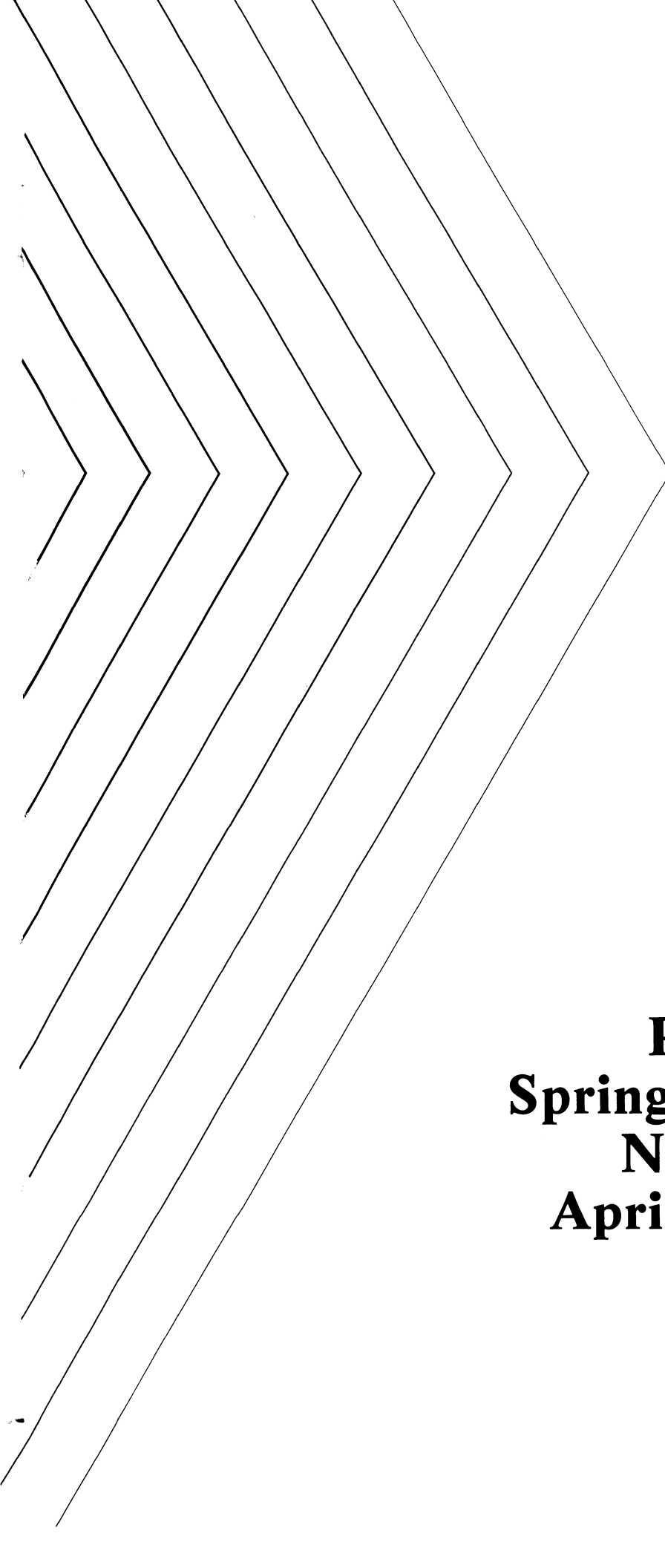


USA DECUS

Nashville, Tennessee



1987 Spring Proceedings



**Proceedings
of the
Digital Equipment
Computer Users
Society**

USA Spring 1987

**Papers Presented at
Spring, 1987 Symposium
Nashville, Tennessee
April 27, - May 1, 1987**

Printed in the U.S.A.

"The following are trademarks of Digital Equipment Corporation"

ALL-IN-1	FALCON	Q-bus
BASEWAY	IAS	Rainbow
DATATRIEVE	LA100	RSTS
DEC	MASSBUS	RSX
DEClab	MicroPDP-11	RT
DECmate	MicroPower/Pascal	UNIBUS
DECnet	Micro/RSX	VAX
DECpage	MicroVAX	VAXcluster
DECSYSTEM-10/20	MicroVMS	VMS
DECUS	PDP (et al.)	VT100 (et al.)
DECwriter	PDT	Work Processor
DIBOL	P/OS	WPS-PLUS
Digital logo	Professional	

Copyright ©DECUS and Digital Equipment Corporation 1987 All Rights Reserved

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation or DECUS. Digital Equipment Corporation and DECUS assume no responsibility for any errors that may appear in this document.

POLICY NOTICE TO ALL ATTENDEES OR CONTRIBUTORS "DECUS PRESENTATIONS, PUBLICATIONS, PROGRAMS, OR ANY OTHER PRODUCT WILL NOT CONTAIN TECHNICAL DATA/INFORMATION THAT IS PROPRIETARY, CLASSIFIED UNDER U.S. GOVERNED BY THE U.S. DEPARTMENT OF STATE'S INTERNATIONAL TRAFFIC IN ARMS REGULATIONS (ITAR)."

DECUS and Digital Equipment Corporation make no representation that in the interconnection of products in the manner described herein will not infringe on any existing or future patent rights nor do the descriptions contained herein imply the granting of licenses to utilize any software so described or to make, use or sell equipment constructed in accordance with these descriptions.

Ada is a trademark of the U.S. Government, XEROX, and XNS are trademarks of Xerox Corporation, IBM, PROFS, PC-XT, and BITNET are trademarks of International Business Machines Corporation, UNIX is a trademark of AT&T Bell Laboratories, CP/M, PL/I are trademarks of Digital Research, Inc., MS-DOS is a trademark of Microsoft Corporation, TSX-PLUS is a trademark of S&H Computer Systems Inc, R:BASE.4000 is a trademark of Microrim, Intel 8088 is a trademark of Intel Corporation, LOTUS 1-2-3 is a trademark of Lotus Development Corporation, MULTIPLAN is a trademark of Microsoft Corporation, Mylar is a trademark of E.I. DuPont de Nemours & Co., PLOTLN is a trademark of Image Research and Compugraphic Corporation, MUMPS is a trademark of Massachusetts General Hospital, Macintosh is a trademark and licensed to Apple Computer, Inc., Multibus is a registered mark of Intel Corporation, 8086 is a trademark Intel Corporation, VENIX is a trademark of VenturCom., Inc, Appletalk, and Apple II are a trademarks of Apple Computers, Inc., INGRES is a trademark of Relational Technology, Inc, Scribe is a trademark of Unilogic Ltd, UniLINK is a trademark of Applitek, HYPREhannel is a trademark of Network Systems Corporation, TIway is a trademark of Texas Instruments, Inc, TCP/IP is a trademark of Darpa, 32000 is a trademark of National, Cyber 180 is a trademark of Control Data, Modbus is a trademark of Gould, Inc, 68000 is a trademark of Motorola, Inc.

The articles are the responsibility of the authors and therefore, DECUS and Digital Equipment Corporations, assume no responsibility or liability for articles or information appearing in the document.

The views herein expressed are those of the authors and do not necessarily express the views of DECUS or Digital Equipment Corporation.

Table of Contents

Artificial Intelligence SIG

- A Visual Rule Editor — Rule*Calc
John R. Thorp, John W. Lewis 1

DAARC SIG

- Laboratory Environment for the Development of
Microprocessor-Based Fluidic Sensor Systems
Steven J. Choy 7

- A Report Generation Language for Control
Engineers
David H. Geer, Jay A. Turner 13

- Spatial/II - A Technical Overview
Mark L. Palmer 25

DATATREIVE/4GL SIG

- Advanced DATATRIEVE Record Definitions
Bart Z. Lederman 35

- Solving Equations in DATATRIEVE
Bart Z. Lederman 45

- VAX DATATRIEVE Security Using Environment
Accounts and ACLs
Michael G. Graham 53

EDUSIG

- Making an Inexpensive Rainbow Workstation for a
Chemistry Lab
John D. Bak, David M. Hayes 63

- Student/Faculty Communications by Computer
Claude M. Watson 69

- Faculty Retraining: A Report from the Front
Edward A. Boyno 73

- Using VAX/VMS to Teach Computer Organization
Linda Lankewicz 79

Graphics Applications SIG

- Readability of VMS Documentation: Then and
Now
C. Eric Kirkland, William P. Brenneman 89

- Overview of Human Factors and Software
Engineering
C. Eric Kirkland 95

- Postscript Applications Using a MNC/DEClab-23
Computer
O. Guetta, D. Fortney, A. Dubois 103

IAS SIG

- Experiences with an IAS-VMS DECnet System
Frank R. Borger 119

Large Systems SIG

- AMAR — A TOPS Performance Monitor
Betsy Ramsey 125

- Planning and Implementing a Large Network
Leslie Maltz 133

Table of Contents

High End VAX System Update Warren Sander	137	Programming in the RSX Indirect Command Language Thomas R. Wyant, III, Arnold S. De Larisch	331
A Practical Exercise in System Sizing Warren Sander, Daniel A. Deufel	147	RT-11 SIG	
High End VAX Configuration: Putting the Pieces Together High Performance Systems Group	153	Moving Decision Points Outward From Applications and Utilities and into Command Level Maarten van Swaay	347
DECSYSTEM-20 Technical Update Mark Pratt.	173	Site Management and Training SIG	
TOPS-10/20 MS/MX Internals Mark Pratt.	177	Analysis of VMS Accounting Data for Determination of Computing Resource Consumption Nancy J. Martin	353
VMS Internals for TOPS-10/20 System Programmers David Wager.	179	Computer Room Design and Construction: A Case History Brent Teeter.	359
Languages & Tools SIG		UNISIG	
Developing LSE Source Code Templates James M. Briggs, Raymond J. Bentz	245	Wading Through Net.News: There's Gold in Them Thar Hills!! Kurt L. Reisler.	365
VAX/VMS Application Performance Louise Wholey	251	VAX Systems SIG	
Networks SIG		Integration of Input/Output Devices Using Silicon Compilation Dr. Robert Couranz, Edwin Rogers, Laurence Specter	373
Transferring Data Between Heterogenous Computers: A Tool to Maintain the Integrity of Foreign Data Steven J. Kempner.	285	Defending Against Trojan Horses, Viruses and Worms Robert A. Clyde	381
Gaining Control: Information Distribution in a Multi-Vendor Corporate Environment Tom Cheatham	291	VAX Systems Coexisting in a Multivendor Environment Robert C. Groman	387
VMS, XENIX, UNIX and MS-DOS Transparent Resource Sharing E. Berelian, L. Farmer, H. Kilman, P. Schoen, P. Wang, J. Vij.	295	The Allocation and Mounting of Magnetic Tapes Under VMS Clyde T. Poole	391
Office Automation SIG		Mysteries of VAX/VMS System Parameters Revealed Steven Szep	395
Documenting Single-Package Systems Michael J. Doyle.	311	An Introduction to VAX/VMS System Tuning Steven Szep	409
RSX SIG			
Introduction to the RSX, P/OS, and RT Indirect Command File Processor Thomas R. Wyant, III	317		

Table of Contents

Unnatural Resources: Working Sets, Quotas, and Limits Steven Szep	415	VMS Disk Performance Management Wef Fleischman	453
Utilizing VAX Uptime Steven Szep	419	On-Line Security Monitoring System Marino J. Niccolai, Linda B. Lankewicz	461
Risk Assessment in System Security Steven Szep	427	Refereed Paper Competition Submissions	
Linda: A Tuner's Home Companion Steven Szep	433	Estimating Development and Run Time Resources: A Practical Example Anthony C. Picardi	463
An Evaluation of Record I/O Versus Block I/O From a Programmer's Viewpoint Darylene Colbert.	443	Improving Technical Manuals through Reader Analysis Thomas L. Warren.	475

FOREWARD

This Proceedings is published by DECUS (Digital Equipment Computer Users Society), a world-wide society of users of computers, computer peripheral equipment and software manufactured by Digital Equipment Corporation. The U.S. Chapter of DECUS has approximately 56,000 active members.

DECUS maintains a library of programs for exchange among members and organizes meetings on local, national and international levels to fulfill its primary functions of advancing the art of computation and providing a means of interchange of information and ideas among members. Two major technical symposia are held annually in the United States.

For information on the availability of back issues of U.S. Chapter Proceedings as well as forthcoming DECUS symposia, contact the following.

DECUS U.S. Chapter
219 Boston Post Road, BP02
Marlboro, MA 01752-1850

All issues of past Proceedings are available on microfilm from:

University of Microfilms International
300 North Zeeb Road
Ann Arbor, MI 48106



PREFACE

This volume of the Proceedings contains papers which were presented at Symposia sponsored by the Digital Equipment Computer Users Society during the Spring and Summer of 1987. It includes submissions from the Spring National Symposium.

The Spring 1987 Symposium was held at the Opryland Hotel in Nashville, Tennessee, from April 27 through May 1, 1987. 5730 DECUS members met in Nashville for a week of education, information, cooperation, sharing, and a good deal of camaraderie.

Digital Equipment Corporation is a very large company. Long overshadowed by the enormous bulk of International Business Machines, Digital adopted a small-town mentality for many years — - kids selling computers to kids. But Digital has been growing at a tremendous rate. Financial analysts are beginning to recognize the trends that Digital users have seen brewing for three years. Last year's computer industry slump, combined with Digital's quadrupling of stock values over a three year period, has let the rest of the world see that Digital is ready to play ball with the big boys.

DECUS, in many ways, has not taken Digital seriously enough. Most DECUS leaders remember Digital as it was five years ago, and their attitudes and decisions are based on their own lifetime of working with Digital. As natural attrition works on this group, though, replacements are coming from a new community of business people (men AND women) that have always known Digital as a corporate computing resource. If seventy percent of DECUS worked for a University or Research and Development center five years ago, then seventy percent now work for corporations, software and hardware vendors, and one-man consulting outfits (specializing not in real-time data acquisition, but corporate MIS planning). These new members (and new leaders) not only change the face of DECUS; they ARE DECUS.

Mark Grundler, who has admirably led the Communications Committee since 1984, has resigned to spend more time with his new wife and child. All of us send our best wishes to Krista and Mark, and thank him for his dedication and hard work. Beverly Welborne has agreed (via unanimous acclamation!) to take up the baton.

My thanks on behalf of the attendees of the Spring National Symposium go out to Scott Pandorf and Emily Kitchen, the DECUS volunteers who led the Symposium Committee. They worked together with DECUS staff members Nancy Wilga, Joanie Mann, Gloria Caputo, Maria Hance, Rosemary Lupo, and Beverly Dandeneau to put together the Nashville meeting. The leadership of the entire Symposium Committee is sincerely appreciated. Judy Mulvey deserves my special thanks for her exhausting work for the Communications Committee and all of our congratulations on her recent marriage. My colleague, Cheryl Smith did the production and layout work for the Proceedings, and her time and energy is most sincerely appreciated.

ARTIFICIAL INTELLIGENCE SIG

A Visual Rule Editor – Rule*Calc™

John R. Thorp
John W. Lewis

Martin Marietta Laboratories
1450 S. Rolling Rd.
Baltimore, Md 21227

ABSTRACT

Most current expert system tools require a middleman, the knowledge engineer, to code and test the thousands of rules that constitute the expert system. This knowledge acquisition process typically requires hundreds of tedious code and test cycles. The visual rule editor, Rule*Calc™, is an environment for developing rule-based systems without the continuing assistance of a knowledge engineer. Rules can be entered on the screen in a Lotus 1-2-3® style format via a syntax similar to that of traditional decision tables. The semantics of the rules is derived from that of the EMYCIN system. Rule entry/debug time is well below that required by the expert to formulate rule content.

INTRODUCTION

The most critical resource of expert system design has been the domain expert, i.e., an individual whose degree of proficiency in some trade is of great value to his or her organization. With the maturing of expert system technology, it has become computationally feasible to codify the expert's knowledge and use this information to emulate the expert's decisions. However, experts often have neither the time nor the inclination to learn how to codify their knowledge.

Therefore, the task of transferring the knowledge from the expert to the machine is typically performed by a second individual, a knowledge engineer, who knows how to translate expert's knowledge into software. The knowledge engineer must then run through several cycles of interviewing the expert, coding the knowledge, and verifying the validity of the rules. Each iteration may require several days, and the entire process is normally repeated successively to build more complex versions. The resulting system may contain hundreds of rules.

There are several disadvantages to this approach: first, the services of a knowledge engineer are an added expense; second, the expert may not be available for the uninterrupted meetings necessary for the transfer of knowledge; and finally, as the size of the expert system grows, it will begin to suffer from many of the problems that currently plague large software projects.

Our proposed solution, Rule*Calc™, solves these problems by providing the expert with a straightforward interface to the machine.

FUNDAMENTAL CONCEPTS

Directed acyclic AND/OR graphs were selected for Rule*Calc™'s conceptual representation of expert systems since most users are familiar with propositional logic. This representation thus provides an interface that can be effectively used by individuals from different disciplines. A tabular screen interface was also created to access and manipulate the graphs.

AND/OR Graphs

An AND/OR graph contains a hierarchy of nodes with inputs coming from level N+1 and outputs going to level N-1, where N is the node's depth in the graph. Each node performs some test on its inputs (i.e., and, or, not) and, when a particular input configuration occurs, "fires" and sends new values to its neighbors.

It is computationally beneficial to view graph-oriented logical operations differently from linear logical expressions. Thus, nodes do not need values on all inputs to determine the output values. Short-circuit evaluation may be used to determine the node's value as soon as it is firm. For instance, should the first input to an *OR node* be true, the value of the output must be true. It is unnecessary to know the values of any other inputs to the *OR node*. A similar statement may be made about an indeterminate value on one of the inputs to an *AND node*.

Since our inference engine uses a tri-state logic (true, false, and unknown), special conventions are needed in addition to those used in traditional logic (Table 1). Our model uses an open-world assumption

so that if a node's value cannot be deduced, its value is set to unknown.

NOT	AND			OR		
	T	F	?	T	F	?
T	F	T	T	F	T	T
F	T	F	F	F	F	F
?	?	?	?	F	?	?

Table 1. Truth tables for Rule*CalcTM's standard logic operations.

Mapping AND/OR Graphs to Expert Systems

From the designer's viewpoint, an expert system shell can emulate a forest of AND/OR graphs. The roots of the graphs correspond to goals in the expert system and terminal nodes map to facts which obtain their values from a source outside the system. All other nodes in the graphs are intermediate nodes, which receive values from lower level nodes and send values to higher level nodes.

For this analogy to work, it is necessary to place some restrictions on the configuration of the graph. Rule-nodes AND the values on level N+1 and fact-nodes OR the values on level N+1, as was described above for AND/OR graphs. The top level, level 1, of a graph always contains only fact-nodes; thereafter, even-numbered levels of the graph contain rule-nodes, and odd levels, fact-nodes.

Digraph Oriented Chaining Methods.

Rule*CalcTM uses a combination of forward and backward chaining to carry on a dialogue with the user. The algorithm begins by selecting a top-level goal and back-chains from it to a terminal node (or fact). This fact receives a value, either from the user or an embedded procedure. The new value is then projected as far as possible through the graph via forward chaining. If a goal-fact is not satisfied, the process starts again.

The backward-chaining algorithm consists of one procedure and one loop.

```

SELECT-BEST goal-fact
REPEAT-UNTIL current fact is a terminal
  SELECT-BEST rule related to current fact
  SELECT-BEST fact related to current rule.
  
```

The SELECT-BEST function is discussed later.

The system propagates newly set values by moving the values up the rule/fact graph according to the standard logical operations. However, to avoid

incorrectly deducing that a fact is unknowable, it does not set facts to unknown unless ALL immediately subordinate rule-nodes are unknown or false.

Search Strategies

The inference engine is wholly contained in a single subroutine, allowing it to be removed and replaced by a functionally equivalent module. Using this feature, we have implemented several "graph following" algorithms behind the uniform screen interface.

The first method used to navigate the graph was a left-to-right, depth-first search (DFS).¹ This is the simplest and most straightforward method of coding the high-level back-chaining description. The graph's traversal path is selected at each node based on the evaluation of a desirability function (i.e., SELECT-BEST), thus allowing the path to be dynamically reconfigured.

The desirability function for each node is calculated from the cost of test of the node's subordinates, the possibility that traversal of the node would lead to a correct solution and the percentage of the subproblem represented by the node that has been solved. Weights are assigned to each of the criteria and varied to tune the system and reflect the importance of each parameter. Specifically, these parameters are designated Node_Cost, Node_Possibility, and Node_Percent_Solved. A simple desirability function can be described as follows:

When multiple subordinate nodes exist, the inference engine will select the node with the highest certainty factor (CF), where:

$$CF = \text{Node_Possibility} / \text{Node_Cost}$$

When a choice must be made between several facts with equal CF's, facts closest to the top of the table are selected first. Similarly, when the engine must choose between several rules, rules with lower rule numbers take priority.

Alternate search strategies with identical desirability functions were tested, including branch and bound and A* algorithms. While both algorithms reduced search time and mean queries before solution, they also made it more difficult for designers to ensure the coherence of the questioning in end-user dialogues. Therefore, DFS is installed in the current version of Rule*CalcTM.

Conventional forward chaining was enhanced to include the desirability function and percolate values forward. New costs and probabilities for related, unset nodes were recalculated from the minimum and maximum values, respectively, of the remaining subordinate nodes.

THE EDITOR INTERFACE

The editor's user-interface consists of three parts, each of which serves to edit some property of the facts. In some cases, there are custom editors tailored to facilitate changes to properties of a particular type. Fact actions and fact relationships, two of the more important properties which may be manipulated in the editor, each have their own editor format.²

The Fact-Relationship Editor

To facilitate learning, the relationship between facts and rules is displayed in a tabular format. Specifically, the rule-editor screen is seen as a matrix whose columns and rows correspond to rules and facts, respectively. If there is no relation between a fact and a rule, the entry in the matrix is blank. However, if the fact is involved in the rule, the intersection is filled with the symbol representing the rule operation to be performed on the fact.

Valid rule operations are:

Category	Entry	Key	Action
Antecedents	=	=	Test if fact is true.
	~ =	~	Test if fact is false.
Consequents	-T	t	Set fact to true.
	-F	f	Set fact to false.

Most operations in the editor may be performed by a single key stroke, including fact/rule access, fact/rule cut and fact/rule paste.

The Fact-Action Editor

Actions are blocks of text that are executed or are placed on the screen when events related to the fact's value occur. This editor provides a means of manipulating the actions associated with a given fact. All actions related to a fact are displayed in windows on one screen, and the user may move between these windows or between screens to perform editing operations.

The Run-Time Environment

Users' run-time needs differ widely: designers need a flexible environment with rule-tracing facilities; novice end-users need a system that provides clear queries and statements with additional explanations when necessary; and more sophisticated end-users may wish to direct the system to pursue a particular line of reasoning.³

In Rule*CalcTM, the expert system designer can switch between the end-user environment and the editor with a single key stroke. This ability enhances the rule-testing and repair facilities. In the end-user environment, the programmer has access to the

query-dialogue window, as well as a rule-stack debugger. The debugger may be used to dynamically trace chaining and observe fact value assignments.

Alternately, the end user has access to two query modes: novice and skilled. The novice mode provides an interface that will guide the user to the proper conclusion, while the skilled mode allows the experienced technician to enter facts already inferred or tests already performed and thus guide the system in a particular direction.

In addition to optional conversation styles, Rule*CalcTM offers several different interface styles. The standard keyboard/CRT combination is supported, as well as a mouse and menu interface on terminals that support the mouse. Furthermore, the system may query the user via a speech synthesizer and will accept responses entered through a standard telephone keypad.

CODE GENERATION

Rule*CalcTM serves as a standard representation for expert systems that can be translated into other computer languages. The run-time environment can be recreated by programs written in traditional languages, and Rule*CalcTM has the ability to automatically generate these programs. This feature may be used to port the expert system to other machines or other operating systems.

To demonstrate this capability, we selected Common LISP as our target language.⁴ The generated code allows users to add application-specific code to the output program to customize screen formats or direct the program to acquire data from peripheral devices. This function of Rule*CalcTM has generated several application programs in the diagnostic and maintenance area.⁵

The language generation feature may be tailored to write in a target language that is a general purpose computing language. Initial investigations indicate that it is feasible to generate expert systems in PASCAL, ADA, PROLOG, FORTRAN 77, and C.

CONCLUSIONS

Several ideas incorporated in the final design of Rule*CalcTM have aided the production of in-house expert systems. Providing a visual editor capable of single key commands increases the interactivity of the program and allows the user to see the effects of a change immediately. Presenting forward and backward chaining as graph operations allows the designer to grasp these concepts with a minimum of training time.

In addition, Rule*Calc™ offers several solutions to common expert system problems. Due to the style of interface, domain experts may code the expert system without help at their convenience. The high-level language code-generation facility eliminates the task of recoding the developmental prototype to a distribution-grade program, and the generated code automatically reduces the number of software design errors by providing a structured base for a larger custom system.

Trademarks for Rule*Calc™ and Lotus 1-2-3® are owned by Martin Marietta Corporation and Lotus Development Corporation, respectively. The software presented in this report is for Martin Marietta use only and not intended for public release.

REFERENCES

- [1] Aho, A. V., Hopcroft, J. E., and Ullman, J. D., *The Design and Analysis of Computer Algorithms*, Reading, Mass.: Addison-Wesley, 1974.
- [2] Lewis, J. W., "An effective graphics user interface for rules and inference mechanisms," in, *Human Factors in Computing Systems*, Amsterdam, Netherlands: North-Holland, 1984.
- [3] Waterman, D. A., *A Guide to Expert Systems*, Reading, Mass.: Addison-Wesley, 1986.
- [4] Steele, G. L., Jr., *Common LISP: The Language*, Hudson, Mass.: Digital Press, 1984.
- [5] Lewis, J. W., and Wysocki, E. M., "Applicability of Expert System Technology To FAA Maintenance," *ATCA Fall Conference Proceedings*, 1986.

DATA ACQUISITION, ANALYSIS, RESEARCH, AND CONTROL SIG

Laboratory Environment for the Development of Microprocessor-Based Fluidic Sensor Systems

Steven J. Choy

U.S. Army Laboratory Command
Harry Diamond Laboratories
Adelphi, MD 20783-1197

Abstract

The Harry Diamond Laboratories Fluid Control Group uses a variety of embedded microprocessors as an integral part of its realtime control systems. Typical processors include the Digital Equipment Corporation (DEC) J11 and the Motorola MC68000 integrated into a GESPAC G64 bus-based system. A laboratory environment for the development of these fluidic sensor systems has been established which consists of a DEC VAX 11/780 computer, loosely coupled to a DEC μ PDP 11/73 controlling a variety of analog and digital devices, attached to a IEEE-488 bus. The hardware/software components of the laboratory environment and how they are being used to develop new realtime systems are presented. The discussion also focuses on insights acquired and problems experienced when dealing with interfaces between DEC and non-DEC components.

Overview

As microprocessor hardware technology becomes faster, more complex, and available in smaller and cheaper packages, the domain of applications to realtime systems has become unbounded. General-purpose microprocessors with tremendous computing abilities can now be affordably integrated into tiny realtime systems where, not too long ago, this was not considered feasible. The hardware possibilities appear to be endless.

This "utopia" of microcomputer processing does not come without cost. It takes a considerable amount of time and effort to design, develop, integrate, and test these processors into a workable and usable system. In order to harness this computing power into a useful productive system in a timely manner, a flexible development environment must be established, allowing an engineer to experiment and analyze the possibilities for integration of these microprocessors in an embedded realtime control system. This paper discusses a laboratory environment developed by the Fluid Control Group at Harry Diamond Laboratories (HDL) for the explicit purpose of designing and developing prototype realtime control systems using embedded microprocessors. The discussion presents an overview of both the laboratory hardware and software.

Fluidics, A Brief Summary

Fluidics is a technology that uses liquids and gases to perform sensing, logic, amplification, and control functions *without moving mechanical parts*. Fluidics is finding its way into systems that require high reliability because of the absence of moving parts, and low maintenance costs. Some of the

applications pioneered at HDL, for this technology include temperature sensing, hydraulic stabilization, and angular rate sensing. The last of these applications can be used for building low-cost, reliable, autonomous navigation systems. (Since the theory of operation behind these devices is beyond the scope of this paper, the reader is referred to reference [1] for more background in this area.) The discussion in this paper focuses on the laboratory environment devised at HDL for using microprocessors and fluidic elements in developing such navigation systems.

Laboratory Hardware

The laboratory hardware can be divided into two major categories: (1) there is the digital processing hardware used for both development and for the actual application, and (2) the analog interfacing hardware used for measurement and control. Some of this hardware is for development and testing, and some of this hardware is an integral part of the application.

The interrelationship of the various laboratory hardware components is shown in Figure 1. The digital processing hardware involves three systems:

1. An embedded application processor (i.e., DEC J11, Motorola MC68000) for controlling the realtime application.
2. A DEC VAX 11/780 system for cross-development of the embedded microprocessor used in the application system.
3. A DEC μ PDP 11/73 system for data acquisition and environmental control used for testing and simulating the application system.

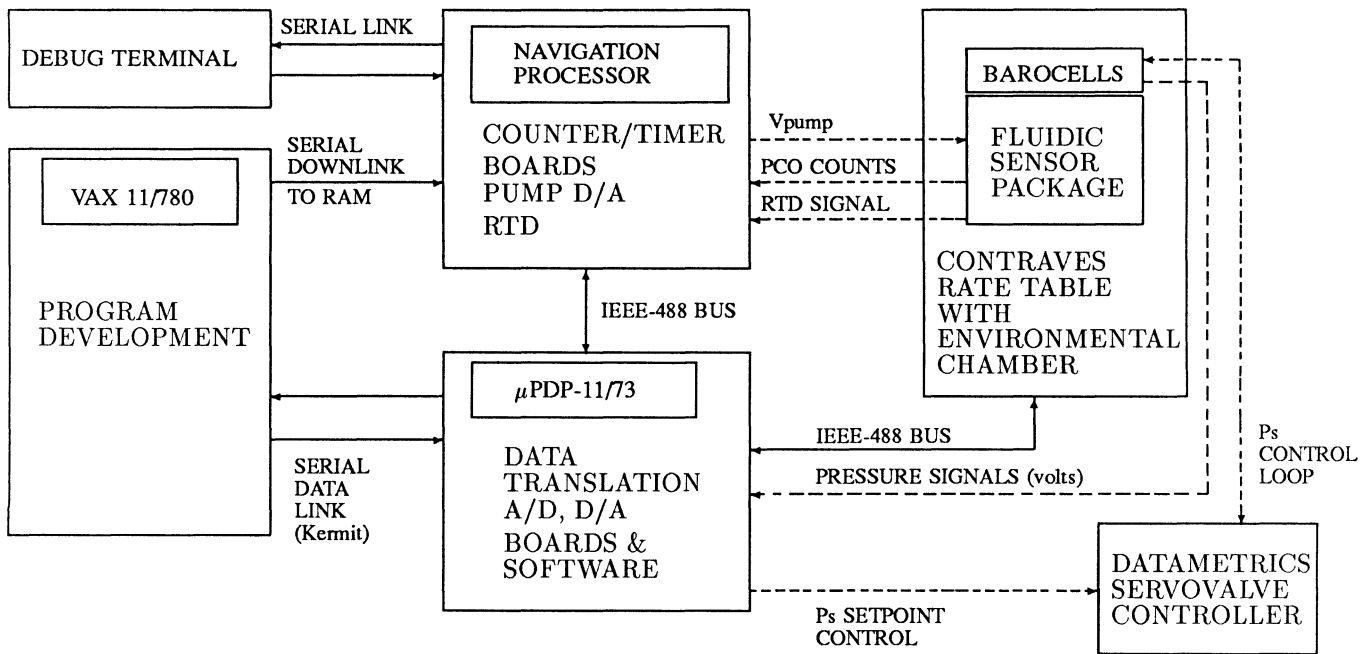


Figure 1: Laboratory hardware configuration

The analog hardware is made up of control components for simulating various environmental parameters as well as measuring components for analyzing the system response to different controls. Connected on an IEEE-488 bus is a Contraves combination temperature control chamber and rate table, used for sensor testing under different environmental conditions. Also attached to this bus are several Hewlett-Packard multi-meters and a platinum temperature probe for high-accuracy temperature readings.

Connected on the Q bus via a DTI (Data Translation Inc.) interface is a 12-bit, digital-to-analog converter (D/A) used for servo pressure control. On the input side of the interface are a number of counters to collect data from the fluidic oscillators, along with several analog-to-digital converters (A/D) for monitoring the Barocell pressure transducers, flowmeter inputs, and other miscellaneous sensor inputs.

The Embedded Processing System

The embedded processing hardware used by the navigation system is built around a G-64 bus-based system available from Gespac Inc. This hardware was chosen because of it is relatively small and compact and has low power requirements (owing to the use of CMOS parts). In addition, Gespac provided boards with a wide range of powerful processors, including the DEC J11, the Motorola 680X0, the National Semiconductor 32010, and the Intel 80X86. These processing boards, along with the availability of various supporting peripherals such as counters, parallel and serial input/output (I/O) interfaces, A/D's and D/A's, and other digital support devices, made it possible to quickly assemble and arrange a prototype system without lengthy hardware development times.

The particular example navigation system described in this paper uses an 8-MHz M68000 processing board with 16

Kbytes of random access memory (RAM) and 128 Kbytes of electrically programmable read-only memory (EPROM). Figure 2 shows the basic layout of the system and its supporting devices. In this system, a realtime clock is used to set a fixed rate (usually in the range from 10 to 100 Hz) to sample two realtime inputs.

The angular rate of the system comes from the fluidic rate sensor via a 32-bit counter, and the distance traveled comes from a distance sensor via a 16-bit A/D. The distance direction is a transistor-transistor logic (TTL) level that is connected to the CTS pin of an RS-232 port on the multifunction I/O board. The system communicates to an operator by way of a 64 by 256 pixel flat panel liquid crystal display (LCD) overlaid by a 4 by 10 sectioned touch panel. Two serial ports are available for interactive terminal debugging and program downloading from the VAX to the on-board RAM. During the development stage, an extra 32-Kbyte memory board is used along with an IEEE-488 bus interface for communication to the μ PDP11.

The Cross Development System

Software for the embedded processor hardware is developed on a VAX 11/780 (running VMS, not ULTRIX) using a cross compiler system. As shown in Figure 1, the two systems are linked via a standard 9600-baud RS-232 serial link. During the program development phase, the test programs for the embedded processor are loaded into RAM over the serial link. When the program logic has been "debugged," the object module is converted to hexadecimal and sent to a DATAIO EPROM programmer connected to the VAX on another serial interface.

Using the VAX as a software development system (as opposed to using the embedded processor) provides several advantages:

1. The VMS operating system provides an excellent envi-

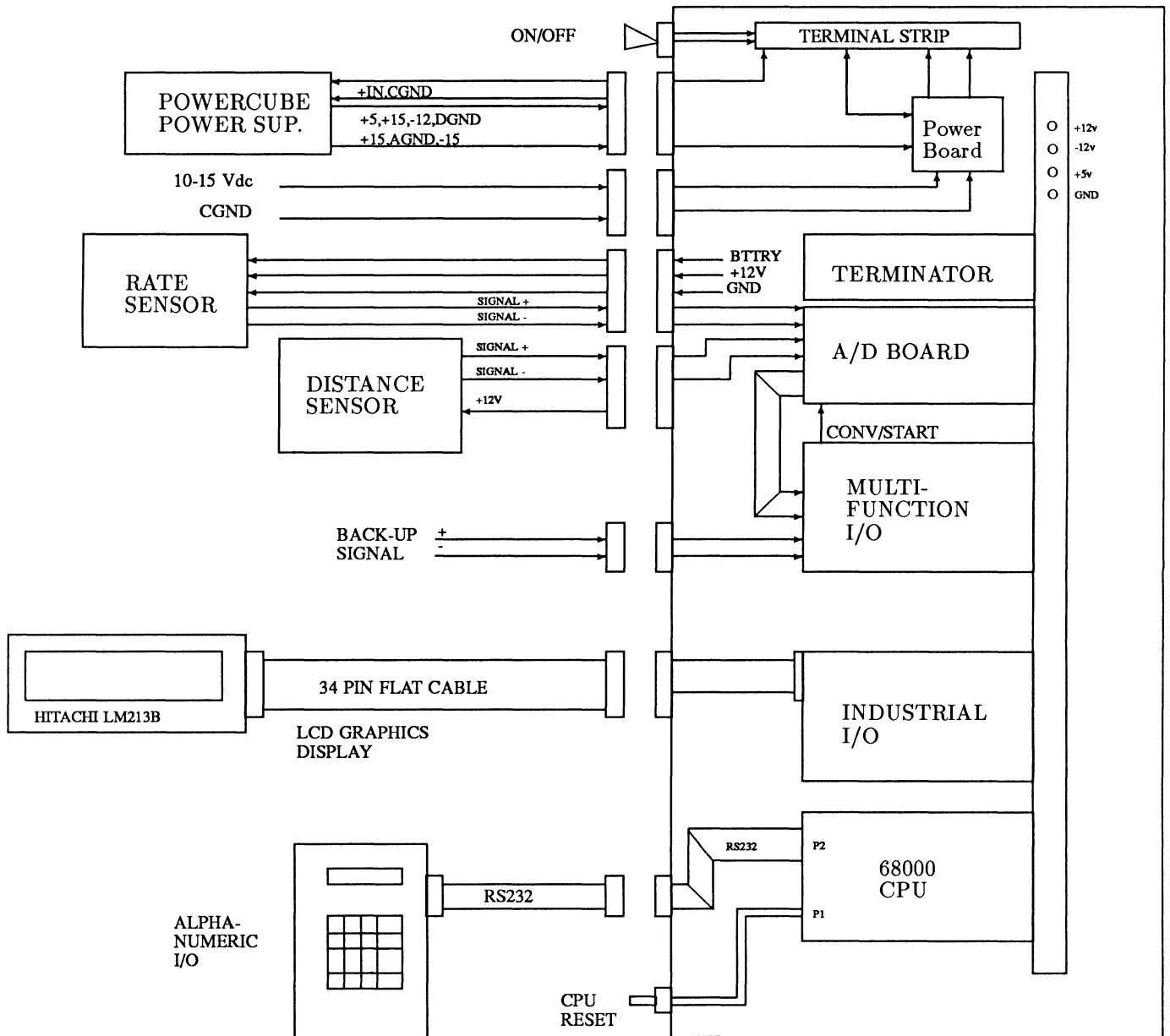


Figure 2: Embedded processor hardware

ronment for program development, including support for program source code preparation, debugging, documentation, management, and backup.

2. The VAX is a multi-user system, allowing several persons to work on the software development simultaneously.
3. Since the VAX does not require the use of the embedded processor, hardware development can also be occurring simultaneously with the software development.
4. The VAX has a variety of shared peripherals available such as line printers, plotters, and disks, which are not normally connected to an embedded processor.
5. Since most of the software development is in a high-level language, device-independent algorithms can be de-

bugged completely without the actual hardware ever having to be used.

6. By having a central repository for all software, much of the developed program code is easily reused and shared among different systems. This reduces the development effort significantly for subsequent systems.

In addition to providing cross-development facilities for the embedded processor, the VAX can also be used as a file server for the embedded processor during data collection periods. Through a simple, compact protocol on the serial interface, test programs loaded into the embedded processor can make file system service calls to open, close, read, and write transparently to VMS files.

The Environmental Control System

The μ PDP11 is an RSX11M based system that is responsible for all the test measurement and control in the laboratory. It is also used for analyzing collected test data and producing calibration information for the fluidic sensors. The calibration information is forwarded to the VAX via a serial link to be incorporated into the embedded processing software. The information is usually in the form of numerical tables that are reprocessed and converted to binary images on the VAX and eventually burned into PROM's. The data can also be analyzed further and plotted on the VAX, where high-quality laser plotters are available.

As described above, most of the test and measurement hardware is connected to the μ PDP11 via an IEEE-488 bus. The remaining components are connected to the μ PDP11 via interface hardware on the Q bus provided by DTI. In the laboratory setup, the collection of data requires the cooperation of two processors.

- The main control processor (the μ PDP11) establishes and monitors the test environment.
- The embedded application processor (the M68000) collects the data in realtime and forwards the data to the μ PDP11.

In a typical setup, the embedded processor is downloaded with a data collection test program that communicates with the μ PDP11 over the IEEE-488 bus. The μ PDP11 sets the environmental test chamber (also connected on the IEEE-488 bus) to the required test conditions and then commands the embedded processor to begin data collection from the fluidic sensor. The sample size and the sample rate along, with other variables involved in the data collection, are all settable by the embedded processor via commands from the μ PDP11. The collected data values are buffered in the embedded processor's memory and sent to the μ PDP11 via the IEEE-488 bus.

This cooperative processing configuration allows an engineer to easily test the application system in different environments via hardware simulation in the laboratory. In addition, the μ PDP11 can be used to monitor the fluidic sensor output in parallel with the application processor to provide a performance reference for the application system. This performance information can be used to calibrate individual application systems.

Support Software

As mentioned in the previous section, using the VAX as a centrally located program development system allows much of the developed software to be reused in subsequent systems. The HDL fluidics control group uses its own "generic" embedded realtime operating system, which provides an ever-expanding library of system-level support, such as device drivers for different peripheral chip interfaces, memory management, task management and communications. In addition, application software libraries have been developed to support such functions as graphics display, user interfacing, operator input, and command parsing.

Most of the software is written in the C language using a cross compiler. Because of the special attention given to processor independence, most of the code can be transported to different target processor boards. The cross-development system provides for a universal linker and librarian. Thus the same linker and librarian may be used for different target processors, providing uniformity for the program development process.

Two target processors have been used thus far, the DEC J11 and the Motorola M68000. From a software viewpoint, the two processors are architecturally similar. Both processors provide several general-purpose registers, using similar addressing modes. Both use memory mapped I/O, along with privileged and user operating modes.

The J11 has the advantage of providing hardware-assisted floating-point math in both single- and double-precision modes. The M68000 only provides for integer calculations. In the context of the navigation software, in order to minimize drift error, the nature of the mathematics requires a high level of precision over a wide dynamic range. The double-precision math processor in the J11 is ideal for this calculation. In order to implement the same calculations on the M68000 in realtime, it is necessary to construct abstruse integer math algorithms that operate on even more abstruse integer representations of the data structures.

However, the M68000 has the advantage of being able to directly address more memory than the J11 because of its logical 32-bit architecture. The J11 requires the programming of a memory management unit when addressing beyond the abilities of the 16-bit program counter. This advantage is minimal for the navigation systems since the programs rarely extend beyond 48 Kbytes.

The Embedded Processor Operating System

The HDL-developed embedded realtime operating system is not only being used on various G-64 bus-based boards, but is also being used extensively with VERSABUS- and VMEBUS-based processor boards, as well as in-house-designed processor boards. The transportability of the system is attributed to the use of high-level language programming and system modularity. Features of the operating system include

- multitasking or foreground/background system configurations including support for multiprocessor as well as uniprocessor systems;
- memory management utilities to support dynamic storage allocation and the manipulation of data on stacks, heaps, queues, and dequeues (linked or contiguous data structures);
- integral runtime debugging utility;
- extensive runtime library support; and
- modular software components, making the system adaptable and reconfigurable to small, minimal configurations as well as large, complex ones;

Depending on the particular system being developed, the operating system can be configured for multiple task support or for foreground/background support. Often the overhead (both processor and memory) incurred by multitasking is undesirable, and so some systems are configured without this feature. In addition to using less overhead, foreground/background systems are often less complicated to use (depending on the application).

However, multitasking support provides dynamic creation and deletion of tasks along with task prioritization and several forms of intertask communication. Dynamically created message systems and shared data sections are available, along with semaphores for intertask synchronization and shared resource protection. The software library also includes a message system for intertask, interprocessor communication. When multiple processors are used, the operating system provides constructs for remote booting of a task to another processor along the system bus (i.e., VMEBUS).

Although the embedded operating system has no disk drivers or file system support, these functions, when needed, can be fulfilled in one of two ways. Drivers are provided to make the VMS file system transparently available to the embedded operating system via an RS-232 serial interface and standard C language I/O library calls. Using this method, the VAX can act as a file server to several different slave systems. The disadvantage to using this method is the slow transfer rate of the data. However this technique is useful for quickly generating programs to log test data onto the VAX for later analysis. It is also useful for building command files of comprehensive diagnostic sequences that are normally expecting interactive command input from a terminal. It provides the functionality of a disk file system during the development phase without the need to actually interface a disk drive and associated controller hardware to the embedded processor system. Using familiar C language I/O library functions such as *fprintf()* and *fscanf()* provides an easy-to-use interface to the file system.

When an application requires an integral disk drive with the system, then the embedded operating system is used as a bootstrap to a commercially available, reconfigurable, UNIX-like operating system that provides complete file system support. At this point, the HDL-developed operating system disappears and is replaced with the "UNIX clone" operating system. This method requires writing appropriate disk controller device drivers. Currently no disk devices are used on the GESPAC bus-based systems. However, drivers are written to support disk controllers from Interphase, Inc., and Motorola, Inc., on both VERSABUS and VMEBUS.

The Software Library

The software library provides support for a variety of hardware interface chips. Drivers are available for a number of serial interfaces (UART's) as well as counters, timers, peripheral interface adapters (PIA's), D/A's, and A/D's. Some of the UARTS currently supported include the NEC7201, the MC8650, the SIG2661, the MC68681, the SCN2681, and the Z8530. Counter chips supported include the Z8536, the

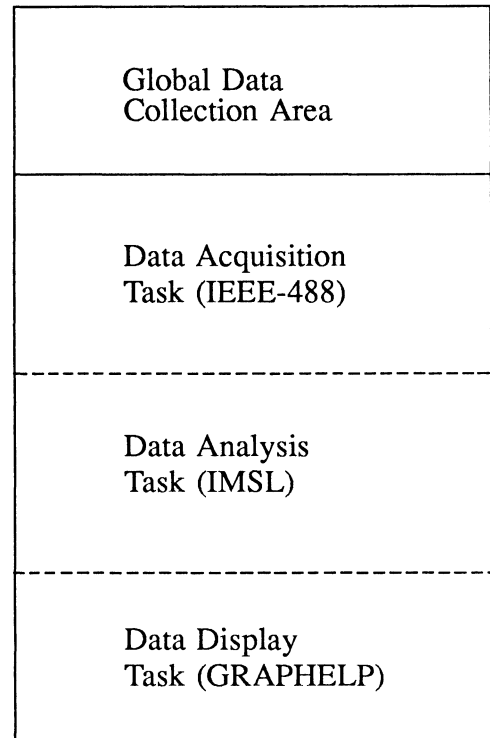


Figure 3: Memory resident RSX11M software modules

MC6840, the AMD2915, and the INTEL8254. Drivers are also written for the INTEL IEEE-488 bus chip set.

Higher level application software libraries exist to support bitmap graphics, including vector-to-raster routines and software font character generators. These routines are useful for driving flat-panel displays such as the one used in the fluidic navigator. Routines to provide vector graphics using Tektronix 4010 protocols are also contained in the application software library along with VT100 screen manipulation routines (ANSI X3.64 escape sequences) and easy to use command parsing routines. These functions, along with the standard C language library functions for formatted I/O and character manipulation, make it easy to quickly generate custom interactive hardware diagnostic and testing packages for new systems.

All the functions are stored in modular object libraries for easy access and reusability. When a new function is required that is almost provided for by a pre-existing library routine, the source code of the original library function can be accessed and tailored to the modified requirement. In this manner the software development time is still shorter than for developing a completely new function.

Data Acquisition Software

The data acquisition and analysis software is written completely in FORTRAN-77 running under RSX11M on the μ PDP11. The software is broken into four basic memory resident modules, as shown in Figure 3. There are three tasks that communicate with each other via a single RSX11M global data section.

The data acquisition task is responsible for the environ-

mental control and collection of data along the IEEE-488 bus. Note that all software related to device interaction was built upon already existing interface libraries provided by the hardware manufacturers (i.e., DEC for the IEEE-488 interfacing, and DTI for the D/A's, A/D's, and counters). Once the data values are stored in the global data section, a second task can be activated to analyze the data. In reality, there are several different analytical tasks built upon a commercially available subroutine library called IMSL. A third task is available to display raw and processed data on a medium resolution (1024 by 800) graphics display terminal. The display software is built on an in-house-developed graphics package called GRAPHELP which runs on both the μ PDP11 and the VAX. The reader should see reference [2] for more information about the graphics support.

It should be noted that the original data acquisition system was developed under RT11. With that system, the three tasks were run separately, passing data between them through temporary files. The multitasking nature of RSX11M allowed the system to be redesigned as three memory-resident tasks sharing a fixed memory-resident data area. In addition to saving disk space, this structure eliminates the need to perform unnecessary, time-consuming file I/O. The practice under RT11 was to collect lots of data onto the disk before invoking the analysis and display tasks. Under RSX11M the data sets are analyzed as they are collected, and only data sets of "significant interest" are stored on the disk.

Conclusions

The HDL fluidics control group has found the laboratory environment described in this paper to be an extremely useful tool in developing embedded microprocessor-based realtime systems. The close integration of the test hardware with the application hardware has made it possible to quickly and easily develop and test new systems in a timely fashion.

The use of modular software libraries and high-level language programming has made it easy to evolve the system and application code to new processor systems and new application systems as new technology becomes available. The reusability of software has become an important factor in developing new application systems.

Future plans include the migration to embedded processors that run at higher clock rates and support floating-point calculations. These will probably not include the J11 because of its low availability in a commercial product. In addition, work is on-going to provide better integration and reconfiguration for multiprocessor systems. The application software library will continue to expand. It is also believed that closer coupling between the VAX development system and the μ PDP11 environmental control system via a DEC-NET/Ethernet interface (as opposed to the RS-232 serial interface currently in use) will enhance the overall system development process.

References

- [1] Stephen Tenney and John Grills, Development of a Low-Cost Navigation Aid, *Proceedings of The American Society of Mechanical Engineers*, Winter 1986, 86-WA/DSC-4.
- [2] Steven Choy, Interactive Graphics Support For Mini-computer Systems, *Proceedings of the Digital Equipment Corporation Users Society*, Winter 1978.

A Report Generation Language for Control Engineers

David H. Geer
General Electric Co.
Schenectady, NY 12301

Jay A. Turner
Digital Equipment Corp.
Albany, NY 12203

Abstract

Report Generator Language (RGL) is a tool for retrieving data from a controls data base and producing printed reports from that data. It is targeted at Control Engineers and plant operators, who have used computers but whose specialty is not computer programming. RGL features include (1) ease of access to the plant data base, (2) a menu-driven interface, (3) detailed, English error messages, (4) structured programming, (5) powerful output formatting constructs, (6) arithmetic functions, and (7) subroutine capability, including recursion.

RGL has been successfully used for formatting reports of current and historical data from plant sensor data bases, monitoring plant equipment, billing, and diagnosis of equipment failures. Because RGL is easy to learn and requires less coding and less debugging, it is a powerful tool for the users of control systems. This paper discusses the development of the RGL language, its features and its benefits.

Introduction

Modern power plants require many kinds of printed reports. Daily, weekly, and monthly production reports are used by operations management. Thermal performance reports are used by plant engineers to monitor the health of plant equipment. Maintenance management reports keep track of parts inventory and equipment running time to aid in scheduling planned maintenance. Emission reports monitor combustion products in the exhaust gas for regulatory agencies. All of these reports require changes from time to time as plant equipment, operating practice, and regulatory requirements change.

Before RGL was developed, plant engineers had two choices: either learn to program in FORTRAN, BASIC or PASCAL, and learn to use the complex subroutine library supplied by the computer vendor, or rely on the vendor to provide the complete logging package. The first approach meant diverting control or instrument engineers from their normal duties for many months of training. The latter meant static formats for logs and reports, or software development charges from the vendor each time a change was made to plant equipment, process, or administrative procedures.

With RGL control engineers can write their own logs after only three days of training. The toolkit guides the user through the phases of editing, compiling, and testing. The familiar syntax and powerful formatting features allow him to learn to use the language quickly and produce exactly the

output he wants.

The Plant consists of a collection of equipment such as boilers, turbines, pumps, valves, and motors. The plant is monitored by Data Acquisition Systems or DASs. These are special purpose microcomputers which are designed to monitor sensors, convert their readings to digital form, and transmit the data samples to the Station Computer for processing and storage. The station computer is a VAX or Micro-VAX, depending on plant size.

A single sensor is called a point. A sensor reading is called a data point, or sample. The Station Computer maintains a file called the Data Dictionary that lists the names of the DASs, the names of each point, and the information needed to convert samples of each point for printing.

The Station Computer maintains two data bases: The real time data base, which contains the most recent value of each point, and the historical data base, which contains a record of the data points that the system has received over time. The real time data base is keyed on point name, and the historical data base is keyed on point name and time.

What is a Report Generator?

A Report Generator is a system of programs that are designed to make it easy to produce printed output. RGL components are shown in Figure 1.

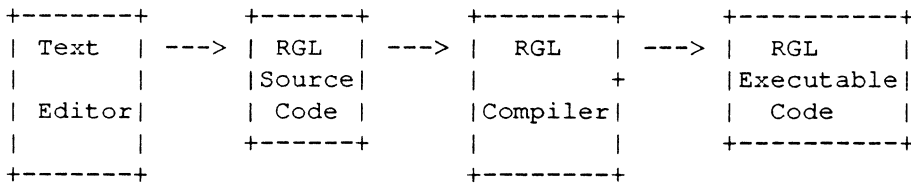


Figure 1: RGL Components

RGL programs (reports) read data and format the results into the output (view) file. The output file is processed with the Report Viewer program to produce printed output and color plots. RGL data flow is shown in Figure 2.

The Viewer program lets the user preview the output before printing or plotting.

Ease of Access to Plant Data

RGL provides routines to get data from the historical and real time databases as well as from the data dictionary file, Maintenance Management files and ASCII text files. In all, RGL supplies over forty data access procedures. The most commonly used procedures for data access are CURRENT, LEVEL, EDGE, and SEARCH.

CURRENT is used for real-time database access. The syntax is: `.CURRENT PointName, UnitName, LatestValue, Time, EngUnits, Status`

A sample of CURRENT code and output is shown in Figure 3. The PointName and UnitName specify the sensor and Data Acquisition System (DAS) where the sensor is located. The CURRENT procedure returns the value of the most recent sample of a point, along with the time at which the sample was taken. The engineering units of the point (e.g. DEG C, or PSI), and a status code (indicating success or failure) are also returned.

To read the value of a specific sensor from the historical database at a specific time, LEVEL is used. The syntax is `.LEVEL PointName, UnitName, Time, Value, EngUnits, Status`

A sample of LEVEL code and output is shown in Figure 4. The "Level" procedure returns the value of the sample at or before "time" of the point "pointname".

The EDGE procedure is used to search for many occurrences of a given point in a time range. A sample of EDGE code and output is shown in Figure 5. To begin the search the FIRST parameter is passed with the value TRUE. In subsequent searches it must be passed FALSE.

Edge returns only transitional values. This includes changes of state of logic signals and changes in value of other points. The syntax is `.EDGE PointName, UnitName, First, StartTime, EndTime, Value, Time, EngUnits, Status`.

The SEARCH procedure expands on the EDGE procedure by providing a means to search for many occurrences of any of several points on several units. A sample of SEARCH code and output is given in Figure 6. An array of point

names and an array of unit names is passed to the procedure. The point name and unit name are returned with the point value. The values are returned in ascending time order. The syntax is `.SEARCH PointList, UnitList, First, StartTime, EndTime, Value, Time, EngUnits, PointName, Unit, Status`

Menu-driven Interface

The Report Generator can be used either through a menu-driven interface, called the Toolkit, or via DCL commands.

The main menu of the Toolkit provides an interface to the EDT editor, RGL compiler and run time system, and the Report Viewer. The user selects options on the menu screens by moving the selection arrow (==>) to the desired option with the arrow keys and pressing RETURN or SELECT. The main menu is shown in Figure 7.

The Toolkit keeps track of what reports are available, and maintains a series of menus from which the user may select a report. The Toolkit remembers the last report selected, and will assume that the user wants to edit, compile, or run that report when he selects a menu option. If there is no current report as yet, then an option that requires a report name will put up the Report Selection Menu. The Report Selection Menu is shown in Figure 8.

Since the user actually is creating a multitude of files in his current directory, and since he may not be familiar with DCL, a Maintenance Menu is provided to help him manage his directory. The Maintenance Menu provides selections to purge or delete reports, and to list the directory. The Maintenance Menu also allows the user to check schedule status for periodically scheduled reports. The Maintenance Menu is shown in Figure 9.

Reports produce intermediate output files that may or may not be directly printable. The Report Viewer allows the user to preview his text or graphic output, and allows him to produce plots, printouts and screen copies of his report output.

Detailed, English Error Messages.

One very important design goal for the Report Generator was that its error and warning messages be as helpful as possible, and that they not be in cryptic computerese. It is assumed that the average user of the Report Generator will not be familiar with VMS error messages, and will need to be told not only what is wrong, but what to do about it.

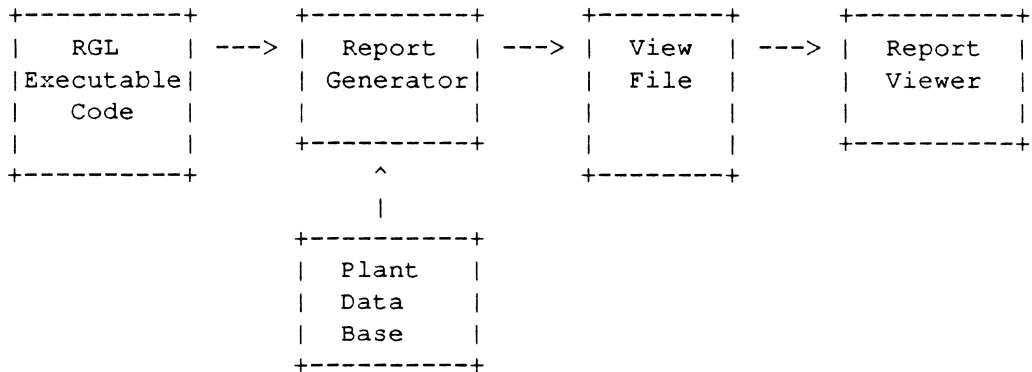


Figure 2: RGL Data Flow

```

.Current "VARS","GTG101",vars,time,units,status
.if status=1
.then
.display vars;" ";units;" at ";time
.else
.display "Error status=";status
.end

```

Output:

12.3 MVARs at 03-MAR-1987 09:13:59.02

Figure 3: CURRENT Procedure

Sample:

```

.Level "VARS","GTG101","02-Feb-1987 13:00:00.00",-
      vars,units,status
.if status=1
.then
.display vars;" ";units
.else
.display "Error status=";status
.end

```

Output:

12.3 MVARs

Figure 4: LEVEL Procedure

Sample:

```
.Set First=True
.Repeat
.Edge "VARS","GTG101",first,-
    "02-Feb-1987 13:00:00.00",-
    "02-Feb-1987 23:59:59.99", -
    vars,time,units,status
.set first=false
.if status=1
.then
    .display vars;" ";units;" at ";time
.end
.until status<>1
.if status<>3
.then
    .display "Error status=";status
.end
```

Output:

```
0.0 MVARs at 02-FEB-1987 13:00:00.00
0.5 MVARs at 02-FEB-1987 13:23:41.35
3.2 MVARs at 02-FEB-1987 17:28:01.27
1.0 MVARs at 02-FEB-1987 23:11:12:41
```

Figure 5: EDGE Procedure

Sample:

```
.table points(3) "L52GX","L4","L30B"  
.table units(4) "GTG101","GTG201","GTG301","GTG401"  
.Set First=True  
.Repeat  
  .search points,units,first,-  
    "02-Feb-1987 13:00:00.00",-  
    "02-Feb-1987 23:59:59.99", -  
    value,time,engunit,point,unit,status  
  .set first=false  
  .if status=1  
    .then  
      .display unit;":";point;"=";value;" ";engunit;  
      .display " at ";time  
    .end  
  .until status<>1  
  .if status<>3  
    .then  
      .display "Error status=";status  
    .end
```

Output:

```
GTG201:L52GX=1.0 at 02-FEB-1987 15:41:32.21  
GTG301:L30B=0.0 at 02-FEB-1987 17:00:32.99  
GTG101:L52GX=1.0 at 02-FEB-1987 24:11:37.17
```

Figure 6: SEARCH Procedure

+-----+
Report Development Toolkit

```
==> Create a new report  
      Edit the report source file  
      Compile the report  
      View the compilation log file  
      Test run the report  
      View the report source listing  
      View the report output  
      Switch the scheduling status  
      Maintenance Menu  
      Select a different report
```

Make a selection, then press return.
+-----+

Figure 7: Main Menu

```

+-----+
Report Development Toolkit

80COL          DD          LOCAL2          PNTSEARC
ADDT           DISPLAY    LONG          POWER
ALPHA          DUMMY      LYSHIFT     PRINT
ALPHA ==> ECHO          LYSHIFT2    PRTAB
ARGSORT        EDGE       MMIO         PU
ARITH          ELSE       MMIO2        READINT
ARITH2         ERR        MMIO3        REPEAT
ARRAY          EV         MMIO4        RETURN
ARUN           FACTORIAL  NESTED       RL2TM
CALL1          FOR         NESTED2      SDANAL
CALL2          FORWARD   NOT          SEARCH
CALL3          IF         NOT2         SET
CALLTAB        IF2        OPERAND      SHIFT
CAT            INC         PAREN       SINCOS
CONCAT         INCLUDE   PASSWRONG   STRING
CTIM           LOCAL     PERIODIC    SUANAL

Make a selection, then press return.                MORE...

```

Figure 8: Report Selection Menu

```

+-----+
Report Development Toolkit (Maintenance)

Purge a report
Purge all reports
Delete a report
==> List the Directory
Schedule status

Make a selection, then press return.

```

Figure 9: Maintenance Menu


```

Program Simplest(input,output);
var
  i:integer;
begin
  readln(i);
  if i>0
  thenn
    writeln('i=',i)
  else
    writeln('abs(i)=-i');
end.

```

Pascal produces the following error messages:

```

00007      0 1      thenn
                1
%Pascal-E-SYNTHEN, (1) Syntax: THEN expected
%Pascal-E-ENDDIAGS, Pascal completed with 1 diagnostic

```

Figure 11: Simple Pascal Program

```

.getkb str
.str2int str,i
.if i>0
.thenn
.print "i=",i
.else
.print "abs(i)=", -i
.end

```

Figure 12: Simple RGL Program

```

.THENN
!
An unresolvable problem was found on source line number 4.
Unknown directive or boolean operator, "THENN".

A DOT (.) was found, but it was not part of a real number or a
known keyword. Check and see if you misspelled a directive or
boolean operator (.AND, .OR, or .NOT).

.PRINT "i=",I
!
Source line number 5 may not produce the desired results.
.THEN expected.

The .IF directive has been found without a matching .THEN directive.
.THEN has two forms:
    .IF condition .THEN
and
    .IF condition
    .THEN
No statements may come in between the .IF and the .THEN.

RGL finished with 1 warnings, and 1 errors
No object file has been produced.

%DT_X00RPT-F-ABORT, report generator terminated abnormally

```

Figure 13: Error Messages

```

.REPEAT
statements
.UNTIL condition

.FOR variable = Start,End
statements
.NEXT

.IF condition .THEN
statements
.ELSE
statements
.END

.SUBROUTINE name parameters
statements
.END

.RETURN

.EXIT

```

Figure 14: RGL Program Structures


```

=====
Name:  $name                               $date
Addr:  $address1
      $address2

Monthly fuel bill:

Gas used:  $gasused $gasunits @ $gasprice = $$Gasbilled
Oil used:  $oilused $oilunits @ $oilprce  = $$Oilbilled

Total billable = $$billable
Pay by : $payduedate
=====

```

Figure 15: Unformatted Print Example

```

.ECHO 1 - Write to both display and
file.
.ECHO 2 - Write to the display only.

```

Forms may be drawn on the screen using ECHO and unformatted print statements, for example. RGL provides the keyboard input routines needed to handle simple forms.

Arihmetic Functions

RGL provides expression evaluation for a a small set of arithmetic operators and functions. These were created specifically to give the plant engineer the tools to write thermal performance calculations. Performance reports can be written using either real time data or historical data.

Binary operators:

```

+      Addition
-      Subtraction
*      Multiplication
/      Division
^      Exponentiation

```

Unary functions:

```

.SIN x  sine of x
.COS x  cosine of x
.TAN x  tangent of x
.ATAN x arctangent of x
.ABS x  absolute value of x
.LN x   natural log of x
.LOG x  base-10 log of x
.EXP x  e^x.
.SQRT x square root of x.

```

All angles are in radians.

Subroutine Capability

As RGL was used by engineers for more and more sophisticated applications, it became clear that RGL needed subroutine capability. Programs were becoming longer and longer, and typically contained repetitive code.

Subroutine capability was added in the last major release of RGL.

RGL supports subroutines with strongly-typed arguments, and with local variables stored on the stack to permit recursion.

Since a subroutine's name and argument list must be declared before it can be used, RGL has a FORWARD statement. This allows a subroutine's body to be defined separately from the declaration of its name and arguments.

RGL's subroutine syntax is somewhat of a cross between FORTRAN and Pascal, but without the parenthesis.

Figure 15 shows the .FORWARD construct, while Figure 16 shows recursion.

Conclusions

The Report Generation Language (RGL) has been very well received by its customers. Easy access to plant data using only a few function calls has encouraged experienced programmers to use the language, rather than learn dozens of complicated subroutines in Pascal or Fortran. The menu driven Toolkit and English error messages have made the language accessible to engineers and others who are not trained programmers. The use of structured flow control statements results in code that is easy to read and maintain. Powerful output formatting constructs, like unformatted print and printusing, allow users to compose the output directly on their screens.

RGL provides an easy, understandable, and cost-effective way to produce plant reports. This approach has made it successful.

```
.SUBROUTINE A INT:INTEGER
.FORWARD

.SUBROUTINE B
  .FOR I=1,10
    .CALL A I
  .NEXT I
.END

.SUBROUTINE A
  .PRINT INT
.END

.CALL B
```

Figure 16: Example of FORWARD Construct

```

.SUBROUTINE FACTORIAL X:INTEGER,Z:INTEGER
!
! X IS THE INPUT AND Z IS THE OUTPUT
!
.IF X>1
  .THEN
    .CALL FACTORIAL X-1,Z
    .SET Z=Z*X
  .ELSE
    .SET Z=1
  .END
.END ! of FACTORIAL

.SUBROUTINE FACT I:INTEGER
!
! ANOTHER WAY TO WRITE FACTORIAL
!
.LOCAL J:INTEGER
.IF I>1
  .THEN
    .SET J=I-1
    .CALL FACT J
    .SET I=I*J
  .ELSE
    .SET I=1
  .END
.END ! of FACT

! here is the main body of the program
.set i=5
.call fact i
.print "Fact 5 = ";i

.set fact = 0
.call fact fact
.print "fact 0 =";fact

.call factorial 5,i
.print "Factorial 5=";i

.set zero=0
.call factorial 0,zero
.print "factorial 0=";zero

```

Figure 17: An Example of Recursion

SPATIAL/II - A TECHNICAL OVERVIEW

Mark L. Palmer
Digital Equipment Corporation
Marlboro, Massachusetts

ABSTRACT

This paper briefly discusses what a spatial database is and is used for, and describes the components of Digital's Spatial Database product, Spatial/II.

SPATIAL DATA

Spatial data is any information which requires specification of locality in a dimensional frame of reference. In particular, it is information which represents entities existing in 3 dimensions.

Spatial data may exist implicitly in a collection of data the purpose of which does not require that the spatial elements be managed per se. An example of this might be a customer database application which stores customers' street addresses along with their financial figures and order information.

A large and growing number of applications, however, need to perform operations on the geometric, or spatial, entities "behind" the other data they manage. In the above example, the "address" data provided would not be enough to write a program which finds possible delivery routes connecting a group of customers within a given area. By making the spatial data associated with the street addresses explicit in terms of coordinates, representing certain spatial relationships, and performing operations on this data, the problem could be addressed.

Some application areas with needs to manage spatial data explicitly are:

- o Astronomy
- o Automated Cartography
- o Robot Vision
- o Geological Exploration
- o Facilities Management
- o Molecular Modelling
- o Automated Navigation
- o Land/Geographic Information Systems

Much progress has been made towards isolating the set of operations required and common to all spatial entities, and providing them in systems designed to manage spatial data explicitly. These operations apply and are useful regardless of whether the spatial entities in question represent railroads or molecular structures.

SPATIAL DATABASES

Current data and database models are proving inadequate for managing spatial data as needed by the above-mentioned application areas. Here are some reasons why:

1. Size - A typical spatial application uses tens of gigabytes of data. Current database implementations aren't built to handle these amounts, or if they do, access based on locational keys is too slow to provide interactive response.

2. Graphics - Management and query of spatial data is inherently a graphic operation. The ability to work with pictures of geometric entities is essential. Traditional database models don't provide utilities which allow graphic plotting and manipulation of their contents based on coordinates. Systems which produce images from the data and store them separately are not adequate, since the images don't reflect successive changes to the data and become incorrect.

3. Cost of data capture - Obtaining spatial data is expensive. It is labor-intensive (e.g. digitizing) or requires expensive technology (e.g. remote sensing). Traditional data models make it difficult to preserve this investment. They provide no means of isolating spatial data for use with different sets of thematic data.

4. Storage utilization - Spatial entities are composed of widely varying amounts of coordinates which are needed in queries. Most traditional models require definition of fixed numbers of numeric attributes.

5. Spatial Operations - Certain operations are essential to query and manage spatial data and occur so frequently that they must be fast.

These operations in traditional data model implementations are too slow or are not feasible within the model itself. Some examples:

- o recognizing entities partially or fully inside of others
- o finding line segment intersections over large areas
- o recognizing, representing, and using connectivity between entities
- o neighbor finding
- o quick, locationally-based search for entities.

SPATIAL/II

Digital's Spatial/II product is intended for use as a standard for representing and accessing spatial data, providing diverse applications utilizing spatial data a common base on which to build.

The product consists of a file structure, a callable interface to routines which manipulate the file structure, a library of geographic and cartographic routines for use in processing the data, and a set of utilities which allow generic spatial data manipulation and management.

Data Pool

Each Spatial/II file has a set of components which work together to provide access to the data:

- o Header
- o Tables
- o Index
- o Data Dictionary

Means of manipulating (creating, modifying, and deleting) each component are available both via the callable interface and also interactively, via use of the utilities.

Header

The "Header" component provides storage for information about the file as a whole: its bounding figures, creation date, size, topic, password, etc. Part of the header is maintained by Spatial/II, the other part is available for user-defined purposes.

Tables

The "Tables" component allows users to store formatted "matrices" of information which are used in

processing their data. A table may define what graphic symbols are to be used when displaying a file, or the format for displaying records as text. It may also be used to store "filters", which are ways of specifying spatial and attribute constraints for a set of records to express queries. Tables may be transferred between files.

Index

The "Index" component allows storage of indices, which are lists of record numbers. Users may generate and use indices in many ways, for example to mark the subset of records which satisfy a given filter in order to access them quickly in the future, or to provide multiple orderings for different types of spatial processing on a file.

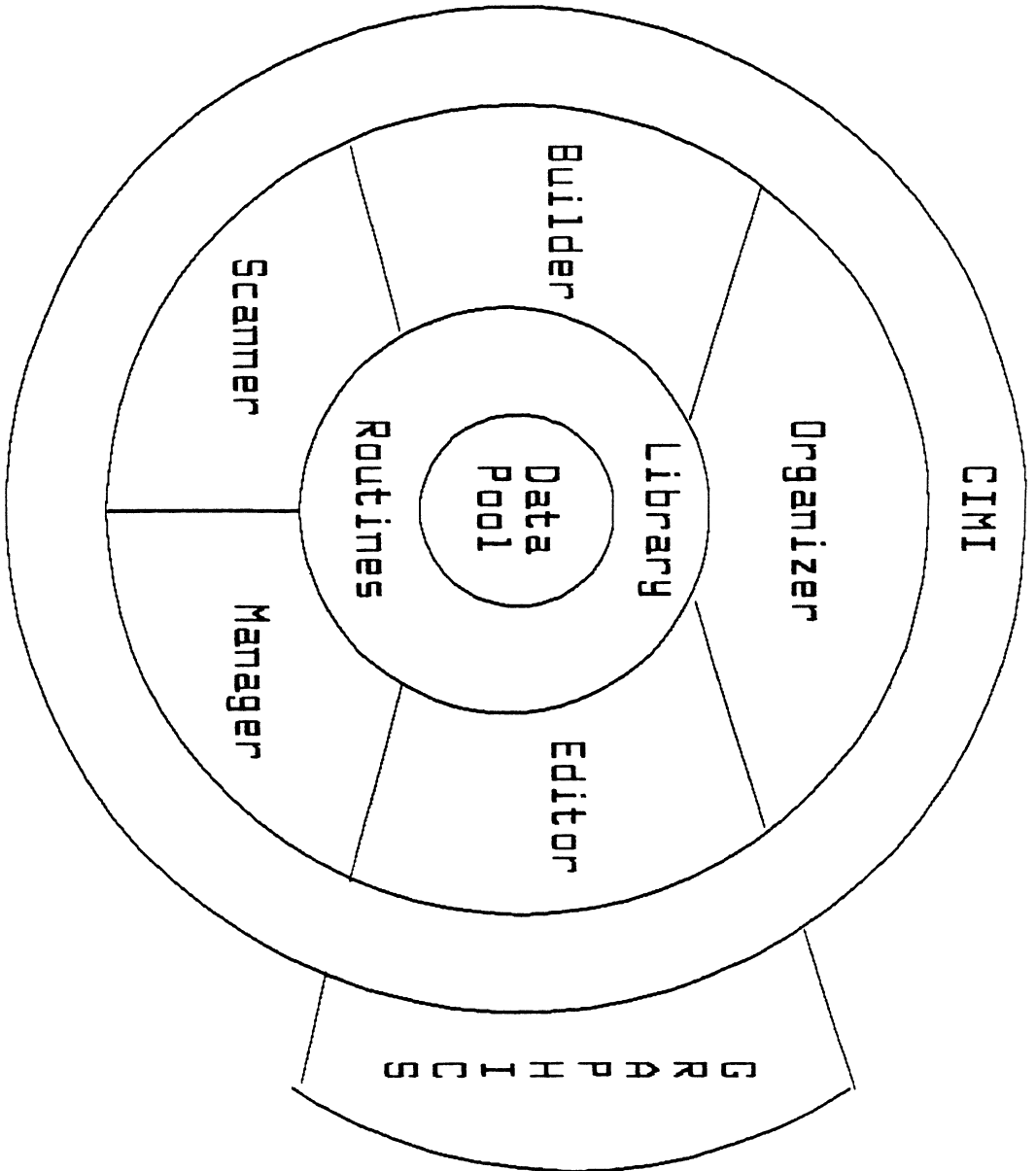
Data Dictionary

The "Data Dictionary" component keeps track of the organization of attributes on each record. User definitions created to extend the information kept about the spatial data are stored here.

A distinction is made between Locational, Topographic, and Thematic attributes. Locational attributes are those which specify position of an entity (these are usually coordinates or quantities). Topographic attributes consist of pointers to other records which have specific spatial relationships (e.g. "parent" and "neighbor" relationships). Thematic attributes are those which provide information about what a spatial entity represents (e.g. street name, oil well output, etc.).

Locational and Topographic attributes are defined and managed by the Spatial/II software.

Spatial/II



Thematic attributes may be defined by the user. The data dictionary allows synonyms, or "aliases", for attribute names to be created; it also supports multiple versions of an attribute within a record.

The last and largest component of a Spatial file is the data itself, which the other components exist to assist in manipulating. This is the set of actual records with Locational, Topographic, and Thematic attributes.

Spatial/II file structure employs a vector-type representation. Topographic operations are especially facilitated by using VMS indexed files for direct access - records keep "pointers" to topographically related records by using actual record numbers. Locational attributes need not be continually processed to guide access since topographic relationships are explicitly represented. In most cases, Topographic information may be automatically generated and is preserved by the operation of the various utilities.

Data Entities and Structure

A Spatial/II file may contain one of the following data types (listed in order of increasing complexity):

- o Point
- o Resel
- o Chain
- o Polygon
- o Triangulated Irregular Network (TIN)

Points are simply X,Y, and Z coordinate triples with which Thematic attributes may be associated. Points are typically used to represent objects such as oil wells and telephone poles.

A Resel is a rectangular plane defined within a grid coordinate system. The Resels in a file completely cover all grid cells in the coordinate system. Each resel is defined by lower left and upper right corners. Resels are useful for data in "map sheets" format, for example to represent areas of different ground elevation.

Each Chain is a set of coordinates which have an order and a direction. At least 2 Points (start and end) are needed; intermediate points are called "Detail Points". Topographically, Chains may have parents and may share end points. Locational attributes such as width and length are provided. Chains are useful for representing land features such as roads and rivers, or for molecular structures.

A Polygon is a set of Chains which may share endpoints but do not otherwise intersect. Topographic attributes include: parent, child, and right and left neighbor. Locational attributes include type (convex or concave), center, area, status (open or closed) etc.

Polygons are often used in cadastral maps to identify land parcels.

TIN files are typically used in 3D modelling of land surfaces such as mountains or multiple layers of geologic substratum. A Triangle is a plane in three dimensions enclosed by three Chains. Locational attributes of Triangles include centroid, area, slope, and direction. Some Topographic attributes are: parent, child, vertices (points), neighbor, and nesting level.

The primary entity represented by a file is made of a hierarchy of components which are usually present as data. Polygons are made of chains, which are made of nodes. Triangles in a TIN may be grouped into polygons; the sides of a Triangle are Chains. Corresponding entities between data types are homologous, i.e. Chain records found in Triangle, Chain, and Polygon files have common attributes.

Each entity and its components has a set of system-defined attributes which are maintained by the utilities supplied; further attributes may be added by users, extending the record structure. This is why Chains can be used to represent rivers as well as chromosomes. A standard attribute naming scheme is used, so the only difference between attributes across file types is in the names' prefix.

CALLABLE INTERFACE

The callable interface allows programs to access and manipulate records which make up spatial entities, as well as providing a library of geometric and cartographic procedures which operate on the records. The routine library supplied is used to by the utilities, which allow users to manipulate data interactively.

Each component of Spatial/II has a group of procedures which provide the set of operations required to manipulate that component. Procedures are supplied which create, delete, and modify entries in the data dictionary, indices, tables, and file management components.

Since the Spatial/II routine library is written in FORTRAN, it is very easy to call from VAX FORTRAN. The documentation gives examples of how these calls are made from VAX FORTRAN.

UTILITIES SUPPLIED

All of the utilities interact with the user via a common user interface, called the Command, Input, and Messaging Interface (CIMI).

The menu structure associated with controlling any utility is non-hierarchical. A function may be executed by typing almost any abbreviation of its name from "anywhere" in the menu. Most functions are named with two short words which are unique enough that they can usually be abbreviated using two letters.

The menu structure is also easily changeable by users interactively. A user may, for example, rename a function to his liking and relocate the function to a place in the menu tree which is more suited to his purposes without exiting the utility.

Manager

The Manager provides for manipulation of the data at the "file" level, and management of file access to various users. info kept about each user, or "owner", and each file, including passwords for both owners and files.

The utility allows file creation and deletion, modification of information associated with each file, as well as "import" of data into the system. User accounts may be similarly manipulated.

Builder

Builder allows "fleshing out" the topographical information in a file by analyzing existing info to determine relationships which are then stored. For example, Builder allows generating polygons from chains.

Organizer

This allows manipulation of spatial data on the file level while preserving topographic data in the file. Organizer allows files to be split apart and later rejoined, and for previously unrelated files to be merged.

Editor

The Editor provides access to individual files on a record basis and is the most complex utility. It allows users to graphically display and manipulate their spatial data.

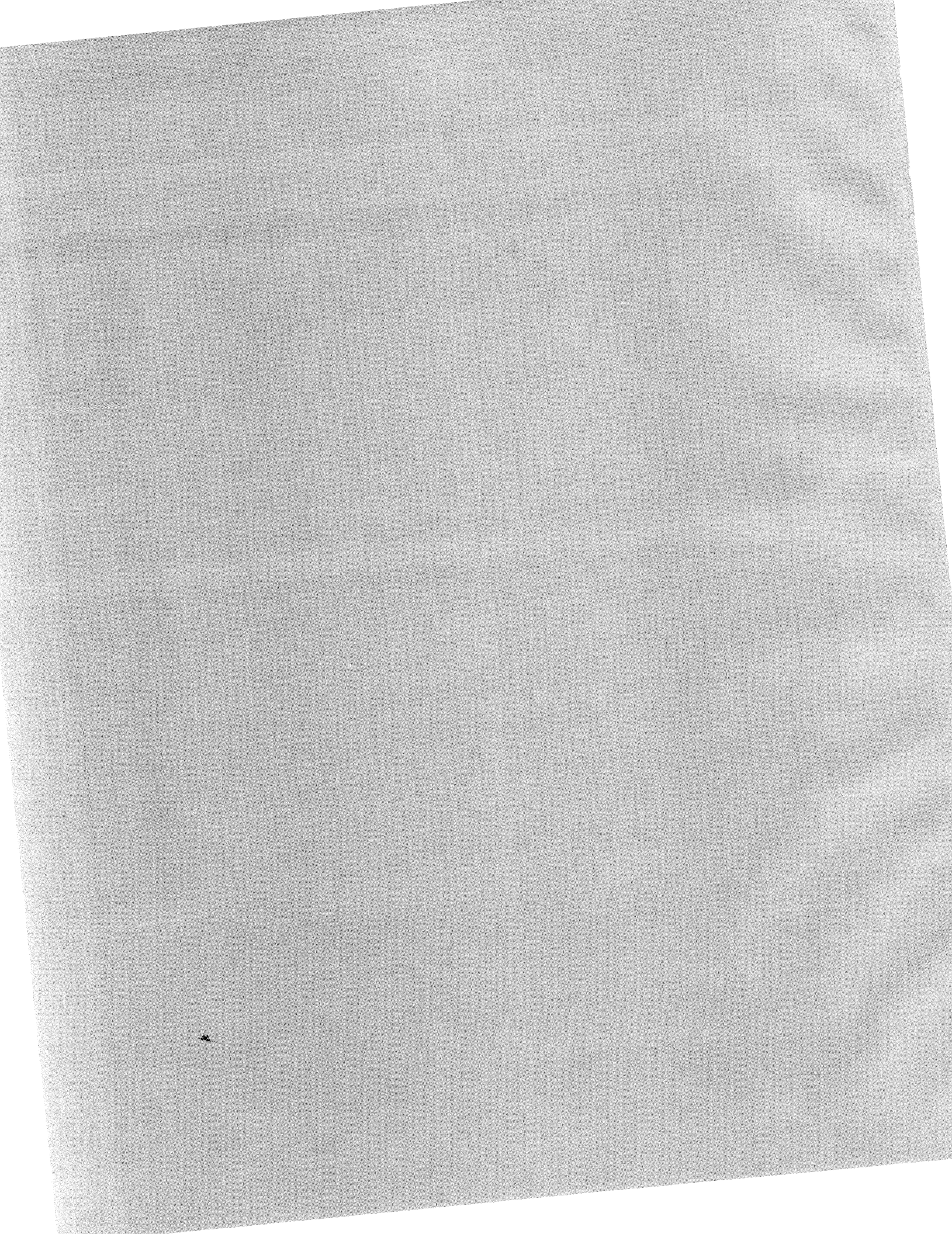
Each component of a Spatial/II file is accessible:

- o Tables
- o Indices
- o Data Dictionary
- o Header
- o graphic data editing (add, delete, modify)

By defining and activating "filters" (part of the tables component) spatial queries may be conducted and their results made available graphically. Also, the display arrangements for graphic and textual information to specify labelling, coloring, symbol use, etc. in accordance with ranges and limits set up using the Editor.

Scanner

The Scanner is used to detect locational and topographic errors in the data, and also derives geometric locational data (areas, bounds) by processing existing locational info (eg bounding poly, rect, flow, etc), and adds this information to the other spatial information already stored.



Advanced DATATRIEVE Record Definitions

B.Z.Lederman
ITT World Communications
New York, NY 10004-2464

Abstract

This session is intended to illustrate some of the more advanced features of DATATRIEVE record definitions. Lower case text indicates commands typed in by a user, upper case is printed by DTR or is material stored in the CDD. Please keep in mind that most examples are "stripped down", showing only the fields necessary to illustrate the principles being demonstrated: "real applications would require additional fields, and in most cases more descriptive field names. Most of these examples use advanced features found in VAX-DTR and DTR-20, and unfortunately will not work in DTR-11 (or PRO-DTR).

Introduction

Reading a file whose records differ in length and field layout is a common problem. In the following sample file, there are records whose total length is not given directly by a field in the record.

```
$ type var.seq

01 10 bytes.
02 15 byte record
01 10 bytes.
03 This is 20 bytes...
02 15 byte record
04 This is 25 bytes long...
01 10 bytes.
03 This is 20 bytes...
02 15 byte record
01 10 bytes.
04 This is 25 bytes long...
01 10 bytes.
```

You can just define a text field the length of the longest record, but you get "Record too Short..." error messages, and the short records are padded with blanks or zeroes. Also, it would be hard to look at the individual data items within each record. A first try at a better record definition could be:

```
DTR> show var_rec

RECORD VAR_REC
01 VAR_REC.
  10 TYPE PIC 99 EDIT_STRING Z9.
  10 REC_LEN COMPUTED BY
    TYPE VIA VAR_LEN_TAB.
  10 TOP.
  15 VARIABLE OCCURS 0 TO 30 TIMES
```

```
DEPENDING ON REC_LEN.
  20 VTEXT PIC X.
10 A REDEFINES TOP.
  20 FILLER PIC X.
  20 NBRA PIC 99 EDIT_STRING Z9.
  20 FILLER PIC X.
  20 TXTA PIC X(6).
10 B REDEFINES TOP.
  20 FILLER PIC X.
  20 NBRB PIC 99 EDIT_STRING Z9.
  20 FILLER PIC X.
  20 TXTB PIC X(11).
10 C REDEFINES TOP.
  20 FILLER PIC X.
  20 TXTC1 PIC X(7).
  20 FILLER PIC X.
  20 NBRC PIC 99 EDIT_STRING Z9.
  20 FILLER PIC X.
  20 TXTC2 PIC X(8).
10 D REDEFINES TOP.
  20 FILLER PIC X.
  20 TXTD1 PIC X(7).
  20 FILLER PIC X.
  20 NBRD PIC 99 EDIT_STRING Z9.
  20 FILLER PIC X.
  20 TXTD2 PIC X(13).
```

;

This record definition depends upon a table that converts the record type to a record length. This happens to be in a domain table in this example, but could also be in a dictionary table.

```
DTR> show var_tab_rec

RECORD VAR_TAB_REC
01 VAR_TAB_REC.
```

```

10 TYPE PIC 99 EDIT_STRING Z9.
10 LENGTH PIC 99 EDIT_STRING Z9.

```

```
;
```

```
DTR> show var_len_tab
```

```

TABLE VAR_LEN_TAB FROM VAR_TAB_DOM
USING TYPE : LENGTH
END_TABLE

```

```
DTR> print var_tab_dom
```

```

TYPE LENGTH
1      10
2      15
3      20
4      25

```

If you print this domain, you get the first field by default.

```
DTR> print var
```

```

REC
TYPE LEN VTEXT
1  10
    1
    0
    b
    y
    t
    e
    s
    .
2  15
    1
    5
    b
    y
    t
    e
    r
    e
    c
    o
    r
    d
    .
    .
    .

```

and so on. This is very useful in cases where you want to get each character in the record separately, such as for “parsing” data, and you get the length of the text without having to add an FN\$STR.LENGTH function to DTR. However, if you want all of the data in a single field:

```
DTR> for var print a
```

```

NBRA  TXTA
10  bytes.
15  byte r
10  bytes.
Illegal ASCII numeric "Th".
0  s is 2
15  byte r
Illegal ASCII numeric "Th".
0  s is 2
10  bytes.
Illegal ASCII numeric "Th".
0  s is 2
15  byte r
10  bytes.
Illegal ASCII numeric "Th".
0  s is 2
10  bytes.

```

and the same happens for all other REDEFINED fields, because the numeric fields don’t “line up”. One alternative is to use a CHOICE statement in a procedure to get the proper field to print out. (You can also use IF-THEN-ELSE statements to accomplish the same result, and that approach will also work with DTR-11, but CHOICE is more compact.)

```
DTR> show var-print
```

```

PROCEDURE VAR_PRINT
FOR VAR BEGIN
    PRINT TYPE, CHOICE OF
        TYPE = 1 THEN A
        TYPE = 2 THEN B
        TYPE = 3 THEN C
        TYPE = 4 THEN D
        ELSE " "
    END_CHOICE
END
END_PROCEDURE

```

```
DTR> :var-print
```

```

TYPE
1  10 bytes.
2  15 byte record
1  10 bytes.
3  This is 20 bytes...
2  15 byte record
4  This is 25 bytes long...
1  10 bytes.
3  This is 20 bytes...
2  15 byte record
1  10 bytes.
4  This is 25 bytes long...

```

1 10 bytes.

This suits many applications, but is sometimes inconvenient. An alternative is a record definition (actually a VIEW) that will print out the proper fields by default. (See figure 1 following.)

This has the slight drawback that, since there is nothing which identifies unique records in this example, all records of a given type are obtained for each record in the view. In cases where there was an additional field with a unique key, this would not be a problem: in this case, however, some additional work can solve the problem. (See figure 2 following)

This is one way to one complete set of records. Another method is:

```
DTR> find vari
```

```
[12 records found]
```

The following is not quite what we want.

```
DTR> for current print av
```

```
NBRA  TXTA
```

```
10 bytes.
10 bytes.
10 bytes.
10 bytes.
10 bytes.
10 bytes.
10 bytes.
10 bytes.
10 bytes.
10 bytes.
10 bytes.
10 bytes.
```

Execution terminated by operator.

but you can do this:

```
DTR> select first
```

Now, you can do some interesting things, like separating the different groups of similar records.

```
DTR> print av
```

```
NBRA  TXTA
```

```
10 bytes.
10 bytes.
10 bytes.
10 bytes.
10 bytes.
```

```
DTR> print bv
```

```
NBRB  TXTB
```

```
15 byte record
15 byte record
15 byte record
```

```
DTR> print cv
```

```
TXTC1  NBRC  TXTC2
```

```
This is 20 bytes...
This is 20 bytes...
```

```
DTR> print dv
```

```
TXTD1  NBRD  TXTD2
```

```
This is 25 bytes long...
This is 25 bytes long...
```

Normally I would discourage the use of FIND and SELECT, but in this case it can be used to sort and separate all records of a given type.

Still, this is not quite what we were looking for. If you can put a CHOICE statement into a procedure, why not put it into the record definition.

```
DTR> show cvar_rec
```

```
RECORD CVAR_REC USING
```

```
01 CVAR_REC.
```

```
10 TYPE PIC 99 EDIT_STRING Z9.
```

```
10 REC_LEN COMPUTED BY TYPE VIA
VAR_LEN_TAB.
```

```
10 FILLER PIC X.
```

```
10 TOP.
```

```
15 VARIABLE OCCURS 0 TO 30
TIMES DEPENDING ON REC_LEN.
```

```
20 FILLER PIC X.
```

```
10 A REDEFINES TOP.
```

```
20 ANBR PIC 99.
```

```
20 FILLER PIC X.
```

```
20 ATXT PIC X(6).
```

```
10 B REDEFINES TOP.
```

```
20 BNBR PIC 99.
```

```
20 FILLER PIC X.
```

```
20 BTXT PIC X(11).
```

```
10 C REDEFINES TOP.
```

```
20 CTXT1 PIC X(8).
```

```
20 CNBR PIC 99.
```

```
20 CTXT2 PIC X(9).
```

```
10 D REDEFINES TOP.
```

```
20 DTXT1 PIC X(8).
```

```
20 DNBR PIC 99.
```

```
20 DTXT2 PIC X(14).
```

```
10 TEXT COMPUTED BY CHOICE OF
TYPE = 1 THEN ATXT
```

```

DOMAIN VARI OF VAR USING
01 VARIK OCCURS FOR VAR.
  10 TYPE FROM VAR.
  10 REC_LEN FROM VAR.
02 AV OCCURS FOR VAR WITH TYPE = 1.
  10 NBRA FROM VAR.
  10 TXTA FROM VAR.
02 BV OCCURS FOR VAR WITH TYPE = 2.
  10 NBRB FROM VAR.
  10 TXTB FROM VAR.
02 CV OCCURS FOR VAR WITH TYPE = 3.
  10 TXTC1 FROM VAR.
  10 NBRC FROM VAR.
  10 TXTC2 FROM VAR.
02 DV OCCURS FOR VAR WITH TYPE = 4.
  10 TXTD1 FROM VAR.
  10 NBRD FROM VAR.
  10 TXTD2 FROM VAR.
;

DTR> print vari

      REC
TYPE LEN NBRA  TXTA  NBRB   TXTB      TXTC1  NBRC  TXTC2   TXTD1  NBRD   TX
  1  10   10  bytes.  15  byte record This is  20  bytes... This is  25  bytes
      10  bytes.  15  byte record This is  20  bytes... This is  25  bytes
      10  bytes.  15  byte record
      10  bytes.
      10  bytes.
  2  15   10  bytes.  15  byte record This is  20  bytes... This is  25  bytes
      10  bytes.  15  byte record This is  20  bytes... This is  25  bytes
      10  bytes.  15  byte record
      10  bytes.
      10  bytes.
  1  10   10  bytes.  15  byte record This is  20  bytes... This is  25  bytes
      10  bytes.  15  byte record This is  20  bytes... This is  25  bytes
      10  bytes.  15  byte record
      10  bytes.
      10  bytes.

```

Execution terminated by operator.

Figure 1: Example


```
DTR> show print-first-vari
```

```
PROCEDURE PRINT_FIRST_VARI
DECLARE N PIC 9.
N = 1
FOR VARI BEGIN
    WHILE N = 1 BEGIN
        N = N + 1
        PRINT
    END
END
END_PROCEDURE
```

```
DTR> :print-first-vari
```

REC	TYPE	LEN	NBRA	TXTA	NBRB	TXTB	TXTC1	NBRC	TXTC2	TXTD1	NBRD	TX
1	10	10	bytes.	15	byte record	This is	20	bytes...	This is	25	bytes	
		10	bytes.	15	byte record	This is	20	bytes...	This is	25	bytes	
		10	bytes.	15	byte record							
		10	bytes.									
		10	bytes.									

Figure 2: Example

```

TYPE = 2 THEN BTXT
TYPE = 3 THEN CTXT1 ||| CTXT2
TYPE = 4 THEN DTXT1 ||| DTXT2
ELSE ""
END_CHOICE.
10 NUMBER EDIT_STRING Z9 COMPUTED
BY CHOICE OF
TYPE = 1 THEN ANBR
TYPE = 2 THEN BNBR
TYPE = 3 THEN CNBR
TYPE = 4 THEN DNBR
ELSE 0
END_CHOICE.
;
DTR> for cvar print fn$log10(number)
1.000
1.176
1.000
1.301
1.176
1.398
1.000
1.301
1.176
1.000
1.398
1.000

```

```
DTR> print cvar
```

REC	TYPE	LEN	TEXT	NUMBER
1	10	bytes.		10
2	15	byte record		15
1	10	bytes.		10
3	20	This is bytes...		20
2	15	byte record		15
4	25	This is bytes long...		25
1	10	bytes.		10
3	20	This is bytes...		20
2	15	byte record		15
1	10	bytes.		10
4	25	This is bytes long...		25
1	10	bytes.		10

Just to prove that NUMBER is really numeric

Now we finally have the data in the form we want. Something which is not visible when you look at this print-out on paper is that the field TEXT always prints out the length of the actual field: it does not pad short records with spaces or zeroes which is what would happen if you just defined one field of 25 bytes (you also don't get the "Record too Short" error messages).

There are a number of applications where data validation in the record definition is desired. In this example, the employee number contains a sort of "check sum", where the last two digits are the sum of the first two. This sort of thing is sometimes done to verify that the data does not contains errors (I'd rather depend on the operating system facilities, but some people would prefer this). This particular check sum is a bit crude, and done only to demonstrate the methods which may be used. If you were going to do this a lot, it would be worthwhile to define a new FN\$— function to do the computation, especially if the check method was more complicated

such as some sort of “rule of 11”, but not everyone wants to add functions to DTR. The interesting part of all this is that you can define a VALID IF clause to work on parts of the same field it validates, and that the fields used can be defined after the VALID IF clause.

```
DTR>show empno_rec
```

```
RECORD EMPNO_REC
01 EMPNO_REC.
  10 EMPLOYEE_NUMBER PIC 99999
    VALID IF CK = (N1 + N2 + N3).
  10 NBRS REDEFINES EMPLOYEE_NUMBER.
    20 N1 PIC 9.
    20 N2 PIC 9.
    20 N3 PIC 9.
    20 CK PIC 99.
;
```

```
DTR> print empno
```

```
EMPLOYEE
NUMBER

12306
65617
98724
11002
00101
32308
```

Something that users don’t always realize is that a COMPUTED BY field can be anywhere in the record definition, and does not have to be computed from fields that come “ahead” of it in the definition. DTR will read and parse the entire record definition to resolve all field names before doing anything with the record: thus, a field can, in some cases, even be computed by itself.

With this definition, you can prevent invalid numbers from being stored.

```
DTR> store empno
Enter EMPLOYEE_NUMBER: 32301
```

```
Validation error for EMPLOYEE_NUMBER.
Re-enter EMPLOYEE_NUMBER: 32308
```

You can also find out if all the numbers currently in the domain are still valid (something which a normal VALID IF won’t do):

```
DTR> for empno print ck, (n1 + n2 +
n3)

CK
```

```
06 6
17 17
24 24
02 2
01 1
08 8
08 8
```

Now look at what happens if an invalid number is present in the domain.

```
DTR> print empno
```

```
EMPLOYEE
NUMBER

12306
65617
98724
11002
00101
32301 [this number is invalid]
```

```
DTR> print empno with ck ne (n1+n2+n3)
```

```
EMPLOYEE
NUMBER

32301
```

We can use DTR to go in and fix any checksums. (I would advise looking at the data first to be certain it really is valid, unless you want to do something like this to add checksums to data that was stored previously without checksums.)

```
DTR> ready empno modify
DTR> for empno with ck ne (n1+n2+n3)
begin
CON> modify empno using ck = n1+n2+n3
CON> end
```

```
DTR> print empno
```

```
EMPLOYEE
NUMBER

12306
65617
98724
11002
00101
32308
```

While thinking up stuff for this presentation, I came up with the following example which, quite frankly, I didn’t think would work.

```
DTR> show sci_rec

RECORD SCI_REC
01 SCI_REC.
  10 SCI_NOT USAGE REAL
    EDIT_STRING 99.99.
  10 N2 COMPUTED BY *."N2".
;
```

Depending upon how you access the domain, you can be prompted for N2 once per record (might be used to make the system pause during loops), once per domain, or not at all.

```
DTR> for sci print sci_not

SCI_NOT

00.01
00.88
01.20
09.80
23.40
```

```
DTR> print sci
Enter N2: 30

SCI_NOT  N2

00.01 30
00.88 30
01.20 30
09.80 30
23.40 30
```

```
DTR> for sci print sci_rec

SCI
NOT      N2

Enter N2: 30

00.01 30
Enter N2: 20

00.88 20
Enter N2: 10

01.20 10
Enter N2: 1

09.80 1
Enter N2: 0

23.40 0
```

Having done this, I'm not at all sure what I would use it for, but one possibility might be to calculate prices from a

stored price list where the discount might change for different customers, or where you might have to convert prices into foreign currencies (where the exchange rate changes daily). The factor by which the prices are multiplied could be entered as the prompted field, and then this field can be used to multiply the stored price into a COMPUTED BY field with the net price.

Some COMPUTED BY fields are more useful than others. For example, if several departments share a data base and you want to make sure that each department enters the correct sequence of numbers (this example assumes a valid range of numbers for each department, just to make it more difficult):

```
DTR> show po_rec

RECORD PO_REC
01 PO_REC.
  10 DEPT PIC XXX.
  10 PO_NUMBER PIC 99999 VALID IF 1 = CHOICE OF
    (DEPT = "AAA" AND PO_CHECK BETWEEN 01 AND
20) THEN 1;
    (DEPT = "BBB" AND PO_CHECK BETWEEN 21 AND
40) THEN 1;
    (DEPT = "CCC" AND PO_CHECK BETWEEN 41 AND
60) THEN 1;
    ELSE 0
  END_CHOICE.
  10 PO_CHECK REDEFINES PO_NUMBER.
  20 DEPT_NO PIC 99.
;
```

```
DTR> print po

      PO
DEPT NUMBER

AAA  01001
BBB  21001
```

```
DTR> store po
Enter DEPT: AAA
Enter PO_NUMBER: 01002
DTR> store po
Enter DEPT: BBB
Enter PO_NUMBER: 01003
```

```
Validation error for field PO_NUMBER.
Re-enter PO_NUMBER: 21002
```

This isn't bad, but it could be better. Why store the department number and verify it, when you could change the record definition and force it to always be correct? (This time I'm assuming one prefix per department.)

```
DTR> show po_rec

RECORD PO_REC
01 PO_REC.
  10 DEPT PIC XXX VALID IF DEPT = "AAA",
"BBB", "CCC".
  10 HIDEIT.
  20 FILLER PIC 999.
  10 REAL_STUFF REDEFINES HIDEIT.
  20 DEPT_SEQ PIC 999.
  10 PO_NUMBER PIC 99999 COMPUTED BY CHOICE OF
    DEPT = "AAA" THEN DEPT_SEQ + 01000
    DEPT = "BBB" THEN DEPT_SEQ + 02000
    DEPT = "CCC" THEN DEPT_SEQ + 03000
    ELSE "00000"
  END_CHOICE.
```

We can also force the sequence number to be correct.

```
DTR> show store-po

PROCEDURE STORE_PO
DECLARE MAXSEQ PIC 999.
DECLARE TMPDEP PIC XXX.
TMPDEP = FN$UPCASE(*."Department")
MAXSEQ = MAX(DEPT_SEQ) OF PO
        WITH DEPT = TMPDEP
STORE PO USING BEGIN
    DEPT = TMPDEP
    DEPT_SEQ = MAXSEQ + 1
END
END_PROCEDURE
```

```
DTR> :store-po
Enter Department: bbb
DTR> print po
```

```
      PO
DEPT NUMBER

BBB 02001
BBB 02002
AAA 01001
BBB 02003
```

```
DTR> :store-po
Enter Department: bbb
DTR> print po
```

```
      PO
DEPT NUMBER

BBB 02001
BBB 02002
AAA 01001
BBB 02003
BBB 02004
```

If you try to store a department which has no records yet, you get an error message, but you also get the correct result anyway:

```
DTR> :store-po
Enter Department: ccc

Can't take MAX,MIN,or AVERAGE of
zero objects.
DTR> print po

      PO
DEPT NUMBER

BBB 02001
```

```
BBB 02002
AAA 01001
BBB 02003
BBB 02004
CCC 03001
```

Something which I have run into, and which others have asked for at past Q&A sessions, is how to get non-VMS date strings into the VMS/DTR date type, especially when you are not able to restructure the data. The following very non-standard date and time is the type of data I've actually encountered.

```
$ type date.seq
86:01:02 1003A
85:03:14 120P
86:09:29 1100P
86:11:11 332A

DTR> show date_rec

RECORD DATE_REC
01 DATE_REC.
    10 INPUT.
        20 I_YEAR PIC 99.
        20 FILLER PIC X.
        20 I_MONTH PIC XX.
        20 FILLER PIC X.
        20 I_DAY PIC 99.
        20 FILLER PIC X.
        20 T_HOUR PIC XX.
        20 I_HOUR PIC 99 COMPUTED
            BY T_HOUR.
        20 I_MINUIT PIC 99.
        20 I_AP PIC X.
    10 O_DATE COMPUTED BY
        FN$DATE(I_DAY | "-" |
            I_MONTH VIA MONTH_TABLE |
            "-19" | I_YEAR) .
;
```

The date part is easy: you just need a table to turn the numeric month into an upper case alphanumeric month.

```
DTR> show month_table

TABLE MONTH_TABLE
01 : "JAN",
02 : "FEB",
03 : "MAR",
04 : "APR",
05 : "MAY",
06 : "JUN",
07 : "JUL",
08 : "AUG",
09 : "SEP",
10 : "OCT",
11 : "NOV",
12 : "DEC"
```

END_TABLE

DTR> print datei

I	I	I	T	I	I	I	O
YEAR	MONTH	DAY	HOUR	HOUR	MINUIT	AP	DATE
86	01	02	10	10	03	A	2-Jan-1986
85	03	14	1	01	20	P	14-Mar-1985
86	09	29	11	11	00	P	29-Sep-1986
86	11	11	3	03	32	A	11-Nov-1986

Not too bad: but when you have to add the time things get a little bit more complicated. I've shown only the hour and minuit here, but you can add seconds and fractions of a second as well. Note that I'm also using FILLER to hide the input fields, so by default only the wanted fields print.

DTR> show date_rec

```
RECORD DATE_REC
01 DATE_REC.
 10 HIDEIT.
   20 FILLER PIC X(14).
 10 INPUT REDEFINES HIDEIT.
   20 I_YEAR PIC 99.
   20 FILLER PIC X.
   20 I_MONTH PIC XX.
   20 FILLER PIC X.
   20 I_DAY PIC 99.
   20 FILLER PIC X.
   20 T_HOUR PIC XX.
   20 I_HOUR PIC 99 COMPUTED BY
      T_HOUR.
   20 I_MINUIT PIC 99.
   20 I_AP PIC X.
   20 A_HOUR COMPUTED BY CHOICE OF
      (I_AP = "A" AND T_HOUR = 12)
      THEN 00
      (I_AP = "P" AND T_HOUR < 12)
      THEN T_HOUR + 12
      ELSE T_HOUR
      END_CHOICE.
   20 B_TIME COMPUTED BY
      ((A_HOUR * 60) + I_MINUIT) *
      600000000.
 10 O_DATE COMPUTED BY
      FN$DATE(I_DAY | "-" |
      I_MONTH VIA MONTH_TABLE |
      "-19" | I_YEAR |||
      FN$TIME(B_TIME)).
;
```

The hard part is converting the AM/PM time to a 24 hour time, then getting it to print in the proper format. There are a number of ways it might be done depending upon the exact input format: in this case I convert the hour and minute to "clunks", then use FN\$TIME to put it back to characters long enough to use FN\$DATE to put the date and time back to

clunks. This might seem a bit "clunky", but it's actually the easiest way to get it to work every time. The alternative is to make all of the fields "print" in the FN\$DATE function the way the day and year do. (It is sometimes also possible to do this sort of thing in DTR-11: though there are no FN\$—functions, DTR-11 will handle dates with embedded times in clunks.)

DTR> print datei

O_DATE

2-Jan-1986
14-Mar-1985
29-Sep-1986
11-Nov-1986
7-Aug-1986
4-Jul-1976

DTR> for datei print i_hour, i_minuit,
CON> i_ap, fn\$time(o_date)

I	I	I	
HOUR	MINUIT	AP	FN\$TIME
10	03	A	10:03:00.00
01	20	P	13:20:00.00
11	00	P	23:00:00.00
03	32	A	03:32:00.00
12	01	A	00:01:00.00
12	58	P	12:58:00.00

The net result is that O_DATE now contains the complete date and time in VMS format, and all of the normal DTR Boolean comparisons will work.

Solving Equations in DATATRIEVE

B.Z.Lederman

ITT World Communications
New York, NY 10004-2464

Abstract

This paper highlights some of the methods of solving equations by using the mathematical, logical and statistical functions available in DATATRIEVE. This paper will not attempt to teach equation solving, but will highlight the facilities available in DATATRIEVE, demonstrate some approaches to solving problems, and will point out some of the difficulties or limitations to the process.

Why?

The first question many people will ask is: "Why would anyone want to solve mathematical equations in DATATRIEVE? Isn't DATATRIEVE a data retrieval and reporting language?" The answer, briefly, is that there may be applications which are totally non-mathematical (perhaps a library card catalog) that would be implemented in DATATRIEVE with no math functions; and some totally numerical applications (such as a computer controlled milling machine or Fourier analysis of a video image) that would probably not be implemented in DATATRIEVE; but there are no sharp boundaries between data retrieval which is totally non-mathematical and data retrieval which requires some math. It is perfectly reasonable to implement applications which primarily store and retrieve data in DATATRIEVE which also require some math, such as inventory control, accounting, payroll, and probably as many other applications as there are people using DATATRIEVE, and it may be easier to implement the entire application in DATATRIEVE than to do some pieces in DATATRIEVE and other pieces in some other language. There is also the practical consideration that many people who are able to quickly learn and use DATATRIEVE do not have any "traditional" programming background, and may not have access to other programming resources and are faced with the prospect of doing it entirely in DATATRIEVE or not doing it at all.

One Alternative: Callable DATATRIEVE

There is an alternative for programs which require a large amount of math but for which you would still like to use DATATRIEVE for data storage, retrieval, and reporting, and that is to use the call interface. You can write a program in most, if not all, VAX languages (and with some limitations, PDP-11 and TOPS) and have it call DATATRIEVE: this allows you to write the math portion in your favorite language, and then have it pass data to or retrieve data from DATATRIEVE and execute DATATRIEVE statements from within your program. It is even possible for the DATATRIEVE task

to be running on a separate system linked to yours via DECnet, which is often advantageous. Solving equations in other languages is outside the scope of this presentation, however, and this is mentioned here simply to inform you about some of the alternatives available.

Basic Requirements

DATATRIEVE has all of the basic requirements for solving mathematical or logical equations, which are:

- Mathematical Operators, such as addition, subtraction, multiplication, and division
- The ability to control the flow of calculations by logical (Boolean) operators (IF-THEN-ELSE).
- The ability to repeat an action until a condition is met (FOR and WHILE).

While this may not seem to be a very large repertoire, it is enough to solve almost any equation: it is, in fact, all that any computer has, or what any person would have if the equation were to be solved by hand.

Other Functions

"Traditional" computer languages (such as FORTRAN, BASIC, Pascal, etc.) may have exponentiation (which can be added to DATATRIEVE), but otherwise they generally have only the same basic math operators. For convenience, most languages have libraries of functions for commonly used complex calculations (such as Logarithms, Trigonometry, Statistics, etc.), and so does DATATRIEVE. In addition, it is possible to add new functions to DATATRIEVE, either as "true" functions, or by writing procedures which are then used like subroutines or functions. (Unfortunately, DATATRIEVE-11 / PRO-DATATRIEVE doesn't have "true" functions, but users can still write their own procedures that are used like subroutines or functions.)

First Example

In order to illustrate the process, I will set up a sample domain and run through a series of examples. The record definition is:

```
01 SAMPLE_REC.
  03 ITEM PIC 9.
  03 A PIC 999 EDIT_STRING ZZ9.
  03 B PIC 999 EDIT_STRING ZZ9.
  03 C PIC 999 EDIT_STRING ZZ9.
  03 T1 PIC 9999 EDIT_STRING ZZZ9.
  03 T2 PIC 9(6) EDIT_STRING
ZZZ,ZZ9.
;
```

The domain is SAMPLE, and ITEM is a keyed field. This very simple domain is for demonstration purposes only.

The first example will be to calculate T1 by the formula $T1 = (A + B) * C$. While this could easily be done by making T1 a COMPUTED BY field it serves as a simple starting point. (In DATATRIEVE-11, it is not possible to sort on a computed field, but it will be possible to sort on T1.) The FOR statement will be used as it is the easiest way to perform the same calculation for every record in a domain or collection.

For demonstration purposes, I've put the following data into the sample domain.

ITEM	A	B	C	T1	T2
1	3	5	7	0	0
2	7	5	3	0	0
3	2	6	4	0	0
4	7	3	4	0	0

A possible command sequence is to perform the calculation is:

```
READY SAMPLE MODIFY
FOR SAMPLE MODIFY USING T1 = (A + B) *
C
PRINT SAMPLE SORTED BY DESC T1
```

After the commands, it looks like this:

ITEM	A	B	C	T1	T2
1	3	5	7	56	0
4	7	3	4	40	0
2	7	5	3	36	0
3	2	6	4	32	0

Next: Running Totals

Something which will find a greater range of applications than the first examples is running totals: for this, it is necessary to store data from one record to another in some sort of variable or field, and this raises the first important point concerning "programming" in DATATRIEVE, which is that there are no

default variables as there are in BASIC or FORTRAN. All fields must be DEFINED in a record or DECLARED, and you must make the field large enough to hold the data planned for it. Starting with the same sample domain, the commands would be:

```
DECLARE RUNNING PIC 9(6) .
RUNNING = 0
FOR SAMPLE MODIFY USING T1 = (A + B) *
C
FOR SAMPLE SORTED BY DESC T1
  MODIFY USING BEGIN
  RUNNING = RUNNING + T1
  T2 = RUNNING
END
```

Since the running total will be in field T2, RUNNING has been declared to be the same size as T2 (though it doesn't have to be: it just has to be large enough to hold the largest number which will be encountered). Notice that RUNNING *must* be initialized to zero: DATATRIEVE does not initialize any fields, though sometimes you get lucky and get a blank area of memory. In this example, the data is placed in the current collection rather than storing the running totals as the collection is being totaled by field T1 rather than by the primary key field of the sample domain. The domain (sorted by descending T1) now looks like this:

ITEM	A	B	C	T1	T2
1	3	5	7	56	56
4	7	3	4	40	96
2	7	5	3	36	132
3	2	6	4	32	164

The running totals are now in place, and the current collection is ready for the report writer. Since the original version of this paper other methods of obtaining running totals in reports have appeared, but this method is still useful for obtaining running totals outside of the report writer, or when you want to obtain totals to store into a new domain or for other calculations.

More Difficult: Square Roots

The next step in difficulty will be to calculate the square root of a number (this is useful for standard deviation and other statistical calculations) using the Newton-Raphson method. First, to test my algorithm, I will make a procedure which will accept a number and calculate the square root, printing out the value to see if it's correct.

```
DEFINE PROCEDURE TEMP
DECLARE T1 PIC 9999
  EDIT_STRING ZZZ9.
DECLARE ROOT PIC 9999V99
  EDIT_STRING ZZZZ.Z9.
DECLARE TRY PIC 9999V99
```

```

EDIT_STRING ZZZZ.Z9.
DECLARE DIF PIC S9999V99
EDIT_STRING SZZZZ.Z9.
T1 = *.INPUT
TRY = 2
DIF = 1
WHILE DIF > 0.01 BEGIN
    ROOT=T1/TRY
    TRY=(ROOT+TRY)/2
    DIF=ROOT - TRY
    IF DIF<0 DIF=DIF*-1
    PRINT ROOT, TRY, DIF
END
PRINT T1, ROOT, TRY, DIF
END_PROCEDURE

```

There are several important points in this procedure. First, to repeat a previous statement, it is necessary that all fields be declared, and that they be large enough to hold the expected data. Notice that ROOT, TRY and DIF all have 2 decimal places reserved: if they did not, the square root would be calculated to the nearest whole number only. Note also that DIF has space reserved for a sign, as the difference between the last try and the present try could be positive or negative. Again, TRY and DIF must be initialized as DATATRIEVE does not initialize variables.

The WHILE statement is indispensable for this type of calculation as there are no labels and no GOTOs in DATATRIEVE. There is generally only two methods of performing repetitive calculations: the FOR statement which is used to perform some operation once on each record of a domain, and the WHILE statement for other repetitive calculations as it is not tied to a domain. In this case, the WHILE statement repeats until DIF (the difference between the present guess and the previous guess) is less than .01, this being the chosen limit of accuracy as the numbers were declared to have two decimal places. Incidentally, the constant 0.01 could be another field or variable, but if it is explicitly stated as it is here, it *must* have the leading zero.

The next three lines are the algorithm: divide the number by a guess and average the difference between the guess and the answer to form the next guess, repeating the process until the required accuracy is obtained. Notice that while spaces around math operators are usually optional, if you enter the second line as `DIF = ROOT - TRY` DATATRIEVE will tell you that field `ROOT_TRY` is undefined or used out of context. I have deliberately “squeezed” everything together in this example to show that DATATRIEVE is reasonably tolerant of variations in programming “style”, but I recommend using spaces between items to make things more “readable”, to avoid the minus sign versus dash problem, and it usually makes things easier to edit.

The next line forces the value of DIF to be positive (the absolute value) to meet the condition of the WHILE statement, otherwise any negative value for DIF would end the calculation prematurely. The loop ends not when DIF is calculated but at the end of the block, which is how most “do loops” operate. The print line within the BEGIN-END block is a de-

bugging aid: by placing a print statement here I can watch the values for each variable for each pass through the loop and determine if my logic is correct. When the procedure is correct, this line may be removed so that the final answer is printed by the last PRINT statement. Another method of debugging is to place the commands in an indirect command file: this way syntax errors are more visible as each line is printed when read in by DATATRIEVE. (Remember to \$ SET VERIFY to see things happening on a VAX.)

Two samples of the printout (with debug) look like this:

```

DTR> :TEMP
Enter INPUT: 25

    ROOT      TRY      DIF
12.50      7.25      5.25
 3.44      5.34      1.90
 4.68      5.01      .33
 4.99      5.00      .01

```

```

T1  ROOT      TRY      DIF
25  4.99      5.00      .01

```

```

DTR> :TEMP
Enter INPUT: 35

    ROOT      TRY      DIF
17.50      9.75      7.75
 3.58      6.66      3.08
 5.25      5.95      .70
 5.88      5.91      .03
 5.92      5.91      .01

```

```

T1  ROOT      TRY      DIF
35  5.92      5.91      .01

```

DTR>

One of the advantages of DATATRIEVE is that it appears to the user as an “interpreter”, like the original BASIC: this means that you can take statements and execute them immediately without having to go through some intermediate compilation process. Since you can also edit your procedures from within DATATRIEVE, and examine your data before and after executing the procedure within DATATRIEVE, the development cycle can be quick, and the user only has to work with one product (or two, if you count the editor separately). If you are working on a procedure, like this one, you can run it, see if it’s correct, make whatever changes are necessary, and re-run the procedure all from within DATATRIEVE.

Making the Procedure Useful

Now that this procedure works, I will put it into a form where it can be used elsewhere, and call it SQRT. This is a way to build up a library of "functions" or "subroutines" usable in DATATRIEVE (which will even work in DATATRIEVE-11).

```

DEFINE PROCEDURE SQRT
IF T1 LE 0 ABORT "No Negative Numbers"
DECLARE ROOT USAGE IS REAL.
DECLARE TRY USAGE IS REAL.
DECLARE DIF USAGE IS REAL.
TRY = 2
DIF = 1
WHILE DIF > 0.01 BEGIN
    ROOT = T1 / TRY
    TRY = (ROOT + TRY) / 2
    DIF = ROOT - TRY
    IF DIF < 0 DIF = DIF * -1
END
END_PROCEDURE

```

The print statements and definition of T1 have been removed: T1 must be defined before the procedure is called (so the calling procedure will make the space reservation and assign a value to it before calling this procedure), and ROOT will contain the answer when finished. As there are no argument lists as there may be when calling subroutines in other languages, it is the responsibility of the person writing the procedure to document carefully the fields which must be defined before the procedure is used, what types of fields they should be, and what field will contain the answer when finished.

The first line in this procedure is very important: in order for any equation to yield the correct answer, the input data must be correct (remember Garbage In, Garbage Out?). Since negative numbers have no real square root, it is necessary to insure that input to this procedure is not negative. The variable declarations are also slightly different. Rather than limit the range and accuracy of the procedure, the use of REAL variables allows these fields to accept very large or very small values: this is very handy for cases when you may not know just what values the variables will have, and it occupies less space than a large number with one character per byte (the default DISPLAY data type). This procedure is now ready to be used as part of another procedure. For example, let us put the sum of A, B and C into T1, and 100 times the square root of T1 into T2.

```

READY SAMPLE MODIFY
FIND SAMPLE
FOR CURRENT MODIFY USING BEGIN
    T1 = A + B + C
:SQRT
    T2 = 100 * ROOT
END

```

The current domain now looks like this:

ITEM	A	B	C	T1	T2
1	3	5	7	15	387
2	7	5	3	15	387
3	2	6	4	12	346
4	7	3	4	14	374

It should be noted that there are alternate methods of dealing with an incorrect value for T1. One method is:

```

DEFINE PROCEDURE SQRT
    DECLARE ROOT USAGE IS REAL.
    ROOT = 0
    WHILE T1 GT 0 BEGIN
        DECLARE --- variables as before
        ---
        --- initialize variables ---
        WHILE ...
            ---- procedure as before ----
    END
END
END_PROCEDURE

```

In this case, the entire procedure will be executed only if T1 is greater than zero, otherwise nothing is done, and ROOT defaults to zero (the rest of the procedure is unchanged). Another alternative is:

```

DEFINE PROCEDURE SQRT
    DECLARE ROOT USAGE IS REAL.
    IF T1 GT 0 BEGIN
        ---- procedure as above ----
    END ELSE
        ROOT = 0
    END
END_PROCEDURE

```

Here the IF-THEN-ELSE statement is used to execute the procedure if T1 is valid, and return a dummy value of zero for the root if T2 is invalid. The last three lines could be condensed into one, but writing it this way brings it closer to "normal" programming. One caution: most "structured" programmers would put the ELSE statement at the beginning of a new line, to clarify the structure. This is *not* possible in DATATRIEVE: the verb ELSE *cannot* be the first word on a line. (Generally speaking: there are some "tricks" that can be done, but they generally aren't worth doing.)

The last example could also be performed in this manner:

```

DEFINE PROCEDURE SQRT
    DECLARE ROOT USAGE IS REAL.
    IF T1 GT 0 BEGIN
        ---- procedure as above ----
    END
    IF T1 LE 0 ROOT = 0
END_PROCEDURE

```

This appears to be both less structured and less efficient than the previous version, but it has one advantage in that it “compiles” faster under some circumstances (and uses less pool space in DATATRIEVE-11, though this won’t bother VAX-DTR or DTR-20 users). In the first version, all of the statements from the IF to the last END (before the END_PROCEDURE) must be “compiled” before any part of the IF statement is executed, including evaluation of the IF condition itself, and this takes time (and pool). In the second version, each IF statement is “compiled” separately and executed separately. If you are going to be going through a repetitive series of calculations many times, it’s usually faster to put all of the statements into one big WHILE statement or BEGIN-END block, let it all be compiled once when you enter the routine, and then let it run. The program may seem to “stall”, while the statements are compiled, but once done the procedure will run very fast, about as fast as code compiled in other languages. Although it sometimes appears to be one, DATATRIEVE is not an interpreter (like the original BASIC) where individual lines of code are interpreted and executed before moving on to the next line: each “block” of code is compiled, then executed. If, however, you have a procedure which will probably be executed once only (or once in a while), and it contains a large number of IF conditions, it may be better to put them in as separate IF statements so you won’t waste time compiling everything just to execute one small statement. [For DATATRIEVE-11 users, in this example, the savings in pool will be small because the second IF statement is so short, but with more complicated IF-THEN-ELSE conditions this method of breaking up the computations into smaller segments can save a considerable amount of pool, and is especially useful in DATATRIEVE-11.]

It should be noted that the statement

```
WHILE DIF > 0.01 BEGIN
```

could have been written in many different ways. One could also say WHILE (DIF > 0.01 OR DIF < -0.01) BEGIN, or WHILE DIF BETWEEN -0.01 AND 0.01 BEGIN, or any other valid Boolean expression. If any of these had been used, the line

```
IF DIF < 0 THEN DIF = DIF * -1
```

which converts negative values to positive values would not be required.

It should also be noted that the procedure name SQRT isn’t really very descriptive. It would probably be better to call it something like SQUARE_ROOT or even SQUARE_ROOT_2 (for 2 decimal places), and place it in a common dictionary for everyone to use once it has been debugged. The above example of use would then look more like this:

```
READY SAMPLE MODIFY
FIND SAMPLE
FOR CURRENT MODIFY USING BEGIN
  T1 = A + B + C
  :CDD$TOP.USER$LIBRARY.SQUARE_ROOT
  T2 = 100 * ROOT
END
```

(DATATRIEVE-11 procedures will look like the original examples.)

Procedures versus Functions

The next important question is: should this be a procedure, or should a function be added to DATATRIEVE? This is going to depend a lot on what other facilities are available, how often the function will be used, and on how many systems the function will be used.

Adding a function to DATATRIEVE is not at all difficult, especially if you are adding one of the VMS library routines, as you don’t have to write any code. Doing the calculation in a function is often more efficient as DATATRIEVE doesn’t have to “compile” the code each time it’s used, and this is especially important if the function is going to be used often. If you want to add a function of your own, however, the first step is to write a subroutine to implement the function in some programming language that supports the VMS calling standard. This is going to be the first stumbling block for many DATATRIEVE users, who don’t have a traditional “programming” background. Next, the function must be linked into the DATATRIEVE image: this isn’t difficult, but many system managers resist change. Of more practical difficulty is what happens if your DATATRIEVE procedures have to be distributed to many different systems (for example, if you have developed something that is going to be throughout a company or corporation). If you do everything as DATATRIEVE procedures, you can distribute the DATATRIEVE code and be certain it will work: if you depend upon special functions, then you must be certain that those functions have been built into every DATATRIEVE image where your code will run. This can be especially difficult if the corporation is widely distributed, and, as often happens, different systems are running different versions of the operating system and DATATRIEVE.

An Application: Least Squares Data Fit

One more example of this type of operation will be fitting a trend line to data in a domain. This is the “least squares” method of fitting the best line to a set of data points, and is often used for such things as predicting future growth. Though there is a least squares fit for many plots in DATATRIEVE, the values are not retrievable for use within DATATRIEVE and this procedure will make the values usable for storage in a domain or other use.

The procedure is:

```
DEFINE PROCEDURE TREND
  DECLARE SUMX USAGE IS REAL.
  DECLARE SUMY USAGE IS REAL.
  DECLARE SUMXY USAGE IS REAL.
  DECLARE SUMXSQ USAGE IS REAL.
  DECLARE SUMYSQ USAGE IS REAL.
  DECLARE SLOPE USAGE IS REAL.
  DECLARE INTERCEPT USAGE IS REAL.
  DECLARE FIT USAGE IS REAL.
  DECLARE TEMP USAGE IS REAL.
```

```

DECLARE N USAGE IS INTEGER.
N = 0
SUMX = 0
SUMY = 0
SUMXY = 0
SUMXSQ = 0
SUMYSQ = 0
READY SAMPLE
FOR SAMPLE BEGIN
    SUMX = SUMX + ITEM
    SUMY = SUMY + T1
    SUMXY = SUMXY + (ITEM * T1)
    SUMXSQ = SUMXSQ + (ITEM * ITEM)
    SUMYSQ = SUMYSQ + (T1 * T1)
    N = N + 1
END
TEMP = ((SUMX * SUMY / N) - SUMXY)
SLOPE = TEMP / ((SUMX * SUMX / N) -
    SUMXSQ)
INTERCEPT = (SUMY - SLOPE * SUMX) / N
FIT = SLOPE * TEMP /
    (SUMYSQ - (SUMY * SUMY / N) )
PRINT SLOPE USING ZZZ9.9999,
    INTERCEPT USING ZZZ9.9999,
    FIT USING ZZZ9.9999
FINISH SAMPLE
RELEASE N
RELEASE TEMP
RELEASE FIT
RELEASE INTERCEPT
RELEASE SLOPE
RELEASE SUMYSQ
RELEASE SUMXSQ
RELEASE SUMXY
RELEASE SUMY
RELEASE SUMX
END_PROCEDURE

```

The procedure follows the same rules as before as to declaring all variables and initializing them. The FOR statement is used to process the domain and sum up some values which will be required for the calculation. The question might arise as to why the procedure is summing up the values for X (ITEM) and Y (T1) and counting up the number of items in N when it could simply FIND the domain and then use the SUM and COUNT commands to have DATATRIEVE do the work. The answer is that the procedure has to go through the domain once anyway to sum the squares of the variables and the products of the two variables, and it is more efficient to also sum the other values at the same time than to have DATATRIEVE make additional passes through the domain to to the summing and counting, especially if this were to be done on a large domain. It is a good general rule to gather as much data at one time as possible to save time in processing (but don't store values you won't need). This is also shown by the use of an intermediate calculation for the value of TEMP: this expression is used in two other places, and it is more efficient

to use four bytes of pool to store the value than to calculate it twice, and it is also faster. The data now in the domain and the answers look like this:

ITEM	A	B	C	T1	T2
1	0	0	0	1200	0
2	0	0	0	1800	0
3	0	0	0	1600	0
4	0	0	0	1900	0
5	0	0	0	1800	0
6	0	0	0	2100	0

```

DTR> :TREND
      SLOPE      INTERCEPT      FIT
137.1429 1253.3334      0.6954
DTR>

```

The statements which were missing from previous examples but are included here are FINISH and RELEASE. In DATATRIEVE-11, pool is always a scarce resource, it is good practice to free up pool space by closing out domains and releasing space reserved for variables which are no longer used. On the VAX, it is often thought that, since the system uses virtual memory, there is an unlimited supply. This is not true: memory is **not** unlimited, and keeping around structures you don't need will eventually cost you something. Although only global variables actually require explicit release, it is best to get into the habit of releasing resources as soon as possible: in this example, if the RELEASE statements were not included, the variables would still be stored in pool after the procedure was finished.

A Few Suggestions

At this point, the reader should have a grasp of what is possible in the way of equation solving in DATATRIEVE. More complex problems may be approached by breaking them down into smaller sections, each of which should yield to one of the methods presented. For those who plan to go further with this approach, the following subjects in the DATATRIEVE manual will be of interest: the ABORT, DECLARE, FOR, WHILE, CHOICE, and IF-THEN-ELSE commands; arithmetic and Boolean expressions; (procedures and indirect command files; optimization; and especially the section dealing with the USAGE clause, which describes the internal format of the different types of numbers. COMP {INTEGER, BYTE, WORD, LONG, QUAD} is usually the most efficient type of storage; for real numbers REAL {FLOAT} and DISPLAY (the default) should be the next most efficient. The author recommends avoiding COMP_{PACKED}, COMP_{ZONED}, and COMP_6 except when needed to read data written by other programs, and DATE (except for date calculations).

Where to find Equations

Readers may be interested in knowing where to find equations in suitable form for solution in DATATRIEVE (or other computer languages). Books on the particular subject (for example, a book on statistics for standard deviation or trend line fitting) are a good beginning, especially the older books which give instructions for solving the equations by hand; and even better, books which show how to solve the equations on pocket calculators. When such calculators were more expensive than they are now, and most had only four functions (rather than the specialized math or financial calculators now available), a number of books showing how to break down trigonometric functions, financial equations, etc. into a form which could be solved on a four function calculator were published, and these methods should be easily transferred to DATATRIEVE. They will also give worked examples, so the user can compare the answer obtained in DATATRIEVE with the answers in the book to determine if the equation has been correctly solved. Another good source is the manuals provided with programmable pocket calculators, (if you can still find one) which often give the formula and a worked example: the trend line example was obtained in this way. There are also books published for high-school and college math classes containing nothing but formulas, and some have functions expanded into series, which are particularly suitable for solution by computer. Finally, for those wishing to solve trigonometric functions, the Fortran-IV (Fortran-77) manual set contains an appendix describing the methods used to provide those functions and the accuracy obtained.

Built-In Functions

VAX-Datatrieve has the following built-in functions which might be used for mathematical operations (not including the date functions):

```
FN$ABS FN$ATAN FN$COS FN$EXP
FN$FLOOR FN$HEX FN$LN FN$LOG10
FN$MOD FN$NINT FN$SIGN FN$SIN
FN$SQRT FN$TAN
```

Many users appear reluctant to use these functions as they are not "english-like" as is the rest of DATATRIEVE, but in fact are really quite simple to use. For instance, the square root example could be reduced to:

```
FOR SAMPLE MODIFY USING BEGIN
  T1 = A + B + C
  T2 = 100 * FN$SQRT(T1)
END
```

or, if you aren't really interested in the intermediate value of T1:

```
FOR SAMPLE MODIFY
  USING T2 = 100 * FN$SQRT(A + B + C)
```

Or you can modify the original record definition:

```
01 SAMPLE_REC.
  03 ITEM PIC 9.
  03 A PIC 999 EDIT_STRING ZZ9.
  03 B PIC 999 EDIT_STRING ZZ9.
  03 C PIC 999 EDIT_STRING ZZ9.
  03 T2 COMPUTED BY FN$SQRT(A + B +
C) .
;
```

Or, if you don't want to store the value:

```
FOR SAMPLE PRINT FN$SQRT(A + B + C)
```

Functions thus have the advantage that they can be incorporated into places where procedures cannot be used, or cannot be easily used.

To add your own functions to DATATRIEVE, you have to modify a file, DTRFND.MAR, supplied with DATATRIEVE. When installing DATATRIEVE, you are asked if you want to save certain customization files: say YES to save the function file. Although this is a Macro-32 language source file, it doesn't really look like assembler language as it simply consists of function definitions. For example, the following function definition adds a function which raises a number to a power.

```
; FN$POWER - Raise a real number to
;               a real power
;
; Output is a floating value in R0, R1
; Input is two floating values
;   passed by immediate value
;
$DTR$FUN_DEF FN$POWER, OTS$POWRR, 2
  $DTR$FUN_OUT_ARG
  TYPE = FUN$K_VALUE,
  DTYPE = DSC$K_DTYPE_F
  $DTR$FUN_HEADER HDR = <"Power">
  $DTR$FUN_IN_ARG
  TYPE = FUN$K_VALUE,
  DTYPE = DSC$K_DTYPE_F, ORDER = 1
  $DTR$FUN_IN_ARG
  TYPE = FUN$K_VALUE,
  DTYPE = DSC$K_DTYPE_F, ORDER = 2
  $DTR$FUN_END_DEF
```

In this instance you don't have to write your own routine to do the work as it uses a routine in a library supplied with VMS. More function definitions like this, and a DATATRIEVE procedure that generates the definitions, may be found in the DATATRIEVE / Fourth Generation Languages SIG Library tape, which is in the DECUS library and on the VAX SIG Symposia tape.



VAX DATATRIEVE Security Using Environment Accounts and ACL's

by Michael G. Graham
Sanders Associates
95 Canal Street
Nashua, NH 03061

Abstract

The security and integrity of database system information is paramount, particularly as it applies to personnel and financial records of groups and/or companies. External security enhancements now include such devices as *Defender II* type security call-back schemes, which require the use of special passwords and having the computer call back the user, if connected to the system by modem. This presentation addresses the issue of *internal* security, that of preventing otherwise authorized system users from accessing sensitive database information residing on the machine. This presentation concerns itself specifically with DATATRIEVE Security. The scheme defined herein uses a two-fold approach.

Since DATATRIEVE data files reside within VAX/VMS sub-directories, and since the DATATRIEVE Domain and Record Definitions, Procedures, and Tables reside within the Common Data Dictionary (CDD), security precautions for both must be implemented. It is assumed that the data is to be shared by some but not all system users. Security for the actual data files will be first discussed, followed by a discussion of DATATRIEVE security and a layered approach to data integrity.

Data File ACL's

The ordinary SET PROTECTION scheme for files is nullified within an Environment Account. The entire purpose of the Environment is to allow users within a *common* need to access files within that Environment. However, there are still instances wherein certain data files may require restrictions, limiting the access to *specified* users within the group. This can be accomplished by the use of FILE and/or DIRECTORY ACL's. The ACL offers a way to match the specific access you want to grant or deny to specific users for each object.

Identifiers are the means of specifying the users in ACL. There are three types of identifiers:

- UIC identifiers that depend on the User Identification Code (UIC) that uniquely identify each user on the system.
- General identifiers that are defined by the Security Manager in the system rights database to identify groups of users on the system.
- System-defined identifiers that describe certain types of users (BATCH, NETWORK, DAILUP, INTERACTIVE, LOCAL, REMOTE).

UIC identifiers conform to the specific UIC's. The Security Manager creates and assigns the general identifiers and UIC's to the system users with the Authorize Utility (AUTHORIZE). System-defined iden-

tifiers are automatically defined by the system when the rights database is created at system installation time.

The next step in this protection scheme is to define what access to grant or deny to the holders of each of these identifiers, for each file that needs this level of protection. Because there may be more than a few identifiers needed to represent differing access needs for each object, it is fairly common to create a whole list of entries, each of which define groups of access rights to grant or deny. Such a list is called an access control list (ACL), and each entry within the list is called an access control list entry (ACE). ACL's offer the user an opportunity to fine-tune the action taken when access is sought to an object. You can provide an ACL on any object to permit as much or as little access as is desirable in each case. They can even cause security alarms to be set off when access to an object succeeds or fails.

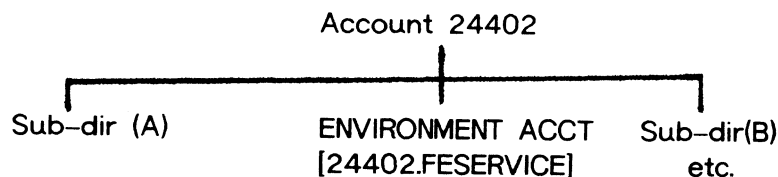
The obvious advantage to having an intricate scheme of file ACL's also has its negative side. While file ACL's enhance the security of the system, the payment comes in user time required to generate and maintain them and the processor time required to perform the functions that ACL's mandate.

For a full description of File ACL's, their use and implementation, refer to the *VAX/VMS DCL Dictionary* and the *VAX/VMS Utilities Reference Manual*.

ENVIRONMENT Accounts

An Environment Account can be implemented on any VAX system, and it permits the designated "owner" to define who may or who may not access the files residing within that account. When a user logs into an Environment Account, it is essentially the same as logging into a regular account, with the exception that passwords are not normally used. An Environment Account usually appears as a sub-directory to some "user" account on the system. Whose account the Environment is established in is usually dictated by the needs of the group, department, etc., which

will be accessing the databases and data. An example of an actual Environment Account is diagrammed below for illustration:



In the above illustration, a user account [24402] (my company Clock Number) contains several sub-directories. One of those sub-directories is named "FESERVICE" and is an Environment Account. The account even contains a LOGIN.COM file so that specifically tailored commands can be included, as well as setting the default CDD dictionary for DATATRIEVE use. Another part of the Environment includes an ENV USERS.DAT file, which when used with specific commands, allows the "owner" to ADD, DELETE, MODIFY, and PRINT the list of users who have authorized access to the Environment. From this point, additional sub-directories can be built as required.

The interesting feature here is that the normal process of access to user files is bypassed, all other sub-directories in your VMS account can be protected as desired, and only those individuals who are on the Authorized Access List to the Environment may be allowed into the Environment. You may also wish to keep a record of who accessed the Environment Account and when. Figure 1 illustrates an example of a .COM file which I wrote for that purpose. As each Environment Account user accesses the account, he is automatically logged into a record file, which is periodically scanned for security and accounting purposes. This .COM file is one of the immediately executed files as the user accesses the account. Figure 2 illustrates the LOGIN.COM file as I have it implemented on my account. Any bell and whistle you wish to add can, of course, be done.

To have an Environment Account established within your own User Account, you must see your System Manager. The Manager will also provide you with

```

$CONTEXT = " "

$PID = F$EXTRACT(0,17,F$PID
(CONTEXT)1

$TIMEIN = F$EXTRACT
(0,17,F$TIME())

$NAME = F$GETJPI(PID,
"USERNAME")

$OPEN/APPEND OUTFIL
DISK:[24402.FESERVICE]TIMEIN.LOG

$WRITE OUTFIL "USER: ",NAME,
"LOGON: ",TIMEIN1

$CLOSE OUTFIL

```

Figure 1

full instructions as to the methods to maintain the account. You are now, in effect, an Account Manager. If your DATATRIEVE account is structured such that the full dictionary path name is specified in the Domain Definitions for all your DATATRIEVE data files (which it should be), then those data files which are to be shared by your Environment Users *must* reside within the Environment.

To illustrate my point, my DATATRIEVE account for the Field Engineering Services Department has the CDD pathname CDD\$TOP.USERS.FESERVICES. Within that dictionary object are sub-dictionaries. One such dictionary is PERSONNEL. Within the sub-directory PERSONNEL is a Domain entitled FSE. The Domain Definition for that database, following the above guideline, is:

```

"DOMAIN FSE USING FSE REC ON
DISK:[24402.FESERVICES.DBASE]
FSE.DAT"1

```

¹Note: The above example would be typed on one line.

You can also see that there is a sub-directory within the environment entitled ".DBASE".

Within that sub-directory, all DATATRIEVE datafiles reside. By always specifying the full pathname for the .DAT file, you ensure that *only* those who have access to the Environment can access the file, as well as reduce confusion if a person has neglected to set his default directory to the right place.

```

$SET NOON

$SET PROT=(S:RE,O:RWED,G,W)
/DEFAULT1

$DTR == "$SYS$SYSTEM
:DTR32.EXE"1

ASSIGN/PROCESS "CDD$TOP.USERS
.FESERVICES" CDD$DEFAULT1

$ASSIGN DISK:[24402.FESERVICE.HLP]
RESUME HLP$LIBRARY 11

$RESET ::=SET DEF DISK:[24402
.FESERVICE]1

$BASE ::=@DISK:[24402
.FESERVICE.DBASE]
DBASE.COM1

$USERS ::=@DISK:[24402
.FESERVICE.USERS]
USERS.COM1

$HLP ::=@DISK:[24402
.FESERVICE.HLP]
HLP.COM2

$VID*EO ::=@DISK:[24402
.FESERVICE.VIDEO]
VIDEO.COM1

$CAI ::=SET DEF DISK:[24402
.FESERVICE.CAI]1

```

—continued—

Figure 2

```

$SHORT      ::=SET TERM/
            WIDTH=80

$LONG       ::=SET TERM
            WIDTH=132

$SHORT

$@TIMEIN

$WRITE SYS$OUTPUT "Welcome to the
FESERVICE Environment Area."
"In case of disaster, call Mike Graham,
X-52061"

$WRITE SYS$OUTPUT " "

$TYPE DISK:[24402.FESERVICE]NOTICE
.TXT1

$WRITE SYS$OUTPUT " "

$EXIT

```

Figure 2, continued

DATATRIEVE Security

The CDD also has a protection enhancement which allows the "owner" of the Dictionary Object to protect it in a variety of ways. Essentially, there are thirteen different levels of protection which can be implemented on any dictionary object. The choices, and how to implement them, can be somewhat confusing. Protecting the data files within the Environment Account is only closing one of the barn doors. To keep the cows at home, to ensure that *only* authorized personnel have access to both data and CDD objects, Access Control Lists (ACL's) should be implemented within DATATRIEVE. The new *DATATRIEVE User's Guide* (a super book, by the way) contains a rather lengthy discussion on implementing

Access Controls. However, through experience and a lot of plain knocking my head against brick walls, I have devised a rather simple, but quite effective case method of CDD item protection. Two overall general rules apply:

- There are only *two* people authorized to create sub-dictionaries, define Domains and Records, and create Procedures and Tables.
- All other users are confined to reading writing to, or modifying established databases.

If this perhaps sounds restrictive, it is meant to. The principles involved here are based on military and government control systems. The reason for having two full-access personnel is really self-explanatory. People get ill, take vacations, have babies, etc. There should be a backup person with full access to manage the database system in the event of the *untimely demise* of the System Manager. It is also a proven fact that the potential for system compromise increases in direct proportion to the number of people who have unlimited system access.

The following illustration is of a DATATRIEVE system using the CASE method of implementing DATATRIEVE security. The system uses six cases. Each case provides for a certain level of security within the system. To view, add, or delete names from the DATATRIEVE ACL's, the following DATATRIEVE terms are used:

DEFINEP	define an ACL for a given individual or UIC.
DELETEP	delete an individual from the ACL for the object.
SHOWP	list the ACL for the specified object.

The normal method of defining an Access Control List consists of adding users to each object in the following manner:

```

DEFINEP [objectname] [position#]
GRANT=[priv].DENY=[priv],
BANISH=[priv]2

```

²Note: The above example would be typed on one line.

At Level 1, people can go different directions, depending on how you structure the ACL's for each sub-directory. Suppose SMITH and WESSON have a need to access the Procedures of databases in SUB (A), but not ACES. By the same token, only ACES has access to the objects in SUB (B). Everyone has access to SUB (C) (the Domain and Record Definitions reside there, or at least the master databases do). Only YOURNAME has access to SUB (D).

```
DEFINER SUB (A) 1 USER=
YOURNAME,GRANT=ALL2
```

```
DEFINER SUB (A) 2 USER=
SMITH,GRANT=PRS,DENY=D,
BANISH=FG2
```

```
DEFINER SUB (A) 3 USER=
WESSON,GRANT=PRS,DENY=D,
BANISH=FG2
```

```
DEFINER SUB (A) 4 UIC=
[*,*],DENY=ALL2
```

```
DEFINER SUB (C) 2 USER=
SMITH.GRANT=PRS,DENY=D,
BANISH=FG2
```

```
DEFINER SUB (C) 3 USER=
WESSON.GRANT=PRS,DENY=D,
BANISH=FG2
```

```
DEFINER SUB (C) 4 USER=
ACES.GRANT=PRS,DENY=D,
BANISH=FG2
```

```
DEFINER SUB (C) 5 UIC=
[*,*],DENY=ALL2
```

```
DEFINER SUB (D) 1 USER=
YOURNAME,GRANT=ALL2
```

```
DEFINER SUB (D) 2 UIC=
[*,*],DENY=ALL2
```

Here again, note position 4. The catchall is used at the end of *every ACL!!!* At this level, SMITH and WESSON can pass into SUB (A), ACES can't, and of course YOURNAME is *always* there. The CASES at this level used were CASE 1 for YOURNAME, CASE 3 for SMITH and WESSON, CASE 6 for everyone else. To define the ACL's for SUB (B), (C), and (D), you would use the exact same format, only changing the names:

```
DEFINER SUB (B) 1 USER=
YOURNAME,GRANT=ALL2
```

```
DEFINER SUB (B) 2 USER=
ACES,GRANT=PRS,DENY=D,
BANISH=FG2
```

```
DEFINER SUB (B) 3 UIC=
[*,*],DENY=ALL2
```

```
DEFINER SUB (C) 1 USER=
YOURNAME,GRANT=ALL2
```

At each Level 2, each object, be it Domain Definition, Record Definition, Procedure, or Table, must be defined. This is particularly true in those areas where all personnel will have access to the same databases. For example, OBJECTS 1 and 2 within SUB (C) are a Domain and Record Definition respectively, you would use CASE 4 for SMITH and WESSON, CASE 5 for ACES, and of course, CASE 6 as the last entry. Example:

```
DEFINER OBJECT1 1 USER=YOURNAME
,GRANT=ALL2
```

```
DEFINER OBJECT1 2 USER=SMITH,GRANT
=EPRSW,DENY=D,BANISH=FG
```

```
DEFINER OBJECT1 3 USER=WESSON,GRANT
=EPRSW,DENY=D,BANISH=FG
```

```
DEFINER OBJECT1 4 USER=ACES,GRANT
=EPRSW,DENY=D,BANISH=FG
```

```
DEFINER OBJECT1 5 UIC=[*,*],DENY=ALL4
```

⁴Note: The process for OBJECT2 would be identical.

User ACES can only access the database for read privileges, whereas users SMITH and WESSON can access the same database for read/write/modify privileges.

However, none of the users except YOURNAME (remember, the *boss!* has privileges which would allow them to modify, delete, create, or otherwise perform actions within the sub-dictionary which could be detrimental to the system. Absolute control over the system is retained by the System Manager. One additional thought: the above scheme also proves the argument that a SMITH and WESSON beats ACES *always!*

The alphabet soup of letters used in conjunction with the DEFINEP command merely determines the extent of privileges within your DATATRIEVE account. If care is taken, you can use the CASE method for most applications. If other privileges are required, consult the *DATATRIEVE User's Guide*. This particular scheme has been in effect for over three years at my company on an extremely large database system, and it has never failed. It does, however, place a burden on the System Manager to ensure that ACL's are kept current.

In conclusion, by combining the use of an Environment Account and File ACL's to protect the DATATRIEVE data files, and by using the CASE method to create ACL's within a layered DATATRIEVE account, you can afford your DATATRIEVE system the maximum in internal security.



Making an Inexpensive Rainbow Workstation for a Chemistry Lab

John D. Bak and David M. Hayes
Department of Chemistry, Union College
Schenectady, New York 12308

Abstract

Chemical kinetics is the study of how quickly and by what means chemical reactions proceed. Some reactions are so fast and complicated that data must be taken very quickly and then lengthy calculations done to get results. Computers speed these studies greatly. The system described herein uses a *DEC-Rainbow* microcomputer as a terminal for a workstation. The system may collect 8k of buffered data at a rate of 1MHz and then upload the data to a VAX for analysis using the same Rainbow as a graphics terminal. The program for data analysis will then accept the data and also other information about the chemical reactions in the same symbolic format that chemists use so that the analysis may be done.

Introduction

This system was specifically designed to collect and analyze data from an instrument called a *flash photolysis spectrophotometer*, but it may also be used with other instruments. The way the "flash rig" works is that the chemicals to be reacted are put in a glass container, and then flashed by high intensity light from a xenon flash lamp or a laser. The length of this flash is typically less than 10 microseconds. Some of the energy of the light is absorbed by the chemicals and causes them to react. The reactions studied by this method may go to completion in as short a time as a hundred microseconds. The reaction is monitored by passing a continuous probe light through the sample cell and measuring the variation in transmitted intensity at a particular wavelength as a function of time. The transmitted light intensity can usually be correlated with the concentration of particular reactants, transients or products in the sample cell. This transmitted light is first converted to an electric current by a photomultiplier tube and then converted to a voltage by passing the current through a known resistance. This voltage is directly proportional to transmitted light intensity at the selected wavelength through the sample cell. This is the input the interface circuit between the instrument and the Rainbow.

There are two basic needs that we want this system to fulfill. First we want to be able to collect data by computer. Second, we want to be able to analyze this data using the college's VAX cluster. The basic system is an instrument connected to an interface circuit with a buffer which is connected to a Rainbow. The Rainbow acts as the data collection station and also as a VAX terminal for the school's VAX cluster, where data analysis software resides. These two basic components, the data

collection system and the data analysis software will be described herein.

Data Collection

System Requirements

There are several characteristics that this system has to have in order to be used to study the reactions that we have in mind. The minimum needs are:

- Variable timing: 1Mhz maximum rate
- A 2000 data point buffer
- Greater than 8 bits per data word
- Capability to upload data to VAX
- Graphics with hardcopy
- Must be inexpensive

The core of our system is a Rainbow microcomputer with a graphics board and an LA50 printer. This gives us our graphics and hardcopy capabilities as well as being a terminal to the VAX, so that data may be uploaded right after it is collected. The analog to digital (A/D) converter in the interface between the instrument and the Rainbow is a HAS1201 made by Analog Devices. This converter has a maximum rate of 1.05MHz at 12 bits per word. We wanted more than 8 bits to give us the sensitivity to make accurate readings in areas where the converter's full range was not being utilized. To get variable timing an 8253-5 programmable interval timer is used. It is configured in such a way that it gives us data sampling intervals ranging from 1 μ sec to about 8.9 years (more about how that is done later). The interface circuit has a 16k byte buffer so that it gives 8k words. Finally an 8251A USART is used to communicate between the Rainbow and the buffer circuit. The cost for the interface circuit was under \$1000 (this excludes the price of the Rainbow and printer). The

most expensive item was the A/D converter at \$512. The final characteristics we ended up with are:

- Variable sampling rate from 1 μ sec to 8.9 years per data point
- 8k word buffer for data
- 12 bit per data word
- Rainbow graphics and LA50 printer
- Also usable as a VAX terminal
- Inexpensive: less than \$1000

Using the System

The system has an 8085 microprocessor which receives commands from the Rainbow and then acts on them. To use it one first sets up the internal registers of the circuit by sending them commands from the Rainbow. The data collection process is started either by sending a command from the Rainbow, or through the remote start input on the circuit itself. The circuit then sends the data it collects back to the Rainbow and the process can start over again.

The circuit operates in three possible modes :

- Buffered operation.
- Real time operation.
- Programmed operation.

Buffered Operation

This mode allows rapid collection rates of up to 1MHz. This is possible because all the data is first stored in the 8k buffer before it is sent to the Rainbow. The number of sets of data to be taken must first be specified. Each of these sets of data will be taken in succession after the start collecting signal is received. This is a very nice feature because the rate at which each set is taken can be different, so that in the beginning data may be taken quickly, but at the end data may be taken more slowly as the reaction slows. After the number of sets of data to be taken is loaded, the periods for each set are loaded. The restriction on the data collection intervals is that each period after the first must be an integer multiple of the period before it. In other words, the period for data set two will equal the period for data set one times the number entered for data sets two timer register, and data set three's period will equal the period for data set two times the value entered for data set three's timer register. The number entered for period one is in halves of microseconds—1 and is between 0 and 65535. So if a period of 60 μ sec is desired the register is loaded with 119, and if a rate of 1MHz is desired a 1 would be loaded. The next thing to be loaded would be the number of points to be taken for each set. The total number of points taken cannot exceed 8191 points because of the buffer size. Finally, the remote start input would be enabled if the start collection signal is to come from outside, or the data collection could be started from the Rainbow. After data collection, the data would be transmitted to the Rainbow where it may be stored on floppy disk.

Real Time Operation

This mode is only usable at lower rates of data collection. Because the data words are 12 bits long, two bytes are required to transmit one word, so the maximum rate of data transfer into the Rainbow is about 600 words per second at the rate of 9600 baud. To use this mode the timer registers are loaded with the period of the sampling rate as before and then either the remote start is enabled or the timer is started from the Rainbow. Now the circuit will send each data word to the Rainbow as it is collected without buffering it. To stop the process, a command to stop is sent from the Rainbow.

Programmed Operation

Since there is an 8085 microprocessor in the interface circuit, small programs may be loaded into the circuit. This option is included for completeness and maximum flexibility.

Commands

The commands that the interface circuit can accept are:

- **Stop and reset**
- **Get 1,2 or 3 sets of data**
Set up to take buffered data at one, two or three different rates
- **Set up for real time collection**
- **Get one data point**
- **Load timer register 1,2 or 3**
Loads the timer registers. Each register is 16 bits
- **Load count registers 1,2 or 3**
Loads number of points to be taken with each rate for the buffered data. Total cannot exceed 8191.
- **Select timing from counter 1,2 or 3**
When taking real time data, selects which timer the timing will be taken from.
- **Start/Stop Timer**
Starts or stops timer without waiting for remote start.
- **Allow/Disallow remote start**
- **Load Temporary program**
May load a temporary program. Up to 15 may be loaded as long as they fit into the memory restrictions.
- **Run Temporary program.**
- **Run diagnostic tests**
- **Load Status Register**
This is an important register with hardware switches. More will be explained later.

Hardware Design

The interface circuit itself is a small microprocessor system with DMA (Direct Memory Access) capabilities for storing data from the A/D converter. The CPU is an 8085A and is used to control the states of the circuit and also to generate a 2MHz system clock from which all the timing is derived.

The A/D converter is a HAS1201 made by Analog Devices. This unit has a 1.05MHz maximum conversion rate, internal track and hold circuitry and 12 bit resolution. The one drawback to this unit is that there is no end of conversion signal. This is gotten around by tying the start conversion signal and the register strobe together so that the start conversion signal is also used as a pseudo end conversion signal. This does mean that the output will be delayed one period of the clock but this is only a minor problem that can be compensated for by programming.

The system's DMA control circuitry consists mainly of two sets of four 74LS191 presetable up/down counters. The first set is operated in count-up mode and is used to generate the addresses for DMA operations. When the CPU is held, the outputs of these counters are put on the address lines and they are incremented every time there is a start conversion signal. The other set is preloaded with the number of data points to be collected and then counts down with each start conversion signal until it reaches zero and then it interrupts the CPU.

The system has four memory chips. The first one is an 8k EPROM with the operating system on it. The next is an 8k RAM for variables, the stack and a ny temporary programs that might be loaded. The last two are set up for DMA operations to be preformed on them. One chip stores the most significant byte (MSB) and the other stores the least significant byte (LSB) of the A/D converter's data word. These two chips share the same address space when the CPU is held, but not the same data bus, but when the CPU is operating, one chip is in the 8k past the other and the two data busses are linked into one.

