EY-2279E-SG-0002

VAX/VMS INTERNALS I

Student Workbook

Prepared by Educational Services
of
Digital Equipment Corporation

Second Edition, October 1986

Printed in U.S.A.

The manuscript for this book was created using DIGITAL Standard Runoff. Book production was done by Educational Services Development and Publishing in Nashua, NH.

CONTENTS

2 THE PROCESS

3 SYSTEM MECHANISMS

4 DEBUGGING TOOLS

5 SCHEDULING

6 PROCESS CREATION AND DELETION

7 SYSTEM INITIALIZATION AND SHUTDOWN

8 USING THE LINKER

FIGURES

TABLES

EXAMPLES

Student Guide

INTRODUCTION

The VAX/VMS Operating System Internals course is intended for the student who requires an extensive understanding of the components, structures, and mechanisms contained in the VAX/VMS operating system. It is also an aid for the student who will go on to examine and analyze VAX/VMS source code.

This course provides a discussion of the interrelationships among the logic or code, the system data structures, and the communication/synchronization techniques used in major sections of the operating system.

Technical background for selected system management and application programmer topics is also provided. Examples of this information include:

- The implications of altering selected system parameter values
- The implications of granting privileges, quotas, and priorities
- How selected system services perform requested actions.

Information is provided to assist in subsequent system-related activities such as:

- Writing privileged utilities or programs that access protected data structures
- Using system tools (for example, the system map, the system dump analyzer, and the MONITOR program) to examine a running system or a system crash.

This course concentrates on the software components included in (and the data structures defined by) the linked system image. Associated system processes, utilities, and other programs are discussed in much less detail.

GOALS

- Describe the contents, use, and interrelationship of selected VAX/VMS components (job controller, ancillary control processes, symbionts), data structures (SCB, PCB, JIB, PHD, P1 space), and mechanisms (synchronization techniques, change mode dispatching, exceptions and interrupts).

- Describe and differentiate system context and process context.

- Discuss programming considerations and system management alternatives in such problems as:

  - Assigning priorities in a multiprocess application
  - Controlling paging and swapping behavior for a process or an entire system
  - Writing and installing a site-specific system service

- Use system-supplied debugging tools and utilities (for example, SDA, XDELTA) to examine crash dumps and to observe a running system.

- Describe the data structures and software components involved when a process is created or deleted, an image is activated and rundown, and the operating system is initialized.

- Describe how the following interrupt service routines are implemented:

  - AST delivery
  - Scheduling
  - Hardware clock
  - Software timers

- Briefly describe the components of the I/O system, including system services, RMS, device drivers and XQPs.

- Briefly describe how RMS processes I/O requests, including the user-specified and internal data structures involved.

- Describe certain additional VMS mechanisms used on a VAX system in a cluster (for example, synchronization and communication mechanisms).

NON-GOALS

- Writing device drivers (see the VAX/VMS Device Driver course)
- Writing ancillary control processes, ACPs (see the VAX/VMS Device Driver course)
- Comprehensive understanding of RMS internals
- DECnet internals (see the DECnet courses)
- Layered product internals
- Command language interpreter internals
- System management of a VAXcluster

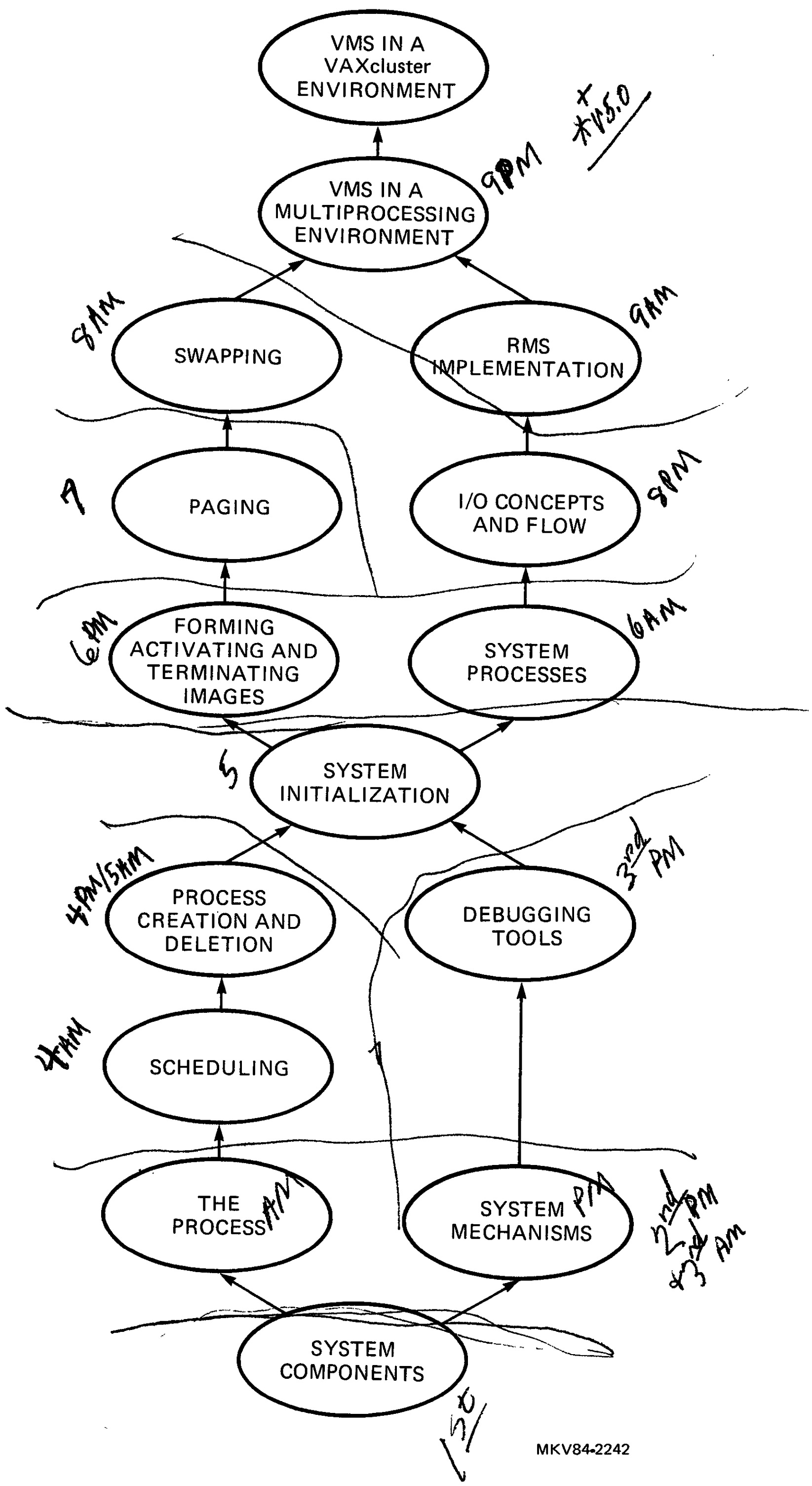
PREREQUISITES

- Ability to program in at least one VAX native language. This may be obtained through language programming experience and completion of an appropriate language programming course (for example, Assembly Language Programming in VAX-11 MACRO). In addition, completion of the Introduction to VAX-11 Concepts course is recommended.
- Ability to read and comprehend programs written in VAX-11 MACRO is required. In addition, ability to program in VAX-11 MACRO or BLISS is recommended.
- Completion of one of the Utilizing VMS Features courses.

RESOURCES

1. VAX/VMS Internals and Data Structures
2. VAX/VMS System Dump Analyzer Reference Manual
3. VMS Internals I and II Source Listings

COURSE MAP


VMS IN A VAXcluster ENVIRONMENT
VMS IN A MULTIPROCESSING ENVIRONMENT
SWAPPING
RMS IMPLEMENTATION
PAGING
I/O CONCEPTS AND FLOW
FORMING ACTIVATING AND TERMINATING IMAGES
SYSTEM PROCESSES
SYSTEM INITIALIZATION
PROCESS CREATION AND DELETION
DEBUGGING TOOLS
SCHEDULING
THE PROCESS
SYSTEM MECHANISMS
SYSTEM COMPONENTS
MKV84-2242

COURSE OUTLINE

I. System Components

A. How VMS Implements the Functions of an Operating System

B. How and When Operating System Code is Invoked

C. Interrupts and Priority Levels

D. Location of Code and Data in Virtual Address Space

E. Examples of Flows for:

1. Hardware clock interrupt
2. System event completion
3. Page fault
4. RMS request for I/O
5. $QIO request for I/O

F. Examples of System Processes

1. Operator Communication (OPCOM)
2. Error logger (ERRFMT)
3. Job controller (JOB_CONTROL)
4. Symbionts (SYMBIONT_n)

G. Software Components of DECnet-VAX

II. The Process

A. Process vs. System Context

B. Process Data Structures Overview

1. Software context information
2. Hardware context information

C. Virtual Address Space Overview

1. S0 space (operating system code and data)
2. P0 space (user image code and data)
3. P1 space (command language interpreter, process data)

D. SYSGEN Parameters Related to Process Characteristics

III. System Mechanisms

A. Hardware Register and Instruction Set Support

B. Synchronizing System Events

1. Hardware Interrupts
2. Software Interrupts
   Example: Fork Processing
3. Requesting Interrupts
4. Changing IPL
5. The Timer Queue and System Clocks

C. Process Synchronization Mechanisms

1. Mutual Exclusion Semaphores (MUTEXes)
2. Asynchronous System Traps (ASTs)
3. VAX/VMS Lock Manager

D. Exceptions and Condition Handling

E. Executing Protected Code

1. Change Mode Dispatching
2. System Service Dispatching

F. Miscellaneous Mechanisms

1. System and Process Dynamic Memory (Pool)

G. SYSGEN Parameters Controlling System Resources

IV. Debugging Tools

A. VAX/VMS Debugging Tools

B. The System Dump Analyzer (SDA)

1. Uses
2. Requirements
3. Commands

C. The System Map File

D. Crash Dumps and Bugchecks

1. How bugchecks are generated
2. Sample stacks after bugchecks
3. Sample crash dump analysis

E. The DELTA and XDELTA Debuggers

V. Scheduling

A. Process States

1. What they are (current, computable, wait)
2. How they are defined
3. How they are related

B. How Process States are Implemented in Data Structures

1. Queues
2. Process data structures

C. The Scheduler (SCHED.MAR)

D. Boosting Software Priority of Normal Processes

E. Operating System Code that Implements Process State Changes

1. Context switch (SCHED.MAR)
2. Result of system event (RSE.MAR)

F. Steps at Quantum End

1. Automatic working set adjustment

G. Software Priority Levels of System Processes

VI. Process Creation and Deletion

A. Process Creation

1. Roles of operating system programs
2. Creation of process data structures

B. Types of Processes

C. Initiating Jobs

1. Interactive
2. Batch

D. Process Deletion

E. SYSGEN Parameters Relating to Process Creation and Deletion

VII. System Initialization and Shutdown

A. System Initialization Sequence

B. Function of initialization programs

C. How memory is structured and loaded

D. Start-up command procedures

E. How hardware differences between CPUs affect initialization

F. Shutdown procedures and their functions

G. Auto-restart sequence

H. Power-fail recovery

VIII. System Processes

A. For selected VAX/VMS processes:

1. Job controller
2. Symbionts
3. Error Logger
4. OPCOM

We will be describing their:

1. Primary Functions
2. Implementation
3. Methods of communication with other VMS components
4. Basic internal structure (on a module basis)

IX. Forming, Activating and Terminating Images

A. Forming an Image

1. PSECTs in source/object modules
2. Format and use of the image header

B. Image Activation and Start-Up

1. Mapping virtual address space
2. Overview of related data structures
3. Image start-up (SYS$IMGSTA)
4. Installing Known Files

C. Image Exit and Rundown

1. $EXIT system service
2. Termination Handlers
3. DCL Sequence

D. SYSGEN parameters relating to image formation, activation and termination

X. Paging

A. Basic Virtual Addressing

1. Virtual and physical memory
2. Page table mapping

B. Overview of Page Fault Handling

1. Resolving page faults
2. Data structures in the process header

C. More on Paging

1. Free and modified page lists
2. The paging file
3. Cataloging pageable memory (the PFN database)

D. Global Paging Data Structures

E. Summary of the Pager

XI. Swapping

A. Comparison of Paging and Swapping

B. Overview of the Swapper, the System-Wide Memory Manager

C. Maintaining the Free Page Count

1. Write Modified Pages
2. Shrink Working Sets
3. Outswap Processes

D. Waking the System-Wide Memory Manager

E. Outswapping a Process

1. Swap files
2. Scatter/Gather
3. Partial Outswaps

F. Inswapping a Process

XII. I/O Concepts and Flow

A. Overview of I/O components and flow

B. Components of I/O system

1. RMS
2. I/O system services
3. XQPs, ACPs
4. Device drivers

C. The I/O database

1. Driver tables
2. IRPs
3. Control blocks

D. Methods of data transfer

XIII. RMS Implementation and Structure

A. User-specified data structures (FABs, RABs, and so on)

B. RMS Internal Data Structures

1. Process I/O Control Page (for example, default values, I/O segment area)
2. File-Oriented and Record-Oriented Data Structures (IFAB, IRAB, BufDescBlk, I/O Buffer)

C. RMS Processing

1. RMS Dispatching
2. RMS routines and data structures
3. Examples of flows of some common operations

XIV. VMS in a Multiprocessing Environment

A. Loosely coupled processors

B. Tightly coupled processors (11/782)

1. MP.EXE structures
2. Scheduling differences
3. Startup /shutdown

C. Clustered processors

XV. VMS in a VAXcluster Environment

A. Cluster synchronization and communication mechanisms

1. Distributed lock manager
2. Distributed job controller
3. Interprocessor communication

B. System initialization and shutdown differences

1. VMB, INIT and SYSINIT differences
2. Joining a cluster
3. Leaving a cluster

C. SYSGEN parameters relevant to the VAXcluster environment

D. Relevant system operations

System Components

INTRODUCTION

This module introduces the major software components supplied in or with the VAX/VMS operating system. As an overview of the operating structure, it gives a review of facilities introduced in previous VAX/VMS courses. New terms and logic components are introduced, but detailed discussion of them is generally deferred until later modules of this course.

This module does not provide a complete catalog of all facilities, modules, and programs in the operating system. It provides an understanding of the relationships and coordination among the various software components.

Software components can be classified by several attributes, including:

- Implementation form (service routine, procedure, image, or process)
- "Closeness" to the linked system image (part of SYS.EXE, linked with system symbol table, privileged known image, and so forth)
- Access mode (kernel, executive, supervisor, or user)
- Address region (program, control or system)
- Memory-resident characteristics (paged, swapped or shared)

OBJECTIVES

For each selected VAX/VMS software component, briefly describe:

1. Its primary function
2. Its implementation (process, service routine, or procedure; in which address region it resides; what access modes it uses)
3. The method or methods by which it accomplishes communication

RESOURCES

Reading

- VAX/VMS Internals and Data Structures, System Overview

Additional Suggested Reading

- VAX/VMS Internals and Data Structures, Chapters on I/O System Services, Interactive and Batch Jobs, and Miscellaneous System Services.

Source Modules

Facility Name

SYS
DCL,CLIUTL
DEBUG
RTL
RMS
F11A,F11X,MTAACP
REM,NETACP
JOBCTL,INPSMB,PRTSMB
OPCOM
ERRFMT

TOPICS

I. How VMS Implements the Functions of an Operating System

II. How and When Operating System Code Is Invoked

III. Interrupts and Priority Levels

IV. Location of Code and Data in Virtual Address Space

V. Examples of Flows for:

   A. Hardware clock interrupt

   B. System event completion

   C. Page fault

   D. RMS request for I/O

   E. $QIO request for I/O

VI. Examples of System Processes

   A. Operator Communication (OPCOM)

   B. Error logger (ERRFMT)

   C. Job controller (JOB_CONTROL)

   D. Symbionts (SYMBIONT_n)

VII. Software Components of DECnet-VAX

THREE MAIN PARTS OF VMS

Scheduling and Process Control

Functions

- Assign processor to computable process with highest priority
- Attend to process state transitions
- Facilitate synchronization of processes
- Perform checks and actions at timed intervals

Code and Data

- Scheduler interrupt service routine
- Report system event code (IPL 3)
- Hardware clock and software timer interrupt service routines IPL 22 or 24 for HRDWE clock (IPL 7 for SFWE interrupt)
- System services ($WAKE)

Memory Management

Functions

- Translate virtual addresses to physical addresses
- Distribute physical memory among processes
- Protect process information from unauthorized access
- Allow selective sharing of information between processes

Code and Data

- Pager fault service routine and swapper process
- PFN database, page tables
- System services ($CRETVA)

I/O Subsystem

Functions

- Read/write devices on behalf of software requests
- Service interrupts from devices
- Log errors and device timeouts

Code and Data

- Device drivers, device-independent routines
- I/O data structures
- System Services ($QIO)

The Parts of the Operating System

```
VAX/VMS V4.0  on node COMICS  26-SEP-1984 13:34:35.10   Uptime    0 11:13:52
  Pid     Process Name     State  Pri      I/O        CPU         Page flts Ph.Mem
00000080 NULL              COM     0         0   0 09:10:38.72          0      0
00000081 SWAPPER           HIB    16         0   0 00:01:08.46          0      0
00000084 ERRFMT            HIB     8       834   0 00:00:07.34         67     88
00000085 OPCOM             LEF     8       133   0 00:00:01.62        625     58
00000086 JOB_CONTROL       HIB     9      4110   0 00:00:45.73        155    299
00000088 SYMBIONT_0001     HIB     6      1161   0 00:01:19.87       7514     45
00000109 SOUZA             LEF     7      8777   0 00:00:50.47      14077    445
0000008B NETACP            HIB    10      3375   0 00:01:25.81       4121   1500
0000008C EVL               HIB     6        32   0 00:00:00.73        265     44  N
0000008D REMACP            HIB     9       111   0 00:00:00.55         72     41
0000018F HANDEL            LEF     7      2631   0 00:00:31.96      14528    150
00000110 BACH              LEF     6     15106   0 00:01:58.01      20174    400
00000191 STRAVINSKY        LEF     9      6689   0 00:01:14.64      16548    372
00000096 OPERATOR          LEF     7    122767   0 00:19:34.03       6974    499
00000197 CHOPIN            LEF     4      4140   0 00:00:43.43       9015    129
00000218 MARSH             LEF     4     17492   0 00:04:25.90      59864    150
0000019E BATCH_509         COM     4      1076   0 00:00:16.36       7318    312  B
000001AA SCOTT_KEY         LEF     4      2788   0 00:00:48.76      11152    127
0000012D HUNT              CUR     4     17262   0 00:02:22.36      23639    178
0000013A _TTA3:            LEF     4      1765   0 00:00:32.21       9565    138
```

Example 1 Sample SHOW SYSTEM Output

- List of processes on the system
- Images running in process context
- Only the "upper layer"
- Notice lack of:
  - Scheduling program
  - I/O handling programs
  - System service code

Functions Handled "Below" User Level

- Scheduling of processes for CPU time
  - Highest-priority process
- Memory management within a process
- System services
  - $CREPRC
  - $GETxxx
  - $CREMBX
- Record Management Services (RMS)
  - OPEN
  - GET, PUT
  - CLOSE
- I/O Code to handle peripherals
- Time Management
- Basic resource management

INVOKING SYSTEM CODE

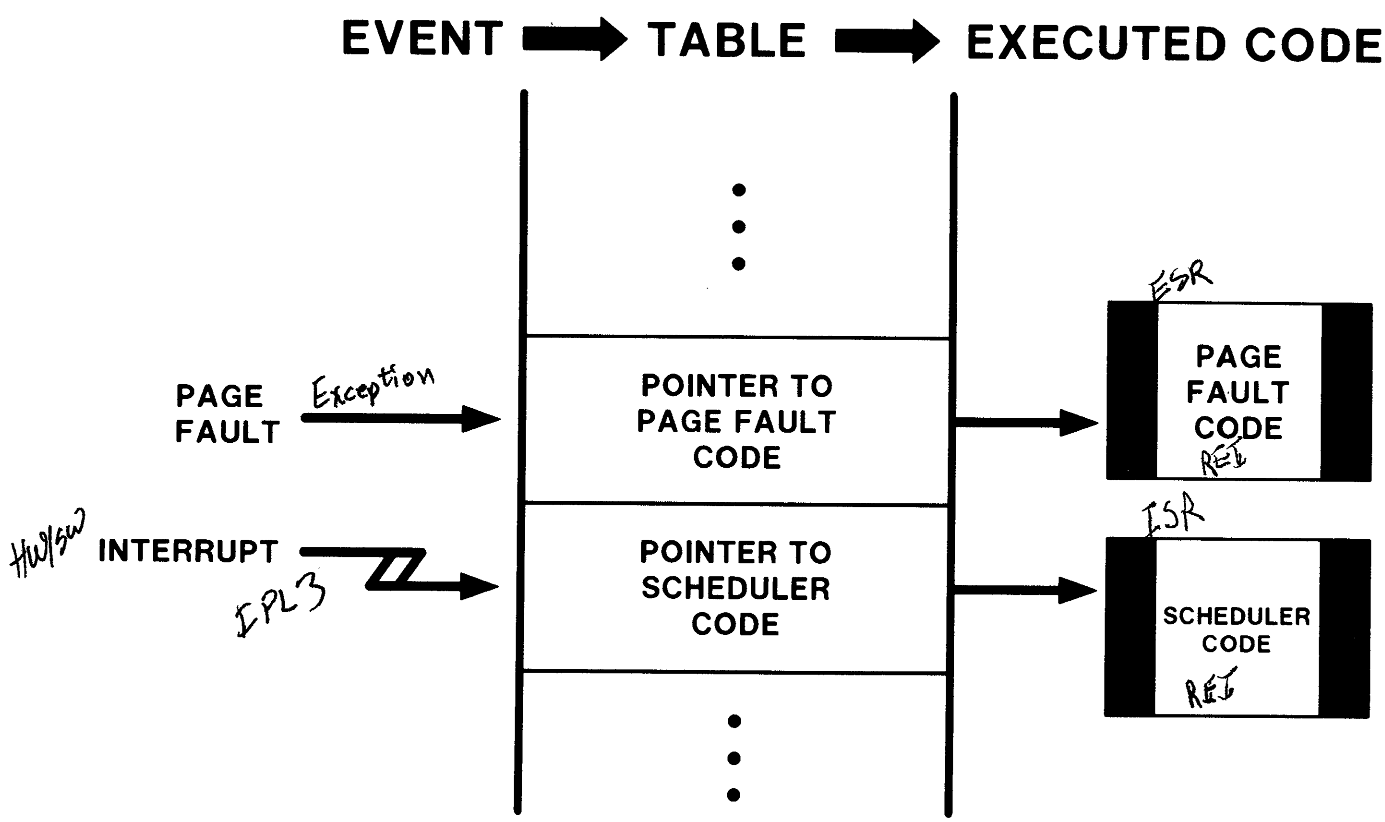


Figure 1 Invoking System Code

- 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

(HRDWE addr) SCBBR pts to SCB base "PA"

Interrupts vs. Exceptions

Table 1 Differences Between Interrupts and Exceptions

Interrupts	Exceptions
Asynchronous to the execution of a process	Caused by process instruction execution
Serviced on the system-wide interrupt stack in system-wide context	Serviced on the process local stack in process context
Change the interrupt priority level to that of the interrupting device	Does not alter interrupt priority level
Cannot be disabled, although lower-priority interrupts are queued behind higher-priority interrupts	Some arithmetic traps can be disabled

except: 1.) K stack corruption [IPL 31] 2.) machine check [IPL 31]

Hw

Sw

Traps	Faults	Aborts
Not Recoverable	Recoverable	Not Recoverable
does next inst	"Backed up Inst"	
(Div by ∅)	(Page Fault)	(Machine Check)

HARDWARE MAINTAINED PRIORITY LEVELS

- Processor is always operating at one of 32 possible hardware-maintained priority levels (0 - 31).
- Operating at a higher level causes hardware to block interrupts at the same and lower levels from being serviced.
- Hardware determines which code will execute after an interrupt occurs.
- How to get into and out of different levels:

  1. Interrupt

     Into - Hardware requests interrupt (for example, from a terminal). Levels 16 through 31. Software requests interrupt (uses MTPR instruction). Levels 0 through 15.

     Out of - Use REI instruction.

  2. Block Interrupt

     Into - Software raises priority level (uses MTPR).
     Out of - Software lowers priority level (uses MTPR).

- These hardware-maintained priority levels are called Interrupt Priority Levels (IPLs).

Two Types of Priority

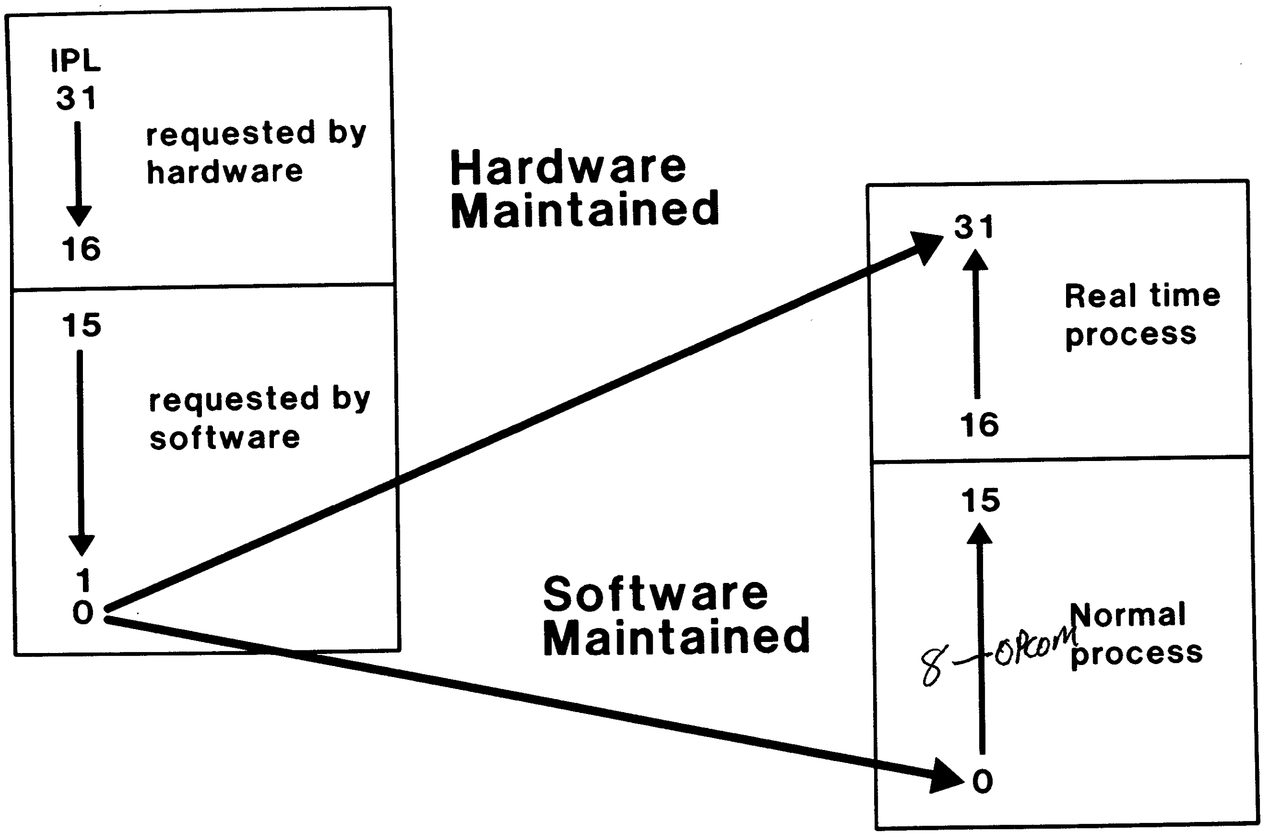


Figure 2 Two Types of Priority

MTPR #n, #PR$_xxx (move to CPU reg)

MTPR #n, #PR$_IPL (changes IPL)

I/O device interrupts:
IPL 20-23
(relative to bus priority)

Interrupt Servicing Sequence

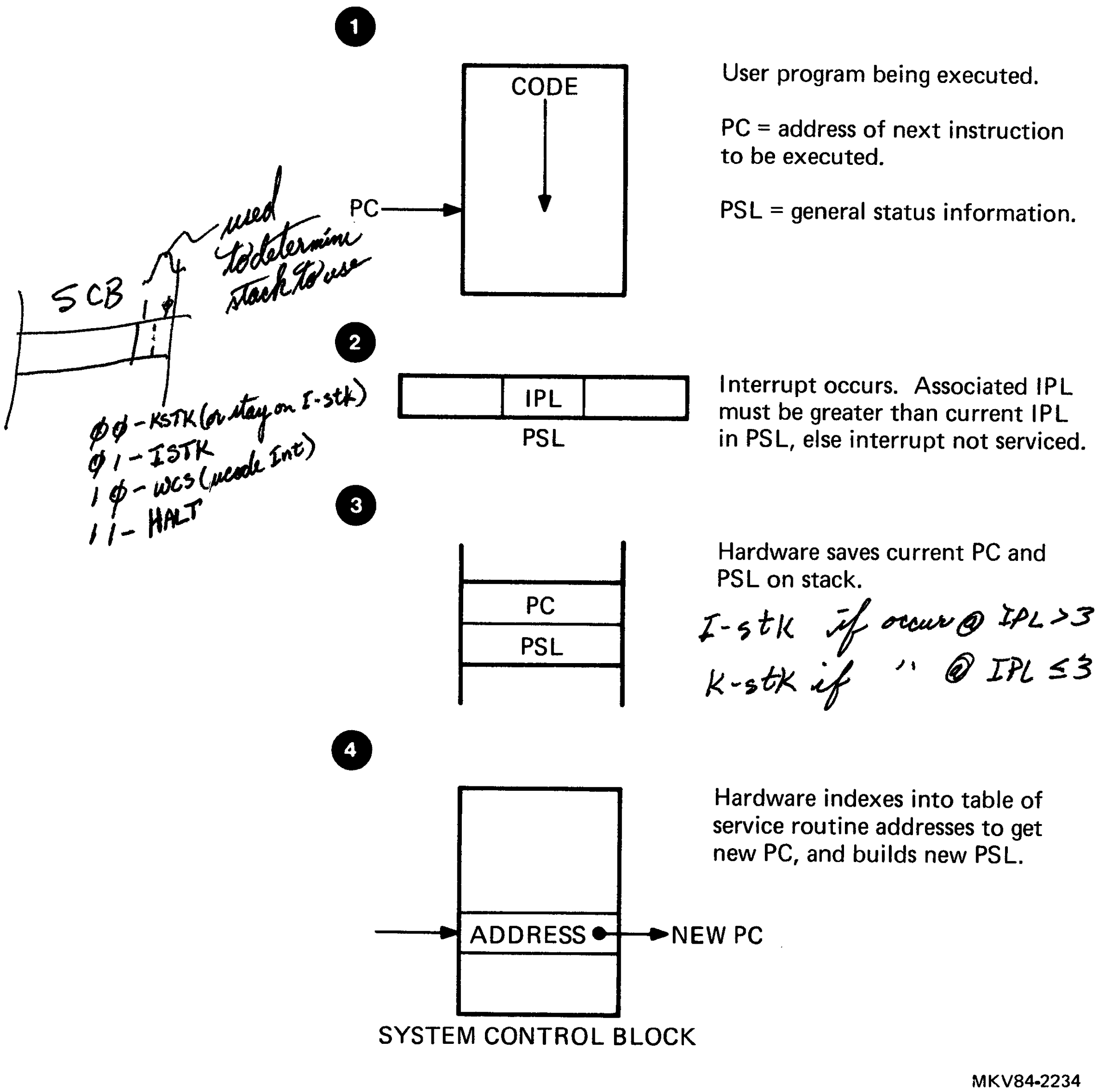
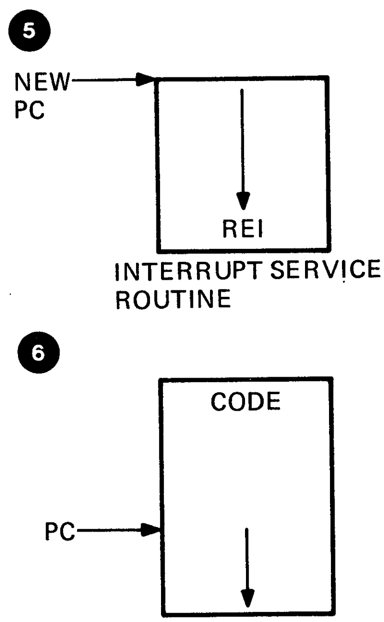


Figure 3 Example of Interrupt Servicing
(Sheet 1 of 2)



Interrupt service routine executes at new IPL.

At end, interrupt dismissed with REI instruction (making sure old PC and PSL are at top of stack).

REI

- Pops PC, PSL from stack
- Checks PSL (ensure to privilege of current mode)
- Moves PC, PSL to CPU registers
- Transfers control to PC

Interrupted program continues exectution.

MKV842235

Figure 3 Example of Interrupt Servicing (Sheet 2 of 2)

ACCESS MODES AND COMPONENTS

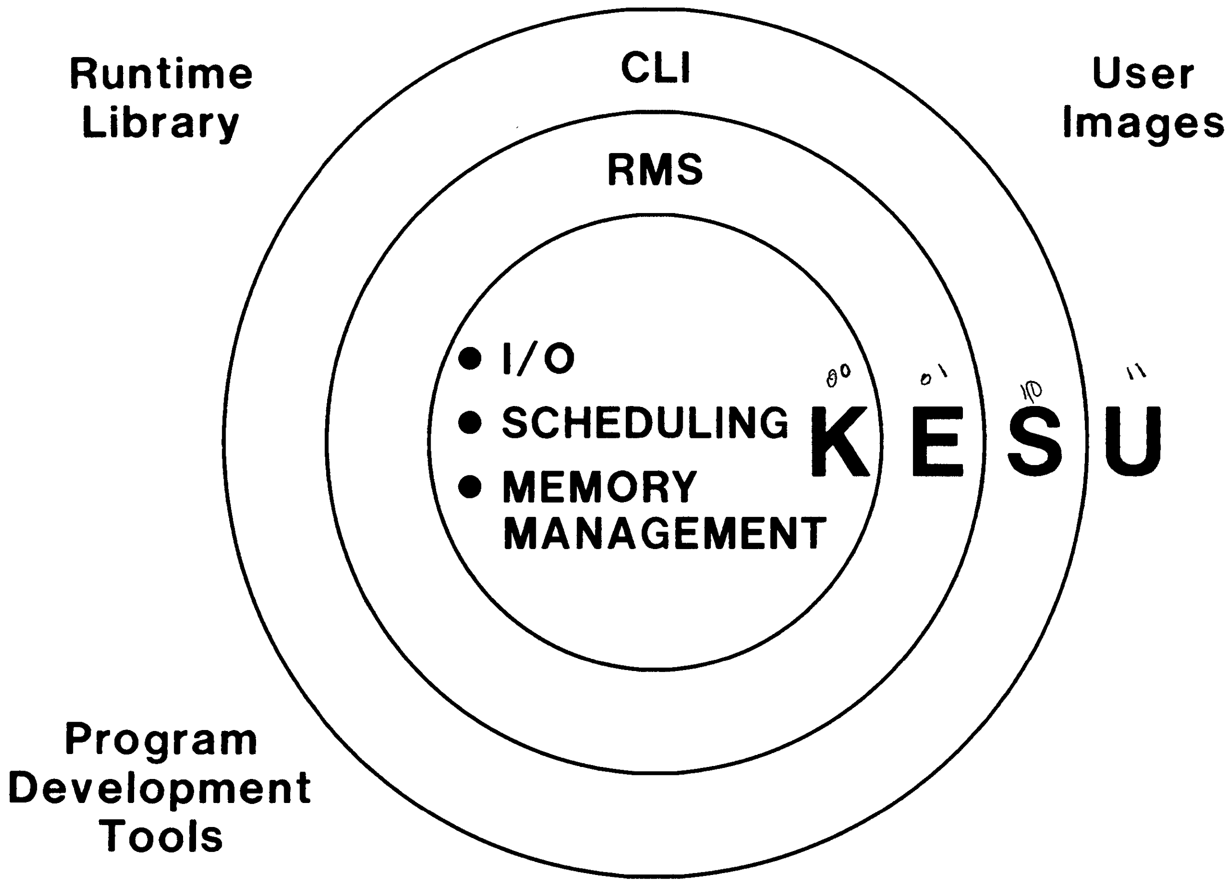


Figure 4 Access Modes and Components

- 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

LOCATION OF CODE AND DATA

by convention: 1st page is "no access"

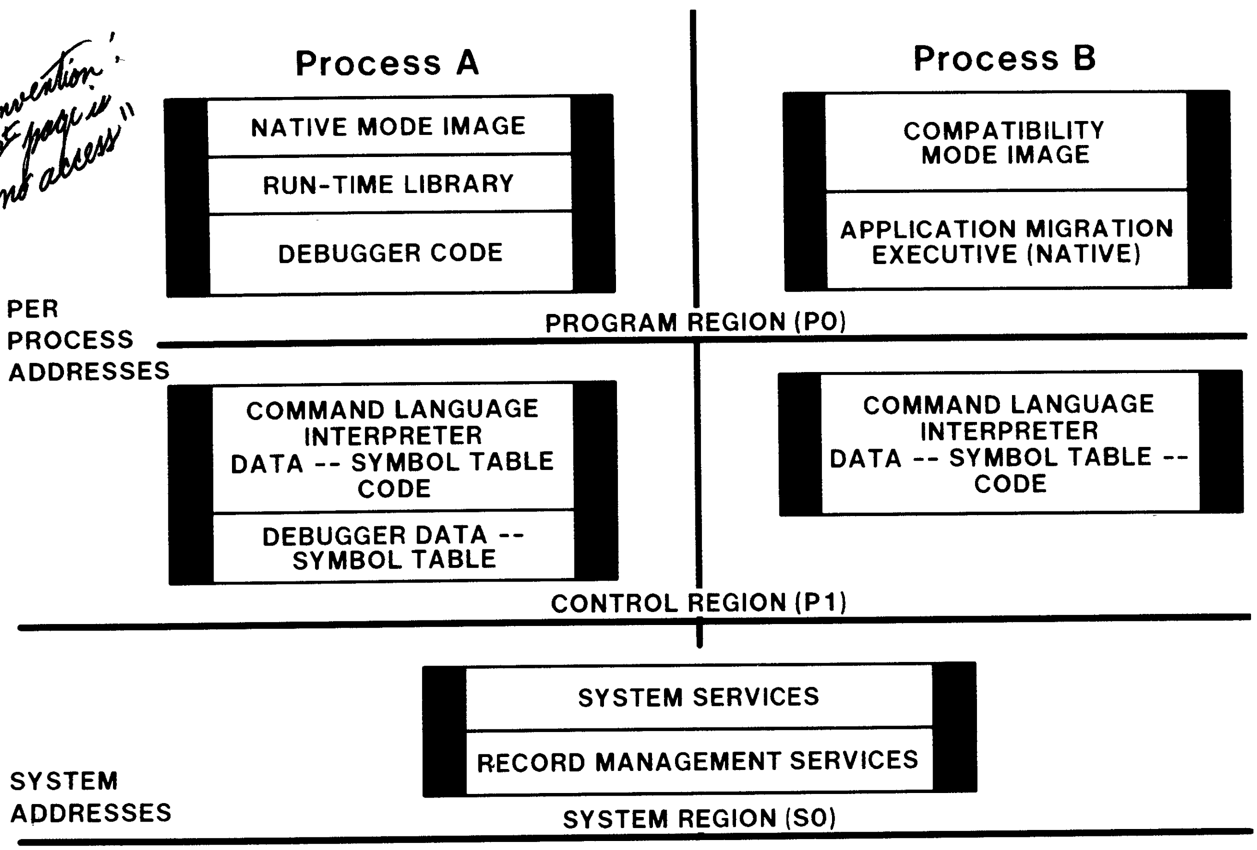


Figure 5 Location of Code and Data in Virtual Address Space

- Images running within processes use several different types of software components
- PØ space (program region) - user's code and data
- P1 space (control region) - process-specific information; stored by the operating system
- PØ space and P1 space are mapped differently for native and compatibility mode images
- SØ space (system space) - operating system code and data; one copy shared by all processes

XQP is mapped through P1 space

Entry Paths Into VMS Kernel

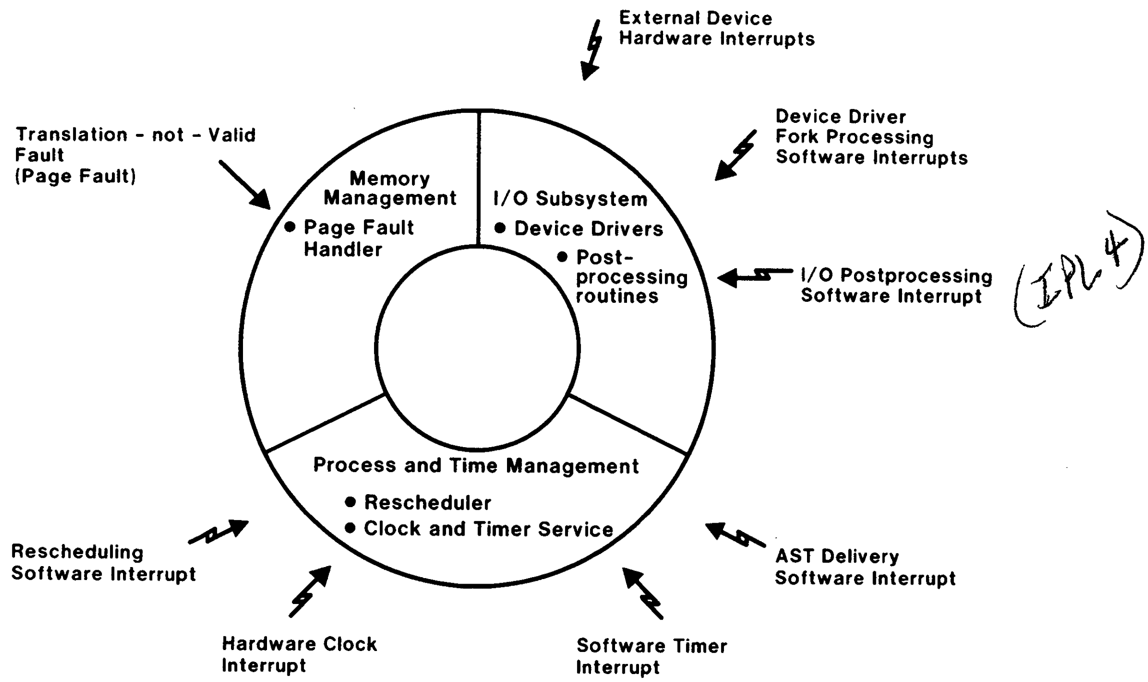


Figure 6 Entry Paths into VMS Kernel

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

Table 2 Summary of System Components and Functions

Function	System Component
Assigns CPU to highest-priority computable process	SCHEDULER
Moves working set between disk and memory	SWAPPER
Moves pages from disk to memory	PAGER
Updates system clock and quantum field, check for servicing at intervals	HARDWARE CLOCK ISR
Performs servicing at intervals • Checks for quantum end • Causes events to be posted • Checks device timeout • Wakes swapper and error logger	SOFTWARE TIMER ISR
Handles requests to/replies from operator	OPCOM
Writes errors to error log file	ERROR LOGGER
Maintains volume structures for driver	ANCILLARY CONTROL PROCESS
Maintains disk and file structure for Files-11 ODS-2 disks	FILES-11 XQP
Creates processes for print jobs, batch jobs, interactive jobs	JOB CONTROLLER
Controls devices, service device interrupts, check for and report device errors	DRIVERS
Handles printing of files	PRINT SYMBIONTS
Handles process state transitions resulting from event completion	REPORT SYSTEM EVENT

THREE TYPES OF SYSTEM COMPONENTS

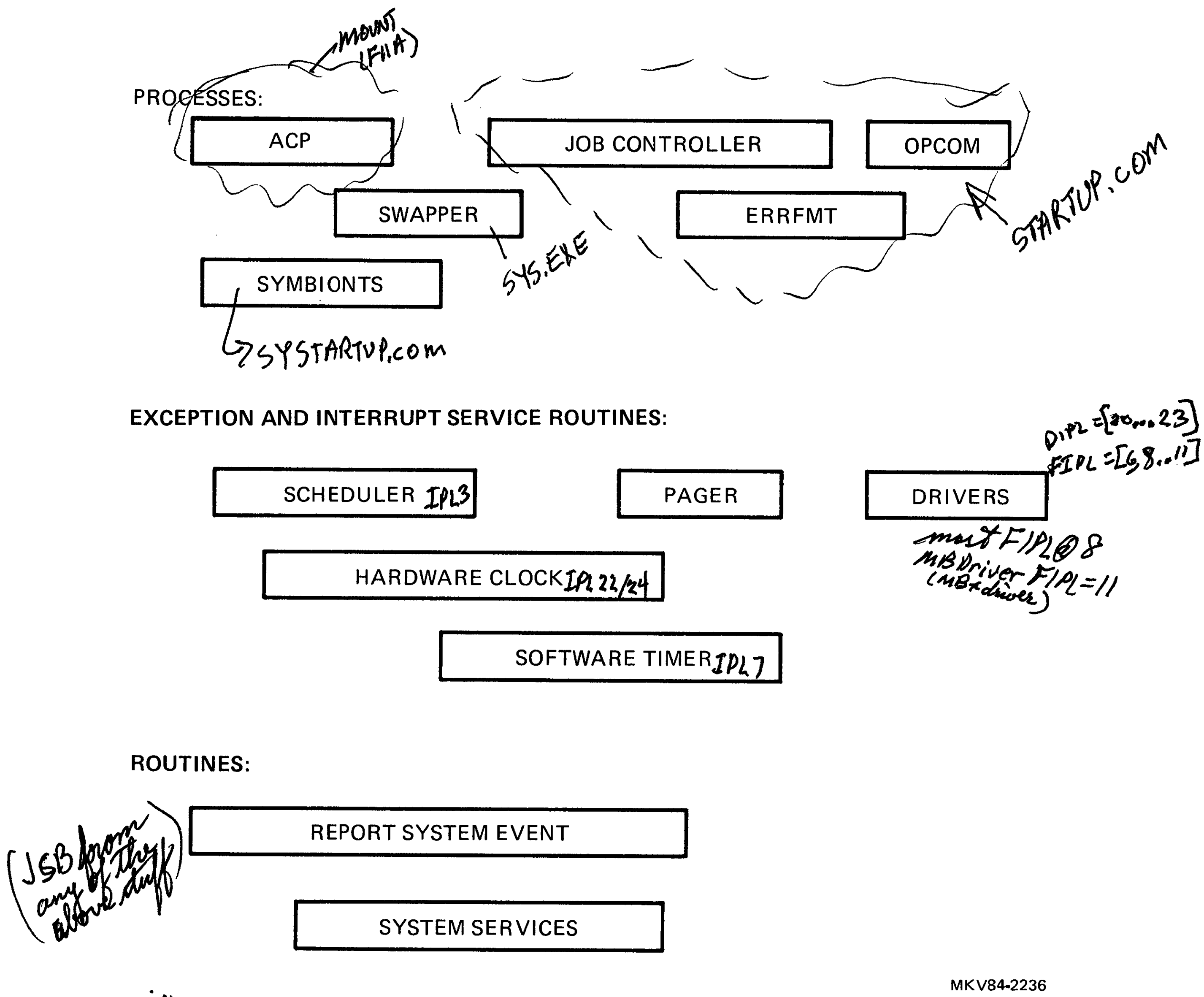


Figure 7 Three Types of System Components

INTERACTION OF VMS COMPONENTS

Hardware Clock Interrupt

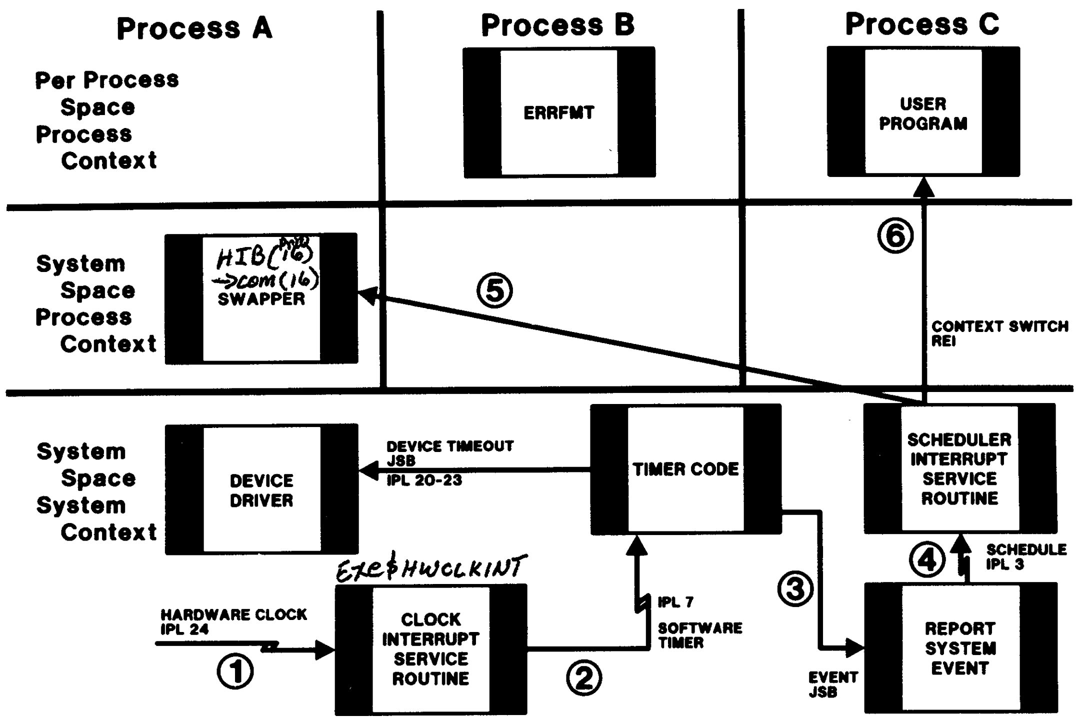


Figure 8 Hardware Clock Interrupt

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

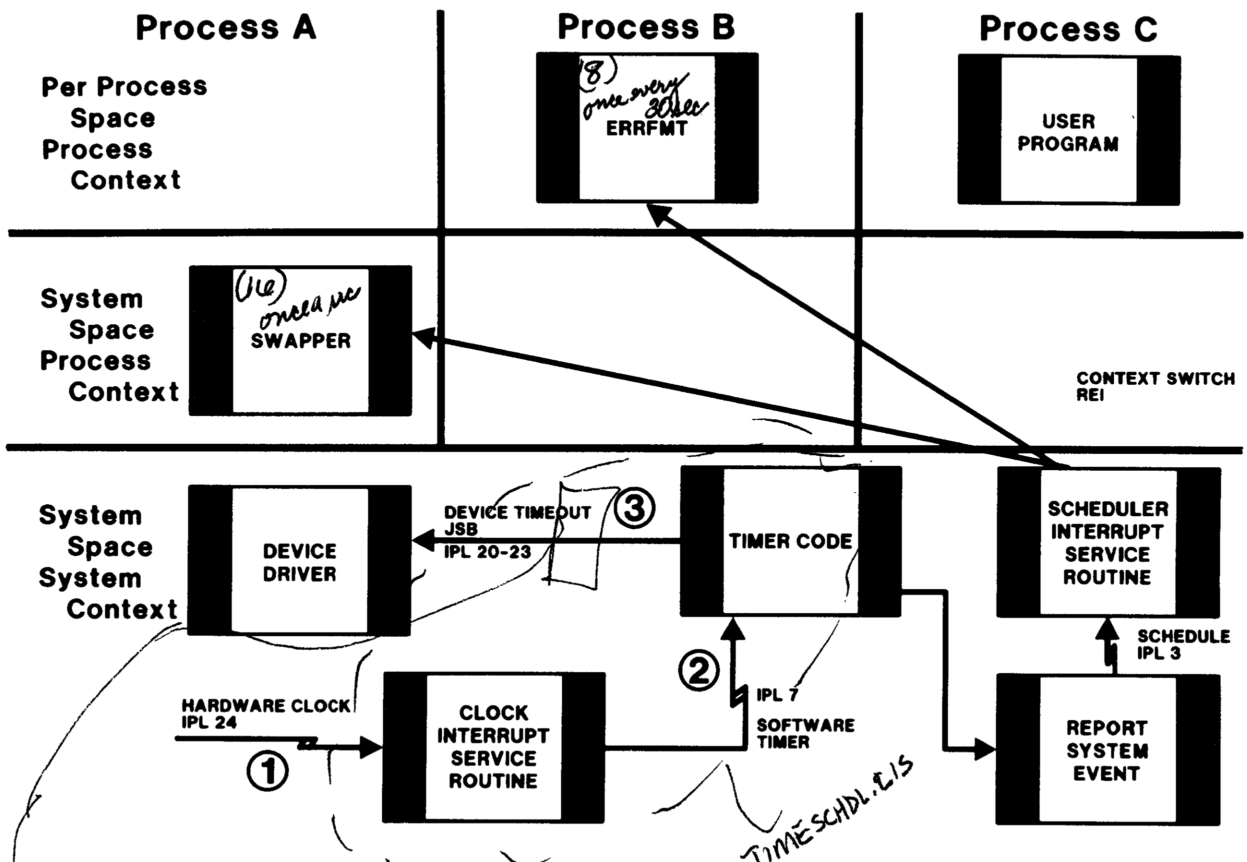


Figure 9 Periodic Check for Device Timeout

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

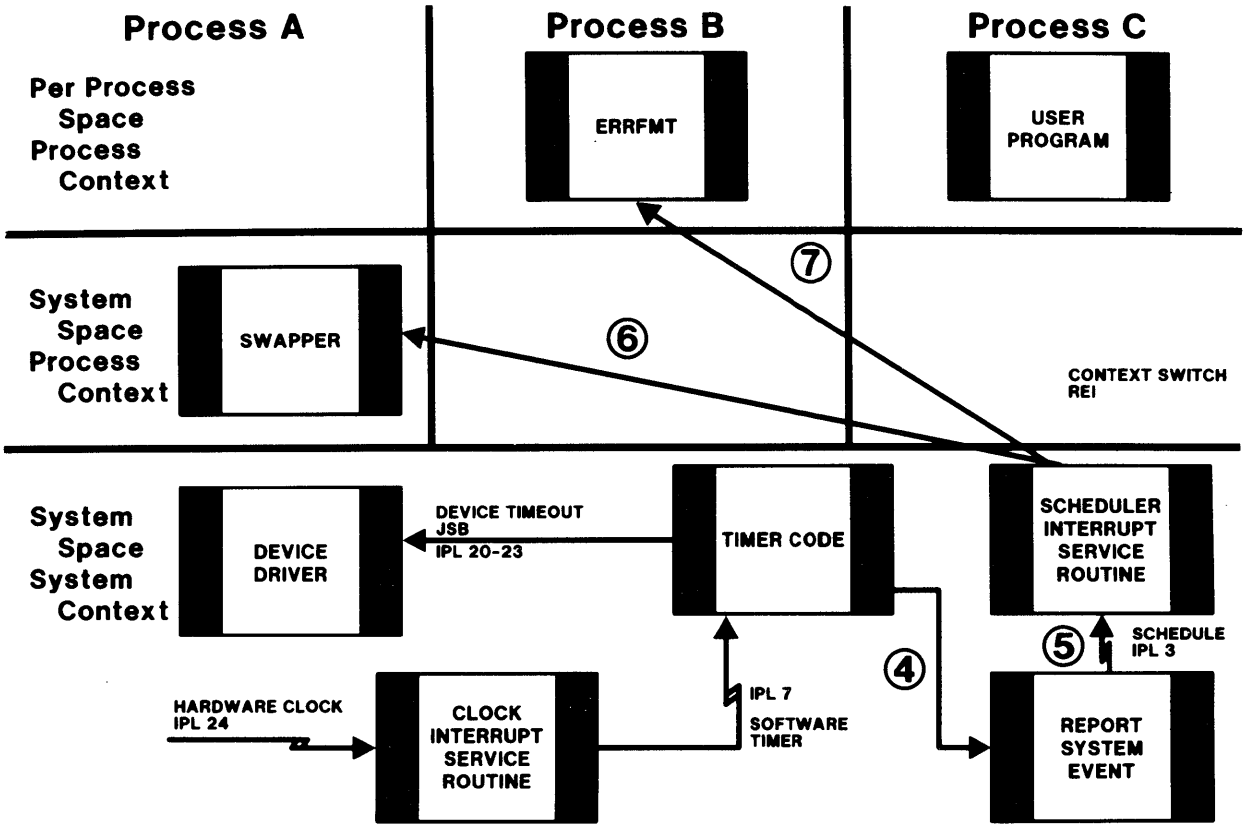


Figure 1Ø Periodic Wake of Swapper, Error Logger

4. The same system subroutine can wake the swapper process and the error logger process.

5. Scheduler interrupt is requested.

6,7. Swapper and error logger will eventually run.

System Event Reporting

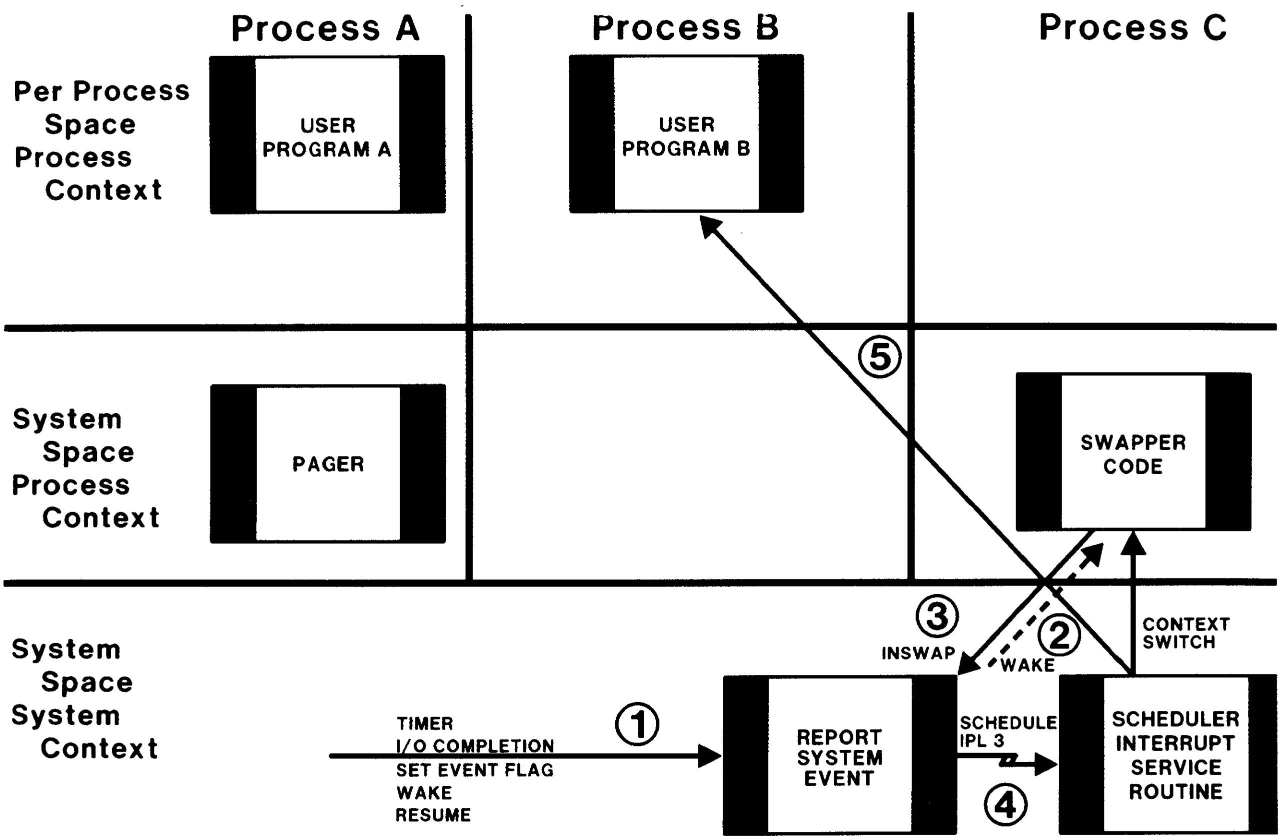


Figure 11 System Event Reporting

Page Fault

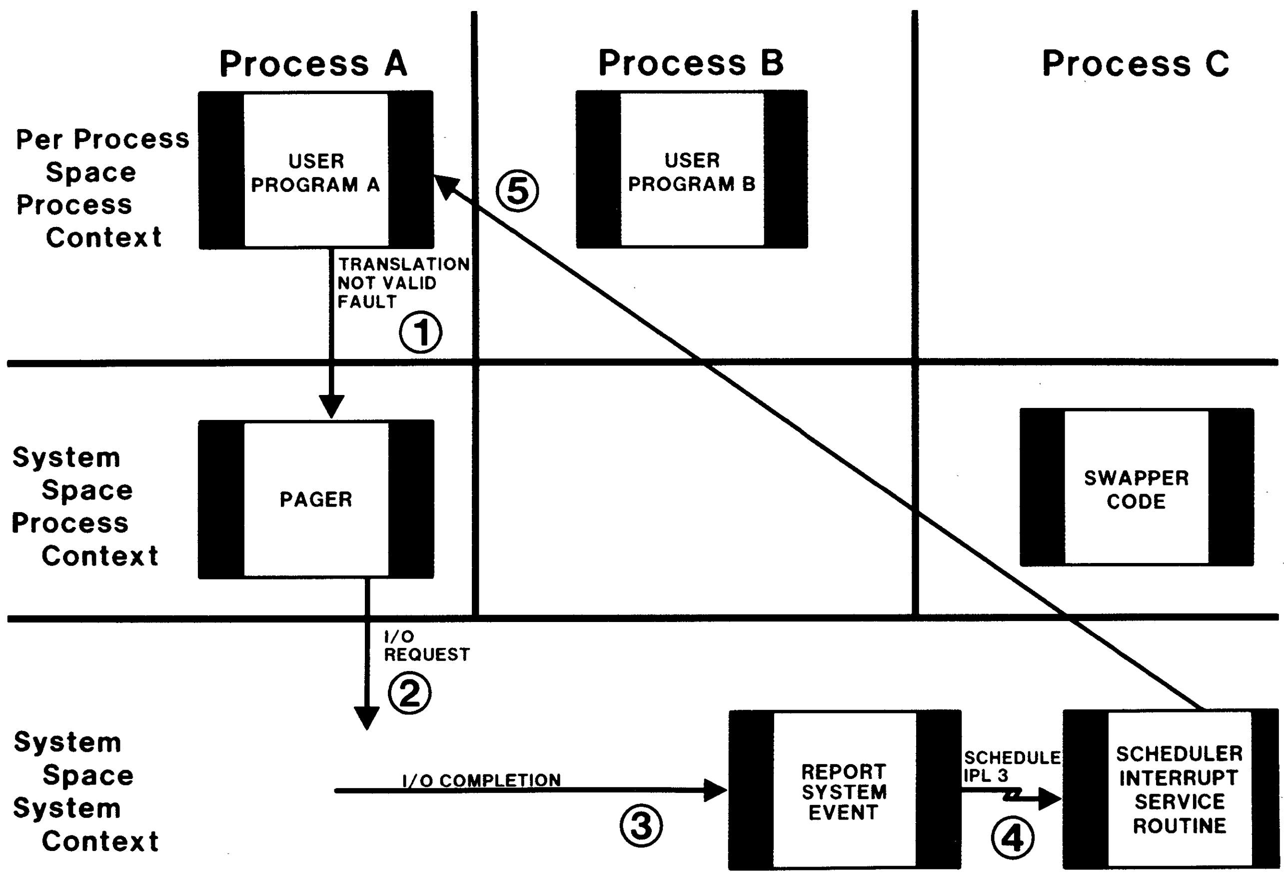


Figure 12 Page Fault

Data Transfer Using RMS

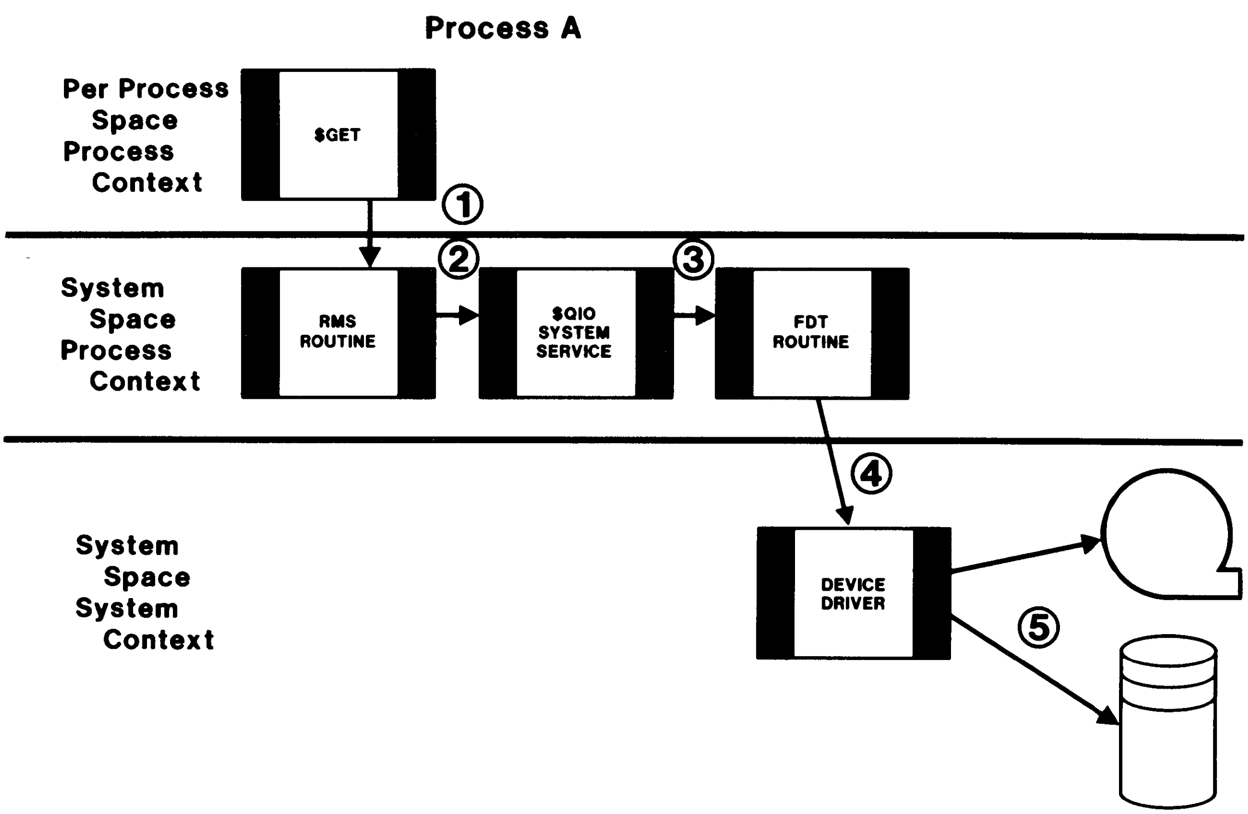


Figure 13 Data Transfer Using RMS

File Manipulation Using RMS

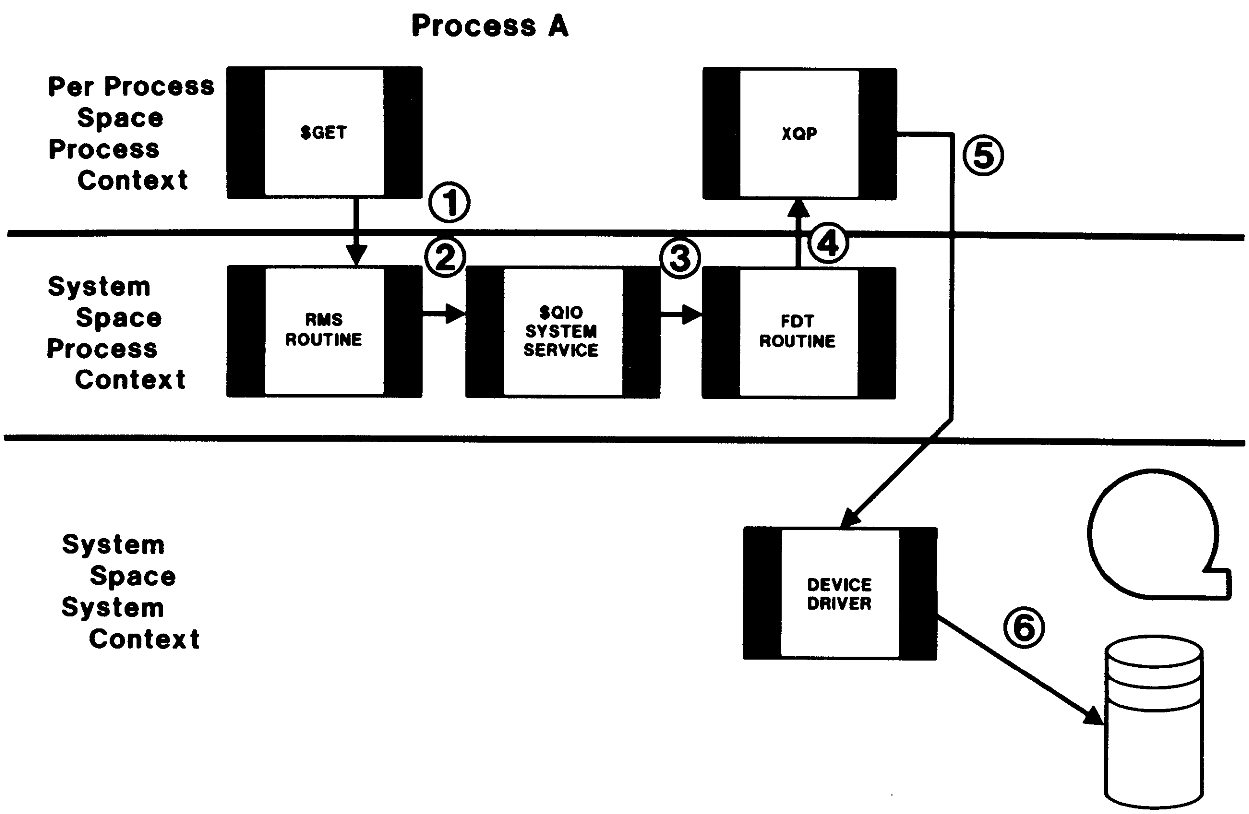


Figure 14 ODS-2 File Manipulation Using RMS

When the ODS-2 file structure is imposed on a disk volume, the following operations require the intervention of the eXtended QIO Procedures (XQP) to interpret or manipulate the file structure.

- File open
- File close
- File extend
- File delete
- Window turn (for read or write)

File Manipulation Using RMS

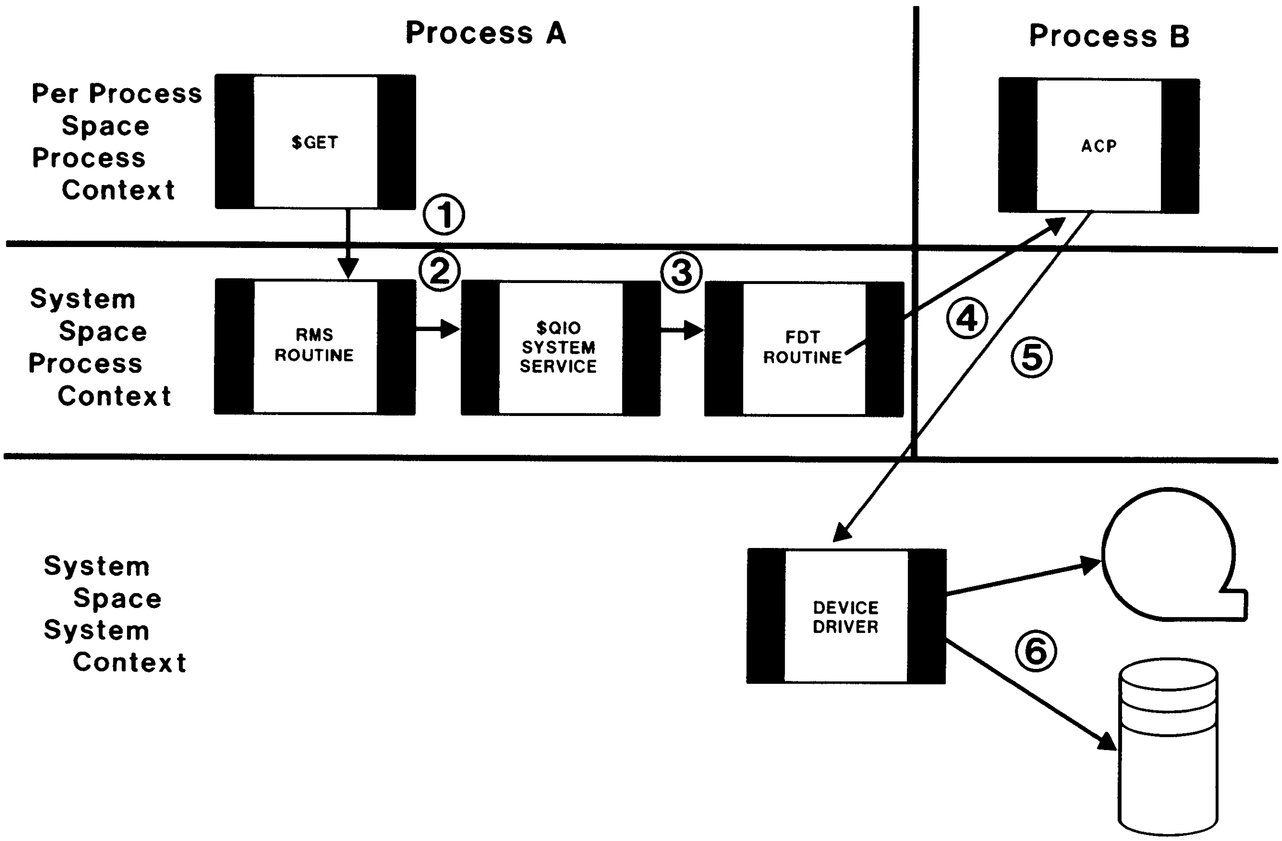


Figure 15 File Manipulation Using an ACP

Ancillary Control Processes (ACPs) help drivers implement:

- Magnetic Tape File Structure
- Network Operations
- ODS-1 On-Disk File Structure

Data Transfer Using $QIO

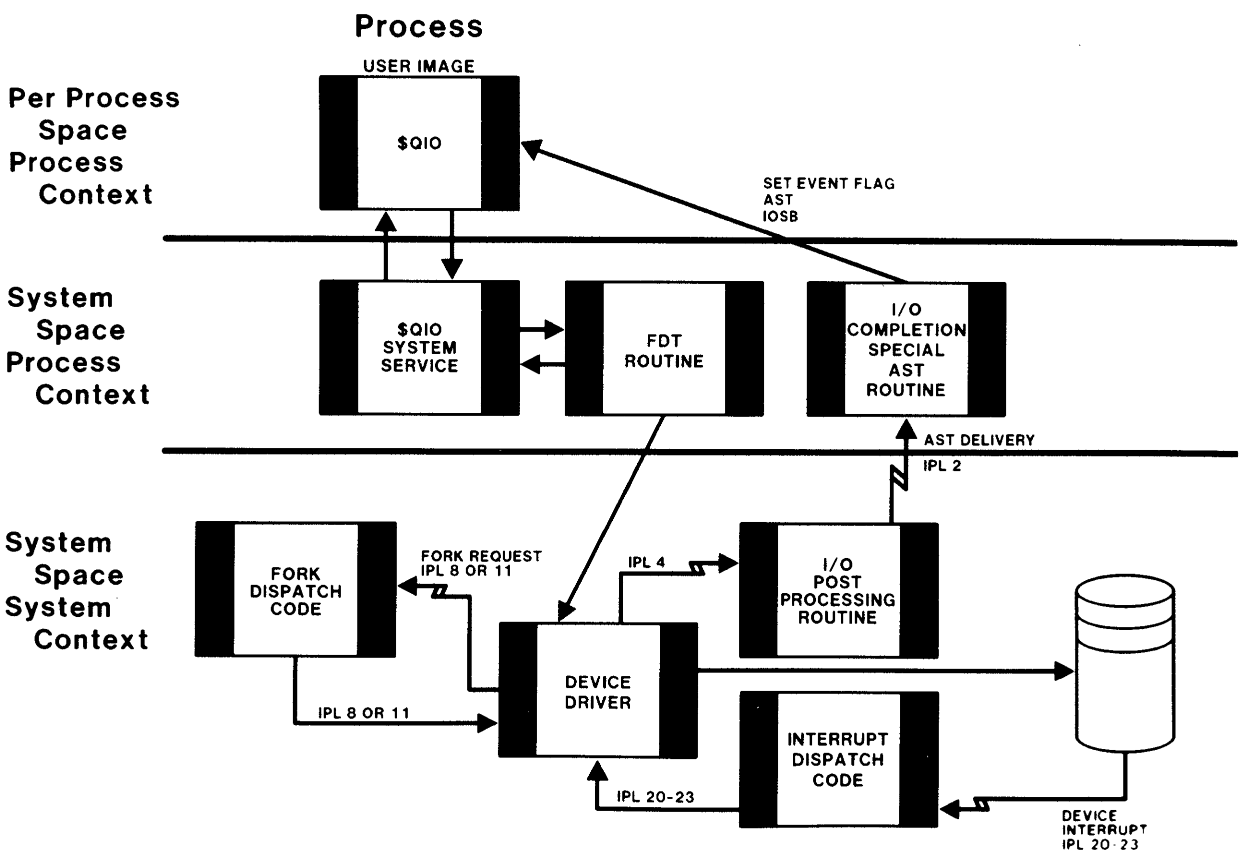


Figure 16 Data Transfer Using $QIO

$QIO Sequence of Events

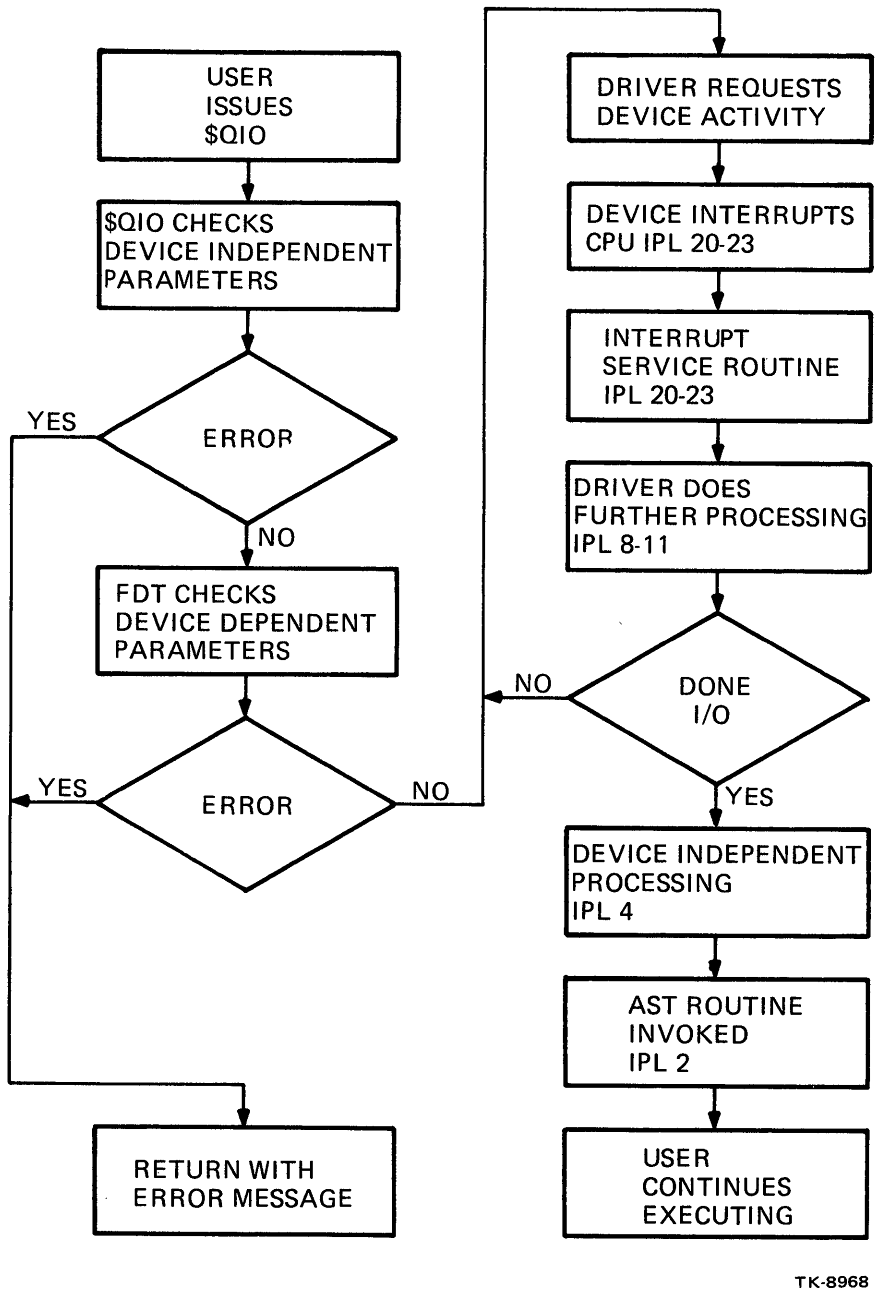


Figure 17 $QIO Sequence of Events

EXAMPLES OF SYSTEM PROCESSES

OPCOM, Error Logger

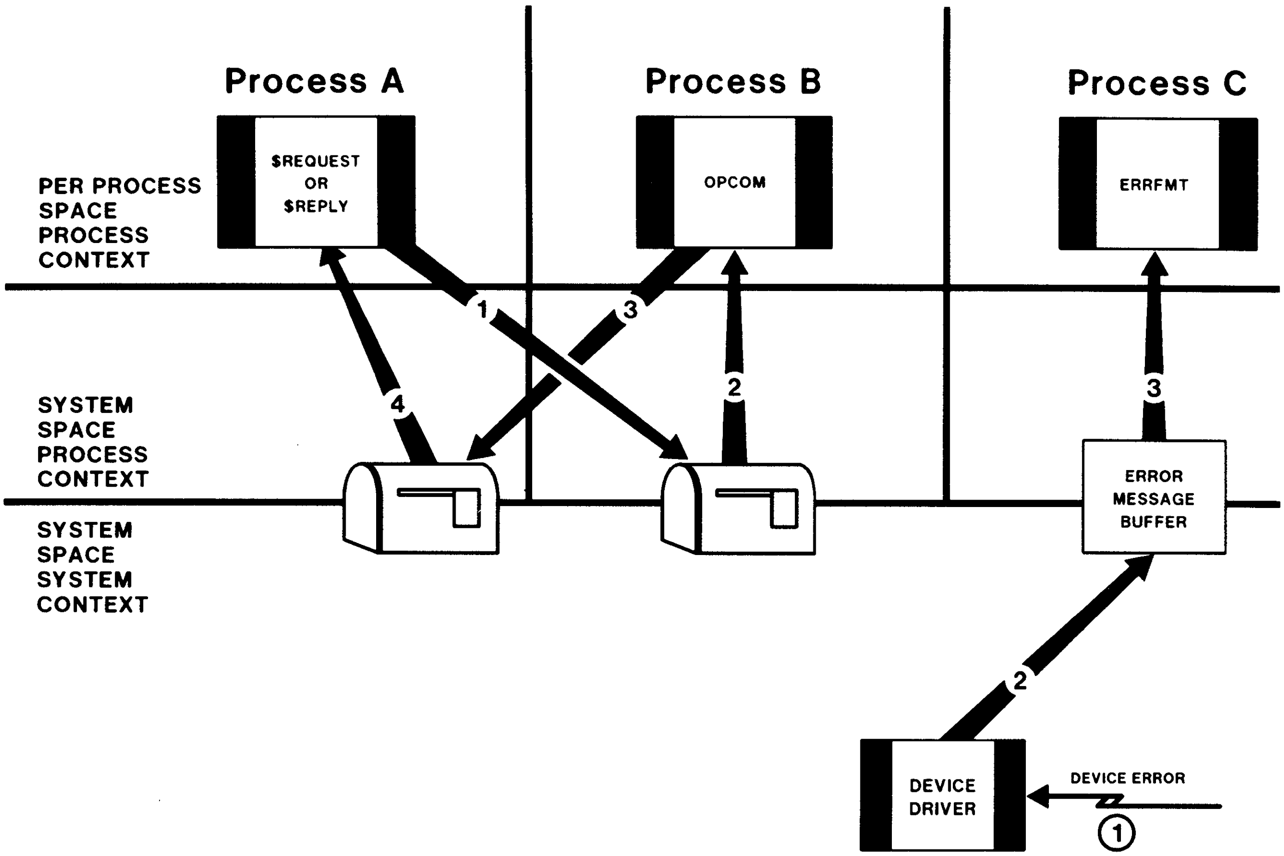


Figure 18 OPCOM, Error Logger

OPCOM Process

- Handles requests to, and responses from, the system operator

Error Logger

- Has buffers in memory in which detected errors are recorded
- Writes to the error log file