The USART (Universal Synchronous/Asynchronous Receiver/Transmitter) is the circuit's link to the outside world. The data received pin of the chip has been tied to the RST5.5 interrupt on the 8085A, so that whenever a command is received, it will be able to get the CPU's attention. This allows the chip to be stopped in the middle of an operation.

The timing circuitry is the most complex part of the circuit. The heart of the circuitry is the 8253-5 programmable interval timer. This chip has three 16-bit gated, presetable repeating down counters. The outputs of these counters are tied to the clock input of the next counter, with the first tied to the 2MHz system clock. Timer one has a 2MHz input, timer two gets its input from timer 1 and timer 3 gets its input from timer 2. This gives the chip the range of a 48 bit counter, but more flexibility, because the outputs of these counters are also multiplexed so that the counter that the start conversion

signal is derived from may be chosen from among the three. The fourth input to the multiplexer is taken from the device select logic for the A/D converter, so that a conversion may be started by the CPU directly. The gates for the counters are active high and the flip/flop that controls this may be set either from the remote start signal, or from a bit in the status register. The flip/flop is reset when a terminal count is reached in the DMA counter for the number of points to collect, or when it is reset by setting the status register bit to zero. The final piece of the timing circuitry is the status register. This important register is used to set the modes of operation for the circuit. Its contents are as follows:

Bit:	Use:
0,1:	Used as address for multiplexer to get start conversion source.
2:	1 allows remote start for timer, 0 remote start gives a CPU interrupt.
3:	Start timer
4:	DMA/ $\overline{\text{CPU}}$.
5:	Remote start mask: 0 blocks remote start completely.
6,7:	Unused

Bits 2 and 5 work in conjunction with each other where bit 5 will block the remote start signal completely, but when it is set to allow the signal in, bit 2 will select what happens with the signal, a 1 starting the timer and a 0 interrupting the CPU. Bit 3 will start the timer regardless of what else is on. Bit 4 selects what happens when the timer starts. If it is a 1, then the CPU is held and a DMA operation will be performed to collect the data, otherwise the start conversion signals will also be sent as CPU interrupts so the CPU will be able to collect the data from the A/D converter directly. Finally bits 0 and 1 are used to encode the multiplexer, a 00 being the device select for the A/D converter and the other numbers being the respective timer registers on the 8253-5 timer.

Buffered Data Collection Process

The multiple data sampling rates of the buffered data are achieved through programming. First the timer registers are each loaded. The first timing rate is put into timer register one, the second into two, and the third into three. Then the number of points to be taken is loaded into the DMA register and the DMA address register is set to the beginning of the DMA memory. Next the status register is loaded to allow remote start on the timer and to select DMA operation and finally start conversions from timer 1. When the remote start comes, the data will be collected until the end count is reached and the CPU will be interrupted. The end count register will now be loaded with the number of points to be collected at the second rate and then the status register will be loaded. DMA will be selected with the start conversions from timer 2, but this time the *start timer* bit will be set so that as soon as the register is loaded the timer will start and the CPU will be held again. Now data collection will proceed and when the terminal count is reached, the CPU will

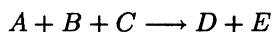
be interrupted again. Now if a third set is to be taken the process will be repeated but with data for the third collection rate. The neat part of this system is that the address register is only loaded at the beginning of the process, so that it will continue to be incremented as the data is taken, but the first point taken in a set of data will be right after the last point taken in the previous set because the register still contains the old number. After the last set is taken the data will be sent out to the Rainbow.

Data Analysis

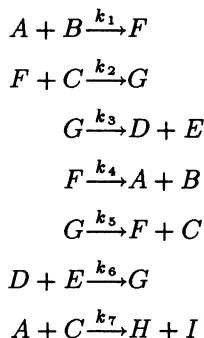
The program that was written to analyze the data is in **FORTRAN-77** and uses routines from **IMSL** to integrate the equations and **RGL (ReGIS Graphics Library)** to make the graphics output on the Rainbow.

Some Basic Chemistry

In order to understand the program it is necessary to first understand how equations for the rates of chemical reactions are derived. First, all chemical reactions can be broken down into a series of steps which describe the reaction. These steps describe the interactions between each of the species in the reaction. For example, take the following reaction:



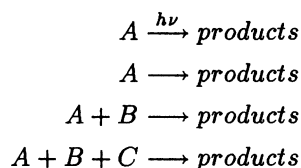
This equation means that reactants A, B and C combine to give products D and E. This might be broken down into the following mechanism:



Reactions 1 to 3 are the basic mechanism which gives us the products but there are other processes which also occur. Reactions 4 to 6 are the reverses of 1 to 3, and 7 is a reaction that uses the reactants up, but does not contribute to the system of interest; this is a competing or side reaction. All of these things must be taken into account when developing chemical mechanisms. The k 's over the arrows are called the rate constants. This is a proportionality constant that helps tell how quickly each step of the mechanism proceeds with respect to the others.

These mechanism steps are important because they can be easily converted into differential equations showing

the rate that the step proceeds at. There are four types of mechanism steps:

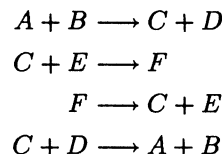


These may be converted to a rate equation as follows:

$$\begin{aligned} v &= \Phi I_a \\ v &= k[A] \\ v &= k[A][B] \\ v &= k[A][B][C] \end{aligned}$$

In these equations the v is the rate of each step in the mechanism above. The first equation is a photochemical reaction so it is different from the rest. Here the rate is proportional to the amount of light absorbed (I_a) and the *quantum yield* (Φ) which is the ratio of the number of molecules that react to the number of photons of light absorbed. The other ones are easier. The rate of the step is proportional to the product of the concentrations (denoted by the square brackets) of the reactants. Since the rate for each step is proportional to the concentration of the reactants, the rate constant converts this proportionality to an equality. The rate constants are very important quantities because a mechanism step does not depend on what mechanism it is in. If the rate constant for a particular mechanism step can be found in one mechanism it will be exactly the same if that mechanism step is found in an entirely different chemical reaction.

Now that we have the rates for each step in the mechanism we can find the rate of change of each species in the reaction. This is done by adding together the rate of all steps where the species is formed and subtracting the rate of all steps where the species is used up. For example, take this sample mechanism:



With rate equations:

$$\begin{aligned} v_1 &= k_1[A][B] \\ v_2 &= k_2[C][E] \\ v_3 &= k_3[F] \\ v_4 &= k_4[C][D] \end{aligned}$$

From this we can see the rates of change for the some of the species in the reaction would be:

$$\begin{aligned} \frac{\partial[A]}{\partial t} &= -v_1 + v_4 \\ \frac{\partial[F]}{\partial t} &= v_2 - v_3 \\ \frac{\partial[C]}{\partial t} &= v_1 - v_2 + v_3 - v_4 \end{aligned}$$

These are a set of coupled non-linear differential equations which can be numerically integrated, using the Gear method because of the sizes of the terms involved, to give concentration versus time for each species in the reaction.

Now that concentration versus time data can be generated for a given mechanism and set of rate constants, we can compare this with the experimental data for the same reaction. When we find agreement between the two this shows that we have discovered a plausible mechanism and the corresponding set of rate constants. This approach is used in the program to solve for the mechanism and rate constants. One puts in a proposed mechanism and rate constants, the computer integrates the differential equations corresponding to that mechanism and then plots both the experimental data and the calculated data on the screen. The mechanism and rate constants can then be modified until there is agreement between the two.

Running the Program

In order to use the program, the experimental data must first be loaded into VAX data files. For photochemical reactions, a VAX file must also be created giving the light intensity versus time profile for the incident light. After this is done the program may be run. It is menu driven to make it easy to use. A mechanism, a set of rate constants, and the initial concentrations are entered into the program. They may be saved for later use. The program will then automatically calculate the differential equations so that they may be integrated. This is an improvement over other kinetic simulation programs where the differential equations are coded into subroutines of the program and every time the mechanism is changed that subroutine has to be rewritten. When the program is run, it will plot both the experimental and simulated concentration versus time data for any chemical species in the reaction. Changes may be made to the mechanism and the data may be replotted, all interactively, until agreement is reached between the experimental and simulated data. The program also gives hardcopy of the graphs and the mechanism, as well as tabular results of the data.

How the mechanism is stored

The data structure that was developed for storing the mechanism is interesting. It consists of a series of arrays where *maxm* is the maximum number of steps that can be stored in the mechanism:

```

rtype(maxm):   Type of reaction step.
nlhs(maxm):   Number of reactants in step.
lhs(4,maxm):  Internal code for each reactant
                 in the step.
nrhs(maxm):  Number of products in step.
rhs(4,maxm): Internal code for each product in
                 the step.

```

The data structure is simple and it allows a general routine to evaluate the velocities (*v*) for each step as follows:

```

function evalv(eqn, j, i,t ,c)
implicit none
integer eqn, j, lp
C eqn is the step being evaluated
double precision t, c(j), i
C t is current time, c(j) is concentration
C of each species in the reaction
external i
C evaluates light intensity at time t
include 'commondef.for/nolist'
C data structure for mechanism
goto(10, 20, 30, 40), rtype(eqn)
write(6,*)' Error, no mechanism step'
write(6,*)'type ',rtype(eqn)
stop
10 evalv= k(eqn) * i(t)
return
C photochemical
20 evalv= k(eqn) * c(lhs(1,eqn))
return
C single reactant
30 evalv= k(eqn) * c(lhs(1,eqn) *
* c(lhs(2,eqn))
return
C two reactants
40 evalv= k(eqn) * c(lhs(1,eqn) *
* c(lhs(2,eqn) * c(lhs(3,eqn))
return
C three reactants
end

```

This function will return as its value the velocity of the mechanism step specified. It may be called by another routine (see below) that evaluates the first derivative of the function called by the DGEAR integration routine in IMSL. This subroutine is defined as follows:

```

subroutine evmech(j, t, c, dc)
integer j, q, r
double precision t, c(j), dc(j),
* v, evalv
C c(j) is concentration of each species
C dc(j) is the first derivative returned by
C this routine.
C t is current time
external evalv
C function for velocity of mechanism step
include 'commondef.for/nolist'
C mechanism and other information
do 5 q=1,j
5 dc(q)=0.0d0
C zero array
do 10 q=1,m
v=evalv(q, j, t, c)

```

These same techniques can be applied to the process of evaluating assignments. When the material is due, it can be collected electronically with the date and time included. The speed and accuracy of collection and evaluation provides better and more timely feedback to the student.

The ease of comparison of students programs makes it easier for the instructor to identify duplicate code, indicating undesirable collusion, while still encouraging students to work together to help each other learn.

A spread sheet is maintained in the "leader account" that is linked to one in the student library. The student can examine a complete and up-to-date progress report at any time without access to the source record.

Course Preparation of a Computer Course

Tests, handouts, syllabi, problems, solutions, and sample programs are prepared by instructors using the editor - word processor and the language interpreters and compilers. The heirarchical directory structure encourages the cataloging of this material in appropriate subdirectories. The instructor accounts are assigned in groups by subject matter. The normal default access protection is to share the reading and executing of all files and directories with the other member instructors in the staff account group. However, protection can be changed when privacy is desired. The normal group access encourages the sharing of materials. Team activities frequently evolve. These can be formalized for better organization by assigning a coordinator to multi-section classes. The team approach better reaches the part-time and night time instructor. The syllabus and handouts with dates, recommended deadlines, and grading standards are collected and prepared by the course coordinator and distributed electronically to the individual instructors. These can be edited, added to and tailored by each instructor.

Common problems can be detected and dealt with as a group without the necessity of scheduling a group meeting.

These management activities are a bonus in that they place little, if any, burden on the computer system. Most of the activities take place at off peak times when there is no contention for ports. Instructors with modems can do much of their activity at times when there are few, or no, other users on the system.

Some of these activities can be performed by stand-alone micro computers, but many of the benefits are lost without the communication and central file storage system provided by a minicomputer. The specifics required for an operating system to support the endeavors described above are: Multi-tasking capabilities, heirarchical directories, and a modifiable command language such as UNIX or VMS with adequate protection schemes. The new wave of micro computers based on 32 bit processors should provide superb and cost effective delivery systems.

APPENDIX

Managing a Class Leader account

Example: Leader account name [BASQ.BASQ00]
Student account names [BASQ.BASQ01] to [BASQ.BASQ30]
Library access LIB:
Master library CPSSLIB:

Local VMS commands

UP
DOWN
NEXT
HOME
CAT

Commands for viewing student directories

\$ DIR or CAT [BASQ.BASQ*]	All accounts in group
\$ DIR or CAT [BASQ.BASQ10]	Student number 10
\$ DIR or CAT [BASQ.BASQ*]*.BAS	All source code files
\$ DIR or CAT [BASQ.BASQ*]*.EXE	All executable files

Commands to view student files

\$ TYPE [BASQ.BASQ*]*.*	All files
	(Unwise to type EXE files)
\$ TYPE [BASQ.BASQ*]*.BAS	All source code
\$ TYPE [BASQ.BASQ*]*.OUT	All output files
\$ TYPE [BASQ.BASQ*]PROB1.*	All PROB1 files
\$ TYPE [BASQ.BASQ*]PROB1.OUT	PROB1 output files

Commands to check the operation of a student file

\$ RUN [BASQ.BASQ10]PROB1.EXE	Runs a student program
\$ PRINT [BASQ.BASQ*]PROB1.BAS	Prints students source

Commands for checking errors by bringing students source file to the instructors account.

```
$ DIR [BASQ.BASQ10]
$ COPY [BASQ.BASQ10]PROB1.BAS PROB1_10.BAS
$ BASIC
```

```
Ready
RUN PROB1_10.BAS
```

The above commands print to the screen. A file of this output can be created for later printing by using a simple command procedure that redirects the screen output to a file. This process is controlled by assigning a short symbol to run the command procedure.

FIGURE 2


```

$! Procedure directs system output of commands made in default
$! directory to a temporary file in the default directory then
$! restores the terminal to normal operation.
$! by Claude M Watson   May 2, 1985
$!
$ DEFINE SYS$OUTPUT TEMP_FILE.PRT !Assigns system output to a file.
$ SET TERM/NOANSII/NODEC          !Sets terminal to ignore escape
                                   !sequences
$ 'P1' 'P2' 'P3'                  !Accepts up to three words
$ DEASSIGN SYS$OUTPUT             !Returns system output to terminal CRT
$ SET TERM/ANSII/DEC=2           !Resets terminal to recognize escape
                                   !sequences

```

Sample of files in LIB: the 9th week of the Winter Term 1987.

```
$ @PRINTSCREEN CAT LIB:
```

```
$ PRINT TEMP_FILE.PRT
```

```
Directory L$CPS_[L$CLS.L$BASQ]
```

BUBBLE.EXE;1	10/10	31-MAY-1986 17:41	(RWED,RWED,RE,RE)
LAB5.DIA;2	3/3	13-FEB-1987 15:19	(RWED,RWED,RE,RE)
LOGIN.COM;1	1/1	12-MAY-1986 08:25	(RWED,RWED,RE,RE)
MESSAGE.DOC;16	2/2	9-MAR-1987 12:10	(RWED,RWED,RE,RE)
MESSAGE1.DOC;1	1/1	9-JAN-1987 15:48	(RWED,RWED,RE,RE)
MESSAGE2.DOC;1	1/1	5-FEB-1987 11:00	(RWED,RWED,RE,RE)
MESSAGE3.DOC;1	2/2	6-FEB-1987 07:52	(RWED,RWED,RE,RE)
MESSAGE4.DOC;1	2/2	16-FEB-1987 11:49	(RWED,RWED,RE,RE)
MESSAGE5.DOC;1	1/1	17-FEB-1987 13:36	(RWED,RWED,RE,RE)
MESSAGE6.DOC;1	1/1	27-FEB-1987 11:46	(RWED,RWED,RE,RE)
OLD.DIR;1	2/2	7-JAN-1987 14:04	(RWE,RWE,RWE,RE)
PIC2.DAT;1	14/14	21-MAY-1986 08:27	(RWED,RWED,RE,RE)
PIC3.DAT;1	13/13	21-MAY-1986 08:27	(RWED,RWED,RE,RE)
PIC4.DAT;4	13/13	10-JUN-1986 12:40	(RWED,RWED,RE,RE)
PIC5.DAT;3	15/15	10-JUN-1986 12:41	(RWED,RWED,RE,RE)
PIC6.DAT;3	9/9	10-JUN-1986 12:42	(RWED,RWED,RE,RE)
RESUME.QUES;1	1/1	9-JAN-1987 15:53	(RWED,RWED,RE,RE)
SAMPLE.BAS;1	2/2	10-FEB-1987 09:56	(RWED,RWED,RE,RE)
SES.EXE;1	9/9	25-MAY-1986 12:20	(RWED,RWED,RE,RE)
SORT.EXE;2	20/20	19-MAY-1986 09:17	(RWED,RWED,RE,RE)
STRING2.S86;10	17/17	5-JUN-1986 12:13	(RWED,RWED,RE,RE)
STR_EXAMP.BAS;11	4/4	2-JUN-1986 10:00	(RWED,RWED,RE,RE)
STR_EXP.BAS;3	7/7	4-JUN-1986 15:01	(RWED,RWED,RE,RE)
STR_EXP.EXE;1	10/10	4-JUN-1986 16:22	(RWED,RWED,RE,RE)
T2_Q22.BAS;1	4/4	16-FEB-1987 16:14	(RWED,RWED,RE,RE)
TAB.DAT;1	11/11	16-NOV-1986 21:28	(RWED,RWED,RE,RE)
TRANS.DAT;4	1/1	30-JAN-1987 10:26	(RWED,RWED,RE,RE)

Total of 27 files, 176/176 blocks.

FIGURE 1

FACULTY RETRAINING

Edward A. Boyno
Montclair State College
Upper Montclair, New Jersey 07043

Suggestions on programs for retraining faculty in Computer Science from someone who has been through such a program.

There are 21 colleges and Universities of the state of New Jersey ranging from Princeton University on down to unaccredited schools, and a score of two year institutions. For reasons that are similar everywhere, they have difficulty attracting and retraining instructors of Computer Science. During the past 10 years, enrollment in their Computer Science programs has mushroomed. The only staffing solution available before 1984, and still the most used solution, was the employment of adjuncts for many of its courses. A solution that I suspect is widely used elsewhere. Because of the number of "high tech" industries in our region (Bell Labs, for example) as well as a number of pharmaceutical and insurance companies we are, in general, able to get high quality part-timers, but the situation is still not good.

In 1983, The Department of Higher Education (DHE) proposed to increase the pool of available, full-time, Computer Science instructors by retraining certain of its existing faculty. I was a participant in the first year of the "Faculty Retraining Program" (FRP), an admitted guinea pig. It is this program that I'd like to speak about today.

I hold a Ph.D. in pure mathematics from Rutgers University. Prior to the summer of 1984 I had had zero experience with computers and I was (and am) a full-time faculty member at Montclair State College

a former "teachers' college" that now offers a full range of liberal arts courses and grants masters degrees in several areas including computer science. I am exactly the person that the FRP was aimed at (I almost said "designed for" but "aimed at" is more accurate). I was one of thirty persons that began the program. All but three of us held a Ph.D. in one of the natural sciences, the other three held masters degrees in mathematics. All but three of us held tenure, and almost exactly half of us were in the senior ranks. We were not a shabby group.

About half were mathematicians, and most had had no previous experience with computing beyond the hobbyist level. Its fair to say that I was the modal student (if not a model one).

The program consisted of a fifteen month course of study from June 1984 to August of 1985 held at Stevens Institute of Technology in Hoboken, New Jersey resulting in an MS degree. The curriculum was developed by the DHE in conjunction with Dr. Lawrence Levine, the programs director, and the Computer Science faculties at Rutgers and Stevens.

Stevens is a Ph.D. granting institution that is widely reputed, at least in the East for its Engineering and Applied Mathematics programs. It is one of the Colleges and Universities that have a special relationship with DEC. Every incoming freshman must purchase a PRO-350, at a very low price, and more

recently, they've received a grant to, essentially, network the entire campus. In passing I must note that Stevens possesses THE most spectacular view imaginable of the New York skyline.

As you might guess, the hardware available to us was exclusively from DEC. Levine had wangled the exclusive use of nine 350's for us. There were several other 350's one of which was connected to a VAX, used by other Levine students for CAI development and were occasionally had accessible to us. We were also given student accounts on a DEC-10 that is available for instructional purposes.

The courses we took were with a couple of notable exceptions usual graduate courses that is we weren't spoon fed special courses. We were, however placed in special sections, took our courses in lock step and (importantly, it turned out) were not given the freedom to choose our instructors.

The faculty for the program for the most part full-time Stevens staff (for whom, though, we were well paid overload). We did have two adjuncts, one of whom was a "regular" part-timer, the other being a special import.

For the duration of the program, a variety of undergraduate students were available to help us learn the machines we used and with programming difficulties. I'm going to have some not-particularly-pleasant things to say about Stevens, but no criticism at all can be attached to these "kids". They were uniformly pleasant, bright, knowledgeable and PATIENT.

Last, but by no means least, is the method in which the program was financed. Tuition at Stevens for an MS degree is something over \$10,000. The DHE, fearful that we would take our degrees and run, refused to grant us the money preferring to LEND it

to us and then to forgive the loan at the rate of 25% per year, thus binding us to the State for four years. When I joined the program, I did not understand the nature of the fiscal responsibility I was accepting, believing the assurances of the DHE and Dr. Levine that all would go well in the long run. It was not until a month after we had all signed promissory notes for the first half of our tuition that the excrement hit the fan. The awakening came in the nature of the loan forgiveness agreement that the DHE had promulgated. It seemed that if FOR ANY REASON we were unable to complete the program in the allotted time, we would become liable for the entire loan. Completion of the program was defined to be the successful receipt of the MS degree. There was absolutely no provision made for someone leaving the program because of illness or family emergency. There was no provision made for people who simply found the program unsuitable to their needs. There was no provision for anything! Needless to say, we were more than a little unhappy with this and negotiation did produce a marginally acceptable agreement, but I still can't go on sabbatical without risking abrogation of the agreement. If there's any lesson to be learned from the Stevens program it is that other ways must be found to finance such programs if faculty are going to join them. I might add that in a second cycle, when all the monetary facts were known to the participants AND an escape provision that we did not have was added, only twelve people joined the program. A third cycle has had to be cancelled.

As I begin my discussion of the program per se, I'll ask you to keep in mind that, at all time, all of us were aware that if we washed out, it would have been in a sea of red ink.

For me, and I reiterate that I consider myself the absolute average participant, the fifteen months that I spent in the program were a nightmare. A year later I can see the worth of what I've received much more clearly, but the rate of return will have to accelerate if I'm ever to recover the physical and psychical costs.

It is clear to me that I myself was responsible for at least some of my problems. Most mathematicians view computer science with disdain. I've learned the hard way that it's a discipline of its own. However, in June of 1984 I still thought that I would breeze through the next year or so with little if any effort, I would even go so far as to say I was anticipating a year and a half of vacation. Thus, I was completely unprepared for the shock of being a student again.

I resented having to fight with other students for parking places. I'd forgotten what it's like to wait on line for an hour at a college bookstore, or what it's like to argue with a clerk in the business office over an error in a bill. I learned the joy of being all ready to get to work only to find that the terminal room was locked and the only secretary who had a key had just left for a two hour lunch.

I found it terribly hard to have to study something that I wasn't an expert at. Submitting to someone else's classroom discipline was also very hard and taking tests again was dreadful experience, especially when I got a "B" on my very first one. In short I HATED being a student.

Any program of this sort is going to involve a certain amount of "Student Shock Syndrome" but in our case absolutely nothing was done to alleviate it. If good financial arrangements is lesson number one of Retraining programs, then making provi-

sion for easing the participants back into the classroom must be lesson number two. I understand that the original plans for such programs called for counselors to be available to minimize ego damage we certainly could have used one.

Let me turn my attention for a second to the actual course of study that we followed. As I said, it's been a year now, a year in which I've taught several Computer Science courses, and I look back on the curriculum with somewhat more expert eyes.

Our first summer was designed to "bring us up to snuff" mathematically and computer-scientifically(?). We were given courses in discrete math structures, probability and "Intro. to Computer Science I and II", one of only two courses specifically created for us. What we weren't given was a course in programming! The "Intro." course did contain a small programming component but for the most part we were expected to learn PASCAL on our own. Lesson number three: give the retrainees a programming course. The lack of same haunted many of my fellows for the rest of the program.

In the fall of 1984 we took three courses: Data Structures, Machine Architecture and Programming Languages I. I have no complaints about any of these but in hindsight it would have been much better to eliminate the ill conceived summer courses and replace them with a real programming course and Data Structures. The only exposure to assembly language programming I've had to date was tiny little bit of "MACRO 10" (The "quaint and curious" MACRO 10 as one of my colleagues called it) in the "Architecture" course and while I haven't yet noticed a pronounced gap in my background, I really wish we'd been given a real course in a "real" assembly language. Typical of the program as well, the assembly

language segment of the curriculum which could have come almost anytime after the first two weeks, was postponed until the last three weeks to ensure that we'd have the maximum competition with other students for machine time!

The spring of 1985 was consumed by Finite Automata, Operating Systems and Programming Languages II. Which later turned out to be a course in Compiler design. In that semester, I taught two graduate courses, wrote an operating system for a simulated machine and wrote a compiler for a subset of PASCAL. It nearly killed me.

Lesson number four: Give faculty full released time for the retraining program.

Lesson number five: If you violate lesson number four at least organize the courses so that the programming intensive course don't all fall in the same time period.

Spring '85 was very bad time for me and many of my colleagues. The stress had accumulated to a terrible level. As bad as my description of this semester might sound, it was much worse.

To explain some of our distress, I have to spend a few minutes talking about the men who staffed our program. I really expected that Stevens would trot out their best and brightest faculty for this program ... after all, I'm in a position to counsel students on their choice of graduate programs, so if for no other reason than advertising, I expected to be dazzled. What we got was pretty much a cross section of their faculty. Two were absolutely brilliant, others were good, some were awful. We saw an instructor who sometimes seemed uncertain of his SUBJECT matter and another who no more belonged in the classroom than I belong in the Green Berets. We had an instructor who threatened

to fail some of us if we didn't stop complaining about him. We had an instructor who put a question on an exam that couldn't be answered with the given information (his solution for the problem was blatantly fallacious). A couple of our instructors literally played favorites. Pet students didn't have to do all the work that the rest did. In another course, all of programming assignments were graded by other, "regular", graduate students and I'll go to my grave believing that this one, mercifully anonymous person, assigned grades by counting the number of comments in the program. I hope you can understand that we often had the feeling that grades were being distributed in an essentially random manner. If you recall, we could be expelled from the program (thus assuming what was by now a debt of over ten thousand dollars) if, for any reason, we didn't complete the masters program on time! At the time, I considered it completely possible that one of these capricious instructors would give me an F for no other reason than that I had been a very vocal critic of the program. (By this time I had already published a highly critical article in the New Jersey AAUP newsletter). Some of my colleagues were struggling along with exactly "B" averages, for them a "C" would have been fatal. Let me add that some of my impressions of the program have softened after a year or so, but I'm still convinced that the grading at Stevens Tech is a stochastic process.

Lesson six: Choose your faculty very carefully.

I can't emphasize this too strongly. A classroom full of experienced college faculty members is a far different audience than the run of the mill graduate class. We were very demanding consumers.

We could spot lack of preparation, bluffs, laziness and all of the other bad habits that poor instructors fall into, and, as I've alluded to, we complained loudly. Moreover, our weaknesses were not the weaknesses of a "normal" class nor were our strengths. One instructor constantly berated members of my group for being poor programmers but made no attempt to improve our programming skills, another spent hours teaching us Boolean Algebra when it was clear that most of us knew a lot more about it than he did. The people at Stevens were clearly unprepared for a class such as ours. Very little was done to shore up our weak points and nothing was done to take advantage of our considerable strengths. In fact, I think it's fair to say that they seemed surprised we possessed skills that might be of use to the program.

Lesson seven: Tailor the program to the students.

Let me return to my main narrative. Most of us did survive the Spring of 1985 despite the fact that we often felt like the characters of Sartre's No Exit.

Summer of '85 was an anti-climax. Levine had managed to provide a selection of courses for us to choose from, allowing us to select two of "Expert Systems", "Database Management", "Computation Chemistry" and for the mathematicians, "Program Verifiability and the Theory of Computation" (the second of the specially provided courses and an absolutely brilliant course it was). We finished the program, again in lock step, in "Systems Programming" and "Algorithms". As before, I've since learned the value of these courses and have no quarrel with anything except the fact that they came last. The "Systems Programming" course probably should have

come before either the "Operating Systems" or the course in compilers.

This last leads me naturally to the most serious flaw in the program (after the financial arrangements) the overall lack of coordination and planning. In the course of fifteen months I wrote no fewer than four machine simulators (interpreters) for four different courses. It seems to me now that it would have been so easy to agree on a simulator so that students could write practice operating systems and the like and to use that simulator for the entire program building on it as they went along. I know of at least one such simulated machine that is specifically designed for that purpose.

I'm going to hazard a final piece of advice:
Lesson eight: Plan carefully.

There are many substantive questions that have to be asked and answered before any project of this sort is undertaken and I'd like to pose some of them for you. What ought to be the most obvious (but did not seem to occur to any of the people involved in planning the Stevens project) is "What sort of Computer Science are we talking about?". At Rutgers, Stevens and in fact at Montclair State, our programs are highly mathematical as opposed to what one might call Data-Processing-intensive. It's very clear to me now that the lack of mathematical background was a great handicap to the non-mathematicians in my group. Those of you with training in mathematics will understand that there is a "mathematical" way of thinking about things that many "laymen" have difficulty acquiring. (It is my personal belief that the ability to think "mathematically" is a talent like drawing or hitting a baseball that some very intelligent and/or gifted people just don't have.) There is some mathematical component

in any reasonable Computer Science curriculum but I'm sure that it could be minimized.

This first issue leads me directly to the second: Which faculty will you retrain? There are a good number of college presidents right now who would love to turn their Geography departments into a nest of computer scientists but even if the Geographers were willing, I'd question whether the transition could be successful on a side scale. Most Social Scientists and experts in the Humanities that I know simply don't have the mathematical background to just jump in and study Computer Science. I can see a Computer Science program with no course in Finite Automata and maybe you could avoid a course in algorithms, but I can't see leaving out Data Structures and I can't see Data Structures without very mathematical things like "Trees" and "Recursion".

Rumor has it that the DHE originally planned to invite ANY faculty member in the state to join the program. Mercifully, wiser heads prevailed!

Another important consideration, I think is the matter of awarding degrees. Few of the programs that I've seen offered to do so. A year ago I would have agreed that the "training was the thing", but now I'm not so sure. I'm glad to have the formal recognition of my accomplishment and Montclair State is pleased to display my new credential in its catalog. People in academia seem to place great store by letters after one's name.

Finally, and I always seem to wind up here, is the matter of paying for the whole thing. It seems to me that an organization that benefits from a retraining program ought to be the one that pays for it. That seems simple enough, but at least one of the people that started at Stevens with me but who

had to leave at the half way point is paying off a \$5000 bill.

It is my strongly held belief that any such program should be completely funded by the agencies that sponsor them. Books and other materials should also be paid for (mine were not) and the grant should be as free of strings as possible.

Using VAX/VMS to Teach Computer Organization

Linda Lankewicz
Spring Hill College
Mobile, Alabama

Abstract

VAX/VMS provides tools which enhance the teaching of Computer Organization and increase the likelihood of achieving the course objectives. These include the Debugger, the TPU editor, and System Services and Run Time Library routines. Students need concrete experiences when mastering material in a foundations course. Using these VMS features, the professor can provide students with sufficient materials so that they can grasp the details and have an opportunity to consider the broader picture of the operating system environment.

Computer Organization is the first upper division course taken by computer science majors. The course prerequisites are two programming courses in which students solve problems using a high-level language. Computer Organization introduces students to computer architecture and the machine instructions used to invoke activity within the framework of that architecture.

The course is a hurdle in the computer science curriculum which must be mastered before continuing in the program. Based upon their success in the course, students decide to major or minor in computer science or to change majors. Students consider the course difficult because of the amount of material that must be covered and because of the unfamiliarity of the subject. Faculty consider the course difficult to teach because the students' experience is limited and because there are few supporting instructional materials available. The professor would like to stimulate the students and challenge them without discouraging them.

While some students will take a second organization course later, the goal of the first course is to introduce students to the underlying organization of a computer. The first course covers

- the representation of information in the computer
- processor and memory structure
- assembly language programming
- the operation of the assembler and the linker

The tendency might be to spend most of the time on the syntax of an assembly language. One would prefer that students leave the course with more than that. When a subject has as many intricacies as assembly language, students become immersed in details. They tend to see the trees rather than the forest, intent on the brackets used in the displacement mode rather than considering the advantages of selecting one

addressing mode over another. While it is difficult to overcome this, students should be given the opportunity to reach a higher level of understanding.

The goal in a Computer Organization course is that the student understand the relationship between a computer's organization and programming. In reaching that goal the objectives for the student include the following:

- understand how information is represented in a computer system
- understand how simple data structures such as pointers, arrays, and stacks are implemented
- understand how programming features such as procedure calls, recursion, and macros are handled
- program in an assembly language

When devising course objectives, one should consider how the course fits into the curriculum. The goal of Computer Organization is not so much to provide students with proficiency in another language, but to give them the foundation for understanding a computer system which will be needed in the Data Structures, Organization II, and Operating Systems courses.

Students should leave a Computer Organization course with a clear picture of the utility of data structures such as pointers and stacks. In this course, students can see how these data structures are implemented on a machine. This view will be helpful when using the data structures in a high-level language. Students in Computer Organization should become comfortable with calling procedures and passing parameters from both internal and external modules including modules written in different languages. These ideas can be expanded later in the Operating Systems course to demonstrate principles of interprocess communication and synchronization.

Students need concrete experiences when mastering material in a foundations course. VAX VMS provides tools which

enhance the teaching of Computer Organization and increase the likelihood of achieving the course objectives. These include the Debugger, the TPU editor, and System Services and Run Time Library routines. Using these VMS features, the professor can provide students with sufficient materials so that they can grasp the details and have an opportunity to consider the broader picture of the operating system environment.

The Debugger is an excellent tool for programming at any level. Students should be introduced to the Debugger in their first programming classes. But the Debugger is especially effective for teaching Computer Organization. Using the Debugger at the beginning of the course when considering how information is represented in a computer allows students to see the twos complement representation of negative numbers; the binary, hexadecimal, and decimal representation of integers; and the difference between the ASCII, integer, and floating point representation.

The Debugger command to examine data in memory is EXAMINE. The qualifier /BYTE, /WORD, /LONG, or /QUAD can be used to specify the size of the memory to be examined. The qualifiers to specify the data type include /ASCII, /INTEGER, /BINARY, /HEX, and /FLOAT. The default display consists of integer longwords in hexadecimal.

In the assembler program shown below, memory is designated for the variables ATWO, ITWO, FTWO, NTWO which contain the ascii, positive integer, floating point, and negative integer representations of the number two.

```

atwo:  .ascii    /2/
itwo:  .long     2
ftwo:  .float    2
ntwo:  .byte     -2
;
      .entry    program, ^m<>
;
      movl     #1, r0
      ret
      .end     program

```

Using the Debugger, a student can observe how these representations differ. The command EX/BYTE/BIN ATWO displays the ascii two stored in ATWO. One byte is displayed in binary as 00110010. The command EX/LONG/BIN ITWO displays the longword integer representation of ITWO in binary while the command EX/LONG/BIN FTWO displays the floating point representation of the number two. The command EX/BYTE/BIN displays the twos complement representation of a negative two as shown in figure 1.

The Debugger can be used for class demonstrations interactively or by capturing a Debugger session. A captured session is safer when it is important to ensure that specific material is covered in class. The commands to capture Debugger sessions are listed below. The commands and resulting output of the Debugger session will be recorded in a file, and that file may be displayed for the class in an editor or printed on overheads.

```

DBG> SET LOG filename
DBG> SET OUTPUT LOG

```

Capturing Debugger output does not provide students with the full-screen display that is useful when stepping through machine code. The interactive use of the Debugger can stimulate discussion if students are asked "what if" questions about the outcome of Debugger sessions. For the source code shown below involving subtracting, incrementing, and adding values, students can discuss the outcome of each instruction while the professor steps through the code interactively.

```

w:      .byte    0
x:      .byte    127
y:      .byte    9
z:      .byte    11
;
      .entry    program, ^m<>
;
      subb3     z, y, w
      incb     x
      addb     y, z
;
      movl     #1, r0
      ret
      .end     program

```

When discussing the processor structure, interactive Debugger displays are useful because the contents of the registers can be seen. The register display on the upper right of the screen can be invoked with the command DISPLAY REG or with the keypad keys PF1 7. Registers 0 through 15 are displayed along with part of the stack and the Process Status Word (PSW).

While stepping through code, students can see how the Program Counter (PC) maintains the location of the next instruction. In the Debugger display shown in figure 2, the PC is pointing to address 0000020D which is the next instruction to be executed as indicated by the arrow in the source code.

Also apparent is the use of register 0 for a status code at the end of each program. Moving the number one into R0 indicates the successful completion of a program. Students can move other numbers into R0 to see the resultant error messages. They begin to understand that their source code is a routine that executes within a larger framework of the operating system.

A classroom demonstration of how the PSW bits are set when negatives, zeroes, overflows, or carries are encountered helps students understand how branching is accomplished. For the same source code shown above, the register display provides information as to the setting of the PSW bits. Students can see the bits change as each instruction is executed. Seeing the effect of each instruction on the PSW bits leads to a discussion of branching. In the example shown above, the subtraction instruction has been executed subtracting 11 from 9 giving -2. The negative and carry bits have been set. A branch-if-negative (BNEG) instruction at this point would result in a branch because the N bit of the PSW is set.

The Debugger is also useful for demonstrating how pointers work. Students can see that an address of a value is placed

```

-OUT -output-----
.MAIN.\ATWO:      00110010
.MAIN.\ITWO:      00000000 00000000 00000000 00000010
.MAIN.\FTWO:      00000000 00000000 01000001 00000000
.MAIN.\NTWO:      11111110

```

Figure 1: Debugger Output

```

INST -scroll-instruction          REG
00000206: SUBB3  B^.MAIN.\Z  |R0:00000000  R10:7FFEDDD4  @SP:00000000
>0000020D: INCB  B^.MAIN.\X  |R1:00000000  R11:7FFE33DEC  +4:00000000
00000210: ADDB2  B^.MAIN.\Y  |R2:00000000  AP :7FF473CC  +8:7FF473CC
00000215: MOVL  S^#01,R0  |R3:7FF47394  FP :7FF47384  +12:7FF473BB
00000218: RET                               |R4:00000000  SP :7FF47384  +16:000008A7
                                           |R5:00000000  PC :0000020D  +20:000005FF
                                           |R6:7FF47049  @AP:00000006  +24:00000005
                                           |R7:0001E4DD  +4:7FFE6440  +28:00000204
                                           |R8:7FFED052  +8:7FF9802C  +32:00000000
                                           |R9:7FFED25A  +12:7FFE640C  +36:00000001
                                           |N:1    Z:0    V:0    C:1  +40:0000000D

```

Figure 2: Debugger Output

in a register or in memory and referenced indirectly. Often students have a vague idea of how pointers work when using a high-level language. Once they view pointers at the Debugger level, they understand the concept.

The advantage of using pointers can be shown with an example using arrays. Students might be asked to consider the addition of the contents of two arrays. They might suggest adding each item in one array to the corresponding item in the second array as shown below. For word arrays containing 100 items, it would be necessary to have 100 ADDW instructions. (See figure 3)

Then students can be shown how to accomplish the task by using registers to hold the addresses of the arrays as shown in figure 4. The arrays are added by referring to them indirectly using the registers. Students can see that the amount of code is reduced since each array item can be referenced in a loop by the register name rather than by individual memory locations. The advantage of using a pointer becomes apparent since incrementing the pointer enables one to reference the entire array.

The source code for the above treatment of arrays is shown below. Stepping through this code with the Debugger allows students to see how R0 is used as a counter for looping, how array addresses are stored in R1 and R2, and how these pointer registers are incremented by two in order to reference the next words in the arrays.

```

num=4
array1:  .word    5,2,8,9
array2:  .word    4,6,9,2
;

```

```

                .entry  program, ^m<>
;
start:  movw    #num, r0
        moval  array1, r1
        moval  array2, r2
;
loop:   addw   (r1), (r2)
        addw   #2, r1
        addw   #2, r2
        decw   r0
        bgtr  loop
;
        movl   #1, r0
        ret
        .end   program

```

This demonstration can be altered to cover autoincrement, indexed, and displacement addressing modes. For example, the ADDW instruction shown above can be altered to ADDW(R1)+, (R2)+ for autoincrement mode. Students have difficulty perceiving why addressing modes are necessary. Using the Debugger the professor can present a natural progression of addressing modes for handling large blocks of information.

One problem associated with teaching assembly language is how to accomplish I/O before students have been introduced to macros or procedure calls. I/O routines can confuse the issues at early stages of the course. Students need to examine contents of memory and registers, to view results in binary or hexadecimal, and to step through a program to see branching in terms of the PC and the PSW. I/O routines alone would not

ARRAY1	ARRAY2			
5	4	ADDW	ARRAY1,	ARRAY2
2	6	ADDW	ARRAY1+2,	ARRAY2+2
8	9	ADDW	ARRAY1+4,	ARRAY2+4
9	2	ADDW	ARRAY1+6,	ARRAY2+6
.	.	.		
.	.	.		
.	.	.		

Figure 3: Arrays

	ARRAY1		ARRAY2
R1: address of ->	5	R2: address ->	4
of	2	of	6
ARRAY1	8	ARRAY2	9
	9		2
	.		.
	.		.
	.		.

Figure 4: Arrays

provide these experiences.

Even after students use macros or subroutine calls for I/O, the Debugger is useful for demonstrating the following:

- subroutine branches vs procedure calls
- referencing arguments by AP or SP
- passing parameters by value, by reference, or by descriptor
- internal and external routines

These ideas are important for later courses. A Data Structures class may require that a user stack be created for the passing of parameters. An Operating Systems course may demonstrate the readers and writers algorithm for interprocess communication by calling system routines. Students understand these concepts better when they have worked with them at the Debugger level.

In the source code shown below, four arguments are pushed onto the stack: the address where the result will be stored, the number 4, the number 3, and the number 2. The subroutine CALC is used to multiply the first two arguments, add this to the third argument, and store the result in memory.

```

result:  .long
        .entry  program, ^m<>
;
        pushal result
        pushl  #4
        pushl  #3
        pushl  #2
        jsb   calc
        addl  #16, sp

```

```

ret
;
calc:
mull3   4(sp), 8(sp), r6
addl3   r6, 12(sp), @16(sp)
rsb
.end    program

```

The arguments are referenced using the SP. The three numeric arguments are referred to as 4(SP), 8(SP), and 12(SP). In the Debugger display shown below, students can see why these arguments are referenced in this manner. They see that the JSB command causes the address needed for the return from the subroutine to be placed on the stack so that the first argument would be 4(SP). The use of @16(SP) for storing the result could be confusing, but the Debugger display shows that 16(SP) contains an address for the result, making it necessary to use @16(SP).

The PC in the display in figure 5 is at 21F. The return from the subroutine (RSB) is accomplished by popping the return address 215 from the stack. After the return from the subroutine, the instruction ADDL #16,SP is used to change the SP rather than popping the arguments off the stack.

This same program can be altered to demonstrate the use of CALLG and CALLS. In the source code shown below, the code for CALLS appears to be almost identical to that using JSB. However, the differences become apparent when using the Debugger. The CALLS causes a call frame to be pushed onto the stack. This call frame contains the masked R6, the old FP and AP, and the address 216 for restoring the PC after returning from the CALC routine. The arguments which were on the stack are now referenced by the AP. Thus the multiply instruction becomes MULL3 4(AP),8(AP),R6.

```

INST -scroll-instruction          REG
00000204: ENTRY MASK ^M<>      |R0:00000000  R10:7FFEDDD4  @SP:00000215
00000206: PUSHAL B^.MAIN.\RE    |R1:00000000  R11:7FFE33DC   +4:00000002
00000209: PUSHL S^#04           |R2:00000000  AP :7FF473CC   +8:00000003
0000020B: PUSHL S^#03           |R3:7FF47394  FP :7FF47384  +12:00000004
0000020D: PUSHL S^#02           |R4:00000000  SP :7FF47370  +16:00000200
0000020F: JSB L^.MAIN.\CA     |R5:00000000  PC :0000021F  +20:00000000
00000215: ADDL2 S^#10,SP        |R6:00000006  @AP:00000006  +24:00000000
00000218: RET                    |R7:8001E4DD   +4:7FFE6440  +28:7FF473CC
00000219: MULL3 B^04(SP),B^      |R8:7FFED052   +8:7FF9802C  +32:7FF473B8
>0000021F: ADDL3 R6,B^0C(SP)   |R9:7FFED25A  +12:7FFE640C  +36:000008A7
00000225: RSB                    |N:0    Z:0    V:0    C:0  +40:000005FF

```

Figure 5: Debugger Output

```

result:    .long
           .entry    program, ^m<>
;
           pushal   result
           pushl    #4
           pushl    #3
           pushl    #2
           calls    #4, calc
           ret
;
calc:      .word     ^m<r6>
           mull3    4(ap), 8(ap), r6
           addl3    r6, 12(ap), @16(ap)
           ret
           .end     program

```

Using the Debugger students can see how returning from the CALLS causes the removal of the call frame from the stack and the restoration of the registers to their prior states. This idea is important in understanding how recursion and context switching are implemented.

A similar demonstration can be used with the CALLG procedure call. An argument list is created instead of using the stack. The CALLG instruction includes the names of the argument list and the called routine.

```

result:    .long
args:      .long     4, 2, 3, 4
           .address  result
;
           .entry   program, ^m<>
;
           callg   args, calc
           ret
;
calc:      .word     ^m<r6>
           mull3    4(ap), 8(ap), r6
           addl3    r6, 12(ap), @16(ap)
           ret
           .end     program

```

Students should know how a stack differs from an array after examining both data structures. They should be able to

discuss differences between implementation methods for subroutine and procedure calls. They should know how recursion works after they stepping through short recursive programs to see how call frames are pushed onto and popped from the stack.

In addition to the Debugger, VMS provides the TPU Editor, another tool for the Computer Organization course. In any programming environment, the editor should work hand in hand with the Debugger. At Spring Hill College our system manager, Glenn Bell, modified the TPU Editor to permit the use of the Debugger within the Editor. This improves the Debugger experience since students can modify assembler code and view the results in the Debugger without having to exit and enter the editor repeatedly. While in the Editor, the code can be compiled and linked with or without the DEBUG option. Additionally, the programmer can spawn a process to pop out of the Editor to read mail or word process, then pop back to the same location in the Editor.

This modification of the TPU Editor simplifies the edit-debug experience for the student and encourages the use of the Debugger. Students can modify code in multiple windows and view the effect in the Debugger without exiting the TPU Editor. In addition, output is produced in a window so that it may be saved as a file for display or printing.

The third VMS enhancement for a Computer Organization course is the availability of system routines. Many of the Computer Organization textbook examples are trivial. Students want to do more than add or sort two lists of numbers. The VMS Run Time Library and System Services routines open the door to many interesting programming assignments. These system routines increase students' understanding of how information is represented in the computer, how to use the stack, and how to pass arguments to a procedure.

Listed below are some of the System Services and Run Time Library routines which may be used by students at this level. A programming assignment might require students to write a MACRO program to set an alarm. The solution would involve spawning a process which would schedule its own wakeup, hibernate, wakeup at the prescribed time, and ring a bell. This could be accomplished by calls to the routines LIB\$SPAWN, \$\$SCHDWK, and \$HIBER.

```

INST -scroll-instruction          REG
>00000219: MULL3  B^04 (AP), B^ |R0:00000000  R10:7FFEDDD4  @SP:00000000
0000021F: ADDL3  R6, B^0C (AP) |R1:00000000  R11:7FFE33DC   +4:20400000
00000225: RET      |R2:00000000  AP :7FF47370   +8:7FF473CC
00000226: HALT    |R3:7FF47394  FP :7FF47358  +12:7FF47384
                |R4:00000000  SP :7FF47358  +16:00000216
                |R5:00000000  PC :00000219  +20:7FF47049
                |R6:7FF47049  @AP:00000004  +24:00000004
                |R7:8001E4DD   +4:00000002  +28:00000002
                |R8:7FFED052   +8:00000003  +32:00000003
                |R9:7FFED25A  +12:00000004  +36:00000004
                |N:0      Z:0      V:0      C:0  +40:00000200

```

Figure 6: Debugger Output

Activity	SS or RTL Routines
hiber/wake a process	\$HIBER, \$WAKE, \$SCHDWK
set a timer	\$SETIMR
spawn processes	\$LIB\$SPAWN
use event flags	\$LIB\$GET_EF, \$LIB\$FREE_EF, \$WAITFR, \$ASCEFC

Later Operating Systems class projects might include using event flags to synchronize the reading and writing of information to an area of memory (\$CREMBX, \$MGBLSC), queuing an I/O request (\$ASSIGN, \$DASSGN, \$QIO), locking resources (\$ENQ), or obtaining information about processes (\$GETJPI).

One set of Run Time Library routines which students particularly enjoy is the Screen Management routines (SMG). With calls to SMG routines windows can be created and inverted, blinking, or large characters can be displayed. Exercises using SMG calls require that students be proficient at passing parameters by value, by reference, and by descriptor. For example, to write a line of double-width text on the screen, the routine SMG\$PUT_CHARS_HIGHWIDE is called and the arguments listed below may be used.

Argument	Type	Access	Mechanism
IDENTITY	longword unsigned	R/O	reference
TEXT	character string	R/O	descriptor
LINES	longword signed	R/O	reference
RENDITION	longword unsigned	R/O	reference

The arguments include an identity name for the display, the text to be displayed, the number of lines to advance after the display, and a rendition mask whose bits indicate whether the text should be blinking, bolded, reversed, or underlined. These parameters have different data types, and they are passed with different mechanisms. The identity, advance lines, and rendition mask are passed by reference. If a CALLS instruction is used, the address of identity, advance lines, and rendition mask are pushed onto the stack.

A descriptor must be created for the text to be displayed. The Run Time Library documentation provides the details. A descriptor is created in memory consisting of the length of the text to be displayed, the type and class of the descriptor, and a pointer to the actual text. The address of the quadword descriptor is pushed onto the stack.

The source code shown below contains the descriptor

DSC_VAX which points to the text to be displayed named VAX_MSG. The actual message to be printed on the screen in large letters is "VAX-11/750."

```

dsc_vax:  .word      len_vax_msg
          .byte      dsc$k_dtype_t
          .byte      dsc$k_class_s
          .address   vax_msg
;
vax_msg:  .ascii     /      VAX-11/750
          /

len_vax_msg = .-vax_msg

```

In order to call the SMG routine to print the large letters, the arguments are pushed onto the stack in reverse order. In the example below, the address of the rendition mask is pushed onto the stack followed by the address of the number of lines to advance, then the address of the quadword descriptor and the address of the display identity. The CALLS instruction contains the number of arguments.

```

pushal   renmask
pushal   one
pushaq   dsc_vax
pushal   display_idl
calls    #4,g^smg$put_chars_highwide

```

A class assignment might require the creation of a login menu using calls to the Screen Management routines. The menu choices might include word processing, editing, or the use of a database. Selection of a menu item would spawn a process for the chosen activity then return the user to the menu.

Students enjoy a programming experience in which they can apply the things they have learned in the course. Such programming requires that students understand how data types are represented in the computer, how to use pointers and stacks for arguments, how to execute procedure calls, and how to use macros to make repetitive code more efficient.

VMS provides a means for students to put the concepts to use and see the practical rationale for them. VMS is effective

in an academic environment because of its openness. Students get a perspective of the operating system which is not easily gained on other systems. They begin to see the operating system as a set of layered modules. At the end of the Computer Organization course, students should be able to program in assembly language and understand its execution within the organization of the computer.



GRAPHICS APPLICATIONS SIG

Readability of VMS Documentation Then and Now

C. Eric Kirkland, Ph.D.

Integrated Microcomputer Systems
Rockville, Maryland

William P. Brenneman

Computer Systems Resource
Charlottesville, Virginia

Abstract

This article introduces the methods and theory of numerical analysis of text known generally as readability analysis. The foundations for readability are presented, along with current competing formulae. To illustrate the discussion, selected VMS documents for Version 3.x will be compared and contrasted with corresponding Version 4.x sections. Though readability analyses *per se* should not be used as a basis for rewriting documents, techniques for improving documentation are summarized.

Introduction

A longstanding, significant criticism of computer systems is that the overall quality of the training manuals and technical documentation is quite low. (Maynard, 1979; Nickerson, 1982) Often the personnel who would most benefit from an automated system may prove the least likely to read any of the documents. What may be the oldest joke in computing is the pithy statement "When all else fails, read the manual." Unfortunately, there has been little effort to comprehensively address this problem either in the education of technical writers or other professionals. (Wright, 1977)

One method for addressing this problem is the analysis of the actual readability of the documentation. This can be accomplished using computer programs for counting unfamiliar words, sentence length, and so forth. From these measures, a prediction of the readability of the text may be calculated. The analytic methods also may be used to predict the efficacy of various revisions of a text.

Readability

Historically, the study of readability has been the province of composition. A unifying principle of composition, as stated by Herbert Spencer, is "so [to] present ideas that they may be apprehended with the least possible mental effort." (1881, p. 11) To this end, the use of familiar words in short, simple sentence structures provides the maximum economy of the reader's attention and, therefore, the best comprehension.

A definition of readability is difficult to prepare. On the one hand, the authors understand that a definition is critical to understanding. But on the other hand, once a definition is offered, one is faced with a seemingly endless series of

exceptions and modifiers. Notwithstanding these reservations, here is a definition the authors have found useful.

Hirsh suggested that readability refers to the "easiness with which a reader understands a text". (1977, p. 9) Given that two texts could convey identical meaning, the text which evoked the meaning with the lesser effort would be judged the more readable.

A convenient method of judging readability is the comprehension test. The literature includes studies which have sought to measure readability in a direct, comprehension-referenced method using standard texts and either multiple-choice or completion tests.

But this approach may be unfeasible or too expensive. For example, the body of documentation to be studied may be too voluminous or may require an expert in a given field as the reader. In such circumstances, readability analysis provides an alternative.

Readability analysis is the application of a computational formula to produce an index that predicts the difficulty or ease with which people will be able to read and comprehend the material. The index is calculated from the surface structure of the text. Features of the surface structure include the words themselves, the punctuation, spelling, sentence length, and so forth. Typically, the prediction is expressed as a grade-equivalent index indicating the level of skill required of the reader to reach a given level of comprehension.

Readability formulae stem from the advent of formal, statistical analysis of text. This may be traced to the explosive growth and diversity of interest in mental measurement following World War I. The success of standardized tests and analytic methods provided a paradigm for the further study of human abilities.

In 1921, Thorndike published a volume entitled *The Teacher's Word Book* which provided 10,000 words that had been laboriously stratified by the frequency of occurrence in texts. This provided the springboard for the numerical analysis of text factors related to human comprehension by Vogel and Washburn.

Vogel and Washburn (1928) used multiple regression analysis to isolate factors that were most highly correlated with comprehension scores. They found the number of different words, the total number of prepositions, the number of words not in the Thorndike list, and the number of simple sentences (Simplicity was judged by the authors.) produced a multiple correlation of 0.845 with comprehension scores. Thus, these four factors account for roughly two-thirds of the variance.

Of course, few ideas are entirely new to mankind. Quite similar factors had been suggested by Herbert Spencer in 1881 as being important in writing. His list of factors included word length in syllables, familiarity of words, abstract level of words and sentence length.

Nonetheless, the technique of counting various text factors, particularly vocabulary familiarity, has been carried forward in a number of formulae as general factors which affect readability. However, before getting too involved in the various formulae, an overview of factors that affect comprehension is needed.

Factors Affecting Comprehension

The syntax of the written composition will have a major influence on the comprehensibility of the text. Convolved sentences with multiple prepositional phrases and embedded clauses require more effort to understand.

Of course, semantics will impact understanding. The more difficult the writer's predicate (intention), the harder it is to express in words. Consequently it is more difficult to understand the full meaning.

The reader's motivation, ability and interest will have a profound effect, too. These are major compounding factors in the comprehension studies that attempt to measure readability. For example, does the testing itself invoke a set of enabling or disabling attitudes and motivations that affect the ability to pass the comprehension test that looms at the end of the session?

Readability Factors

The general method for developing readability formulae has been to take a standard text, give comprehension tests to readers, and then utilize a multiple regression approach to develop a formula that predicts the scores that would be obtained by other readers.

Two common factors are sentence length in words and number of unfamiliar words. The sentence length may be viewed as a measure of syntactic complexity. Procedurally, word familiarity is judged by the percentage of words absent from a list common words. The unfamiliarity of the words may be viewed as a measure of semantic complexity.

Note that because the reading materials have to be developed with a given audience in mind and because the test-takers are representative of only a restricted population, most readability indices have a limited range of grade-equivalent scores that can be obtained. Thus, the text book for a graduate course in physical chemistry might be given a readability index of ninth grade if it were assessed using a formula that could not provide a higher estimate of difficulty.

Two Formulae

Two of the many formulae will be given additional coverage in this section. These are the Dale-Chall and the Flesch. Both are all-purpose measures with grade-equivalent ranges that would include most adult readers' abilities.

Dale-Chall

Edgar Dale and Jeanne Chall introduced a readability formula in 1948 that is still one of the most widely used readability formulae. With this formula, syntax is approximated by sentence length in words; semantics is approximated by computing percentage of words not include in the Dale list of 3,000 common words. A multiple correlation of 0.70 with scores on the McCall-Crabbs "Standard Test Lessons in Reading" was originally obtained.

Flesch

A contemporary of Dale and Chall, Rudolf Flesch published a formula known as the Reading Ease Formula in 1948. It also obtained a multiple correlation of 0.70 with the McCall-Crabbs (above) using the number of syllables per 100 words and the average sentence length in words as the independent variables. This formula has an unusual scale of scores wherein 100 is considered easy "for any literate person" (1948, p. 229) and zero (0) is considered virtually unreadable. Flesch did, however, provide a table for converting the scores to grade equivalences.

Peter Kincaid has revised the Flesch to create a new formula that produces grade level equivalent scores for adult reading materials. This has subsequently been adopted as a Department of Defense (DoD) standard for military specifications: MIL-M-38784A, Amendment 5, 24 July 1978. A more recent edition of the standard, MIL-M-38784B, 16 April 1983, includes procedures for calculating this index in Section 4, Quality Assurance Provisions. This formula, therefore, has been rejuvenated and given new importance.

VMS Documentation

The preparation of any large body of documentation such as the VAX/VMS series requires the concerted effort of a large number of people. These people have a monumental job facing them with a comprehensive product such as that provided by DEC with VMS and all its layered products. The dedicated technical writing and documentation staff which DEC obviously has employed are to be congratulated for the progress

they have made in the usability and completeness of their finished product.

Even a cursory inspection of the VMS 3.x and the VMS 4.x documents reveals some startling differences. For example, with VMS 4 the graphic layout of the pages was dramatically changed with multiple type fonts, graphics, bolding and other graphic devices. These presentation features may affect the usability of the documents. Also, these changes can be implemented with a documentation template system, hence economically.

But what exactly is DEC doing about the readability of their documents?

To analyze the complete set of VMS documentation would be straightforward, but the authors neither had the time nor the resources to invest in this effort. Instead, a sample of DCL commands was obtained by asking co-workers to list their favorites. Admittedly, this is neither scientific nor particularly comprehensive. Therefore, the results presented below should be viewed purely as applying to these command ONLY and not the complete set of VMS documentation.

DCL Commands Analyzed

COPY	RUN
DELETE	SET DEFAULT
DIRECTORY	SHOW DEFAULT
HELP	SHOW DEVICES
PRINT	TYPE
PURGE	

Table 1

The following table is based on the results of the analyses. The scores for the Dale-Chall and DoD analyses are in grade-equivalents. Flesch scores range from 0 (unreadable) to 100 (easily read), as was noted above.

Readability Analysis

DOCUMENT	READABILITY FORMULA		
	DALE	FLESCH	DOD
VMS 3.x	13.99	54.09	10.04
VMS 4.x	14.22	56.02	9.71

Table 2

Clearly, there is not much difference; but no test of significance was warranted given the relatively poor sampling technique.

Before the analysis was conducted, the authors' intuition was that the VMS 4 was much more readable than the VMS

3 documents. This apparently was driven by the presentation factors, not the verbal content. This intuition was simply incorrect. Appearances were deceiving.

Composition

Before any readability analysis can take place, the document has to be written. This may seem obvious, but the predecessor of readable text is a well-prepared composition. No application of readability can replace good composition. Finding skilled writers is a major challenge.

The documentation life cycle commences with the requirements analysis phase and continues through the final acceptance of the system. Each major milestone is marked by a written deliverable to the customer. Indeed, guidelines for programming emphasize that programs are written to read by people; and, therefore, clear, concise and relevant comments are required in the code itself.

Yet system developers often neglect the documentation or delay it as much as possible. Documentation has somehow come to be viewed as a last moment – or optional – accompaniment to the product rather than an ongoing record of accomplishment and usability.

Often the actual deliverable documents are poorly conceived and executed. Alphonse Chapanis, a world-renowned human factors engineer, suggested the following:

Report rejected. Too windy, too hard to read, too long. Final payment on this contract is being held up until a readable report is received. (1965, p. 14)

In the development of any complex system, humans play the critical role; and, according the principle of least effort, people will find the easiest solution to their work. Since the computer system and its considerable documentation will doubtless be part of the overall man-machine system, make it easy on the people who will use the system by adopting standards, being consistent, and using plain language that they understand.

General Guidelines

The resolution to improve documentation must be a corporate commitment in order for the effort to succeed. The skills required for writing effective prose must be nurtured and rewarded, and the corporation must consider the documentation an integral part of quality assurance. Without corporate leadership, any effort will likely fail.

Standards

Given the commitment, the first step toward improving documentation is to adopt a standard for deliverable and developmental documentation. The standard must include both presentation and content guidelines. Ideally, it will include standard tables of contents and outlines for writers to follow.

One standard with which the one of the authors (Kirkland) is intimately familiar is DoD STD 7935 for automated

data processing (ADP) systems. The objective of 7935 is to provide managers and developers of automated systems with a uniform set of documents that address milestones in the software life cycle. These uniform documents serve to guide the customer with the progress of the project, and provide a permanent record of the technical achievement of the project. Later these documents are used to guide maintenance and, possibly, general deployment of the ADP system.

Note well, however, that the use of standards neither limits creativity nor forces formulaic documents. Instead, standards ensure the development of a consistent set of deliverables to the customer that is organized and presented in a consistent manner subsystem by subsystem.

Writing

Documents are written to be read by people. Readers often are not familiar with the programs and the systems. Each document, therefore, must provide essential background context to allow the document to be read and understood.

Ideas for improving the quality of technical writing have been offered by many authors. Seldom have these ideas been backed by experimental evidence to support them.

One refreshing report was offered by Hartley, Trueman and Burnhill (1980), in which they give a list of suggestions for improving technical writing that they had found to have no effect on comprehension. These non-effective rules included using the active voice, using simpler wording, shortening sentences, shortening paragraphs, and providing procedures in numbered lists.

But these suggestions look like they were taken directly from a "How to Write Good Technical Documents" guide. In their final analysis, however, these changes did NOT make the text any easier to comprehend!

Also, note that we can infer that certain of these suggestions would have the effect of reducing the computed readability indices for the given text. For example using simpler (in the sense of more common) words and shorter sentences is guaranteed to reduce the predicted readability.

It is for reasons such as these readability analyses should NEVER be used to guide rewriting. By examining the formulae discussed above, an author could quite easily lower the readability index and yet not improve the comprehensibility.

There are several handbooks, however, that these authors suggest should be standard accessories for writers. The list includes Hirsh (1977), Strunk and White (1972), *The Chicago Manual of Style* (1982), and the *Harbrace College Handbook* (1986). (See references.) Strunk and White's is the authors' favorite.

Sentence Composition

The following suggestions are adapted from the Harbrace College Handbook. A similar list could be made from other texts.

- Write in complete sentences using correct grammar.
- Make relationship between coordinate clauses clear.

- Place idea to be emphasized in independent clause, and subordinate idea in dependent clause.
- Avoid ambiguous references using pronouns.
- Express parallel ideas using parallel grammatical form.
- Eliminate superfluous words.
- Limit the use of the passive voice.
- Vary the length and format of sentences.

Composition of Longer Texts

The need for modular structure is obvious in programming. But often the modular structure of text is ignored. The basic module is the paragraph. The following, again adapted from the Harbrace College Handbook, gives some insights into paragraphs, and longer chunks of connected discourse.

- Limit paragraphs to a single topic.
- Clearly state the topic, generally near the beginning of the paragraph.
- Arrange the details of a paragraph according to some plan.
- Develop the paragraph with supporting details.
- Strengthen the coherence of paragraphs by using connectives and linking expressions.
- Group related ideas together. Arrange the groups using an outline.

Presenting Technical Information

Patricia Wright (1977) has suggested a number of things to improve the presentation of technical information so it can be remembered. Some of these are summarized below.

- Present information verbally rather than using flowcharts or tables.
- Verbal lists using short sentences are effective.
- Provide introductory verbal outlines of the material that will follow.
- Use Headings in Table of Contents and Body of the text. Number them using arabic numerals separated by "points".
- Minimize the use of serial numbers, acronyms and neologisms.

Note that outlines are important both for the writer and for the reader. Writers should use them to guide the writing, so the general outline should be part of the standard. The outline should be presented to the reader, possibly in the form of the Table of Contents.

Consistency

In addition to these general guidelines listed above, the importance of consistency cannot be over stated. Use standard English. Use correct grammar and punctuation. If the application being developed is industry-specific, then use the vernacular of the target population.

Beta Test

No document is ever complete the first time off the printer. Just as the system should be beta tested, so should the documents. Unfortunately, the beta testing of the documentation is a universally neglected activity.

If the users do not spontaneously provide feedback on the documents, then ask for reader feedback using a simple form or checklist. DEC includes a reader's comment form in every manual.

Also, collect the beta test documents and study them. Did the users highlight things? Did they write interpretations or notes? Are some pages dog-eared and stained? These are important bits of information that will help during the rewriting and, hence, will lead to a better product.

Rewriting

Having left a document for one or more days, a thorough re-reading will highlight its weaknesses. Also, the collection of the beta test documents will provide important clues for the sections that need the most work.

One of the principles of writing that has proved quite useful to these authors is the notion of linearity as was suggested by Hirsh (1977).

- Organize ideas into related clusters and present them as a group.
- Make one idea carry forward logically to the next idea.
- Make connections between ideas clear.
- Minimize demands on memory by frequent closure of ideas.

Taking an analogy from programming, though it is standard to use subroutines, avoid them in writing documentation. State what needs to be stated in-line, sequentially.

Summary and Recommendations

It is important that a set of mutually-supportive goals are developed to bridge systems development and technical writing about the systems. The two activities are complementary and interdependent. Lacking documentation on the available software tools, developers would be stuck in the endless task of reinventing the tools.

The extremely limited study reported herein suggests the readability of the VMS documentation has not been addressed. This would be most unfortunate because as system designers

and developers we must be assured our personnel can use the operating system and related tools effectively.

Though this study suggests the documents are comprehensible by high school graduates, the issue of their routine comprehensibility remains. Can system users understand the documentation under non-test conditions? Do they have the necessary cognitive skills to read comfortably at a twelfth grade level in their normal work environment? Will they be able to use the tools provided by DEC after reading the instructions or will they be forced to experiment?

The understandability of the VMS documentation can only be improved by careful rewriting by skilled writers. This would be an arduous and expensive process, but the benefits to the user community would, in these authors opinion, be worth the effort.

DEC has made considerable progress in applying the graphic effects that improve the appearance of the VMS documentation. These changes may improve the usability of the documentation.

All of us can improve the usability of our documentation by providing an index and a table of contents for reference, by using highlighting techniques to help the reader focus on the important factors, and by writing the document for linear cognitive processing. VMS provides a free set of tools that can be used to meet these requirements: text editors (EDT/TPU) and RUNOFF. So among VMS users, no one can claim they do not have the means for providing these improvements to their documents.

Finally, never write a document without asking yourself if it will make sense to the readers. "Write so you can be understood by your elders!" (Hopper, 1987)

Acknowledgements

Special thanks are due Maureen E. Kane for her contributions to this effort. She developed the graphics that accompanied the original presentation at DECUS, and she provided editorial assistance in the final writing of this article.

References

- Chapanis, A. Words, words, words. *Human Factors*, 1965,7, 1-17.
- Dale, E. and Chall, J. A formula for predicting readability. *Educational Research Bulletin* (Ohio State), Jan 21 and Feb 17, 1948, 11-20 and 37-54.

Department of Defense. *Automated Data Systems Documentation Standards*. DoD-STD-7935, Feb 15, 1983.

Flesch, R. A new readability yardstick. *Journal of Applied Psychology*, 1948, 32(3), 221-233.

Hartley, J., Trueman, M. and Burnhill, P. Some observations on producing and measuring readable writing. *Programmed Learning and Educational Technology*, 1980, 17(3), 164-174.

Hirsh, E. D., Jr. *The Philosophy of Composition*. Chicago: University of Chicago Press, 1977.

Hodges, J. C. and Whitten, M. E. with Webb, S. S. *Harbrace College Handbook* (10th ed.). New York: Harcourt Brace Jovanovich, 1986.

Hopper, G. M., Admiral (USN, Ret). Personal communication. March 31, 1987.

Maynard, J. A user-driven approach to better user manuals. *IEEE Computer*, 1979, 12, 7275.

McCall, W. and Crabbs, L. Standard Test Lessons in Reading. *Teachers College Record*, 1925, 27(3).

Nickerson, R. S. Why interactive systems are sometimes not used by the people who might benefit from them. *Human Factors*, 1982, 24, 509-519.

Spencer, H. *Philosophy of Style*. New York: D. Appleton and Co, 1881.

Strunk, W., Jr. and White, E. B. *The Elements of Style* (2nd ed.). New York: Macmillan, 1972.

The Chicago Manual of Style (13th ed.). Chicago: University of Chicago Press, 1982.

Thorndike, E. *The Teacher's Word Book*. New York: Bureau of Publications, Teachers College, Columbia University, 1921.

Vogel, M. and Washburn, C. An objective method of determining grade placement of children's reading materials. *Elementary School Journal*, 1928, 28, 373-381.

Wright, P. Presenting technical information: a survey of research findings. *Instructional Science*, 1977, 6, 93-134.

Overview of Human Factors and Software Engineering

C. Eric Kirkland, Ph.D.

Integrated Microcomputer Systems
Rockville, Maryland

Abstract

This article provides a general overview of Human Factors Engineering and its relevance to Software Engineering. It is assumed the reader has only a very limited knowledge of human factors engineering and a considerable knowledge of software engineering. Due to its multidisciplinary nature, only a brief coverage of human factors is possible; however, areas of emphasis were selected which are most germane to software engineering. These include the parallels between structured design principles and those of human factors, and practical guidelines for improving the design of systems.

Introduction

Human Factors Engineering (HFE) is an applied science that concerns itself with the design of things that people use. In particular it studies people and their relationship with machines and environments. Outside the United States, HFE is commonly known as ergonomics. Though distinctions between the two have been offered, today the distinction is so substantially blurred that for the purposes of a general introduction they may be considered identical.

The focus of HFE is the design and creation of objects, facilities, products, equipment and environments that are usable by people. Included are the procedures for carrying out work and other activities.

The objectives are to enhance effectiveness, to improve efficiency, to maintain or enhance human values, and to satisfy human needs. Examples that come to mind immediately are health, safety, and job satisfaction.

The approach is to synthesize knowledge about human abilities, behaviors, characteristics, and motivations with knowledge of job performance requirements. The goal is to design and build an harmonious working system that includes the person, the machine and the work environment.

To apply HFE approaches and techniques to software engineering is quite straightforward. The first and most important step is to always strive to include the person as an integral component of the system. With this perspective, the human needs are the principal motive for the actual development process.

Historical Perspective

During World War II, the discipline of HFE emerged as a distinct discipline of importance. This emergence was in response to the need for training young men to control complex weapons systems. (McCormick and Sanders, 1982) The HFE

focus on optimizing the combination of humans and machines was a natural accompaniment to the increased complexity of the tasks at hand. For example, these weapons systems often exceeded the complexity of any machine the recruits had ever seen. These men had to be trained quickly and effectively to meet the demands of the war. Once trained, their skill in the use of the systems would profoundly affect their chances of survival. So the HFE emphasis on modifying the machine to make it more understandable and usable by people was an obvious choice: There simply wasn't enough time to modify the men.

Of course, as long as people have used tools, there has been an interest in improving the safety and efficiency of the tools. Prior to modern medical advances, even a blood blister from too rough a handle was life threatening. Since the human musculature provided the power for the tool, improvements in efficiency lessened the immediate burden of work.

Between 1750 and 1890, machines became dominant in industry. Jacquard revolutionized the textile industry with the punch-card controlled loom, thereby providing Hollerith with a paradigm that marked the path that would later lead to computers. With machines human muscles no longer supplied the power; the human brain, however, continued to provide control. Interestingly, the then-current technological forefront came to be considered a model for the way the human brain functions: The brain was termed a thinking machine.

Between the last quarter of the nineteenth century and the onset of World War II, the telecommunications, automobile, and aviation industries flourished. Everyone needed a motor vehicle and a telephone. The impact of the telephone was such that the brain came to be compared to the switchboard.

Much of this period leaves man in control of the machine, however human strength and endurance are not issues because the machine provides the power and a portion of the control. The human role is reduced to a more supervisory or monitoring function.

The advent of the computer further changed the role of workers by assuming more of the control functions. In many highly complex activities, such as flying commercial jet aircraft, the human being has become little more than an interested observer. Thus, vigilance becomes an important problem: It simply is not easy to maintain the level of interest required. Also, the quality of work life may be negatively affected by the automation of the system.

Computers aid, relieve and extend human capabilities in ways that otherwise could not be imagined. Unlike any machine that has come before it, a computer helps a person analyze data and make decisions. The computer is an assistant for thinking; but, as would be predicted, the computer has become the model for human brain function.

In addressing these many areas of human-machine systems, several academic disciplines and professions contribute techniques, tools and understandings. The sciences include psychology, sociology, anthropology, physiology, biology, mathematics and statistics. The professions include industrial engineering, architecture and education. With such a mixture at its foundation, HFE understandably brings many diverse views to the problem of building human-machine systems.

Task Analysis

One of the principal tools of HFE is task analysis. Task analysis produces a carefully specified description of a task and its constituent processes. The tools required and the flow of information from one portion of the task to another are included, as are the environmental and economic constraints.

The technique of hierarchical decomposition is critical to the success of task analysis. The decomposition yields a system specification that includes a list of sub-functions and procedures that can be used in the detailed design of the overall system. Task analysis may be used to provide a list of skills or training requirements that will be needed by the system users. In addition, the task requirements may be shaped into a job design description that guides personnel selection.

Herbert Simon has argued that all complex systems involving humans are artificial in the sense that they are "man-made, as opposed to natural". (1969, p. 4) Furthermore, all complex systems either are hierarchies or can be depicted as such.

Hierarchies allow the designer the tremendous freedom to focus on the goals and functions of a system without becoming embedded in the details of any particular sub-function. Once the overall functional description of the hierarchy is completed, the designer may exit the process altogether, leaving the details to other crafts people. In order for this technique to succeed, however, the designer must carefully and completely describe all the necessary interfaces between sub-functions and processes.

Software Engineering

The hierarchical decomposition of tasks described above applies equally well to software engineering (SE). Indeed, the

reader could easily have thought the above was taken from a text on structured analysis and design. The common thread of both efforts is the specification of complex systems in terms of discrete process descriptions and interfaces between functional components.

One of the techniques of structured analysis and systems design is the data flow diagram. [For details on data flow diagramming techniques, see De Marco (1979) or Teague and Pidgeon (1985).] Figure 1 presents a data flow for the analysis and design phases of the information life cycle.

Upon receipt of the user request, a survey is initiated to identify the user needs. A feasibility report is generated to describe the system that is being requested and to present recommendations to the organization. This report and the organizational goals are then incorporated in a requirements analysis.

The requirements analysis is the crucial step in the life cycle. It produces three key components: (1) Functional Description, (2) Physical Requirements, and (3) Budget and System Development Plan.

The Functional Description is a complete specification of required functions and system interfaces. The entire environment, including both manual and automated systems already in use, must be clearly described. The physical requirements for the system must be discussed because these will affect the hardware that will subsequently be selected. Of course, the budget and schedule will have to be approved by management before any work takes place.

Interviews and checklists are useful in the collection of user requirements. Simple observation of the people as they work provides valuable insights, too. Who works with whom and for whom? Who really does the work? Who is responsible for doing the work? Seldom will an analyst find an organization where the official chain-of-command is the only one to consider.

The preliminary design of the product produces a system specification that guides the preparation of the detailed system design. During this phase the analysts prepare an overall system configuration plan, select an implementation language, identify major modules and their interfaces, establish controls and data structures, and provide a test plan. A system hardware configuration and performance requirement document is forwarded to the hardware study.

The hardware study produces an order for equipment, so it must incorporate user requirements, performance, and configuration information from the preliminary design. Budgetary limits and product availability impose additional constraints on the selection of the vendor. The final system configuration is forwarded to the detailed design process.

The detailed design phase produces a structurally decomposed system at the level of individual modules. These specifications must include data structures, algorithms, internal program controls, interface requirements, interprocess communications techniques, and other factors that influence the overall program design. Also addressed are factors affecting projected sizes, required timing, storage, and other hardware environment constraints. Standards for test data and a plan for testing are very important as system constraints. Not only must the formal technical specifications "describe everything

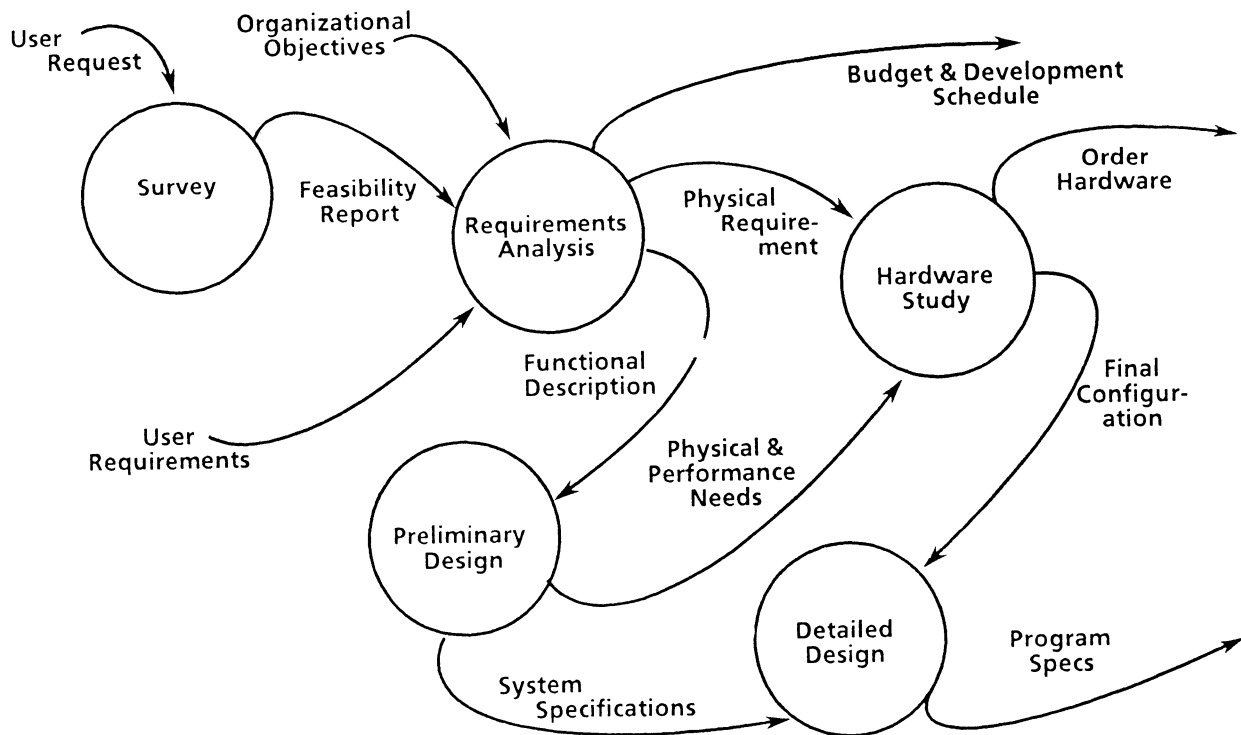


Figure 1: Information Life Cycle

the user does see, including all interfaces; it must refrain from describing what the user does not see.” (Brooks, 1975, p. 62)

Note that the human factors related to the system have been given very little attention. Few texts on systems analysis and design offer any guidance on including human beings in the design. Certainly, humans are interviewed and feedback on the design is solicited. Once the user requirements phase is completed, however, little interest in the ultimate users remains. The human engineering issues aside from performance simply are not given the importance they deserve.

Synthesis

Human-machine systems are the focus both of HFE and software engineering. Both disciplines rely on the hierarchical decomposition of complex systems. Both disciplines seek to produce an efficient, safe, usable system. There should be tremendous harmony between these disciplines. It is entirely possible, however, that many software engineers have given only cursory attention to the human component of their system designs.

Often the software engineer seeks to develop a completely automated system. There is no analysis of what functions would be left for the person who uses the system. Such systems often leave only trivial tasks for the human being. For example, the system users’ primary jobs may be just monitoring the system. Even worse, the humans who use the system

may have been given nonsense “make work” of no consequence at all.

Of course, humans perceive their inconsequential role in such systems quite quickly. Their reactions may range from an initial frustration and anger with the system to a final state of resignation and apathy. This can be catastrophic in situations where the system designers have left failure override in human hands.

Though the designer may have envisioned the workers suddenly springing into action to “save the day”, the overall system failure may be unavoidable because the workers really are not able to bring their full abilities to the problem in a short period of time. The lack of meaningful work may have caused such general mental and physical fatigue that the people simply are not equipped to handle the emergency.

It is this author’s personal philosophy that this state of affairs should never exist. People should be given meaningful work that contributes to their personal satisfaction with their work and their self-esteem. This should be done even if it means taking work away from the automated system. People need jobs to help them attain their own individual potential.

Integrated Design

In approaching the integration of Human Factors Engineering with Software Engineering it is important that a set of explicit goals be developed. The designer must systematically address

not only the user requirements but also human factors. The following list can be used to assure basic human factors have been included in the analysis. (Meister, 1971)

1. What are the system outputs and inputs?
2. What operations produce the outputs from the inputs?
3. What system functions can be assigned to the human?
4. What level of training and skill is required of the users?
5. Are the tasks compatible with human capabilities?
6. What interface will be used between the human and the system?
7. Are the machine and the human smoothly integrated?

The collection of outputs and inputs is part of the requirements analysis phase. The actual target users are the source of this information.

Each of the operations that are required to produce the output can be established by preparing a data flow diagram or work breakdown structure. Task analysis techniques are useful, too.

Once the system data flow is established, the partitioning of those functions to be assigned to the human can be completed. De Marco has suggested that the diagram of the system be partitioned to highlight the domain of change. (De Marco, 1979)

Next, consider the training, skills and human capabilities. Can a person actually function in the ways required? Would the human have to be modified to use the system? Of course, re-designing the machine would be far easier and more economic than re-designing the human being.

The issues of the integration of the man and the machine are important. How can the system be designed such that the human and machine subsystems complement each other, anticipate each others requirements, and share the work? With automated systems it is possible the machine will work substantially faster than the human being. Will the human being be pressured by the machine's prompt for more input? Perhaps the software could be modified to occasionally provide a break in the monotony.

Task Allocation

The partition of the overall system functions to allocate certain tasks to the automated system and other tasks to humans must be undertaken early in the design phase. In making this partition, it is important that the human be given paramount importance. Kantowitz and Sorkin use the phrase "Honor Thy User" (1983, p. 13). This means that the most important thing the system designer can do is to provide the human a meaningful role that is designed to maximize the value of the human in the system.

One approach to the partitioning task is to first assign tasks to the machine for which it has a "natural" advantage. Then examine the remaining tasks to insure those left for the

human are useful and reasonable. If not, then give certain functions back to the human. This approach will counteract the overwhelming tendency to automate every function of a system.

Models

Another approach to the problem of including people in the design is model building. Indeed model building is one of the tools in common to the disciplines of HFE and SE. Models provide an abstraction of the overall system and its components such that the users can understand the system and such that the developers can build the system.

Rubinstein and Hersh (1984) suggest that a "Use Model" should be developed that describes how people will use the system. This model should specify the problems the system will solve and how it will integrate with the workers environment. It should depict the relationships of the people using the same system and explain how their efforts interrelate.

Starting from the Functional Description, the developer must build a system model that is representative of the domain of work. The model must be understandable by the users, who must be able validate both the manual and automated functions that are specified. The model should reflect the sociological and political influences that shape the work environment. In other words, the model must be ecologically valid.

If the users cannot understand the model when it is presented by the designer, then quite simply the model is useless: Go build a better model. Simplify the design, and limit the scope of proposed system. It also may be helpful to decrease the conceptual load on the users by making the model more relevant to their work and their knowledge. Though the author has found data flow diagrams to be approachable by most users, this is not always the case.

Rubinstein and Hersh also suggest that a system "myth" (metaphor) be developed, much like the "desk top myth" has been developed for office workers. But be sure to pick a myth that is representative of the work being done. It is intuitive that a "kitchen myth" would not have succeeded in the office environment. But what of the desk top myth on a shop floor? Is it ecological valid?

Once this model is understood and accepted, it will be the major input to the design phases that follow, and it can be used for the development of a prototype.

User Characteristics

As part of the overall requirements analysis, the detailed characteristics of the user population must be collected. The following items are a minimum set of questions that must be answered.

1. Who are the users?
2. What level of knowledge to they have?
3. What capabilities are required?
4. What skills must they have?

5. What attitudes must they have?
6. What types of training will be needed?

As may be obvious, these questions quickly become intertwined. If the user population includes business managers and executives, then the answers that would be obtained are quite different from those that would be obtained if the user population was data entry clerks and typists.

In 1973 James Martin listed management and executive user characteristics with an insight that remains interesting today. (p. 438) He stated that these users are highly intelligent, will not remember mnemonic commands, are too busy to attend a training course, and are highly impatient. They, also, will reject a system if it is confusing or does not provide worthwhile results. This suggests a system for these users had better be well-designed and easily usable.

There are other characteristics of the users that should be considered. Bloom's Taxonomy (Bloom, 1956) provides a useful hierarchy of human cognitive powers.

1. Knowledge
2. Comprehension
3. Application
4. Analysis
5. Synthesis
6. Evaluation

As the hierarchy is traversed from knowledge to evaluation, the concomitant load on the users' abilities is increased. This increase in cognitive load must be carefully considered.

Knowledge is the possession of facts, possibly only by rote memorization. Comprehension is the capacity to actually use the knowledge (facts) stored away in memory. It implies an ability to thread together the facts such that the pieces fit into a whole that is usable.

Application is a demonstration that the person has the comprehension of the matter at hand. In contrast, comprehension only implies the person could apply the knowledge; not that he or she will.

Analysis implies the breakdown of the information at hand into its constituent parts and the detection of their interrelationships.

Synthesis is defined as taking the pieces and building a coherent whole. Generally, this implies taking the pieces of previous experience and previous analyses and recombining them to form something new.

Evaluation rests on the foundation of human judgment. Both criteria of performance and standards are applied. Solutions are questioned and competing strategies are weighed one against the other. It involves judging the accuracy, completeness, economy, and efficiency of solutions.

Most people function most comfortably at only so high a level in this hierarchy. Having worked closely with the people in developing the functional description, the analyst should be able to pick the conceptual level that fits the user population.

Usability Specifications

John Whiteside of DEC has offered the appealing suggestion that a key component of a system specification should be a usability specification. The usability specification must provide a set of clear and measurable factors related to system performance and usability under actual conditions.

Setting usability as a goal for designers does not make it happen. Designers must be given the time and the corporate commitment to usability that will afford usability the same priority as other engineering factors.

Whiteside notes that "usability of a system is more a function of quality than style. Indeed ... the best indicator of usability relates to the strength of an interface's design and amount of effort put into refining and debugging it." (1986, p. 28)

Model building and prototyping (above) complement usability analyses by establishing a low-cost platform for users to experiment with the system and for the usability measures to be collected.

Language and Documentation

One of the most common mistakes in building systems (and therefore one of the most common complaints) concerns the language used both by the system and in its documentation. The author provides a thorough description of this problem area elsewhere in these *Proceedings*, so a repetition here is unwarranted. A summary of factors that will improve the documentation and the language include the following:

1. Provide an End User's Manual as a first deliverable.
2. Provide overviews, an index and a table of contents.
3. Use graphic devices to highlight important points.
4. Beta test the documentation.
5. Rewrite.

The understandability of a document is improved by providing overviews: "Most documents fail in giving too little overview." (Brooks, 1975, p. 165) Also, the usability of the documentation is improved by providing an index and a table of contents and by using highlighting techniques help focus the reader's attention to important points.

Beta test the documentation along with the system. If the users do not spontaneously provide feedback on the documents, then ask them for suggestions to improve the comprehensibility and usability of the documents.

Rewrite the documentation following beta test. Unless this is included in the original plan, the time it takes will likely preclude its happening. But unless rewriting is undertaken, then poor manuals will continue to plague computer system users.

Of the standard three structured programming notions (sequence, selection and iteration), only sequence is applicable to writing. State what needs to be stated in-line, sequentially.

Make each idea carry forward logically to the next. Explicitly link ideas. Minimize demands on the reader's memory by frequent closure of ideas: Do not leave ideas unresolved and unclear while expounding a tangential or interjectory idea.

Summary

Human factors engineering is an applied science that studies people and their interaction with machines. Since the most pervasive human-machine interaction today is likely the human-computer interaction, human factors offers a number of approaches and tools that complement software engineering. This is particularly true for the initial design, and the final delivery and acceptance phases.

This article has listed a number of additional, human factors that should be included in the design of automated systems. These include an analysis of the user characteristics, a task allocation plan, a system model, and usability specifications.

An integrated information life cycle, also, addresses the comprehensibility of the system and its documentation. Improvements in the individual documents, both language and presentation factors, can help. The consistency and predictability of the dialog can be improved, too.

The traditional life cycle management approach ignores the user population during much of the course of system development. Often the functional description is the only document delivered to the end user population before the system is developed. The integrated approach advocated in this paper shifts the focus from the system per se to the users of it; and it adds an End User's Manual, which should be a key deliverable following the approval of the preliminary design.

Figure 2 is a data flow diagram for an integrated information life cycle that includes these additional factors.

One final hierarchy of interest is the hierarchy of human needs specified by Abraham Maslow. The lowest level of Maslow's hierarchy is physiologic survival. Directly above it are safety and then love. These are the most basic three human needs. As system designers we have a responsibility to design systems that do not endanger the people who use them, but other than that there is little we can do with respect to these three needs.

The next higher three needs are self-esteem, information and understanding. It is here that our system designs can have the most impact. Our systems must provide people tools that enhance their self-esteem by providing them the expression of meaningful work, work that serves a purpose and does not denigrate their ability or their stature as fellow human beings.

As a final, simple technique, place yourself in the end user's situation. Ask yourself if the system is reasonable and has internal consistency and integrity. Ask yourself if the work will be meaningful and satisfying. If your own human values and needs are supported by the system, then, to at least that extent, your efforts at improving the human factors will have been successful.

Acknowledgements

Special thanks are due Maureen E. Kane for her contributions to this effort. She developed the graphics that accompanied the original presentation at DECUS, and she provided editorial assistance in the final writing of this article.

References

- Bloom, B. (Ed.) *Taxonomy of Educational Objectives, Handbook I: Cognitive Domain*. New York: David McKay, 1956.
- Brooks, F. *The Mythical Man Month*. Reading, MA: Addison-Wesley, 1975.
- De Marco, T. *Structured Analysis and System Specification*. Englewood Cliffs, NJ: Prentice-Hall, 1978.
- Kantowitz, B. and Sorkin, R. *Human Factors: Understanding People-System Relationships*. New York: John Wiley and Sons, 1983.
- Martin, J. *Design of Man-Computer Dialogues*. New York: Prentice-Hall, 1973.
- McCormick, E. and Sanders, M. *Human Factors in Engineering and Design* (5th ed.). New York: McGraw-Hill, 1982.
- Meister, D. *Human Factors: Theory and Practice*. New York: Wiley, 1971.
- Rubinstein, R. and Hersh, H. *The Human Factor: Designing Computer Systems for People*. Bedford, MA: Digital Press, 1984.
- Simon, H. *The Sciences of the Artificial*. Cambridge, MA: M.I.T. Press, 1969.
- Teague, L., Jr. and Pidgeon, C. *Structured Analysis Methods for Computer Information Systems*. Chicago: Science Research Associates, 1985.
- Whiteside, J. Usability engineering. *Unix Review*, June 1986, 4(6), 22-37.

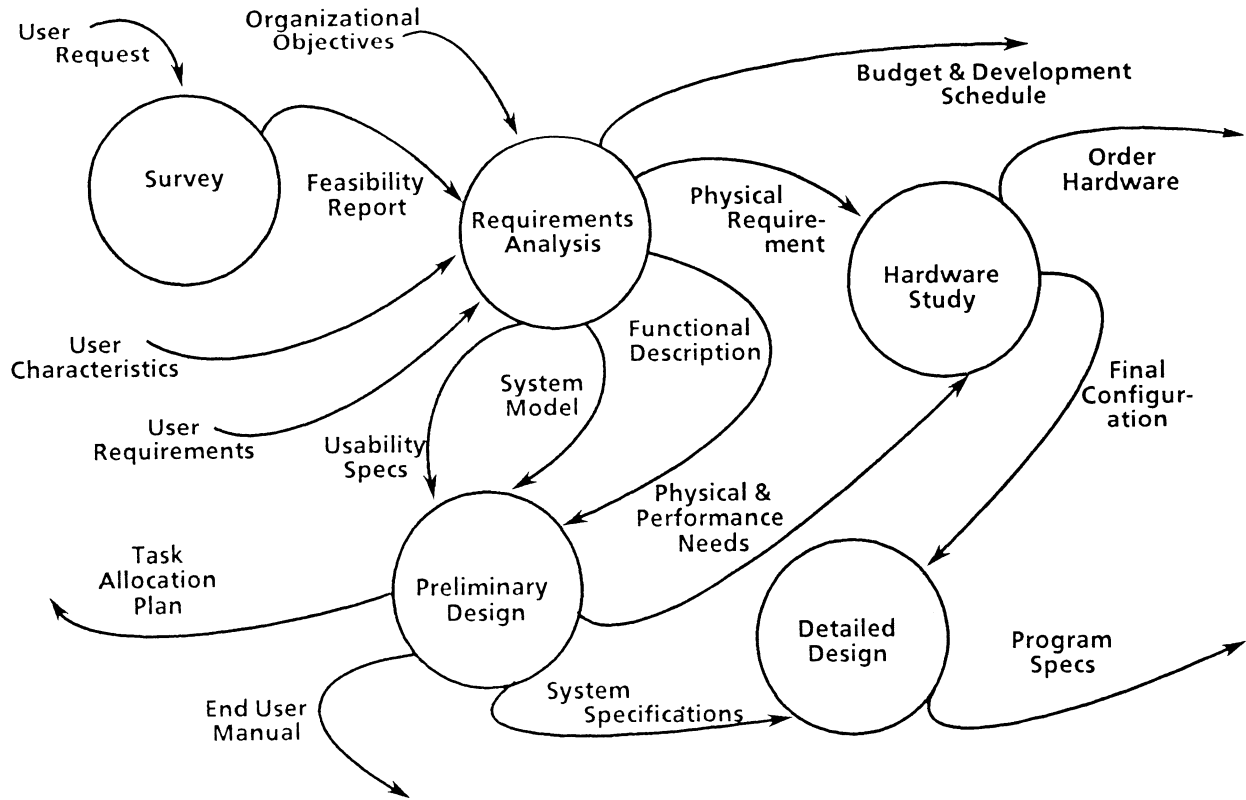


Figure 2: Integrated Information Life Cycle

POSTSCRIPT APPLICATIONS USING A MNC/DECLAB-23 COMPUTER.

O. Guetta, D. Fortney, A. Dubois
Uniformed Services University of the Health Sciences
Medicine Department
Bethesda, MD 20814 USA

Abstract:

POSTSCRIPT is a high-level interpretive programming language designed to describe the appearance of text and graphics on printed pages and to be implemented on printers. However, to transfer POSTSCRIPT programs and data from a mass storage device to the printer, a link between the printer and a computer is necessary. Therefore, we connected the printer to the printer port of a MNC/DECLAB-23 computer, which is using an RT-11 operating system and supporting FORTRAN IV. POSTSCRIPT codes and their corresponding data were sent to the printer using I/O routines written in FORTRAN. The emulator of the printer interpreted the printing instructions and processed the data to produce the desired formatted output pages. We developed a batch word processing program written in POSTSCRIPT to format and print text files (abstracts, articles, documentation, ...). The text files were created and edited using a terminal. They contained, in addition to the text, several sequences of characters describing their output format. Other applications, such as listing programs and tabulating, plotting and curve fitting experimental data, were also written in POSTSCRIPT. This method allows freedom and flexibility to develop POSTSCRIPT programs, and can be used to connect any printer with a POSTSCRIPT emulator to the serial output port of any computer.

INTRODUCTION

The Postscript language¹ is a high-level interpretive programming language designed to describe the appearance of text and graphics on printed pages. The Postscript interpreter is implemented on the printer. The present paper describes a method used to operate the printer and develop Postscript programs, and demonstrates some of the applications that we have developed.

OPERATING MODES OF THE PRINTER

The two operative modes of the printer are the interactive mode and the batch mode.

1. Interactive mode

The interactive mode is the mode by which a user may interact with the printer from a terminal connected directly to it. A job consists of a dialogue in which the user issues a Postscript statement and the Postscript

interpreter executes it and prompts for the next statement. While the user is sending a Postscript statement, the interpreter echoes characters and provides error messages; therefore, this mode is useful for experimenting with Postscript. Another advantage of this mode is that, thanks to the Postscript language, the printer can be used as a general purpose personal computer. However, this mode is not convenient to transfer large Postscript files or data files.

2. Batch mode

The batch mode is the mode by which a user may operate the printer as a printing device for another computer. The printer is in this case connected to an open port of the computer. A job consists of the execution of a Postscript program sent by the computer to the printer. A Postscript program generally consists of two parts:

a - the prologue, containing specific definitions and procedures, but not executed immediately;

b - the script, that consists of references both to Postscript operators and to definitions made in the prologue, and interspersed with operands and data

required by the procedures.

The batch mode allows development of programs with large data input, and is therefore the mode we used in our applications.

MATERIAL

To develop Postscript programs, we used:

1 - a DIGITAL MNC/DECLAB-23 computer, with an RT-11 interactive, single user, real-time operating system and FORTRAN IV programming language;

2 - a DIGITAL PDP 11/44 computer, with an RSX-11M interactive, multiuser, multitasking, real-time operating system and FORTRAN 77 programming language;

3 - an APPLE LaserWriter Plus printer with Postscript emulator that was connected to the printer port of either the computer or the terminal.

The advantage of connecting the printer to the printer port of the terminal is that whatever is sent to the printer is visible on the screen, and messages coming back from the printer can be read. The drawback of this type of connection is that the printer port of the terminal has to be opened, and therefore the keyboard cannot be used during the time of the transfer. In a single job environment, if the user inadvertently hits one key on the keyboard, it will echo on the screen and be sent to the printer, then crashing the program. In a multitasking environment, for the same reason, the user would not be able to run another task at the same time.

The applications presented in this paper have been developed on the MNC computer with the LaserWriter connected to the printer port. However, the method that we use to develop and transfer Postscript files to the printer is independent of the type of computer or printer.

METHOD

The data to be printed are first stored in a data file. The Postscript program that will be used to print the data with the desired format is then developed, debugged and stored in a Postscript file. At the time of printing, the Postscript prologue followed by the script and its corresponding data are sent to the printer from a program using I/O routines written in Fortran. The interpreter interprets and executes the printing instructions contained in the Postscript program and processes the data to get the desired output page. This method is useful to print large data files, and also allows the insertion of new Postscript statements (for instance to save and restore given states, to reset variables, ...) between different data sets.

POSTSCRIPT APPLICATIONS

1. Printing a file

The first application is a program that prints a listing of a file.

Figure 1 shows the Postscript prologue, which contains three main procedures:

1 - *rd* is a procedure that reads a line of characters terminated by a newline character from the current file, and stores these characters in the string called *char*;

2 - *pr* is a procedure that first decreases *y1* (vertical printing position), then sets the current point to coordinates (*x1,y1*), and finally prints the string *char*;

3 - *pr1* is a procedure that first sets the current point to the left top of the page and prints the title of the file stored in the variable *title*, then sets the current point to the right top of the page and prints the value of the page counter *ip*.

This Postscript prologue is stored in the file named OGLT9.POS.

Figures 2a and 2b show a listing of the Fortran program that transfers OGLT9.POS and the data file to the printer.

- Unit 5 corresponds to the terminal port, unit 7 corresponds to the printer port.

- After a few lines of declarations, the program asks the user the name of the data file to print, and stores it into *NAME1*.

- File OGLT9.POS is opened; each line of this file is read and immediately sent to the printer, until the end of file is reached and the file is closed.

- The name of the file to print, stored in *NAME1*, is sent to the printer and stored in the string *title*. Statement 215 demonstrates one way of passing a Fortran variable to a Postscript variable.

- The file to print is then opened, and the line counter *NBL* is set to zero.

- Each line of the data file is then read with an 132A1 format corresponding to the line width of the terminal and stored in the string *LINE*. Statement 225 sends the Postscript procedure *pgr* to the printer, which is executed immediately and expects a line of characters in the current file. The Fortran variable *LINE* is therefore sent to the printer immediately following *pgr* procedure. We demonstrated here a second way of passing a Fortran variable to a Postscript variable.

- *NBL* is then incremented:

* If it is less than or equal to 45 (maximum number of lines per page taking into consideration the vertical spacing and the font height chosen), another line is read and the same loop is again executed.

* If *NBL* is greater than 45, statement 245 sends procedure *pr1* to the printer, which prints the title and the page number, and the *showpage* command which allows the current page to be printed. Statement 255 increments the page counter *ip* and resets the vertical position *y1*. Another line is then read until the end of file

is reached and the file is closed.

At the same time, the listing of this program illustrates an output page.

2. Plotting a signal

The second application is a program that plots a signal over time. The peaks are determined using a peak routine and marked with a + sign on each graph, as illustrated in figure 3. The signal, which is the output of an A/D converter, consists of a sequential file in which all the points are evenly spaced over time. The time of each peak found by the peak routine is stored in another sequential file. In order to avoid printing too many pages but to have good resolution, the aim of the program is to plot 3 graphs per page, each graph representing 8 minutes of the signal. The time between two consecutive points is known from the rate of digitization. The minimum and maximum values are calculated for each signal.

The Postscript program contains three main procedures:

a - the first procedure initializes each graph by defining the position of the origin, drawing a box around the graph, reading from the Fortran program the title of the study and the minimum and maximum values, placing ticks on the axes, labeling the axes, and initializing the current point;

b - the second procedure reads from the Fortran program the time and amplitude of a point and traces a line to this point. This procedure is invoked for each data point;

c - the third procedure reads from the Fortran program the time of each peak and prints a + at this location.

The Fortran program divides the time signal into 8 minutes segments. For each segment, the program sends to the printer the command to initialize the graph, the command to plot a point immediately followed by the time and amplitude of the point, and the command to plot the peak followed by the time of the peak. The Fortran program also sends to the printer commands to reinitialize Postscript variables, move the origin of each graph, and restore given states.

3. Tabulating data

Another important application for biomedical research is a data tabulation program.

a - Figure 4a shows an example of a formatted data file. The first line represents the title of the study, composed of 80 alphanumeric characters. The second line is the number of measurements in the file. The following lines represent the different measurements, each one composed of 9 numbers written with a specific

format.

b - Figure 4b shows one type of output. The title of the study is printed in bold characters and therefore emphasized; on top of each column is written what each number represents in the column; in each column, the numbers are right-justified.

The Postscript program used to generate this output contains two main procedures:

* one procedure to print the box, the title of each column, and the title of the study transferred from the Fortran program;

* another procedure to read the 9 numbers to print from the Fortran program, to move to the right column, and to print each number.

The Fortran program first sends to the printer the commands to initialize the page, then reads each line of the data file and sends the commands to execute the second procedure immediately followed by the 9 numbers, until the end of the file is reached.

Figure 4c shows a second type of output, which uses the same data file for input. Each parameter is plotted over time, and marked with a + on each graph. The points are connected with a line that allows visual determination of major variations of a parameter.

The Postscript program contains two main procedures:

* the first procedure initializes each graph by positioning the origin, drawing the box, writing the title of the study, placing the ticks on the axes, labeling the axes, and initializing the current point;

* the second procedure reads from the Fortran program the time and amplitude of a point, prints a + at this location, and traces a line from the previous point to this point.

For each graph, the Fortran program first sends to the printer the command to initialize the graph, then for each point the command to plot it, immediately followed by its time and amplitude. The Fortran program also sends to the printer commands to reinitialize Postscript variables, to move the origin of each graph, and to restore given states.

Figure 4c shows that some of the peaks seem to occur at the same time on different tracings, suggesting a correlation between some of these parameters. Figure 4d examines this possible correlation and shows, using the same input file again, a third type of output. Each of these four graphs represents a plotting of the two variables K solid and K liquid as a function of one parameter (mean frequency for graph 1, motility index for graph 2, relative amplitude for graph 3, and area for graph 4). Each point is marked with a + (for solids) or with an x (for liquids). For each variable, correlation coefficients are calculated using a linear regression program and printed under each graph. The two regression lines (solid line for solids and dotted line for liquids) are traced on each graph.

Both Postscript and Fortran programs are similar to the programs of the previous output.

4. Word-processing

The last application described in this paper is a program that we call batch word-processing program. The aim of the program is to allow the user to print any document (abstract, article, memo, ...) with a specific format by using only the editor to create the text file.

Figure 5a shows a listing of a text file. The file contains, in addition to the text, several sequences of characters describing the output format.

The Fortran part of this word-processing programs reads the text file and sends to the printer each word separately. A word is considered to be text delimited by a space on each side.

The Postscript part of this program contains procedures that analyze each word received from the Fortran and procedures that interpret the instructions hidden in the special characters describing the desired format.

Figure 5b shows the output produced by the word-processing program using the text file shown in figure 5a, and illustrates some of the available features.

CONCLUSION

This method allows freedom and flexibility to develop Postscript programs that can be used to interface any computer with any printer supporting Postscript.

REFERENCE

PostScript Language Reference Manual, Adobe Systems Inc. Addison-Wesley Publishing Company, Inc., Menlo Park, CA, 1985.

```
%
% Initialization
%
/str 200 string def
/inch { 72 mul } def
/ip 1 def
/x1 .8 inch def /y1 10 inch def
/Courier-Bold findfont 12 scalefont setfont
%
%
% Procedure rd reads line of characters from current
% file and store characters in string char.
%
/rd { currentfile str readline pop
      /char exch def
    } def
%
%
% Procedure pr prints string char.
%
/pr { /y1 y1 height 8 mul 7 div sub def
      x1 y1 moveto
      char show
    } def
%
%
% Procedure pgr combination of rd and pr.
%
/pgr { rd pr } def
%
%
% Procedure pr1 prints file name and page number.
%
/pr1 { .8 inch 10.3 inch moveto title show
       6.5 inch 10.3 inch moveto
       (page ) show ip cvlit str cvs show
    } def
```

Figure 1 - Printing a file: Postscript prologue

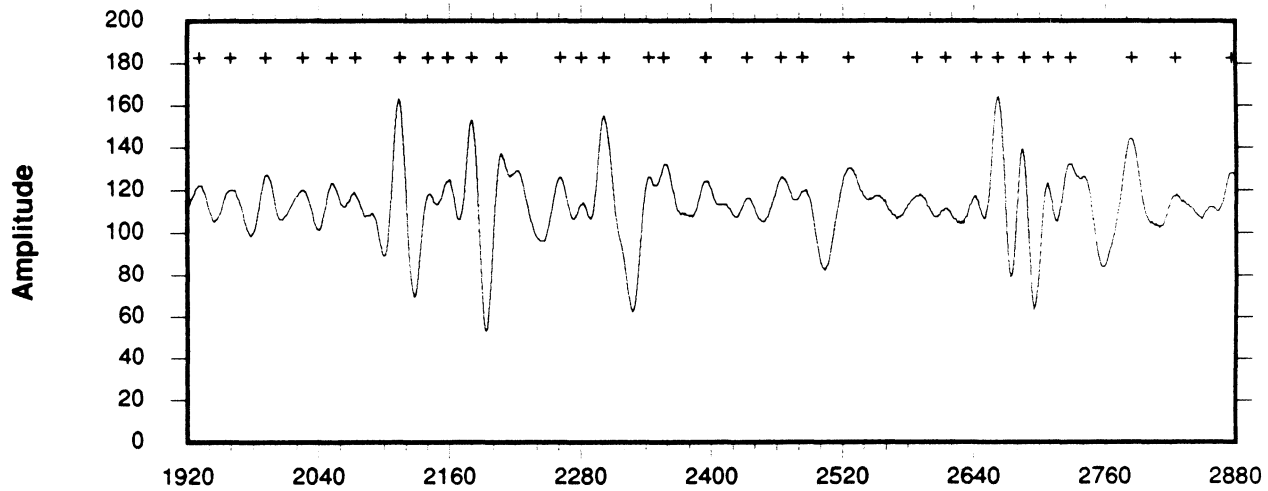
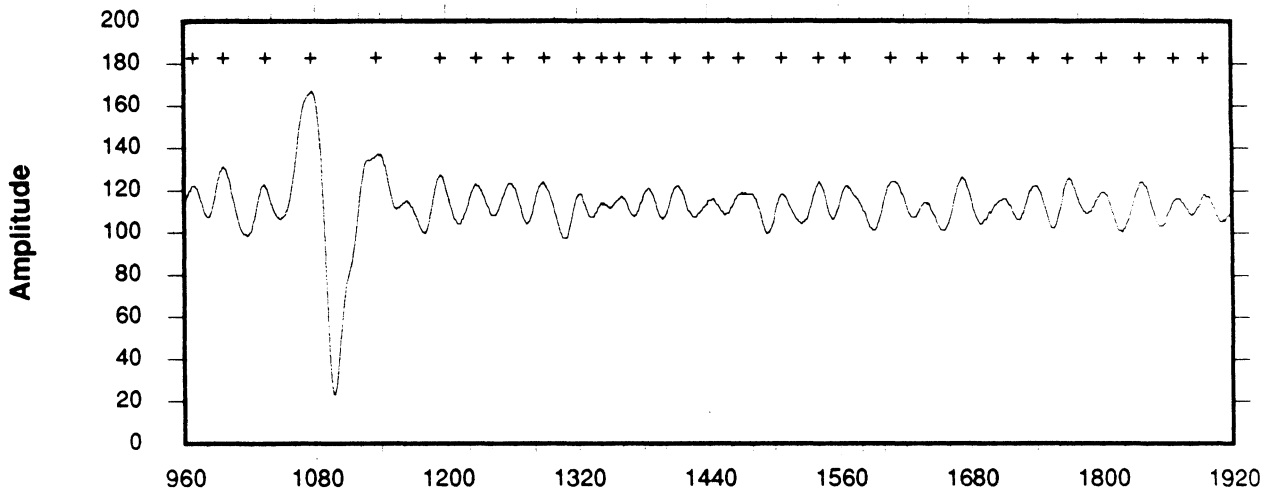
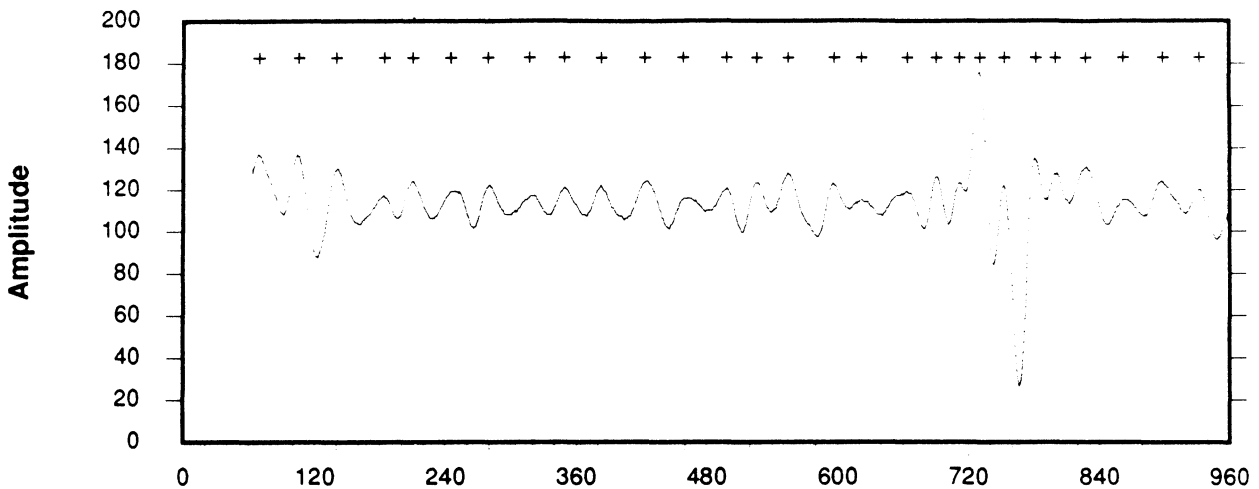
```
C
C      Unit 5 = terminal ; Unit 7 = printer
C
C      Initialization
C
      BYTE LINE(132),NAME1(28)
135     FORMAT(132A1)
145     FORMAT(1X,132A1)
C
C
C      Ask name of the file to print and store it in NAME1
C
      WRITE(5,115)
115     FORMAT(' ENTER NAME OF THE FILE TO PRINT : ', $)
      READ(5,125)(NAME1(I),I=1,28)
125     FORMAT(28A1)
C
C
C      Send OGLT9.POS to the printer
C
      OPEN(UNIT=1,NAME='OGLT9.POS',TYPE='OLD')
10     CONTINUE
      READ(1,135,END=11)(LINE(I),I=1,132)
      write (7,145)(LINE(I),I=1,132)
      GO TO 10
11     CONTINUE
      CLOSE(UNIT=1)
C
C
C      Save current state of printer's VM
C
      write (7,205)
205     format (' /state1 save def')
C
C
C      Send filename to printer
C
      write (7,215)(NAME1(I),I=1,28)
215     format (' /title (' ,28A1,' ) def')
C
```

Figure 2a - Printing a file: Fortran program (1)

```
C
C      Open file to be printed, set NBL to 0
C
      OPEN (UNIT=1, NAME=NAME1, TYPE=' OLD' )
      NBL=0
C
C
C      Read one line from file:
C          if end of file GO TO 99
C          if not send to printer 'pgr' and LINE
C
20      CONTINUE
      READ (1, 135, END=21) (LINE (I) , I=1, 132)
      write (7, 225)
225     format (' pgr')
      write (7, 145) (LINE (I) , I=1, 132)
C
C
C      Increment NBL and test for end of page
C
      NBL=NBL+1
      IF (NBL.LE.45) GO TO 20
      NBL=0
      write (7, 245)
245     format (' pr1 showpage')
      write (7, 255)
255     format (' /ip ip 1 add def  /y1 10 inch def')
      GO TO 20
C
C
C      End of printing: print title, show page,
C      restore state, close file
C
21      CONTINUE
      write (7, 265)
265     format (' pr1 showpage statel restore')
      CLOSE (UNIT=1)
      END
```

Figure 2b - Printing a file: Fortran program (2)

MK 702D. 12 MARCH 87. SK 20-22. CONTROL .30-.85 RESIN. LP001 + HP001 FILTER.



Time in half seconds

1 integer equal 6.0 mcV or mg.

Figure 3 - Plotting a signal

MK 225S. 16 JAN 87. SK 22-23. CONTROL WATER PH 7. CORRELATION FILE.

22

3.	92.499	90.575	2.5991	3.2997	0.00	0.	0.	0.
6.	77.226	43.432	6.0154	24.4994	3.36	440.	131.	80.
11.	71.969	19.991	1.4100	15.5183	3.00	235.	78.	35.
14.	72.677	19.452	-0.3263	0.9111	3.75	294.	78.	39.
17.	71.187	18.360	0.6700	1.8687	2.61	272.	104.	35.
21.	70.582	17.706	0.2184	0.9280	2.71	168.	62.	23.
31.	70.937	17.229	-0.0502	0.2731	4.00	290.	72.	43.
41.	70.424	14.618	0.0726	1.6434	3.49	406.	112.	59.
51.	48.039	9.137	3.8252	4.6992	3.00	468.	156.	63.
62.	20.522	3.030	7.7320	10.0343	3.74	383.	103.	60.
71.	18.803	2.745	0.9720	1.0976	3.65	496.	136.	73.
81.	17.723	2.526	0.5915	0.8314	2.84	209.	73.	34.
91.	17.159	2.461	0.3234	0.2607	0.00	0.	0.	0.
102.	16.851	2.504	0.1647	-0.1575	3.00	354.	118.	57.
111.	7.761	1.546	8.6144	5.3580	0.00	0.	0.	0.
122.	1.473	1.159	15.1074	2.6192	3.82	444.	117.	70.
131.	0.949	0.935	4.8850	2.3863	0.00	0.	0.	0.
142.	0.940	0.885	0.0872	0.5031	3.51	356.	104.	51.
151.	0.996	1.062	-0.6377	-2.0091	3.00	331.	110.	46.
161.	0.916	0.778	0.8373	3.1118	4.10	470.	116.	62.
171.	0.865	0.941	0.5729	-1.9022	1.71	212.	124.	49.
181.	1.131	1.077	-2.6813	-1.3499	3.47	406.	117.	67.

Figure 4a - Tabulating data: Formatted data file

MK 225S. 16 JAN 87. SK 22-23. CONTROL WATER PH 7. CORRELATION FILE.

TIME min	% SOLID	% LIQUID	K SOLID %	K LIQUID %	FREQ. cpm	MOTILITY μVxcpm	REL. AMP. μV	AREA μVxmin/min
3.	92.499	90.575	2.5991	3.2997	0.00	0.	0.	0.
6.	77.226	43.432	6.0154	24.4994	3.36	440.	131.	80.
11.	71.969	19.991	1.4100	15.5183	3.00	235.	78.	35.
14.	72.677	19.452	-0.3263	0.9111	3.75	294.	78.	39.
17.	71.187	18.360	0.6700	1.8687	2.61	272.	104.	35.
21.	70.582	17.706	0.2184	0.9280	2.71	168.	62.	23.
31.	70.937	17.229	-0.0502	0.2731	4.00	290.	72.	43.
41.	70.424	14.618	0.0726	1.6434	3.49	406.	112.	59.
51.	48.039	9.137	3.8252	4.6992	3.00	468.	156.	63.
62.	20.522	3.030	7.7320	10.0343	3.74	383.	103.	60.
71.	18.803	2.745	0.9720	1.0976	3.65	496.	136.	73.
81.	17.723	2.526	0.5915	0.8314	2.84	209.	73.	34.
91.	17.159	2.461	0.3234	0.2607	0.00	0.	0.	0.
102.	16.851	2.504	0.1647	-0.1575	3.00	354.	118.	57.
111.	7.761	1.546	8.6144	5.3580	0.00	0.	0.	0.
122.	1.473	1.159	15.1074	2.6192	3.82	444.	117.	70.
131.	0.949	0.935	4.8850	2.3863	0.00	0.	0.	0.
142.	0.940	0.885	0.0872	0.5031	3.51	356.	104.	51.
151.	0.996	1.062	-0.6377	-2.0091	3.00	331.	110.	46.
161.	0.916	0.778	0.8373	3.1118	4.10	470.	116.	62.
171.	0.865	0.941	0.5729	-1.9022	1.71	212.	124.	49.
181.	1.131	1.077	-2.6813	-1.3499	3.47	406.	117.	67.

Figure 4b - Tabulating data: Output 1

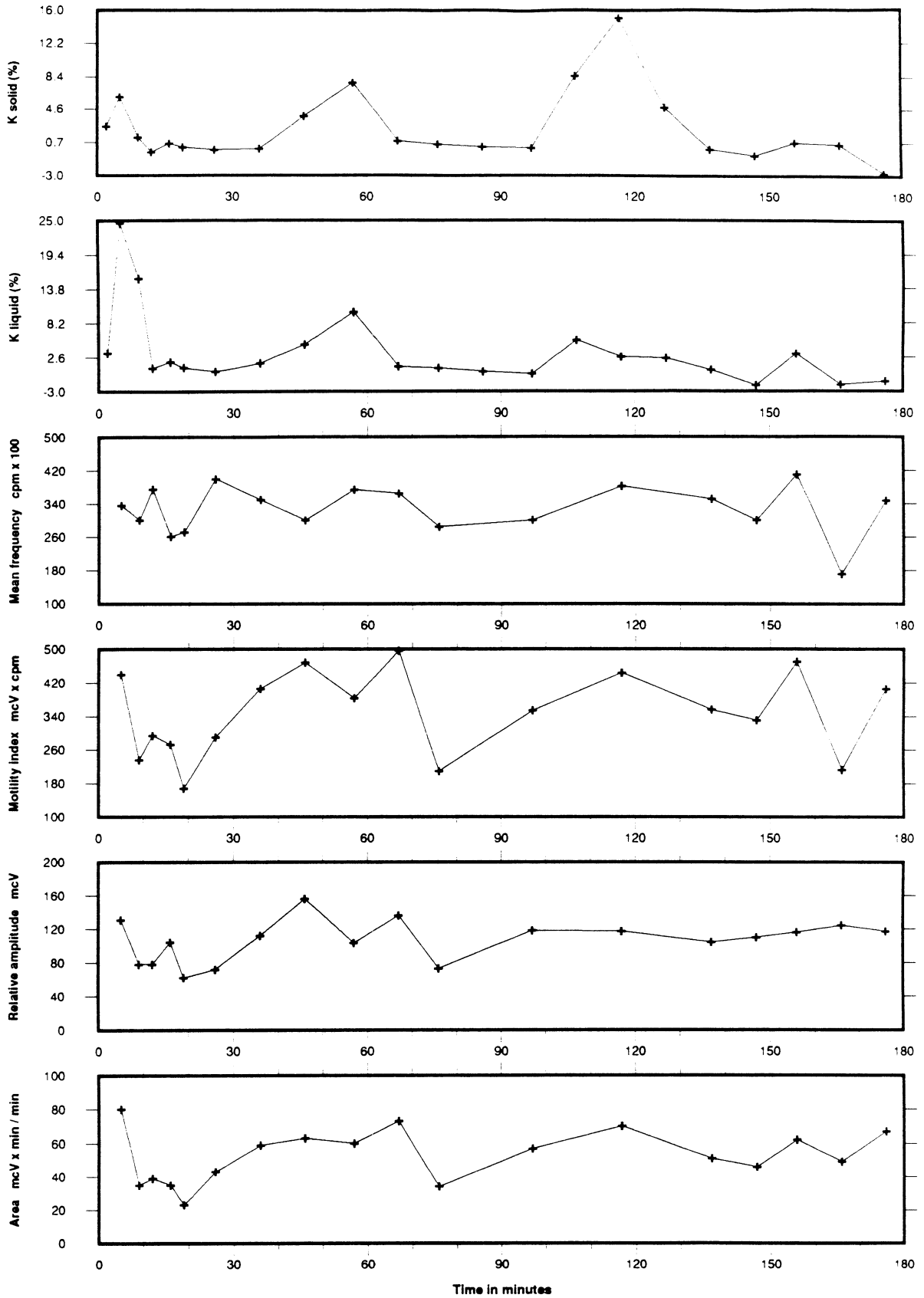
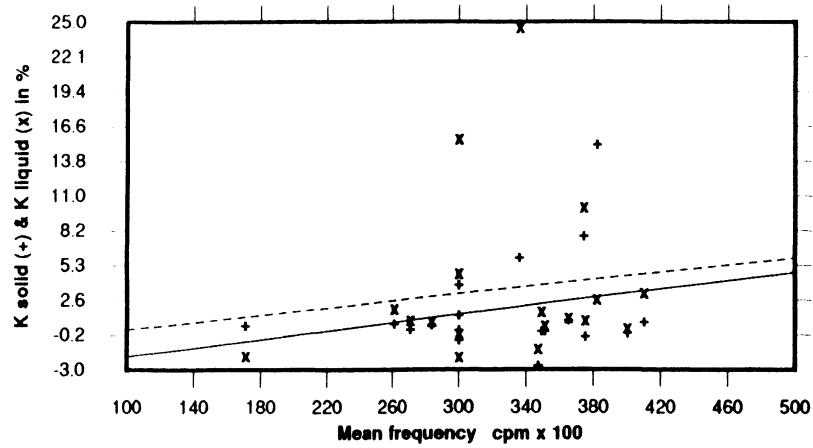


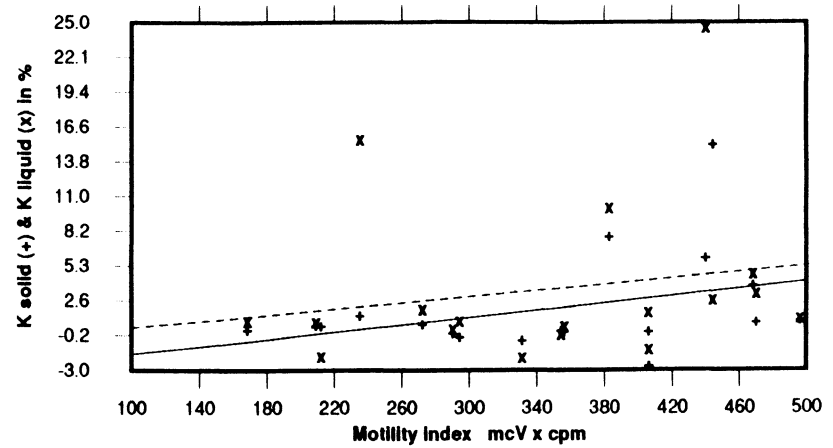
Figure 4c - Tabulating data: Output 2

CORRELATION GASTRIC EMPTYING / GASTRIC MOTILITY

MK 225S. 16 JAN 87. SK 22-23. CONTROL WATER PH 7.

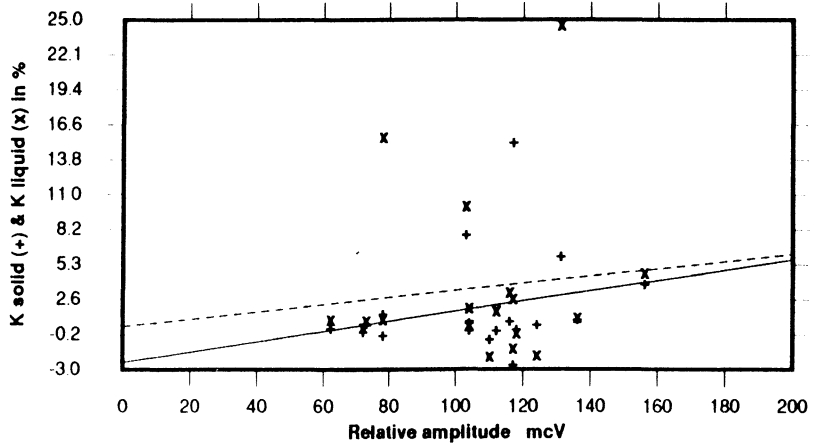


Solid (+,-): Nb points = 18 Eq: $Y = 0.017 X - 3.609$ $R = 0.24$
 Liquid (x,..): Nb points = 18 Eq: $Y = 0.014 X - 1.195$ $R = 0.13$

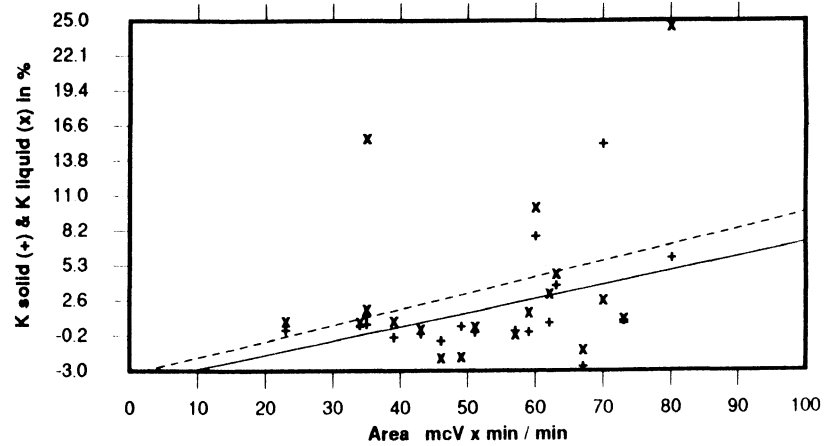


Solid (+,-): Nb points = 18 Eq: $Y = 0.015 X - 3.125$ $R = 0.36$
 Liquid (x,..): Nb points = 18 Eq: $Y = 0.012 X - 0.792$ $R = 0.18$

114



Solid (+,-): Nb points = 18 Eq: $Y = 0.041 X - 2.442$ $R = 0.25$
 Liquid (x,..): Nb points = 18 Eq: $Y = 0.029 X + 0.435$ $R = 0.11$



Solid (+,-): Nb points = 18 Eq: $Y = 0.113 X - 3.999$ $R = 0.43$
 Liquid (x,..): Nb points = 18 Eq: $Y = 0.130 X - 3.307$ $R = 0.30$

Figure 4d - Tabulating data: Correlation

@13,16,,,,,10,,,,1,.9,7.6,10.1,2.6,0.05

This batch word-processing program allows formatting and printing of any text file. Each text file is created and edited using a terminal and contains, in addition to the text, several sequences of characters describing the output format. ^L^L

@24,22,,1.5,5.5

At any time,^040

@9

a change of font is possible,^040

@29

even in the middle of a sentence or a

w

@8

o

@29

rd. ^L

@33,16,,3,2.25

^L

Text can be printed at any place on the page, with or without right justification, ^L

@,,,,,3

or centered. ^L^L

@9,,,1,6.5,1,,,,1

Underlining is available

@,,,,,,,,,2

, and a box can be drawn around the text. ^L^L

@13

Superscript and subscript are available: ^L

^05.040 (X+Y)^3742^375=X^3742^375 + 2XY + Y^3742^375 ^L

^05.040 \123^374i,^040i=1,10^376

(a^374i^376 + b^374i^376) =

\123^374i,^040i=1,10^376 a^374i^376 +

\123^374i,^040i=1,10^376 b^374i^376 ^L

Figure 5a - Word-processing: Text file

This batch word-processing program allows formatting and printing of any text file. Each text file is created and edited using a terminal and contains, in addition to the text, several sequences of characters describing the output format.

At any time, a change of font is possible, even in the middle of a sentence or a **w**ord.

Text can be printed at any place on the page, with or without right justification, or centered.

Underlining is available, and a box can be drawn around the text.

Any character not available directly from the keyboard can be printed by using the asciicircum character followed by its octal code (for example, ^276 prints à).

A symbol is printed by using the backslash character followed by its octal code (for example, \252 prints ♠).

Superscripts and subscripts are available:

$$(X+Y)^2=X^2 + 2XY + Y^2$$

$$\sum_{i,i=1,10} (a_i + b_i) = \sum_{i,i=1,10} a_i + \sum_{i,i=1,10} b_i$$

Figure 5b: Word-processing: Output

Experiences with an IAS-VMS DECNET system

Frank R. Borger

Michael Reese - University of Chicago
Center for Radiation Therapy
Chicago, Illinois

Abstract

The Michael Reese - University of Chicago Center for Radiation Therapy currently supports two VAX 11/750's running VMS and a PDP11/44 running IAS version 3.2. The three machines are linked with DECNET to provide ease of communication between the systems, and to ease the transition involved in transferring treatment planning software to the VAX systems, (where the addressing limits of the PDP11 has been a serious limitation.

This paper will discuss the tribulations involved in getting the system up, along with problems and work-arounds inherent in running a mixed (version 3 and version 4) DECNET system.

It all started innocently enough. We postulated, "Of course we need the two VAXes linked with DECNET, but we want the PDP11 system included too." It was a battle to get funding for DECNET-IAS, but as it turned out, it was more of a battle to get it up, after we won the battle to get the software.

I have had people look at me as if I were an idiot when I said I LIKED programming in machine language, but I just experienced something that even I have to admit was a painful experience. I installed DECnet on our IAS version 3.2 system. Without having installed DECnet before. Without having any software support. And, it turns out, I installed what was a version 3.1 DECnet which I had to update to match version 3.2 of IAS.

In response to an angry letter to DEC concerning my problems I was told that DECnet has been put on hold. (Now they tell me, after I have it up and running.) DEC finally has released a version 3.2 DECnet, so this is somewhat academic, but its a good horror story of how not to send out a product.

The AUTOPATCH procedure

After I copied all the standard DECnet distribution to our system disk, I copied the contents of the AUTOPATCH tape we received to [225,200] and tried to execute the autopatch procedure as documented in [225,200]DECNET.DOC. The write-up on the autopatch tape showed a complicated procedure which copied everything from [225,200] to the appropriate UIC using the /NV switch and then did a full generation of DECNET. The actual command file only tried to copy the updated files to the correct account, and also kept failing with the error message:

```
NGN -- UNDEFINED SYMBOL <EXITST>  
.IF <EXITST> = <SUCCES> .RETURN
```

If I had to do it over again, I would just look at the command file and do manually what the command file tries to do. There are only about 6 accounts that need updating.

Generating the NETPLN command file

Once I had copied the updated files over, I followed the standard NETPLN command file. This was the only part of the procedure that executed correctly.

Generating the command build files

Next I tried to generate the DECnet command build files via a NGN @[11,67]NETGEN. NGN had all kinds of errors with .IFNINS commands for PIP, LOA, UNL, UFD, LBR, MAC, TKB and STK (slow TKB.) I finally installed all the things NGN tested for and commented out the .IFNINS commands. (Our system has been modified to do flying installs of tasks such as LBR, TKB etc to save on pools space, and this may have been a factor in the problem.) I also commented out some .IFNLOA commands concerning source and destination disks.

- Since I had commented out NETGEN.CMD's call to BLDNET, during one of the previous bombs, I then directly executed BLDNET.CMD via a NGN @BLDNET. That is when the first main problem developed. Many of the TKB commands generated errors of the form:

TKB - *DIAG*-Load address out of range in module

This turned out to be due to one of the new “features” of the new task builder. Examination of the maps of the several tasks in question disclosed that the problem was due to the following.

- The new TKB links all RW modules alphabetically, followed by all the RO modules, and then changes any RO modules to RW. (Even if one applies the “/RW” switch.) This mainly has an effect upon psects named \$\$name or ..name.
- The version 3.2 HNDLIB has all modules in psect \$\$HNDL rather than in the .blank psect.

A quick solution to this problem was to copy the version 3.1 HNDLIB object library to the new system disk as HNDLIBV31, and edit any command file that referenced HNDLIB to use the version 3.1 library.

The Network Loader Task Abort

The last and greatest bug occurred when everything had built correctly and we tried to bring up the system. The network loader task would consistently abort with an odd address trap. A day or so of debugging finally found the error. It's actually been around for a long time, but only surfaced due to the new ordering of modules by the new TKB. Bear with me and I will explain what went wrong.

- The routine \$PRIO in NTL was doing a BAD thing. IT was doing a 1000 byte read into a 2-byte buffer in psect ..BUF. (Evidently the designer wanted to be able to change the size of the buffer, probably by an explicit expansion in the TKB command file. Unfortunately they didn't do it.)
- This would never have worked except that:
 - Under the version 3.1 ordering, psect ..BUF was the last thing in that overlay segment.
 - There were longer overlay sharing the same space as the one containing \$PRIO and ..BUF.
 - The longest overlay segment was followed by a second root of a co-tree.
- Under version 3.2 TKB, psect ..BUF, (the last RW psect,) was followed by the first RO psect, which contained general SYSLIB routines, notably \$MUL among others. \$MUL was promptly overwritten with garbage.

This final problem was fixed by expanding the psect to its proper size adding the following command to NTLBLD.CMD (the TKB command file for ...NFT.)

```
EXTSCT=..BUF:776
```

Your friendly DEC salesperson

After the system was finally up and running correctly, I got a call from our friendly DEC salesman telling me I could never get the system up unless I bought software support and got update B. (It turns out update Z wouldn't have helped, what I needed was an update to the DECnet

package, not to IAS.) I strongly objected to this in a letter to DEC in Maynard, stating that I believed that either

- When we bought version 3.2 of IAS it should have supported all layered products
- or
- When we bought DECnet IAS we should have received a package that worked on the current version of IAS, version 3.2.

Conclusions concerning installation

In any case, my final analysis is that DEC's field test sights could not have included any DECnet users, else they would have known there were problems and would (or should) have included some notification in the release notes to warn users that DECnet would not function.

A final disclaimer from my end. Our DECnet installation is an end node connected to a couple of VAXes, so we have not been able to test routing, down line loading, or virtual terminals. The problems with the new TKB and version 3.2 HNDLIB would probably bomb some other tasks that we didn't need.

For anyone who desperately needs DECnet up on version 3.2, doesn't have the money to buy the 3.2 version (which is now out,) let me summarize what needs to be done.

- Copy the version 3.1 DECnet update to [225,200] and then manually copy the files to the correct account.
- Make a copy of version 3.1 HNDLIB.OLB, (called HNDLIBV31 for example) and edit any TKB command file that references the library to use the version 3.1 edition.
- Edit NTLBLD.CMD to include the line:
EXTSCT=..BUF:776

An alternative method that should also work would be to link the version 3.1 TKB and SLOTKB under 3.2 and use them to generate the DECnet components.

Now It's Working How Do We Use It ?

Although VMS has been designed around DECnet, IAS antedates DECnet. Commands for file access on VMS are almost transparent. (You just add “NODENAME:” before the normal filespecification.) For IAS to VMS transfers it isn't so easy. The solutions were different at each end of the network.

At the IAS end, you used NFT (Network File Transfer) instead of PIP to transfer files. NFT wanted a network specification of username and password in its command line, which is not a good idea. (You can leave it on the screen of a CRT or on the paper of a hard-copy terminal when you leave.) The alternative is to use the ALIAS facility under DECNET-IAS. The Alias facility lets you define a short alias for the long access string necessary for DECNET access. In our case, as part of

the Network startup command file, I define the following Alias:

```
SET ALIAS FRANK DESTINATION...
MRVAX/username/password:: SCOPE GLOBAL
```

Note that the appropriate account must be set up on the vax in question, and that MRVAX is the node name of the Michael Reese VAX 11/750. This alias then can be used to transfer files. The following command will get a file from the remote vax, edit it, and put it back.

```
MCR>NFT SY:=FRANK::PROGRAM.FOR
MCR>KED PROGRAM.FOR
MCR>NFT FRANK::=PROGRAM.FOR
```

Some other problems do exist:

- IAS version numbers are octal, VAX version numbers are decimal. A ';' transfer from the VAX to the PDP can really screw up.
- Transfers to the IAS system do not default to the highest version number if a file exists, but NO WARNING MESSAGE IS GENERATED.
- Wild card file name transfers will run into the 9 character limitation of 9 characters.

For copying files from the VAX end of the network, We developed a slightly different method. Our login.com file defined the following symbol:

```
$ get11 ::= @[frank.com]get11
```

The get11.com command file contained the following:

```
#! get file from pdp11 via decnet
$ fra := "fra"
$ han := "han"
aa := "copy mrspot""user pass""::'p2' sy:
bb := "copy mrspot""user pass""::'p2' sy:
$ if 'p1' .eqs. 'fra' then 'aa'
$ if 'p1' .eqs. 'han' then 'bb'
```

This allowed us to transfer the same file from the PDP11 to the VAX using the command:

```
get11 fra program.for
```

Similar command files allow us to send files to the PDP11, and to delete files on the PDP11, as in:

```
put11 fra program.for del11 fra program.for;*
```

This is somewhat unsatisfactory in that the username and password are available in the command files.

Set Host

The SET HOST command is one of the greatest things about DECNET. Unfortunately it does not work well across the network for the following reasons.

The RMT remote terminal handler works well connecting to other IAS systems. It unfortunately does not know about VMS systems, and will not connect to a VAX system.

Set host from VMS to IAS almost works. It fails (in our case,) for the following reasons:

- RMT (or more specifically HT....) do not support the IO functions 'read with timeout', 'send XOFF', and 'read with special terminator'.
- Our system had somewhat older versions of MCR and PDS which we had adapted to our uses some time ago. These programs did a 'read with timeout' when getting the user's command.
- The absence of a 'read with special terminator' command precluded our using our fancy command line recalling and editing MCR, (ECR.)
- HT.... does not pass the ESC character properly. This has a minor effect that you can not terminate MCR command lines with an escape to make MCR go away, and has a major effect in that you can't use KED, etc over the network. The minute you go into screen mode, you cant issue escape sequence commands from the keypad, ergo you can't get out of KED. Nice catch-22 DEC.

Other Minor Fixes and Enhancements

- We ended up redoing MCR to not do a read with timeout. The task is installed as ...RCR, and the network terminals HT0: thru HT3: are set to use this CLI via the command:

```
MCR>SET HTnn:/CLI=RCR
```

- None of the command files know about MCR mode IAS. We had to change many install commands

```
from: ins xxxxxx/task=$$$xxx
to:   ins xxxxxx/task=...xxx
```

- We set DECNET up to use a different copy of the PDSUFP file. Currently there is a copy of PDSUFP on disk SD0: that is used for normal terminal logins and another copy on disk LB0: that is used for network logins. This restricts the access to only those users we want to have cross system access.

What's Really Nice About DECNET

I have a poor-man's DECserver for my terminal. It's a three-way RS-232 switch that enables me to connect to the IAS 11/44, the VMS VAX/750, or the 11/10 in my office running RT11 or diagnostics. (Trying to remember the subtle differences between the 3 systems, differences between KED/RT and EDT/VMS, etc. really makes my day.) But have you ever needed to debug a task that does fancy screen I/O ? You have to do it running the debugger at another terminal. With DECNET, I do a SET HOST MRSPOT from the VMS system, and presto, I have both parts running at the same terminal, just switch back and forth from IAS to VMS.

The IAS DECNET Wishlist

- Hopefully, DEC will get their act together and fix HT.... for the latest release of DECNET IAS.
 - Why can't RSX11M mail be installed on IAS? This would really make my day. Now if I have prepared a document on IAS, I must DECNET it over to an account on the VAX before I can put it in the VMS mail system.
 - Although SET HOST now works from VMS to IAS, the reverse still does not work. Supposedly an unsupported version of RMT knows how to talk to VMS systems. I would like to see it.
 - Under VMS I can copy the context of a SET HOST session to a disk file. Unfortunately this does not work for a SET HOST to the IAS system. I wish it would.
 - I know that transfers to the PDP11 where names are longer than 9 characters won't work, but give me a switch that does the following:
 - Truncates names to 9 characters and types to 3.
 - Lists changes that are made, (possibly to a file.)
 - Possibly prompts for a new filespec if names or types are too long.
 - HT.... is a handler, and DEC had distributed sources to handlers for a long time. Why can't I get sources to HT....? (If necessary, I WILL disassemble the damn thing.)

LARGE SYSTEMS SIG

AMAR—A TOPS Performance Monitor

Betsy Ramsey
American Mathematical Society
Providence, RI

Abstract

AMAR is a DEC-written utility in the public domain which collects system performance data on a DEC-10 or DEC-20 computer, analyzes and reports on the data, and maintains the data in historical databases. AMAR is used at the American Mathematical Society to answer capacity planning and load balancing questions, and to justify additional hardware purchases.

Overview

AMAR is an acronym for *Automatic Measuring, Analysis, and Reporting*.

AMAR is a DEC-written product that monitors system usage and workload data on a continuous basis. It has reporting modules that also attempt to analyze the system usage data items. AMAR rolls up the raw data records into historical records which are maintained in database files. AMAR can retain up to one year's worth of data.

Because the American Mathematical Society is a DEC-20 site, this paper will describe AMAR-20. AMAR-10 has similar capabilities.

History

In the early 1980's, the American Mathematical Society was experiencing severe performance problems on its DEC-20 computer. Digital software engineers analyzed our system performance using WATCH, the bundled TOPS-20 performance tool (similar to the VMS Monitor utility). AMS was informed that the machine needed more memory. The memory was purchased, but performance remained bad. Further analysis revealed that the machine was at the limit of its CPU capacity. The AMS Board of Trustees was forced to make an unexpected and unwelcome purchase of a second DEC-20 to relieve the load. Knowing how unhappy we were, our salesman asked Digital if there wasn't something they could do to ensure that this wouldn't happen to us again. The salesman was told that DEC had an internal tool called AMAR that could track system utilization over a period of time, and that would predict when certain thresholds would be exceeded. AMAR was expensive, but AMS gladly purchased it. The American Mathematical Society has been running AMAR

since early 1983.

In spring 1983, Digital announced that the 36-bit line of computers was going to be phased out, but that they would continue hardware and software development for five years, until 1988. Accordingly, DEC made new releases of the TOPS-10 and TOPS-20 operating systems in 1986. Unfortunately, those new releases required changes to AMAR which, because of personnel changeover, DEC no longer had the expertise to make. At the demand of those customers who had purchased AMAR, DEC sent AMAR to Systems Concepts, a third party hardware/software shop, to be updated for TOPS-10 V7.03 and TOPS-20 V6.1. The TOPS-20 upgrade was completed in June 1986. The TOPS-10 upgrade will be finished soon.

Digital has placed AMAR in the public domain. The TOPS-10 V7.02 and TOPS-20 V5.x versions of AMAR are available from the DECUS Program Library. The V7.03 and V6.x versions will be submitted to the Library as well.

The Two AMARs

AMAR comprises two components: System AMAR and Workload AMAR.

System AMAR continuously collects data on system performance and utilization variables. These variables include most of those collected by the WATCH program, such as USED, FILW, SKED, NRUN, UPGS, and so forth. In addition, System AMAR collects device data on disks and tapes.

Workload AMAR continuously collects job-specific data such as job number, terminal number, user name, account string, program name, CPU utilization, working set size, and page fault activity.

AMAR works as follows:

- Data collection programs run continuously, usually as a SYSJOB subjob.
- Database management programs run once a day to process the raw data. Hourly data is rolled up into daily, weekly, monthly records. Old data is deleted.
- Report generation programs run to generate automatic daily, weekly and monthly reports. Ad hoc reporting is available as well.

AMAR is intended to replace the TOPS-20 WATCH utility. It collects much of the same data as WATCH, but has the ability to synthesize it. In addition, because of its monitor snoop space requirements, AMAR cannot be run simultaneously with WATCH.

AMAR is not intended to replace the SYSDPY utility. SYSDPY is an excellent tool for in-depth examination of a performance problem at the time it is occurring.

In many ways, AMAR is similar to the VAX SPM tool. SPM offers more reporting mechanisms for system data, but it lacks AMAR's comprehensive workload coverage.

Reports

The best indication of AMAR's abilities are in the reports it generates.

System AMAR Reports

System AMAR can produce five standard reports.

- System Utilization

This report summarizes system utilization for one day, a week, or a month. It is useful for monitoring system performance and spotting problems. The report consists of a graph of system utilization, a summary of user-specified key utilization items in prime and non-prime time, and a breakdown of the key items over an appropriate interval (hours for daily reports, days for weekly reports, and weeks for monthly reports).

- Typical Day

This is a composite report which summarizes weekly or monthly data into a "typical day". It contains much the same information as the System Utilization Summary reports, but uses average values rather than actual values.

- Trend Analysis

This report presents a summary of system usage over the past 13 weeks (for weekly reports) or the past 12 months (for monthly reports), including a summary

utilization graph for the period. The monthly reports attempt to predict trends in utilization. These reports are useful for capacity planning.

- Disk

This report, available on a daily, weekly or monthly basis, summarizes utilization of the disk subsystem. It reports mount time, use time, and read, write, seek and other statistics for both logical structures and physical drives. It is useful for balancing disk I/O across available channels.

- Tape

This report summarizes tape drive use and MTIO's.

The contents of these reports can be tailored slightly by modifying a Report Description File (.RFD). Through the use of this file, names of key utilization items can be changed, and their threshold values and warning messages altered. The number and order of key utilization items can be modified for each report. The format of the reports cannot be changed, however.

Workload AMAR Reports

Workload AMAR reports are generated with user-specified report parameters, and are thus more flexible than System AMAR reports. By selecting the appropriate database file and group and sort items for the report, the user can examine the system workload in a variety of ways.

Some of the items which may be used for grouping and sorting a Workload AMAR report are as follows.

JOB	job number
TTY	terminal number
USR1 ... USR3	user name (up to 15 chars)
ACT1 ... ACT3	account string (up to 15 chars)
PNAM	program name
BATCH	batch job indicator
CPU%	CPU percentage (USED)

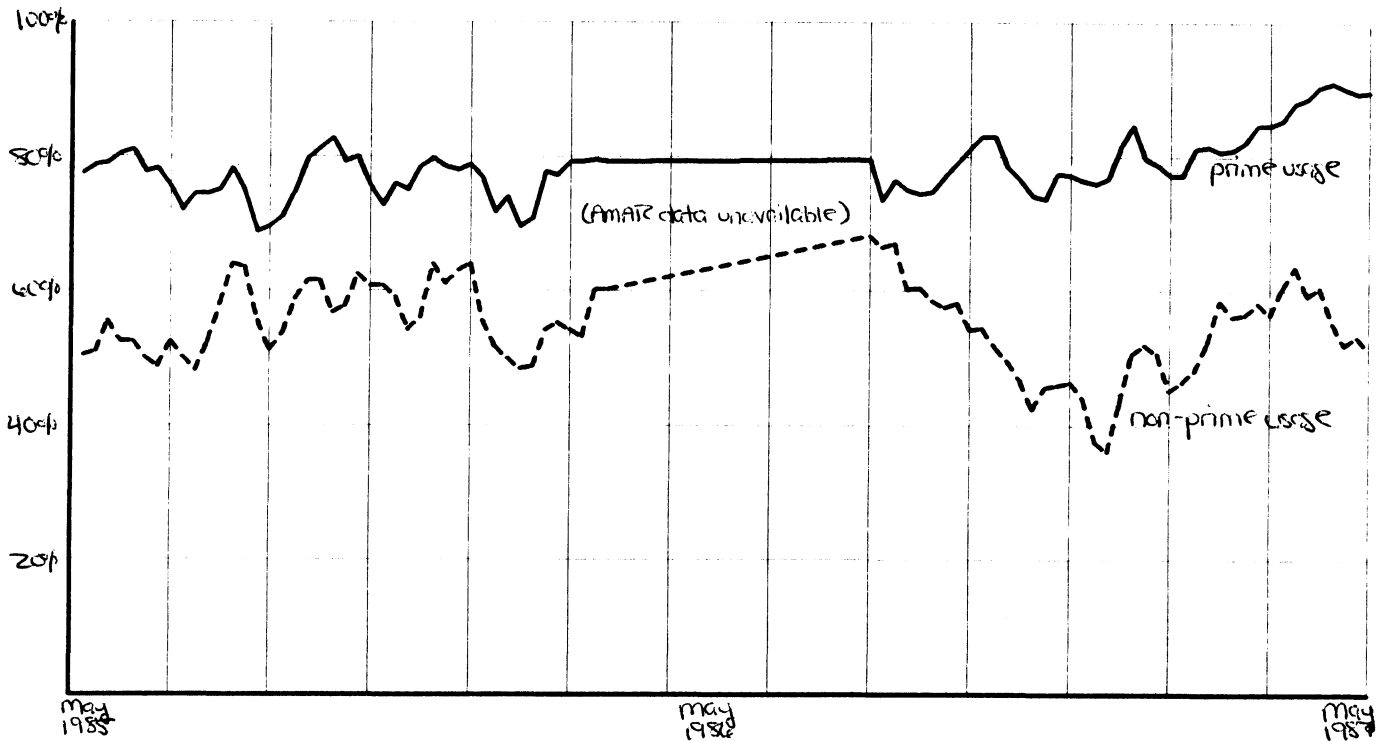
AMAR at AMS

The American Mathematical Society has been using AMAR for almost four years. We find it to be an invaluable tool for tracking system performance and performing capacity planning functions.

AMS Tailoring

AMS has tailored the nightly batch control files so that AMAR generates only those reports that we find useful. We have modified the System AMAR .RFD file so that the reports are easier to interpret.

Figure 1: AMS Two-Year CPU Utilization Summary Report, derived from AMAR data.



AMS generates these AMAR reports:

- System AMAR

System Utilization Summary	daily, weekly, monthly
Typical Day	monthly
Trend Analysis	weekly and monthly
Disk	weekly and monthly
Tape	weekly and monthly

- Workload AMAR

User/Program Shift Report	daily, weekly, monthly
User Shift Report	weekly and monthly
Program Shift Report	weekly and monthly
Report on All Programs	monthly
Report on All Users	monthly

AMS does not use the System AMAR report description file (xxxxDR.RFD) as supplied on the distribution tape. Instead, we use a modified version which renames all the system utilization items to their WATCH names. (For example, we renamed item % IDLE IO TIME to be FILW.) This makes it easier to interpret the data, since all of DEC's TOPS performance literature refers to this data by WATCH names.

The Workload AMAR reports are double-spaced. To save paper and disk space, AMS edits these files after they are produced to make them single-spaced.

Use of System AMAR

AMS uses data from the System AMAR reports for a number of purposes.

Track System Utilization

AMS defines system utilization as the percentage of the CPU that is not idle. AMAR reports the IDLE figure for prime and non-prime periods, and AMS uses that to compute the system utilization value. This figure is added to an ongoing data file which is run through a set of Fortran programs. These programs produce a laser printer graph (Figure 1) of system utilization over the past two years, where each point on the graph is a weekly four week average of the system's utilization. This graph allows us to see past trends, and to determine how much capacity remains on the system.

The System Utilization reports can show us exactly how our system is being used. For example, Figure 2 shows the prime time breakdown of CPU utilization into USED, the amount of time the system spent executing user processes; system overhead (SKED and BGND), which is large on this system because we have over 90 users logged in during the day; I/O wait (FILW and SWPW); and IDLE, the amount of time the CPU was totally idle.

Monitor Daily Performance

If a system experiences a particularly bad day, the Daily Utilization Summary permits us to determine the hours when there was little or no extra CPU capacity. Armed with that data, we can turn to Workload AMAR to determine the cause.

Monitor Memory Demands

WATCH, with its Active Swap Ratio, still provides the best indication of memory demand. System AMAR allows us to come close to that, however. AMS looks at two figures: swap rate (SWPW) and the swap ratio (UPGS/UMEM). We have used these figures to justify the purchase of additional memory.

Monitor I/O Load

The Disk report allows AMS to determine whether the disk traffic is balanced over the I/O channels. AMS has twice rearranged the disk drives on the channels as a result of AMAR data. The I/O load is now distributed as evenly as possible over the channels.

Use of Workload AMAR

Despite the lengthy list in the previous section, AMS is much more dependent on Workload AMAR than System AMAR. WATCH can provide basic system data, and that type of data is relatively easy to summarize. Workload data is very bulky, however, and much more difficult to roll up and report.

AMS uses Workload AMAR for several purposes.

Capacity Planning and Load Balancing

AMS has two DEC-20s, both of which average between 80% and 95% system utilization during prime time (in the winter). For load balancing purposes, the Computer Services Division (CSD) is often called upon to answer the question "What if we put additional people on system A? Can it handle it?". AMAR allows us to answer that question.