Print Jobs

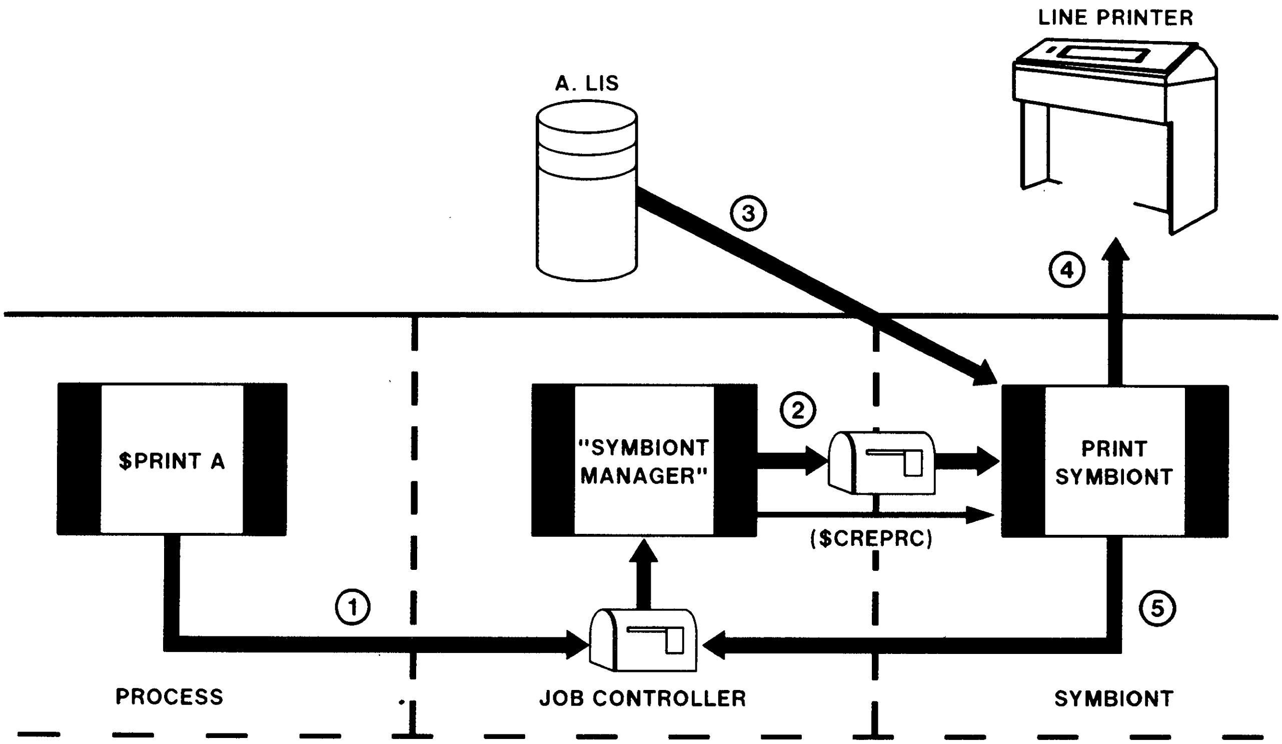


Figure 19 Print Jobs

Batch Jobs

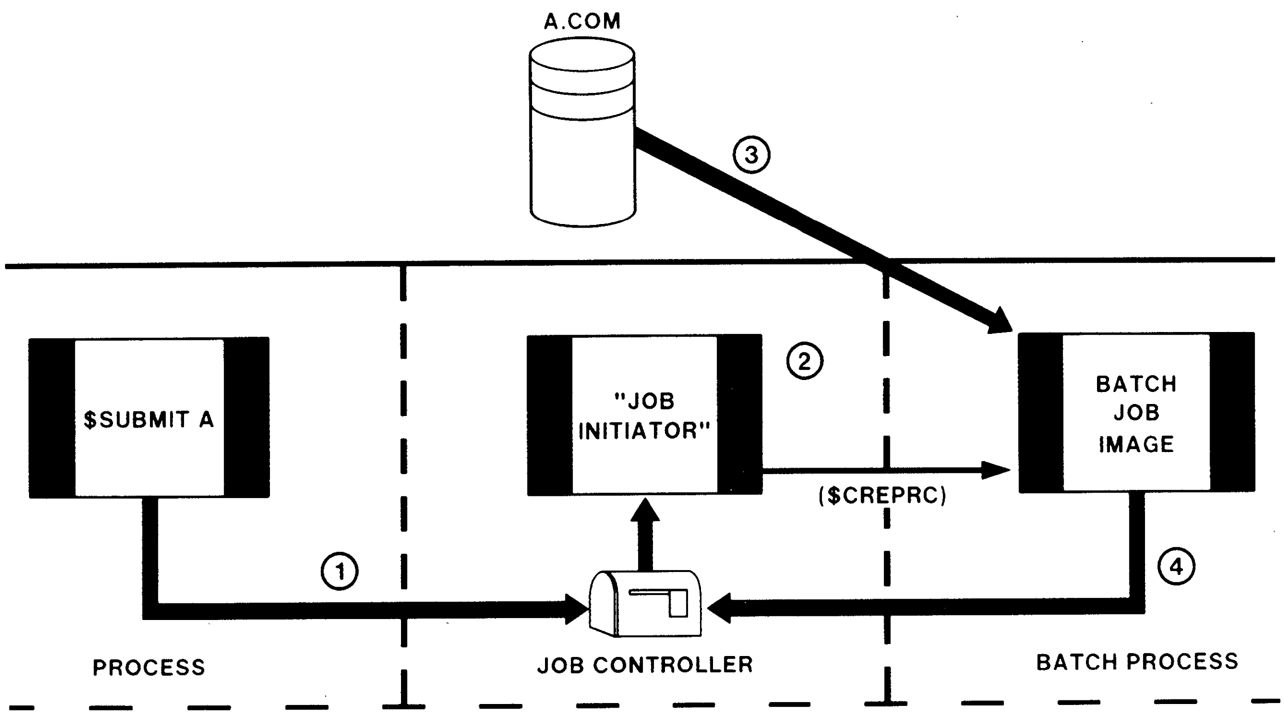


Figure 20 Batch Jobs

Terminal Input

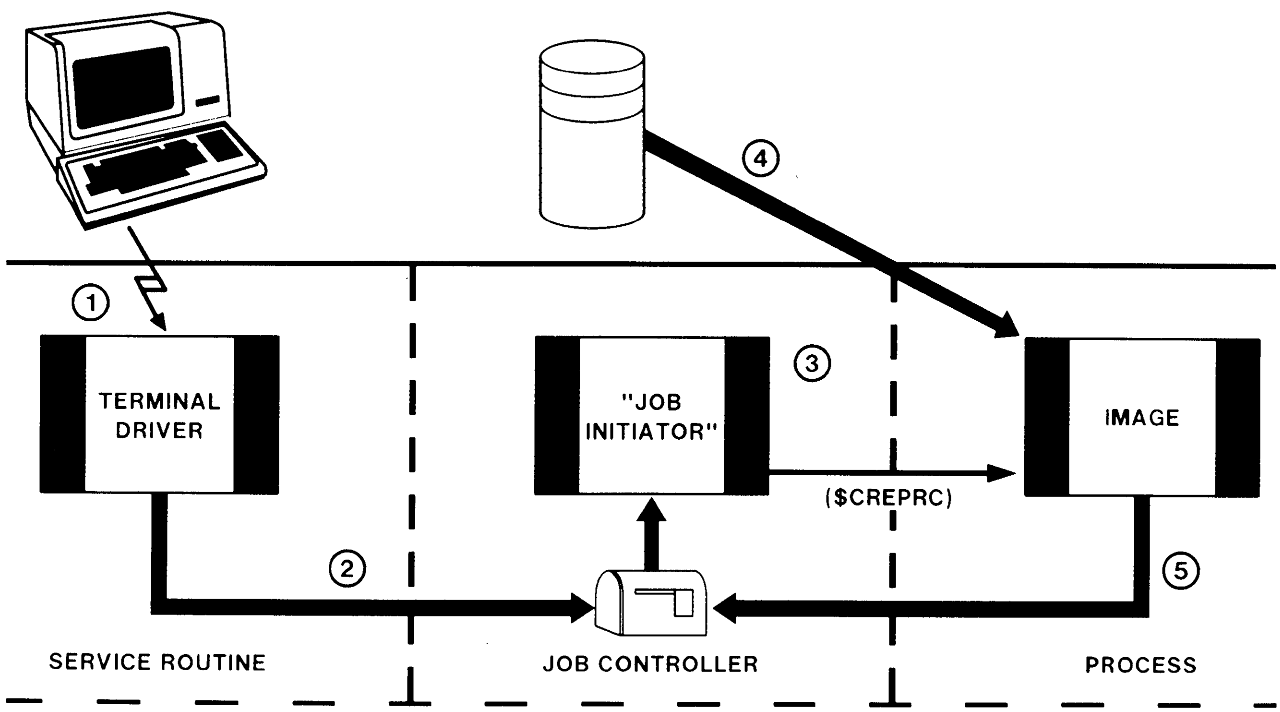


Figure 21 Terminal Input

Card Reader Input

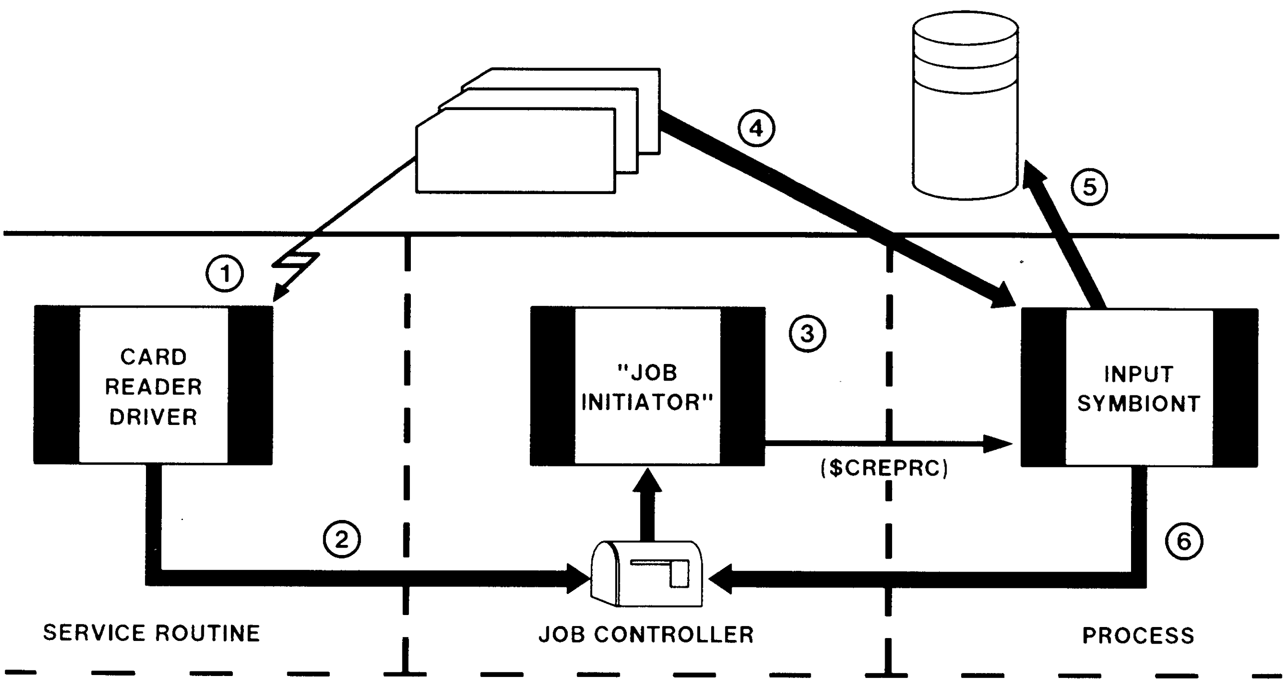


Figure 22 Card Reader Input

SOFTWARE COMPONENTS OF DECnet-VAX

Data Link Device Drivers

- XMDRIVER, XDDRIVER, XGDRIVER - handle synchronous DDCMP links (DMR11, DMP11, DMF32)
- XEDRIVER - for DIGITAL Ethernet UNIBUS Adapter (DEUNA)
- XQDRIVER - for DIGITAL Ethernet Q-bus Adapter (DEQNA)
- CNDRIVER - handles Computer Interconnect (CI)
- NWDRIVER - for X.25 (used for datalink mapping)
- Terminal drivers - for asynchronous DECnet (DDCMP protocol)

NETDRIVER and NETACP

- Implement routing, and End Communications Layer (ECL)
- NETDRIVER handles the time-critical functions (for example, transmit or receive data).
- NETACP handles the non-time-critical functions (for example, setting up logical link).

RMS, DAP Routines, and FAL_n

- Implement application layer for file transfer operations

RTTDRIVER, REMACP, and RTPAD

- Implement application layer for remote terminal access

Netserver

- Collection of programs used to start up a network user process on a remote node

Special DECnet Components

EVL

- Event logger process - collects and filters network event information; passes it to the correct destination
- Created at network start-up if event logging enabled

SERVER_n Process

- Process ready to handle a logical link

NCP, NML, MOM, MIRROR, NDDRIVER

- For network management
- For special functions (down-line load, up-line dump, device loopback tests)

DECnet Remote File Access

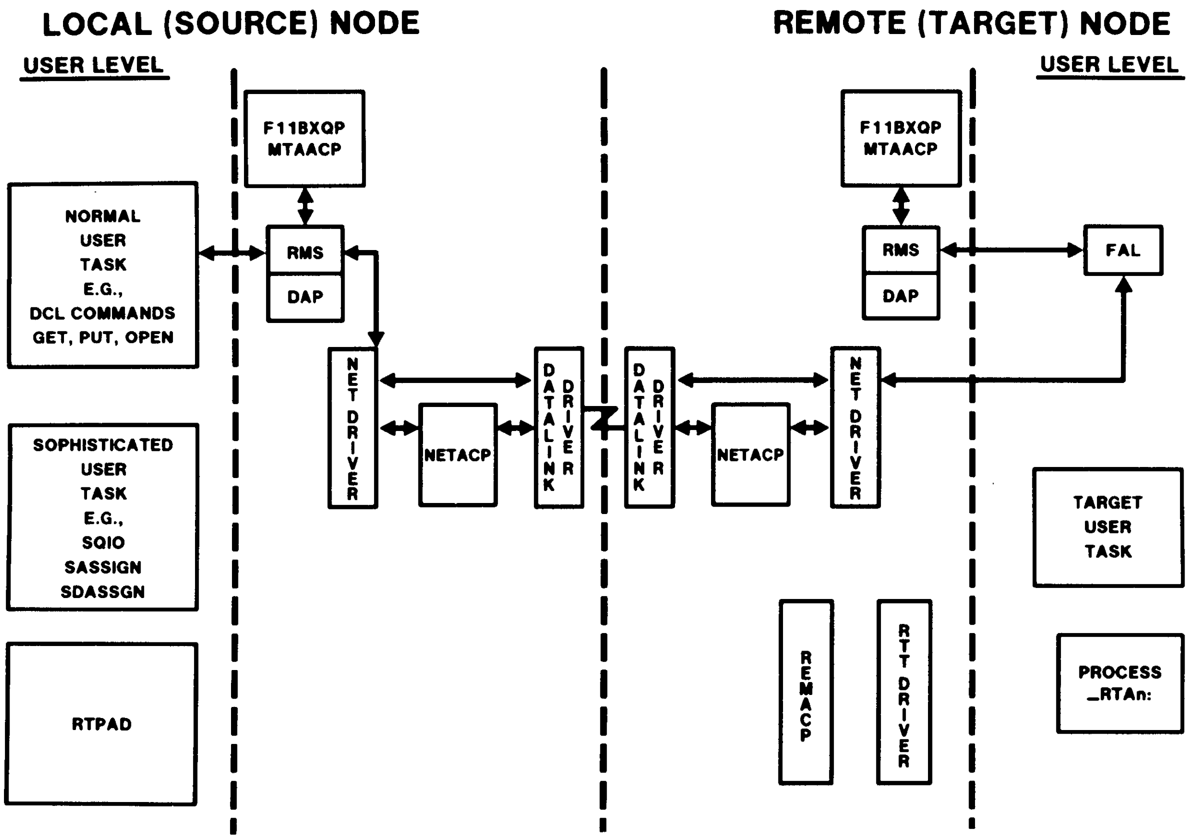


Figure 23 DECnet Remote File Access

- User issues DCL command, such as:

  TYPE NODEB"NAME PASSWORD"::DISK$:[DIRECTORY]FILENAME.TYP

- RMS detects "::" in file specification
- RMS and NETDRIVER use internal $QIOs
- NETACP process on each node sets up data structures to support logical link
- FAL_n process issues requests to RMS on remote node

SUMMARY

- How VMS Implements the Functions of an Operating System
- How and When Operating System Code is Invoked
- Interrupts and Priority Levels
- Location of Code and Data in Virtual Address Space
- Examples of Flows for:
  - Hardware clock interrupt
  - System event completion
  - Page fault
  - RMS request for I/O
  - $QIO request for I/O
- Examples of System Processes
  - Operator Communication (OPCOM)
  - Error logger (ERRFMT)
  - Job controller (JOB_CONTROL)
  - Symbionts (SYMBIONT_n)
- Software Components of DECnet-VAX

APPENDIX
ADDITIONAL DECnet-VAX INFORMATION

DECnet Protocols

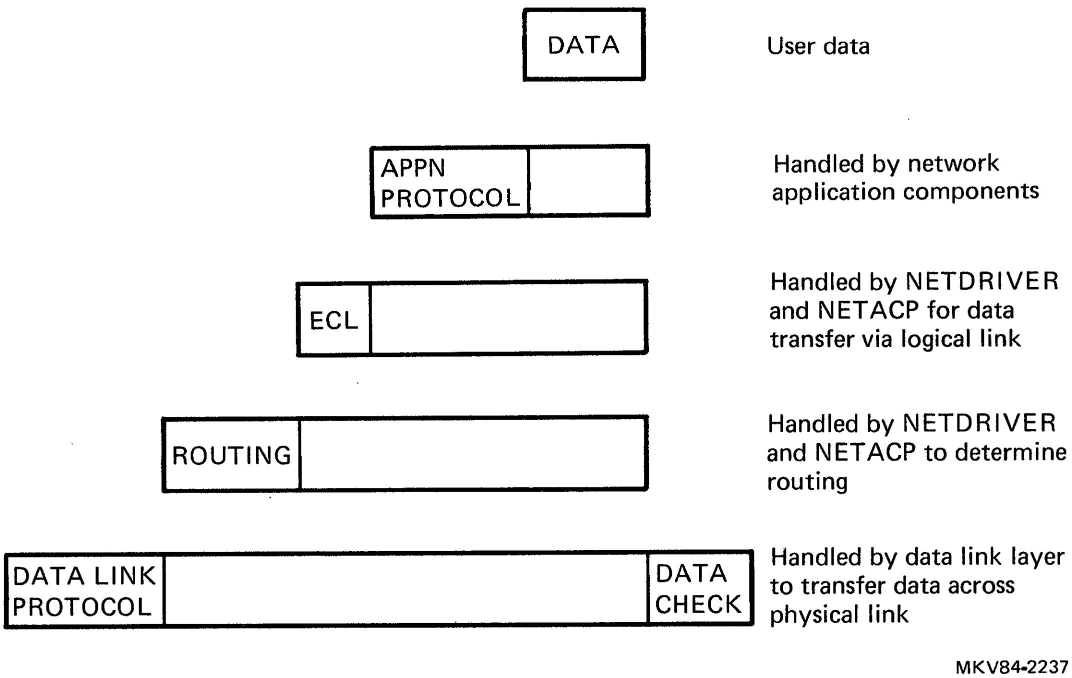


Figure 24 DECnet Protocol Layers

DECnet Task-to-Task Communication

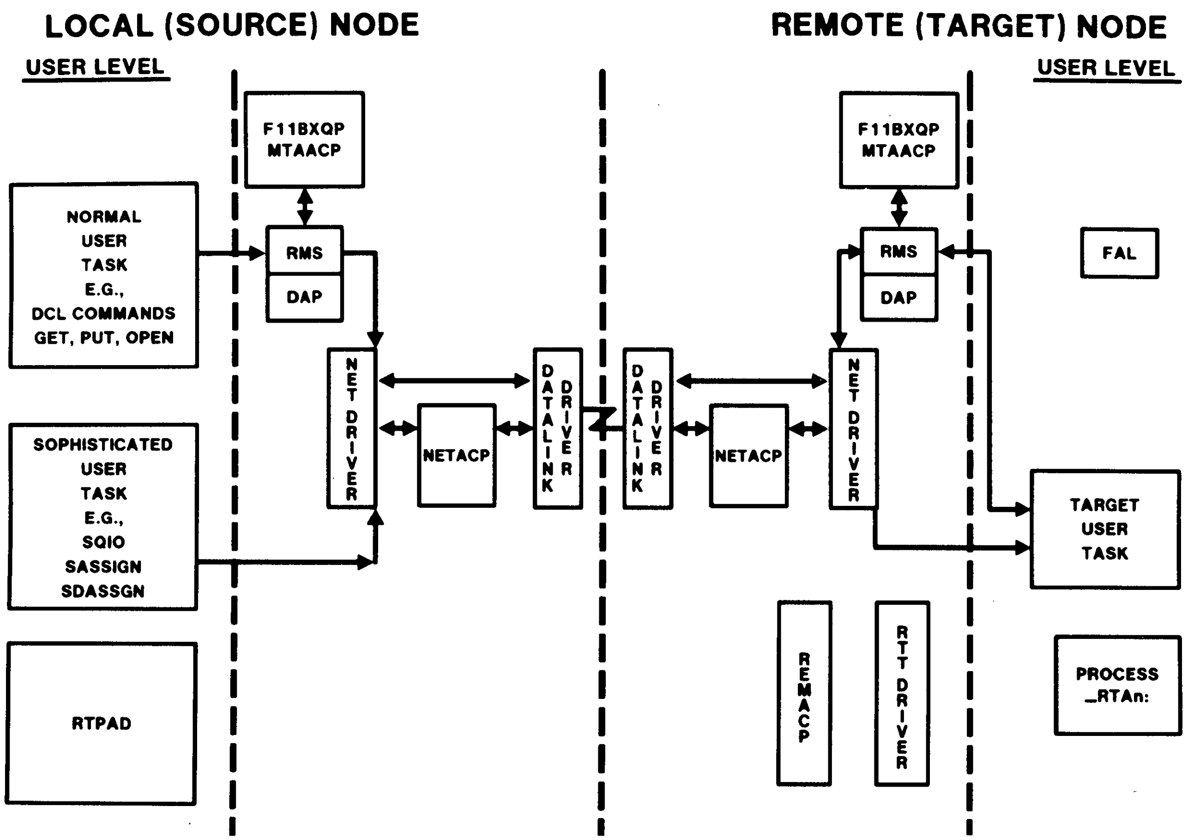


Figure 25 DECnet Task-to-Task Communication

Transparent Task-to-Task Communication

- For example, on the source node, the user issues:

  $DEF XXX NODEB"""USERID PASSWORD"""::"""TASK=YYY"""

  and in the program:

  OPEN (NAME=XXX ....)

- The OPEN command is passed to RMS.
- RMS checks the translation and sets up a logical link with the remote program YYY.
- The procedure is similar to remote file access with the following differences:
  - The command procedure YYY.COM must reside on the directory of USERID on NODEB (SYS$LOGIN).
  - The remote program uses the logical name SYS$NET to accept connection.

    for example, OPEN (NAME=SYS$NET ....)
  - The two programs must cooperate. For example, when one program issues a Read, the other issues a Write.

Nontransparent Task-to-Task Communication

- Bypass RMS and issue $QIOs directly to the NETDRIVER.

DECnet Performing Set Host Operation

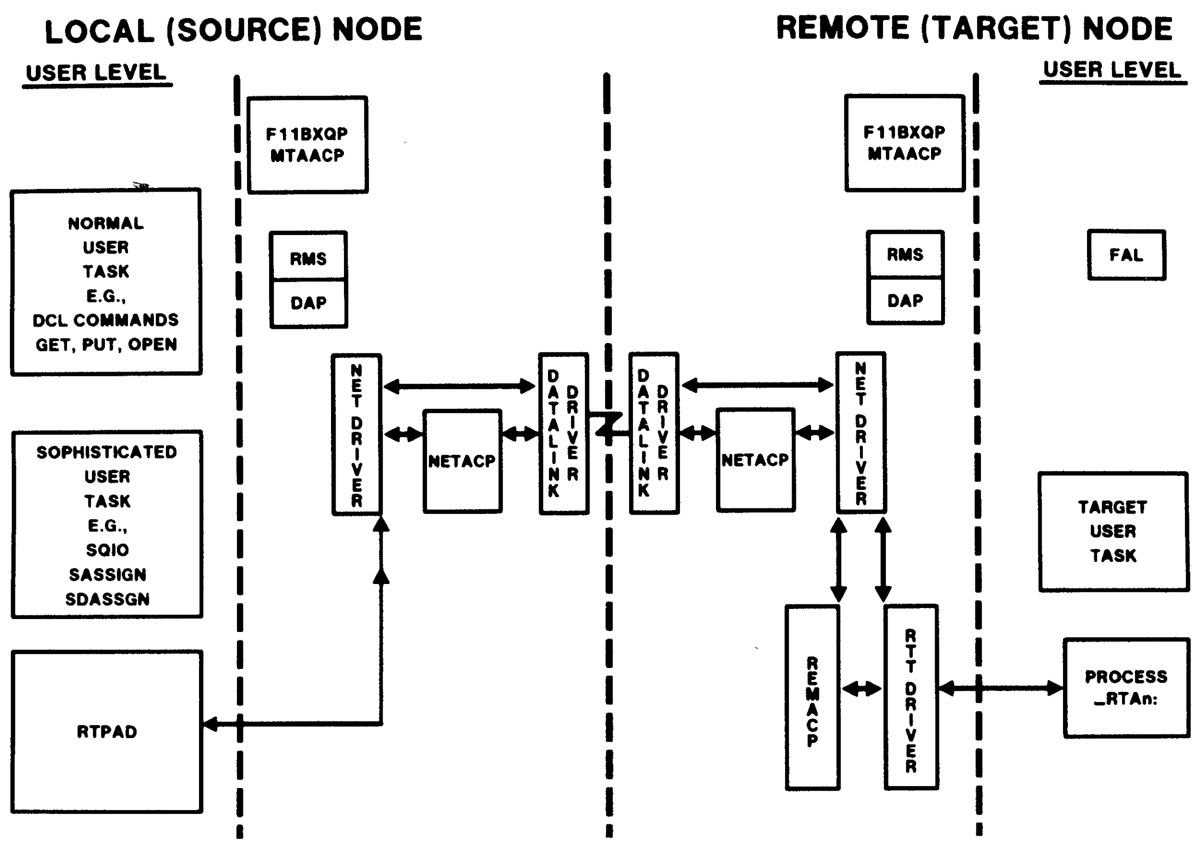


Figure 26 Performing Set Host Operation

- $SET HOST invokes RTPAD program
- Process is created on remote system to handle requests
- Local terminal appears to be connected to remote system

The Process

INTRODUCTION

This module details a familiar part of VAX/VMS: the process. The definition of a process is fundamental to understanding the operating system. The process is the representation of each user of the system. Several of the software components of the system itself are also processes.

The process is the basic scheduling entity of VAX/VMS. A group of one or more processes forms the basic accounting entity of VAX/VMS: the job. Some features and resources are only defined for each process, while others are shared among all the processes in a job. Three major classes of attributes and resources can define a process and the operations performed within it.

- Hardware process context (GPR's; M/m reg)
- Software process context (PCB/PHD/JIB)
- Virtual address space (and associated memory management data)

Hardware context includes the contents of the hardware processor registers that contain perprocess values (separate from system-wide ones). Examples of these registers include:

- The general-purpose registers (R0 through R11)
- The frame pointer (FP) [R13], argument pointer (AP) [R12], the four perprocess stack pointers (KSP,ESP,SSP,USP), and the current stack pointer (SP) [R14]
- The processor status longword (PSL) and the program counter (PC)
- Hardware registers that define the state of the AST queue and the locations and sizes of the process page tables.

PR$_ASTLVL
0 KAST
1 EAST
2 SAST
3 VAST
4 - no AST's pending

Software context defines the resources and attributes used by the VAX/VMS software but not used by the VAX-11 hardware. Examples of this type of information include:

- Resource quotas, privileges, and accumulated accounting values
- Scheduling or software priority
- Link fields to operating system data structures and queues
- Identification fields such as user name, UIC, process name, and process ID.

Virtual address space includes the mapping information for, and the contents of, the perprocess address regions, the program (or P0) region, and the control (or P1) region. In addition, all processes implicitly share the system region. Software executing in any of the three address regions, but using the hardware and software context of a process is said to be "executing in the context of the process." Software components using only system address space and the interrupt stack execute in system context (outside process context). Examples include interrupt service routines and device drivers.

OBJECTIVES

1. Describe the similarities and differences of system context and process context.

2. Using the System Dump Analyzer on either a crash dump file or the current system, examine and interpret the software process control block, process header, job information block, and control region of a specified process.

3. Describe how the various process data structures are used.

   - When the structures are modified

   - Which structures are reset to default or initial values

4. Discuss the SYSGEN parameters that relate to process characteristics, and the effects of altering those parameters.

RESOURCES

Reading

- VAX/VMS Internals and Data Structures, system overview, chapters on use of listing and map files, and naming conventions.

Additional Suggested Reading

- VAX/VMS Internals and Data Structures, chapters on executive data areas, data structure definitions, and size of system virtual address space.
- VAX/VMS System Dump Analyzer Reference Manual

Source Modules

Facility Name	Module Name
SYS	SHELL
	SYSIMGACT
	SYSBOOT
	SCHED
	PAGEFAULT
	SWAPPER
	SYS.MAP

TOPICS

I. Process vs. System Context

II. Process Data Structures Overview

   A. Software context information

   B. Hardware context information

III. Virtual Address Space Overview

   A. SØ space (operating system code and data)

   B. PØ space (user image code and data)

   C. P1 space (command language interpreter, process data)

IV. SYSGEN Parameters Related to Process Characteristics

PROCESS VS. SYSTEM CONTEXT

Process Context

- Software Context, including
  - Privileges
  - Quotas
  - Scheduling priority
  - IDs (user name, UIC, Process ID)
- Hardware Context, including
  - General Purpose Registers (RØ- R11, AP, FP, PC)
  - Stack pointers (4)
  - Processor Status Longword (PSL)
- Virtual Address Space
  - Program region (PØ)
  - Control region (P1)
  - System region (SØ)

System Context

- System virtual address space (SØ)
- The interrupt stack

PROCESS DATA STRUCTURES OVERVIEW

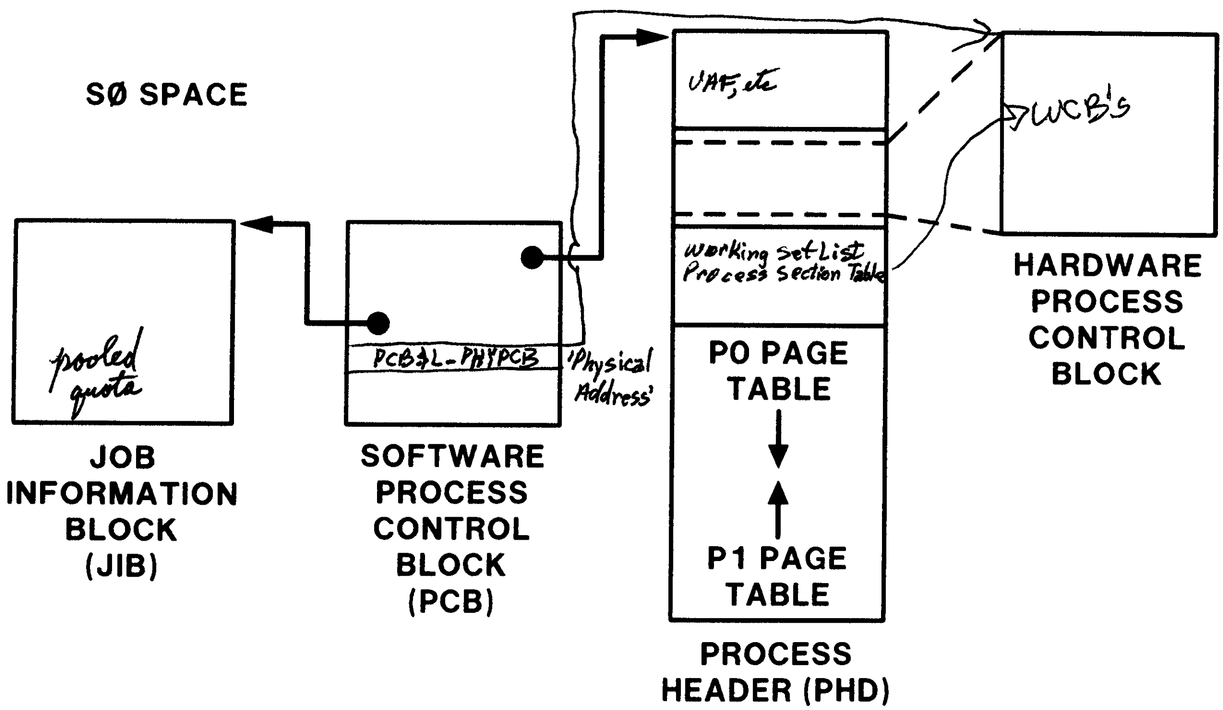


Figure 1 Process Data Structures

- 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
- Hardware Process Control Block
  - Contains saved hardware context
- Job Information Block (JIB)
  - Keeps track of resources for a detached process and all its subprocesses.

Software Process Control Block (PCB) (pg 922 IDSM)

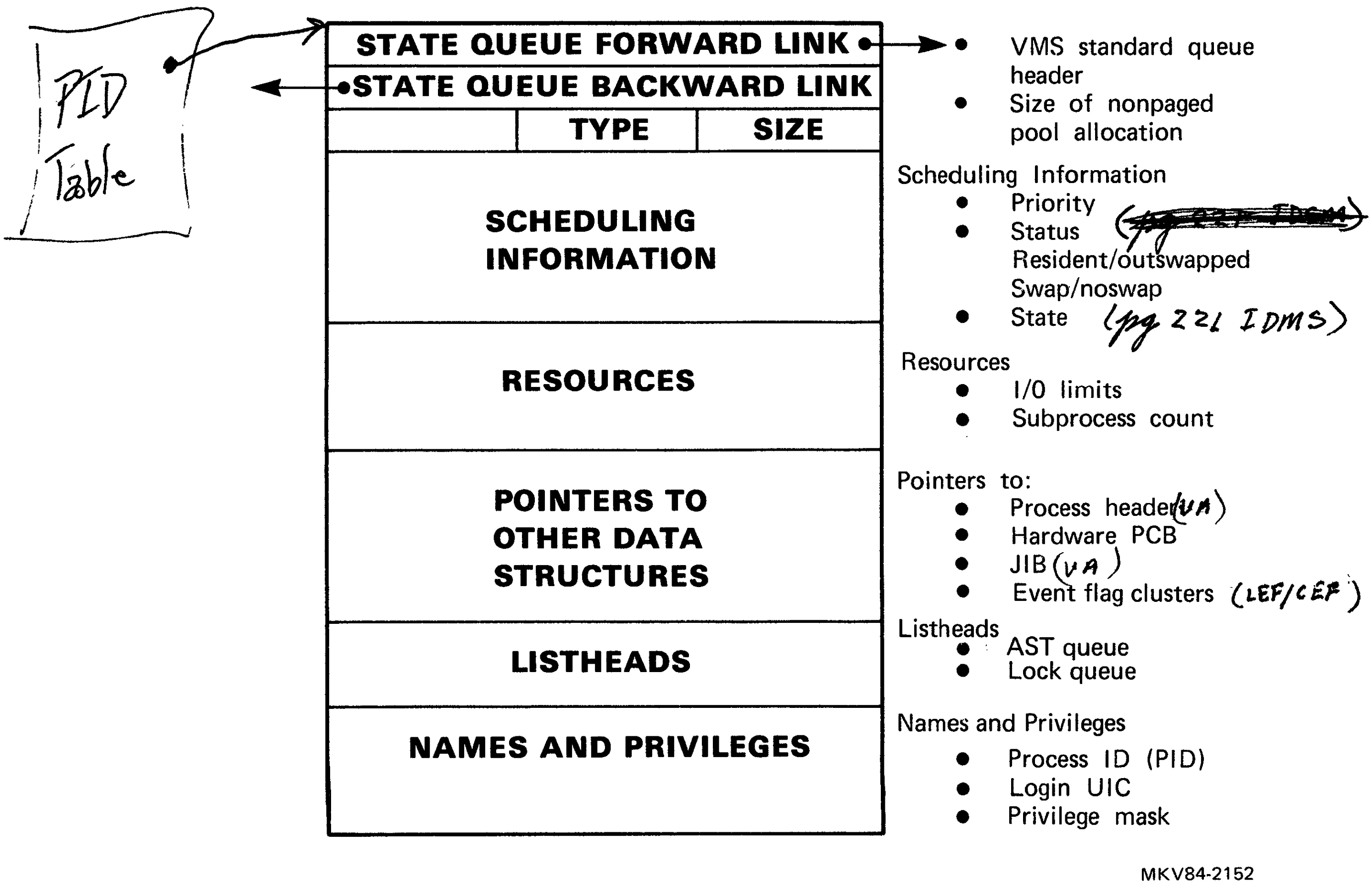


Figure 2 Software Process Control Block (PCB)

	where	pageable	swappable	
(Sfwe) PCB	NP pool	No	No	
PHD	Balance Slots	Yes / No	Yes / Yes	m/m info / HW PCB
JIB	NPP	No	No	

(pg 924 IDSM) swappable

Process Header (PHD)

SWP$GL_BALBASE

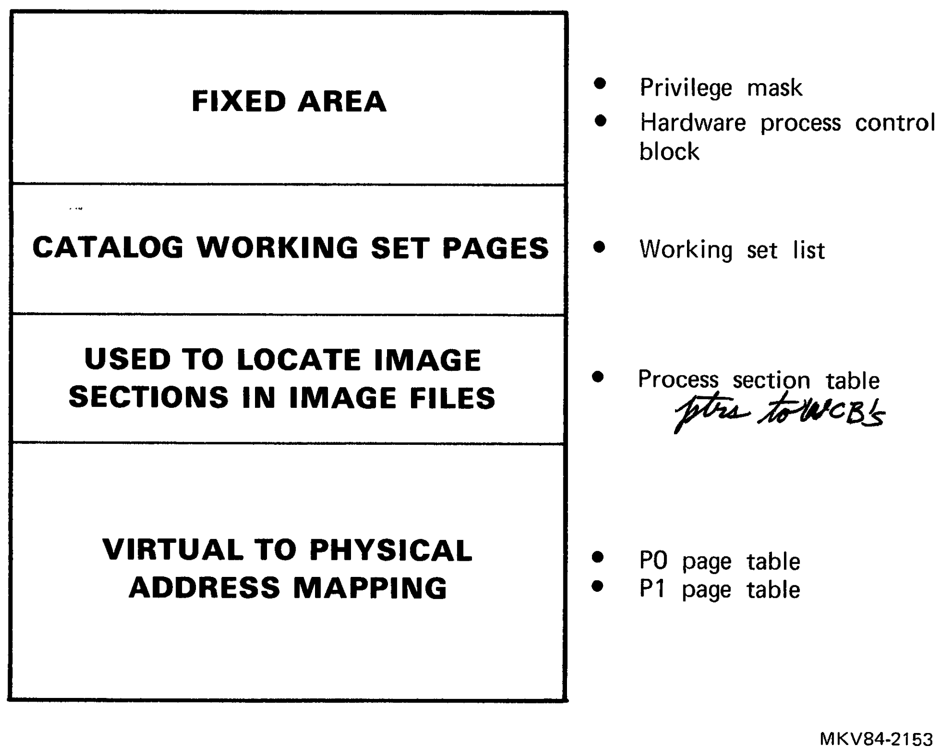


Figure 3 Process Header (PHD)

more balance slots
=> large SPTE's

Hardware Process Control Block

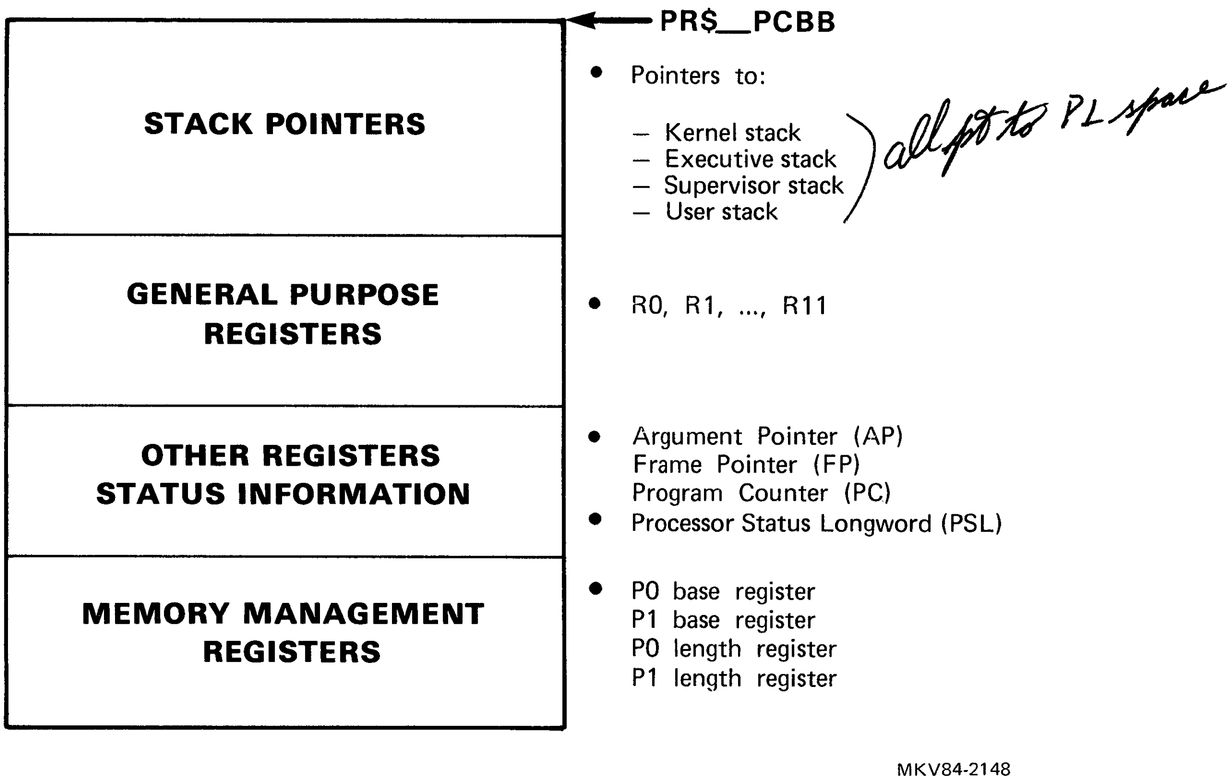


Figure 4 Hardware Process Control Block

- PR$_PCBB contains the physical address of the hardware PCB for the current process.