First, we find out which existing user(s) the new users will behave like, a surprisingly easy thing to do ("Oh, they'll be doing the same thing as John Doe."). Then we use Workload AMAR to determine how much of the system the existing users are consuming (Workload AMAR reports the WATCH USED figure), and we compute the appropriate value for the new users. With that data in hand, we can look at the System AMAR data to see if the system can handle it.

Monitor Daily Performance

When System AMAR indicates a period where there was little or no extra CPU capacity, Workload AMAR can report which users and programs were running at the time. Applications which "hog" the system can be spotted, and possibly run at some other time.

Track Application Growth

By examining Workload AMAR reports from previous years, AMS can determine how the use of their applications has changed over the period.

Provide Information for Management

With two heavily loaded systems, users, which at AMS includes top management, become disgruntled at times over system performance or the lack of it. Workload AMAR allows CSD to inform these users of exactly the purposes for which the system is being used. Workload AMAR has proven itself to be a priceless political tool.

Operational Considerations

There are some things to consider before deciding whether or not to run AMAR. These considerations point out both positive and negative aspects of AMAR.

- AMAR data collection involves much less overhead than WATCH.

For a prime time interval (8am-5pm), the System AMAR and Workload AMAR data collection programs together use between two and three minutes of runtime. WATCH, by comparison, uses between six and seven minutes of runtime during the same period.

- AMAR data occupies a great deal of disk space.

The data collection files must reside on a permanently-mounted structure. These files occupy between 3000 and 4000 disk pages.

The historical database files and most AMAR programs can reside on any structure. Depending on the database configuration you select, these files will occupy between 7000 and 25000 pages.

- You must rebuild your monitor.

If you support a full 128 users on your system, you will have to rebuild your monitor to contain 22 pages of snoop space. (The monitor contains only 12 snoop pages by default). You can usually free up enough monitor space by eliminating monitor support for devices you do not have (such as physical plotters, card

punches, extra tape drives). In addition, for better Workload AMAR reports, you should make the JOBPNM table resident.

The AMAR installation guide contains information and suggestions on how to go about this, but it assumes that you know how to build the monitor (sources are not required).

- Once AMAR is installed, it requires very little maintenance.

AMAR has failed only rarely in the four years AMS has run it. Most of its failures were due to a corrupted raw data file. We simply deleted the file, and AMAR corrected itself. The default batch jobs supplied with AMAR do a fair amount of error checking and correcting. Once you have AMAR tweaked to your satisfaction, the reports will be generated and the database files managed without user intervention.

AMAR reports generated at AMS are appended to the end of this paper.

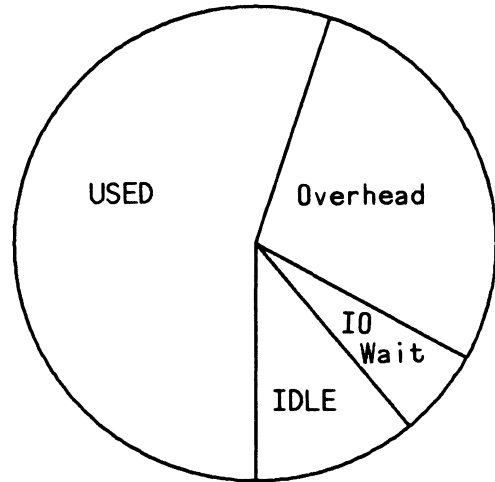
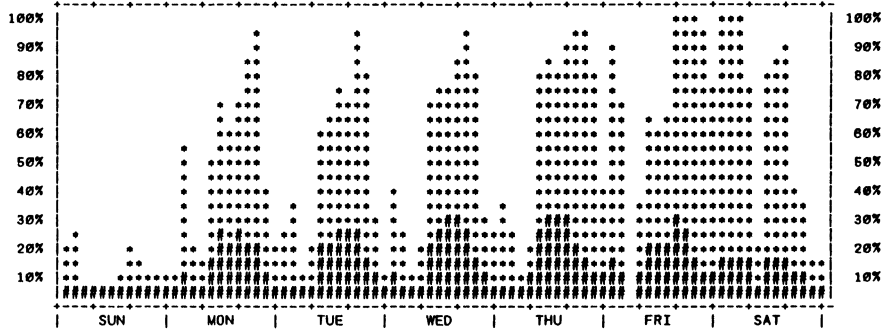


Figure 2: Prime Time CPU Usage

- AMAR -
WEEKLY UTILIZATION SUMMARY REPORT

AMS SALES DEC-20
SYSTEM: SALE PRIME TIME: 0800 - 1700

----- CPU UTILIZATION (+) OVERHEAD (#) -----



----- EACH COLUMN REPRESENTS 2 HOURS -----

----- SUMMARY OF KEY UTILIZATION ITEMS -----

-----AVERAGE-----	_CPU	USED	SKED	BGND	FILW	SWPW	TCOR	IDLE	NRUN	NBAL
---PRIME TIME---	72%	46%	9%	13%	6%	3%	.2%	22%	2	3
---NON-PRIME TIME---	41%	34%	4%	2%	6%	0%	.1%	53%	1	1
-----AVERAGE-----	UMEM	UPGS	DMRD	DMWR	DKRD	DKWR	TTYU	PTYU		
---PRIME TIME---	3977 *	4454 *	5	4	7	5	2	80		
---NON-PRIME TIME---	3977 *	2817	0	0	9	3	0	15		

- AMAR -
WEEKLY UTILIZATION DETAIL REPORT

AMS SALES DEC-20
SYSTEM: SALE PRIME TIME: 0800 - 1700

--- PRIME TIME ---

KEY UTILIZATION ITEMS	AVERAGE OF ----- DAILY AVERAGE -----							
	CURRENT -WEEK-	-SUN-	-MON-	-TUE-	-WED-	-THU-	-FRI-	-SAT-
CPU	72%		63%	69%	75%	82%*	73%	
USED	46%		39%	43%	47%	52%	51%	
SKED	9%		8%	9%	9%	10%	8%	
BGND	13%		11%	12%	15%*	17%*	11%	
FILW	6%		8%	7%	6%	6%	5%	
SWPW	3%		4%	4%	3%	3%	3%	
TCOR	.2%		.1%	.2%	.2%	.2%	.2%	
IDLE	22%		30%	24%	19%	13%	22%	
NRUN	2		2	2	3	3	2	
NBAL	3		2	3	4	4	3	
UMEM	3977 *		3978	3978	3976 *	3976 *	3976 *	
UPGS	4454 *		4479 *	4449 *	4448 *	4399 *	4495 *	
DMRD	5		4	5	5	5	4	
DMWR	4		4	5	5	5	3	
DKRD	7		7	8	8	8	6	
DKWR	5		5	4	5	5	4	
TTYU	2		1	1	2	2	1	
PTYU	80		77	85	85	80	71	

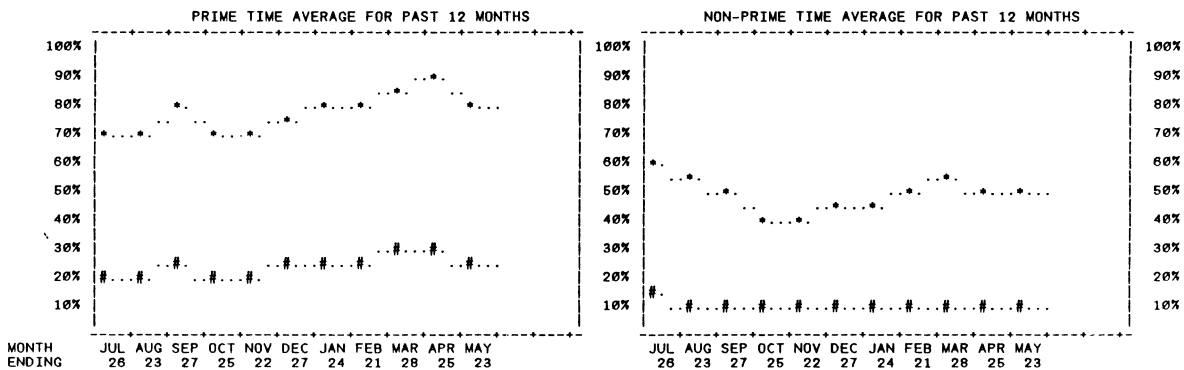
--- NON-PRIME TIME ---

KEY UTILIZATION ITEMS	AVERAGE OF ----- DAILY AVERAGE -----							
	CURRENT -WEEK-	-SUN-	-MON-	-TUE-	-WED-	-THU-	-FRI-	-SAT-
CPU	41%	9%	38%	32%	35%	44%	80%*	81%
USED	34%	6%	31%	26%	29%	37%	69%	62%
SKED	4%	1%	4%	4%	4%	4%	8%	6%
BGND	2%	1%	2%	3%	3%	3%	3%	3%
FILW	6%	2%	6%	6%	6%	7%	9%	6%
SWPW	0%	0%	0%	0%	0%	0%	0%	0%
TCOR	.1%	.0%	.1%	.1%	.1%	.1%	.1%	.1%
IDLE	53%	89%	57%	62%	58%	49%	11%	32%
NRUN	1	0	1	1	1	1	1	1
NBAL	1	0	1	1	1	1	2	2
UMEM	3977 *	3978	3978	3978	3977 *	3976 *	3976 *	3976 *
UPGS	2817	2318	2730	2796	2995	3094	2949	3035
DMRD	0	0	0	0	0	0	0	0
DMWR	0	0	0	0	0	0	0	1
DKRD	9	1	10	8	8	10	12	14
DKWR	3	1	4	3	4	7	4	3
TTYU	0	0	0	0	0	1	0	0
PTYU	15	13	16	17	17	17	16	15

* = OVER LONG TERM LIMITS > = GREATER THAN OR EQUAL TO < = LESS THAN OR EQUAL TO

AMS SALES DEC-20
SYSTEM: SALE PRIME TIME: 0800 - 1700

----- CPU UTILIZATION (*) OVERHEAD (#) AND BOTH (0) -----



PRIME TIME AVERAGES OF KEY UTILIZATION ITEMS												LONG TERM LIMIT
FISCAL MONTH ENDING	Q1M1W4 JUL 26 M-10	Q1M2W4 AUG 23 M-09	Q1M3W5 SEP 27 M-08	Q2M1W4 OCT 25 M-07	Q2M2W4 NOV 22 M-06	Q2M3W5 DEC 27 M-05	Q3M1W4 JAN 24 M-04	Q3M2W4 FEB 21 M-03	Q3M3W5 MAR 28 M-02	Q4M1W4 APR 01 M-01	Q4M2W4 MAY 23 M-00	
CPU	68	67	75	67	69	75	78	78	84*	85*	80*	>80%
USED	48	47	51	46	47	50	52	49	53	55*	52	>80%
SKED	11	10	11	9	9	10	10	10	12	12	11	>13%
BGND	7	8	10	9	9	10	11	14	16*	16*	14	>15%
FILW	7	8	6	8	7	6	6	6	5	4	5	>20%
SWPW	2	3	3	3	4	4	3	3	3	2	3	>5%
TCOR	.2	.2	.2	.2	.2	.2	.2	.2	.3	.3	.2	>1.0%
IDLE	25	25	19	26	24	19	18	18	11	11	15	<10%
NRUN	2	2	4	2	3	3	4	3	5	5	4	>30%
NBAL	3	3	4	3	3	4	4	4	5	6	4	>20%
UMEM	3977*	3792*	3977*	3978*	3978*	3978*	3978*	3978*	3953*	3978*	3978*	<3977
UPGS	4384*	4182*	4387*	4423*	4385*	4317*	4325*	4353*	4291*	4339*	4404*	>3978
DMRD	4	4	5	5	5	5	5	5	6	6	5	>40%
DMWR	4	4	5	4	5	5	5	5	6	5	5	>40%
DKRD	9	10	9	8	9	9	8	8	9	10	9	>40%
DKWR	6	5	6	5	5	6	5	6	6	6	6	>40%
TTYU	32	24	26	22	24	19	14	4	1	1	1	>40%
PTYU	32	38	54	58	59	63	64	81	89	87	84	>90%

* = EXCEEDS LONG TERM LIMIT > = GREATER THAN OR EQUAL TO < = LESS THAN OR EQUAL TO ----- CONTINUED NEXT PAGE -----

AMS SALES DEC-20
SYSTEM: SALE PRIME TIME: 0800 - 1700

PRIME TIME TREND OF KEY UTILIZATION ITEMS

ITEM	FIRST MONTHLY AVG	TABLE OF RELATIVE USAGE PER MONTH	LAST MONTHLY AVG	--11 MONTH-- TREND -- GROWTH LINE /MONTH FIT	PREDICTED RANGE OF VALUES WHERE AVERAGE WILL FALL IN 06 MONTHS 12 MONTHS	PREDICTED PERIOD WHEN LONG TERM LIMIT REACHED	LONG TERM LIMIT
CPU	68	[8 7 8 7 8 8 8 8 9 9 9]	80	+1.68 71%	80-100	BY 87OCT	>80
USED	48	[8 8 9 8 8 8 8 8 9 9 9]	52	ERRATIC VALUES			>80
SKED	11	[8 8 9 7 7 8 8 8 9 10 8]	11	ERRATIC VALUES			>13
BGND	7	[5 6 8 6 6 7 8 10 12 12 10]	14	+0.93 85%	17-26	BY 87JUL	>15
FILW	7	[10 10 8 10 10 8 8 8 7 6 7]	5	-0.29 70%	0-5	NONE LIKELY	>20
SWPW	2	[6 8 7 9 10 11 7 9 8 6 8]	3	ERRATIC VALUES			>5
TCOR	.2	[7 8 9 7 7 9 9 9 11 10 8]	.2	ERRATIC VALUES			>1.0
IDLE	25	[11 11 8 11 10 9 8 8 5 5 7]	15	-1.37 71%	0-14	BY 88MAR	<10
NRUN	2	[6 6 9 6 6 8 9 8 11 12 9]	4	ERRATIC VALUES			>30
NBAL	3	[6 7 9 6 6 9 9 8 11 12 9]	4	ERRATIC VALUES			>20
UMEM	3977	[8 8 8 8 8 8 8 8 8 8 8]	3978	ERRATIC VALUES			<3977
UPGS	4384	[8 8 8 8 8 8 8 8 8 8 8]	4404	ERRATIC VALUES			>3978
DMRD	4	[6 6 9 8 8 8 8 9 10 10 9]	5	ERRATIC VALUES			>40
DMWR	4	[6 6 9 8 8 8 8 9 10 9 9]	5	ERRATIC VALUES			>40
DKRD	9	[9 9 8 7 8 8 8 8 9 9 8]	9	ERRATIC VALUES			>40
DKWR	6	[9 8 8 7 8 8 8 8 9 9 9]	6	ERRATIC VALUES			>40
TTYU	32	[18 14 15 12 13 11 8 2 1 0 1]	1	-3.37 92%	0-0	NONE LIKELY	>40
PTYU	32	[4 5 7 7 8 8 8 10 11 11 11]	84	+5.67 92%	106-145	BY 87JUL	>90

NON-PRIME TIME TREND OF KEY UTILIZATION ITEMS

ITEM	FIRST MONTHLY AVG	TABLE OF RELATIVE USAGE PER MONTH	LAST MONTHLY AVG	--11 MONTH-- TREND -- GROWTH LINE /MONTH FIT	PREDICTED RANGE OF VALUES WHERE AVERAGE WILL FALL IN 06 MONTHS 12 MONTHS	PREDICTED PERIOD WHEN LONG TERM LIMIT REACHED	LONG TERM LIMIT
CPU	55	[10 9 9 7 7 8 8 9 10 8 8]	46	ERRATIC VALUES			>80
USED	44	[10 9 9 7 7 7 8 9 10 9 8]	39	ERRATIC VALUES			>80
SKED	9	[14 11 9 6 6 6 7 8 9 8 8]	5	ERRATIC VALUES			>13
BGND	2	[8 8 9 6 7 7 8 9 11 9 9]	3	ERRATIC VALUES			>15
FILW	13	[17 9 8 7 7 7 7 8 7 7 7]	6	ERRATIC VALUES			>20
SWPW	0	[2 8 1 1 1 53 1 1 8 1 1]	0	ERRATIC VALUES			>5
TCOR	.1	[10 10 11 6 6 6 10 11 12 10 10]	.1	ERRATIC VALUES			>1.0
IDLE	31	[6 7 8 10 10 9 9 8 7 8 8]	48	ERRATIC VALUES			<10
NRUN	1	[11 10 9 6 6 6 7 9 11 9 9]	1	ERRATIC VALUES			>30
NBAL	2	[11 10 9 6 6 6 8 9 11 9 8]	1	ERRATIC VALUES			>20
UMEM	3977	[8 8 8 8 8 8 8 8 8 8 8]	3978	ERRATIC VALUES			<3977
UPGS	2887	[9 8 9 8 8 8 8 8 9 8 9]	2922	ERRATIC VALUES			>3978
DMRD	0	[11 11 8 5 5 5 6 8 11 7 7]	0	ERRATIC VALUES			>40
DMWR	1	[11 10 9 6 6 6 6 8 9 10 8 9]	0	ERRATIC VALUES			>40

PREDICTIONS ARE ONLY MADE USING MONTHLY DATA. GIVEN THE CURRENT TREND, THERE IS A 90% CHANCE THAT THE ACTUAL VALUES WILL FALL WITHIN THE PREDICTED RANGES. UNCHANGING VALUES: REFERS TO RELATIVELY CONSTANT VALUES. ERRATIC VALUES: REFERS TO THE LACK OF A STRONG LINEAR PATTERN IN THE DATA. * = EXCEEDS LONG TERM LIMIT > = GREATER THAN OR EQUAL TO < = LESS THAN OR EQUAL TO ----- CONTINUED NEXT PAGE -----

GENERAL USAGE SUMMARY

	PRIME TIME	NON-PRIME TIME
HOURS THEORETICALLY AVAILABLE	45:00	123:00
HH:MM SYSTEM WAS UP	45:00	121:01
HH:MM AMAR MEASURED THE SYSTEM	45:00	119:47
DKRD	7	9
DKWR	5	3
DMRD	5	0
DMWR	4	0
SWFS	60%	90%

PRIME TIME PACK SUMMARY

PACK NAME	TOTAL TIME (HH:MM) MOUNTED - IN USE	% OF TIME MOUNTED - IN USE	% MOUNTED TIME IN USE	PACK LURD	PACK LUSK	PACK LUWQ	PACK LUFS_	PACK LUWR
CASS 0	4:41 - 1:20	10% - 3%	28%	0	0	.0	98%	0
CSD 0	44:39 - 19:25	99% - 43%	43%	0	0	.0	14%	0
D2 0	44:39 - 18:58	99% - 42%	42%	0	0	.0	8%	0
FS 0	44:39 - 24:54	99% - 55%	56%	1	0	.0	8%	0
PS 0	44:39 - 44:37	99% - 99%	100%	9	9	.0	2%	7
SA 0	44:39 - 38:50	99% - 86%	87%	2	1	.0	16%	1

UNIT NAME	TOTAL TIME (HH:MM) MOUNTED - IN USE	% OF TIME MOUNTED - IN USE	% MOUNTED TIME IN USE	UNIT PURD	UNIT PUSK	UNIT PUWQ	UNIT PUFS	UNIT PUWR
RP000	44:39 - 19:25	99% - 43%	43%	0	0	.0	14%	0
RP001	44:39 - 24:54	99% - 55%	56%	1	0	.0	8%	0
RP002	44:39 - 18:58	99% - 42%	42%	0	0	.0	8%	0
RP004	2:17 - 0:29	5% - 1%	21%	0	0	.0	96%	0
RP005	2:24 - 0:51	5% - 2%	35%	0	0	.0	96%	1
RP101	44:39 - 44:37	99% - 99%	100%	9	9	.0	2%	7
RP201	44:39 - 38:50	99% - 86%	87%	2	1	.0	16%	1

NON-PRIME TIME PACK SUMMARY

PACK NAME	TOTAL TIME (HH:MM) MOUNTED - IN USE	% OF TIME MOUNTED - IN USE	% MOUNTED TIME IN USE	PACK LURD	PACK LUSK	PACK LUWQ	PACK LUFS_	PACK LUWR
AMAR 0	1:21 - 0:41	1% - 1%	51%	1	1	.0	24%	1
CASS 0	5:21 - 0:05	4% - 0%	2%	0	0	.0	97%	0
CC 0	1:44 - 0:28	1% - 0%	25%	1	0	.0	18%	0
CHECKD0	0:26 - 0:26	0% - 0%	100%	15	5	.0	12%	1
CSD 0	118:20 - 9:16	99% - 8%	8%	0	0	.0	13%	0
D2 0	118:33 - 15:02	99% - 13%	13%	0	0	.0	8%	0
FS 0	118:13 - 24:44	99% - 21%	21%	2	1	.0	9%	0
PS 0	118:55 - 89:32	99% - 58%	58%	1	1	.0	2%	1
SA 0	118:45 - 40:32	99% - 34%	34%	4	1	.0	17%	2

----- CONTINUED NEXT PAGE -----

AMAR WORKLOAD REPORT

SITE: AMS SALES DECSYSTEM-20 SYSTEM: SALE
 REPORT DESCRIPTION: WEEKLY REPORT BY PROGRAM (PRIME SHIFT) - THREE MINUTE CUTOFF
 INPUT FILE: 87424 .DB0 (FISCAL YEAR: 87 QUARTER: 4 MONTH: 2 WEEK: 4 WEEKDAYS)

7:59 - 15:59 WEEKDAYS (-HOLIDAYS) FROM: MON 18-MAY-87 TO: FRI 22-MAY-87 (5 DAYS) MEASURED: 100% = 40.00 HOURS

GROUPED BY: PNAM		SORTED BY: CPU%		CUTOFF: 0.63% OF CPU		ACCOUNT		PRGRM		PAGES		CPU%		SWAP		FILE		IFA		RSP		SEC		SR B		TTY		LOGIN AT			
JOB #	FRK #	AVG	MEM	IN	DEMO	USER	NAME	NAME	NAME	(WS)				PF/S	PF/S				/MIN	/RSP								DAY	TIME		
88.8	307.3	105.0	2.80	*****	INTERVAL	TOTALS	*****	46.8	43.15	4.1	6.2	59	1145.5	0.1	4																
43.4	138.8	41.0	0.75	????????		????????		EXEC	45.2	14.06	1.0	2.0	64	181.1	0.2	4															
12.0	46.6	18.2	0.67	????????		??-??-??		EMACS	46.6	9.91	0.8	0.9	89	397.2	0.1	4	T														
15.1	41.1	16.5	0.19	????		??-??-??		MM	33.3	2.68	0.5	0.4	44	60.5	0.2	4															
1.2	4.0	2.0	0.13	???		??-??-??		NCPCAL	91.6	2.22	0.1	0.1	115	72.7	0.1	3	T														
1.0	18.2	8.0	0.22	OPERATOR		OPERATOR		SYSJOB	22.9	2.01	0.7	0.4	30	261.7	0.0	3	T	DET													
1.0	7.0	2.0	0.10	OPERATOR		OPERATOR		MMAILR	34.7	1.35	0.2	0.3	40	14.5	0.4	5	T														
0.0	0.2	0.2	0.05	???		??-??-??		RUN	84.1	1.20	0.0	0.2	62	1.5	0.4	4															
0.2	0.4	0.3	0.03	???		??-??-??		PRJRPT	107.0	0.72	0.0	0.1	129	1.8	0.2	4	T														
1.0	7.0	1.3	0.03	OPERATOR		OPERATOR		SYSJOB	26.0	0.71	0.1	0.1	65	15.7	0.1	2	T														
0.0	0.2	0.1	0.04	???		SFS-MAINT		FNDREC	38.0	0.68	0.0	0.3	24	0.1	1.6	24	T	371													
75.7	263.5	89.5	2.22	*****	SUBTOTALS	THRU CUTOFF	*****	45.8	35.51	3.5	4.8	61	986.9	0.1	4																
13.2	43.8	15.5	0.57	*****	SUBTOTALS	AFTER CUTOFF	*****	49.7	7.64	0.7	1.4	51	158.6	0.1	6																

Planning and Implementing a Large Network

Leslie Maltz

Stevens Institute of Technology

ABSTRACT

Planning and implementing a large corporate or campus-wide network is a complex task. should be taken into consideration during the planning process. Specific examples are cited from the campus-wide network that is being implemented at Stevens Institute of Technology. nodes ranging from microcomputer workstations through highend systems from a variety of vendors. costs, support challenges, organizational structure, and budgetary implications.

SESSION SUMMARY:

WHAT IS A NETWORK?

Networks were:

- Small (number of nodes)
- Timesharing systems
- Point to point connections
- Homogeneous systems
- Managed facilities

WHY NETWORK?

This is a key questions that should be answered at the onset of the planning process. The answer depends on the individual institution. Answers range from cost saving for communications to competitiveness with other organizations, with many other possible answers.

CONCERNS:

The following is a partial list of concerns that should be considered as part of the planning process:

- Capabilities
- Installability
- Ease of use
- Performance
- Reliability
- Maintainability
- Compatibility
- Evolvability
- Growth potential
- Cost
- Timeliness
- Institutional goals
- Bandwidth
- Security
- Functionality
- Management
- Extensibility

NEEDS ASSESSMENT:

Much work should be done to answer basic questions associated with a needs assessment survey. Some of these questions cover establishing the reasons for networking, determining what forms of networking already exists, the development of a master plan for computing resources, estimating usage and patterns, identification of communities of interest when possible, assessing existing computing resources, areas of risk, security requirements, etc.

NETWORK FUNCTIONALITY:

Below is a possible list of network functionality. Each institution should determine its own proposed list.

- Task-to-task communications
- Resource sharing
- File transfer
- Electronic mail
- Gateways
- Specialized resources
- Standards
- Support for a multi-vendor environment
- Estimated maximum number of nodes
- Acceptable level of security

...

SITE SURVEY:

Related to the needs assessment is the need for accurate information on the building and grounds that will be wired. This survey should include the identification of the number and location of proposed connections; collection of accurate blueprints; determination of regulations, codes, and zoning rules; identification of restrictions; inter/intra-building status; identification of existing wiring; geography of area to be networked, ...

POPULATION:

Who are the current and potential users of the proposed network? What are their work patterns; what are their mobility patterns? Are there identifiable groups or communities of interest? How much can be said about the hours of usage, resources needed/used, growth potential, needs of any kind?

TOPOLOGY AND TECHNOLOGY:

Once all the surveying and assessments have been completed, it soon becomes apparent that compromise is necessary. It will likely be necessary to weigh tradeoffs in performance versus costs; in optimal versus affordable solutions.

PROTOCOLS:

Networks are usually capable of supporting multiple protocols. The limitation is often the number than can be supported by the network support staff as opposed to a maximum number. Some protocols being supported at sites with large quantities of computing resources are DECnet, TCP/IP, XNS, SNA, and RSCS. The specific protocols that should be supported at an institution depend on the answers to the questions asked as part of the site survey, needs assessment, and concerns, and functionality surveys.

PROTOTYPES:

Whenever feasible, the development of and experimentation with prototypes can be extremely valuable, and may result in considerable savings in the long run. Prototypes may include sub-networks and clusters of systems or workstations, file servers, projections systems, print/plot servers, mail servers, and more. The specific implementations depend on the scope of the plan, funds available for prototyping, timing, and prior experience.

MULTI-VENDOR ENVIRONMENT:

Many institutions are committed to supporting a multi-vendor environment. At Stevens the list of vendors includes DEC, IBM, H-P, AT&T, SUN and many others. Decisions regarding the design and implementation of the campus-wide network are based on the requirement of the support of a multi-vendor environment.

GATEWAYS:

With a growing need for communication and connectivity with individuals and resources outside our own institutions, it is important to provide access to external networks and facilities. This is

often accomplished via gateways. Some of the external networks include BITNET, NSFNET, ARPANET, CSNET, and CCNET.

STANDARDS:

Standards are the one thing that makes it possible to design and implement large networks. The recognition of established external standards such as those of ANSI, IEEE, and OSI is essential. There is also a need to establish internal standards to facilitate the design, implementation, and growth. The establishment/adherence to standards will permit controlled evolution and growth, multi-vendor support, and ease of support.

SUPPORT:

While many possible permutations of support organizations may exist and function well, the distribution of a variety of resources on a network can often be supported by a centralized support organization. This can be quite successful as well as economical. An alternative would be distributed support.

SERVERS:

A full function network includes a variety of servers. Some of the possible servers are file, compute, communications, print, plot, name, and time.

MANAGEMENT:

Networks are not only a complicated resource to plan; they require an ongoing effort to manage. This function includes the following management activities:

- Configuration
- Fault isolation
- Security
- Accounting
- Performance
- Documentation
- User assistance/training

INFORMATION DISTRIBUTION:

It is important to keep all members of the institution well informed on all aspects of the design, implementation, and support processes. Much support for the effort is gained through adequate information flow. The lack of such information can result in misinformation and resentment.

LICENSING:

With the implementation of large networks, it is becoming increasingly important to have realistic,

consistent, and affordable licensing policies. Many vendors do not currently have such policies. One consequence is the proliferation of unlicensed software.

CONCLUSION:

Proper planning before beginning implementation is a must. Many potential problems can be avoided through the development of a detailed plan. This phase of the effort should not be minimized. Lacking of planning can result in limitations in functionality and growth, and costs can be substantially higher.

**High End
VAX system
Update**

**Warren Sander
High Performance Systems**

This session's objectives are:

- **Review High End VAX processor announcements for the last six months**
- **Review VAX Software changes in the last six months**

Current High End Processors

- **VAX 8500/VAX 8530**
- **VAX 8600/VAX 8650**
- **VAX 8550/VAX 8700**
- **VAX 8800**
- **VAX 8974/VAX 8978**

VAX 8530

- **Announced March 5th**
- **VAX 8530 replaces VAX 8500**
- **30% to 50% faster**
- **4 to 4.5 times the VAX 8200**
- **Backplane supports up to 80MB memory**
- **VMS Version 4.5 / Ultrix Version 2.0**

VAX 8600 and VAX 8650

- **MS86-DA New 64MB memory boards**
- **Mother/daughter boards**
- **Up to 260MB main memory**
 - **4 MS86-DA**
 - **1 MS86-AA**
- **VMS 4.5/Ultrix 2.0 support 128MB**
- **Future VMS supports full 260MB**

VAX 8974

The VAX 8974 is complete system including hardware, software, and support services providing up to 25 times the performance of the VAX 8200.

VAX 8974 Features and Benefits

- **High Availability Configuration**
- **Mainframe Class I/O Subsystem**
- **Enhanced system management features**
- **Easy incremental growth**
- **Significant price/performance and cost of ownership features**

VAX 8974 Hardware

- 4 VAX 8700 central processors
Up to 512 MB memory
Up to 16 VAXBI channels
- 2 HSC70 I/O processors
12 disk I/O channels
2 tape I/O channels
- Dual Ported SA482 storage array
- Full CI and Ethernet support
- VAXcluster Console System

VAX 8974 Software

- VMS
- DECnet
- VAX Volume Shadowing
- VAX Performance Advisor
- All VAX/VMS software products available at significant savings

VAX 8974 Services

- Pre-installation site evaluation
- Customer Support Plan
- Hardware and Software Installation
- DECstart, Media and Documentation
- 1 year resident Systems Engineer
- 1 year hardware warranty
- 1 year software support

VAX 8978

The VAX 8978 is a complete system including hardware, software, and support services providing up to 50 times the performance of the VAX 8200.

VAX 8978 Features and Benefits

- High Availability Configuration
- Mainframe Class I/O Subsystem
- Enhanced system management features
- Easy incremental growth
- Significant price/performance and cost of ownership features

VAX 8978 Hardware

- 8 VAX 8700 central processors
Up to 1024 MB memory
Up to 32 VAXBI channels
- 4 HSC70 I/O processors
24 disk I/O channels
4 tape I/O channels
- Two dual ported SA482 storage array
- Full CI and Ethernet support
- VAX Cluster Console System

VAX 8978 Software

- VMS
- DECnet
- VAX Volume Shadowing
- VAX Performance Advisor
- All VAX/VMS software products available at significant savings

VAX 8978 Services

- Pre-installation site evaluation
- Customer Support Plan
- Hardware and Software Installation
- DECstart, Media and Documentation
- 1 year resident Systems Engineer
- 1 year hardware warranty
- 1 year software support

SA482

The SA482 Storage Array is Digital's newest high capacity, high performance, online storage solution.

SA482

- 2.5 GB in 5.5 square feet
- Connects to HSC50, HSC70, UDA50, KDB50, KDA50
- Four independent spindles
- One-year warranty
- Lower cost per Megabyte
- Reduced maintenance fees

SA482 Performance

- Peak Transfer Rate
2.4 MB/sec/spindle
- Average Seektime
24 milliseconds/spindle
- Rotational latency
8.3 milliseconds/spindle

- Average access time
32.3 milliseconds/spindle
VMS Update
VMS Version 4.4

- Faster VAXcluster state transitions
 - . Requires new CI port driver
 - . Requires CI microcode, Revision 7.0
- Cluster node name via ALIAS

- VAXcluster support for:
 - . VAX 8200/VAX 8300
 - . VAX 8500/VAX 8550
 - . VAX 8700/VAX 8800

- HSC70 Support

VMS Version 4.4 Networks

- Cluster node name via ALIAS
- NETNODE.DAT broken into LOCAL and REMOTE database files
- NCP parameter TRANSMIT PIPELINE with DMR11s provides more efficient use of satellite links
- Multiple network support in VAX PSI configurations
- Sysgen LRPSIZE set to 1504 instead of 576 if Ethernet is present

VMS Version 4.4 MONITOR Features

- MONTIOR remote nodes within a VAXcluster
- New CLUSTER class
- Record and playback multiple nodes in a single file
- New output format for the recording file

VMS Version 4.4
SECURITY Enhancements

- New DYNAMIC attribute
- SET RIGHTS_LIST
- New System Services:
\$GETUAI
\$SETUAI
\$CHECK_ACCESS
- Node specific identifier, created at boot time

SYS\$NODE_nodename

- . Limit access to layered products
 - . Restrict users in a non-homogeneous VAXcluster
- US Department of Defense C2 security certification

VMS Version 4.4
HSC Based Volume Shadowing

- Two Compatible Disks are logically identical
- Provides High Data availability
- Transparent for FILES-11 and ODS-2 volumes

VMS Version 4.4
RMS Enhancements

- Support for decending keys
- Fully shared sequential files
- RMS performance improvements like adaptive locking
- File specifications may now include a hyphen ("-")

VMS Version 4.4
Miscellaneous Changes

- DCL commands for subroutine support

- Debugger enhancements
 - . Vertical windows
 - . New keypad definitions
 - . Support for VAX DIBOL & VAX SCAN
 - . New SHOW STACK command
 - . Debugging permitted for shareable images
- System Dump Analyzer
 - . ATTACH and SPAWN commands
 - . New EVALUATE, EXAMINE and SEARCH commands

VMS Update

VMS 4.5 is a maintenance release consisting of 105 separate updates.

VMS 4.4 must be installed before a system can be updated to VMS 4.5

VMS 4.5 can be installed in a VAXcluster as either a rolling or concurrent update but VMS 4.5 cannot coexist in a cluster with VMS 4.3.

VMS 4.5
CI Port Driver

The new version of the CI Port Driver Image (Version 7.0) fixes the following problems:

- Miscellaneous Error #5, Internal Queue Retry Expired
- Arbitration Timeout
- Buffer Length Violation

You can identify which version of the microcode you are running as follows:

```
$ SHOW CLUSTER/CONTINUOUS  
COMMAND> ADD RP_REVIS
```

VAX 8200/8250
VAX 8300/8350
VAX 8530/8550
VAX 8700/8800

The low-order word is the RAM version and the high-order word is the PROM version.

VMS 4.5 Permanent MONITOR Server Process

Creating a permanent MONITOR server process on each member in a cluster at bootstrap can significantly reduce the startup time for MONITOR/CLUSTER commands

To create a detached server process, add the following lines in SYSS\$MANAGER:STARTUP.COM:

```
$ DEFINE /SYSTEM / EXECUTIVE_MODE VPM$SERVER_LIVE TRUE  
$ RUN /DETACH /PAGE_FILE=10000 SYSS$SYSTEM:VPM.EXE
```

VMS 4.5 DMB32 Layered Product Support

VAX 8200/8300/8500/8550/8700/8800 systems that include DMB32 communications processors must install the DMB32 layered product in order to use the SYNCHRONOUS port.

This software is not included in the VMS Update Kit

ULTRIX

ULTRIX Version 2.0 now supports all VAXBI based systems:

with KDB50 connections to disks.

VAX Performance Advisor

The VAX Performance Advisor (VPA) is a ruled based performance analysis tool that runs as a layered product on VAX/VMS.

VPA gathers data from all nodes on a VAXcluster and identifies and reports possible performance problems which it substantiates with its data and recommends solutions to the problems.

VMS Update

VMS 4.6 is focused on Maintenance but the kit has been remastered so it is a complete distribution

- Scheduled to submit to SDC in June
- FCS starting in July (for the US)

VMS Version 4.6 Major New Features

- VAX Volume Shadowing: 3 Member Shadow Sets
- Local Area VAXclusters: Up to 26 Satellite Members.
- Full 260MB Memory Support for VAX 8600 & VAX 8650
- Processor support for VAX 8250, VAX 8350, & VAX 8530

VMS Version 4.6 Other New Features

- SET TIME/CLUSTER Command
- AUTOGEN Enhancements
- LAT/VMS New Features
- Limited Support for Dual-ported HSC Tapes
- NCP SHOW CIRCUIT Command Changes
- New Debugger Features

SET TIME/[no]CLUSTER

The SET TIME/[no]CLUSTER command will update the time on ALL nodes present in the VAXcluster to the time specified or to the time on the node the command is executed on if no time is specified

Example:

```
$ SYNC_CLOCKS:
$   SET TIME /CLUSTER
$   WAIT 6:00:00
$   GOTO SYNC_CLOCKS
```

This procedure sets the time on all the cluster nodes to the value obtained from the local time-of-year clock, waits, then resets the time again.

AUTOGEN Enhancements

A user specified startup file can now be defined in place of SYSTARTUP by using the symbol STARTUP in MODPARAMS.DAT

Example:

```
STARTUP = "SYSS$MANAGER:MY_STARTUP.COM"
```

AUTOGEN will now calculate a value for QUORUM using either the current value or the initial cluster quorum.

AUTOGEN now understands and manipulates secondary page and swap files in MODPARAMS.DAT

Example:

```
SWAPFILE2_NAME = "SWAP$DISK:[SWAPFILE]SWAPFILE.SYS"
SWAPFILE2_SIZE = 30000
```

You still need to use SYSGEN to install the secondary files in SYSTARTUP.COM but the files will be created if they don't exist

LAT/VMS Features

VMS Version 4.6 includes new Local Area Transport (LAT) software which includes support for asynchronous printers connect to LAT terminal servers. The support consists of a new LAT port driver, LAT control program and LAT print symbiont.

VAX/VMS Software Announcements New Products

VPA - VAX Performance Advisor
VAX Data Distributor
VAX SQL - Structured Query Language
SSU - Session Support Utility
VAX Software Project Manager

VAX/VMS Software Announcements New Versions

VAX ACMS V2.1	VAX DBMS V3.2
VAX TDMS V1.7	VAX Rdb/VMS V2.2
VAX INFO V1.2	VAX Datatrieve V4.0
VAX DECalc V3.0	VAX DECalc-PLUS V3.0
VAX RALLY V1.1	VAX TEAMDATA V1.1
VAX DECReporter V2.0	VAX SCAN V1.1
VAX VTX V3.0	ALL-IN-1 V2.2
VAX COBOL V3.4	VAX COBOL Generator V1.1

VAX Performance Advisor Version 1.0

- VPA analyzes system workload data and makes recommendations on how to improve performance.
- Support for both a single VAX processor and VAXcluster systems is provided.
- Analysis of data can be performed from any VAX processor in a VAXcluster system.
- VPA identifies system bottlenecks as well as processes that may be using inordinate amounts of system resources.
- User can request data to support recommendations made by VPA.
- Histograms of CPU utilization, physical memory usage, disk I/O and terminal I/O for each node in a VAXcluster system are available.
- User can define collection intervals for automatic collection of performance data

VAX Data Distributor Version 1.0

- Centralized Storage of Definitions, Schedules, and Status Information
- VAX Data Distributor Syntax
 - Define transfers, specifying the locations of source and target databases
 - Select records and fields to extract and replicate
 - Transfer data automatically through user-defined schedules or execute the transfer on demand
 - Show transfer and schedule definitions and status
 - Perform automatic retry if network failures occur
 - Enable logging to record transfer events if desired
- Database Security
 - Transfer Database
 - Source Database
 - Target Database

VAX Data Distributor Version 1.0 requires:

- Rdb/VMS V2.2 or later
- MicroVMS or VMS Version 4.4 or later
- DECnet (if Data Distributor will be run on multiple nodes)

VAX SQL Version 1.0

- VAX SQL is layered on DSRI.
- VAX SQL is designed for compatibility with other SQL products.
- VAX SQL may be used with remote as well as local databases.

- VAX SQL includes an interactive DML and DDL utilities
- Includes language preprocessor support for VAX COBOL, VAX FORTRAN, and VAX PL/I.
- Dynamic SQL can accept or generate SQL statements at run time
- VAX SQL can read or write metadata from the CDD

VAX Software Project Manager Version 1.0

- Graphical, Multi-user, software development project management tool
- Fully integrated planning, controlling and estimating functions
- Includes CPM, WBS, PERT, GANTT and precedences and Estimation based on Boehm's industry-standard COCOMO model

VT330 and VT340 Session Support Utility Version 1.0

- Allows a VT330/VT340 to operate two session over one wire
- SSU runs under VMS 4.4 or later
- SSU runs on and valid VAX/VMS host configuration

VAX ACMS Version 2.1

- Improved performance
- VAX Language-Sensitive Editor (LSE) support
- Supports ACL's
- Major New Documentation
- Major documentation changes, including:
 - "Design for Performance" chapter in ACMS Design Guide
 - Error Messages now available via DCL and ACMS HELP

VAX DBMS Version 3.2

- Doubles Performance in some cases
- Adding AREAS without unloading/reloading
- FIND/FETCH/STORE by Database Key (DBKEY)
- UNIQUE clause
- Enhanced statistics package

VAX Datatrieve Version 4.0

- Support for the VAX Language-Sensitive Editor (VAX LSE)
- A new OPTIMIZE qualifier
- New sample domain and record definitions
- New VAX DATATRIEVE PLOT commands
- New arguments to specify object types for EXTRACT/EDIT
- A SHOW FIELDS enhancement for RMS sources
- A logical name to define default READY access
- Several improvements to the DAB.PAS file
- A new function called FN\$DCL
- A logical name to set stack size
- New arguments for the DEFINEP command

VAX TDMS Version 1.7

- VAX Language-Sensitive Editor support
- DEFINE KEY capability with full LK201 support
- Improved performance for USE FORM
- Enhancements to the RETURN instruction
- SPAWN and ATTACH commands from RDU and FDU
- Enhancement to the OUTPUT instruction

VAX Rdb/VMS Version 2.2

- Reduced I/O for small update transactions
- New precompilers -- VAX C and VAX Pascal
- Support for VAX Data Distributor
- Support for VAX SQL
- Miscellaneous features
 - True VAXcluster wide database shutdown
 - Additional RDO and precompiler support for DSRI

VAX RALLY Version 1.1

- Improvements to Documentation and Sample Applications
- Brief Application Report Option
- Improved Performance with AFILE Creation

VAXinfo I, VAXinfo II, & VAXinfo III Version 1.2

New version of VAXinfo including the newest versions of all products:

<u>VAXinfo I V1.2</u> (Q*740)	<u>VAXinfo II V1.2</u> (Q*AB1)	<u>VAXinfo III V1.2</u> (Q*738)
----------------------------------	-----------------------------------	------------------------------------

CDD V3.3	CDD V3.3	CDD V3.3
Datatrieve V4.0	Datatrieve V4.0	Datatrieve V4.0
TDMS V1.7	TDMS V1.7	TDMS V1.7
Rdb/VMS V2.2	Rdb/VMS V2.2	DBMS V3.2
	ACMS V2.1	ACMS V2.1

VAX DECalc Version 3.0

- Enhanced performance and reliability.
- Enlarged grid size -- to 9999 rows by 702 columns (A1 to ZZ9999).
- New name format -- grids are stored as VMS filenames.
- Nested IF/THEN/ELSE function and logical operators AND, OR and NOT
- Consolidation of multiple grids
- Sorting -- "Block" sorting similar to Lotus 1-2-3.
- Moving -- "Block" moving similar to Lotus 1-2-3.
- "\$" absolute and relative cell referencing similar to Lotus 1-2-3

VAX DECalc-PLUS Version 3.0

- Interactive debugger support for external routines
- More performance and reliability dealing with external routine templates
- Larger grid size -- 9999 rows by 702 columns (A1 to ZZ9999)
- Consolidation of multiple grids
- Sorting -- block sorting similar to Lotus 1-2-3
- Moving - block move similar to 1-2-3
- Spread sheets stored as VMS files
- "\$" absolute and relative cell referencing similar to Lotus 1-2-3
- Nested IF/THEN/ELSE function and logical operators AND, OR, and NOT

VAX TEAMDATA Version 1.1

- VAX Xway Support
- Improved Performance - memory use reduced by 20%
- Improvements and Additions to Documentation

VAX DECreporter Version 2.0

- Multiline titles (up to 4 lines of text in each title)
- Ability to specify subtotals, average, maximum or minimum statistics
- Auto formatting (allows the user to automatically set column positioning)
- User selectable 80 or 132 column report width
- Print queue menu selection allows the user to specify an output printer
- Parenthetical computation expressions
- Boolean selection criteria

VAX SCAN Version 1.1

- VAX SCAN V1.1 is a maintenance release that adds support for newly announced processors

VAX VTX Version 3.0

- New VISTA (VTX Infobase Structure Tool and Assister) enables IP's to build and maintain infobases more effectively in an interactive environment
- New ALL-IN-1 Terminal Control Program (TCP)
- WPS-PLUS to VTX Print Queue
- Up to 32,767 continuation pages
- Context-Sensitive Keywords
- Template pages
- Support for 132-column mode
- Performance Enhancements
- Support for additional terminals

ALL-IN-1 Version 2.2

- Includes WPS-PLUS, VAX FMS V2.3 and Message Router VMsmail Gateway V2.1
- Same performance as ALL-IN-1 V2.1
- VAX VTX 3.0 can be fully integrated

VAX COBOL Version 3.4

- Fixes an error found in the V3.3 ANSI validation.
- Validated and a high level with NO errors against 1974 ANSI and FIPS PUB 31-1 (amended 21-2)
- Current ANSI 1985 features:
 - Contained Programs
 - Scope delimiters
 - EVALUATE verb
 - In-line PERFORM
 - REFERENCE MODIFICATION statement

VAX COBOL Generator Version 1.1

- Can use RMS files or Rdb/VMS databases
- Print files from within the generator
- GENERATE DOCUMENT to produce documentation to support the application design



A Practical Exercise In System Sizing

Warren Sander
Daniel Allen Deufel

High Performance Systems

Objectives Of This Session

- Understand the Basics of System Sizing
- For an arbitrary case, design a system that meets the functional objectives
- Be able to select relevant data for base system sizing

The Convention Center System

A Practical Description

The Convention Center System is intended to supply a number of computing services to a major convention and conference center. The system will provide services to the three major groups:

- The Convention Center Management
- The Convention Promoter
- The Convention Attendee

Goals

- Provide Convention Promoter facilities to maintain and manage show information
- Provide the Convention Center with automated planning services
- Provide show visitors with (perceived) instantaneous response to requests and queries

- Provide a redundant service environment

Convention Center Management Requirements

The Convention Center management needs tools to help it plan the various events, exhibits, sessions, and logistics that are associated with a conference or convention. In addition, the Conference Center will provide visitor reservation and hotel reservation services to their clients.

Convention Promoter Services

The Convention Center will be offering a series of services to the promoters using the facilities. When the promoter schedules a show, a number of tools are made available to enable him to begin scheduling facilities, organizing display areas, etc. As the show date approaches, the Convention Center System will be used to pre-register visitors.

During the show the Convention Center System will be used to register arriving visitors, provide Telephone Message Services, handle visitor Literature Requests. Staffed Information Desks will be located around the Convention Center to handle visitor queries.

Finally, the system will give the promoters tools to oversee their show as it runs its course.

Convention Attendee Services

During a show, the Convention Center System will provide visitors with a number of services. These services include an Electronic Mail system, on-line convention news, a literature request system, and information about local attractions and places to eat. These services may be obtained at 200 terminals located in the Convention Center and the major hotels in the area.

Convention System Functional Description

The CCS (Convention Center System) allow the Convention Center Staff to provide all Convention Promoters with an integrated system which can plan all phases of a convention. The CCS will also provide the Promoter with a full range of convention services for not only the convention staff and exhibitors but for the attending population also. CCS integrates all convention related functions into a single menu driven environment that is easy to use for both experienced and inexperienced promoters and exhibitors.

Convention System Applications

The following applications are proposed:

- Exhibit, Session and Event Planning System - (PLAN)
- Visitor Registration and Logistics System - (VRLS)
- VIP and Facilities Scheduling - (VIPS)
- Information Desk Support - (INFO)
- Telephone Message Services - (TELS)
- Literature Order/Request and Fullfilment - (LITO)
- Information System for Population at Large - (PALS)
- Promoter Administration Services - (PADS)

Subsystem Planning

For this exercise we need to examine the demands each of the applications will make on the following pieces of our system:

- CPU Capacity
- Memory Capacity
- Channel I/O Capacity
- Spindle I/O Capacity
- Storage Capacity

Key User Groups

Application	Users	# (estimated)	Totals
PLAN	- Exhibit, Session and Event Managers	5	35
	- Content Specifiers/Planners	30	
VRLS	- Pre-registration Data Entry Operators	8	74
	- Event Check-in Supervisors and Managers	6	
	- Event Check-in Operators	50	
	- Convention Process Managers (Security,logistics,event sizing etc.)	10	
VIPS	- VIP's	20	40
	- VIP Secretaries	15	
	- Facilities Coordinators and Managers	5	
LITO	- Literature Abstract Writers	2	2
INFO	- Information Desk Supervisors and Managers	5	25
	- Information Desk Operators	20	
TELS	- Telephone Operators	5	10
	- Message Center Operators	5	
PALS	- PAL "Hostesses"	20	50000
	- Convention Visitors	50000	
PADS	- Key Promoter Management	30	30

Visitor Registration and Logistics System

This application establishes and maintains a database of all key visitor data which is required not only to assure:

- Event Access
- Visitor Transportation and Lodging Logistics
- Dynamic Event Programming and Operation

but also serves as the Information Support Tool/Source for:

- Promoter Administration (Statistics, Reports and Inquiries)
- Lead Recording and Reporting
- Information Desks
- Telephone Message Handling
- Staff Assignment and Deployment

VRLS Work Load

<u>MAX Operations Per Hour</u>	<u>CPU Per Operation</u>	<u>I/O's Per Operation</u>
1,200	4.0	32

	<u>CPU Seconds Per Hour</u>	<u>I/O's Per Hour</u>
Totals	4,800	38,400

Notes: "CPU" units are the CPU seconds required to process one operation on a VAX-11/780

"I/O" units are the number of I/O requests generated during the processing of one operation

VIP and Facilities Scheduling System

This application provides scheduling of Individuals and Facilities (Conference rooms, Guided Tours, etc.) and maintains Master Files/Listings for each. It assures orderly and smooth execution of key appointments and events in a "hustle-bustle" environment. "VIPS" is linked with additional applications (see "PADS") and provides the Customer VIP with a "total" Office Information System while at the conference.

VIPS Work Load

<u>MAX Operations Per Hour</u>	<u>CPU Per Operation</u>	<u>I/O's Per Operation</u>
150	1.5	10

	<u>CPU Seconds Per Hour</u>	<u>I/O's Per Hour</u>
Totals	225	1,500

Notes: "CPU" units are the CPU seconds required to process one operation on a VAX-11/780

"I/O" units are the number of I/O requests generated during the processing of one operation

Information Desk Support System

This application provides Information Support to dedicated accounts at fixed locations. It is a menu sub-set of several Convention Community Services applications such as, Visitor name-, Company name-, Hotel- look-up, Event- and Session-schedules, Exhibit Content etc.

INFO Work Load

<u>MAX Operations Per Hour</u>	<u>CPU Per Operation</u>	<u>I/O's Per Operation</u>
600	2.8	24

	<u>CPU Seconds Per Hour</u>	<u>I/O's Per Hour</u>
Totals	1,680	14,400

Telephone Message Services System

This application provides the means of transmitting an incoming telephone message which arrives at a (ideally) central location to the intended recipient (Visitor or Staff). Process logic assists the receiving operator in message screening and diversion according to message type (emergency, general info., etc.) and timeliness for posting to Electronic Mail System (see PALS), Hotel, Message Board or Security.

TELS Work Load

<u>MAX Operations Per Hour</u>	<u>CPU Per Operation</u>	<u>I/O's Per Operation</u>
600	6.0	37

	<u>CPU Seconds Per Hour</u>	<u>I/O's Per Hour</u>
Totals	3,600	22,200

Information System for Population at Large

This application is a multifunctional Information System available to any Convention Visitor composed of:

- **Electronic Mail** A Visitor may send mail to any other Visitor.
- **Visitor Inquiry** A Visitor may inquire on the presence of another Visitor.
- **Convention News** Worldwide News, Convention-, Company-, and City- Information with items updated continuously during the course of the event.
- **Literature Services** A free-text search facility in the content of the convention or sponsoring organization related literature and, when found, a choice of ordering it on-line for mailing to Visitor's address.
- **Information Services** Free-text search into convention content as related to exhibit, demos, vendors, etc. Alphabetic listings of products and vendors. Graphic display of exhibit area ("You are here, and this is where you find your item of interest", etc.)

PALS Work Load Breakdown

<u>PALS Task</u>	<u>MAX Operations Per Hour</u>	<u>CPU Per Operation</u>	<u>I/O's Per Operation</u>
E-Mail	100	8.0	100
Visitor Inquiry	100	1.7	6.8
Lit. Services	100	3.2	15.0
Convention News	300	0.90	4.2
Info Services	800	0.90	4.2

PALS Work Load

<u>PALS Task</u>	<u>CPU Seconds Per Hour</u>	<u>I/O's Per Hour</u>
E-Mail	800	10,000
Visitor Inquiry	170	680
Lit. Services	320	1,500
Conv. News	270	1,260
Info Services	720	3,360
Totals	2,280	16,800

Applications Work Load Profile

<u>Task</u>	<u>CPU Seconds Per Hour</u>	<u>I/O's Per Hour</u>
VRLS	4,800	38,400
VIPS	225	1,500
INFO	1,680	14,400
TELS	3,600	22,200
PALS	2,280	16,800
Totals	12,585	93,300

CPU Requirements

Base Daily CPU Requirements:

- Given: Demand = 12,585 CPU (11/780) Seconds per Hour
= 3.5 * VAX-11/780s
- Given: VAX 8550 ~ 5 * VAX-11/780
- Given: A Clustered VAX 8550 ~ 0.8 * VAX 8550
~ 4 * VAX-11/780
- Therefore, 1 VAX 8550 should be able to handle the peak applications demands (not including preregistration and staging of future shows).

Memory Requirements

For ALL-IN-1 based applications plan on the following:

- The 1st thru 4th users require 5MB of memory each
- The 5th thru Nth users require 0.5MB of memory each
- Therefore, 124 Users require:

$$(5MB * 4) + (120 * 0.5MB) = 80MB \text{ of Main Memory}$$

Simultaneous System Users

<u>Application</u>	<u>Users</u>	<u># (estimated)</u>	<u>Totals</u>
PLAN	- Content Specifiers/Planners	20	20
VRLS	- Pre-registration Data Entry Operators	8	
	- Event Check-in Supervisors and Managers	4	
	- Event Check-in Operators	50 *	
	- Convention Process Managers (Security,logistics,event sizing etc.)	4 *	66
VIPS	- VIP's	5 *	
	- VIP Secretaries	15 *	
	- Facilities Coordinators and Managers	5	25
LITO	- Literature Abstract Writers	2	2
INFO	- Information Desk Supervisors and Managers	5	
	- Information Desk Operators	20 *	25
TELS	- Telephone Operators	5 *	
	- Message Center Operators	5 *	10
PALS	- PAL "Hostesses"	20	
	- Convention Visitors	140 *	160
PADS	- Key Promoter Management	30	30
			30
		Total	338 **

* - These are the 254 users involved with the current show

** - This is the projected Worst Case condition.

Channel I/O Requirements

- The CSS applications generate 93,300 I/O Requests per hour or 26 I/O Requests per second
- Each HSC5X-BA can handle roughly 70 I/O Requests per second
- Each HSC70 can handle roughly 600 I/O Requests per second

Spindle I/O Requirements

- The CSS applications generate 93,300 I/O Requests per hour or 26 I/O Requests per second
- One SA482 spindle can handle roughly 35 I/O Requests per second

Storage Requirements

System:

- 1 Spindle for System
plus 1 Shadow Spindle
plus 1 Spindle for Page/Swap for each CPU

Live Show:

- 50KB Per Visitor
- Maximum of 50,000 Visitors per show
- 50KB * 50,000 = 2.5GB = 1 SA482
plus 1 SA482 for Shadowing

Show Staging:

- Must support 6 shows in staging
1 Spindle per staged show
plus an equal number of shadow spindles

Terminal Requirements

Terminals will be located in the following locations:

<u>Location</u>	<u>Number of Terminals</u>
<u>Convention Center</u>	
Preregistration	8
Registration	50
VIP Center	20
Information Stations	20
Message Center	5
Telephone Center	5
Promoter's Center	20
Convention Center Floor	100
Dial Up	24
<u>Hotels</u>	
8 terminals at each of 12 Hotels	96
Total	348

Hardware Order

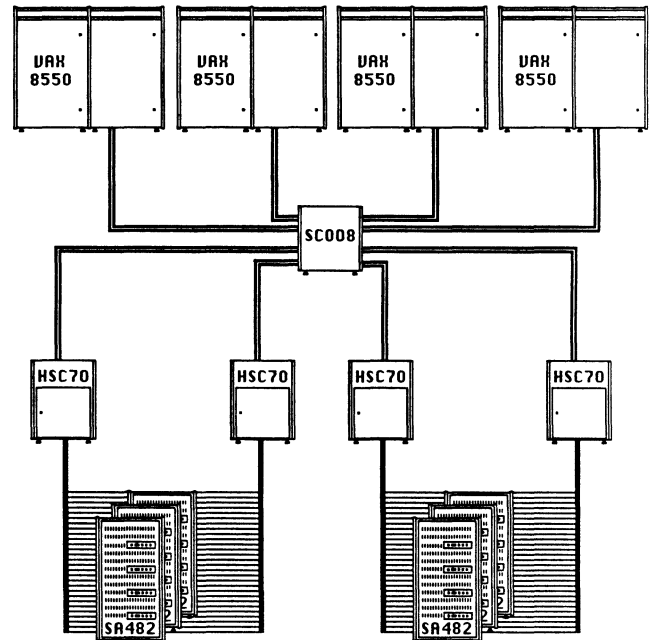
Quan	Part Number	Description
4	855CB-AP	8550 CPU 32MB CI PD UP SW 208/60
12	MS88-CA	16MB 256K 8800/8700/8500 MEM
4	HSC70-AA	HSC70 Base
24	HSC5X-BA	Disk Controller
2	HSC5X-CA	Tape Controller
6	SA482-AA	SA482 Storage Arrays
1	SC008-AC	8 Node Star Coupler w/CAB
2	TA78-BF	TA78 Tape Drive Master
2	TU78-AF	TU78 Tape Drive Slave
1	LPS40-AA	PrintServer 40
1	DJ-630C2-AA	VAXcluster Console System
8	FOCHA-AE	VCS Fiber Optic Conn Kit
8	BN25J-50	50 Meter Fiber Optic Cable for FOCHA
12	LN03-AA	LN03 Laser Printer
6	DELNI-AA	Local Network Interconnect
26	BC26V-25	SDI Cables - 25 Feet Long
4	BNCIA-20	CI Cables - 20 Meters Long
51	BNE3H-20	IEEE802 PVC STR AUI CBL
32	H4000	Ethernet Transceiver
3	BNE2A-ME	500 Meter Cable, Ethernet Teflon
6	12-19816-01	Cable Terminator
4	DEREP-AA	Local Ethernet Repeater
360	BC22D-50	50FT Cable, Null Modem
348	VT340-AA	VT340 Color Term
33	DSRVB-AA	DECserver-200 w/Modem CTL
3	DF100-RM	Multiple Modem, rack mount enc.
24	DF124-AM	2400/1200 bps Modem, module only
6	DSRZA-BA	MUXserver-100 120v
12	DFMZA-BA	DECmux II 120V
24	BC22F-10	10FT Cable, RS232
24		19.2 kbps Modems (from Phone Co.)

Software License Order

Quan	Part Number	Description
1	Q2740-UZ	VAXInfo I lic/warr
3	Q2740-QZ	VAXInfo I vaxcl lic only
1	Q2741-UZ	VAX Teamdata lic/warr
3	Q2741-QZ	VAX Teamdata vaxcl lic only
1	Q2A86-UZ	VAX Rally lic/warr
3	Q2A86-QZ	VAX Rally vaxcl lic only
1	Q2965-UZ	VAXSET package lic/warr
3	Q2965-QZ	VAXSET vaxcl lic only
1	Q2099-UZ	VAX COBOL lic/warr
3	Q2099-QZ	VAX COBOL vaxcl lic only
1	Q2365-UZ	VAX COBOL generator lic/warr
3	Q2365-QZ	VAX COBOL generator vaxcl lic
1	Q2AB2-UZ	VAX VOL SHAD lic/warr
3	Q2AB2-QZ	VAX VOL SHAD vaxcl lic only
1	Q2ZCC-UZ	VAX Perf Advisor lic/warr
3	Q2ZCC-QZ	VAX Perf Advisor vaxcl lic
1	Q2031-UZ	VAX VTX lic/warr
3	Q2031-QZ	VAX VTX vaxcl lic only
1	Q2033-UZ	VAX VTX TC/CON lic/warr
3	Q2033-QZ	VAX VTX TC/CON vaxcl lic
1	Q2031-UZ	VAX VALU lic/warr
3	Q2031-QZ	VAX VALU vaxcl lic only
1	Q2960-UZ	VAXNotes lic/warr
3	Q2960-QZ	VAXNotes vaxcl lic only
1	Q2AAA-UZ	ALL-IN-1 lic/warr
3	Q2AAA-QZ	ALL-IN-1 vaxcl lic only
33	Q2Z06-UZ	DECSEVER-200 lic/warr
6	Q2ZAW-UZ	MUXSERVER-100 lic/warr
1	Q2ZAV-UZ	SSU Licence, 8700/8550
3	Q2ZAV-QZ	SSU vaxcl license, 8700/8550
1	Q2A82-UZ	VAX SW PROJ MGR LIC
3	Q2A82-QZ	VAX SW PROJ MGR VAXCL LIC

Software Kits Order

Quan	Part Number	Description
1	Q2740-HM	VAXInfo I upd
1	Q2741-HM	VAX Teamdata upd
1	Q2A86-HM	VAX Rally upd
1	Q2965-HM	VAXSET package upd
1	Q2099-HM	VAX COBOL upd
1	Q2365-HM	VAX COBOL generator upd
1	Q2AB2-HM	VAX VOL SHAD upd
1	Q2ZCC-HM	VAX Perf Advisor upd
1	Q2031-HM	VAX VTX upd
1	Q2033-HM	VAX VTX TC/CON upd
1	Q2031-HM	VAX VALU upd
1	Q2960-HM	VAXNotes upd
1	Q2AAA-HM	ALL-IN-1 upd
1	Q2Z06-HM	DECSEVER-200 upd
1	Q2ZAW-HM	MUXSERVER-100 upd
1	Q2ZAV-HM	SSU Licence, 8700/8550 upd
1	Q2A82-HM	VAX SW PROJ MGR upd
1	QL797-HM	PRINTSRVR40 CLIENT upd
4	QL798-HM	PRINTSRVR40 SUPHOST upd
1	Q2001-HM	VAX/VMS UPD 16MT9
1	Q2D05-HM	DECNET-VAX F/F UPD



High End VAX Configuration Putting the Pieces Together

High Performance Systems Group
Digital Equipment Corporation
Marlboro, MA

This session will cover Digital's High End VAX Systems, VAXclusters, and VAXcluster Software and Services. We will discuss definitions, configuration rules and general purpose options for high end systems. Then, an example configuration of a high end VAXcluster will be provided.

This session will not cover (1) why to buy one CPU over another; (2) performance characteristics of individual processors; (3) realtime or non-general purpose options.

For the purpose of this session, a high end system is a clusterable VAX processor that has greater than twice the power of a VAX 8200/780. Therefore, we will be discussing the following systems: VAX 8530/8550; VAX 8600/8650; VAX 8700/8800; VAX 8974/8978. Also, for the configuration exercise, we will assume that all terminal communications will take place through LAN-based servers.

Configuration Rules VAX 8530/8550

All base systems include:

- 1 VAX BI Channel
- 1 Ethernet Interface
- 20 Megabytes of Memory
- Console Terminal
- Floating Point Accelerator
- Room for 8530-8550 Upgrade

Site Prep Info:

	<u>8530</u>	/	<u>8550</u>	<u>8530</u>	/	<u>8550</u>
Nominal Voltage	208		208	380/416		380/416
Frequency HZ	60		60	50		50
Current AC Amps	16		16	8		8
Thermal - Watts	3500		3200	3500		3200
- BTU/Hour	12,000		12,000	12,000		12,000
Power Plug	L21-20R		L21-20R	516R6W		516R6W
Size	60x27x30		60x27x30	152x68.5x76		152x68.5x76
Weight	966 Pounds		650 Pounds	495 kg		295 kg

Expansion Cabinet: H9652-EC/ED

Needed for:

- 2nd VAX BI BA32-BA/BB + DB88-AD
- UNIBUS Options BA11-AW/AX + DNBUA-A
- CI Interface CIBCI

Maximum of 2 of the above allowed

Site Prep:

Nominal Voltage V	120/208	240/416
Frequency HZ	50-60	50-60
Phases	3	3
Current AC Amps	X+2.8	X+1.4
Thermal - Watts	X+252	X+252
- BTU/Hour	X+860	X+907
Power Plug	L21-30R	(no NEMA Available)
Size	63.3 x 26.3 x 30 inches	153 x 66.7 x 76.2 cm
Weight	X+400 Pounds	X+181.6 kg

X = Power/Weight of internally mounted options

Configuration Rules VAX 8530/8550

	VAX BI Slots Available	Panel Units Available
Base Cab.*	4	10
DB88-AD in H9652-EC/ED	5	37

Option	VAX BI Slots Used	Pannel Units Used
Ethernet Port (DEBNT)	1	2
CI Port (CIBCI)	2	0
KDB50	1	2
TU81-Plus	1	1
DMB32	1	4
DWBUA	1	0
VAX BI Channel #2	1	0

All systems are now shipping with the new memory backplanes which support configurations up to 80 MB

8500 to 8530 Upgrade:

New Console distribution with new microcode which will upgrade current VAX 8500 systems to VAX 8530 systems.

Configuration Rules VAX 8700/8800

All base systems include:

8700

- 1 VAX BI Channel
- 1 Ethernet Adaptor
- 32 MB Memory
- Console Terminal
- Battery Backup
- Floating Point Accel.
- Space for 2nd CPU

8800

- 2 VAX BI Channels
- 1 Ethernet Adaptor
- 32 MB Memory
- Console Terminal
- Battery Backup
- Floating Point Accel.
- UNIBUS Adaptor,
Backplane, CI Port,
& CI Cables

Site Prep Info:

	<u>8700</u>	/	<u>8800</u>	<u>8700</u>	/	<u>8800</u>
Nominal Voltage	208		208	380/416		380/416
Frequency HZ	60		60	50		50
Current AC Amps	22		22	11		11
Thermal - Watts	3700		6000	3700		6000
- BTU/Hour	12,600		26,750	12,600		26,750
Power Plug	560R9W		560R9W	532R6W		532R6W
Size	60.5x74x30		60.5x74x30	154x188x76.2		154x188x76.2
Weight	966 Pounds		650 Pounds	495 kg		295 kg

Memory

All current systems are shipping with
new backplane which supports
configurations up to 128MB

Configuration Rules VAX 8700

H9652-EC/ED Expansion cabinet (up to two per system) provides space for any two combinations of:

- BA32-BA/BB (VAX BI exp. box)
- BA11-AW/AX (UNIBUS exp. box)

	VAX BI Slots Available	Panel Units Available
Base Cabinet*	10	26
2nd VAX BI** Chan. + DW88-AC	5	-
3rd VAX BI Chan. + DW88-AE	5	37
4th VAX BI Chan. + DW88-AD	5	-

* One Ethernet controller in each available configuration

** DB88-AC converts first VAX BI Channel with 11 slots into 2 VAX BI Channels with 5 slots each

VAX 8800

	VAX BI Slots Available	Panel Units Available
Base Cab.	7	28
3rd VAX BI Chan.+ DW88-AE	5	37
4th VAX BI Chan.+ DW88-AD	5	-

Configuration Rules

VAX 8600/8650

All base systems include:

- 1 DB86 SBI Adaptor
- 1 DW780 UNIBUS Adaptor
- 4, 16, or 32MB of Memory
- 1 Year Full Warranty

8600/8650

Nominal Voltage V	120/208	240/416
Frequency HZ	60	50
Current AC Amps	22	11
Thermal - Watts	6,500	6,500
- BTU/Hour	22,000	22,000
Power Plug	DF6516FRAB	DF3401FRAB
Size	60.25x73.25x30	153.7x186x76.2
Weight	1,700 Pounds	875 kg

Expansion Cabinet H9652-F & H9652-C (similar to the H9652-E)

Base cabinet contains space for:

- **CI780 VAXCluster Interface**
- **FP86 Floating Point Accelerator**
- **2nd DW780 UNIBUS Adaptor**
- **2nd DB86 SBI Adaptor**
- **861UP-AA 8600 > 8650 Upgrade**

Front end cabinet contains:

- **RL02 Front end disk**
- **BA11 UNIBUS expansion box**
- **24 Panel units**

Maximum of 2 SBIs per system for a total of 12 SBI devices:

2 (max)	CI780	(1 if DR780 is configured)
4 (max)	DR780	(1 if CI780 is configured)
4 (max)	RH780	
4 (max)	DW780	

VAX 8600/8650
Memory

Minimum of 4MB is included in base system, expandable up to 260MB in the 8 slot memory backplane

MS86-AA 4MB memory (up to 8)
MS86-CA 16MB memory (up to 4) *
MS86-DA 64MB memory (up to 4) *†

*** Each MS86-CA/DA takes 2 memory slots**

† VMS V4.5 & ULTRIX 2.0 support only 128MB of memory. Future versions will support a full 260MB of memory

(4) MS86-DA 64MB memory = 256MB
(1) MS86-AA 4MB memory = $\frac{4\text{MB}}{260\text{MB}}$

VAX 8974/8978

Complete systems consisting of:

	<u>8974</u>	<u>8978</u>
VAX 8700 CPU	4	8
HSC70	2	4
Disk Channels	12	24
Tape Channels	2	4
SA482 - 2.5GB	1	2
TA78	1	2
VAXcluster Console	1	1
Star Coupler	8 Node	16 Node
NI Tranceiver Cables	6	11
VCS Cables Kits	6	12
DELNI	1	2
DECserver-200	1	1
VMS		
DECnet		
VPA		
Volume Shadowing		

Site Prep Info:

	<u>8974</u>	<u>8978</u>
Power Requirements	60 KVA, 20 KW	115 KVA, 40 KW
Heat Dissipation (per Hr.)	60.02 MJ (58.86 KBTU)	124.19MJ (117.72 KBTU)
Foot Print	28.08 M ² (312 FT ²)	54.29 M ² (603.25 FT ²)
Weight	3951 kg (8780 lb.)	7834 kg (17410 lb)

Disk Interfaces

	<u>KDB50</u>	<u>UDA50</u>
Bus	VAX BI	UNIBUS
Data throughput		
- Burst	3 mb/sec	3 mb/sec
- Sustained	1 mb/sec	750 kb/sec
Seek Optimization requests	up to 20	up to 20
Sector buffering	up to 42	up to 52
DMA	yes	yes
Max. SDI Disks	4	4
Slots	1 VAX BI	2 UNIBUS Hex

VAXcluster Controllers

	<u>HSC50</u>	<u>HSC70</u>
Max SDI Disks	24	32
Max Channel cards	6	8
Control Processor	F-11	J-11
Data memory	128kb	256kb
Program memory	256kb	1024kb
Control memory	128kb	256kb
Instruction cache	0	8kb
Load device	TU58	RX33
System boot time	6 min	1 min
Aux power supply	Optional	Included
I/O Per Second	200	310

Disk Channel	HSC5X-BA	4 Disk drives
Tape Channel	HSC5X-CA	4 Tape masters

Aux Power Supply HSC5X-EA/EB Required for HSC50 with more than 3 channels

Disks

	<u>RA60</u>	<u>RA81</u>	<u>SA482</u>
Type	Removable	Fixed	Fixed
Number of Spindles	1	1	4
Size (MB)	205	456	2,560
Peak Xfer (MB/Sec)	1.98	2.2	2.4/Spindle
Avg Access (mSec)	50	36.3	24/Spindle
Avg Latency (mSec)	8.33	8.33	8.33/Spindle
Media Surfaces (data)	6	7	32
Sectors/Track	42	51	57
Single Track Seek	6.7	6	3

Tapes

	<u>Tx78</u>	<u>Tx81</u>
Density	1600/6250	1600/6250
Speed (IPS)	125	75/25
Rewind Speed (IPS)	440	192
Transports/Master	4	1
Masters/Channel	4	4
Capacity (MB/2400' tape)	40/145	40/145

Cluster Configuration Rules

- 16 nodes per cluster
- 1 Star Coupler per cluster
- Each CPU or HSC counts as a node
- Maximum of 45 meters between Star Coupler and any node

Preconfigured System:

- New VAX Installation
- Stand alone Applications
- Lower performance than a VAXcluster
- Includes Disk, Tape & Communications
- Fully runnable system

VAXcluster SBB:

- CPU Configured for a VAXcluster
- Highest Performance
- Use new/existing disk & tape

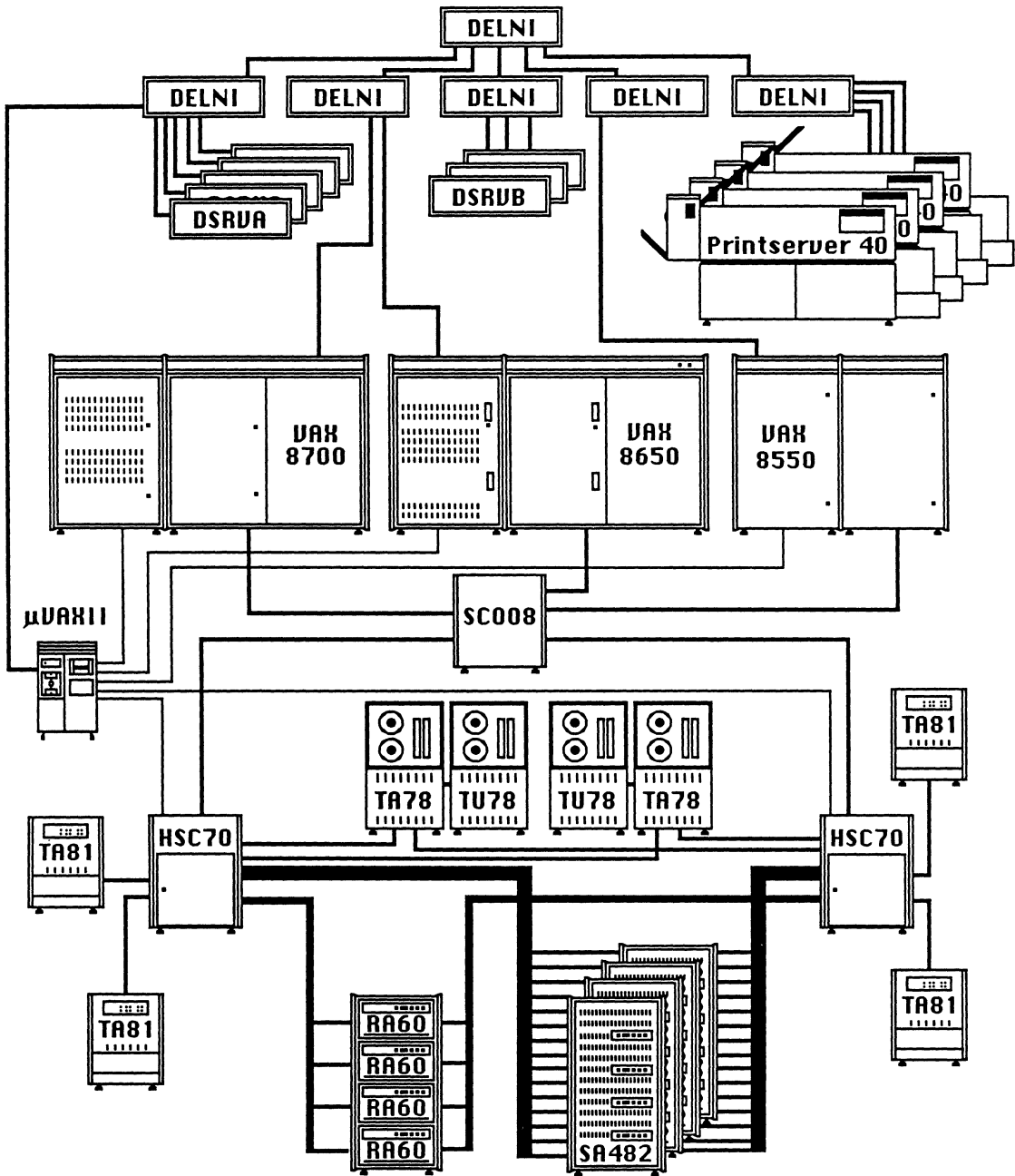
VMS System Building Blocks:

- Replace existing system
- Use existing disk & tape
- Use existing communications
- No VAXcluster hardware (except VAX 8800)

Building A VAXcluster From Scratch

The target system is a three CPU VAXcluster including:

- 1 VAX 8550 (64MB)
- 1 VAX 8700 (96MB)
- 1 VAX 8650 (96MB)
- 2 HSC70 Storage Controller
- 4 SA482 Storage Arrays
- 4 RA60 Disks
- 200 Ethernet Based Local Terminal Connections
- 24 Ethernet Based Dialup Terminal Connections
- 8 1600/6250 bpi Tape Drives
- 4 40 Page/minute Ethernet Based Laser Printers
- 1 VAXcluster Console System

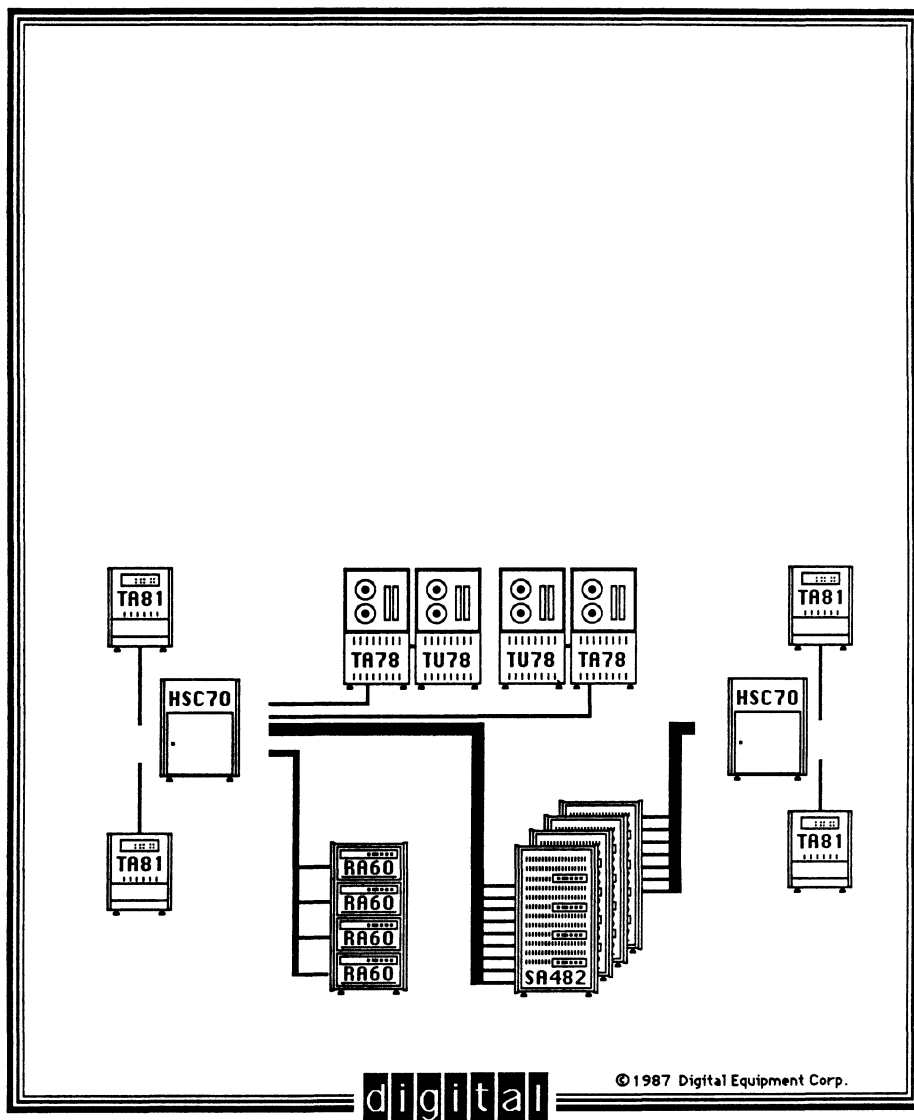


Step 1 - Order the Disks, Tapes, and HSCs

Order:

- 4 SA482-AA/AD 2.56 GB Storage Array
- 1 RA60-JA/JD (4) RA60 w/4 Hi Cab
- 2 TA78-BF/JB Dual Acc Master
- 2 TU78-AF-AJ 1600/6250 Slave Drive
- 4 TA81-AA/AB PE/GCR 25/75 ips for HSC
- 2 HSC70-CA/CB Base HSC70 w/VCS interface

What we have now is this:



Step 2 - We need to order channels for the HSC70's

How many?

Given:

- 1 SA482 per Channel
- 4 TA81s per Channel
- 4 TA78s per Channel

Therefore:

- 4 SA482s = 4 Disk Channels
- 4 RA60s = 1 Disk Channel
- 4 TA81s = 1 Tape Channel
- 2 TA78s = 1 Tape Channel

Therefore we need 5 Disk Channels and 2 Tape Channels

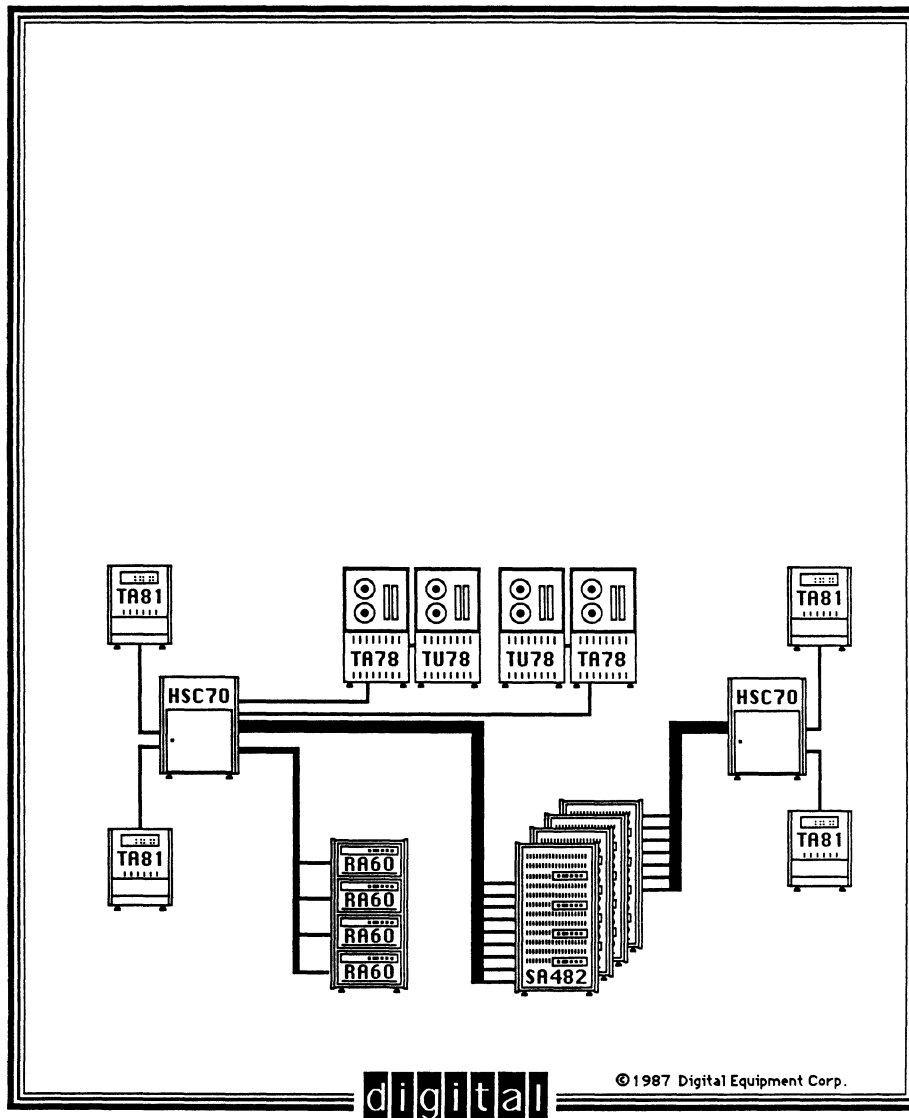
Order:

- 5 HSC5X-BA Disk Data Channel
- 2 HSC5X-CA Tape Data Channel

We want to dual port the disks. Therefore we need additional channels.

Order:

- 5 HSC5X-BA Disk Data Channel



Step 3 - Order dual porting cables

All SA482s, RA60s, TA78s, & TA81s come with one BC26V-12 SDI cable

Order:

22 BC26V-25 Shielded SI Cable 25ft

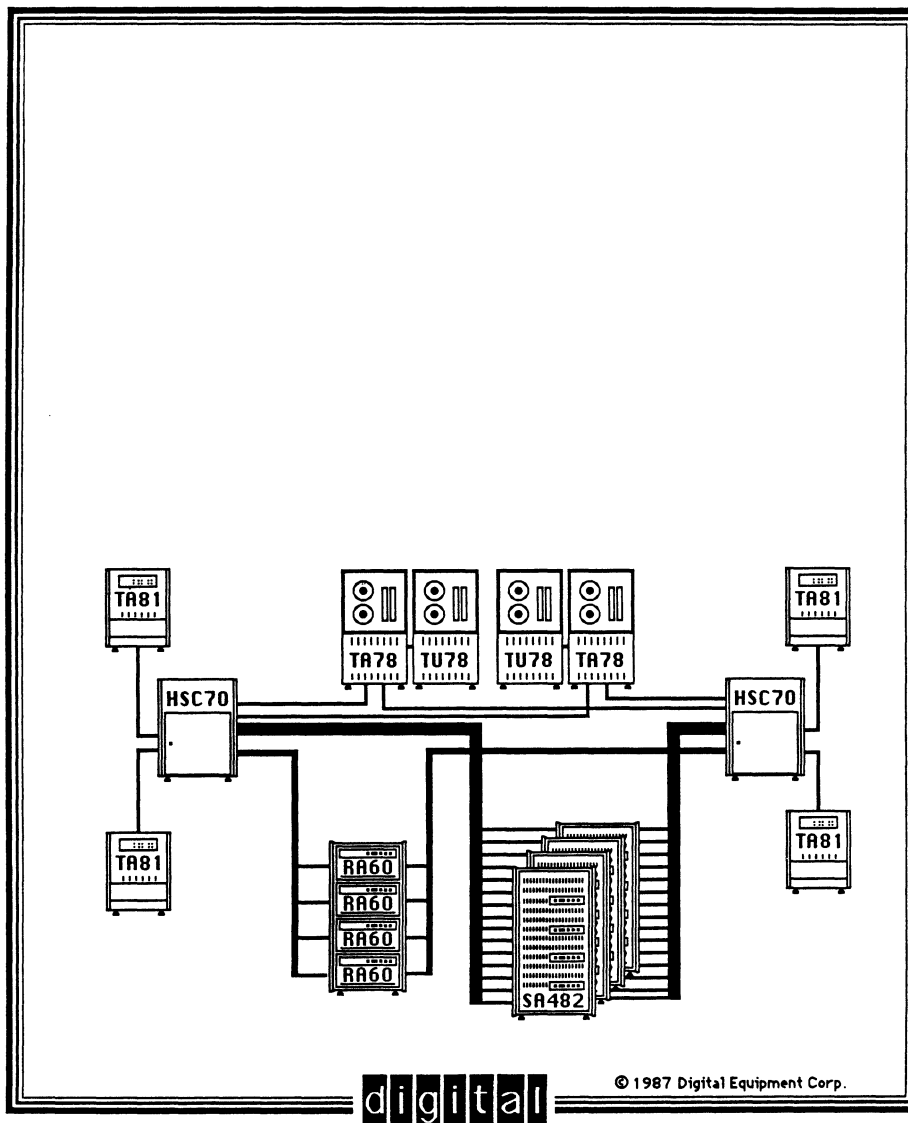
SDI cables come in 12, 25, 50, & 80 foot lengths

Note: The disk cabinets are 22 inches wide so there won't be much slack left using only 12 foot cables. Therefore we will order 25 foot cables for dual porting and have Field Service move the cables around so that they fit.

Given:

- 4 SA482s = 16 SDI
- 4 RA60s = 4 SDI
- 4 TA81s = 0 SDI (TA81s can't be dual ported!)
- 2 TA78s = 2 SDI

Therefore we need 22 SDI cables



Step 4 - Order the CPUs

4 MS88-CA 16MB memory exp
 1 FOCHA-AC/AD VCS fiber conn kit

Order the 8550 VAXcluster SBB:

1 855CB-AP/AT VAX 8550 Cpu 32MB CI 1yr sw Pdup
 8550 CPU 32MB 256k mem FP
 VAX BI channel (one)
 CI Port, BNCIA-20, exp cab
 Ethernet comm interface
 Console Terminal
 One year hardware warranty
 Q2001-UZ VMS Paid Up Lic.
 Q2D05-UZ DECnet F/F Paid Up

Order the 8650 VAXcluster SBB:

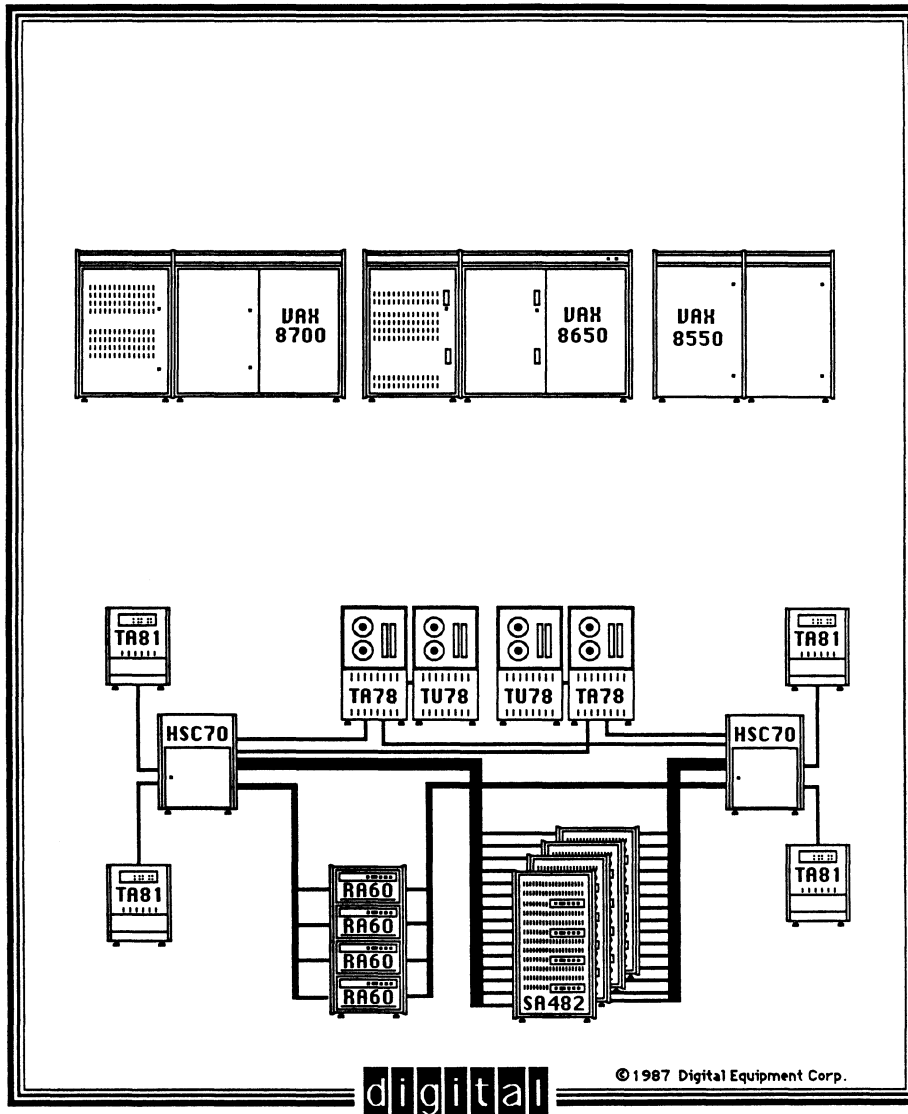
1 865CD-AP 8650 QK001-UZ 32MB 120/60 HOST
 8650 CPU 32MB(256k) memory
 CI780-MA 8600 SBI-CI adapter
 BA11-A UNIBUS EXP
 (2) DD11-DK, DD11-CK
 DELUA-M Ethernet interface
 CK-DELUA-KM (cab kit for DELUA)
 (1) BNCIA-20
 QK001-UZ VAX/VMS Lic/warranty
 QKD05-UZ DECnet license

2 MS88-CA 16MB memory exp
 1 FOCHA-AC/AD VCS fiber conn kit

Order the 8700 VAXcluster SBB:

1 871CB-AP/DP 8700 CPU 32MB 256k mem FP,BBU
 VAX BI Channel
 CI Port, BNCIA-20
 Ethernet comm interface
 Console Terminal
 One year hardware warranty
 Q2001-1P VMS Paid Up Lic.
 Q2D05-1P DECnet F/F Paid Up Lic.

1 MS86-DA 64MB memory exp
 1 FOCHA-AC/AD VCS fiber conn kit
 1 FP86-AA Floating Point Accelerator



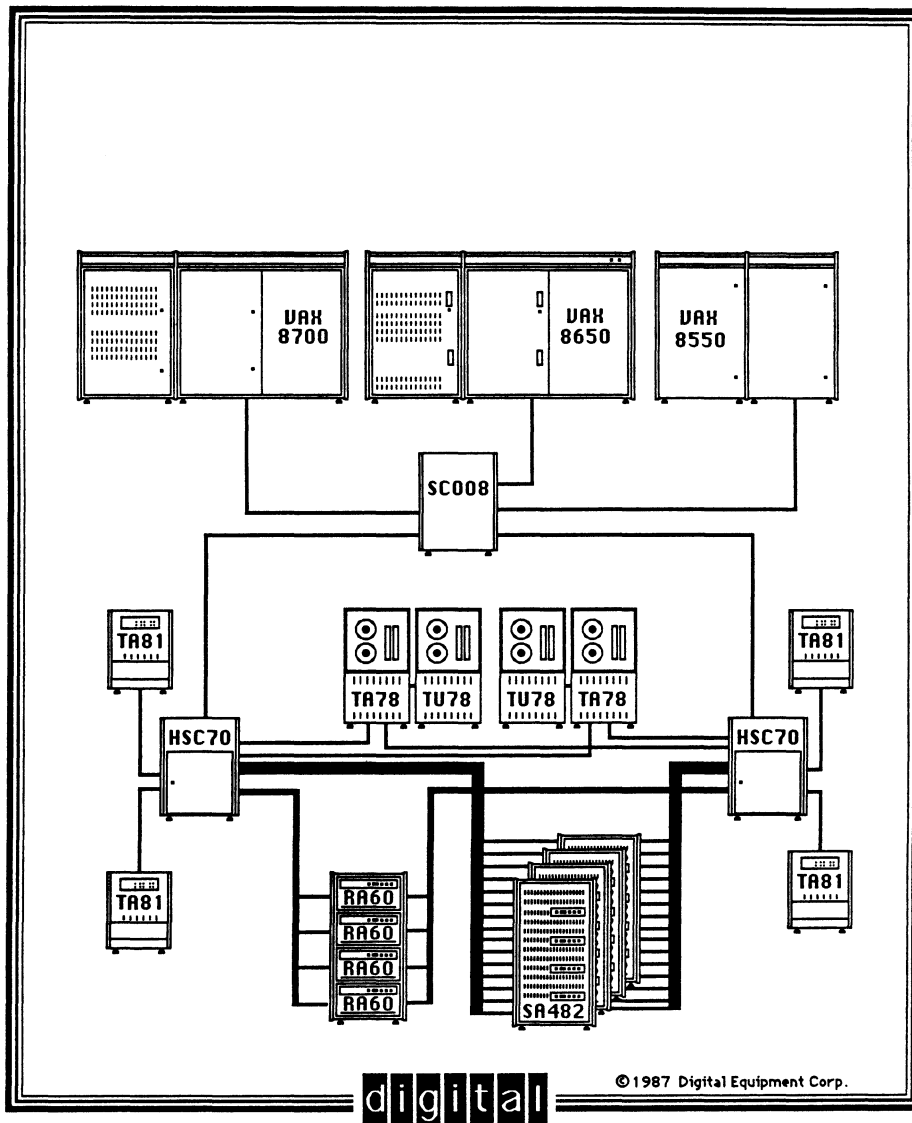
Step 5 - Now connect the CPUs to the storage subsystem

Order a Star Coupler and CI Cables:

- 1 SC008-AC 8 Node Star Coupler
- 2 BNCIA-20 CI Cable Set - 20 meters

Note: The VAXcluster System Building Blocks all come with a set of 20 meter CI Cables. While it is not required, it is suggested that all CI cables be the same length.

Note: The HSC70's do not come with a set of CI Cables.



Step 6 - Now order communications equipment

Given:

200 Local lines @ 8 lines per DECServer-100
= 25 DECServer-100s

24 Dialup lines - 8 lines per DECServer-200
= 3 DECServer-200s

Order:

25	DSRVA-AA/AB†	8 line term server
3	DSRVB-AA/AB	DS200 RS232 8 line termsrv
3	DSRVB-B*	Country kit

† DSRVA-AB needs Country Kit DSRVA-A*

Step 7 - Order the Laser Printers

Given:

160 pages per minute worth of Laser Printing equipment needed

Printserver 40 delivers 40 pages per minute

Order:

4 LPS40-AA/A3 Printserver 40

Note: The Printserver 40 base price includes the licenses for the printserver software but does not include the media or documentation

Step 8 - Order Network Equipment & Cables

Now we need to order network connections and cables for the VAX CPUs, the DEC Servers, and the Printservers

Note: For this example we will build our network around cascaded DELNIs and will not use Ethernet cable and transceivers. In local mode five DELNIs connected to a sixth give us 40 network connections.

Given:

3	VAX CPUs
25	DECserver-100s
3	DECserver-200s
4	Printserver 40s
1	VAXcluster Console System

We need 36 network connections.

Order:

6	DELNI-AA/AB	Local Network interconnect
6	DELNK-A*	DELNI country kit (Non US only)
5	BNE3A-5	5 Meter Ethernet cable
32	BNE3A-10	10 meter Ethernet cable
4	BNE3A-20	20 meter Ethernet cable

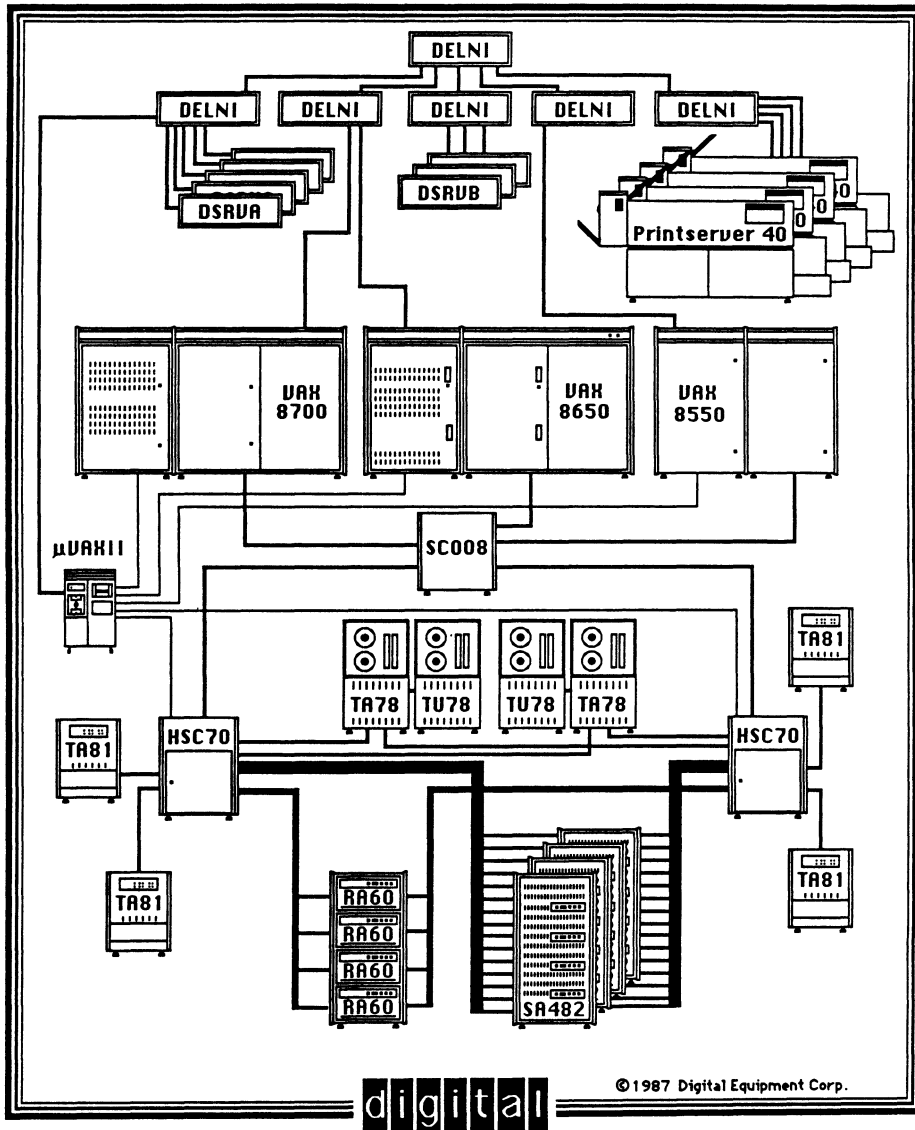
Step 9 - Order the VAXcluster Console System

Order the VAXcluster Console System:

1	DJ-630C2-A2/A3	VAXcluster Console System DH-630Q3-FA/F3 MVII system QZ002-C5 MVMS Lic,Bin,Doc QZ002-H5 MVMS 1-8 User Pd QZV01-UZ VCS lic QZV01-H5 VCS bin,doc H7133-A Power Supply
---	----------------	---

5	BN25J-LL	Fiber Optic Cable for FOCHA†
1	QZD05-UZ	DECnet-Microvax F/F
1	QZD05-H5	DECnet-Microvax media & Doc
1	QZD05-95	DECnet-Microvax DPMC
1	QZD05-15	DECnet-Microvax Installation

† For non-US orders country kits are needed, two for DJ-630C1-A3 and one for each FOCHA-AD

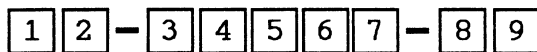


Step 10 - Now Order the Software:

We want to order the following software products:

VAX/VMS	VAXINFO I
DECnet Full Function	VAXset
VAX Volume Shadowing	Pascal
VAX Performance Advisor	Fortran
VAXnotes	Cobol
DECserver-100	DECserver-200

Software Options



- 1 & 2 Not normally used except as subparts in BOMs
- 3 Software Type - Normally "Q"
- 4 Processor Type
 - 2 = 8550/8700/8974/8978
 - K = 8600/8650
- 5 - 7 Product Identifier (100 = Fortran)
- 8 License & Service
 - J = Period Payment (BI machines only)
 - U = Single use license
 - Q = Single use license for VAXcluster node
 - B = Startup Package Level III
 - 7 = Startup Package Level II
 - 5 = Startup Package Level I
 - I = Installation Service
 - 9 = DECsupport
 - 8 = Basic
 - 3 = Self maintenance
 - K = Additional Update
 - H = Binaries & Documentation
 - G = Documentation only
- 9 Media
 - M = 1600 BPI Tape
 - 5 = TK50
 - Z = None

Example:

Fortran for an 8700 CPU with:
Installation
DECservice
Paid Up License
2 Copies of Documentation
& Doc Service

Q2100-UZ	VAX FORTRAN w/Warranty
Q2100-IZ	Installation
Q2100-9M	DECservice
Q2100-GZ	Documentation
Q2100-HM	Media & Documentation
(2) Q2100-KZ	Documentation Services

Example:

Fortran for an 8700 CPU with:
Installation
DECservice
Periodic Payment License
2 Copies of Documentation
& Doc Service

Q2100-1B	VAX FORTRAN Init Lic Charge
Q2100-JP	VAX FORTRAN Pri-PPL
Q2100-IZ	Installation
Q2100-9M	DECservice
Q2100-GZ	Documentation
Q2100-HM	Media & Documentation
(2) Q2100-KZ	Documentation Services

VMS and DECnet licenses are included in system building blocks

VAX Volume Shadowing

Q2AB2-UZ VAX Volume Shadowing lic w/warr
Q2AB2-QZ VAX Volume Shadowing vaxc lic w/warr
QKAB2-QZ VAX Volume Shadowing vaxc lic w/warr

VAX Performance Advisor

Q2ZCC-UZ VAX Performance Advisor lic w/warr
Q2ZCC-QZ VAX Performance Advisor vaxc lic w/warr
QKZCC-QZ VAX Performance Advisor vaxc lic w/warr

VAXset

VAXset consists of the following products:

- VAX Language Sensitive Editor (LSE)
- Source Code Analyzer (SCA)
- VAX Performance and Coverage Analyzer (PCA)
- VAX DEC/CMS
- VAX DEC/MMS
- VAX DEC/Test Manager

Q2965-UZ VAXset License w/warr
Q2965-QZ VAXset vaxc lic w/warr
QK965-QZ VAXset vaxc lic w/warr

VAXinfo I

VAXinfo I consists of the following products:

- VAX Common Data Dictionary (CDD)
- VAX TDMS
- VAX DATATRIEVE
- VAX Rdb/VMS

Q2740-UZ VAXinfo I license w/warr
Q2740-QZ VAXinfo I vaxc lic w/warr
QK740-QZ VAXinfo I vaxc lic w/warr

Now for the rest of the licenses:

Fortran

Q2100-UZ VAX FORTRAN license w/warr
Q2100-QZ VAX FORTRAN vaxc lic w/warr
QK100-QZ VAX FORTRAN vaxc lic w/warr

Cobol

Q2099-UZ VAX COBOL license w/warr
Q2099-QZ VAX COBOL vaxc lic w/warr
QK099-QZ VAX COBOL vaxc lic w/warr

Pascal

Q2126-UZ VAX PASCAL license w/warr
Q2126-QZ VAX PASCAL vaxc lic w/warr
QK126-QZ VAX PASCAL vaxc lic w/warr

VAXnotes

Q2960-UZ VAXnotes license w/warr
Q2960-QZ VAXnotes vaxc lic w/warr
QK960-QZ VAXnotes vaxc lic w/warr

DECserver-200

(3) Q2Z06-UZ DECserver-200 lic w/warr

DECserver-100

(25) Q2925-UZ DECserver-100 lic w/warr

Step 11 - Now Order Installation Services

2 Q2025-BM New VAXcluster SPS lvl III Base Node
Q2025-5Z New VAXcluster SPS lvl III 8550/8700
QK025-5Z New VAXcluster SPS lvl III 8650

VAXcluster System DECsupport Level III Startup Package

These services cover all Digital eligible licensed software on all VAX nodes within the cluster for a fixed price. (there are some exceptions) The services consist of the following components:

- Critical advisory and scheduled preventive support
- Telephone Support Access to a VAXcluster support team for cluster specific problems
- Access to Digital Software Information Network (DSIN)
- A special VAXcluster System Newsletter/DISPATCH
- One set of software media and multiple copies of documentation kits for all licensed product updates
- Access to the Software Problem Reporting (SPR) system
- VAXcluster software installation and DECstart
- One year of VAXcluster DECsupport
- A DECPLAN training account

Step 12 - Now Order the Media

Now we have purchased the needed licenses for our layered products. This means we have the right to run them on our VAXcluster system. We have ordered the rights to run the software. We must now order the 'H' kit which is media and documentation:

Q2001-HM	VAX/VMS media and documentation
Q2D05-HM	DECnet Full Function media and doc
Q2ZCC-HM	VAX Performance Advisor media/doc
Q2965-HM	VAXset media/doc
Q2740-HM	VAXinfo I media/doc
Q2100-HM	VAX FORTRAN media/doc
Q2099-HM	VAX COBOL media/doc
Q2126-HM	VAX PASCAL media/doc
Q2960-HM	VAXnotes media/doc
Q2Z06-HM	DECserver-200 media/doc
Q2925-HM	DECserver-100 media/doc
QX926-H7	HSC70 media/doc
QL798-HM	Support Host Software
(2) QL797-HM	VAX/VMS client software

These are no cost because of the VAXcluster Level III SPS that was ordered but must still be ordered so that we get the correct media and documentation for our layered products.

Now we have our system properly configured and we have all the licences, media and documentation. We also get installation of products and DECstart which will install all and customize our startup files for us. We need a training plan to spend money in the DECPLAN account and we are finished.

DECSYSTEM-20 TECHNICAL UPDATE

Mark Pratt

Tops-20 Monitor Group

TOPS-20 AUTOPATCH STATUS

- o Tape 15 Shipped 16 Mar 87
- o Tape 16 Planned 21 Aug 87
- o See Buzz Hamilton in booth

TOPS-20 DOCUMENTATION STATUS

- o Notebook update # 28 being distributed now. Contains FORTRAN v11 documentation
- o Autopatch tape 16 will include new documentation for DDT

MAJOR GOALS OF TOPS-20 RELEASE 7

MAINTAINABILITY / RELIABILITY

- Dump on Bugchk
- * CFS Node Joining Message
- Auto-Answer Startup
- * Cluster GALAXY
- EXEC Cleanup
- * Increase # of Structures
- Offline Structures
- * Cluster Dump

COMPLETION OF CFS

- Cluster ENQ/DEQ
- * Cluster GALAXY
- Cluster Data Gathering
- * Increase # of Structures
- Login Structure
- * CFS Node Joining Message
- * Cluster Dump

* project which meets multiple goals

NEW FEATURES

- Printer Support (4 projects)
- Command Editor
- EXEC
- Recognition
- Unprivileged OPR
- * Cluster GALAXY
- PMOVE/PMOVEM use in monitor

* project which meets multiple goals

* project which meets multiple goals

DUMP ON BUGCHK PROJECT

- o Continuable dump of BUGCHK/INFs
- o Multiple dump structures
- o DOBOPR program
- o DOB% JSYS
- o Faster BOOT

CLUSTER DUMP PROJECT

- o Takes a simultaneous crash dump of all systems in the cluster
- o Invoked at console or can be invoked from within the monitor
- o Broadcasts a message to all systems to take a cluster dump
- o Attempts to dump entire cluster however some systems may not get the message
- o Digital use only
- o Not documented

OFFLINE STRUCTURES PROJECT

- o Automatically prevents new access to structures which have gone offline
- o Helps to prevent hung jobs by not allowing new IORBs to be queued
- o Enabling and timeout period is controlled by SETSPD
- o Does not unhang already hung jobs

PMOVE/PMOVEM USE IN MONITOR

- o PMOVE/PMOVEM - Physical memory move instructions
- o Helps CI/NI performance but actual monitor performance hasn't been measured
- o 5 to 7 times faster than monitor routines
- o Microcode 442 which went out on an update tape last year
- o Optionally utilized by AP tape 15 monitor
- o Required by Release 7

CLUSTER ENQ/DEQ PROJECT

- o Logically extends ENQ/DEQ to cluster environment
- o Must explicitly use new ENQ function to enable
- o Once enabled, becomes permanent for that process only
- o Local ENQs work as they do in 6.1
- o Available for customer applications and third party software, however there are no plans to upgrade Digital products

LOGIN STRUCTURE PROJECT

- o Allows common PS: to be shared by systems within the cluster
- o BS: structure - System specific swapping, startup procedures, files, etc
- o Any CI or MASSBUS disk which is available to become PS:
- o Based on Stanford changes but this is not NON-PS: login
- o Some utilities will change
- o Anything which writes to common areas may have to change
- o Enabled thru SETSPD

CLUSTER DATA GATHERING PROJECT

- o Remote access to monitor information of other systems within the cluster
- o Only informational JSYSes simulated
- o Cluster SYSTAT
- o New INFO% JSYS
- o General purpose monitor message SYSAP
- o Cluster-wide access for TTMSG
- o Enabled thru SETSPD except for Cluster GALAXY functions

INCREASE # OF STRUCTURES PROJECT

- o Allows up to 64 structures

CFS NODE JOINING MESSAGE

- o Console messages are output when a node cannot join the cluster

AUTO-ANSWER STARTUP PROJECT

- o Automatic response to "Why Reload" and "Run CHECKD" questions if no operator has responded
- o Defaults to "Other" and "No" with a comment "Question timeout"

EXEC

- o Fix cosmetic bugs
- o Fix known problems
- o Fix documentation conflicts
- o Some new minor features
- o New GALAXY support

COMMAND EDITOR PROJECT

- o Command journaling in monitor
- o Editor code exists in the EXEC
- o VMS style command editing will be supported

RECOGNITION PROJECT

- o Recognizes filespecs, keywords, and switches to the end of ambiguity
- o Provides better question mark help
- o Does not work in directory field

CLUSTER GALAXY PROJECT

- o Simplifies dismounting of structures within the cluster
- o Extensions to the SHOW commands to return remote information
- o Cluster SEND-all
- o Compile time option

UNPRIVILEGED OPR PROJECT

- o Allows non-Wheel/Oper users to perform certain OPR commands
- o Controlled by a privileged OPR command and requires new priv bit

PRINTER SUPPORT PROJECT

- o We are investigating remote printing services for:
 - o LAT 5.1
 - o within Tops-20 cluster
 - o to VAXes
 - o to terminals

FIELD TEST

- o January 88
- o Looking for FT sites, let us know by August 87

TOPS-10/20 MS/MX INTERNALS

Mark Pratt

Tops-20 Monitor Group

MS/MX BACKGROUND INFO

- o MS creates a .MAI file
- o MS sends IPCF packet to MX
- o MX knows local node names and Decnet node names
- o MX validates the addresses
- o MX creates an .ENV file and queues work requests
- o MX sends response back to MS

MX SENDERS/LISTENERS

- o MX maintains a simple round robin scheduler
- o Tasks which get scheduled are:
 - o Decnet sender
 - o Decnet listeners (many, both SMTP and MAIL-11)
 - o Local Mail sender
 - o Local Mail listener (IPCF handler)

HOW MS/MX HANDLES ARPANET

- o MS sends IPCF packet to MX
- o MX validates the addresses and kicks back unknown nodes
- o MX sends response back to MS
- o MS checks response and looks for addresses which have the "Don't know this node" error
- o MS passes addresses thru special routines to identify network
- o Address is found to be Arpa name and the message gets queued to MMAILR

NEW MAIL SENDERS/LISTENERS

- o Currently, all senders and listeners use common modules called NMXNET and MXLCL to do I/O and scheduler interfacing
- o New Decnet mail protocols are no problem. Non-Decnet mail protocols need NMXNET/MXLCL equivalents
- o NETTAB contains network data structures. Local and Decnet are set up now along with some non-used Arpa data structures



VMS Internals for TOPS-10/ 20 System Programmers

David Wager

Digital Equipment Corporation

Marlboro, MA

ABSTRACT

VMS is a virtual system, and has, relative to TOPS-10/20, more modularity of code, more flexibility in application design (with some exceptions), more features (depending on your point of view), and certainly more code. These differences come with a price, however. VMS is not as internally efficient across all loads as a highly customized operating system like TOPS. Software installation and on-line O/S debugging are more difficult. -

INTRODUCTION

This paper presents an overview of VMS operating system internals for the experienced systems programmer. It assumes that the reader has an in-depth knowledge of the TOPS-10 or TOPS-20 operating system, and focuses on the way familiar operating system functions such as job scheduling, memory management and I/O processing are handled in VMS.

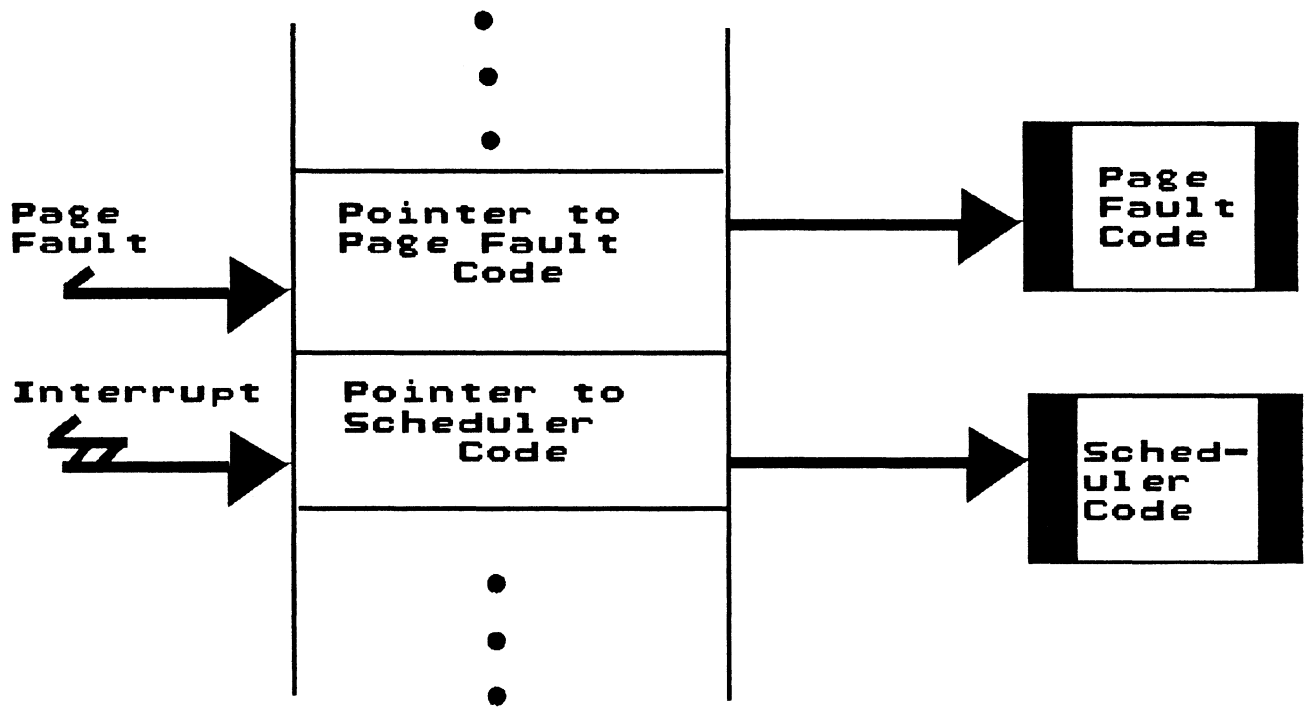
This paper is primarily figure-oriented. It is much easier to discuss operating system internals via diagrams. Each figure contains text briefly describing the particular operating system feature or component. Specific items presented include the following:

invoking system code
hardware and software priority
operating system access modes and components
entry paths into the kernel
actions taken on hardware clock interrupt
actions taken on device timeout check
actions taken on swapper wakeup
system event reporting
page fault handling
RMS data transfer
\$QIO data transfer
process data structures
virtual address space
synchronization techniques
AST delivery
exception and interrupt dispatching
system service dispatching
memory allocation
process states and transitions
process creation
image formation, layout, and installation
working sets
XQPs and ACPs
I/O database
VAXcluster components

These figures can be thought of as supplementary material to Digital's VAX/VMS Internals and Data Structures (Lawrence J. Kenah and Simon F. Bate, Digital Press, 1984). -

INVOKING SYSTEM CODE

EVENT → TABLE → EXECUTED CODE



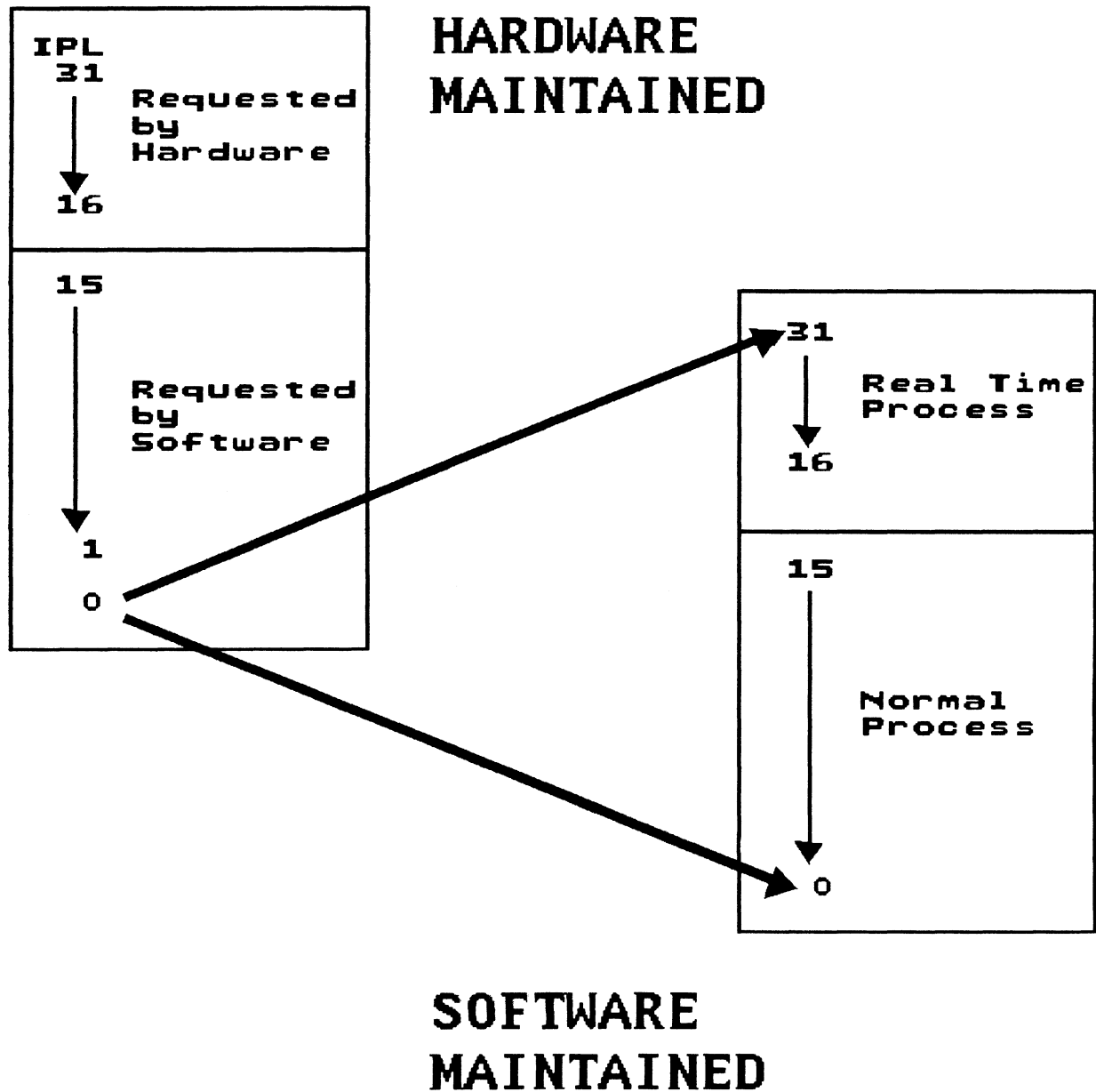
VAX/VMS driven by interrupts and exceptions

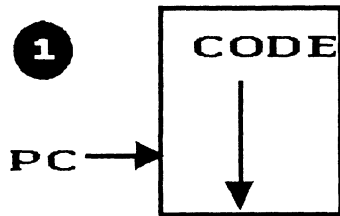
On interrupt or exception, hardware vectors to correct code

Example, Page Fault

- Page Fault occurs
- Hardware vectors through table
- Page Fault code executes

TWO TYPES OF PRIORITY

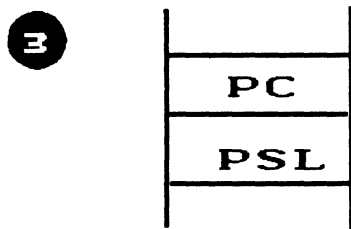




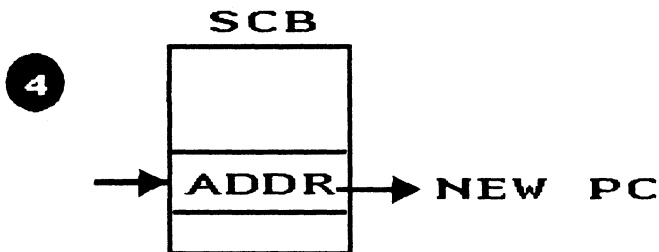
User program being executed.
 PC = address of next instruction to be executed.
 PSL = general status information.



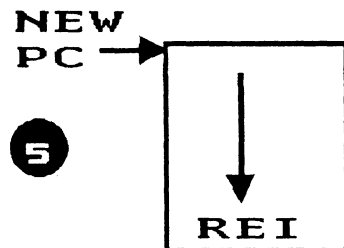
Interrupt occurs. Associated IPL must be greater than current IPL in PSL, else interrupt not serviced.



Hardware saves current PC and PSL on Stack.

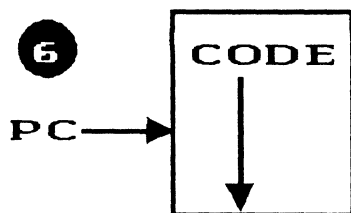


Hardware indexes into table of service routine addresses to get new PC, and builds new PSL



Interrupt service routine executes at new IPL.

At end, interrupt dismissed with an REI instruction (making sure old PC and PSL are at top of the Stack.)

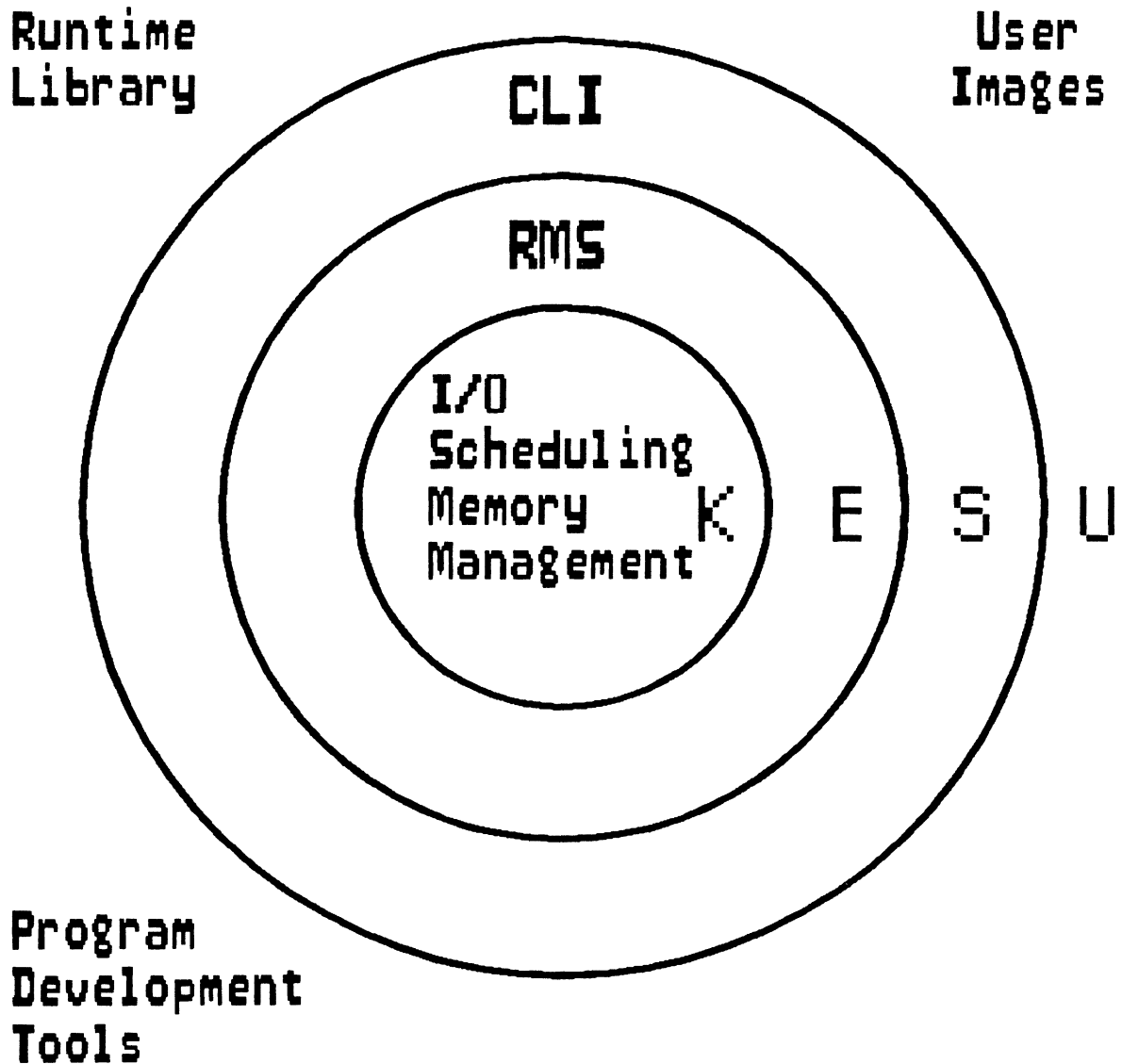


REI

- Pops PC, PSL from Stack
- Checks PSL
- Moves PC, PSL to CPU registers
- Transfers control to PC

Interrupted program continues execution.

ACCESS MODES AND COMPONENTS

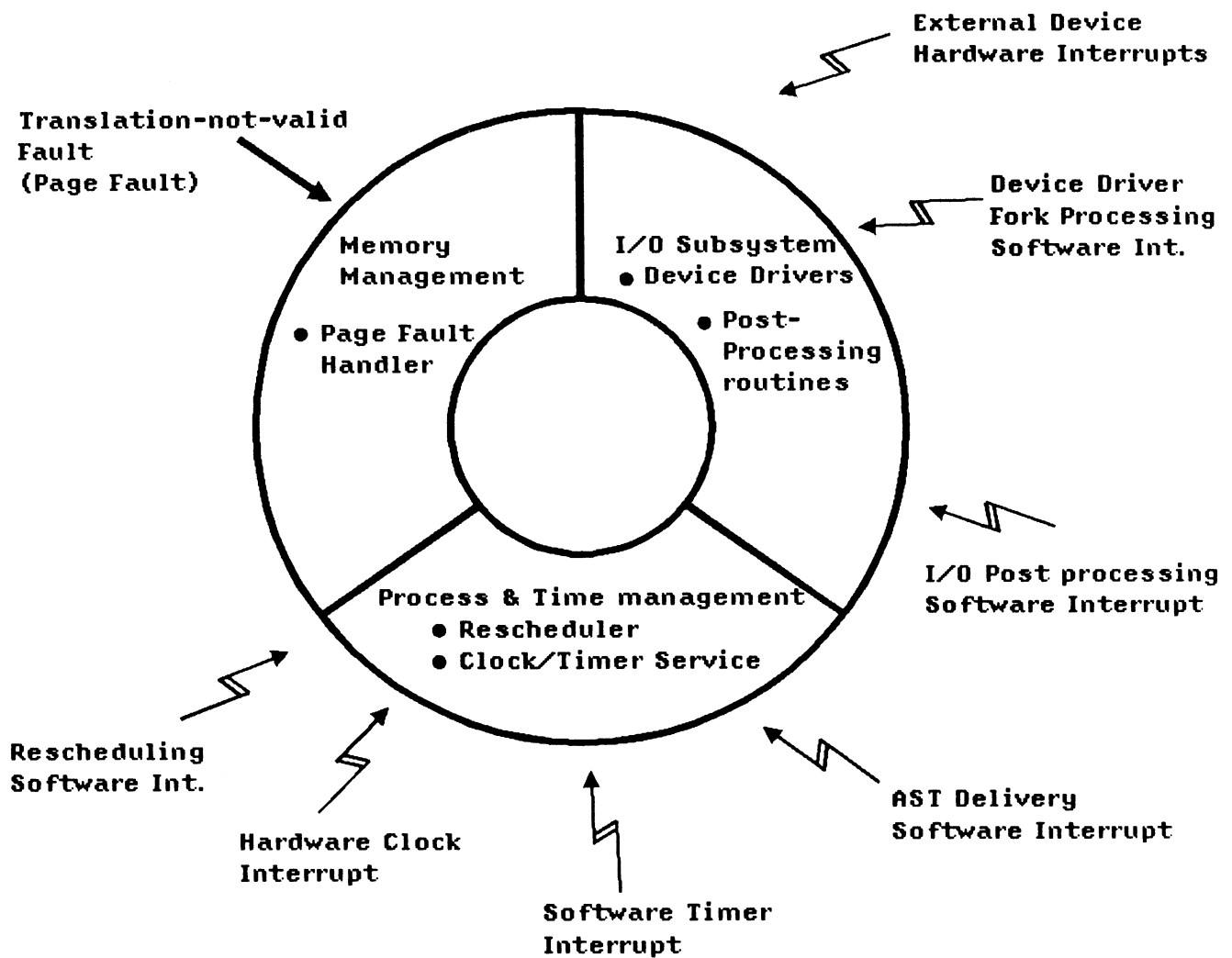


Access Modes and Components

Kernel of the operating system is protected from user by several layers of access protection

User normally accesses protected code and data through the Command Language Interpreter (CLI), Record Management Services (RMS), and system services.

System services - routines in operating system kernel that may be called by the user by means of a well-defined interface.



Entry Paths into VMS Kernel

Memory Management

- Brings virtual pages into memory

Process and Time Management

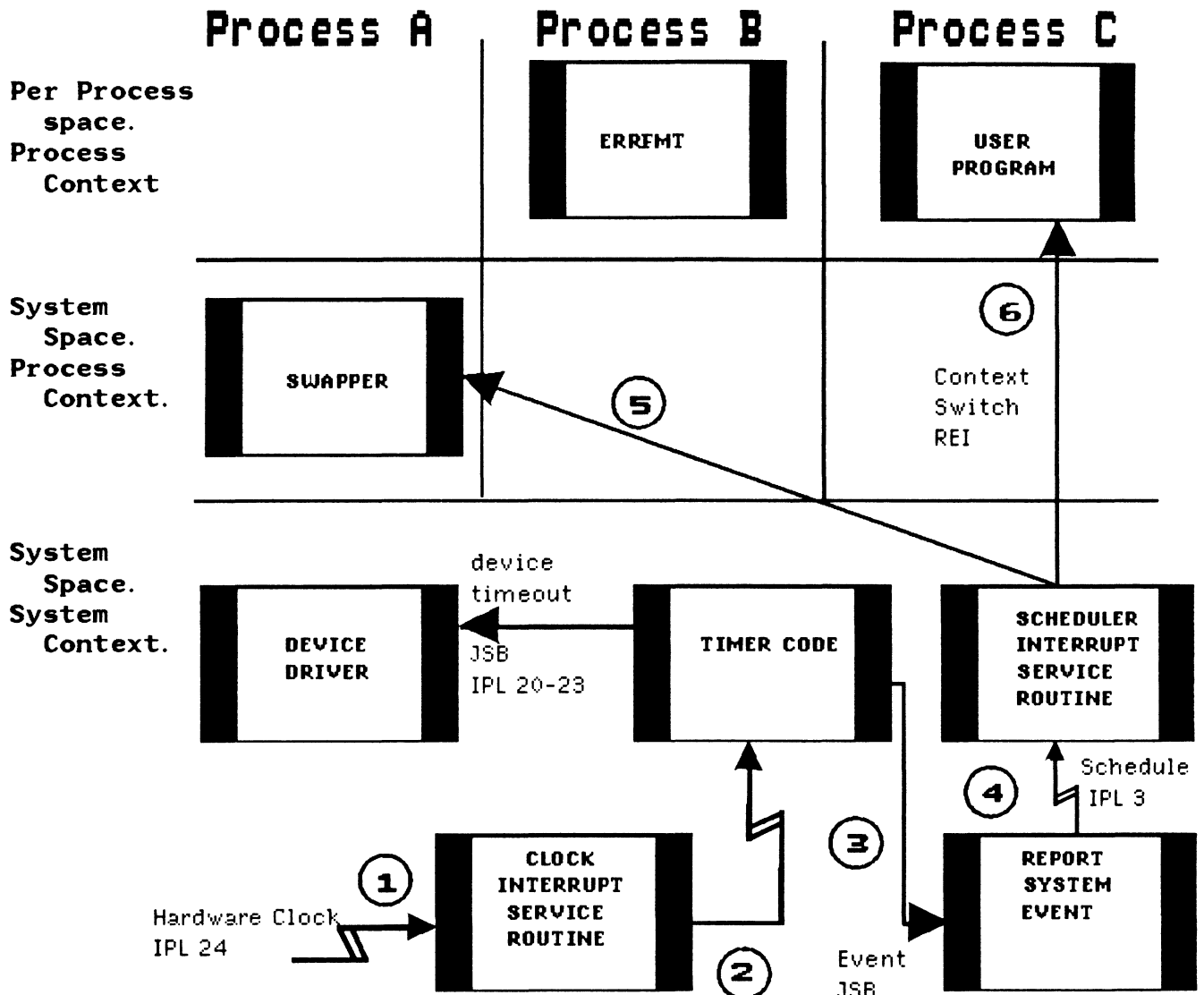
- Saves and restores context of process
- Updates system time
- Checks timer queue entries (TQES), quantum end
- Causes events to be processed

I/O Subsystem

- Reads/writes device
- Finishes I/O processing

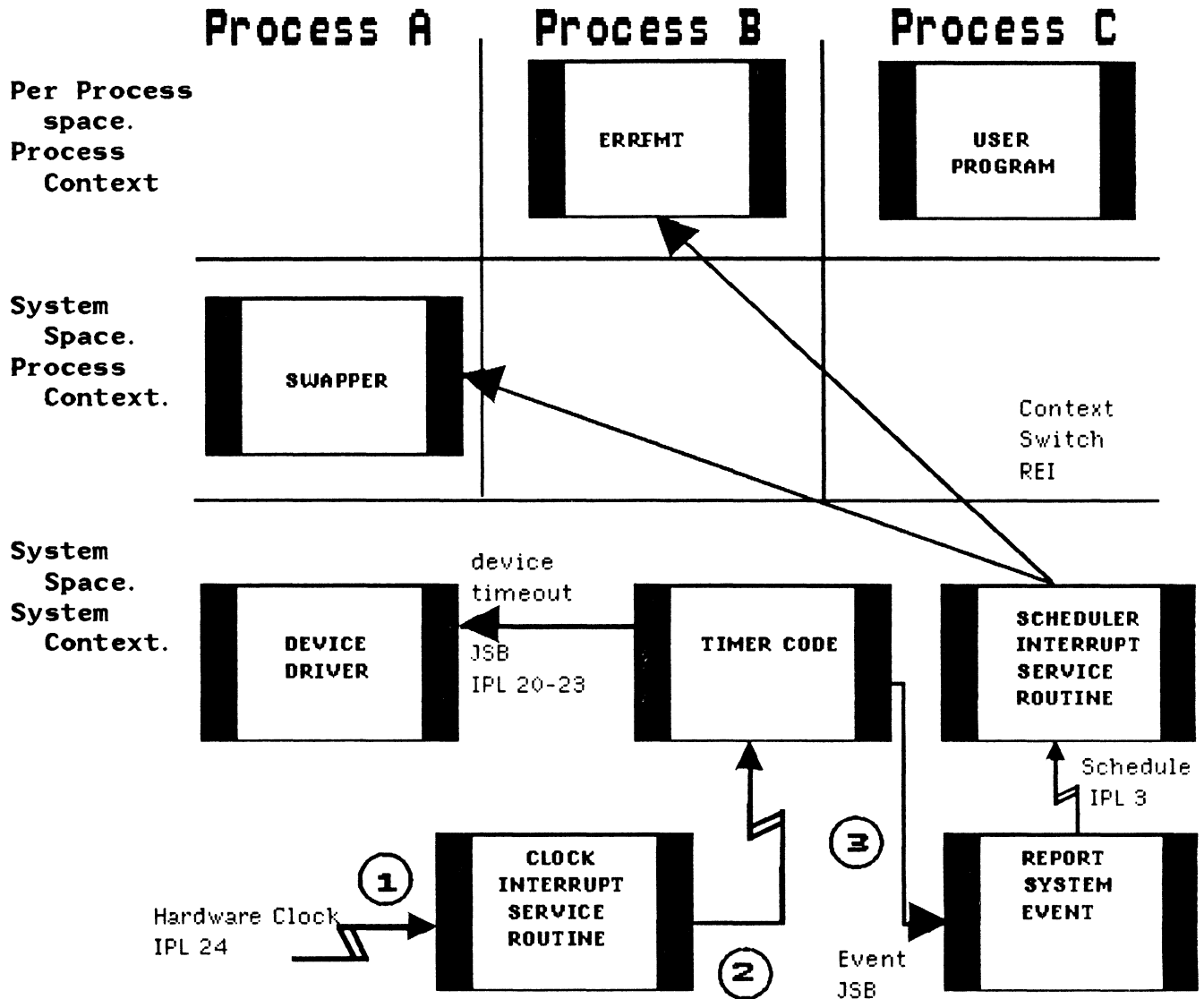
INTERACTION OF VMS COMPONENTS

Hardware Clock Interrupt



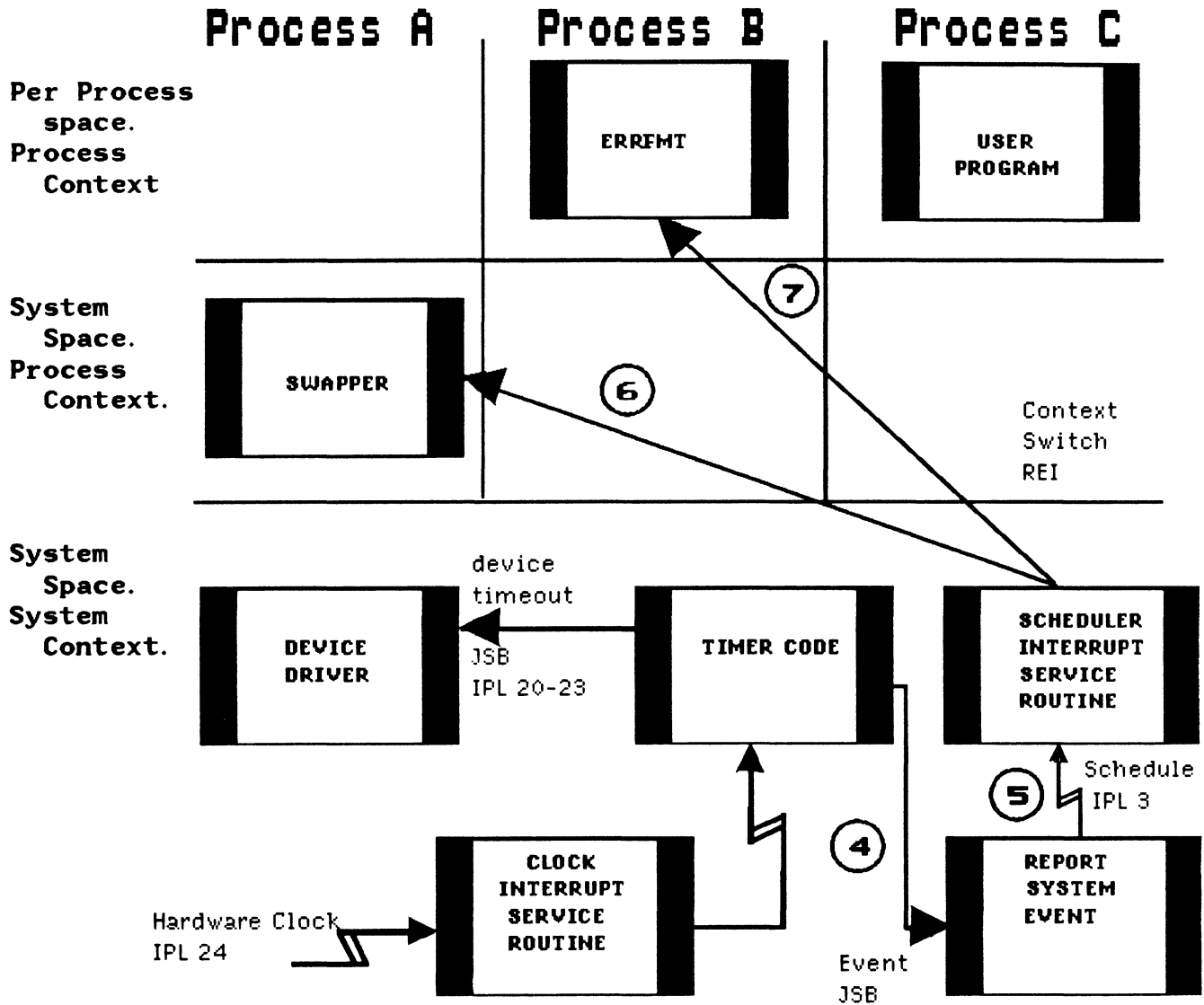
1. **Clock**
 - Updates system time and quantum field
 - Checks first timer queue entry
2. **Timer**
 - Checks for quantum end
 - Causes events to be processed
3. **Report System Event**
 - Changes process state
 - May request scheduler interrupt
4. **Scheduler**
 - Current <--> Computable
5. **Swapper**
 - Inswaps computable process
6. **Scheduled user program runs**

PERIODIC CHECK FOR DEVICE TIMEOUT



1. Hardware Clock Interrupt
2. Once every second, a timer queue entry becomes due that causes a system subroutine to execute.
3. This system subroutine checks for device timeouts, calls drivers to handle timeouts.

PERIODIC WAKE OF SWAPPER, ERROR LOGGER



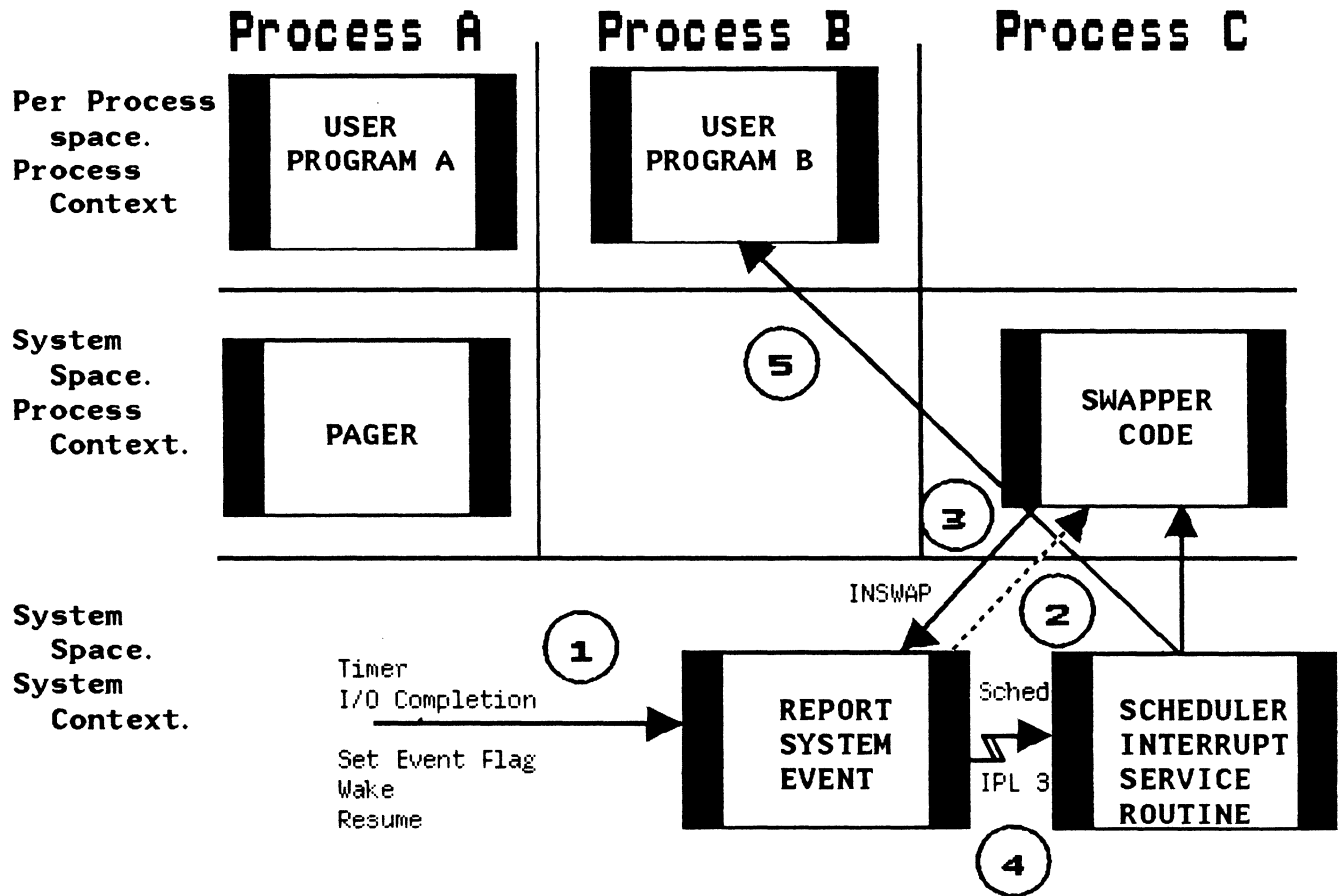
4. The same system subroutine can wake the swapper process and the error logger process.

5. Scheduler interrupt is requested.

6. } Swapper and error logger will eventually run.

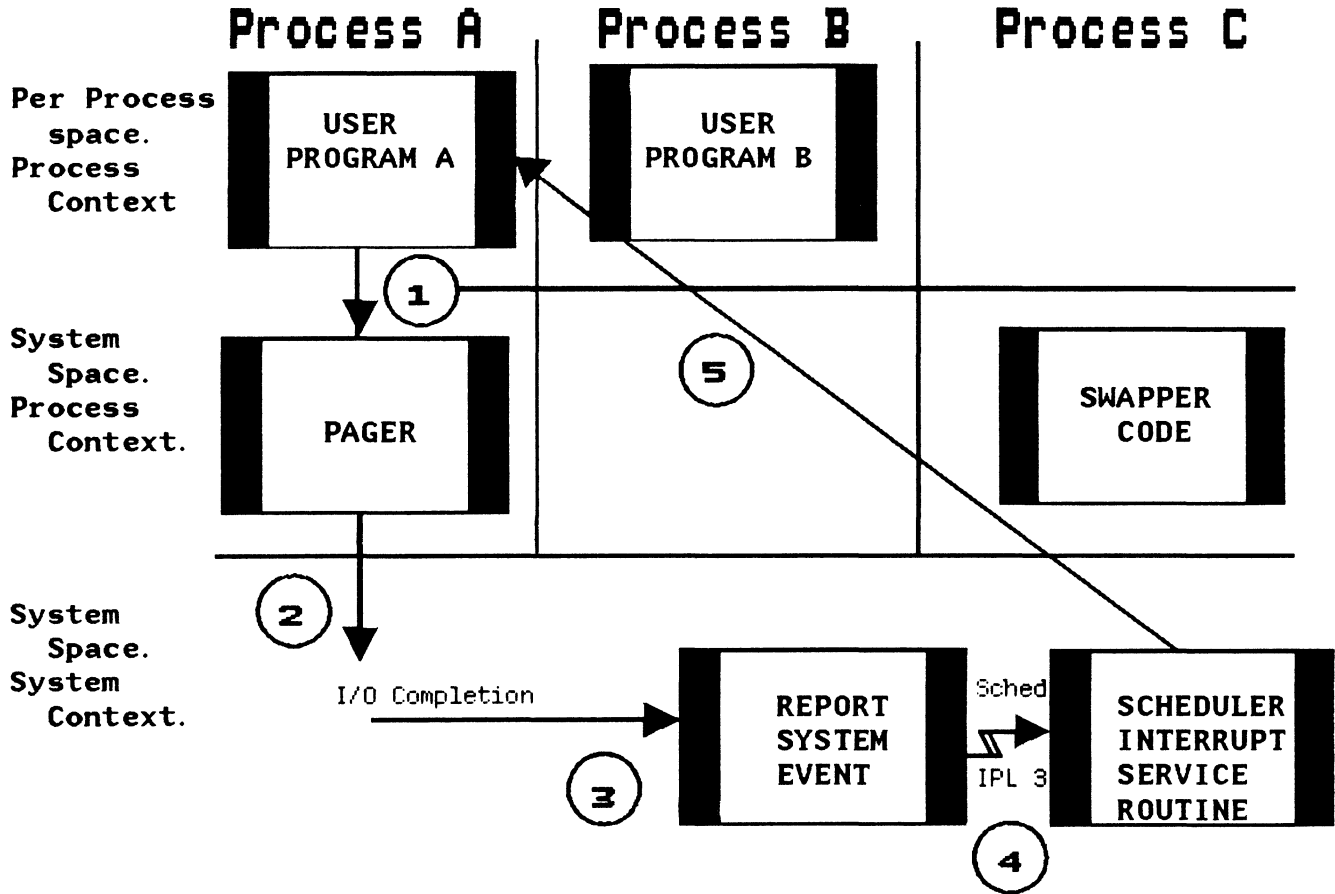
7.

SYSTEM EVENT REPORTING



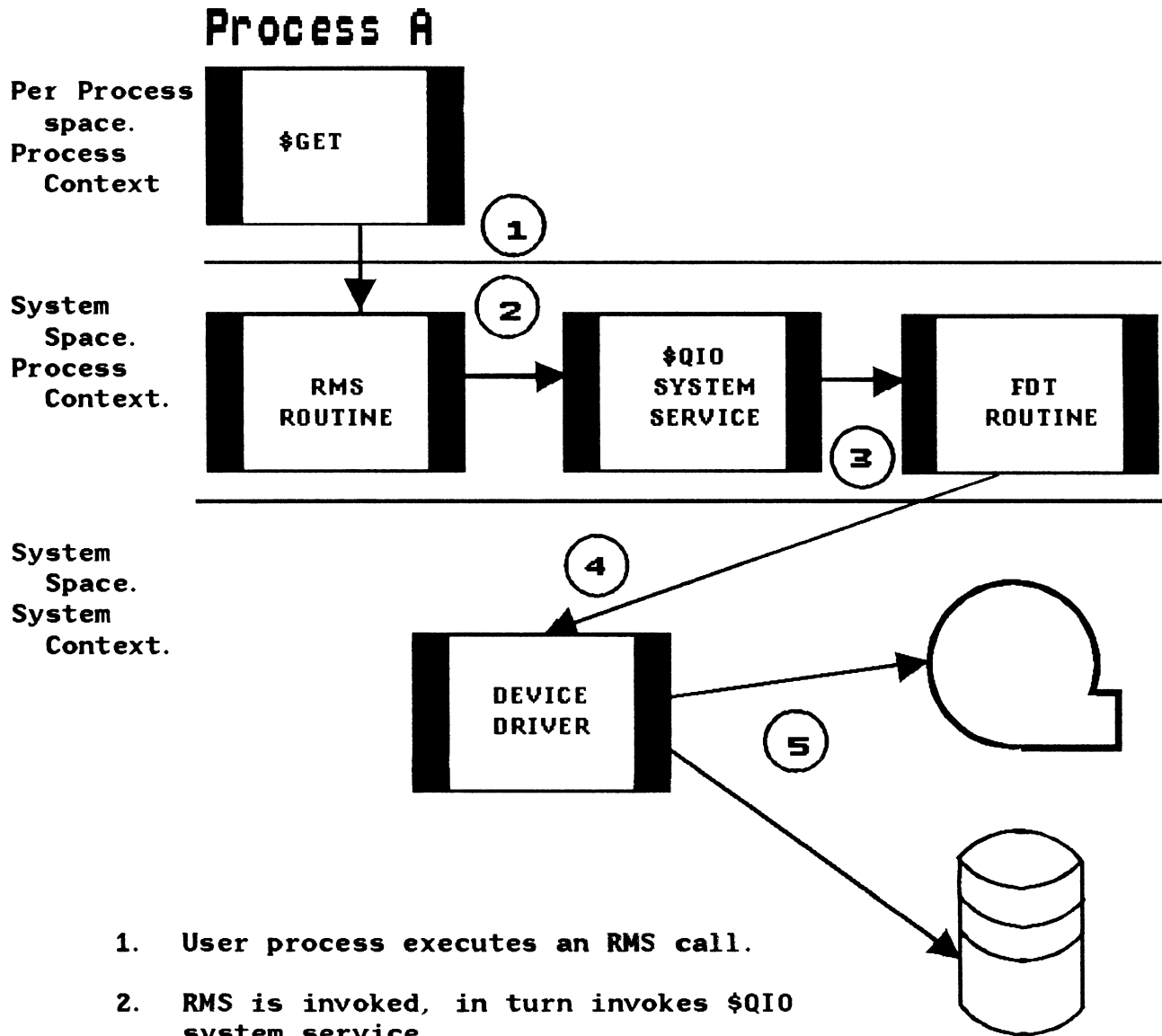
1. Event (Timer, I/O completion, etc) happens and is noted by RSE.
2. RSE sends a WAKE to the Swapper.
3. Swapper inswaps process.
4. RSE notes completion of inswap and requests Scheduler.
5. Process is scheduled to run.

PAGE FAULT HANDLING



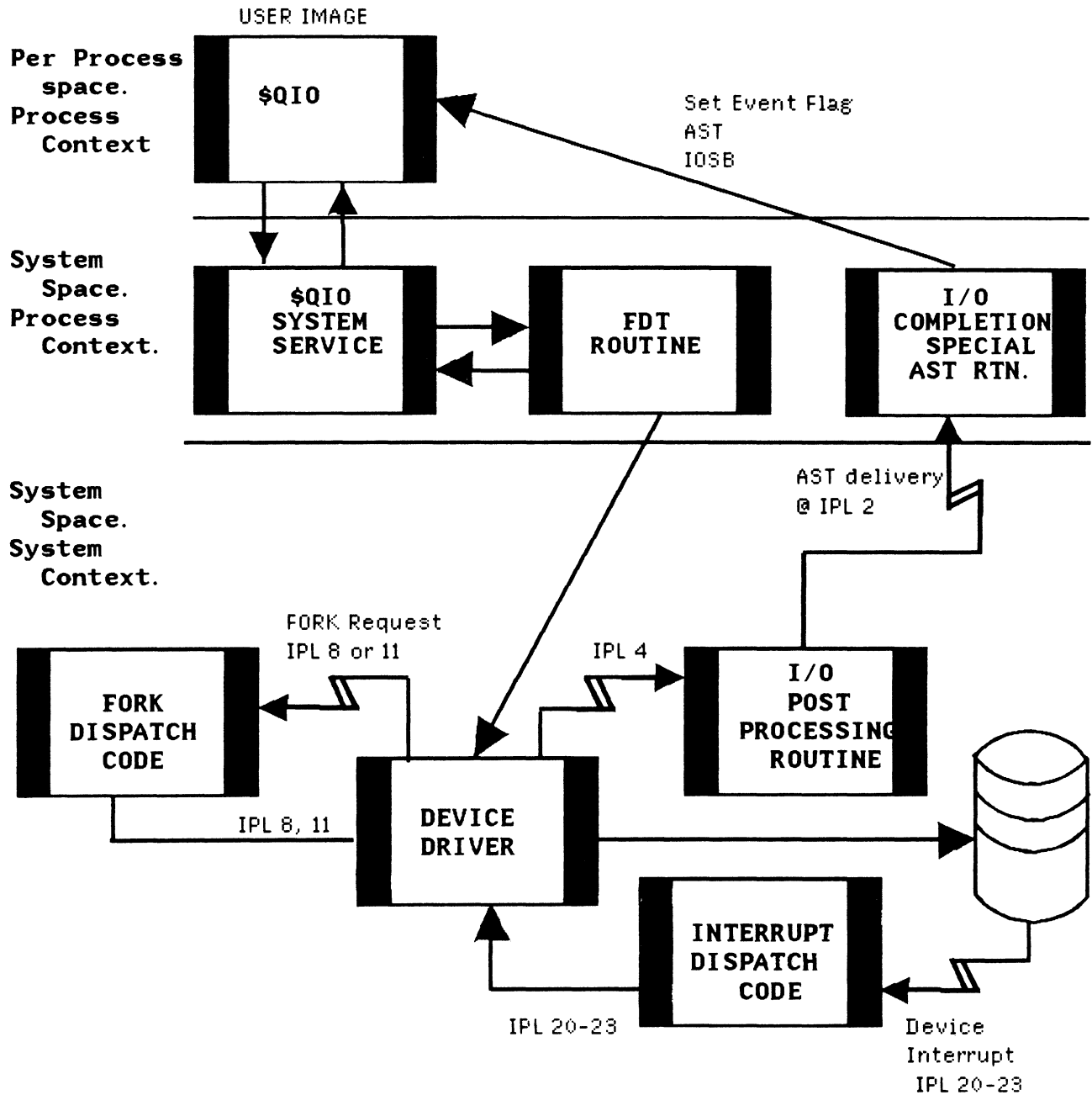
1. User process generates a "Translation not Valid" page fault.
2. Pager notes page faults and resolves.
3. Upon I/O completion for page fault, RSE is notified.
4. RSE requests scheduling interrupt for user process.
5. User process is made current and resumes.

DATA TRANSFER USING RMS

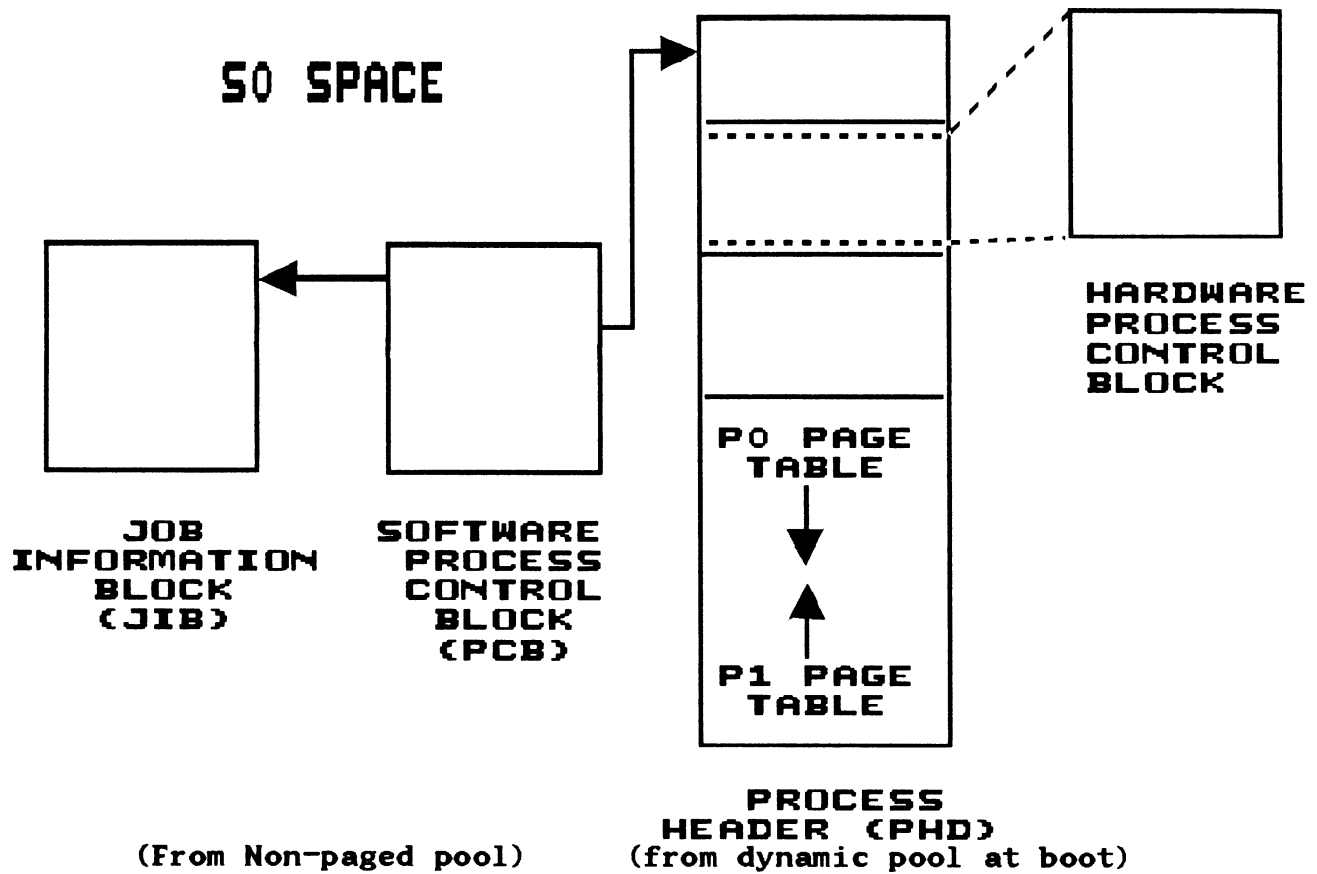


1. User process executes an RMS call.
2. RMS is invoked, in turn invokes \$QIO system service
3. \$QIO invokes FDT (Function Decision Table) routines (extensions to \$QIO).
4. FDT routine invokes the appropriate device driver.

DATA TRANSFER USING \$QIO

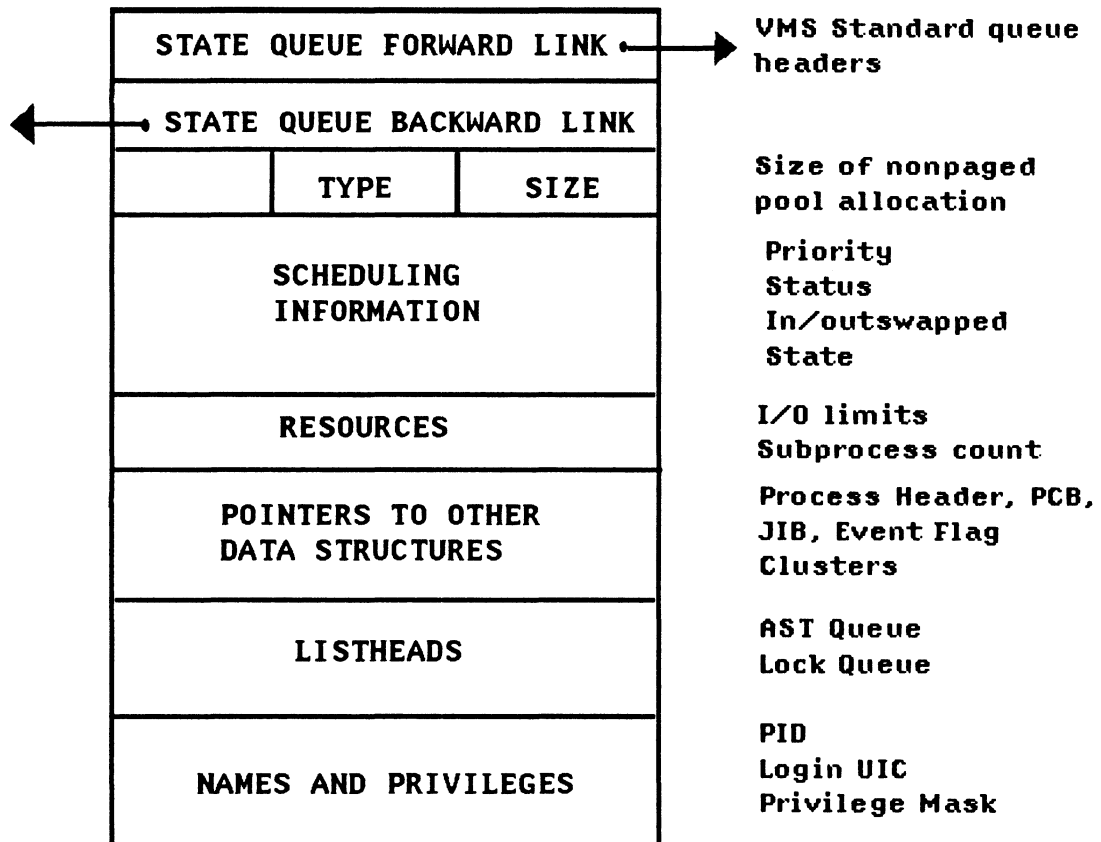


PROCESS DATA STRUCTURES OVERVIEW



- **Software Process Control Block (PCB)**
 - Holds process-specific data that must always be available (for example, process state, priority).
 - Contains pointers to other process data structures.
 - Not paged, Not swapped
- **Process Header (PHD)**
 - Contains process memory management information.
 - Contains hardware process control block.
 - May be outswapped
- **Hardware Process Control Block**
 - Contains saved hardware context
- **Job Information Block**
 - Keeps track of resources for a detached process and all subprocesses.

SOFTWARE PROCESS CONTROL BLOCK (PCB)



PROCESS HEADER (PHD) AND JOB INFORMATION BLOCK (JIB)

PHD

FIXED AREA
WORKING SET PAGE CATALOG
IMAGE FILE IMAGE SECTION LOCATION TABLE
VIRTUAL TO PHYSICAL ADDRESS MAPPING

Privilege Mask
Hardware Process Control
Block

Working Set list (WSL)

Process Section Table (PST)

P0 page table

P1 page table

JIB

STACK POINTERS
GENERAL PURPOSE REGISTERS
OTHER REGISTERS STATUS INFORMATION
MEMORY MANAGEMENT REGISTERS

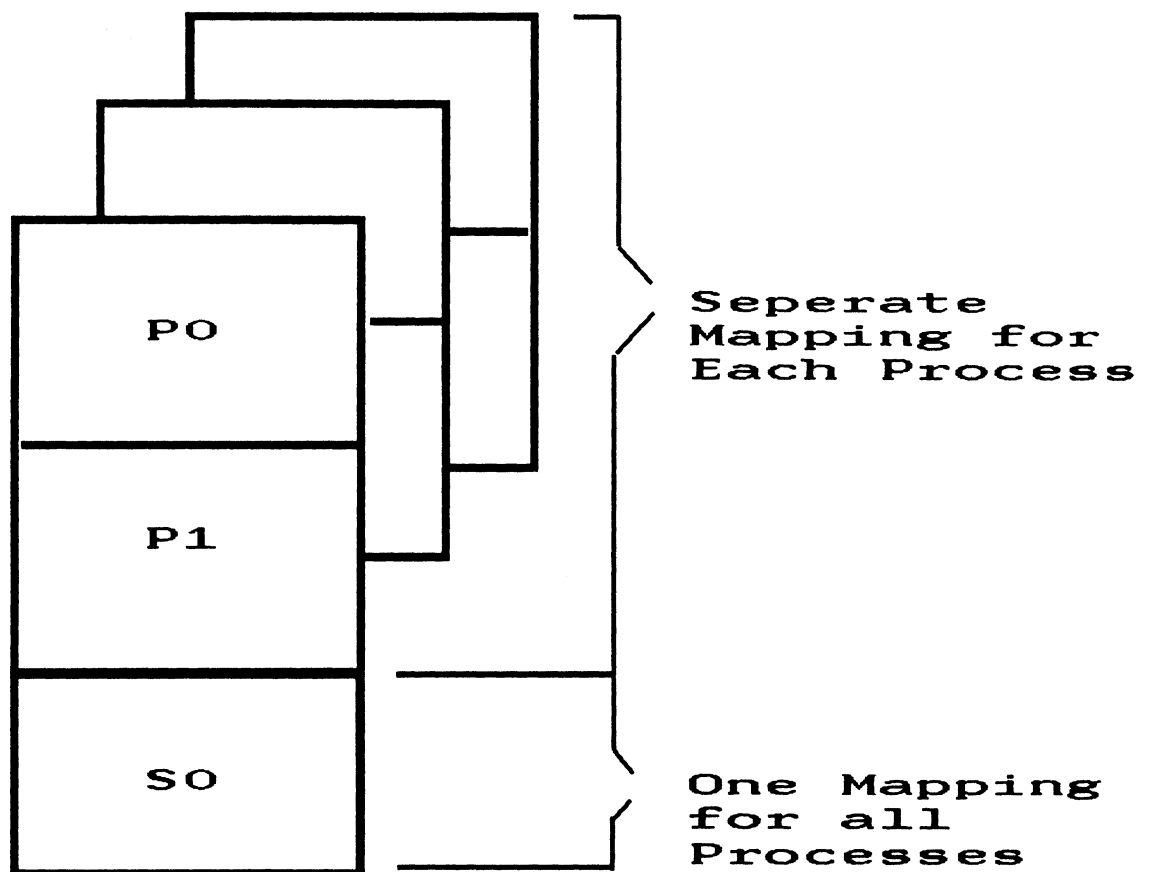
Kernel stack pointer
Executive stack pointer
Supervisor stack pointer
User stack pointer

R0, R1, ..., R11

Argument Pointer (AP)
Frame Pointer (FP)
Program Counter (PC)
Processor Status Longword (PSL)

P0, P1 Base Registers
P0, P1 Length Registers

VIRTUAL ADDRESS SPACE OVERVIEW



PROCESS VIRTUAL ADDRESS SPACE

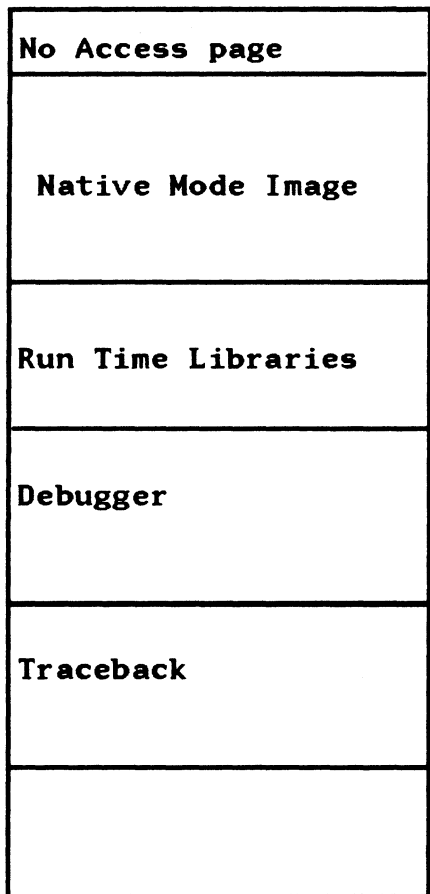
- P0 - Image, Run-time Library Debugger
- P1 - Command Language Interpreter, stacks, file system XQP, I/O data areas
- S0 - System services, RMS, other executive code and data

SO VIRTUAL ADDRESS SPACE CONTENTS

SYSTEM SERVICE VECTORS	System Service code, Scheduler, Report System Event (RSE)
EXECUTIVE CODE AND DATA	
FILE HANDLING ROUTINES	RMS.EXE, \$GET, \$PUT, etc.
ERROR MESSAGE TEXT	SYSMSG.EXE (pageable)
DESCRIPTION OF PAGES IN PHYSICAL MEMORY	PFN Database (used to map thru)
SHARED DYNAMIC DATA STRUCTURES	Paged Pool Global Section descriptors
SHARED DYNAMIC DATA STRUCTURES, DRIVERS	Non-paged Pool Software process control blocks Unit control blocks Lookaside lists I/O request packets Timer queue elements
STACK USED WHEN INTERRUPTS OCCUR	Interrupt stack
TABLE FOR VECTORING BY HARDWARE TO SERVICE ROUTINES	System Control Block (SCB)
PROCESS HEADER STORAGE	Balance Slots (balance set implementation)
VALID SYSTEM VIRTUAL ADDRESS LOCATIONS	System Header - System working set list - Global section table
PAGE LOCATION IN VA SPACE	System Page Table
GLOBAL PAGE LOCATIONS	Global Page Table

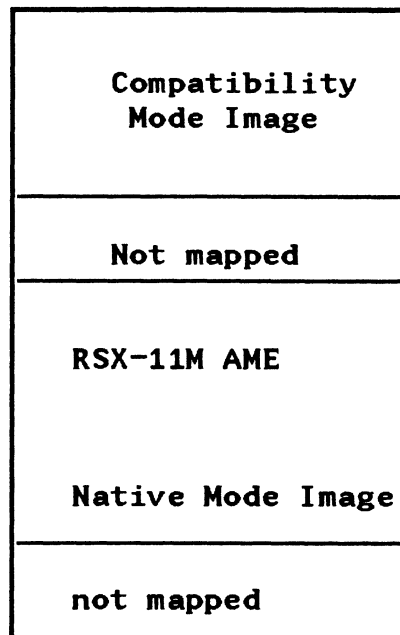
P0 VIRTUAL ADDRESS SPACE LAYOUT

Native Mode Image



0
1FF
POLR pages
3FFFFFFF

Compatibility Mode Image

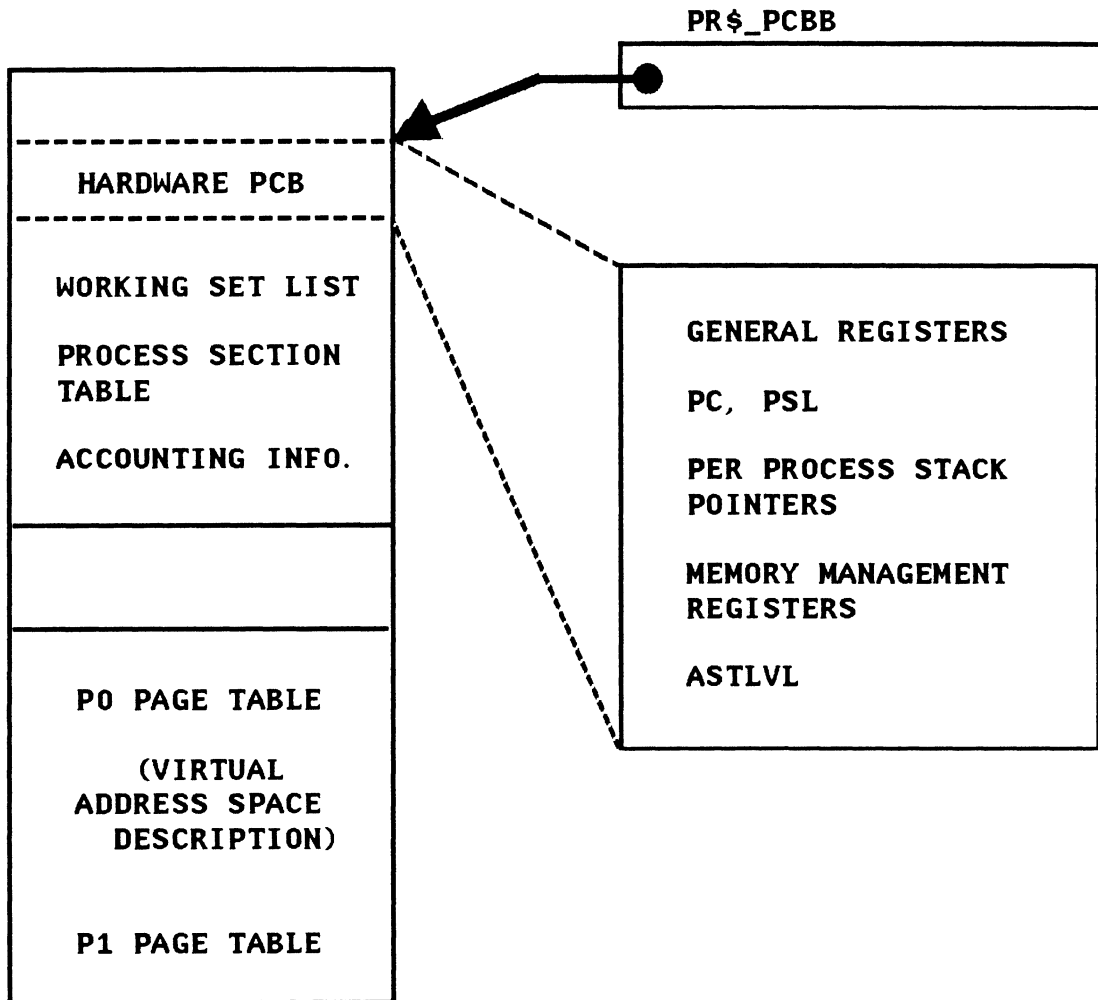


0
End of Comp. Mode image
177777 = FFFF
POLR pages
3FFFFFFF

P1 VIRTUAL ADDRESS SPACE LAYOUT

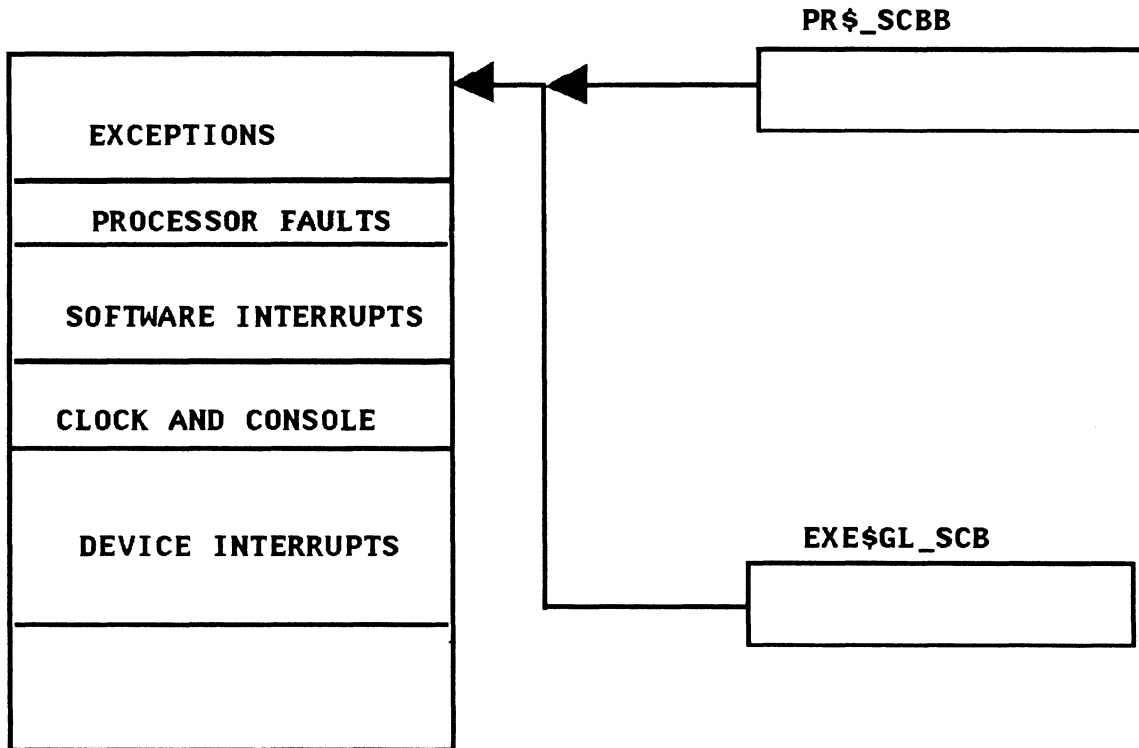
Image Speci- fic		40000000
-----	USER STACK	
	per Process Message Section(s)	CTL\$GL_CTLBASVA
Process	CLI Symbol Table	
	CLI Image	CTL\$AG_CLIMAGE
	Files - 11 XQP	CTL\$GL_F11BXQP
s p e c i f i c	Image I/O Segment	
	Process I/O Segment	PIO\$GW_PIOIMPA+ IMP\$L_IOSEGADDR
	Process Allocation Region	CTL\$GL_ALLOCREG
	Channel Control Block Table	
-----	P1 Window to Process Header	CTL\$GL_CCBBASE
Static	Process I/O Segment	CTL\$GL_FMLH
	Per Process Common Area	
	Compatibility Mode Data page	CTL\$GL_CMCNTX
	Security Auditing Impure Data	NSA\$T_IDT
	Image Activator Context	CTL\$GL_IAFLINK
	Generic CLI Data Pages	CTL\$AL_CLICALBK
	Image Activator Scratch Pages	
	Debugger Context	
	User System service/Msg vectors	CTL\$A_DISPVEC
	Image Header Buffer	MMG\$GL_IMGHDRBUF
	Kernel Stack	CTL\$AL_STACKLIM
	Executive Stack	
	Supervisor Stack	
	System Service Vectors	P1SYSVECTORS
	P1 Pointer Page	CTL\$GL_VECTORS
	Debugger Symbol Table	
		7FFFFFFF

HARDWARE CONTEXT



SYNCHRONIZING SYSTEM EVENTS

Hardware Interrupts and the SCB



SYSTEM CONTRTOL BLOCK

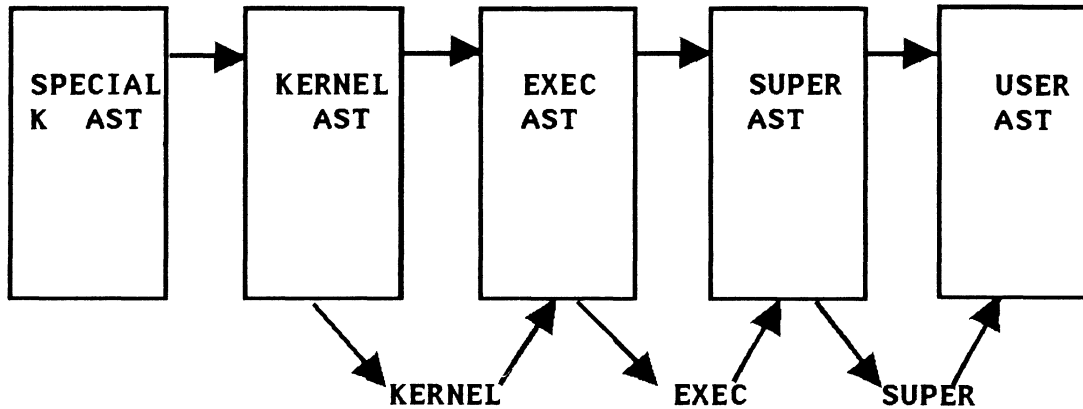
System Control Block (SCB) - physically contiguous area of system space

Hardware register PR\$SCBB contains physical address of the SCB

Hardware gets service routine address from longword in the SCB

Size of the SCB is CPU specific

AST DELIVERY



Delivery of an AST depends on:

- The current access mode of the process
- Whether the access mode of the AST is enabled
- Whether an AST is already active in the same access mode

Certain system ASTs have special precedence (special kernel ASTs)

- I/O completion
- \$GETJPI on another process

REI checks for deliverability of pending ASTs

Deliverability of ASTs is recorded in ASTLVL

ASTLVL contains

- Access mode of first deliverable AST in queue (for example, ASTLVL = 1 for executive mode AST)
- Or, the value 4 if:
 1. There are no ASTs in the queue
 2. AST delivery is disabled
 3. An AST is active in the same access mode

HARDWARE INTERRUPTS

FUNCTION	VALUE	NAME
POWER FAIL INTERRUPT	30	(None)
CLOCK INTERRUPTS	24	IPL\$HWCLK
DEVICE INTERRUPTS	20 - 23	UCB\$B_DIPL*

* Offset into Device's Unit Control Block (UCB)

- Interrupt Priority Levels (IPLs) above 15 reserved for Hardware Interrupts
- Peripheral devices interrupt at IPL 20 to 23
- IPL\$_xxxx - IPL level (see \$IPLDEF)

TO BLOCK	RAISE IPL TO	NAME
ALL INTERRUPTS	31	IPL\$_POWER
CLOCK INTERRUPTS	24	IPL\$_HWCLK
DEVICE INTERRUPTS	20 - 23	UCB\$B_DIPL*
ACCESS TO SCHEDULER'S DATA STRUCTURES	8	IPL\$_SYNCH
DELIVERY OF ASTs (prevent process deletion)	2	IPL\$_ASTDEL

* Offset into Device's Unit Control Block (UCB)

- Can use IPL to block interrupt servicing
- For example, to block AST delivery, raise to IPL\$_ASTDEL
- IPL\$_SYNCH used to coordinate access to the scheduler's database

SOFTWARE INTERRUPTS AND IPL

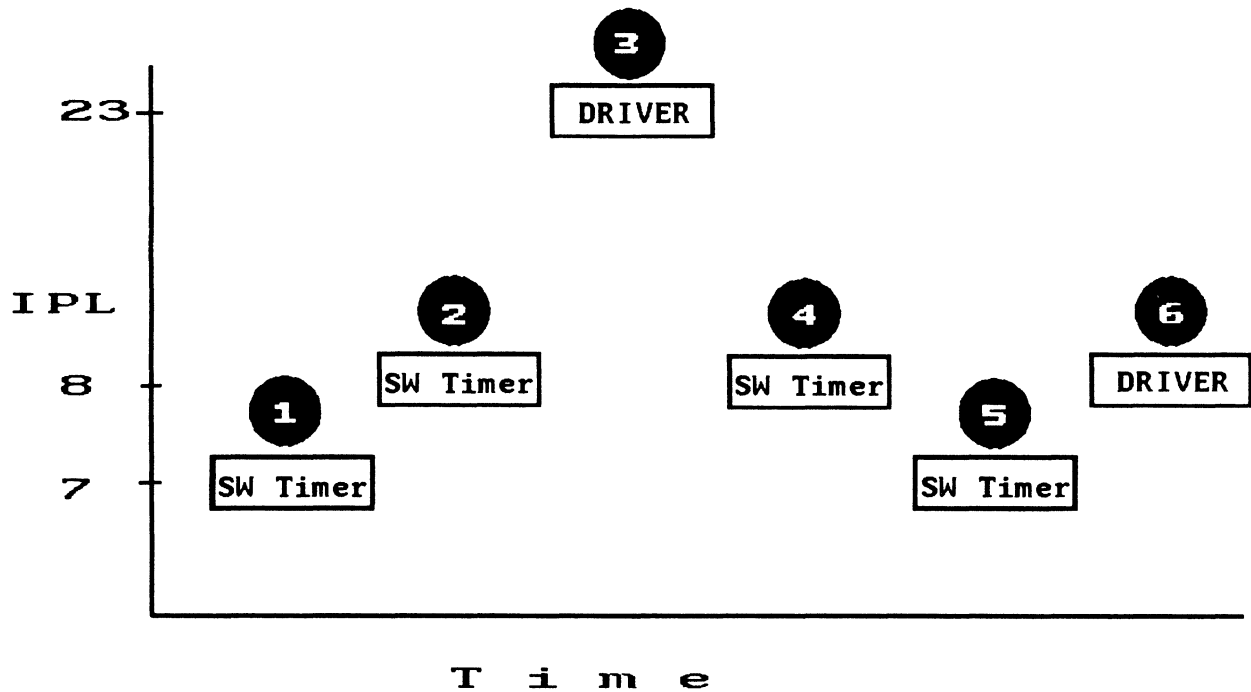
FUNCTION	VALUE	NAME
(unused)	15 - 12	
FORK DISPATCHING	11	IPL\$_MAILBOX
FORK DISPATCHING	10	
FORK DISPATCHING	9	
FORK DISPATCHING	8	IPL\$_TIMER
SOFTWARE TIMER INTERRUPT	7	IPL\$_SYNCH
FORK DISPATCHING	6	IPL\$_TIMERFORK
USED TO ENTER XDELTA	5	
I/O POST-PROCESSING	4	IPL\$_IOPOST

RESCHEDULING INTERRUPT	3	IPL\$_SCHED
AST DELIVERY INTERRUPT	2	IPL\$_ASTDEL
(unused)	1 - 0	

Interrupt Priority Levels (IPLs) 1 through 15 are reserved for software interrupts.

Driver fork level stored at offset UCB\$_FIPL in UCB (see \$UCBDEF)

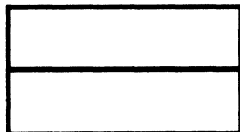
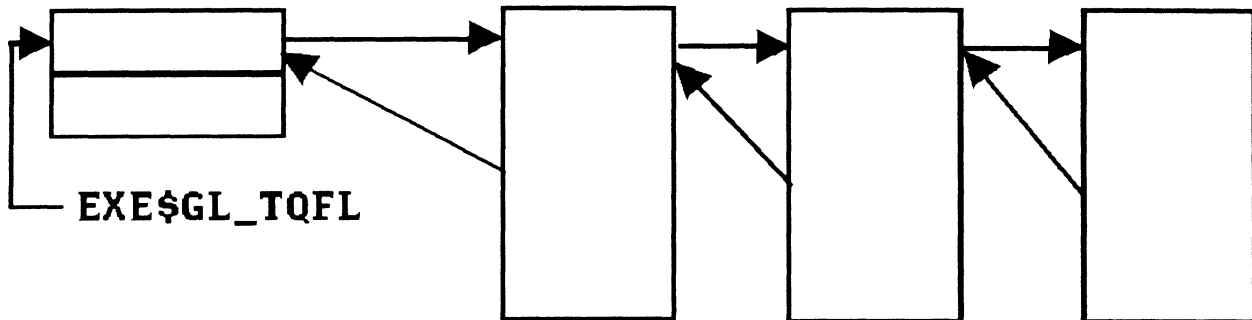
USING IPL to SYNCHRONIZE SYSTEM ROUTINES



1. Software timer invoked at `IPL$_TIMERFORK` (IPL 7)
2. Software timer raises to `IPL$_SYNCH` (IPL 8) to synchronize
3. Device interrupt - driver code at IPL 23
Driver requests interrupt at IPL 8 and issues an REI
4. Software timer resumes at `IPL$_SYNCH`
5. Software timer lowers IPL back to `IPL$_TIMERFORK`
6. Driver code executes at IPL 8

CLOCKS AND TIMER SERVICES

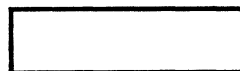
Timer Queue (ordered by expiration)



EXE\$GQ_SYSTIME
(10 ms)



PRxxx\$_TODR
(xxx=number associated with processor type)

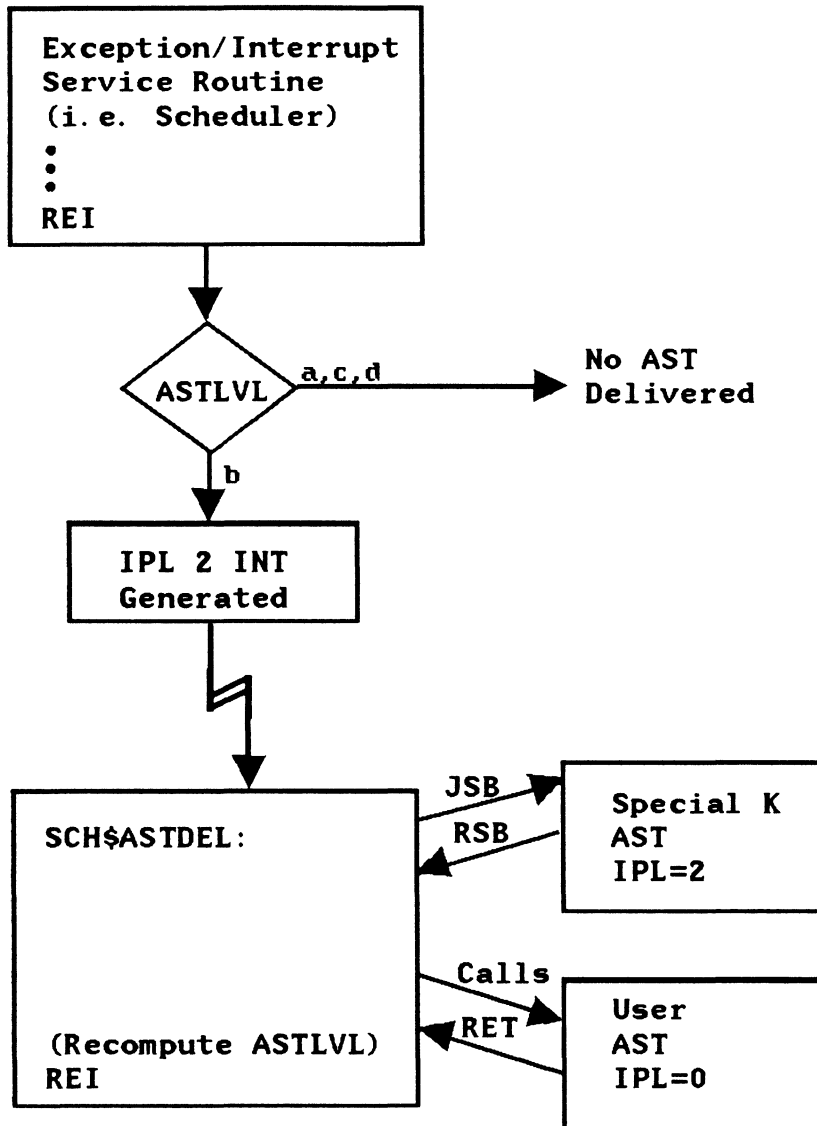


PRxxx\$_NICR (Next Interval Count)



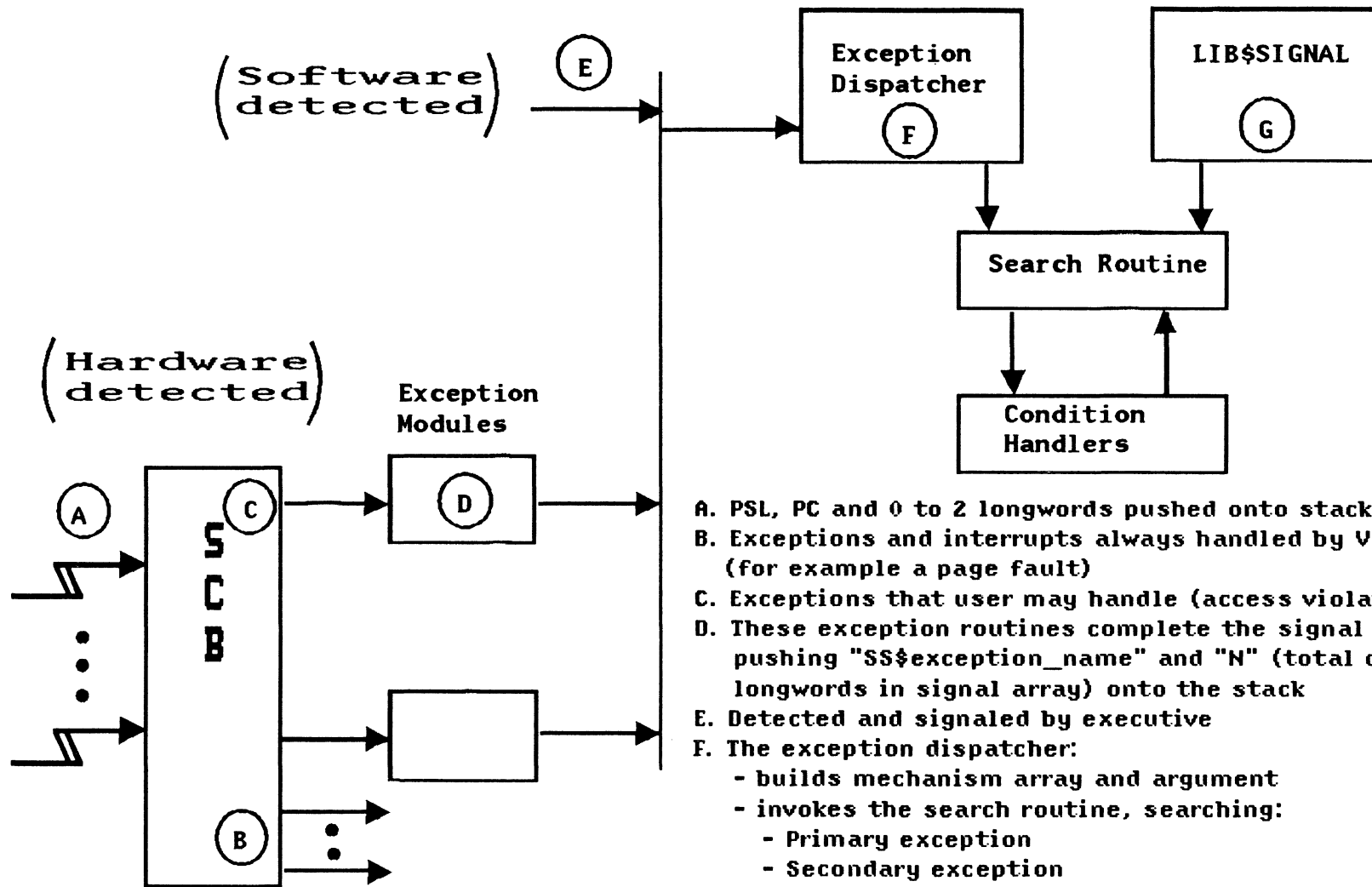
PRxxx\$_ICR (Interval Count)

AST DELIVERY SEQUENCE



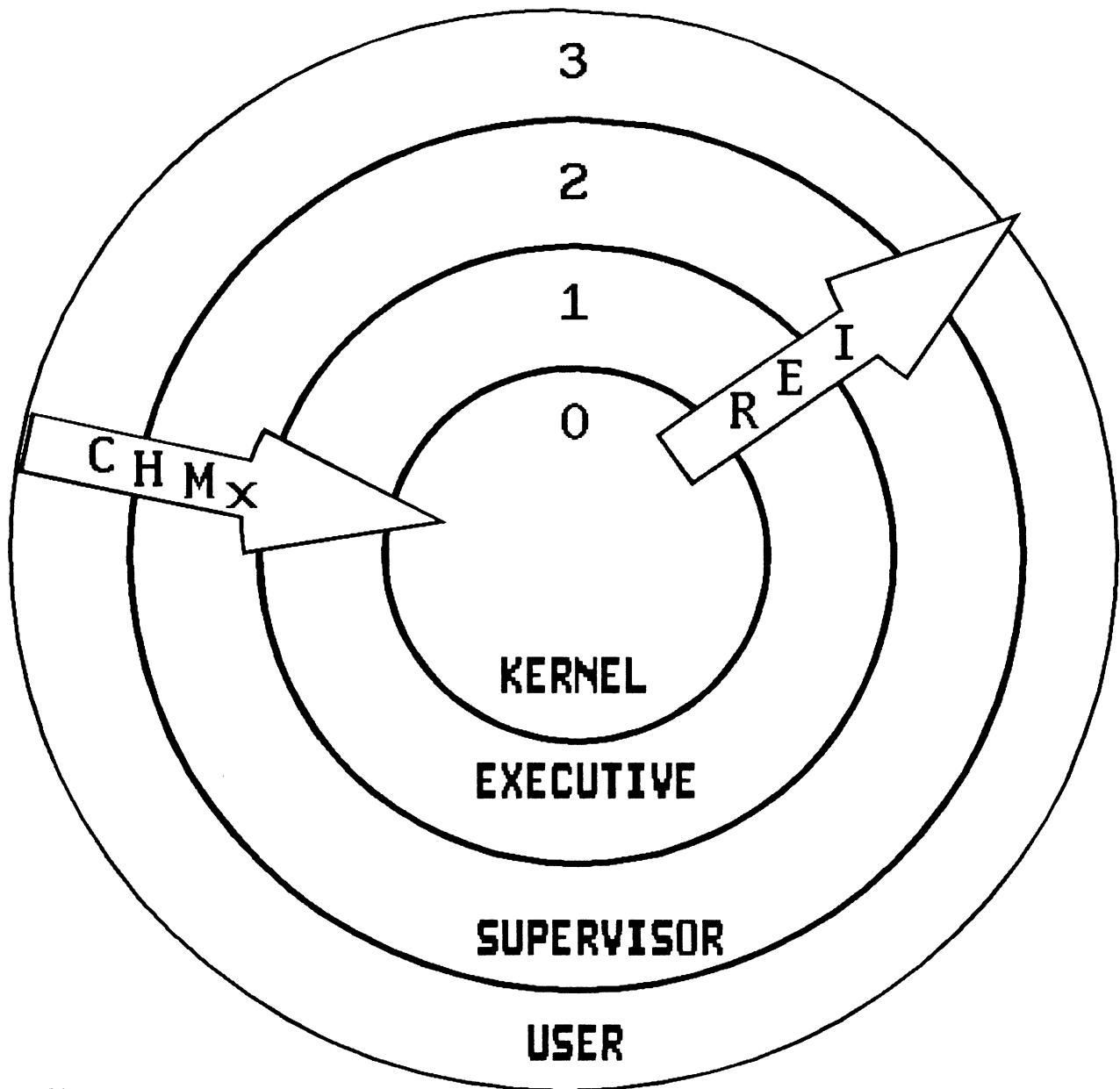
Rule	Example
a) $ASTLVL > \text{new access mode}$	User AST (3) > kernel access mode (3)
b) $ASTLVL \leq \text{new access mode}$	Super AST (2) \leq super access mode (2)
c) Interrupt stact active	(IS) bit set in PSL
d) Final IPL \geq 2	Process code a elevated IPL(\geq 2)

EXCEPTION AND INTERRUPT DISPATCHING



- A. PSL, PC and 0 to 2 longwords pushed onto stack
- B. Exceptions and interrupts always handled by VMS (for example a page fault)
- C. Exceptions that user may handle (access violation)
- D. These exception routines complete the signal array by pushing "SS\$exception_name" and "N" (total of longwords in signal array) onto the stack
- E. Detected and signaled by executive
- F. The exception dispatcher:
 - builds mechanism array and argument
 - invokes the search routine, searching:
 - Primary exception
 - Secondary exception
 - Call frames
 - Last Chance
- G. Alternate Condition handling mechanism
 - Signalled by RTL or user calling LIB\$LIBRARY
 - Search routine in same order as F above.

ACCESS MODE TRANSITIONS



CHM_x:

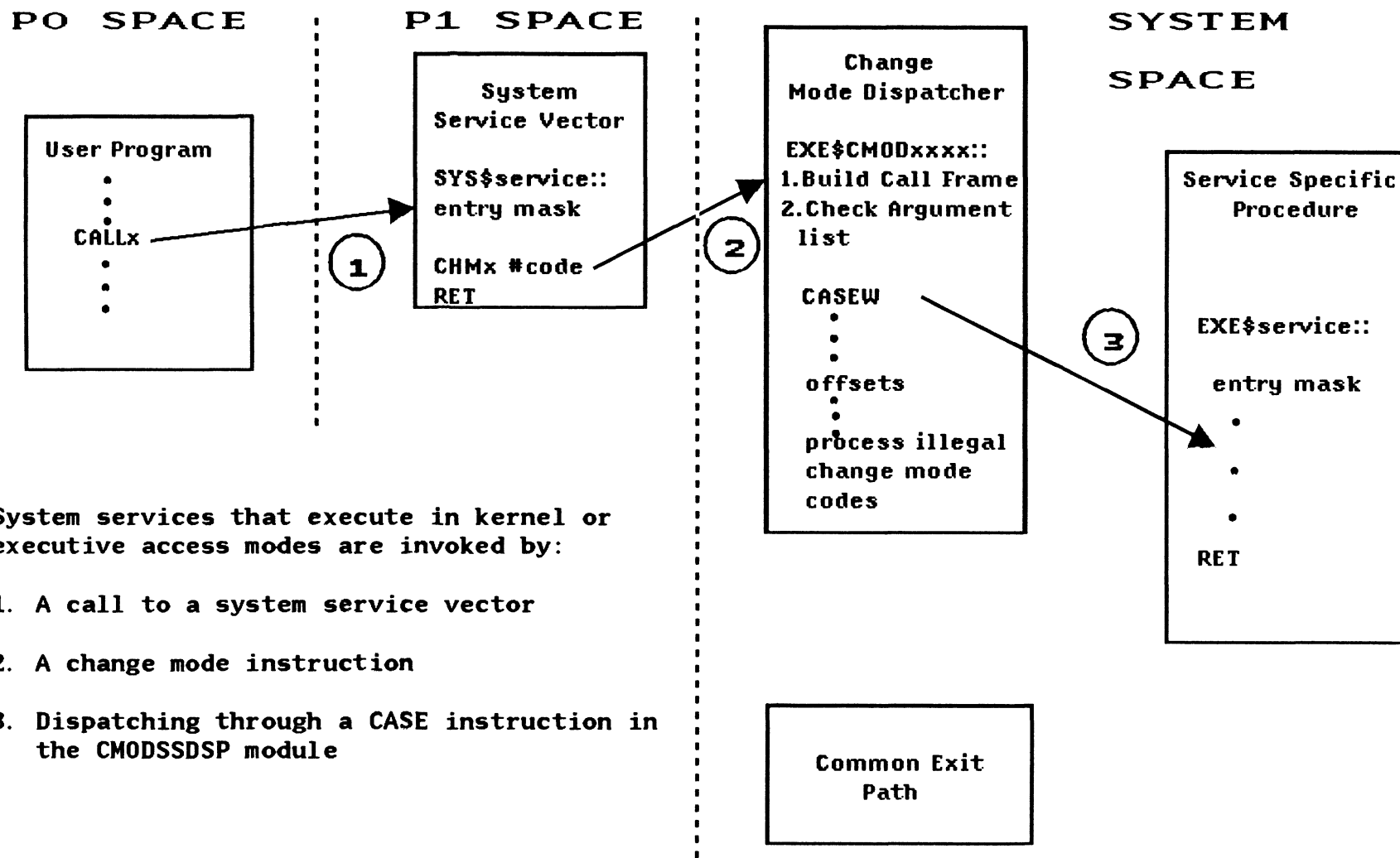
Only way to move from less privileged to more privileged access modes

REI:

Only way to move from more privileged to less privileged access modes

Checks for illegal or unauthorized transitions

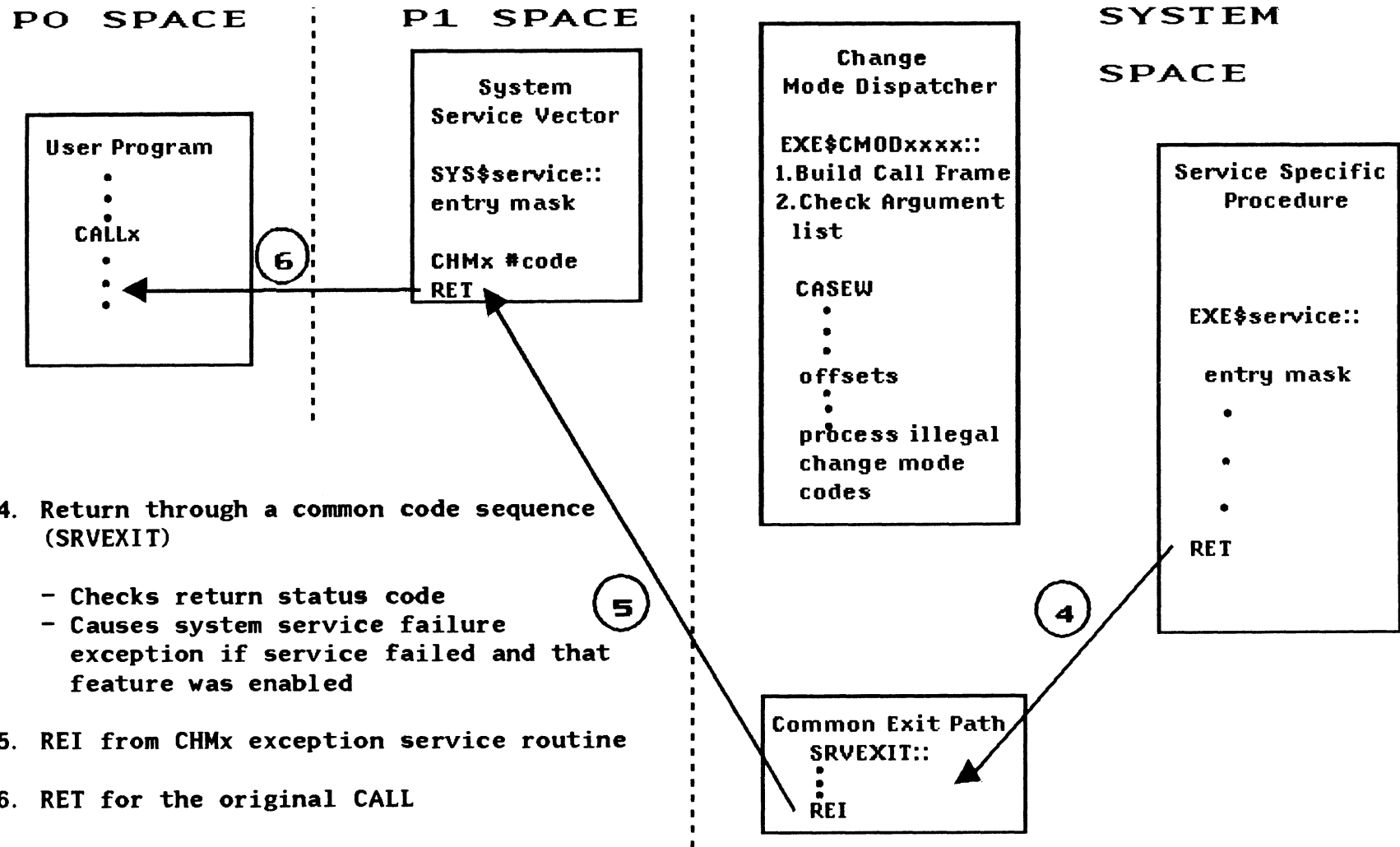
PATH TO SYSTEM SERVICES



System services that execute in kernel or executive access modes are invoked by:

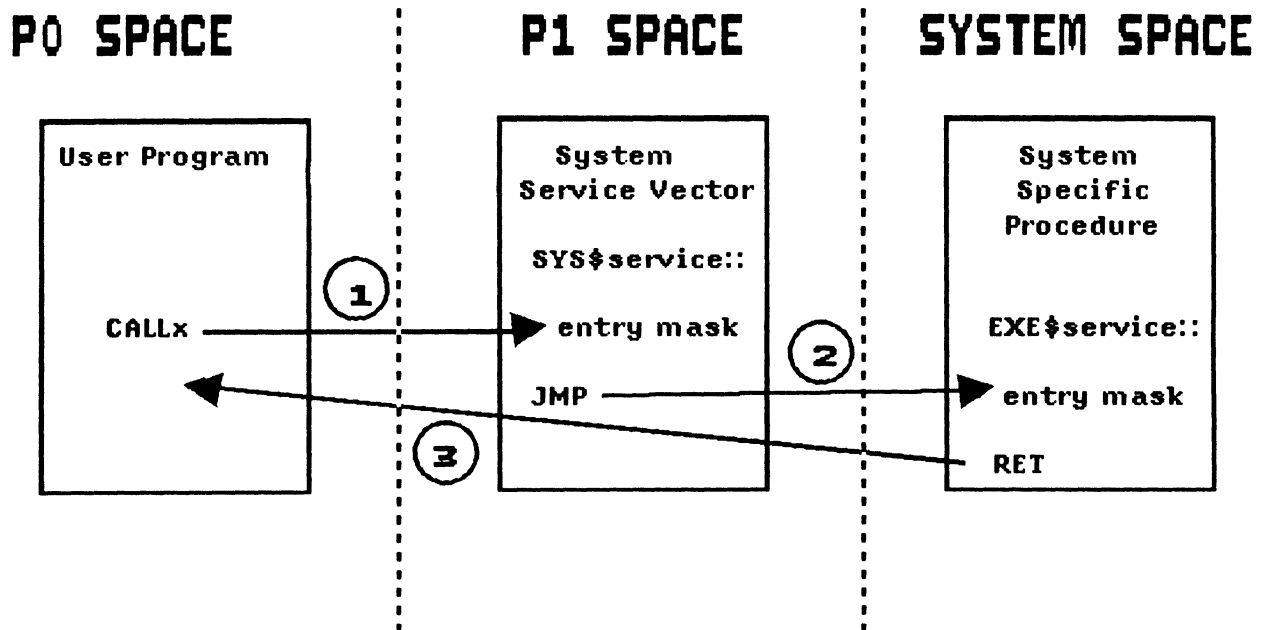
1. A call to a system service vector
2. A change mode instruction
3. Dispatching through a CASE instruction in the CMODSSDSP module

RETURN FROM SYSTEM SERVICE



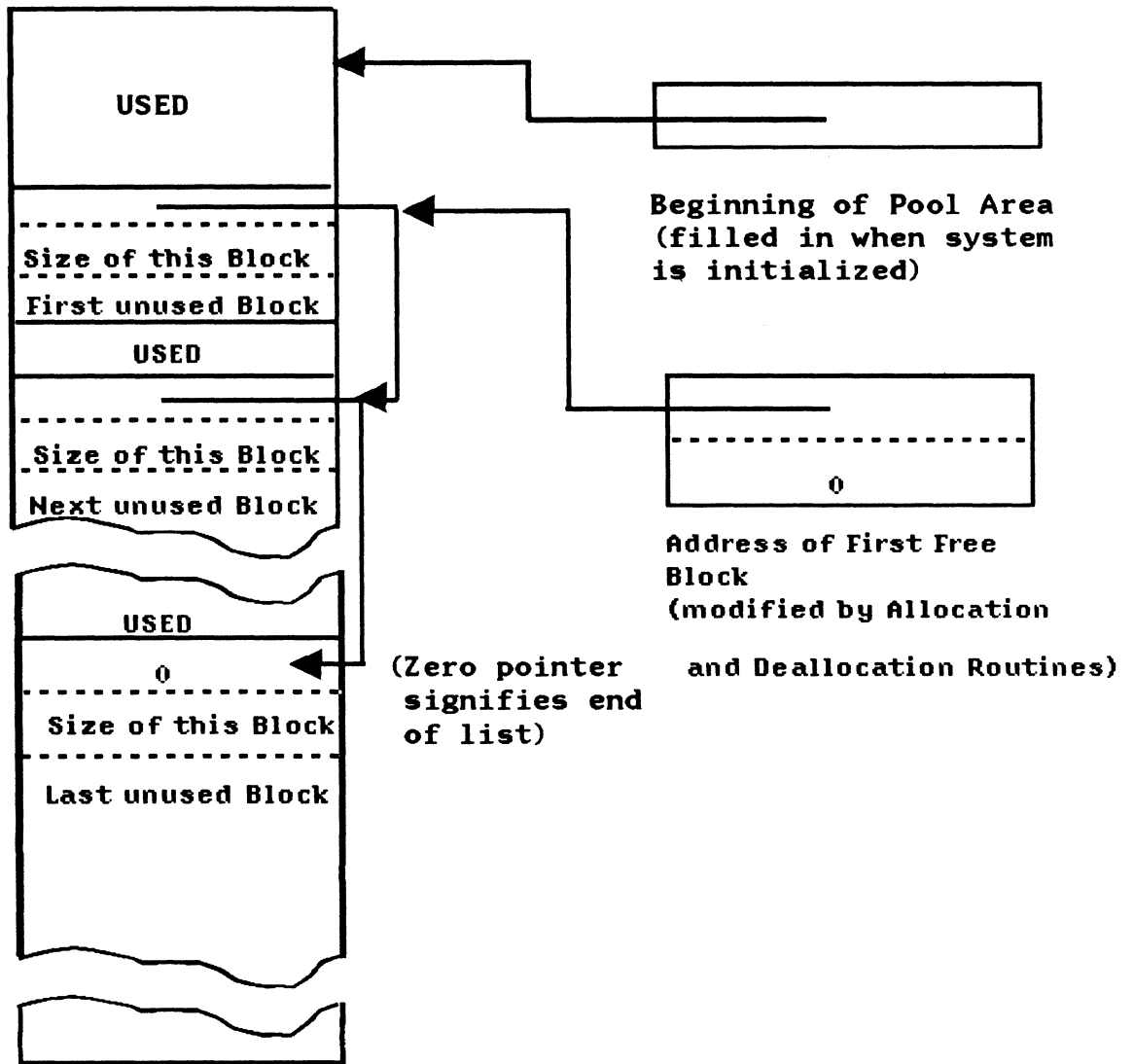
4. Return through a common code sequence (SRVEXIT)
 - Checks return status code
 - Causes system service failure exception if service failed and that feature was enabled
5. REI from CHMx exception service routine
6. RET for the original CALL

NONPRIVILEGED SYSTEM SERVICE



1. Invoked with a CALL statement.
2. System services that do not require a change of access mode have a simpler control passing sequence.
 - \$FA0
 - Timer Conversion Services
3. These services are not checked by SRVEXIT for error status codes.

DYNAMIC MEMORY

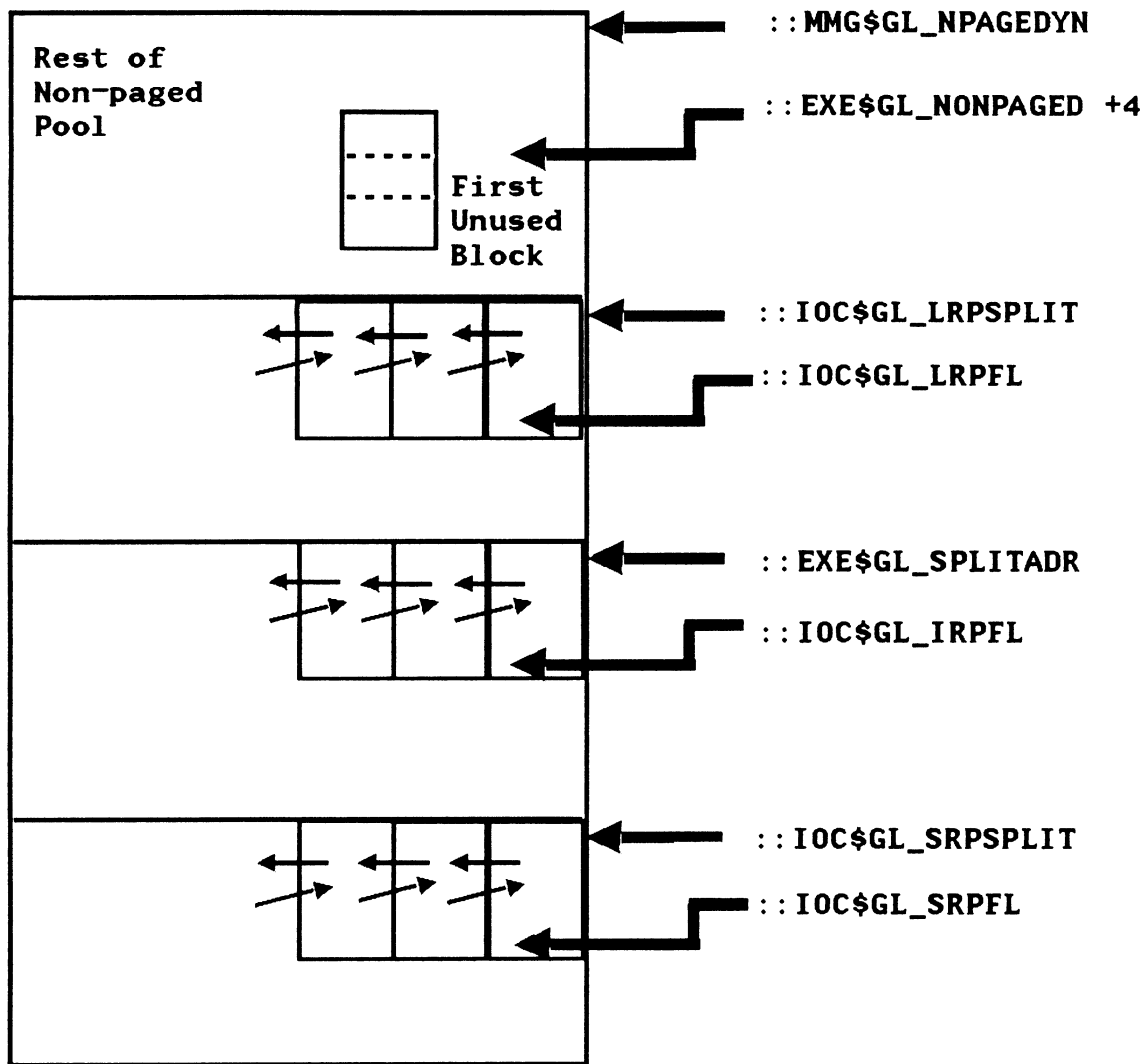


Used for the management of data structures that must be allocated and deallocated after the system or process is initialized.

Free blocks are stored in order of ascending addresses.

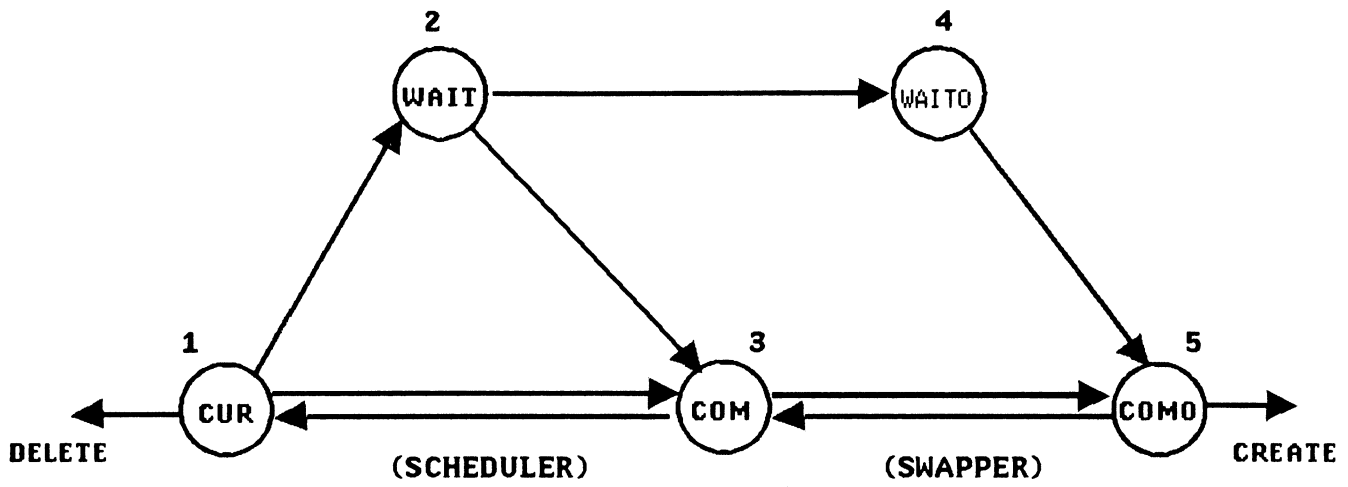
Number of bytes allocated for paged pool determined by SYSGEN parameter PAGEDYN.

ALLOCATING NONPAGED POOL



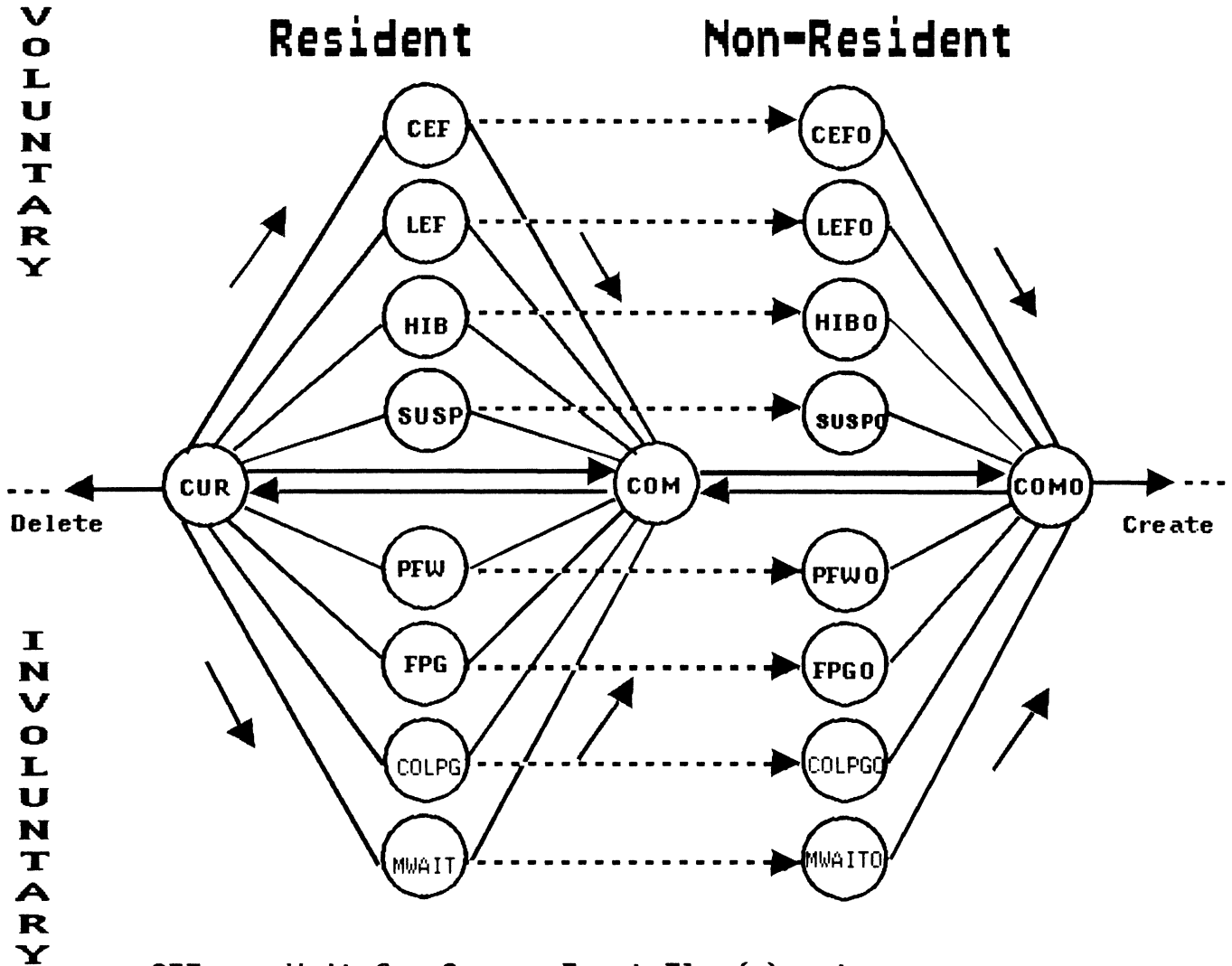
The above are examples of several preallocated nonpaged pool data structures and their associated listheads.

THE PROCESS STATES



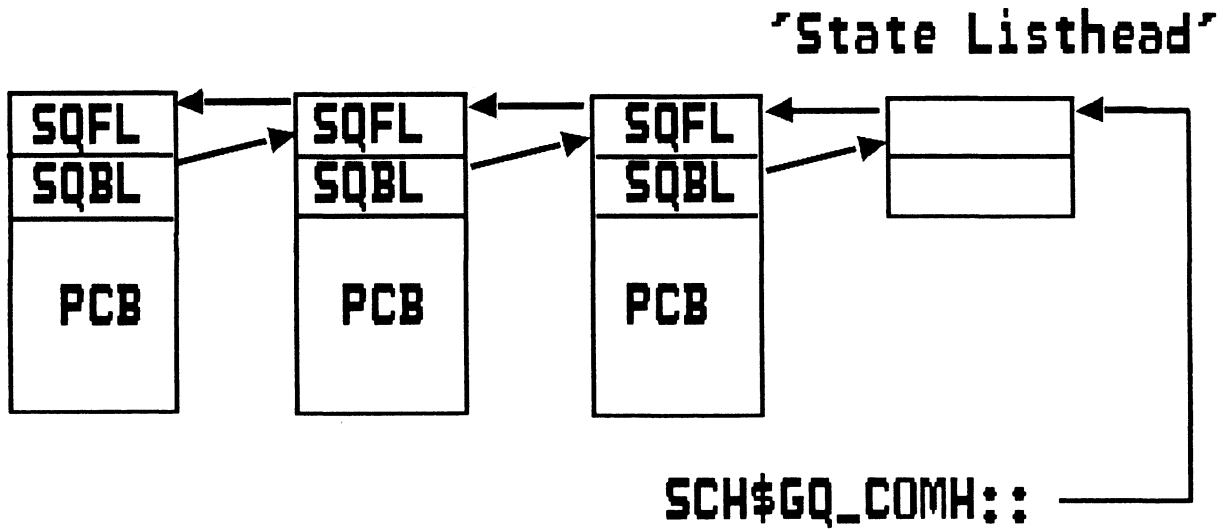
1. **CURRENT - executing**
2. **WAIT - removed from execution to wait for event completion**
3. **COMPUTABLE - ready to execute**
4. **WAIT OUTSWAPPED**
5. **COMPUTABLE OUTSWAPPED**

PROCESS WAIT STATES



- CEF - Wait for Common Event Flag(s) set
- LEF - Wait for Local Event Flag(s) set
- HIB - Hibernate until wake-up
- SUSP - Suspended until resumed
- >COM - Removed from execution at quantum end or preempted
- PFW - Page read in progress
- FPG - Wait for free page availability
- COLPG - Wait for shared page to be read in by another process
- MWAIT - Wait for miscellaneous resources or mutex
- Delete - Process has been logged out or deleted

HOW PROCESS STATES ARE IMPLEMENTED



The state of a process is defined by:

- The value in the PCB\$W_STATE field
- The PCB being in the corresponding state queue

State queues are circular

The current state is not implemented as a queue

- Just a longword pointer (SCH\$GL_CURPCB)
- Queue structure not necessary because only one process in the current state

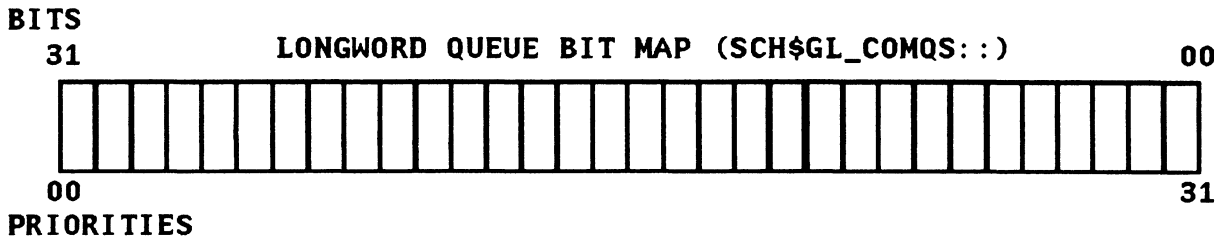
VAX instructions for manipulating queues:

- INSQUE new_entry, predecessor
- REMQUE out_entry, return_address

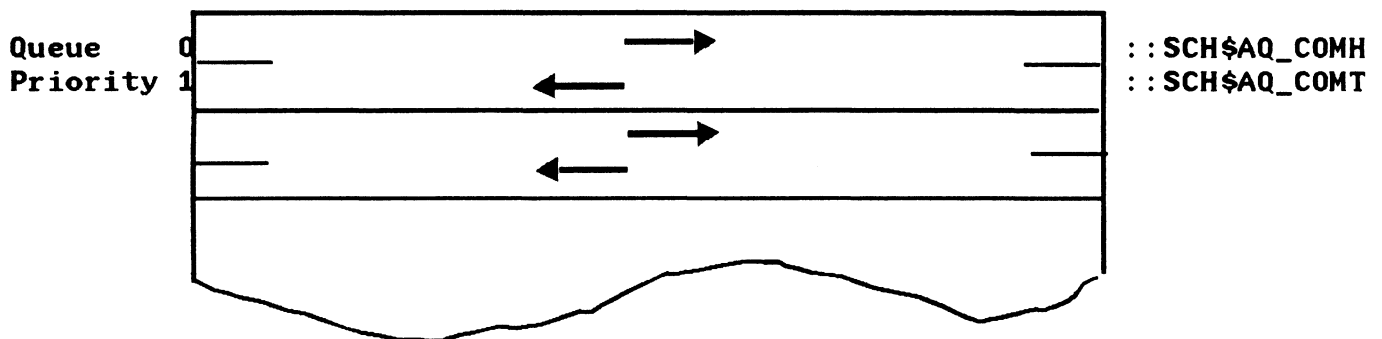
IMPLEMENTATION OF COM AND COMO STATES

BITMAP (1 EACH FOR COM, COMO)

FOR STATE COM



LISTHEADS (32 each for COM, COMO)



COM state implemented as a collection of queues

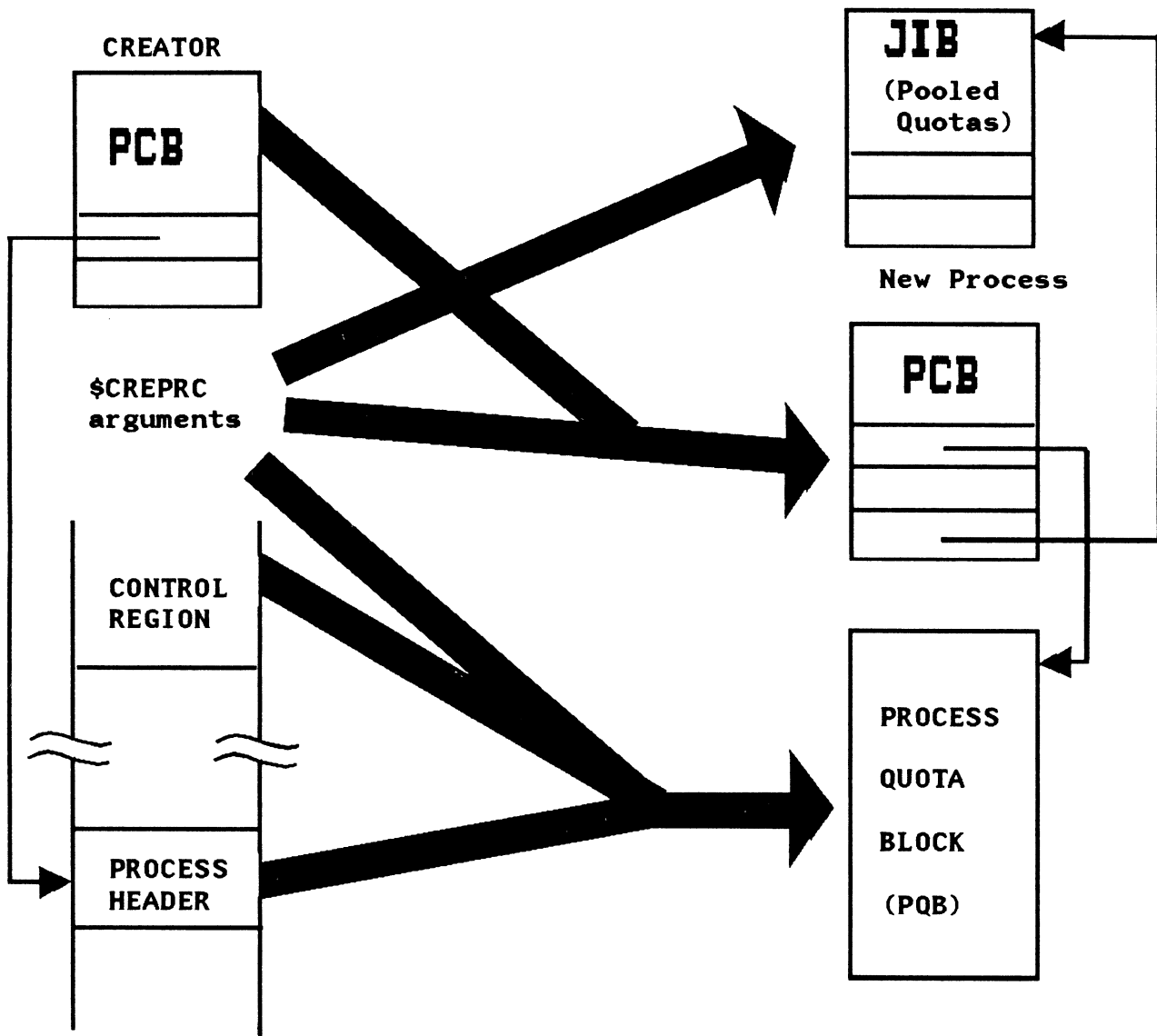
Designed to speed scheduler's search for highest-priority computable process

- A queue for each software priority
- Summary longword records nonempty COM queues
- Internally, software priority stored as inverted value (ie as 31 minus priority)

COMO state is implemented like COM state

- 32 more queues
- Another Summary longword

CREATION of PCB, JIB, and PQB



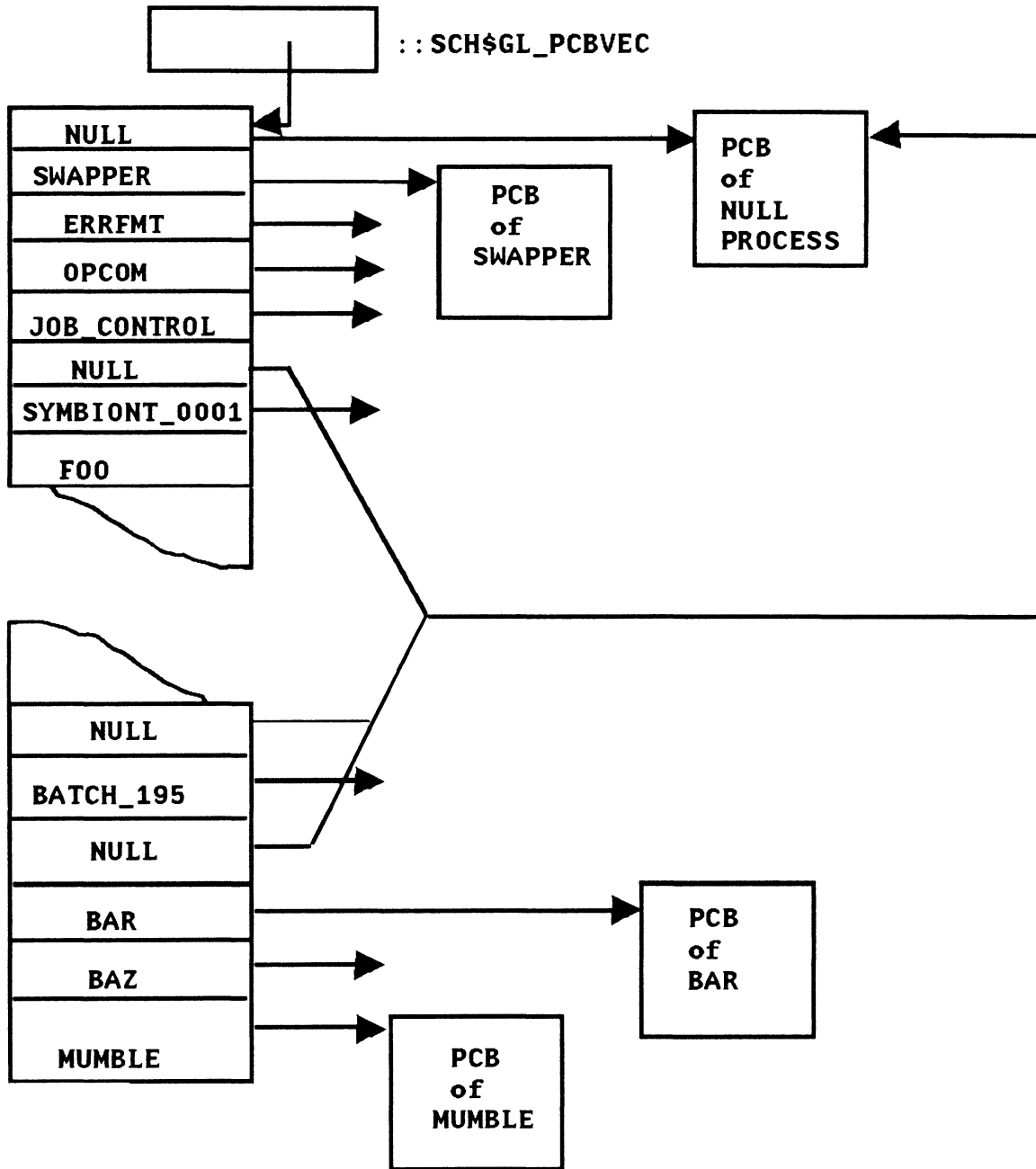
1. \$CREPRC allocates new data structures

- PCB
- JIB (if new process is detached)
- PQB (temporary)

2. These new data structures are filled from:

- \$CREPRC arguments
- Creator's PCB
- Creator's control region
- Creator's process header
- System defaults

PCB VECTOR



On process creation, search for an unused vector

Unused vectors point to Null's PCB

Table of pointers to ALL PCBs

Index into table is contained in PID

SCH\$GL_PCBVEC to start of table

VMS SYSTEM PROCESSES

Proc. Name	Base Priority	Image Name	Comments
NULL	0	part of SYS. EXE	
SWAPPER	16	part of SYS. EXE	System wide memory manager
ERRFMT	7	ERRFMT. EXE	Cleans up error log buffer
OPCOM	6	OPCOM. EXE	Operator Comm. Manager
JOB_CONTROL	8	JOBCTL. EXE	Queue and Actng. mgr. (Quasar)
SYMBIONT_n	4	PRTSYMB. EXE	Output symbionts
NETACP	8	NETACP. EXE	DECnet ACP
EVL	4	EVL. EXE	Network Event logger
REMACP	8	REMACP. EXE	Remote ACP

NOTE:

OPCOM coupled with JOB_CONTROL mimic certain functions of QUASAR and BATCON on TOPS-10/20 systems.

ACPs are roughly analogous with ACJ (Access Control Jobs on TOPS-20).

SYMBIONTS are roughly analogous with spoolers (LPTSPL, CDRIVE, SPRINT, etc and handle high speed to slow speed device interfacing.

FORMING AN IMAGE

Program Sections

Object Code is organized into program sections (PSECTs)

- By VAX-11 MACRO assembler
- By high-level language compilers
- Depending on properties of the code, or explicit PSECT directives

PSECT attributes are assigned by

- MACRO programmers
- Some defaults applied by the MACRO assembler
- High-level language compilers

Mnemonic	Attribute	Mnemonic	Attribute
WRT	Writable	NOWRT	Not Writable
RD	Readable	NORD	Not Readable
EXE	Executable	NOEXE	Not Executable
PIC	Position Independent	NOPIC	Not Position Ind.
LCL	Local	GBL	Global
CON	Concatenated	OVR	Overlaid
SHR	Potentially shareable	NOSHR	Not shareable
VEC	Protected (vector)	NOVEC	Not protected

FORMAT OF AN IMAGE FILE

.PSECT A	CON,	EXE, NOWRT
	A1	
.PSECT B	CON, NOEXE, NOWRT	
	B1	
.PSECT C	CON, NOEXE,	WRT
	C1	
.PSECT D	OVR, NOEXE,	WRT
	D1	
.PSECT E	CON,	EXE, NOWRT
	E1	

LINKER 

.PSECT A	CON,	EXE, NOWRT
	A2	
.PSECT B	CON, NOEXE, NOWRT	
	B2	
.PSECT D	OVR, NOEXE,	WRT
	D2	

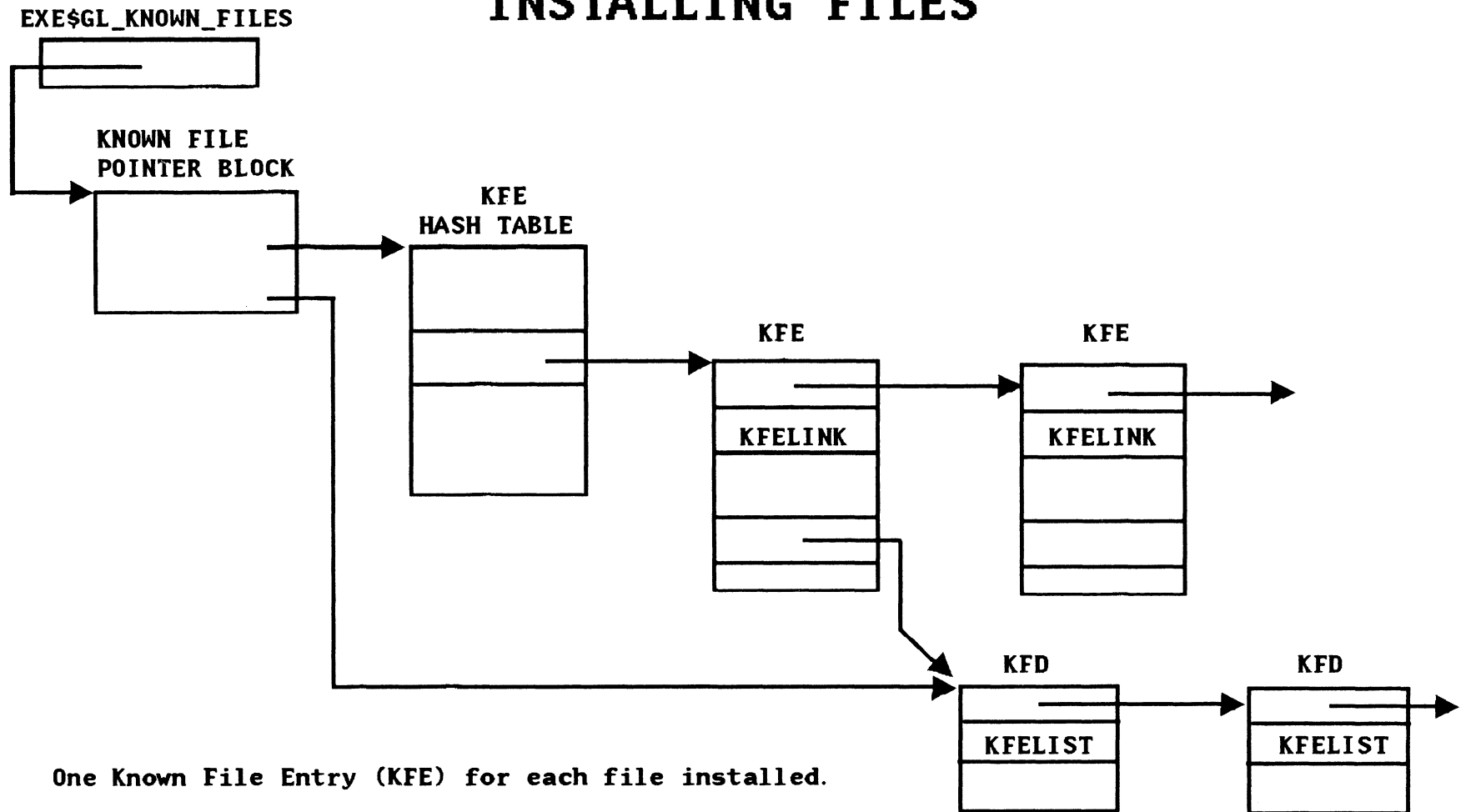
I	R/O	NOEXE	Image
II	R/W C/R	NOEXE	Section
III	R/O	EXE	Descriptors
IV	R/W C/R	EXE	
V	R/W DZRO	NOEXE	(Image Header)
VI	R/O GBL	EXE	
VII	RO, CR, GBL	NOEXE	
B1, B2			Image Sections
C1, D1, D2			
A1, A2, E1			
FIXUP VECTORS			

Image sections stored in the image file

- I. Read-only data
- II. Read/Write data (copy-on-reference)
- III. Executable code
- IV. Fixup vectors
- V. User stack is stored in demand zero
- VI. Additional image sections for global shareable RTL's as well as transfer vectors and code
- VII. Private impure data (copy-on-reference)

NOTE: All image headers live in the INDEXF.SYS in MFD [000000] for any given disk, and all images have image headers.

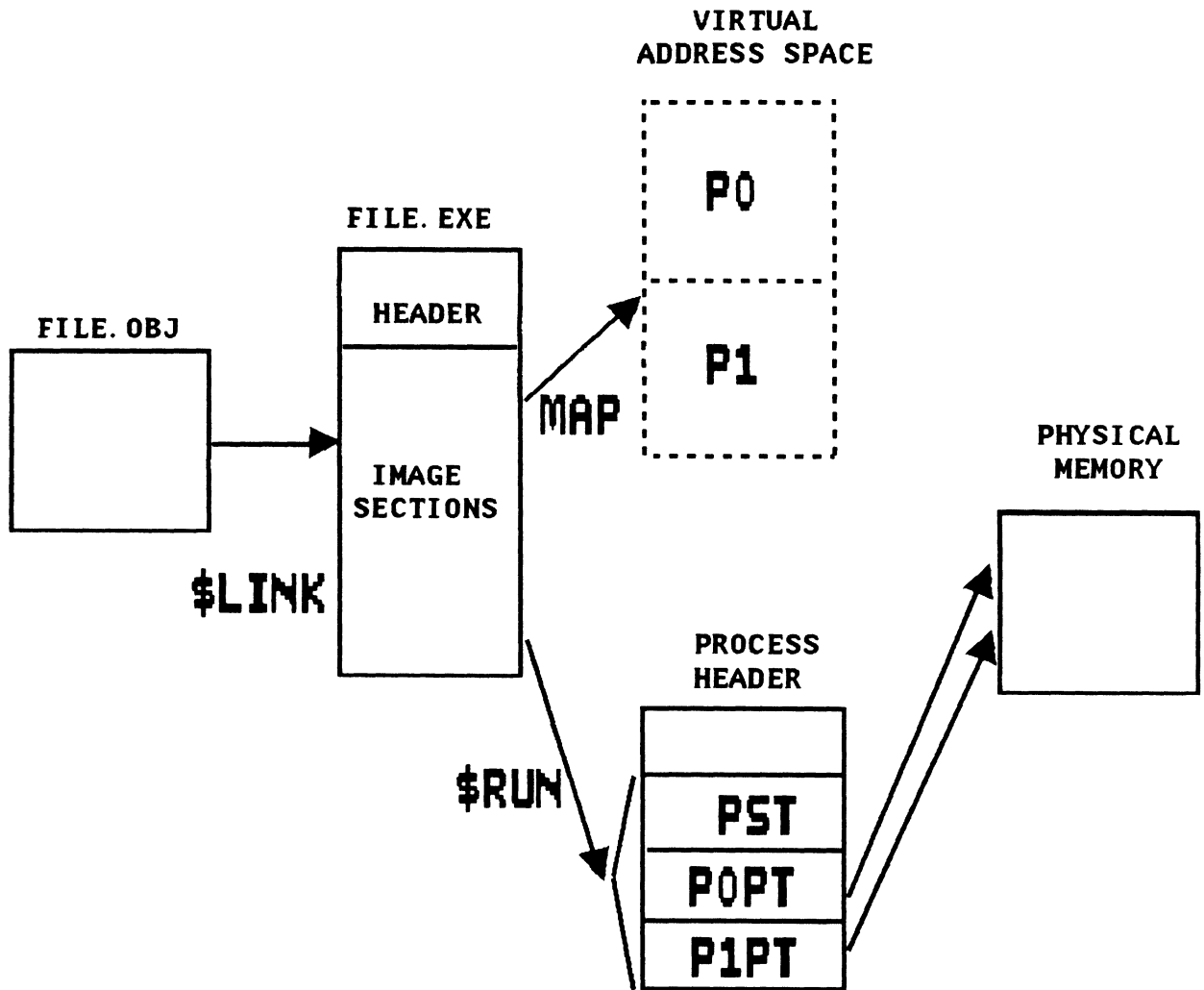
INSTALLING FILES



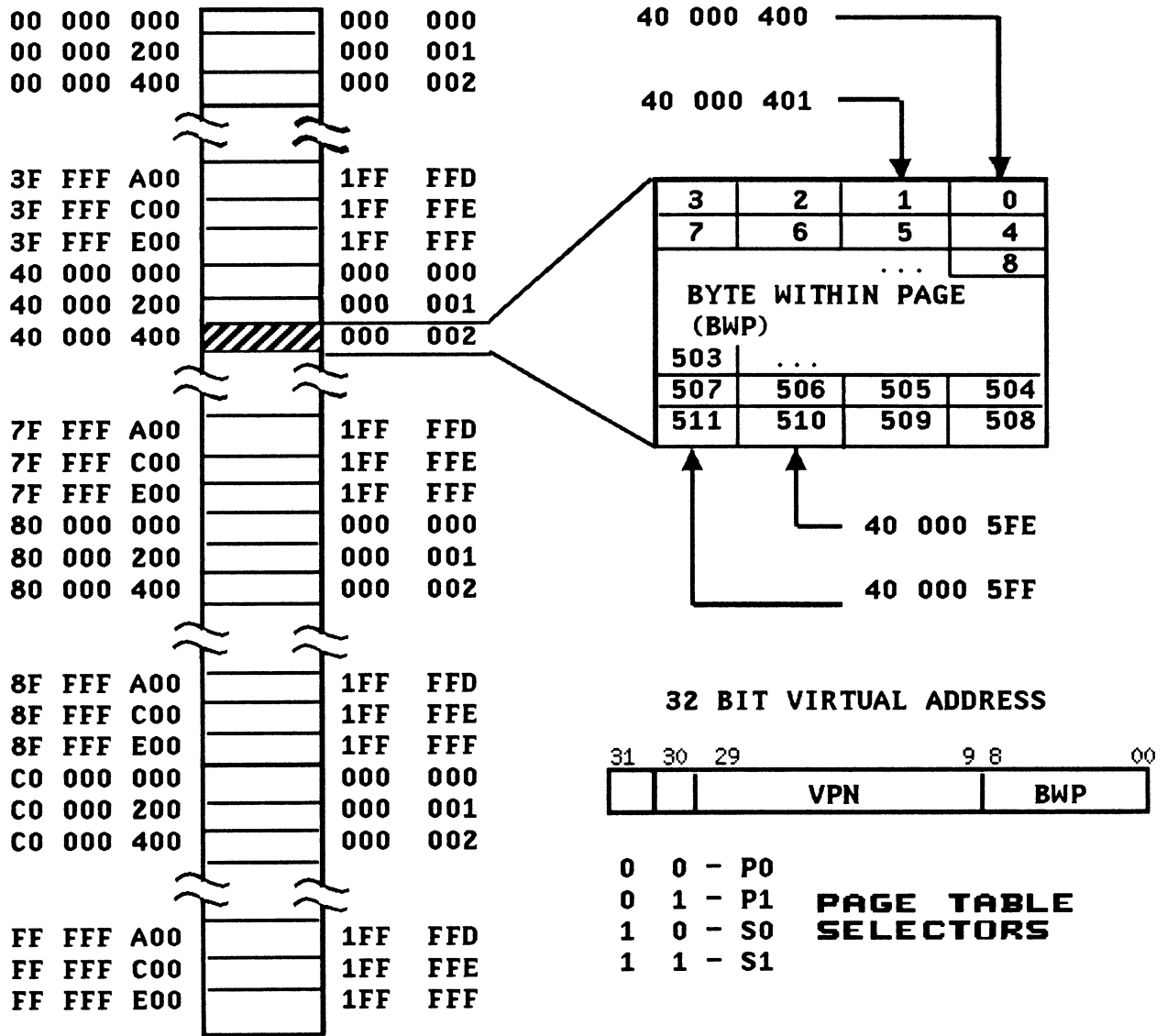
One Known File Entry (KFE) for each file installed.

Can INSTALL a file with various attributes

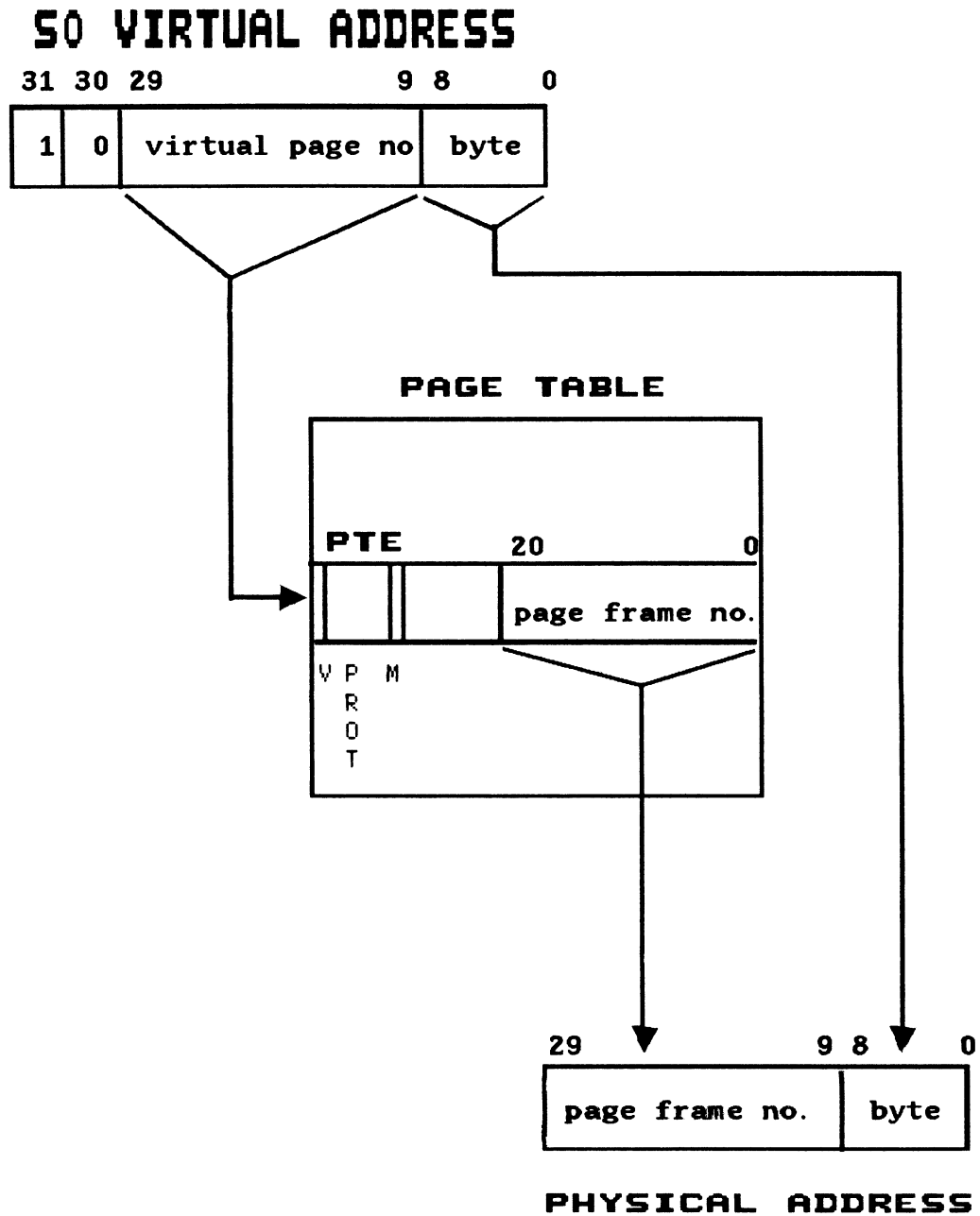
One KFD for each unique device, directory, and file-type combination



VIRTUAL ADDRESS SPACE



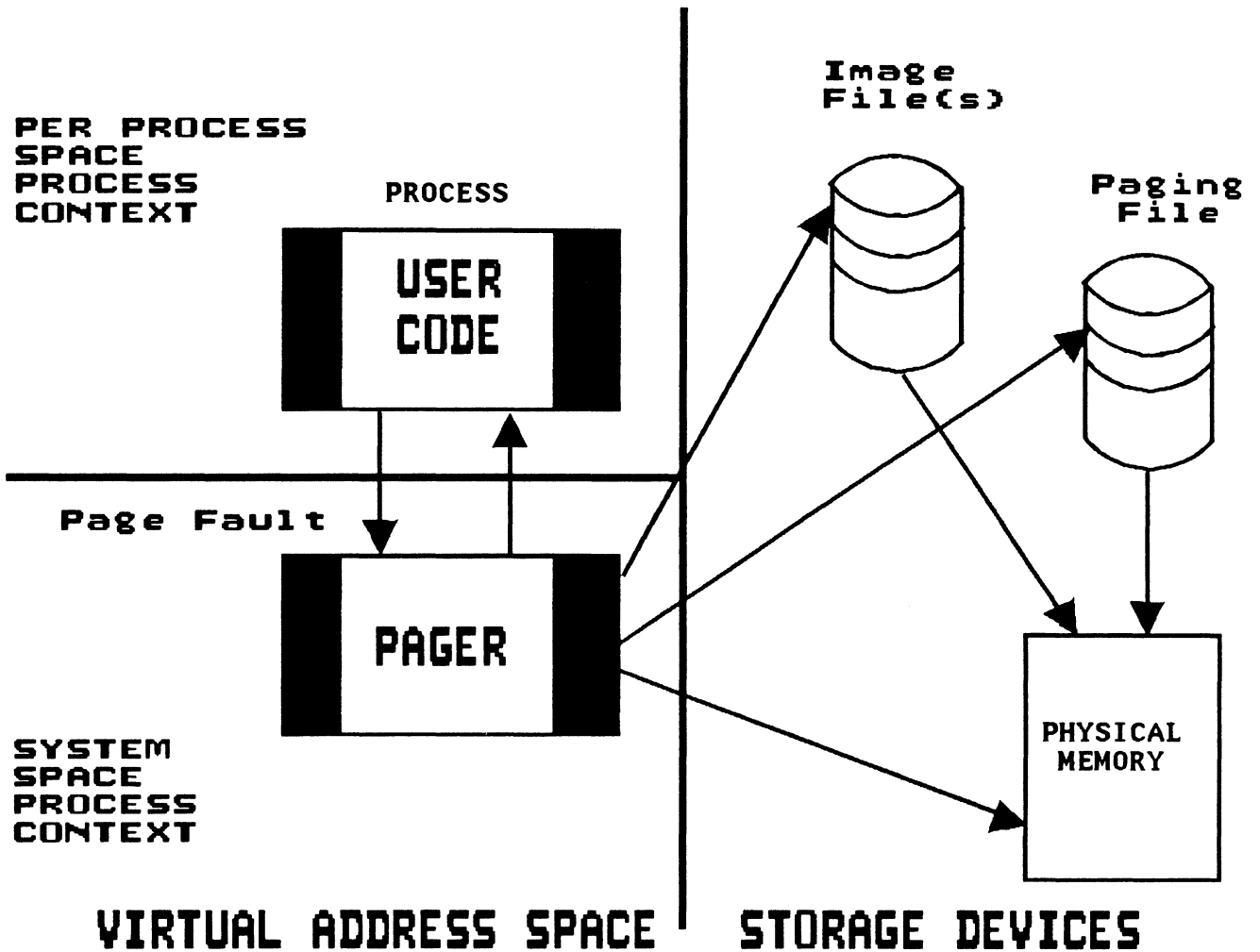
SO VIRTUAL ADDRESS TRANSLATION



NOTE:

- V - "Virtual" bit
- M - "Modified" bit
- PROT - Protection field

OVERVIEW OF PAGE FAULT HANDLING



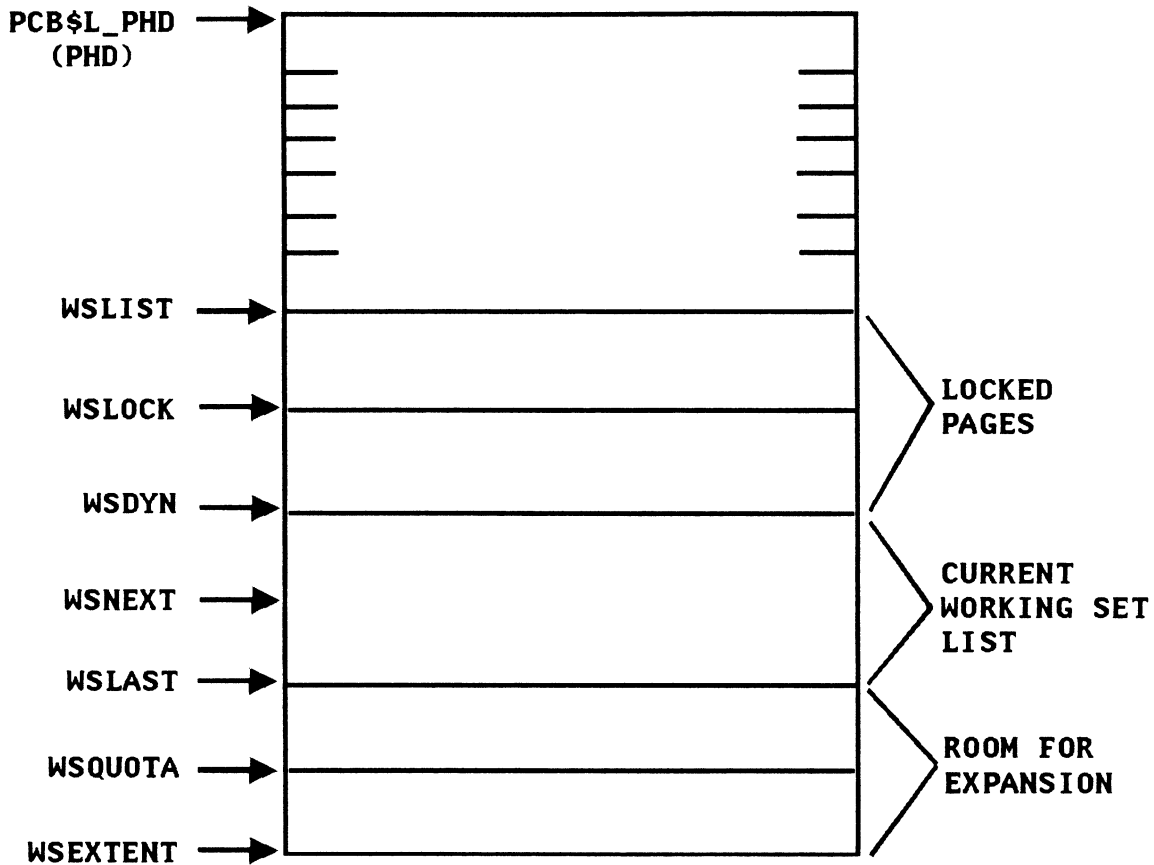
Pager is an exception service routine executing within the context of the process that incurred the page fault

Page not in memory - read I/O issued to image file or page file

Page in memory - taken from free or modified page list, or valid global page

WORKING SET LIST

PHD



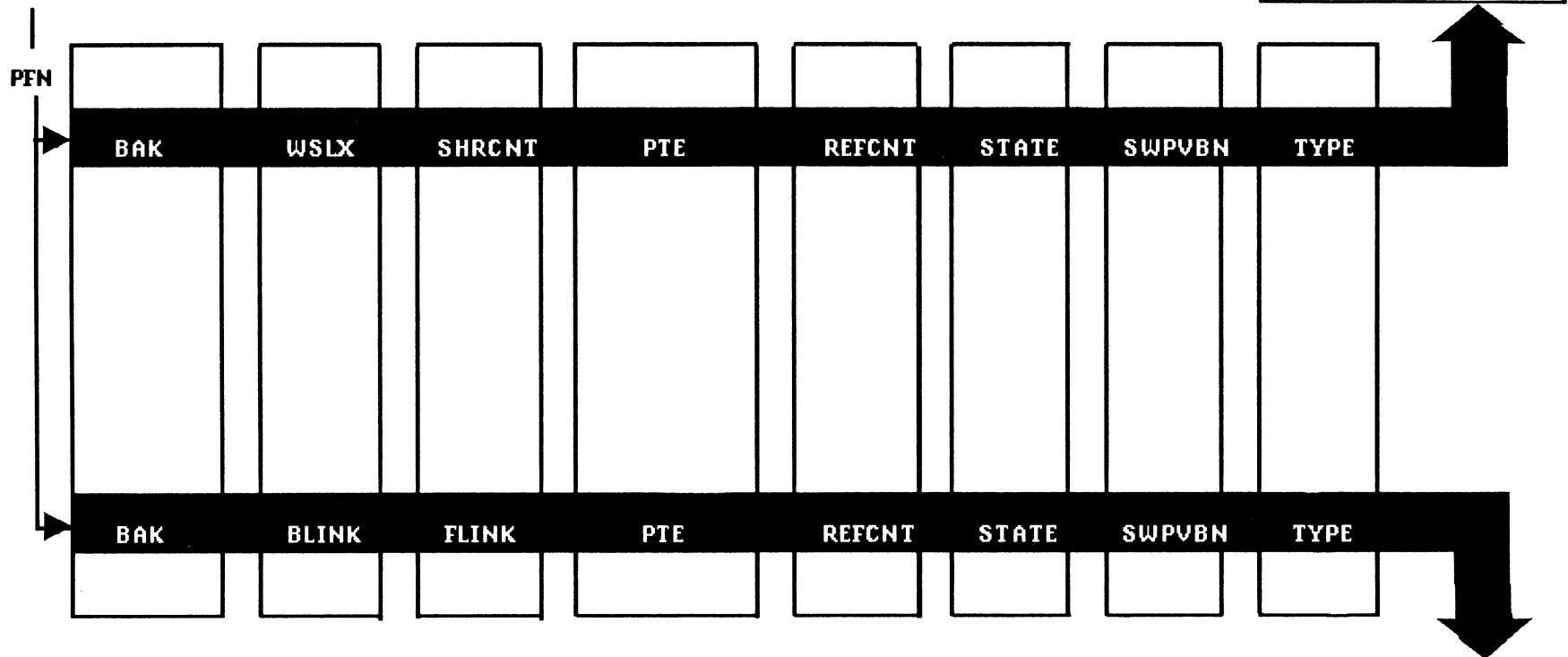
WSLAST can move to

- **WSQUOTA** if few free pages (free page count < BORROWLIM)
- **WSEXTENT** if many free pages (free page count > BORROWLIM)

WSNEXT - latest entry put in working set list

Page replacement is first-in/first-out (VAX/VMS does not have page-aging capabilities in it's microcode like TOPS)

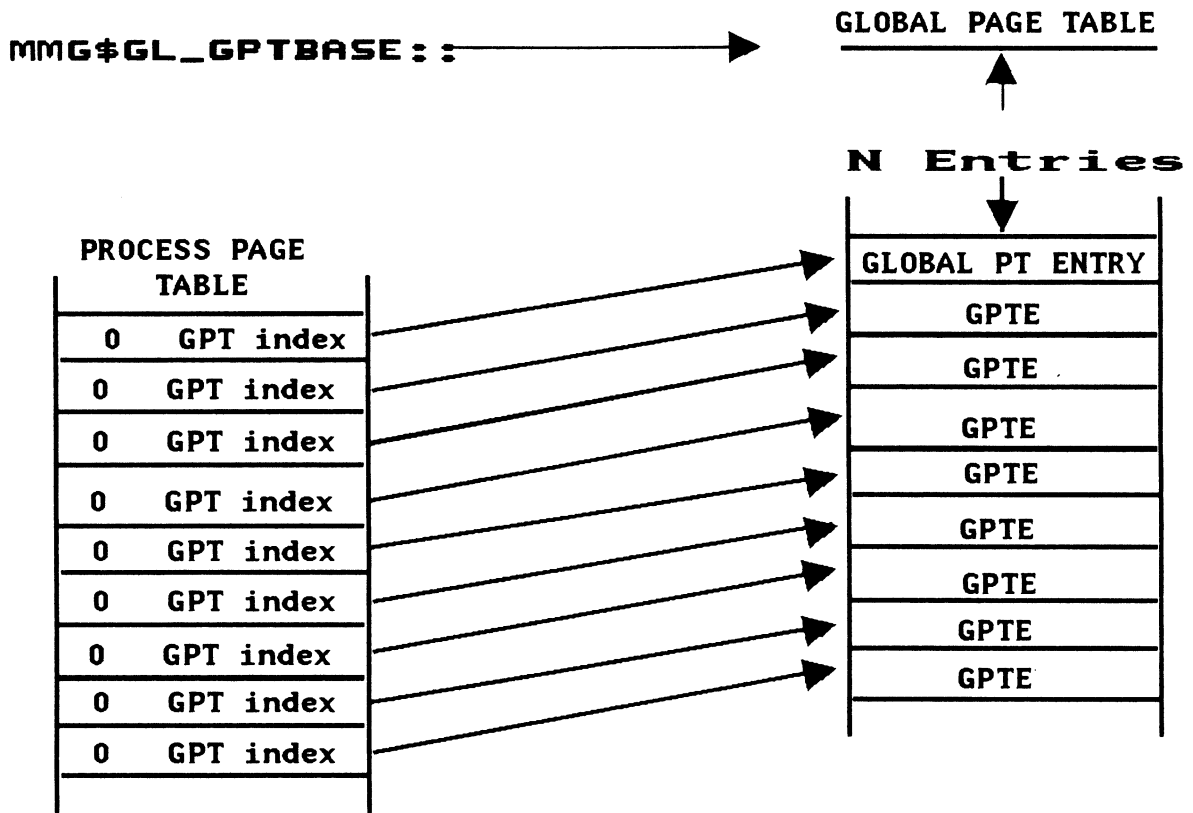
PFN DATABASE



BAK Where page should go if it must leave memory (original PTE)
WSLX, BLINK index into working set list, or link if in Free/Modified page lists
SHRCNT, FLINK number of processes sharing page, or link if in FPL or MPL
PTE virtual address of PTE that maps this page (SVAPTE)
REFCNT number of reasons not to put page on FPL or MPL
STATE specifies list of activity and saved Modify Bit
SWPVBN virtual block number in swap file or page file
TYPE type of page - for example, process, system global, etc.
PFN Physical page number, index into arrays

Free or modified page list

GLOBAL PAGING DATA STRUCTURES

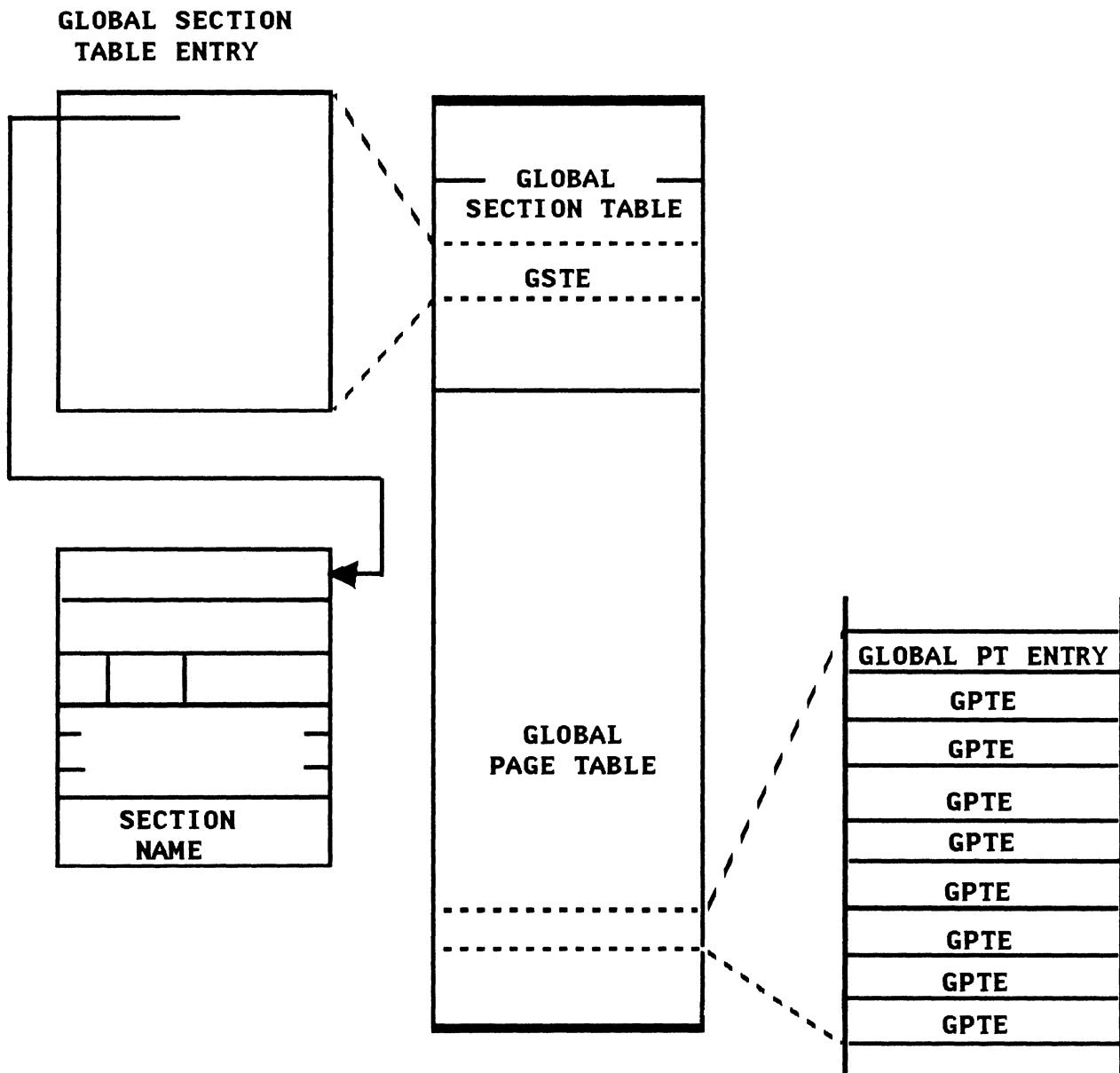


GLOBAL PAGE TABLE

Central location for global page information

Mapped into S0 space

GLOBAL SECTION DATA STRUCTURE RELATIONSHIPS

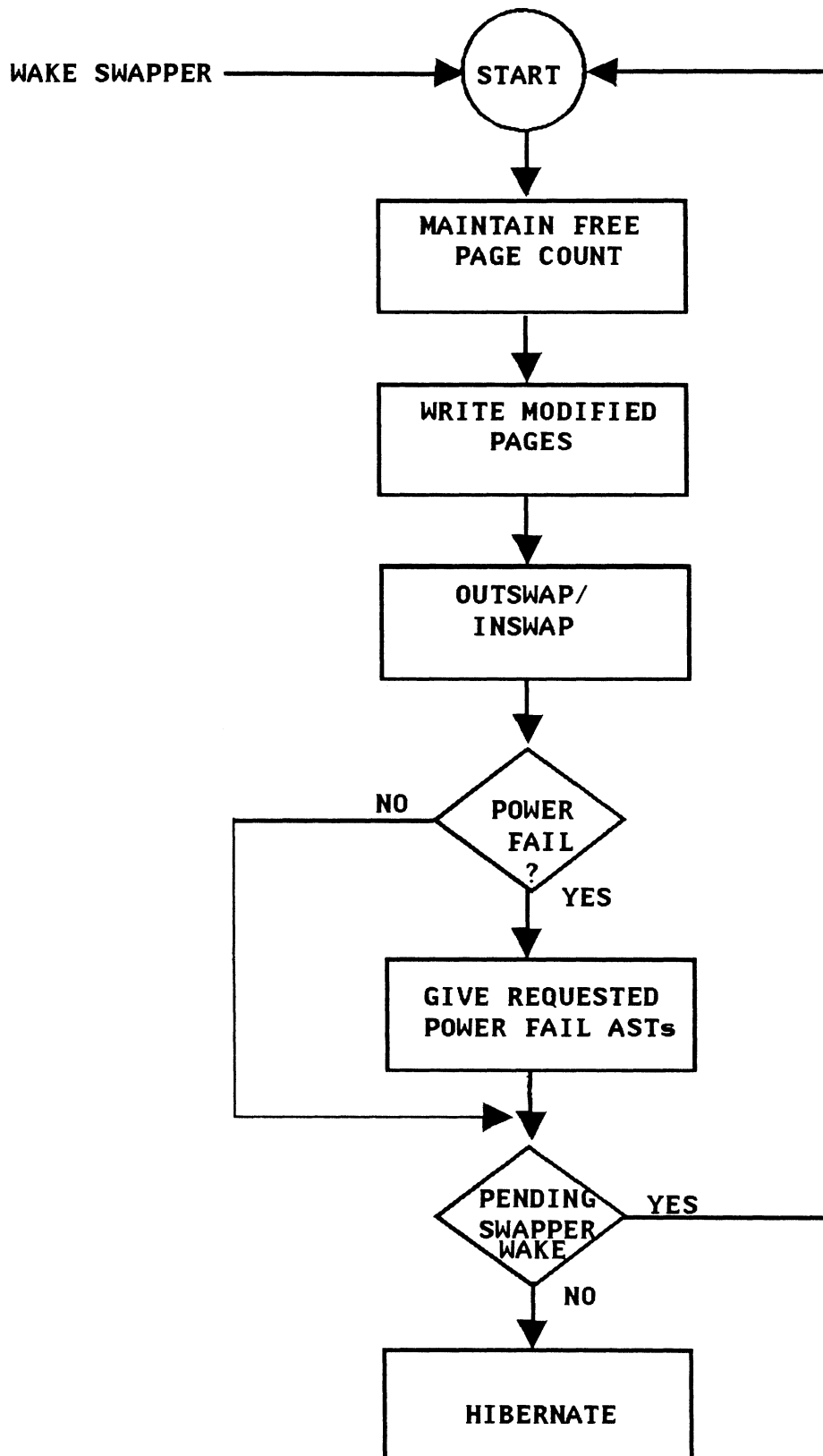


Three data structures contain global section information:

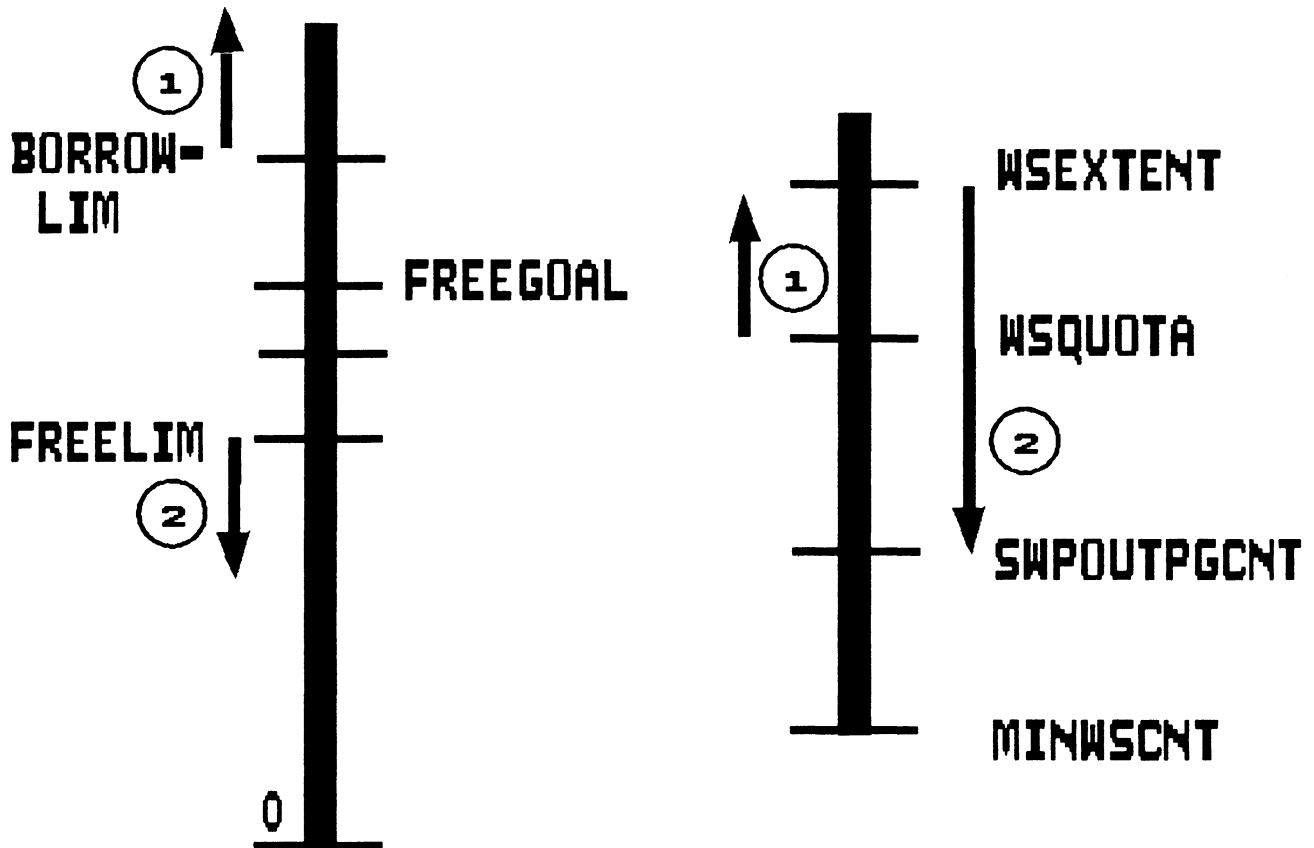
1. Global Page Table
2. Global Section Table (similar to process section table)
3. Global Section Descriptors (allow the location of global section information by name)

GSDs are placed in either a system queue or a group queue

SWAPPER - MAIN LOOP



EXPANSION AND CONTRACTION OF WORKING SETS

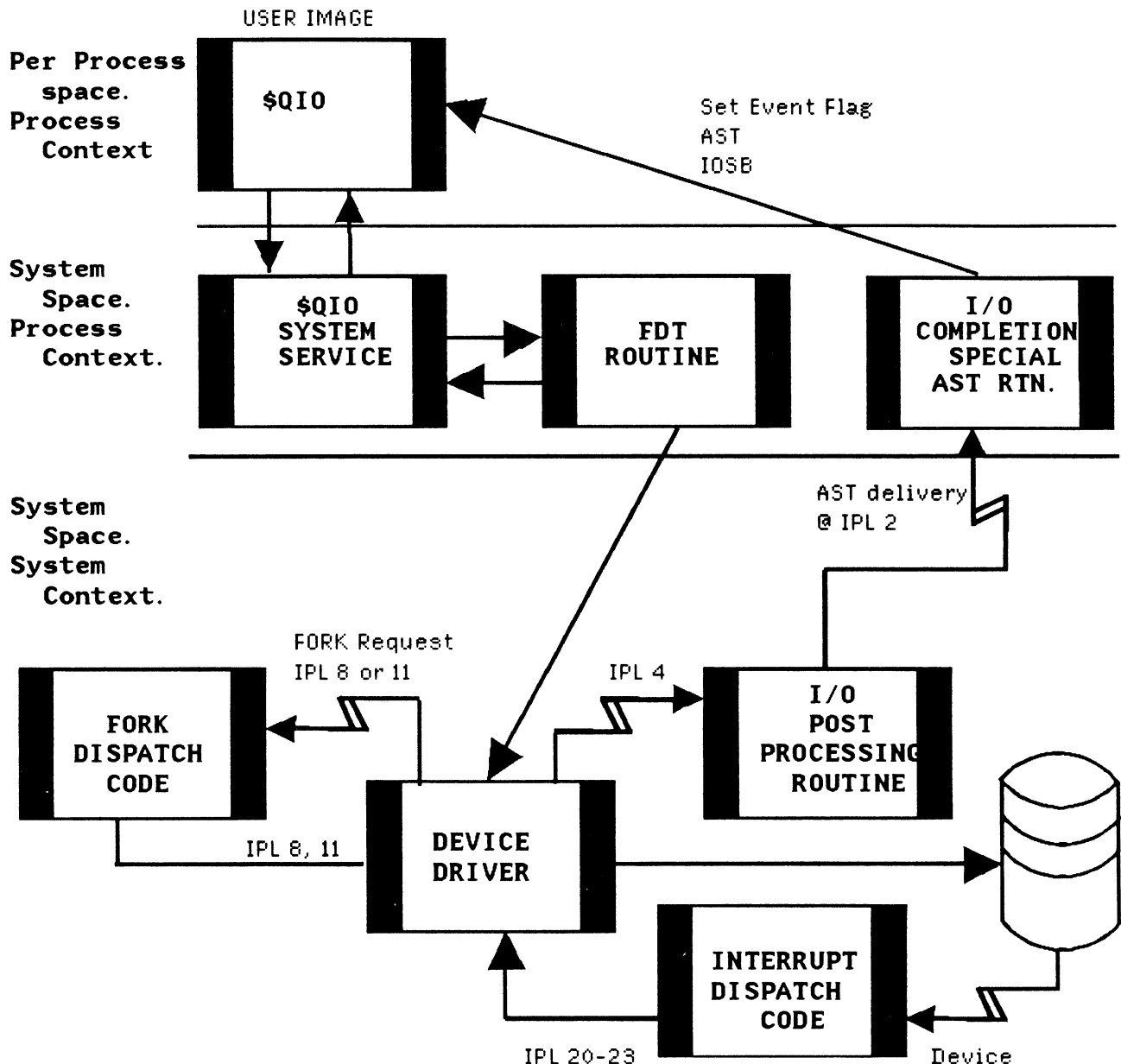


**NUMBER OF PAGES
ON FREE PAGE LIST**

**NUMBER OF PAGES
IN WORKING SET**

1. If free page count > BORROWLIM, working set may grow past WSQUOTA to WSEXTENT. (at end of current run Quantum)
2. If free page count < FREELIM, swapper will attempt to:
 - Shrink working sets from WSEXTENT to WSQUOTA
 - Shrink working sets from WSQUOTA to SWPOUTPGCNT

INPUT / OUTPUT (FULL)



Preprocessing done by RMS, \$QIO and FDT routines

Device control and data manipulation done by driver

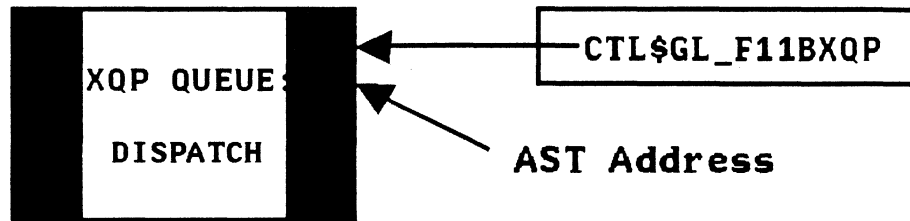
Final clean up done by I/O post and I/O completion routines

FDT (Function Decision Table) Routines are device specific extensions to \$QIO (like RPxKON on TOPS-10)

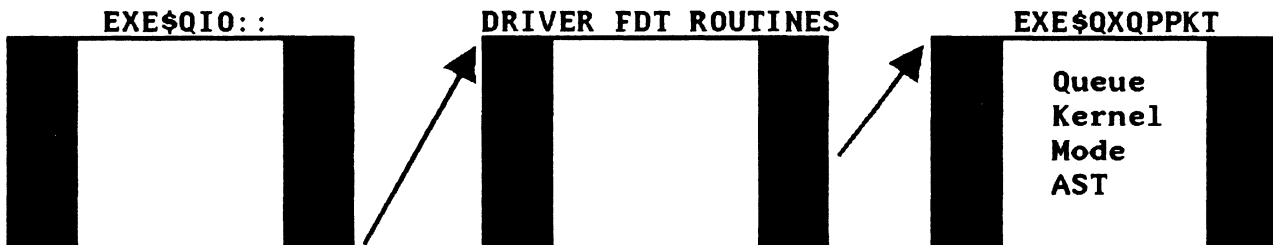
Fork Dispatch Code allows device drivers to continue processing at lower IPL level without destroying synchronization. NOT PROCESSES

XQPs

P1 SPACE



S0 SPACE



XQPs: (extended queue packets)

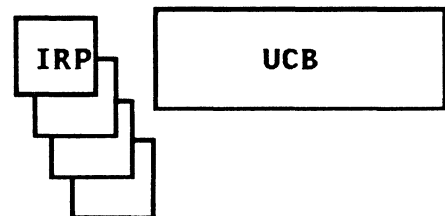
Reside in P1 space

Are used in process context

FDT routines JSB to EXE\$XQPPKT (at IPL\$ASTDEL

EXE\$XQPACP invokes the XQP by means of an AST

When finished, XQP queues IRP to the driver's Unit Control block (UCB)



NOTE:

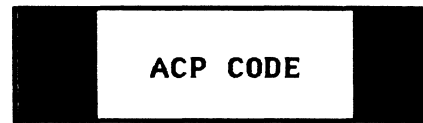
IRP - Individual Request Packet (an IORB on TOPS-20)

UCB - analogous with TOPS 10/20 UDB

FDT - analogous with TOPS 20 routines like PHYP4.MAC, etc, or TOPS-10s RPxKON.

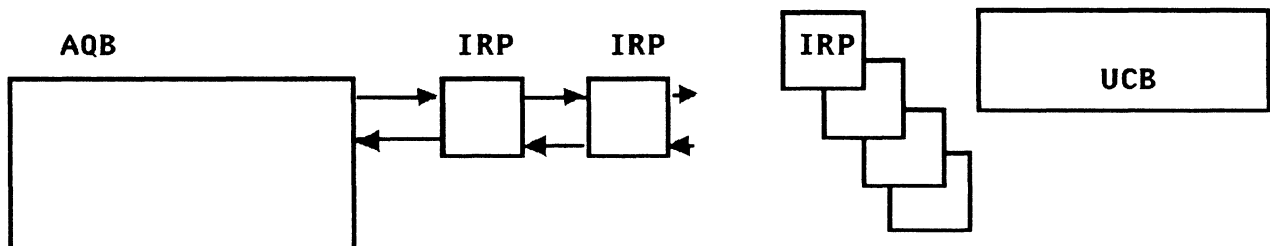
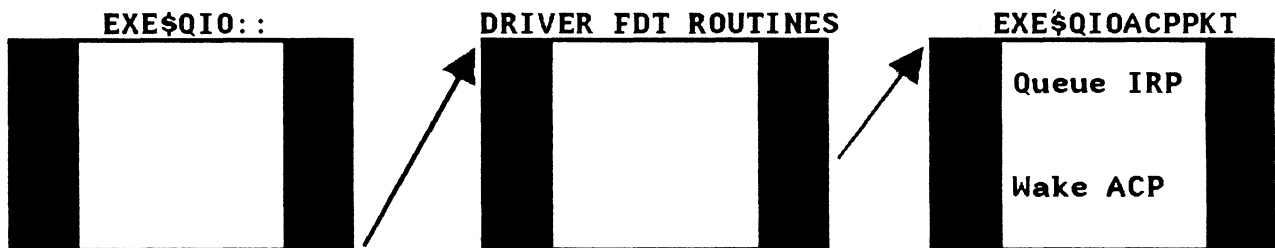
ACPs

P0 SPACE



P1 SPACE

S0 SPACE



ACPS (AST Control Blocks) (not to be confused with ACPs - ancillary control process (analogous with ACJs or DAEMONS) confused yet?)

Are separate processes

FDT routines JSB to EXE\$QIOACPPKT (= < IPL\$SYNCH)

EXE\$QIOACPPKT queues IRPs to ACP Queue Block (AQB) and wakes ACP

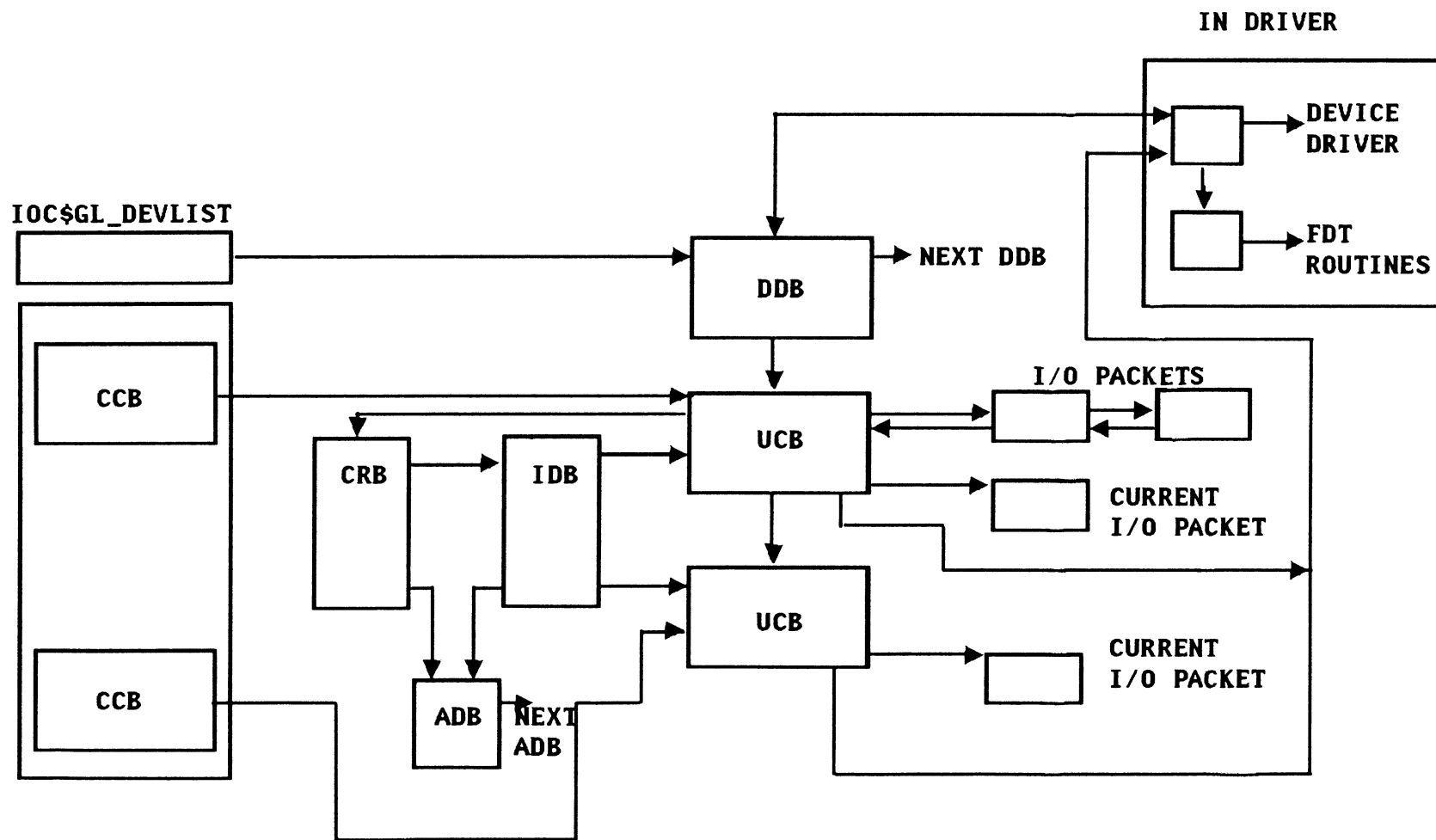
When finished, ACP queues IRP to proper UCB

THE I/O DATABASE

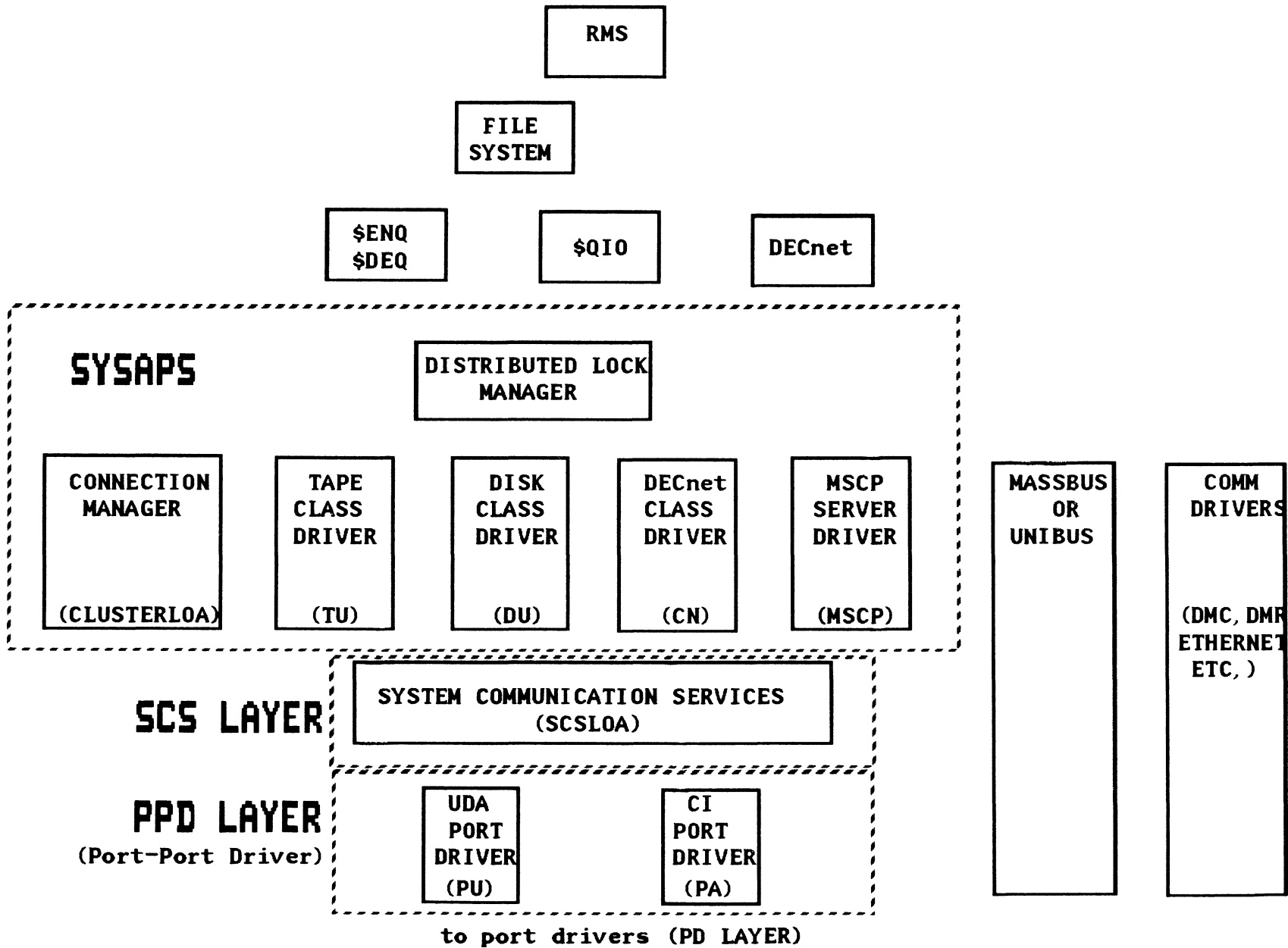
NAME	FUNCTION	COMMENTS
IRP	Carries information for a specific I/O request	Created by \$QIO in nonpaged pool
CCB	Links a 'channel' to a specific device unit	Created by \$ASSIGN in P1 space
DDB	Contains information common to all devices on a controller	One per device type (one for DBA, etc.)
UCB	Contains information for a device unit. Used as a listhead for storage by the driver	One per device unit (one for DBA1:, etc.)
DDT	Contains entry point addresses for driver routines	Used by VMS to select the correct routine
FDT	Contains list of valid functions and their FDT routine addresses	Used by \$QIO to select routines for the proper
CRB	Contains information and list-heads for a particular controller	Used especially by devices that share a controller (for example, DBA1: and DBA2: share controller DBA)
IDB	Contains information including a table of UCB addresses for units under a controller	Used by drivers and VMS
ADP	Contains information including mapping registers and data paths	Used by drivers and VMS

Name	Definition	TOPS 10/20 Definition
IRP	- Individual Request Packet	Like an IORB
CCB	- Channel Control Block	Like a KDB
DDB	- Device Data Block	Like a KDB and UDB mixed
UCB	- Unit Control Block	Like a UDB
DDT	- Driver Dispatch Table	Like Channel Logout Areas
FDT	- Function Descriptor Table	Like RPxKON or PHYPx.MAC
CRB	- Controller/Channel Request Block	Like Channel Command List
IDB	- Interrupt Dispatch Table	Like old style RH vectoring
ADP	- Adaptor Control Block	

SUMMARY LAYOUT OF I/O DATABASE

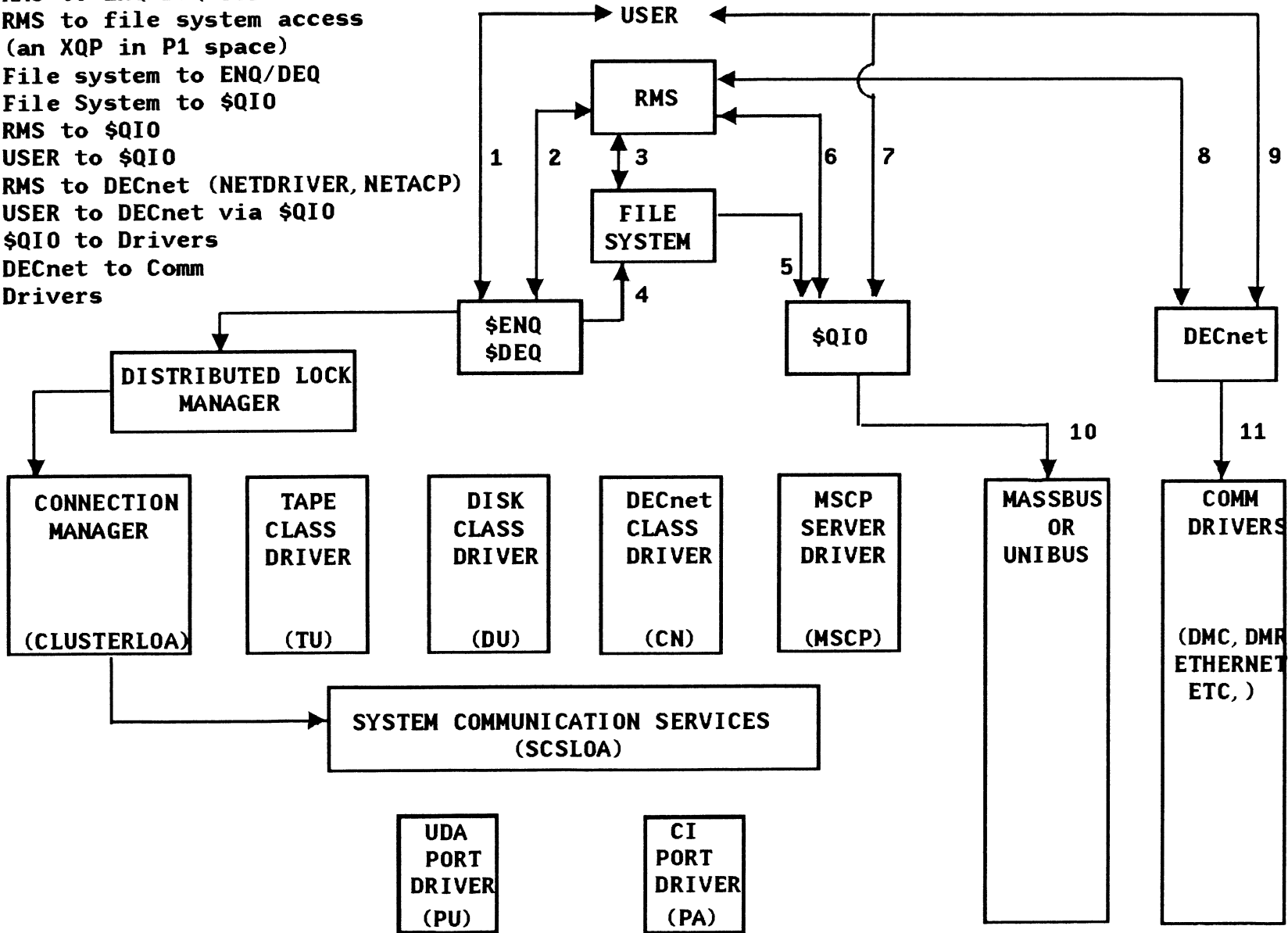


VAXcluster SOFTWARE COMPONENTS



VAXcluster SOFTWARE COMPONENTS

1. USER to ENQ/DEQ services
2. RMS to ENQ/DEQ services
3. RMS to file system access
(an XQP in P1 space)
4. File system to ENQ/DEQ
5. File System to \$QIO
6. RMS to \$QIO
7. USER to \$QIO
8. RMS to DECnet (NETDRIVER, NETACP)
9. USER to DECnet via \$QIO
10. \$QIO to Drivers
11. DECnet to Comm Drivers



SYSTEM PROCESSES in a VAXcluster

PROCESS NAME	PRIORITY	IMAGE NAME	COMMENTS
CACHE_SERVER	16	FILESERV.EXE	Flushes the system-wide caches
CLUSTER_SERVER	8	CSP.EXE	Envelop for cluster jobs (cluster OPCOM)
CONFIGURE	8	CONFIGURE.EXE	Dynamic device configuration manager

PROCESS NAME	ERROR LOG FILE	PRIVILEGES	UIC
CACHE_SERVER	cache_server_error.log	all	[1, 4]
CLUSTER_SERVER	cluster_server_error.log	all	[1, 4]
CONFIGURE	configure_error.log	CMKRNL, PRMMBX BYPASS, SHARE	[1, 4]

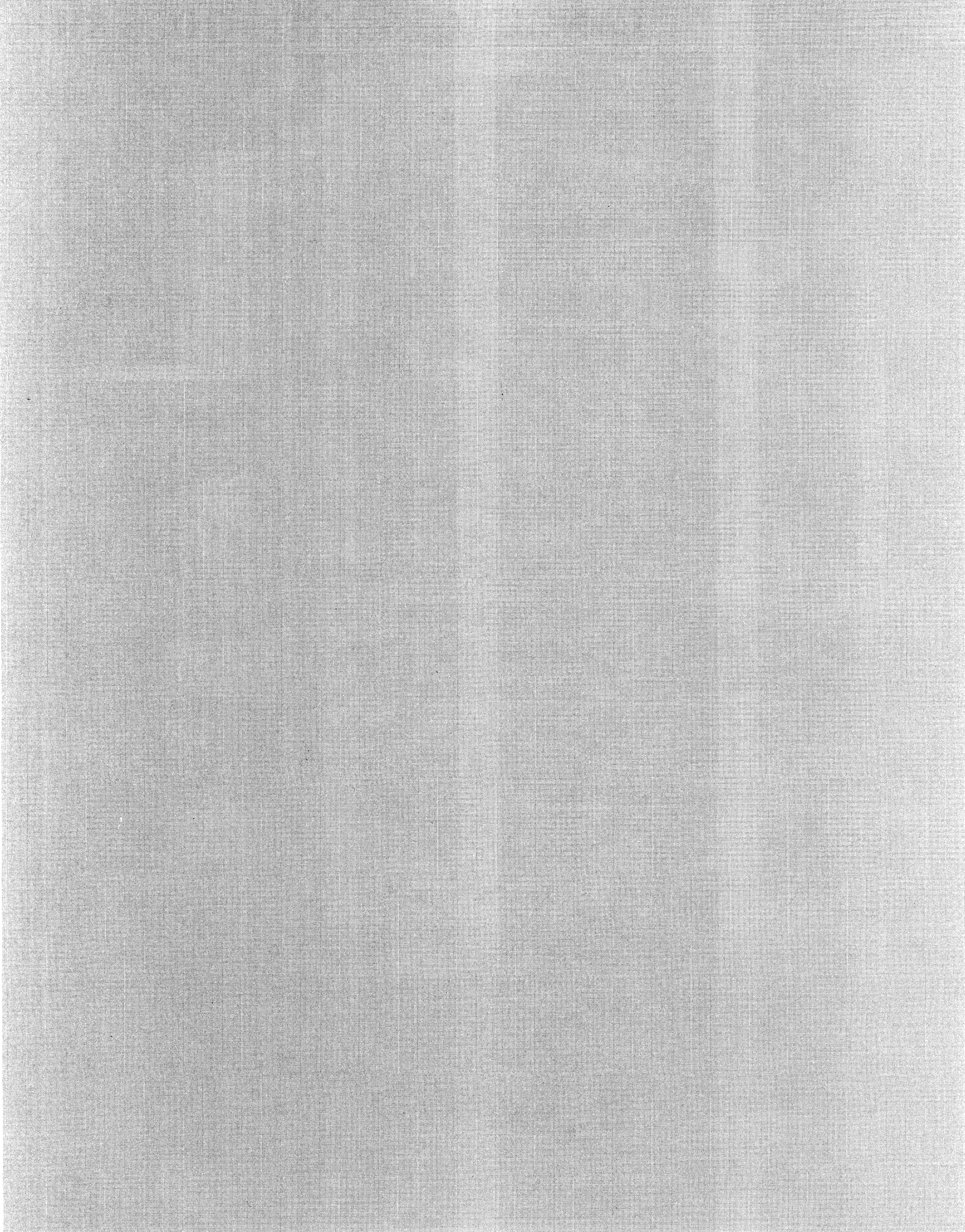
CACHE_SERVER and CLUSTER_SERVER are only created if system is a member of a VAXcluster.