Privileged vs. General Registers

Privileged

- Can only be accessed in kernel mode using MTPR, MFPR instructions

- Types:

  Pointers to Data Structures

  Hardware Process Control Block (PR$_PCBB)
  System Control Block Base (PR$_SCBB)

  Hardware Error Registers

  SBI Error on VAX-11/780 (PR$_SBIER)
  Cache Error on VAX-11/750 (PR$_CAER)

  Clock Registers

  Time of Year on VAX-11/730 (PR730$_TODR)
  Interval Count on VAX-11/780 (PR780$_ICR)

  Other Registers

  Interrupt Priority Level (PR$_IPL)
  Software Interrupt Summary (PR$_SISR)

General

- Can be accessed in any access mode using most instructions

- R0-R11,AP,FP,SP,PC

Job Information Block

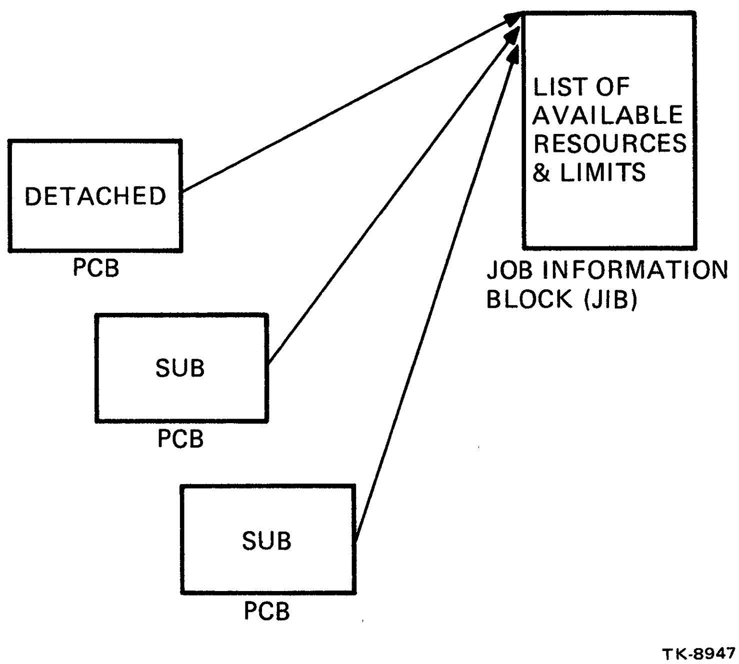


Figure 5 Job Information Block (JIB)

- Job consists of a detached process and its subprocesses.
- Job information block (JIB) keeps track of resources allotted to a job, such as:
  - Limit on number of subprocesses (PRCLIM)
  - Open File Limit (FILLM)

VIRTUAL ADDRESS SPACE OVERVIEW

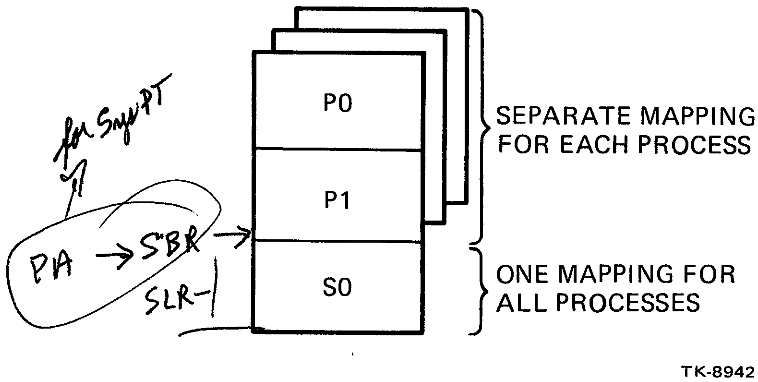


Figure 6 Virtual Address Space

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, Record Management Services, other executive code and data

S0 Virtual Address Space

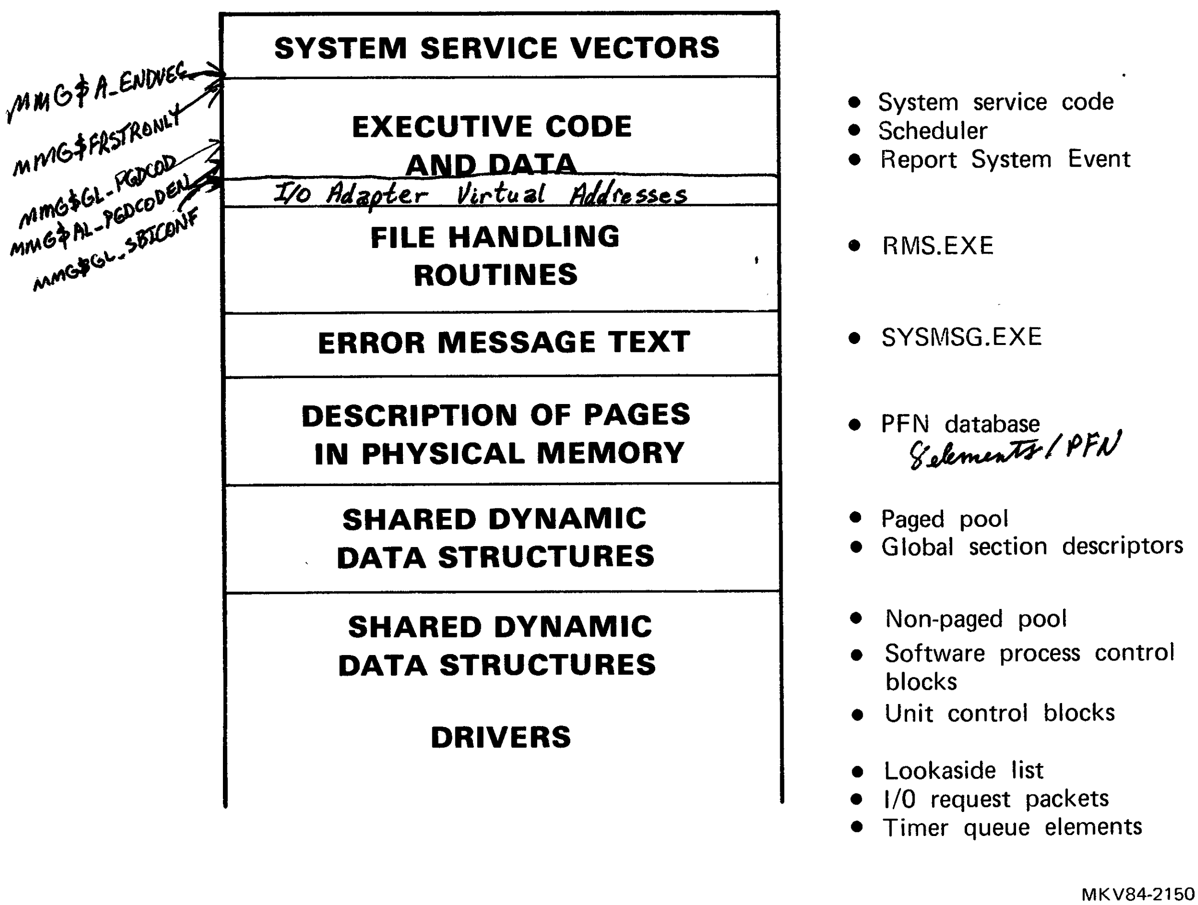


Figure 7 S0 Virtual Address Space - Low Addresses

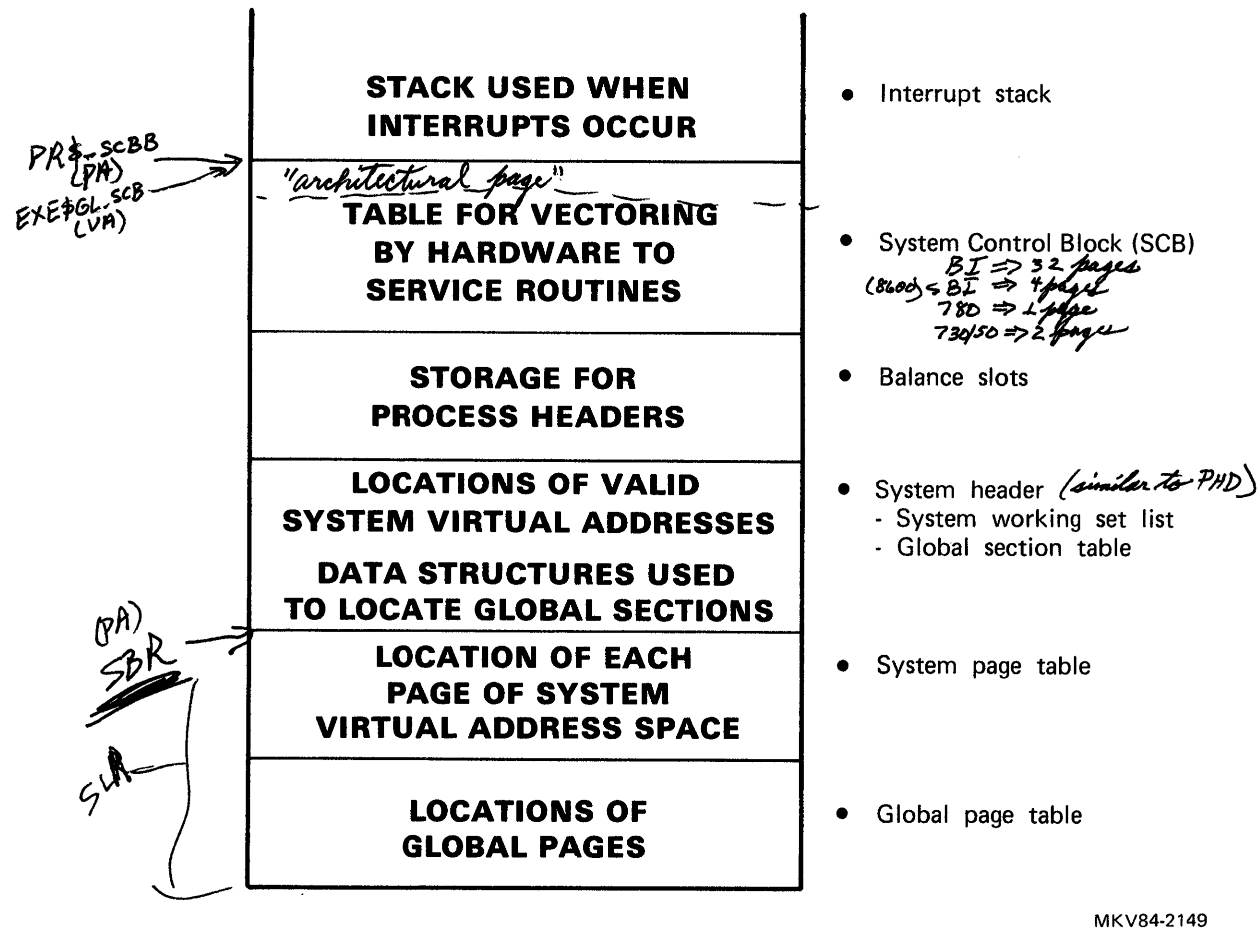


Figure 8 S0 Virtual Address Space - High Addresses

P0 Virtual Address Space

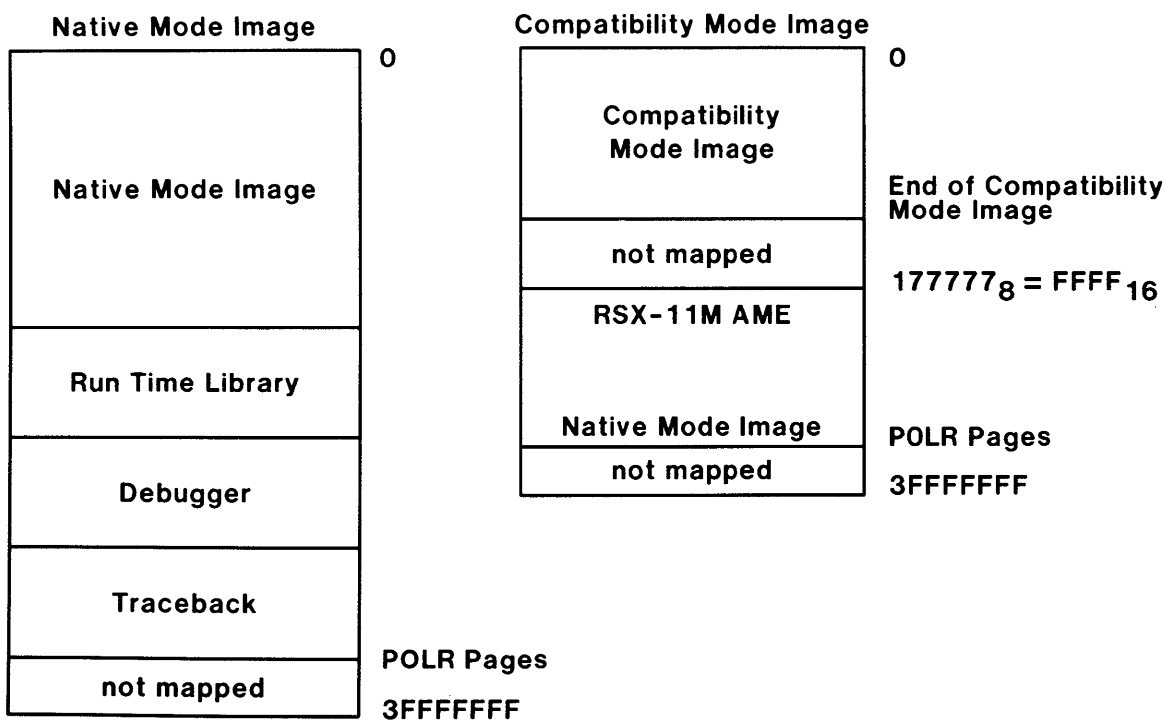


Figure 9 PØ Virtual Address Space

P1 Virtual Address Space

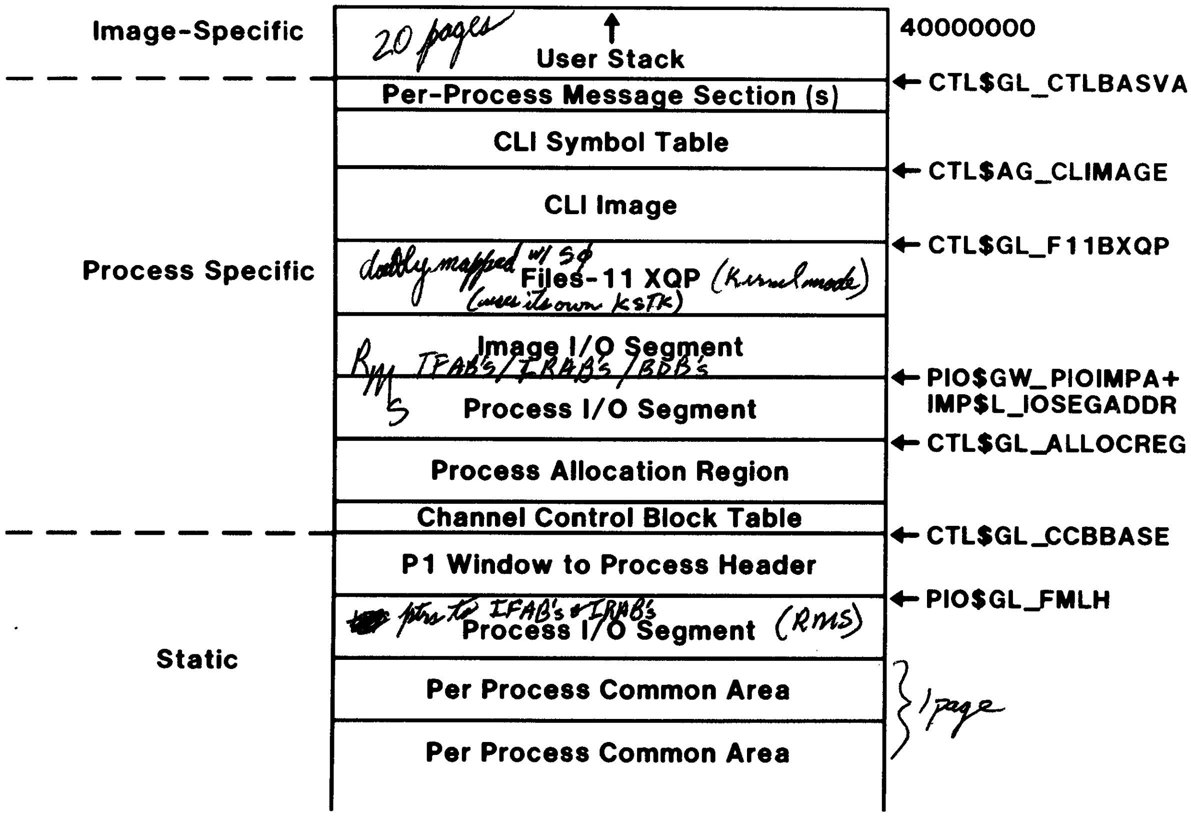


Figure 10 P1 Virtual Address Space - High Addresses

P1 space is built from high addresses toward low addresses.

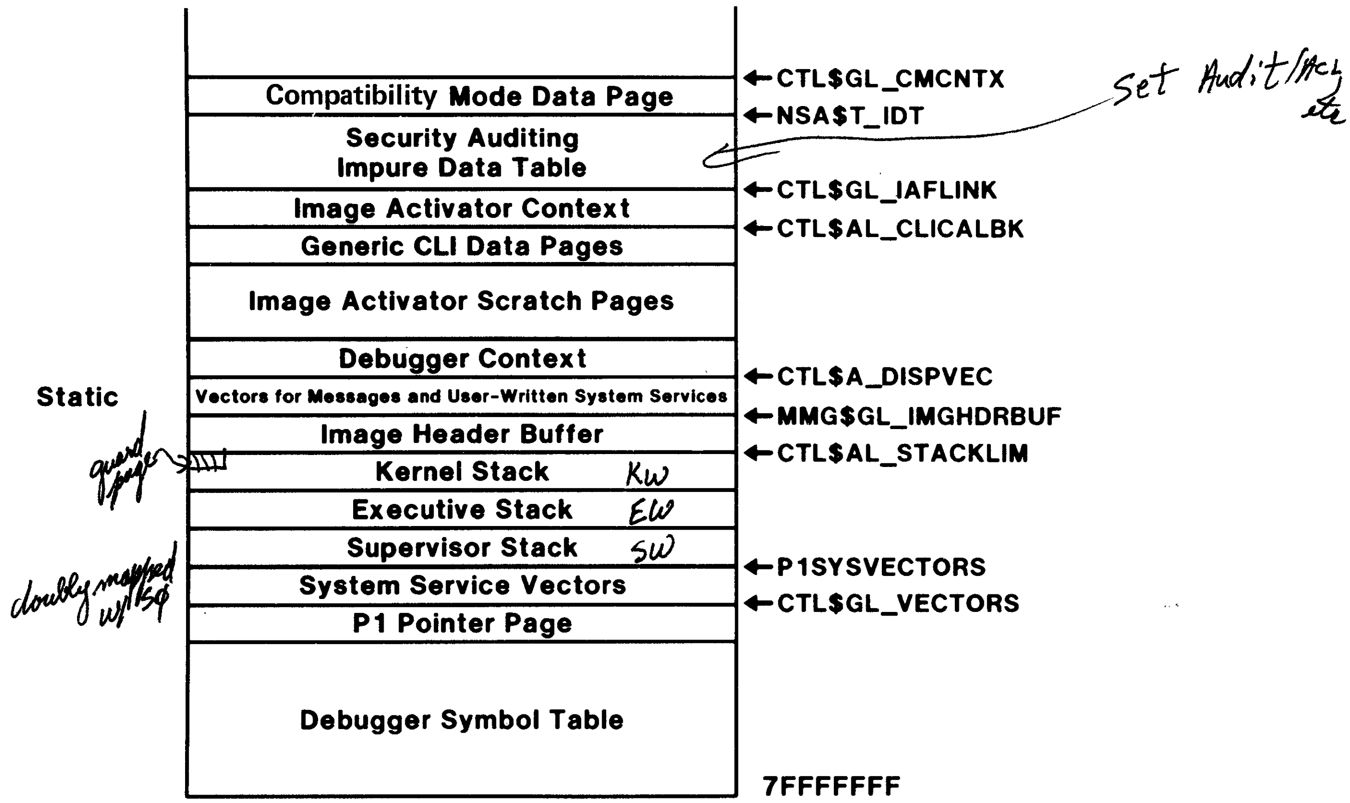


Figure 11 P1 Virtual Address Space - Low Addresses

Image-Specific - Deleted on image exit
Process-Specific - Changes according to SYSGEN parameters and CLI used
Static - Does not change

Table 1 Function of P1 Space

Function	P1 Area
Images	Command Language Interpreter (DCL, MCR, user-written)
Symbol tables	Symbolic Debugger Command Language Interpreter
Pointers	System service vectors User-written system service vectors P1 window to process header (maps to PHD in S0 space) P1 pointer page (i.e., CTL$GL_CTLBASVA; addresses of exception vectors) Perprocess message vectors
Stacks	Kernel, executive, supervisor, user
RMS data	Image I/O segment Process I/O segment
File system code	Files-11 XQP
Error message text	Perprocess message section
Storage area	
• Data stays around between images	Perprocess Common Area (LIB$GET_COMMON)
• Logical names	Process allocation region
Other data areas	Generic CLI data pages Image activator scratch pages Image header buffer Compatibility mode data page (used by AME) Channel control block table (links process to device)

SUMMARY

Table 2 SYSGEN Parameters Relevant to Process Structure

Function	**Parameter**
Size of the CLI symbol table	CLISYMTBL
Limit on use of process allocation region by images	CTLIMGLIM (*)
Number of pages in the process allocation region	CTLPAGES (*)
Default number of pages created by the image activator for the image I/O segment	IMGIOCNT (*)
Number of pages for the process I/O segment mapped by PROCSTRT	PIOPAGES (*)

(*) = special SYSGEN parameter

System Mechanisms

INTRODUCTION

Many of the operations associated with an operating system can be described in terms of software components manipulating data structures. A variety of control mechanisms must be established to ensure that components competing for common resources do not interfere with each other or cause a system "deadlock." Several hardware instructions provide support for these software mechanisms. Additional mechanisms control the accessibility of data structures.

The implementation of an interrupt priority structure provides a hardware-arbitrated mechanism for synchronizing device requests, some software component requests (such as scheduling and AST delivery), and synchronizing the accessibility of some protected data structures. Interrupts are the result of asynchronous events occurring within VMS and the hardware configuration.

Available mechanisms for synchronizing the activities of processes include:

- Interrupt Priority Levels (IPL)
- The System Timer Queue
- Mutual Exclusion Semaphores (MUTEXes)
- Asynchronous System Traps (ASTs)
- The VAX/VMS Lock Manager

Exceptions are another mechanism used by VMS. Exceptions are synchronous events that result from actions within a particular process. Common examples include:

- Translation-not-valid fault (page fault)
- Divide-by-zero trap

Execution of most system services and record management services occurs as a result of change mode to kernel and change mode to executive exceptions (CHMK and CHME instructions).

Dynamic memory (pool) is used to provide storage for various classes of VMS data structures. Process data structures are allocated from a dynamic memory area in the control (P1) region. System-wide data structures are allocated from either paged or nonpaged pools depending on the types of system components accessing them.

OBJECTIVES

To understand the operations of VMS, and to write system-level programs, the student must be able to:

1. Describe how the various VAX/VMS protection, communication, and synchronization mechanisms are implemented, and why each of them is used.

2. Discuss the SYSGEN parameters controlling various system resources (for example, memory), and the effects of altering those parameters.

RESOURCES

Reading

- VAX/VMS Internals and Data Structures, chapters on condition handling, system service dispatching, software interrupts, AST delivery, the lock manager, synchronization techniques and dynamic memory allocation.

Additional Suggested Reading

- VAX/VMS Internals and Data Structures, chapters on hardware interrupts, and timer support
- VAX-11 Architecture Handbook, chapters on special instructions, and exceptions and interrupts
- VAX-11 Hardware Handbook, chapters on privileged registers

Source Modules

Facility Name	Module Name
SYS	ASTDEL,SCHED CMODSSDSP EXCEPTION,SYSUNWIND MEMORYALC MUTEX SYSENQDEQ TIMESCHDL SYSSCHEVT,SYSCANEVT FORKCNTRL IOCIOPOST
SYS$EXAMPLES	USSDISP.MAR,USSLNK.COM USSTEST.MAR,USSTSTLNK.COM
Macros	IFWRT,IFNOWRT,IFRD,IFNORD IFPRIV,IFNPRIV SETIPL,DSBINT,ENBINT,SAVIPL
RTL	LIBSIGNAL

TOPICS

I. Hardware Register and Instruction Set Support

II. Synchronizing System Events

- Hardware Interrupts
- Software Interrupts
  Example: Fork Processing
- Requesting Interrupts
- Changing IPL
- The Timer Queue and System Clocks

III. Process Synchronization Mechanisms

- Mutual Exclusion Semaphores (MUTEXes)
- Asynchronous System Traps (ASTs)
- VAX/VMS Lock Manager

IV. Exceptions and Condition Handling

V. Executing Protected Code

- Change Mode Dispatching
- System Service Dispatching

VI. Miscellaneous Mechanisms

- System and Process Dynamic Memory (Pool)

VII. SYSGEN Parameters Controlling System Resources

HARDWARE REGISTER AND INSTRUCTION SET SUPPORT

Table 1 Keeping Track of CPU, Process State

Function	Implementation	Name
Store processor state	Register	Processor Status Longword (PSL)
Save, restore process state	Instruction	SVPCTX, LDPCTX

Processor Status Word

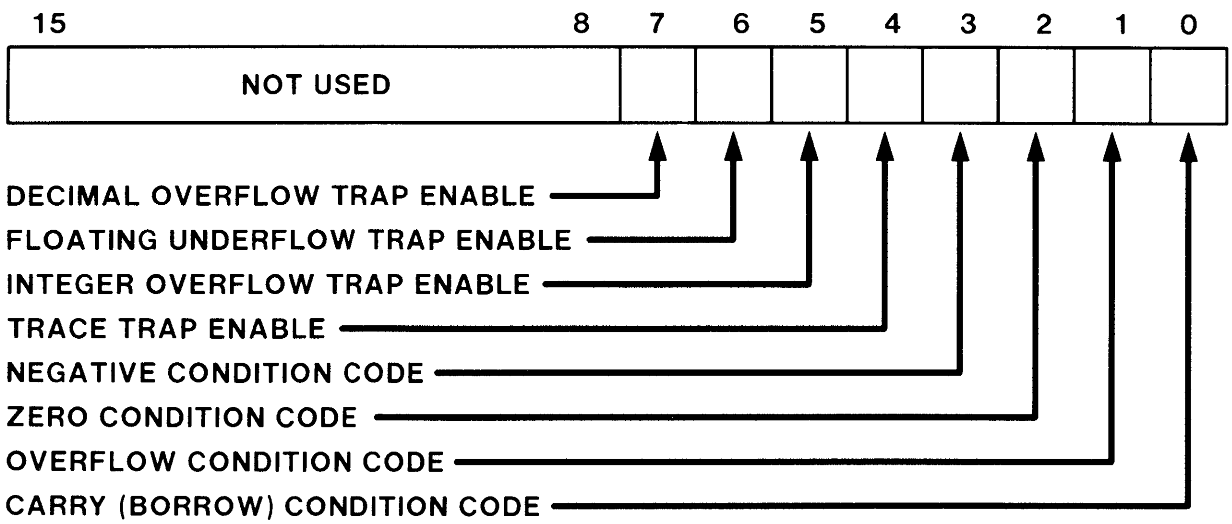


Figure 1 Processor Status Word

- Low-order word of Processor Status Longword (PSL)
- Writable by nonprivileged users through:
  - Special Instructions
  - Entry masks
  - Results of most instructions

can't write (MOVL)
can read (MOVPSL)

Processor Status Longword (PSL)

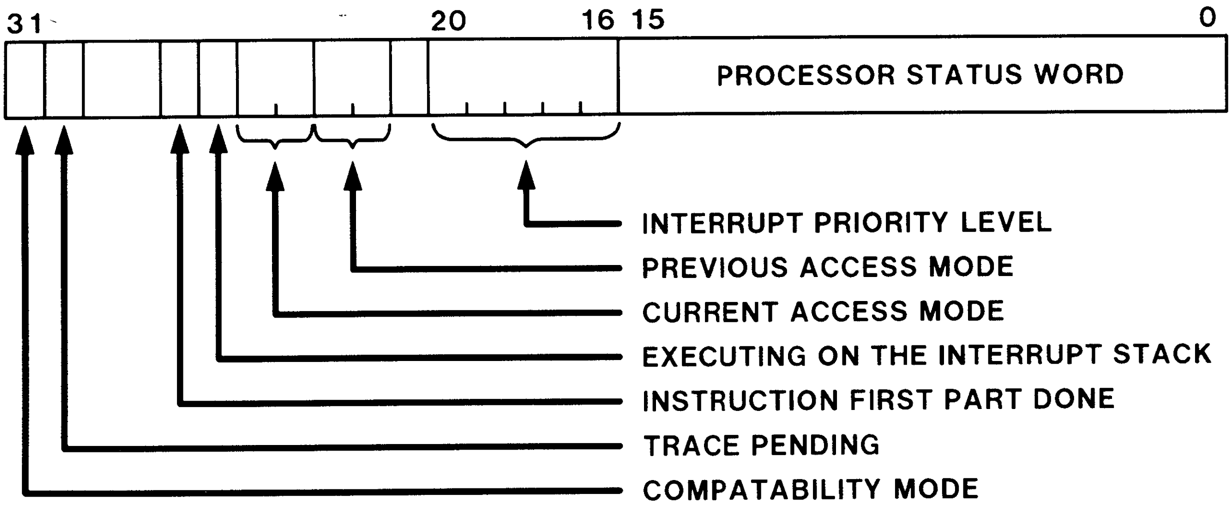


Figure 2 Processor Status Longword (PSL)

- High-order word of most interest to system programmers
  - Contains processor status information
  - Read-only to nonprivileged users
  - Changed as a result of REI and MTPR instructions
  - May be changed as a result of interrupts and exceptions
- PSL is part of process hardware context

entry mask
<15> DV bit - decimal overflow ena
<14> IV bit integer " "

Hardware Context

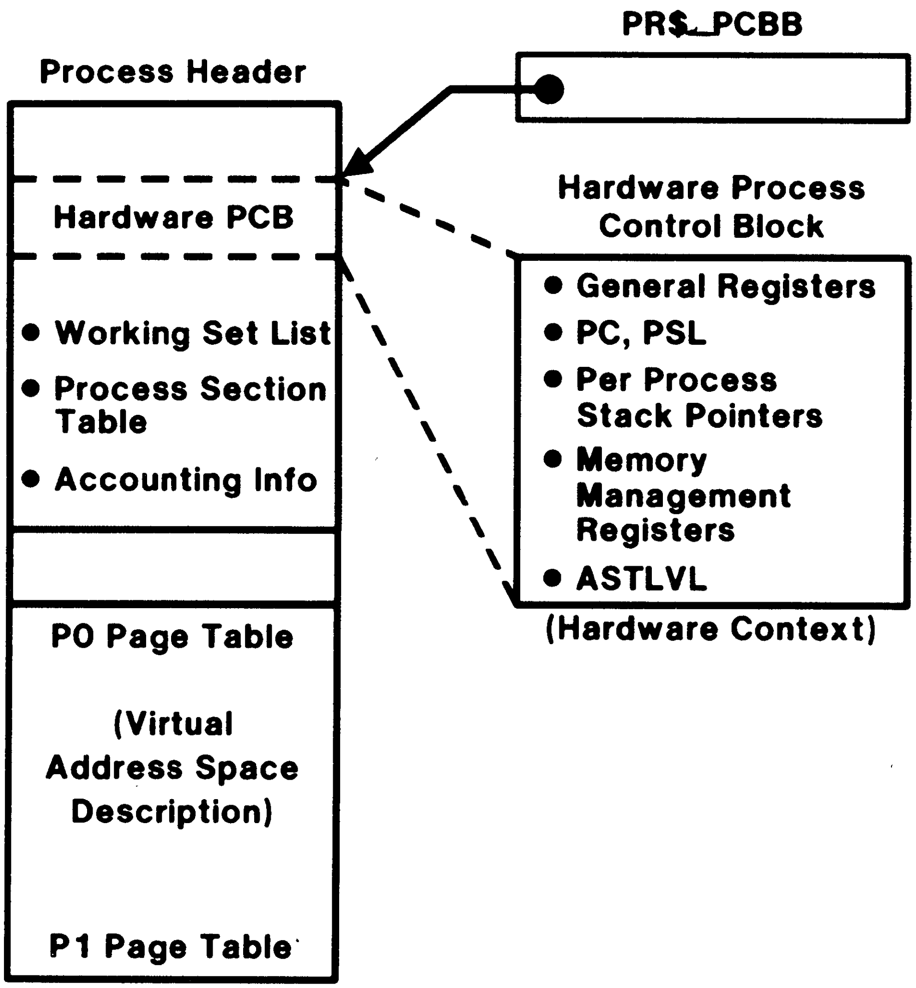


Figure 3 Hardware Context

- Hardware PCB contains hardware context while process not current
- VAX instructions for saving and restoring hardware context (SVPCTX and LDPCTX)

SYNCHRONIZING SYSTEM EVENTS

Hardware Interrupts and the SCB

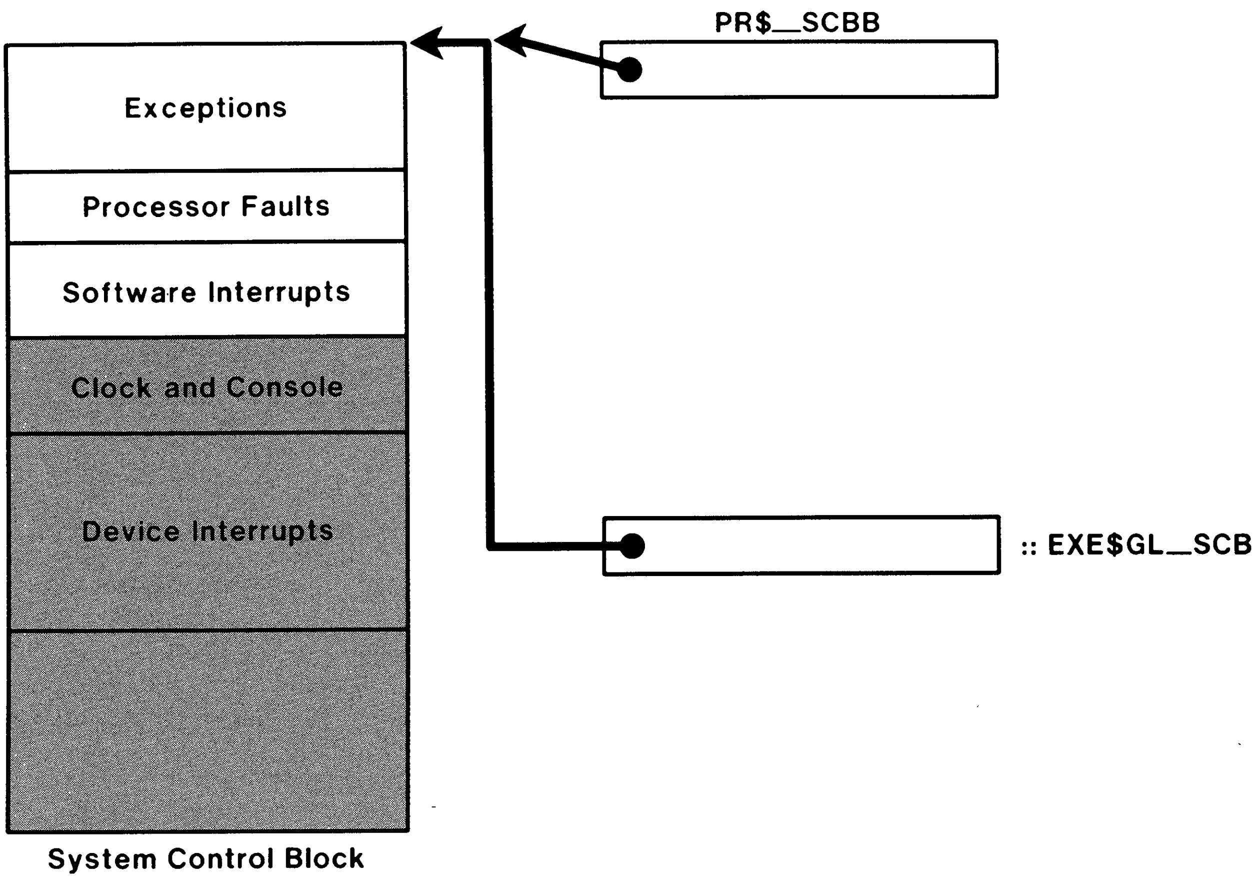


Figure 4 Hardware Interrupts and the SCB

- System Control Block (SCB) - physically contiguous area of system space
- Hardware register PR$_SCBB contains physical address of SCB
- Hardware gets service routine address from longword in SCB
- Size of SCB is CPU-specific.

Hardware Interrupts and IPL

Table 2 Hardware Interrupts and IPL

FUNCTION	VALUE (decimal)	NAME
Power Fail Interrupt	30	
Clock Interrupts	24	IPL$_HWCLK
Device Interrupts	20-23	UCB$B_DIPL*

*** Offset into Device's Unit Control Block**

- Interrupt Priority Levels (IPLs) above 15 reserved for hardware interrupts
- Peripheral devices interrupt at IPL 20 to 23
- IPL$_xxxx - IPL level (see $IPLDEF)

Software Interrupts and the SCB

uses PR$_SIRR

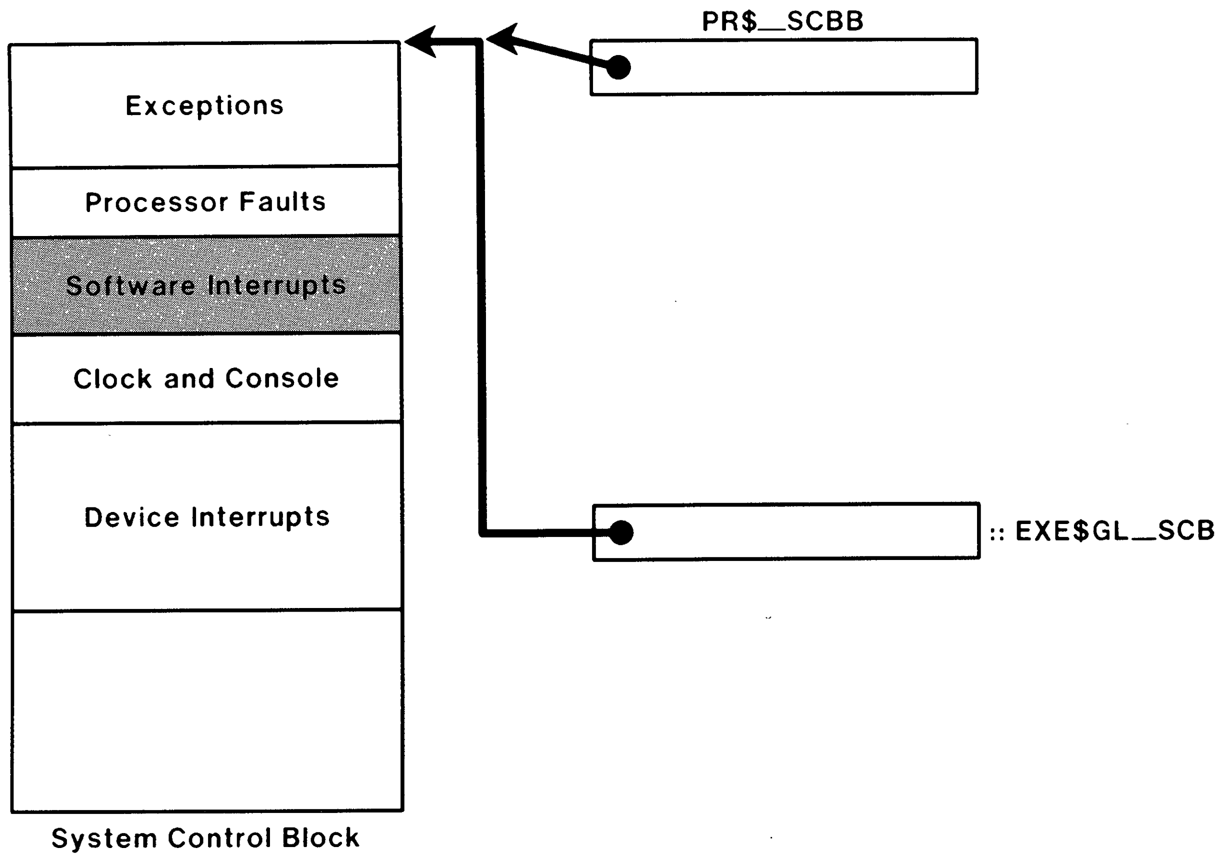


Figure 5 Software Interrupts and the SCB

- Hardware gets service routine address from longword in SCB.