CONFIGURE is only created if device PAA0: exists.

All images reside in SYS\$SYSTEM

All error log files reside in SYS\$MANAGER

LANGUAGES AND TOOLS SIG



James M. Briggs
 Raymond J. Bentz
 RCA Aerospace and Defense
 Electronic Systems Department
 Moorestown, NJ 08057

ABSTRACT

The VAX Language Sensitive Editor (LSE) helps a user construct syntactically correct programs in languages for which LSE templates have been developed. DEC provides templates for the programming languages which it supports, such as FORTRAN and Pascal.

This paper discusses the steps necessary for a user to create LSE templates for other programming languages. They include how to get started; how much of the syntax to support using LSE; the adding of help messages; the process of creating the template file including translating from the BNF description of the computer language; and adding features which support nonsyntactic local programming standards. These steps are illustrated by examples from creating an LSE editing environment for the Navy CMS-2 programming language.

INTRODUCTION

Syntax-directed editing tools (see pp. 387-389 of [1]) can make entering computer programs simpler, but they have drawbacks. Because of the requirement of program syntactical correctness, input of the program is too structured. Consequently, much of the convenience usually associated with editors like VAX EDT is lost. Secondly, no easy way seems to exist to create a syntax-directed editor for a new language.

With introduction of the VAX Language Sensitive Editor (LSE), an opportunity arose to simplify the entry of computer programs using its template-driven language editing features, and still keep all of the desirable features in EDT.

We succeeded in building a language editing environment for the Navy CMS-2 language using LSE. In doing that, we made certain decisions about how much syntax to automate, how to use language help features, and how to translate from the formal definition of the language into the LSE template language. This paper discusses these aspects of building a language-sensitive editing environment and provides insight into what we did in creating a CMS-2 language editor.

VAX LSE

The VAX Language-Sensitive Editor (LSE) is an advanced text editor which aids in the creation of syntactically correct programs. Its normal text editing features provide a screen-oriented text editor similar to EDT. The text editing features include split-screen editing and the ability for users to redefine the key definitions to tailor the editing environment.

Language Sensitive Editing

LSE's language editing features simplify the entry of computer programs. LSE language editing is based on the use of predefined language templates for the computer language being used, for instance, FORTRAN or Pascal. While editing a program in a language for which the templates exist, the LSE user constructs a program by successive expansions of either placeholders or to-

kens. Placeholders indicate where in the code a user must supply additional (possibly optional) text information. Tokens are keywords which the user may enter in order to cause an associated template to appear. LSE keyboard commands allow a user to EXPAND a placeholder or token, to GOTO the next or previous placeholder, or to ERASE an optional placeholder.

When one presses the expand key over either a placeholder or a token, a template is copied into the text buffer. This template consists of formatted text which will become part of the program and possibly more placeholders requiring additional expansion. The user fills out the required text information as desired.

Using templates for language editing enables the user to:

- Enter programs more quickly,
- Minimize syntax errors,
- Achieve a uniform coding style.

Note that the language specific features of LSE do not force the entry of a syntactically correct program, although the use of the templates makes it a lot easier. In addition, at any time, the user still has access to all of the jornahtext editing commands; thus, the user has the choice of using language specific editing features or standard text editing commands, whichever is more convenient.

Uses of Language Sensitive Editing

Although LSE's language editing features are directed to simplified entry of computer program source, it is possible to use it for structuring other text entry wherever one can create acceptable input templates. Such possibilities exist in a wide range of data entry applications, for instance, in simplifying RUNOFF document input.

LSE can satisfy a wide range of user needs for program editing. The templates and help information make it ideal for teaching the programming language; an experienced user will appreciate the ability to quickly expand tokens while using the EDT-like features.

CMS-2 Programming Language

The CMS-2 language was developed by the U.S. Navy in the late 1960s. It is a high-order language extensively used in developing weapons systems for the Navy. This large language contains a rich set of data structures for both global and local data, supporting system procedures, procedures and functions, and a full set of control structures supporting structured programming. Compilers and cross-compilers are available to support development of CMS-2 programs for the standard Navy computers[2].

GETTING STARTED

Before developing templates for a new programming language, a few things need to be done.

Get Some Experience

Before thinking seriously about implementing the templates for a new language, one should use LSE with a familiar DEC-supported language such as FORTRAN or Pascal. Check out the style of DEC's templates, DEC's use of menus for placeholders, optional items versus required items, indentation style, and help support.

Get Some Documentation

One should acquire the LSE User's Guide [3], the language's user's manual, in our case [2], and obtain a syntax description for the language. We were able to obtain the syntax definition in a Backus-Naur Form (BNF) from the user's manual. The user's manual will become the source of help information, examples, and, sometimes, even the style. The syntax description is necessary for creating the language templates.

Make Some Decisions

After becoming experienced with LSE, one needs to make some basic decisions which will affect the production of the templates. Not only can very complete templates and help information be created, but also a minimal language editor by adding templates for only a few tokens of the language. So, a decision must be made concerning how much of the language syntax needs to be implemented. This will determine the number and depth of expansion of placeholders and the tokens which are supported. Although our CMS-2 language editor is fairly complete, we decided not to implement the templates for expressions. Consequently, expressions become terminal placeholders and the user is responsible for their correct entry.

Templates can also support the entry of nonsyntactic source information. In the CMS-2 language editor, templates were used to add comments including module purpose, revision history information, and programmer identification.

File organization decisions have to be made. In implementing the CMS-2 editor, we chose to support two separate file types (.CTS and .CS2). The CTS (compile-time system file) contains the major header information needed to compile the source and includes separate CS2 files. The CS2 files contain either complete SYS-PROCs (system procedures) or SYS-DDs (system data designs). Thus there are two DEFINE LANGUAGE statements in our definition (see Figure 1) each of which has distinct language editing capabilities. The correct templates are used by the editor automatically when the user chooses to edit a file of type .CTS or .CS2.

```
DEFINE LANGUAGE CTS -
/CAPABILITIES = NODIAGNOSTICS -
/COMPILE_COMMAND = "QCMS2" -
/FILE_TYPES = (.CTS) -
/IDENTIFIER_CHARACTERS = -
"ABCDEFGHIJKLMNPOQRSTUVWXYZ01234567890" -
/INITIAL_STRING = -
" {cms2_system_declaration}" -
/OPT = ("[" , "]" ) -
/OPTL = ("[" , "]" ... ) -
/REQ = ("{" , "}" ) -
/REQL = ("{" , "}" ... )

DEFINE LANGUAGE CMS2 -
/FILE_TYPES = (.CS2) -
/IDENTIFIER_CHARACTERS = -
"ABCDEFGHIJKLMNPOQRSTUVWXYZ01234567890" -
/INITIAL_STRING = -
" {system_element}" -
/OPT = ("[" , "]" ) -
/OPTL = ("[" , "]" ... ) -
/REQ = ("{" , "}" ) -
/REQL = ("{" , "}" ... )
```

Figure 1. DEFINE LANGUAGE Extracts from CMS2.LSE

Finally, decisions need to be made concerning the style of the source. This includes indentation style and the layout of the control structures. The editing templates will automatically enforce these decisions.

Try to Automate LSE Template Building

One notices very quickly that much repetition is involved in entering the LSE template definitions. For instance, in our CMS-2 DEFINE PLACEHOLDER definitions, there are always the same eight statements with, at most, minimal changes. We decided early that a good idea would be to implement an LSE language editor using LSE. We actually implemented three LSE editors: one for the CMS-2 programming language, one for the LSE definitions, and one for simplifying the entry of HELP library information.

The editor for the LSE definitions was designed to make template entry as effortless as possible. Menus were used for almost all commands, qualifiers and qualifier values. Optional qualifiers with a fixed set of values were implemented as menus of tokens representing the values. When one of these menu items is selected, the token expands into both the qualifier and its value. The most commonly used menu choices were placed toward the top of their menu to minimize user keystrokes when choosing them. With the use of this editor, template entry is almost automatic and requires minimal knowledge of the format of the LSE template language. This allows the user to concentrate on the problem of defining the target language.

BUILDING LANGUAGE TEMPLATES

In building the LSE language templates, the best guide to have is a structured description of the language such as provided by Backus-Naur Form (BNF) production rules. The BNF notation translates easily into the LSE template language. See Figures 2 and 3 for equivalent BNF and LSE descriptions of the same example.

Before we started creating the templates for our CMS-2 language editor, we first collected all of the BNF rules for the CMS-2 language into a data file, CMS2.BNF. This was constantly referred to in writing our CMS-2 editing templates.

```

<program block> ::= <procedure block>
                  <function block>
                  <exec-proc block>

<procedure block> ::= [(EXTDEF)] PROCEDURE <name>
                    <formal i-o params>
                    [EXIT <name list>] $
                    <program body>
                    END-PROC <name> $

```

Figure 2. Extracts from CMS2.BNF

```

DEFINE PLACEHOLDER program_block -
  /LANGUAGE = CMS2 -
  /NOAUTO_SUBSTITUTE -
  /DESCRIPTION = "PROCEDURE, FUNCTION, or EXEC-PROC" -
  /TOPIC_STRING = "PROGRAM_BLOCK" -
  /TYPE = MENU
    "procedure" /PLACEHOLDER /NOLIST
    "function" /PLACEHOLDER /NOLIST
    "exec_proc" /PLACEHOLDER /NOLIST
END DEFINE

DEFINE PLACEHOLDER procedure -
  /LANGUAGE = CMS2 -
  /NOAUTO_SUBSTITUTE -
  /DESCRIPTION = "CMS-2 PROCEDURE" -
  /TOPIC_STRING = "PROCEDURE_BLOCK" -
  /TYPE = NONTERMINAL
  "[extdef] PROCEDURE (name)"
  " [formal_input_parameters]"
  " [formal_output_parameter]"
  " [exit_phrase] $"
  " (program_body)"
  "END-PROC (name)"
  "$"
END DEFINE

DEFINE TOKEN PROCEDURE -
  /LANGUAGE = CMS2 -
  /PLACEHOLDER = procedure

```

Figure 3. Extract from CMS2.LSE

Top-down or Bottom-up Approach

Two basic approaches to building the LSE template file from the BNF description of the language are top-down and bottom-up.

The top-down approach is begun by expanding a high-level non-terminal and working down to lower levels. This is a results-oriented approach. Small parts of the editor become quickly available for debugging and use. This approach unfolds easily from the BNF. Defining productions that have multiple right-hand sides (see Figure 2) is simple this way: define each production as a "menu" and worry about each menu item later on. However, this approach can seem to get out of hand because it appears to be expanding in many directions at one time.

The bottom-up approach is begun by expanding the low-level non-terminals and working up to the higher level ones by using the low-level ones as building blocks. This approach seems to lead to a better structured template file because each placeholder is defined in terms of already existing placeholders/tokens. However, it does not flow smoothly from the BNF; one has to look at the BNF "backwards" to understand how each placeholder needs to be defined. Furthermore when using this approach, one really has nothing to show for the effort until it is almost finished.

Because the CMS-2 language editor was our first experience with defining the templates, and because of the large size of the target language, we used a top-down approach to the task. This was a direct method for attacking the problem. It allowed us to define templates for small parts of the language which could be tested early on in the development effort.

Defining the Templates

Once the approach is decided on, the steps necessary to define the templates are quite simple. Choose a production to expand. Create a placeholder which has the left hand side of the production as its name. For each keyword in the production, include it in the body of the placeholder as a terminal. For each nonterminal in the production, include a placeholder for it in the body of the placeholder being defined. Productions that can expand into different right hand sides should have bodies defined as being of "menu" type. Continue doing this until all productions have been expanded to the level of detail desired.

Example

As an example, consider the <program block> production in Figure 2. Since <program block> can expand into one of <procedure block>, <function block> or <exec-proc block>, the logical way to expand <program block> is using a menu (See Figure 3). We did not use the /DUPLICATION or /SEPARATOR qualifiers since the {program_block} placeholder is not a "list" type placeholder. We used the /NOAUTO_SUBSTITUTE qualifier because we do not want the next occurrence of the placeholder {program_block}, which could occur anywhere in the file, to be replaced in this expansion. On the individual menu items, we used the /PLACEHOLDER and /NOLIST qualifiers to specify that the items are placeholders that are not to be expanded as "list" placeholders. Since {program_block} is a required placeholder, as indicated by the { }, the menu item placeholder selected will take on this "required" attribute. See page 5-30 of [3].

Now that the {program_block} has been defined, we move on to one of the menu items. The definition of {procedure} is an example of a nonterminal placeholder. See Figure 3. The body of the placeholder came almost directly from the BNF production of (procedure_block). The keywords PROCEDURE and END-PROC were taken literally from the BNF. The [extdef] placeholder expands into (EXTDEF). The nonterminal (formal i-o params) expands into the two placeholders [formal_input_parameters] and [formal_output_parameters]. When expanded, each of these will contain the necessary keywords and additional placeholders. Since the EXIT (name list) phrase is optional, it has been defined as the optional placeholder [exit_phrase]. If [exit_phrase] is expanded, the required format will be inserted into the editing buffer. The body of the procedure will come from the required placeholder {program_body}. In contrast with the "menu" type body of {program_block}, upon pressing the expand key, the body of the {procedure} placeholder is copied directly into the current edit buffer. This placeholder is made up of terminal (END-PROC), optional ([exit_phrase]) and required ({program_body}) placeholders. Even though much of the placeholder body is optional, enough of the syntax of the CMS-2 language is presented to aid the programmer in the coding of a procedure body. White space should be added liberally to increase the readability of the resultant code according to one's standard programming style.

In developing the CMS-2 editing templates, we had to make a hard decision as to what level of the syntax we should provide support. At some point in the expansion, the implementor will want to stop providing templates and just have a placeholder expand into a

textual description of what is expected. In most cases, having IDENTIFIER as a terminal placeholder that expands to "A string of letters and digits beginning with a letter" was detailed enough. In cases where EXPRESSION was used in the BNF, as in the CONDITION of an IF statement or in the right hand side of an ASSIGNMENT statement, it was not clear as to what level to take the expression placeholder. What we decided to do was to change the BNF slightly; the EXPRESSION in the IF-condition was replaced by a nonterminal {if_condition}, which in turn expands to {expression} which expands to "A valid CMS-2 expression". The assumption here is that the programmer will know what is and is not a valid expression. The {if_condition} nonterminal was added to show the programmer what was expected but to allow him to enter whatever he thought was the proper thing. In some cases one can provide supplementary HELP information.

Tokens vs. Placeholders

LSE tokens provide a convenient method of placeholder expansion. Instead of descending through several layers of menus, the user can just type in a language keyword or a simple token (such as ASSIGNMENT), press the expand key; the user is placed at the same language editing level as if he had used all of the "normal" placeholder expansions.

Deciding whether to use a placeholder or a token to define language constructs is sometimes difficult. Since all language elements can be defined as placeholder, and placeholders provide a bit more flexibility than do tokens, we used the approach of first defining everything as placeholders. Tokens were added later for the convenience of the user of the editor. These tokens can be defined to "point" to the desired placeholder. See Figure 3 for how this was done with PROCEDURE.

The typical tokens are keywords of the language. For example, the token IF is defined to expand into the {if_statement} placeholder. Typing IF followed by the expand key inserts the same template into the editing buffer as does working down to {if_statement} placeholder and expanding it.

This leads to a little more structure in the template definition. One can have many different tokens defined as the same placeholder; both language keywords and other "special" tokens can make using the editor easier. Tokens that are not language keywords (such as, ASSIGNMENT or STATEMENTS) can be defined to insert a certain type of clause into the buffer or to bring up a menu. If one wants an assignment statement, instead of descending through the parse tree of {statement} ... => {simple_statement} => (set_phrase), one could just descend as far as {statement} ..., enter ASSIGNMENT followed by the expand key, and be looking at the expansion of the SET phrase. Figure 4 illustrates the case where the user could enter STATEMENTS, press the expand key, and get a menu of all possible simple statements. The user can then use the menu to choose the desired statement.

A token defined as a placeholder can also be used in a placeholder definition to provide default information. If the user does not want to use the default, he can expand the token and a placeholder or menu can show up in its place.

Size of the Template File

Much work can go into the generation of the language template file. This is, of course, dependent on the complexity of the language and on choices of how much syntax to support.

```

DEFINE PLACEHOLDER simple_statement -
  /LANGUAGE = CMS2
  /NOAUTO_SUBSTITUTE -
  /DESCRIPTION = "CMS-2 Simple Statement" -
  /TOPIC_STRING = "SIMPLE_STATEMENT" -
  /TYPE = MENU
    "begin_block"           /PLACEHOLDER/NOLIST
    "debug_phrase"         /PLACEHOLDER/NOLIST
    "direct_code_block"    /PLACEHOLDER/NOLIST
    "exec_phrase"          /PLACEHOLDER/NOLIST
    "for_block"            /PLACEHOLDER/NOLIST
    "goto_phrase"          /PLACEHOLDER/NOLIST
    "input_output_phrase"  /PLACEHOLDER/NOLIST
    "pack_phrase"          /PLACEHOLDER/NOLIST
    "procedure_call_phrase" /PLACEHOLDER/NOLIST
    "proc_switch_call_phrase" /PLACEHOLDER/NOLIST
    "resume_phrase"        /PLACEHOLDER/NOLIST
    "return_phrase"        /PLACEHOLDER/NOLIST
    "set_phrase"           /PLACEHOLDER/NOLIST
    "shift_phrase"         /PLACEHOLDER/NOLIST
    "stop_phrase"          /PLACEHOLDER/NOLIST
    "swap_phrase"          /PLACEHOLDER/NOLIST
    "vary_block"           /PLACEHOLDER/NOLIST
END DEFINE

DEFINE TOKEN STATEMENTS -
  /LANGUAGE = CMS2 -
  /PLACEHOLDER = simple_statement

```

Figure 4. Extracts from CMS2.LSE

We implemented three LSE editors for editing HELP files (.HLP), LSE template files (.LSE) and CMS-2 source programs (.CTS and .CS2).

The HELP editor is quite simple. It contains five placeholders and one token, and consists of approximately 80 source lines.

The LSE template editor contains 49 placeholder and 18 tokens, and consists of approximately 600 source lines.

The CMS-2 language editor contains 233 placeholders and 32 tokens, and consists of approximately 2700 lines. The BNF production rules consist of 381 left hand sides and 829 right hand sides. As noted earlier, we did not fully implement all of the CMS-2 syntax.

As seen from the CMS-2 example, a great deal of work is required to implement the LSE templates, both in the planning stages and in the template input.

PROVIDING LANGUAGE HELP

In addition to structuring the user input through series of expandable templates, LSE provides at least three ways to give the user help:

- Help with use of the keyboard,
- Help with terminal placeholders,
- Help with the language.

Help with the keyboard is provided automatically by LSE (see Chapter 3 of [3]; it requires no programming, and it is available by pressing the HELP (PF2) key. It gives information on the functional assignment of the keys in the keypad and the other defined keys.

Help with the meaning of terminal placeholders is given to the user when expanding a terminal placeholder (see pages 1-16 and 5-29 of [3]); it takes the text which has been put into the body of the placeholder and, instead of copying it into the buffer, it presents it as a tutorial message. A standard example of this is the message "A string of letters and digits starting with a letter" given upon expanding the IDENTIFIER terminal placeholder.

Language help can be viewed as a way of keeping a language user manual on line so that the user can easily request it during expansion of a placeholder or a token. If the help information has been previously placed into a help library with the keys specified as topic strings in the placeholder and token definitions, then it is simple for the user to get the help information by pressing the sequence GOLD, HELP (PF1, PF2) when the cursor is located on the placeholder or token of interest. Full screen help is provided using the VAX HELP facility. The user is given help initially for the token or placeholder at the cursor location, but can continue within the help library, and finally return to editing.

To create the language help information, one must:

- Create a HELP library,
- Specify the HELP library name in the DEFINE LIBRARY statement,
- Specify the library keys as the topic strings in the DEFINE TOKEN and DEFINE PLACEHOLDER statements.

In deciding how to create the language help information, one should decide on which tokens and placeholders require help and consider combining help for more than one placeholder or token into the same help item. The source for the help information can be the programming language users manual. Remember that the user probably has access to the user's manual. So, select enough information to remind the user of the purpose of the item; give parameters and calling sequences, if appropriate, and include examples.

An example of programming the language help for the CMS-2 programming language is indicated in Figures 3 and 5. Figure 3 shows LSE templates for the CMS-2 language program_block and procedure. The /TOPIC_STRING indicates the HELP key names. These key names are part of the HELP file shown in Figure 5. Because of the way the help file is created, if the user requests help for the program_block template, then additional help information will be indicated also for procedure_block and function_block.

For more information on HELP libraries, see pages LIB-6 to LIB-11 of [4]; for how to use the VAX Librarian to create them, see pages LIB-1 to LIB-3 of [4]; for associating the help information with the language, see the information on pages 5-24 through 5-32 of [3] for /HELP_LIBRARY and /TOPIC_STRING.

COMPILING THE PROGRAM

The user can compile the program without leaving the LSE editor. For the DEC-supported programming languages, this is as simple as typing COMPILE at the LSE> prompt. Furthermore, one can use the results of the compilation to locate compiler-found errors in the source program. Typing REVIEW after the compilation is finished (or COMPILE/REVIEW as one step) brings up a split screen in which the compiler diagnostics are at the top of the screen, and the source program at the bottom. LSE commands can then be used to locate the program errors and to correct them. (See pages 5-12 to 5-14 for the COMPILE command and page 5-71 for the NEXT ERROR command in [3].

```

1 PROGRAM_BLOCK
Program blocks contain the procedural statements of the
source program. Procedures, functions, and exec-procs
contain the statements that define the processing operations
to be performed by the object program.

2 PROCEDURE_BLOCK
The PROCEDURE and END-PROC statements delimit a procedure
within a system procedure. A procedure may be called using
a procedure call phrase or a procedure switch call phrase.

2 FUNCTION_BLOCK
The FUNCTION and END-FUNCTION statements delimit a function
within a system procedure element. A function must have at
least one input parameter and results in a single output value.
A function is called by coding the function name followed by
the actual input parameters enclosed in parentheses within
an expression.

```

Figure 5. Extract from CMS2.HLP

For the DEC-supplied languages, being able to review compiler diagnostics and easily make changes to the program source is based on the compiler writing a diagnostics file (of file type .DIA). The format of this file appears not to be documented.

For the CMS-2 compiler, error messages are written to a message file (of file type .MSG). Using the split screen editing features of LSE, we can put the message file into a buffer displayed in the upper half of the screen and view it while editing the program source in the lower half of the screen. What we lose is automatic location of next errors and the possibility of automatic correction. If we could have found documentation for the format of the diagnostics file, it would have been a simple matter to write a filter which would process the message file produced by the CMS-2 compiler and generate the appropriate diagnostics file.

To use the COMPILE command, one defines it in the DEFINE LANGUAGE statement when creating the language templates. In the case of the CMS-2 compiler, we only want to be able to compile sources of type .CTS (compile time system). So we defined the /COMPILE_COMMAND for the CTS language but not for the CS2 language. Further, we defined the compile command to execute a command file. The command file contains two DCL commands: the first is a SET COMMAND which defines the CMS2 verb; the second is the actual CMS2 command with the required compiler options. When the user types the COMPILE command, LSE spawns a subprocess to perform the compilation. The subprocess executes the command which is formed by appending the name of the file being edited to the indicated /COMPILE_COMMAND. In our command file, 'p1' receives the name of the edit file. Our users define the CMS2 verb in their login command files in the same way as the first line of the command file; but when LSE starts the subprocess to perform the CMS-2 compilation, that command definition is lost. See Figures 1 and 6 for examples of how we set up the COMPILE command for the CMS-2 compiler.

```

SET COMMAND DISK#PG11:[MTASSEXE,REV000]CMS2.CLD
CMS2 /OPT=X/COM=(CMP,*.CMP)/INC=(CS2,*.CS2) 'p1'

```

Figure 6. COMPILE_COMMAND file CMS2.COM

CONCLUSION

Implementing a language sensitive editor using LSE is a straightforward task. LSE provides for a full program creation environment by its convenient support of language help and compilation. One should prepare for its implementation as in any other programming. Having a set of production rules in a BNF form makes the template production simpler.

REFERENCES

- [1] Meyrowitz, N., and van Dam, A., "Interactive Editing Systems: Part II," Computing Surveys, Vol. 14, No. 3, September 1982, pp. 353-415.
- [2] User Handbook for CMS-2 Compiler, NAVSEA 0967-LP-598-8020, 30 May 1986.
- [3] VAX Language-Sensitive Editor User's Guide, Order No. AA-FY24A-TE, Digital Equipment Corporation, Maynard, Massachusetts, July 1985.
- [4] VAX/VMS Librarian Reference Manual, Order No. AA-Z419A-TE, Digital Equipment Corporation, Maynard, Massachusetts, September 1984.

VAX/VMS Application Performance

Louise Wholey
Measurex Corporation
Cupertino, CA 95070

Abstract

This paper reviews some performance tools available from DEC and third party vendors that were used to do application performance studies on the Measurex VAX/VMS VISION real-time process control application. DEC's SPM tool, especially the PC (program counter) sampler, was found to be most useful. The paper presents a useful technique for analyzing SPM PC sampler data for the entire program address space, including modules within Measurex's own shareable images. The shareable images include a set of user-written system services, which must be executed as a protected shareable image.

Introduction

This is a tutorial on how to do application performance. Both DEC and third party products will be examined for their utility in tuning the Measurex VISION real-time process control application. Special consideration is given to handling applications with shareable images. Useful tools, commands to use them and the results of those commands will be presented.¹

Measurex Application

The Measurex VISION application, which is the basis for undertaking a performance study, is a process control system consisting of about forty processes running as a process tree, most of which run at real-time priorities. Real time under VAX/VMS is defined as running at priority greater than or equal to sixteen. The main effect of real-time is that there is no working set adjustment. Thus the working set quotas determine the maximum number of physical pages in a process. In the VISION application, there are also three time-sharing processes, the block chain, video and button builders. These builders are CPU and I/O hogs.

One of the constraints is that each process is mapped to two Measurex sharable images. The performance of the individual modules within these shareable images is of interest. Since the application is written in C, each process is also mapped to several DEC run-time library shareable images. The internal behavior of DEC's run-time libraries cannot be changed to suit the application. What is significant for tuning is how much time is spent in them and which routines call the run-time library excessively.

¹Each section of the paper can be read separately. Figures are included at the end of the paper.

Objectives

The first step in such a study is to select a few specific objectives. For the Measurex VISION application, the first objective is that the entire system, including the operating system and all of its pool space, must fit in 5 1/2 MB of memory. Next, the CPU usage and execution speed of the builders are of interest. When the builders run, a control engineer sits at a video screen configuring a system to meet customer requirements. This may be a long time for a complex system. Each execution normally leaves behind audit trail files, which may be reused to configure the same or similar systems. The reruns from audit trail files often take hours to complete. Thus, a second and very important objective is to speed up the builders.

Finally, the throughput of the application is very dependent upon the speed of the interprocess communication services. Early in the performance studies, message services using interprocess ASTs were written to replace the slow mailbox services of VMS. Improvement in these services will speed up all phases of the application.

Strategy

Tuning requires a systematic approach. First, one must know what gains are desired. Then, one measures the performance of the current system, which might be the time to do a standard operation. While doing that measurement, one may look for "hot spots", that is the high resource usage, such as high CPU time, I/O or page faults and note unexpected behaviors. After improvements are made, the same operation needs to be remeasured to show that improvements have been made. For example, on VISION, the time to run a particular audit trail file might be ten hours. After changing the system the same audit trail run might take 8 hours, showing a twenty percent improvement.

Tools

The following sections describe some of the tools available. Included is a discussion of the merits of each tool and some of their idiosyncrasies. Sample commands are given to provide a starting point for performance studies. Details on the use of each tool should be obtained from the documentation.² The results of using the tools to study the VISION application are presented.

VAX/VMS Utilities

Some tools are shipped with the VAX/VMS operating system. They are available on every system and are generally quite useful.

Image Accounting

The VAX/VMS accounting utility may be used to look at gross measures in a large time-sharing system context. In image accounting mode, it produces start/stop times, image activation counts, working set values, page faults, execution times and I/O counts. This tool has not been useful for tuning the VISION application.

Show

Show is a very helpful VAX/VMS utility. Show Memory and Show System can be used to obtain a system overview in order to see if the system is adequately configured for the application. The command

```
§ SHOW MEMORY
```

generates a summary summary of how memory is being utilized. Figure 1 shows an example. Problems in the memory configuration, such as no free memory, swapped processes, few free blocks of pool, or shortage of swap or page file space should be fixed before application performance studies are begun.

```
§ SHOW SYSTEM
```

Show System's display contains a great deal of useful information. Figure 2 shows an example taken on the VAX/VMS environment running the video builder, VDB. High page fault activity stands out clearly; the page fault column can be read as a histogram. Processes with high page faults should be investigated. If the processes are part of the application to be tuned, that becomes part of the tuning effort. If they are competing processes, then some corrective action needs to be taken first, if possible.

I/O counts can be read the same way. In the example in Figure 2, one of the VISION builders, VDB, has a very high I/O count and rate.³ While one may wish to question how high is appropriate, a high rate *is expected* for this process

²All the manuals are listed in the references section.

³Relative rates can be obtained by dividing by CPU time, though I/O rate is normally given in units of I/O counts per elapsed time.

since its job is to build video frames on the disk. One other process shows a high I/O count. BDM, the bulk data manager, performs I/O to configure the application data region on the disk.

The physical memory column shows actual pages in use by every process. The value may be less than or equal to the corresponding process working set.⁴ If the number of physical pages is significantly less than the working set, then it may be possible to remove pages from the working set without causing many faults. The physical memory numbers include both process private and global pages.

CPU time should be referenced to see the relative significance of the numbers in the other columns. The process with the most CPU time could easily have the most page faults but may not be a problem. If a process does not have much CPU time and has a high number of page faults, it should be investigated further.

Monitor

Monitor is the tool distributed with the VAX/VMS system for use in system and application performance studies. Several monitor commands are particularly useful.

```
§ MONITOR PROCESS/TOPCPU
```

This command tells what processes are taking the most CPU time. If a process is not taking CPU time, then it is not normally a problem. Study of the top CPU users for VISION during the initial stages of testing on VAX/VMS showed the TIMER job to be the top CPU user. TIMER job consists of an AST routine that answers real-time clock interrupts and sends a message to the job's main code, which wakes up to see what work needs to be done. It was initially taking about 6 to 8 percent of CPU. After replacing VMS mailbox services with Measurex's message services, the TIMER process CPU usage dropped to about 2 or 3 percent of the CPU.

While mailbox message services may not be the most ideal means of communicating between and within processes on VAX/VMS, the control software was originally written for another system where such communication was based on message services. Writing efficient message services for VAX/VMS means the application design does not have to change. Moreover, the new message services are 95% coded in C and are, therefore, portable.

```
§ MONITOR PROCESS/TOPFAULT
```

The Top Fault display is an excellent tool to use to select the working set quota for each process. Processes that fault heavily need to be given larger working sets. If memory is short, some memory can be reclaimed from processes never seen among the top faulters, or that accrue physical memory slowly.⁵ When every process has an adequate working set, few page faults are seen.

⁴Working set is not shown in the Show System display.

⁵Show System may be an adequate tool to see this. Show Process/id=n/continuous run at image initialization is another choice.

\$ MONITOR MODES

Monitor Modes tells how much CPU time is going into useful work, which is normally user mode activity. The time on the interrupt stack and in kernel and exec mode is overhead. Interrupt stack time is hardware and software interrupt processing. Kernel mode is scheduling, page faulting and system services. Exec mode is RMS processing. The VISION application builder programs require a significant proportion of system activity since they do considerable file I/O and invoke many special system services.

SPM

SPM is a VAX/VMS layered product that includes a large number of reports and displays designed mainly for VAX/VMS system tuning. All of its capabilities are very useful, however, for application tuning. The two parts of the product used for this study are the system tuner and the PC sampler.

SPM is a complex and comprehensive product. It takes practice to learn how to use it effectively and to gain insights into interpreting the results.

SPM System Tuning

Valuable information on application behavior can be found in the extensive tabular, display and graphical reports of the SPM System Tuner. It provides a detailed overview of the impact of the application on the system. Process data is also available for working set and page fault analysis.

SPM System Tuning - Collection

Collection of data is done by starting a detached process.

```
$ SPM COLLECT = TUNE-  
    /CLASS=(ALL,PROCESS) -  
    /DISK=DUA0-  
    /DEV=(GP,CX,SC) -  
    /OUT=spm_tun.dat
```

The class modifier is required in order to get process data. The disk and device data collection should be altered to suit the environment being observed. A file naming convention turned out to be very useful for retrieving old files from nightly backup save sets. The fact that the tuner runs as a detached process means that it can be run easily on a single terminal system.

The tuner will run until requested to stop.

```
$ SPM COLL=TUN /STOP
```

Control y does not stop a detached process. The command

```
$ SPM COLL=TUN /NEW_FILE
```

is an alternate way to close the data file for analysis without stopping SPM.

SPM System Tuning - Analysis

Analysis of data can be done many ways. The following command will produce the four graphs indicated and a tabular output including data from all collection classes. Specific output and data file names may be used as shown.

```
$ SPM REP=LOG-  
    /GRAPH=(SUM,CPU,MEM,PAGE) -  
    /CLASS=ALL-  
    /OUT=spm_tun.rpt-  
        spm_tun.dat
```

Version 3 of SPM collects and can display a huge amount of data, but many graphs have no data in them. For example, idle devices and all VMS mailboxes are plotted even though they may have not done any I/O. The extra plots can be eliminated by the use of proper qualifiers and parameters on the command line. Finding out which graphs have the interesting information may require at least one run taking all graphs.

SPM System Tuning - Tabular Report

The tabular form in Figure 19 contains a wealth of information about CPU usage, page faulting, swapping, I/O, file system, and other system usage. For the VISION application the report shows 94% of memory is in use (highlighted at the top middle of Figure 19). Since the system has 6 megabytes total, then 5.6 MB is used by the application. That meets the original requirement.

A surprise lies under the CPU+IO Idle item in the CPU and I/O Overlap box of the Final Statistics output (right side, highlighted). The system shows about 24% of the elapsed time is totally unused. The CPU is idle and no I/O is taking place. The block builder is running to configure customer block chains (control sequences) from an audit trail file. The expectation is that system should be very busy, pausing from CPU busy to do disk I/O and update the video screen, which displays the current audit trail command line. The display is an operator graphics console with no hardware scrolling function. The pauses in CPU and I/O usage are the result of the time taken by the graphics chip to refresh the screen several times as it emulates scrolling. The refresh consists of sequentially moving the old lines up on the screen and then inserting the new one at the bottom. No one had thought about the effect of this behavior on throughput. Saving twenty percent of the wall clock time means a five hour run would take only four hours. Since all of the commands are also written into a log file, the display only needs to indicate that the builder is still running.

SPM System Tuning - Graphical Reports

A variety of graphs are available from SPM. A few of them are included in this paper to illustrate specific findings, but the SPM manual set should be referred to for a complete coverage.

The first example, Figure 3, is the CPU utilization graph. The plot shows a sudden cut-off when the application and everything running on the system stopped. A look at the disk

allocation graph in Figure 4 shows why - the disk became 100% allocated. Figure 5 shows the high cost of running out of disk space in terms of the I/O rate as the disk approaches full. Also, the tabular report shown in Figure 19 shows a very low file cache hit rate for the bitmap cache. Considerable extra I/O was required to allocate space.

The page fault graph in Figure 6 shows the page fault behavior plotted by time. Image activation causes many page faults in a short time in order to fault into memory the image and data. The graph shows that the working sets are completely adequate since there are almost no faults after image activation.

SPM PC sampling

The SPM PC sampler is the best tool for a finely detailed analysis of where a program is spending its time. It is currently the only tool for analysis of exec mode and kernel mode activity. The technique described here enables looking at the CPU time used by all the modules within the program. For VISION that includes the main program, the Measurex shareable images, and the system.

SPM PC Sampling - Collection

Unlike the tuner, the PC sampler does not detach from the terminal. Thus, either an extra terminal must be dedicated to collection, or a spawn/nowait command may be issued. If using spawn/nowait, a control y to stop the main process will also stop the collector. The sample command

```
$ SPM COLL=SYS /END="+5" spm_pc.dat
```

causes SPM to collect for 5 hours using the output data file spm_pc.dat. One advantage the PC collector has over the tuner is that the data file may contain a node name. Running VISION for 5 hours produces about 50,000 disk blocks of data. Since the microvax is usually short of disk space, the data is normally sent over ethernet to a VAXcluster disk. NETACP is set to priority 18 to keep the net up.⁶ Another way to solve the file size problem is to pre-select the process on which to take samples. The collector filters the samples, collecting only for the requested process.

```
$ SPM COLL=SYS /END="+5" /ID=pid  
spm_pc.dat
```

The selected process' pid is first obtained from Show System.

SPM PC Sampling - Analysis

There are two primary forms that the analysis output from the SPM PC sampler can take, a system-wide view and a detailed analysis for a given process. Both analyses start the same way.

```
$ SPM REP=SYS /OUT=spm_pc.rpt  
spm_pc.dat
```

⁶SWAPPER and REMACP are also run at priority 18.

This command produces a typically voluminous report. Parts of such a report are shown in Figure 14. The summary block at the top of the first page tells the start/stop times for the run. Below it on the left is a list of processes with their PID's and, on the right, a list of drivers and other loadable system code. The load addresses for RMS and the emulation code for the microvax can be used to define address buckets (discussed later) for these modules.

The next page of output, entitled Processor Usage by Process, shows, on the left, what percent of all the samples were for a particular process. The big users in Figure 14 are the block data manager, bdmmain019, that writes system configuration data into a disk file, and the block builder, BLK-BLD019. Another surprise is that the block builder takes only half as much CPU time as the data manager program. Thus, the focus for tuning the block builder has to include the BDM program. The last sample page is part of the System Module Usage report, which lists the percent of time spent in the various system modules. This part of the report includes the null process, drivers, RMS as a whole, and the microvax emulation modules, though not all of them are shown in the sample.

Process Address Space Layout

For the *detailed* analysis of SPM PC samples, one needs to define address space layout of each process. The address space of VISION processes is drawn in Figure 7. A shared application data area is located at the lowest addresses, followed by program code, then shareable images, P1 space, and S0 space. The data is located below the code because the application software uses absolute addresses to reference the data.⁷ The data addresses in the application region are the same for all processes. Since code is position independent, the shareable image code will start at a unique address for each process depending on the size of the program code.

The Measurex shareable images, identified as VOS and USS, are always loaded first by the image activator. The address space analysis technique presented in this paper depends on this. The DEC run-time libraries (RTL's) are grouped as an undifferentiated unit covering the remainder of P0 space. P1 space has very little code; the transfer vectors to VMS system services and RMS are there, as well as is the XQP file processing code.⁸ Finally, S0 space is the VMS code. The SPM distribution kit provides the definition of the layout of the VMS address space in a command procedure named SPM-SYSTEM.COM, which generates the file SPMSYSTEM.DEF.

The VISION application includes a set of user-written system services, the USS shareable image, which can only be executed as an installed protected shareable image. Special link arrangements that change shareable image code to non-shared code cannot be done for the privileged code in user-written system services. Thus the special technique presented here for handling shareable image address space evolved due to this constraint.

⁷VISION was originally coded for an unmapped system.

⁸File processing includes directory lookups, enters, removes, file creates, opens, closes, changes, and volume space allocation.

Conversion to Image PC Samples

Generating a histogram of CPU time spent in these various address areas of the program, requires a total of five analysis steps. Two steps have already been covered, collecting PC samples and system wide analysis. The third step is to convert system wide PC sample data to image PC samples. The command

```
$ SPM CONVERT=SYS -  
  /ID=pid -  
  /OUT=xxx.pcs -  
      spm_pc.dat
```

causes the conversion to image PC samples for a process named xxx whose PID is "pid". The output is xxx.pcs, the image PC sample file.

Program Address Space Definition for SPM

The fourth step is the definition of address space buckets (named address ranges) for each VISION process to be analyzed. SPM can read the link map for the address ranges of all the modules located in the psect \$CODE. An excerpt from a map showing the \$CODE psects is in Figure 8. For the analysis of each VISION process, SPM has to read the main program link map, the VOS and USS shareable image maps, and the system definition file supplied with spm, spm-system.def (renamed sys.def for easy typing).

The sequence of operations to acquire and manipulate all of the data is complex. A command procedure, INQUIREBK-TDEF.COM, was written to make it easier to do repetitively. A copy of the procedure is included in Figure 15. It is invoked to carry out the fourth step.

```
$ @INQUIREBK-TDEF
```

The procedure reads the process map xxx.map, the VOS and USS shareable image maps, and the spm-supplied file sys.def (a copy of spmsystem.def). It requests the user to enter some data displayed from the maps, from which it calculates offsets to the VOS and USS. The procedure builds an image definition file according to the SPM address space definition language. The procedure includes the final SPM step of translating the image definition data into an image bucket file:

```
$ SPM DEFINE=IMAGE image.def  
  /buck=xxx.bkt
```

The output file, xxx.bkt, contains the process module address buckets for the final step in the analysis.

```
$ SPM REP=IMAGE-  
  /BUCK=IN=xxx.bkt-  
  /OUT=xxx.rpt-  
      xxx.pcs
```

The result is a video display of the PC histograms, for process xxx, which can be stepped through by entering carriage return or "killed" by typing K. The file xxx.rpt can be printed for detailed study.

In principle, any number of shareable images can be configured this way. The problem with a general implementation of this technique is knowing where in the process the image activator will place each shareable image. This information is unavailable at link time. One technique to solve this problem is to write a program to read and report the image activator scratch area in P1 space to get the starting and ending address of each shareable image for the process. Iac\$gl_image_list points to each shareable image descriptor; offset 72 is the starting and 76 is the ending address of the shareable image.⁹ Another technique is to code the shareable images to report their address ranges after activation, by including data definitions and code to do the reporting.¹⁰

BLKBLD Program Histogram

The PC histogram for the block builder program is shown in Figure 16. The SPM output has been edited for display purposes, leaving "..." where text was removed. There were twelve pages of modules in the BLKBLD main program, each showing PC samples accounting for less than 1% of the total CPU time. The scale makes one asterisk equal to 645.7 counts.

The transfer vectors for each of the shareable images are shown separately, just before the histogram for the shareable image with the same name. First the VOSXFR and the VOS, the non-privileged shareable image are shown, then the USSXFR and the USS, the user-written system services. After the USS is the remainder of P0 space identified as Program RTL. This includes the VAXCTRL, FORRTL, LIBRTL, MTHRTL, and SMGSHR.

The last part of the histogram shows P1 space, the location of the file primitives and the transfer vectors for system and RMS services. After P1 space is an edited view of system space, identified as PROGRAM SYS. This is S0 space.

BLKBLD Program Results

This PC histogram shows two spikes of high PC counts. One is for the module SYSENQDEQ and the other is for module ZZZ. Two modules are labeled SYSENQDEQ; together they add up to 22.6% of the CPU time. These are lock services. The other high CPU use is in module ZZZ. The address definition for ZZZ covers the remainder of S0 space beyond the parts defined by SPMSYSTEM.DEF. If this address range is compared to the addresses shown in the SPM PC sampler overview report in Figure 14, the range is seen to correspond to loadable parts of the system: RMS code, device drivers, and microvax emulation code. The high time in this module is likely to be RMS execution.

BLKBLD program improvements - Record locking

Clearly the block builder can run faster if the locks are removed. The reason for the locks is that a file of common

⁹See the fiche on Debug. Remember, though, it may change, since it is undocumented information.

¹⁰The data psect attributes must be manipulated to cause proper sort by the linker.

symbol tabel data is shared among several programs. The programs are actually doing block I/O, randomly accessing fixed length 512 byte records, but to have locking done, they were coded as record I/O. A quick check with the system architect revealed that the need for VMS to do the locking has vanished. Other computer systems running the VISION software do not have built-in record locks; thus, the programs have do their own locking.

By setting the user-provided interlock bit (UPI) in the FAB shared access request field,¹¹ the record locking is removed. The result is shown in figure 9. The counts in the SYSENQDEQ modules have dropped to 1.5%, a saving of 20% of the machine by not having these locks.

The SPM System Tuner shows the effects of the lock removal in Figure 18. The System Summary graph was the result of running for a few hours with the original VOS shareable image that included record locking, then stopping the application, changing the VOS to eliminate locks, and restarting the application. Less CPU time is used after the change and more I/O is done for the same elapsed time.

BLKBLD program changes - Block I/O

Baited by such success, attention turned to the spike for ZZZ in program SYS. If ZZZ is mostly RMS record I/O activity, and the only reason to use record I/O is to have locks, then the record I/O can be replaced by block I/O. The result is shown in Figure 10. The most obvious result if the focus remains on ZZZ is that the CPU usage in that code dropped from 23 to 15%. That sounds like an 8% saving for block I/O.

The machine, however, is not being used in the same way as it was earlier. Program P1 now shows 20% of the CPU time, up from 6% before locking was removed and 10% in the study with record locks removed.¹² The throughput is probably higher with block I/O, but an elapsed time study needs to be done to verify that. It is also possible that some of the internal buffering by RMS for record I/O was lowering the number of I/O requests. Only a throughput study will produce the answer as to which uses the machine more efficiently.

MUDM Results

Other processes in the VISION application were observed with the SPM PC Sampler histogram technique. The results for MUDM, the memory update manager, are shown in Figure 17. This program receives requests from other processes to periodically update memory locations in the shared data region. It receives clock tokens (a special type of message) from the TIMER job indicating a certain time period has expired. For MUDM the histogram shows that most of the time is spent receiving messages.

¹¹FAB\$M.UPI in FAB\$B_SHR

¹²The increase from 6 to 10% was not noticed earlier by the author and does not appear on the Figure 10 histogram.

Message service modules

The MUDM data shows a surprisingly large amount of time spent in the message service modules. Earlier studies had shown the services to be faster than these graphs indicate. A The USS link map for this run reveals that MUDM is running message service modules coded in C instead of Macro. Some of the modules were recoded for optimal performance on the VAX. After changing the object module library to include the macro routines, and relinking and reinstalling the USS, the application showed a 2 to 3% CPU saving for every process. This oversight may have gone unnoticed for a long time without the help of SPM.

Module names vs. entry point names

The key to making sense from the CPU spikes in system space is to know the names of modules in the system. System programmers are more likely to recognize entry point names than module names. A conversion between the two is helpful. Sorting the system map file, SYS\$SYSTEM:SYS.MAP, by module name produces a listing of all the entry points in a given module.

This technique was quite helpful for the MUDM study, where a spike (14% of the CPU) appeared for the module EXSUBROUT. Figure 11 shows an excerpt from the system map showing all the entry points in EXSUBROUT. The timer queue insertion and deletion modules, INSTIMQ and RMV-TIMQ, are probably where the CPU time is being spent. The receive message services include an option to time out a receive message request after a selectable time period.

The other peak in the MUDM Program SYS histogram is in CMODSSDSP which has 12.8% of the PC hits. This is the change mode dispatcher, the routine that transfers control to kernel or exec mode for both built-in system services and user-written system services. The improvement indicated by these results is to code the the message services to stay in kernel mode and call the timer routines directly, rather than separately call the Measurex message services and the VMS timer services.

BDM Results

The results for the process BDM, the bulk data manager, shown in Figure 12, present quite a different picture of where CPU time is being spent. BDM spends 77% of the time in the run-time library (Program RTL) area. Tuning this process can be done by finding out what statements in the program cause the calls to the run-time libraries. Measurex has acquired two tools, PCA and I-MON, that can determine what is calling the run-time libraries by looking at stack frames.

Stack Traceback Tools

Both PCA and I-MON have the capability of doing stack traceback for PC samples which lie outside the main program address space. Using this type of facility, PC samples in shareable images and system space are reported as if they had oc-

curred in the main program code. This is done by scanning stack frames until an acceptable address is found.

PCA

PCA is a VAX/VMS layered product designed for application performance and coverage analysis. It can do a variety of sampling operations on a process:

- PC sampling
- PC addresses of page faults
- System service calls
- File I/O
- Coverage and execution counts
- Chargeback of VMS and RTL calls

PCA works as a debugger. It requires special linking of the program with a debugger in order to have the debug symbol table available.

PCA - Preparation

To prepare for the use of PCA, the following compile and link commands are used:

```
$ compile /DEBUG program

$ LINK /DEBUG=SYS$LIBRARY:PCA$OBJ.OBJ
      program
```

The preparation phase is awkward for VISION. Most of the images are too big to be compiled and linked with the debugger. The block builder image, for example is, nearly 1600 blocks. With the debugger used for the link step only, it grows to 5500 blocks. Microvax disk space is a problem. There are some variations on preparation that help.

First, if the program has been previously linked with the any debugger, the logical name LIB\$DEBUG can be defined to be SYS\$LIBRARY:PCA\$COLLECTOR.EXE. This causes PCA to be selected as the debug module at image activation time. In addition, if the source has not been compiled with debug, PCA can still be used. Rather than have source line information, the data is charged by module, which is adequate for the VISION study.

```
Module data if link /debug
Source code line if compile /debug
```

PCA - Collection

The program being observed is run by the PCA collector in the same way DEBUG runs a program. The following example starts collecting PC samples with the stack traceback facility enabled. The program for which collection is being done can also be a subprocess, as is the case in VISION.

```
$ RUN program
PCAC> SET PC_SAMPLING
PCAC> SET STACK_PCS
PCAC> SET DATAFILE file
PCAC> GO
```

The collection stops on program completion or termination by any means, including control y.

PCA - Analysis

The PCA analyzer needs to be told what to do. The following command tells it to report pc sampling, to trace samples back to the main module, to display only address buckets containing some data (/nozero), and to view the data by module.

```
$ PCA file
PCAA> PLOT /PC_SAMPLING -
      /MAIN -
      /NOZERO -
      PROGRAM_ADDRESS BY MODULE
```

This command was used on the BDM process after re-linking the image with the debugger and collecting samples as indicated.

BDM Results

The results of using PCA on BDM are shown in Figure 13. The author of BDM had speculated that the activity causing 77% of the PC samples to fall in the RTL area would be disk read and write requests. Disk writes account for half the activity, while reading and deleting file space are another 33%. The remaining 18% is associated with receiving messages. There are no surprises here. The obvious way to improve this program is to decrease the amount of I/O it does. That is being done.

PCA - Pagefault analysis

If page faults had been a problem, PCA could have been used to do a detailed analysis of what part of the code causes page faults. The request

```
PCAC> SET PAGE_FAULT
```

would collect page fault data.

PCA and DEC's Software Productivity Tools

PCA is part of Digital's VAXset collection of integrated software productivity tools, which includes LSE, SCA, CMS, MMS, DTM and PCA. PCA is expected to assist in finding execution "hot spots", such as excessive activation counts or inefficient modules, and to determine whether all parts of an application have been executed during testing.

I-MON

I-MON is a product of Bear Computer Systems. It is designed to be very easy to use. I-MON allows the user to avoid the problems associated with other tools that require a special program link; it uses the trace tables in the executable image to define sampling buckets. This is completely transparent to the user. In addition, dynamic features of the display allow the user to zoom in on program detail when a problem is seen.

The following command will start I-MON collecting data on the process identified by pid.

```
$ IMON /ID=pid /SELECT=com
```

I-MON begins by doing PC sampling with data displayed by module. When a problem is seen on the terminal display, the user can dynamically expand the view of the module to see the instructions displayed by line number or hex address. Alternatively, the tool can show the caller a heavily used module. I-MON will log all terminal activity to a file for convenient playback or hard-copy of any session. It can also gather data directly into a file, bypassing the terminal, by using /OUT=file /SAMP=2000 qualifiers on the command.

Summary

The results that have been presented in this paper indicate that progress has been made toward the original performance objectives. The VISION processes fit in 5.6 MB of memory with essentially no page faults except during initialization. The block builder runs faster by 20-40%. The macro message services were restored, saving 2-3% for every process. There is also a possibility for further improvements by changing the timeout code in the receive message services. The work is not yet complete; other builders require further study and the system needs to be studied during process control activity.

Block Builder Results

The screen refresh on the graphics terminal will be altered to save about 20% elapsed time. Record locking is eliminated saving another 22% of the cpu. Block I/O vs. record I/O elapsed times still need to be measured. If record I/O happens to be faster, then RMS buffering can be examined for further tuning of the file I/O.

BDM Results

BDM, the bulk data manager, is the highest user of both CPU time and I/O resources during block builder tests. Lowering the number of I/O's appears to be the only way to improve its throughput. There are plans to have the block builder do some internal buffering so that it does not request as many I/O operations from BDM. Additionally, a future hashing scheme may cut down drastically on BDM's I/O rate.

Tools Evaluation

SPM has proven to be a valuable tool for finding unexpected behaviors in programs. The SPM PC sampler is extremely

valuable, though parts of the output may require assistance from a system programmer for a complete analysis of the application. The SPM System Tuner's results are of use to both programmers and system managers. PCA or I-MON should be regularly used by the developers of each program during the integration stage of development to be sure the program behaves as the author intends. The problems uncovered by using these tools enable programmers to put their efforts into the right places.

Acknowledgements

I would like to thank Mary Tremaine for preforming the miracle of transcribing the tape of my talk at DECUS.¹³ Without that effort this paper would not have been published. In addition, I would like to express my gratitude to my boss, Ron Lau, for making it possible for me to do the performance studies and for allowing me some company time to complete the final editing of the paper. Finally, I want to thank my wonderful family, my husband Jim and daughter Mary Wholey, for being tolerant of my absences during preparation of this material.

¹³I said everything included in this paper three times!

References

VAX/VMS DCL Dictionary, VAX/VMS Version 4.4, AA-Z200C-TE, Digital Equipment Corporation, April 1986, Show Memory, p. DCL-567, Show System, p. DCL-598.

VAX/VMS Monitor Utility Reference Manual, VAX/VMS Version 4.4, AA-Z423B-TE, Digital Equipment Corporation, April 1986, Monitor Modes, p. MON-62, Monitor Processes, p. MON-69

Guide to VAX SPM, VAX SPM Version 3.0, September 1986, AA-G139A-TE, Digital Equipment Corporation, September 1986.

VAX SPM Reference Manual, VAX SPM Version 3.0, AA-R580C-TE, Digital Equipment Corporation, September 1986.

VAX PCA User's Reference Manual, VAX Performance and Coverage Analyzer Version 1.1, VAX/VMS Version 4.4, AA-EB54A-TE, Digital Equipment Corporation, August 1986.

I-MON, VMS Image Monitor, Version 4.0, Bear Computer Systems, North Hollywood, CA, 1985.

VAX/VMS Guide to VAX/VMS Performance Management, VAX/VMS Version 4.0, AA-Y515A-TE, Digital Equipment Corporation, September 1984.

VAX/VMS Internals and Data Structures, Lawrence J. Kenah and Simon F. Bate, Digital Press, Burlington, MA, 1984. For VAX/VMS Version 3.

VAX/VMS Internals and Data Structures, Version 4.4, Lawrence J. Kenah, Ruth Goldberg and Simon F. Bate, Digital Press, Burlington, MA, Preliminary and Partial Edition, 1987.

System Memory Resources on 16-APR-1987 07:02:57.07

Physical Memory Usage (pages):		Total	Free	In Use	Modified
Main Memory (9.00Mb)		18432	4192	14213	27
Slot Usage (slots):		Total	Free	Resident	Swapped
Process Entry Slots		60	21	39	0
Balance Set Slots		58	21	37	0
Fixed-Size Pool Areas (packets):		Total	Free	In Use	Size
Small Packet (SRP) List		1000	627	373	96
I/O Request Packet (IRP) List		1000	790	210	208
Large Packet (LRP) List		25	14	11	1584
Dynamic Memory Usage (bytes):		Total	Free	In Use	Largest
Nonpaged Dynamic Memory		349696	111520	238176	104272
Paged Dynamic Memory		276992	79024	197968	78064
Paging File Usage (pages):		Total	Free	In Use	Total
DISKSUDWL:[SYS0.SYSEXE]SWAPFILE.SYS		11304	21696	33000	33000
DISKSUDWL:[SYS0.SYSEXE]PAGEFILE.SYS		9929	8071	18000	18000

Of the physical pages in use, 3069 pages are permanently allocated to VMS.

Figure 1: Show Memory

```

VAX/VMS V4.5 on node SCSI 16-APR-1987 07:02:48.45 Uptime 1 13:41:25
Pid Process Name State Pri I/O CPU Page flts Ph.Mem
00000040 NULL COM 0 0 1 01:50:18.46 0 0
00000041 SWAPPER HIB 18 0 0 00:00:13.89 0 0
00000102 OPIVD4019 LEF 19 26 0 00:00:03.10 603 397 S
00000103 OPIVD5019 LEF 19 26 0 00:00:03.04 602 397 S
00000044 JOB CONTROL HIB 9 870 0 00:00:04.66 145 269
00000045 ERPFMT HIB 8 1207 0 00:00:11.43 70 93
00000046 OPCOM LEF 8 294 0 00:00:02.69 1972 61
00000047 NETACP HIB 18 748 0 00:00:26.25 289 209
00000048 EVL HIB 6 55 0 00:00:13.42 29827 47 N
00000049 PEMACP HIB 18 151 0 00:00:00.69 80 55
000000CA OPILP 019 LEF 19 28 0 00:01:29.01 1642 475 S
000000CC VDEJOB019 COM 4 670583 0 01:06:47.70 1236 1100 S
0000010E SPM TUNE LEF 24 3389 0 00:00:45.50 178 300
000000D4 BATCH 70 LEF 4 5540 0 00:00:32.96 2434 946 B
000000D5 mlmain171 LEF 8 67 0 00:00:04.39 655 491 S
000000D6 bdmmain171 LEF 5 298519 0 01:19:41.43 1025 893 S
00000097 019VOS HIB 19 1874 0 00:00:22.70 3072 945
000000D8 BLKBLD171 LEF 5 79175 0 00:06:54.15 1511 2268 S
000000A5 mlmain019 LEF 17 66 0 00:00:04.22 653 495 S
000000A6 mlmain019 LEF 17 32 0 00:00:03.74 608 450 S
000000A7 ml3main019 LEF 17 34 0 00:00:03.78 611 453 S
000000A8 timmain019 HIB 20 70058 0 00:16:01.60 761 375 S
000000A9 bdmmain019 LEF 16 24646 0 00:06:13.29 1236 1000 S
000000AA datmain019 LEF 20 43 0 00:01:26.58 3035 625 S
000000EB mudmain019 LEF 19 69 0 00:02:49.87 849 649 S
000000EC PLOTSK019 LEF 18 34 0 00:00:17.33 654 515 S
000000ED kbmain019 LEF 18 33 0 00:00:03.15 953 425 S
0000006E LTAL: 019 CUR 4 16521 0 00:08:11.38 22890 389
000000EF GPI 019 LEF 19 32 0 00:00:03.21 629 441 S
000000F0 vct019 LEF 19 30 0 00:00:03.03 676 501 S
000000F1 vgc019 LEF 19 36440 0 00:00:54.62 627 397 S
000000F2 alarmmain019 LEF 19 26 0 00:00:03.85 582 414 S
000000F3 VIDEO_4019 LEF 19 269 0 00:01:20.83 965 475 S
000000F4 symmain019 LEF 19 49 0 00:00:04.06 662 496 S
000000F7 VIDEO 5019 LEF 19 27 0 00:00:03.15 876 475 S
000000F8 spimaIn019 LEF 17 71 0 00:00:04.74 740 580 S
00000079 003VOS HIB 7 1474 0 00:00:15.75 2333 350
000000FA OPIVD2019 LEF 19 27 0 00:00:03.09 604 398 S
000000FB OPIVD3019 LEF 19 26 0 00:00:03.09 604 399 S

```

Figure 2: Show System

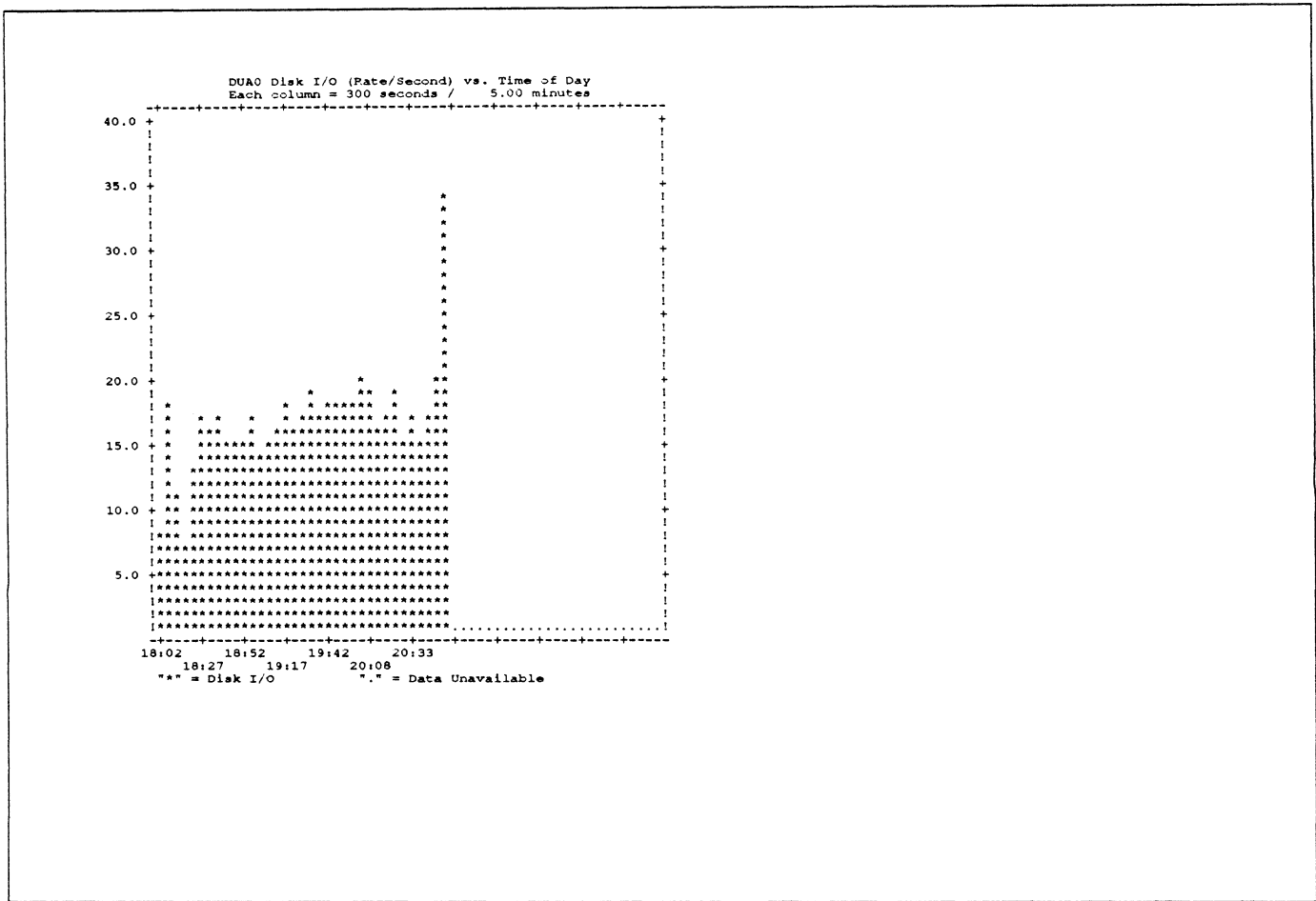
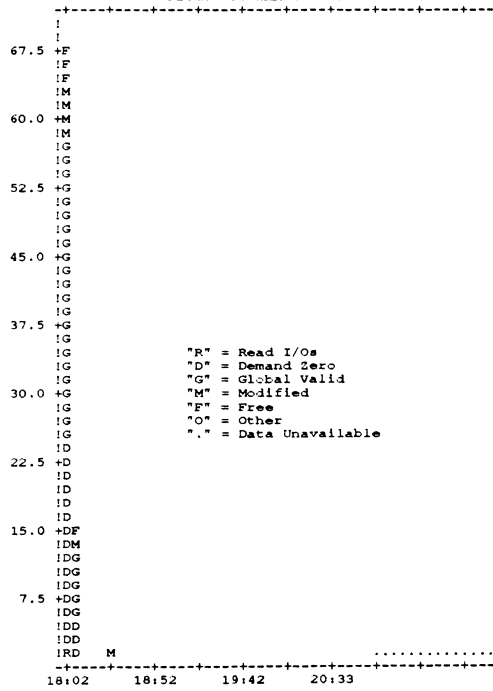


Figure 5: Disk I/O Rates Report

Page Faults (Rate/Second) vs. Time of Day VAX SPM V3.0-01
 From: 16-MAR-1987 18:02:30.16



"R" = Read I/Os
 "D" = Demand Zero
 "G" = Global Valid
 "M" = Modified
 "F" = Free
 "O" = Other
 "." = Data Unavailable

Figure 6: Page Fault Rate Report

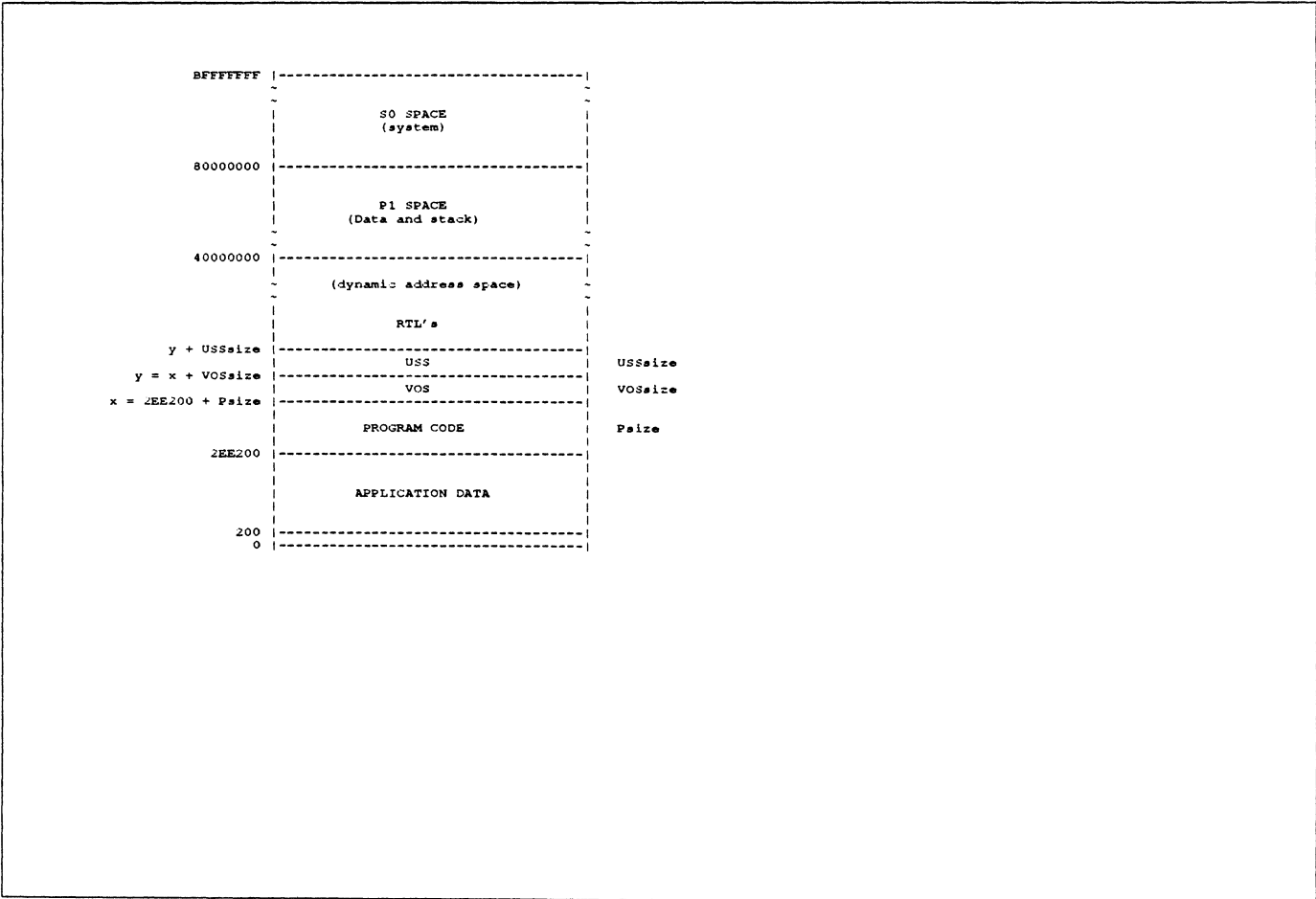


Figure 7: Program Address Space

Psect Name	Module Name	Base	End	Length	Align
-----	-----	---	---	-----	----
\$CODE		00241000	002427F5	000017F6 (6134.) BYTE 0
	TIMMAIN	00241000	0024136A	0000036B (875.) BYTE 0
	TIMGLOBAL	0024136B	0024136B	00000000 (0.) BYTE 0
	CLOCK	0024136B	0024136B	00000000 (0.) BYTE 0
	SETCLOCKS	0024136B	00241593	00000229 (553.) BYTE 0
	READVMSCK	00241594	0024171B	00000188 (392.) BYTE 0
	ADDSIGNAL	0024171C	002419DF	000002C4 (708.) BYTE 0
	SUBSIGNAL	002419E0	00241ABE	000000DF (223.) BYTE 0
	ADDELDTIM	00241ABF	00241BF3	00000135 (309.) BYTE 0
	SUBDELDTIM	00241BF4	00241CBE	000009CB (203.) BYTE 0
	SNDTOK	00241CBF	00241E38	0000017A (378.) BYTE 0
	SNDMSG	00241E39	00241EC8	00000090 (144.) BYTE 0
	CLKUPD	00241EC9	00241FBE	000000F6 (246.) BYTE 0
	FILEUPD	00241FBF	0024209D	000000DF (223.) BYTE 0
	CHECKTIME	0024209E	002421FE	00000161 (353.) BYTE 0
	MISCFGNS	002421FF	00242204	00000006 (6.) BYTE 0
	SETGMT	00242205	00242278	00000074 (116.) BYTE 0
	SETLOCOFF	00242279	002422B4	0000003C (60.) BYTE 0
	CUTCCK	002422B5	002422F0	0000003C (60.) BYTE 0
	SNDCLKTOK	002422F1	0024231D	0000002D (45.) BYTE 0
	NUMSEC	0024231E	00242428	0000010B (267.) BYTE 0
	LOADCLOCK	00242429	002427F5	000003CD (973.) BYTE 0

Figure 8: \$CODE Psect part of map

Removed Record Locking
 PROGRAM BLKBLD BY MODULE
 PROGRAM SYS BY MODULE

```

+-----+
SYSWAIT |
8000B60A:8000B6E5 |** 0.84%
SYSENQDEQ |
8000BC1C:8000C97C |** 1.09%
IOSUBNPAG |
8000CD71:8000D966 |** 1.13%
IOSUBRAMS |
8000D967:8000DB03 |***** 2.25%
LOADMREG |
8000DB04:8000DC8E |** 0.78%
CMODSSDSP |
8000FC00:8000FD0C |* 0.52%
8000FD78:8000FFDF |***** 3.54%
. . .
SYSENQDEQ |
800143A6:8001465C |* 0.38%
. . .
ZZZ |
8002904F:BFFFFFFF |***** 23.81%
+-----+
  
```

Scaling: 418.32 counts/asterisk

Figure 9: Image PC Histogram

```

PROGRAM BLKBLD BY MODULE
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
BLOC
00268400:0026844C |*****| 0.02%
GNMREF
0030C07E:0030C194 |*****| 2.37%
GSYMD
0030C68B:0030C801 |***| 1.20%
BFILES
0031EA00:0031F347 |***| 1.05%

PROGRAM P1
-----+-----+-----+-----+-----+-----+-----+-----+-----+
P1
40000000:7FFFFFFF |*****| 20.49%
-----+-----+-----+-----+-----+-----+-----+-----+-----+
Scaling: 671.84 counts/asterisk

PROGRAM SYS BY MODULE
-----+-----+-----+-----+-----+-----+-----+-----+-----+
IOLOCK
80006AFF:80006D8C |*****| 2.63%
SYSACFFDT
80008B11:8000925B |*****| 1.88%
SYSQIOREQ
800098AA:80009CDB |*****| 3.45%
ASTDEL
80009D90:8000A040 |*****| 4.57%
SYSENQDEQ
8000B1C1:8000C97C |****| 1.56%
IOSUBRAMS
8000D967:8000DB03 |*****| 2.71%
CMODSSDSP
8000FC00:8000FDOC |*| 0.56%
8000FD78:8000FFDF |*****| 3.64%
ZZZ
8002904F:BFFFFFFF |*****| 15.08%
-----+-----+-----+-----+-----+-----+-----+-----+-----+
Scaling: 671.84 counts/asterisk

```

Figure 10: Image PC Histogram of BLKBLD

-----+
 ! Symbol Cross Reference !
 -----+

Symbol	Value	Defined By	Referenced By ...
EXE\$CLEANUP_ORB	800104B4-R	EXSUBROUT	LNMSUB
EXE\$CHKDELAACES	80010453-R	EXSUBROUT	
EXE\$PROBER	8000A78A-R	EXSUBROUT	SYSCHKPRO
EXE\$CHKWRTACCES	80010467-R	EXSUBROUT	MBDRIVER
EXE\$CHKPHYACCES	80010458-R	EXSUBROUT	SYSQIOREQ
EXE\$PROBEW	8000A7D3-R	EXSUBROUT	EXCEPTION
EXE\$PROBEW_DSC	80010500-R	EXSUBROUT	SYS\$IMGACT
EXE\$INSTIMQ	80008A78-R	EXSUBROUT	SYS\$CHEVT
EXE\$CHKCEACCES	80010458-R	EXSUBROUT	
EXE\$PROBER_DSC	800104FC-R	EXSUBROUT	SYS\$IMGACT
EXE\$CHKRDACCES	8001045D-R	EXSUBROUT	IOSUBNPAG
EXE\$BUFQUOPRC	8000A73D-R	EXSUBROUT	
EXE\$CHKLOGACCES	80010453-R	EXSUBROUT	SYSQIOREQ
EXE\$PRTIMQ	80009A9F-R	EXSUBROUT	SYS\$CANVT
EXE\$NAXACMODE	800104ED-R	EXSUBROUT	COMDRVSUB
EXE\$CHKXEACCES	80010462-R	EXSUBROUT	
EXE\$MULTIQUOTA	8000A756-R	EXSUBROUT	
EXE\$BUFFRQUOTA	8000A731-R	EXSUBROUT	SYSACPFDT
EXE\$SNGLEQUOTA	8000A753-R	EXSUBROUT	SYSQIOREQ
EXE\$VAL_IDNAME	80010545-R	EXSUBROUT	SYSRDBRES

Figure 11: System Map Excerpt

```

PROGRAM BDMMAIN BY MODULE
BDMMAIN          +-----+
00253600:0025378D |                                     0.05%
.
.
Image PC Histogram          VAX SPM V3.0-01 Page 8
PROGRAM RTL
RTL              +-----+
0026A200:3FFFFFFF *-----* 77.11%
                  |
                  +-----+
Scaling: 2640.00 counts/asterisk

```

Figure 12: Image PC Sampling for BDM

Program Counter Sampling Data (790491 data points total)

Bucket Name	PROGRAM ADDRESS\	
BDMMAIN		0.0%
ALLOCATE		0.0%
ALSTRET		0.0%
BDMCLNUP		0.0%
BDMRQINDX		0.0%
BDMSETUP		0.0%
BLOSTRET		0.0%
CALCADR		0.0%
CHKDATA		0.0%
CHKIDEV		0.0%
CHKINDEX		0.0%
CHKREQST		0.0%
CHKSTORE		0.0%
CLOSEIMAG		0.0%
COPYKEY		0.0%
DASTRET		0.0%
DATAREQST		0.0%
DATASTORE		0.0%
DATFSP		0.0%
DEALDISC	*****	14.5%
DEALLOC		0.0%
DFKEYINS		0.0%
DFKEYSRCH		0.0%
DINDXINS		0.0%
DINDXSRCH		0.0%
EXITFILE		0.1%
EXITREQST		0.0%
EXITSTORE		0.0%
INDEXRTN		0.0%
INITDATA		0.0%
INITREQST		0.0%
INITSTORE		0.0%
KEYDELETE		0.0%
LINKREQ		0.0%
MAILMSG		0.0%
NMADRCHK		0.0%
OPENFILE		0.0%
OPSTRET		0.0%
PTRICE		0.0%
READATA	*****	18.7%
REPFIL		0.0%
RISTRET		0.0%
SISTRET		0.0%
SRCHSTORE		0.0%
STOREINDX		0.0%
UPDATINDX		0.0%
WRITDATA	*****	48.5%
CHRNCFY		0.0%
BFILL		0.0%
RCVMSGHDR	*****	18.0%
SNMSGHDR		0.1%

Figure 13: PCA Analysis of BDM

```

-----
VAX SPM V3.0-01
PC SAMPLER
Data Collection Start Time: 8-APR-1987 18:34:34.02
Data Collection Stop Time: 8-APR-1987 23:34:31.55
Total Elapsed Time: 04:59:57.53
Sampling Interval: 1
-----

```

Processes at Start Time:

Process Name	PID
NULL	00000080
SWAPPER	00000081
bdmain019	00000402
datmain019	00000503
ERRFMT	00000084
OPCOM	00000085
JOB_CONTROL	00000086
mudmain019	00000507
PIOTSK019	00000508
NETACP	00000089
EVL	0000008A
REMACP	0000008B
KBINIT019	0000050C
SYMBIONT_000	0000008D
OPI_019	0000050E
vct019	0000050F
alarmmain019	00000510
vgout019	00000511
VIDEO_4019	00000512
FEXEXEC019	00000513
FEXSCHED019	00000514
symmain019	00000515
OPIVD2019	00000516
splmain019	00000517
OPIVD3019	00000418
OPIVD4019	00000519
019VOS	0000049A
OPIVD5019	0000051B
OPILP_019	0000039C
BLKBLD019	0000051D
BB_CCTASK019	0000051E
SPM_TUNE	000003BE
TXA4:	0000049F
master1019	000004EA
master2019	000004EB
master3019	000004EC
timmain019	000003ED

Drivers at Start Time:

Driver Name	Start	End
SPMTIMER	8017FA50	80180380
DDDRIVER	80181FD0	801829E0
SCDRIVER	8017D3E0	8017E150
GPDRIVER	8017D210	8017D3E0
CXDRIVER	8017CF00	8017D210
PIPEDRIVER	8017C5A0	8017CF00
LTRDRIVER	80179320	8017C5A0
CTDRIVER	801770C0	80178B90
RTTDRIVER	80176510	80176F30
NDDRIVER	801723E0	80172D40
NETDRIVER	8016E7F0	801722F0
LCDRIVER	80200A00	80200F86
YCDRIVER	8016CEB0	8016D7C0
XGDRIVER	80169310	80169CCE0
DZDRIVER	801666F0	80167A10
XEDRIVER	80163160	80166A90
TSRDRIVER	80162240	80163000
DUDRIVER	801A7410	801AACB9
FUDRIVER	801A5C60	801A7405
TTDRIVER	801A0940	801A5C56
OPERATOR	800018CE	8000196E
NLDRIVER	80001895	80001ED2
MBDRIVER	8000185C	80001E5E
RMS CODE	8002E600	80048200
UBAO_ADAPTER	80155C00	80165E60

Processor Usage by Process

Process Name	Samples	%	Total Time System Time		System Time %	Process by Access Mode				IPL>0	IPL>2	
			(Seconds)	(Seconds)		KRNL	EXEC	SUPR	USER			CMPY
NULL	1048086	58%	10480.86	10480.86	100%	100%	0%	0%	0%	0%	0%	0%
SWAPPER	516	0%	5.16	5.08	98%	100%	0%	0%	0%	0%	99%	99%
bdmain019	287878	15%	2878.78	504.35	17%	13%	5%	0%	89%	0%	13%	6%
datmain019	5996	0%	59.96	25.31	12%	61%	0%	0%	38%	0%	21%	15%
ERRFMT	327	0%	3.27	1.74	53%	80%	18%	0%	0%	0%	43%	26%
OPCOM	36	0%	0.36	0.34	94%	88%	0%	0%	11%	0%	80%	80%
JOB_CONTROL	68	0%	0.68	0.47	69%	79%	5%	0%	14%	0%	51%	33%
mudmain019	17886	0%	178.86	93.19	52%	66%	0%	0%	33%	0%	31%	26%
PIOTSK019	1358	0%	13.58	4.49	33%	48%	0%	0%	51%	0%	19%	14%
NETACP	2438	0%	24.38	2.88	11%	100%	0%	0%	0%	0%	7%	6%
EVL	324	0%	3.24	3.01	92%	91%	1%	1%	5%	0%	85%	83%
REMACP	14	0%	0.14	0.13	92%	100%	0%	0%	0%	0%	92%	50%
KBINIT019	29198	1%	291.98	76.06	26%	58%	0%	0%	41%	0%	20%	11%
SYMBIONT_000	0	0%	0.00	0.00	0%	0%	0%	0%	0%	0%	0%	0%
OPI_019	0	0%	0.00	0.00	0%	0%	0%	0%	0%	0%	0%	0%
vct019	7006	0%	70.06	22.36	31%	56%	0%	0%	43%	0%	19%	15%
alarmmain019	0	0%	0.00	0.00	0%	0%	0%	0%	0%	0%	0%	0%
vgout019	18029	1%	180.29	75.67	41%	58%	0%	0%	41%	0%	28%	19%
VIDEO_4019	28977	1%	289.77	83.48	28%	49%	0%	0%	50%	0%	15%	10%
FEXEXEC019	0	0%	0.00	0.00	0%	0%	0%	0%	0%	0%	0%	0%
FEXSCHED019	9498	0%	94.98	40.65	42%	61%	0%	0%	38%	0%	23%	17%
symmain019	0	0%	0.00	0.00	0%	0%	0%	0%	0%	0%	0%	0%
OPIVD2019	0	0%	0.00	0.00	0%	0%	0%	0%	0%	0%	0%	0%
splmain019	4	0%	0.04	0.04	100%	100%	0%	0%	0%	0%	25%	25%
OPIVD3019	0	0%	0.00	0.00	0%	0%	0%	0%	0%	0%	0%	0%
OPIVD4019	0	0%	0.00	0.00	0%	0%	0%	0%	0%	0%	0%	0%
019VOS	4743	0%	47.43	35.53	74%	75%	14%	0%	10%	0%	48%	20%
OPIVD5019	0	0%	0.00	0.00	0%	0%	0%	0%	0%	0%	0%	0%
OPILP_019	7014	0%	70.14	23.35	33%	50%	0%	0%	49%	0%	17%	12%
BLKBLD019	163960	9%	1639.60	774.89	47%	59%	11%	0%	28%	0%	31%	20%
BB_CCTASK019	0	0%	0.00	0.00	0%	0%	0%	0%	0%	0%	0%	0%
SPM_TUNE	3302	0%	33.02	6.71	20%	73%	7%	0%	18%	0%	34%	30%
TXA4:	0	0%	0.00	0.00	0%	0%	0%	0%	0%	0%	0%	0%
master1019	0	0%	0.00	0.00	0%	0%	0%	0%	0%	0%	0%	0%
master2019	0	0%	0.00	0.00	0%	0%	0%	0%	0%	0%	0%	0%
master3019	0	0%	0.00	0.00	0%	0%	0%	0%	0%	0%	0%	0%
timmain019	66688	3%	666.88	228.14	34%	51%	0%	0%	48%	0%	26%	21%
PID = 0000051F	1713	0%	17.13	10.67	62%	53%	13%	29%	3%	0%	45%	28%
PID = 000004A0	1597	0%	15.97	9.49	59%	51%	14%	29%	4%	0%	42%	25%
PID = 00000521	82	0%	0.82	0.74	90%	86%	0%	0%	4%	0%	74%	35%
PID = 00000522	7420	0%	74.20	61.79	83%	47%	40%	7%	4%	0%	40%	22%
PID = 00000523	723	0%	7.23	5.64	78%	75%	13%	0%	11%	0%	70%	32%
PID = 00000524	613	0%	6.13	5.24	85%	80%	16%	0%	2%	0%	76%	34%
PID = 00000525	614	0%	6.14	5.30	86%	83%	14%	0%	2%	0%	78%	31%
PID = 00000526	2840	0%	28.40	21.20	74%	73%	5%	0%	20%	0%	69%	49%
PID = 000003A7	2446	0%	24.46	19.18	78%	76%	4%	0%	18%	0%	74%	58%
PID = 00000528	719	0%	7.19	7.12	85%	81%	13%	0%	5%	0%	77%	37%
INTERRUPT STACK	77640	4%	776.40	776.40	100%	100%	0%	0%	0%	0%	100%	100%

```

+-----+
| System Module Usage |
+-----+

```

```

1--- Filter: INTERRUPT STACK ---1

```

System Module	Module Samples	% Total Module Samples	Filter Samples	% of Module Samples	Filter Samples	% Total Filter Samples
IOCS\$PURGDATAP	220	0.02%	220	100.00%	0.28%	0.28%
EXES\$LOAD_NOP	3086	0.23%	0	0.00%	0.00%	0.00%
EXES\$LOAD_KCJF	1023	0.08%	0	0.00%	0.00%	0.00%
EXES\$LOAD_KRUF	520	0.04%	0	0.00%	0.00%	0.00%
EXES\$LOAD_KSPR1	645	0.05%	0	0.00%	0.00%	0.00%
EXES\$LOAD_KSPR2	1100	0.08%	0	0.00%	0.00%	0.00%
SCSS\$ALLOE_RSPID	216	0.02%	0	0.00%	0.00%	0.00%
SCSS\$DEALL_RSPID	137	0.01%	137	100.00%	0.18%	0.18%
SCSS\$LKP_RDTWAIT	1	0.00%	0	0.00%	0.00%	0.00%
PAT\$A_NONPGD_DAT	366	0.03%	366	100.00%	0.47%	0.47%
PMS\$END_IO	410	0.03%	410	100.00%	0.53%	0.53%
PMS\$END_RQ	387	0.03%	385	99.48%	0.50%	0.50%
PMS\$START_IO	366	0.03%	1	0.27%	0.00%	0.00%
PMS\$START_RQ	16	0.00%	0	0.00%	0.00%	0.00%
EXES\$SFAIL	1	0.00%	0	0.00%	0.00%	0.00%
IOCS\$IOPPOST	5697	0.42%	5697	100.00%	7.34%	7.34%
IOCS\$BUFFPOST	2277	0.17%	2262	99.34%	2.91%	2.91%
IOCS\$QNTSEG	2	0.00%	2	100.00%	0.00%	0.00%
IOCS\$QNTSEGL	30	0.00%	2	6.67%	0.00%	0.00%
IOCS\$WAKCP	3125	0.23%	0	0.00%	0.00%	0.00%
IOCS\$DIRPOSTL	1891	0.14%	0	0.00%	0.00%	0.00%
EXES\$POWERFAIL	46	0.00%	0	0.00%	0.00%	0.00%
MMGS\$PAGEFAULT	427	0.03%	0	0.00%	0.00%	0.00%
MMGS\$SVFCTX	7	0.00%	7	100.00%	0.01%	0.01%
MMGS\$PGLTWAIT	552	0.04%	0	0.00%	0.00%	0.00%
MMGS\$WLEPFN	65	0.00%	0	0.00%	0.00%	0.00%
MMGS\$FREWSLE	342	0.03%	0	0.00%	0.00%	0.00%
MMGS\$FREWSLX	61	0.00%	0	0.00%	0.00%	0.00%
MPH\$INVALIDHK	35	0.00%	0	0.00%	0.00%	0.00%
MMGS\$ERE_TRYSKIP	366	0.03%	0	0.00%	0.00%	0.00%
MMGS\$DELSLEX	62	0.00%	0	0.00%	0.00%	0.00%
MMGS\$DELSLEPPG	143	0.01%	0	0.00%	0.00%	0.00%
MMGS\$ININMPFN	149	0.01%	0	0.00%	0.00%	0.00%
MMGS\$MAKEWSLE	334	0.02%	0	0.00%	0.00%	0.00%
MMGS\$LOCKPGB	182	0.01%	0	0.00%	0.00%	0.00%
MMGS\$INCPTRF	1001	0.07%	0	0.00%	0.00%	0.00%
MMGS\$DECPTRF	1350	0.10%	1199	88.81%	1.54%	1.54%
MMGS\$DECPHDREF	1	0.00%	0	0.00%	0.00%	0.00%
MMGS\$DECPHDREF1	5	0.00%	1	20.00%	0.00%	0.00%
MMGS\$INIBLDPKT	23	0.00%	0	0.00%	0.00%	0.00%
MMGS\$ALLOCPFN	62	0.00%	0	0.00%	0.00%	0.00%
MMGS\$DELCOMPFN	314	0.02%	0	0.00%	0.00%	0.00%
MMGS\$REMPFNH	24	0.00%	0	0.00%	0.00%	0.00%
MMGS\$REMPFN	264	0.02%	0	0.00%	0.00%	0.00%
MMGS\$RELPFN	199	0.01%	0	0.00%	0.00%	0.00%
MMGS\$DALCRKASTORE	51	0.00%	0	0.00%	0.00%	0.00%
MMGS\$INSPFNH	19	0.00%	0	0.00%	0.00%	0.00%
MMGS\$INSPFNT	256	0.02%	0	0.00%	0.00%	0.00%
MMGS\$IOLCK	5531	0.41%	0	0.00%	0.00%	0.00%

Figure 14: SPM PC Sampling Report

```

$! INQUIREBKTDEF.COM Procedure to define image buckets for SPM
$Type sys$input
This procedure requires program map to be in this directory.
Procedure will search vos and vosuss maps for size data.

User is prompted for the location of vos and vosuss maps.
User will be prompted for information for the analysis.

Note: Procedure works only for the microvax VOS (not uvos).

$set noon
$prog='P1'
$if "'P1'.eqs." then -
$inquire prog -
"Please enter program name to examine (must match map name)"
$write sys$output ""
$write sys$output -
"Need location vos and vosuss maps (directory or dev:dir)."
$write sys$output -
"Logical name like mxv_share is ok. No answer is current directory."
$inquire loc "Vos, use_location"
$xxx:=f$environment("DEFAULT")
$if loc .eqs. "" then assign [] mxv_loc
$if loc .nes. "" then assign 'loc' mxv_loc
$write sys$output "Searching program map for address data ..."
$SEARCH 'PROG' MAP/WIN=(3,35)/OUT=SEA.TXT DEFAULT
$write sys$output -
"The following line from the map shows size of program."
$write sys$output -
"Add base of fixup vectors plus #pages times 200"
$write sys$output "(number just to left of base is # pages)."
```

```

$SEARCH SEA.TXT FIXUP
$delete sea.txt/0
$vosbase='P2'
$if "'P2'.eqs." then -
$inquire vosbase "How big is main program including data region?"
$! Here the dcl procedure constructs 'prog'.def
$defaddr := DEFINE ADDRESSES: MAP ""'prog'.map""
$
$open/write progdef 'prog'.def
$write progdef "DEFINE UNITS: PROGRAM, MODULE"
$write progdef " program 'prog'"
$write progdef defaddr
$write progdef "END"
$close progdef
$
$! Here the dcl procedure constructs vos.def, use.def, and rtl.def
$vosbaseN=ix/vosbase/
$SEARCH mxv LOC:mxv_VOS.MAP "VOSXFR "
$inquire vosxfrsize - "How big is vosxfr?"
$vosxfrsizeN=ix/vosxfrsize/
$!vosxfrsizeN=ix400
$endvosxfrN=vosbaseN+vosxfrsizeN-1
$endvosxfr=f$fac("IXL",endvosxfrN)
$SEARCH mxv LOC:mxv_VOS.MAP "$CODE "
$inquire voscodoff - "What is offset to vos $code?"
$voscodoffN=ix/voscodoff/
$!voscodoffN=ix00
$mxv_codeN=vosbaseN+voscodoffN
$mxv_code=f$fac("IXL",mxv_codeN)
$
$vosline1:= program vosxfr, 'vosbase'-endvosxfr/
$vosline2:= program vos, , , 'mxv_code'
$defaddrvosline:=DEFINE ADDRESSES: MAP ""'mxv_loc:mxv_VOS.MAP""
$
$open/write vosdef vos.def
$write vosdef "DEFINE UNITS: PROGRAM, MODULE"
$write vosdef vosline1
$write vosdef vosline2
$write vosdef defaddrvosline
$write vosdef "END"
$close vosdef
$
$SEARCH mxv LOC:mxv_VOS.MAP/WIN=(0,8) "VOSXFR "
$inquire vossize - "How big is vos?"
$vossizeN=ix/vossize/
$!vossizeN=ix9600
$ussbaseN=vosbaseN+vossizeN !end of vos = start of use
$ussbase=f$fac("IXL",ussbaseN)
$SEARCH mxv LOC:mxv_VOSUSS.MAP "VOSUSSXFR "
$inquire ussxfrsize - "How big is ussxfr?"
$ussxfrsizeN=ix/ussxfrsize/
$!ussxfrsizeN=ix200
$endussxfrN=ussbaseN+ussxfrsizeN-1
$endussxfr=f$fac("IXL",endussxfrN)
$SEARCH mxv LOC:mxv_VOSUSS.MAP "$CODE "
$inquire usscodoff - "What is offset to use $code?"
$usscodoffN=ix/usscodoff/
$!usscodoffN=ix00
$usscodeN=ussbaseN+usscodoffN
$usscode=f$fac("IXL",usscodeN)
$ussline1:= program ussxfr, 'ussbase'-endussxfr/
$ussline2:= program use, , , 'usscode'
$defaddrussline:=DEFINE ADDRESSES: MAP ""'mxv_loc:mxv_VOSUSS.MAP""
$
$open/write ussdef use.def
$write ussdef "DEFINE UNITS: PROGRAM, MODULE"
$write ussdef ussline1
$write ussdef ussline2
$write ussdef defaddrussline
$write ussdef "END"
$close ussdef
$
$SEARCH mxv LOC:mxv_VOSUSS.MAP/WIN=(0,9) "VOSUSSXFR "
$inquire ussize - "How big is use?"
$ussizeN=ix/ussize/
$!ussizeN=ix200
$rtlbaseN=ussbaseN+ussizeN !end of use = start of rtl
$rtlbase=f$fac("IXL",rtlbaseN)
$
$spm define=image 'prog'.def /address='prog'.adr /bucket=n1:
$spm define=image vos.def /address=vos.adr /bucket=n1:
$spm define=image use.def /address=use.adr /bucket=n1:
$
$create image.def
! image.def file to define buckets for prog analysis with spm
DEFINE OPTIONS: LIST, PRINT, ABSADDR
DEFINE UNITS: PROGRAM, PHASE, MODULE
$append 'prog'.adr image.def

```

```

$append vos.adr image.def
$append uss.adr image.def
$open/append imagedef image.def
$write imagedef "PROGRAM RTL,      ''rtlbase'-3fffffff"
$write imagedef "PROGRAM P1,      40000000-7fffffff"
$close imagedef
$append sys.def image.def
$      !like spmsystem.def, but added last address definition
$
$open/append imagedef image.def
$write imagedef "DEFINE SAMPLING"
$write imagedef "  PROGRAM ''prog' BY MODULE"
$write imagedef "  program vosxfi"
$write imagedef "  PROGRAM VOS BY MODULE"
$write imagedef "  program ussxfi"
$write imagedef "  PROGRAM USS BY MODULE"
$write imagedef "  PROGRAM RTL"
$write imagedef "  PROGRAM P1"
$write imagedef "  PROGRAM SYS BY MODULE"
$write imagedef "END"
$close imagedef
$
$delete 'prog'.adr;0,vos.adr;0,uss.adr;0
$delete 'prog'.def;0,vos.def;0,uss.def;0
$write sys$output "Now create buckets ..."
$
$I procedure to define image buckets from address data
$assign 'prog'/bkt.lis sys$error
$assign 'prog'/bkt.lis sys$output
$spm define=image image.def /bucket='prog'.bkt
$deassign sys$output
$deassign sys$error
$SEARCH 'prog'/BKT.LIS "*"          !VIEW ERROR MESSAGES
$write sys$output -
"Should have buckets defined and ready to use in ''prog'.BKT."
$write sys$output -
"Print ''prog'/BKT.LIS For a list of buckets and error messages."
$delete image.def;0
$exit

```

Figure 15: INQUIREBKTDEF.COM

Image PC Histogram VAX SPM V3.0-01 Page 1
 PROGRAM BLKBLD BY MODULE

BLOC		
00268400:0026844C		0.01%
.....		

Image PC Histogram VAX SPM V3.0-01 Page 13
 PROGRAM VOSXFR

VOSXFR		
0031DC00:0031DFFF	*	0.26%
.....		

Scaling: 645.70 counts/asterisk

Image PC Histogram VAX SPM V3.0-01 Page 14
 PROGRAM VOS BY MODULE

LKPCONE		
003200D5:003200F3		0.03%
LKPCONN		
003200F4:0032023A		0.19%
ALLOCBUFF		
003202E4:00320442		0.14%
BFILES		
003206C5:00321008	**	0.77%
MXCLOSE		
003213DF:00321429		0.00%
COPYBUF		
0032142A:003214FD		0.04%
CREPROC		
00321616:00321AE2		0.00%
MADELETE		
003226C3:0032273E		0.00%
GETBUFTR		
00322CCA:00322DAF		0.16%
GETBUFSIZ		
00322DB0:00322E49		0.06%
.....		

Scaling: 645.70 counts/asterisk

Image PC Histogram VAX SPM V3.0-01 Page 15
 PROGRAM USSXFR

USSXFR		
00328400:003285FF	*	0.37%
.....		

Scaling: 645.70 counts/asterisk

PROGRAM USS BY MODULE

SENDUTIL		
003291DE:003292BA		0.18%
ADDTOK		
00329364:00329537		0.20%
.....		
PROBEW		
00329F8F:00329FC0	*	0.28%
READTOK		
0032A232:0032A28F	*	0.50%
FREETOKR		
0032A290:0032A2AA		0.03%
REMQUE		
0032A2AB:0032A2BF		0.17%
SCHDACT		
0032A2C0:0032A307		0.03%
SETIPL		
0032A308:0032A30E		0.11%
VARALLOC		
0032A36C:0032A4E6	*	0.59%
VARFREE		
0032A4E7:0032A64A	*	0.30%
CHFNCPY		
0032A773:0032A782	*	0.36%
.....		

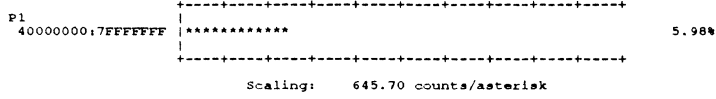
Scaling: 645.70 counts/asterisk

PROGRAM RTL

RTL		
0032AE00:3FFFFFFF	****	2.17%
.....		

Scaling: 645.70 counts/asterisk

PROGRAM P1



PROGRAM SYS BY MODULE

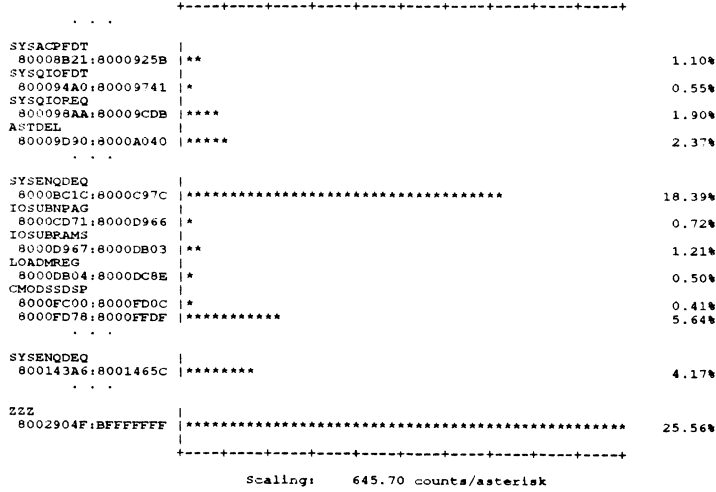


Figure 16: PC Histograms

PROGRAM MUDMAIN BY MODULE

```

MUDMAIN
00244400:00244730 |*****| 1.73%
RCVMSGHDR
00252E82:00252FA3 |*****| 3.32%
    
```

Scaling: 40.50 counts/asterisk

PROGRAM VOSXFR

```

VOSXFR
00253600:002539FF |****| 1.03%
    
```

Scaling: 40.50 counts/asterisk

PROGRAM VOS BY MODULE

```

ERRFNSVMS
00255082:00255518 |**| 0.43%
IPCDISC
0025593A:00255AD4 | | 0.01%
GETBUFTR
002586CA:002587AF | | 0.01%
GETTYPE
00258A9B:00258AD9 |****| 1.25%
RECEIVE
0025939C:0025956F |*****| 9.15%
VOS_TO_VMS_TIME
0025AF45:0025AF52 | | 0.04%
INSTALL
0025B81E:0025B82B |*****| 2.34%
CHRNCPY
0025B864:0025B873 |*****| 1.92%
BFILL
0025B874:0025B886 | | 0.01%
    
```

Scaling: 40.50 counts/asterisk

Image PC Histogram VAX SPM V3.0-01 Page 4

PROGRAM USSXFR

```

USSXFR
0025DE00:0025DFFF |*****| 2.23%
    
```

Scaling: 40.50 counts/asterisk

Image PC Histogram VAX SPM V3.0-01 Page 5

PROGRAM USS BY MODULE

```

RECUTIL
0025F2D0:0025F3A1 |*****| 5.95%
INSQUE
0025F866:0025F86F | | 0.01%
MEM_LOCK
0025F905:0025F935 | | 0.01%
PROBEW
0025F98F:0025F9C0 |***| 0.97%
READTOK
0025FC32:0025FC8F |*****| 2.17%
REMQUE
0025FCAB:0025FCBF |*****| 4.79%
SETIPL
0025FD08:0025FD0E |*****| 1.94%
VARALLOC
0025FD6C:0025FEE6 | | 0.01%
CHRNCPY
00260173:00260182 |***| 0.82%
    
```

Scaling: 40.50 counts/asterisk

Image PC Histogram VAX SPM V3.0-01 Page 6

PROGRAM RTL

```

RTL
00260400:3FFFFFFF |*****| 2.47%
    
```

Scaling: 40.50 counts/asterisk

PROGRAM P1

P1	40000000:7FFFFFFF	*****	4.19%
----	-------------------	-------	-------

Scaling: 40.50 counts/asterisk

PROGRAM SYS BY MODULE

SYSLOAVEC	800034D0:800036AF	*****	1.92%
SYSLKWSET	80007E3C:80008125	****	1.13%
EXSUBROUT	80008A78:80008B10	*****	14.12%
SYSCANEVT	800093B6:800093EE	*****	1.59%
SYSSCHEVT	80009742:800098A9	*	0.10%
ASTDEL	80009D90:8000A040	*****	7.44%
RSE	8000A37D:8000A675	*****	1.69%
SCHEM	8000A6A4:8000A72C	****	1.05%
MEMORIALC	8000A814:8000ADD4	*****	2.24%
POSTEF	8000AEFF:8000B06B	*****	3.07%
SYSEVTSRV	8000B205:8000B2A3	**	0.60%
SYSWAIT	8000B60A:8000B6E5	*****	1.85%
CMODSSDSP	8000FD78:8000FFDF	*****	12.80%
EXSUBROUT	80010353:80010577	****	0.88%
ZZZ	8002904F:8FFFFFFF	*	0.28%

Scaling: 40.50 counts/asterisk

Figure 17: Image PC Histogram for MUDM

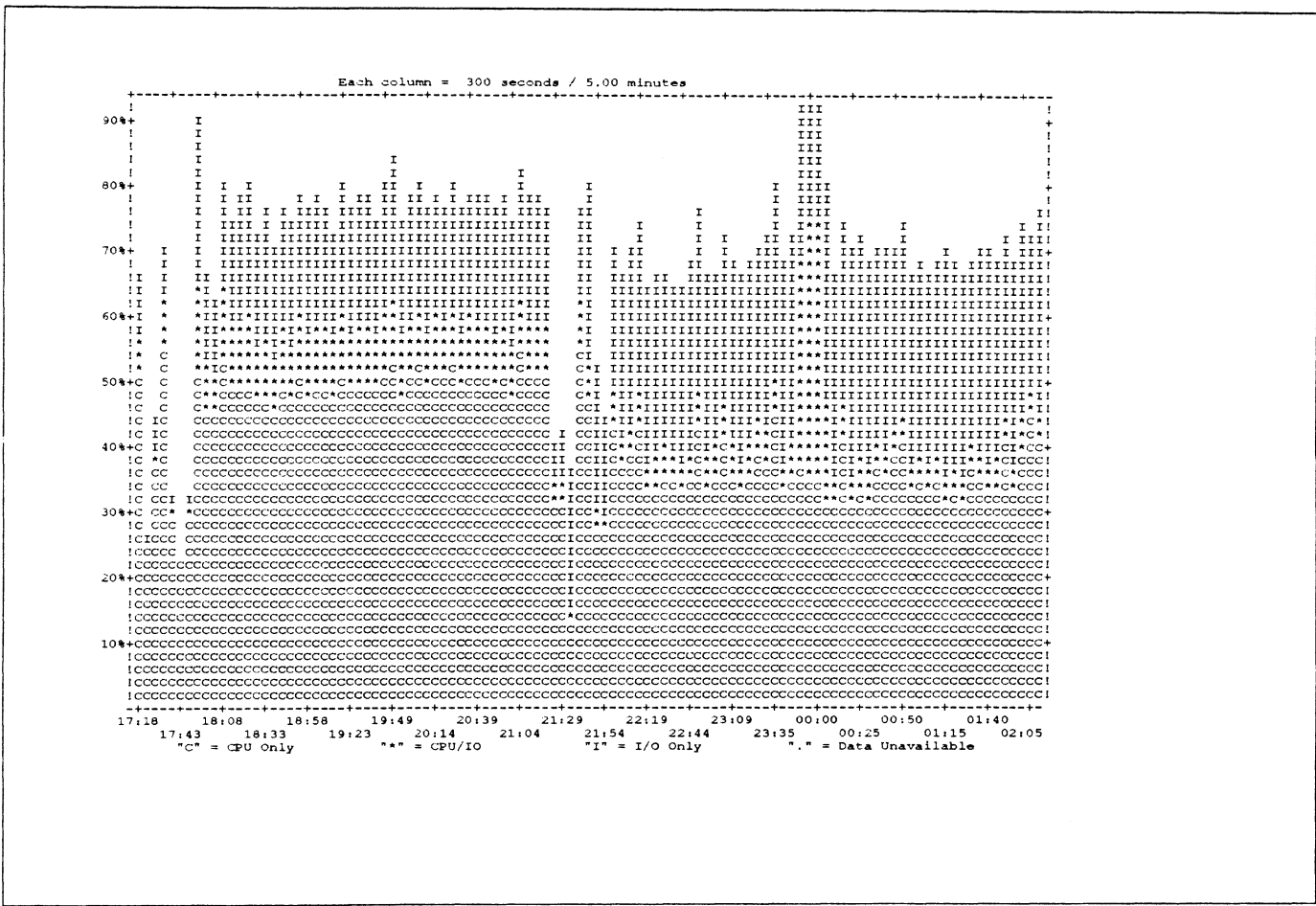


Figure 18: System Summary (Percent) vs. Time of Day

```

*****
*****              FINAL Statistics              *****
*****              Data Analyzed: from 16-MAR-1987 18:02:30.16 to 16-MAR-1987 20:58:31.65 *****
*****
----- AVE Process-Memory Counts ----- Memory Utilization ----- AVE Mem/CPU ----- Swapper Counts -----
| Proc  Balset  Free  Modify| Total  Paged  User  Modify| Queues  | Header  Header  Swaper  |
| Count Count  Pages  Pages|MEMutl  MEMutl  MEMutl  MEMutl| Mem    CPU  |InSWP  OutSWP  InSWP  OutSWP  CPU %  | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 39    34     724   269| 94.1%  92.4%  91.8%  3.1%| 0      11   | 0      1      0      0      0.0%  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
----- CPU Idle ----- CPU Busy ----- CPU and I/O Overlap -----
| Total Page  Swap  Pg+Swp| Inter  |
| Idle  Wait  Wait  Wait  | Stack  Kernel  Exec  Super  User  Compat| System  Task  | CPU+IO  CPU  I/O  Multi  CPU+IO| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 52.0% 0.2% 0.0% 0.2%| 5.1%  18.4%  4.4%  0.0% 20.1%  0.0%| 27.9% 20.1%| 23.6% 40.6% 24.5% 33.9% 11.3%|
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
----- Paging Rates (per second) -----
| Page System  Pages  Read  Pages  Write  Free  Modify  Bad  Dzero  Gvalid  Trans  WritIn| Hard  Soft  |
| Faults Faults  Read  I/Os  Written I/Os  List  List  List  Faults Faults Faults  Prog| Faults Faults| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2.7    0.0    0.4    0.1    0.5    0.0    0.2    0.3    0.0    0.8    1.2    0.0| 0.0| 2.7%  97.3%|
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
----- I/O Rates (per second) ----- File I/O Rates (per second) -----
| Direct  Buffrd  Lognam  Mailbx  Mailbx  | Window  Window  Split  Erase  File  | AVE  |
| I/Os    I/Os    Trans  Reads  Writes  | Hits    Turns  I/Os  I/Os  Opens  | Open  | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 19.7    1.6    0.3    0.1    0.1    | 14.9    0.3    0.1    0.0    0.0  | 171  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
----- File Cache Attempt Rate (per second) ----- File cache Effectivness -----
| Dir  Dir  Quota  File  File  Extent  Bit  | Dir  Dir  Quota  File  File  Extent  Bit  |
| FCB  Data  Data  Id    Hdr    | Map  | FCB  Data  Data  Id    Hdr    | Extent  Bit  | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.0   0.1   0.0   0.0   0.5   0.1   1.6  | 99.2% 99.2% 0.0% 100.0% 97.4% 87.5% 0.9%  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
----- Disk Statistics -----
| Work  |
| Avail  Paging  Swping  Contlr  Rate  Serv  Resp  |
| %      %      %      %      (/s)  Time  Time  |
|-----|-----|-----|-----|-----|-----|-----|-----|
| DUAO  | 35.8  0.9  0.3  100.0  16.6  22   24   | 0.4  98.6  |
|-----|-----|-----|-----|-----|-----|-----|-----|
----- Server Statistics -----
| Work  |
| Avail  Paging  Swaping  Queue  |
| %      %      %      Length  |
|-----|-----|-----|-----|
| PUA   | 33.9  0.9  0.3  0.4  |
|-----|-----|-----|-----|

```

Figure 19: SPM Tabular Report

TRANSFERRING DATA BETWEEN HETEROGENEOUS COMPUTERS: A TOOL TO MAINTAIN THE INTEGRITY OF FOREIGN DATA

Steven J. Kempler
NASA/Goddard Space Flight Center
Laboratory of Extraterrestrial Physics
Greenbelt, Maryland

ABSTRACT

The Interactive Format Conversion System (IFCS) is a package designed to facilitate the transfer of data between heterogeneous computers. IFCS has expanded to include the implementation of data conversion between four unrelated computers, in one system. The system has the generalized capability of: 1) accepting input data from a number of devices (disk, tape, data line); 2) performing useful data conversions, and; 3) producing output on a variety of devices. The structure of the data conversion subsystem simulates a subset of the presentation layer in a network communications link by converting input data into an internal machine independent format and then converting the internal data to the output format. This conversion subsystem is derived by the inputs that specify the requirements of the particular application. The Transportable Applications Executive (TAE) is used to provide a consistent user interface and tie the various subsystems together. This talk describes the capabilities, operating procedures, benefits and future considerations of IFCS.

1.0 INTRODUCTION

1.1 Background

The transfer of space-derived data between computers, both heterogeneous and homogeneous, have become more common and increasingly desirable. The International Organization of Standardization (ISO) has established a seven layer model for networking. (Tanenbaum describes this in detail in his book Computer Networks.) See Figure 1. The five lower layers of the networking model are being addressed by various organizations, including the National Bureau of Standards (NBS), International Standards Organization (ISO), and the IEEE. Currently, the sixth layer, the Presentation layer, is being developed on a case-by-case basis many times over for a variety of space-related data. This layer specifically performs transformations on data, such as text compression, format conversions, encryption, etc.

1.2 Current Effort

Problem: Currently, Presentation layer, specifically format conversion software, is being developed on a case-by-case basis many times over for a variety of space-related science data, leading to much duplication of effort and code.

Solution: The Interactive Format Conversion System (IFCS) is a subset of Presentation Layer Software. It generalizes format converting by interactively generating software that transforms data from and to the desired machine formats. The generated code is transportable and can be generated for a particular application and used on a number of different computers. Such a system solves the problem for the users of space-derived data that has not been attacked in any general sense up to now.

IFCS presently resides on the Laboratory of Extraterrestrial Physics (LEP) (Code 690) VAX 11/785, in Building 2 at Goddard Space Flight Center. This computer supports a wide range of scientific space-related data and the analysis of this data. Included are data from Mariner, Voyager, ISEE and IMP satellites. In addition, the LEP VAX supports many data analysis packages and numerical libraries. Therefore, the need has grown to transfer data on the LEP VAX to other computers as well as visa versa, to support the needs of scientific data analysis in familiar environments. This need is not limited by any means.

2.0 CAPABILITIES

2.1 General Capabilities

Generally, IFCS is capable of creating a transportable data conversion routine to exact user specification, compiling and linking the routine to a user developed program or the General Format Conversion Utility (GFCU, described below) and executing the GFCU to convert data.

2.2 Functional Capabilities

The first step of IFCS is the conversion generator program. Required inputs include input and output record definition and optional namelist file names, machine format associated with the data and the name of the output conversion routine. The conversion generator analyzes the input and output record definitions and builds a file (a subroutine) that contains all the actual field conversion routines in exact order as defined by the record definition. The library containing these lower level conversion routines is the heart of IFCS. Each routine performs a different function. (i.e. convert DEC R*4, convert to IBM R*8, etc. See PERFORMANCE CAPABILITIES for a further discussion on converting.) The conversion generator creates the routine which in turn accesses these pre-existing routines when run. The second step in IFCS is to compile and link the conversion routine. It may be linked to a user developed program or it may utilize the General Format Conversion Utility (GFCU). This utility will perform all general input of data, convert the data using the conversion routine and output the results. Finally, GFCU or the user application program is executed to perform the format conversion.

In addition, IFCS includes a Machine Definition program for when it becomes desirable to add new machines to the system (IFCS presently supports DEC VAX, IBM and SIGMA 9 computers). New conversion library routines will also need to be implemented.

IFCS utilizes Transportable Applications Executive (TAE) to enhance its functional capabilities as well as fulfill design

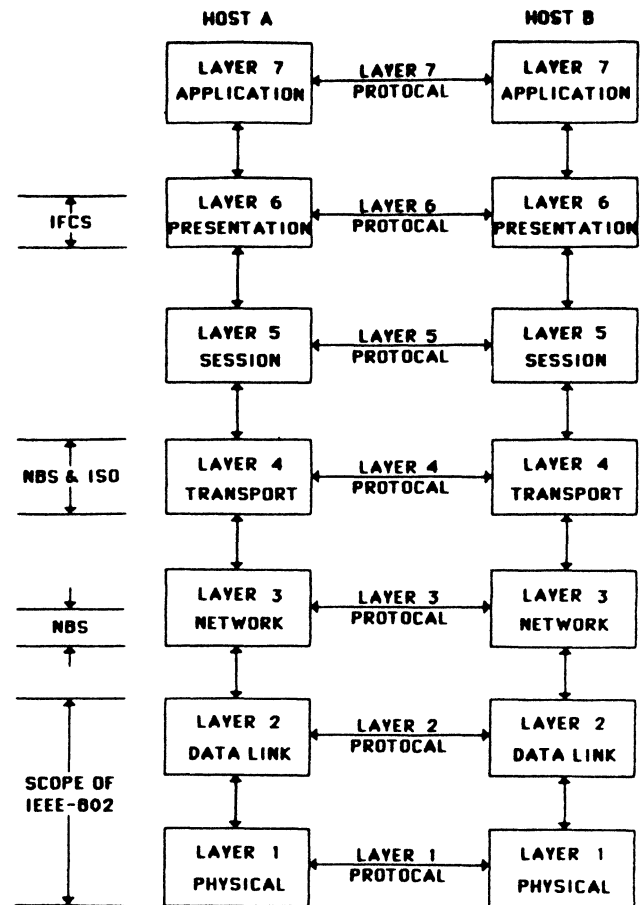


FIGURE 1 - SEVEN LAYER NETWORKING MODEL

objectives. Inputs are entered using TAE standards. This provides ease for the experienced user and support for the less experienced user. Menus and help files provide information for the first time user. In addition, IFCS can be easily transported to any installation that maintains TAE.

2.3 Performance Capabilities

IFCS has several important performance capabilities. A primary capability is its use of namelists. This provides the user with the ability to convert only certain fields of data from the input record. GFCU is capable of inputting data from up to three input sources, converting the data and outputting to a single sink. Also, GFCU can input and output to tape or disk. Most important is that IFCS uses an intermediate data format when converting data. That is, every field transformed is actually converted twice (i.e. IBM -> intermediate form -> VAX). This design was implemented so that when additional machines are added, source code will increase at a much smaller rate (Figure 2). (All new routines will convert to or from intermediate format.) Finally, IFCS is capable of checking for precision loss and overflow when it is desirable to convert values to utilize less space (i.e. R*4 to

NUMBER OF CONVERSION ROUTINES REQUIRED AS A FUNCTION OF THE NUMBER OF MACHINES IN /FCS

CONVERTING ONE DATA TYPE TO THE SAME DATA TYPE:

NUMBER OF MACHINES (M)	ONE ROUTINE DOES CONVERSION (M*(M-1))	WITH /FCS INTERMEDIATE FORMAT (M*2)
2	2 (1 IN EACH DIRECTION)	4 (2 IN EACH DIRECTION)
3	6	6
4	12	8
5	20	10

CONVERTING ONE DATA TYPE TO ANY OF 4 DATA TYPES:

M	16 * M * (M-1)	4 * M * 2
2	32 (16 IN EACH DIRECTION)	16 (8 IN EACH DIRECTION)
3	96	24
4	192	32
5	320	40

ORDINARILLY, IT TAKES 32 ROUTINES TO BE ABLE TO CONVERT ANY 6 DATA TYPES (R*4, R*8, R*16, I*2, I*4, C*8) FROM ONE MACHINE TO ANY OF THOSE DATA TYPES ON ONE OTHER MACHINE.

USING /FCS INTERMEDIATE FORMAT IT TAKES ONLY 16.

FIGURE 2

I*2). A value, of the users choice, representing BAD data is inserted. Also, the user may choose a tolerance level in which /FCS will stop if BAD must be inserted too many times.

Figure 3 lists the binary data types that /FCS is presently capable of converting from and to.

3.0 OPERATING PROCEDURES

User inputs to the system include global variables which are defined and used throughout the /FCS session. After /FCS is launched (Figure 4), the CVTGEN executes by receiving communications (inputs) from the user: data definition files, file and machine names. CVTGEN generates a stand alone conversion routine, that may be linked to a user supplied program or link to the General Format Conversion Utility (CVTLINK). To execute the General Format Conversion Utility (GFCUTIL), the data input devices (containing the data to be converted) and data output device, and device attributes must be communicated (input) to the utility, as well as the number of records to convert. The result is converted data.

/FCS is able to perform six basic operations through the use of the TAE menus (Figure 5). /FCSGBL is a global procedure

BINARY DATA TYPES PRESENTLY SUPPORTED

FOR DEC VAX	FOR IBM 3081/360
REAL*4 - R4	REAL*4 - R4
REAL*8 - R8	REAL*8 - R8
REAL*16 - R1	REAL*16 - R1
INTEGER*2 - I2	INTEGER*2 - I2
INTEGER*4 - I4	INTEGER*4 - I4
COMPLEX*8 - C8	COMPLEX*8 - C8

FOR XEROX SIGMA 9	FOR CDC CYBER 750
REAL*4 - R4	REAL*10 - R0
REAL*8 - R8	
INTEGER*4 - I4	
COMPLEX*8 - C8	

FIGURE 3

that allows the user to define certain variables. CVTGEN, CVTLINK and GFCUTIL are the three steps for developing and executing a data conversion program (Figure 6). The procedure, /FCS, combines the previous three PDF's in one procedure. Finally, MACHDEF allows the programmer to implement additional machines.

USER INPUTS FOR /FCS

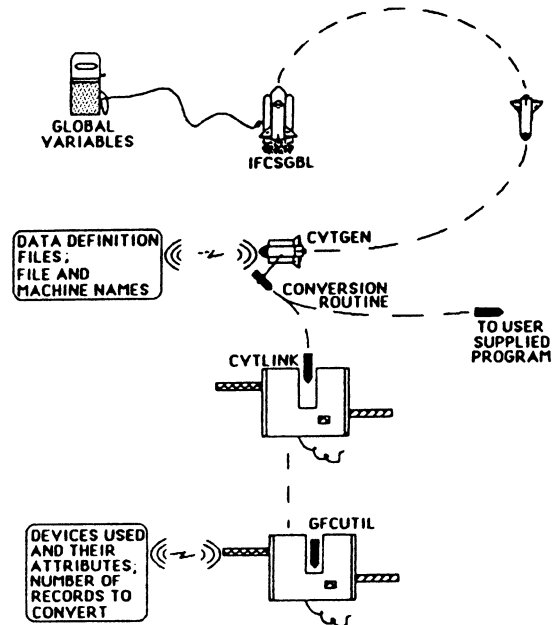


FIGURE 4

■ "ROOT", library "DISKUSERS3:VLSJFY.PLS.DEMO")

INTERACTIVE FORMAT CONVERSION SYSTEM

- 1) TO ALTER IFCS GLOBAL VARIABLES (IFCSGBL)
- 2) TO GENERATE AN IFCS CONVERSION ROUTINE (CVTGEN)
- 3) TO LINK THE CONVERSION ROUTINE TO THE GENERAL FORMAT CONVERSION UTILITY (CVTLINK)
- 4) TO EXECUTE THE GENERAL FORMAT CONVERSION UTILITY (GFCUTIL)
- 5) TO EXECUTE PROCs CVTGEN, CVTLINK AND GFCUTIL (IFCS)
- 6) TO IMPLEMENT A NEW MACHINE'S ATTRIBUTES (MACHDEF)

Enter: selection number, HELP, BACK, TOP, MENU, COMMAND, or LOGOFF.
?

FIGURE 5

3.1 IFCSGBL

Two variables set in this global are used throughout IFCS.

- the name of the IFCS generated conversion routine to be linked and executed.
- the number of input sources (up to three).

3.2 CVTGEN

CVTGEN represents the first step if IFCS (Figure 7). Using information received from user created internal files and user input, CVTGEN creates a FORTRAN routine that, when executed, will receive data according to the specified format, convert the data to the desired machine and output only the data fields within the record that are of interest. The conversion routine is made up of a series of calls to

INTERACTIVE FORMAT CONVERSION SYSTEM (IFCS)

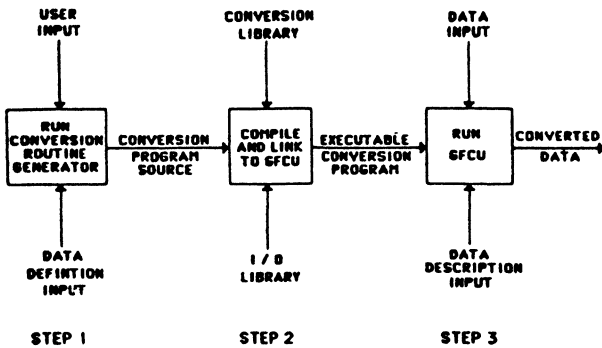


FIGURE 6

CONVERSION ROUTINE GENERATOR

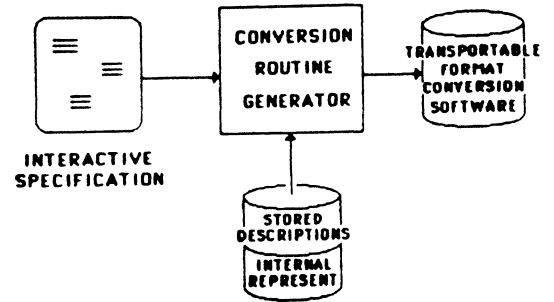


FIGURE 7

pre-existing lower level routines. For each data field, a lower level routine is accessed to perform the correct bit manipulations to move that field into and out of the intermediate format (as described earlier).

Outputs

The CVTGEN output is the reusable conversion routine created to user specification.

3.3 CVTLINK

At this point the conversion routine may be utilized with a user application or it may be linked to the General Format Conversion Utility (GFCU). This procedure compiles and links a conversion routine to the GFCU. The name of the object and load modules will be the same as the source file, which are defined in IFCSGBL. No interactive inputs are required for this step.

3.4 GFCUTIL

The third step, GFCUTIL, actually converts the specified data (Figure 8). This procedure, using the tape I/O library, provides a means for reading tapes created on and writing tapes for other machines in any format, as well as reading and writing to disk. Generally, GFCU acquires the input data, performs the specified conversions, and outputs the results.

Outputs

The output generated is a file containing the desired converted data.

GENERAL FORMAT CONVERSION UTILITY (GFCU)

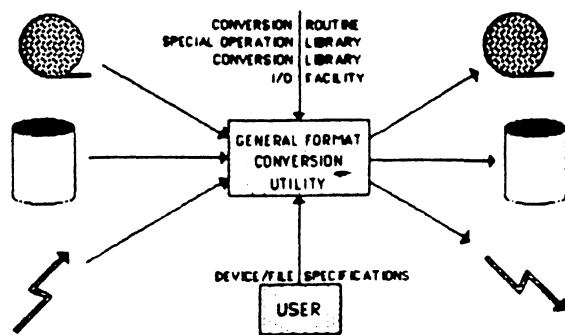


FIGURE 8

3.5 IFCS

The purpose of this operation is to combine the three steps of IFCS into one procedure. This provides much convenience when it is desired to create, link and execute IFCS all at once.

3.6 MACHDEF

This operation is primarily used by the IFCS manager. When a new machine is implemented into IFCS, the manager must: create lower level conversion algorithms that convert data to and from the intermediate format; provide for any tape formatting dissimilarities that the new machine has to the existing machines (in the tape I/O library) and; execute MACHDEF. MACHDEF is a software maintenance program that implements the characteristics of any newly added machine to IFCS. As mentioned, only the IBM, VAX and SIGMA 9 are presently supported. This software receives the machine characteristics and places them in a machine definition file.

Outputs

The output of this process is the addition of the new machine specifications in the machine definition table.

4.0 DESIGN CONSIDERATIONS

The primary objective in designing and implementing IFCS was to develop a system that allows characteristics of the source and target data streams, along with identifier information, to be easily specified interactively. IFCS utilizes this input to produce transportable computer code that maintains the semantics of the data as they are transformed from one computer to another. In addition, IFCS was developed to be friendly and flexible. The user need not supply more

information than is absolutely necessary for the function to be performed. Also, interactive input requirements must be unambiguous and check for invalid inputs. IFCS is made flexible enough to handle a wide spectrum of possible data inputs. Finally, the system was designed to isolate host specific code so that it may be transported with minimal change.

5.0 DESIGN DEPENDENT ENHANCEMENTS

Several enhancements were implemented easily into IFCS due to careful planning in the design phase of IFCS development. An important one involved adding the capability to convert data to and from SIGMA 9 format and CDC format. Because of the use of an intermediate format, the number of routines required for this implementation was minimized. That is, creating one routine that converts a particular data type to the intermediate format, opens the door for that data to be reconverted to any data type in any of the available machine formats. Converting CDC data is particularly interesting because of its 60 bit words. IFCS allows you to convert these words to R*4 or R*8 variables on the desired output machine, according to the precision requirements of the application.

Another enhancement provided the capability to check for overflow conditions after conversion was performed. Before executing GFCUTIL to commence data conversion, the user may alter the default values to be inserted for REAL or INTEGER values that overflow when converted. Also the user may enter a tolerance number. This is the number of overflows that will be tolerated before terminating the program.

The third important enhancement is the implementation of code to convert VAX and IBM REAL*16 and COMPLEX*8. If you need it, it's there! Again, by adding a 'convert from' and a 'convert to' routine for each data type on each machine, allows conversion to any available data type.

6.0 ADVANTAGES OF USING IFCS

Several advantages are realized by IFCS users. First off, it is an easy to use FORTRAN program. There are no hardware to hookup or software protocols to learn. IFCS has an easy to use front-end program interface, that provides on-line help, prompts, default input values, input value checks. IFCS also has the capability to convert any type of data between any of four heterogeneous computers in one system. This is a very unique feature. Furthermore, IFCS is not specific to any particular data record definition. In fact, the user defines the record definition by simple IFCS rules. This is to say, IFCS capabilities are not limited by record definition. The use of namelists so that only part of an input record need be

converted has proven advantageous. Other advantages are that the GFCUTIL can read and write a variety of VAX tape formats and read and write standard labeled tapes for VAX, IBM and SIGMA 9. Finally, once the conversion routine is created, it may be implemented into a user supplied application or I/O program. In other words, IFCS need not be used as an entire unit, but its individual procedures may be utilized as required.

7.0 FUTURE CONSIDERATIONS

Plans exist for IFCS on all fronts. Enhancements to the system include adding a provision for special data types (for example, spacecraft telemetry). Enhancements for I/O include implementing electronic communication into GFCU. Adding other machines to the system, such as UNIVAC, is another immediate consideration, as well as implementing IFCS on other machines. The latter would require that: 1.) TAE be present; 2.) IFCS code be transferred to the new installation, and; 3.) The lowest level conversion routines be altered for the new machine. (As mentioned, the generated conversion routine stands alone and can be transferred to another machine without any adjustments.) From an operational point of view, optimizing the speed of IFCS is being addressed.

ACKNOWLEDGEMENTS

I wish to acknowledge William Mish, Thurston Carleton, Len Moriarty, Frank Ottens and Allan Silver for their recommendations concerning IFCS.

REFERENCES

- Carlson, Patricia A., et al., Primer for the Transportable Applications Executive, NASA/GSFC, January, 1984.
- Century Computing, Inc., Application Programmer's Reference Manual for the Transportable Applications Executive, March, 1984.
- Century Computing, Inc., User's Reference Manual for the Transportable Applications Executive, March, 1984.
- Computing Surveys, Vol. 13, No. 4, December, 1981.
- Tanenbaum, Andrew S., Computer Networks, Prentice-Hall, 1981.

Tom Cheatham

Linkware Corporation
128 Technology Drive
Waltham, MA 02154

ABSTRACT

OSI is a revolutionary information distribution standard. However, since most currently installed network hardware (terminal emulation boards for PCs, for example) does not support this standard, preparing your network for OSI can be very costly. A new information distribution technology is described that provides the benefits of OSI on your current network hardware.

PROBLEM

As many of you are aware, the International Standards Organization has developed a set of communications standards known as the Open Systems Interconnect model, or OSI. The primary purpose of this OSI model is to standardize the semantics and formats of protocol data units so that information may be effectively transmitted between unlike computers.

It is not enough, however, to get bits safely and accurately between machines. Application connectivity must exist to put information to good use. The OSI standard addresses this problem by providing standardized "all purpose" application specifications so that vendors can develop useful applications.

Because of these information transmission and application standards, many data communications experts consider OSI to be a revolutionary information distribution standard. But before two computers can talk using OSI, all connecting network equipment must conform to OSI. Since most currently installed network hardware (such as terminal emulation boards for PCs) does not support this standard, preparing your network for OSI can be very costly.

An obvious question to any software vendor addressing the mixed vendor market is how to take advantage of the semantic communications so elegantly specified in the OSI standards, yet do so over the user's existing hardware.

SOLUTION

Many existing commercial networks support full-duplex virtual circuit capabilities. These offerings include DECnet, SNA, TCP/IP, MSNET, NFS, HyperChannel, FastPath, X.25, etc... Existing user investment in these networking solutions is tremendous. An OSI-based application design that views each of these tested technologies as subnet options would enable cross-vendor communications over any of them.

This paper presents an application design that demonstrates how the OSI "all-purpose application" protocol known as FTAM (File Transfer and Access Management) can be made to work effectively over any installed subnet. The key to this concept is the subnet support layer, whose function it is to select the technology supporting the requested target machine (or node), make the necessary virtual circuit connection to the node, and then process the sending and receiving of data over that virtual circuit.

The FTAM protocol, as specified in the OSI standard, provides for the semantic transfer of data, and its structure, between unlike systems. This makes it possible, for example, for 60 bit machines to communicate with 16 bit machines (transmitting a 16 bit PC floating point number to a 60 bit CDC machine); something that heretofore was not standardized and was very difficult to implement. The OSI reference model (including FTAM) does not, however, support existing proprietary network technologies.

The three primary components of the proposed design, which work together to support existing networks, are the virtual filestore, the subnet switching mechanism, and process responders. Before describing these components, it is important to define two terms. These are "initiator FTAM" and "responder FTAM". Initiator FTAM is the FTAM application in the computer that initiates a request for information; responder FTAM is in the computer that receives the request for information.

The Virtual Filestore

A key concept in the FTAM model is the virtual filestore which is an abstract collection of files and manipulation machine which accepts primitives from the FTAM responder service interface. The virtual filestore in turn front-ends a mapping scheme into the operating system's native file services. The service interface provided by the initiator FTAM service layer and the interface into the virtual filestore are identical. Hence it makes perfect sense to allow the user interface region to alternatively access the virtual filestore directly. This allows the user interface region the option of manipulating the local filestore as well as remote filestores.

The Subnet Switching Mechanism

The FTAM protocol machine interfaces with the presentation layer which in turn interfaces with the OSI Session Services Interface (SSI). The OSI Presentation Layer Passes a primitive, and its accompanying parameters, to the OSI SSI. In a true OSI network, that primitive would be used by the Session Layer to perform the requested service. The session primitive and its accompanying parameters can instead be bundled into a non-OSI protocol data unit and transmitted across ANY available network to a remote machine. There the primitive and Parameters can be unbundled and presented, in OSI Session interface format, to the Presentation Layer on the remote site. In this way, the proposed design application "fools" the OSI Application and Presentation Layers into thinking they are operating in a pure OSI environment. This enables the use of existing network capabilities while reaping the benefits of the vendor independence inherent in the OSI protocols.

The only real requirement of the existing network technology is that it be possible to create full-duplex virtual circuits between processes. The bundled OSI Session Services primitives are the "new" protocol exchanged over that connection. The subnet switching layer uses configuration tables to determine which network technology to use to make the initial connection and which parameters should be applied.

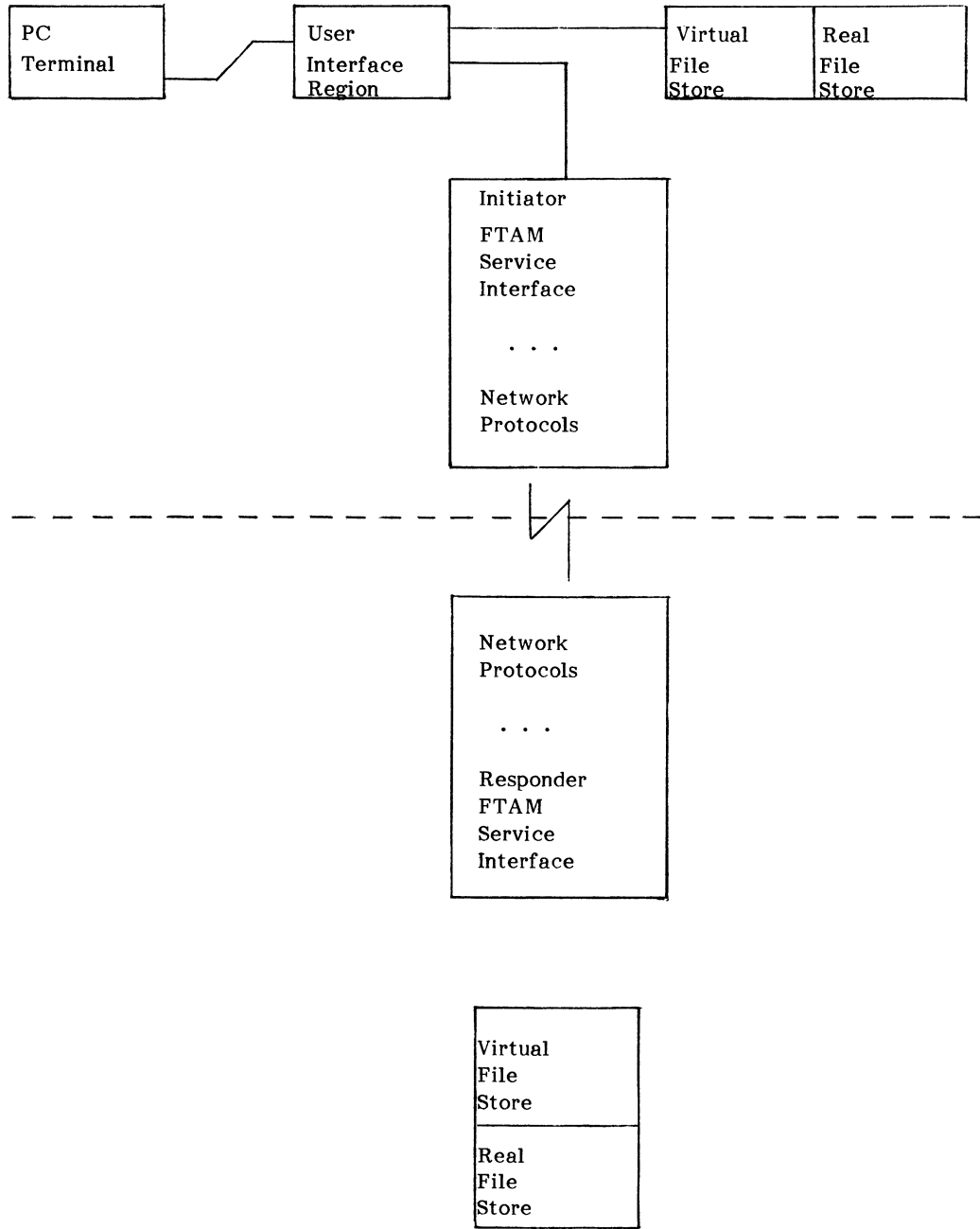
Process Responders

Process responders are required in this model to accept the subnet connection and receive the bundled OSI Session Services primitives.

There are two common methods for establishing communications with a remote process. The first and often the simplest model is the spawn option. Spawning process-responders in response to remote requests is often a service provided by the existing network capability (DECNET, TCP/IP, etc...). If the network does not supply spawning services they can often be added as part of the software package which must then support some protocol elements to effect that service.

The second model is to create permanently resident multi-user responders which multiplex connections at the application layer. This method is supported by such network technologies as SNA, Network Systems HyperChannel, and Ungermann-Bass Net/One.

Both models can be supported by the FTAM application model. In either case, once the responding process has accepted the connection, the "bundled" OSI session primitive protocol completes the picture.

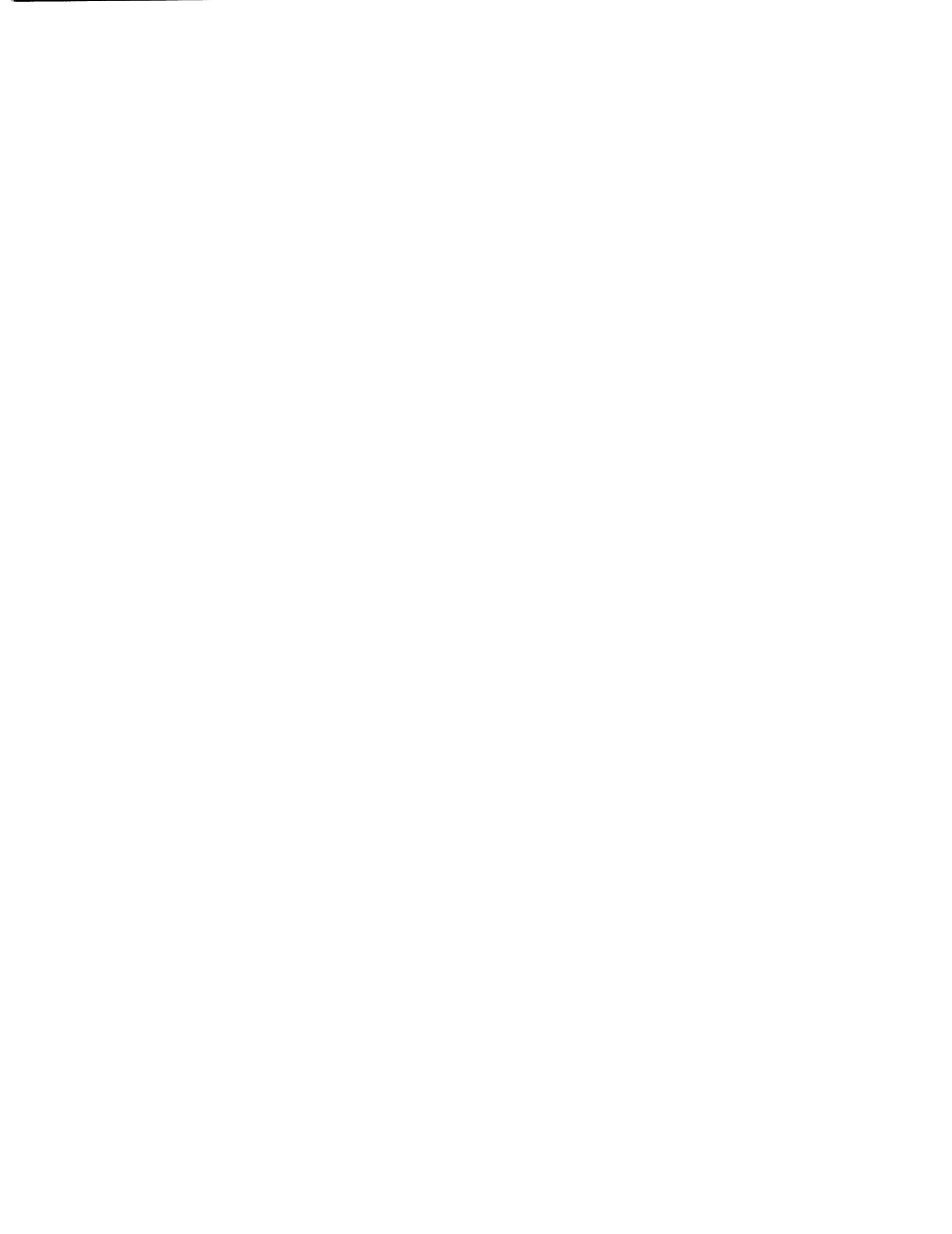


CONCLUSION

The FTAM protocol defined in the OSI standards provides for the semantic transfer of data, and its structure, between unlike systems. Although this protocol does not support existing network technologies, it is possible to build an application based on the FTAM protocol which enables cross-vender communications over existing commercial networks.

The keys to the success of such an application are: 1) a virtual filestore on each node that is accessible to local and remote users, 2) a subnet switching mechanism that determines which technology to use to connect nodes, and that fools the OSI Application and Presentation Layers into thinking they are operating in a pure OSI environment, and 3) process responders to complete the connect request from the initiating process.

This application model can be extended to other OSI "all-purpose" application protocols (CCITT X400 MHS, VTP, TOPP, MMFS, Network Directory Services) in the same manner with the same basic result- OSI functionality over your current network investment.



VMS, Xenix, Unix, and MS-DOS Transparent Resource Sharing

E. Berelian, L. Farmer, H. Kilman
F. Schoen, P. Wang, J. Vij

ITT ADVANCED TECHNOLOGY CENTER
SHELTON, CONNECTICUT

ABSTRACT

This paper documents the results of an R&D case at the ITT Advanced Technology Center on Distributed Business Communications. The objective was to define and prototype a substrate that would add a layer of distribution and resource transparency at the user level over a network of heterogeneous machine environments. That is, it presents a *Single System Image* to end-users. This single system image provides networking transparency of resources. Resources were defined as being files, programs, and devices. All applications can be ported unmodified to this environment and can transparently access network resources. The system was called Business Communication Subsystem (BCS) and included the following environments: VAX/VMS, Xenix, MS-DOS, and BSD UNIX.

BCS DEFINITION

INTRODUCTION

This paper defines BCS and discusses its rationale and generic architecture. The paper then details a test bed implementation of a subset of the concepts defined. Finally, a comparison of BCS with other competitive approaches is discussed.

Business Communication System (BCS) is a substrate for business communication services with an open communications architecture to provide a transparent distributed environment for off-the-shelf applications. These applications access network resources as if they were local. This substrate is portable to voice/data PABX's and LAN's with appropriate data communication

services.

The major BCS issues of providing a substrate for business communication services, transparency, and open architecture are discussed further below.

Substrate for Business Communications Services

The problem of heterogeneous networking is a complex one. Any reasonable implementation must by necessity be limited to a set of "core" services and depend on off-the-shelf packages for application-specific functions. If BCS were to define a new *Network Operating System* it would then be limited to a restricted set of application packages. We designed BCS to capitalize on the existing set of software applications on each target environment and we extended the underlying network *substrate* to facilitate transparent network resource sharing. That is, BCS provides run-time extensions to environments such as MS-DOS and XENIX so programs written for them can run without modification and be able to access local and remote resources.

Transparent Distributed Environment

In a distributed environment, if access to remote resources can be accomplished as if these resources were local, then to the end-users and the using applications, distribution will be truly transparent. To solve the accessibility and integration problems described previously BCS should provide a *Single System Image (SSI)* which will make resources available in a transparent manner. The SSI environment is the system software that resides on all cooperating processors and provides users and applications with the illusion of operating on a

single centralized computer. Hence, the Single System Image is presented from the user/applications' viewpoint and not the networks'. SSI should provide for a global name space which is used to identify objects in the system such that:

- Names do not have to carry location information,
- It is at the same time possible to specify location,
- Objects cover files, programs, and devices.

A primary goal of BCS is to provide a ubiquitous interface which provides transparency of the details or existence of the "glue" between dissimilar system components. For a system to provide complete transparency, it must address the following:

- application transparency,
- location transparency,
- control transparency, and
- performance transparency.

The first three items form a hierarchy where application transparency is the foundation of the ubiquitous interface. *Performance transparency* spans across all levels of this hierarchy. Performance transparency implies that a networked environment should perform within the performance bounds of a true single system.

Application transparency assures that applications operate correctly without modification. This implies no source changes, re-compiling, re-linking, or re-loading. This will allow the full use of off-the-shelf applications.

Location transparency is the capability to access resources (files, devices, and programs) without reference to their network locations. This capability may be provided by the implementation of a global name space for all resources across a network.

Control transparency ensures consistency of the control of network processes. Processes can be transparently created, executed, and controlled from multiple points in the network and can communicate with cooperating processes.

Open Communications Architecture

The complexity of the problem also requires that a framework be first developed to allow open communications: the framework provides a layered organization according to ISO-OSI recommendations [Ref. 14]. It also provides ISO OSI recommended services at layer 4 which can be easily ported onto a variety of networks providing layers 1 to 3. Services at layers 5 to 7 are defined as either accepted standards or defacto standards. This facilitates the acceptance of BCS within other vendors' product lines.

BCS ARCHITECTURE

1. Conceptual Model

Figures 1 and 2 show a conceptual model for a typical business communications system. The basic concept is that users utilize services via some communication network. The users are hosted on what is labeled as the end-user systems (PC's, data terminals, telephones, etc.). The providers of multi-user services are hosted on the servers (VM,

MVS, VMS, XENIX, etc.), supporting off-the-shelf packages such as: PROFS and ALL-IN-1. The communication network can be realized as either a virtual wire (i.e. ISO Layer 3 functions), or as a partner with the end-user systems and servers in providing integrated business services (i.e. ISO Layer 4 to 7 functions).

In this model (Figure 2), BCS is a system that tightly glues together different machines over a communication network. The Intel 286/310 and MicroVAX servers ("internal servers") share an object name space and transparently share resources. They use standard operating systems that have been augmented to cooperate over a network. On these internal servers reside off-the-shelf, unmodified, applications. Users on either PCs running MS-DOS or terminals connect via the communications transport layer into these internal servers. MS-DOS machines can also reside inside the BCS umbrella as transparent servers. The native operating systems interfaces and shared object name space allow users to visualize these internal servers as a single system with the access and power of the union of all the resources. For example, a Xenix user would visualize the common pool of BCS resources in a Xenix context (syntax and semantics), while an MS-DOS user would view these same resources within a MS-DOS context.

BCS also provides access to external servers, i.e. VMS, XENIX, MS-DOS, VM and MVS. These servers provide the end-user with loosely coupled services including: file transfer, electronic mail, and remote execution. These external machines do not share the BCS name space and cannot cooperate with internal servers in the highly

integrated fashion previously described.

Thus, the BCS architecture can be positioned as shown in Figure 3 in the OSI model. The lower three layers are provided by the communication network (e.g. voice/data PABX or LAN). Layer 4 is the interface between BCS and the communication network. BCS protocols necessary to provide the BCS services are at Layers 5 through 7. Finally, at the top of this figure are the off-the-shelf applications that access network resources via BCS. The communication backbone layers (i.e., 1 to 3) can be any network that provides basic Layer 3 services. This functional organization of BCS functions has the advantage that it can be ported to any communication network, only the translation of BCS services to the particular data communication network services has to be accomplished inside of Layer 4. The separation of functions within the ISO-OSI framework gives BCS a high level of network and hardware independence.

Software Structure

Using the analysis technique sketched above, detailed studies of end-user's resource-sharing requirements were performed. The result is the functional hierarchy shown in Figure 4.

The lowest rectangle is the *Data Communication Network*. As mentioned in the previous section this can be any network that provides basic OSI Layer 3 services. The BCS software structure above the *Data Communication Network* is independent of the network.

BCS provides:

- A *Data Communication Services* functional area augments the particular data communication network chosen for a given implementation. This delta is necessary to provide the required data communication services for the other BCS functional areas.
- A host *Operating System* interface area, so modifications can be done to access the network and cooperate with the other BCS pieces. These modifications are done without affecting the external operating system interfaces.
- A *Name Server* to provide the common name space for network objects.
- An *Inter-Process Communication* facility for use by various BCS components (i.e. remote execution, loosely coupled services, network management) to communicate between cooperating processes on two or more servers.
- *File Management* is the facility that provides a common set of file functions across heterogeneous sets of machines.
- *Remote Execution* allows the execution of batch and interactive programs independent of user or program location.
- *Loosely Coupled Services* are those basic services to external servers (i.e. file transfer, electronic mail, and application access).
- *Shared Devices* is the ability of a cooperating machine to offer a device it hosts to the BCS network. This allows applications and users to access a remote device as if it were local.
- *Data Management* is the area that addresses the sharing of data between

diverse applications and environments.

- *Network Management, Maintenance, Administration, and Security* provide BCS with the necessary control of the network and its objects.