Software Interrupts and IPL

Table 3 Software Interrupts and IPL

FUNCTION	VALUE (decimal)	NAME
(unused)	15-12	
Fork Dispatching	11	IPL$_MAILBOX
Fork Dispatching	10	
Fork Dispatching	9	
Fork Dispatching	8	IPL$_TIMER
		IPL$_SYNCH
Software Timer Interrupt	7	IPL$_TIMERFORK
Fork Dispatching	6	
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 reserved for software interrupts
- Driver fork level stored at offset UCB$B_FIPL in UCB (see $UCBDEF)

Example of Fork Processing

1. IPL 23 interrupt occurs

2. Driver interrupt service routine executes

   - Processing done at IPL 23
   - Queue 'context block' (UCB) to fork dispatcher (block contains PC)
   - Request IPL 8 interrupt
   - Continue processing at IPL 23
   - REI when done at IPL 23

3. IPL 8 interrupt is recognized

4. Fork dispatcher service routine executes

   - If queue empty, REI
   - Dequeue UCB
   - JSB to PC in UCB

     PC is usually in driver code
     Routine exits with RSB when done

   - Loop back

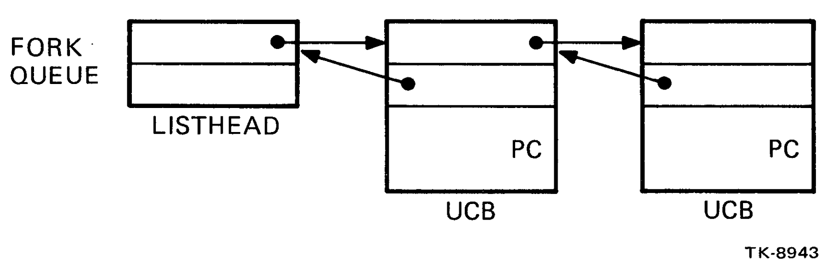
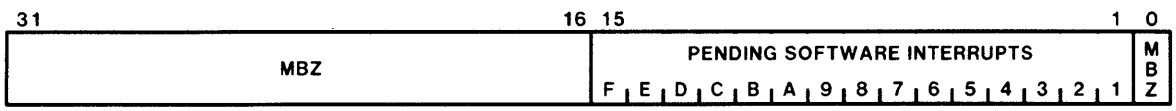


Figure 6 Fork Queue

Software Interrupt Requests

PR$_SIRR **Software Interrupt Request Register (Write Only)**



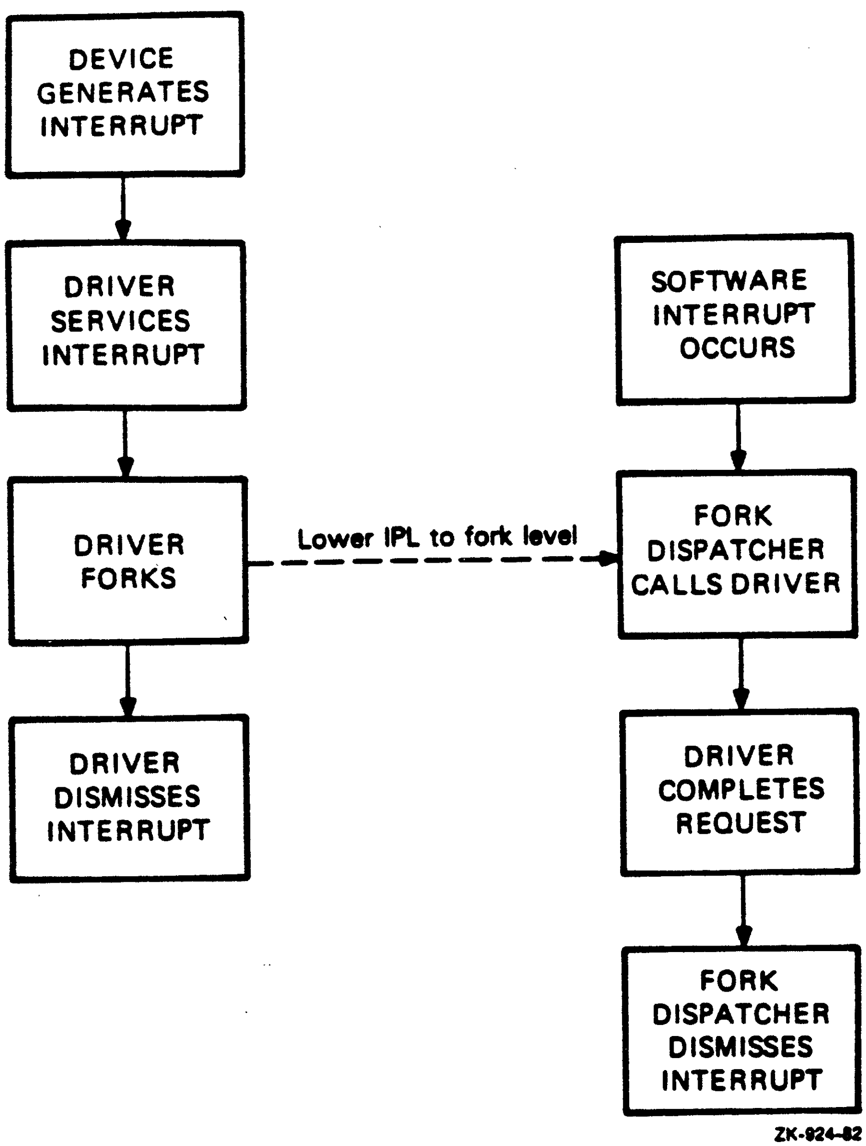
PR$_SISR **Software Interrupt Summary Register (Read/Write)**

Figure 7 Software Interrupt Requests

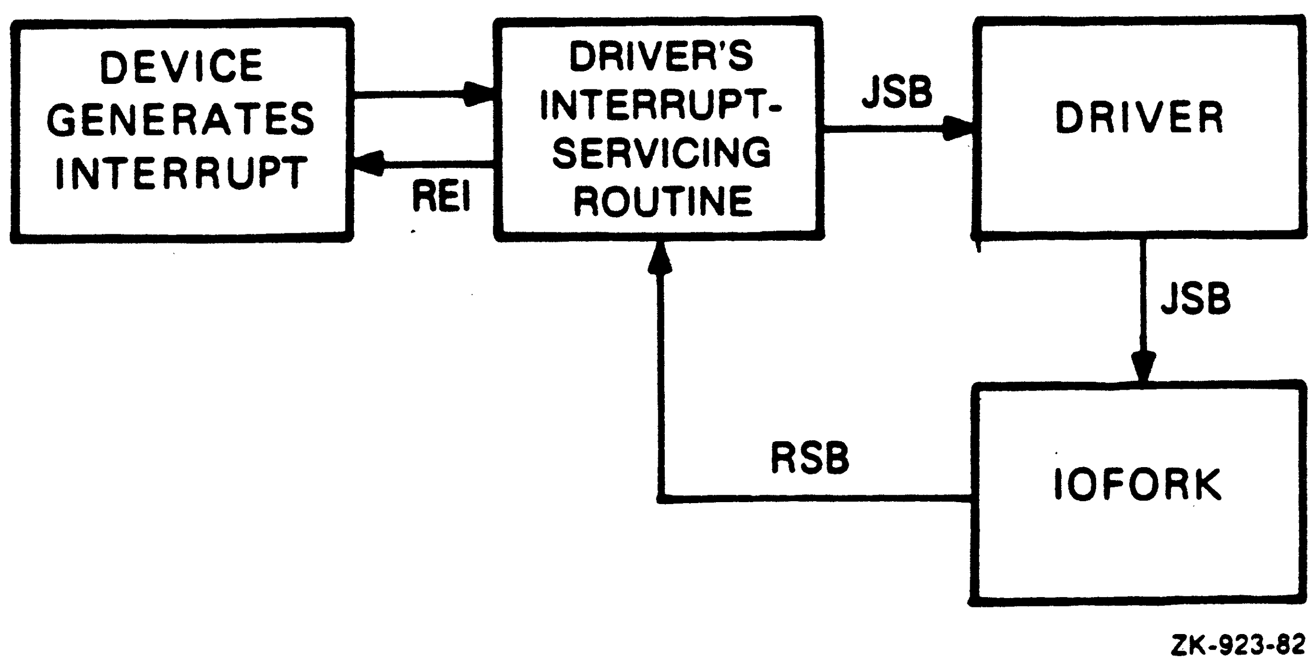
- Software Interrupt Summary Register
  - Bits 1 through 15 correspond to IPLs 1 through 15.
  - Bit set indicates pending software interrupt request.
  - Interrupt is serviced as IPL drops below specified level, when REI is issued.
- Software Interrupt Request Register
  - To set bit in SISR, write IPL value to SIRR.
  - Use SOFTINT macro:

```
.MACRO  SOFTINT  IPL
        MTPR     IPL,S^#PR$_SIRR
.ENDM   SOFTINT
```

Reactivation of a Driver Fork Process



Creating a Fork Process After



from Interrupt to Fork Process Context

To lower its priority, the driver calls a VAX/VMS fork process queuing routine (by means of the IOFORK macro) that performs the following steps:

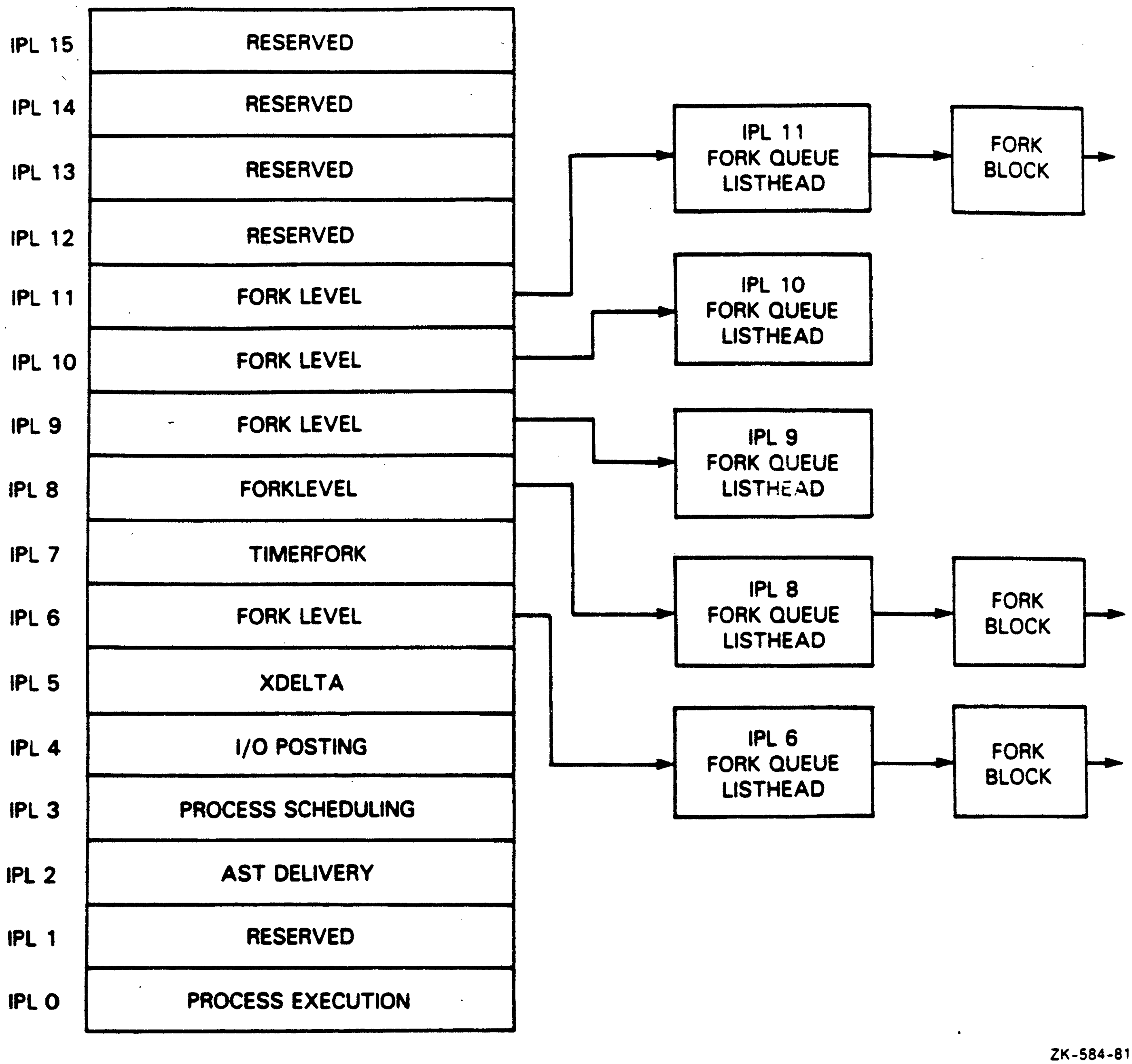
1. Disables the timeout that was specified in the wait-for-interrupt routine
2. Saves R3 and R4 (these are the registers needed to execute as a fork process) (UCBL_FR3, UCBL_FR4)
3. Saves the address of the instruction following the IOFORK request in the UCB fork block (UCB$L_FPC)
4. Places the address of the UCB fork block from R5 in a fork queue for the driver's fork level
5. Returns to the driver's interrupt-servicing routine

The interrupt-servicing routine then cleans up the stack, restores registers, and dismisses the interrupt. Figure 5–7 illustrates the flow of control in a driver that creates a fork process after a device interrupt.

Fork Block

<table>
<tr><td colspan="3">Fork Queue Forward Link</td></tr>
<tr><td colspan="3">Fork Queue Backward Link</td></tr>
<tr><td>Fork IPL</td><td>Type</td><td>Size</td></tr>
<tr><td colspan="3">Saved PC</td></tr>
<tr><td colspan="3">Saved R3</td></tr>
<tr><td colspan="3">Saved R4</td></tr>
</table>

Fork Dispatching Queue Structure



Activating a Fork Process from a Fork Queue

When no hardware interrupts are pending, the software interrupt priority arbitration logic of the processor transfers control to the software interrupt fork dispatcher. When the processor grants an interrupt at a fork IPL, the fork dispatcher processes the fork queue that corresponds to the IPL of the interrupt. To do so, the dispatcher performs these actions:

1. Removes a driver fork block from the fork queue
2. Restores fork context
3. Transfers control back to the fork process

Thus, the driver code calls VAX/VMS code that coordinates suspension and restoration of a driver fork process. This convention allows VAX/VMS to service hardware device interrupts in a timely manner and reactivate driver fork processes as soon as no device requires attention.

When a given fork process completes execution, the fork dispatcher removes the next entry, if any, from the fork queue, restores its fork process context, and reactivates it. This sequence is repeated until the fork queue is empty. When the queue is empty, the fork dispatcher restores R0 through R5 from the stack and dismisses the interrupt with an REI instruction.

Unit-Control Block (UCB)

UCB$L_FQFL*			
UCB$L_FQBL*			
UCB$B_FIPL*	UCB$B_TYPE*	UCB$W_SIZE*	
UCB$L_FPC			
UCB$L_FR3			
UCB$L_FR4			
UCB$W_SRCADDR*		UCB$W_BUFQUO*	
UCB$L_ORB*			
UCB$L_LOCKID*			
UCB$L_CRB*			
UCB$L_DDB*			
UCB$L_PID*			
UCB$L_LINK*			
UCB$L_VCB*			
UCB$L_DEVCHAR			
UCB$L_DEVCHAR2			
UCB$W_DEVBUFSIZ		UCB$B_DEVTYPE	UCB$B_DEVCLASS
UCB$L_DEVDEPEND			
UCB$L_DEVDEPND2			
UCB$L_IOQFL*			
UCB$L_IOQBL*			
UCB$W_CHARGE*		UCB$W_UNIT*	
UCB$L_IRP			
UCB$B_AMOD*	UCB$B_DIPL	UCB$W_REFC*	
UCB$L_AMB*			
UCB$L_STS			
UCB$W_QLEN		UCB$W_DEVSTS	
UCB$L_DUETIM*			
UCB$L_OPCNT*			
UCB$L_SVPN*			
UCB$L_SVAPTE			
UCB$W_BCNT		UCB$W_BOFF	
UCB$W_ERRCNT		UCB$B_ERTMAX	UCB$B_ERTCNT
UCB$L_PDT*			
UCB$L_DDT*			
reserved			

Blocking Interrupts

Table 4 Blocking Interrupts

WHAT TO BLOCK	RAISE IPL TO (decimal)	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**

- Can use IPL to block interrupt servicing
- For example, to block AST delivery, raise to IPL$_ASTDEL
- IPL$_SYNCH used to coordinate access to scheduler's database

Summary of IPL Mechanism

- IPL determines which component gets the CPU
  - IPL of interrupt determines which service routine is called
- Can alter current IPL
  - To raise, use SETIPL or DSBINT
  - To lower:

    If at original level (IPL has not been raised), request interrupt at lower level with SOFTINT, then REI

    If at elevated level, lower to original level with SETIPL or ENBINT
  - REI enforces the rules
- Altering of IPLs can be used to synchronize system routines and processes
  - Current IPL blocks interrupts at same and lower IPLs
  - Convention: Raise IPL to IPL$_SYNCH to access system-wide database (PCBs, PHDs, etc.)
  - Convention: Raise to IPL$_ASTDEL to prevent process deletion

Using IPL to Synchronize System Routines

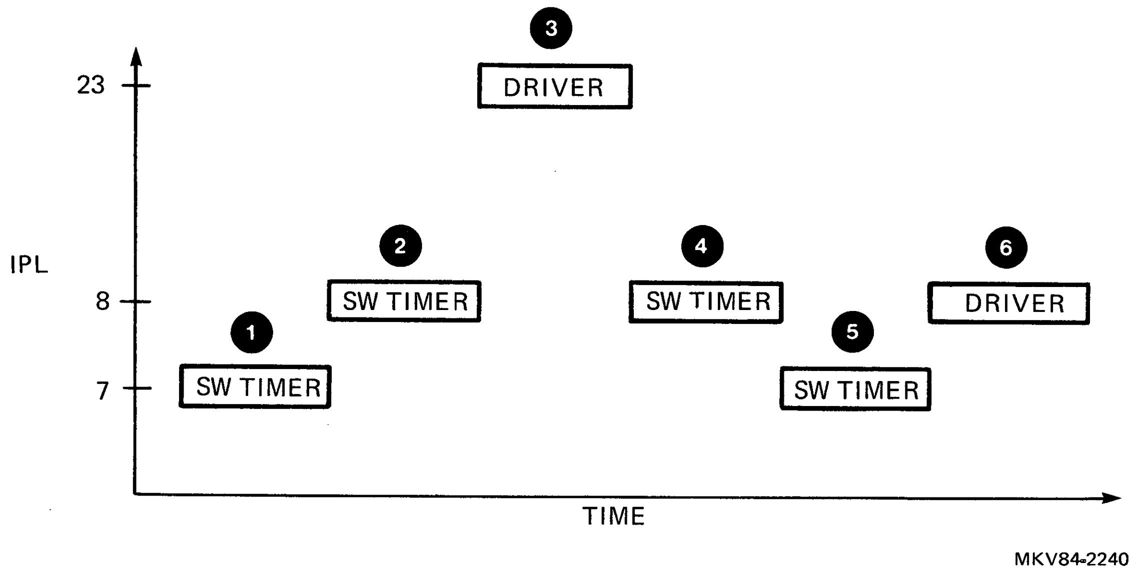


Figure 8 Raising IPL to SYNCH

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 REI
4. Software timer resumes at IPL$_SYNCH
5. Software timer lowers IPL back to IPL$_TIMERFORK
6. Driver code executes at IPL 8

System Timer Queue and System Clocks

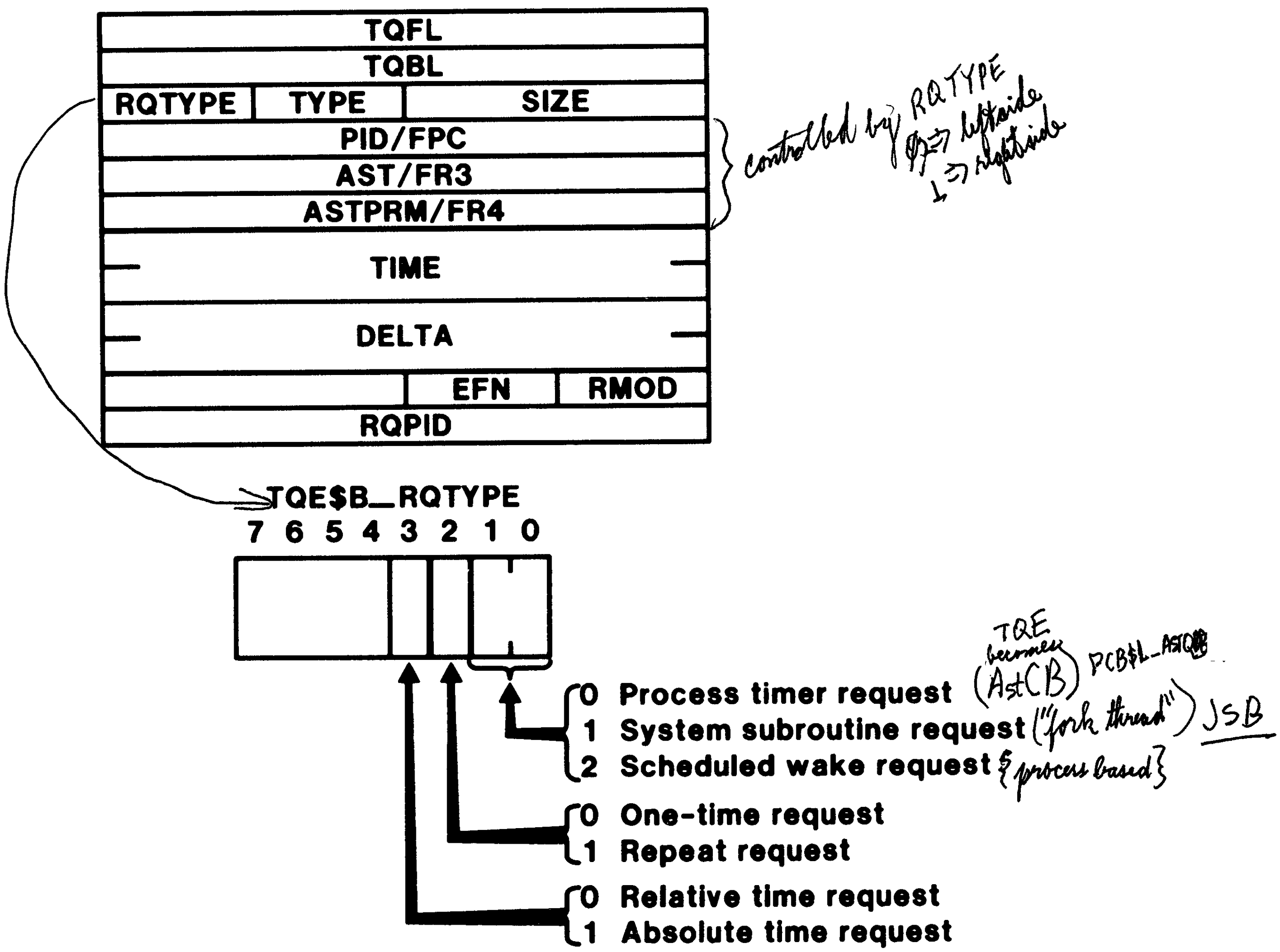


Figure 9 Timer Queue Element

- Timer queue is ordered by absolute expiration time.
- Scheduled wake-up and system subroutine requests may have a delta time specified for recurring events.
- The AST routine, AST parameter, and event flag fields are filled from the system service argument list.

```
        .SBTTL  INSERT ENTRY IN TIME DEPENDENT SCHEDULER QUEUE
;+
; EXE$INSTIMQ - INSERT ENTRY IN TIME DEPENDENT SCHEDULER QUEUE
;
; THIS ROUTINE IS CALLED TO INSERT AN ENTRY IN THE TIME DEPENDENT SCHEDU
; QUEUE. THE ENTRY IS THREADED INTO THE QUEUE ACCORDING TO ITS DUE TIME.
; THE QUEUE IS ORDERED SUCH THAT THE MOST IMMINENT ENTRIES ARE AT THE FR
; OF THE QUEUE.
;
; INPUTS:
;
;       R0 = LOW ORDER PART OF EXPIRATION TIME.
;       R1 = HIGH ORDER PART OF EXPIRATION TIME.
;       R5 = ADDRESS OF ENTRY TO INSERT IN TIME QUEUE.
;
;       IPL MUST BE IPL$_TIMER.
;
; OUTPUTS:
;
;       SPECIFIED ENTRY IS INSERTED INTO THE TIME DEPENDENT SCHEDULER QU
;       ACCORDING TO ITS DUE TIME.
;-

        .PSECT
EXE$INSTIMQ::                                   ;INSERT ENTRY IN TIME QUEUE
        MOVQ    R0,TQE$Q_TIME(R5)               ;SET ABSOLUTE DUE TIME
        MOVAL   W^EXE$GL_TQFL,R3                ;GET ADDRESS OF TIME QUEUE LISTH
        MOVL    R3,R2                           ;COPY ADDRESS OF TIME QUEUE LIST
10$:    MOVL    TQE$L_TQBL(R2),R2               ;GET ADDRESS OF NEXT ENTRY
        CMPL    R3,R2                           ;END OF QUEUE?
        BEQL    20$                             ;IF EQL YES
        CMPL    R1,TQE$Q_TIME+4(R2)             ;COMPARE HIGH ORDER PARTS OF TIM
        BLSSU   10$                             ;IF LSSU NEW ENTRY MORE IMMINENT
        BGTRU   20$                             ;IF GTRU NEW ENTRY LESS IMMINENT
        CMPL    R0,TQE$Q_TIME(R2)               ;COMPARE LOW ORDER PART OF TIME
        BLSSU   10$                             ;IF LSSU NEW ENTRY MORE IMMINENT
20$:    INSQUE  TQE$L_TQFL(R5),TQE$L_TQFL(R2) ;INSERT NEW ENTRY IN TIME
        RSB                                     ;
```

Example 3 EXE$INSTIMQ (from module EXSUBROUT)

- MAKETQE

  - Allocates two blocks from nonpaged pool
  - Places code to execute periodically in first block
  - Makes second block TQE that invokes code in first block
  - Records address of TQE block in site-specific longword
  - After program run, user can log out

    Code will still be executed periodically

    No process overhead involved

    Independent of CURRENT process

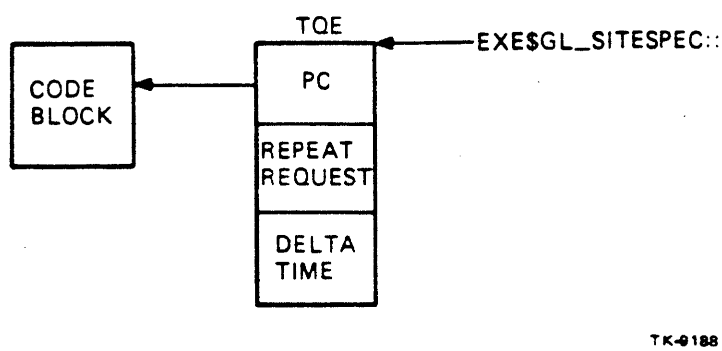


Figure 2 Sample System Programs

- STOPTQE

  - Removes TQE from queue
  - Deallocates TQE and code block
  - Clears site-specific longword

```
        .TITLE  MAKETQE -- Inserts TQE into timer queue
        .IDENT  /V01/
;++
;
; ABSTRACT:
;
;       This program places a segment of code into nonpaged pool,
;       and then establishes a TQE which invokes that routine
;       every tenth of a second.
;
; SIDE EFFECTS:
;
;       Non-paged pool is used to hold the TQE, and the code that
;       executes.
;
; PROGRAMMER:
;
;       Vik Muiznieks   15-MAY-1980
;
;--
;
;       External symbols
        $IPLDEF                                 ; IPL definitions
        $TQEDEF                                 ; TQE definitions
;
;       Local symbols
HEADER = 12                                     ; size of header
DYN_C_MY_TYPE = 120                             ; my block type
;
;       Local storage
        .PSECT  NONSHARED_DATA  PIC, NOEXE, LONG
DELTA:  .LONG   10000*100                       ; delta repeat time
        .LONG   0                               ; of .1 seconds
;
;       This is the code that executes every .1 seconds in response to
;       the TQE.  The timer interrupt service routine transfers control
;       to the code with a JSB instruction at IPL$_TIMER (7).  Note that
;       the code must be PIC (position independent) since it is being COPIED
;       to the system buffer (and executes at arbitrary system addresses).
;
COPY_START:                                     ; start of code to be
                                                ; copied into pool
        INCL    @UPDATE                         ; This is where the
                                                ; routine could do
                                                ; useful work
        RSB                                     ; return control to
                                                ; timer interrupt
                                                ; service routine
UPDATE: .LONG   0                               ; will hold address of
                                                ; location to be incremented
COPY_LEN = . - COPY_START                       ; size of copied code
;
;       Program entry point
;
        .PSECT  CODE    PIC, SHR, NOWRT
START:  .WORD   0                               ; null entry mask
        $CMKRNL_S  ROUTIN=10$                   ; enter kernel mode
        RET                                     ; all done

10$:    .WORD   ^M<R2,R3,R4,R5>                 ; save registers used
        .ENABL  LSB                             ; enable local symbol block
        TSTL    G^EXE$GL_SITESPEC               ; if in use, error
        BEQLU   15$
```

```
        MOVL    #SS$_IVMODE,R0
        RET
;
;       Allocate pool to hold code.  Code must be placed in system
;       space so that it can execute in ANY process context.  HEADER extra
;       bytes will be allocated for a header (since the code block may
;       later be deleted by running program STOPTQE).  The program will
;       use the first word in the third longword to store the size of
;       the block.  Normally the system uses the first two longwords
;       for forward and backward links.  In this case, the first
;       longword will be incremented each time the routine specified
;       by the TQE executes.  The second longword will not be used.
;       Note that IPL is raised to IPL$_ASTDEL before the block of pool
;       is allocated.  This is done so that the process can not be
;       deleted while it has the address of the block in a register
;       (and no other record of the block is maintained elsewhere in
;       the system).
;
15$:    MOVL    #COPY_LEN+HEADER,R1             ; size of pool needed
        SETIPL  #IPL$_ASTDEL                    ; so process not deleted
        JSB     G^EXE$ALONONPAGED               ; allocate pool
;
;       The above routine destroys R0-R3, and returns in R2 the
;       address of the allocated block of pool.
;
        BLBS    R0,20$                          ; proceed if no error
        SETIPL  #0                              ; lower IPL before exiting
        MOVZWL  #SS$_INSFMEM,R0                 ; indicate error
        RET                                     ; return error code
20$:    MOVL    R2,UPDATE                       ; save address of block
        CLRQ    (R2)+                           ; clear location to be update
                                                ; point R2 to 3rd longword
        MOVW    R1,(R2)+                        ; fill in size field
        MOVZBW  #DYN_C_MY_TYPE,(R2)+            ; fill in type field and
                                                ; point R2 to start of code
        PUSHL   R2                              ; save address of code
        MOVC3   #COPY_LEN,COPY_START,(R2)       ; copy code to buffer
                                                ; NOTE -- R0-R5 altered
;
;       Allocate a TQE.  Note that the routine allocates the TQE at
;       IPL$_SYNCH, but returns control at IPL$_ASTDEL (so process
;       cannot be deleted before it can deallocate pool used for TQE).
;       The routine destroys R0-R4, and returns the address of the TQE
;       block in R2.
;
        JSB     G^EXE$ALLOCTQE                  ; allocate TQE block
        BLBS    R0,40$                          ; continue if no error
        MOVL    (SP)+,R0                        ; else, get code address
                                                ; and clean up stack
        SUBL    #HEADER,R0                      ; account for header
        JSB     G^EXE$DEANONPAGED               ; deallocate code block
        MOVZWL  #SS$_NOSLOT,R0                  ; return error code
        BRB     50$                             ; and exit
;
;       Initialize TQE and insert TQE into queue (using system routine).
;       The routine expects the TQE address in R5.  It copies the
;       due time into the TQE, and inserts the TQE in the queue at
;       the appropriate point.  Since the current time is passed
;       (in R0 and R1) as the due time, the TQE should be placed
;       at the head of the queue, and delivered after the next
;       timer interrupt.
;
;       The address of the TQE is also stored in a global location
```

```
;       in the executive reserved for site-specific use.
;
40$:    MOVB    #TQE$C_SSREPT,TQE$B_RQTYPE(R2)  ; indicate system sub.
                                                ; and repeat request
        MOVQ    DELTA,TQE$Q_DELTA(R2)           ; set repeat time-.1 sec
        MOVL    (SP)+,TQE$L_FPC(R2)             ; starting address of code;
                                                ; also cleans up stack
        MOVL    R2,G^EXE$GL_SITESPEC            ; save TQE address for
                                                ; program that will
                                                ; cancel TQE request
        ASSUME  IPL$_SYNCH EQ IPL$_TIMER
LOCK_START:

        SETIPL SYNCH                            ; accessing system data base
        MOVQ    G^EXE$GQ_SYSTIME,R0             ; get current abs. time
        MOVL    R2,R5                           ; copy TQE address for
        JSB     G^EXE$INSTIMQ                   ; queuing routine
        MOVZWL  #SS$_NORMAL,R0                  ; set success status
50$:    SETIPL  #0                              ; lower IPL
        RET                                     ; all done
        .DSABL  LSB                             ; disable local symbol block
;
;       By placing the SYNCH label after the code that must execute
;       at IPL$_SYNCH, the page with the SETIPL SYNCH instruction and
;       the page with the SYNCH label are guaranteed to be in the
;       process's working set.  Since the code will not span more
;       than 2 pages, there is no way to have a page fault above IPL 2,
;       even though the pages have not been locked into the working
;       set (with the $LKWSET system service).
;
SYNCH:  .LONG   IPL$_SYNCH
LOCK_END:
        ASSUME LOCK_END-LOCK_START LE 512

        .END    START
```

```
$ set process/priv=cmkrnl
$
$ RUN/NODEBUG MAKETQE
$
$ RUN/NODEBUG MAKETQE
%SHR-F-IVMODE, invalid mode for requested function
$
$ RUN/NODEBUG STOPTQE
Value in EXE$GL_SITESPEC = 801FEA00
Value in field = 0000010F
Value in field = 0000010F
Value in field = 0000010F
$
$ RUN/NODEBUG STOPTQE
MAKETQE program has not been run.
$
$ RUN/NODEBUG MAKETQE
$
$ RUN/NODEBUG STOPTQE
Value in EXE$GL_SITESPEC = 80205A00
Value in field = 0000003A
Value in field = 0000003A
Value in field = 0000003A
```

Example 6 Sample Run

```
        .TITLE  STOPTQE -- Removes TQE from timer queue
        .IDENT  /V01/
;++
;
; ABSTARCT:
;
;       This program displays the contents of the location being updated
;       by the routine specified in a TQE (thrice).  It then cancels the
;       TQE request, and deallocates the block of pool being used to
;       contain the TQE routine.
;
; SIDE EFFECTS:
;
;       Non-paged pool is returned to the system.
;
; PROGRAMMER:
;
;       Vik Muiznieks   15-MAY-1980
;
;--
;
;       External symbols
        $IPLDEF                                 ; IPL definitions
        $TQEDEF                                 ; TQE definitions
;
;       Local symbols
HEADER = 12                                     ; header size for code block
LOOP_CNT = 3                                    ; loop counter
;
;       Local storage
        .PSECT  NONSHARED_DATA  PIC, NOEXE, LONG
LKWSET: .ADDRESS START_LOCK                     ; starting address
        .ADDRESS END_LOCK                       ; ending address
TTCHAN: .WORD   0                               ; TT channel
TT:     .ASCID  /SYS$COMMAND/                   ; descriptor for terminal
CTR:    .LONG   STR_END - STRING                ; $FAO control string
        .ADDRESS STRING                         ; descriptor
CTR1:   .LONG   STR1_END - STR                  ; $FAO control string
        .ADDRESS STR                            ; descriptor
STR:    .ASCII  *Value in EXE$GL_SITESPEC = !XL*; converts to hexadecimal
STR1_END:
STRING: .ASCII  *Value in field = !XL*          ; converts to hexadecimal
STR_END:
FAOLEN: .LONG                                   ; $FAO output length
OUT:    .LONG   35                              ; Output string desc.
        .ADDRESS BUFF
BUFF:   .BLKB   35                              ; Actual output string
BAD_MESSAGE:                                    ; used in case MAKETQE
        .ASCII  /MAKETQE program has not been run./ ; not yet run
BAD_SIZE = . - BAD_MESSAGE
;
;       Entry point for routine
        .PSECT  CODE    PIC, SHR, NOWRT
START:  .WORD   0                               ; null entry mask
        $CMKRNL_S       ROUTIN=10$              ; enter kernel mode
;       Note that most of the work being done in kernel mode by this
;       example really could be done in user mode.  There is not much
;       need to enter kernel mode before label START_LOCK.
        RET                                     ; all done
10$:    .WORD   ^M<R2,R3,R4,R5,R6>              ; save registers used
        $LKWSET_S INADR=LKWSET                  ; lock pages in working set
        BLBS    R0,15$                          ; proceed on success
        RET                                     ; stop on error
```

```
15$:    $ASSIGN_S DEVNAM=TT,CHAN=TTCHAN            ; get channel to terminal
        BLBC    R0,25$                             ; exit on error
20$:    MOVL    G^EXE$GL_SITESPEC,R2               ; get TQE address
                                                   ; if negative, system address
        BLSS    30$                                ; stop if not negative
        $OUTPUT CHAN=TTCHAN,LENGTH=#BAD_SIZE,BUFFER=BAD_MESSAGE
        $DASSGN_S CHAN=TTCHAN                      ; deassign terminal channel
        RET                                        ; all done
25$:    BRW     ERROR                              ; solve BLBC byte displacemen
30$:    MOVL    TQE$L_FPC(R2),R6                   ; get code address
        SUBL2   #HEADER,R6                         ; point to update location
        MOVZBL  #LOOP_CNT,R4                       ; set loop count
        $FAO_S  CTRSTR=CTR1,OUTLEN=FAOLEN,-        ; format EXE$GL_SITESPEC
                OUTBUF=OUT,P1=R2                   ; for debugging
        BLBC    R0,25$                             ; test for errors
        $OUTPUT CHAN=TTCHAN,LENGTH=FAOLEN,BUFFER=BUFF ; print value
        BLBC    R0,25$                             ; test for errors
40$:    $FAO_S  CTRSTR=CTR,OUTLEN=FAOLEN,-         ; format counter which
                OUTBUF=OUT,P1=(R6)                 ; changes every .1 seconds
        BLBC    R0,25$                             ; check for error
        $OUTPUT CHAN=TTCHAN,LENGTH=FAOLEN,BUFFER=BUFF ; display counter
        BLBC    R0,ERROR                           ; check for error
        SOBGTR  R4,40$                             ; loop a few times
START_LOCK:                                        ; code must be locked in
                                                   ; working set so no page
                                                   ; faults above IPL 2
        SETIPL  #IPL$_SYNCH                        ; raise IPL to synch
        REMQUE  (R2),R0                            ; remove TQE from queue
        JSB     G^EXE$DEANONPAGED                  ; deallocate TQE
        MOVL    R6,R0                              ; get address of code block
        JSB     G^EXE$DEANONPAGED                  ; deallocate code block
        CLRL    G^EXE$GL_SITESPEC                  ; clean-up location so this
                                                   ; program cannot be rerun
                                                   ; until MAKETQE rerun
        SETIPL  #0                                 ; enable interrupts
END_LOCK:                                          ; end of locked down code
        $DASSGN_S CHAN=TTCHAN                      ; deassign terminal channel
        MOVZWL  #SS$_NORMAL,R0                     ; return success status
        RET                                        ; all done
ERROR:  MOVL    R0,R6                              ; save exit status code
        $DASSGN_S CHAN=TTCHAN                      ; deassign terminal channel
        MOVL    R6,R0                              ; restore exit status code
        RET                                        ; all done
        .END    START
```

Clocks and Timer Services

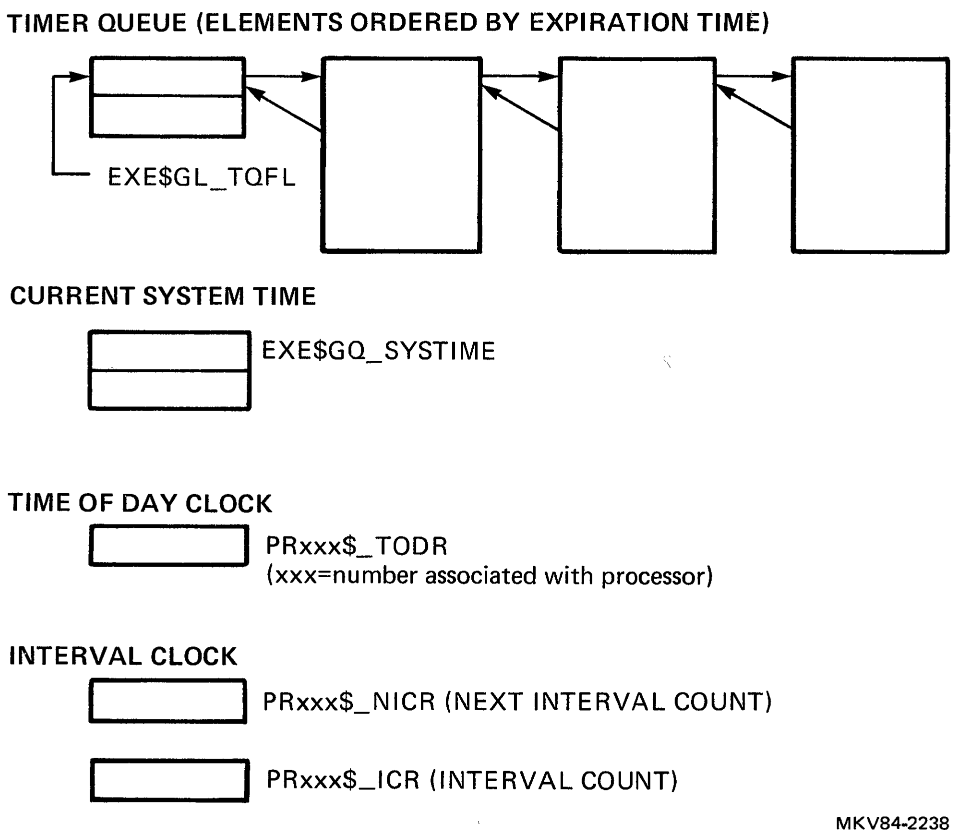


Figure 10 Clocks and Timer Services

Summary of System Synchronization Tools

Table 5 Summary of System Synchronization Tools

Function	Implementation	Name
Arbitrate interrupt requests	Hardware-maintained priority	Interrupt priority level (IPL)
Service interrupts and exceptions	Table of service routine addresses	System control block (SCB)
Synchronize execution of system routines	Interrupt service routines	Timer, SCHED, etc.
Request software interrupt	MACRO	SOFTINT
Synchronize system's access to scheduler data structures	MACRO - raise IPL to IPL$_SYNCH	SETIPL or DSBINT
Continue execution of code at lower priority	Queue request, SOFTINT, REI	FORK

PROCESS SYNCHRONIZATION

Table 6 Process Synchronization Mechanisms

Function	Implementation	Name
Synchronize certain system-level activities of processes	Adjust IPL (SETIPL macro)	IPL
Allow process to request action at a certain time	Queue of requests and hardware and software clock interrupts	Timer queue
Synchronize access to data structures by processes	Semaphore	Mutex
Allow process to execute procedure on completion of event	REI IPL 2 interrupt service routine	Asynchronous system trap (AST)
Allow processes to synchronize access to resources	$ENQ(W) and $DEQ system services	VMS lock manager

Mutual Exclusion Semaphores (MUTEXes)

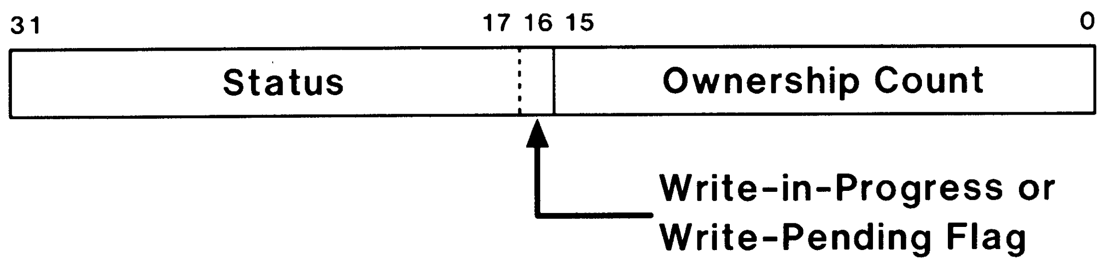


Figure 11 A Mutex

- Protect data structures against conflicting accesses by multiple processes
- One writer or multiple readers are allowed
- Examples:
  - Group logical name tables
  - System logical name table
- To access the data structure, first place a lock on the mutex
- Mutex locking is only possible in process context

SEMAPHORE

For articles on related subjects *see* CONCURRENT PROGRAMMING; DEADLOCK; LOCKOUT; MONITORS; PARALLEL PROCESSING; and PETRI NETS.

Semaphores are synchronization primitives used to coordinate the activities of two or more programs or processes that are running at the same time and sharing information. They are used for elementary interprocess communication, to guarantee exclusive access to shared data, to protect a section of code that must be executed without certain kinds of interruptions (such a code segment is called a *critical region* or *critical section*), or to allocate a set of identical scarce resources.

Two operations are defined on semaphores: *P*, or wait, and *V*, or proceed. The usage protocol for a shared resource is as follows: A process that needs control of a resource executes a *P* operation on the semaphore associated with that resource. The system suspends the process until the resource is available, and then allows it to proceed. When the process is finished with the resource, it executes a *V* operation on the semaphore to release the resource for use by another process. The resource may be any hardware or software component, including data structures, physical devices, or code segments. A semaphore may also be used to indicate when it is safe for execution to proceed past a certain point in the program. The usage protocol is slightly different when a semaphore is used to coordinate interprocess communication. For example, if process *A* requires data produced by process *B* before it can execute further, a semaphore can be used to block *A* until *B* provides the data and releases *A* with a *V* operation.

One case of special interest is the *mutex* (for mutual exclusion) semaphore, which allows only one process to use the resource at once. This is particularly useful for protecting a data structure from being updated simultaneously by more than one process.

Semaphores are often implemented with counters. For example, a typical implementation of a semaphore (call it SEM) might involve:

- Initialization of SEM. (Set the counter of SEM to the total number of instances of the resource; e.g., for a mutex semaphore, to 1.)
- P(SEM). (If the counter of SEM is greater than zero, decrement it by one and allow the calling process to proceed; otherwise, block the calling process and switch to another—unblocked—process.)
- V(SEM). (If there is a blocked process waiting on SEM, then select and awaken some blocked process; otherwise, increment the counter of SEM by one.)

The bodies of these routines must be indivisible (uninterruptible operations). The *P* and *V* notation is due to Dijkstra, who, motivated by the counter implementation, used his native Dutch to get *P* from *proberen te verlagen* ("to try to decrease") and *V* from *verhogen* ("to increase").

REFERENCE

1968. Dijkstra, Edsger W. "The Structure of the 'THE'-Multiprogramming System," *Comm. ACM* **11**, *No. 5*: 341–346 (May).

M. SHAW

List of Data Structures Protected by Mutexes

Data Structure	Global Name of Mutex [1]
Logical Name Table	LNM$AL_MUTEX
I/O Database [2]	IOC$GL_MUTEX
Common Event Block List	EXE$GL_CEBMTX
Paged Dynamic Memory	EXE$GL_PGDYNMTX
Global Section Descriptor List	EXE$GL_GSDMTX
Shared Memory Global Section Descriptor Table	EXE$GL_SHMGSMTX
Shared Memory Mailbox Descriptor Table	EXE$GL_SHMMBMTX
(not currently used)	EXE$GL_ENQMTX
Line Printer Unit Control Block [3]	UCB$L_LP_MUTEX
(not currently used)	EXE$GL_ACLMTX
System Intruder Lists	CIA$GL_MUTEX
Object Rights Block Access Control List [4]	ORB$GL_ACL_MUTEX

[1] When a process is placed into an MWAIT state waiting for a mutex, the address of the mutex is placed into the PCB$L_EFWM field of the PCB. The symbolic contents of PCB$L_EFWM will probably remain the same from release to release but the numeric contents change. The numeric values are available from the system map SYS$SYSTEM:SYS.MAP

[2] This mutex is used by the Assign Channel and Allocate Device system services when searching through the linked list of device data blocks and unit control blocks (UCBs) for a device. It is also used whenever UCBs are added or deleted, for example, during the creation of mailboxes and network devices.

[3] The mutex associated with each line printer unit does not have a fixed location like the other mutexes. As a field in the unit control block (UCB), its location and value depend on where the UCB for that unit is allocated.

[4] The mutex associated with each object rights block (ORB) does not have a fixed location like the other mutexes. As a field in the object rights block, its location and value depend on where the ORB is allocated.

The mutex itself consists of a single longword that contains the number of owners of the mutex (MTX$W_OWNCNT) in the low-order word and status flags (MTX$W_STS) in the high-order word (see Figure 2-1). The owner count begins at -1 so that a mutex with a zero in the low-order word has one owner. The only flag currently implemented indicates whether a write operation is either in progress or pending for this mutex (MTX$V_WRT).

Table of contents

```
0000
0000           .TITLE  MUTEX - MUTEX WAIT ROUTINES
0000           .IDENT  'X-1'
0000
0000   ;
0000   ;****************************************************************************
0000   ;*                                                                          *
0000   ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
0000   ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
0000   ;*  ALL RIGHTS RESERVED.                                                    *
0000   ;*                                                                          *
0000   ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
0000   ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000   ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
0000   ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
0000   ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
0000   ;*  TRANSFERRED.                                                            *
0000   ;*                                                                          *
0000   ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
0000   ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
0000   ;*  CORPORATION.                                                            *
0000   ;*                                                                          *
0000   ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
0000   ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
0000   ;*                                                                          *
0000   ;*                                                                          *
0000   ;****************************************************************************
0000
0000   ;++
0000   ; FACILITY: EXECUTIVE, SCHEDULER
0000   ;
0000   ; ABSTRACT:
0000   ;       THIS MODULE CONTAINS THE ROUTINES WHICH IMPLEMENT THE MUTEX
0000   ;       LOCK AND UNLOCK SERVICES FOR INTERNAL EXECUTIVE USE.
0000   ;
0000   ;
0000   ; ENVIRONMENT:
0000   ;       MODE = KERNEL
0000   ;
0000   ;--
0000   ;
0000   ;       .PAGE
0000           .SBTTL  HISTORY                 ; DETAILED
0000   ;
0000   ; AUTHOR:       R. HUSTVEDT  CREATION DATE: 25-AUG-76
0000   ;
0000   ; MODIFIED BY:
0000   ;
0000   ;       V03-003 SSA0022         Stan Amway              2-Apr-1984
0000   ;               Backed out SSA0005. It was temporary.
0000   ;
0000   ;       V03-002 SSA0005         Stan Amway              10-Jan-1984
0000   ;               Added code to maintain PMS MWAIT transition counters.
0000   ;               The counters (in MDAT) and supporting code will be removed
0000   ;               before V4 release.
0000   ;
0000   ;       V03-001 ROW0168         Ralph O. Weber          3-MAR-1983
```

```
0000     58 ;            Change W^ references to G^.
0000     59 ;
```

```
    0000           .SBTTL  DECLARATIONS
    0000
    0000   ;
    0000   ; INCLUDE FILES:
    0000   ;
    0000
    0000           $DYNDEF                          ; STRUCTURE TYPE DEFINITIONS
    0000           $IPLDEF                          ; IPL DEFINITIONS
    0000           $MTXDEF                          ; MUTEX DEFINITIONS
    0000           $PCBDEF                          ; PCB DEFINITIONS
    0000           $PRDEF                           ; PROCESSOR REGISTER DEFINITIONS
    0000           $PRIDEF                          ; PRIORITY INCR CLASS DEFS
    0000           $PSLDEF                          ; PSL DEFINITIONS
    0000           $SSDEF                           ; SYSTEM STATUS CODES
    0000           $STATEDEF                        ; SCHEDULER STATE DEFS
    0000           $WQHDEF                          ; WAIT QUEUE HEADER DEFS
    0000   ;
    0000   ; EQUATED SYMBOLS
    0000   ;
    0000
00000000           .PSECT  AEXENONPAGED,BYTE        ; NONPAGED EXEC
```

```
                              0000          .SBTTL  SCH$RWAIT - RESOURCE WAIT
                              0000
                              0000     ;++
                              0000     ; FUNCTIONAL DESCRIPTION:
                              0000     ;        SCH$RWAIT SUSPENDS THE EXECUTION OF A PROCESS UNTIL REQUIRED
                              0000     ;        RESOURCES ARE AVAILABLE.
                              0000     ;
                              0000     ; CALLING SEQUENCE:
                              0000     ;        SETIPL/DSBINT #IPL$_SYNCH
                              0000     ;        PUSHL   <PSL>
                              0000     ;        BSB/JSB SCH$RWAIT
                              0000     ;
                              0000     ; INPUT PARAMETERS:
                              0000     ;        R0 - RESOURCE NUMBER FOR WHICH TO WAIT
                              0000     ;        R4 - PCB ADDRESS
                              0000     ;        00(SP) - PC AT WHICH TO RESUME
                              0000     ;        04(SP) - PSL WITH WHICH TO RESUME
                              0000     ;
                              0000     ; IMPLICIT INPUTS:
                              0000     ;        SCH$GQ_MWAIT - MUTEX WAIT QUEUE HEADER
                              0000     ;        PCB OF CURRENT PROCESS
                              0000     ;
                              0000     ; OUTPUTS:
                              0000     ;        R0-R3 PRESERVED
                              0000     ;
                              0000     ; IMPLICIT OUTPUTS:
                              0000     ;        *** TBS ***
                              0000     ;
                              0000     ; SIDE EFFECTS:
                              0000     ;        *** TBS ***
                              0000     ;
                              0000     ;--
                              0000
                              0000     SCH$RWAIT::                                  ;;; RESOURCE WAIT ENTRY POINT
00 00000000'GF     50     E6  0000             BBSSI   R0,G^SCH$GL_RESMASK,10$ ;;; SET WAITING FLAG
                   7E     11  0008     10$:    BRB     WAITR                        ;;; AND ENTER WAIT STATE
                              000A
```

```
                    000A         .SBTTL  SCH$LOCKWNOWAIT - LOCK MUTEX FOR WRITE WITHOUT WAITING
                    000A
                    000A     ;++
                    000A     ; FUNCTIONAL DESCRIPTION:
                    000A     ;       SCH$LOCKWNOWAIT LOCKS THE SPECIFIED MUTEX FOR EXCLUSIVE WRITE ACCESS
                    000A     ;       TO THE PROTECTED STRUCTURE.  IF ANOTHER PROCESS HAS ALREADY CLAIMED
                    000A     ;       THE MUTEX, THEN THIS ROUTINE RETURNS A FAILURE INDICATION.
                    000A     ;
                    000A     ;
                    000A     ;
                    000A     ; CALLING SEQUENCE:
                    000A     ;       BSB/JSB SCH$LOCKWNOWAIT
                    000A     ;
                    000A     ;
                    000A     ; INPUT PARAMETERS:
                    000A     ;       R0 - ADDRESS OF MUTEX
                    000A     ;       R4 - PCB ADDRESS OF CURRENT PROCESS
                    000A     ;
                    000A     ; IMPLICIT INPUTS:
                    000A     ;       SCH$GQ_MWAIT - MUTEX WAIT QUEUE HEADER
                    000A     ;       PCB OF CURRENT PROCESS
                    000A     ;       MUTEX LOCATED BY R0
                    000A     ;
                    000A     ; OUTPUTS:
                    000A     ;       R0 LOW BIT SET IF LOCKED SUCCESSFULLY
                    000A     ;          LOW BIT CLEAR IF MUTEX IN USE
                    000A     ;       R1-R3 PRESERVED
                    000A     ;       IPL = ASTDEL
                    000A     ;
                    000A     ; IMPLICIT OUTPUTS:
                    000A     ;       *** TBS ***
                    000A     ;
                    000A     ; SIDE EFFECTS:
                    000A     ;       *** TBS ***
                    000A     ;
                    000A     ;--
                    000A     SCH$LOCKWNOWAIT::
                    000A             SETIPL  #IPL$_SYNCH             ;;; RAISE TO SYNCH IPL
0B 60     10    E6  000D             BBSSI   #MTX$V_WRT,(R0),20$     ;;; SET WRITE PENDING
          60    B6  0011             INCW    MTX$W_OWNCNT(R0)        ;;; RAISE OWNER COUNT
          05    12  0013             BNEQ    10$                     ;;; RETURN FAILURE IF BUSY
   50     01    3C  0015             MOVZWL  #SS$_NORMAL,R0          ;;; INDICATE SUCCESSFUL COMPLETION
          32    11  0018             BRB     LKEX                    ;;; AND MERGE WITH COMMON EXIT CODE
          60    B7  001A     10$:    DECW    MTX$W_OWNCNT(R0)        ;;; CORRECT COUNT
          50    D4  001C     20$:    CLRL    R0                      ;;; SET FAILURE RETURN INDICATION
                    001E             SETIPL  #IPL$_ASTDEL            ;;; LOWER TO ASTDEL
                05  0021             RSB                             ; AND RETURN
```

```
                        0022            .SBTTL  SCH$IOLOCKW - LOCK I/O DATA BASE MUTEX FOR WRITE
                        0022    ;++
                        0022    ; FUNCTIONAL DESCRIPTION:
                        0022    ;       SCH$IOLOCKW RETURNS TO THE CALLER WHEN THE I/O DATABASE MUTEX
                        0022    ;       HAS BEEN LOCKED FOR WRITE ASSURING EXCLUSIVE ACCESS.
                        0022    ;
                        0022    ;
                        0022    ;
                        0022    ; CALLING SEQUENCE:
                        0022    ;       BSB/JSB SCH$IOLOCKW
                        0022    ;
                        0022    ;
                        0022    ; INPUT PARAMETERS:
                        0022    ;       R4 - PCB ADDRESS OF CURRENT PROCESS
                        0022    ;
                        0022    ; IMPLICIT INPUTS:
                        0022    ;       SCH$GQ_MWAIT - MUTEX WAIT QUEUE HEADER
                        0022    ;       PCB OF CURRENT PROCESS
                        0022    ;       I/O DATABASE MUTEX
                        0022    ;
                        0022    ; OUTPUTS:
                        0022    ;       R0 = ADDRESS OF I/O DATABASE MUTEX
                        0022    ;       R1-R3 PRESERVED
                        0022    ;       IPL = ASTDEL
                        0022    ;
                        0022    ; IMPLICIT OUTPUTS:
                        0022    ;       *** TBS ***
                        0022    ;
                        0022    ; SIDE EFFECTS:
                        0022    ;       *** TBS ***
                        0022    ;
                        0022    ;--
                        0022
                        0022    SCH$IOLOCKW::                           ; LOCK I/O DATA BASE FOR WRITE ACCESS
50   00000000'EF    9E  0022            MOVAB   IOC$GL_MUTEX,R0         ; GET ADDRESS OF I/O DATABASE MUTEX
```

```
                 0029         .SBTTL  SCH$LOCKW - LOCK MUTEX FOR WRITE
                 0029 ;++
                 0029 ; FUNCTIONAL DESCRIPTION:
                 0029 ;       SCH$LOCKW RETURNS TO THE CALLER WHEN THE SPECIFIED MUTEX
                 0029 ;       HAS BEEN LOCKED FOR WRITE ASSURING EXCLUSIVE ACCESS TO THE
                 0029 ;       PROTECTED STRUCTURE.
                 0029 ;
                 0029 ;
                 0029 ;
                 0029 ; CALLING SEQUENCE:
                 0029 ;       BSB/JSB SCH$LOCKW
                 0029 ;
                 0029 ;
                 0029 ; INPUT PARAMETERS:
                 0029 ;       R0 - ADDRESS OF MUTEX
                 0029 ;       R4 - PCB ADDRESS OF CURRENT PROCESS
                 0029 ;
                 0029 ; IMPLICIT INPUTS:
                 0029 ;       SCH$GQ_MWAIT - MUTEX WAIT QUEUE HEADER
                 0029 ;       PCB OF CURRENT PROCESS
                 0029 ;       MUTEX LOCATED BY R0
                 0029 ;
                 0029 ; OUTPUTS:
                 0029 ;       R0-R3 PRESERVED
                 0029 ;       IPL = ASTDEL
                 0029 ;
                 0029 ; IMPLICIT OUTPUTS:
                 0029 ;       *** TBS ***
                 0029 ;
                 0029 ; SIDE EFFECTS:
                 0029 ;       *** TBS ***
                 0029 ;
                 0029 ;--
                 0029
                 0029 SCH$LOCKW::                                  ; LOCK MUTEX FOR WRITE
                 0029 10$:    SETIPL  #IPL$_SYNCH                 ;;; RAISE TO SYNCH IPL
08 60   10   E6  002C         BBSSI   #MTX$V_WRT,(R0),30$         ;;; SET WRITE PENDING
        60   B6  0030         INCW    MTX$W_OWNCNT(R0)            ;;; RAISE OWNER COUNT
        02   12  0032         BNEQ    20$                         ;;; WAIT IF BUSY
        16   11  0034         BRB     LKEX                        ;;; MERGE WITH COMMON EXIT CODE
                 0036
                 0036 20$:                                        ;;; MUST WAIT FOR EXCLUSIVE USE
        60   B7  0036         DECW    MTX$W_OWNCNT(R0)            ;;; CORRECT COUNT
        43   10  0038 30$:    BSBB    WAITM                       ;;; AND WAIT FOR MUTEX
        ED   11  003A         BRB     10$                         ; REPEAT LOCK ATTEMPT WHEN
                 003C                                             ; RESCHEDULED
```

```
                           003C        .SBTTL  SCH$IOLOCKR - LOCK I/O DATABASE MUTEX FOR READ
                           003C ;++
                           003C ; FUNCTIONAL DESCRIPTION:
                           003C ;      SCH$IOLOCKR RETURNS TO THE CALLER WHEN NO WRITERS OWN THE I/O
                           003C ;      DATABASE MUTEX THUS ASSURING THE I/O DATABASE WILL REMAIN UN-
                           003C ;      CHANGED UNTIL THE MUTEX IS RELEASED. IPL IS RAISED TO PREVENT
                           003C ;      AST DELIVERY WHILE THE MUTEX IS OWNED AND THE PROCESS WILL NOT
                           003C ;      BE OUTSWAPPED.
                           003C ;
                           003C ; CALLING SEQUENCE:
                           003C ;      BSB/JSB SCH$IOLOCKR
                           003C ;
                           003C ; INPUT PARAMETERS:
                           003C ;      R4 - CURRENT PROCESS PCB ADDRESS
                           003C ;
                           003C ; IMPLICIT INPUTS:
                           003C ;      SCH$GQ_MWAIT - MUTEX WAIT QUEUE HEADER
                           003C ;      PCB OF CURRENT PROCESS
                           003C ;      I/O DATABASE MUTEX
                           003C ;
                           003C ; OUTPUTS:
                           003C ;      R0 = ADDRESS OF I/O DATABASE MUTEX
                           003C ;      R1-R3 PRESERVED
                           003C ;      IPL = ASTDEL
                           003C ;
                           003C ; IMPLICIT OUTPUTS:
                           003C ;      *** TBS ***
                           003C ;
                           003C ; SIDE EFFECTS:
                           003C ;      *** TBS ***
                           003C ;
                           003C ;--
                           003C
                           003C SCH$IOLOCKR::                          ; LOCK I/O DATABASE FOR READ ACCESS
    50   00000000'EF   9E  003C        MOVAB   IOC$GL_MUTEX,R0        ; GET ADDRESS OF I/O DATA BASE MUTEX
```

```
                           0043            .SBTTL  SCH$LOCKR - LOCK MUTEX FOR READ
                           0043     ;++
                           0043     ; FUNCTIONAL DESCRIPTION:
                           0043     ;       SCH$LOCKR RETURNS TO THE CALLER WHEN NO WRITERS OWN THE
                           0043     ;       SPECIFIED MUTEX.  THUS THE STRUCTURE PROTECTED BY THE MUTEX
                           0043     ;       WILL REMAIN UNCHANGED UNTIL THE MUTEX IS RELEASED.  IPL IS
                           0043     ;       RAISED TO PREVENT AST DELIVERY WHILE THE MUTEX IS OWNED AND
                           0043     ;       THE PROCESS WILL NOT BE OUTSWAPPED.
                           0043     ;
                           0043     ; CALLING SEQUENCE:
                           0043     ;       BSB/JSB SCH$LOCKR
                           0043     ;
                           0043     ; INPUT PARAMETERS:
                           0043     ;       R0 - ADDRESS OF MUTEX
                           0043     ;       R4 - CURRENT PROCESS PCB ADDRESS
                           0043     ;
                           0043     ; IMPLICIT INPUTS:
                           0043     ;       SCH$GQ_MWAIT - MUTEX WAIT QUEUE HEADER
                           0043     ;       PCB OF CURRENT PROCESS
                           0043     ;       MUTEX
                           0043     ;
                           0043     ; OUTPUTS:
                           0043     ;       R0-R3 PRESERVED
                           0043     ;       IPL = ASTDEL
                           0043     ;
                           0043     ; IMPLICIT OUTPUTS:
                           0043     ;       *** TBS ***
                           0043     ;
                           0043     ; SIDE EFFECTS:
                           0043     ;       *** TBS ***
                           0043     ;
                           0043     ;--
                           0043
                           0043     SCH$LOCKR::                                     ; LOCK MUTEX FOR READ
                           0043            SETIPL  #IPL$_SYNCH                     ;;; RAISE TO SYNCH IPL
      30 60    10    E0    0046            BBS     #MTX$V_WRT,(R0),RDWAIT          ;;; WAIT IF WRITE PENDING OR
                           004A                                                    ;;; IN PROGRESS
               60    B6    004A            INCW    MTX$W_OWNCNT(R0)                ;;; INCREASE OWNER COUNT
      0A A4    0C    91    004C     LKEX:  CMPB    #DYN$C_PCB,PCB$B_TYPE(R4)       ; CHECK FOR PCB
               25    12    0050            BNEQ    20$                             ; BUG CHECK IF NOT PCB
            0E A4    B6    0052            INCW    PCB$W_MTXCNT(R4)                ;;; NOTE IN PCB ALSO
      01    0E A4    B1    0055            CMPW    PCB$W_MTXCNT(R4),#1             ; IS THIS THE FIRST MUTEX IT OWNS?
               18    12    0059            BNEQ    10$                             ; BR IF OWNS MORE THAN 1 MUTEX
   28 A4    0B A4    90    005B            MOVB    PCB$B_PRI(R4),PCB$B_PRISAV(R4); SAVE CURRENT PRIORITY
   29 A4    2F A4    90    0060            MOVB    PCB$B_PRIB(R4),PCB$B_PRIBSAV(R4) ; SAVE BASE PRIORITY
      0B A4    10    91    0065            CMPB    #16,PCB$B_PRI(R4)               ; IS THIS A REAL TIME PROCESS?
               08    1A    0069            BGTRU   10$                             ; BR IF SO
      0B A4    0F    90    006B            MOVB    #15,PCB$B_PRI(R4)               ; ELSE FORCE TO LOWEST RT PRIORITY
      2F A4    0F    90    006F            MOVB    #15,PCB$B_PRIB(R4)              ; AND SET PRIORITY BASE TO RT
                           0073     10$:   SETIPL  #IPL$_ASTDEL                    ;;; DROP TO ASTDEL IPL
                     05    0076            RSB                                     ;;; AND RETURN
             00AC    31    0077     20$:   BRW     NOTPCB                          ;
                           007A
                           007A     RDWAIT:                                        ;;; MUST WAIT FOR READ
            C6 AF    DF    007A            PUSHAL  SCH$LOCKR                       ;;; RETRY AFTER WAIT
                           007D
                           007D     WAITM:                                         ;;; WAIT FOR MUTEX TO FREE
```

```
                    6E   DD  007D   345          PUSHL   (SP)                          ;;; FORM PC, PSL ON STACK
                 04 AE   DC  007F   346          MOVPSL  4(SP)                         ;;; BUILD PSL
      04 AE   05   10 02 F0  0082   347          INSV    #IPL$_ASTDEL,#PSL$V_IPL,#PSL$S_IPL,4(SP) ;;; SET IPL TO ASTDEL
            4C A4   50   D0  0088   348 WAITR:   MOVL    R0,PCB$L_EFWM(R4)             ;;; SAVE ADDRESS OF MUTEX
        00000000'GF 64   0E  008C   349          INSQUE  (R4),G^SCH$GQ_MWAIT           ;;; INSERT AT HEAD OF WAIT QUEUE
           00000008'GF   B6  0093   350          INCW    G^SCH$GQ_MWAIT+WQH$W_WQCNT        ;;; INCREMENT COUNT IN QUEUE
            2C A4   02   B0  0099   351          MOVW    #SCH$C_MWAIT,PCB$W_STATE(R4)      ;;; SET STATE
                 FF60'   31  009D   352          BRW     SCH$WAITL                     ;;; WAIT WITH STACK CLEAN, STATE SET
                             00A0   353
```

```
                              00A0         .SBTTL  SCH$RAVAIL - DECLARE RESOURCE AVAILABILITY
                              00A0
                              00A0  ;++
                              00A0  ; FUNCTIONAL DESCRIPTION:
                              00A0  ;       SCH$RAVAIL IS CALLED TO SIGNAL THE AVAILABILITY OF THE SPECIFIED
                              00A0  ;       RESOURCE AND RELEASE ANY WAITING PROCESSES.
                              00A0  ;
                              00A0  ; CALLING SEQUENCE:
                              00A0  ;       BSB/JSB SCH$RAVAIL
                              00A0  ;
                              00A0  ; INPUT PARAMETERS:
                              00A0  ;       R0 - RESOURCE NUMBER
                              00A0  ;
                              00A0  ; IMPLICIT OUTPUTS:
                              00A0  ;       *** TBS ***
                              00A0  ;
                              00A0  ; SIDE EFFECTS:
                              00A0  ;       *** TBS ***
                              00A0  ;
                              00A0  ;--
                              00A0
                              00A0  SCH$RAVAIL::                                  ; DECLARE RESOURCE AVAILABILITY
7D 00000000'GF    50    E7    00A0          BBCCI   R0,G^SCH$GL_RESMASK,EXIT            ; CLEAR AND TEST WAITING FLAG
                              00A8          DSBINT  #IPL$_SYNCH                   ;;; BLOCK SYSTEM EVENTS
                  45    11    00AE          BRB     UNLOCK                        ;;; MERGE WITH COMMON CODE
```

```
                                    00B0        .SBTTL  SCH$IOUNLOCK - UNLOCK I/O DATABASE MUTEX
                                    00B0  ;++
                                    00B0  ; FUNCTIONAL DESCRIPTION:
                                    00B0  ;       SCH$IOUNLOCK RELEASES OWNERSHIP OF THE I/O DATABASE MUTEX AND
                                    00B0  ;       RE-ACTIVATES ANY WAITING PROCESSES IF THE MUTEX HAS BECOME
                                    00B0  ;       AVAILABLE AS A CONSEQUENCE OF THIS UNLOCK REQUEST.
                                    00B0  ;
                                    00B0  ; CALLING SEQUENCE:
                                    00B0  ;       BSB/JSB SCH$IOUNLOCK
                                    00B0  ;
                                    00B0  ; INPUT PARAMETERS:
                                    00B0  ;       R4 - PCB ADDRESS OF CURRENT PROCESS
                                    00B0  ;
                                    00B0  ; IMPLICIT INPUTS:
                                    00B0  ;       SCH$GQ_MWAIT - MUTEXT WAIT QUEUE HEADER
                                    00B0  ;       PCB OF CURRENT PROCESS
                                    00B0  ;       I/O DATABASE MUTEX
                                    00B0  ;
                                    00B0  ; IMPLICIT OUTPUTS:
                                    00B0  ;       *** TBS ***
                                    00B0  ;
                                    00B0  ; SIDE EFFECTS:
                                    00B0  ;       *** TBS ***
                                    00B0  ;
                                    00B0  ;--
                                    00B0
                                    00B0  SCH$IOUNLOCK::                          ; UNLOCK I/O DATABASE MUTEX
       50   00000000'EF    9E       00B0        MOVAB   IOC$GL_MUTEX,R0           ; GET ADDRESS OF I/O DATABASE MUTEX
```

```
                                00B7         .SBTTL  SCH$UNLOCK - UNLOCK MUTEX
                                00B7  ;++
                                00B7  ; FUNCTIONAL DESCRIPTION:
                                00B7  ;        SCH$UNLOCK RELEASES OWNERSHIP OF THE SPECIFIED MUTEX AND
                                00B7  ;        RE-ACTIVATES ANY WAITING PROCESSES IF THE MUTEX HAS BECOME
                                00B7  ;        AVAILABLE AS A CONSEQUENCE OF THIS UNLOCK REQUEST.
                                00B7  ;
                                00B7  ; CALLING SEQUENCE:
                                00B7  ;        BSB/JSB SCH$UNLOCK
                                00B7  ;
                                00B7  ; INPUT PARAMETERS:
                                00B7  ;        R0 - MUTEX ADDRESS
                                00B7  ;        R4 - PCB ADDRESS OF CURRENT PROCESS
                                00B7  ;
                                00B7  ; IMPLICIT INPUTS:
                                00B7  ;        SCH$GQ_MWAIT - MUTEXT WAIT QUEUE HEADER
                                00B7  ;        PCB OF CURRENT PROCESS
                                00B7  ;        MUTEX
                                00B7  ;
                                00B7  ; IMPLICIT OUTPUTS:
                                00B7  ;        *** TBS ***
                                00B7  ;
                                00B7  ; SIDE EFFECTS:
                                00B7  ;        *** TBS ***
                                00B7  ;
                                00B7  ;--
                                00B7
                                00B7  SCH$UNLOCK::                                    ; UNLOCK MUTEX
                                00B7          DSBINT  #IPL$_SYNCH                     ;;; RAISE TO SYNCH IPL
          0A A4     0C    91    00BD          CMPB    #DYN$C_PCB,PCB$B_TYPE(R4); STRUCTURE MUST BE PCB
                    63    12    00C1          BNEQ    NOTPCB                          ;
             0E A4  B7          00C3          DECW    PCB$W_MTXCNT(R4)                ;;; NOTE UNLOCK IN PCB
                    25    12    00C6          BNEQ    10$                             ;;; MORE STILL OWNED
      2F A4   29 A4       90    00C8          MOVB    PCB$B_PRIBSAV(R4),PCB$B_PRIB(R4) ; RESTORE SAVED BASE PRIORITY
        51    28 A4       90    00CD          MOVB    PCB$B_PRISAV(R4),R1             ; GET ORIGINAL PRIORITY
          0B A4     51    90    00D1          MOVB    R1,PCB$B_PRI(R4)                ; RESTORE IT
      00000000'GF   51    90    00D5          MOVB    R1,G^SCH$GB_PRI                 ; AND ANNOUNCE IT
52   00000000'GF    20    00 EA 00DC          FFS     #0,#32,G^SCH$GL_COMQS,R2; FIND PRIORITY OF NEXT COMPUTABLE PROCESS
             52     51    91    00E5          CMPB    R1,R2                           ; CHECK FOR DELAYED PREMPTION
                    03    1B    00E8          BLEQU   10$                             ; NO, CONTINUE
                                00EA          SOFTINT #IPL$_SCHED                     ; ELSE RESCHEDULE WHEN IPL DROPS
                    60    B7    00ED  10$:    DECW    MTX$W_OWNCNT(R0)                ;;; DECREMENT OWNERSHIP COUNT
                    31    18    00EF          BGEQ    EXITN                           ;;; EXIT IF NOT LAST
          2D 60     10    E7    00F1          BBCCI   #MTX$V_WRT,(R0),EXITN           ;;; EXIT IF NO WRITE IN PROGRESS
                                00F5                                                  ;;; OR PENDING
                    11    BB    00F5  UNLOCK: PUSHR   #^M<R0,R4>                      ;;; SAVE PCB ADDRESS
   53     00000000'GF     DE    00F7          MOVAL   G^SCH$GQ_MWAIT,R3               ;;; GET ADDRESS OF WAIT QUEU
             54     63    D0    00FE          MOVL    (R3),R4                         ;;; AND HEAD PCB
             52     02    9A    0101          MOVZBL  #PRI$_RESAVL,R2                 ;;; SET PRIORITY INCREMENT CLASS
             54     53    D1    0104  10$:    CMPL    R3,R4                           ;;; CHECK FOR END OF QUEUE
                    17    13    0107          BEQL    30$                             ;;; YES, DONE
          4C A4     6E    D1    0109          CMPL    (SP),PCB$L_EFWM(R4)             ;;; IS PROCESS WAITING FOR THIS MUTEX
                    0C    12    010D          BNEQ    20$                             ;;; NO, SKIP IT
                    64    DD    010F          PUSHL   (R4)                            ;;; SAVE FLINK
                  FEEC'   30    0111          BSBW    SCH$CHSE                        ;;; CHANGE TO EXECUTABLE STATE
                 08 A3    B7    0114          DECW    WQH$W_WQCNT(R3)                 ;;; DECREASE QUEUE LENGTH
                    10    BA    0117          POPR    #^M<R4>                         ;;; RESTORE FLINK
```

```
          E9   11  0119   467          BRB     10$                 ;;; AND CONTINUE
     54   64   D0  011B   468 20$:     MOVL    (R4),R4             ;;; FLINK ON TO NEXT PCB
          E4   11  011E   469          BRB     10$                 ;;; AND CONTINUE
          11   BA  0120   470 30$:     POPR    #^M<R0,R4>          ;;; RESTORE REGISTERS
                   0122   471 EXITN:   ENBINT                      ;;; ENABLE INTERRUPTS
               05  0125   472 EXIT:    RSB                         ; AND RETURN
                   0126   473
                   0126   474 NOTPCB:  BUG_CHECK NOTPCB,FATAL      ; STRUCTURE NOT PCB
                   012A   475          .END
```

```
BUG$_NOTPCB             ********   X   02
DYN$C_PCB             = 0000000C
EXIT                    00000125 R     02
EXITN                   00000122 R     02
IOC$GL_MUTEX            ********   X   02
IPL$_ASTDEL           = 00000002
IPL$_SCHED            = 00000003
IPL$_SYNCH            = 00000008
LKEX                    0000004C R     02
MTX$V_WRT             = 00000010
MTX$W_OWNCNT          = 00000000
NOTPCB                  00000126 R     02
PCB$B_PRI             = 0000000B
PCB$B_PRIB            = 0000002F
PCB$B_PRIBSAV         = 00000029
PCB$B_PRISAV          = 00000028
PCB$B_TYPE            = 0000000A
PCB$L_EFWM            = 0000004C
PCB$W_MTXCNT          = 0000000E
PCB$W_STATE           = 0000002C
PR$_IPL               = 00000012
PR$_SIRR              = 00000014
PRI$_RESAVL           = 00000002
PSL$S_IPL             = 00000005
PSL$V_IPL             = 00000010
RDWAIT                  0000007A R     02
SCH$CHSE                ********   X   02
SCH$C_MWAIT           = 00000002
SCH$GB_PRI              ********   X   02
SCH$GL_COMQS            ********   X   02
SCH$GL_RESMASK          ********   X   02
SCH$GQ_MWAIT            ********   X   02
SCH$IOLOCKR             0000003C RG    02
SCH$IOLOCKW             00000022 RG    02
SCH$IOUNLOCK            000000B0 RG    02
SCH$LOCKR               00000043 RG    02
SCH$LOCKW               00000029 RG    02
SCH$LOCKWNOWAIT         0000000A RG    02
SCH$RAVAIL              000000A0 RG    02
SCH$RWAIT               00000000 RG    02
SCH$UNLOCK              000000B7 RG    02
SCH$WAITL               ********   X   02
SS$_NORMAL            = 00000001
UNLOCK                  000000F5 R     02
WAITM                   0000007D R     02
WAITR                   00000088 R     02
WQH$W_WQCNT           = 00000008

                              +-----------------+
                              ! Psect synopsis  !
                              +-----------------+

PSECT name      Allocation          PSECT No.   Attributes
----------      ----------          ---------   ----------
.  ABS  .       00000000  (    0.)  00 (   0.)  NOPIC   USR   CON   ABS   LCL NOSHR NOEXE NORD   NOWRT NOVEC BYTE
$ABS$           00000000  (    0.)  01 (   1.)  NOPIC   USR   CON   ABS   LCL NOSHR   EXE   RD     WRT NOVEC BYTE
AEXENONPAGED    0000012A  (  298.)  02 (   2.)  NOPIC   USR   CON   REL   LCL NOSHR   EXE   RD     WRT NOVEC BYTE
```

```
                                        +--------------------------+
                                        ! Performance indicators !
                                        +--------------------------+

Phase                      Page faults    CPU Time       Elapsed Time
-----                      -----------    --------       ------------
Initialization                      33    00:00:00.03    00:00:00.33
Command processing                 874    00:00:00.22    00:00:01.66
Pass 1                             392    00:00:01.87    00:00:07.71
Symbol table sort                    0    00:00:00.25    00:00:00.27
Pass 2                              26    00:00:00.43    00:00:00.87
Symbol table output                  6    00:00:00.01    00:00:00.25
Psect synopsis output                4    00:00:00.01    00:00:00.01
Cross-reference output               0    00:00:00.00    00:00:00.00
Assembler run totals              1338    00:00:02.82    00:00:11.12

The working set limit was 1650 pages.
49006 bytes (96 pages) of virtual memory were used to buffer the intermediate code.
There were 50 pages of symbol table space allocated to hold 889 non-local and 12 local symbols.
475 source lines were read in Pass 1, producing 13 object records in Pass 2.
22 pages of virtual memory were used to define 21 macros.

                                        +----------------------------+
                                        ! Macro library statistics !
                                        +----------------------------+

Macro library name                          Macros defined
------------------                          --------------
_$11$DUA75:[SYS.OBJ]LIB.MLB;1                     12
_$11$DUA75:[SYSLIB]STARLET.MLB;2                   6
TOTALS (all libraries)                            18

993 GETS were required to define 18 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS$:MUTEX/OBJ=OBJ$:MUTEX TMP$:MUTEX.MAR+EXECML$/LIB
```

Obtaining and Releasing Mutexes

- Example - to obtain the paged pool mutex
  - In your routine

```
MOVAL   G^EXE$GL_PGDYNMTX,RØ
MOVL    G^SCH$GL_CURPCB,R4
JSB     G^SCH$LOCKR     ;read
           or
JSB     G^SCH$LOCKW     ;write
```

  - When returns, process has mutex
  - Process should remain at IPL 2 or greater while it owns a mutex

- Example - to release the paged pool mutex
  - In your routine

```
MOVAL   G^EXE$GL_PGDYNMTX,RØ
MOVL    G^SCH$GL_CURPCB,R4
JSB     G^SCH$UNLOCK
SETIPL #Ø          ; if no longer hold any mutexes
```

  - All mutex symbols defined in module SYSCOMMON, except for line printer mutex in LPDRIVER.