Hardware Configuration

The BCS hardware configuration is dependent on the data communication subnet on which it is built. As mentioned previously BCS can exist on any network that provides basic ISO level 3 services. The BCS software interface is defined at the transport level (layer 4) and all upper layers communicate via this defined interface. In its initial implementation phase, BCS is implemented on a popular Local Area Network (Ethernet). Subsequent phases will concentrate on the porting of BCS to a voice/data PABX. The major goal of these future implementations would be to bridge the voice and data within BCS's transparent environment.

TEST-BED IMPLEMENTATION

Figure 5 shows the organization of the test-bed from a physical view-point. This figure illustrates BCS hardware configuration as applied to a Local Area Network. The protocol structure in this case is Ethernet for the first two layers, a null Layer 3 (we did not address inter-networking in the Test-bed), ISO transport class 4 (utilizing Intel's ina960 product) as Layer 4, and OpenNET (using the standard file sharing protocols specified by Intel, Microsoft, and IBM [Ref. 8]) as Layers 5 through 7. Some upper layer protocols must be added to implement the BCS functional areas listed above.

The Test-Bed demonstrates XENIX, and VMS Servers and MS-DOS engines connected via a LAN as part of a transparent distributed environment. They share the same object name space and can access network data, programs, and devices transparently. In this example we show sets of homogeneous servers (ITT 993X and iAPX 286/310s running XENIX, MicroVAX II running MicroVMS and VAX-11/750 running VMS), cooperating as heterogeneous servers in the same *Single System Image* network. Services in this network can be accessed from any MS-DOS or XENIX machine.

Details of hardware and software components that are purchased and/or developed are given below.

Hardware Implementation

In Figure 5, the servers (ITT 993X, iAPX 286/310, VAX-11/750, and MicroVAX) and the MS-DOS machines share an Ethernet backbone. The end-user terminals are connected to a terminal server (MicroVAX) on the Ethernet. This terminal server provides the virtual link from the terminal to any target host and also performs data compression and multiplexing. The ITT 993X, VAX-11/750, and iAPXes provide file and print services with the 993X also serving as the communication gateway to IBM hosts. These links to external servers are noted in the general BCS model (Figure 2). They are shown here connected to an ITT 94XX, but they might as well be directly connected to the Ethernet as a gateway interface.

The test-bed was implemented by integrating the

following hardware systems, LAN, and communications interfaces:

- Server 1: consisting of an iAPX 286/310 with 1 MB RAM, a 40 MB hard disk, an Intel 311 expansion box for an additional 40 MB hard disk and tape unit, and an iSXM 552 Ethernet communication board.
- Server 2: consisting of an iAPX 286/310 with 1 MB RAM, a 40 MB hard disk, an Intel iSXM 552 Ethernet communication board and an Okidata 2350 dot matrix printer.
- Server 3: consisting of an ITT Courier 993X Application Processor with 5 MB RAM, a 40 MB hard disk, an APDA device adapter board (for attachment to the Courier 994X cluster controller), and an iSXM 552 Ethernet communication board.
- Server 4: consisting of a DEC MicroVAX II with 3 MB RAM, a 71 MB hard disk, and an Excelan EXOS 203 Ethernet communication board.
- Server 5: consisting of a DEC VAX-11/750 with 8 MB RAM, 1GB hard disk, and an Excelan EXOS 204 Ethernet communication board.
- Network Monitor: consisting of an ITT XTRA XP with 0.5 MB RAM, a 20 MB hard disk, an Ungermann-Bass NIU Ethernet communication board, color-graphics adapter card and a color-graphics monitor.
- Workstation: consisting of an ITT XTRA XP with 0.5 MB RAM, a 20 MB hard disk, an Ungermann-Bass NIU Ethernet communication board and a SmarTEAM 103/202 modem.

Software Implementation

Software modules were implemented for

application support (inter-processor mail), file access, terminal access and network management in the test-bed environment. The majority of the code is in the C language with the exceptions being the VMS and MicroVMS device drivers that required the use of the MACRO-32 assembler. There was 12 thousand lines of code (KLOC) written which represents 100Kbytes of object spread over four different systems (VMS, MicroVMS, XENIX and MS-DOS).

The software modules are summarized below. The relationship of protocol layers used for the file and terminal access in the test-bed implementation are given in Figures 6 and 7, respectively.

The software modules implemented are:

- MicroVMS Software: consisting of a master virtual terminal interface and an EXOS 203 driver with supporting utilities.

The virtual terminal interface is implemented to allow users to setup single or multiple sessions to any BCS servers. The EXOS driver is a MicroVAX II VMS device driver that handles low level interface functions to and from the EXOS 203 controller board. The EXOS board hosts the ISO transport class 4 software which functions as ISO layers 1 through 4. The EXOS support utilities include iNA960 software downloading, status monitoring and reporting, and network statistics collection and transmittal.

- XENIX Software: consisting of slave virtual terminal support, network management support, and inter-processor mail support modules.

Virtual terminal support is needed because the XENIX machines also act

as *virtual terminal servers* or slaves, which interact with the MicroVAX *virtual terminal consumer* or master. Functionally, these modules interface to the session layer and iSXM 552 interface driver.

Network management and inter-process mail support modules are used to generate responses to the configuration and statistics requests from the Network Monitor (ITT XTRA XP), and to enhance the mail services to/from Workstation users.

- VMS Software: consisting of a MS-DOS and Xenix file and print services. In addition, the same set of MicroVMS EXOS drivers and utilities ported to EXOS 204/UNIBUS and VMS.
- MS-DOS Software: consisting of inter-processor mail support and network monitor modules.

The inter-processor mail support module performs identical functions as its XENIX counterpart, i.e., enhancing the send and receive mail functions to/from XENIX users.

The network monitor software extracts information from the various protocol layers of the network, performs analysis and finally presents the data graphically on the XTRA color-graphics monitor.

Additionally, a number of off-the-shelf software packages were integrated for a demonstration of the Test-bed implementation. These are: IBM PC-DOS 3.1, MS-NET, DEC VAX VMS 4.1, Microsoft XENIX 3.0, XENIX mail, kermit, and Intel OpenNET.

COMPETITIVE APPROACHES

The document "A Study of Single System Image

Environments for the Business Services Subsystem (BSS)" [Ref. 2] details an analysis of products that operate over heterogeneous hosts and provide some level of resource transparency. The products are either commercially available or subjects of university research. The paper defines a base architecture and services, and details the selection criteria. This analysis resulted in the selection of a system to be used in the Test-Bed implementation. The system chosen for the Test-Bed was the combination of XENIX-NET and MS-NET. Its functionality, adherence to networking standards, cost of implementation/porting, and performance were noted. XENIX-NET and MS-NET, however, do not provide many of the features that LOCUS and other "distributed operating systems" [Ref. 12] provide such as:

- a single file with data distributed and/or replicated over many sites,
- distributed processes to support the creation and communication with remote processes,
- application-locator services to find out, e.g., whether there is a PASCAL compiler on the system, and forward user request to the appropriate server,
- other utilities including time services, boot services, etc..

These features were not part of the selection criteria of the study.

Since the technical memorandum was released two major products have come into prominence. The Network File System [Ref. 5] from Sun Microsystems Inc. and Remote File System [Ref. 4] from AT&T. RFS is targeted at UNIX networking while NFS has a heterogeneous goal.

RFS from AT&T is the choice for UNIX networking. RFS adheres strictly to UNIX semantics and standards, supports a transparent application interface, and provides a uniform methodology for implementing varying network protocols. However, RFS is a new technology and currently does not support heterogeneous hosts (e.g. XENIX, MS-DOS, VMS). RFS does, however, provide a defined software architecture in which these heterogeneous ports can be implemented.

The NFS system as specified by SUN Microsystems is a heterogeneous implementation. It has been ported to MS-DOS and VMS systems as well as a wide spectrum of UNIX systems (BSD 4.2/3, UNIX System V Release 2, Xenix System III and V, and Microport UNIX). It supports common file operations (UNIX like) over this set of machines, but does not address the issue of location or object transparency.

DECnet-DOS from Digital Equipment Corporation ties together the MS/PC-DOS world and the VMS/MicroVMS worlds utilizing the DECnet protocol suite. DECnet-DOS provides: virtual terminal emulation (CTERM or LAT), bidirectional file transfer (using a DOS based utility and VMS using FAL), a unidirectional VMS mail utility for DOS, transparent file access library, transparent task-to-task communication library, and a network disk driver for DOS which provides virtual network disks (similar to PC-NET/MS-NET functionality). All of these services run over an existing Ethernet or Asynchronous DECnet phase IV network. DECnet-DOS in conjunction with its sister DECnet-ULTRIX ties together VMS, ULTRIX, and MS-DOS. Each

environment maintains its name space and administrative autonomy.

VAX/VMS Services for MS-DOS from Digital Equipment Corporation is the new entry in the heterogeneous distributed computing marketplace. It is the NETBIOS implementation of DECnet-DOS designed specifically for the VAXmate processor. It provides: virtual terminal (VT220 emulation) support, virtual MS-DOS disk on VMS with full file sharing, and a standard IBM NETBIOS interface for MS-DOS network applications. This product does not support XENIX or UNIX (non-ULTRIX) connectivity, a global name space, or VMS client services to MS-DOS.

There are other connectivity products from SYNTAX and Datability that address the MS-DOS and VMS connectivity using industry standard protocols.

CONCLUSIONS

Overall, we believe that the BCS Test-bed succeeded in demonstrating the feasibility of a *Single System Image* for distributed business communications: it is indeed possible to integrate existing heterogeneous machines (MicroVAX, ITT Courier AP, Intel 286/310 and ITT XTRA) into a networked system and preserve distribution and resource transparency over the diverse machine environments.

In summary, BCS is similar to NFS in its heterogeneous nature, and to RFS and LOCUS in its full support of UNIX file operations. However,

NFS, LOCUS, and RFS use a remote mounting technique which masks the network entry point of a file system tree. BCS uses a network layer above each machine's file system root. In addition, a BCS logical name or symbolic link mechanism is used to syntactically hide the existence of the network to users and applications.

To conclude, we emphasize that although the current BCS test-bed only supports a limited set of capabilities, it shows that the *Single System Image* concept is indeed feasible, and that the BCS architecture is a sound approach towards its implementation.

ACKNOWLEDGMENTS

There are many individuals who contributed to the ideas, analysis, specification, and Test-Bed implementation of BCS. We would like to thank Santanu Das, Director of the Applied Technology Division, for his support of the project. We are especially grateful to Shastri Divakaruni, and Walter Guilarte for their management support and guidance. The initial conceptual view of BCS was assisted by John Thalhamer, Doreen Lawson, and Baldev Singh. A final thank you to Hsiaosu Hsiung for his contributions in the specification and Test-Bed implementation of BCS.

REFERENCES

1. H. Kilman and F. Schoen, "A VAX/VMS Implementation of XENIX-NET and MSNET", Proceedings from 1987 Spring DECUS
2. E. Berelian, W. Guilarte, H. Kilman, "A Study of Single System Image Environments for the Business Services Subsystem (BSS)", Doc. No.

MF851127, ITT Advanced Technology Center, February 8, 1985.

3. R.S. Divakaruni and H. Kilman, "Futuristic Trends in an Integrated Office Environment", May 11, 1985 proceedings from VENCOT '85
4. AT&T, "UNIX Version V Release 3.0 Overview", AT&T Information Systems, 1986.
5. Sun Microsystems, Inc., "Networking on the Sun Workstation", Part No. 800-1324-03, Rev. B., dated February 17, 1986.
6. M. J. Weinstein, T. W. Page, B. K. Livesey and G. J. Popek, "Transactions and synchronization in a distributed operating system", ACM Proc. 10-th Symposium on Operating System Principles, December 1985.
7. For a recent review, see A. S. Tanenbaum and Robbert Van Renesse, "Distributed Operating Systems", Computing Surveys, vol. 17, no. 4, December 1985.
8. Microsoft Corp., Intel Corp., "Core File Sharing Protocol", PN 136329-001 Rev.A, Version 1.6, June 20, 1985, and "Extended File Sharing Protocol", PN 136330-001 Rev.A, Version 1.6, June 12, 1985.
9. Intel Corp., "XENIX Networking Software User's Guide", Order No. 135147-001, 1985.
10. Intel Corp., "Intel OpenNET Architecture Xenix Network File Services User's Guide", 1985
11. Locus Computing Corporation, "The LOCUS Distributed System Architecture", Edition 3.1, June 1984.
12. Popek, G., Walker, B., Chow, J., Edwards, D., Kline, C., Rudisin, G., and Thiel, G., "LOCUS A Network

Transparent, High Reliability
Distributed System", UCLA.

13. Hac, A., "Distributed File Systems - A Survey", SIGOPS February 1985
14. CCITT VIIIth Plenary Assembly, "Data Communications Networks. Open Systems Interconnection (OSI) System Description Techniques Recommendation X.200", Red Book Volume VIII Fascicle VIII.5, October 1984.
15. Intel Corp., "ina960 Programmer's Reference Guide", 1985
16. International Standards Organization, "Information Processing Systems - Open Systems Interconnection - Transport Protocol Specification", ISO/TC 97/SC 16/WG 6, DIS 8073 Rev., June 29, 1984.
17. M. J. Hatch, M. Katz and J. Rees, "AT&T's RFS and Sun's NFS: A Comparison of Heterogeneous Distributed File Systems", Unix/World, December 1985.

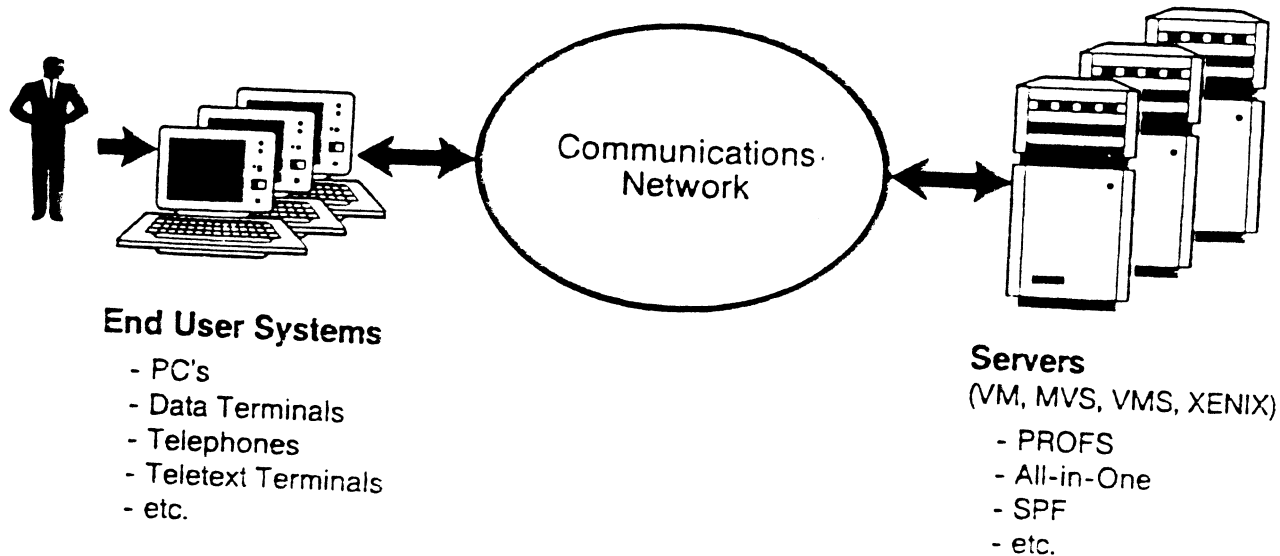


Figure 1: Conceptual Model for Business Systems

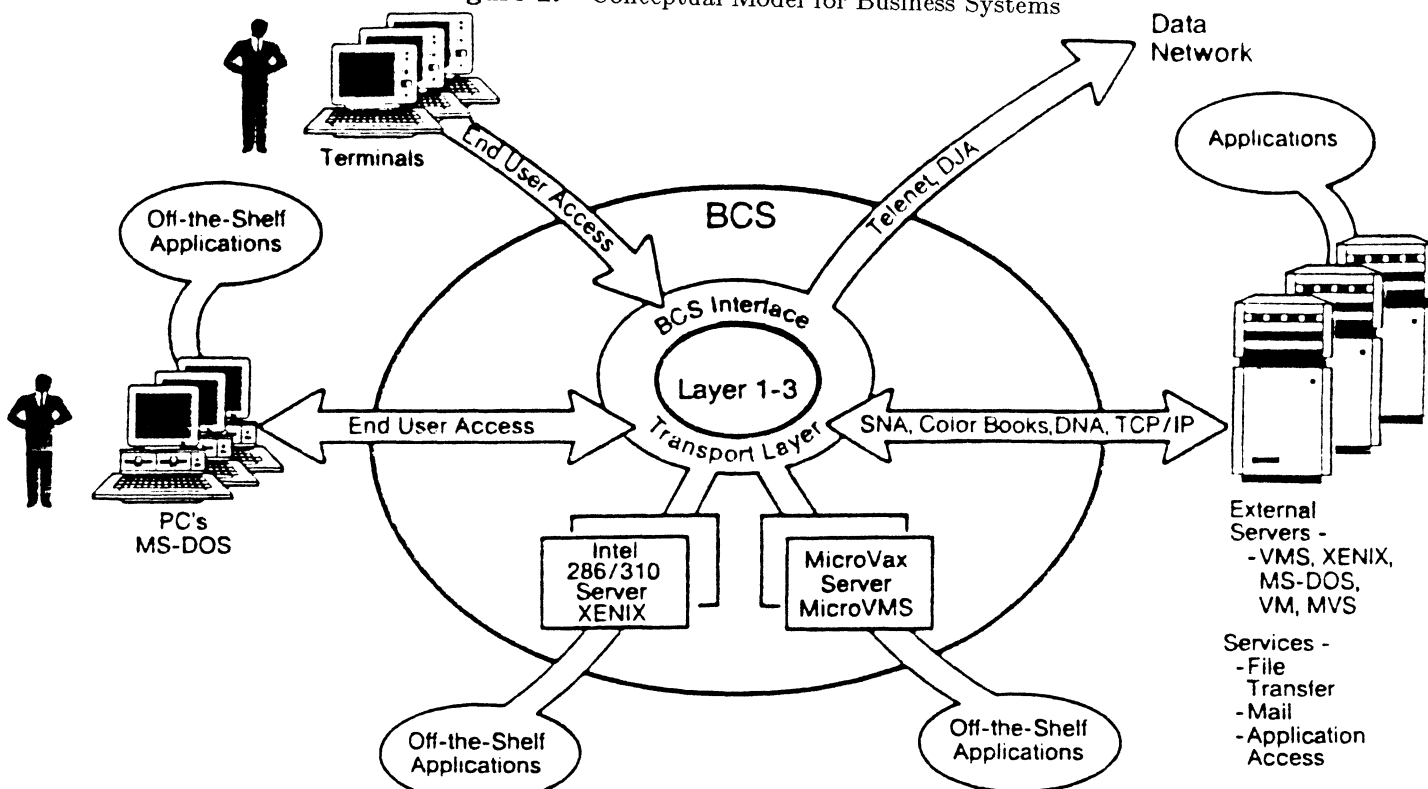


Figure 2: Practical Model for Business Systems

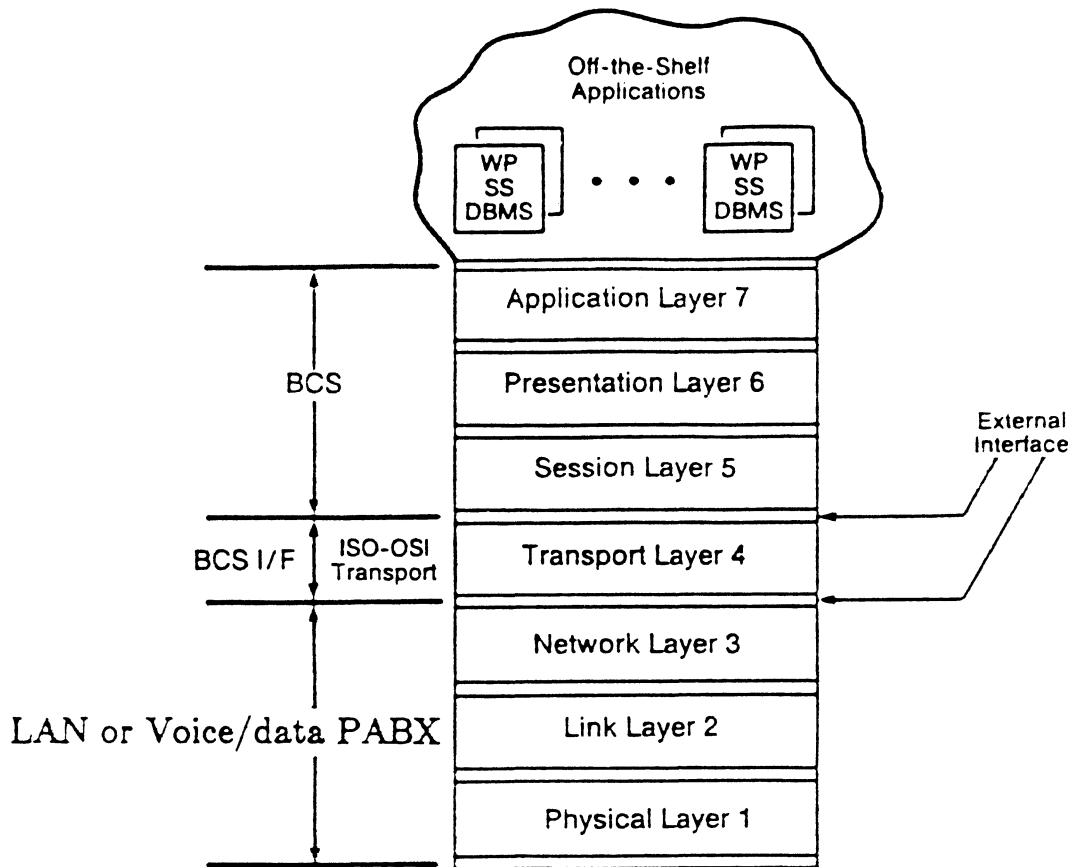


Figure 3: BCS Architecture (OSI-ISO)

Shared Devices	Loosely Coupled Services	Data Management	Network Management, Maintenance, Administration
Remote Execution			
File Management			
Inter-Process Communication			
Security	Data Communications Services	Name Server	
		Operating System Interface	
Data Communications Network LAN or Voice/Data PABX			

Figure 4: BCS Software Structure

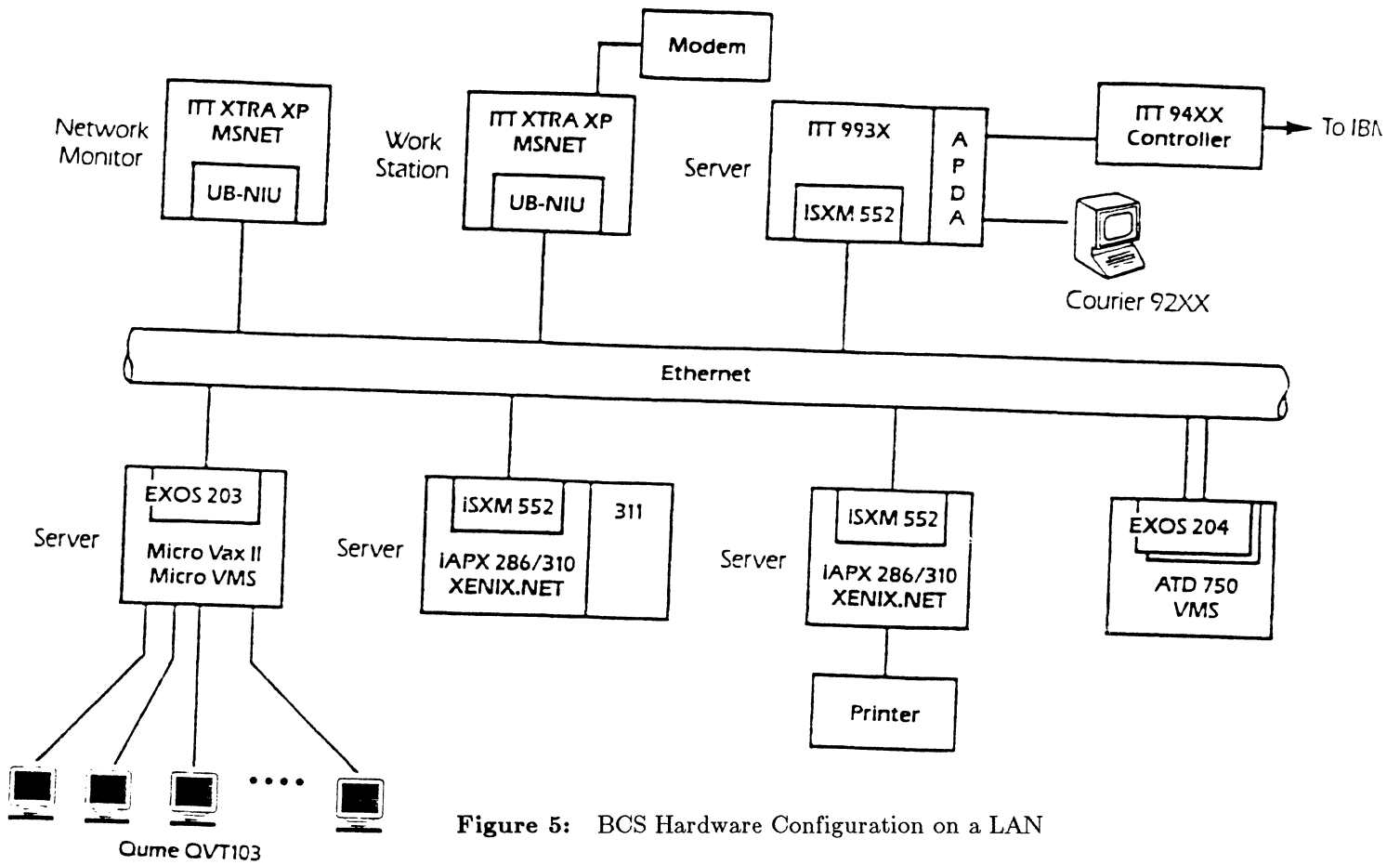


Figure 5: BCS Hardware Configuration on a LAN

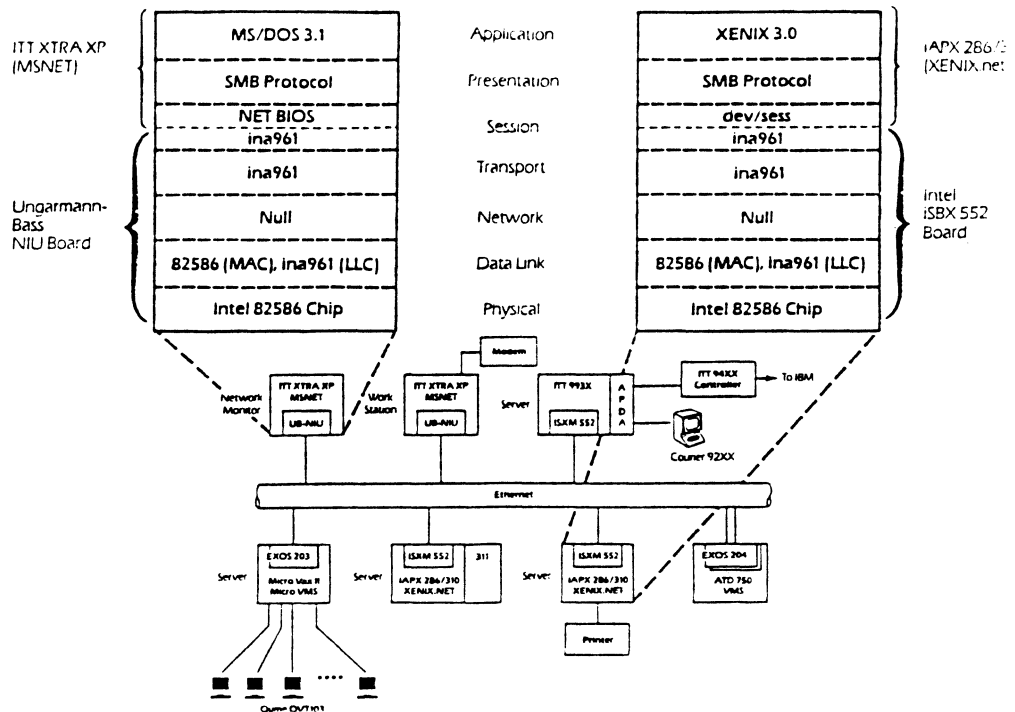


Figure 6: File Access Protocol Architecture

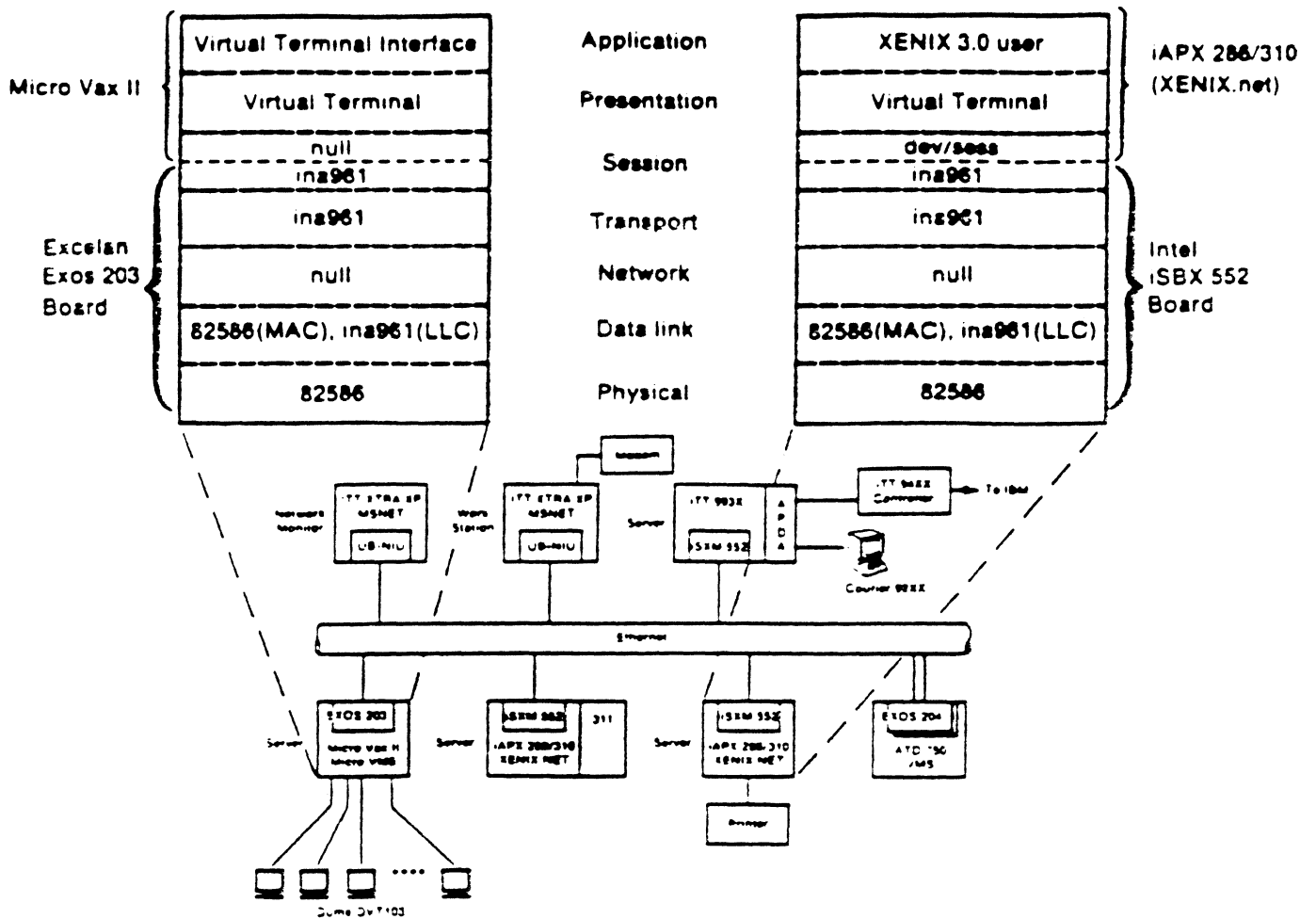


Figure 7: Virtual Terminal Protocol Architecture

OFFICE AUTOMATION SIG

Documenting Single-Package Systems

Customer Benefits and Documentation Challenges

Michael J. Doyle

Digital Equipment Corporation

Maynard, Massachusetts

ABSTRACT

What are single-package systems? Why do customers like them? What are the challenges in documenting them? This paper discusses the benefits (to customers) and the challenges (to creators) of single-package systems and the documentation that accompanies them.

Office Systems Documentation (OSD) is a group of writers, editors, and publications personnel responsible for providing documentation for several of DIGITAL's office automation (OA) products. We recently documented PC ALL-IN-1, which presented a new documentation challenge for us: how to document a single-package system.

What Are Single-Package Systems, Anyway?

Single-package systems are "ready-to-run" systems that contain the components necessary to solve one or more computing problems. This does not mean that all of the components come in a single physical package. Instead, single-package is an ordering concept that refers to one order number for a complete solution—with the focus on solution.

A single-package system consists of:

- Hardware
- Software
- Services
- Documentation

Single-package systems are not meant to be customized to any great extent, and they require no system generation. They very much represent a "solution in a box."

Let's take a look at PC ALL-IN-1 as a single-package system. PC ALL-IN-1 consists of a MicroVAX II Server that allows PCs to network together and, optionally, connect into a larger corporate network. Some of the main components of this system are:

- MicroVAX II Server (preconfigured)
- VT220 console terminal package
- LN03 laser printer package
- ThinWire Ethernet hardware for ThinWire connections
- Cable concentrator for asynchronous connections
- Software for the Server
- Software for the PCs
- Services and onsite Server Administrator training
- Documentation

Except for the PCs and the cabling from the PCs to the Server, everything needed to create a PC network is included in the package.

What are the Benefits to Customers?

The biggest benefit to the customer of single-package systems is that the systems are easy to order. The customer doesn't have to figure out what parts are needed to solve a problem. In fact, the customer doesn't even have to figure out a solution.

The customer need only *recognize* the problem and find the order number that provides the solution to that problem. This solution includes the necessary hardware, software, services, and documentation. Since the system has been tested and used as a system, not as a bunch of parts which may or may not solve the problem, the customer is assured that all the pieces work together as a unit.

In addition, because the pieces are packaged as a system, installation and maintenance is simplified. The installation and maintenance processes are performed for the system, not for each individual piece of the system.

All the benefits to the customer are easy to recognize: single-package systems are the simplest solution to a computing problem. However, there seems to be an inverse relationship between simplicity on the customer's part and complexity on the creator's part. This relationship holds true for both the creators of the product and the creators of the documentation.

What are the Challenges to Documentation?

Writers in OSD have historically provided user communications (hardcopy and online documentation) for OA software. We have also provided help with software interface issues, and help with the quality assurance of our software products.

Because PC ALL-IN-1 was, in the true sense of the phrase, and "all in one" package, we needed to modify our product responsibilities, what we write, who we write for, and how we work.

Product Responsibilities

Each component of PC ALL-IN-1 represents an important part of the system; each component also represents a dependency. The MicroVAX II is developed by one group; the VT220 console terminal is developed by another group; and the LN03 laser printer package is developed by yet another group. The PC ALL-IN-1 documentation team faced communication and organizational challenges because of the number of groups involved in the product.

To what extent did the number of groups involved affect our product responsibilities? Figure 1 shows the major groups involved in bringing a traditional software product to market. Notice that historically the development of OA products rests with one software development group.

Figure 1
Groups Working On Traditional OA Product

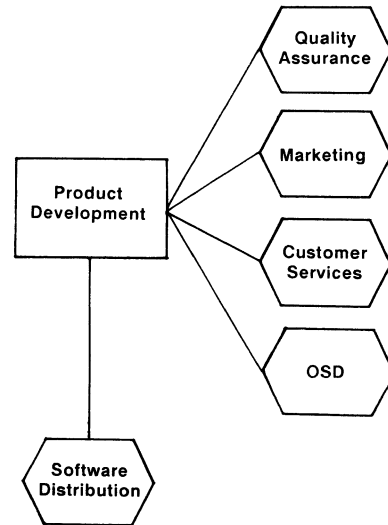
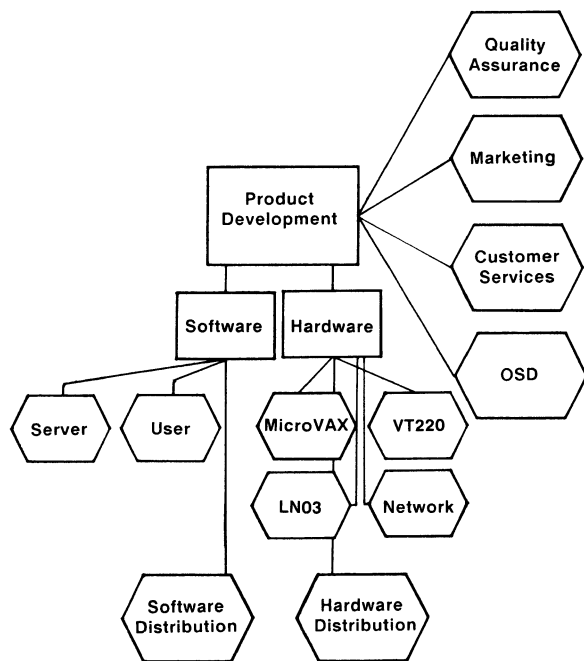


Figure 2, however, shows how the major groups that make a single-package system are dependent on many development groups - both software and hardware. The number of dependencies increases dramatically, and the importance of organization and communication becomes clear.

Figure 2
Groups Working On PC ALL-IN-1



Because of the number of development groups contributing to the product, a good portion of the responsibility of keeping a system focus fell on the shoulders of documentation. To a great extent, it became our responsibility to make the product look and work like a single system, through well-integrated documentation.

What We Write

As a group, our experience in the past has focused on layered products, which generally require user, system manager, and installation documentation. With PC ALL-IN-1, we needed to provide documentation for an entire system. We had to provide documentation for:

- Planning for the system. We documented the work that needs to be done at the customer site in preparation for the installation of a PC ALL-IN-1 system.
- Ordering the system. We documented the components of the basic system package. We also documented the additional packages that are available for the system, including the additional printer, modem, PC, and cabling packages.

- Installing the hardware. We documented the steps for hardware installation, tying all of the hardware pieces together as a system.
- Installing the software. We documented the steps for installing the software on both the MicroVAX II Server and the PCs.
- Using the system. We documented the various components of the user software, including the WPS-PLUS editor, electronic mail, and the document File Cabinet.
- Maintaining the system. We documented the steps for maintaining the Server, including information on how to maintain user accounts and backup data.

With PC ALL-IN-1 we were no longer responsible for documenting a component of a system; instead, we were responsible for documenting the system itself - all of the components working as a unit.

Who We Write For

The different components of the system required documentation that was targeted to the user of that component. There is no single audience for an entire system.

PC ALL-IN-1 challenged us to write to these different audiences. Instead of writing solely for the OA user sitting at a terminal, we wrote for the:

- Customer planning and ordering the system
- Hardware specialist installing the system
- Users using the system
- Server Administrator maintaining the system

How We Work

Challenges in how we work were the most difficult to meet because they required changes in our basic work habits. We had to move from informal work habits that resulted from working with small project groups, to more formal habits that were needed to work with the many development groups contributing to the project. As you can probably guess, it is a very difficult process to move from informal to formal work habits. Certainly it is more difficult than the reverse (formal to informal).

Our main job as technical writers is to make sure that the information we provide is technically accurate. Because of the variety of groups involved in developing the system, our major challenges seemed to focus on:

- Getting accurate information
- Getting thorough technical reviews
- Resolving conflicting technical comments

Conclusions

Are single-package systems worth the effort? Because they are the simplest solution for some computing problems (not all - some), the answer has to be a resounding YES.

The success of single-package systems clearly depends more heavily on documentation than the success of traditional layered products. With single-package systems, documentation is responsible for:

- Documenting all parts of the system, including hardware, software, and services.
- Documenting all tasks associated with the system, including planning, ordering, installing, using, and maintaining.

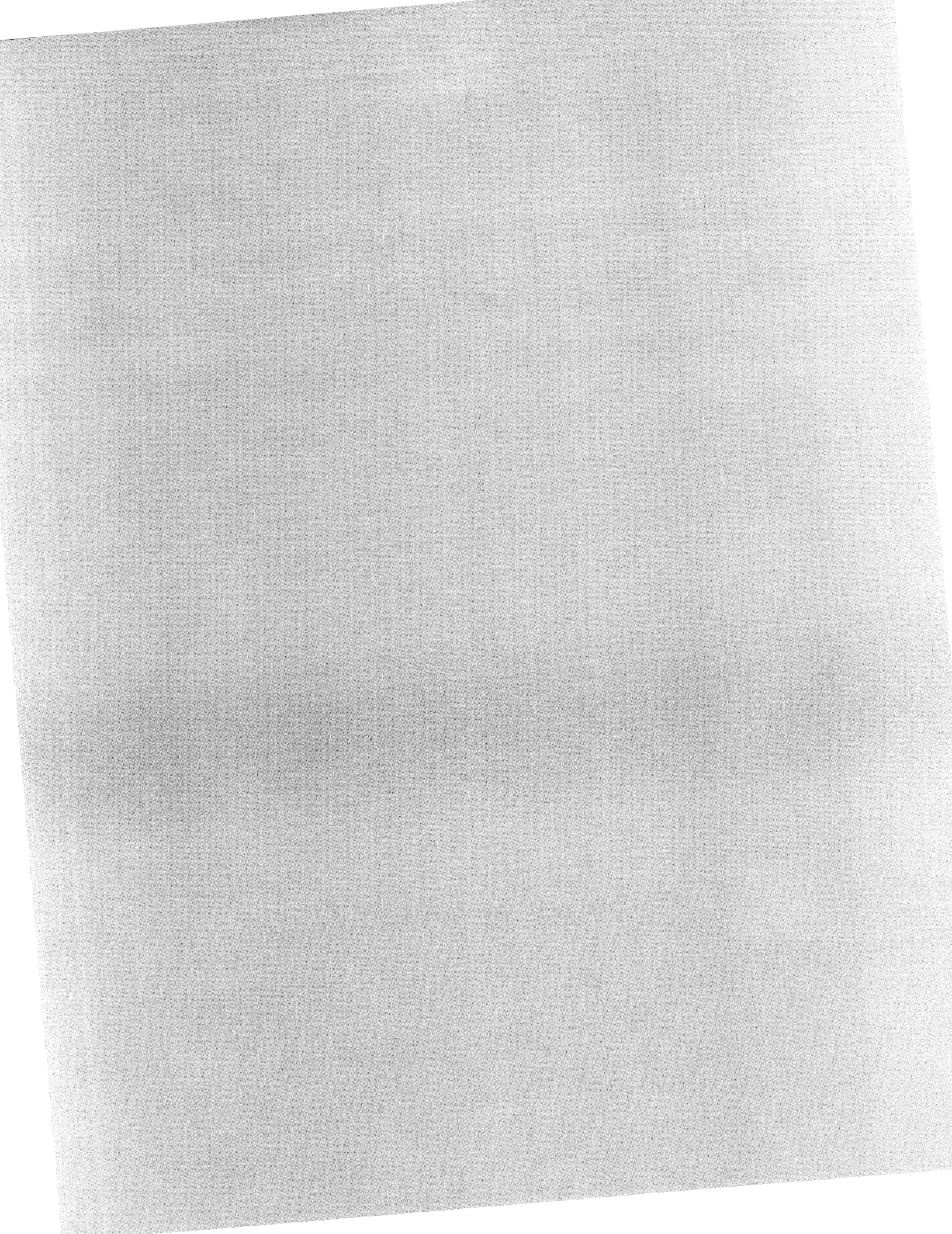
- Keeping a system focus.

To meet the challenges that such responsibilities present, writers must follow the same two-step cycle that all people involved in new areas must follow: we must learn by trial and error, then learn from experience.

We must pay attention to both our successes and failures, and in the future emphasize our successes and weed out the failures. We must listen to and seek out comments from all users, listen to their feedback, and respond. Our documentation goal is to make the information users need as accessible as possible.

Future work on the documentation for single-package systems must also strive for fully integrated documentation. For a single-package system to be completely successful, it must look and feel like it was developed by one small development group.

Fully integrating the documentation for a single-package system requires writers to further develop communication, organization, and negotiation skills as we work with larger numbers of development groups for a single product. Because of the diversity in the developmental background of the components in a single-package system, documentation must provide the communication thread that ties everything together.



Introduction to the RSX, P/OS, and RT Indirect Command File Processor

Thomas R. Wyant, III
E. I. DuPont de Nemours
Richmond, Virginia

Abstract

This paper presents a survey of the basic capabilities of the indirect command processor (ICP) that is common to RSX, P/OS, and RT. A useful and efficient subset of the ICP directives is presented, some common misconceptions are addressed, and known bugs in the various implementations of the ICP are highlighted.

Introduction

Goal

The goal of this paper is to help the reader or listener to make efficient use of the Indirect Command File Processor (ICP) under RSX, P/OS, and RT.

Caveats

The current releases of all the operating systems are assumed. However, this paper should be useful as far back as:

- RSX-11M V3.2
- RSX-11M+ V1.0
- VAX-11 RSX V1.0
- P/OS V1.0
- RT-11 V5.0 (FB and XM)

Where there are differences between the Indirect Command File Processors of these systems (and I am aware of these differences) they are indicated. Command lines in examples are all for RSX MCR.

I am not the final authority on the ICP in all its multifarious versions. Errors in research and transcription do occur. I apologize in advance for these, but assume no responsibility for their consequences.

Acknowledgement

Allen A. Watson's paper, "Indirect Command Files for New RSX Users", presented at the Spring, 1983 DECUS US Symposium, was both an inspiration and a reference for this paper.

Background

Historical Perspective

The RSX Indirect Command Processor (hereinafter known as the ICP) is the RSX system component that allows you to group MCR or DCL commands in a file and have them executed as a group. The same basic processor was put in P/OS when it was spun off from the RSX mainstream, and the same logic has within the last couple years been exported to RT-11.

Invoking the ICP

To feed a command file to the ICP:

RSX

```
>@filename
```

(Default filetype is .CMD)

P/OS

```
$ @filename
```

(Default filetype is .CMD)

RT

```
. SET KMON IND  
. @filename
```

or

```
. IND @filename
```

(Default filetype is .COM)

In all cases the input is to a CLI prompt. @filename in response to some other prompt has nothing to do with the ICP. For example,

```
PIP>@filename
```

feeds the file to PIP's own command processor.

Contents of an ICP file

Each line of an ICP file is composed of one of the following:

- An “external” comment:
 - Begins with ; .
 - Is displayed when encountered (unless .ENABLE QUIET is in effect).
 - Has no other effect.
- An “internal” comment:
 - Begins with . ; .
 - Is not displayed when encountered.
 - Has no other effect.
- An ICP directive:
 - Begins with . .
 - Is interpreted and executed by the ICP.
 - Most of this paper deals with these.
- A CLI command:
 - Is any line that doesn’t meet the above criteria.
 - Is issued as a command to the current CLI, as though you typed it. If the current CLI can’t handle it, that’s your problem, not ICP’s.

Normally, only one of the above may appear on a line of an ICP file. The exceptions to this rule are:

- Most ICP directives may be followed by an internal comment. The output directives are the major exception to this.
- The ICP conditional directives may be followed by a comment, a CLI command, or another ICP directive (including another conditional).

Processing a line of an ICP file

A record (line) in an ICP file is processed in the following steps:

1. Read it.
2. Perform symbol substitution (if enabled).
3. Decide what category it falls in. If it’s an ICP directive:
 - (a) Parse the first “word”.
 - (b) Load the overlay that processes it (overlaid versions).
 - (c) Complete parsing.
 - (d) Execute.

Why is this important? Because:

- Symbol substitution occurs very early in the processing of a line. Many “LISP”-ish behaviors of the ICP are based on this.
- Many versions of the ICP are overlaid, so grouping like operations together can improve performance.
- Running an overlaid ICP off a floppy disk can require a good deal of patience.

Symbols

What are Symbols?

- They are named data stores used by the ICP.
- They are not accessible outside the ICP (though the ICP can make their VALUES available).
- Their names are 1-6 RAD50 characters (excluding .), and MUST contain at least one non-numeric character. Note that:
 - Certain releases of the ICP have had trouble with embedded dollar signs, or with symbol names that begin with a numeric character.
 - There are certain predefined Special Symbols, which are named according to the same convention, but have their names enclosed in <>. These do not conflict with ordinary names.
- They come in three flavors:
 - Logical (True of False).
 - Numeric (16-bit integer values).
 - String (0 to 132 bytes).
- They can be assigned values:
 - By computation within the ICP.
 - By querying the user.
 - In special cases, by reference or on entry to the ICP.
- Their values can be tested.
- They are normally local to the command file in which they were created. However, if you .ENABLE GLOBAL, symbols whose name begins with a \$ are available for the life of the ICP run.

Logical Symbols

- Take on the values TRUE or FALSE.
- Can be assigned values by:

```
.SETT symbol ! sets it TRUE.  
.SETF symbol ! sets it FALSE.  
.SETL symbol expression
```

- Logical expressions consist of logical symbols (including the Special Logical Symbols `!TRUEi` or `!FALSEi`), connected by the following operators:

```
!      (logical OR) .
&      (logical AND) .
#      (logical NOT) (RSX, P/OS) .
^      (logical NOT) (RT) .
```

Expressions are evaluated left to right, without regard for usual precedence of operators. Operations may be grouped with parentheses.

- Values can be tested by `.IFT`, `.IFF`, or `.IF` directives. These directives are discussed under “Control”.
- Can take on values entered from your terminal, using the `.ASK` directive. This directive is discussed under “Terminal I/O”.
- Substitution yields “T” for TRUE, or “F” for FALSE. Substitution is discussed later in this section.

Numeric Symbols

- Take on 16-bit integer values. These are unsigned. Under RSX and P/OS, you can treat them as signed values if you

```
.ENABLE OVERFLOW
```

- Can be assigned values by:

```
.SETN symbol expression
.INC symbol          ! Adds 1.
.DEC symbol          ! Subtracts 1.
.SETT [mask] symbol ! RT-11 only
.SETF [mask] symbol ! RT-11 only
```

The `.SETx [mask] symbol` operations set or clear the masked bits in the given numeric symbols.

- Numeric expressions consist of numeric symbols or constants, connected by the following operators:

```
+      (addition);
-      (subtraction);
*      (multiplication);
/      (integer division);
!      (bitwise OR);
&      (bitwise AND);
#      (bitwise NOT) (RSX, P/OS);
^      (bitwise NOT) (RT) .
```

Expressions are evaluated left to right, without regard for usual precedence of operators. Operations may be grouped with parentheses.

- Numeric constants are octal by default unless you append a decimal point, or unless you:

```
.ENABLE DECIMAL      ! (RSX, P/OS)
.DISABLE OCTAL       ! (RT)
```

- Values can be tested by the `.IF` directive. This directive is discussed under “Control”.
- Can take on values entered from your terminal, using the `.ASKN` directive. This directive is discussed under “Terminal I/O”.
- The default radix of a symbol is decimal, unless the expression that computed its value consisted only of octal symbols and constants. This default radix can be changed by:

```
.SETO symbol ! (set to octal);
.SETD symbol ! (set to decimal) .
```

- Substitution yields the value of the symbol, in the current default radix of the symbol. Substitution is discussed later in this section.

String Symbols

- Take on the value of a 0-132 byte string.
- Can be assigned values by:

```
.SETS symbol expression
```

If you `.DISABLE LOWERCASE`, the string is uppercased before being assigned to the symbol.

- String expressions consist of string symbols, substrings, or constants, connected by the following operator:

```
+      (concatenation) .
```

Expressions are evaluated left to right. Operations may NOT be grouped with parentheses, but then with only one operator, why would you need to?

- A string constant is constructed as follows:

```
"this is a string constant"
#so is this, for RSX and P/OS only#
```


The quoting character may not appear in the string constant.

- Substrings can be extracted by the construction:

```
symbol[start:end]
```

which represents the bytes between the start and end positions, inclusive. Any valid numeric expression can be used for “start” and “end”. Also, * can be used, representing the last character in the string.

- Values can be tested by the .IF directive. This directive is discussed under “Control”.
- Can take on values entered from your terminal, using the .ASKS directive. This directive is discussed under “Terminal I/O”.
- Substitution yields the bytes in the string. Substitution is discussed later in this section.

Symbol Substitution

- Occurs only when enabled by .ENABLE SUBSTITUTION. Under P/OS and RT, substitution is enabled by default. Under RSX, it is disabled by default.
- Is called for by enclosing a symbol name in apostrophes.
- If you attempt substitution on an undefined symbol, an error occurs.
- Substitution in a line of an ICP file occurs BEFORE the line is parsed. Therefore it can occur anywhere in a line, and in any kind of line.
- Under RSX and P/OS, you can get format control by following the symbol name (within apostrophes) by a percent sign (%) and one or more of the following:

```
D    (substitute decimal value);
O    (substitute octal value);
Rn   (right justify in "n" byte
      field);
Ln   (left justify in "n" byte
      field);
Z    (fill with leading zeros);
S    (signed value);
C    (do blank compression);
X    (substitute RAD50 string
      for number);
V    (substitute value for first
      byte, or a byte for value).
```

Examples (all of which assume .ENABLE SUBSTITUTION):

- Assembling and task building an arbitrary module:

```
.ASKS MODULE What module
MAC 'MODULE'='MODULE'
TKB @'MODULE'BLD
```

Effects:

- You are prompted for the name of a module;
 - That module is assembled and taskbuilt.
- Inserting control characters:

```
.SETN NJUNK 33
.SETS ESCAPE "'NJUNK%V' "
```

Effect:

- String Symbol ESCAPE now contains an escape character.
- Using format control to set the size of a field:

```
.SETN NJUNK 1
PIP FILE.'NJUNK%R3Z'/LI
```

Effect:

- A directory listing of file FILE.001 is produced.

Determining the Characteristics of Symbols

- Finding out whether a symbol exists:

```
.IFDF symbol    ! If it's defined
.IFNDF symbol   ! If it's not.
```

Note that the line

```
.IFDF symbol ;'symbol'
```

will result in an error if the symbol is undefined. The substitution is attempted BEFORE the line is parsed.

- Characteristics of a symbol:

```
.TEST symbol
```

causes the following Special Symbol values to be set:

<SYMTYP>
 = 0 if the symbol is a Logical Symbol,
 = 2 if the symbol is a Numeric Symbol,
 = 4 if the symbol is a String Symbol;

<OCTAL>
 = <TRUE> if octal (numeric and
 string only);

<STRLEN>
 = Length of string (string symbols only);

<ALPHAN>
 = <TRUE> if (uppercase) alphanumeric
 (string symbols only);

<NUMBER>
 = <TRUE> if a number (string symbols
 only);

<RAD50>
 = <TRUE> if a RAD50 string (string only).

Manipulating Substrings

- o Finding substrings of a string:

.TEST string1 string2

returns the following Special Symbol value:

<STRLEN>
 = the position of first occurrence of
 string2 in string1, or 0 if it
 doesn't occur.

Note that:

- The strings may be either string constants or string symbols.
- Extracting substrings based on character position was discussed under String Symbols.
- o Substrings based on character locations:

.PARSE string1 string2 symbol1 symbol2 ...

Takes string2 as a list of separators, and picks string1 apart. symbol1 gets everything up to the first separator, and so on.

- If there are more symbols than separator characters in string2, the last separator character gets reused.
- If all separators do not occur IN ORDER in string1, symbols corresponding to the missing separators come back with the null string.

- Special Numeric Symbol ;STRLEN; contains the actual number of substrings processed by .PARSE. This includes explicitly null substrings, but not symbols set null because .PARSE could not find any more separators.

For example:

```
.PARSE <UIC> "[,]" JUNK1 GROUP MEMBER
JUNK2
```

sets String Symbols JUNK1 and JUNK2 null, GROUP to your current UIC group, and MEMBER to your current UIC member.

Control

Conditional Directives

- o The general syntax of the Conditional Directive is

.IFx condition statement

The "statement" (directive, CLI command, or whatever) is executed ONLY if the condition is satisfied.

- o There are a number of cases of the conditional directive:

Syntax	Satisfied if:
.IFT symbol	symbol is true
.IFF symbol	symbol is false
.IFDF symbol	symbol defined
.IFNDF symbol	symbol not defined
.IFLOA driver	driver loaded
.IFNLOA driver	driver not loaded
.IF symbol rel expr	relation satisfied
.IFINS task	task installed
.IFNINS task	task not installed
.IFACT task	task active
.IFNACT task	task not active
.IFT [mask] symbol	numeric symbol has any masked bits set
.IFF [mask] symbol	numeric symbol has any masked bits clr

Note that:

- The "task"-oriented conditionals are available only in RSX and P/OS.
- The "[mask]"-ed forms of the conditionals are available only under RT.
- o The following relations are valid in a .IF directive:

Syntax	Satisfied if value of symbol is

= or EQ	equal to expression
<> or NE	not equal to expression
> or GT	greater than expression
< or LT	less than expression
>= or GE	greater than or equal to expression
<= or LE	less than or equal to expression

The expression must be of the same type as the symbol.

- o Tests can be connected using:
 - .OR - satisfied if either condition is met.

.IFx condition .OR .IFx condition

- .AND - satisfied if both conditions are met.

.IFx condition .AND .IFx condition

- Parentheses after the first .IFx group tests.

.IFx .AND (.IFx .OR .IFx)

Labels

- o Are used to identify locations as targets of .GOTO or .GOSUB directives.
- o Are formed in the same way as symbol names but do not conflict with them.
- o Must occur (at least) once in the same command file as all .GOTOS or .GOSUBS that refer to them, or you get an error.
- o Need not be unique - but you can get "strange" results if they're not.
- o Come in two flavors:

- Standard labels:

- * Defined by the syntax:

.label: (more stuff on same line)

- * Are found by scanning the command file forward from the .GOTO or .GOSUB to the end of the file and then (if the file is on disk) rewinding and scanning forward to the .GOTO or .GOSUB directive;

- * Substitution does not occur when scanning for labels.

- Direct-access labels:

- * Defined by the syntax

.label:

with nothing else on the same line;

- * Location is cached, and loaded directly when referenced by a .GOTO or .GOSUB directive;

- * If cache fills, earliest defined label is dropped (ie: it reverts to being a standard label);
- * Obviously, there are no direct-access labels if the command file is not on disk.

Transfer of Control

- o "Standard" GO TO:

- Syntax:

.GOTO label

- Unconditionally transfers control to the given label.

- The given label must occur in the same command procedure as the .GOTO that refers to it.

- o "Computed" GO TO:

- No explicit support for this.

- Can be implemented using symbol substitution and smart choice of labels, as in:

.ENABLE SUBSTITUTION

.SETN OPTION 0

.QUERY: .ASKS FILE Which file

.ASKN [0:2] OPTION Which option

.GOTO OPT'OPTION'

.OPT0: .EXIT

.OPT1: PIP 'FILE'/LI

.GOTO QUERY

.OPT2: PIP 'FILE'/SP

.GOTO QUERY

- o "Assigned" GO TO:

- No explicit support for this.

- Can be implemented using symbol substitution and smart choice of labels, as in:

.ENABLE SUBSTITUTION

.SETS ASSIGN "LABEL1"

.

.

.

.GOTO 'ASSIGN'

.

.

.

.LABEL1:

File I/O

Basic File Operations

- o Input directives:

- To open an existing sequential file for input:

`.OPENR filename`

- To read the next sequential record into a String Symbol:

`.READ symbol`

If the end of the file is encountered, Special Symbol `¡EOF¿` is set `¡TRUE¿`, and the symbol's value is untouched.

- Output directives:
 - To open a new sequential file for output:

`.OPEN filename`

- To open an existing sequential file to append records to it:

`.OPENA filename`

If the specified file does not exist, the effect is the same as `.OPEN`.

- To write a line of text to a sequential file:

`.DATA text`

- To copy lines from the command procedure to a sequential file:

`.ENABLE DATA`

causes all lines in the command procedure to be written to the output file, until a

`.DISABLE DATA`

is encountered. Substitution is performed (if enabled) before the output lines are written. Note that:

- * Labels in a `.ENABLE DATA` block are recognized during a label search. This is a restriction.
- * Some versions of the ICP don't recognize `.DISABLE DATA` unless it is left justified in the record.

- Miscellaneous file I/O directives:

- To close a file:

`.CLOSE`

- To close and delete a file (under RT only):

`.PURGE`

- Restrictions:

- There are no I/O directives for relative, indexed, or stream files. You can use `RMSDES`, `RMSCNV`, and `RMSIFL` to convert between these file organizations and sequential, if it is appropriate to your application.
- There is no way to read more than 132 bytes of any file record.

Miscellaneous Capabilities

- To operate on multiple files:

All file I/O directives will take an optional File Number after the directive, but before any arguments. For example:

```
.OPEN #1 KANGA.ROO
.DATA #2 This data goes to File 2.
```

The File Number is a hash mark (#) and a number from 0-3. #0 is the same as omitting the file number. You can, of course, generate the number by symbol substitution:

```
.DATA #'FILENO' This data is written
.DATA #'FILENO' to some file, but
.DATA #'FILENO' which one is not
.DATA #'FILENO' determined until the
.DATA #'FILENO' ICP is run.
```

- To determine file attributes (RSX only):

Special Symbol `¡FILATR¿` is loaded with the first 7 words of the FCS file descriptor block for the most recently `.OPENed` file (as a string of decimal numbers, separated by commas). This includes such useful information as how big the file is, what its largest record is, and more.

Terminal I/O

`.ASKx` Operation

- The `.ASKx` directives are used to prompt for and validate symbol values.
- The `.ASKx` directives always return either a valid value or some specific exception condition.
- Validation failure causes reprompting automatically.
- By default, entry of `^Z` causes the ICP to exit.
- Syntax:

```
.ASK [df:tm] Logical_symbol pmp
.ASKN [lo:hi:df:tm] Numeric_symbol pmp
.ASKS [lo:hi:df:tm] String_symbol pmp
```

Where:

- lo = lowest valid value (.ASKN) or lowest acceptable length (.ASKS).
 - hi = highest valid value (.ASKN) or highest acceptable length (.ASKS).
 - df = default value.
 - tm = timeout on question.
 - * Under RSX and P/OS, this can be disabled with the .DISABLE TIMEOUT directive.
 - * Under RT, this will not work unless you have a system clock, and issue the .ENABLE TIMEOUT directive.
 - pmp = prompt string.
- All .ASKx parameters are optional - except that “lo” and “hi” must be either both specified or both omitted.
 - Trailing colons in parameter block can be omitted. If all parameters are omitted, the square brackets can be, too.
 - If the default answer is taken, ;DEFAULT_i comes back TRUE.
 - If the timeout expires, ;TIMOUT_i comes back TRUE.
 - You can get uppercase conversion on a .ASKS by issuing the .DISABLE LOWERCASE directive.

.ASKx Exception Handling

You don't get exceptions returned unless you .ENABLE them. Exception conditions which can be trapped in this way are:

- .ENABLE ESCAPE
 - Causes the escape character to be a valid response for any .ASKx directive.
 - Special Symbol ;ESCAPE_i (and its synonym ;ALTMOD_i) come back TRUE if an escape character is entered.
 - This is generally used to break out of the normal logic sequence (eg - to print help text).
- .DISABLE CONTROL-Z (RSX, P/OS only)
 - Allows \hat{Z} to be trapped by your command procedure.
 - Special Symbol ;EOF_i comes back TRUE when a Control/Z has been entered.
 - This is generally used to break out of normal logic sequence (eg - SYSGEN), or with SET /SLAVE=TI: to create captive command procedures.

The manual specifies that the default answer is returned for exceptions. Not all versions of ICP support this.

Miscellaneous Input

You can also .OPENR a terminal. This is the only way to do I/O to a terminal other than TI: (RSX, P/OS) or TT: (RT).

Output

- By default:
 - Each .ASKx displays:


```
>* prompt [parameters]: (under RSX)
$ * prompt [parameters]: (under P/OS)
* prompt [parameters]: (under RT)
```

 - “External” comments display:


```
>; comment text (under RSX)
$ ; comment text (under P/OS)
; comment text (under RT)
```
 - CLI commands are displayed as:


```
>Command (under RSX)
$ Command (under P/OS)
. Command (under RT)
```

- You can disable the extra stuff by issuing

```
.DISABLE DISPLAY (Under RSX and P/OS)
.DISABLE PREFIX,SUFFIX (Under RT)
```

If you do this:

- Each .ASKx displays:


```
prompt
```
- “External” comments display:


```
comment text
```
- CLI commands are displayed as:


```
Command
```

 - You can get rid of “External” comments and CLI commands completely by:


```
.ENABLE QUIET
```

 - You can also (of course) .OPEN a terminal for output.

Modules

Internal

- Modules internal to the current command file can be created by using the .GOSUB - .RETURN construction.
- Module entry syntax:


```
.GOSUB label arguments
```

 - Argument passing:

- The arguments are available in the reserved String Symbol `COMMAN`.
- This contains a literal copy of anything on the `.GOSUB` command line after the label.

- Module exit syntax:

```
.RETURN
```

returns to the first line after the `.GOSUB` directive.

For example:

```
.ENABLE SUBSTITUTION
.GOSUB STORE Arthur Dent
  .GOSUB STORE 6*9=42
  .EXIT
.STORE:
; The argument is "'COMMAN'".
.RETURN
```

This displays (under `RSX`):

```
>; The argument is "Arthur Dent".
>; The argument is "6*9=42".
>@ <EOF>
```

External

- Other command files can be called just as though from the CLI prompt.
 - The module entry syntax is

@file arguments

- The arguments are loaded into reserved String Symbols as follows:

- * `COMMAN` - Contains the entire invoking command line, uppercased and with blank compression.

- * `P0-P9` - Are loaded as though by:

```
.PARSE COMMAN " " P0 P1 P2 P3 ....
```

- You can exit the module in two ways:

- * Return to the calling module. The syntax is

```
.EXIT status
```

Where "status" is a number to be returned to the caller in Special Symbol `¡EXSTAT¡` (the default being 1). If executed from the top level, the ICP is terminated and the status is returned to the parent task (if any). The end of the command procedure is equivalent to `.EXIT`

- * Terminate the ICP. The syntax is

```
.STOP
```

- You can pass results back to the caller in global symbols, or in Special String Symbol `¡EXSTRI¡` by:

```
.SETS <EXSTRI> results
```

- You can also chain between command files:

```
.CHAIN file
```

The only parameter passing is by global symbols, or (under `RSX` and `P/OS` only) by using

```
.CHAIN file/LO
```

which causes all local symbols to be preserved in the new command file.

External Environment

What Kind of System Are You On?

- Your operating system type is encoded in Special Numeric Symbol `¡SYSTEM¡` (in octal), as follows:

```
0 for RSX-11D.
1 for RSX-11M.
2 for RSX-11S (nice trick!).
3 for IAS.
4 for RSTS (for expansion?).
5 for AME, or VAX-11 RSX.
6 for RSX-11M+.
7 for RT-11 SJ.
10 for RT-11 FB (or RTEM-11).
11 for P/OS.
```

- Under `RSX`, a string describing your operating system type is available in Special String Symbol `¡SYTYP¡`. This is a new feature, and may or may not be in `P/OS` and `RT`.

- Under `RSX`, a string describing your operating system version is available in Special String Symbol `¡VERSN¡`. This is a new feature, and may or may not be in `P/OS` and `RT`.

- Your system name is available in Special String Symbol

```
<NETNOD> (for RSX and P/OS)
```

```
<MONNAM> (for RT)
```

By default, this is the name given at `SYSGEN`. However (at least under `RSX`) if you are running `DECnet`, it will contain your `DECnet` Node Name.

The Device Configuration

- o Determining the status of a driver:
 - .IFLOA driver is satisfied if the driver is loaded.
 - .IFNLOA driver is satisfied if the driver is not loaded.
- o Determining the status of a device:

.TESTDEVICE name

(where "name" is a physical or logical device name) returns, in Special String Symbol ;EXSTRI_i , the string

"phys, n1, n2, n3, n4, flags"

where:

- Phys is the physical name of the device.
- n1 through n4 represent:
 - * Under RSX and P/OS, the contents in octal of the words U.CW1 through U.CW4. This is the same information that is returned by GETLUN in the third through sixth words of the buffer.
 - * Under RT, n1 is the device size as a decimal number, with trailing dot. n2 through n4 are always zero.
- Flags are three letter device status indications, separated by commas. One from each group will be returned:

* Driver status:

LOD, driver is loaded.

UNL, driver is not loaded.

* Device status:

ONL, device is online.

OFL, device is offline.

UNK, status is unknown (only by RT).

* Mount status:

MTD, device is mounted.

NMT, device is not mounted.

* Mount type (RSX and P/OS only):

FOR, device is mounted foreign.

NFO, device is not mounted foreign.

* "Publicity" (RSX and P/OS only):

PUB, device is set public.

NPU, device is not set public.

* Allocation (RSX and P/OS only):

NAL, not allocated.

ALU, allocated to this terminal.

ALO, allocated to another terminal.

Be warned that some versions of the ICP get confused whether to return ALO or ALU.

* Attachment (RSX and P/OS only):

NAT, not attached.

ATU, attached by this copy of the ICP.

ATT, attached by another task.

If the device is not in the system, ;EXSTRI_i returns "NSD,".

Other Things About the System

- o Under RT only, you can find out what volume is on a given device by issuing

.VOL symbol device

which loads the volume label into the string symbol.

- o The current date and time are returned in the Special String Symbols

- ;DATE_i (blank under RT if there is none);

- ;TIME_i .

- o You can check for the existence of a file using

.TESTFILE file

- The status of the search is returned in Special Numeric Symbol ;FILERR_i (1 = success).

- The file name is returned in Special String Symbol ;FILSPC_i . Some versions of the ICP have trouble, due to a bug in RSX SYSLIB module EXPFN. This bug is fixed under the current release.

- o Under RSX and P/OS, the state of a task can be checked:

- .IFINS task is satisfied if the task is installed.

- .IFNINS task is satisfied if the task is not installed.

- .IFACT task is satisfied if the task is active.

- .IFNACT task is satisfied if the task is not active.

- o Under RSX and P/OS, you can check on a partition or common block using

.TESTPARTITION partition name

this loads Special String Symbol ;EXSTRI_i with

"name, base, size, type, "

where:

- name = The name of the partition;

- base = The partition base address in 64-byte blocks (octal);

- size = The partition size in 64-byte blocks (octal);

- type = one of the following:

- * SYS (system controlled);

- * USR (user-controlled);

- * NSP (no such partition).

The Context of the ICP File

- The device type of your terminal is returned in Special Numeric Symbol ;TTYPE_i, under RSX and P/OS. Prior to the current release, this was initialized when the ICP was invoked, and not affected thereafter by SET /DEV.
- The physical device name of your terminal can be obtained under RSX and P/OS by using

```
.TESTDEVICE TI:
```

- Your current default device can be obtained by issuing

```
.TESTDEVICE SY: ! RSX and P/OS
```

```
.TESTDEVICE DK: ! RT
```

- Your current default directory is returned in Special String Symbol ;DIRECT_i under RSX with named directory support and P/OS. If you are in NON-AMED mode, ;DIRECT_i = "[]".
- Your protection UIC is returned in Special String Symbol ;UIC_i under RSX and P/OS. If you don't have named directory support, ;UIC_i contains the default UIC instead.
- Your current CLI is returned in Special String Symbol ;CLI_i under RSX. Be warned that if the CLI override bit is set (which normally occurs only in SYSLOGIN.CMD), this CLI is not necessarily the one your CLI commands are passed to.
- You can translate logical names under RSX with logical name support and P/OS, using

```
.TRANSLATE logical
```

The translation is returned in Special String Symbol ;EXSTRI_i. ;EXSTRI_i will be empty if translation failed.

- The name of the command file which is currently executing is loaded into Special String Symbol ;FIL-SPC_i on entry to the command file. If you want it, you must preserve it in another string symbol of your choosing.

Executing under Multiple CLIs

RSX users may be faced with the task of writing a command file that will execute under more than one Command Line Interpreter (CLI). There are at least the following ways to approach this:

- Group CLI commands together in blocks, and execute the appropriate block. Example:

```
.ENABLE SUBSTITUTION
.GOTO '<CLI>'
.MCR: PIP *.SAV;*/DE/NM
      PIP *.SAV;*/RE=*.DAT;*
```

```
.EXIT
.DCL: DELETE *.SAV;*
      RENAME *.DAT;*.SAV;*
      .EXIT
```

- Force the CLI as desired on entry to the procedure. Example:

```
.IF <CLI> <> "MCR" MCR SET /MCR=TI:
```

It would probably be polite to set it back on exit.

- Force individual commands to the desired CLI. Example:

```
.ENABLE SUBSTITUTION
.SETS MCR ""
.IF <CLI> <> "MCR" .SETS MCR "MCR "
'MCR' PIP *.SAV;*/DE/NM
'MCR' PIP *.SAV;*/RE=*.DAT;*
.EXIT
```

- Modify the ICP to set the CLI override bit on entry, and clear it on exit.

Synchronizing With Tasks

- Under RSX:
 - The ICP will wait for the following types of tasks to complete before proceeding:
 - * Tasks run with the install/run/remove version of the "RUN" command.
 - * Tasks run as CLI commands.
 - The ICP will not wait for the following types of tasks to complete before proceeding:
 - * RUN or CLI commands with the .XQT directive prefixed.
 - * RUN of an installed task
 - You can resynchronize with a task by issuing:

```
.WAIT task
```

- Under P/OS
 - The ICP will wait for the following types of tasks to complete before proceeding:
 - * Tasks run with the "RUN" command.
 - * Tasks run as CLI commands.
 - The ICP will not wait for the following types of tasks to complete before proceeding:
 - * RUN or CLI commands with the SPAWN command prefixed,
 - The ICP has no way to resynchronize with a SPAWNed task, but it would be possible for a user-written task run as a CLI command to provide this functionality.

- o Under RT
 - The ICP will wait for the following types of tasks to complete before proceeding:
 - * Tasks run with the "RUN" command.
 - * Tasks run as CLI commands.
 - The ICP will not wait for the following types of tasks to complete before proceeding:
 - * Tasks run using the FRUN or SRUN command.
 - The ICP has no way to resynchronize with a task initiated with FRUN or SRUN, but it would be possible to write a task to provide this functionality.

Interfacing with your own code

Passing a command line to your own code

- o Under all systems covered, you must run your task as a CLI command for this to work.
- o Under RSX and P/OS:
 - You can run your code using

```
>INS task/NAME=...nam
>NAM command
```

or using

```
>RUN task/CMD="NAM command"
```

- Your code will look like this:

```
LOGICAL*1 CMDBUF(80)
INTEGER*2 CMDLEN
CALL GETMCR(CMDBUF,CMDLEN)
```

Note that:

- * If CMDLEN comes back negative, no CLI command is available.
 - * The command name is also passed. You must remove it yourself. This may leave you with no command.
 - * You should not process the command buffer beyond character CMDLEN. Location CMDLEN+1 contains the terminating character (if any), and the rest of it is undefined.
 - * Under RSX-11M+, if the command ends in "-", it means that it was longer than 80 characters. You must CALL GETMCR again to get more.
- o Under RT:

- You must run your task using the so-called "CCL" facility:

```
. task command
```

- Your code will look like this:

```
LOGICAL*1 CMDBUF(80)
LOGICAL*1 CMDPMP(5)
DATA CMDPMP /'C','m','d','>',"200/
CALL GTLIN(CMDBUF,CMDPMP)
```

Note that:

- * The command line is terminated with an ASCII null character.
- * I know of no way to prevent DCL from munging around with the command line.

Passing Status Back to the ICP

You can pass a limited amount of data from your task directly back to the ICP. This information is returned to the ICP in Special Numeric Symbol ;EXSTAT_i.

- o Under RSX and P/OS, you can pass back a word. Your code looks like:

```
INTEGER*2 EXSTAT
EXSTAT = 1
CALL EXIT(EXSTAT)
```

Note that:

- You can (if you like) CALL EXST instead of EXIT.
- Some values of exit status have (by convention) a special meaning to the system:

```
1 = Success;
0 = Warning;
2 = Error;
4 = Fatal.
```

- o Under RT, you can pass back a byte. Your code looks like:

```
INTEGER*2 EXSTAT
EXSTAT = 1 .OR. IPEEK("53)
CALL IPOKE("53,EXSTAT)
CALL EXIT
```

The above code seems to be what the manuals call for under RT-11. However, under the only system available to me for testing (RTEM-11 V2.0, running RT-11 V5.1), the only thing that works right is that the ICP is aborted if EXSTAT is 20 (octal). I am informed that this works correctly under RT-11 V5.4.

Other Ways to Communicate

- You can use Extended Logical Names to exchange moderate amounts of information under RSX-11M+ and P/OS. The ICP will:

- Create logicals using the MCR DLG command.
- Read logicals created by the program using the .TRANSLATE directive.

Your code will:

- Create logicals using CALL CLON or CALL CLOG.
- Read logicals created by the ICP using CALL TLON or CALL TLOG.

Under RSX-11M+, See the release notes on these calls. There are bugs in the documentation.

- You can use disk files to exchange large volumes of information.

Debugging

- You can trace the ICP directives as they are executed by using

```
.ENABLE TRACE
```

The directive trace is turned on by default if you invoked the ICP using:

```
@file/TR ! RSX and P/OS
@file/T ! RT
```

This is the only way to do a trace in earlier versions of the ICP.

- You can suppress the execution of CLI commands by using

```
.DISABLE MCR ! All systems
.DISABLE DCL ! RT
```

CLI commands are displayed anyway, with a comment marker (!) in front. If you invoke the command procedure by:

```
@file/-MCR ! RSX and P/OS
@file/-CLI ! RSX and P/OS
@file/N ! RT
```

CLI commands are suppressed by default. This is the only way to suppress CLI commands in earlier versions of the ICP.

References

- [1] Watson, Allen A., *Indirect Command Files for New RSX Users*, in RSX/IAS SIG Symposium Handout, Spring 1983 DECUS US Symposium.
- [2] Pro/Tool Kit Command Language and Utilities Manual (The primary reference for the ICP under P/OS).
- [3] P/OS System Reference Manual (Documents calls to EXST and GETMCR).
- [4] RSX LB: [1, 2] ICP .HLP (On-line help file for ICP. Contains some information that is not in the manual).
- [5] RSX-11 Executive Reference Manual (Documents calls to EXST, GETMCR, CLON, TLON, AND GETLUN).
- [6] RSX-11 I/O Operations Guide (Documentation for the contents of ;FILATR₀).
- [7] RSX-11M/M-PLUS Indirect Command Processor Manual (The primary reference for the ICP under RSX).
- [8] RSX-11M/M-PLUS RMS-11 Utilities (The reference for RMSDES, RMSCNV, and RMSIFL).
- [9] RT-11 Programmer's Reference Manual (Documents calls to GTLIN, IPEEKB, and IPOKEB).
- [10] RT-11 Software Support Manual (Documents layout of System Communication Area, which contains the error byte).
- [11] RT-11 System Users Guide (The primary reference for the ICP under RT).

Programming in the RSX Indirect Command Language

Thomas R. Wyant, III
E. I. DuPont de Nemours
Richmond, Virginia

Arnold S. De Larisch
Florida Atlantic University
Boca Raton, Florida

Abstract

This paper presents a compendium of advanced techniques for using the RSX Indirect Command Processor (ICP). These include the use of command procedure libraries, arrays and other structured data types, binary file I/O, screen handling both with and without FMS, error control, command line processing, and multiple precision arithmetic. This paper is aimed at an audience that is familiar with the RSX ICP, though "Introduction to the RSX, P/OS, and RT Indirect Command File Processor" (RX001) should give sufficient background.

Caveats

The current releases of RSX are assumed. The techniques presented here may or may not work under previous versions of RSX, or under IAS, P/OS, or RT.

Since the things covered in this paper are "off the beaten track", they are more likely than usual to be affected by bugs or other differences between releases of the ICP. The examples cited all work under RSX-11M+ V3.0 C. I will mention bugs where I am aware of their existence.

I am not the final authority on the ICP in all its multifarious versions. Errors in research and transcription do occur. I apologize in advance for these, but assume no responsibility for their consequences.

The examples are all from working command procedures, which will be submitted to the Spring, 1987 RSX SIG tape. Some reformatting has been necessary to make the ICP code fit on the page properly. I have tried to preserve the functionality, but I have no way to test the example code out of context.

Acknowledgement

Allen A. Watson's paper, "Nifty Things to Do with RSX Indirect Command Files", presented at the Spring, 1983 DECUS US Symposium, was both an inspiration and a reference for this paper.

Command Procedure Libraries

It can be desirable to break a very large command procedure into separate modules. These will be easier to keep track of if they are kept in a Command Procedure Library. Examples include:

```
LB: [1, 2] INDSYS.CLB
LB: [200, 200] SYSGEN.CLB
LB: [137, 10] NETGEN.CLB
```

A Command Procedure Library is just a Universal Library (as documented in the LBR manual), created with the command `>LBR library.CLB/CR:::UNI:CMD`

Once the command procedure library is created, modules can be inserted and removed just as they are for a macro or object library.

Modules in a Command Procedure Library can be executed without extracting them from the library with the command `>@library/LB:module`

Where:

library is the name of your command file library, with default file type `.CLB`. If you omit this, the current library is assumed if the command is issued from inside a library, otherwise `module.CMD` is used. The latter functionality is useful when debugging.

module is the name of the module in the library to be executed. If you omit this, the module `.MAIN.` is executed.

If you do not specify the `/LB` switch when invoking a Command Procedure, the ICP checks the attributes of the file to determine whether it is a Command Procedure Library. So, you can call your Command Procedure Library `library.CMD`, and execute module `.MAIN.` with the command `>@library`.

You can store things other than Command Procedures in a Command Procedure Library. For instance, you could

store the sources for a software package there, and have the `.MAIN` module extract the sources and build the package.

Under all releases of the ICP that support command procedure libraries, you can (officially) use the `.TESTFILE` directive to test for the presence of a module in a Command Procedure Library or other universal library with the command `.TESTFILE library/LB:module`.

This returns the value of 1 in Special Numeric Symbol `<FILERR>` if the library file exists and the module is in it. However, some of the ICP releases involved (including at least M+ V2.1C) have a bug: `<FILERR>` will be 1 if the library file exists, regardless of whether the desired module is in it.

Structured Data Types

Although the ICP has explicit support only for simple variables, there are a number of ways in which arrays and other structured data types may be built. All of these are based on building symbol names at execution time, using substitution on other symbols. An assortment of techniques is presented here.

Using a String Symbol as an Array

Small arrays can be stored in a String Symbol and extracted by forming a substring. In order to do this,

- All elements in the array must be the same size.
- The total size of all elements in the array must not exceed the maximum size of a String Symbol (132 bytes).
- The location of each element in the string must be manually calculated from the `INDEX` of the desired element and the `SIZE` of the elements, as follows:

```
.SETN START (INDEX-1)*SIZE+1
.SETN END START+SIZE-1
```

This works best with arrays where the element is one byte long, as the index can be used directly as both the start and the end of the substring.

Example

`PRN.CMD` is a utility designed to print a file on an LA-series printer connected to the printer port of a VT100- or VT200-series terminal.

The horizontal pitch on an LA50 is set by an escape sequence of the form `"esc;inumberw"`. The array `.SETS S$PCHR "5555568800224444"` translates any desired integer pitch from 1 through 16 to the nearest equivalent escape sequence argument.

When the desired horizontal pitch is determined (and stored in Numeric Symbol `N$HPIT`), the escape sequence required to set the printer to that horizontal pitch is built using the String Expression `ESCAPE+"["+S$PCHR[N$HPIT:N$HPIT]+"]w"`

Using a String Symbol as an Attribute List

As a variant on arrays, a String Symbol can be used to store a list of attributes to be associated with the name of the string symbol, or a portion thereof. These attributes can be tested for in place using the `.TEST` directive, or extracted using the `.PARSE` directive. In order to set up an attribute list:

- The attributes are listed in the String Symbol's value, punctuated by a unique character.
- Attributes may be subdivided by using another unique character, to any level desired.

If the primary use of the attribute list is to search using `.TEST`, it may be helpful to start the list off with a punctuating character. However, there is a special case if the attributes are all one byte long, and the only purpose of the list is to use it with the `.TEST` directive: the punctuation may be omitted.

Example

`PRN.CMD` is a utility designed to print a file on an LA-series printer connected to the printer port of a VT100- or VT200-series terminal.

The command interface to `PRN` is screen driven. The arrow keys and (on the LK201 keyboard) the "Previous Screen" and "Next Screen" keys have different functions, but are implemented in basically the same way: a counter is incremented (or decremented) and clamped to a desired range. The `¡PF1¿` key can also be used to amplify the action of an arrow key; this is done by adding (or subtracting) a number greater than one, which is dependent on what arrow key was used.

All of this is accomplished by setting up String Symbols named after the six keys of interest. The escape sequence parser delivers control for all six keys to the same piece of code, which `.PARSES` the aforementioned String Symbols and plugs the derived attributes into its subsequent operations.

The following attributes turn out to be needed:

- The name of the Numeric Symbol updated by the key;
- The amplification factor to apply if `¡PF1¿` is in effect;
- The sign of the operation on the Numeric Symbol modified by this key (+ or -);
- The limit beyond which the Numeric Symbol may not go;
- The type of test to make against this limit (eg: `<`, `>`, ...).

The String Symbols defining the keys are assembled from the concatenation of the appropriate attribute values, separated by commas:

```
.SETS ARROWA "N$FLD,8,-,0.,<"
.SETS ARROWB "N$FLD,8,+,N$FMAX,>"
.SETS ARROWC "N$CVAL,10,+,N$CMAX,>"
.SETS ARROWD "N$CVAL,10,-,N$CMIN,<"
.SETS ARROW5 "N$SCR,8,-,0.,<"
.SETS ARROW6 "N$SCR,8,+,N$SMAX,>"
```

These define (in order) the up, down, right, and left arrows, and the Previous and Next Screen keys.

The following code is then used to execute all six keys (recognition of the keys is discussed later under "Screen Handling Without FMS"):

```
.SETS KEY ARROW'CHAR'
.PARSE KEY ", " AXS FAC SGN LIM TST
.IFT GOLD .SETN ESCA0 ESCA0*'FAC'
.SETN 'AXS' 'AXS''SGN'ESCA0
.IF 'AXS' 'TST' 'LIM' .SETN 'AXS'
'LIM'
```

This code is entered with:

- CHAR containing the last character of the escape sequence that defines the key struck; this is forced to "5" or "6" for the Previous and Next Screen keys, which are named by a different convention than the arrow keys;
- ESCA0 containing the numeric value of the first argument in the escape sequence; this is forced to 1 for the Previous and Next Screen keys;
- GOLD set to \downarrow TRUE \downarrow if the \downarrow PF1 \downarrow key was the last key struck; otherwise it is false.

If (for example) the down arrow key is struck, the code is entered with CHAR = "B", ESCA0 = 1, and GOLD = <FALSE>. After substitution, this gives:

```
.SETS KEY ARROWB
.PARSE KEY ", " AXS FAC SGN LIM TST
.IFT GOLD .SETN ESCA0 ESCA0*8.
.SETN N$FLD N$FLD+ESCA0
.IF N$FLD > N$FMAX .SETN N$FLD N$FMAX
```

This has the effect of moving the cursor forward 1 field on the screen (or eight fields if the \downarrow PF1 \downarrow key is in effect).

Using Groups of Symbols as an Array

An array of arbitrary size can be constructed by using a group of symbols with similar names. The names of symbols in this group consist of a constant part (which can be thought of as the array name) and a variable part (which can be thought of as the array subscript). The symbols in the array need not be the same type, and only those elements that are actually used need to be defined. An array can have more than one dimension, provided the naming convention for the elements is chosen appropriately.

Array elements are referred to by using symbol substitution to construct the name of the element's symbol out of the array name and the symbol(s) used to index the array. Arrays can be indexed by symbols of any type.

The formation of arrays in this manner is subject to the following restrictions:

- The naming convention used to map array elements onto symbol names must never result in a symbol name more than six characters long.

- The naming convention must give rise to a unique symbol name for each element in an array. This is usually a problem only for multi-dimensional arrays, where the use of "%Rn" format control can be helpful.

- Symbol substitution can not be done on an arbitrary array element. The value of that array element must be assigned to another, "constant-named" symbol, which is used instead in the desired substitution.

Example

UPS.COMD is a command procedure to send a number of files (up to 15) to another person or persons over DECmail-11. A screen is displayed, with spaces for the user to enter the addressees, the subject matter, and the names of the files to send. All the files desired are built into a single temporary file, which is sent in batch mode.

The file names are stored in an array of symbols, named S\$FNnn, where "nn" is a two digit number from 00 through 14. All the file name fields on the screen are processed by the same code, which subtracts two from the field number to get the "nn" used to construct the name of the array element (subroutine ASKE is discussed under "Screen Handling Without FMS"). The file processing code follows:

```
.; Prompt for the name of the next
.; file.

.SETN N$ROW N$FLD+6.
.SETN N$FILE N$FLD-2.
.SETS S$FILE S$FL'N$FILE%DR2Z'
.GOSUB ASKE 40;'N$ROW%D';10\S$FILE'
.IFT <EOF> .GOTO EXIT

.; Convert the file name to
.; uppercase.

.DISABLE LOWERCASE
.SETS TEXT ''TEXT%C''
.ENABLE LOWERCASE

.; If the file name is null, go
.; process the associated escape
.; sequence, if any.

.IF TEXT = "" .GOTO ESCPSI

.; Check for existence of the file,
.; and get the full file name.

.SETS S$ERR "File 'TEXT' not found."
.TESTFILE 'TEXT'
.IF <FILERR> <> 1 .GOTO INPERR

.; Store the file name in its slot in
.; the array.
```

```
.SETS S$FL'N$FILE&DR2Z' <FILSPC>

.; Redisplay the fully qualified file
.; name.

.SETS TEXT "40;'N$ROW%D';10"
.GOSUB PLOTF 'TEXT'\ '<FILSPC>'

.; Go handle the associated escape
.; sequence, if any.

.SETS S$ERR ""
.GOTO ESCPSI
```

Content-addressable Memory

This is really just an extension of the concept of using a group of symbol names as an array. There is no logical reason why you can't use a String Symbol as an array subscript, provided the contents of the symbol give rise to a valid symbol name. The same consideration applies to allowing the array name to shrink to zero bytes. What you have left is a system where the entire symbol table is an array, and the symbol names are chosen to describe the information stored in the symbol. You should be aware that:

- The String Symbol that contains the information to be looked up had better be validated first to be sure its contents represent a legal symbol name.
- You must check for existence of the symbol before you use it, as a random symbol name is probably *not* defined in the symbol table.
- Since it is probably not desirable to literally use the whole symbol table, a subset can be selected (eg: all symbols with alphanumeric names), and symbols used otherwise in the command procedure can be selected to fall outside this subset (eg: names with embedded dollar signs).

Example

CRASHDUMP.CMD is a crash dump analyzer suitable for sites where the procedure for generating the crash dump analysis varies from system to system. This procedure accepts the name of the crashed system as input, and runs the appropriate analysis.

This command file contains a table that lists, for each system, some help text, the name of the crash dump analyzer to use, the memory size, the crash device, the starting block on the crash device, and the name of the executive symbol table. This information is stored in a symbol named after the system, thus:

```
.SETS FENNY "PDP-11/84;;256;DU;;"
.SETS MARVIN "PDP-11/03;CDA42;28;DY;;"
.SETS ZAPHOD "PDP-11/74;;1024;DR;;"
```

All other symbols used in this command procedure contain embedded dollar signs, so any alphanumeric symbol name

is likely to be of interest. The reserved symbols **COMMAN** and **P0** through **P9** can be excluded by explicitly testing for them.

When this command file is executed, it can simply prompt for the system name, insure that the entry is a valid symbol name (and is in fact the name of an existing symbol), and extract the information needed from symbol:

```
.DISABLE LOWERCASE
.ASKS [0:6] S$$SYS What system name
.ENABLE LOWERCASE
.IFF <ALPHAN> .GOTO SYSERR
.IFNDF 'S$$SYS' .GOTO SYSERR
.PARSE 'S$$SYS' ";" .....
```

Searching a (Very) Sparse Array

One of the disadvantages of using sparsely populated arrays is the inefficiency of weeding out nonexistent elements when iterating over the entire array. A better alternative may be to iterate over the entire symbol table, searching for elements of the array.

DEC has provided Special String Symbol **<NXTSYM>**, which can be used to search the symbol table. Each time this symbol is referred to, it returns the name of the next symbol in the Symbol Table. The search is initialized with the command `.SETS <NXTSYM> ""`. The end if the symbol table is indicated when **<NXTSYM>** returns a null string.

If you intend to use this technique, you should make note of the following points:

- You must refer to **<NXTSYM>** only once in each iteration, or you will skip symbols. A useful way to do this is using the command `.SETS SYMNAM <NXTSYM>`
- The documentation of `¡NXTSYM¡` contains warnings about its availability for general use. My experience is that you might have to fiddle a bit to get it to work. See `LB: [1, 2] INDSYS/LB: INDDMP` for an example. I recommend against defining any new symbols inside the iteration loop.

Example

CRASHDUMP.CMD is a crash dump analyzer suitable for sites where the procedure for generating the crash dump analysis varies from system to system. This procedure accepts the name of the crashed system as input, and runs the appropriate analysis.

If the user input (obtained in the previous example) does not represent a valid system name, the symbol table is scanned, and a list of all valid system names is produced. This is accomplished by the following code:

```
.SYSERR:.;
;
; System "'S$$SYS'" is invalid.
.SYSHLP:.;
;
```

```

; Valid system names are:
.SETS <NXTSYM> ""

.; loop through Symbol Table:
.SYSHLL:

.; Get next Symbol name.

.SETS $$$SYS <NXTSYM>

.; If there is none, done.

.IF $$$SYS = "" .GOTO SYSASK

.; If not alphanumeric, skip.

.TEST $$$SYS
.IFF <ALPHAN> .GOTO SYSHLL

.; If not a String Symbol, skip

.TEST '$$$SYS'
.IF <SYMTYP> <> 4 .GOTO SYSHLL

.; If it is COMMAN or P0-P9, skip

.TEST $$$NOGO ", '$$$SYS', "
.IF <STRLEN> > 0 .GOTO SYSHLL

.; If it is a legal system name,
.; pick it apart and display
.; the identifying text;

.PARSE '$$$SYS' ";" $$HELP $$JUNK
;          '$$$SYS%L6' - '$$HELP'

.; Go get the next Symbol:

.GOTO SYSHLL

```

Binary File I/O

I/O on files containing binary data can be done with the ICP, by converting the bytes in the file record to numeric values, and then assembling the byte values in ways appropriate to the field in which they occur. The ICP can not process records more than 132 bytes long, but otherwise any sequential file can be read. Files opened for output will have variable length records with "list" carriage control. RMSDES can be used to create sequential files with other attributes, which can then be opened by the ICP for appending data.

The binary data is interpreted by using symbol substitution with numeric values. These bytes are then assembled in an appropriate manner to yield the data in the record.

Example

SYMDMP.COMD is a command file that reads a .OBJ or .STB file, and displays the types and values of the symbols it finds there.

The decoding of a binary byte is relatively straightforward:

```

.SETS $$B0 $$REC[5:5]
.SETN O$FLG '$$B0%V' &377

```

This extracts byte 5 of the record and stores its value in the Numeric Symbol O\$FLG.

Binary words are processed by extracting two consecutive binary bytes, and combining them in the correct order:

```

.SETS $$B0 $$REC[1:1]
.SETS $$B1 $$REC[2:2]
.SETN O$B0 '$$B0%V' &377
.SETN O$B1 '$$B1%V' &377
.SETN O$W O$B1*400+O$B0

```

This extracts the binary word starting at byte 1 of the record, and stores it in Numeric Symbol O\$W.

RAD-50 words are processed in the same manner as binary words, and then converted to ASCII using %X format control:

```

.SETS $$B0 $$REC[1:1]
.SETS $$B1 $$REC[2:2]
.SETN O$B0 '$$B0%V' &377
.SETN O$B1 '$$B1%V' &377
.SETN O$W O$B1*400+O$B0
.SETS $$$SYM "'O$W%X' "
.SETS $$B0 $$REC[3:3]
.SETS $$B1 $$REC[4:4]
.SETN O$B0 '$$B0%V' &377
.SETN O$B1 '$$B1%V' &377
.SETN O$W O$B1*400+O$B0
.SETS $$$SYM "'$$SYM' 'O$W%X' "

```

This extracts the six RAD-50 characters stored in bytes one through four of the record, and converts them to ASCII in String Symbol \$\$\$SYM.

Screen Handling

Screen Handling With FMS

The RSX-11M+ ICP comes with an interface to FMS-11. You can use this interface to generate a form-driven application. Demonstration of the FMS capability of the ICP is provided by: @LB: [1, 2] INDSYS.CLB/LB: FMSDEM. In order to create your own FMS-11 driven command procedure:

- Forms must be designed and inserted in a form library, just as for any FMS application. This implies the need for an FMS license.
- The .FORM directive is used to display forms and gather input.

- You can use `.IFENABLED FMS` to determine if FMS support is available.
- Special Numeric Symbol `<FILER2>` will contain the status code for the previous FMS operation. You can also (if necessary) use `.IFDF <FILER2>` to determine whether `.IFENABLED FMS` will produce a syntax error.

Screen Handling Without FMS

If you don't own RSX-11M+ and FMS-11, you can write your own screen handler for the ICP. This is not a trivial undertaking, and several points must be observed to get your screen handler to work:

- The ICP will have to do all input through the `.ASKS` directive. The prompt sequence of the `.ASKS` directive is used to position the cursor for input.
- The ICP must be conditioned to accept escape sequences by issuing the `.ENABLE ESCAPE-SEQUENCE` directive.
- The terminal driver must be conditioned to recognize escape sequences and pass them to the ICP by issuing the MCR command `SET /ESCSEQ=TI:` or its DCL equivalent.
- You must issue the `.DISABLE DISPLAY ICP` directive to prevent unwanted characters from being displayed on the screen by the `.ASKS` directive.
- The ICP must parse the escape sequence off the end of the `.ASKS` input string, and interpret it.

In addition to the above, there are several points which are not absolutely essential, but which are highly recommended to improve the performance and maintainability of your screen handler:

- You should issue the `.DISABLE DETACH ICP` directive, so that input from additional keystrokes will not be lost if you type faster than the ICP can process your input. Note that if you do this, programs run by the ICP may not have access to your terminal.
- Constant escape sequences (eg: home cursor, clear screen, setup sequences) should be assigned to appropriately named String Symbols on initialization. This enhances portability, and makes it easier to handle multiple terminal types.
- Variable escape sequences (eg: cursor positioning) should be generated in `.GOSUB` modules, for the same reasons.
- Literal escape sequences should not be embedded in the command file. If you violate this rule, you may not be able to `TYPE` the procedure on your terminal.
- Input should also be done in a `.GOSUB` module, so that the escape sequence can be easily parsed off the rest of the input.

- Avoid leaving the cursor on the last line of the screen. If you cannot avoid this, repaint the screen after the inevitable scroll-up.

If you want to get really sophisticated, you can `SET /NOECHO=TI:` and have the ICP take care of displaying the characters on the screen.

Example

`PRN.COMD` is a utility designed to print a file on an LA-series printer connected to the printer port of a VT100- or VT200-series terminal.

Both the ICP and the terminal driver must be initialized to handle the escape sequences involved with screen input:

```
.ENABLE SUBSTITUTION
.DISABLE DISPLAY
.DISABLE DETACH
.ENABLE ESCAPE-SEQUENCE

'MCR' SET /LOWER=TI:
'MCR' SET /ESCSEQ=TI:
'MCR' SET /BUF=TI:132.
```

Next, a selection of control characters suitable for ASCII terminals is defined:

```
.SETN NJUNK 16      ! Shift out
.SETS SO "'NJUNK&V'"
.SETN NJUNK 17      ! Shift in.
.SETS SI "'NJUNK&V'"
.SETN NJUNK 33      ! Escape
.SETS ESCAPE "'NJUNK&V'"
.SETN NJUNK 217     ! Single shift 3
.SETS SS3 "'NJUNK&V'"
.SETN NJUNK 233     ! Ctrl Seq Init
.SETS CSI "'NJUNK&V'"
```

After symbols have been defined for the individual control characters, control sequences to perform specific functions can be built. The following are suitable for ANSI terminals:

```
.SETS HOME ESCAPE+"[H" !Home cursr
.SETS CLEAR ESCAPE+"[J" !Clr screen
.SETS CLRLIN ESCAPE+"[K"!Clear line
.SETS BOLD ESCAPE+"[1m" !Bold video
.SETS REV ESCAPE+"[7m" !Revers vid
.SETS NML ESCAPE+"[m" !Normal vid
.SETS BOTTOM ESCAPE+"[24;1H"
```

Last, a control sequence is defined to initialize the terminal to the desired state. The following initializes a DEC VT100 or VT200 series terminal by homing the cursor and clearing the screen, loading the normal ASCII character set into `G0` and the graphics character set into `G1`, and selecting `G0`:

```
.SETS INIT HOME+CLEAR+ESCAPE+"(B"
.SETS INIT INIT+ESCAPE+" )0"+SI
```

Now that the constant control sequences are taken care of, we need a subroutine to prompt for the current field. The example given below is entered by the command `.GOSUB ASKE size;line;column/text`.

This positions the cursor at the given line and column, and displays the given text in a reverse video field of width 'size'. The `.ASKS` directive is used to get the response. The text part of the response is returned in String Symbol `TEXT`, and the escape sequence in String Symbol `ESCSEQ`. If `TEXT` is null, it is loaded with the input text string. Finally, the user's input is redisplayed (in upper case) in the field. The code to do all this is:

```
.ASKE:

.; Separate the command into its
.; components.

.PARSE COMMAN ";" FLDSIZ COMMAN
.PARSE COMMAN "\" COMMAN FLDTXT

.; Call on POSITN to build the escape
.; sequence that positions the
.; cursor.

.GOSUB POSITN 'COMMAN%C'

.; Build the prompt string for the
.; field.

.TEST FLDTXT
.SETN PAD 'FLDSIZ' .-<STRLEN>
.SETS FLDTMP BLANKS[1:PAD]
.SETS FLDTMP REV+FLDTXT+FLDTMP
.SETS FLDTMP COMMAN+FLDTMP+COMMAN

.; Get the input from the field.

.SETS ESCSEQ ""
.DISABLE LOWERCASE
.ASKS [::FLDTXT] TEXT 'FLDTMP'
.ENABLE LOWERCASE

.; Select the processor for the next
.; field.

.SETS S$ERR ""
.INC N$FLD
.SETS S$GOTO "DISPAT"
.IFT <EOF> .RETURN

.; Strip the terminating escape
.; sequence from the field.

.TEST TEXT ESCAPE
.IF <STRLEN> = 0 .TEST TEXT CSI
.IF <STRLEN> = 0 .TEST TEXT SS3
.IF <STRLEN> = 0 .GOTO ASKX
```

```
.SETS ESCSEQ TEXT[<STRLEN>:*]
.SETS TEXT TEXT[1:<STRLEN>-1]

.ASKX:.;

.; If there was no text entered,
.; supply the default.

.IF TEXT = "" .SETS TEXT FLDTXT

.; Redisplay the field.

.TEST TEXT
.SETN PAD 'FLDSIZ' .-<STRLEN>
.SETS FLDTMP BLANKS[1:PAD]
.SETS FLDTMP REV+TEXT+FLDTMP
.SETS FLDTMP COMMAN+FLDTMP+BOTTOM
.SETS FLDTMP FLDTMP+NML+CLRLIN+HOME
;'FLDTMP'
.RETURN
```

The above subroutine relies on subroutine `POSITN` to generate the escape sequence to position the cursor. The calling sequence for `POSITN` is `.GOSUB POSITN line;column` and the escape sequence is returned in String Symbol `COMMAN`. The following is suitable for an ANSI compatible terminal:

```
.POSITN:
.SETS COMMAN "'COMMAN%C' "
.SETS COMMAN ESCAPE+"[" +COMMAN+"H"
.RETURN
```

Once the text part of each field has been processed, the escape sequence that terminated it (if any) must be handled. This is done by a finite state machine, where each character of the escape sequence is dispatched for processing based on what it is and the current state of the system. In the example, the name of the current state is stored in String Symbol `ESCTYP`, and each character is handled by executing a `.GOTO` to a label composed of the state name and the ASCII code for the character (in octal). The example parses ANSI escape sequences, composed of an introductory sequence (`<ESC>` or `<CSI>` or `<SS3>`), some arguments (decimal numbers separated by semicolons), and a terminating character. The sequences `<ESC>[` and `<ESC>O` are recognized as alternates for `<CSI>` and `<SS3>`, respectively:

```
.; Initialize the escape sequence
.; parser:

.ESCPSI:
.SETS ESCTYP "INI" ! Parser
"state"
.SETN ESCAMX 0. ! Arguments
.SETN ESCA0 0. ! First
argument
```

```

.; Main parser loop:

.; Strip off the next character (if
.; any), convert it to a number, and
.; do a "computed" GO TO based on
.; current parser state and
character
.; code:

.ESCPSR:
  .IF ESCSEQ = ""      .GOTO 'S$GOTO'
  .SETS CHAR ESCSEQ[1:1]
  .SETS ESCSEQ ESCSEQ[2:*]
  .SETN CVALUE 'CHAR%V'
  .ONERR ESCPSE
  .GOTO 'ESCTYP' 'CVALUE'

.; Any unrecognized characters end
.; up here.

.ESCPSE:.;
  .SETN N$FLD N$OFLD
  .SETF GOLD
  .GOTO 'S$GOTO'

.; First character = <ESC>; set
.; state:

.INI33:.;
  .SETS ESCTYP "ESC"
  .GOTO ESCPSR

.; First character is <SS3>, or

.INI217:.;
  .; First was <ESC> and second is
  .; "O"; set state:

.ESC117:.;
  .SETS ESCTYP "SS3"
  .GOTO ESCPSR

.; Got <CSI>nnn~ = one of the "F"
.; keys. Dispatch appropriately.

.CSI176:.;
  .GOTO FKY'ESCA0%D'

.; Got <SS3>P = PF1 - use it as
.; shift key:

.SS3120:.;
  .SETN N$FLD N$OFLD
  .SETT GOLD
  .GOTO ESCPSI

```

Error Control

It can be useful to attempt an operation even though it may produce an error in the ICP. Although the ICP can not be set to ignore such errors, it can be set to dispatch them to the label of your choice for handling.

To cause errors to be trapped to your error handler, issue the ICP directive `.ONERR` label. The next error encountered will cause control to be transferred to the given label.

Errors are divided into numbered classes, as described in the ICP documentation. You can set bits in Special Numeric Symbol `<ERRCTL>` to determine which classes are trapped to your error handler. Untrapped errors will cause the ICP to abort. By default, only Class 1 errors are trapped. On entry to the error handler, Special Numeric Symbol `<ERRNUM>` contains the Error Class Number of the error encountered. The `.ONERR` directive must be reasserted after each error trapped.

The Error Classes are pretty broad (there are only two), and don't tell you very much about what actually caused the fault. If you are expecting more than one source of error, you will need to build your own logic to distinguish between them.

The manual says you should not resume processing after trapping a Class 2 Error, as the state of the ICP is indeterminate. I have found that it works in some cases, but recommend trying each case out before you build an application around it.

Example

`PRN.CMD` is a utility designed to print a file on an LA-series printer connected to the printer port of a VT100- or VT200-series terminal.

This example follows on from the one in the previous topic. The escape sequence parser may receive an escape sequence that it is not equipped to handle. It would be nice to recover from the ICP error that results. A trap for this is set in the `.ONERR` directive just before the main parser loop dispatches the character. If the character turns out to be unrecognized, the escape sequence processing is aborted and reinitiated from that point. This will probably result in more aborts until the buffer is empty, or until an `<ESC>`, `<CSI>`, or `<SS3>` is encountered. The code here also appeared in the previous example:

```

.; Initialize the escape sequence
.; parser:

.ESCPSI:
  .SETS ESCTYP "INI" ! Parser
  "state"
  .SETN ESCAMX 0.    ! Arguments
  .SETN ESCA0 0.    ! First
argument

.; Main parser loop:

.; Strip off the next character (if
.; any), convert it to a number, and
.; do a "computed" GO TO based on

```

```

.; current parser state and
character
.; code:

.ESCPSR:
  .IF ESCSEQ = "" .GOTO 'S$GOTO'
  .SETS CHAR ESCSEQ[1:1]
  .SETS ESCSEQ ESCSEQ[2:*]
  .SETN CVALUE 'CHAR&V'
  .ONERR ESCPSE
  .GOTO 'ESCTYP' 'CVALUE'

.; Any unrecognized characters end
.; up here.

.ESCPSE:.;
  .SETN N$FLD N$OFLD
  .SETF GOLD
  .GOTO 'S$GOTO'

```

Parsing an MCR- or DCL-like Syntax

Command Files can be invoked in much the same way as a CLI command, and the parameters passed are available to the Command File in String Symbols P0-P9. Parsing these parameters normally takes place in two phases:

- Parsing the file specification(s);
- Parsing the switches and options.

You need to design a command syntax that is both clear and easily parsed. Either MCR or DCL can serve as a model.

You get a “nicer” parser if you can process all the file specifications in one loop, with an inner loop for the switches. This way, all the work can be done in one place.

You may wish to check for a null command line, and get the information you need through .ASKx directives.

If the syntax for switches is properly defined, your switch parser code will be completely generic - that is, you can add switches without modifying the parser. This is done by:

- Defining a consistent symbol name convention for storing switch settings. In the example, V\$xx is used, where xx represents the switch name, and the Symbol Type of V\$xx determines how the switch is processed.
- Defining a consistent and restricted switch syntax. In the example, no switch may have more than one argument.

Example

SYMDMP.COMD is a command file that reads a .OBJ or .STB file, and displays the types and values of the symbols it finds there.

The command syntax for SYMDMP (which also has an interactive mode) is

```
>@SYMDMP outfile=infile/switches in MCR syntax, or>@SYMDMP infile/switches outfile in DCL syntax. The legal switches are:
```

/BR insert page breaks in the output file

/SP submit the output file to the print spooler

Switches may appear on either the input or the output file.

The first part of the parsing process is to define all switches and their defaults, and separate the actual switch specifications from the rest of the command:

```

.; Define and initialize the
.; command switches:

  .SETF V$SP ! /SP (spool)
  .SETT V$BR ! /BR (page break)

.; Determine processing mode
.; (interactive or command line):

  .IF P1 = "" .GOTO PROMPT

.; Get the file specs, from either
.; MCR or DCL syntax:

  .IF P2 <> "" .GOTO SWIEXT
  .PARSE P1 "=" S$OUT S$FIL
  .IF S$FIL <> "" .GOTO SWIEXT
  .SETS S$FIL S$OUT
  .SETS S$OUT ""

.; Peel the switches off the file
.; specifications:

.SWIEXT:.;
  .PARSE S$FIL "/" S$FIL S$SWIT
  .PARSE S$OUT "/" S$OUT S$JUNK
  .SETS S$SWIT "/" + S$SWIT + "/" + S$JUNK

```

Once the actual switches have been isolated, it is simple to loop through the list of them, checking existence and validating:

```

.SWITLP:
  .IF S$SWIT = "" .GOTO PROCES

.; Peel the next switch off, and
.; get its arguments:

  .PARSE S$SWIT "/" S$SWX S$SWIT
  .PARSE S$SWX ":" S$SWX S$SWP

.; Figure out whether it is asserted
.; or negated:

```

```

.SETT L$ASRT
.IF S$SWX = "" .GOTO SWITLP
.IF S$DSH = S$SWX[1:1] .GOTO
SWITNM
.IF S$NO <> S$SWX[1:2] .GOTO
SWITAS
.SETS S$SWX S$SWX[2:*]
.SWITNM:.;
.SETS S$SWX S$SWX[2:*]
.SETF L$ASRT
.SWITAS:.;
.SETS S$SWX S$SWX[1:2]

.; See if this switch has a
.; corresponding V$sw symbol:

.TEST S$SWX
.IFF <ALPHAN> .GOTO SWIBAD
.IFNDV V$'S$SWX' .GOTO SWIBAD

.; Dispatch the rest based on the
.; symbol type:

.TEST V$'S$SWX'
.GOTO SWIT'<SYMTYP>'

.; Logical symbol. Set its value to
.; the switch polarity:

.SWIT0:.;
.IF S$SWP <> "" .GOTO SWINPR
.SETL V$'S$SWX' L$ASRT
.GOTO SWITLP

.; Numeric symbol. Set its value to
.; the switch parameter:

.SWIT2:.;
.IFF L$ASRT .GOTO SWINNG
.TEST S$SWP
.IFF <NUMBER> .GOTO SWIIVP
.SETN V$'S$SWX' 'S$SWP'
.GOTO SWITLP

.; String symbol. Set its value to
.; the switch parameter:

.SWIT4:.;
.IFF L$ASRT .GOTO SWINNG
.SETS V$'S$SWX' S$SWP
.GOTO SWITLP

```

Multiple Precision Arithmetic

The ICP is capable of doing arithmetic on 16-bit signed or unsigned integer values. Occasionally, this is not sufficient.

Normally, access to the carry bit is necessary for extended precision, and this is not available in the ICP. However, if the operations are performed eight bits at a time, the ninth bit can be used as the carry bit in addition and subtraction. Since the product of two eight bit numbers is never more than sixteen bits, multiplication can also be done eight bits at a time, summing the cross products at the end. Division has to be done by the shift and subtract method, and is the slowest of the four conventional operations.

Obviously, a 32 bit result can not be stored in a 16-bit Numeric Symbol. However, a pair of symbols will do nicely. I recommend the use of a two-element array, created as described earlier under "Using Groups of Symbols as an Array". Alternatively (or in addition), the values can be converted to decimal and stored in a String Symbol.

To isolate the low-order byte of one of the operands, it suffices to perform a bitwise logical AND with the value 377 (octal). The high byte is obtained by dividing by 400 (octal). Assembling the resultant bytes into a word is the reverse of these steps.

Example

BRU.CMD is a preprocessor for BRU, the Backup and Restore Utility. It prompts the user for how the operation is to be done, and constructs a BRU command (along with the necessary device allocations, mounts, dismounts, CON commands, and so on) based on the user's input and the current state of the system.

One of the options available to the user is to initialize the output disk in a manner different than the input disk. For this option, it was desired to use the same algorithm to calculate initial and maximum index file size as is used by the INITIALIZE command. The maximum index file size depends on the volume size, and the calculation must be done in double precision.

The following subroutine will add two 32 bit numbers, each stored in a pair of Numeric Symbols named by the convention "xxxxxn" where "xxxxx" is the double precision "variable" name passed to the subroutine, and "n" is 0 or 1. The calling sequence is .GOSUB VADD variable variable. The sum is returned in the left-hand variable. The code to do this is:

```

.VADD:

.; Extract the variable names from
.; the argument list.

.SETS COMMAN "'COMMAN%&C'"
.PARSE COMMAN " " ST$A ST$B

.SETN OT$B0 'ST$B'0
.SETN OT$B1 'ST$B'1

.; Separate the first addend into its
.; constituent bytes.

```

```
.SETN OT$A0 'ST$A'0&377
.SETN OT$A1 'ST$A'0/400&377
.SETN OT$A2 'ST$A'1&377
.SETN OT$A3 'ST$A'1/400&377
```

```
.; Separate the second addend into
.; its constituent bytes.
```

```
.SETN OT$B3 OT$B1/400&377
.SETN OT$B2 OT$B1&377
.SETN OT$B1 OT$B0/400&377
.SETN OT$B0 OT$B0&377
```

```
.; Add the corresponding bytes of the
.; two addends, with carry.
```

```
.SETN OT$C0 OT$A0+OT$B0
.SETN OT$C1 OT$A1+OT$B1
.SETN OT$C1 OT$C1+(OT$C0/400&377)
.SETN OT$C2 OT$A2+OT$B2
.SETN OT$C2 OT$C2+(OT$C1/400&377)
.SETN OT$C3 OT$A3+OT$B3
.SETN OT$C3 OT$C3+(OT$C2/400&377)
```

```
.; Strip out the carry bits.
```

```
.SETN OT$C0 OT$C0&377
.SETN OT$C1 OT$C1&377
.SETN OT$C2 OT$C2&377
.SETN OT$C3 OT$C3&377
```

```
.; Reconstitute the sum.
```

```
.SETN 'ST$A'0 OT$C1*400+OT$C0
.SETN 'ST$A'1 OT$C3*400+OT$C2

.RETURN
```

Slaved "Captive" Accounts with Indirect

Since the privilege structure under RSX is an all or nothing proposition, often accounts are required which have limited access to certain privileged instructions. These accounts are usually referred to as captive accounts. Some special techniques are required in order to produce a truly "Captive" account. First, the command procedure must be "bomb proof", that is, it can not unexpectedly exit. This is a necessity since the account must be slaved to prevent unsolicited access to the command line interpreter (CLI). If the command procedure bombs, the terminal is locked up and can only be fixed by a privileged user's intervention (from another terminal).

There are only a few general extra steps that need to be taken when creating a command file of this type. First, Control-Z recognition must be enabled. Doing this enables the the command file to trap Control-Z's typed in response to an .ASKx directive. Next, you must provide for general error recovery using an .ONERR directive. I discovered this

when a user accidentally hit the BREAK key. Evidently an error signal is passed back to indirect by the terminal driver. An example of these techniques will be shown under the Menu Driven Command Files.

Menu Driven Command Files

The best way to limit access to the system is using a menu type command file. This file will put up a list of options from which to choose. The user may not do anything which is not available as an option. Below is a typical Menu Driven Captive Account for an operator. Notice that the only access is to the pre-defined choices.

```
.DISABLE DISPLAY
.ENABLE CONTROL-Z
.ENABLE DECIMAL
.ENABLE QUIET
.ENABLE SUBSTITUTION
.OPEN TI:
.OPTION:.DATA
.DATA
.DATA OPERATOR COMMAND FILE
.DATA 0 EXIT
.DATA 1 SHOW USERS
   .DATA 2 SET TIME
.DATA 3 SHUTDOWN
.DATA
.START: .ONERR OPTION
.ASKS CMD ENTER NUMBER >
.IFT <EOF> .GOTO OPTION
.TEST CMD
.IF <STRLEN> = 0 .GOTO OPTION
.IFF <NUMBER> .GOTO OPTION
.GOSUB 'CMD'
.ONERR
.GOTO OPTION
.0: BYE
.EXIT
.1: DEV /LOG
.RETURN
.2: .ASKS HR ENTER TIME HH:MM:SS >
TIM 'HR'
.IF <EXSTAT> <> <SUCCES> .GOTO 2
.RETURN
.3: RUN [1,54]SHUTUP/TASK=BYEBYE
.WAIT BYEBYE
.RETURN
```

This file should be placed into the appropriate privileged directory. Further, the account should marked as slaved in the system accounting file. If additional security is needed, an audit trail of each command that is entered could be easily added. An audit trail can help to ensure accountability for the actions of an individual.

Menu driven command files could also be used in cases where security is not really an issue. If, for example, the syntax for a particular command or set of commands is rather

awkward, a menu could be very beneficial. Further, if you have an occasional user who only needs access to a few commands, this could provide a simple mechanism to accommodate them.

The .TESTDEVICE directive

One of the more complex and less understood directives in INDIRECT is the .TESTDEVICE directive. This directive allows a command file to acquire information about any device in the system. The information about the device is stored in a string symbol <EXSTRI>. As the description in the INDIRECT manual indicates, the information passed back in the variable is the full physical name of the device, four device-characteristics words, and device attribute information. This information is separated by commas and can be easily separated by the .PARSE command.

Of the information given, the most interesting and obscure information is held in the four device-characteristics words. The definitions of the bit fields in these words can be found in the appendix of the *RSX-11M/11M-PLUS Guide to Writing an I/O Driver*. Yes, that dreaded manual of the RSX manual set. If you turn to Appendix A, you will see the System Data Structures and Symbolic Definitions. If you scan this appendix, you will come to the UCBD\$ definitions. This is the section that defines what U.CW1 thru U.CW4 mean.

I discovered the usefulness of this directive when trying to write a command file which would log out all non-privileged users when backup's began. This seems like a rather simple task until you try to determine who is logged in and non-privileged. It just so happens that the information sought happened to be in the second device characteristic word of the TT driver. By masking all but the bits which were needed, the information could be easily determined.

The necessary information to accomplish this task was taken from Appendix A of the Device Driver Writer's Manual in the UCBD\$ section. The Terminal Dependent Characteristics Word 2 (U.CW2) Bit Definitions permits the following information to be found.

U2.LOG=400 User Logged on Terminal (0=Yes)

U2.PRIV=10 Unit is A Privileged Terminal (1=YES)

The following example will display for each TT: type device whether or not it is logged in and whether or not it is privileged.

```
.ENABLE SUBSTITUTION
.DISABLE QUIET
.ENABLE DISPLAY
.DISABLE DECIMAL
.SETS TYPE "TT"
.SETN NN 0
.10:
.SETS DEV TYPE+"NN%ZL2' '+':"
.TESTDEVICE 'DEV'
.SETS ABC <EXSTRI>
.IF ABC = "NSD," .EXIT
```

```
.PARSE ABC "," PHYDEV CW1 CW2 CW3 CW4
CW5
.SETN UCW2 'CW2'
.SETN U2PRV 10
.SETN U2LOG 400
.SETF LOGOUT
.SETF PRIV
.SETS XXX ""
.SETN PV UCW2&U2PRV
.SETN LO UCW2&U2LOG
.IF PV > 0 .SETT PRIV
.IF LO > 0 .SETT LOGOUT
.IFT LOGOUT .SETS XXX XXX+"NOT LOGGED
IN "
.IFF LOGOUT .SETS XXX XXX+"LOGGED IN
"
.IFT PRIV .SETS XXX XXX+"AND
PRIVILEGED "
.IFF PRIV .SETS XXX XXX+"AND NOT
PRIVED "
.; !Display the results
;TERMINAL 'PHYDEV' is 'XXX'
.INC NN !Increment Unit Number
.GOTO 10
```

Another interesting application of the .TESTDEVICE directive is to provide a limited ability to communicate on a system-wide basis. For example, at our site we wish to disable logins to non-privileged users during backups. The mechanism which RSX provides disables logins for all users. Clearly, we needed a different mechanism than was available. Instead of hacking up DEC's code, we choose to control logins in LB: [1,2]SYSLOGIN.COMD. The problem was how to inform SYSLOGIN.COMD that backups were in progress. The answer was rather simple but not obvious. A global assignment of logical name BK: to the system disk was made when we wished to disable non-privileged logins. Thus, in our SYSLOGIN.COMD we performed a .TESTDEVICE BK: and if it wasn't assigned it would return NSD in <EXSTRI>. When logins could be enabled again, the assignment to symbol BK: would be cleared.

An excerpt from LB: [1,2]SYSLOGIN.COMD file dealing with the device assignment technique.

```
.NMOD:
.IF P5 <> "P" .GOSUB CKTERM
.IF P5 = "P" .GOTO SLAV
SET /NOPRIV=TI:
.SLAV:
.IF P6 = "S" .GOTO OVER
.IF P7 = "T" SET /NOSLAVE=TI:
.OVER:
CLI /UNOVR
.SETS FILE P1+P2+"LOGIN.COMD"
.TESTFILE 'FILE'
.IF <FILERR> = <SUCCES> .CHAIN
'FILE'/LO
.;
.; WOOPS SLAVED ACCOUNT WITH
```

```

.; NO LOGIN.CMD FILE
.;
.IF P6 = "S" .XQT BYE
.EXIT
.CKTERM:
.;
.; THIS PORTION OF THE ROUTINE
.; DETERMINES IF BACKUPS ARE IN
PROGRESS
.; AND INFORMS THE USER OF SUCH.
.;
.TESTDEVICE BK:
.SETS TEST <EXSTRI>
.IF TEST = "NSD," .RETURN
.IFINS ...CA. CLR
.DISABLE QUIET
;
;*****
;* LOGINS DISABLED      *
;* BACKUPS IN PROGRESS *
;* TRY AGAIN LATER     *
;*****
;
.ENABLE QUIET
.DELAY 3S
BYE
.EXIT

```

Here is an excerpt from our BACKUP .CMD file which provides the necessary code to enable and disable non-privilege logins.

```

ASN LB:=BK:/GBL !DISABLE NON-PRIV
LOGINS
BRU etc...
ASN =BK:/GBL !ENABLE NON-PRIV LOGINS

```

Since device assignments can be made for a specific group, this could be used as a mechanism to restrict access to a system according to group number designation. For example, if we wish to restrict access to group 377 during backups the following assignment could be made: ASN LB:=BK:/GRP=377.

Read With No Echo

From time to time, it becomes necessary for a command file to be able to read in some information from the keyboard without echoing the data back to the screen. This is usually associated with passwords and other private data. Indirect does not directly support read with no echo, however the full-duplex terminal driver does support noecho mode. Hence a simple example is in order.

```

.ENABLE SUBSTITUTION
.DISABLE DISPLAY
.ENABLE QUIET
SET /NOECHO=TI:
.ASKS PASWRD Enter password >

```

```

.ASKS PASWD2 Reenter password >
SET /ECHO=TI:
.OPEN TI:
.IF PASWRD = PASWD2 .DATA Match!
.IF PASWRD <> PASWD2 .DATA No Match!
.CLOSE
.EXIT

```

The above example simply reads in two strings without echo and compares them for equality and inequality. The command procedure prints a statement as to the equality or inequality of the input data.

System Wide Indirect Command Files

If you are in a development environment, you have probably created many useful command files. There is an easy way to have a system-wide indirect command directory which would be searched whenever a command file was not found in a user's local directory. It also reduces unnecessary duplication of command files, eliminates typing device and directory specifications, and allows easy updating of command files.

By default, a command file search begins in the user's own directory and concludes in LB:[1,54] for 11M or LB:[3,54] for 11M+. To change this default behavior to search another directory, one needs to simply decide which UFD (User File Directory) to use on the LB: device (usually the system boot disk). Calculate the octal representation of the chosen directory. Then rebuild Indirect with a modified global symbol. Finally, run VMR to remove the old INDIRECT and re-install the new one into the system image.

The Indirect build file, which is created by the Sysgen procedure, needs to be modified slightly. The file name is dependent upon the flavor of RSX being used. If you are using RSX-11M [1,24]ICPBLD.CMD needs modification or if the RSX-11M-PLUS is being utilized then [1,24]ICMBLD.CMD, ICMRESBLD.CMD or ICMFSLBLD.CMD needs to be changed. The name of the file is dependent upon the resident library which is used in building it. You must also answer YES to the question in Sysgen "Do you wish to modify any files before building?". Sysgen will pause after creating the build file, to permit modifications. The modifications necessary are outlined below.

The group and the member number can each be represented in an 8-bit binary pattern as indicated below for [1,3].

```

0 000 000 100 000 011  Binary
G GGG GGG GMM MMM MMM  Group/Member
number
0 0 0 4 0 3  Octal

```

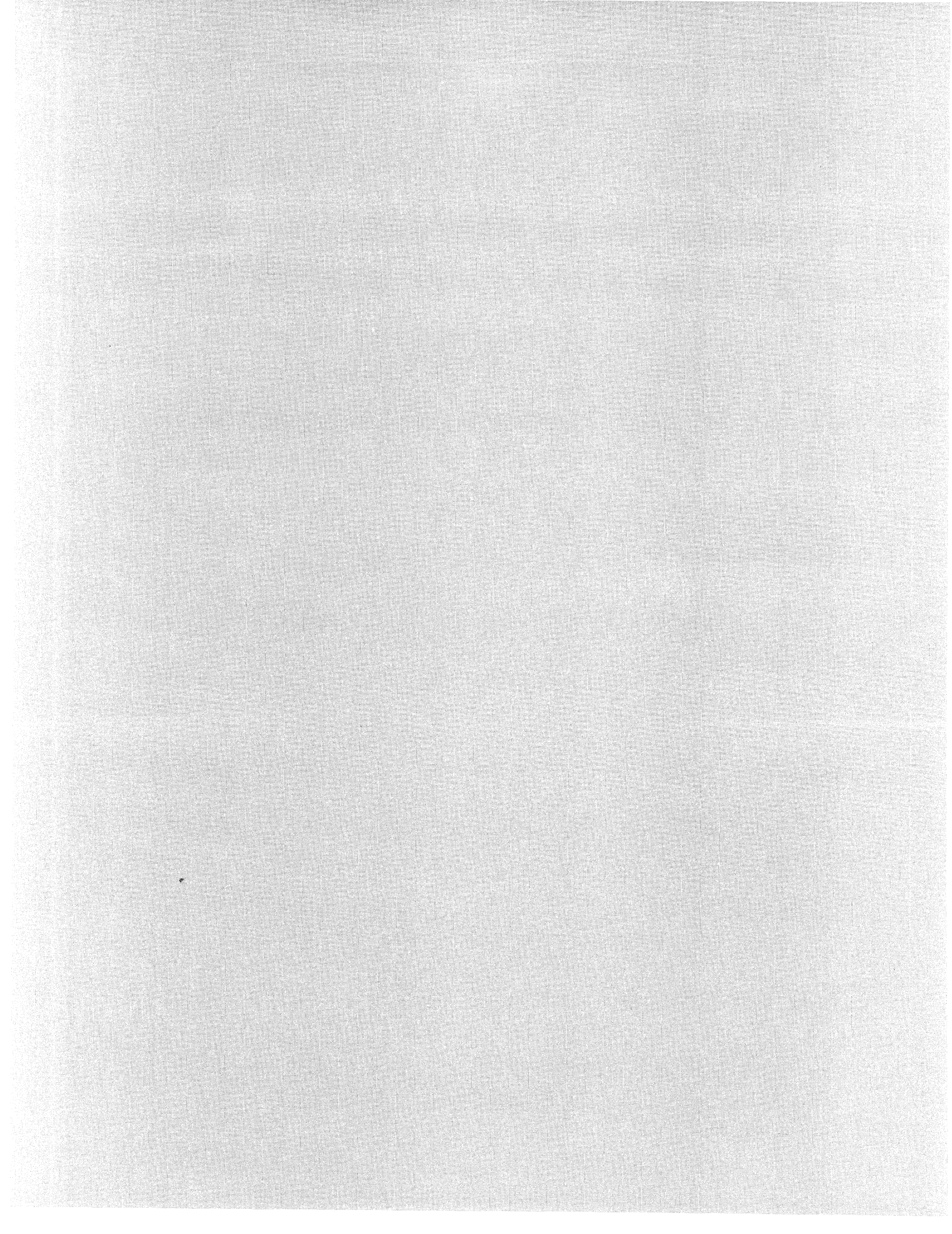
Assuming that we wish to rebuild Indirect so that the default system command file directory is LB:[1,3] on an RSX-11M-PLUS with Supervisory Mode Library support. The following procedure would accomplish this.

1. Invoke Sysgen to rebuild INDIRECT

2. Find the line GBLDEF=D\$CUIC:1 in the Build File
3. Change the line to GBLDEF=D\$CUIC:000403
4. Finish re-building INDIRECT
5. Run VMR to do (1) REM ...AT (2) INS
LB: [3,54] ICMFSL/TASK=...AT./INC=10000

References

- [1] Watson, Allan A, Nifty Things to Do with RSX Indirect Command Files, *RSX/IAS SIG Symposium Handout*, Spring 1983 DECUS US Symposium
- [2] Watson, Allen A, Nifty Things to Do with RSX Indirect Command Files, *The DEC Professional*, March, 1984.
- [3] DeLarisch, Arnold S, *RSX 11 System Management, A Beginner's Perspective*.
- [4] *LA50 Printer Programmer Reference Manual* (Documents printer escape sequences).
- [5] RSX LB: [1,2] ICP.HLP, (On-line help file for ICP. Contains some information that is not in the manual.)
- [6] RSX LB: [1,2] INDSYS.CLB, (Sample command routines.)
- [7] *RSX-11M/M-PLUS RMS-11 Utilities*. (The reference for RMSDES.)
- [8] *RSX-11M/M-Plus Task Builder Manual*. (Documents object file layout.)
- [9] *RSX-11M/M-PLUS Indirect Command Processor Manual*. (The primary reference for the ICP under RSX.)
- [10] *RSX-11 Utilities Manual*. (Reference for the librarian task (LBR).)
- [11] *RSX-11M-PLUS Guide to Writing an I/O Driver*. (Reference for the UCB data structures.)
- [12] *VT220 Programmer Pocket Guide*. (Documents escape sequences for VT2xx terminals.)



Moving Decision Points Outward From Applications and Utilities Into Command Level

Maarten van Swaay
Department of Computer Science
Kansas State University
Manhattan, Kansas 66506

Abstract

The user command interface of an operating system has come to be regarded as the outermost level in a layered hierarchy. In keeping with this view one should attempt to move decision points outward from the lower operating layers, so that all required actions can be defined at command level. That will allow the description of those actions in a form that matches the form of other commands normally entered from the keyboard, so that only a minimal amount of programming skill will be required. The strategy is illustrated with a utility that handles modem initialization, dialup and login dialog in preparation for a session handled by a terminal emulator.

Introduction

Commands entered by the user of a system can be seen as the executed path of a program that resides in the mind of the user (Figure 1). Messages displayed by various utilities invoked by user commands serve as performance reports that must be interpreted by the user for the selection of subsequent commands. If the command sequence for a given session is long and elaborate it will become attractive to collect the sequence into a command file. The command file then becomes a program that can call various system utilities and user applications as procedures.

The program represented by the command file replaces the program that was previously held in the mind of the user. Thus the command program must have authority to make decisions based on performance reports returned to it (Figure 2). Conversely, performance messages produced by various utilities must no longer be directed at the user, but must serve as performance reports returned to the command program.

The user who wants to remain unaware of the inner workings of the system may feel uncomfortable with the idea of having to write a command program, but that same user should not find it difficult to write a simple list of the commands he/she would otherwise have entered directly at the keyboard. The IND utility provides extensive support for the capture of performance reports, for text substitution, and for program flow control. That flexibility comes at a price, however: a typical IND program bears little resemblance to the familiar string of keyboard commands.

The older and simpler *.COM file is considerably more flexible than many users recognize, and it has the virtue of close similarity to conventional keyboard commands. In com-

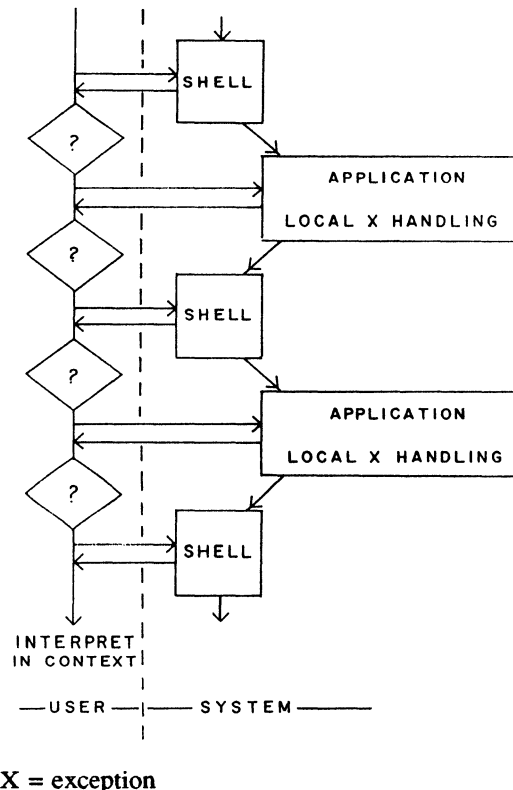


Figure 1: Manual invocation of applications

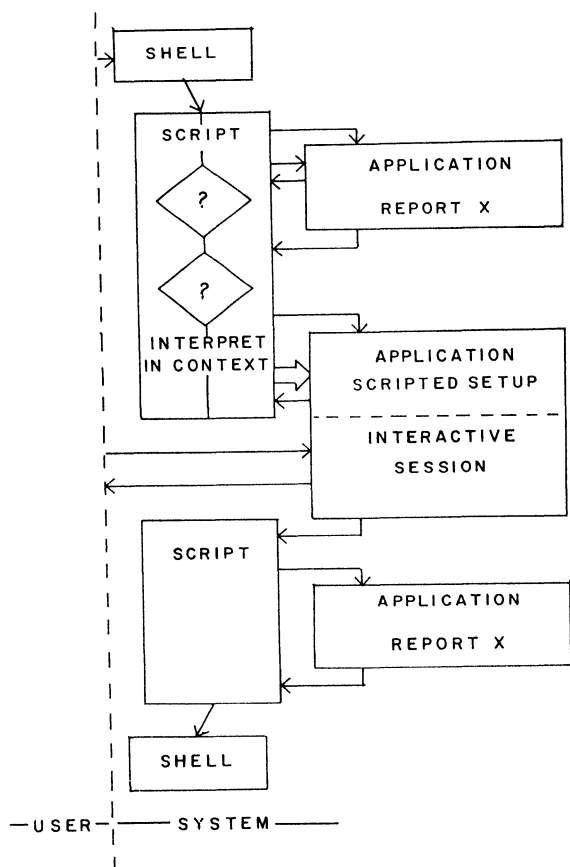
combination with the RT11 facility by which programs can return command strings on exit, the *.COM file can handle branches and aborts at command level. Iteration loops can be implemented by repeated invocation of command files, but the approach is clumsy at best. In practice many command sequences are found to be loop-free, however.

A package that controls a modem and establishes a connection to a remote host may serve to illustrate the strategy outlined above. In the absence of standards for the behavior of remote terminals one will normally find several terminal emulators on any given system, each of which is designed to handle sessions with a corresponding remote host. It is obviously possible to build facilities into each emulator to handle the modem control and login dialog. A cleaner approach is based on a separate **DIAL** utility that initializes a session with a remote host and then hands over control to the appropriate emulator. The initialization proceeds via several stages:

- acquisition and setup of serial port
- setup of modem behavior
- connection to remote site by dialup
- penetration of data switches at remote site
- login dialog with remote host
- transfer of control to appropriate emulator

At each stage allowance must be made for the possibility of failure. For this illustration we may decide that local failures (unaccessible serial port, unresponsive modem) should result in termination of the attempt. On failure to establish a connection we may wish to try another phone number, or we may wish to revert to a manual attempt. **DIAL** supports these choices by discarding the current command file after local failure. On failure of the dialup stage **DIAL** returns a command string @SY:DIAL.ABT; thereby replacing the current command file with the command file DIAL.ABT, which can contain the desired alternate command sequence. With only minor enhancements **DIAL** could be made to accept the specification of a replacement command file at each stage of its progress. In the absence of any exceptions **DIAL** will exit on completion of its assigned task, and the next statement in the command file will start the appropriate emulator.

For some remote hosts the penetration and login dialog cannot be entirely defined in a script. An example of such a situation would be a smart remote modem that must make a selection between several protocols. Such modems may require a variable number of <CR> characters to achieve proper setup. In addition some remote sites may respond only after unpredictable delays. It would be possible, but not practical, to build facilities into **DIAL** to handle such situations. Instead, **DIAL** is designed to switch between user input and command file input where necessary. **DIAL** recognizes a command file statement beginning with <.M> as a command to switch to keyboard control; a keystroke of <LF> serves to switch control back to the command file.



X = exception

Figure 2: Invocation from a shell script

Because **DIAL** must handle a variety of remote sites, **DIAL** expects a minimum of cooperation from the remote site. **DIAL** expects at least one return character after each transmitted string, and it allows up to 30 seconds for that return. The long patience serves primarily to allow for delays introduced by the local and remote modems and by the phone system. Beyond the first returned character **DIAL** will treat two seconds of silence as indication of the end of response. Finally **DIAL** has the capability to test the first returned character against an expected return supplied to it from a command file. Even with these limited capabilities **DIAL** has reliably established connections to four very dissimilar remote hosts for more than 1000 sessions.

The source code of **DIAL** (MACRO-11) and a description of its structure and use will be included on the Nashville RT11 tape. Figures 3 and 4 show an example consisting of a pair of *.COM files used to initiate a session with the departmental VAX-11/780 (UNIX BSD 4.3) at Kansas State University.

```

! DK:VAX.COM  DATE: 5-APR-87  TIME: 17:41
! modem control commands according to HAYES protocol
!
R DIAL
.C Resetting modem ...      ! .C allows display of comments by DIAL
.R                          ! recognized by DIAL as modem reset command
! switch to manual mode and enable modem speaker, to test for phone-in-use
.M ... listen for line in use ... ATM1 if busy ...
ATM2D
! proceed if line is available by striking <LF> on keyboard
! null line below aborts modem dial sequence used to enable speaker

ATM1V0TD 987-6543 <1        ! primary phone number, expected return = "1"
! the remote modem requires one null line for synchronization

! the data switch requires one null line to wake it up

nnnnnnn                    ! user name (UNIX BSD4.3
ppppppp                    ! password
! null line below approves default terminal type built into user account

^C                          ! dismiss DIAL
R FDX                       ! start emulator

```

Figure 3: An Example Command File

```

! SY:DIAL.ABT  DATE: 5-APR-87  TIME: 17:45
!
R DIAL
ATM1V0TD 987-6541 <1        ! alternate phone number

nnnnnnn
PPPPPPP

^C
R FDX

```

Figure 4: An Example Command File

SITE MANAGEMENT AND TRAINING SIG

Analysis of VMS Accounting Data for Determination of Computing Resource Consumption

Nancy J. Martin

USA LABCOM, Harry Diamond Laboratories
Adelphi, Maryland

Abstract

The CAD/CAM Systems Group at the U.S. Army's Harry Diamond Laboratories provides the staff with a general-purpose scientific computing environment. Using a VAX 11/780¹ computer, the Group supports approximately 250 users both on and off site. The tracking of specific computing resource usage is essential for billing and budget-planning considerations by the Group as well as by the users. Group members have developed software, using VMS accounting data, to analyze information about resources consumed on a user-by-user basis. Reports are generated detailing this information in an easily understood manner. The presentation will describe the data analysis process, with emphasis on the techniques used to convert the VMS accounting data to a more useable form.

Background

Whenever a computer system expands its user base beyond the bounds of the group which originally owns it, some basic operational questions arise, such as who pays for the system operation. In a laboratory environment such as exists at Harry Diamond Laboratories (HDL), the money funding Group A is probably different from that funding Group B, and both are different from that funding Group C. If Group A operates the computer, it probably does not want to fund the computing activities of these other groups. Therefore, a need arises for determining the resources consumed by each group.

The CAD/CAM Systems Group at HDL has developed a software package for doing this. The Group operates the computer in a time-sharing system with cost recovery.² Users provide a project number to charge against, and the Group collects consumption data and calculates the charges. This charging information is then sent to the HDL's Cost Accounting Section, which performs the actual billing. The collected data allow the Group to predict future performance bottlenecks and set appropriate rates for cost recovery. In addition, users can estimate their own computing requirements from past consumption data.

VMS logs every user action in the form of binary data records and provides a utility which collects the binary consumption data and translates them into ASCII format. These data can be used as input into programs that will perform the actual charge-back operation, i.e., collect the resource data and compute the charges incurred. At the time this package

was done, commercially available charge-back programs did not always collect the desired data for every possible situation. Therefore, the decision was made by the Group to write its own software. Furthermore, the Group decided to use the binary accounting data directly and bypass the VMS utility.

As is, the VMS utility collects no overall usage statistics, the format is incorrect for the existing charge programs, no count is given for the number of print jobs, and, because of a special output device - a laser printer/plotter-the print page count is incorrect. A plot with large numbers of vectors is seen by the printer driver as having many pages, when in reality there is only one page printed. Furthermore, there is also a bug in the current version of the accounting logger.³ When a privileged user submits a batch job or prints a job with the /USER switch, the accounting logger should create a record as if the named user had done the print or submit. However, the generated record does not contain the account number of the selected user.

Because of the funding situation at HDL, a user may have many account numbers, each corresponding to a different project number, associated with that username. Only one of these accounts is active at any given time, but it is possible for a user to have more than one associated account number over the accounting period. Since user data are collected by user and account number, the VMS utility will not connect the user data with the proper account when the account number is missing.

It would be possible to run the utility numerous times with different switch options and then feed the results into a program that would format the output, fix the job count, etc., but the overhead involved in repeated runs of the utility

¹VAX and VMS are trademarks of the Digital Equipment Corporation

²Cost recovery is a scheme in which user funding exactly offsets operational costs.

³VMS Version 4.4

is prohibitive, especially if the data file is large. The Group decided that writing its own program would be more cost and time effective; in addition, the code could be modified to fit changing conditions within HDL.

VMS Accounting Data Format⁴

SYSS\$MANAGER:ACCOUNTNG.DAT is the VMS accounting log file. The file consists of a series of variable length records, each with the same general format (Fig. 1). The records consist of a record header and a number of information packets, the exact number of packets depending on the record type. There are presently eight types of accounting records generated by system events:

- Process termination
- Image termination
- Login failure
- System initialization
- Print job
- Forward accounting file link
- Backward accounting file link
- User supplied data

However, for this paper, only the first five record types listed above are of interest.

The record header consists of a length field, which contains the length of the entire record; a type field; and a system time field. The system time is the time the record was logged, and is in 64-bit format. The type field contains the record type and, in the case of a process termination record, the type of process in the subtype field. There are six process types:

- Interactive
- Subprocess
- Detached
- Batch
- Network

The type of process affects the statistics to be collected. For example, since the Group does not charge for connect time on batch jobs, there is no need to collect that information for a batch process record.

Each record consists of one or more of the following data packets:

- Identification
- Resource

⁴The source for the following information is Appendix A, Supplemental ACCOUNTING Information, of the accounting utility documentation, volume 5A of the VAX/VMS documentation set.

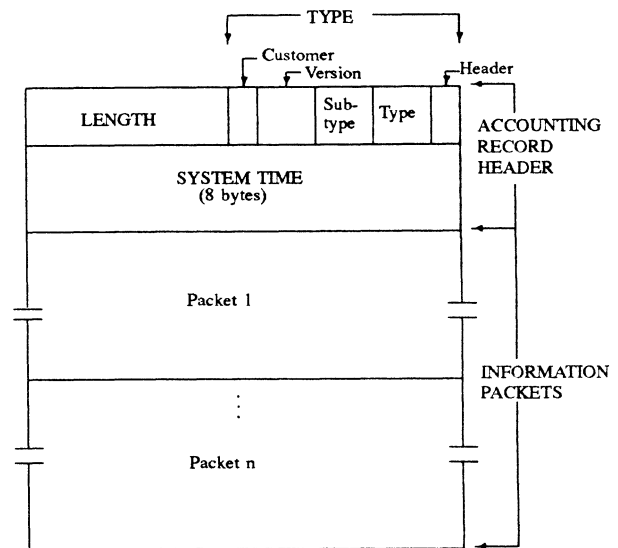


Figure 1: General Record Format.

- Print resource
- Image name
- File name
- User data

These packets have the same general format as shown in Figure 2. For HDL charge-back software, only the first three packet types are of interest. These packets contain the following information:

- Identification
 - Username
 - Account number
 - Queue
 - Jobname
- Resource
 - System start time
 - CPU time
 - Page and I/O faults
 - Volume mounts
- Print Resource
 - Pages printed

Like the accounting records, the packets start with a header field which contains packet length and type. The rest of the packet is made up of fixed-length data fields and, optionally, a variable-length data area. For example, the identification packet consists of a series of fixed-length fields containing process information and offsets into a variable-length data area (Fig. 3). The resource packet, on the other hand, contains only

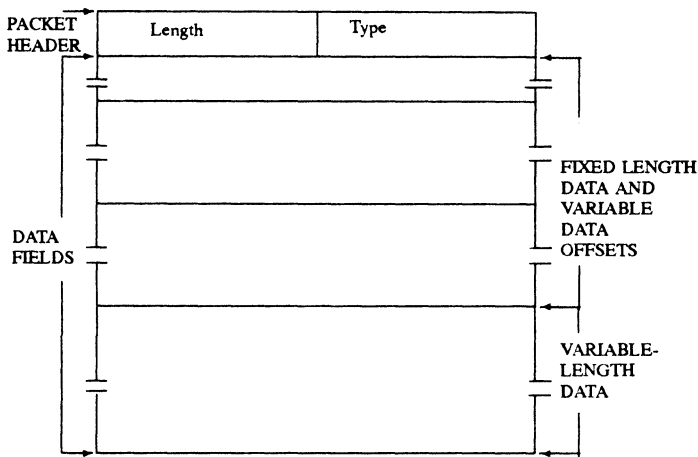


Figure 2: General Packet Format.

fixed-length data. The offsets are counted from the beginning of the packet, and all variable data are counted; i.e. the first byte contains the length of the data.

The records contain the following packets of interest:

- Process termination
 - Identification Packet
 - Resource Packet
- Image termination
 - Identification Packet
 - Resource Packet
- Login Failure
 - Identification Packet
- System Initialization
 - Identification Packet
 - Resource Packet
- Print
 - Identification Packet
 - Print Resource Packet

Data Extraction

There are two types of data extraction to be performed. The first is a direct extraction performed on the fixed-length data and the offset fields. For variable-length data, a double extraction is needed, once to get the offset value, and the second time to get the data itself.

The data extraction can be performed with a system run-time library function, LIB\$EXTZV. The function returns an integer value and expects three arguments, the position relative to a base address, the size of the field to be extracted, and the address of the base. The base address can refer to either

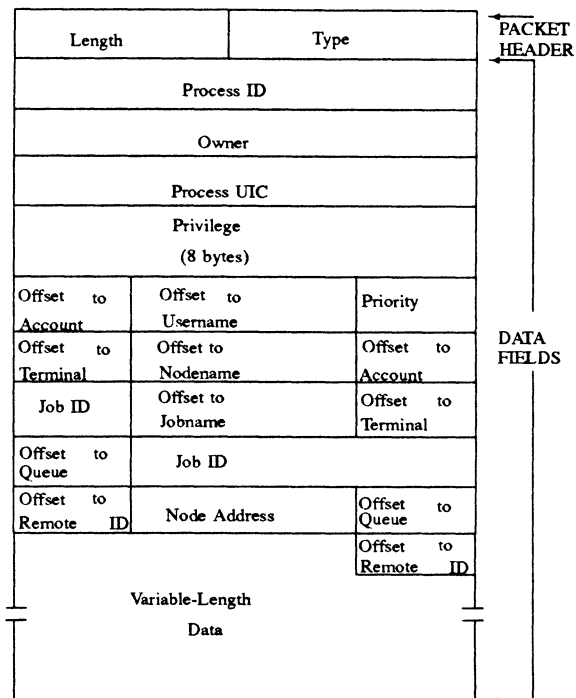


Figure 3: Identification Packet Format.

the start of the record or to the start of the packet, if it is known. Since the largest field which can be extracted using this function is a longword, or 32-bits, this function can be used directly on any of the data fields except the 64-bit time fields. To read the 64-bit data, an INTEGER*4 array of size two should be declared and consecutive calls to the function performed.⁵

The FORTRAN EQUIVALENCE statement is used in the reading of the variable-length data. The record is stored in a variable that is declared BYTE. This variable is equated to a CHARACTER variable, and the desired data are extracted using straightforward string manipulation.

A set of symbolic definitions is available that contains the location of the fixed-length data fields within the records or packets, thus easing the process of extracting data. The definitions are contained in the symbolic definition macro \$ACRDEF. Other symbols representing record and packet types are also contained there.

Data Manipulation

All integer data, except for the 64-bit time manipulation, involve standard addition and subtraction. Thus the process of totaling the user collection data is straightforward. The addition or subtraction of the 64-bit data requires the system library routines LIB\$ADDX and LIB\$SUBX. Both routines require three arguments - operand (or minuend), operand (or subtrahend), and the result - all of which are INTEGER*4 arrays of two elements.

⁵ All the programs are written in FORTRAN-77, so any programming references are FORTRAN specific.

jobname is extracted. Since all plot jobs have the same unique name, if the jobname matches that unique name, a page count of one is used; otherwise, the true print count is extracted and used.

The solution to the /USER problem was to get the current⁶ account number from the SYSUAF.LIS file and charge to that number.

In addition, with only minor modifications to the code (something which is impossible with most commercial software as the source code is unavailable), the Group is able to monitor resource usage on specific devices or classes of processes. If a system is heavily saturated, the system manager will be able to pinpoint those users who run excessive numbers of subprocesses. The manager will also be able to track the usage of terminals or tape drives.

Conclusion

Once the decision was made to bypass the accounting utility, the job of writing the software was straightforward since VMS provides all the tools for analyzing the binary accounting data. In addition, by writing its own software, the Group is able to control which data are collected and make any changes necessary to reflect a changing computing environment.

References

- [1] *Programming in VAX Fortran*, Digital Equipment Corporation, 1984
- [2] *VAX/VMS Volume 5A System Management*, Digital Equipment Corporation, 1986
- [3] *VAX/VMS Volume 8B System Routines*, Digital Equipment Corporation, 1986

⁶At the time the accounting program was being run

TOTALS FOR FILE: sample.dat

ACCOUNTING PERIOD: JAN 1987

USER	LOGINS	CONN-TIME	CPU-SECS	BUF I/O	DIR I/O	PAGE FLTS	VOL-MOUNTS	PRINT-JOBS	PAGES	ACCNT
USER1	1	0:44:24	164.	3127	349	3996	0	0	0	ACC047
USER2	1	0:16:18	88.	536	125	2820	0	0	0	ACC051
USER3	1	1:47:45	79.	32460	549	3038	0	0	0	ACC178
USER4	0	00:00:00	18.	1859	32	4190	0	13	13	ACC226
USER5	5	1:51:42	214.	36541	2937	18290	0	4	24	ACC069
USER6	3	0:54:03	57.	6556	699	5959	0	15	35	ACC054
USER7	4	3:12:37	402.	144107	5138	19233	0	3	3	ACC120
USER8	1	0:10:35	16.	5058	188	1354	0	0	0	ACC045
USER9	1	0:23:27	119.	1482	1693	9087	0	0	0	ACC239
USER10	1	0:22:43	115.	1567	526	5982	0	5	5	ACC214
USER11	5	0:56:17	121.	7050	1191	17141	0	5	12	ACC208
USER12	6	3:37:13	432.	106151	5540	33504	0	1	6	ACC195
USER13	1	0:20:44	32.	1165	459	4663	0	0	0	ACC019
USER14	11	3:38:53	1702.	22951	8212	73642	9	85	151	ACC194
USER15	8	1:24:58	3853.	250706	144550	308478	1	4	11	ACC002
USER16	3	2:31:58	363.	26996	4040	35440	0	0	0	ACC075
USER17	2	2:05:44	204.	16284	1361	16653	0	10	130	ACC247
USER18	1	0:01:18	7.	154	82	1264	0	0	0	ACC238
USER19	4	2:30:03	310.	19288	5500	33265	0	3	35	ACC176
USER20	5	2:15:17	578.	10094	16658	21103	4	2	11	ACC142
USER21	3	0:43:38	144.	1353	816	5689	0	5	52	ACC137
USER22	1	0:20:14	27.	7331	268	2271	0	1	1	ACC145
USER23	1	0:30:19	248.	1150	614	15444	0	9	9	ACC219
USER24	5	1:15:41	72.	8893	767	10111	0	1	1	ACC227
USER25	7	6:41:43	724.	71208	4906	48578	0	5	13	ACC007
USER26	7	3:09:30	256.	69315	2589	20713	0	9	43	ACC046
USER27	4	1:23:52	70.	1142	695	7332	0	8	14	ACC056
USER28	1	1:03:21	103.	5207	349	16747	0	0	0	ACC224
USER29	2	0:49:10	90.	9693	951	10678	0	0	0	ACC248
USER30	6	0:01:59	17.	334	167	3400	0	0	0	ACC000
USER31	4	1:03:26	190.	23136	3262	12946	0	0	0	ACC143
USER32	1	0:01:10	7.	127	87	1176	0	0	0	ACC003
USER33	5	1:04:24	105.	24506	1191	7815	0	7	16	ACC249
USER34	1	0:22:44	106.	14104	910	7334	0	0	0	ACC234
USER35	2	0:01:29	11.	275	115	2156	0	0	0	ACC014
USER36	0	00:00:00	1216.	478	2095	78066	0	6	6	ACC131
USER37	5	3:11:21	1085.	9826	1739	34081	0	2	2	ACC221
USER38	1	0:01:06	9.	198	165	1406	0	0	0	ACC005
USER39	1	0:16:58	32.	958	1341	2315	1	1	4	ACC213
USER40	2	2:30:50	299.	14124	2846	22237	0	6	131	ACC059

TOTAL_RECORDS READ: 672
 TOTAL_LOG FAILURES: 30

TOTAL LOGINS: 123
 TOTAL CONNECT TIME: 53:39:13
 TOTAL_CPU: 13684
 TOTAL PRINT JOBS: 210
 TOTAL PAGES PRINTED: 728

Figure 4: Sample Output from Data Collection Program.



Brent Teeter, P. E.
 Naval Weapons Center
 China Lake, CA. 93555

ABSTRACT

Once a computer system has been purchased, the anxious optimism of waiting for it to arrive will fade into the realization of where to put it when it arrives. By considering the computer system site early in the procurement process, headaches can be reduced for system startup, system operation, maintenance, and expansion.

INTRODUCTION

Good computer room design practice entails addressing six concerns: setting up the design team, physical requirements, electrical requirements, cooling requirements, security, and contractor interactions. With proper attention to these needs, a well designed computer room will provide a reliable environment for the computer and will improve work performance of the people that use it.

GETTING STARTED

There are a number of concerns that should be addressed before the design gets underway. The first concern is setting up the design team. People who have a vested interest in the results should be used to assist in the design and review. An example of such a person is the system manager. He will have to live with the computer room that results from the design. Other good sources of people are those who have shown an interest in the project, who have good memories, and who have the time to pay attention to the construction. A third source of help is DEC Field Service. Field Service can provide specifications and requirements for much of the computer room.

Once the design team is established, team members should keep a historical record of all interactions with contractors and consultants. This record may be useful in the future if performance problems occur.

When the computer room design is finished, it should be reviewed by at least two knowledgeable people outside of the design team. This review is necessary because people who are intimately involved in construction designs sometimes miss details. Finally, due to the high time demands placed on design team members, it may be advantageous to hire a consultant. However, consultants do not have the vested interest that employees have.

PHYSICAL REQUIREMENTS

The first task in designing a computer room is to determine the size and weight of each cabinet and peripheral that will be placed in the room. Adequate room for growth should be allowed as well as clearance to allow rear doors of cabinets to be opened.

The easiest method of determining equipment placement is to make a floor plan using a convenient scale (ie. 1/4 inch equals 1 foot), cut out each peripheral and cabinet floor footprint to the same scale, and place them in the floor plan until all specifications have been met. While laying out equipment locations, it is important to consider workflow. Workflow considerations make users of the computer room more efficient. An example of workflow is to group console terminals of multiple machines together so that the system manager can use them with a minimum of movement and effort.

An important benefit of using a floor plan is that when the computer room construction is completed, the plan can be used to accurately locate equipment in the room. Each cabinet can be located by taping out the location of it.

If the amount of equipment in the computer room justifies it, a raised floor should be considered. Raised floors have several benefits, among them being:

- * The floor acts as an air conditioning plenum.

- * The floor aids in the natural flow of convective cooling air.

- * It protects data and power cables from damage.

- * Raised floors are cleaner than non-raised floors.

However, raised floors have concerns that must be addressed:

* How do heavy cabinets/peripherals enter and exit the room?

* Air distribution can be blocked by having too many pipes and cables under the floor

* The locations of cables that exit and enter the floor in relation to other equipment must be determined so that they will not be blocked.

* All concrete and drywall must be sealed with concrete sealer to prevent blistering.

Another physical computer room requirement is that the room be treated as an environmental entity. The walls of the computer room should go from the floor to the roof of the building, effectively separating the computer room from the building.

Since computer room noise control is an important consideration, effective steps to reduce noise are necessary. These include using static free carpets, sound absorbing materials on the walls and separating the computer room into noise zones. Equipment that is noisy is grouped in one zone and quiet equipment is grouped in another zone.

ELECTRICAL REQUIREMENTS

In order to determine the electrical requirements for the computer room, these specifications for each cabinet and peripheral are needed as well as their tolerances:

- * Volts
- * Current
- * Phase
- * Plug type
- * Peak Power (Peak Current)

The final specifications should also allow for growth since once the wiring is in place it can be expensive to increase capacity.

All power receptacles for computer equipment should be isolated ground type sockets with grounding occurring at a central point. This central point grounding minimizes ground loops which can induce noise into the system. The local electrical code should however be examined about regulations concerning isolated ground sockets - some municipalities will not allow them.

All power lines feeding the computer should be dedicated to the computer. There should be no other electrical equipment on the line. The main feed line for the computer room should be checked completely

from the distribution transformer of the building to the computer room for other noise producing equipment that might affect computer operations. If there is any doubt about the quality of the power, a power line monitor can be used (rented or purchased) to check for disturbances. This monitor should be allowed to run for as long as possible since some power line disturbances are season dependant. A good example is the summer thunderstorms that occur in some areas of the country.

If a power line monitor reveals noise and power problems on the electrical system, the following solutions can be tried in order of increasing severity:

- * Filters - low cost and easy
- * Constant Voltage transformers
- * Motor - Generator Sets
- * Uninterruptable Power Supplies - High cost and difficult

In any computer room electrical system, it is very helpful to use a Power Distribution System (PDS). These systems provide some filtering but mostly provide isolation. Newer PDS systems, called Power Conditioning Systems (PCS), provide substantial filtering. PDS/PCS systems are useful because they modularize the electrical distribution and installation process. There is only one connection that a licensed electrician must make, thus speeding up the installation. When purchasing a PDS it is best to pick the one with the highest input voltage available. This provides greater noise reduction than using lower input voltage PDS units.

In addition to receptacles for computer cabinets and peripherals, convenience outlets should be included in the design. These are the electrical outlets that will be used for vacuum cleaners and other noise (electrical) producing equipment. Because of this noise, these receptacles should be placed on a different feeder line than the PDS.

Closely related to electrical requirements are lighting requirements. Generally, it is very desirable to use light dimmers in the computer room. These can decrease the heat load placed on the air conditioners. However, some dimmers are Radio Frequency Interference (RFI) sources. For this reason, the particular brand of dimmer should be carefully examined for RFI before it is installed.

ENVIRONMENTAL REQUIREMENTS

In order to determine the environmental requirements for the room, the BTUs of heat for each peripheral must be determined. The total heat load produced by all electrical equipment including lights must be capable of being cooled by the air conditioning equipment. As always, the environmental specifications should allow for growth.

The temperature limits for each peripheral must also be known. All peripherals have two types of limits: static and dynamic. Static limits establish the overall range in which the equipment can operate. Dynamic limits specify how fast the temperature can change per unit time (usually in degrees per hour). Generally disk drives have the most critical dynamic limits because read/write head alignment depends upon uniform temperature throughout the drive.

The air flow direction for each peripheral and cabinet should also be known. This information will determine where to place raised floor vents (if used) and determine whether certain peripherals and cabinets are compatible. The usual flow direction is front to back and bottom to top. The raised floor vents can then be located in order to assist this natural flow of air.

When specifying the air conditioning units, it is usually better to specify two small units rather than one large unit. Thus, if one unit fails, the computer facility can still operate in a degraded mode. Likewise, the larger the computer room is in volume, the more time there is to shut down the system when the air conditioners fail.

In order to minimize contamination, there should be a source of air that will maintain a positive pressure in the computer room. This positive pressure will tend to push dirt and dust out of the room. If a raised floor is used, the concrete slab and drywall underneath it should be sealed with concrete sealer to reduce the number of particles that are produced as the concrete ages.

SECURITY REQUIREMENTS

In choosing the location of the computer room, careful attention must be paid to physical security. Security involves room location, fire suppression, electrical noise, and protection instrumentation. If possible, pick an interior room. Interior rooms are more temperature stable than exterior rooms. They are also less susceptible to external electromagnetic interference (EMI). However, if self contained air conditioners are planned for the room, a room with an external wall(s) becomes necessary.

Due to the high value equipment in computer rooms, all computer rooms should be protected from a potential fire. Smoke detectors should be installed in the room—generally under the raised floor. However if a raised floor is not used then they can be installed on the wall.

Handheld fire extinguishers should be placed near the computer room exits. Thus, if a fire occurs, people looking for fire extinguishers will already be near an exit should they change their mind about fighting a fire. These extinguishers should be

filled with Halon 1211 or 1301. Halon 1301 is less toxic to humans than 1211 but both halons are excellent fire suppression agents. For large computer rooms, under floor self contained halon systems are available.

Ceiling sprinklers are another method of fire suppression. However, since most damage in a fire occurs from water damage and since there is a high electrical shock hazard in a computer room, ceiling sprinklers should be used as a backup to Halon systems. Also, sprinklers should be used that can turn themselves off when the computer room temperature decreases to a set value so that flooding does not occur. Except for ceiling sprinkler pipes, water lines in the ceiling should be avoided. At minimum, they should be kept away from equipment.

Another security concern is electromagnetic interference (EMI). EMI can occur from many sources including welders, motors, heavy industrial equipment and even other computers. The solution to EMI is usually to tie all equipment to a common ground, move equipment away from the source (since EMI strength is proportional to the square of the distance), and surround the computer with copper screen.

Once the security issues have been addressed, different types of detectors can then be interfaced to the computer room Power Distribution System (PDS). These sensors connect to the PDS through the Building Interface Alarm box (BIA). Some of the detectors that can be used are: smoke, fire, water, over/under temperature sensors and over/under voltage sensors. If any of these sensors detects an out of bounds condition, it will trigger a power shutdown of the PDS.

A last concern for computer room security is environmental data gathering. Instruments such as temperature and humidity recorders provide a record of the stability of the environment. Other instruments can provide data on other desired data such as voltage levels.

CONTRACTOR INTERACTIONS

In order to protect the company and the computer room design team, all interactions with the contractor should be conducted through one specific contact. There should also be an alternate contact to serve as a backup should the primary contact be unavailable. Both contacts should be aware of what they legally can and cannot do regarding the construction contract. Other people in the company, while not designated as contacts, can serve very usefully as eyes and ears during the construction process. In this manner, they can keep the construction contacts apprised of information they might not normally know.

In the construction contract, there should be penalties for late completion of

the work. If there are not penalties, construction may drag on for an excessive time.

Finally, during construction of the computer room, disruptions to normal business can be minimized by scheduling the contractor to work at times convenient to the company.

SUMMARY

Computer room construction requires attention to a very large number of details. If motivated people are used on the design team, and these people have access to the proper information, then a successful computer room design will result.

REFERENCES

1. Digital Equipment Corporation, Power Distribution System Technical Guide, 1982.
2. Digital Equipment Corporation, The Power Distribution System Configuration and Ordering Guide, 1981.

Wading through Net.News There's Gold in Them Thar Hills!!

Kurt L. Reisler
Hadron, Inc
9990 Lee Highway
Fairfax, VA 22030
...!seismo!hadron!klr

Abstract

One of the greatest resources available to the members of the UNIX¹ community is an accumulation of special interest "groups" that circulate through net.news. Using the facilities of uucp and USENET, net.news acts as a roughly-structured "gazette", containing vast amounts of technical information (and misinformation), news (and rumors), sources, and bug fixes.

This article will address the methods of accessing net.news, how to make use of it effectively, and examples of the good, the bad, and the really ugly things that can be gathered through net.news.

What is Net.News?

What is generally referred to as "net.news²" is actually a collection of standard Unix utilities and special purpose software, used by a very large³ number of Unix systems for the exchange of

- technical information
- public domain source code
- public domain executables
- bug reports
- "recreational" information
- rumors
- reviews of all sorts
- anything else

Each of these "areas of interest" is usually circulated through the use of designated "news groups". These groups can be either moderated or unmoderated. Distribution of news groups can be local, city-wide, national, or international in scope. Distribution of the "news" is accomplished via "usenet", a loosely coupled network of Unix systems that extends around the world.

In this paper I hope to be able to provide you with an overview of "net.news", why you should get involved with it, and what you need to get started.

¹UNIX is a registered trademark of AT&T Bell Labs

²pronounced as net dot news

³exact numbers change daily

A Brief Historical Perspective

The following is liberally excised from "USENET Software: History and Sources (Last changed: 15 March 1987)" by Gene Spafford. The concept of usenet was started by two graduate students at Duke University of North Carolina in late 1979. Tom Truscott and Jim Ellis thought that hooking up Unix systems with available software would be an excellent way of exchanging information within the Unix community. With the help of Steve Bellovin, a graduate student at the University of North Carolina, they were able to establish the first two usenet sites in early 1980. The software used to drive this initial setup was made up of existing Unix utilities and shell scripts. Eventually this was rewritten in C and was distributed as the "A" release of the news software.

In 1981, the news software was rewritten by Mark Horton (a U.C. Berkeley graduate student) and Matt Glickman (then a high school student). The rewrite was needed to increase the functionality of the software, and to better handle the increasing flow of news across the net. The initial public release of the "B" news software was version 2.1 in 1982. The last release of the "B" software was version 2.10.2, released in 1984.

With the release of the 2.10.2 version of the news software, responsibility for the coordination, maintenance and enhancement of the news software became the responsibility of Rick Adams, at the Center for Seismic Studies. What followed was a restructuring of the news groups, and a new release of the news software as version 2.11. There have been several bug-fix patches for the current version.

The growth of "net.news" has been phenomenal. What

started in 1980 as 2 sites, has grown to an estimated 6,500⁴ sites with an estimated 176,000 active readers. There are currently over 246 news groups.

The Software-Side of Net.News

The current release of the news software makes use of a collection of special programs for reading, distributing and managing the news. A large portion of it is standard Unix uucp⁵ and mail software. In addition to the programs which are written in C, there are a large number of shell scripts.

Standard UNIX components

The news software is based on two standard⁶ components of the Unix operating system, uucp and mail. The uucp components that the news makes use of include uucp, uux, uucico, and uupoll. These programs allow cooperating Unix systems to move mail packets between systems and to remotely execute the appropriate news and mail software. In addition, the news software relies heavily on the Unix file system structure. For further details about the standard Unix components, consult your Unix documentation set.

Special "News" Software

As is traditional with "contributed" software within the Unix community, the news software is distributed in source form. It can usually be obtained from the news site that will be providing you with a news feed, or from the DECUS UNISIG Tape, available through the DECUS Tape Library or LUG distribution tree. When the distribution has been properly customized and compiled, you have a large collection of executables and shell command files. These are the programs that are used to receive, distribute, manage and read the news messages. In addition to the sources, there is considerable documentation (some of it in the source code). The documentation should be thoroughly read, as starting up a new news system is not an adventure for the uninformed.

How Does it all work?

The mechanics of net.news are relatively straightforward. All news "articles" are, in reality, mail messages that are moved from site to site over the usenet. These messages can contain only ASCII characters, but other than that restriction, they can contain almost any sort of information. There are utilities⁷ that allow a user to send non-ASCII messages, such as executables, archives, etc.

When a user posts a news item, it is sent to the user's news host as a mail message. From there it is sent along usenet to each news host that it connects to. In this manner, an article can move around the world, from host to host in a

⁴March 1987

⁵unix-to-unix copy

⁶in as much as anything is standard under Unix

⁷such as compress, uuencode, uudecode

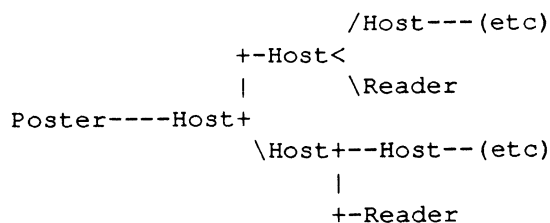


Figure 1: Simplified News Flow

```

Article 2239 of net.sources:
Path: hadron!sundc!seismo!lll-crg!hoptoad!rdm
From: rdm@hoptoad.uucp (Rich Morin)
Newsgroups: net.sources
Subject: sharks - SHell ARchive checKing Script
Message-ID: <1163@hoptoad.uucp>
Date: 30 Sep 86 04:04:07 GMT
Date-Received: 1 Oct 86 09:00:32 GMT
Organization: Santa Forda Computer Laboratory
Lines: 118
Keywords: paranoia shar security trojan horse
```

Figure 2: Sample message header

relatively short period of time. An over simplified version of this can be seen in Figure 1.

The exact path that a news item follows may not always be the most efficient path. In fact, the path that a news item follows is very seldom the same path that a normal mail message to a uucp address follows. The path that a message traveled through can be seen in the message header (see Figure 2). The message header obviously carries a lot of additional information. It indicates which of the news groups the article was included in, the subject of the message, the message ID, the article number in *your* news directory structure, the date the message was posted, the date the message was received, the organization of the poster of the message, the length of the message in lines, and any keywords that the poster may have assigned to the news item. This information may or may not be of interest to the reader and the news reader software can be tailored to show the reader only that portion of the header that is of interest to them.

A map of the usenet "backbone" can be seen in Figure 3. The dashed lines in Figure 3 are normal connections, while the dotted lines are restricted connections. It is along this backbone that news flows to the various distribution "trees". A node that is part of the backbone distribution is referred to as the "root" of that particular distribution "tree". A node that is at the end of the distribution "tree" is referred to as a "leaf" node, while a node that passes the news on to a number of other nodes, is referred to as a "branch" node. As complex as it seems, it all does work—in defiance of the laws of nature and Murphy⁸.

As news packages are received, they are "unpacked"

⁸well, most of the time

comp.ai	Artificial intelligence discussions.
comp.binaries.mac	Encoded Macintosh programs in binary. (Moderated)
comp.bugs.2bsd	Reports of UNIX version 2BSD related bugs.
comp.bugs.4bsd	Reports of UNIX version 4BSD related bugs.
comp.bugs.sys5	Reports of USG (System III, V, etc.) bugs.
comp.databases	Database and data management issues and theory.
comp.emacs	EMACS editors of different flavors.
comp.graphics	Computer graphics, art, animation, image processing.
comp.laser-printers	Laser printers, hardware & software. (Moderated)
comp.mail.uucp	Mail in the uucp network environment.
comp.org.decus	DEC Users' Society newsgroup.
comp.os.research	Operating systems and related areas. (Moderated)
comp.os.vms	DEC's VAX line of computers & VMS.
comp.sources.d	For any discussion of source postings.
comp.sources.unix	Postings of public-domain sources. (Moderated)
comp.std.unix	Discussion for the P1003 committee on UNIX.
comp.sys.dec	Discussions about DEC computer systems.
comp.sys.sun	Sun workstation computers. (Moderated)
comp.sys.workstations	Various workstation-type computers. (Moderated)
comp.text	Text processing issues and methods.
comp.unix	Discussion of UNIX features and bugs. (Moderated)
comp.unix.wizards	Discussions, bug reports, and fixes on and for UNIX.
sci.astro	Astronomy discussions and information.
sci.bio	Biology and related sciences.
sci.crypt	Different methods of data en/decryption.
sci.space.shuttle	The space shuttle and the STS program.

Figure 4: Extract of Technical News Groups

source, and information on hardware and software systems.

sci. Discussions intended as technical in nature and relating to the established sciences.

A listing of some of the names and descriptions of the technical groups is shown in Figure 4. A full listing of the news groups and their descriptions will be published in a future issue of the *DECUS SIGs Newsletters*.

The technical news groups are the most heavily read of all the groups on the net. Of primary interest are the news groups that circulate public domain sources, such as net.sources. In addition, the discussions that take place in groups like comp.bugs.4bsd and comp.os.unix can always be informative. In short, the technical news groups are where you and your programming staff can learn about problems and solutions, as well as obtain actual sources which can be compiled on your own system. Access to the technical news groups can be the biggest positive gain from access to net.news.

Administrative News groups

Since the net.news software is an evolving beast, the maintenance and enhancement of the software and the administration of the network are coordinated through special administrative, or "admin-dot" news groups. The administrative news groups contain discussions of news software problems and solutions, as well as announcements of new news groups. It is a wealth of information for both the neophyte and experienced news administrator.

misc.consumers	Consumer interests, product reviews, etc.
misc.handicap	Items of interest for/about the handicapped.
misc.jobs	Job announcements, requests, etc.
misc.kids	Children, their behavior and activities.
misc.legal	Legalities and the ethics of law.
misc.taxes	Tax laws and advice.
rec.arts.books	Books of all genres, shapes, and sizes.
rec.arts.drwho	Discussion about Dr. Who.
rec.arts.movies	Discussions of movies and movie making.
rec.arts.poems	For the posting of poems.
rec.arts.sf-lovers	Science fiction lovers' newsgroup.
rec.arts.startrek	Star Trek, the TV show and the movies.
rec.arts.tv	The boob tube, its history, and past and current shows.
rec.arts.wobegon	"A Prairie Home Companion" radio show discussion.
rec.food.cooking	Food, cooking, cookbooks, and recipes.
rec.photo	Hobbyists interested in photography.
rec.skiing	Hobbyists interested in skiing.
rec.sport.football	Discussion about football.
rec.video	Video and video components.
talk.politics.theory	Theory of politics and political systems.
talk.religion.misc	Religious, ethical, & moral implications.
talk.religion.newage	Esoteric and minority religions & philosophies.
talk.rumors	For the posting of rumors.

Figure 5: Extract of Non-technical News Groups

Non-Technical News groups

In addition to the technical news groups, there are a large number of non-technical news groups circulating on the usenet. Such groups are now being designated with a variety of news prefixes:

misc. Groups addressing themes not easily classified under any of the other headings or which incorporate themes from multiple categories.

soc. Groups primarily addressing social issues and socializing.

talk. Groups largely debate-oriented and tending to feature long discussions without resolution and without appreciable amounts of generally useful information.

news. Groups concerned with the news network and software themselves.

rec. Groups oriented towards hobbies and recreational activities.

A sampling of the "non-technical" news groups can be seen in Figure 5. These groups are home to discussions about movies, TV, cooking, cars, photography, video gear, politics, etc. It is the non-technical groups that cause some heartburn to a few system managers, who feel that they have "no place" in a "technical" network. In reality, most programmers spend a relatively small amount of their "news time" with a large number of the non-technical news groups. They usually settle down with one or two of these as an alternative to the technical discussions of the technical and administrative news groups.

Effects on Productivity

Access to net.news can have a very beneficial effect on overall programmer productivity. It give the programmer access to a huge number of technical resources outside of their immediate office. A majority of the sources that are circulated via usenet can be used to enhance the programmer's job. In addition, a lot of the bug fixes and system enhancements can be used to correct problems and generally make the use of a local system more pleasant. Net.news is also an avenue for getting "how do I" questions answered.

What can Net.News do to you?

There are a number of drawbacks to receiving net.news. In my opinion, the advantages far outweigh the disadvantages, but in attempt to stay even-handed, I will present some of them.

Effects on Productivity

Yes, reading of net.news also has a negative side on productivity. Reading the news takes time. Depending on the volume of news groups being read, it can take a lot of time. I would guess that most programmers spend 1 to 2 hours a day reading the news. This may be from home or during lunch, or on company time. The time may be higher on a Monday or after returning from a holiday or business trip¹⁰. There is also the time spent reviewing, compiling and testing programs that are distributed via the source news groups. But, there is a trade-off here in terms of clock time lost, versus productivity gains¹¹. In many cases, techniques that are learned through the review or use of source code distributed through the net can enhance the knowledge, technical skills and productivity of programmers.

Questionable Software

The problem of the "Trojan Horse" program is an on-going one in the net.news world. The problem is multiplied by the circulation of executable binaries rather than sources in some of the news groups. A Trojan Horse program is one that claims to do one thing, usually useful, but in reality does something quite different. In many cases, the these programs will either damage files on your system, make attempts to compromise system security, or just be annoying¹². There are many ways of dealing with a potential Trojan Horse program. If it was obtained in source, than these should be reviewed before the program is compiled and run. If it is a binary, you should wait a few days before you try to run it. Usually such pranks are found and widely broadcast on the net. As with all public-domain software, caution is the best protection.

It should be noted that the number of Trojan Horse programs that are distributed across the net is actually very small. The number of prank programs seems to peak around April Fool's Day for obvious reasons. However, it should also be

pointed out that for each of the "good" programs that are received, there are a number that just plain don't work, and are usually not worth the effort to make work.

Religious Wars

Getting connected to net.news, opens up the possibility of getting involved in a number of technical and non-technical discussion groups. Occasionally, these degenerate into "Religious Wars" which can be time consuming. Religious wars usually revolve around such major topics as:

- editors
- versions of Unix
- spelling and grammer
- programming techniques
- network policy
- news groups naming conventions

Not much good ever comes of the religious wars on the network. They are seldom if ever resolved, and on occasion have resulted in personal attacks (non-physical) that have resulted in individuals unsubscribing from certain news groups.

System effects

Getting involved with net.news can have a very definite effect on your system itself. You can expect a dramatic increase in disk usage. This is due to the news software itself, the news articles as they come in, the archived news articles as they are "aged", and the copies of news articles and software that are being stored by the readers of the news on your system. In addition, you can expect an increase in modem usage on your system as your news host provides you with the days articles, and as your users post responses and new articles. Finally, if you are receiving a "full news distribution¹³", you can expect to be receiving approximately 1.5 megabytes of news a day.

Conclusions

Net.news is a user maintained network that helps to keep the users of Unix systems in touch with each other. News "articles" are actually messages that are sent between cooperating Unix systems, using standard uucp software and additional "news" software. These messages can contain a variety of information, including bug reports, bug fixes, source code, executables, movie reviews, rumors, jokes and just about anything else that can be entered into a computer. Although there is some cost to participating in net.news, in terms of phone bills, system performance and programmer productivity, it is felt¹⁴ that the gains far outweigh the costs.

One thing has become evident in making presentations about net.news at DECUS sessions. There is great interest in

¹⁰this is know as "catching up" with the news

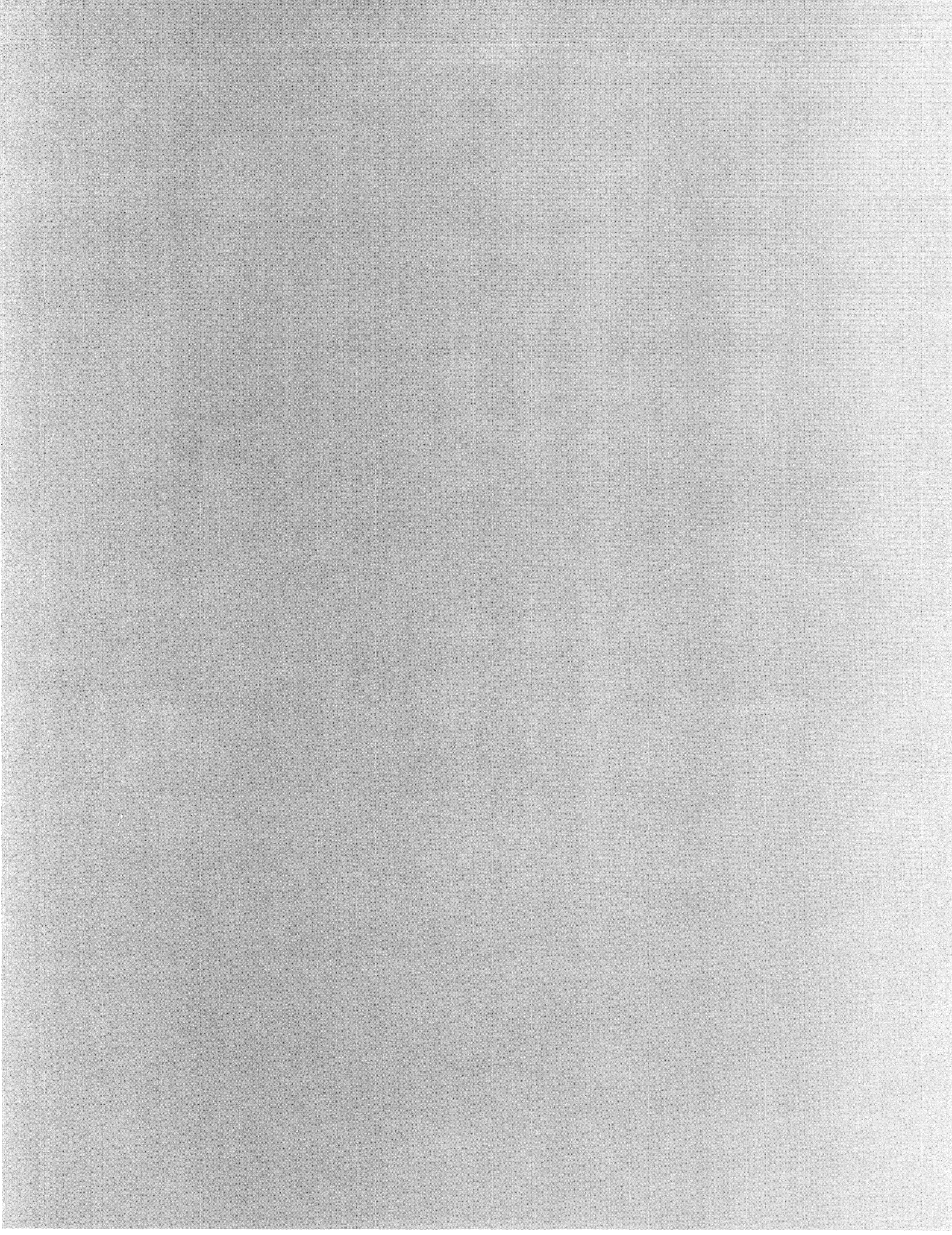
¹¹"What? Me Biased?"

¹²Anything received around 1 April is *ALWAYS* suspect.

¹³i.e. all of the available news groups

¹⁴at least by this author

getting access to net.news on the part of the attendees. They know where to get the news software, and they are able to get it installed. The information that is lacking is "who to contact to obtain a news feed." To attempt to correct this situation, I have posted a message in the "news.admin" news group, requesting that news site administrators who can provide "feeds" for new news sites contact me. I am in the process of compiling this information into a list that will be published on a regular basis in the *DECUS SIGs Newsletter*, as well as a few of the DEC-oriented commercial publications.



INTEGRATION OF INPUT/OUTPUT DEVICES USING SILICON COMPILATION

Dr. Robert Couranz, Edwin Rogers, Laurence Specter
Raytheon Company
Sudbury, Massachusetts

ABSTRACT

The current trend in military electronics, especially computers, is to exploit to the maximum extent possible existing commercial designs. Although the commercial computer architectures and associated implementation technologies have significantly improved performance and reduced the size of the core of the computer, the I/O functions must now be addressed. In many cases, the I/O requirements of the militarized computer are quite different than those of the commercial counterpart. Thus, it is necessary to develop highly integrated I/O designs to compliment the level of integration of the remainder of the computer. This paper addresses that problem and provides a case study where a silicon compiler was used to rapidly design an ASIC for a specialized military interface.

1 THE NEED OF THE MILITARY MARKETPLACE

1.1 Time to Meet Customer Requirement

The military market is not unlike the commercial market, in that time is a key issue. In the commercial environment, time-to-market is tied to when a manufacturer needs to release a product to meet market and company needs or in response to competitive pressures. Normally, the product will be sold to users having a range of applications that the product was designed to address.

The military time-to-market is first driven by the interval between contract award and the delivery of the first form, fit, and function unit. Then a second interval, that between acceptance of qualification units and production comes into play. Prior to contract

award there has been extensive proposal effort expended and design at the system and subsystem level has been completed. Also, critical areas of hardware have been proven and a high confidence in the ability to produce the more basic elements of the system or subsystem has been developed. Still, the detailed design of the complete deliverable unit is held until contract award because the cost of design is part of contract.

1.2 Limited Production Run is Standard

When a commercial vendor such as a computer manufacturer introduces a new product to the market, the anticipation is that large quantities will eventually be produced. With the exception of some selected items such as munitions, the production run for any given military product is usually limited. A radar system might number less than 100 units. Aircraft of a given type might reach

several thousand but over many years of production and various system revisions. Thus the electronics complement across the production run may change considerably. Computer subsystems associated with these larger systems must be configurable to support many production buys in quantities of 10 to 100. Thus, the ability to configure the processor through the development of standard and custom I/O interfaces becomes critical.

1.3 Environmental Limits Range From Commercialized Military to Space Qualified

Based on initiatives of the Defense Science Board, the armed services are trying to develop specifications that reflect the true operating environment that will be seen by the equipment being procured. Studies have shown that equipment in the field will see varying environmental requirements as a function of how close to the actual battlefield the equipment is to be employed.

One approach to the low severity environment problem is to take commercial hardware and repackage the boards in a more rugged, air cooled chassis. This approach suffers from the lack of a solid configuration management strategy. The organization repackaging the commercial components can only supply a ruggedized version of the commercial module. In addition, the air cooled chassis exposes the boards, components, and connectors to the contaminants in the cooling air such as high humidity and salt air. Finally, there is no upgrade path in the same form, fit, and function if the system environmental requirements are increased.

Alternately, a commercialized military unit provides a full military conduction cooled package designed to meet the rigors of the full environmental range and all applicable military equipment specifications such as MIL-E-5400 and

MIL-E-16400. Because the chassis is conduction cooled, the contaminants of the atmosphere never reach the components, connectors, or circuit boards. Configuration management is under control of the manufacturer since the basic module is built, not procured. Finally, the cost of the mechanical package is a minor part of the overall unit cost. Thus, the unit cost can be made consistent with the environmental specifications by the selection of the real cost driver, the semiconductor component quality. Should the environmental requirements be increased, the units can be upgraded by exchanging modules.

1.4 Technology Transfer

Traditionally there has been a significant lag between the development of a product for a commercial user and the availability of the same product capability in the military market. This is especially true in the area of computer electronics where the commercial vendors are pressing the state-of-the-art for every bit of performance and functionality available to capture market share. The first step in translating the best technology to the military user is to be able to capture commercial technology and make it available in commercialized military to full military quality. Just capturing the basic commercial processor does not allow the application of the technology to the military problem. Military I/O and peripherals must be provided to make the hardware useful.

2 MILITARY VAX(R) COMPUTER FAMILY

The Military VAX[®] Computer Family consist of a range of processors of graduated size and capability providing the right level of computing power for any specified application. After meeting stringent Raytheon military specification requirements, the computer's performance will be validated by DEC(R) before they wear the VAX label (Figure 1).

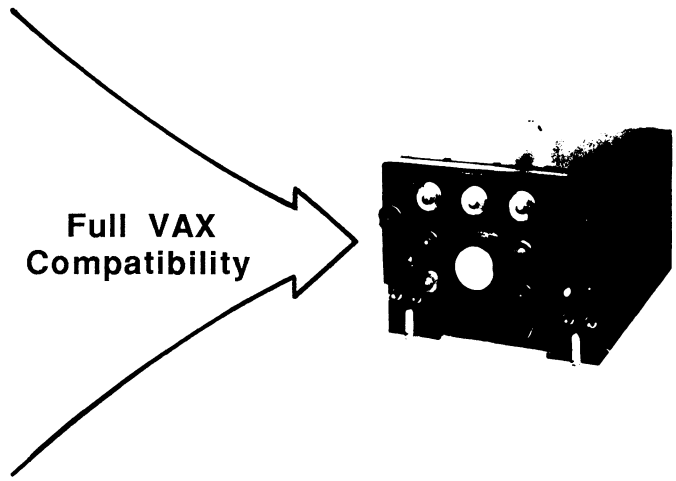
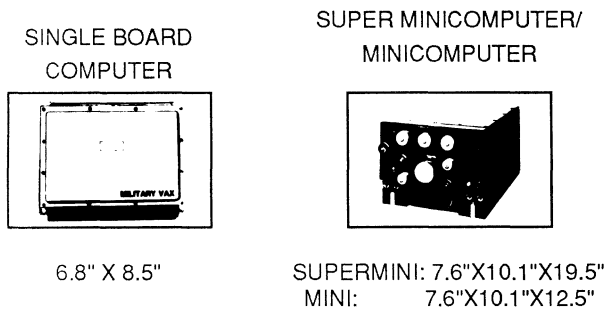


Figure 1. Military Computer Family VAX[®] Compatibility

The smallest member of the family is the Model 810 Single Board Computer which is intended for deeply embedded, specific application intensive environments. It consists of a CPU with floating point, a limited amount of on-board static RAM and EPROM, and two serial channels. Both the I/O and memory bus are available for incorporating additional I/O or memory (Figure 2).

The Model 860 Supermini computer is a full ATR Long and supports one or two of the Model 830 CPUs. It also supports up to 32 Megabytes of DRAM, two I/O busses, and six I/O slots with provisions for I/O expansion. It is intended to support large data management applications or smaller applications which anticipate significant growth.



All Militarized Vax[®] Computer Family products are built in two versions; a reduced cost mil-spec implementation using high quality commercial grade components and an extreme performance version using Mil-Std 883B or better components. All models of the MVCF will support the DEC VAXBI[®]. Thus, I/O modules are being designed for application across the complete product line.

Figure 2. Physical Characteristics Of The MVCF

3 IMPLEMENTING THE MIL-STD-1553B INTERFACE MODULE

The Model 830 Minicomputer contains a CPU with floating point and 256 Kbytes of cache memory. It supports up to 16 Megabytes of DRAM, two bus adapters, and three internal I/O slots. The Model 830 is packaged as a full ATR Short and is well suited to large real-time applications or small data management requirements.

The MIL-STD-1553 interface is a serial bus employed by all branches of the armed forces. As a result, it is a natural interface to be selected as one to be offered as part of the MVCF I/O group. Since the basic processors employ high levels of integration, specifically 1.25 micron CMOS devices from Raytheon's Microelectronics Center, it becomes mandatory to produce I/O elements that are configured to the

same level of integration. In order to reach the level of integration required and to minimize time-to-market, the development of application specific integrated circuits appeared necessary.

3.1 Requirements Placed On The MIL-STD-1553B Module

The functional requirements for a 1553 bus are set forth in Military-Standard 1553, Revision B, Notice 2. An example

of the application of the 1553 bus structure as applied to a hypothetical signal intelligence gathering application is shown in Figure 3. This architecture uses a rugged two wire data transmission path for high reliability and low weight, and dual data buses (or channels) for redundancy. The primary requirements for Raytheon's 1553B (R1553B) are for a 1 Million bit per second serial data stream with start-of and end-of message indicators, a device

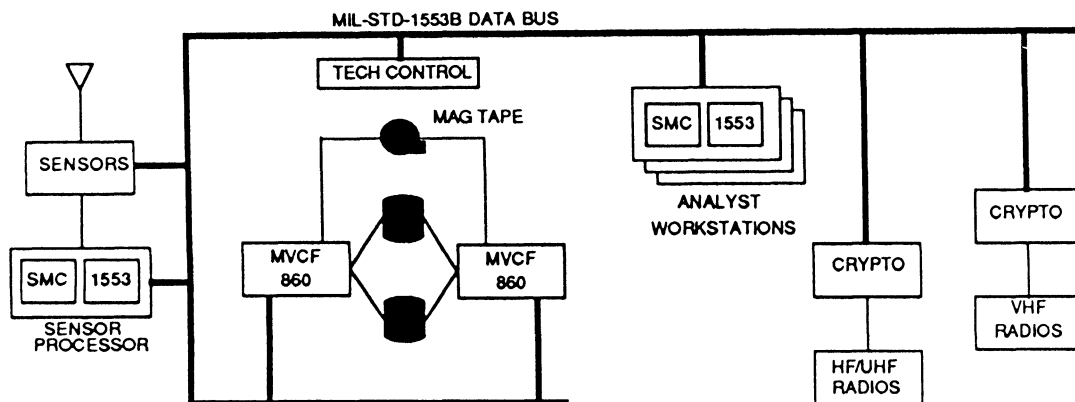


Figure 3. Example of MIL-STD-1553B In A Signal Intelligence System

address mechanism, and device control and status mechanisms. Other requirements include error detection of single bit message errors, a message broadcast capability, and a token passing mechanism for rotating bus control.

3.2 Design Of The Module

The R1553B provides an intelligent interface (Figure 4) between the 1553 data bus and Raytheon's militarized VAX Backplane Interconnect (VAXBI)[®]. The R1553B is configured as dual redundant data buses (1 active and 1 standby), and can, under program control, act as a bus controller (BC) or a remote terminal (RT). A BC sends and receives commands, status, or data while an RT accepts commands from a BC and also sends and receives status and data.

3.3 Functions To Be Performed

The R1553B performs at the full 1 Megabit per second serial data rate of the 1553 which translates to approximately 93,500 bytes per second data rate on the VAXBI[®]. The R1553B further off-loads the CPU by providing a 128K byte message buffer for storing multiple messages (up to 128) and also schedules host programmable functions such as auto-retry. The R1553B also handles interrupt latencies by making use of an interrupt history list by sequentially storing interrupts.

The R1553B implements a programmable interrupt feature for handling conditions such as message errors, end of message list, remote terminal busy, auto-retry failure, and self-test failure. On-board built-in-test

circuitry supplemented by CPU diagnostics allow detection of greater than 90% of all failures. The R1553B is implemented on a single multilayer printed-circuit card assembly employing low-power surface mounted components.

3.4 Selection of Elements/Functions To Be Integrated Into The ASIC

The design shown in Figure 4 can be implemented with three classes of integrated circuits, LSI, memory, and small scale and medium scale "glue"

logic. The 1553 and VAXBI[®] interface circuits are available as off the shelf Military LSI and do not offer advantages for further integration. The same can be said for the memory devices used in the design. This leaves the Data Path Logic, DMA Controller and VAXBI[®] Interface Controller which are comprised of SSI and MSI functional elements and offer advantages in size, weight, and power when integrated as a higher level function. The data path and DMA (DPDMA) function has been selected here for further description. The DPDMA

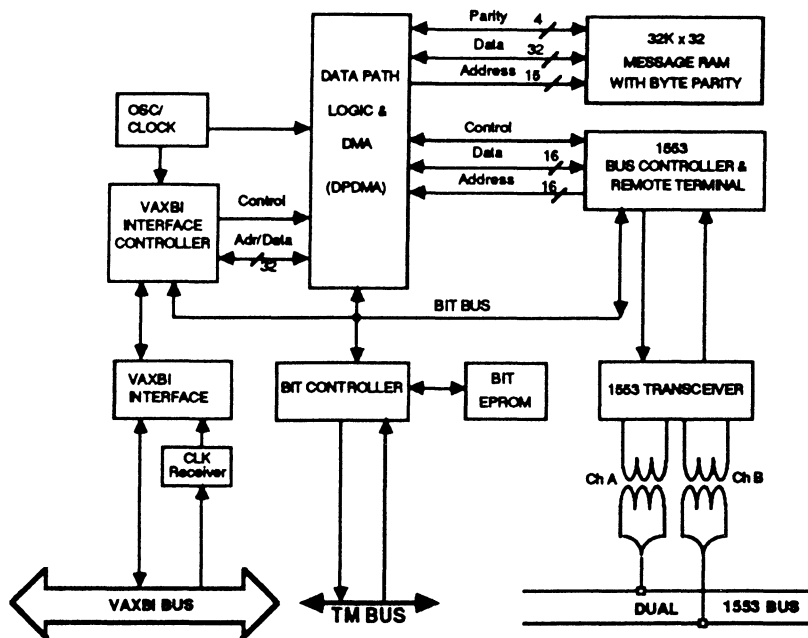


Figure 4. R1553B VAXBI[®] Interface Adapter

generates and checks the message RAM parity logic for the message RAM, multiplexes and demultiplexes the 16-bit 1553 bus controller data path with the 32-bit message RAM and VAXBI[®] data paths, and provides address registers, word count registers, and sequencing logic for the DMA function. In addition, the DPDMA circuit resolves memory access conflicts between the 1553 and VAXBI[®] with arbitration logic. It also has BIT for self-test of the chip.

3.5 Sizing Of Module In Terms Of Surface Area Required

A discrete (non-LSI) version of the DPDMA function can be implemented with approximately 70 SSI and MSI circuits and occupy an area of 81 square inches. With an available board area of approximately 70 square inches, the discrete implementation would not fit on a single module. The options to the designers are to develop an interface that requires two modules, reduce the

functionality of the design, or to introduce an Application Specific Integrated Circuit (ASIC). The CPU has been reduced to two modules through the extensive use of VHSIC level technology. It would be unrealistic to produce I/O modules of much lower complexity on the same or greater form factor as the CPU. Thus, the judicious choice of functions to be integrated into an ASIC is the only viable option.

3.6 Compiler Design Procedure

The VLSI compiler implementation of the 1553 interface device originated from a requirements specification consisting of an architectural block diagram and system timing requirements. A preliminary feasibility study was first performed to determine the approximate silicon die size and I/O count required. This was accomplished through an exploratory design cycle using the Silicon Compilers Inc GENESIL[®] compiler function set. By trying different circuit configurations, an optimum architecture was achieved. The selected logic blocks plus and estimated routing area resulted in a projected die size of 324 x 324 mils using a VHSIC 1.25 micron CMOS technology.

Having shown design feasibility, the actual silicon compiler implementation began by first selecting a design hierarchy that would allow for layout optimization of critical blocks. The design entry was accomplished using a high level specification format that is unique to the silicon compiler environment. As each block specification is completed, a layout is performed and then optimized to meet system requirements. When all lower level blocks are routed, the chip level floorplan is performed. This step determines the I/O pad placement and global chip routing strategy.

3.7 The Resultant Design Concept

The first pass of the 1553 chip produced a device with a die size of 320 x 350 mils. (Figure 5) The device contains 28672 CMOS transistors implemented in a VHSIC 1.25 micron technology. The estimated power dissipation is 250 mw. The device will still have further layout optimization, timing analysis, and functional simulation before the final product is released for silicon fabrication. Of note is the fact that the time interval from the definition of the functions to be integrated into the ASIC to the point where device simulation could be started was approximately two man weeks.

4 IMPACT OF VLSI ON THE OVERALL INTERFACE DESIGN

The advantages of using a VLSI replacement for the DPDMA function is shown in Table 1. The comparison shown is for the entire circuit card assembly.

	DISCRETE	VLSI	CHANGE
Total Number of ICs	119	49	-59%
Power (Watts)	16	8.7	-46%
Area Used (Sq. In.)	81	47	-42%
Reliability (MTBF)	34,000	50,000	+32%

Table 1. Comparison of VLSI and Discrete Implementations

As can be seen from Table 1, the use of the compiled ASIC would make major positive contributions to the module design. The primary issue is the reduction of the surface area required to

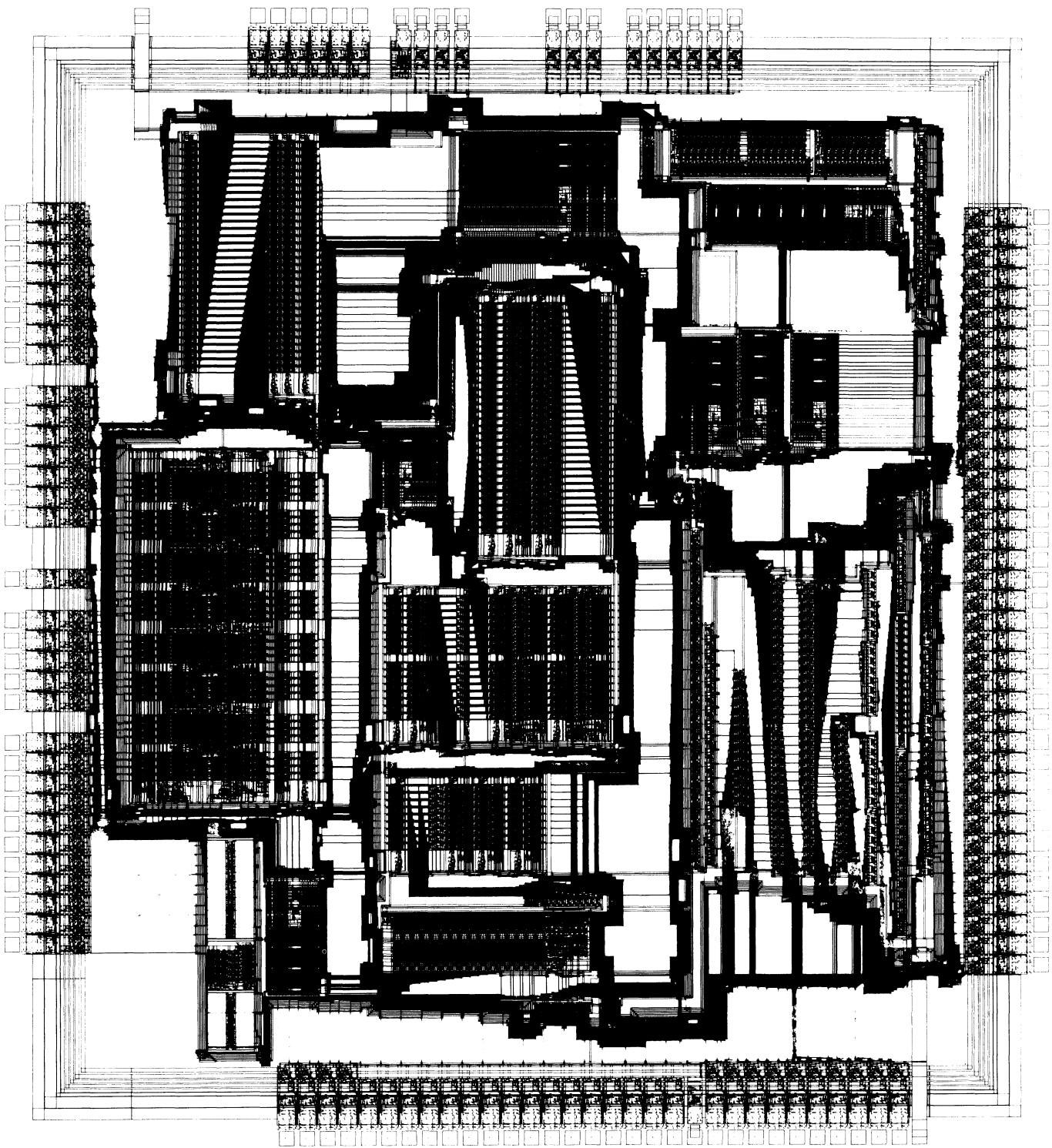


Figure 5. ASIC for MIL-STD-1553B Interface

implement the specified functionality to the size that will fit on a single module. Two other major benefits were fallouts from the ASIC application were the decreases in power and increase in computed MTBF. These both were the result of the reduced number of integrated circuits required for the function.

The integration of computer electronics has been largely directed toward the central processing unit and memory. As

the level of integration of these units increase, it becomes a necessity to increase the integration of I/O devices. The benefits reported in this paper are believed to be indicative of those that can be achieved in both commercial and military systems.

DEC, VAX, and VAXBI are trademarks of the Digital Equipment Corporation. GENESIL is a trademark of Silicon Compilers, Inc.

Defending against Trojan Horses, Viruses and Worms

Robert A. Clyde
Clyde Digital Systems
Orem, Utah

ABSTRACT

Trojan horses, viruses and worms are a potential threat to any system. Mandatory access controls under VMS would provide a partial defense against them. This defense could be strengthened by increased security awareness, integrity checking, surveillance, and analysis of the surveillance data.

The Enemy

This paper focuses on attacks on a computer system by trojan horses, viruses and worms. Viruses and worms are extensions of the trojan horse theme. These attacks center around a seemingly innocent program that contains covert logic. When this program is invoked by an unsuspecting user it tampers with objects that the user can access so that the intruder can achieve some ultimate goal. A good way to understand the relationship between trojan horses, viruses and worms is to view them in the context of a concerted attack using all three.

A prospective intruder creates a trojan horse--a program containing covert logic--and entices other people to use it. The trojan horse's covert logic also contains a virus. When an unsuspecting user executes the trojan horse, the virus spreads itself to other programs to which the user has write access. Furthermore, each time one of the infected programs runs it also spreads the virus. Each virus also looks for an opportunity to insert a worm into a particular system program. This worm contains the necessary logic to allow the intruder to penetrate the system at will. For example, the worm might be placed in the LOGINOUT program so that whenever the intruder types a certain sequence for the password he is logged in with all privileges.

So in summary we have the following relationships:

- Trojan Horse - Contains covert logic and can introduce a virus via unsuspecting users.
- Virus - Spreads the virus to other hosts (e.g., programs), thereby breaking down the defenses of the system so that eventually a worm will be inserted. The worm will be inserted when someone with write access to the targeted system program runs an infected program.
- Worm - Penetrates the security of the operating system.

Note the insidious nature of such an attack on your system. Removing the worm that allows penetration will not provide a complete remedy since the virus will be active and will eventually insert another worm. Complete recovery from such an attack will require

neutralizing the virus. The purpose of this paper is to discuss measures that could be taken in order to prevent and detect such attacks.

It is possible to have a trojan horse without a virus and a virus without a worm. For example, if the goal of the intruder is simply access to certain files rather than penetration of the operating system, then a virus or trojan horse alone will suffice. Also, note that an initial trojan horse is the key to the attack. Methods that will combat trojan horses will be effective in combatting viruses and worms as well.

Security Awareness

Increased security awareness on the part of system managers and users must be the initial focus for controlling this and most security problems. In particular, users and system managers should be properly suspicious when presented with a *gift horse* program. If there are any questions at all, the source code should be reviewed by a reliable expert.

Users should periodically check the protection codes and access control lists set on their files to see if there have been any changes. Generally, executable programs should be set so that there is no write access. This will help limit the spread of a virus.

Terminals must be locked up or logged out when not in use. If this is not done, an intruder can avoid the use of a trojan horse and directly insert a virus or a worm via a terminal left logged into someone else's account. For example, if a user with SYSPRV leaves his terminal logged in, an intruder could insert a worm directly into the LOGINOUT program from that terminal. In reality unattended logged in terminals probably account for more security breaches than trojan horses. A terminal should be locked in cases where logging out would be difficult or very inconvenient. Although Digital's terminal server has a locking mechanism, VMS does not. Nevertheless, it is possible to write a program to perform terminal locking at the operating system level.

Discretionary Access Controls

The standard VMS operating system comes equipped with discretionary access controls in the form of access control lists and protection codes that can be set on most objects. These access controls are known as *discretionary* since the owner of the object is able, at his *discretion*, to modify the access control lists and protection codes.

Because of this, discretionary access controls are not an effective defense against trojan horses, viruses and worms. Consider the following trojan horse scenario:

1. A user named JOHN writes a game called XTREK and sets its protection so that anyone can execute it.
2. JOHN places covert logic in XTREK so that it sets any files to which the user has access to world read and write.
3. JOHN then sends a mail message to everyone on the system proclaiming the wonders of his XTREK program and inviting all to try it.

4. SAM reads the message and decides to run XTREK. The game runs fine, but also changes SAM's files so that anyone can read or write to them. XTREK is able to do this since VMS allows the owner of the files to modify the access controls.
5. Now JOHN is able to read and write SAM's files.

Mandatory Access Controls

Historically, mandatory access controls have been touted as the primary defense against trojan horses, viruses and worms[1]. In practice mandatory access controls are set in place by the system security officer--they cannot be modified by normal users. If SAM owns files which have had mandatory access controls placed on them so that only TOP SECRET users can read or write to them, then SAM cannot lower the files' classification to allow users at the SECRET level to access them. So if SAM runs the XTREK program it will be unable to lower the classification on TOP SECRET files. Thus, the mandatory access controls have defeated the trojan horse in this example.

VMS does have a latent capability for providing mandatory access controls[2]. Digital has recently announced a security product which activates those controls[3]. Nevertheless, mandatory access controls are not a cure-all.

VMS with mandatory access controls contains numerous covert channels[1]. Covert channels are communication channels inherent in the system which were not originally designed as such. Covert channels are only significant on systems which have mandatory access controls and which are running multiple security levels. Therefore, a trojan horse in a program run by a TOP SECRET user could use a covert channel to transmit TOP SECRET information to an intruder at the SECRET level. So for example, if VMS allowed the SHOW USER command to be issued by users at any level, the process name field could be used as a communication channel. Thus the trojan horse in the XTREK program could change its process name to contain TOP SECRET data. JOHN at the SECRET level could then read this data by issuing a SHOW USER command.

The method for handling the covert channel threat is to

1. Identify as many of the covert channels as possible.
2. Remove as many covert channels as possible.
3. Monitor the remaining covert channels.

Also, how mandatory are the mandatory access controls on VMS? If on a VMS system there are privileges (e.g., BYPASS, DOWNGRADE, and UPGRADE) which are able to bypass the mandatory access controls, then when a program with a trojan horse is executed by a user with those privileges it could access protected information and make it available to an intruder. Therefore, a complete penetration scenario involving viruses and worms, like the one described earlier involving LOGINOUT, would give the intruder all privileges--including the BYPASS and SECURITY privileges.

Controlled Program Creation

Another method for reducing a system's exposure to trojan horses, viruses, and worms is to restrict the insertion and creation of programs on the system. This may involve only

acquiring software from known, reliable sources. On particularly sensitive systems it may be necessary to perform a source code review and certification before placing an outside program on the system.

In conjunction with this it may also be necessary to restrict the creation of executable programs. This can be partially accomplished by controlling access to the various compilers, assemblers and linkers on the system. However, executable code could still be downloaded from a PC or some other system. By monitoring terminal input it would be possible to detect downloading of executable code and flag this as a potentially suspicious event.

Integrity Checking

Trojan horses, viruses and worms function by compromising the integrity of programs and files on the system. Consequently, their presence may often be detected by checking to see if any programs have been changed or if any file protections have been modified. Such an integrity check could consist of the following:

1. Compare the protection codes and ACLs of system files to a previously determined standard.
2. Check for viruses and worms by comparing system programs and files to a previously determined standard (i.e., perform a CRC).
3. Compare the protection codes and ACLs of all files to a previously determined standard (i.e. a system backup from an earlier time period).

While these procedures will detect that system integrity has been compromised, they do not provide sufficient information for identifying the source and method of the initial intrusion. Collecting such information requires surveillance.

Surveillance of System Use

As indicated by the earlier sections, it may be difficult to totally prevent the insertion of a trojan horse onto a system. Nevertheless, a vigilant system security officer can employ surveillance in an effort to discover one of the following conditions:

1. Original insertion of a trojan horse, virus or worm.
2. Abnormal use or access as a result of a trojan horse, virus or worm making certain files or services available.

The following are potential sources of surveillance data:

1. VMS security alarms.
2. VMS accounting log.
3. Monitoring terminal I/O.
4. Monitoring system service use.

Only the first two sources are inherently available with VMS[4]. The other two would require additional system-level programming.

Once surveillance data has been collected, it must be analyzed. This can either be done manually or automatically. If it is done manually, the volume of data would most likely preclude any type of review other than spot checking. A computerized analysis, on the other hand, could greatly reduce the burden to the system security officer.

Analysis of Surveillance Data

The purpose of collecting and analyzing surveillance data is to detect any type of suspicious activity--not just trojan horses, viruses, and worms. Nevertheless, this method should also be effective against these particular intrusions into the system. (Of course, the surveillance system must have tamper resistant mechanisms of its own.)

For example, the VMS alarms could flag any use of the AUTHORIZE program. A security officer knowing who is supposed to be able to run AUTHORIZE would then be able to recognize that AUTHORIZE had been run by an intruder. (Note, however, that if the intruder acquired the SECURITY privilege, he could disable the alarms before running AUTHORIZE.)

Monitoring system service requests would make it possible to detect such things as:

- Increases in privilege level
- Use of executive and kernel mode

The analysis program could search through the monitored terminal I/O and perform pattern matching in order to detect such things as follows:

- Browsing through directories
- Execution of AUTHORIZE, SYSGEN, INSTALL, etc.
- Displaying of sensitive information
- Downloading of executable code

Perhaps the most important benefit of monitoring terminal I/O is that it provides the system security officer with a complete record of what a particular user did at a terminal. Thus if a user entered a trojan horse via the terminal, this action would be recorded. If a trojan horse has made certain sensitive data available, the terminal surveillance would contain a record of what the intruder did with that data. Thus, the terminal surveillance data may constitute valuable evidence if disciplinary action or prosecution becomes necessary.

Conclusion

Trojan horses, viruses and worms function by compromising the integrity of programs and files on the system. A concerted attack consisting of all three can be particularly troublesome. Although mandatory access controls provide some defense against trojan horses, viruses and worms, they may not provide a sufficient defense. Surveillance and integrity checking may be implemented on a VMS system with or without mandatory access controls. The use of surveillance coupled with integrity checking can provide a potent defense against trojan horses, viruses and worms.

References

- [1] U.S. Department of Defense. DoD Computer Security Center. *Department of Defense Trusted Computer System Evaluation Criteria*, CSC-STD-001-83 (Aug. 15, 1983).
- [2] Blotcky, S., Lynch, K. and Lipner, S. "SE/VMS: Implementing Mandatory Security i VAX/VMS." *Proceedings of the 9th National Computer Security Conference*. Gaithersburg, MD: September 1986, pp. 47-54.
- [3] Shannon, Terry C. "DEC Tries to Make VMS Even More Secure." *Digital Review*. November 10, 1985, pp. 60-61.
- [4] Digital Equipment Corporation. *Guide to VAX/VMS System Security*. Maynard, Massachusetts: July 1985.

Created March 12, 1987

VAX Systems Coexisting in a Multivendor Environment

Robert C. Groman

Woods Hole Oceanographic Institution
Woods Hole, Massachusetts 02543

Abstract

VAX systems are very popular; however, there are many other computing options available including scientific workstations, super minicomputers and parallel processing systems. Networked personal computers are also available to the computer center and department level projects. This paper reviews the issues in designing, implementing and supporting a multivendor, networked environment.

Introduction

This paper reviews the issues facing the system manager and systems support group responsible for designing, implementing and supporting a multivendor environment. Computer networks, tying together both large and small computers from different companies, offer the computer user a rich diversity of computing options. The integration and smooth operation of these networks pose a number of problems that must be resolved if the interconnected systems are to work well together.

Background

The Woods Hole Oceanographic Institution, a private, non-profit corporation doing basic research and education in oceanography and related fields, offers a good opportunity to study the impact of a multivendor computing environment. The Institution consists of 300 researchers and technical staff who acquire their own funding through competitively sought grants and contracts. They work within one of the five departments: Biology, Chemistry, Geology and Geophysics, Ocean Engineering and Physical Oceanography. An Administration group is responsible for keeping things together, but since it is the researchers that provide the funding, these researchers are ultimately *in charge*. (The Institution has been described as a controlled anarchy, a fitting phrase.) As an example of researchers independence, there are over 400 personal computers at the Institution representing 38 different models and vendors. The Administration's rationale for this is that in a research and development environment, too many *standards* can do more harm than good. This view has merit, although not without some negative consequences.

The Information Processing and Communications Laboratory within the Department of Ocean Engineering is responsible for providing computer and data communication expertise and for operating the general and some project specific VAX computer systems at the Institution. The facility consists of a VAXcluster (8800, 785, 780, 750), a separate VAX 8250, several microVAX II based systems, and hundreds of personal

computers. The VAX systems, running VMS, offer a range of compilers and software packages including Fortran, C, Pascal, SPSS, SAS, UNIRAS, DISSPLA/TELLAGRAF, IMSL, ILS and SMP. Researchers use the clustered VAXes for computer modeling, data processing and graphics displays. A broadband coaxial cable (SYTEK Localnet 20) runs throughout the Institution (the two campuses are located about a mile apart) and provides 9600 baud terminal access to the computers.

Distributed Computing

At first, Institution members wondered whether the *central computer model*, i.e. a central computer group providing computing resources, would survive the arrival of personal computers (PC) and project specific machines such as the UNIX based Ridge, Sun systems and microVAXes. However, their arrival increased the demand on the central computer! One reason for this is that the PC's provide an opportunity for people to learn what computers can do for them. As applications outgrow PC's, they migrate to the larger central machines. Of course, the central site computers also must change with the times. They must be replaced by faster, more cost effective components if they are to continue to be competitive with the powerful, new personal computers and workstations.

Some scientists also need access to the higher end systems like the Cray X/MP and Cyber 205 class machines. These machines were accessed via dial-up leased lines either directly or via a local machine acting as a remote batch terminal using the 3780 protocol. The researchers usually have compute and memory intensive applications that take weeks to run on a machine the speed of a VAX-11/780. Routinely running week long jobs becomes impractical due to the long run time and the absence of an uninterruptable power supply.

Another motivation for using a supercomputer can be cost. Recently, the National Science Foundation established the supercomputing centers which offer free computer time to qualified NSF funded projects. Cost is also a strong motivating factor for acquisition of personal computers, workstations and project specific machines [2]. Funding agencies appear to

be less willing to provide money for computer time [1]. This puts pressure on researchers to look for the least expensive compute cycles available, including supercomputers and the newer minisupercomputers such as from Alliant, Convex and others.

Computer System Evaluation

The computer market is constantly changing and that makes it difficult to follow and understand. The lifetime for computer hardware technology is said to be less than three years. It is no wonder that computers become outdated almost immediately after their introduction. This allows computer vendors to leap frog one another with better, faster machines. The more agile a company is, the better able it is to bring to market the *newest* hardware. Since it is usually the smaller companies that are the more agile, it is no surprise that larger companies (including IBM and DEC) are often late to market with their products.

The UNIX operating system has been proposed as a standard in order to insulate computer users from this fast changing technology. The theory is that you can buy the newest, fastest computer today (in your price range) and upgrade to another vendor's UNIX based system whenever you want, without worrying about migrating your applications or investing in education time to learn another operating system. This argument has merit, but is somewhat premature. UNIX is not standardized yet, and until it is, conversions between UNIX based systems still involves some effort. There is also the issue of whether UNIX is a better operating system than VMS (or any other vendor's proprietary operating system). This debate appears to be a *holy war* – each group believing it has found the one true operating system. UNIX and VMS are both good, while each also has its weaknesses.

The pressure for faster cpu cycles leads potential buyers to review vendor benchmark results. Is an Intel based 80386 or Motorola 68020 based cpu faster than a VAX 785? The answer is *yes* if you look at the *correct* benchmarks. Computer users are becoming more knowledgeable about the factors affecting a computer's power in order to better evaluate a product's claims. These factors include cpu speed, memory capacity and input/output bandwidth or throughput.

Central processing unit (cpu) speed is the most often quoted yardstick for measuring the power of a computer system. A MIPS (million instructions per second) rating of 1 for the VAX-11/780 is routinely quoted although some Digital equipment owners use the VAX-11/780 itself as the basic unit, (using the term VUP - VAX unit of performance) – a particularly self satisfying term for DEC's marketing group. However, quoting cpu speed using MIPS is often misleading if not inaccurate. The uncertainty comes from the imprecise definition of the term *instruction*. On a Reduced Instruction Set Computer (RISC) architecture, instructions are simple, by design. One instruction may load a register while another instruction adds two registers together. On a complex instruction set computer (CISC), like the VAX, instructions can accomplish many things such as the single instruction that implements the Fortran DO loop construct. The RISC machine may have a higher MIPS rating but not necessarily run your

application faster than a CISC machine.

For this reason, running specific benchmarks are a better way to compare machines. The choice of benchmarks, however, is critical. The best approach is to duplicate your applications mix on the machine you want to test. This is usually too difficult and time consuming. A useful compromise is to run some of your applications separately or *stand alone* assuming that there will be a linear relationship when the applications are run together. (This assumption is reasonable but not always true. For example, if there are real-time applications that are run along with other jobs, the actual job mix should be tried.) Standalone tests comprise the bulk of the benchmark tests and are often relied upon to determine the relative speed of two systems. The problem comes from extrapolating these results. For example, if a standalone program works twice as fast on a new machine, it is too easy to assume that all applications will run twice as fast. The reality can be much different. Factors that can effect how other applications may fare include 1) hardware speed assist for certain arithmetic operations but not for others; 2) compile time optimizations; 3) excess memory paging due to nonlocality of references; 4) speed of input/output operations; and 5) special coding to take advantage of multiprocessing or parallel processing.

Memory capacity effects the *power* of a computer system in two ways. For an operating system that does not support virtual addressing, the memory capacity puts an upper limit on the size of a computer program, and therefore its complexity. (Techniques such as overlaying allow a partial work around to this limit but at the expense of longer development and execution time.) For operating systems that do support virtual addressing, the virtual address space is usually large enough to accommodate almost all applications. However, the computer typically cannot support as much physical memory as virtual memory. When an application exceeds the physical memory limit, paging and/or swapping will occur, causing serious performance problems in the worst case. Memory capacity then can limit how useful a particular system configuration is to your needs, even if its cpu speed is superior. Another factor is memory cost. An entry level computer system may be aggressively priced to compete favorably with its competition. However, memory upgrades (perhaps essential to your application) may be very expensive due to lack of competition. The final system cost will be much higher than expected.

The input/output (I/O) bandwidth of a machine must also be considered when evaluating the power of a computer system. Most vendors know how to design a balanced system so that the I/O subsystems match the speed of the cpu. It does no good to have a very fast cpu wait for information from memory or secondary storage devices like disks and tapes. Many factors can effect the I/O bandwidth. These include 1) the speed and width of the data path between memory and cpu; 2) the speed and number of channels between external devices and the internal data path; 3) individual subsystem components such as disk drive access time and disk controller transfer rate capacity; and 4) design of the internal data paths – e.g. do memory transfers wait for slower disk transfers on the same path or is there a separate data path for cpu to memory

transfers.

There are many other factors that can effect the overall performance of a computer system. It is important to recognize that they do exist and to carefully use vendors' claims about performance in your evaluation.

Tying Computers Together

By 1985, it became clear that Institution scientists needed easy access to a full range of computing resources and that a network would be the mechanism to provide that easy access. The Institution expanded the role of the computer center to include data communications and provided funding. The network design is based on the network in use at the Massachusetts Institute of Technology.¹ The MIT network uses the TCP/IP protocol and the concept of a *backbone* facility providing high speed connections among local, building based networks. We will eventually use fiber optic cable as our backbone although until its installation, we are using the broadband cable to provide our interbuilding connections. Ethernet, baseband cable, is used within buildings to provide local area networking among PC's, workstations and the VAX systems. Gateways are used to provide access to other buildings and other networks. The TCP/IP protocol is supported on the VAXes via Wollongong's WIN software and DECnet-TCP/IP connections are possible via gateways. Novell's Netware software provides a local area network environment for department's PC's used mainly for word processing applications.

A microwave link to MIT provides video, voice and data channels to MIT and, through MIT's network, to other sites including NSF's supercomputing centers (NSFnet). We are also installing a high speed satellite link to the Cray system at NCAR and will be part of the University Satellite Network (USAN). We will continue our connection to the Space Physics Analysis Network (SPAN) Decnet network as well. Due to the interconnection of many of these networks, there can be more than one way to communicate between nodes, providing redundant communications paths.

Flexibility is Costly

The efforts to provide users with a full range of computing resources accessed via a local area network take their toll in complexity and personnel support costs. It was essential that a separate Data Communications Group be formed to handle the network implementation, to provide ongoing hardware and software advice to people wanting to connect to the network, and to handle problem calls in a timely manner. It does not take long for users to become dependant on their network connections and any break in service is considered a serious problem.

Similarly, the network has made it easier for people to connect up computers with dissimilar operating systems. With this flexibility comes additional demands for systems help.

¹It is often counter productive and costly to always develop ones own, separate problem solutions. The *not invented here* syndrone must be overcome in order to build on other peoples' successes.

In addition to VMS support, we must provide problem resolutions for the MS/PC-DOS and the various flavors of the UNIX operating system. A separate PC support group within the central computer facility is available to handle the PC and more recently UNIX questions. As the use of UNIX increases, however, more (possibly separate) support will be required for UNIX systems. In order to keep the support task manageable, only a limited number of PC software products are fully supported (e.g. Wordmarc/Composer, T_EX/L_AT_EX, Supercalc, Symphony). Users can purchase whatever software they choose, but their expectations for how much help they can expect to get must be set correctly.

Conclusions

Due to its need for a diversity of computing resources, the Institution routinely used machines ranging from personal computers to supercomputers. The next step was to connect these machines together to maximize their use. The result is the creation of three types of networks: 1) a very local area network (VLAN) consisting of personal computers for departmental word processing and microVAX based VAXclusters for scientific use within a lab; 2) local area network (LAN) within one or more buildings that connect the researcher's PC's and workstations to other scientists and to the secretarial word processing PC's; and 3) wide area network(s) (WAN) that connect the Institution's networks to national and international networks.

The future probably will be more of the same – more personal computers, workstations, project specific supermini-computers, central mid-range computers and access to remote supercomputers. The new minisupercomputers may also play a role if use demands. It is necessary to balance the requirements for diversity and flexibility against the limitations imposed by available funds. Support costs (including people and equipment) for the network, computer operating systems and application programs will become unacceptably high unless the balance is maintained.

Acknowledgment

I thank Debbie Marena for the care and speed with which she prepared this manuscript.

The Woods Hole Oceanographic Institution, Information Processing and Communications Laboratory (project number 5602) provided support for this work. Woods Hole Oceanographic Institution contribution number 6501.

References

- [1] Gordon Bell, NSF, Personal Communication, 26 May 1987.
- [2] Groman, Robert C., The VAX Generation: Management Decisions, in Proceedings of the Digital Equipment Computer Users Society, Anaheim, CA, December 1984.
- [3] Notkin, David; Hutchinson, Norman; Sanislo, Jan; and Schwartz, Michael. Heterogeneous Computing Envi-

ronments: Report on the ACM SIGOPS Workshop on
Accommodating Heterogeneity CACM Vol 30, No. 2,
February 1987.

The Allocation and Mounting of Magnetic Tapes Under VMS

Clyde T. Poole

The University of Texas at Austin
Department of Computer Sciences
Austin, Texas

Abstract

The allocating of tape drives and the mounting of magnetic tapes under VMS is handled by a set of immature commands that are not necessarily suited for use by large VAX sites. This paper attempts to point out the weaknesses in the system and suggests possible changes that would make the system more usable.

Introduction

Most VAX/VMS¹ sites fall into one of two categories regarding the way magnetic tapes are handled.

The first is the traditional VAX site: one or more VAX's with a relatively small number of users in a "friendly" environment. The users handle the allocation of tape drives and the mounting of magnetic tapes for themselves. If all the tape drives are allocated, a user needing to mount a tape, asks around and persuades someone to relinquish a drive.

The second type of site has only recently begun to use VAX's extensively. They are usually larger sites than the first type, with large numbers of users spread across a large physical area. Magnetic tapes are mounted for the user by a trained operator. Physical access to the machine room is usually restricted. These sites either have or have had other large computers manufactured by DEC (DECsystem-10's and/or DECSYSTEM-20's²) or other vendors.

This paper is concerned with the problems of sites of the second type. The current tape allocation and mounting system in VMS seems sufficient for sites of the first type. In this paper I will present a list of problems along with some possible solutions to these problems. I will also present and examine some other possibilities and considerations.

Problems and Solutions

The following is a list of problems that have been identified. Following each problem is one or more solution. The solution numbers are prefixed with the letter S. Where more than one solution has been presented, the capital letters (A-Z) have been used to separate the solutions from each other. These problems and their associated solutions are in no particular order and therefore their order should not imply any sense of preference or seriousness.

¹VAX and VMS are trademarks of Digital Equipment Corp.

²DECsystem-10 and DECSYSTEM-20 are trademarks of Digital Equipment Corp.

1. The ALLOCATE command can be used to allocate any unallocated tape drive to a process. The operator cannot prevent this allocation except by allocating the drive to an operator process. This, in turn, prevents the MOUNT command from ever succeeding because all the tape drives are allocated. This also allows a malicious or ignorant user to stop all access to tape drives by simply allocating them all.

S1A. The ALLOCATE command is really not necessary (or desirable) at most sites. Simply making it a privileged command would solve the problem.

S1B. In TOPS-10³ there was a command called RESTRICT which limited direct allocation (ASSIGNment in TOPS-10) of a restricted device to the operator and privileged system utilities. The operator also had a command, REASSIGN, that allowed him to pass an allocated device to another process (job in TOPS-10). Implementing a similar pair of commands in VMS and allowing the MOUNT command to allocate a restricted tape drive to a user would work fine. When a user deallocated (DEASSIGNed in TOPS-10) the restricted device, its restricted nature returned automatically. This would not require any changes visible to the users of the ALLOCATE command. Note that simply using the device protection scheme and/or ACL's on a device does not produce the same behavior.

2. The REQUEST command is not a clean interface for requesting that the operator mount a particular tape. This command requires the user to perform a multi-step process to get a tape properly mounted. First the user allocates a drive. Then he uses the REQUEST command to ask the operator to load a particular tape on a particular drive. The user finally issues a MOUNT command that may now fail because the operator forgot to put the write ring in the tape reel. Many sites have implemented .COM

³TOPS-10 is a trademark of Digital Equipment Corp.

files to make this operation a single step for the user. The fact that so many have done so suggests the need for a single-step mount process.

S2. The MOUNT command alone should pass enough information to the operator and VMS to allow the completion of a mount request. If a tape is labeled, VMS should check a data base of tape protections and ensure that the user is allowed access to the tape being requested. If he is not allowed access, he should be notified and the operator should not get any mount request indication. It might be appropriate for this to be a security alarm condition. If he is allowed access, the operator should be asked to mount the tape. If the user has asked that a foreign tape (one that is not in the protection data base, see S5 below) be mounted, then a visual identification string should be required by the MOUNT command and the operator request should clearly indicate that the operator is responsible for ensuring that the user should have access to the tape. If it is an unlabeled tape, the operator should tell VMS which drive he mounted the tape on. If it is a labeled tape, VMS should detect the mount, read the label and complete the request with no further operator intervention.

3. The mount system allocates the tape drives in a manner that causes higher usage of the low numbered drives than of the high numbered drives.

S3A. The solution S2 above solves this problem also.

S3B. If VMS must tell the operator which tape drive to use, it should at least rotate through all available drives of the appropriate density.

4. The MOUNT command does not have a visual identifier switch. Many sites allow foreign, unlabeled tapes to be mounted. The user needs a clean method of indicating that such a visual identifier should be checked before mounting a tape.

S4. The solution S2 above solves most of this problem. The support for a visual identifier should be extended to the error logging process to facilitate detection of failing tapes.

5. There is no tape protection system. A user can ask the operator to mount any tape. The current system relies on the visual identification of the tape and its allowed users.

S5. Implement a known-tape protection and ownership data base. As a very minimum this data base should contain:

- (a) the UIC (or rights identifier) of the owner of the tape
- (b) the allowed access types for each allowed user (including the owner); read-only, read-write, write-only for interactive processes and the same set for batch processes
- (c) an indication of whether the tape is magnetically labeled or not and the type of label (ASCII, EBCDIC, etc.)

- (d) the tape magnetic and visual label (they could be the same but should not have to be)
- (e) the date of the last successful mount
- (f) a count of the number of times it has been mounted
- (g) an expiration date
- (h) the date it was added to the data base
- (i) a location field (which cabinet, room, building, etc.)
- (j) volume set name and relative position in volume set
- (k) user defined, reserved fields

The allowed user list should be by UIC (with wild cards allowed) or rights identifier. The utility for building and maintaining this data base should allow the owner of a tape to change the protection of the tape without the intervention of anyone else. In addition, the /OVERRIDE=ID switch to the MOUNT command must be made a privileged operation when the tape is a protected tape, i.e., one in the ownership data base.

6. There is no mount queuing system. If all tape drives are currently allocated, all succeeding MOUNT commands fail.

S6A. Implement a mount queuing system using a first come first served scheme with consideration given to different tape densities (see S13 below).

S6B. Implement a mount queuing system as S6A above with the added feature that the operator have the ability to override the system and select which request he will service next.

S6C. Implement an absolute priority mount queuing system with operator override of request priority possible.

7. The MOUNT command blocks DCL. If a user wants to mount more than one tape at the same time he must mount each in a sequential fashion.

S7. The MOUNT command should make a mount request and then return to DCL command level by default. Succeeding MOUNT commands would be processed in the same way. Interactive users would get a message as each request they had pending was satisfied. A MOUNT/WAIT command could be used in batch to insure that a batch command file waited for all necessary mount requests to be completed before continuing. A MOUNT /CHECK or a SHOW QUEUE /TAPE command would allow a user to examine the status of any mount requests he has pending.

8. There is no way for the batch queuing mechanism to tell if all the tape resources necessary for a particular batch job are available before the job is started. Tape deadlock situations are easily produced and difficult to remedy.

S8. Implement a switch to the SUBMIT command where a list of required resources could be provided such as:
/RESOURCES=(TAPES:(DENSITY:1600,COUNT:2))

9. Any automated tape protection system will probably require that the protections revolve around the magnetic tape label. Users can currently change the label written on a tape to anything they want using the INITIALIZE or MOUNT utilities.
- S9A. Make the INITIALIZE command an operator privileged command and take the /INITIALIZE switch out of the MOUNT command.
- S9B. Add a new privilege called "May Initialize Tapes" and only give it to the operator by default.
10. The BACKUP utility reinitializes tapes as it uses them (especially system backups). This will cause problems with any tape protection and/or allocation system that might be implemented.
- S10. Make BACKUP respect existing labels. It should be emphasized that when BACKUP is supplied with a list of tape labels, the intention is that BACKUP use those tapes. BACKUP should verify the tape mounted has the appropriate label before it writes on it and leave the same label when it is finished writing on it.
11. The DISMOUNT /NOUNLOAD command allows the user to leave a tape loaded but unallocated. This is a security problem. If the user has not allocated the drive, another user can now allocate the drive and get access to the first user's tape.
- S11A. Remove the /NOUNLOAD switch from the DISMOUNT command. This assumes that an appropriate REWIND command will be implemented.
- S11B. Make /NOUNLOAD a privileged command. This assumes that an appropriate REWIND command will be implemented.
- S11C. Allow the /NOUNLOAD switch only if the tape was allocated with the ALLOCATE command. (see solution S1A)
12. The DISMOUNT command does not force a DEALLOCATE of the drive. This causes two problems. The first is a continuation of problem 11 above. The second is related to problem 1 above. Users tend to forget that they allocated a drive. If they do not enter a DEALLOCATE command, the drive will remain unusable to the rest of the user community until the offending user logs out.
- S12. Solutions S1A and S1B above solve this problem.
13. Users must know the device names of the tape drives available to them. This includes knowing that some drives are 800-1600 bpi drives, some are 1600 bpi only, some are 1600-6250 bpi, some are streaming and some are start/stop. On systems with many tape drives, this can be confusing.
- S13. Implement a /DRIVE=density switch to the MOUNT command that will pick a tape drive of the appropriate density. If =density is left off, assume some default, probably 6250. If this switch is present, don't require a device specification.
14. Sites that do not have 24 hour per day, 7 day a week operator coverage have a problem. There is currently no way for a user to determine that an operator is or is not on duty. A user should probably not be able to make a mount request and batch jobs that need tapes should not start if there is no operator on duty. The REPLY /DISABLE=TAPE option is not sufficient.
- S14. Implement a "no operator on duty" flag that can be queried in some simple manner and make the MOUNT command check to make sure there is an operator before making a request. Solution S8 above should also be extended to check this flag.
15. If a user requests that a tape be mounted "write-enabled", the mount system does not check to make sure that the drive is hardware write-enabled; that is, the write ring is in place in the mounted tape. The reverse is also true. A tape can be mounted with the write ring in place when the user requested read-only access.
- S15. Have the operator portion of the mount system check the status of the write ring and require that the operator remount the tape with the ring in or out as requested by the user. The user should never know that this happens.
16. The operator has no clean method of making a tape drive completely inaccessible. This is especially needed when a tape drive is broken. Allocating the drive to an operator process is not a good solution.
- S16A. Implement a command similar to the DETACH device command in TOPS-10. This command completely removes the device specified from visibility to any device dependent commands. This also implies that the ATTACH device command is needed to restore the visibility of the device.
- S16B. Make the command SET DEVICE /NOAVAILABLE work for magnetic tape drives.
17. There is no method of removing an allocated tape drive from another process. This is especially painful when a user allocates a drive, uses the drive and then starts a long running program without deallocating the drive. The user would normally gladly give up the drive if he did not have to lose all the run time he may have already accumulated.
- S17. Implement some privileged command that will allow the operator to remove an allocated device from another process without interfering with the running image.
18. VMS does not support standard IBM⁴ EBCDIC labels and tapes.

⁴IBM is a registered trademark of International Business Machines Corp.

- S18. The tape mounting system should recognize and process standard IBM labels. The EBCDIC translation would only be applied to the tape label portions of the volume set; no translation need be applied to the data file sections of the volume set.
19. There is no method of redirecting a request for a device. It sometimes occurs that only a particular drive will read a particular tape. This shouldn't happen but it does.
- S19. Allow the operator to redirect a mount request from a specific drive `xxxx:` to another specific drive `yyyy:`. This eliminates the need for a `MOUNT` followed by a `DISMOUNT` followed by another `MOUNT`.
20. The `MOUNT` and `DISMOUNT` command pair do not produce sufficient accounting information. Specifically, the total elapsed time that a drive is under the control of a user is not collected.
- S20. Improve the accounting entries provided by the mount system so that the total elapsed time that a drive is allocated by a user is collected in some way. A possible method is to provide both a "mount" and a "dismount" entry such that the elapsed time between the entries could be calculated.

presented in this paper comes from the investigation and implementation of the University of New Orleans, `MOUNT` and Magtape Inventory System. Because of this, I owe special thanks to Charles Boyd, Robert Adam II and I. Joseph Autin; all of the UNO, Computer Research Center staff.

Other Considerations and Possibilities

I realize that systems like the tape protection data base may be very site dependent. Digital would also probably like to make any system it implemented into a separately licensed (and paid for) product. It therefore might be more appropriate to make the mount utility into a user modifiable system, somewhat like the user modifiable print symbionts. "Exits" might be provided for attaching tape security subroutines for example. I would be happy to work with Digital on the design of such a system.

Many of the problems I have presented are related to security. These need attention as soon as possible. Most have very simple and straight-forward solutions.

At least one of the problems presented concerns `BACKUP`. `BACKUP` should have no regard for the labels on a tape, except for the requirement that one be present. The labeling of tapes is an administrative nicety for tracking and protecting tapes. Most sites need to have consistent labeling for tracking tape failures and usage. Letting `BACKUP` change the tape label defeats this property of labeling.

A general observation about mounting tapes is that the user has too much control over the system and the operator has none. In large computer installations, the orderly use of non-shared resources requires that the operator have some level of control over those resources. The `MDA` (mountable device allocator) and `GALAXY` systems under `TOPS-10` could be used as examples of an implementation that works.

Acknowledgments

The problems and solutions presented in this paper are the results of the efforts of many people. Most of the information

Mysteries of VAX/VMS system parameters revealed

Steven Szep
Chase Manhattan Bank
1 New York Plaza
New York, NY 10081

Abstract

This paper will present VMS system parameters from an "Internals" perspective. In the first part, we relate the various components of VAX/VMS to particular subsets of the complete parameter set. In the second part, we take the same approach with respect to a single process.

The System

System initialization

Before a VAX/VMS 4.x system can operate, some initialization (or, "bootstrap") program must execute to configure the system and read the Exec into memory. Parts of this operation are specific to the type of VAX processor; others are common across all VAX computer systems.

Sysboot itself occurs in two phases:

- Init, which loads the code which is part of the Exec;
- Sysinit, which loads the code which sets up some process context.

Basically, Init turns on memory management and sets up those data structures whose size or content depend on system parameters. Sysinit, however, opens the system files, creates system processes, maps RMS and the system message file, and creates the process which invokes the system startup command procedure (SYSTARTUP).

Once Init has succeeded in turning on memory management, it is free to make references to system addresses – particularly, in order to initialize those dynamic data structures whose listheads are stored in static global locations in system (S0) space. This involves the allocation of memory from nonpaged pool.

The size of the packets for the three lookaside lists are calculated; the lists are then formatted and linked together.

The system header is treated by the VMS memory - management sub-system as the occupant of a balance slot whose index is equal to the system parameter BALSETCNT.

Sysinit, like Init, consumes large amounts of nonpaged pool and some paged pool. However, the sizes of various control blocks are not directly related to system parameters.

The swap file is divided into swap spaces. Each space is a multiple of the system parameter MPW_WRTCLUSTER. The maximum number of processes the system can support is taken as the minimum of the swap file space count and the system parameter MAXPROCESSCNT.

System virtual address space

The Exec image SYSSYSTEM:SYS.EXE contains the code for VMS, but very little data. Since many of its data structures are not created until the system is booted, their sizes can be determined from the appropriate system parameters.

The number of bytes in a page is 512. If 511 is added to an expression for a number of bytes before an integer division takes place, this represents a rounding up to the next - highest page boundary.

Since a page table entry is 4 bytes long, a page of page table entries maps 128 pages. (Here the rounding up factor is 127.)

System virtual address space (SVA) is "configured" as follows. The area which will contain the linked Executive, the RMS image, and the system message file has its size determined by the system parameter SPTREQ. There must also be enough pages here to map the I/O adapters and to reserve a system virtual page for each device unit whose driver requests one.

The space reserved for the paged dynamic memory area depend on the system parameter PAGEDYN.

The space reserved for nonpaged pool is the sum of the size of the nonpaged memory and the size of the lookaside lists – that is, small request packets (SRPs), I/O request packets (IRPs), and large request packets (LRPs). The system parameter NPAGEDYN determines the size of this space. The size of each lookaside list is determined by the size of the request packets and the number of packets in the list.

The system parameter INTSTKPAGES states the size of the Interrupt Stack.

The size of the area devoted to balance slots depends on BALSETCNT, times the size of a process header. Thus, we can see that we reduce BALSETCNT in order to support a large process virtual address space. Likewise, we reduce VIRTUALPAGECNT in order to support a large number of concurrently - resident processes.

The size of the system page table depends on two system parameters, SYSMWCNT and GBLSECTIONS. The size of the

global page table depends on one – namely, GBLPAGES.

For calculations depending on the amount of available physical memory, VMS uses the minimum of the size of physical memory and the sysgen parameter PHYSICALPAGES.

The system parameters FREELIM and MPW_LOLIM set the lower-limit thresholds for the number of pages on the Free Page and Modified Page Lists, respectively.

Dynamic memory allocation

VMS maintains three separate areas for the dynamic allocation of storage:

process allocation region data structures required only by a single process;

paged dynamic memory data structures used by several processes, but not required to be permanently non-resident (for example, group and system logical names, global sections, known file entries, resident image headers);

nonpaged pool data structures and code used by the portions of VMS which are not procedure - based; data structures and code shared by processes, but not paged (for example, PCB and sequence vector, the Swapper's I/O page table, page-file bitmap, modified page - writer's arrays, adapter control blocks – for all external adapters located at boot-time, device driver code and associated data structures – for devices either located through the autoconfiguration phase of Sysgen or explicitly loaded via the Sysgen commands LOAD or CONNECT).

Nonpaged pool contains four regions: the three lookaside lists (for SRPs, IRPs, and LRP) and the remainder of variable length. Nonpaged pool differs from paged pool (and the process allocation region) in that it can possibly be extended as part of the normal system's operations. For each region, there are two relevant system parameters: one specifies its initial size; the other, its maximum size.

The size of the variable - length region of nonpaged pool is controlled by NPAGEDYN and NPAGEVIR. Although both are expressed as a number of bytes, both are rounded down to an integral number of pages. During initialization, sufficient contiguous system page table entries (SPTes) are allocated for the maximum size of the region, equal to whichever of the sysgen parameters NPAGEDYN and NPAGEVIR is greater. Physical memory pages are mapped using a portion of these allocated SPTes.

During system operation, a failure to allocate from this region results in an attempt to expand the region. The physical pages are allocated to fill in the next portion of available SPTes. VMS's deallocation strategy requires that this extended dynamic area be virtually contiguous with the previously existing one. The four regions must also be adjacent.

This strategy is the reason that the maximum number of SPTes must be allocated at once for each region – even though some of them are initially unused.

The lookaside lists for SRPs, IRPs, and LRPs are allocated in the same manner – using the following system parameters:

Type	Size
SRP	SRPSIZE
IRP	160
LRP	LRPSIZE

Initial and maximum counts are maximized. SRPSIZE is rounded up to a 16 - byte boundary; the maximum size of the SRP list is rounded up to a page boundary. LRPSIZE is also rounded up to a 16 - byte boundary, and the maximum size of the list is rounded up to a page boundary.¹

Dynamic nonpaged pool expansion enables automatic "system tuning". However, an inadequate initial allocation size will result in increased overhead due to the expansion carried out during allocation requests. Also, unnecessary PFN database is built for physical pages added to nonpaged pool during expansion: the penalty here is 18 bytes (4%) per page.

For a too - large maximum allocation, the penalty is 1 SPTe for each unused page, which is 4 bytes (1%). If the maximum size of a lookaside list is too small, performance is adversely affected – because VMS is prevented from using the lookaside mechanism for pool requests.

If the maximum size of the variable - length region is too small, processes must wait for nonpaged pool to be allocated and may be placed in MWAIT.

Note: The three lookaside lists are structured into a series of elements, of size = xRPSIZE. In each of the lists, the elements are entered into a doubly - linked list. The system parameters SRPMIN and LRPMIN indicate the smallest sizes for packets in the SRP and LRP lists, respectively.

Process creation

The \$CREPRC system service allocates a process control block (PCB), a job information block (JIB) for detached processes only, and a process quota block (PQB). This routine fills these three structures with the implicit and explicit parameters passed to it. The control blocks are allocated from nonpaged pool.

Two tables in the Exec are used by the \$CREPRC system service when quotas are set for the new process: a minimum quota table and a default table. Each quota or limit in the system has an entry in both tables. The contents of the minimum table are determined by the system parameters prefixed by PQLM, while those of the default table by PQLD.

The default values for each quota are placed inside the PQB. Any quota included in the argument list to \$CREPRC replaces the corresponding default value. Each quota is then forced to its minimum value. Checks are made to insure that the creator process possesses sufficient quota to cover those given to the new process. Finally, the required quotas and working values are moved into the PCB.

¹The output of the DCL command \$ SHOW MEMORY displays the quantity LRPSIZE+64 .

The Shell process

A process comes into existence in state **COMO**. Its swap image exists in the paged portion of the Exec image `SYSS$SYSTEM:SYS.EXE`. This image contains a minimal process header and control region (P1) space.

The selection of a newly - created process for inswap and the actual inswap operation are both performed by the Swapper.

When the Exec image was linked, the Shell process was constructed to look like an outswapped process. However, a process header must be configured via several system parameters. To perform this task, the Swapper determines whether the new process was created from the Shell. If so, it calls a routine to complete the process header before inswap is completed.

The system parameters `PAGTBLPFC` and `PFCDEFAULT` are stored in the process header.

The

`WSQUOTA`, `WSAUTH`, `WSEXTENT`, and `WSAUTHEXTENT` pointers are initialized to the system parameter `WSMAX`. The `WSFLUID` counter is initialized to `MINWSCNT`. The end of the working set list (`WSLAST`) and the default count (`DFWSCNT`) will initially reflect the value of `PQL_DWSDEFAULT`.

New processes

The first code which executes in the context of a newly - created process is the same for every process in the system.

The quotas stored in the process header - CPU time limit and AST limit - are moved from the PQB to the process header.

The working set list pointers are initialized to reflect the quotas passed from the creator process. (Of course, this takes place after minimization with the system - wide value for `WSMAX`.)

The I/O channel table will be created in P1 space. The number of channels permitted is determined by the system parameter `CHANNELCNT`.

Process headers

The size of the process header is related to several system parameters: `PHD$K_LENGTH`, `PROCSECTCNT`, `PQL_DWSDEFAULT`, and `VIRTUALPAGECNT`.

Most of the process header is taken up by the P0 and P1 page tables. The total number of pages allocated for these tables is

The process header pages which do not contain page tables are locked into the process's working set: they always require physical pages. Thus, many processes will have working sets less than `WSMAX`. The initial working set list size is calculated for this very reason.²

²The assumption made here is that most processes will have working sets approximately equal to `PQL_DWSDEFAULT`.

Process spaces

Most of the pieces of P1 space have pre-determined sizes. These sizes are based on the contents of the module called "Shell" in the Exec.

The image I/O segment is created by the Image Activator. It is the RMS impure area for files opened during the execution of a specific image.

Images

Before an image can execute, VMS must take steps to prepare it for execution. Process page tables and other data structures must be set up to locate the correct image file on disk. Address references between shareable images must also be resolved.

In addition, if the debugger or the traceback handler is expected to run when the image executes, the correct hooks must be present to allow them to be invoked.

After the Image Activator has processed all the image descriptors, it calls the `$CRETVA` (Create Virtual Address Space) system service to create the image I/O segment. Its size is determined by the system parameter `IMGIOCNT`, which may be overridden by the "IOSEGMENT" Linker option.

System parameter tables

A common module, Parameter, is linked into both the Sysboot and Sysgen images. It contains information about each adjustable system parameter. This data never changes.

Each parameter occupies a cell in a table of working values. This table can be manipulated via the Sysgen commands `SHOW`, `SET`, and `USE`.

There is also a copy, `Sysparam`, of this working table linked into the Exec image `SYSS$SYSTEM:SYS.EXE`.

Miscellaneous remarks on system data structures

System page table entries (SPTes) are reserved for, and physical memory pre-allocated for, `NPAGEDYN`, `LRPCOUNT`, `IRPCOUNT`, and `SRPCOUNT`.

SPTes are reserved for `NPAGEVIR`, `LRPCOUNTV`, `IRPCOUNTV`, and `SRPCOUNTV`. Physical memory is not pre-allocated for `NPAGEVIR`, `LRPCOUNTV`, `IRPCOUNTV`, and `SRPCOUNTV`: it is allocated on demand from the Free List if there is enough "excess" memory.

Changing system parameters

1. Save existing parameter values before making any changes.
2. Change only a few parameters at one time; make the changes small ones.
3. Observe system behavior relevant to the changes.
4. Monitor system to discover new problems.
5. Evaluate your success.
6. Return to the original values, if necessary.

7. Start all over again?

Autogen

Autogen recalculates some 60 parameters; there are 241 in VMS 4.5 .

Do not be concerned about that big number...If you use Autogen, your system will probably re-boot.

The process

Overview

A process is the environment within which programs execute under VAX/VMS. This entity consists of a hardware context, a software context, and some virtual address space.

A number of processes can run in the available physical memory. To VMS, a process is a schedulable entity. Each process manipulates data, some of which it may share with other processes.

As we shall soon see, the whole memory management strategy of VMS depends on the process's initial working set quota, its working set extent, and its base priority.

VMS consists of the Executive, which is always resident in memory, and several other components. VMS attempts to ensure that each process can complete its work by allocating sufficient system resources to each one.

Image activation lays the groundwork by which the process can bring into memory its first set of pages from the image file.

Physical memory on a VAX system has three major uses: process space, resident Exec, and page caches.

VMS and each process have their own individual working spaces. Actually, each working set contains process-specific lists. At any given time, the pages in a working set include all valid pages in memory for the process, and they may represent only a subset of those in the process's page tables.

There are enough balance slots reserved in physical memory for the maximum number of processes expected to be running concurrently, including VMS itself.

Under VMS, it is important to balance the use of memory and the number of processes running at once. Each process has a QUANTUM of time available to it for doing work. VMS schedules both when and for how long a process executes.

The Swapper schedules the actual usage of physical memory. It keeps track of the pages which are in physical memory, as well as in the paging and swapping files on disk. Each process should have a steady supply of pages for each task.

When a process demands more pages than are available in its working set, some of this process's pages must be moved out to the page cache. Those pages which have been modified are kept on the Modified Page List; the unmodified ones are kept on the Free Page List. If the page cache is full, the Swapper transfers a cluster of pages from the modified cache out to the disk paging file. This action constitutes a page fault.

A page fault also occurs whenever the process needs the pages that are stored in either the image file or the paging file.

Under VAX/VMS, a bottleneck occurs if many processes begin page faulting at the same time. It costs memory to minimize the effects of this phenomenon.

Process creation

\$/CREPRC allocates new data structures: process control block (PCB), a job information block (JIB) if the new process is detached, and a temporary process quota block (PQB). These structures are filled in from the \$/CREPRC arguments, the creator's PCB, the creator's control region, the creator's process header (PHD), VMS defaults (PQL_Dxxx), and VMS minimums (PQL_Mxxx). The process ID (PID) is actually formed at this time.

Note: The VMS parameters MAXPROCESSCNT and BALSETCNT play an important role.

The Swapper's task is to store VMS parameters in the new PHD, to initialize pointers and counters in the new PHD, and to initialize SPTES.

The PROCSTRT component of SYS.EXE performs the following tasks: move PQB information to the PHD and P1 space; create logical name tables; map in the XQP; and, finally, call the image activator.

Scheduling

To help other processes compete with compute-bound processes, priority boosts are applied at the time of certain events – for example, I/O completion or resource available. There are different boosts for different events, but boosts cannot exceed priority 15.

At QUANTUM end, one of several actions might occur:

- an outswapped process may be made computable;
- the Swapper may be invoked;
- the process has run out its CPU limit and can now be deleted;
- AWSA is calculated.

Automatic working set adjustment (AWSA)

The goal of AWSA is to attain the optimal working set size for each process. The balance is between

- the maximum which allows good program performance;
- the minimum which optimizes overall memory usage.

A high page-faulter needs an increased working set size. A low-faulter may find its working set decreased if physical memory is required elsewhere.³

The basic scheme is as follows. The maximum size to which the working set can grow is WSQUOTA. If there are more than BORROWLIM pages on the Free List, however, then the Working Set List can be extended up to WSEXTENT, at QUANTUM end. If there are more than GROWLIM pages on

³The list size is modified, not the number of entries in use.

the Free List, pages can be added above WSQUOTA, upon the resolution of a page fault.

TUNE_INQ.COM is a command procedure to check tuning run.

Image activation

The Image Activator opens the image file, reads the image header, maps the image into virtual address space, and returns to the caller—typically, DCL.

The actual pages of this image are brought into physical memory as needed.

The process section table, or “PST”, locates the image sections on disk. The PST entries are actually built by the Image Activator. Most PST entry information is copied from the image section descriptors, or “ISDs”, in the {filename}.EXE .

A most negelected VMS parameter which applies here is PROCSECTCNT.

Process deletion

After the image runs and finally exits, the process will be deleted.

All traces of the process are removed from the system, and all system resources are returned. Accounting information is passed to the Job Controller.

If the process was a subprocess, all quotas and limits are returned to its creator.

Finally, the creator is notified of the deletion.

Notes on PTOOLS

SPM2.COM fires up SPM experiments. It first checks to see which drives are mounted and which devices are online. This program collects both tuning (SPM\$COLLECT.TUNE.DAT) and capacity (SPM\$COLLECT.CAPACITY.DAT) statistics.

PC.START.COM fires up SPM to collect system-wide PC (PC.DAT) statistics.

PC.LOG.COM reports on system-wide PC statistics.

PC.BY.ID.COM provides this report for a specific process.

PROC.START.COM fires up SPM to collect process metrics.

PROC.LOG.COM reports on process metrics.

PROC.BY.ID.COM provides this report for a specific process.

CAP.LOG.COM is a sample program for reporting capacity statistics.

TUNEX_GRP.COM is a sample program for graphing tuning statistics.

TUNEX_TAB.COM is a sample program for tabulating tuning statistics.

CAP_INQ.COM is a command procedure to check capacity run.


```

$! name: spm2.com
$!
$! purpose: to run SPM experiments on production machines
$!
$! author: s. szep
$!
$! date: 12/20/85
$!
$! revised: 05/07/86
$!
$!
$! create a local symbol
$!
$ wso = "write sys$output"
$!
$! purpose: find out what drives have packs mounted
$!
$ wso "Working..."
$ wso " "
$!
$ drive = ""
$ assign mou.tmp sys$output
$ sh dev /mou d
$ deas sys$output
$ open/read/error=no_way -
disks mou.tmp
$!
$! skip first 3 records
$!
$ read/end_of_file=no_more_disks -
disks line
$ read/end_of_file=no_more_disks -
disks line
$ read/end_of_file=no_more_disks -
disks line
$!
$! loop to get drive names
$!
$ again:
$ read/end_of_file=no_more_disks -
disks line
$ pos = f$locate(":",line)
$! ignore report labels
$ if pos .eqs. f$length(line) -
then goto again
$ pos = pos + 1
$ moudev = f$extract(0,pos,line)
$ one = f$extract(0,1,moudev)
$!
$! ignore RT's

$!

```

```

$ if one .eqs. "R" -
then goto again
$!
$! valid disk drives
$!
$ okay:
$ drive = drive + moudev + ","
$ goto again
$!
$! error path
$!
$ no_way:
$ wso " "
$ wso "*** Error in MOUDEV ***"
$ wso " "
$ goto end_it_all
$!
$! cleanup #1
$!
$ no_more_disks:
$ mou_len = f$length(drive) - 1
$ drive = f$extract(0,mou_len,drive)
$ sh sym drive
$ close disks
$ purge/nocon/keep=2 mou.tmp
$!
$! from: othdev.com
$!
$! purpose: find out other devices on the system
$!
$! table for permissible devices
$!
$ DEV_table = "2XM2XQ2XE2DV2MF2MU"
$!
$!
$ other = ""
$ assign oth.tmp sys$output
$ sh dev
$ deas sys$output
$ open/read/error=nogo -
dvcs oth.tmp
$!
$! skip first 3 lines
$!
$ read/end_of_file=no_more_devs -
dvcs line
$ read/end_of_file=no_more_devs -
dvcs line
$ read/end_of_file=no_more_devs -
dvcs line
$!
$! main read-loop
$!
$! forget about drives
$!
$ more_devs:
$ read/end_of_file=no_more_devs -
dvcs line

```

```

$ pos = f$locate(":",line)
$!
$! ignore report labels
$!
$ if pos .eq. f$length(line) -
then goto more_devs
$ abbrev = f$extract(0,2,line) ! get device "prefix"
$!
$! Main parsing routine.
$! This routine compares the current device
$! against the options in the device table.
$! When it finds a match, it branches to the
$! appropriate label.
$!
$ dev_siz = 2 ! device "prefix" length
$ index = 0 ! init. table index
$!
$! table search
$!
$ check_next:
$ dev_len = f$extract(index,1,dev_table) ! table-element length
$ if dev_len .eq. 0 - ! if a disk or tube,
then goto more_devs ! skip it
$ index = index + 1
$ next_dev = f$extract(index,dev_len,dev_table)
$ if abbrev .eqs. next_dev - ! if in table,
then goto found_it ! a valid device
$ index = index + dev_len
$ goto check_next
$!
$! valid devices
$!
$ found_it:
$ pos = pos + 1
$ othdev = f$extract(0,pos,line)
$ other = other + othdev + ","
$ goto more_devs
$!
$! error path
$!
$ nogo:
$ wso " "
$ wso "*** Error in OTHDEV ***"
$ wso " "
$ goto end_it_all
$!
$! cleanup #2
$!
$ no_more_devs:
$ oth_len = f$length(other) - 1
$ other = f$extract(0,oth_len,other)
$ sh sym other
$ close dvcs
$ purge/nocon/keep=2 oth.tmp
$!
$! *****
$! *
$! 2) Fire up SPM for tuning session... *
```

```

$!      *
$!*****
$!
$! from: tunex.com
$!
$! purpose: SPM tuning expt.
$!
$! setup parameters
$!
$! xdrive = "(" + drive + ")"
$! xdev = "(" + other + "," + drive + ")"
$!
$! start SPM's timer
$!
$! @sys$system:spmtimer
$!
$! start up tuning
$!
$! perf col=tune/int=300/class=all -
$! /DISK='xdrive' -
$! /DEVICE='xdev' -
$! /ending="+23:59:00"
$!
$! wso "TUN2 starting..."
$!
$!*****
$!      *
$! 3) Fire up SPM for capacity-planning session      *
$!      *
$!*****

```

```

$!
$! from: capex.com
$!
$! purpose: SPM capacity expt.
$!
$! start up capacity session
$!
$! perf collect=capac/int=300 -
$! /class=all -
$! /DEVICE='xdrive' -
$! /DISK='xdrive' -
$! /ending="+23:59:00" -
$!
$! wso "CAP2 starting..."
$! wso " "
$!
$! common exit point
$!
$! end_it_all:
$! sss := logout/full
$! sss

```

```

$! name: pc_start.com
$!
$! purpose: to start SPM to collect PC data.
$!
$! date: 09/11/86
$!
$! by: s. szep
$!
$!
$ write sys$output -
"*** Running SPM for PC statistics ***"
$!
$ @sys$system:spmtimer
$!
$ spawn/nowait perf collect=system_pc -
/ending="23:59" pc.dat
$!
$ exit

```

```
//
```

```

$! name: pc_log.com
$!
$! purpose: to use SPM to analyze PC data.
$!
$! date: 09/11/86
$!
$! by: s. szep
$!
$!
$ write sys$output -
"*** Running SPM to analyze PC statistics ***"
$!
$ perf rep=system_pc/out=pc.log pc.dat
$!
$ exit

```

```
//
```

```

$! name: pc_by_id.com
$!
$! purpose: to use SPM to analyze PC data for 1 process.
$!
$! date: 09/15/86
$!
$! by: s. szep
$!
$!
$ write sys$output -
"*** Running SPM to analyze PC statistics for 1 process ***"
$!
$ if p1 .nes. "" -
then goto do_it
$!
$ inquire p1 "Which pid? "
$!
$ do_it:
$ perf rep=system_pc/out=pc.log/id='p1' pc.dat

```

```
$!  
$ exit
```

```
$! name: proc_start.com  
$!  
$! purpose: to start SPM to collect process metrics.  
$!  
$! date: 12/16/86  
$!  
$! by: s. szep  
$!  
$!  
$ write sys$output -  
"*** Running SPM for process metrics ***"  
$!  
$ @sys$system:spmtimer  
$!  
$ spawn/nomwait perf collect=tune -  
/int=60/class=process -  
/nodisk/nodevice -  
/ending="23:59"  
$!  
$ exit
```

```
//
```

```
$! name: proc_log.com  
$!  
$! purpose: to use SPM to analyze process metrics.  
$!  
$! date: 12/16/86  
$!  
$! by: s. szep  
$!  
$!  
$ write sys$output -  
"*** Running SPM to analyze process metrics ***"  
$!  
$ perf rep=log/out=tune_proc.rpt -  
/tab=(interval,final) -  
/nograph/class=process -  
tune_proc.log  
$!  
$ exit
```

```
//
```

```
$! name: proc_by_id.com  
$!  
$! purpose: to use SPM to analyze process metrics data for 1 process.  
$!  
$! date: 12/16/86
```

```
$!  
$! by: s. szep  
$!  
$!  
$ write sys$output -  
"*** Running SPM to analyze process metrics for 1 process ***"  
$!  
$ if p1 .nes. "" -  
  then goto do_it  
$!  
$ inquire p1 "Which pid? "  
$!  
$ do_it:  
$ perf rep=log/out=tune_proc.rpt -  
/tab=(interval,final) -  
/nograph/class=process -  
/id='p1' -  
out=tune_proc.log  
$!  
$ exit
```

```
$! name: cap_log.com  
$!  
$! purpose: to report on SPM cap. log  
$!  
$! date: 8/8/85  
$!  
$! by: s. szep  
$!  
$! revised:  
$!  
$!  
$ perf report=log_file/graph=all/out='p2' 'p1'  
$!  
$ exit
```

```
//
```

```
$! file: tunex_grp.com  
$!  
$! purpose: tuning example graphs  
$!  
$! note: report for "tunex"  
$!  
$! by: s. szep  
$!  
$! date: 8/22/85  
$!  
$! revised:  
$!  
$!  
$ perf rep=log/out=glog1.dat/notab/graph=(all) -
```



```
$! revised:  
$!  
$!  
$ perf collect=tune/inquire  
$!  
$ exit
```

An Introduction to VAX/VMS System Tuning

Steven Szep
Chase Manhattan Bank
1 New York Plaza
New York, NY 10081

Abstract

Bottleneck detection is the primary goal of performance analysis. Resource management is the fundamental problem faced by the technical staff. This paper addresses these issues from the perspective of the mechanisms within VMS which deal with resource allocation of system resources to user processes.

Background on the tuning process

Introduction

To plan a measurement session, the workload and its evolution in time must be known. Measurements must be made in a "controlled" environment: the workload should remain constant, no hardware modifications should be carried out during the session, and no changes should be made to standard operational procedures.

CPU overhead is a problem when the performance tool is active. The number of statistics and the duration of the sampling interval must be chosen wisely, so as to perturb the system as little as possible.

Bottlenecks

A bottleneck is a limitation of system performance due to the inadequacy of a hardware or software component or of the system's organization.

Assumptions made at the time of setting up the system may be proven false as the workload evolves with time.

When the service requests for a given component exceed, in frequency and intensity, the service capacity of that component, the conditions for the appearance of a bottleneck arise. Because of the nature of the requests each process makes sequentially and not simultaneously for most resources, the other parts become more lightly loaded: many active processes end up in the wait queue of the overloaded resources and cannot contribute to other queues.

In a system in which all or most components are overloaded, specific sources of bottlenecks may not be found: the system may be saturated. To improve performance, a more powerful system must replace it or the workload must be reduced.

Bottleneck detection is vitally important: only by acting on the component(s) causing bottlenecks can advantages be obtained which justify the costs of system improvement.

Methodology

The use of Digital's SPM software product requires an iterative method for bottleneck detection based on the off-line interpretation of measurement results obtained from data collection by a standard set of tools. For these experiments to be considered valid, a representative workload is required.

Inefficiency in system performance leads one to ponder possible bottlenecks, their locations, and the availability of methods for their elimination.

Based on preliminary performance data, a hypothesis on what causes the bottleneck is formulated. Its validity is verified by analyzing data collected by further experimentation. When a hypothesis is confirmed, one must take steps either to eliminate the bottleneck or at least reduce its effects.

The removal of one bottleneck sometimes causes another to appear. This bottleneck can, in turn, be studied with the same scheme—which is repeated until the system is balanced (or, free of bottlenecks).

On-line bottleneck detection

The interactive component of SPM allows one to display certain system metrics in real time.

Also, the Monitor utility can provide information useful for hypothesis formulation.

The main advantages of on-line methods are:

- The speed with which important symptoms are detected;
- The ease with which their causes are often found.

System modifications to remove bottlenecks

The choices are few:

- Addition, replacement, or removal of one or more hardware components;
- Modification of Sysgen parameters, after reading AUTOGEN.COM;

- Tuning of the file system (RMS);
- Selecting one or more application programs for tuning;
- Optimizing the program mix.

Whatever is done, one must of course verify the effects on the entire system: the whole tuning process commences again.

Review of resource management

Before you begin a tuning session, you should be knowledgeable in the concepts of VAX/VMS resource management.

Without this understanding, you will encounter unnecessary problems in your attempts at system tuning!

Introduction

A process is a schedulable entity in the system. Under VAX/VMS, a number of processes can run in the available physical memory.

VAX/VMS consists of the Executive, which is always resident in memory, and several other components. It supports main (physical) memory, as well as secondary storage devices (disks and tapes).

Each process does “work”—that is, it manipulates data. The operating system tries to ensure that each process can complete its work as quickly as possible.

Memory management

Physical memory is divided into three main parts. One portion is available for processes; one is reserved for the resident Executive; and, one is for the page cache, where data is stored for movement from and to the disks.

Each disk has only one access path available to transfer data from and to physical memory—that is, to perform disk I/O.

There are enough balance slots reserved in physical memory for the maximum number of processes expected to run concurrently—including the operating system. The operating system and each process have their individual workspaces in physical memory, called “working sets”, which are actually process-specific lists.

A working set includes all of the valid pages in memory for any particular process. Pages in the working set typically represent a subset of the total number of pages in the process’s page tables.

Pages

In VAX/VMS, the basic addressable unit is the “byte”. Bytes are stored in groups of 512, called “pages”. Pages are kept in the working sets or in the section of physical memory called the “page cache”, as well as on disks.

The page is a convenient vehicle for moving a uniform number of bytes into and out of memory.

Note: When a page is written to disk, it is called a “block”.

Quantum

It is important to maintain an even balance in the use of memory and the number of processes running at once. Each process has an available amount of time to perform its work: the “quantum”, itself a system parameter.

The VAX/VMS quantum is a fixed time-slice. If no other process is waiting to exercise its quantum, the current process can retain control of the CPU.

Scheduling and paging

During image activation, the groundwork is laid so that the process can bring in its first set of pages from the image file and use them in its own working set.

The scheduling of physical memory is the task of the Swapper. It keeps track of pages in both physical memory and on disk: the paging and swapping files. It ensures that each process has a steady supply of pages for each task.

When a process’s demand for more pages exceeds those available in the working set, some must be moved to the page cache in order to make room.

In VAX/VMS, there are two sections to the page cache in physical memory:

- Pages whose contents have been modified, which are on the modified page list;
- Pages which have not been modified, which make up the free page list.

When the page cache begins to fill up, the Swapper transfers a cluster of pages from the modified-page cache out to disk, into the paging file. A “page fault” also occurs when a process needs additional pages, which are stored in either an image file or in the paging file. If there is insufficient space in the working set, the process must begin moving pages to the page cache. The process brings in groups of pages from the image file (on disk). The assumption is that the process is likely to reference pages other than the ones just referenced.

Under VAX/VMS, a bottleneck occurs when many processes begin page-faulting at the same time—particularly if there is only one paging file for all processes, and the speed of retrieval is that of loading between and disk—which is slower than memory accesses required to update the memory management database.

To alleviate this problem, you can install additional paging files on separate disks or create a larger page cache.

Automatic working set adjustment (AWSA)

Via AWSA, processes can acquire additional working set size (physical memory) under the control of VAX/VMS. The operating system recognizes the amount of page-faulting occurring for each process and factors this into the operation.

All processes have an initial default limit of pages of physical memory, `WSDEFAULT`. Any process which requires more space in memory is permitted to expand to the amount of the larger limit `WSQUOTA`. This is true only if `WSQUOTA` is less than `WSMAX`, a Sysgen parameter.

To avoid most page faulting (a potentially costly operation), VAX/VMS can extend working set space to “needy” processes—provided that free memory is available. The process can borrow working set space up to the final limit WSEXTENT.

The system manager must also consider the actual number of pages the working set requires. The actual working set count equals the process’s pages, plus any global pages used.

When a process’s working set increases, this growth occurs in increments of the system parameter WSINC. VMS reviews the needs for adding or subtracting pages only at the end of the next-occurring quantum and after the minimum interval established by the system parameter AWSTIME. Thus, VMS samples the page-faulting rate of each process over the adjustment period defined by AWSTIME and QUANTUM.

Example. If QUANTUM = 200 and AWSTIME = 700, then VMS reviews page requirements for a process every time the process consumes 800 milliseconds of CPU time, or every fourth quantum.

The goal of AWSA is to reduce the amount of page-faulting in the system as a whole. VMS compares the current rate of page-faulting each process is undergoing, against the norm established for all processes on the system via PFRATH and PFRATL, which are system parameters defining the upper and lower limits of acceptable page-faulting.

At the end of a process’s adjustment period, if the page-fault rate for the process is high—compared to PFRATH, VMS approves an increase in its working set size in the amount of WSINC, up to the value of its WSQUOTA, for the next adjustment period.

If this increase would place the process above its WSQUOTA, thereby requiring a loan, then VMS checks the availability of free memory against an established system norm, BORROWLIM. Thus, AWSA only permits a process to grow above its WSQUOTA value if there are at least as many pages of free memory as specified by BORROWLIM.

If too many processes attempt to add pages at once, VMS is forced to withdraw its intention of granting additional pages to processes which have already had the benefit of growing beyond their quotas.

When a process page-faults after its working set count exceeds its quota, VMS examines the value of the system parameter GROWLIM before permitting the process to use more of its WSINC loan. This aspect of AWSA is event-driven and is *not* tied into any adjustment period.

If there are as many pages on the free list as required by GROWLIM, VMS continues to permit the process to add pages to its working set. If the number of free pages does not equal or exceed GROWLIM, VMS will not permit the process to grow; in fact, the process will be forced to give up some of its pages before it reads in new ones.

Processes which are not heavily page-faulting can give up some of their working set limit through voluntary decrementing. Processes with a page-fault rate below PFRATL (when PFRATL does not equal 0) are subject to a loss of pages. This reduction occurs at the next quantum end after AWSTIME has elapsed. The amount of reduction is defined by the system parameter WSDEC. No process can be reduced below the

minimum size defined by the Sysgen parameter AWSMIN.

At the time an image exits, the process’s working set limit drops automatically back to WSDEFAULT.

The sizing of working sets

The VMS memory management strategy initially depends on WSQUOTA and WSEXTENT. These values are derived from the SYSUAF. When establishing a user’s account, we must make a conscious decision about the appropriate values for each user. The DCL command \$ SET WORKING_SET and the system service \$ADJWSL can raise or lower these limits for an interactive process.

Sub-processes and detached processes receive their working set characteristics upon creation by the system service \$CREPRC or the DCL command \$ RUN. If specific values are not provided, then a process will receive the default working set characteristics from the appropriate system parameters: PQL_DWSDEFAULT, PQL_DWSQUOTA, and PQL_DWSEXTENT.

When a batch queue is created, the DCL command \$ INITIALIZE/QUEUE establishes the default values for its jobs. These values may even be set to defer the user’s UAF limits. When a batch job runs, these values may be altered via the DCL commands \$ SUBMIT and \$ SET QUEUE/ENTRY.

WSQUOTA should be large enough so that the process can perform reasonably well without a loan, yet small enough so that any single process is not guaranteed an inequitable share of memory when memory is scarce.

The general scheme is to set the initial working set limits on a rule-of-thumb basis, and then to adjust them based on observed behavior. Experience counts!

Working set limits for user programs depend on the code-to-data ratio of the program and on the amount of data in the program. Programs which are mostly code—those which have a limited amount of data or use RMS to perform record processing—require only small working sets. Programs which manipulate large amounts of data internally—such as sort procedures or librarians—require larger working sets.

Via Autogen, set WSMAX to the highest number of pages required by any one process on your system.

In Authorize, for each user set WSQUOTA at the largest number of pages required by a program the user will run interactively. Set his WSDEFAULT to the median number of pages required by a program he will run. Set WSEXTENT to the largest number of pages you anticipate this user will ever need.

Also, for each user set a diskquota via the Diskquota utility. Spread default directories across all available drives (except the system disk). Use ACLs judiciously.

In Systartup, for each batch queue, set WSEXTENT to the largest number of pages required. Set its WSQUOTA to the number of pages which will permit every job to complete within a reasonable amount of time. Set WSDEFAULT at the median number of pages required by all jobs.

This scheme forces users to submit large jobs for batch processing, because the jobs will not run efficiently interac-

tively. (You can force this via ACLs which disallow their interactive usage.) To further restrict interactive users, you can impose CPU time limits in the UAF.

Adjusting AWSA parameters

AWSA depends heavily on the following system parameters:

```
PFRATH WSINC
PFRATL WSDEC
QUANTUM GROWLIM
AWSTIME BORROWLIM
AWSMIN FREELIM
```

PFRATL and WSDEC, which control voluntary decrementing, are particularly sensitive to the application workload.

For certain values of PFRATH and PFRATL, poor system performance may result because of the VMS page-replacement algorithm and because of the time spent maintaining these page-faulting limits.

You can turn off borrowing for a process by setting its WSEXTENT equal to its WSQUOTA. You can circumvent AWSA entirely by using the \$ SET WORKING_SET/NOADJUST command. This command may sometimes cause processes to fault badly. And, you can turn completely turn off AWSA for your system by setting WSINC = 0.

DEC recommends that the AWSA parameters, as set by Autogen, correctly match your operational needs and should be left alone. Before you start changing them, read Autogen carefully and document why you want to do so, as well as what system behavior you predict will occur. (You might be pleasantly surprised at the results, as well as mildly amused!) Always track your changes via a set of interlinked DCL command procedures and archive your system parameter files for historical purposes.

Tuning strategies

If you would like to provide a rapid response by VMS whenever the load on your system demands greater working set sizes—for example, in time-sharing and development environments...

- Set PFRATH low or = 0
- Set AWSTIME low
- Start processes with small WSDEFAULTs
- Provide for either large WSQUOTAs or generous loans, by setting BORROWLIM low and defining large WSEXTENTs.

To tune for a less dynamic response which will stabilize and track moderate needs for working set growth—particularly for production environments...

- Establish moderate values for AWSTIME, WSINC, and PFRATH
- Provide generous WSDEFAULT' s.

Swapper trimming

Swapping consists of writing a process from memory to a reserved disk file, the swap file.

If process requirements dictate, VMS will “swap out” processes to a swapping file on disk so that the remaining processes have the benefit of its available memory without excessive page-faulting. The operating system can also reclaim memory via “swapper trimming”, which is performed by the Swapper.

Swapper trimming is initiated when VMS detects too few pages on the free-page list—that is, whenever the number of free pages falls below the system parameter FREELIM. VMS takes action to obtain **at least as many** pages as specified by FREEGOAL.

First, it checks to see whether the minimum number of pages exists in the modified-page list which make it worthwhile to write them out. (VMS does the comparison to MPW_THRESH.) If the minimum exists, VMS invokes the Swapper to reduce the modified-page list—thereby freeing its pages for the free-page list. If the minimum does not exist, VMS decides to “trim” some processes—that is, it forces them to give back some pages or to be swapped out.

On the process level, the Swapper checks for processes which have borrowed on their WSEXTENT. These may be trimmed back to their WSQUOTA. If this initial trimming fails to produce a sufficient number of pages, because no or only a few loans were outstanding, then the Swapper trims on the second level.

The Swapper now refers to its system-wide trimming value, the system parameter SWPOUTPGCNT, which defines the minimum number of pages any process is permitted to retain in memory before it must be swapped out. The Swapper selects processes as candidates for trimming based on their state. When all the needed pages have been acquired, the Swapper stops its trimming.

Swapping

If trimming on the second level fails to produce enough pages, the Swapper starts to swap out processes from its list of likely candidates. Memory is first reclaimed from suspended processes, and then from dormant processes. The next likeliest candidates are processes in the LEF and HIB states.

Two criteria define a dormant process:

- The non-real time process has current priority less than system parameter DEFPRI (default = 4);
- The computable process without a significant event (page fault, direct or buffered I/O, CPU time allocation) within an elapsed time period, defined by the system parameter DORMANTWAIT (default = 10 seconds).

The Swapper also compares the length of real time a process has been waiting since entering HIB or LEF state to the system parameter LONGWAIT. VMS will hopefully be able to differentiate between those processes which have been idle for some time and are likely to remain idle, from those

which have not been idle very long and might be likely to soon become computable.

VMS selects for outswapping those processes which have been idle for the longest time. By freeing up their pages, the operating system permits *other* processes to satisfy their CPU requirements, so that they can resume execution sooner.

Swapper trimming can be costly if the Swapper trims pages which processes truly need: such processes are forced to fault heavily. To prevent this, you should determine a minimum working set size which permits some work to be performed reasonably efficiently, but below the peak efficiency value. Set SWPOUTPGCNT to *this* value.

You can turn off second-level swapper trimming by increasing SWPOUTPGCNT so that it is never permitted. If you notice excessive paging, you can eliminate it, *but* you force swapping to begin sooner.

For a process with PSWAPM privilege, you can turn off swapping and second-level trimming with the DCL command `$ SET PROCESS/NOSWAPPING`.

Note, though, that Swapper trimming is supposed to be more beneficial than voluntary decrementing. Autogen provides default values which provide for swapper trimming and disable voluntary decrementing.

Memory sharing

Memory sharing permits multiple processes to map to, and thereby gain access to, the same pages of physical memory. This is accomplished under VAX/VMS through the system-wide global page table.

The memory which is saved by sharing is calculated as follows:

```
(# pages of shared read-only code)
* (\# of sharing processes - 1)
```

A small amount of overhead is required: space for global page table entries and global section table entries.

The system parameter GBLPAGES defines the size of the global page table; GBLSECTIONS, the size of the global section table. The system working set size, SYSMWCNT, must be increased whenever GBLPAGES is: Autogen increments SYSMWCNT by 1 for every 128 pages you add to GBLPAGES.

Once a shareable image has been created, it can be installed as a permanently-shared image. Memory is only saved if more than one process is actually mapped to the image at one time. Of course, you must increase your users' working set characteristics—namely, WSDEFAULT, WSQUOTA, and WSEXTENT—to correspond to the anticipated use of shared code.

To determine if there is active sharing on your shareable images, use Install. Type LIST/FULL and observe the values shown for "Current" (the number of current concurrent accesses) / "Maximum" (the highest number of concurrent accesses since installation) shared access counts. If the maximum is less than 3, the overhead for sharing is excessive.

Note that the overhead required to share memory is counted in bytes, while the savings are counted in pages!

Scheduling

The VAX/VMS scheduler controls both when and how long a process executes: it dramatically affects CPU demand. It is important to maintain an even balance in the use of memory and the number of concurrent processes. Each process has an available amount of time to perform its task, its quantum.

Utilizing a modified round-robin scenario, the scheduler rotates control of the CPU among computable processes, so that all computing processes receive frequent and equitable chances to complete their processing requirements. For optimization, it permits operations to overlap; for example, if a process must wait for I/O to complete, another process will run.

Processes receive a chance to execute on a rotating basis, according to process priority and state. Each computable process receives a time-slice for execution equal to the system parameter QUANTUM. Once its quantum begins, each process executes until one of the following occurs:

- A process of higher priority becomes computable;
- The process is no longer computable because of a resource wait;
- The process itself voluntarily enters a wait state;
- Its quantum ends.

If no other process (at the same priority) is waiting to exercise its quantum, the current can renew its quantum and retain control of the CPU.

A change in process state causes the Scheduler to re-examine which process should be permitted to run. The Scheduler selects the computable process with the highest priority. Priorities are numbered 0 - 31, while real-time processes run above 15. (The Swapper runs at 16.)

For processes below priority 16, the Scheduler can increase and decrease process priorities. A "priority boost" is used as follows:

- The Scheduler recognizes events such as I/O completion or a duration of time.
- When such an event occurs and the associated process becomes computable, the Scheduler increases its priority: the amount is related to the event.

A large increase will be given in order to permit a process to run again sooner.

When a process is scheduled, its priority is reduced by 1—in order to permit processes which have received a priority boost to begin to return to their "base priority". (The priority is *never* decreased below this base or increased into the real-time range.)

VMS permits real-time processes to run until either it voluntarily enters a wait state or a higher-priority process becomes computable.

Base priorities

You can modify the base priorities of processes and the value of `QUANTUM`. All other aspects of process scheduling are fixed by the behavior of the Scheduler and the characteristics of your workload.

A process receives a default base priority from the "PRIO" field in the user's UAF record. A process can change its priority with the system service `$SETPRI`. With the DCL command `$ SET PROCESS/PRIORITY`, a user can reduce the priority of his own processes. (A user needs `ALTPRI` privilege to increase these priorities.)

Note: A user needs `GROUP` or `WORLD` privilege to change the priority of other users' processes.

A detached process or sub-process receives its base priority when created by the system service `$CREPRC` or the DCL command `$ RUN`. If none is specified, the priority of the creator is used.

When a batch queue is created, the DCL command `$ INIT/QUEUE/PRIORITY` establishes the default "job" priority. When a user submits a job with the DCL command `$ SUBMIT` or changes his job's characteristics with the DCL command `$ SET QUEUE/ENTRY`, he can adjust the priority downward. (If he has `OPER` privilege, he can also make increases.)

Diagnosis

Introduction

VAX/VMS performance suffers when there is a limiting resource. Which resource—memory, I/O, or CPU—becomes a bottleneck depends on the characteristics of the workload your system is supporting.

There are several simple tests we can use to rule out certain classes of typical problems. If the undesirable behavior is observable, you will be able to make headway fairly rapidly.

It is possible to have overlapping limitations. An iterative approach to diagnosing your VAX system's problems will detect all major limitations.

Note that if there is a significant amount of operating system overhead, the technical staff has probably mis-tuned your VAX system.

Technical assistance

Our software tool LINDA permits the technical user to pinpoint the causes of undesirable behavior on your VAX system. It does this by helping him to isolate particular kinds of performance problems, and by outlining what corrective procedures he can undertake, if any.

Of course, this same user must later monitor the effectiveness of any remedial action taken: if there is no sufficient improvement, he must again employ LINDA to analyze the situation all over again. Perhaps the changes made were too radical and should be scaled back to a more conservative level. Or, perhaps some significant problem had been masked by one

which has been solved, and now there is the need to correct this one.

We quickly come to realize that tuning is primarily an iterative process, and that multiple causes of performance problems can only be uncovered by repeated use of LINDA—until we achieve a satisfactory level of performance.

Unnatural Resources: Working sets, quotas, and limits

Steven Szep
Chase Manhattan Bank
1 New York Plaza
New York, NY 10081

Abstract

In a multi-user environment, the competition for limited system resources must be monitored and controlled. This paper presents a practical approach to workload management on VAX/VMS systems.

Essential Resources

The CPU Resource

The CPU is the central resource in the system: it allocates and initiates demand for all other resources, and it provides instruction service to user processes.

The system manager should observe the following:

- The average size of the compute queue.
- Idle time and process-scheduling wait states.

The memory resource

This resource can be separated into pieces of varying size. These can be allocated to processes simultaneously.

The system manager should observe the following:

- Working sets of resident processes, for appropriate sizes.
- Locality of reference, for an examination of application design.

The Disk I/O resource

The key performance issue is the time it takes to complete an operation. The measure to keep in mind is the average time to execute an I/O request on a particular disk. Keeping this average time low will minimize CPU blockage.

Tuning

Tuning can be defined as the alteration of various system parameters in order to improve overall system performance. Your goal should be to obtain the optimum overall performance possible for your configuration in your particular work environment. Before you start, you need a specific plan for how you will analyze and use the data you will capture.

Tuning cannot cure:

- Improper operation

- Unreasonable expectations
- Inadequate hardware configuration
- Improper device choices
- Hardware malfunctions
- Poor application design
- Inequitable distribution of resources
- Misuse of available resources
- Poor workload distribution.

Why does DEC believe that tuning is rarely required? Autogen establishes initial values for all configuration-dependent system parameters so that they match your particular **hardware** configuration. VMS includes several features which permit it to dynamically tune **itself** during operation. A **common** cause for poor system performance is insufficient hardware capacity.

When you acquire new capacity, you should re-tune your VAX. The same holds true if your workload drastically changes.

Tuning is complex, time-consuming, and prone to error. Fortunately, mis-tuned systems will exhibit symptoms with fairly obvious solutions.

System response time

Overall responsiveness depends on the responsiveness of our three essential resources: CPU, memory, and I/O. In other words, if each resource has adequate capacity, then the entire system will perform satisfactorily.

Our major concerns are:

- How well is each resource responding to service requests?
- How well is the capacity of each resource meeting demand?

- Does any resource have excess capacity? If so, can it be attributed to blockage by another, over-committed resource?

Our investigations will look measure the size of the queue of service requests – for examples, the compute queue or disk I/O queue, and the time it takes VMS to service one such request, which equals the response time for that particular resource.

Analysis

Is each resource shared equitably among processes?

Can the system's consumption of a resource be reduced?

(This will make available more of this resource to users.)

How well distributed is the demand for each resource?

Can some of the activity on one resource be off-loaded to less heavily used types of resources?

MONITOR

The VMS Monitor utility permits us to track several classes of system-wide performance data at specified intervals. Output can be sent directly onto a display terminal for an interactive session, or into a disk file for later playback. It is best to record data via a detached process running at a high priority.

Resource control via Authorize

Each VMS user is limited in the consumption of reusable system resources. We establish these limits when we set up our users' accounts on the VAX. These limits control how a process shares its resource allocations with any sub-processes it may create. We know that these fall into four distinct categories: deductible, non-deductible, pooled, and system-wide. Setting these limits takes a bit of experience. Use Digital's suggestions as a starting point. You will probably find reason to adjust them in the context of your system's evolution later on.

The relevant SYSUAF record fields are:

ASTLM	FILLM	SHRFILLM
BIOLM	JTQUOTA	TQELM
BYTLM	MAXACCTJOBS	WSDEFAULT
CPU	MAXDETACH	WSEXTENT
DIOLM	PGFLQUO	WSQUOTA
ENQLM	PRCLM	

Workload management

Know thy workload! System performance is directly proportional to the efficiency of workload management. The major issues facing the system manager are:

- What is the typical number of users at each time of day?
- What is the typical response time for various tasks for this number of users, at each hour of operation?

- What are the peak hours of operation?
- Which jobs typically run at which time of day?
- Which commonly run jobs are intensive consumers of the CPU? Of memory? Of disk?
- Which applications involve the most image activations?
- Which parts of the system software have been modified or are user-written?
- Are there any known bottlenecks?

Record retention

It is important to summarize your system's behavior under typical workload conditions. Track the following information:

- Page fault rate
- Terminal response time
- CPU usage
- Memory usage
- Operational (KESU) modes
- Number of jobs versus overall performance
- Number of users
- Performance versus time of day

Only with such a historical record will you be able to monitor trends and plan for necessary capacity.

The possible scenarios include:

- Is there a time of day when the workload peaks?
- Is there any way to balance the workload better?
- Could any jobs be better run as batch jobs?
- Could system performance benefit by adopting primary and secondary hours of operation?
- Can applications be designed to work around any known / expected bottlenecks?
- Is code-sharing being utilized, in order to conserve memory?

A common sense approach to workload distribution demands that we run large jobs as batch jobs, restrict system usage to maintain an adequate response time, and design applications to reduce demand on bottlenecks.

Memory management

Technical staff should investigate the use of VMS memory-management mechanisms which can help balance your workload and automatically adjust the system for better resource utilization.

Under VAX/VMS, each process has an independent address space. At any given moment in time, some of its pages are actually resident in physical memory, while the remainder are located in disk files.

VMS manages all processes in several ways:

- Which ones are running in memory;
- How much memory each process can use;
- Which are not in physical memory.

The following concepts are relevant when we discuss VMS memory management:

Balance set Process currently resident in physical memory

Working set The pages of a process currently in physical memory

Working set size The number of pages a process is permitted in physical memory

Working set count The number of pages a process currently has in physical memory

Paging The action required to move a page into or out of a process's working set

Swapping The action required to move a process into or out of the balance set

Page cache The free-page list and modified-page list

A little research will show you that a specific page can be in one of several places:

- In the process's working set
- In an image file (not yet paged in, read-only, discarded after use)
- In the free-page list and available for re-use
- In the modified-page list
- In a system page file (modified, paged out)
- In a system swap file (swapped out).

A page table entry (PTE) contains the location for one page belonging to the process. A given page can:

- Be read in from the paging file
- Undergo a change of state (free or modified), if in cache
- Be initialized, if demand-zero.

Alternatively, a page-fault may:

- Remove a page from a process's working set
- Read additional pages into physical memory
- Wait for more physical memory to become free
- Wait for a write to the modified-page list.

Note that global pages are treated differently. Modified-page writing is itself triggered when:

- The modified-page list is too large
- The free-page list is too small
- Forcing specific pages from physical memory.

The requisite pages may be written to a system page file, a writeable section file, or a system swap file. After the write, the relevant pages are moved onto the free-page list.

The Swapper actually controls these operations. The sequence of actions taken is as follows:

1. Check if free pages are needed
2. Write modified pages, if sufficient
3. Swap, if not enough modified pages
4. Check for in-swap candidate(s)
5. Swap, if not enough free pages.

When an out-swap is required, the Swapper performs these operations:

1. Clean up process headers
2. Check if enough free pages
3. Write modified pages, if sufficient
4. Swap or shrink processes.

Resources for file applications

To use RMS effectively, an application program requires various process and system resources. The appropriate time to configure these resources is during the system design phase: your site might need additional physical memory and/or disk storage to support the new software.

Improving program performance may simply be a matter of allocating larger or more buffers to the applications. The number of buffers and the sizes of buckets and blocking factors can be fine-tuned according to access method.

When a file is created or opened, RMS maintains the specified buffers and data (control) structures, which are charged against the process – via the XQP mechanism. Naturally, the greater the number of files being processed at the same time, the greater amount of memory is required for a specific application. In fact, although the memory requirements for the control structures is fairly constant, buffer memory varies, once again, according to access method.

The process working set is governed by the three process parameters `WSDEFAULT`, `WSQUOTA`, and `WSEXTENT`. It is important to set these correctly in order to ensure a process has sufficient memory to perform the required tasks with a minimum amount of paging.

Shared files should employ global buffers to avoid unnecessary

I/O. The relevant system parameters are: `RMS_GBLBUFQUO`, `GBLSECTIONS`, `GBLPAGES`, and `GBLPAGFIL`.

In the case of asynchronous record I/O, the following `SYSUAF` limits are also important: `ASTLM`, `BIOLM`, and `DIOLM`. Additionally, check `ENQLM` when you have shared files which will be modified at the record level. Finally, note that `FILLM` governs the number of files which a process can have open simultaneously.

File extensions

If a file is extended repeatedly, the extensions may be scattered on the disk. Each extension is called an “extent”; a pointer to each extent resides in the file header. For retrieval purposes, the pointers are gathered together in a structure called a “window”.

When you access an extent whose pointer is not in the current window, the system must read the file header and fetch a new window. This is called a “window turn”. It requires an I/O operation.

Access control lists

ACLs are stored in the file header. The more ACLs you place on a file, the greater becomes the possibility that the file header will have to be extended to accommodate them.

Obviously, applying multiple ACLs on a particular file will impact system performance.

Therefore, place ACLs on all objects with discretion. Use general identifiers to create practical groupings in order to avoid this potential problem.

Application-level performance

The hardest part of tuning is convincing a group of developers that they have to re-think an application’s design in order for you to effectively correct performance problems.

Helping your developers to adopt sound programming practices will result in a performance win for everyone.

Utilizing VAX Uptime

Steven Szep
Chase Manhattan Bank
1 New York Plaza
New York, NY 10081

Abstract

This paper presents a DCL approach to VAX/VMS system tuning. The use of the system utilities MONITOR, SHOW, and SET for performance analysis is illustrated.

Why performance is a critical issue

Hardware is getting cheaper all the time. Software is becoming ever more expensive. People are more expensive yet.

If you can figure out what actually is causing a performance problem, you can find a remedy for it. It is simply a matter of separating the real problem from all the noise.

Some notes on resource limitations

Memory

A large Free List means less paging I/O, but less space for the balance set.

You should control system faulting via the system parameter `SYSMWCNT`. A `HIGH` value removes memory from that available for the user working sets; a `LOW` value may impair VMS performance. In other words, a little faulting never hurts.

The number of outswapped processes is NOT relevant if they are not trying to get inswapped.

Image activations

Image activation under VAX/VMS involves considerable overhead: frequent image activations in a process cause excessive page faulting. The number of demand- zero faults gives a measure of image activations.

Paging induced by image activations cannot be "tuned away". More memory will not help: application re-design is the only alternative.

Page caches

If your overall fault rate is high—once again, this is relative to your own experience—and these are mostly "soft" faults, the page caches may be too large. The page cache (Free List + Modified List) swallows up memory which should be made available to the user working sets. Swapping may also appear.

However, if your overall fault rate is low and these are mostly "hard" faults, the caching is ineffective. Either one

or both are too small. The user working sets, however, have sufficient memory.

Sizing working sets

If your VAX is short on memory, you must re-distribute the working set sizes so that active processes get more memory relative to those which are more inactive.

The working sets of processes which are faulting heavily and are relatively small require more memory for the tasks they are trying to perform. The working sets of processes which have a low fault rate are probably too large for their needs.

If you install more memory, increase the working set parameter values for your heavy faulters.

Automatic working set adjustment (AWSA)

System default values for these parameters should not be modified by novice tuners. If they are out of adjustment, excessive paging and/or rapidly fluctuating working set sizes will occur.

If there is either a large amount of free memory which is not available to the user working sets or many processes are in their `WSEXTENT` regions while others have been outswapped, then you have inappropriate memory loan parameters.

Swapper trimming is more beneficial on most VAX computer systems than is voluntary decrementing because swapper trimming occurs as needed, while voluntary decrementing occurs continuously.

The balance set

If the balance set count is too low, you will see some processes swapped out even if there is free memory available.

Big consumers

Processes with extremely large working sets—either with a `WSQUOTA` which is too large or within their `WSEXTENT` region—may cause other (smaller) processes to swap.

Low-priority compute-bound processes seem to get preferential treatment over ones which handle terminal I/O.

Also, inactive processes—that is, those with no faulting—having large working sets may have swapping disabled. They retain memory at the expense of all other processes.

Introduction to MONITOR

The VMS Monitor utility permits us to track several classes of system-wide performance data at specified intervals. Output can be directly onto a display terminal for an interactive session, or into a disk file for later playback. It is best to record data via a detached process.

MONITOR SYSTEM

The module collects quite a few performance statistics from several of the other components of the VAX/VMS Monitor utility, into a single display.

At a glance, the system manager can see an overview of the activity on the system. This display is similar, in fact, to the DISPLAY component of Digital's SPM product.

Alternatively, one might select `MONITOR ALL/INTERVAL=2`.

The SYSTEM display provides information on:

- Process states
- Page fault rate, including the top faulter. Page faults are indicative of process consumption of physical memory
- Direct and buffered I/O rates, including the top consumers. Direct I/O primarily involves disks and magnetic tapes; buffered I/O involves terminals and like devices.
- Free List and Modified List sizes. These are the VMS page caches.
- CPU busy rate, including the top user process

Note that the NULL process is never displayed here, even if it is a top consumer of a system resource.

A word of caution is appropriate at this juncture. To be eligible for consideration as the “top” consumer, a process must be present and swapped in at **both** the beginning **and** end of the display interval.

The so-called “Top User” statistic is always the *current* one, while the corresponding *overall* statistic may be current, average, minimum, or maximum—depending upon the option typed as part of the command line.

The rates for “top” consumers are calculated based on the user-selectable interval between two successive screen displays. The “overall” rates are based upon the fixed collection interval of the Monitor utility. Therefore, these two values may be different.

The calculations for the page caches—that is, the Free and Modified Lists—are derived from the VAX system's physical memory configuration and the VMS system parameters.

In effect, the SYSTEM display gives you a general “feel” for what's going on “under the hood”.

If the Free List drops to the value of the system parameter `FREELIM`, then you are essentially out of memory—perhaps

because of working set sizing or even because of a lack of actual physical memory.

Check processes which are top consumers of I/O.

It is important to keep in mind that Monitor can only give you an “eyeball” measure of system activity. You will have to resort to SPM or PCA for a more intimate look at a specific image running in a specific process.

MONITOR STATES

This module will display the number of processes in each of the 14 scheduler states:

- Collided Page Wait (**COLPG**) - Waiting for a faulted page in transition.
- Mutex or Miscellaneous Resource Wait (**MWAIT**) - Waiting for the availability of a mutex semaphore or a dynamic resource.
- Common Event Flag Wait (**CEF**) - Waiting for some combination of event flags to be set in a common event block.
- Page Fault Wait (**PFW**) - Waiting for a page to be read because of a page-fault: resident processes.
- Local Event Flag Wait (**LEF**) - Waiting for some event flag(s) to be posted: resident processes.
- Local Event Flag Outswapped (**LEFO**) - Waiting for some event flag(s) to be posted: outswapped processes.
- Hibernate (**HIB**) - Hibernating: resident processes.
- Hibernate Outswapped (**HIBO**) - Hibernating: outswapped processes.
- Suspended (**SUSP**) - Process has executed a suspend request: resident processes.
- Suspended Outswapped (**SUSPO**) - Process has executed a suspend request: outswapped processes.
- Free Page Wait (**FPW**) - Waiting for a free page of memory.
- Compute (**COM**) - Ready to use the processor: resident processes.
- Compute Outswapped (**COMO**) - Ready to use the processor: outswapped processes.
- Current Process (**CUR**) - Using the processor.

The following is a list of the possible **MWAIT** states:

RWAST	Wait for system/kernel AST
RWMBX	Mailbox full
RWNPG	Nonpaged dynamic memory
RWPGF	Page file full
RWPAG	Paged dynamic memory
RWBRK	Breakthrough (wait broadcast)
RWIMG	Image activation lock
RWQUO	Job quota
RWLCK	Lock database
RWSWP	Swap file space
RWMPE	Modified page-list empty
RWMPB	Modified page-writer busy
RWSCS	System Communications Services wait
RWCLU	Cluster state transition wait

Keep in mind that the **CURrent** process is always the process running Monitor.

LEFO processes normally belong to interactive users who have been prompted but who have not responded; however, they might also be processes waiting for disk I/O on a busy system.

The **COMO** state indicates a very crowded system.

Wait states

Voluntary wait states

Most processes in **LEF** are waiting for terminal command input—that is, they are at the **DCL \$** prompt.

A process may enter **LEF** while awaiting I/O completion on a disk or other peripheral. If the I/O resource is approaching capacity, this type of waiting can cause the CPU to be underutilized.

A process in **LEF** may be waiting for a lock to be granted—especially where extensive file sharing exists. Check “**ENQs Forced to Wait Rate**” in the **MONITOR LOCK** display, which gives the rate of lock requests forced to wait before the lock was granted.

A process may enter **LEF**, **CEF**, **HIB**, or **SUSP** when system services are used to synchronize applications. This is a temporary abdication of CPU usage.

Involuntary wait states

These wait states are invoked by VMS to achieve process synchronization in special circumstances.

FPW, **PFW**, and **COLPG** are associated with memory management.

MWAIT indicates a shortage of a system-wide resource—usually page or swap file. This shortage blocks the process from the CPU.

RWSWP relates to the swap file. **RWMBP**, **RWMPE**, and **RWPGF** relate to the page file.

RWAST indicates a process waiting for a resource, the availability of which will be signaled by delivery of an **AST**. Usually, an I/O is outstanding or a process quota has been exhausted.

Note that **MUTEX** is a temporary state.

MONITOR POOL

This display reveals the consumption of the VMS lookaside lists—that is, Small Request Packets (SRPs), Intermediate-sized Request Packets (IRPs), and Large Request Packets (LRPs).

MONITOR DISK

The goal is to balance I/O across all drives—based upon their relative I/O operation rates. (Keep in mind that you must check the individual drive specifications.)

MONITOR FCP

You want to see **high** hit rates here. If you are using the RMS file system extensively in your applications, this will be important to you.

MONITOR PAGE

We want to see a low page fault rate and a very low system fault rate. Keep in mind, however, that some paging is better in most cases than almost any level of swapping activity.

MONITOR PROCESS

If the general *response time* on your VAX/VMS system is low, investigate the top five processes listed for **/TOPCPU**.

If your general *throughput* is low, investigate further the top five processes listed for **/TOPFAULT**.

SHOW

The Show utility gives system managers a “one-shot” display of specific system information.

SHOW SYSTEM

This display will present general information about all processes running on your VAX system.

SHOW MEMORY

This display provides us with the availability and usage of memory resources on the VAX system. It is useful in “eye-balling” conditions related to potential or actual memory limitations.

SHOW ERROR

This display provides error counts for all hardware devices with error counts greater than zero. It is especially useful when a process utilizing a specific device begins to display erratic behavior.

SHOW NETWORK

This display gives us the DECnet addresses and node names currently accessible to your local DECnet node. It provides quick verification of the active link relationships in your network.

SHOW USERS

This option will display identifying information about interactive users.

Please note that you can use PHONE (PHONE: DIR) for remote nodes.

SHOW WORKING_SET

This option will display working set information about your current process.

SET

The SET utility allows you to change characteristics of objects within the VMS operating environment.

SET PROCESS

This VMS utility permits us to change the execution characteristics of a specified process. As system manager, you will often find yourself in predicaments from which this SET command is your only escape.

We may use SET PROCESS as the result of a session with the Linda performance tool.

The available options are:

/RESOURCE_WAIT Enables or disables resource wait mode.

/PRIORITY Changes the process's priority.

/SUSPEND Places the process in a suspended state.

/RESUME Resumes a previously suspended process.

/SWAPPING Enables or disables process swap mode.

SET WORKING_SET/LOG

This VMS utility permits us to re-define working set characteristics of a process. This SET command becomes handy in emergency situations, which is the system manager's "SOP" all the time.

We may use SET WORKING_SET as the result of a session with the Linda performance tool.

/ADJUST Enables or disables modification of the process's working set by VMS. Note: The default is /ADJUST.

/EXTENT=n Specifies the maximum number of pages which can be resident in the process's working set during image execution, where MINWSCNT (in VMS) is less than n and n is less than WSEXTENT (in SYSUAF). Note: If n is greater than WSEXTENT, then VMS sets n = WSEXTENT .

/LIMIT=n Specifies the size to which the process's working set is to be reduced at image-exit time. Note: If n is greater than QUOTA then VMS sets QUOTA = n .

/QUOTA=n Specifies the maximum number of pages which any image executing in the context of this process can request (via the \$ADJWSL system service). Note: If n is greater than WSQUOTA (in SYSUAF), then VMS sets n = WSQUOTA .

Notes on MTOOLS

MONV4.COM

MONV4 uses the System Monitor program to record system-wide performance data. Uses MONVMS4.COM .

SHWS.COM

SHWS displays current working set information.

WHAT.COM

WHAT displays information about interactive processes.

```
$! name: monv4.com
$!
$! author: s. szep
$!
$! date: 06/24/86
$!
$! purpose: Submit MONVMS4.COM as a detached process
$!   to initiate continuous recording for
$!   the current boot.
$!
$! Submit detached MONITOR process to do continuous
$! recording.
$!
$ run sys$system:loginout.exe -
/ uic=[1,4] -
/ input=monvms4.com -
/ output=tmon.log -
/ error=tmon.log -
/ process_name="Monitor" -
/ working_set=100 -
/ maximum_working_set=100 -
/ extent=512 -
/ noswapping
$!
$! End of MONV4.COM
$!
```

```
$! name: monvms4.com
$!
$! author: s. szep
$!
$! date: 06/24/86
$!
$ set default sys$update
$ set noon
$!
$! Begin recording for this boot. The specified
$! /INTERVAL value is adequate for long-term summaries.
$!
$ set process/priority=15
$!
$ monitor/int=5/nodisp/rec=tmon.dat -
/ ending="+23:59:00" all
$!
$! End of MONVMS4.COM
$!
```



```

$! name: shws.com
$!
$! purpose: display working set info
$!
$! note: requires WORLD privilege to display other processes
$!
$ set nover
$ set term/wid=132
$!
$ a = ""
$ pid = ""
$ context = ""
$!
$ if p1 .NES. "" then pid = p1
$!
$ write sys$output -
"                               Working Set Information"
$ write sys$output " "
$ write sys$output -
"                               WS      WS      WS      WS      Pages      Page"
$ write sys$output -
"Username  Processname  Stat Extnt Quota Deflt  Size  in WS  faults  Image"
$ write sys$output " "
$ start:
$ if p1 .EQS. "" then pid = f$pid(context)
$ if pid .EQS. "" then goto done
$ pid = a + pid + a
$ username = f$getjpi('pid,"username")
$ if username .EQS. "" then goto start
$!
$ processname = f$getjpi('pid,"prcnam")
$ imagename = f$getjpi('pid,"imagname")
$ state = f$getjpi('pid,"state")
$ wsdefault = f$getjpi('pid,"dfwscnt")
$ wsquota = f$getjpi('pid,"wsquota")
$ wsextent = f$getjpi('pid,"wsextent")
$ wssize = f$getjpi('pid,"wssize")
$ globalpages = f$getjpi('pid,"gpgcnt")
$ processpages = f$getjpi('pid,"ppgcnt")
$ pagefaults = f$getjpi('pid,"pageflts")
$ pages = globalpages + processpages
$ text = f$fa0("!AS!15AS!5AS!5(6SL)!7SL!AS", -
username,processname,state,wsextent,wsquota,wsdefault, -
wssize,pages,pagefaults,"    " + imagename)
$ write sys$output text
$ if p1 .NES. "" then goto done
$ goto start
$!
$ done:
$ write sys$output " "
$ inquire ask1 "Hit any key to continue"
$ set term/wid=80
$ exit

```

```

$! name: what.com
$!
$! define a local symbol
$!
$ wso == "write sys$output"
$!
$ context = ""
$ privs = f$privileges("group,world")
$ if .not. privs then goto noprivs
$ time = f$time()
$ syi = f$getsyi("version")
$ syi = f$extract(0,4,syi)
$ wso " "
$ wso "VAX/VMS ''SYI' users on ''TIME'"
$ wso " "
$ wso " Username      Process Id      Terminal      Image"
$ wso "-----"
$ if syi .ge. "4.0" -
then goto vfour ! V4.x has different end sequence
$ more_processes:
$ ipid = f$pid(context)
$ proc = f$getjpi(ipid,"prcnam")
$ if proc .eqs. "ERRFMT" -
then goto normal_exit ! V3.x has ERRFMT at end
$ term = f$getjpi(ipid,"terminal")
$ if term .eqs. "" -
then goto more_processes
$ user = f$getjpi(ipid,"username")
$ imag = f$getjpi(ipid,"imagname")
$ pid = f$extract(0,8,ipid)
$ if imag .eqs. "" -
then imag = "      Using DCL or idle"
$ wso ""      ''USER' ''PID'      ''TERM'      ''IMAG' "
$ goto more_processes
$!
$! VMS 4.x
$!
$ vfour:
$ start:
$ pid = f$pid(context)
$ if pid .eqs. "" -
then goto normal_exit
$ proc = f$getjpi(pid,"prcnam")
$ term = f$getjpi(pid,"terminal")
$ if term .eqs. "" -
then goto start
$ user = f$getjpi(pid,"username")
$ imag = f$getjpi(pid,"imagname")
$ pid = f$extract(0,8,pid)
$ if imag .eqs. "" -
then imag = "      Using DCL or idle"
$ wso " ''USER' ''PID'      ''TERM'      ''IMAG' "
$ goto start
$!
$ noprivs:
$ normal_exit:
$ wso " "
$ exit

```


Risk assessment in system security: A software implementation

Steven Szep
Chase Manhattan Bank
1 New York Plaza
New York, NY 10081

Abstract

This paper will describe ideas and techniques which have been found to be useful in designing a software-based risk assessment module, code-named "Morisot", for enhanced system security on VAX/VMS systems. It should be noted that these methods are also applicable to other fields, such as customer credit evaluation and market trend analysis.