Asynchronous System Traps (ASTs)

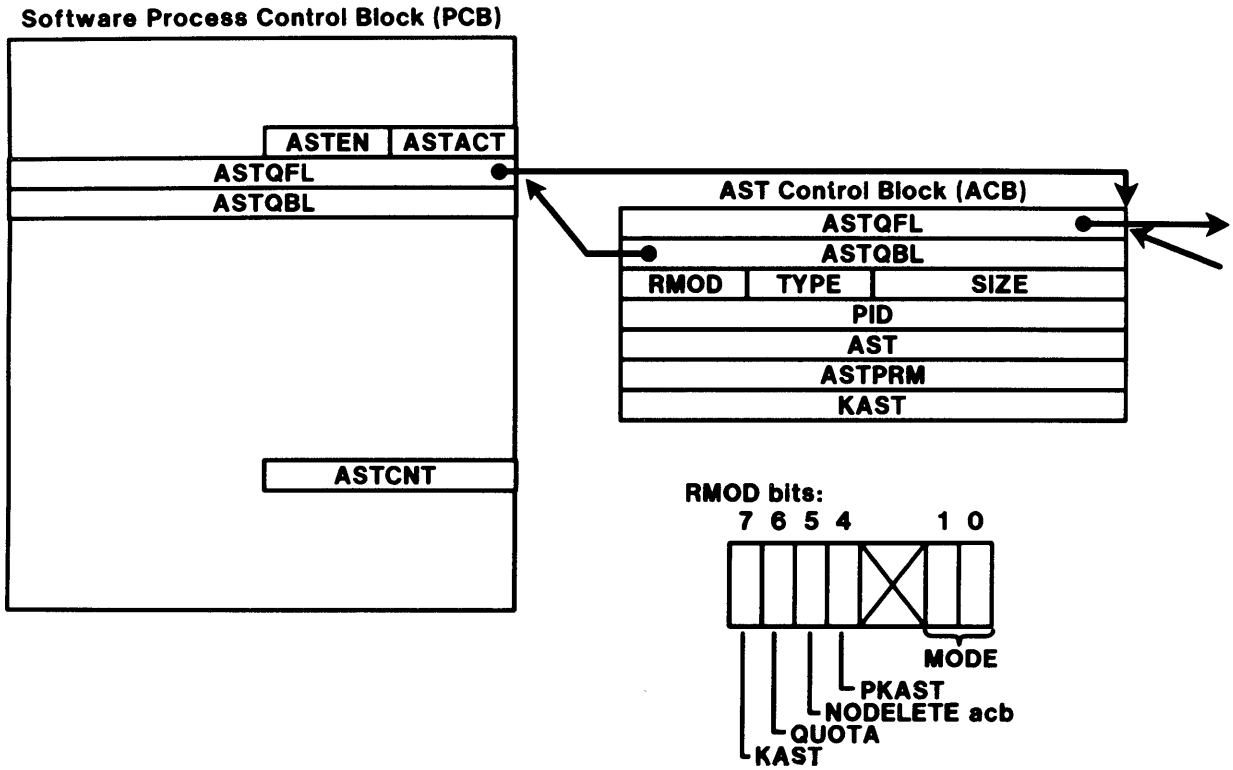


Figure 12 AST Queue off the Software PCB

- Provide an asynchronous tool for communication and synchronization
- AST Control Block (ACB) built when AST requested
- ACBs are queued to the software PCB when the AST is due
  - Queue is ordered by access mode

ASYNCHRONOUS SYSTEM TRAPS (ASTS)

- MECHANISM TO INITIATE THREAD OF EXECUTION
  - WITHIN A PROCESS
  - ASYNCHRONOUSLY TO OTHER ACTIVITY WITHIN PROCESS
  - FREQUENTLY TO NOTIFY PROCESS OF SOME EVENT
  - SOMETIMES TO EXECUTE PIECE OF SYSTEM CODE IN PROCESS'S CONTEXT

- THREAD OF EXECUTION INITIATED
  - AT A PARTICULAR ACCESS MODE
  - FREQUENTLY AS CALLED PROCEDURE
  - SOMETIMES AS SUBROUTINE OF IPL2 ASTDEL SERVICE ROUTINE

- "INTERRUPT" MOST PROCESS WAIT STATES

- DELIVERY TO ALL ACCESS MODES ENABLED BY DEFAULT

- ONLY ONE AST ACTIVE PER PROCESS PER ACCESS MODE

- ASSOCIATED SYSTEM SERVICES

$DCLAST	DECLARE AST
$ENQ[W]	ENQUEUE LOCK REQUEST
$GETDVI	GET DEVICE/VOLUME INFORMATION
$GETJPI	GET JOB/PROCESS INFORMATION
$GETSYI	GET SYSTEM INFORMATION
$QIO[W]	QUEUE I/O REQUEST
$SETIMR	ENQUEUE TIMER REQUEST
$SETAST	ENABLE/DISABLE AST DELIVERY
$SETPRA	SPECIFY POWER RECOVERY AST
$UPDSEC	UPDATE SECTION FILE ON DISK

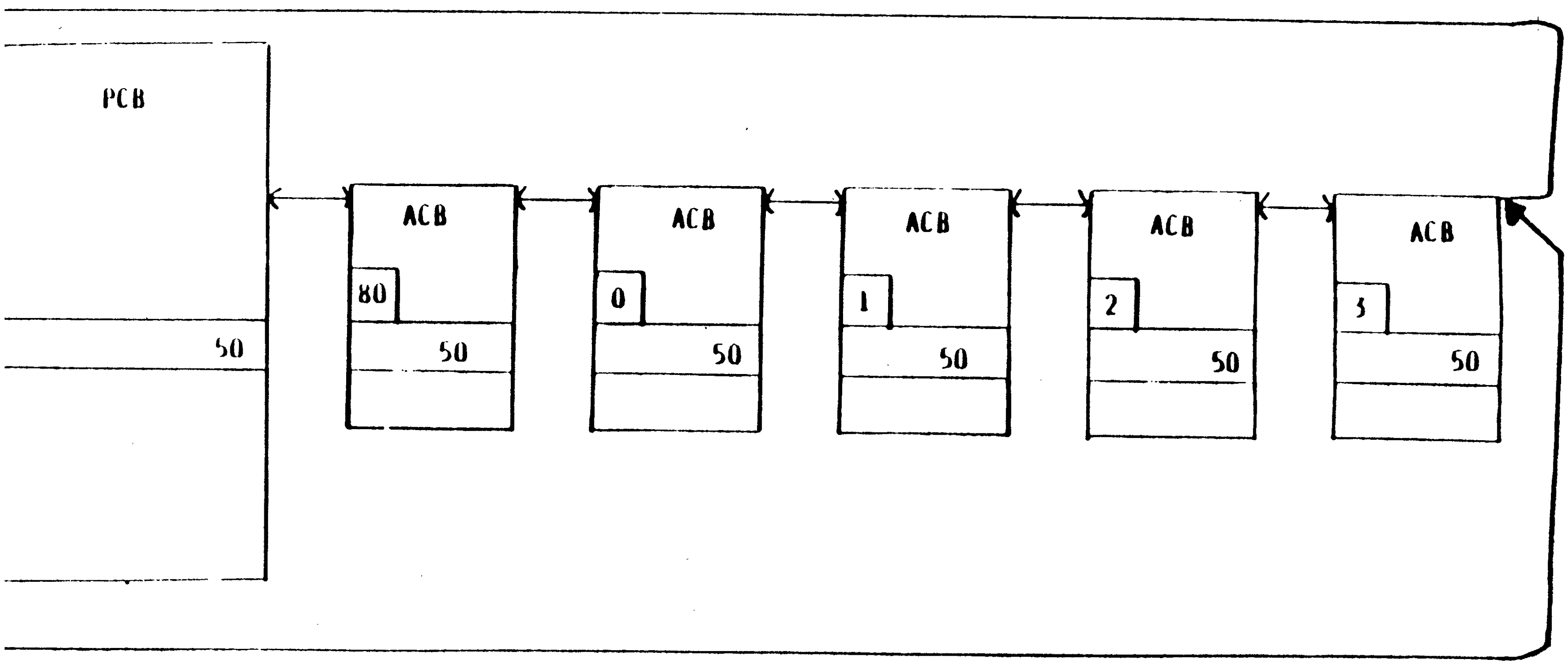
ARCHITECTURE FEATURES

- PR$_ASTLVL
- PHD$B_ASTLVL
- LDPCTX
- REI

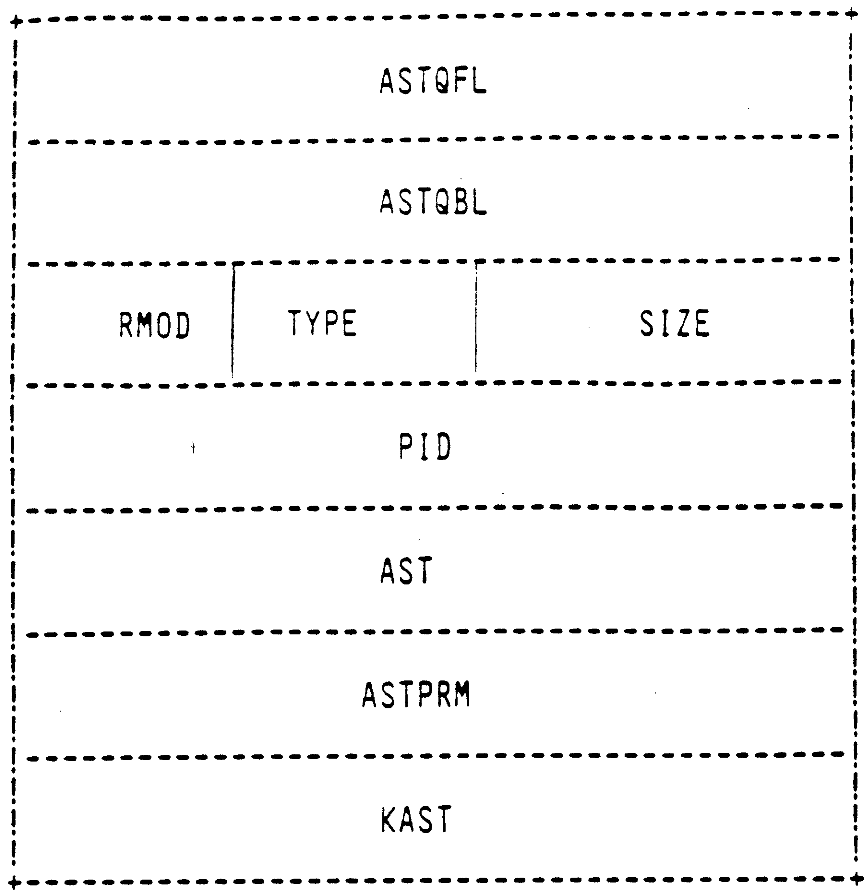
SOFTWARE PCB FIELDS ASSOCIATED WITH ASTS

PCB$L_ASTQFL	LIST HEADER FOR
PCB$L_ASTQBL	ENQUEUED ASTs
PCB$W_ASTCNT	AVAILABLE AST QUOTA
PCB$B_ASTACT	1 BIT FOR EACH ACCESS MODE (1 = AST ACTIVE)
PCB$B_ASTEN	1 BIT FOR EACH ACCESS MODE (1 = AST DELIVERY ENABLED)

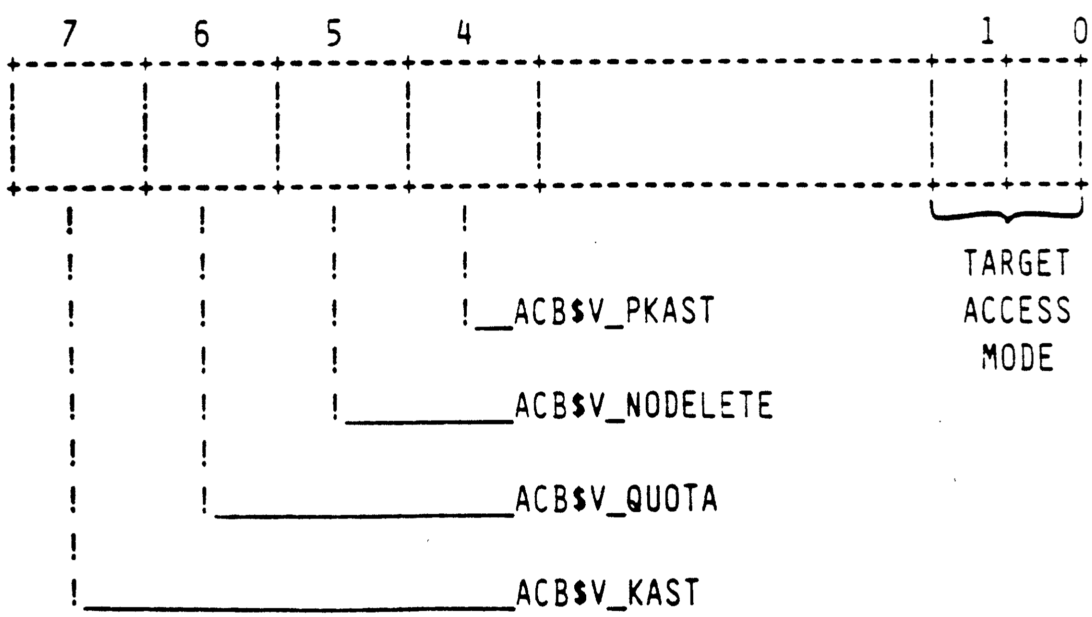
ACBs ARE ENQUEUED IN ACCESS MODE ORDER



AST CONTROL BLOCK



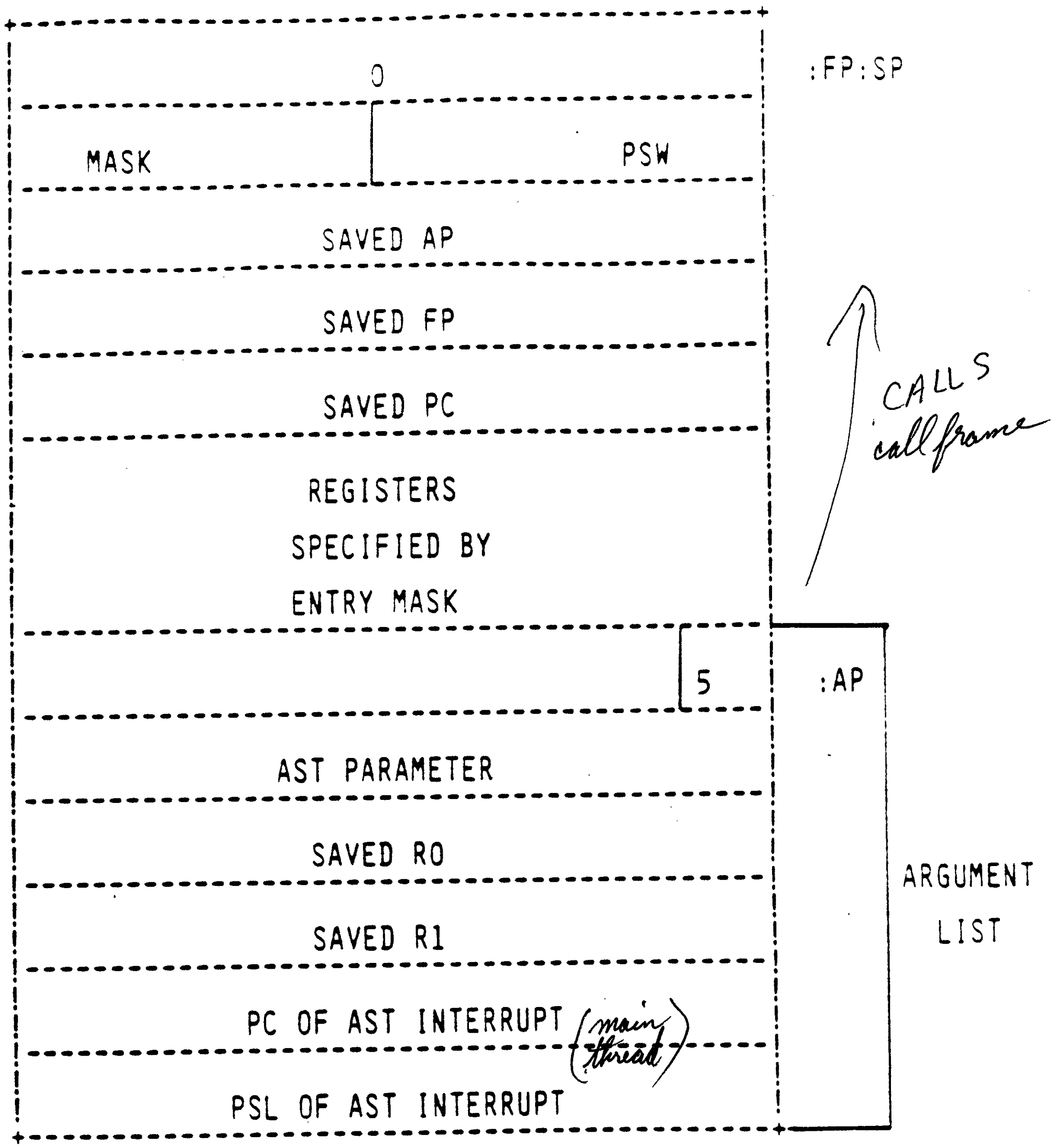
ACB$B_RMOD



SPECIAL KERNEL MODE ASTS

- CANNOT BE DISABLED THROUGH $SETAST
- QUEUED AT FRONT OF AST QUEUE
- DELIVERED THROUGH JSB AT IPL 2
- USED BY VMS EXEC AND UTILITIES
  - $GETJPI - READ INFORMATION ABOUT TARGET PROCESS
  - IOC$IOPOST - POST I/O COMPLETION IN PROCESS CONTEXT
  - EXE$POWERAST - QUEUE PROCESS-REQUESTED AST NOTIFICATION OF POWER RECOVERY
  - DELTA - READ/WRITE VIRTUAL MEMORY OF TARGET PROCESS
  - SDA (ONLINE) - READ VIRTUAL MEMORY OF TARGET PROCESS

AST ROUTINE CALL FRAME



REI Return from Exception or Interrupt

Operation:

```
tmp1 <- (SP)+;    ! Pick up saved PC
tmp2 <- (SP)+;    ! and PSL

if {tmp2<IS> EQLU 1 AND tmp2<IPL> EQLU 0} OR
   {tmp2<IPL> GTRU 0 AND tmp2<CUR_MOD>} NEQU 0} OR
   {tmp2<PRV_MOD> LSSU tmp2<CUR_MOD>} OR
   {tmp2<PSL_MBZ> NEQU 0} OR
   {tmp2<CUR_MOD> LSSU PSL<CUR_MOD>} OR
   {tmp2<IS> EQLU 1 AND PSL<IS> EQLU 0} OR
   {tmp2<IPL> GTRU PSL<IPL>} then {reserved operand fault;

if {compatibility mode implemented} then
begin
        if {tmp2<CM> EQLU 1} AND
           {{tmp2<FPD,IS,DV,FU,IV> NEQU 0} OR
            {tmp2<CUR_MOD> NEQU 3}} then {reserved operand fault;
end
else if {tmp2<CM> EQLU 1} then {reserved operand fault;

if PSL<IS> EQLU 1} then ISP <- SP         !save old stack pointer
                   else PSL<CUR_MOD>_SP <- SP;
if PSL<TP> EQLU 1 then tmp2<TP> <- 1;     !TP <- TP or stack TP
PC <- tmp1;
PSL <-tmp2;
if PSL<IS> EQLU 0 then
        begin
        SP <- PSL<CUR_MOD>_SP;            !switch stack
        if PSL<CUR_MOD> GEQU ASTLVL       !check for AST delivery
              then {request interrupt at IPL 2};
        end;
{check for software interrupts};
{clear instruction look-ahead}
```

AST Delivery

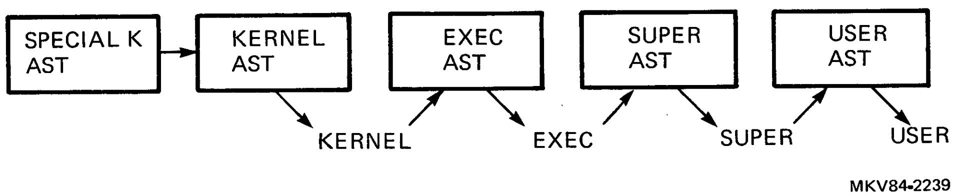


Figure 13 AST Delivery Order

- 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

AST Delivery Sequence

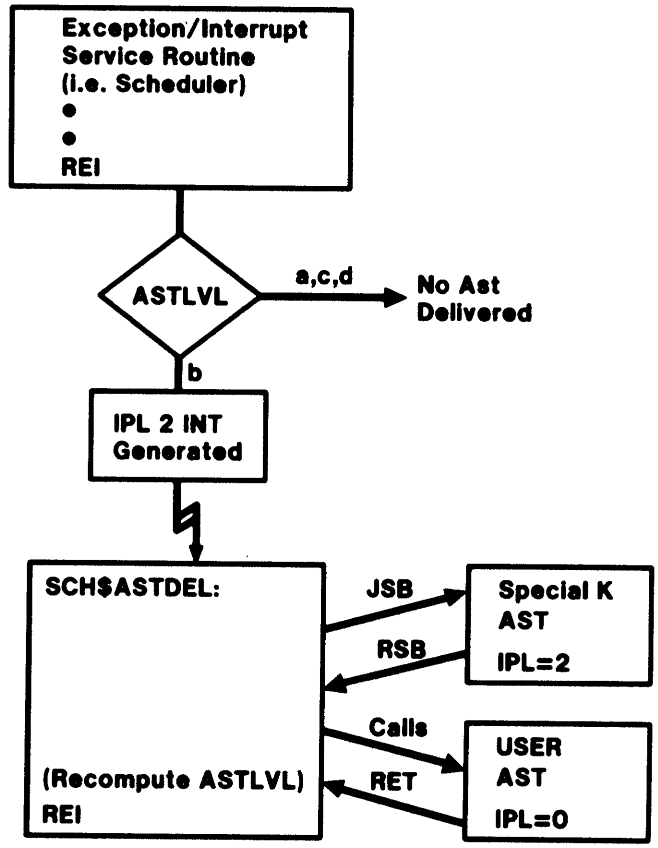


Figure 14 AST Delivery Sequence

Table 7 Rules for Selection of ASTs

Rule	Example
a) ASTLVL > new access mode	User AST (3) > kernel access mode (0)
b) ASTLVL ≤ new access mode	Super AST (2) ≤ super access mode (2)
c) Interrupt stack active	(IS) bit set in PSL
d) Final IPL ≥ 2	Process code at elevated IPL (≥2)

LOCK MANAGER

- SYNCHRONIZES SHARING OF RESOURCES
- RESOURCE - ANYTHING THAT CAN BE GIVEN A NAME
- CLUSTER DEVICE NAME
  - DERIVED FROM THE PATHWAY TO THE DEVICE
  - DEVICE NAME — NODE$DEV:
- SHARED RESOURCE - MUST HAVE UNIQUE NAME ACROSS THE CLUSTER
- DUAL PORTED DEVICE - MUST HAVE THE SAME NAME ACROSS THE CLUSTER

RESOURCES AND RESOURCE LOCKING

Definition of resources -- Any entity on VAX/VMS -- for example

- o Files
- o Data structures
- o Data bases
- o Anything that can be given a name and shared

Definition of locking

- o Lock -- a process's request to access a resource
- o Locks may be granted -- access permitted
- o Locks may be waiting -- access pending (while access is granted to another process)
- o Used to prevent such things as one process reading from a file while another is writing to it.

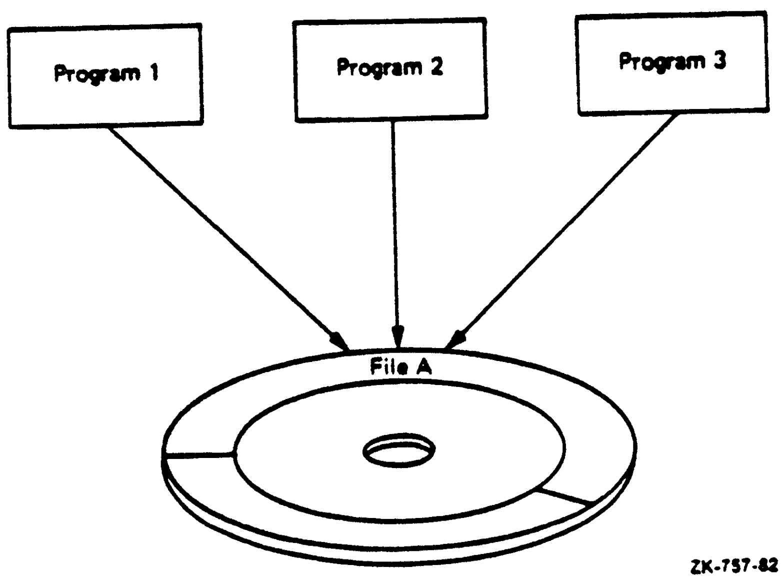


Figure 4-1 Several Programs Sharing a File

Lock Management System provided by VMS (Lock manager)

- o Allows cooperating processes to synchronize access to shared resources
- o Provides a a queuing mechanism
- o Consists of System Services
  - $ENQ -- enqueue a lock, return, notify caller when lock is granted by AST or Event flag
  - $ENQW -- enqueue a lock and wait until it is granted (LEF)
  - $DEQ -- dequeue a lock
  - $GETLKI -- get lock information

Requirements to enqueue a lock

1. Resource name -- indicates which resource is to be locked
2. Lock mode -- indicates how the resource may be shared
3. Address of lock status block -- receives completion status and lock identification (used for all future references to lock)

```
LKSB:   .BLKQ 1                              ; quadword to contain
                                             ; the lock status block
RESOURCE:
        .ASCID  /MY_FILE/                    ; the name of the resourc

          .
          .
          .

        $ENQW_S LKMODE=#LCK$K_PRMODE, -      ; protected read mode
        LKSB=LKSB, -
        RESNAM=RESOURCE
```

Example 4-1 A Simple Lock Request

Operation of the lock manager

The lock manager compares the lock mode of newly requested lock to the lock mode of other locks with the same resource name.

- o If no other lock on same resource -- lock is granted
- o If another process has compatible lock -- lock is granted
- o If another process has incompatible lock -- lock is placed in a wait queue for the resource
- o A process can change lock mode with $ENQ. Called lock conversion.
  - If requested conversion is compatible with existing locks -- conversion is granted
  - If requested conversion is incompatible with existing locks -- lock is place in a conversion queue until the existing incompatible lock is dequeued

Lock queues (on RSB)

- o GRANTED
  - Contains those locks that have been granted
- o WAITING
  - Contains those locks that are waiting to be granted
- o CONVERSION
  - Contains those locks that are granted at one mode and are waiting to be converted to higher lock mode

Table 4-1 The Six Lock Modes

Mode name	Description
LCK$K_NLMODE	NULL MODE. No access granted to the resource. Serves as an indicator of interest in a resource and is converted to higher modes before for access. It is quicker to convert an existing lock than to create a new lock.
LCK$K_CRMODE	CONCURRENT READ. Grants read access to resource. Permits others to read and write at same time.
LCK$K_CWMODE	CONCURRENT WRITE. Grants write access to resource. Permits others to read and write at same time.
LCK$K_PRMODE	PROTECTED READ. Grants read access to resource. Permits others to read. No writers are allowed. "share lock"
LCK$K_PWMODE	PROTECTED WRITE. Grants write access to the resource. Allows it to be shared with concurrent read mode. No other writers are allowed access. "update lock"
LCK$K_EXMODE	EXCLUSIVE. Grants write access to the resource and prevents it from being shared. "exclusive lock"

Table 4-2 Compatibility of Lock Modes

		Mode of Currently Granted Locks					
		NL	CR	CW	PR	PW	EX
Mode of Requested Lock	NL	yes	yes	yes	yes	yes	yes
	CR	yes	yes	yes	yes	yes	no
	CW	yes	yes	yes	no	no	no
	PR	yes	yes	no	yes	no	no
	PW	yes	yes	no	no	no	no
	EX	yes	no	no	no	no	no

Key to Lock Modes

NL -- Null lock
CR -- Concurrent read
CW -- Concurrent write
PR -- Protected read
PW -- Protected write
EX -- Exclusive lock

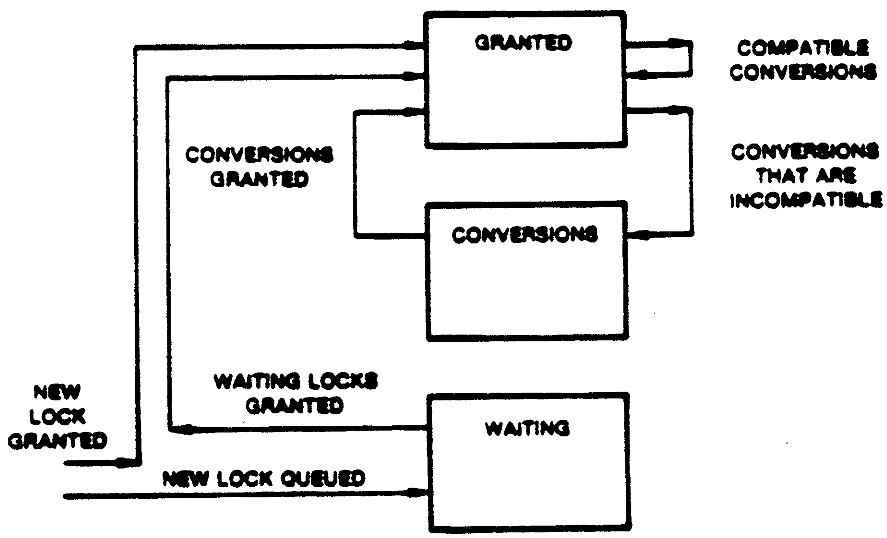
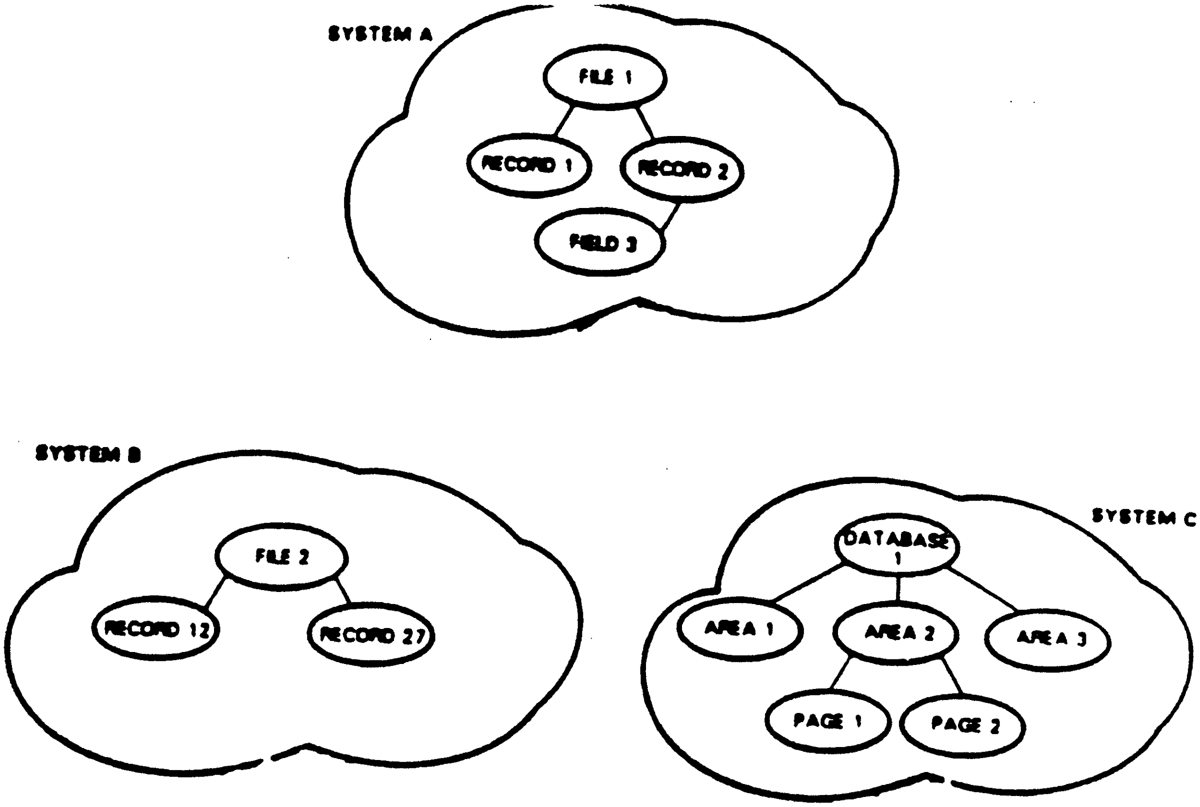


Figure 4-2 Three Lock Queues



RESOURCE	MASTER SYSTEM
FILE 1	A
FILE 2	B
DATABASE 1	C

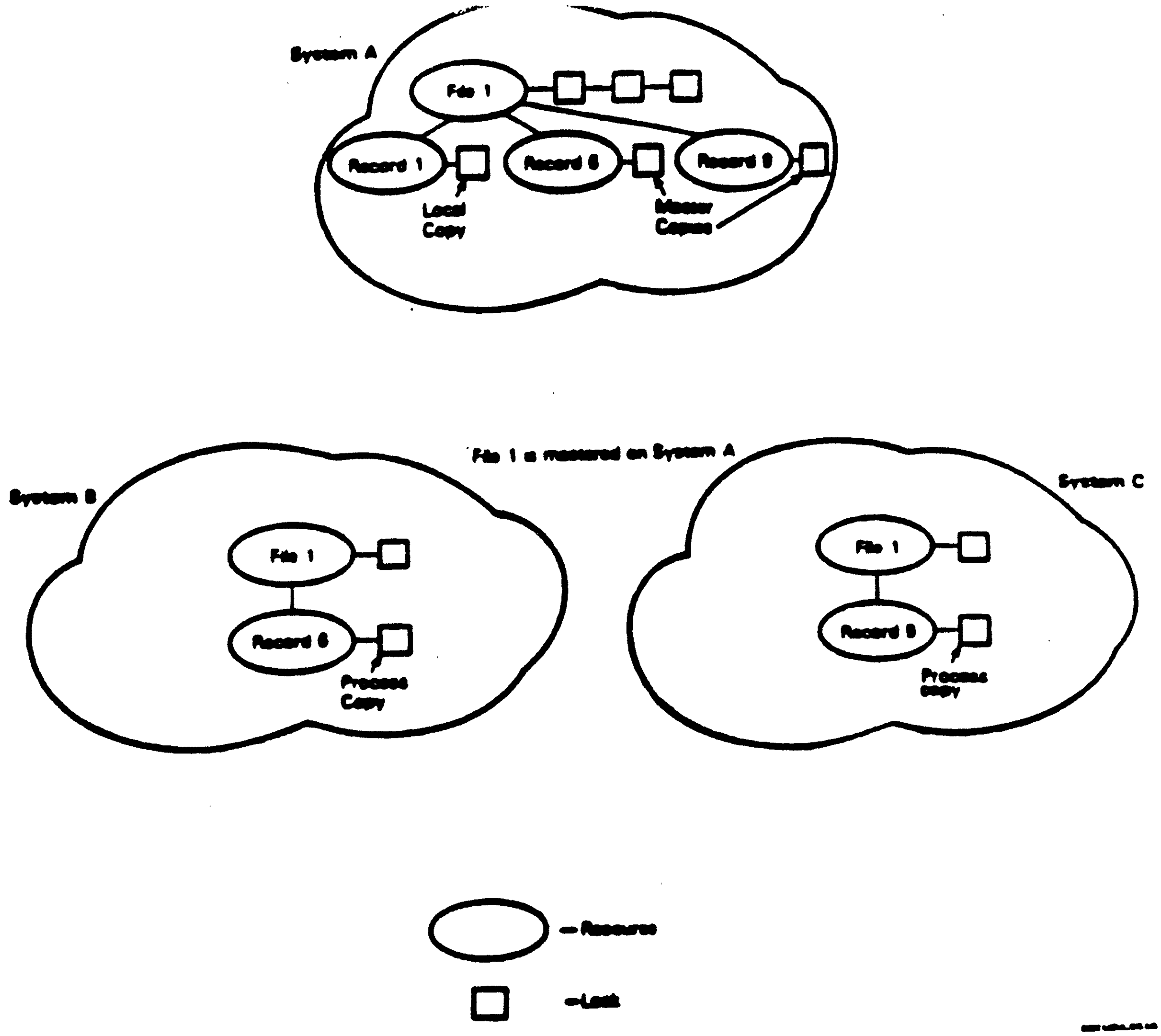
Resource trees -- the set of locks and resources that are common to a given root. Resource trees describe a root resource, related resources, and all locks on them.

Example -- On system A

1. FILE_1 is locked
2. RECORD_1 and RECORD_2 are locked under FILE_1
3. FIELD_3 is locked under RECORD_2

This entire structure is called a resource tree. Any given resource tree is entirely located on one system which is called the master system. (ie. It is said that this system is "mastering the resource")

This tends to distribute the locking activity throughout the cluster.

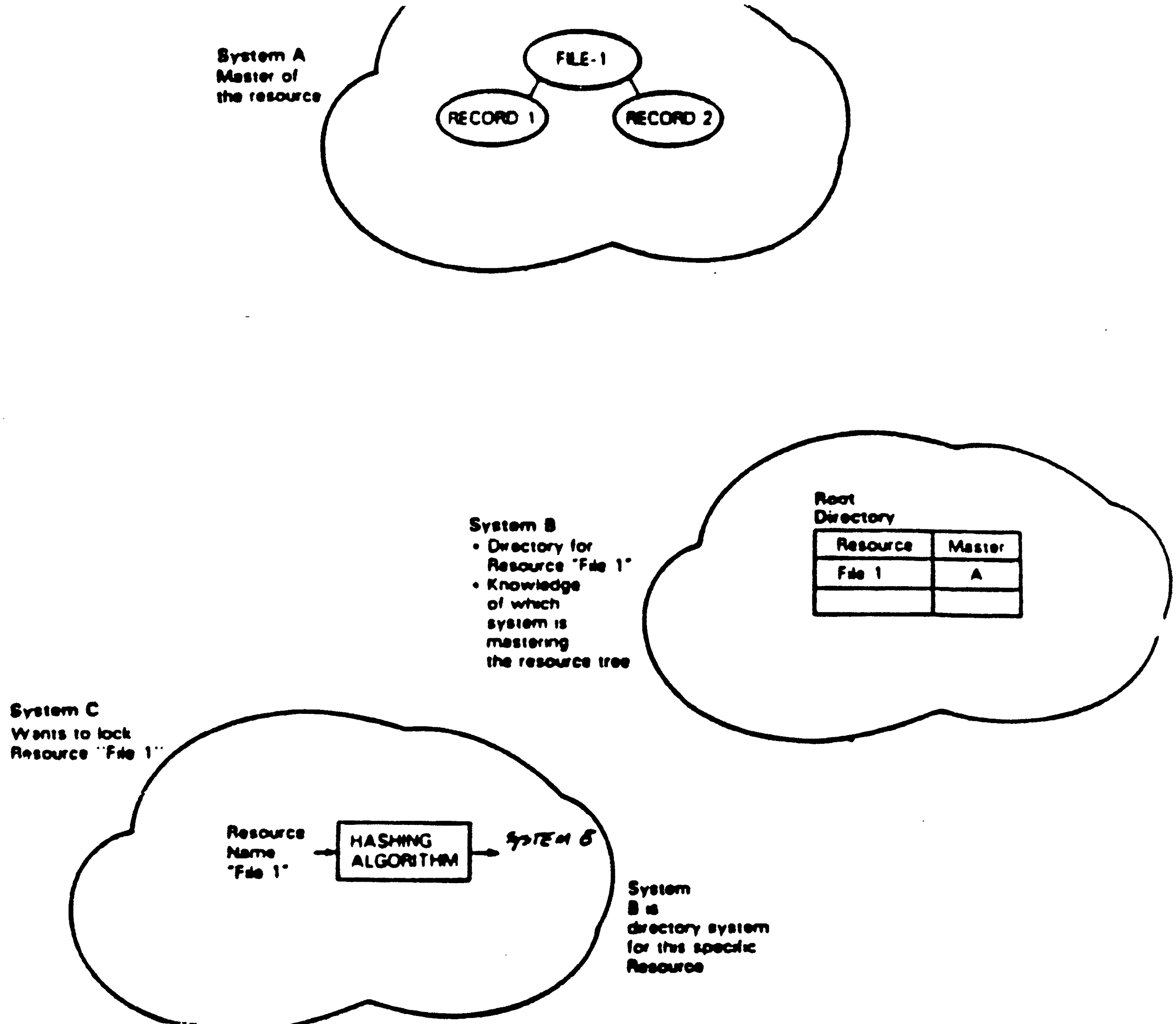


Only one system (the resource master) maintains complete information about a resource tree. All other systems only maintain information about locks that they have an interest in.

Example

1. System A is doing all locking services for entire cluster on the resource tree that it is mastering. It holds the master copies of locks held by remote systems.

2. Systems B and C only maintain information about locks that they have acquired. They have the local copies of locks that they hold. The resource master, if it is another system, holds a corresponding copy of that lock called the master copy.



The knowledge of which system is the master of a resource is distributed in the VAXcluster.

Each system maintains a partial directory that identifies which system is the master of certain resource trees.

A hashing algorithm is used to convert a resource name into the identity of the system that should be the directory system for that resource.