Introduction

Your VAX system is vulnerable to potential abuse for exactly the same reasons that DEC equipment was probably selected by your organization in the first place: an easy-to-integrate open architecture, compatibility across an entire range of user-tolerant processors, and an abundance of clustering and networking hardware and software options.

Once upon a time, security was a simple matter of controlling physical access to a secluded computer room. It is now a matter of corporate significance.

Imagine the possibilities...A dishonest employee can copy and remove proprietary programs and confidential data. A hostile employee can delete important files or even reformat (erase) entire volumes. A malicious intruder can wreak havoc in many ways.

Employees are more professional when provided with a professional environment. Management control is better established through motivation, than through regulation. People first, yes; but tempered by a dose of healthy paranoia.

VMS already provides us with utilities which can monitor system usage, audit users' actions, and issue alarms. Risk management requires that special care be applied in the following situations:

- whenever a potential user attempts to login;
- whenever a user attempts to run a program;
- whenever a program attempts to access a file or device.

Intelligent oversight requires making sense out of the available information and then making decisions based upon this information.

This paper will focus on the latter two situations.

Risk indexing

We begin by indexing the impact of the several possible types of risk/exposure and then proportionately weight our security measures against them.

One, admittedly simple-minded, approach is explained as follows.

If what we lost is *not* primary to our business, the impact becomes merely an annoyance.

If what we lose is necessary to maintain day-to-day business activities, then the impact will be a major annoyance or a minor loss.

If the damage does not make it possible to obey statutory regulations, then the impact expands into a major disruption.

If our loss threatens the very survival of our company, then we will surely suffer a severe disruption of business—that is, a disaster.

Actual risk analysis can be broken down into four components:

1. Identification of risks. For example, the interruption of services or the loss of information.
2. Identification of threats. We will consider this matter when we define our "enemies".
3. Recognition of vulnerabilities. These are flaws in security which can be exploited by the threats of unauthorized entry and access.
4. Documentation of controls Not for corporate-wide exposure: "need to know" only.

The intruder who impersonates a valid user must be prevented from altering data items or data flows within the penetrated system.

An insider who attempts abuse or sabotage must be rapidly exposed.

Trojan horses—or, “code bombs”—must be detected and crushed while the damage they cause is still slight.

In all three cases, rapid containment of the criminal is the primary concern.

We must specially protect those parts of our system which are of major importance. Our risk/exposure-versus-potential impact scenarios apply here.

As a matter of course, we must maintain useful, and complete, audit trails.

To be successful, we must balance user-friendliness and productivity against strict security.

We should prefer to mis-identify a valid user as a possible intruder, *rather than* to permit a real terrorist free-ranging access to our VAX computer systems.

Methods of user authentication

Effective user authentication seems to be an elusive goal. Any access management program has a four - fold purpose:

1. Keep out unauthorized persons.
2. Log who was where, when, and for how long inside the system.
3. Keep in the right people, so that they can get their work accomplished.
4. Deter theft and destruction.

We here list some user authentication methods. (Note that each has its own limitations and problems.)

- Prearranged information in the user’s possession. Examples: passwords, PIN’s, or the magnetized strip
- Piece of hardware in the user’s possession. Examples: a metal key or a computer chip
- Personal feature of the user. Examples: fingerprint, retinal pattern, speech pattern
- Real-time capabilities of the user. Examples: signature dynamics, typing style, facility in using a mouse
- Customary ways of doing things. Example: computer usage pattern
- Skills and information the user possesses. Examples: personal biography, some project history

Morisot combines the last two of these methods to form the foundation of her intrusion - detection model.

Vermin

Who are our enemies? Well, there is the Trojan Horse, a seemingly harmless program which sometimes takes the name of a familiar program. It works in a gentle way until some modified code fragment is activated and files are destroyed.

Next, we have the Virus. This is a small program which evolves, sets out to weaken the effectiveness of the system for doing productive work. It can potentially take control of all system resources.

Also, there is the Worm. This is a sophisticated probe which will wander through a network and enter any nodes with weak defenses.

Less destructive, but no less criminal, is the Browser. This kind of program gathers information by searching—randomly or intelligently—through your file system until it finds something it considers worth stealing.

Next on our roster is the Impostor. This is someone who discovers the username and password of a duly-authorized user and then logs onto the system as that user. It is sad to note that, in many cases, this critical information is revealed to the future impostor by the user himself.

Finally, we have the Snooper. This is a program which resides rather quietly in memory—taking snapshots of user activity for surveillance purposes. This villain typically takes the form of an unauthorized monitor or “spy” program.

Normally, the (discretionary) VAX/VMS security mechanisms are adequate in keeping unauthorized users out. However, code bombs can lead to real headaches. As a result, code reviews and code management are painful but absolutely necessary.

We are talking about billions of dollars, the privacy of clients, and the lost competitive advantage when a strategic plan becomes public. A supposedly “harmless” prank by a hacker or a premeditated attack by a vengeful person can make life miserable for your data center personnel.

Frame technology

A structure gathers together several pieces of information into a fixed pattern. If we think of a structure as a box with many pigeon-holes, then each hole, or “slot”, may contain such pieces of information.

Each slot is labelled according to the type of information it contains. These labels are the “attributes” of this structure. The pieces of information placed inside these slots are their “values”. When values are placed into these slots, an actual database object comes into existence.

Thus, we see that a “structure” defines attributes important in establishing the description of something, while an “object” is a specific instance of that structure. We can think of a structure as a “plan” for the construction of an object. To borrow a bit from Marvin Minsky, we will call our structures “frames”.

Before we proceed, it is time to present some necessary definitions.

Action Performance of a particular task.

Attribute Property of an object, stored in slots in structures.

Frame Knowledge representation of object’s structure.

History Chronological record of significant events.

Inheritance Characteristics of object become those of another.

Instantiation Specification of particular values.

Knowledge base Rules, facts, strategies pertinent to one domain.

Location Source of the trigger.

Log Record of all actions since last system boot.

Monitor Software which oversees risk management.

Object Conceptual entity with multiple attributes.

Script Strategy based upon pre - defined situations.

Slot Storage area associated with object's attributes.

Structure Knowledge representation of an object.

Summary Historical abstract of a user's activity.

Time Occurrence of a trigger.

Trigger Activity which invokes security monitor.

Value Information placed within slots in frames.

Automated reasoning

Morisot is a specific example of "memory-based reasoning" (MBR). MBR requires the recall of stored episodes from the past: there are no "rules" because Morisot works directly from the database. Simply put, MBR makes its decisions by looking for patterns in this data.

Why have we selected MBR as the foundation for our risk manager? (1) When we conceive of "thought" or "experience", we relate it to "memory". (2) Justifications arrived at via so-called "common sense" or from the "obvious" are probably based upon undigested memories of past experience. (3) There is no dependence upon a (restrictive) "domain" model. (This open-endedness will lead to rapid database growth.) (4) There is no requirement for going through a delaying "knowledge-acquisition" phase. (5) The system should degrade gracefully when it cannot arrive at a definitive answer to a problem. (6) Its simple architecture has the happy side-effect of a rapid software development cycle.

The disadvantages of MBR are perhaps too obvious: (1) The database can be arbitrarily large, and searching it can become quite time-consuming. (2) There is no general approach known for searching memory for the "best match" without examining every element of memory. However, we plan to take a relational approach and employ "summarization" strategies. (3) The fundamental "process" is only a collection of weak methods which apply to an extensive memory.

For Morisot, automated risk assessment is accomplished by searching her (relational) database of previous authorization and authentication "problems" for the best match to the current user's situation—that is, she must judge how closely any two of these situations or "patterns" match. Note that the

time to make one decision may become unacceptably large for a sizable database—unless we "tune" our algorithm by restricting our use of the database to a clearly-defined—and, theoretically, dynamically created—subset.

Complexity seems to be inherent in MBR. However, we think we have a solution. We must first separate important "features"—or, attribute/value pairs—in a situation from the unimportant ones. We then come to realize that what becomes important is usually context-sensitive. This fact itself makes it impossible to assign a single "weight" to each feature which will remain constant for the life of the system: these must be re-calculated dynamically with respect to the actual task at hand.

Algorithms

Morisot possesses a large relational database of user history objects, or UHO's. This database has been established by permitting all users "reasonably free" access to the system—in order to "build up" Morisot's memory. We have identified the following data elements:

User process which requests access

Resource the system component being requested

Date calendar time

Event time system time

Action the desired method of access

Response the result of Morisot's risk assessment

Reason why the risk assessment took this form

Note that the first five "slots" represent "predictor" attributes, while the last two have become the "goal" attributes.

Our risk assessment routines will attempt to fill in the "Response" slot for this "User" as follows:

1. Extract from the database the values for each of this user's predictors;
2. Compute a numerical "distance" and a numerical "weight" for each possible predictor, from the current user's (frame) value for each predictor;
3. Use these derived weights and distances to compute a "total distance" measure; and,
4. Finally, select the "n" database records whose total distances from the current user (frame) are the smallest.

Naturally, there are several possible outcomes:

1. No database record is sufficiently similar to the current situation to make the authorization decision.
2. A small number of database records, but hopefully more than one, are retrieved.

3. A significant number of database records for whom authorizations occurred in a similar way in the past are retrieved.
4. There are several authorization patterns among the “n” nearest database records.

In case 1, Morisot realizes that she has never seen this set of events before: she does not know how to handle this user’s request for access to a resource. If this is a reasonable request, we shall grant access, and add this situation to her database; otherwise, we will respond as in case #4.

In case 2, the program may be able to make a tentative decision—even if this is only the second occurrence of this specific situation.

In case 3, our program will very likely make the same decision again—that is, either grant or deny this user’s request.

In case 4, Morisot knows that she cannot come up with a definitive decision: she requires more info, so she invokes the biography prober (during interactive requests) or halts the system (for suspected Trojan horses and worm programs). In either case, an alarm will be automatically transmitted and Morisot’s memory will receive a record of this response.

In summary, Morisot will attempt to make her decisions by “remembering” similar circumstances from the system’s past. She performs this sorcery by counting combinations of features, using these sums to produce a “metric”, using this metric to find any dissimilarity between the current “problem” and every item in the “memory” database, and finally retrieving the “best matches”.

Extreme care must be taken to avoid the following situations:

- No decision is made;
- Data is stored erroneously;
- All the relevant data is not considered.

Disciplined programming and rigorous testing will make sure these problems are resolved in the early stages of implementation.

We must try to complete the risk evaluation rapidly—in order to terminate the activities of a Trojan horse or virus with due speed, without failure.

Data architecture

The contents of Morisot’s database can be classified according to whether they are symptoms of situations, or outcomes of situations, or of a miscellaneous variety (for example, any of the proposed summary and daily objects). In our frame representation for Morisot’s information structures, any empty slots are our “goals”, while the non-empty slots are our “predictors”.

We can restrict the database to a subset in two ways. In the case of predictor restriction, we find the most important attribute, as judged by its weight, and then restrict the database to those frames having the same value in that slot as the current user’s frame. For example, select only frames containing

a specific username/response pair. Note that the combined effect of two predictors is often quite different than their effect separately. This is the main reason why we must calculate weights dynamically.

For the case of goal restriction, we discover plausible values for the goal slot under consideration and then restrict the database to those frames containing one of these values in their goal slots. Note that in this case, weights must be assigned depending on how often they have occurred in the past.

The possible values for the action (“Response”) attribute are:

Probe (an evasion strategy which) attempts to verify the user;

Authorization leads to the granting of access;

Violation leads to an alarm.

The currently defined objects are described below.

<i>Resource History Object: RHO</i>	
Resource	O
Action	O
Date	O
Reason mask	O
User	O
Event time	S

Note: See RRO for implementation strategy.

<i>Resource Daily History Object: RDHO</i>	
Resource	O
Action	O
Date	O
Mode reason mask	O
User count	M
Usage count	M
Event mask	S

<i>User Daily History Object: UDHO</i>	
Resource	O
Action	O
Date	O
Mode reason mask	M
User	O
Usage count	M
Earliest event	M
Last time	M

<i>Resource Request Object: RRO</i>	
Resource	S
Action	S
Date	S
Reason mask	S
User	S
Event time	S
Violation mask	M
Response mask	M

Note: RRO becomes RHO if user authorized to proceed;
RRO becomes AO if user blocked.

<i>Resource Summary Object: RSO</i>	
Resource	S
Action	S
Earliest event	M
Last event	M
Avg. day usage	M
Min. day usage	M
Max. day usage	M
Mode day usage	M
Avg. day # users	M
Min. day # users	M
Max. day # users	M
Mode day # users	M
Mode reason mask	M

<i>Alarm Object: AO</i>	
Resource	S
Action	S
Date	S
Reason mask	S
User	S
Event time	S
Violation mask	M
Response mask	M

Notes: See RRO for implementation strategy.
"User" attribute can have USERNAME's for which there is no entry in SYSUAF.

<i>User Profile Object: UPO</i>	
User	S
Reason mask	S
# attributes	M
Batch mask	S
Detached mask	S
Image mask	S
Interactive mask	S
Login fail mask	S
Network mask	S
Job mask	S
Print mask	S
Process mask	S
Subprocess mask	S
Others ???	-

Conclusion

Considering the available value of the predictors, Morisot will consult the subset of her memory "base" pertaining to one particular user's specific resource request and attempt to fill in the remaining ("goal") slots of the current (active) frame.

As this program moves off the drawing board and into the prototype phase, it is our hope that the quality of Morisot's decisions will *improve* as the (data)base of her memory of past experiences grows.

References

- Stanfill, C and Waltz, D, Toward Memory-based Reasoning, *Communications of the ACM* (December, 1986), pp. 1213-1228.
- Lobel, J, *Foiling the System Breakers*, (McGraw-Hill, 1986).
- Bequai, A, *Technocrimes*, (Lexington, 1987).
- Williamson, G. B., How Secure is Your Ethernet LAN?, *Pageswapper*, Vol. 8 No. 1, (August, 1986).
- Szep, S, Desparately Seeking Access, *Proceedings of the Digital Equipment Computer Users Society*, (Fall, 1985).
- Szep, S, Security Considerations for Network Access, *VAX SIG Session Notes* (Spring, 1986).

Linda: A Tuner's Home Companion

Steven Szep
Chase Manhattan Bank
1 New York Plaza
New York, NY 10081

Abstract

Linda is a homemade program which provides aid or the analysis of performance statistics and offers suggestions for enhancing VAX/VMS system performance. This paper offers insights into the design and use of this software tool.

Designing Linda

Tuning

The basic principles of tuning are easy to learn. As with any unfamiliar methodology, people tend to overestimate the difficulties involved in mastering the tools and the techniques. We hope to make the unfamiliar more familiar to you, and to dispell any anxiety you may have.

Tuning is not exactly fun. A great deal of "disciplined thinking" is involved. Jumping at hasty conclusions is always tempting – especially with management bearing down on you. And ignoring latent problems is potentially catastrophic.

A little technical insight is needed. For instance, we now all understand

- How a VAX works
- What the different components of VMS do
- How users are able to interact with VAX/VMS.

Once we come to understand the basic principles involved in resource allocation – because THAT's the name of the game in tuning, we will be able to use more effectively the tools DEC has provided us to monitor and analyze VAX/VMS performance.

A conceptual model will help us to learn quickly, to solve problems, to reduce uncertainty, and to predict behavior. And, learning some jargon is one antidote for a lack of specialized training.

There is no "generic" system against which to compare yours in order to save time during and speed up your tuning sessions. There is NO way to predict system behavior in the face of changes in your hardware configuration or the program mix.

System tuning methodology

1. Problem definition
2. Diagnosis

- (a) Collection of data
- (b) Determination of the limiting resource(s) (Memory, I/O, CPU)
- (c) Isolation of the cause(s) (Memory, I/O, CPU)
- (d) Compensation for the limitation(s)
- (e) Verification of results

A general approach to SPM

1. System summary

- "CPU only" gives % time busy and no drive busy.
- "CPU / I/O" gives % time for overlaps.
- "I/O only" gives % time for CPU idle and ≥ 1 drive busy.

2. Note periods of intense activity.

3. Reduce data to window in on these "hot spots".

- INTERVAL reports give 1 page for each sample interval.
- FINAL report gives final statistics, averaged over the entire (reduction) period.

Introducing LINDA

LINDA is a program which provides aid for the analysis of historical system statistics and offers suggestions for enhancing system performance.

LINDA is written in Pascal. Basically, it is a software version of the decision tree found in Digital's "Guide to VAX/VMS Performance Management". It consists of an interactive question-and-answer program and an extensive Help facility.

LINDA will leverage scarce expertise and resources in the areas of VAX/VMS performance and capacity planning. It will provide rapid, accurate, and enhanced communications within Chase.

The program LINDA permits the TECHNICAL staff to pinpoint the causes of undesirable behavior on your VAX system. It does this by helping him to isolate particular kinds of performance problems, and by outlining what corrective procedures he can undertake, if any.

Of course, this same user must later monitor the effectiveness of any remedial action taken: if there is no sufficient improvement, he must again employ LINDA to analyze the situation all over again. Perhaps the changes made were too radical and should be scaled back to a more conservative level. Or, perhaps some significant problem had been masked by one which has been solved, and now there is the need to correct this one.

We quickly come to realize that tuning is primarily an iterative process, and that multiple causes of performance problems can only be uncovered by repeated use of LINDA — until we achieve a satisfactory level of performance.

The motivational force behind my decision to create the LINDA program was two-fold:

- Increasing dependence on staff to tune systems;
- Increasing frustration from our dependence on DEC's published "Guide to VAX/VMS Performance Management."

Tuning involves executing the following procedures in an orderly manner:

- Collecting behavioral statistics;
- Evaluating the data against some norm;
- Adjusting system parameters, user values, and application design decisions.

Effecting beneficial changes in system performance implies a great amount of detailed effort and disciplined decision-making. Why not use software to automate as much of this process as possible?

We had schedules to meet: more tuning sessions were being planned than we could possibly hope to complete on time. We also had systems ready to go into production without proper attention having been paid to their actual performance: our developers' credibility was on the line.

We needed a new tool to assist us in detection, diagnosis, and correction of VAX/VMS performance problems. My interest in building tools to assist ourselves in building better systems led me to write this program.

Linda¹ looks at memory issues first. Why we do this is because:

- They are the most frequent
- They cause paging and swapping
- They can lead to I/O and CPU problems.

Figures 1 and 2 are a sample session from our archives. It will give you a flavor for what assistance Linda can provide to the technical staff.

¹The Appendix gives the Pascal code for the "core" routines.

Facts known by Linda

1. Causes of a CPU limitation

- Some process(es) may be blocking others
- Lost CPU time
- High system overhead
- Excessive Interrupt Stack activity

2. Symptoms of a CPU limitation

- Processes waiting in CPU queue
- No idle time
- High system CPU time

3. Where SPM reveals a CPU limitation

- AVE Mem/CPU Queues The CPU queue value must be three or greater, because the NULL process is counted, as well as the process displaced by SPM to capture data.
- Scheduler States and COM queue

4. SPM reporting for no idle time (CPU Idle and Total Idle)

5. SPM reporting for system CPU time high. This CPU Busy statistic can fall into the Interrupt Stack, Kernel Mode, and Executive Mode.

6. Compensating for a CPU limitation. This can be done by specifying explicit priorities for processes or jobs, modifying the SYSGEN parameter QUANTUM or upgrading the CPU.

7. Causes of memory limitations. This includes too little physical memory in the system, inappropriate memory management strategy, and improper memory assignments to users.

Limited memory leads to paging or swapping — that is, system I/O. Page faulting means that the CPU waits.

8. Symptoms of memory limitations include no free memory, high page fault rate, and high swapping rate

9. Where SPM reveals memory limitations

- AVE Process-Memory Counts (including free pages)
- Memory utilization (total and user utilization)

10. SPM reporting for high fault rate (page faults and system faults)

11. SPM reporting for high swap rate

- AVE Mem/CPU Queues (especially Memory queues)
- Swapper Counts (especially InSwap and Swapper CPU percentages)

12. Compensating for memory limitations
 - Reduce the number of image activations
 - Adjust page cache sizes
 - Adjust working set sizes
 - Adjust AWSA parameters
 - Adjust memory loan parameters
 - Adjust swapper trimming / swapping
 - Modify balance set count
 - Prevent active processes from devouring memory
 - Enable swapping for processes
 - Acquire more memory.
13. Causes of I/O limitations
 - Number of device(s) is insufficient
 - Speed of device(s) is inadequate
 - Excessive demand on particular device(s)
 - Insufficient blocking factors
 - Inadequate number of buffers
14. Symptoms of I/O limitations
 - High direct I/O rate
 - High buffered I/O rate
15. SPM reveals I/O limitations in I/O rates (especially Direct and Buffered I/Os)
16. What to search for in SPM reports
 - Determining which I/O device
 - Paging or swapping (system) I/O
 - Poor file system caching
 - Fragmented disk
 - Explicit direct QIO's
 - Explicit buffered QIO's
17. Determining which device: Check all statistics under "Device I/O Rates" and the Busy, Paging, Swapping, and Controller rates under "Disk Statistics."
18. Checking system I/O: Check "Paging Rates" (especially Read and Write I/Os) and "Disk Statistics" (especially Paging and Swapping Rates)
19. Check RMS caching under "File Cache Attempt Rate," and "File Cache Effectiveness ."
20. Detect a fragmented disk by examining "File I/O Rates" (especially Window Turns and Split I/Os).
We use `$ PERFORM REPORT=DISK_SPACE` to analyze online disk space.
21. Detect direct I/O by examining "Process Metrics."
22. Detect buffered I/O by examining "Device I/O Rates" and "Process Metrics."
23. Compensate for I/O limitations by Re-distributing paging and swapping I/O, improving file system cacheing, reducing disk fragmentation, and handling explicit QIOs.

Source code

This appendix contains the Pascal source code for the "core" routines used in the LINDA program.

Note: Explanations, advice, and recommendations given by Linda are contained in the Help library.

```

CONST
  lbr$c_read = %X'01';

(* External RTL routines *)

FUNCTION LBR$INI_CONTROL
  (var library_index: integer;
   func: integer;
   libe_type: integer := %immed 0)
  : integer; extern;

FUNCTION LBR$OPEN
  (library_index: integer;
   fns: [class_s] packed array[i..u: integer]
     of char := %immed 0;
   create_options: int_array := %immed 0;
   dns: [class_s] packed array[i2..u2: integer]
     of char := %immed 0;
   rlfna: array[i3..u3: integer] of integer
     := %immed 0;
   rns: [class_s] packed array[i4..u4: integer]
     of char := %immed 0;
   var rnslen: integer := %immed 0)
  : integer; extern;

FUNCTION LBR$GET_HELP
  (library_index: integer;
   line_width: integer := %immed 0;
   %immed [unbound] procedure routine
     := %immed 0;
   data: integer := %immed 0;
   key_1: [class_s] packed array[i..u: integer]
     of char)
  : integer; extern;

FUNCTION LBR$CLOSE
  (library_index: integer)
  : integer; extern;

(***** MAIN PROCESSING LOOP *****)

  Start_up;

(* Start the questions now. *)

  path := 0;    { Set indicator for stream identifier. }
  blocked := 0; { Set indicator for program termination. }
  response_index := 0; { Set index into response array. }

  REPEAT

    Dec_tree;

  UNTIL blocked <> 0;

```

```

help_stat := LBR$CLOSE(lib_index);
IF NOT ODD(help_stat)
    THEN
        LIB$STOP(help_stat);

(* Save session responses in a text file for future reference *)

(* Initialize the librarian. *)

help_stat := LBR$INI_CONTROL(lib_index,lbr$c_read);
IF NOT ODD(help_stat)
    THEN
        LIB$STOP(help_stat);

(* Open the correct library. *)

help_stat := LBR$OPEN(library_index := lib_index,
                      fns := 'SYS$HELP:TUNE.HLB');
IF NOT ODD(help_stat)
    THEN
        LIB$STOP(help_stat);

///

PROCEDURE Dec_tree;

BEGIN

    path := 0;    { Set indicator for stream identifier. }

    CASE segment OF

        1 : Evaluate_complaint;
        2 : Start_investigation;
        3 : Investigate_memory;
        5 : Analyze_swapping_data;
        6 : Examine_voluntary_decrementing;
        7 : Investigate_swapper_trimming;
        8 : Investigate_swapping_behavior;
        9 : Large_waiting_processes;
       10 : Computable_processes;
       11 : Investigate_scarce_free_memory;
       12 : Investigate_io;
       13 : Investigate_file_activity;
       14 : Investigate_terminal_io;
       15 : Too_many_characters;
       16 : Investigate_cpu;
       17 : Investigate_idle_time;

        OTHERWISE Report_problem;

    END;

END; { End of procedure Dec_tree }

```

```

REPEAT
  Question(reply, string);
  CASE reply OF
    'Y' :      BEGIN
                path := 1;
            END;
    'N' :      BEGIN
                path := 2;
            END;
    'H' :      BEGIN
                path := 0;
                key := 'TUNE1';
                Display_help(key);
            END;
    OTHERWISE Bad_response;
            path := 0;
  END;
UNTIL path <> 0;
IF path = 1
  THEN
    BEGIN
      path := 0;
      segment := 2;
      WRITELN('Initiate preliminary investigation.');

```

An Evaluation of Record I/O Versus Block I/O From a Programmer's Viewpoint

Darylene Colbert
SAS Institute Inc.
Cary, North Carolina

Abstract

This paper describes experiences the author had writing an I/O subsystem for a major software project under VAX/VMS. The use of Block and Record I/O is compared.

Introduction

As a member of the VMS Host Development Group at SAS Institute, I have been involved in the redesign of the Version 5.16 I/O subsystem for the SAS System. I have also worked on the design of the future Version 6 I/O subsystem. Within the SAS System, a large portion of the code is written to be portable across many operating systems. To support this portable code, there is a layer of code at the bottom of the hierarchy which is machine-dependent. It is at this layer that each host development group develops an I/O subsystem to take advantage of individual operating system features. We were concerned about the type of I/O which should be performed and its efficiency and functionality. We specifically needed to know whether to perform block I/O or record I/O at the host level. Therefore, research and testing was undertaken to help us make our decision.

Within the SAS System, the SAS I/O subsystem is responsible for all I/O to SAS data sets. We started with this list of characteristics of SAS data sets on disk:

- They have a sequential file organization, allowing both sequential and random access.
- Most I/O performed is sequential.
- SAS data sets tend to be rather large data files.
- SAS data sets have a fixed record length of 512 bytes.

Before we began, we theorized that block I/O would be more efficient for our type of processing due to sequential access and relatively large fixed record length. However, we also knew that record I/O provided many more parameters that could be used to tune I/O performance.

After the research and analysis was done, we chose to implement our design using block I/O to optimize performance. Many of the features available through RMS were not needed in our application. During the process, we learned of several methods to tune block and record I/O. In addition to presenting the results of our testing, I will go over some of those tuning methods.

This paper presents a comparison of performance between block I/O and record I/O. The first section explains the basic differences between them and the research from which our conclusions were derived. The discussion describes the assumptions of the test programs, the results of the tests, and an analysis of the results. The third section defines and describes a few of the parameters available to optimize performance of both block I/O and record I/O. The fourth section summarizes the results and conclusions of the tests that were performed. The intended audience for this discussion is the programmer who is experienced in general but is not familiar with writing I/O-intensive applications on the VAX.

Comparison of Block I/O and Record I/O

Definition of Terms

What is the difference between block I/O and record I/O? What happens when you make an I/O request? Most programmers are familiar with record I/O. Under VMS, record I/O operations are provided by the record management services, RMS. Record processing under RMS appears to your program as the movement of records directly between a file on disk and your program. This is, in fact, not the case. RMS uses internal memory areas called I/O buffers to read or write blocks of data. Transparently to your program, RMS transfers blocks of a file into or from an I/O buffer. Records within the I/O buffer are then made available to your program when RMS transfers the records between the I/O buffer and it.

Block I/O bypasses the RMS record-processing capabilities entirely. Your program assumes complete responsibility for dividing blocks into records and moving the records to program storage and vice versa. Rather than perform record operations by means of one of the supported record access modes, you process the file as a logical structure consisting of some number of blocks. In block I/O, a program reads or writes one or more blocks by specifying a starting virtual block number in the file and the length of the transfer. Block I/O operations provide you with an intermediate choice between

RMS operations and direct use of the Queue I/O Request system service.

Test Environment

In the test jobs used to compare the two types of I/O, the only variables changed were the type of I/O used and the size of the internal buffers. All other parameters remained constant. The test images are written in C and are equivalent in functionality. The images read and write a data file of fixed length 512 byte records, 25000 blocks long. For medium and large buffer sizes, the results for reading the file were almost identical to those for writing a file. Therefore, this paper refers to the statistics obtained from each interchangeably. The jobs were run on a VAX 8600 with a moderate system load. The test images use a default extension quantity of 160 blocks and a retrieval window size of 255. These two parameters are discussed in more detail in section 4. The only optimization made in the record I/O image was to allow RMS to use the read-ahead (RAH) and write-behind (WBH) options. To allow this, the RAH and WBH flags were specified and the number of buffers used was set to 2.

Each method was tested using various buffer sizes. For block I/O, this corresponds to the program's internal buffer. For record I/O, it corresponds to the buffers used by RMS. The buffer sizes used, in bytes, were 512, 1024, 2048, 4096, 8192, and 16384. A buffer size greater than 16K is atypical for our application. Each image was run multiple times for each buffer size. The resource usage statistics for each buffer size were then averaged to draw our conclusions and produce graphs of each usage statistic by buffer size for block I/O and record I/O.

Test Results

For our research, we chose to look at five statistics for comparison; CPU time, elapsed time, page faults, direct I/O count, and buffered I/O count. These statistics were gathered only for the actual read/write operations and do not include image activation or the open/close operations.

The CPU time used is the total amount of time charged to your process for the CPU. As you can see from Figure 1, the CPU time used is consistently less for block I/O. The CPU time decreases as the buffer sizes get larger.

Elapsed time in these test jobs is the length of time that passes from the beginning of the I/O operations until completion. It includes both CPU time and I/O completion time. Although elapsed time is a more subjective statistic because it varies greatly with system load, it is the one that is most visible to a user. It is the elapsed time that determines how long the user waits while data is being read or written. In Figure 2, you see the elapsed time for block I/O is less than that for record I/O. As the buffer size is increased and fewer disk accesses are required, the elapsed time for both methods decreases.

Page faulting is a function of your working set size; the smaller your working set size, the more paging VMS must do to perform the same amount of work. Figure 3 shows a graph

of the number of page faults incurred during the I/O operations. These jobs were run with a working set quota of 1000 pages, which was not a limiting factor. With that working set size, you see very little paging occurring during the I/O operations. However, if the working set size had been smaller than the physical memory needed for program execution, faulting would have increased. This is true for both block I/O and record I/O.

The direct I/O count reflects the number of times an I/O operation is performed to or from a physical device, in this case, the disk. Notice from Figure 4 that, as the size of the internal buffer increases in both block and record I/O, the direct I/O buffer count decreases. This is because the buffer is filled completely before an access must be made to the disk. So the larger the buffer, the fewer accesses required to write the same amount of data. The interesting aspect of this graph is that the direct I/O count is the same for both block and record I/O for each buffer size. This is expected since the number of accesses required for a specific amount of data is a function of the buffer size, not of the I/O method used.

The last statistic compared is the buffered I/O count. The buffered I/O count reflects buffering done for you automatically by VMS when some form of intermediate buffering is required. For example, requesting I/O on a slow device such as a terminal requires buffered I/O. VMS performs this intermediate buffering in system space and the buffered I/O count is incurred only for this buffering in system space. The program's internal buffer for block I/O, and the RMS internal buffer for record I/O, both reside in program space. Therefore, nothing in our comparisons affects the buffered I/O count. This is seen by the constant line for both record and block I/O in Figure 5.

Analysis of Data

After gathering the data and evaluating each usage statistic, we analyzed the data to see how each result pertained to our application. There are primarily four areas to consider - execution speed, memory usage, file sharing, and development time.

Execution Speed

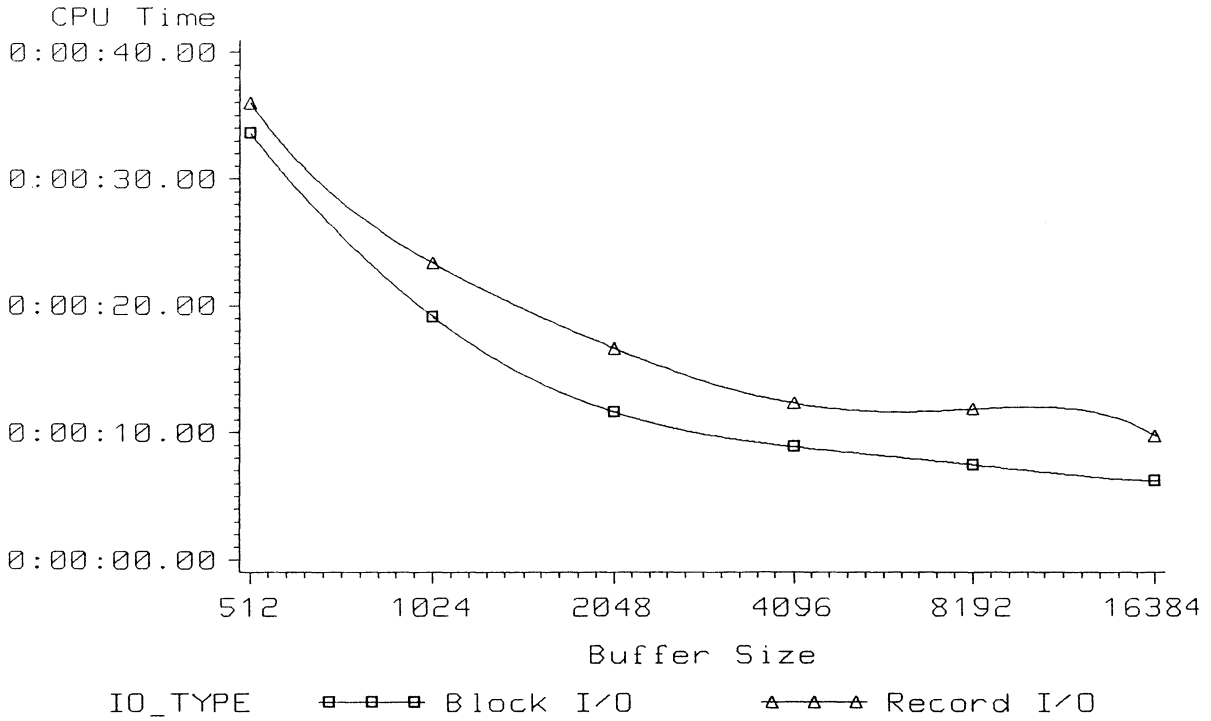
The two statistics which most directly reflect execution speed are CPU time and direct I/O count. For CPU time, block I/O is more efficient. For direct I/O count, there is no difference between block and record I/O. Since increasing execution speed was a primary goal for the I/O subsystem, we chose to implement the system using block I/O.

Memory Usage

The data shows that the larger the buffers, the faster the task processes data. Memory being a limited resource, you must determine a satisfactory balance between execution speed and memory usage. Not only must you have the memory required for the internal buffer, but you must also consider the amount of memory necessary for a reasonable working set size - one

CPU Time (in seconds)

Block I/O and Record I/O

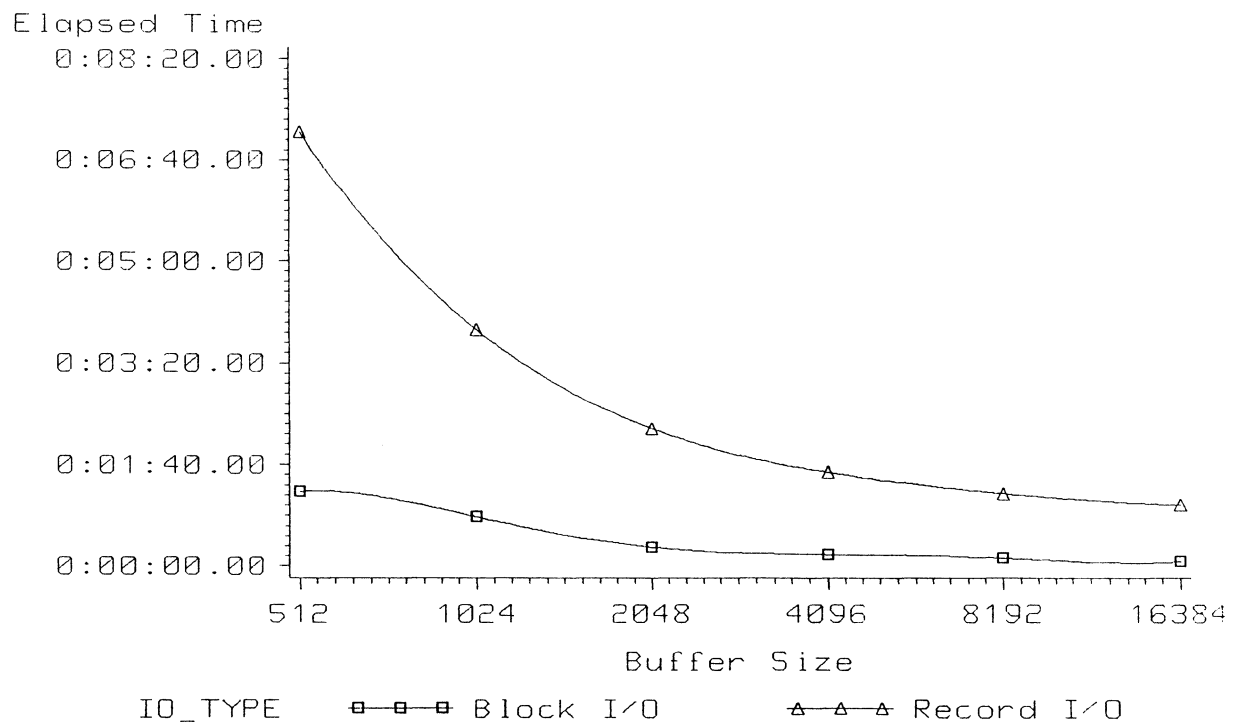


Graph Produced Using SAS/GRAPH

Figure 1: CPU Time

Elapsed Time

Block I/O and Record I/O

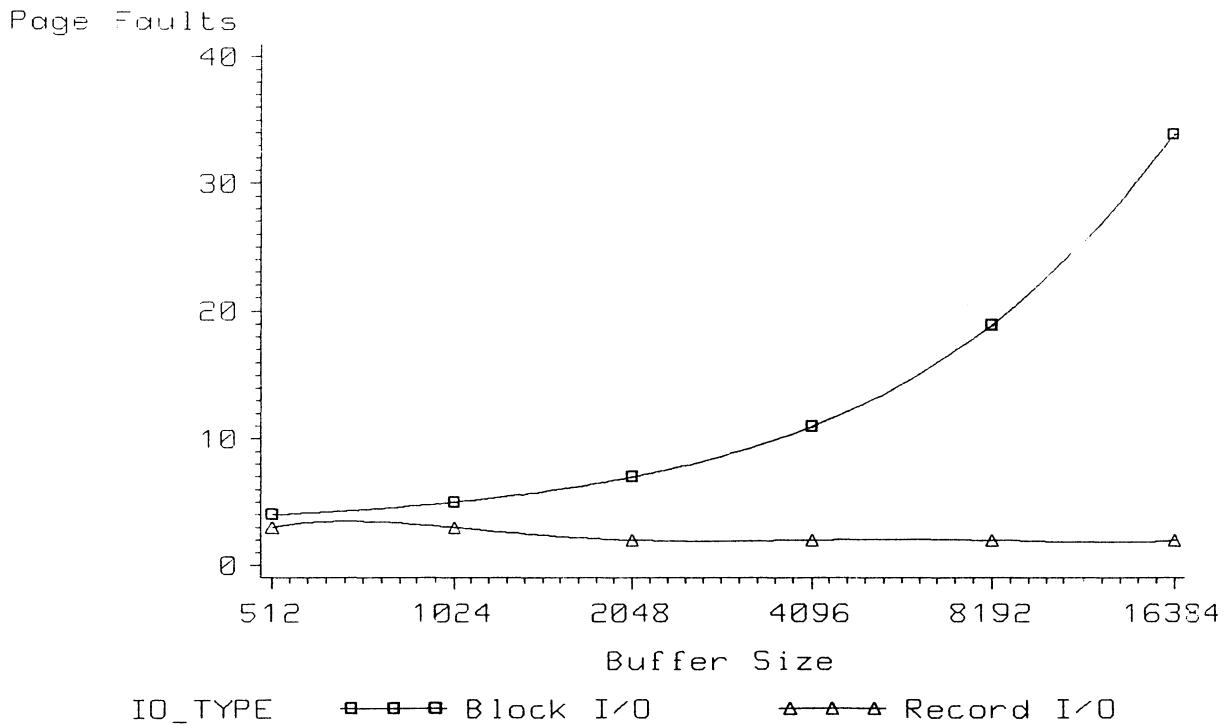


Graph Produced Using SAS/GRAPH

Figure 2: Elapsed Time

Page Faults

Block I/O and Record I/O

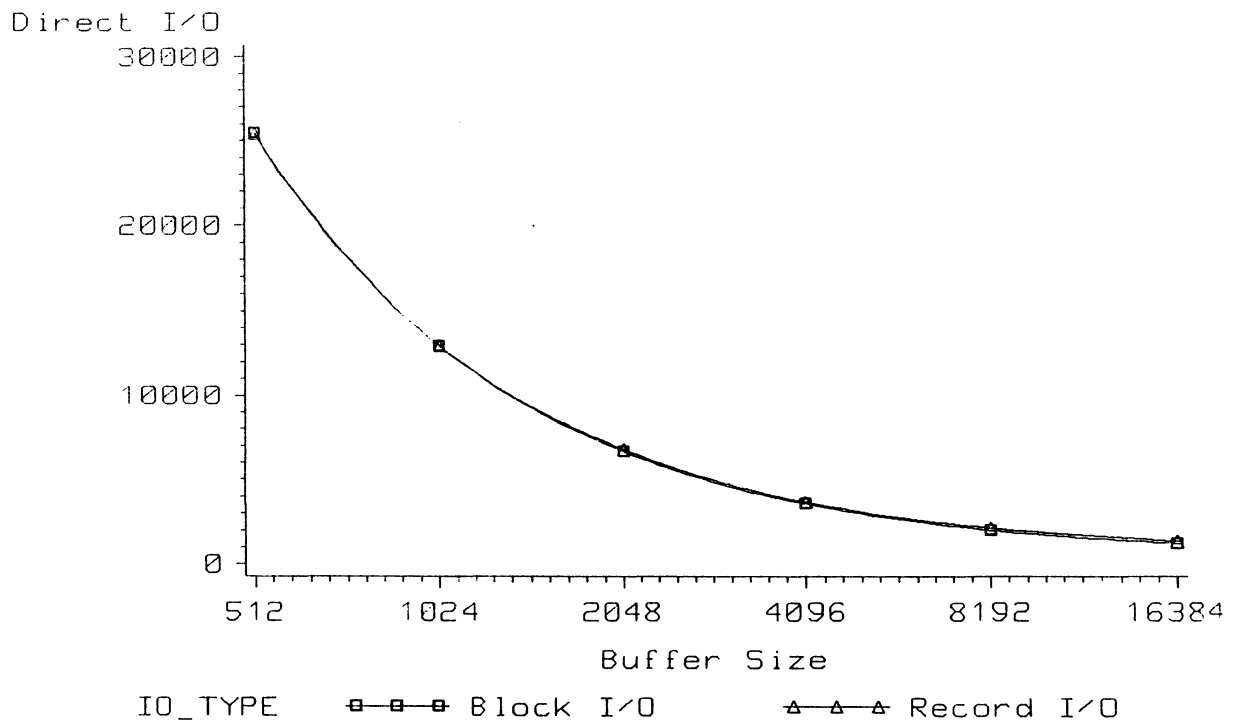


Graph Produced Using SAS/GRAPH

Figure 3: Page Faults

Direct I/O Count

Block I/O and Record I/O

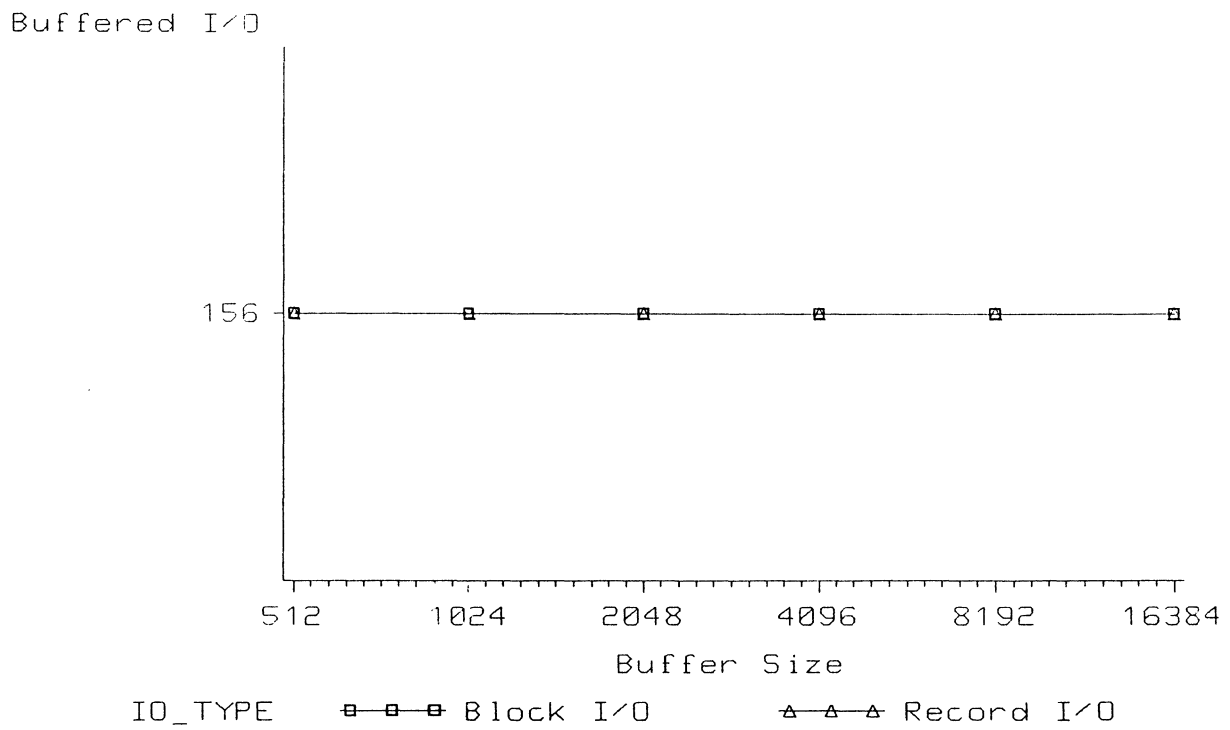


Graph Produced Using SAS/GRAPH

Figure 4: Direct I/O Count

Buffered I/O Count

Block I/O and Record I/O



Graph Produced Using SAS/GRAPH

Figure 5: Buffered I/O Count

that won't create an excessive amount of paging, given the chosen buffer size. From the CPU time graph, we find that execution speed doesn't increase much once your buffer is larger than 4096 bytes. With that buffer size:

- you stay within a reasonable page faulting range,
- the direct I/O count decreases, and
- elapsed time decreases.

For these reasons, we chose an internal buffer size of 4096 bytes to use with our block I/O design.

File Sharing

Typically, many users may want to access a file at the same time. This is called shared access. For our application, this is not relevant since the SAS System does not currently allow shared access to a data file under VMS. However, if you are considering shared access of records within a file, you must use record I/O to take advantage of the record level locking and unlocking features of RMS.

Development Time

For the simple sequential file structure that we must implement, the difference in development time using record or block I/O is not substantial. Therefore, development time was not an important factor in our decision to use block I/O. However, if your application requires the use of indexed files or variable length records, record I/O is the more practical choice. By using record I/O, RMS provides all the logic necessary to use complex record access modes, for example, indexed mode and relative mode. RMS with record I/O also provides complex record formats, like variable length records.

One must also consider future development plans. Once you commit to a method, whether block or record I/O, you cannot easily access and modify the file using the other method. RMS files contain internal information meaningful to RMS itself, so that if you modify an existing file using block I/O, none of the RMS internal structures will be updated. Likewise, if the file is created using block I/O, none of the RMS internal structures will be created and attempting record access through RMS will fail.

Summary of Analysis

To summarize, after consideration of each aspect described above, we chose to implement the next major release of the SAS I/O subsystem using block I/O. The RMS features and tuning parameters provided when you use record I/O are not necessary for our particular application. Therefore, optimizing performance became the deciding factor and block I/O was chosen.

Along with the block I/O format, an internal buffer size of 4096 bytes was chosen as the buffer size to use. This buffer size provides the best tradeoff between memory used for buffering and performance.

During the testing that was done between block I/O and record I/O, we found that the default values for parameters used by RMS did not always reflect the I/O that our application performs. For this reason, you should look into adjusting the parameters that can be set in your program to tune the performance of the I/O operations. For applications that use record I/O, there are many such parameters. However, for block I/O applications like ours, there are only a few parameters that significantly affect performance. This section discusses some of the parameters that can be tuned to improve performance. I will describe the parameters that work for both block and record I/O first, then the parameters that are specific to record I/O.

Specifying Tuning Parameters

Tuning parameters can be adjusted by changing fields in the File Access Block, or **FAB**, and the Record Access Block, or **RAB**. These are predefined structures which RMS uses to give or receive information about the structure of a file and the structure of the records within the file. Alternatively, many parameters can be set with the `DCL SET RMS_DEFAULT` command.

The **FAB** allows you to communicate file-related information to file service calls, such as file characteristics, file specifications, and run-time options. Each of the fields within the **FAB** is assigned a predefined symbolic offset name. Each field name is prefixed with **FAB\$**, followed by a character that specifies the length of the field, followed by an underscore. For example, to refer to the allocation quantity for a file, the symbol **FAB\$L_ALQ** is used. This indicates that we are referring to the **ALQ** field within the **FAB** and that it is a longword-length field.

The **RAB** allows you to communicate record-related information to RMS record services, such as the location, type, and size of the input and output buffers, the record access mode, and tuning options. Just like the **FAB**, the **RAB** uses predefined symbolic offset names along with a standard prefix. For example, to specify the multibuffer count field within the **RAB**, you use **RAB\$B_MBF**. In this example, the name indicates that the **MBF** field is a field in the **RAB** and it is a byte-length field.

Parameters Common to Block and Record I/O

Internal Buffer Size

The parameter that makes the single biggest difference in performance is the buffer size. As the buffer size increases, performance improves. For block I/O, the buffer size is specified in the programmer's code, often as an array of the desired size and type. For record I/O, the buffer size is specified in units of blocks in the **RAB\$B_MBC** field of the **RAB** or with the `DCL` command `$ SET RMS_DEFAULT/BLOCK_SIZE=n`. The optimum size to use depends on the intended use of the file; the only way to determine the best size is to experiment. The default is determined by the `SYSGEN` parameter, **RMS_DMBC**.

Remember that the cost of a larger buffer size is a larger memory usage. As stated previously, we found that an internal buffer size of 8 blocks was optimum for our application.

Default Extension Quantity

There are two parameters that affect performance when adding records to a file and therefore increasing the file size. The first is default extension quantity, which specifies how many blocks to allocate on disk for adding records to a file when the file gets larger than the space initially allocated to it. If you know you will be adding records to a file, you should specify a reasonable default extension quantity to reduce the number of times that the file will be extended. If you use an extension quantity that is too small, you will incur many extensions, causing your file to be fragmented over the disk and resulting in slower access time. However, if your extension quantity is too large, you will be reserving large areas of disk space that may not be used and therefore, wasted. If you do not specify a default extension quantity, RMS will compute a size to use; however, this size may not be optimum. You can approximate a reasonable quantity if you can estimate the average number of records that will be added to the file. The default extension quantity is specified using the `FAB$W_DEQ` field in the `FAB` or with the DCL command `$ SET RMS_DEFAULT/EXTEND_QUANTITY=n`, where `n` is the number of blocks per extension.

Retrieval Window Size

The second parameter that is related to extending a file is the retrieval window size. Each extension to a file is called an extent. If a file is extended repeatedly, the extensions will likely be scattered on the disk. A pointer to each extent, called a retrieval pointer, resides in the file header. For improved performance when reading records from the file, some number of these retrieval pointers are kept in memory in a structure called a window. The more retrieval pointers you keep in memory in the window, the faster your record access will be. The cost of the retrieval pointers is charged to your buffered I/O byte count quota. Since the number of retrieval pointers needed is directly related to the number of existing extents, the default extension quantity and the retrieval window size should be considered together.

The default window size is 7 pointers. Valid values are in the range of 0 through 127, or 255, which specifies that all retrieval pointers be kept in memory, thus mapping the entire file, if possible. In our test jobs, a retrieval window size of 255 was used. By specifying a value of 255, the number of retrieval pointers kept in the window will increase dynamically as the number of extents increases. The performance gain is not seen when writing the file, but rather when the file is read back in. The retrieval window size is specified in the `FAB` in the `FAB$B_RTV` field.

Contiguous Best Try

One way to decrease the number of extents, and therefore retrieval pointers required, is to request the contiguous best try

option. Using this option causes RMS to attempt to allocate the file using a minimum number of extents. It will make the entire file contiguous, if possible. However, if the file cannot be allocated contiguously, it does its best to allocate the largest contiguous areas possible. The only disadvantage of using this option is a slight performance cost at file open time. But the record access improvement definitely outweighs the cost. The contiguous best try option is set in the `FAB` in the `FAB$L_FOP` field by using the symbol `FAB$V_CBT` to turn the option on.

Parameters Specific to Record I/O

Number of Buffers Used

Not only can you specify the size of the internal buffer that RMS is to use when using record I/O, but you can also specify how many buffers of that size you want RMS to use. The RMS buffers are allocated from the process working set. For sequential files, you must specify at least 2 buffers if you use the read-ahead and write-behind options. If you do not intend to use read-ahead and write-behind, one buffer is normally sufficient for sequential files. Increasing the number of buffers takes space in the process working set and can actually degrade performance by causing excessive page faulting if the number is too large. The default number of buffers is one. To specify more than one buffer, use the `RAB$B_MBF` field or the DCL command `$ SET RMS_DEFAULT/BUFFER_COUNT=n`, where `n` is the number of buffers.

Read-ahead and Write-behind Options

With the read-ahead and write-behind options in effect, RMS alternates buffer use between two buffers. With this type of processing, one buffer contains the next records to be read or written to the disk while the second buffer completes I/O. With read-ahead, records are read into a buffer before they are actually needed, so you do not have to wait for I/O to complete when you are ready to process another record. With write-behind, when the first buffer is filled, the next record processed goes to the second buffer. The I/O operation for the first buffer then takes place. The system does not have to wait while the I/O operation completes. Instead, program processing continues.

These options are available only with sequential files and as stated above, you must specify two buffers to use them. The only cost of turning these options on is the cost associated with using two buffers instead of one. With most languages, read-ahead and write-behind are the default operations; with others, you must specify these options explicitly by using a clause in the language. To specify them in the `RAB`, use the values `RAB$V_RAH` and `RAB$V_WBH` in the `RAB$L_ROP` field.

Summary

In conclusion, our decision to use block I/O over record I/O was based on two major considerations:

- the overall performance of block I/O is better than that of record I/O, and
- at this time, we do not need to take advantage of features provided by RMS record I/O.

As our I/O requirements and features change in the future, we may look to RMS to provide the advanced capabilities that we will need.

ODS-2 DISK OPTIMIZATION

Wef Fleischman
Software Techniques, Inc.
DECUS Nashville, Spring 1987

ABSTRACT

Disk structure optimization is a proper concern for all system managers and, if treated systematically, is also one of the most interesting and rewarding pursuits. This article addresses how to approach disk performance optimization for VMS systems. In the process, a cost analysis of a hypothetical 8650 system that is overloaded notes the hidden costs associated with a sub-optimal system. A brief review of basic disk operation, including its mechanical nature and VMS usage, shows the bottlenecks that rob your system of performance. A survey of VMS commands reveals the source of problems and the utilities to fix them, and case histories demonstrate how optimization methods yield beneficial results.

INTRODUCTION

It seems that no matter how many advances are made in computer technology one thing always surfaces as a top wish-list item: *make the system run faster.*

It is human nature to become accustomed to the processing capacity available at our keyboards. We are initially impressed when our new VAX is installed, but powerful resources draw a workload like an electromagnet in a scrap metal yard. Later, when the machine is periodically overloaded it disappoints us and we nostalgically yearn for the better performance that we knew was possible (at least at one time).

Improving the situation is, in many cases, a financial question. Adding hardware is relatively expensive but can improve your system's capacity and, therefore, its performance. However, a better question to ask is, "Are you getting the most from the hardware you already have?"

The remainder of this article focuses on how to get the most from your VAX/VMS system by making sure that your disk resources are performing at peak efficiency. By comparing the cost of buying new hardware with the cost of tuning you can make an appropriate choice for your own installation.

THE COST OF DISK STORAGE

All computers are information processors. We pay a precious premium for having the information nearby: the *accessibility* of the information is what costs dollars. If the data doesn't reach us as fast as it could, we have been short-changed. And when we store data that we don't really use on the disk, it costs "hidden dollars" by depriving us of quick access of the data that we really do use. Figure 1 contrasts the relative price we pay for information stored on various media (notice that magnetic tape is very economical per megabyte, but suffers from relatively slow access time as shown by the dotted line).

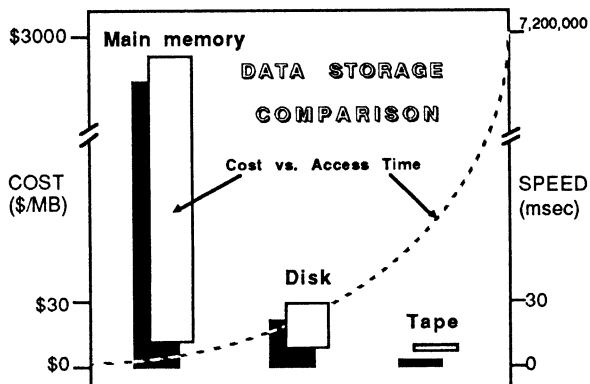


Figure 1

Software Techniques, Inc.

PERFORMANCE TUNING AN OBJECTIVE APPROACH

Performance tuning is best undertaken systematically and generally involves the following steps as listed below and summarized in Figure 2:

1. Identify a *single* problem *objectively*: i.e., "response time is too great in the inventory program".
2. Hypothesize its cause: "disk activity is bottlenecked at DUA2".
3. Identify measurement tools and define a benchmark: "supervisor will measure elapsed time during lookup of five standard inventory items" and "DUA2: direct I/O rate be measured with the MONITOR utility, recording to a file".
4. Perform the baseline benchmark from which comparisons can be made.
5. Experiment with system software modifications: i.e., "DUA2: will be shadowed with DUA3".
6. Perform the post-benchmark and compare results.
7. Repeat until you're satisfied.

OBJECTIVE PERFORMANCE TUNING

1. Identify problem
2. Hypothesize cause
3. Design benchmark
4. Collect baseline data
5. Experiment
6. Collect results
7. Evaluate
8. Repeat, as necessary

Figure 2

DECUS / Nashville Spring 1987

ODS-2 Disk Optimization

It is important to undergo the baseline benchmark. It lends credence and authority to your conclusions by yielding concrete numbers for comparison. The benchmark also quantifies benefits so you can estimate the payback achieved through tuning, and allows you to choose among several different experimental changes. (When making changes, you will almost always elect to try several options because it is as easy to make several changes as it is to make one.)

Key to the process of tuning is the ability to spot problems and know what to do about them. To accomplish this, you should have:

1. At least a cursory understanding of the "mechanical" nature of disk hardware.
2. An understanding of disk usage under VMS (i.e., how ODS-2 works).
3. Practice in "investigative monitoring", and...
4. Knowledge of the tools that let you to *tweak* the system.

Later, we'll talk about how to recognize the bottlenecks and how to deal with them, but let's review some details of disk operation under VMS first.

THE LIMITING FACTORS

It is important to understand that the *mechanical motion* which is intrinsically part of all disk drives¹ also limits their speed. Moving the head armature to the correct cylinder is the largest component of time required, while waiting for the disk to rotate is the second (and lesser) component of *latency*. In Figure 3, the black portion of each bar represents the average time to position the disk armature to the appropriate cylinder. The white portion details the time taken for the desired disk sector to rotate around and become accessible by the read/write heads.

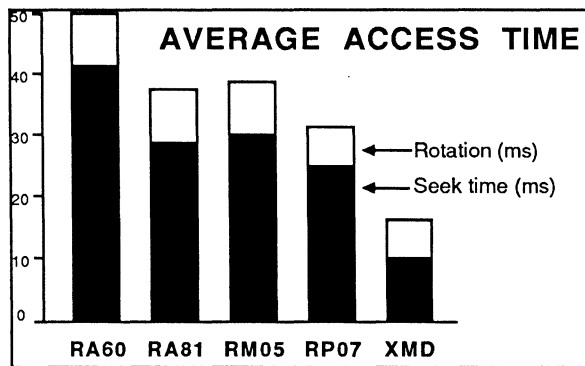


Figure 3

Disks such as the CDC 9772 XMD, shown at the far right, exhibit fast seek times but do relatively little to improve rotational latency. This is because most disk drives rotate at about 3600 r.p.m. (You can expect to see a marked improvement in this statistic over the next five years, however, until improvements are made you must still live with the current limitations.)

Figure 3 clearly illustrates why we need to do everything in our power to minimize the seek distance required by disks. It is the seek distance that is the largest component of access time: if the average seek distance is halved, a busy drive can handle roughly twice the number of user requests.

The disk heads can only be used by one user at a time. On a heavily accessed disk, the heads immediately depart for another area of the disk when finished with the current user's request. If we want to read two sectors, it is optimal to do so in one operation, rather than reading one sector now and coming back later for another sector. Such

¹ Solid state "memory disks", which are constructed of bulk semiconductor memory, are an exception. These "pseudo" disks have access performance similar to main memory.

superfluous seeks consume disk throughput and make them *seem* slow.

ERROR CORRECTING DISKS

Most disks perform some type of automatic error correction these days, including ECC correction, automatic bad block replacement, and other such "transparent" features. Automatic error correction means that the disk, its controller, and/or VMS go out of their way to try to recover from disk failures without the user's (or system manager's) immediate knowledge. This is a great feature, but it has its costs. Error correction takes time and can, in some instances, cause a mysterious loss of performance. For example, have you ever had the experience of not noticing that a disk was having extensive errors until the ERRLOG.SYS file became suspiciously huge? Or not noticed disk errors until a user pointed out a five digit number in the "errors" column of a \$ SHOW DEVICE display? Any marginal disk requiring constant error recovery can exhibit hidden yet noticeable throughput degradation.

Disk error recovery also takes at least twice as long (typically ten times as long) as an error-free data transfer. This is illustrated in Figure 4. Always check the error log of disks that *do not* achieve what you expect as normal throughput.

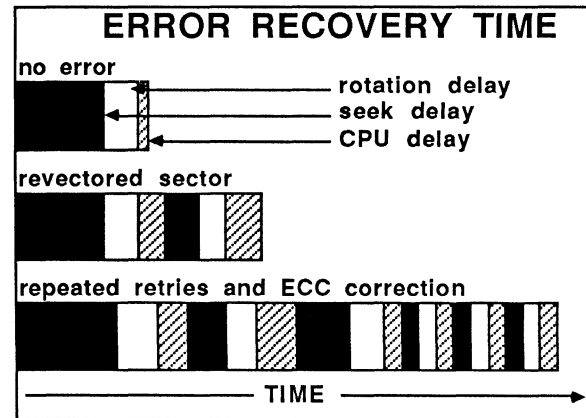
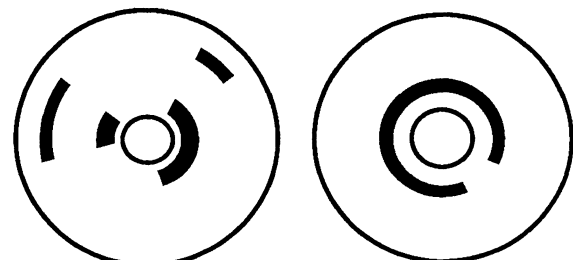


Figure 4

FILE FRAGMENTATION

Two performance problems occur when VMS is dealing with fragmented files. First, the disk heads have to seek all over the disk to retrieve each of the file's fragments, which can number in the hundreds for large files. Second, VMS must step in to assist in locating all those fragments causing a type of overhead called **window turning**. These problems can be eliminated by making fragmented files contiguous as shown in Figure 5.

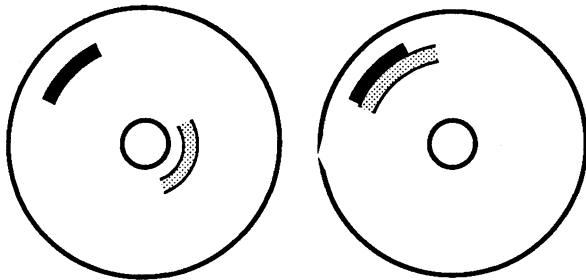


A Fragmented File Made Contiguous

Figure 5

ODS-2 Disk Optimization

The layout of interrelated files is also important. Two files widely separated on the disk and used simultaneously cause the disk to seek alternatively from one end of the disk to the other, a *mechanical motion* that requires *time*. Placing these files together, as shown in Figure 6, allows better access performance.



Two Concurrently Accessed Files Placed Together

Figure 6

ODS-2 FILE STRUCTURE

How does the disk know where to find the contents of file "A" or file "B"? In reality the disk does not. The disk has no knowledge of what the "SYS0" directory means, nor that the pagefile is used for memory management-- the disk is simply a random access block store. Instead, VMS adds the interpretation to particular disk blocks and calls them "boot blocks," "directories" and "files," etc.

VMS is responsible for organizing the system and user files on a disk volume and remembering the location of each. To understand this better, compare VMS file organization to that of books in a library, as illustrated in Figure 7.

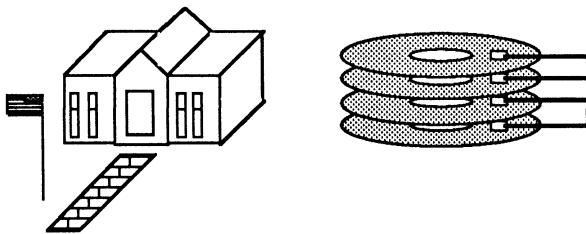


Figure 7

Books in a library are like files on a disk volume. Books are composed of a series of equal-sized pages, and files consist of a number of disk sectors. The librarian (which is F11BXQP² for VMS) dictates the rules for library organization, and is responsible for locating the stored information when it is requested. If a book is too big to fit on a shelf, it can be split to fit on multiple shelves. When a new volume or book arrives, the librarian rearranges the other books to accommodate it.

VMS does not shuffle files around, however, the way a librarian rearranges books. When a file needs to expand or when a new file must be stored, unless given better direction, VMS finds any available space big enough for the file (or its extension). Large files must sometimes be broken into multiple fragments to fit into available contiguous space. It's similar to what might happen if a set of volumes of an encyclopedia had to spread out all over a big library in different rooms.

² This name is derived from "Files-11 (F11) ODS-2 (B) Extended QIO Processor (XQP).

Just as a library has a card catalog that shows the correct bookcase and shelf, VMS keeps its own catalog called the index file as shown in Figure 8.

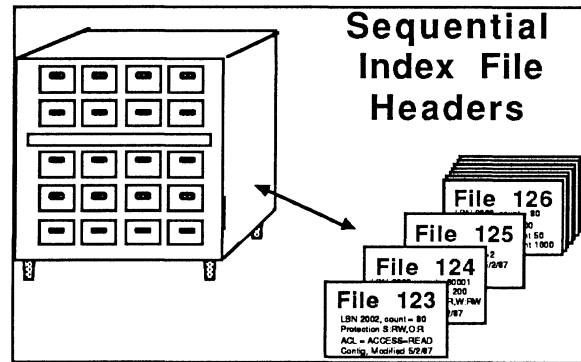


Figure 8

THE FILE HEADER DEFINED

The cards in the card catalog describe where books are located in the library. For VMS, file headers stored in the index file do the same thing.

When you initialize a disk, the "card catalog" is created with a certain number of blank cards. The number of allocated, but unused, headers is controlled by the /HEADERS qualifier to \$ INITIALIZE. In addition, as shown in Figure 9, the index file can expand later and hold even more headers (as controlled by the /MAXIMUM_FILES qualifier to \$ INITIALIZE). When this happens, the index file may become fragmented. This is *undesirable*. It is similar to a library's card catalog becoming so large that another card file has to be placed in another room of the library to hold the overflow. This is as much trouble for VMS as it is for library users, although VMS has been programmed to silently cope with this inconvenience.

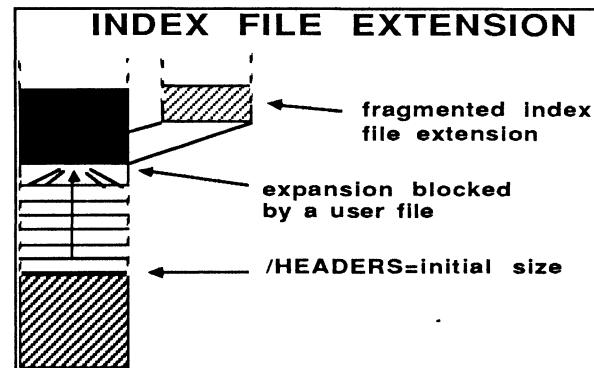


Figure 9

DIRECTORY FILES DEFINED

In VMS, we are accustomed to files being displayed in alphabetical order, and can hierarchically organize files in subdirectories. However, the VMS "catalog card" system stores each new file sequentially on the first available "card". To take all the unorganized "cards" and display them in alphabetical order and in subdirectories, VMS keeps intermediate *directory* files that direct the file name stored to the correct card number in the card catalog. This directory file name is also known as the *file ID* or *FID*.

ODS-2 Disk Optimization

THINGS THAT YOU CAN DO

Now let's center in on the real problem. We want to use all this knowledge about how disks work under VMS to improve performance. Where do you start when you suspect a performance problem and want to do something about it?

One solution, which will be well-received by your DEC salesman, is to add more hardware to enlarge your system's total available capacity-- add more memory. Another option is to off-load some of your current workload to another system, if another system is available and is not over-loaded. Probably, however, the least costly solution to you is to *fine tune* the workload and the hardware you already have to work in better *harmony*.

A COST ANALYSIS

It is difficult to justify expenditures for additional hardware. It can also be difficult to justify time to tune a system if you're already busy and don't know what effort or special training is required. Knowing what you can gain from tuning your system-- and some familiarity with the tuning process-- is what justifies the investment of your time. However, it is the cost gains in particular that convince upper management of the value of time spent on system tuning. To illustrate this, let's look at a hypothetical company.

As at many typical sites, the hypothetical shop shows dramatic rises in computer usage at eight or nine o'clock in the morning. Usage falls briefly around coffee breaks and lunchtime, then falls to a low level after 5:00 P.M. This demand is also moderated by the processing capacity of the computer: as the machine approaches its physical limits people adjust their work. To a large degree, these adjustments are painless and make a great deal of sense to smooth out peak workload. People do such rational things as:

1. Put non-interactive jobs in the batch queue.
2. Change schedules by arriving early and leaving late.
3. Reduce the number of compiles and LINKs during debugging.

What if the system's capacity is *just adequate* for the peak demand level? Further, what happens if the processing load increases to 10% beyond the peak capacity, or what if the machine's performance is reduced 10% due to disk inefficiency? People react. They compensate by making *uncomfortable* adjustments to their working styles. They may use reports rather than on-line inquiry, communicate with colleagues in person rather than using electronic mail³, etc.

At this point, there is increasing pressure on the system manager to restore a "comfortable" computing environment for the users.

To quantify the costs of such an overly busy system, let's assume that our hypothetical company has a VAX 8650, an HSC50 and four RA81 disk drives. This company employs a total of 50 persons who use the computer as shown in Figure 10.

A HYPOTHETICAL COMPANY

HARDWARE	PERSONNEL
VAX 8650	10 managers
HSC50	15 data entry persons
DUA0:	15 secretaries
DUA1:	5 stocking clerks
DUA2:	5 programmer/analysts
DUA3:	

Figure 10

This computer system is currently *over-utilized* and as the computer load surpasses its capacity, employees begin to react as follows:

The Systems Manager reduces daily processing load by running backup and other maintenance functions at night. A half-time computer operator is hired at \$15,000/year. The DEC maintenance contract increases 8% (\$2600/year) as it goes from an 8-hour Monday-Friday contract at \$2770/month to 16-hour coverage at \$2992/month.

Because of frustratingly slow response time managers discontinue using on-line access to sales and inventory data. They request printed reports that cost \$570/year (for one third of an LG01 600 l.p.m. printer at \$11,950 and depreciated over 7 years). Additionally, \$500/year is spent to store and dispose of reports.

Because of the out-of-date, off-line information each manager makes at least one erroneous inventory decision per month. This costs an average of \$15 per decision, or \$1800/year.

Some managers decide to off-load their spreadsheet applications from the VAX to personal computers and five managers buy \$3000 personal computers. With a seven year depreciation schedule this costs \$2143/year.

Managers who previously used the MAIL utility to keep in touch with their employees now use written memos and visit employees. This requires an additional half-hour per day for each manager. Assuming the average salary is \$40,000/year, the two thousand hours spent on communication costs \$2500/year.

The forty supervised employees spend an additional five minutes each day for communication with managers. Salaries are: five programmers/analysts at \$36,000 (180,000); five stocking clerks at \$22,000 (110,000); fifteen data entry operators at \$19,000 (285,000); fifteen secretaries at \$20,000 (300,000). The total salary of non-management is therefore \$875,000, and the extra communication costs an average of \$438/hour or \$22/5-minutes. Over two hundred and fifty working days a total cost of \$5500 is incurred.

To distribute system load more efficiently data entry operators are scheduled in two shifts. To manage the second shift, the lead data entry operators is promoted at a cost of \$3000/year. Two operators cannot accommodate the schedule change and quit. The cost of recruiting replacements is \$1000 per replacement. Therefore, the total costs for splitting the shifts is \$5000.

As shown in Figure 11, the total cost of working a system that is being used at 10% beyond its "comfort" capacity is \$35,670/year.

HYPOTHETICAL COSTS	
2nd Shift Operator	\$15,000
DEC Maintenance	2,600
1/3 of LG01 printer	570
Reports, misc. costs	500
Stale Information errors	1,800
Personal Computers	2,200
Loss of MAIL utility	2,500
Extra employee time	5,500
2nd shift promotion	3,000
Recruiting	2,000
	\$35,670

Figure 11

³ And they are most likely to call you first to complain about the system's slowness.

ODS-2 Disk Optimization

This expense is in-line with that which could occur in any company (if anything, it is most likely to be accused of being *too* conservative). You'll want to perform similar computations specific to your own site and then talk with upper management about saving the company significant costs. When you compare the costs for not acting to improve system performance with the expense of new hardware and the cost of setting aside tuning time, it's obvious which choice offers the least investment of money for the results possible (see Figure 12). Even if the system manager spends just two hours every other week to analyze and tune the system to stay in the "comfort" zone, gains can be made. (At a well-deserved system manager salary of \$60,000/yr, this costs only \$1500 per year, which is far preferable to spending 8 hours a day, 5 days a week listening to complaints that the system is too slow.)

COST COMPARISON

Annual cost of <u>not</u> acting	\$35,670
New RAB1	\$13,000
Tuning effort (2 hours per week)	\$1500

Figure 12

Now that we've worked out the economics, we can appreciate the payback that results if "lost" performance is recouped from hardware through tuning. The areas to look at for tuning improvement include:

1. Adjusting your applications' RMS buffering.
2. Reducing volume fragmentation.
3. Placing files for best access performance.
4. Adjusting memory for XQP caches.

RMS BUFFERS

RMS file processing offers several options that are useful in getting better response time from application programs.

The easiest of these is to increase the size and/or number or type of buffers. This applies to sequential and indexed files. The \$ SET RMS/BLOCK_COUNT/BUFFER_COUNT command can decrease disk accesses by instructing the system to read more data per request. This is more efficient, but requires additional memory in the process' working set. An advantage to using the \$ SET RMS command is that it does not require any changes to your application programs. To use this command, place \$ SET RMS in the login command file for the group of users that use a particular application of interest. (However, be sure that you measure the effect the change makes to objectively determine if any improvement is produced.) Figure 13 shows the output from the \$ SHOW RMS command which reviews one process' settings for RMS buffering.

APPLICATION BUFFER CONTROL

```
$ SET RMS/BUFFER_COUNT
$ SET RMS/BLOCK_COUNT
$ SET FILE/GLOBAL
```

```
$ SHOW RMS
```

Process	MULTI-BLOCK COUNT	Indexed		MULTIBUFFER COUNTS		Sequential			NETWORK BLOCK COUNT
		Relative	Disk	Magtape	Unk	Record			
System	16	0	0	0	0	0	0	0	8
Process	Prolog	Extend		Quantity					
System	0	0	0	0	0	0	0	0	

Figure 13

Software Techniques, Inc.

Figure 14 shows the schematic relationship between the "private" I/O buffers and RMS buffers of two processes that are managed in P1 space to implement multi-buffering, read-ahead and deferred write-behind. The system *global buffer* in Figure 14 allows the two processes to share buffers to a heavily used, shared file.

BUFFER TUNING

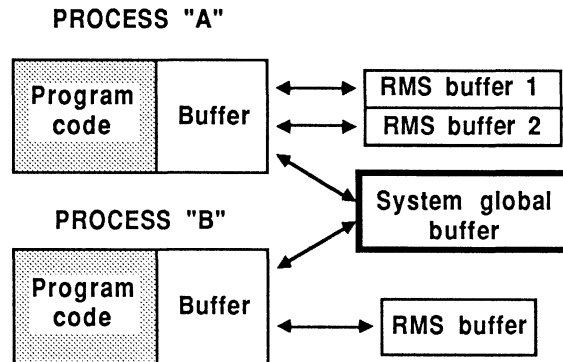


Figure 14

An application can be coded for read-ahead and write-behind processing to smooth the application's apparent interactive performance. This is accomplished automatically by RMS. It pre-fetches input and delays output operations "behind the scenes" to lessen the number of I/O operations physically requested of the disk. Many DEC languages process sequential files by this method by default. You must check the documentation for each particular language to determine if they do this.

As shown above, \$ SET FILE/GLOBAL_BUFFER=N is useful to reduce the aggregate amount of system I/O and main memory required by an application. If multiple processes are using the same file, DEC recommends that you combine the use of global buffers with local RMS multi-buffering.

FILE FRAGMENTATION

As files are created and deleted, every system acquires a certain amount of file fragmentation. But how do you know if this fragmentation is actually slowing system performance? There are four basic tools available to monitor fragmentation and its effect on performance.

\$ MONITOR FCP allows you to observe the "Window Turn Rate" & "Open Rate". The window turn rate is an indication of overhead imposed on VMS for having to contend with fragmented files. When you examine this display, look at the "CPU Tick Rate". This indicates how much CPU time is being used by VMS to handle file requests. Obviously, excessive CPU time indicates that the system is being forced into handling badly fragmented files.

\$ MONITOR FILE_SYSTEM_CACHE allows you to observe the "File Hdr (Hit percentage)". If your HDRCACHE is already large, you have either a high window turn rate or a very high file open rate. If the HDRCACHE is small, you might consider increasing its size.

\$ DUMP/HEADER allows you to list the number of retrieval pointers used to map a file in the "Map Area." portion of its display. It is equipped to give a detailed list of each file fragment, including the disk location and the size of each fragment.

DECUS / Nashville Spring 1987

ODS-2 Disk Optimization

\$ FRAG allows you to see free space location and file fragmented. The FRAG utility, a program included in our disk structuring utility kit, scans an entire disk and reports back the worst fragmented files found. This display is shown in Figure 15.

FRAG Displays the Worst Fragmented Files

Fragment count (ext)	Worst 100 Fragmented Files:
252(2)	[00,001]BCKMGR.LOG;348
101(1)	[SYSTEM]SWAPFILE.SYS;1
27	[001,001]BCKMGR.LOG;347
22	[001,001]BCKDUB1.LOG;3
22	[SYSTEM]ERRLOG.SYS;1
21	[WEF]DISKIT_DSU.EXE;10
18	[DMP]TEST0.PHYS;2

Figure 15

The FRAG utility also surveys the fragmentation of the disk's free space and reports it graphically as shown in Figure 16. The distribution of volume free space is important for several reasons. First and foremost, free space is used to create new files and extend old files. Because free space distribution controls the location and fragmentation of new allocations, if free space is fragmented then all new files created from it are fragmented.

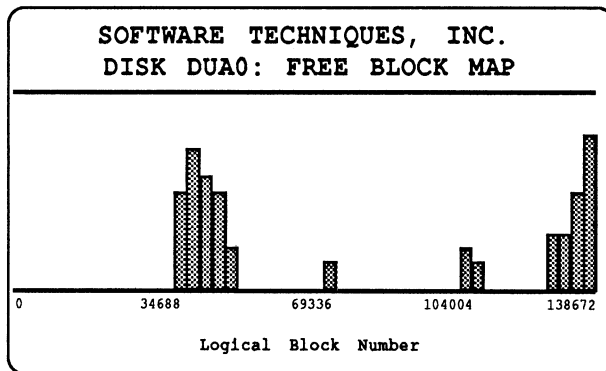


Figure 16

DEFRAGMENTING FILES AND FREE SPACE

The BACKUP utility is the traditional method of defragmenting files, but this is a by-product of the process and not what it was designed to do. BACKUP/RESTORE is a three step security process. First, you perform a full BACKUP, then reinitialize the disk, and finally restore all the files.

\$ BACKUP is not the perfect solution, but is about the best that vanilla VMS offers.⁴ BACKUP/RESTORE usually requires an operator to conduct the procedure, which is time-consuming, especially with tape. You also run the risk of data loss if the save-set is destroyed or mis-sequenced by the operator. Assuming all goes well, the files on restore are defragmented, but your directory files are

⁴ At DECUS/Nashville, DIGITAL announced plans to provide a utility for defragmenting files. However, this utility will not be finished until *after* the next major version of VMS is released, sometime in mid-to-late 1988.

Software Techniques, Inc.

still dispersed across the disk.

\$ COPY allows you to defragment files individually. The /CONTIGUOUS qualifier attempts to find contiguous free space, but uses separate spaces if extra space is needed. Unfortunately, the COPY command does not report the fact that the latter has occurred. The /EXTENSION qualifier can also be used to specify an explicit extend factor for the new file to lessen the fragmentation of future additions to the file when the file grows.

There are many controls at the QIO, RMS or FDL level that qualify what free space is returned by VMS. Volume location can be specified by cylinder number, logical block number, or relative to an existing file. In addition, an approximate location or an exact location can be specified to return an error if the location is not available. If "contiguous only" is specified, VMS returns a single span of sequential disk logical block numbers. "Contiguous best try" asks VMS to do its best, but VMS uses separate spaces on the disk if that is all that's available (in this case no error is returned).

Unfortunately, both QIO and RMS are awkward to use. \$ CREATE/FDL can be a real aid to exercise explicit control over file contiguity and offers a much more friendly user interface.

KEEP A HEALTHY FREE SPACE RESERVE

Disks should be reorganized regularly, and at least 10% of the volume's storage should be kept free. This reduces the rate at which free space is recycled for new file allocations. Disk fragmentation levels escalate rapidly when the free space is recycled frequently. Fragmentation can be moderated by having an adequate free level at all times. If you have a lot of files that are not frequently used, archive them to tape or removable disks. Failing that, move those files to the periphery of the disk so that the disk heads do not have to skip over them while actively accessed disk data.

IMPROVING FILE ACCESSIBILITY

Earlier we spoke of the importance of minimizing the mechanical motion of the disk drive, especially disk head seeks. Defragmenting files can reduce the need for many seeks, but once you've accomplished this, the next goal is to reduce the *distance of the seeks* that must occur in any case. This is accomplished by identifying your most actively utilized files and moving them close together.

\$ SHOW DEVICE/WINDOWS displays all files open on a given disk and can therefore be used to identify the most active files on heavily used disks. (In a VAXcluster, \$ SHOW DEVICE does not list *all* file activity if the disk is mounted /CLUSTER. Files open on such disks are not reflected in the display on other VAXcluster nodes and you must perform the \$ SHOW DEVICE command on each node.)

\$ PROCESS is a utility that we have written which identifies open files by a selected process. This can be useful if you suspect that a particular application program is generating an overload of disk requests. The PROCESS utility helps you identify the files being used and the I/O counts to each file, which is particularly useful for applications whose internal operation is not well documented.

\$ MONITOR PAGE shows you the demand level for disk I/O to the pagefiles. \$ SHOW MEMORY lists the active pagefiles and their actual usage. To make changes in pagefiles or swapfiles, the \$ SYSGEN utility is used. Remember, when SYSGEN extends an existing pagefile or swapfile, that file becomes fragmented if contiguous space is not available.

Installed images are made known to the system because they are often used by the image activator. Even though "installed" these files are repetitively paged into the working sets of each user who runs them. Therefore, these files, along with the pagefile⁵, are good candidates for placing in an optimum disk location.

Often, the middle of the disk is the optimum location for quick file access. This location is shown in Figure 17. However, the ideal definition for the optimum location is that location which allows fastest access to the file. This location can therefore be at the beginning of the disk for a disk with few files.

⁵ Specifically, SYSSYSTEM: *SHR.EXE files are ideally suited for optimum positioning.

DECUS / Nashville Spring 1987

WHERE IS THE "MIDDLE" OF THE DISK?

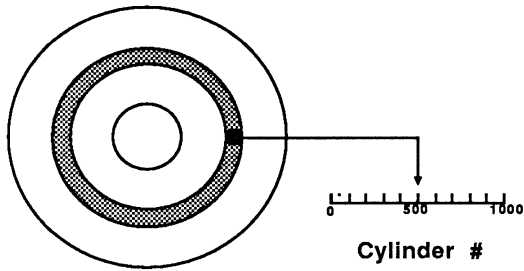


Figure 17

PLACING FILES DELIBERATELY

With QIO and RMS you can request specific allocation areas for new files. The QIO and the associated File Information Block (the FIB) are not easy to use, however. The easiest commands to learn are CREATE and EDIT/FDL as shown in Figure 18. These commands also allow you to specify all of the file options for creating files. (An even easier option is to use a restructuring utility that automatically performs file placement as requested by the system manager.)

CREATING A "PLACED" FILE WITH "\$ CREATE/FDL"

FILE	NAME	[] FILE.EXT
AREA 0	ALLOCATION	2000
	EXACT_POSITIONING	yes
	POSITION logical	35000

Figure 18

BALANCING LOAD ACROSS SPINDLES

If you know where your I/O is taking place you can balance disk load across multiple spindles for better performance. \$ MONITOR DISKS gives you an I/O breakdown by disk. To distribute load, however, think about the following possibilities:

1. Distribute user default directories to idle disks.
2. Use volume sets to distribute indexed files by areas.
3. Use shadow sets to reduce disk head read contention.

The FDL editor lets you distribute the key structure of an indexed file to one physical disk and the data buckets to another. One set of heads can then be kept available at the index buckets while the other disk scavenges for data. Shadow sets can also be used to distribute I/O traffic to multiple spindles. Aside from their value as fail-safe storage, shadow sets provide better read performance by sharing the read load between disks.

SYSTEM CACHE EFFECTIVENESS

The cache areas can be tuned by adjusting the following SYSGEN parameters:

ACP_MAPCACHE controls how many blocks of the allocation BITMAP are loaded.

ACP_HDRCACHE controls how many file headers from the index file are loaded.

ACP_DIRCACHE controls how much memory is used to store blocks of directory files.

ACP_WINDOW controls the default number of window pointers allocated in a window. (This parameter can be overridden when specific volumes are Mounted or specific files opened.)

The file system caches reside in system paged and non-paged pool. Caches in non-paged pool are not "caches" in the strictest sense in that they do not contain actual blocks of disk data. Rather, they contain small shorthand data structures that summarize active areas of the disk. It is important that these caches be adequate in size, but the system defaults and those that AUTOGEN computes are almost always correct for every system. The ANALYZE/SYSTEM utility displays the amount of non-paged pool used via the SHOW POOL/NONPAGED/SUMMARY command. In Figure 19, the "VCA" line represents the "volume cache" and contains the quota block entries, file ID numbers and the list of recently freed extents.

```
SDA> SHOW POOL/NONPAGED/SUMMARY
```

Non-paged dynamic storage pool		
Summary of non-paged pool contents		
50	UNKNOWN	= 71808 (25%)
1	ADP	= 1184 (0%)
1	LOG	= 32 (0%)
10	PCB	= 2880 (1%)
35	UCB	= 17280 (6%)
4	VCB	= 960 (0%)
1	WCB	= 224 (0%)
4	TYPABD	= 1472 (0%)
8	DPT	= 92576 (32%)
5	RBM	= 28144 (10%)
1	VCA	= 2532 (0%)
10	SCS	= 19104 (6%)
1	LOADCODE	= 2752 (0%)
3	INIT	= 39344 (14%)
1	UIS	= 352 (0%)

*Disk Quota Blocks
Free file-ID numbers
Recently freed extents*

Figure 19

SHOW POOL/SUMMARY/PAGED displays the amount of paged pool being used for the caching areas set up by the SYSGEN parameters. Paged pool is loaded with cached blocks of the BITMAP.SYS file, blocks of directory files, the directory index table and file headers. These caches are considerably larger than those that reside in non-paged pool. The SYSGEN parameters controlling the HDRCACHE, MAPCACHE and DINDXCACHE can be increased rather liberally without penalty because the paged pool does not physically reside in memory unless being used.⁶ As shown in Figure 20, the paged pool caches appear in the \$ ANALYZE/SYSTEM display as "UNKNOWN".

```
SDA> SHOW POOL/PAGED/SUMMARY
```

Paged dynamic storage pool		
Summary of paged pool contents		
8	UNKNOWN	= 57824 (51%)
1	PQB	= 2256 (1%)
55	GSD	= 3296 (2%)
73	KFE	= 4704 (4%)
3	MTL	= 128 (0%)
19	KFRH	= 5360 (4%)
1	TWP	= 12336 (10%)
1	RSHT	= 528 (0%)
88	LNLM	= 7440 (6%)
6	KFD	= 320 (0%)
1	KFPB	= 16 (0%)
1	PMB	= 1792 (1%)
2	ORB	= 5472 (4%)
2	QVAST	= 11904 (10%)

*File headers
BITMAP.SYS blocks
Directory blocks
Directory indexes*

Figure 20

⁶ Remember, however, that an increase in these caches' sizes should be accompanied by an increase in NPAGEDYN to accommodate them.

ODS-2 Disk Optimization

RESULTS

From experience, we know that disk tuning really does improve the performance of any system. Here's some example results of what you might expect to see from tuning efforts:

DIRECTORY SEARCHES OPTIMIZED

In an evaluation of our disk structuring methods published by the *DEC Professional Magazine*, a full volume-wide directory search took 37% less time after restructuring the disk. This is shown in Figure 21. The improvement was primarily due to the specific placement of directory files next to the index file.

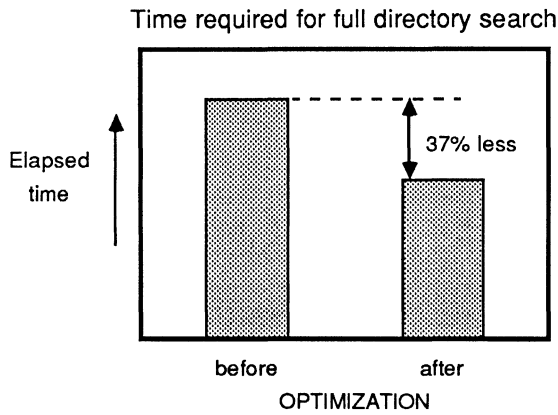


Figure 21

FILE CACHES MORE EFFECTIVELY UTILIZED

In another test, the bitmap cache hit rate measured 14% more hits after restructuring the disk (as measured by VAX/SPM). This is shown in Figure 22. The hit rate improvement was due mostly to the fact that free space was contiguous after restructuring and could therefore be assigned more efficiently.

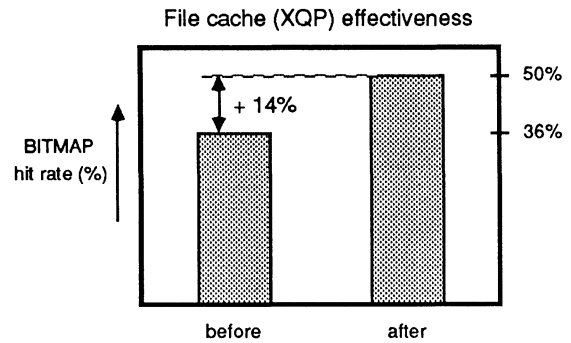


Figure 22

UETP 20-USER LOAD BENCHMARK

The UETP 20 user load was compared before and after restructuring to measure the improvement in disk response time. This is shown in Figure 23. The improvement resulted in a 35% improvement in user response time.

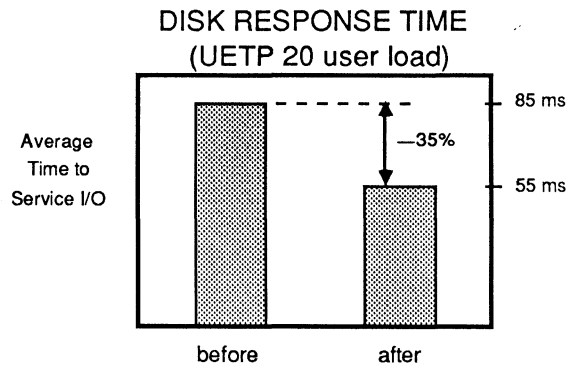


Figure 23

SUMMARY

We began by talking about how a systematic approach to disk tuning gives you the most benefit. The basic operational constraints of disk drives was then explained, so you can understand if you have a performance problem and hypothesize what can be done to eliminate it.

It was mentioned that it is just as important to understand the way VMS organizes the disk, according to ODS-2 structure, as it is to understand where the system might be experiencing bottlenecks in servicing disk requests. We also tried to understand some of the internal mechanisms VMS uses to eliminate the need for some disk accesses through caching and buffering.

Various utilities were then listed for improving system performance, and some tips for locating the source of problems were provided. We also spent some time describing the methods for changing parameters that might help your system.

From this point, it is your own initiative that determine how much system improvement is gained on your system. Disk performance management is one of the most challenging and exciting aspects to site management, and is an outright necessity at any large installation. Practicing these techniques will give you a better understanding of your own system. When you find the bottlenecks that limit your system, you can put these ideas to good use.

To contact the author, please write to:

Wef Fleischman
Software Techniques, Inc.
6600 Katella Avenue, Cypress, CA 90630
714/895-1633

On-Line Security Monitoring System

Dr. Marino J. Niccolai
University of South Alabama
Mobile, Alabama

Linda B. Lankewicz
Spring Hill College
Mobile, Alabama

Abstract

The feasibility of constructing a model of user performance based upon monitored activities is discussed. The premise that users' activities can be determined by their utilization of system resources is supported by statistical analysis. A real-time analysis of activities is developed to allow system managers to detect classes of users. In particular, browsing, the act of searching for hole in the access control system, can be detected. The research shows a high correlation between browsing and the size of the working set, the number of images generated, and page faults. Data was collected using VAX VMS MONITOR and ACCOUNTING utilities and the \$GETJPI System Service.

This paper is the result of a research effort to develop an algorithm for identifying security violations. While VMS provides a means of auditing, the research is concerned with developing an on-line security monitoring system.

A system manager's responsibilities include a number of areas which require knowledge of parameters of resource utilization. These include creating access control lists for directories, setting working set limits when authorizing users, evaluating the users' use of cpu time in ACCOUNTING, and tuning the system. Security is one aspect of the responsibilities. A system manager might establish procedures for the physical security of the system and rely on the mechanisms of the operating system for the internal security.

A system manager develops a certain sense of how activities utilize a particular system. If you spend time monitoring users, you begin to recognize some activities based upon how they are using resources. For the specific system with its parameter settings, an experienced manager develops an intuition about what activity is taking place based upon resource utilization. It would appear that a process' use of system resources could be used to identify the activity of the user. The parameters for the use of system resources are listed below. Disk utilization parameters are faults, reads, and direct I/O requests. Buffered I/O is an indication of terminal usage. The working set size and the number of images generated reflect the memory usage, and the processor time indicates how the cpu is utilized.

Parameters
page faults
page fault reads

amount of buffered I/O
working set size
number of images generated
processor time used

In addition, other resource parameters might be calculated. For example, it would be misleading to look solely at page faults. The number of faults per image would be more indicative of the behavior of the user. Other ratios which were investigated include the processor time per image, the buffered I/O per image, and the ratio of faults to direct I/O.

Calculated Parameters
page faults per image
processor time per image
amount of buffered I/O per image
ratio of faults to direct I/O

Two questions were asked at the outset of the research. Can a process' use of system resources be used to identify the activity of the user? Can a process' use of system resources be used to determine whether the process is a security threat? In some instances monitoring users, a system manager is able to detect unusual activity based upon abnormal patterns of resource utilization. If the research determined that some of these parameters could be used to identify user activities, they could also be used to identify a security threat.

The research was an effort to quantify this process. It consisted of two phases. First a statistical analysis was performed to show that a combination of these parameters could be used to identify user activities. Then a model for a security-

threatening behavior was developed along with an on-line security monitoring system.

The Department of Defense in the "Orange Book," Trusted Computer Systems Evaluation Criteria, states:

The TCB shall contain a mechanism that is able to monitor the occurrence or accumulation of security auditable events that may indicate an imminent violation of security policy. This mechanism shall be able to immediately notify the security administrator when thresholds are exceeded.

It is not sufficient to audit past activities of users. The trusted computing base should contain a mechanism for on-line monitoring of events which might possibly pose a threat to security. If specific activities are identifiable, unusual activities should also be recognizable. One unusual activity, browsing, was selected and a model developed that could be used on other systems.

Browsing consists of attempts by knowledgeable users to compromise the security of the system. No operating system has been able to withstand penetration attempts by knowledgeable users. Such a person might have a legitimate account on the system. The browser's efforts may include probing the operating system for weaknesses, searching for unprotected files, and attempting to change addresses at the channel level. Rather than depending solely on the ability of the operating system to protect its objects, some attempt should be made to dissuade users from activities which match this profile.

To develop an on-line detection of possible security threats, the activities of users on the VAX 11/750 at Spring Hill College were analyzed by examining data collected in ACCOUNTING. The ACCOUNTING Utility provides information about page faults, reads, peak working set, peak page file, direct I/O, buffered I/O, images executed, elapsed time, and processor time.

An ACCOUNTING record is written for each process termination. The system manager is not able to look at ACCOUNTING to see the variations in the parameters over the life of the process. The values recorded are the peak values or the total value when the process terminated. The peak working set size attained is recorded rather than the final working set size.

The values recorded in ACCOUNTING were analyzed to determine whether the parameters could be used to identify the activities of users. A discriminant analysis was performed on three groups of Spring Hill College users whose activities were known. Data was analyzed for 11,0006 user processes during one semester of use.

Group 1 consisted of a senior English class word processing a collection of poetry and prose. Group 2 consisted of a beginning programming class involved in editing and using the Pascal compiler. An interactive statistical software package was used by Group 3, a business statistics class unfamiliar with the VAX.

None of the classes received instruction in the use of other VMS features or software.

The group means for some of the parameters appeared to be different, as shown below. A discriminant analysis of

the data showed that a combination of these parameters could be used to identify the activity of the user. This research is contained in the thesis "Resource Utilization and Security," by Linda B. Lankewicz. The combination of parameters examined could be used to identify a word processing activity 90% of the time and the other two groups 70-75% of the time.

Once it was established that resource utilization parameters could be used to identify user activities, the research effort was directed towards determining how an activity such as browsing would use system resources.

The MONITOR utility gives the current status of a process. It can be used to derive a profile of the activity of a user in terms of shared pages, working set size, direct I/O, page faults, and processor time. MONITOR is used by a system manager to observe the activity on the system in terms of these parameters. It would be helpful if Monitor provided the number of images being generated and the buffered I/O, but these items are not available.

The information provided by MONITOR is hard to digest whether viewing the output interactively or reading it from a file. Its most effective use is in following the activity of one or two processes. The on-line detection system developed during this research would alert the system manager when a process' activity warranted close monitoring.

MONITOR data was captured in a text file at one-minute intervals. An editor was used to delete the system processes and place the time on each row of information as shown below. This data was then imported into LOTUS123. Each line of text file data enters LOTUS123 as one cell so the data must be parsed into separate columns. Then the data can be sorted by process name so that information for each process is grouped together. This allows the system manager to see all of a process' activity. LOTUS123 graphs can be employed to view the changes in working set size, direct I/O, or faults over the life of the process.

For the system in this research, the WSDEFAULT was 200, the WSQUOTA was 500, and the WSEXTENT was 1000. The working set of processes involved in word processing immediately faulted in a working set size over WSQUOTA even if the document was only one word in length. This working set size was maintained as long as the process continued to use the word processor. When a process left the word processor, it was trimmed to WSDEFAULT.

Processes using the editor maintained a working set size in the range 350-550. These processes maintained a steady working set profile although the size was much smaller than that of a process word processing the same size document.

A process involved in browsing had a working set which stayed near or below WSDEFAULT. This was the result of the trimming that occurs with each image exit. Browsing activity is characterized by the generation of many images as the user attempts to examine files or move through the directory hierarchy.

In addition to the working set, other parameters were examined in terms of dynamic behavior for the groups of users. For the particular system used in this study in an academic environment, the values listed below were significant in determining whether a process might be browsing. For another

GROUP MEANS FOR ONE SEMESTER

GROUP	NO. OF IMAGES	PEAK WORKING SET	BUFFERED I/O PER IMAGE	PROCESS TIME PER IMAGE	DIRECT I/O PER IMAGE
1	6	895	498	7.7	71
2	14	516	133	2.1	15
3	11	378	61	1.3	14

MONITOR Output

Process Count: 11 VAX/VMS Monitor Utility Uptime: 0 20:52:31
 PROCESSES

10-JUN-1986 13:38:00

PID	STATE	PRI	NAME	PAGES	DIOCNT	FAULTS	CPU TIME
00000080	COM	0	NULL	0/0	0	0	20:08:30.0
00000081	HIB	16	SWAPPER	0/0	0	0	00:00:29.3
00000084	HIB	8	ERRFMT	0/85	411	67	00:00:11.5
00000085	LEF	8	OPCOM	0/59	79	1094	00:00:02.1
00000086	HIB	8	JOB_CONTROL	0/287	428	174	00:00:24.4
00000087	HIB	6	SYMBIONT_0001	0/46	20	915	00:00:10.6
00000B08	LEF	5	JONES	15/1000	1844	1610	00:01:46.5
00000B97	LEF	9	TTB1:	22/805	83	1087	00:00:07.4
00000D9A	LEF	9	MANAGER	25/775	75	854	00:00:06.0
00000D9D	LEF	4	TTA6:	25/136	1	126	00:00:00.4
00000D24	CUR	6	LANKEWICZ	47/434	67	871	00:00:05.9
12:46:00	00000603	LEF	6 STUDENT3824	49/200	207	1699	00:00:59.7
12:46:00	00000608	LEF	9 STUDENT1487	32/823	229	1149	00:00:27.0
12:46:00	0000060A	LEF	9 STUDENT1820	28/934	452	2098	00:00:38.7
12:46:00	0000060C	HIB	4 LANKEWICZ	59/200	407	5550	00:00:49.1
12:46:00	0000060D	CUR	5 BATCH_555	28/405	53	621	00:00:04.0
12:46:00	0000060E	LEF	5 STUDENT3209	60/182	274	1499	00:00:11.7
12:46:00	000005A0	LEF	7 STUDENT3394	29/1000	2632	2146	00:01:51.6
12:46:00	00000529	LEF	9 STUDENT1836	27/1000	831	1781	00:01:53.8
12:46:00	000005B4	LEF	9 STUDENT0390	27/1000	1472	2348	00:01:40.9
12:47:01	00000608	LEF	7 STUDENT1487	28/858	265	1315	00:00:28.6
12:47:01	0000060C	HIB	4 LANKEWICZ	59/200	407	5550	00:00:49.1
12:47:01	0000060D	CUR	5 BATCH_555	28/405	57	621	00:00:04.0

Figure 1: Edited MONITOR Output

MONITOR Output in LOTUS123

time	pid	username	shared	w.set	dirio	faults	cputime
0806	0E1D	STUDENT8000	84	204	55	576	00:00:03.8
0807	0E1D	STUDENT8000	69	203	59	908	00:00:07.7
0808	0E1D	STUDENT8000	98	216	77	1179	00:00:09.9
0809	0E1D	STUDENT8000	40	150	159	1402	00:00:14.8
0810	0E1D	STUDENT8000	60	175	166	1696	00:00:16.2
0811	0E1D	STUDENT8000	41	145	170	1768	00:00:17.2
0812	0E1D	STUDENT8000	60	166	170	1840	00:00:17.5
0813	0E1D	STUDENT8000	45	200	202	2332	00:00:20.4
0814	0E1D	STUDENT8000	48	161	203	2466	00:00:21.0
0815	0E1D	STUDENT8000	60	212	219	2510	00:00:21.6
0816	0E1D	STUDENT8000	62	184	219	2578	00:00:22.1
0817	0E1D	STUDENT8000	56	157	251	3370	00:00:26.9
0818	0E1D	STUDENT8000	58	168	261	3703	00:00:29.4
0819	0E1D	STUDENT8000	59	171	263	3891	00:00:30.5

Browsing Model

Pages in the Working Set	<= 300
Number of Images	>= 30
Page Faults per Image	<= 150
Page Faults per Working Set Page	>= 15

environment with different parameters limits in place, other values could be determined.

Once it was determined that user activities could be differentiated and the parameters for browsing were identified, the implementation goals were determined to be low overhead, simplicity, and consistency. Any added security on a system has a cost since it will also be competing for system resources. An effort should be made to minimize that cost so that the reduction in performance is minimized. The goal of simplicity is related to efficiency and maintainability. Consistency is a goal because any real security feature must be applied to all users of the system.

Two approaches were considered for the implementation. One was the creation of a shadow process for each login process which would monitor the activities of the user's process. The other implementation would be a single process taking snapshots of all the system's processes.

The creation of shadowing processes at login would offer the advantage of individualized treatment. Yet this inconsistent treatment of processes could be a disadvantage as a shadowing process might be lulled into monitoring its twin process less often. The single process implementation would not bypass any process but take snapshots of all processes at set intervals. This would satisfy the consistency goal of the implementation. It would mean that even the system manager's activities would be monitored. If a system manager forgot to log off at one terminal and someone was using the account to browse, that activity would be detected and an alert signaled.

The shadowing processes approach also was rejected be-

cause of the additional overhead which would be generated by the creation of the shadowing processes. In addition to the overhead of process creation, the number of processes on the system would double. This might be acceptable in a dedicated system, but not in an academic environment nor on most multiprogramming systems.

The snapshots of process behavior involve calls to \$GETJPI to access the parameter values identified as important for detection of browsing. The snapshots are taken by a single process which hibernates, wakes up at the scheduled time, samples the activities of all processes on the system, and decides whether the parameters fit the browsing model. If browsing is detected, a bit is set in a bitmap associated with the port which the process is using.

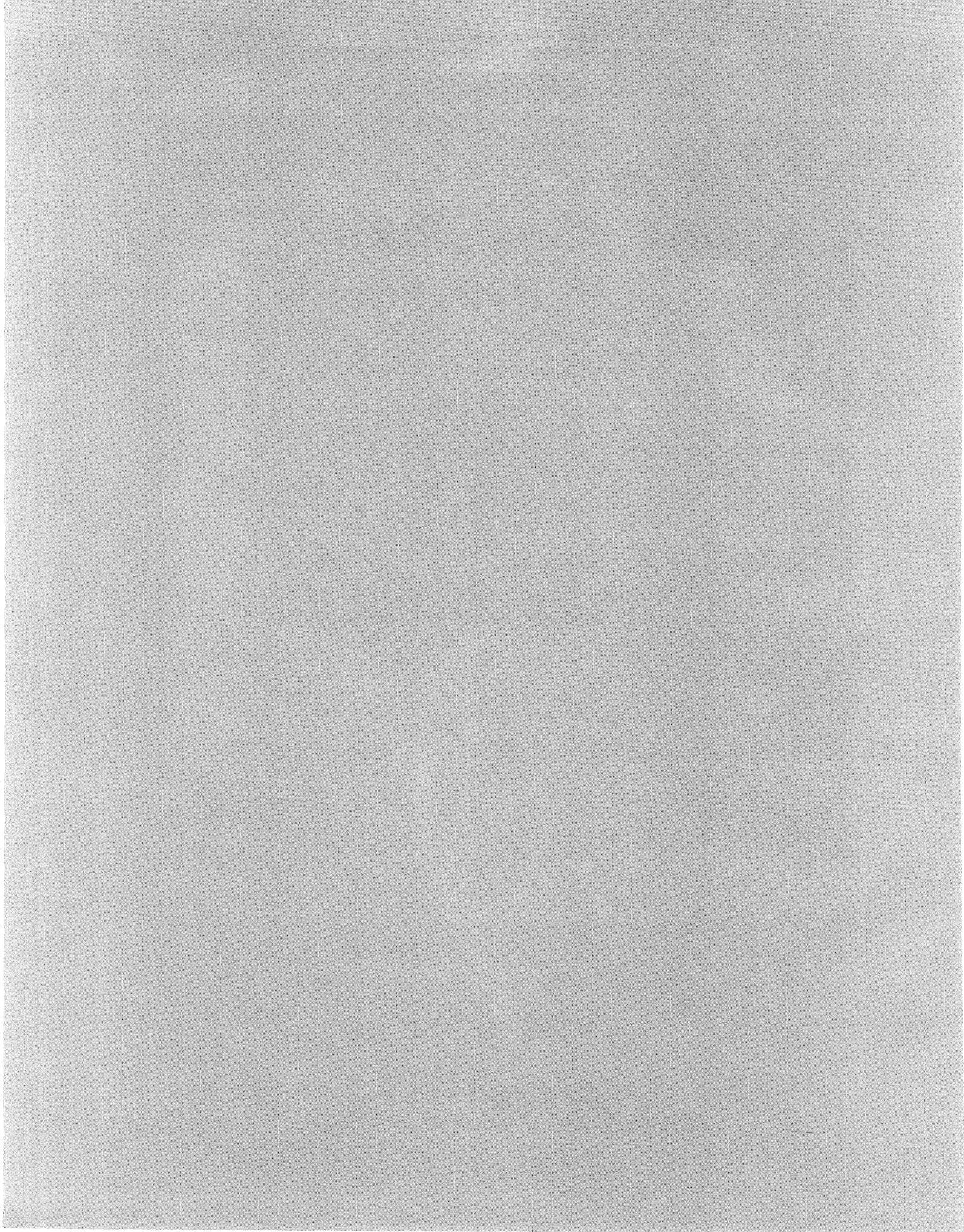
The bitmaps are rotated with each snapshot so that they provide moving windows reflecting the behavior of the users at each port. Some criteria is used to determine when to notify the system manager that the user might be browsing. All users display browsing-type behavior at times when conducting legitimate activities in their accounts. It is sustained browsing behavior that should be noted so that the system manager can investigate further. For the implementation in this research, snapshots were taken at one-minute intervals and the system manager was notified when four consecutive bits were set indicating four minutes of browsing activity.

The research provided a methodology for modeling characteristic user types and showed a statistical differentiation among user profiles. Then an on-line monitoring system was developed to detect a security-threatening behavior.

Further research is needed to refine and extend the analytic model of browsing and to characterize and validate models for a full range of user groups. The work should be extended to other operating systems and the implications for networks should be investigated.

REFEREED PAPER COMPETITION SUBMISSIONS

*This section includes papers that were submitted as entries into the
1987 Spring Refereed Paper Competition
and were not selected as finalists.*



ESTIMATING DEVELOPMENT AND RUN TIME RESOURCES: A PRACTICAL EXAMPLE

Anthony C. Picardi, Sc.D.
Cortex Corporation
Waltham, Massachusetts

ABSTRACT

An empirically based method of estimating development and run time resources is presented. The study is based on a survey of Application Factory users in the DEC VAX/VMS environment. A brief overview of the Application Factory is given. Results of a survey of fifty-two applications developed with the Factory are presented. Application size and complexity is expressed in terms of function points and this measure is then used to estimate development resources based on the statistical relation found in the survey. A worksheet is included so the reader may calculate function points and development effort for both the Application Factory and COBOL. A comparison with another statistically-derived equation relating function points to COBOL effort indicates that Factory development productivity relative to COBOL increases as the size and complexity of the application increases. Survey data show that number of terminals and transaction volume, are the best indicators of hardware resources and that amount of CPU memory is more sensitive to number of users than CPU type. A table is presented which qualitatively relates the surveyed applications to their hardware and run time environments. This survey results indicate the practical limits of what can be quantitatively learned about machine resource requirements via telephone surveys of application developers.

1. APPLICATION FACTORY OVERVIEW

The Application Factory is used to convert an information management application data model and operational specification into an implementation on the full range of DEC VAX computers (and Clusters) running the VMS operating system. An application as it is used here is an integrated set of source modules and data definitions, not just a heap of screen and report programs. The Factory is best used by small groups of developers working in a "prototyping environment" in which entire applications or modules of large integrated applications are developed in one- to three-month time frames. The Factory is most

appropriate for applications where the acceptance criteria emphasize operational functionality, screen and report content, online multiuser flexibility and speed of implementation rather than cosmetic issues of appearance or specific navigational keystrokes.

The core of the Factory is the generator, which is capable of reading specifications about an application and generating a compiled and linked executable image. The process of collecting the specifications has been automated via a menu- and form-oriented user interface with a built-in smart guidance system. Since specifications are viewed as "meta-data" about generic parts of an application, this user interface is in fact a generated Factory application. The input to

the generator is a set of RMS data files while the output is a set of interacting modularized object modules. Data for Factory applications is stored in either RMS files or an Rdb database or a combination of the two.

The Factory development environment also includes a procedural language called Builder which is used via an action diagrammer to flowchart and then generate code for screen and report customizations or for standalone procedures such as purging or importing data. Maintenance to Factory applications is done by changing the specifications, after which the factory automatically regenerates the affected program object modules.

2. SURVEY RESULTS

The survey frame consisted of Factory applications which had been completed over a period of time from June, 1985 to December, 1986. Fifty-two questionnaires were completed either via mail or telephone interviews. Table 1 summarizes the results which describe the applications and their run time characteristics. The average application can be characterized as having 38 screens, 18 reports, 31 datasets and 9 non-screen procedures. This amounts to 810 function points, as defined below, and is the equivalent of approximately eighty-five thousand lines of COBOL code. Non-screen procedures typically

TABLE 1: Summary Statistics for Survey of 52 Factory Applications.

<u>Item</u>	<u>Response</u>
Average number of:	
screens	38
reports	18
non-screen procedures called from menus	9
key screens	29
datasets (files)	31
interfaces to HW/SW systems	1.6
size of largest file	138,953
person-weeks for development	36
development team size	2
function points	810
lines of COBOL-equivalent code	85,100
months prior Factory use	10
Number of applications using/having:	
VAXMAIL, DECNET or VMS Broadcast	15
interacting CPUs or clusters	12
FDL data file tuning	15
performance-related data design	19
response time < 1 second	14
response time 1 to 3 seconds	16
response time > 3 seconds	2
process state monitor	9
audit trail	11
appends from tape/disk	12
data purge/archive	20
procedures automatically startup/shutdown	15
special navigation	9
security	28
installed at more than one site	8
designed for significant later modifications	43
used by or sold to third parties	11

include data purges or appends of data from tapes. A process state monitor was specified in 9 of the applications to control the work flow based on past activities. For example, the company consolidation report could not be run until the divisions run their monthly accounts. Note that most applications, 43, were designed as modules of larger applications or as first cuts to be significantly changed during maintenance. Prior Factory experience is most probably biased upward, since it is a simple average for the team, not taking into account the percent of time an individual worked on the project. It is biased upwards by a few consultants who appeared on many projects for short periods of time with over 120 months of Factory and Builder experience.

3. ESTIMATING DEVELOPMENT AND RUN TIME RESOURCES

The object of this study is to develop a method of estimating the development resources needed to produce business applications with the Application Factory. The procedure is to first describe the application in terms of function points and then derive the required development resources by means of a statistical model relating function points to person-weeks for Factory applications. The rationale for using an implementation independent metric such as function points to measure and compare

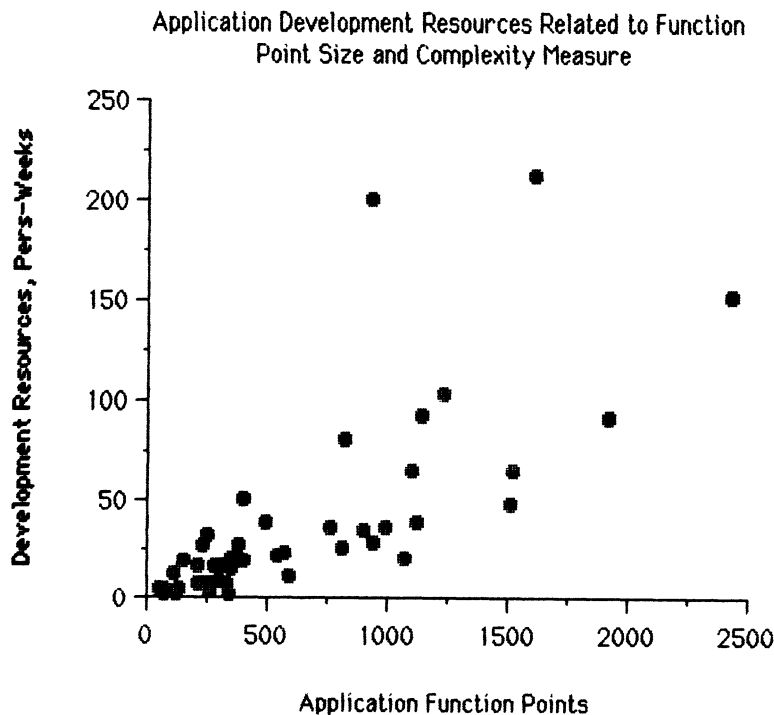
applications is well developed in the literature (1, 2, 3, 5) and has been used to compare COBOL and Factory productivity on a sample of 26 applications (4). The worksheet used to calculate function points is shown in Appendix A and involves the weighted addition of the application's extensive parameters, such as the number of screens, reports and datasets. This sum is then modified by a set of difficulty factors such as the importance of telecommunications or performance, for example.

The relationship between function points and development resources is shown by the scatter plot in Figure 1 for 48 applications. The best fit to these data is the linear model:

$$\text{Person-weeks} = -1.57 + .064 * \text{function points.}$$

This model had a correlation coefficient of 0.74 and an F-statistic, $F(1,46)$, of 55.6 indicating significance for the relation above 99%. Although it would seem that an exponential model would better explain the data by allowing for the "decreasing returns to scale" effect as the project size (function points) increased, such an exponential model resulted in both a lower correlation coefficient and a less significant fit to these data. Attempts to explain more of the sample variance by the addition of development team experience as an independent variable did not result in a significant coefficient for

Figure 1:



experience, although the sign was correct – more experience reduced required person-weeks. The failure of experience to contribute to the relation may be explained by the fact that it was the average months of Factory and Builder experience for the entire Development team unwieghted by the fraction of time each developer spent on the project. Thus teams which had a highly experienced consultant for a short period of time probably ended up with a higher experience score than warranted. An earlier study of a smaller sample found that if the magnitude of participation of the consultant was taken into account, the result was a highly significant difference in efficiency between "novice" and "experienced" developer teams (4).

4. USING FUNCTION POINTS TO ESTIMATE PRODUCTIVITY DIFFERENCES

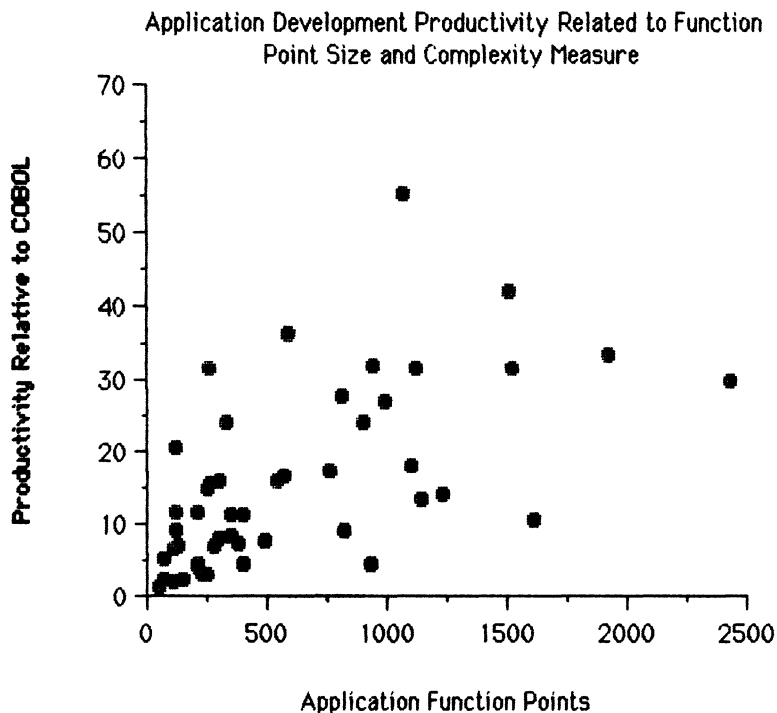
One benefit of using function points to measure application size and complexity is that the same application can be estimated for implementation in a variety of languages, and from this the relative productivities of using those languages can be compared. The only other statistical model for business applications in the DEC environment the author had access to was that used at Dupont and supported by a database of over 200 applications,

most of them in COBOL. This model is presented in appendix A (item 5.2). A productivity metric was defined as the ratio of the estimated person-weeks to develop the application in COBOL (using the model in 5.2) to the actual person-weeks using the Factory (from the survey). A plot of the productivity ratios, in Figure 2, shows a strong and significant increasing trend as the application size increases. Based on 48 cases, the following linear model had a correlation coefficient of 0.60 and an F-statistic, $F(1,46)$, of 26 indicating significance for the relation at greater than 99%:

$$\text{Productivity} = 7.43 + 0.013 * \text{function points.}$$

The average productivity increase over COBOL was a factor of 15 with a standard deviation of 12. It is obvious from the statistical models that the Factory will show increasing advantage over COBOL as the project size increases since the exponential COBOL model will necessarily diverge from the linear Factory model as function points increase. The difference in the two models is supported, however, by the fact that the Factory is an application designed specifically to aid in the configuration management of the many interacting modules that make up a large application, a task that becomes increasingly difficult as the application size and complexity increases.

Figure 2:



Specifically the Factory incorporates a developer guidance system, automatically generated documentation, online data field cross referencing and the automatic selective regeneration and recompilation of modules affected by changes in specifications, for example.

The survey sample was analyzed to discover how differences in efficiency across applications can be explained. Efficiency defined in terms of function points per hour was regressed against project size and experience. Only project size showed a statistically significant but small correlation, with a coefficient of 0.43 and an F-statistic, $F(1,46)$, of 10. This relation, indicated by the scatter plot in Figure 3, included 48 cases and was significant at greater than 99%. In spite of the fact the linear resource model above indicated no significant decrease in efficiency with project size when overall resources were regressed with function points, Figure 3 shows the expected decrease in efficiency to be small but significant when project size is measured in person-weeks:

$$\text{Function points/hour} = 0.82 - 0.0043 * \text{Person-weeks}$$

The fact that experience was once again not found to be significantly correlated with development

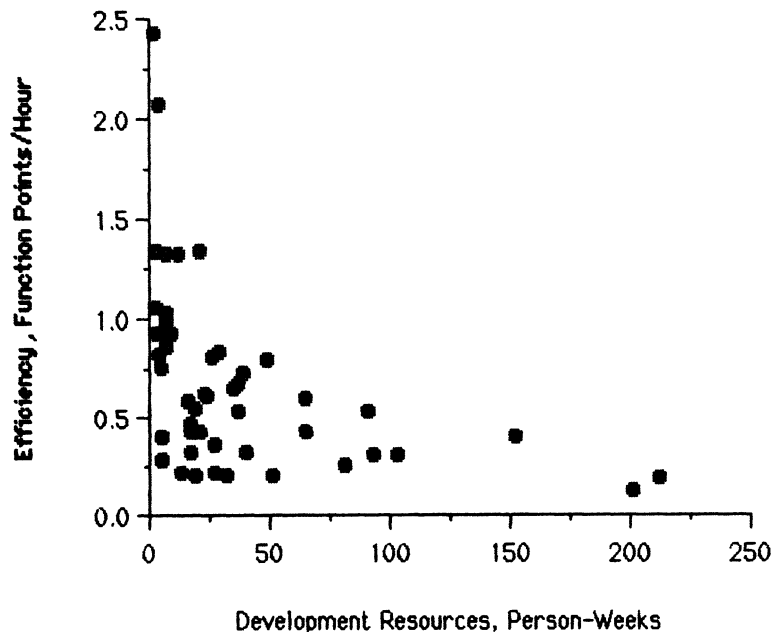
efficiency suggests that a better measure of experience is needed, possibly not limited to Factory and Builder experience alone, but including facility with programming, data modeling and VMS. Finally, no survey data measured the effect of methodology on efficiency and in particular whether development teams that followed a rapid prototyping methodology were more efficient than teams using the Factory in a more traditional manner. Although there was a negative effect of team size on efficiency, this correlation was small and not significant.

5. ESTIMATING RUN TIME RESOURCES

What are the attributes of the run time environment that result in acceptable run time performance? The first step was to investigate the relation between response time and four independent variables: application burden, cpu size, performance related development activity and function points. Respondent were asked to characterize average response time on a typical day, putting the estimate into one of three qualitative categories. Application burden was measured in terms of other simultaneously running applications and the respondent's qualitative estimate of the degree of utilization of the CPU. CPUs, from microVAXs to 8650s, were grouped in six categories roughly

Figure 3:

Relationship Between Development Efficiency and Project Size



according to their processing speed. Performance related development activity was taken as a surrogate for performance related concerns at the site generally and thus the assumed ability to obtain optimal performance from the machine resource. The four independent parameters together resulted in a multiple correlation coefficient of 0.47 which, with 29 cases, yielded an F-statistic, F(4,24), of 1.7 which was not significant. Although each of the independent variables had the expected sign, their individual correlations ranged from 0.07 to 0.24 and only one was significant - application function points at the 95% level. The fact that survey respondent did not have access to quantitative operating system performance measures that could be compared consistently across the sample rendered this approach of little use.

But respondent did know about their machine CPU model and its memory size so these were investigated to see if a predictor of CPU model or memory size could be developed from a knowledge of a single application's run time environment. This environment was expressed in terms of simultaneous users ("terminals"), transaction volume, and application burden (explained above). Multiple stepwise regressions were performed on the survey data to determine the best statistical model.

The only significant predictor of CPU type is

number of terminals, with a correlation coefficient of 0.31 and an F-statistic, F(1,34), of 3.5 which is significant at the 93% level. This relation, indicated by the scatter plot in Figure 4, is dominated by the single application with 180 users on a large machine and was rejected for this reason. Although a favorite question asked by DEC sales persons of the Application Factory Product Manager is "what size of CPU will be needed to run an application developed with the Factory", no significant relationship between application function points and CPU size was found in the survey sample. Indeed the sample includes all sizes of applications running on all types of CPUs. Knowledge of a single application alone is not sufficient to size CPU resources.

In contrast of CPU type, the relation between CPU memory ("megabytes") and terminals was found to be strong and highly significant with a correlation coefficient of 0.79 and an F-statistic, F(2,37), of 60, making it significant at greater than 99%. The best fit linear model is:

$$\text{Megabytes} = 8.73 + 0.60 * \text{number of terminals.}$$

This relation is indicated by the scatter plot in Figure 5.

Although the statistics relating megabytes and terminals are impressive, they still appear to be dominated by a single application. The fact that

Figure 4.

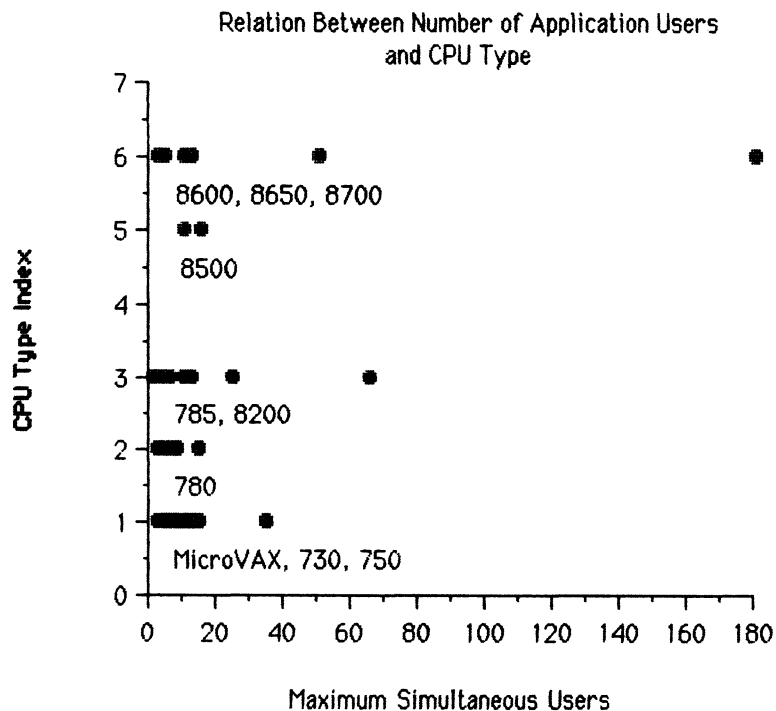


Figure 5:

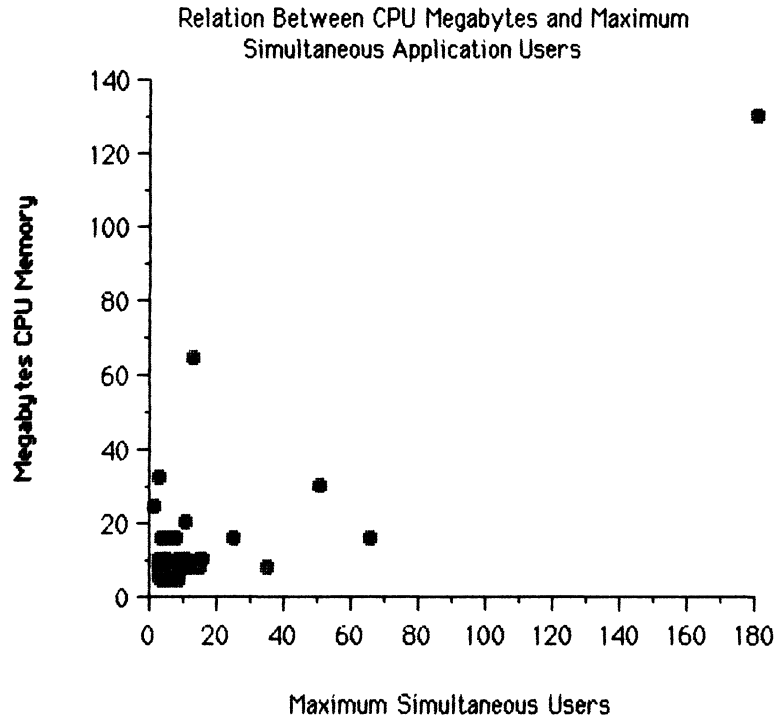


TABLE 2: Application Features and Run Time Environments

<u>Application Name</u>	Application Function Points	CPU Type	Megabytes CPU Memory	Number Simultaneous Users	Transaction Volume Score**	Largest File Size, Records
Apparel Sales Marketing	810	750	6	5		45,000
Inventory-Finished Goods	1125	750	10	14		12,000
Std Cost of Products Estimator	120	785	24	1	1	2,000
Inventory Management and Processing	931	780	8	3	1	10,000
Accounting and Quotation System	1221	8200		1	1	3,000
P/O Tracking System	101	750		2	1	1,400
Bank of (_____) Bankline*	140	780	10	8	1	200,000
Tracking Telex & Telefax	55	750	10	4	1	6,000
(_____) Revenue System*	1107	750	5	2	1	
Demand Billing	105	780	8	2	1	250
Stores Inventory System	63	780	8	5	1	
Deposit System	43	780	8	6	1	400
Maintenance Tracking	527	780	10	14	1	6,000
Wholesale Importer, 4 Applications	560	750		4	1	40,000
Crude Oil Contract/Shipping/Receiving	341	750	4	8	1	12,000
Supply Control	246	750	8	2	1	2,000
Administration System	325	780	16	7	1	15,000
Cortex Corporate Information System	976	750	6	5	1	4,000
Hotline & Bug Reporting	425	750	8	6	1	10,000
Application Factory User Interface	1298	750	8	12	1	2,000
Medical Office Mgt- BC/BS	480	MVAX	4	3	1	5,000
Kevlar Management Information System	2419	8600	30	50	4	
Data Structure Modeler	204	8600	16	4	1	1,000
Finished Product Specification System	203	8650	64	12	1	20,400
Spinning Area Management System	215	MVAX	4	5	1	
Bulk Continuous Fiber Inspect & Pack	1090	785	16	24	4	3,000,000
Bulk Continuous Fiber Scales	243	750	8	14	4	540,000
Estate Donation Probate	802	8500	20	10	1	1,500
Maintenance Work Order Control	269	785		10	1	
Raw Materials Inventory	389	785	16	5	1	900
Sales Order Processing System	1504	750	8	34		
Sales Tracking	751	785	16	6	1	1,000
Product History	393	8600	64	12	1	90,000
Training Registration	196	8600		4	1	300
Project Forecasting	238	785	10	2	1	1,000
Installment Loan Tracking	98	750	8	2	1	2,000
Costing System	344	785	16	65	4	8,500
Beverage Lab Analysis and Reporting	1511	MVAX	9	8	2	1,000,000
Formula Tracking System	249	8600	10	10	1	500
Customer Information System	889	8500	10	15	2	8,000
Inventory-Raw Stock	292	8650	32	2	1	1,000
Telephone Communication Control	578	8600	16	3	1	7,000
Customer Service Database	1910	8600	130	180	4	300,000
Petroleum Products Wholesale Pricing	1595	785	8	12	4	
Oil Market Analysis & Research	367	785	8	10	2	10,000
Animal M/S	1058	MVAX	9	7	1	50,000

* Customer names omitted for proprietary reasons

** 1= 0 to 5,000; 2 = 5,000 to 10,000; 4 = 10,000 to 100,000

resources. Much more must be known about the number of users and the use intensities of other processes. Although the qualitative data exhibited the correct trends, correlations were often too weak and insignificant to form predictive models. CPU memory was found to be highly correlated with number of application users, but this appeared to be dominated by the largest application with 180 users and 128 megabytes of memory. A table describing the application characteristics and their run time environmental parameters is offered so users can derive a qualitative impression of whether they are within the workable range of other users.

7. REFERENCES

1. Albrecht, Allan J., "Measuring Development Productivity", Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium, October, 1979, Pages 83-92.
2. Basili, Victor R., ed. Models and Metrics for Software Management and Engineering. IEEE Institute, New York, 1980.
3. Drummond, Steve, "Measuring Applications Development Performance", Datamation, February, 1984.
4. Picardi, Anthony C., "Productivity Increases With The Cortex Application Factory: Empirical Survey Results", Proceedings of the DECUS Northeast Regional Conference, Boston, Mass, June 5-6, 1986.
5. Zwanzig, Ken, ed. "Handbook for Estimating Using Function Points", GUIDE Project DP-1234, November, 1984.

8. Appendix A: Function Point Estimating Worksheet

Application Name: _____

1. Application Size Estimates (enter number and multiply by given weights)

1.1 Number of Screens	= _____ X 4 = _____
1.2 Number of Reports	= _____ X 4 = _____
1.3 Procedures called from menus	= _____ X 4 = _____
1.4 Number of Key Screens	= _____ X 4 = _____
1.5 Number of Datasets	= _____ X 7 = _____
1.6 Interfaces to HW/SW systems	= _____ X 5 = _____
1.7 Sum of 1.1 to 1.6 = Unadjusted Function Points	= _____
2. Application Difficulty Factors
 - 2.1 Telecommunications (enter indicated score if used)

VAXMAIL (2)	= _____
DECNET (2)	= _____
VMS Broadcast (2)	= _____
Sum of 2.1 =	_____
 - 2.2 Two or more interacting CPUs (enter indicated score if used)

Yes, synchronous, well defined (2)	= _____
Yes, asynchronous, interrupts (2)	= _____
Used on a VAX Cluster (2)	= _____
Sum of 2.2 =	_____
 - 2.3 Performance-related changes (Enter indicated score if done)

FDL tuning (1)	= _____
Performance data/dataview design (2)	= _____
Custom procedures &/or Jobstreams (2)	= _____
Response-time Requirements? (1)	= _____
Sum of 2.3 =	_____

- 2.4 Other simultaneous applications (Enter indicated score if applicable)
- Word processing (2) = _____
- DBMS queries (2) = _____
- Few other applications (<1/2 CPU) (0) = _____
- Many other applications (>1/2 CPU) (2) = _____
- Sum of 2.4 = _____
- 2.5 Transaction volume, records/day (Choose one, enter indicated score)
- 0 to 5,000 (1) = _____
- 5,001 to 10,000 (2) = _____
- 10,001 to 100,000 (4) = _____
- 100,001 and over (6) = _____
- Score for 2.5 = _____
- 2.6 Degree of Customization
- Percent of screens customized (Choose and enter indicated score)
- 0% to 10% (0) = _____
- 11% to 50% (1) = _____
- 51% to 100% (2) = _____
- Percent of reports customized (Choose and enter indicated score)
- 0% to 10% (0) = _____
- 11% to 50% (1) = _____
- 51% to 100% (2) = _____
- Percent of total exe code customized (Choose and enter indicated score)
- 0% to 1% (0) = _____
- 1% to 5% (1) = _____
- Greater than 5% (2) = _____
- Sum of 2.6 = _____
- 2.7 Process state monitor coded (Choose and enter indicated score)
- <20% of processes, <6 states (3) = _____
- Most processes, >6 states (6) = _____
- Score for 2.7 = _____
- 2.8 Specification/Program Re-use (Choose and enter indicated score)
- Automatic documentation only (3) = _____
- Auto doc plus shared data specifications (4) = _____
- Auto doc plus shared process specifications (4) = _____
- Auto doc and all specifications shared (6) = _____
- Score for 2.8 = _____
- 2.9 Used by/sold to third parties? (If yes enter 5)
- Score for 2.9 = _____
- 2.10 Application Custom Features (Enter 1 for each that applies)
- Audit trail (1) = _____
- Append form tape/disk (1) = _____
- Purge/archive (1) = _____
- Automatic startup/shutdown (1) = _____
- Special navigation (1) = _____
- Security (1) = _____
- Sum for 2.10 = _____
- 2.11 Installed at multiple sites (Choose and enter indicated score)
- Two or three sites (3) = _____
- >3 sites (5) = _____
- Score for 2.11 = _____
- 2.12 Designed for later modifications (Choose and enter indicated score)
- Yes, < 30% functionality change (4) = _____
- Yes, >30% functionality change (5) = _____
- Score for 2.12 = _____

3. Calculate adjustment multiplier.

3.1 Add sums and scores for items 2.1 to 2.12 = _____

3.2 Adjustment Multiplier = $.65 + (.01 * (6 + \text{Sum from 3.1})) = \underline{\hspace{2cm}}$

4. Calculation of Function Points.

4.1 Function points = Unadjusted Function Points * Adjustment Multiplier
= Sum from 1.7 above * Multiplier from 3.2 above
= _____

4.3 Lines of COBOL-equivalent code = Function points from 4.1 * 105
= _____

5. Calculation of Development Resources

5.1 Application Factory Person -weeks = $-1.58 + 0.064 * \text{Function Points}$

5.2 COBOL Person -weeks = $(\exp(\ln(\text{Function Points}) - \ln(34)) / 0.59) * 130 / 40$

Improving Technical Manuals Through Reader Analysis

Thomas L. Warren
Department of English
Oklahoma State University
Stillwater, OK 74078-0135

ABSTRACT

Communication transfers information from one who has it to one who needs it and can understand it. Whether reader involvement is casual or interactive, writers must prepare documents based on three points: (1) What does my reader need to know? (2) How can I help my reader understand? (3) What is my reader going to do with the material? Far too often, documentation writers overlook the second and third, while occasionally overlooking (or misjudging) the first. In addition to knowing the reader's needs and abilities, the writer must know whether the reader will access the information sequentially or randomly. This paper, then, focuses on these three points and presents a systematic approach writers can use to prepare more effective, reader-based documentation.

Wendell Johnson, in a 1953 article (1), uses an interesting term to describe communication: The *fateful* process of Mr. A talking to Mr. B. Certainly communication is process, but fateful? Can there be any hope that information can flow from one who has it to one who needs it if that process is fateful? Consider the average conversation. Mr. A chats with Mr. B. Both have other things on their minds: What to say next, what did he just say, what's the solution to this nagging problem, have I paid the gas bill, and so on. A torrent of words issue from the

mouth (or pen) charged with the fateful task of conveying information. Whether they actually do that or not, is another matter.

That manuals are the last thing used to solve problems comes as no surprise to any one who reads them. Certainly, we all have had "fateful" encounters with manuals. Recently, for example, I got a data management program for my office computer. I followed the install routine and turned to the tutorial. There, in line 18 of page 1, I was told to put a specific disk in drive A. This I did. I was then to type a start

command, which I did. Instead of the opening screen, I got an error message. Because I was using the backup copy of the program (as per instructions), I got the original and tried again. The same error message appeared. I then tried another of the original disks and it worked. There was a major error on that first page. Had I not inferred that the real error was in the manual, I would still be trying to run the program.

READERS AND WRITERS

My topic, however, is not debugging manuals or running through a catalog of war stories on errors in manuals. Rather, my topic is a much more important part of communication, focusing instead on the problem of writers failing to know their readers.

Every definition of communication I know includes at least four elements: a sender, a message, a medium for sending the message, and a receiver. If we kept all our communications locked in our heads or could demand that our readers be as knowledgeable as we are about the subject, we would never have fateful breakdowns in communication. But we shouldn't communicate to ourselves or our clones or even pretend to because communication involves the flow of information from one who has it to one who needs it, so manuals could become superfluous for those who know the subject. Because most people aren't knowledgeable, manuals exist. And because the people who write them know more than those who read them, those writers must understand their readers so they can transfer the information. I seriously doubt that any writers totally ignore their readers. In selecting one word over another because it more accurately conveys a meaning, writers show concern for readers. Actually, reader analysis is more complex than deciding if the reader knows the term. Writers may approach that analysis from several angles. Three are

1. A psychological profile
2. A systems analysis profile
3. An informational need profile

Psychological

The psychological profile (best discussed by Pearsall [2]) relies on the writer understanding some demographics about the reader (assumed education, for example) and then constructing a profile based on that information. This analysis actually centers on how well the individual could process text. To look at one example, Pearsall shows that the writing style one adopts for the lay readers (who are outside their field of specialization) has a very high percentage of non-complex sentences (i.e., a high percentage of subject-verb-object constructions). Average lengths are about 15 words per sentence and 50 words per paragraph. The writer assumes that because the reader is not sophisticated in the subject, he or she will have to focus most of the information processing activity on understanding vocabulary and grasping relationships among the words. On the other hand, the expert (one with advanced degrees or many years of experience) requires no such considerations. Sentences are longer as are paragraphs, because the processing time is much reduced. The writer develops a total profile by using the other eleven categories of analysis for the assumed reader. The psychological profile, however, presents the assumed reader outside a context.

Systems

A systems analysis profile requires that the writer be aware of how the individual relates to the rest of the organization. Perhaps more useful when writing reports rather than when writing training manuals, the approach nonetheless has some value for such documents. Knowing the positioning of the individual within the organization suggests a reason for reading. Is the person an executive in upper management? Then the reason for reading the document can be greatly different than if the person is a clerk

needing to learn an application. Such differences certainly call for not only different styles of writing and content, but also organization of the material—including page layout and design*.

Informational

The third method, based on information need, combines elements from both. It begins with the writer asking how much information the reader needs. Certainly, the history of the computer is not necessary if the reader wants to learn a word-processing program. The writer then decides how to write the text to best help that reader understand. The reader must be able to understand the information for it to be useful. Finally, knowing what the reader is to do with the material influences the writing, the organizing, and the layout and design. Materials such as DEC's *Personal Computer: Documentor's Guide*(3) emphasize reader analysis. They tell us to know the reader (DEC's *Guide* names three: novice, someone *familiar* with another operating system [but not yours], and someone *familiar* with a previous version of your product [emphasis added]). What this advice suggests, in effect, is know what your reader already knows and build on that (see Appendix A for a table summarizing the *Guide's* advice on reader analysis). Such advice certainly is valuable because it forces the writer to control the number of inferences the reader must make. But the advice doesn't go far enough. And that's the purpose of this paper: To provide writers with a 3-step process to help them analyze their readers:

1. What does my reader need to know?
2. How can I help my reader understand?
3. What will my reader do with the

information?

1. What Does my Reader Need to Know?

Writers who follow DEC's *Guide* have a fairly good idea of what the reader already knows. The *Guide's* classification system

makes the writer think about the technical sophistication of the reader *relative to the topic*. It suggests indirectly that the writer may make other assumption (such as vocabulary level, sentence length, etc.) about how well the reader can process information. Notice that these assumptions as well evolve from the reader's familiarity with the program. If the writer is working on a reference section of a wordprocessing manual and wants to explain how to merge a mailing list with a document, that writer analyzes the reader by deciding what level of knowledge the reader already has about the command and what he or she needs to know.

If, however, the writer omits a step that is to him or her obvious (as I have found in a merge explanation in a word-processing manual), the reader/operator can become extremely frustrated. The crucial step may have been omitted because the writer forgot, or because the writer assumed that the reader would know what to do. In either case, the reader faces the same situation I faced with the data management tutorial.

A major problem in communicating is for the person who has the information to clearly perceive what the reader needs to know. Just as the writer causes the communication breakdown when assuming too much knowledge, so too could there be problems if the writer assumed too little knowledge. Take an extreme case: The mail merge program assumes that the reader has the computer on and running the program. The writer also assumes that the reader knows where certain keys are and how to use them ("Select," for example). To include information on either of these would frustrate the reader's attempt to find *relevant* information. Erroneous assumptions about the reader's level of computer knowledge could result in too much or too little information. A fateful communication would then occur, although the results of assuming too little knowledge are not as fateful as assuming too much. Guideline one, then, is What does the reader need to know?

*See also my papers in DECUS Fall, 1985 and Spring, 1986 *Proceedings*.

The second guideline relates to helping the reader understand what the writer has written.

2. How can I Help my Reader Understand?

DEC's *Personal Computer: Documentor's Guide* instructs writers to be concerned about the language they use (see Appendix A below). That language will range from concrete, with short sentences and monosyllabic words, to abstractions, long sentences, and, presumably, polysyllabic words. Good advice in so far as it goes and in so far as we assume that length of sentence and word directly relates to reader understanding. But that is a poor assumption to make. Not many people know what *quark* means in quantum physics, yet it is a relatively short word. *Television*, a long word by comparison, is very well-known. Word length is really no measure of difficulty.

Sentence length is likewise identified as being related to the reader's sophistication. Joseph Williams, in his book *Style: Ten Lessons in Clarity and Grace*, presents two samples of long sentences:

1. We have to distinguish two kinds of long sentences; the one you're reading right now, for example, is rather long, sixty-four words to be exact, but it's long simply because I have chosen to punctuate what might be a series of shorter sentences as one long sentence; those semicolons could have been periods—and that dash could have been one too.

2. I can write a different kind of sentence just as long as that but one that doesn't let me trade a comma, semicolon, or dash for a period, because it is composed of several subordinate parts, all depending on a single main clause—a sentence such as the one you are now reading which is also exactly sixty-four words long. (4)

Both are the same length, but the one (#1) is much harder to read than the other because of the way Williams wrote it. Certainly, in a series of instructions on merging a mailing list, the writer is not going to use 64 word sentences. Even in the "Getting Started" section, writers are going to be more aware of length. But my point is that length is really a poor measure of the reader's ability to understand a word or sentence.

I have discussed these two points at length because they are central to the numerous readability formulas that are available for analyzing text. I have already questioned their assumptions about length as a factor in difficulty, and I think that writers need to use other matters in analyzing their readers and helping them understand the information. (5) Advice to writers that focuses on word and sentence length suggests that one need change length to improve readability. While of some value, these measures must be only one of many tools writers use in analyzing text to produce information the reader can easily process.

Understanding information is a complex process. The mind processes the information using memory to develop inferences:

1. Information that is in the passage (such as supplying antecedents for pronouns, or remembering previous steps), and
2. Information outside the context of the passage (information the reader brings to the reading).

When the mind must make numerous inferences or seek information outside (i.e., turning backwards and forwards in a manual looking words up in a dictionary, etc.), the processing slows down and the reader quickly tires of the effort. Consider the following sentence:

No command will be sent by the computer until it has been checked.

The question is who or what needs checking? The command? The computer? The reader must draw an inference. While context and common sense might help, the reader still must work. So, what the writer assumes about the level of sophistication of the reader does make a difference, not only in technical sophistication, but also sophistication in drawing inferences.

These sophistications, however, are not as interrelated as the Guide assumes. A reader may be familiar with an older version of the program, yet have problems processing complex concepts, sentences, or paragraphs. By the same token, a novice may be insulted if forced to read "See Dick Run. See Jane run" sentences. Certainly, writers must make some assumptions about the reader's ability to process information or all manuals would be primer books. The writer, therefore, must know which techniques to use in order to help the reader understand.

While the first guideline reminds the writer of how much content to put in, this second one relates to assumptions about the reader's sophistication. The third guideline has the writer determine what the reader will do with the material.

3. What will my Reader do with the Material?

Readers read for many purposes. Technical manuals provide general information about systems and programs, as well as specific information on how they work and how to use them. Manuals, by definition, are sets of instructions for performing some action. Therefore, elements of successful instructions are important: white space, small steps, appropriate definitions, visuals, and so on. Yet, there are really two major kinds of instructions: what we can call the armchair variety and what we call the workbench variety (a distinction Cunningham makes, [6]).

Armchair Instructions: Armchair instructions are those meant to be read and absorbed into memory so that the reader can go some place and perform them. For example, when you read a set of instructions on controlling the slice of a golf ball, you need

to be able to remember them so that when you are at the golf course, you can perform better. You don't take the instructions with you. Writers must prepare armchair instructions so that they are easily remembered when the reader is away from them.

Workbench Instructions: Workbench instructions, on the other hand, are meant to be read and used on the spot. A manual that tells technicians how to trace a signal problem through a circuit board is used at the spot where they need the material. The demands on memory, consequently, are considerably less. The writer must accommodate readers who shift their gaze from manual to board and back again.

Operators are similar. They must be able to read an instruction, perform the action, and return to the page for the next step. If the writer assumes that readers can hold two or more steps in mind while performing the action and writes accordingly, problems could easily ensue with this demand on memory. Because of the job demands, operators read to perform at two levels.

At one level, operators want to read instructions that will allow them to perform an isolated action or series of actions ("merge", for example). Infrequent use of this material means operators will not need to remember it. On the other hand, a command such as "insert" is one operators will use frequently and will need to learn. The approach to these two writing problems decidedly influences how the writer prepares the text.

What that means involves not only the actual writing, but also layout and design. If operators are going to make immediate use of the material, the writer might be able to ask them to hold a little more in memory than if they were going to memorize or otherwise learn the material. Likewise, if the writer can assume that operators know a particular key sequence ("Insert," for example), the instruction might be a little more complex than if they did not know the sequence. The individual instruction could then need special

attention, being laid out in such a way as to make it easier for operators to locate in case they needed to return to the page. This means using white space or even subpoints; grouping instructions and visuals into chunks is another way.

When operators must learn the material, then small steps, plus the usual white space and visuals are an approach the writer could use. Learning through repetition, as most manuals approach the problem, is another way. Likewise, the writer can link the known with the known, allowing them to relate the new material to the old. One manual, for example, describes moving a block of text in terms of a knife, cutting, and pasting—all activities familiar to most people. (7) So, knowing what the reader will do with the material influences how the writer prepares the material.

CONCLUSION

My point in this paper has been to suggest that while manuals for writers such as those from DEC stress that the writer understand and know the reader, they don't go far enough. Too often the writer understands the reader's sophistication based on how much the reader knows about the program at hand. The result is that the material is confusing and unclear, leading to misunderstandings and certainly additional evidence that the manual should be the last thing read.

Writers should therefore consider three questions as they prepare to write the material and while writing it:

1. What does my reader need to know? (The details)
2. How can I help my reader understand? (Definition, word/sentence/paragraph length, etc.)
3. What is my reader going to do with the material? (Learn, apply immediately)

REFERENCES

1. Johnson, Wendall. "The Fateful Process of Mr. A Talking to Mr. B," *Harvard Business Review*, 31, No. 1 (1953), 49-56.
2. Pearsall, Thomas E. "Introduction," *Audience Analysis for Technical Writing*. Beverly Hills, CA: Glencoe, 1969.
3. Digital Equipment Corporation. *Personal Computer: Documentor's Guide*. Marlboro, MA: DEC, 1983.
4. Williams, Joseph M. *Style: Ten Lessons in Clarity and Grace*, 2nd ed. Glenview, IL: Scott, Foresman and Company, 1985. p. 112.
5. Warren, Thomas L. "Putting Readers Back in Manuals: Computer Manuals and the Problems of Readability." Marlboro, MA: Digital Equipment Corporation Users Society *Proceedings*, Fall, 1985, pp. 457-482. Warren, Thomas L. "Putting Readers Back in Manuals: Computer Manuals and the Problems of Readability—V.2.0." Marlboro, MA: Digital Equipment Corporation Users Society *Proceedings*, Spring, 1987, pp. 387-396.
6. Cunningham, Donald H. *Personal conversation*, June, 1981.
7. Microsystems Engineering Corporation. *MASS-11 Reference Manual: WS-200 Editor*. Hoffman Estates, IL: Microsystems Engineering Corporation, 1984, p. 6-9.

Proceedings of the Digital Equipment Computer Users Society, Marlboro, MA 01752-1850