The hashing algorithm is chosen at the time of cluster formation and when nodes are added or removed from the VAXcluster. It must be the same on all nodes.

It provides a distributed lookup point to identify which system is mastering any given resource.

This directory is held in the lock database in memory and is not to be confused with a directory on a disk.

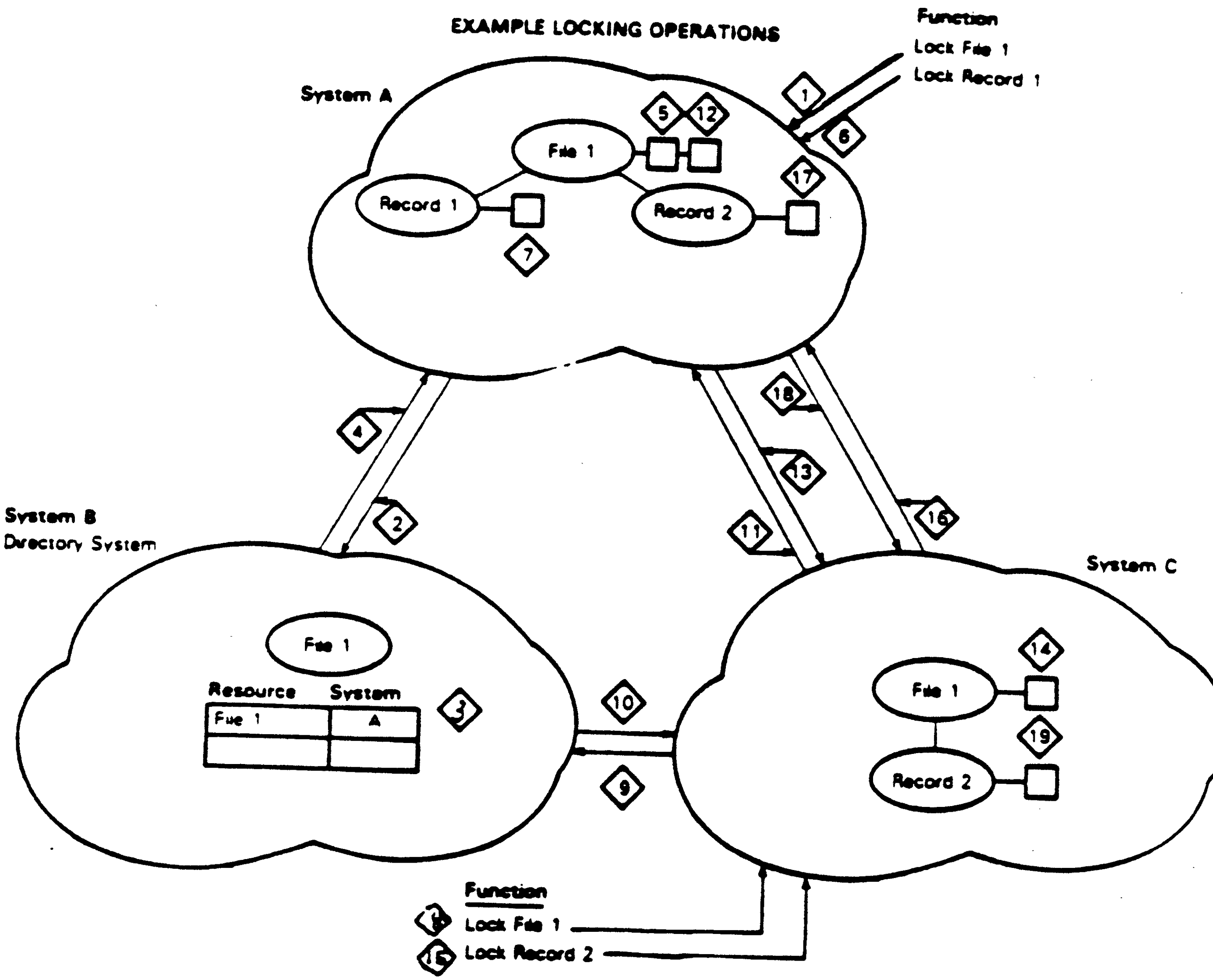


Figure 4-8 Example of Locking Operations

Annotation for Figure 4-8

A. FILE_1 locked on SYSTEM_A
   1. Request for a lock on FILE_1, the hash alglorithm indicates that SYSTEM_B should be the directory system for FILE_1.
   2. Message to Directory system -- "Who is mastering FILE_1?"
   3. No system is mastering FILE_1 so SYSTEM_A is entered into the root directory as master of FILE_1
   4. Message to SYSTEM_A "You are now mastering FILE_1"
   5. SYSTEM_A locks FILE_1

B. RECORD_1 locked on SYSTEM_A
   6. Request for a lock on RECORD_1
   7. Lock is granted -- no CI traffic since SYSTEM_A is mastering the resource

C. FILE_1 locked on SYSTEM_C
   8. Request for lock on FILE_1, the hash alglorithm indicates that SYSTEM_B should be the directory system for FILE_1.
   9. Message to Directory system -- "Who is mastering FILE_1?"
   10. Message to SYSTEM_C -- "SYSTEM_A is mastering FILE_1"
   11. Message to SYSTEM_A -- "Could I lock FILE_1?"
   12. Lock is granted
   13. Message to SYSTEM_C -- "Lock is granted"
   14. Lock data is also kept locally

D. RECORD_2 locked on SYSTEM_C
   15. Request for lock on RECORD_2
   16. SYSTEM_C goes directly to SYSTEM_A, since C already knows that A is mastering the resource
   17. Lock is granted
   18. Message to SYSTEM_C -- "Lock is granted"
   19. Lock data is also kept locally

41

Synchronizing Access Using the VAX/VMS Lock Manager

- Allows cooperating processes to synchronize access to shared resources
- Can be used system-wide or group-wide
- Lock manager is invoked with system services

```
$ENQ(W)   [efn], lkmode, lksb, [flags], [resnam], [parid],
          [astadr], [astprm], [blkast], [acmode], [nullarg]

$DEQ      lkid, [valblk], [acmode], [flags]
```

- Provides a queuing mechanism
- To allow for maximum sharing
  - Locking at various levels of granularity
  - Provides several lock modes
- Lock manager uses event flags to signify completion
- Lock manager uses ASTs
  - Kernel ASTs to perform asynchronous operations in context of the caller
  - Normal ASTs to notify of completion
- Detects locking deadlocks
- Limit on number of locks per process (ENQLM)
- Used by
  - VAX-11 RMS to implement file and record locking
  - Image activator and INSTALL utility to synchronize access to the known file database
  - Files-11 ODS-2 file system

Table 8 Data Structures Supporting the Lock Manager

Purpose	Data Structure	When Created	Size
Describe a lock on the system (owner PID, address of lock status block)	Lock Block (LKB)	When lock requested	Fixed
Catalog all locks on the system	Lock ID Table	At INIT	LOCKIDTBL LOCKIDTBL_MAX
Describe a resource being locked (resource name, lock queues, lock value block, etc.)	Resource Block (RSB)	When first lock placed on resource	Fixed
Given a resource name, locate the resource block	Resource Hash Table	At INIT	RESHASHTBL
Hold the listhead for the process lock queue	Software PCB	Process creation	Fixed

Can access the lock database in several ways:

- Given a resource name, use the resource hash table
- Given a lock ID, use the lock ID table
- To access all locks of a process, use the lock queue on the software PCB

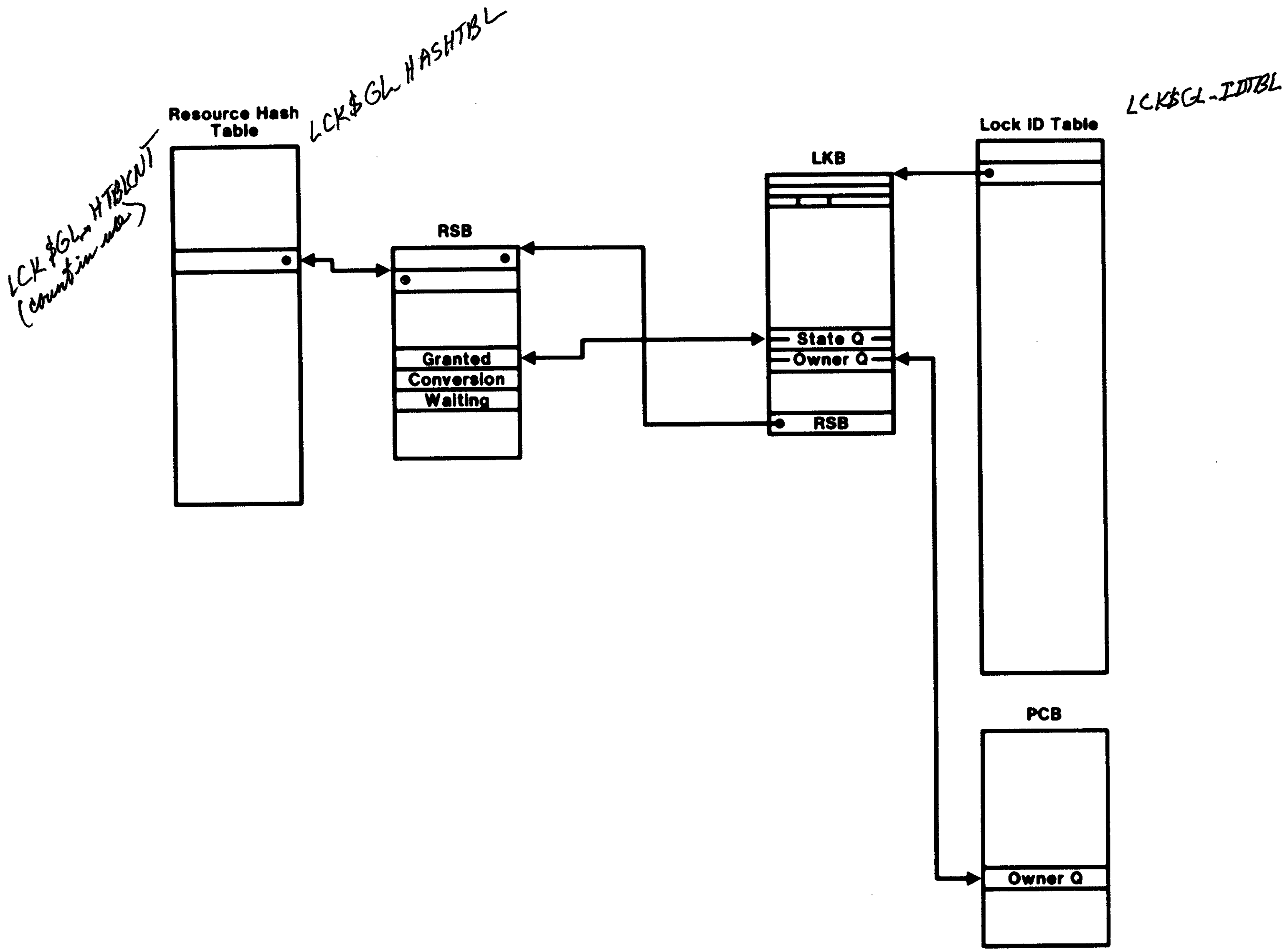


Figure 15 Relationships in the Lock Database

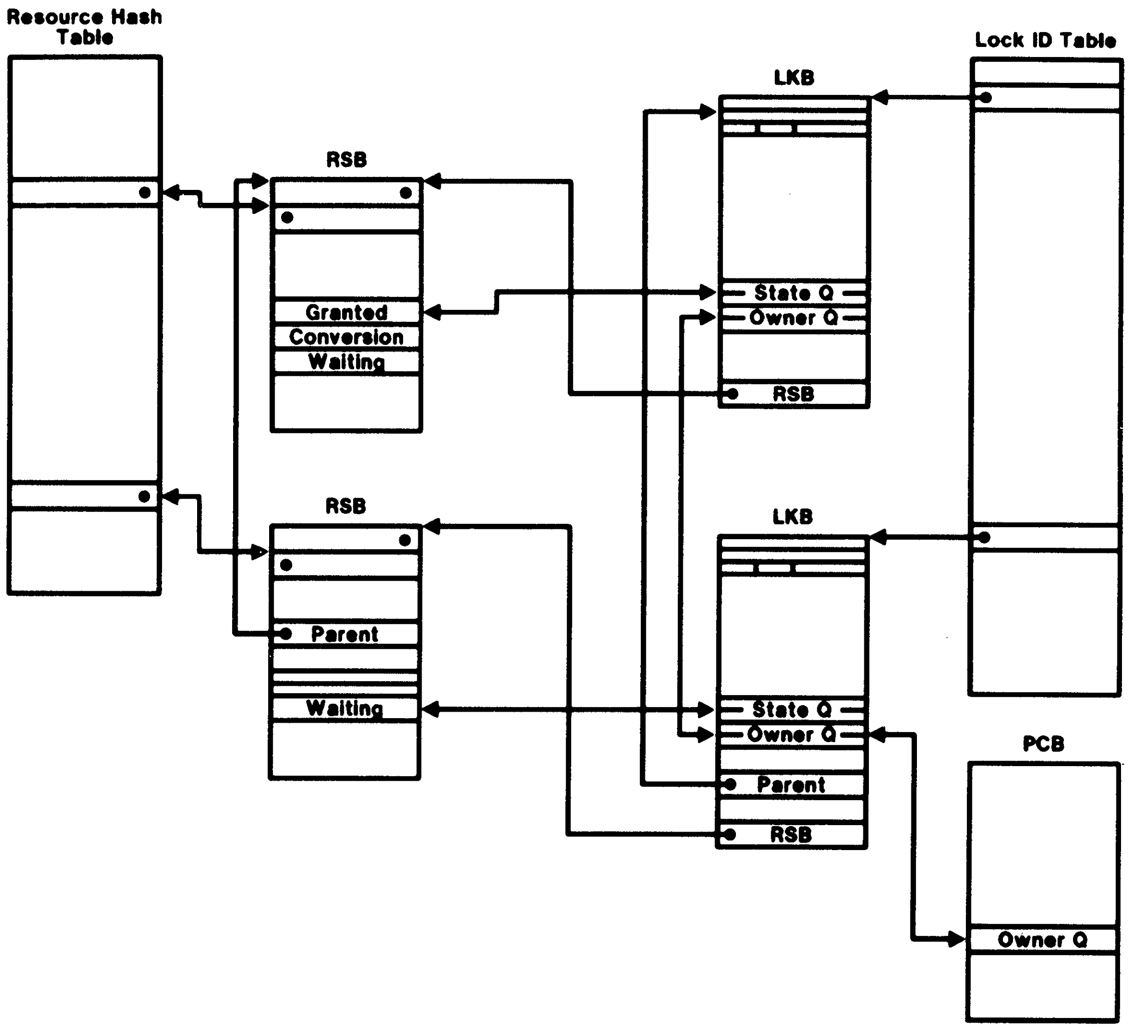


Figure 16 Relationships Between Locks and Sublocks

3.2 Search Sequence

1. PRIMARY EXCEPTION VECTOR for the MODE of the exception
2. SECONDARY EXCEPTION VECTOR for the MODE of the exception
3. All CALL FRAMES in the stack of the MODE of the exception
4. LAST CHANCE EXCEPTION VECTOR for the MODE of the exception

3.2.1 Setting up a Vector Address

Use the following system service macro call to set up an address in any of the three vector locations for one mode.

```
$SETEXV_S vector, addres, [acmode], [prvhnd]
```

Where the [] around an item means you do not have to specify a value because the macro definition provides a default for you.

Vector =	#0 to specify Primary Vector #1 to specify Secondary Vector #2 to specify Last Chance Vector
Address =	The address of your error handling routine. The routine must have an entry mask because the system is going to CALLG to it.
Acmode =	The mode you want to set the vector for. This mode is maximized with the mode you called the system service in.
Prvhnd =	The location to store the previous contents of the vector.

3.2.2 Setting up a Call Frame Address

Use the following instruction to fill in the first location in the currently active call frame.

```
MOVAL address, (FP)
```

Address =	The address of your error handling routine. The routine must have an entry mask.

3.3 Primary and Secondary Exception Vectors

			offset
Kernel Primary	0	CTL$AQ_EXCVEC::	00
Kernel Secondary	0		04
Executive Primary	0		08
Executive Secondary	0		0C
Supervisor Primary	0		10
Supervisor Secondary	0		14
User Primary	0		18
User Secondary	0		1C

Figure 11: Primary and Secondary Exception Vectors

3.4 Call Frame Specifying a Handler Address

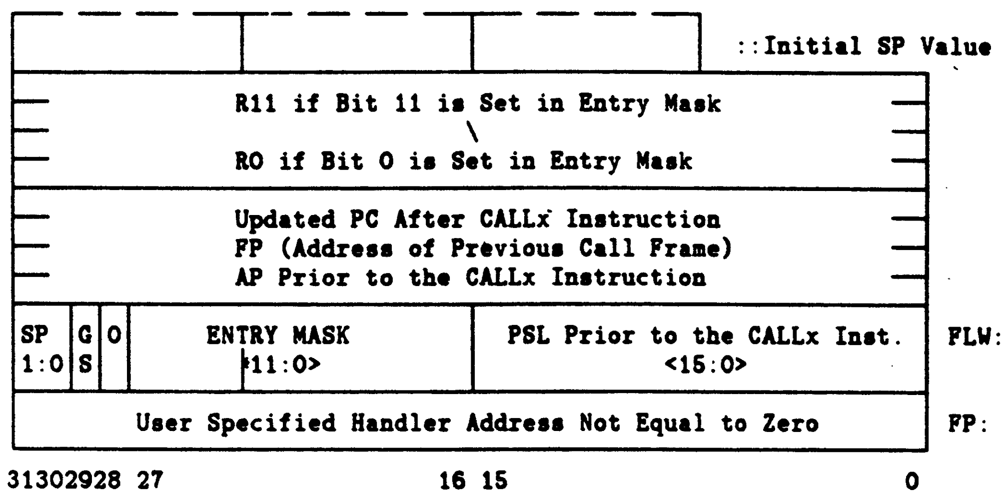


Figure 12: Call Frame

The Debugger creates a call frame with a handler before calling your image. DCL also creates a call frame with EXE$CATCH_ALL as the handler address.

3.5 Last Chance Exception Vectors

			offset
Kernel Last Chance	EXE$EXCPTN	CTL$AL_FINALEXC::	00 Bugcheck, Fatal
Executive Last Chance	EXE$EXCPTNE		04 Bugcheck, Nonfatal
Supervisor Last Chance	0		08
User Last Chance	EXE$CATCH_ALL		0C Exit Image

Figure 13: Last Chance Exception Vectors

2.2.2 System Control Block and Addresses

VECTORS (BITS 1:0)

	00	SERVICE ON KERNEL STACK UNLESS RUNNING ON INTERRUPT STACK
	01	SERVICE ON INTERRUPT STACK
**	10	SERVICE IN WCS, PASS BITS 15:2 TO MICRO PC
	11	HALT

SYSTEM CONTROL BLOCK (SCB)

0	UNUSED, RESERVED		
***4	MACHINE CHECK	ABORT/FAULT/TRAP, PROCESSOR & ERROR.INFO PUSHED ON.SP	EXE$AL_LOAVEC
***8	KERNEL STACK NOT VALID	ABORT	EXE$KERSTKNV
C	POWER FAIL	INTERRUPT	EXE$POWERFAIL
10	RESERVED/PRIVILEGED INSTRUCTION	FAULT, OP-CODES RESERVED TO DEC & PRIVILEDGED INST.	EXE$OPCDEC
14	CUSTOMER RESERVED INSTRUCTION	FAULT	EXE$OPCCUS
18	RESERVED OPERAND	FAULT/ABORT	EXE$ROPRAND
1C	RESERVED ADDRESSING MODE	FAULT	EXE$RADRMOD
20	ACCESS CONTROL VIOLATION	FAULT, VA CAUSING FAULT IS PUSHED ONTO STACK, REASON MASK	EXE$ACVIOLAT
24	TRANSLATION NOT VALID	FAULT, VA CAUSING FAULT IS PUSHED ONTO STACK, REASON MASK	MMG$PAGEFAULT
28	TRACE (TP)	FAULT, ENABLED BY T ON PREVIOUS INSTRUCTION	EXE$TBIT
2C	BREAKPOINT	FAULT	EXE$BREAK
30	COMPATIBILITY	TRAP, TYPE CODE PUSHED ON STACK (TABLE A)	EXE$COMPAT
34	ARITHMETIC	TRAP, TYPE CODE PUSHED ON STACK (TABLE B)	EXE$ARITH
38-3F	UNUSED, RESERVED		
40	CHMK	TRAP, OPERAND WORD PUSHED ONTO STACK SIGN EXTENDED	EXE$CMODKRNL
44	CHME	TRAP, OPERAND WORD PUSHED ONTO STACK SIGN EXTENDED	EXE$CMODEXEC
48	CHMS	TRAP, OPERAND WORD PUSHED ONTO STACK SIGN EXTENDED	EXE$CMODSUPR
4C	CHMU	TRAP, OPERAND WORD PUSHED ONTO STACK SIGN EXTENDED	EXE$CMUDOSER

50	SBI SILO COMPARE
54	CRD/RDS
*58	SBI ALERT
*5C	SBI FAULT
*60	CPU TIMEOUT (VMS: ASYNCHRONOUS WRITE TIMEOUT)
61-83	UNUSED, RESERVED

84	SOFTWARE LEVEL 1
88-8C	SOFTWARE LEVEL 2-3
90-BC	SOFTWARE LEVEL 4-F
C0	INTERVAL TIMER
C4-E4	UNUSED, RESERVED
*F8	CNSL RECEIVE INTR
*FC	CNSL TRANSMIT INTR
FF	UNUSED, RESERVED
*100-13C	SBI REQ 4
*140-17C	SBI REQ 5
*180-1BC	SBI REQ 6
*1C0-1FC	SBI REQ 7

* These offsets are 11/780-specific.
** Interrupt serviced in WCS.
Go to 10E0 which contains a RETURN1 unless changed.
*** Vector must select interrupt stack

REASON MASK BIT BREAKDOWN

BIT	VALUE	MEANING
0	0	PROTECTION VIOLATION
	1	LENGTH VIOLATION
	NA	NOT USED ON TRANSLATION NOT VALID (PAGE FAULT)
1	0	NORMAL MEMORY REFERENCE
	1	REFERENCE TO A PTE
2	0	READING
	1	WRITING

TABLE B

VAX-11 Native Mode Codes

CODE	CONDITION
1	INTEGER OVERFLOW TRAP
2	INTEGER DIVIDE BY ZERO TRAP
3	FLOATING OVERFLOW TRAP
4	FLOATING DIVIDE BY ZERO TRAP
5	FLOATING UNDERFLOW TRAP
6	DECIMAL STRING OVERFLOW TRAP
7	DECIMAL STRING DIVIDE BY ZERO TRAP

TABLE A

Compatibility Mode Codes

CODE	CONDITION
0	RESERVED OP-CODE
1	BREAKPOINT
2	IOT
3	EMT
4	TRAP
5	ILLEGAL INSTRUCTION
6	ODD ADDRESS

BR (8–7)	TR (6–2)	00 (1–0)

PERIPHERAL INTERRUPT VECTOR

MKV84-2563

Figure 5: System Control Block and Addresses

EXCEPTIONS AND CONDITION HANDLING

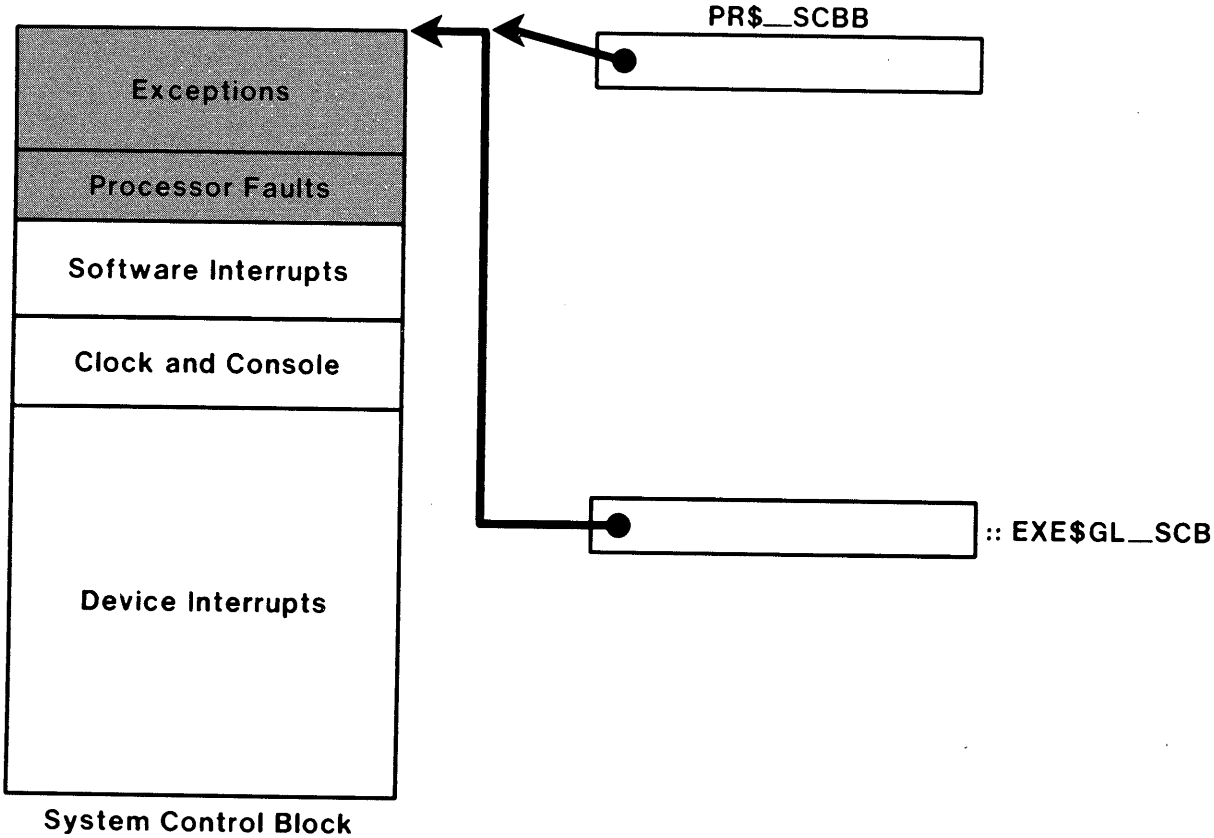


Figure 17 Exceptions and the SCB

- Exceptions are serviced by system routines
- Exception Service Routines (ESRs) are dispatched through the SCB

Exception and Interrupt Dispatching

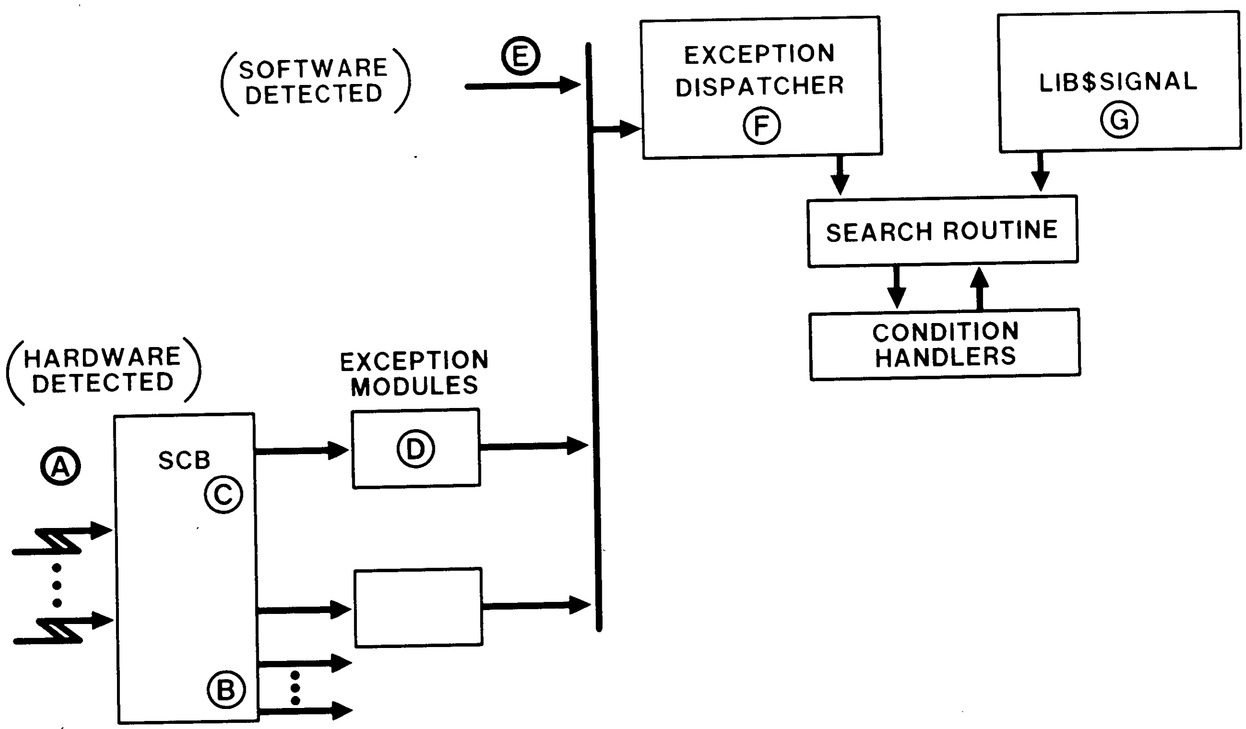


Figure 18 Exception and Interrupt Dispatching

Notes on Figure 18

A. PSL, PC and Ø to 2 longwords pushed onto stack

B. Exceptions and interrupts always handled by VMS (for example, page fault)

C. Exceptions that user may handle (for example, access violation)

D. These exception routines complete the signal array by pushing "SS$exception_name" and "N" (total of longwords now in signal array) onto the stack.

E. Detected and signaled by executive

F. The exception dispatcher

   1. Builds mechanism array and argument

   2. Invokes the search routine. Search order is:

      a. Primary exception
      b. Secondary exception
      c. Call frames
      d. Last chance

G. Alternate condition-handling mechanism

   1. Signaled by RTL or a user calling LIB$SIGNAL or LIB$STOP

   2. Search mechanism - same as (F)-2.

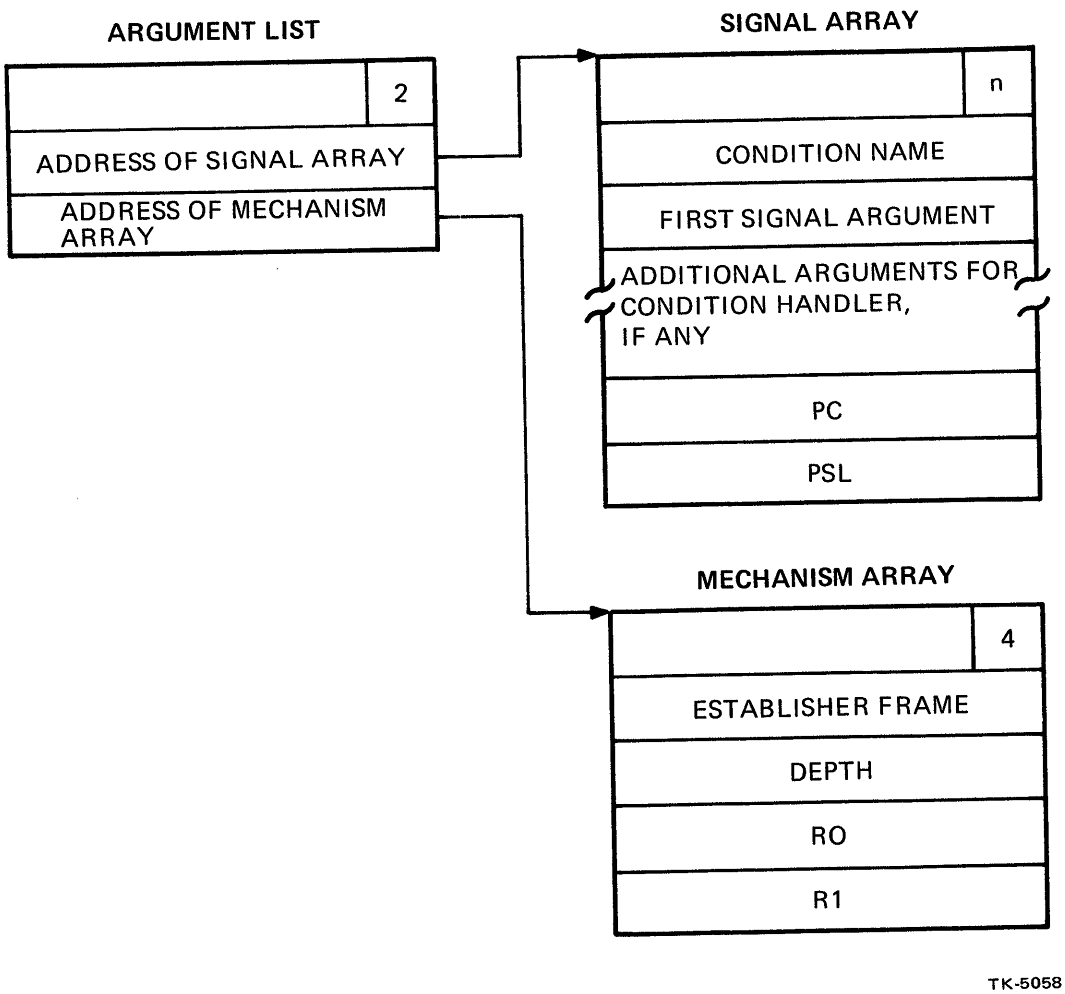


Figure 19 Condition Handler Argument List

HOW A USER EXECUTES PROTECTED CODE

Table 9 Executing Protected Code

Function	Implementation	Name
Protect memory from read/write	Hardware-maintained access modes	Kernel, executive, supervisor, user
Change access mode	Instruction	CHMx, REI
Enter system service, RMS, user-written system service	Call --> instruction	CALL_x --> CHMx

Access Mode Transitions

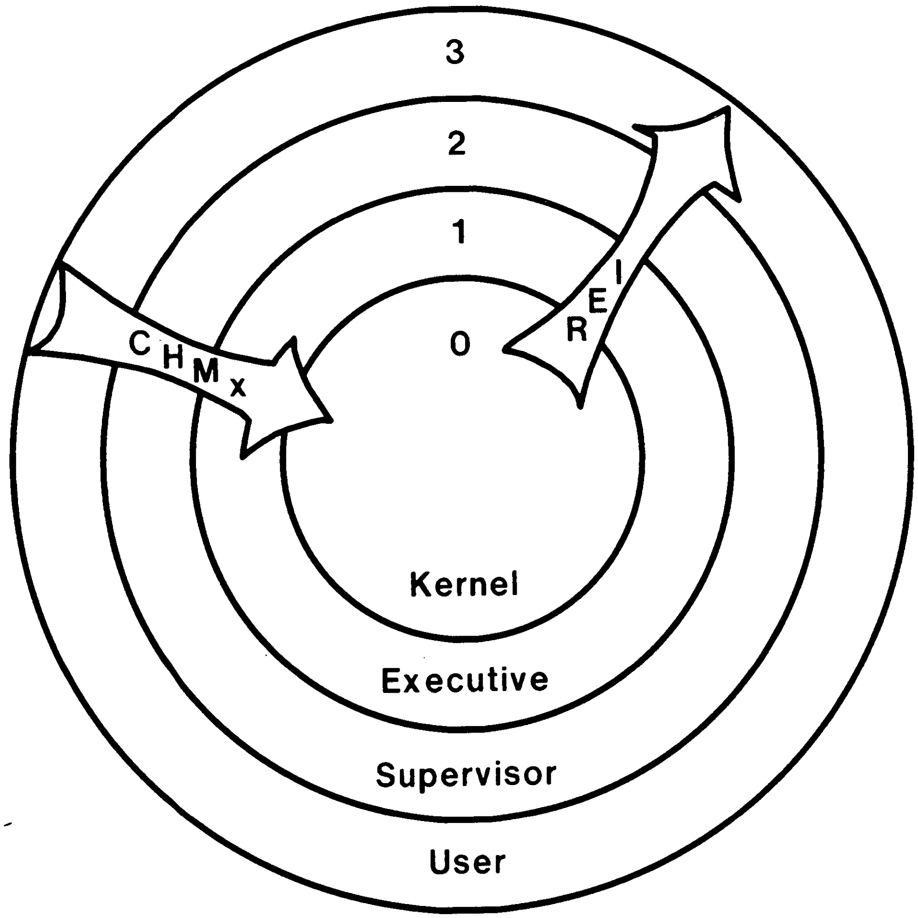


Figure 20 Access Mode Transitions

CHMx:

- 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

CHMx and REI Instructions

CHMx code-number

- Stack pointer switches to new mode
- PSL, PC and sign-extended code-number pushed onto stack

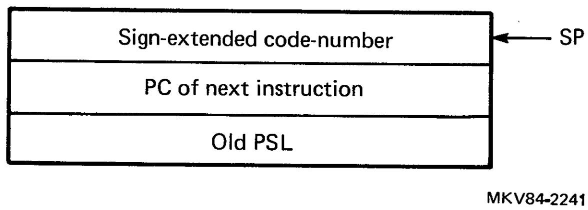


Figure 21 Stack After CHMx Exception

- PSL zeroed (except for IPL, Current Mode, Previous Mode)
- Current mode of PSL moved to previous mode field
- Current mode changed to new mode
- New PC taken from system control block (SCB)
- Code-number determines routine to execute in new mode

REI

- Replaces current PC and PSL with two longwords popped from the stack. Before doing so,
  - Various checks are made to protect the integrity of the system.
  - Checks for pending ASTs.
  - Checks for pending software interrupts.
  - After placing the PC and PSL in temporary registers, the SP is switched to the appropriate access mode based on the PSL current mode field.

REI Is Used in Various Situations

- To provide user-initiated access to system code and data:

```
CHMx code-number
     .
     .
     .
REI
```

- To switch to compatibility mode:

```
PUSHL PSL (Bit 31 set)
PUSHL PC
REI
```

- To dismiss any other exception
- To service and dismiss a hardware interrupt:

```
Hardware Interrupt (IPL 16 through 31)
     .
     .
     .
REI
```

- To service and dismiss a software interrupt:

```
Software Interrupt (IPL 1 through 15)
     .
     .
     .
REI
```

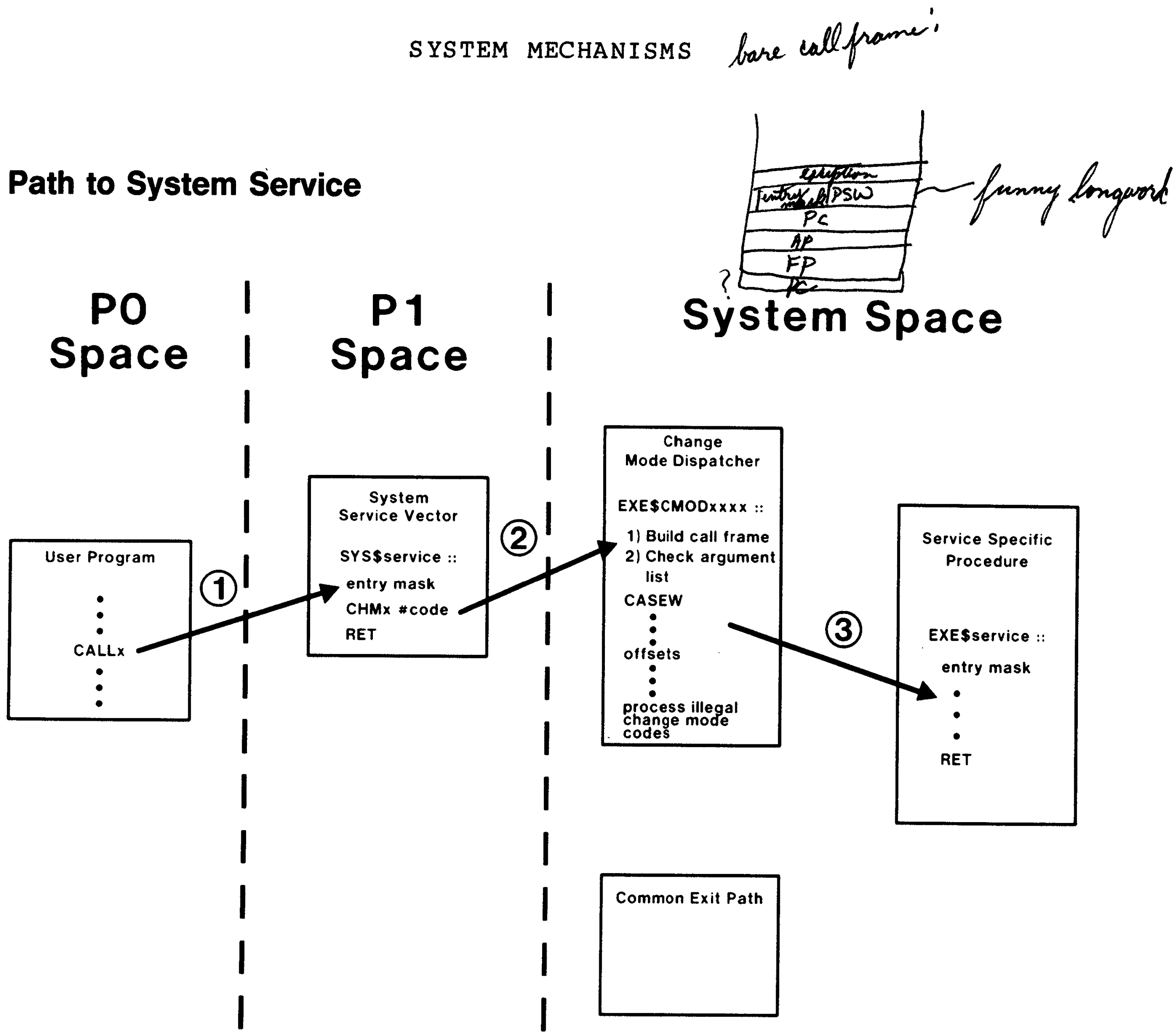


Figure 22 Path to System Service

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

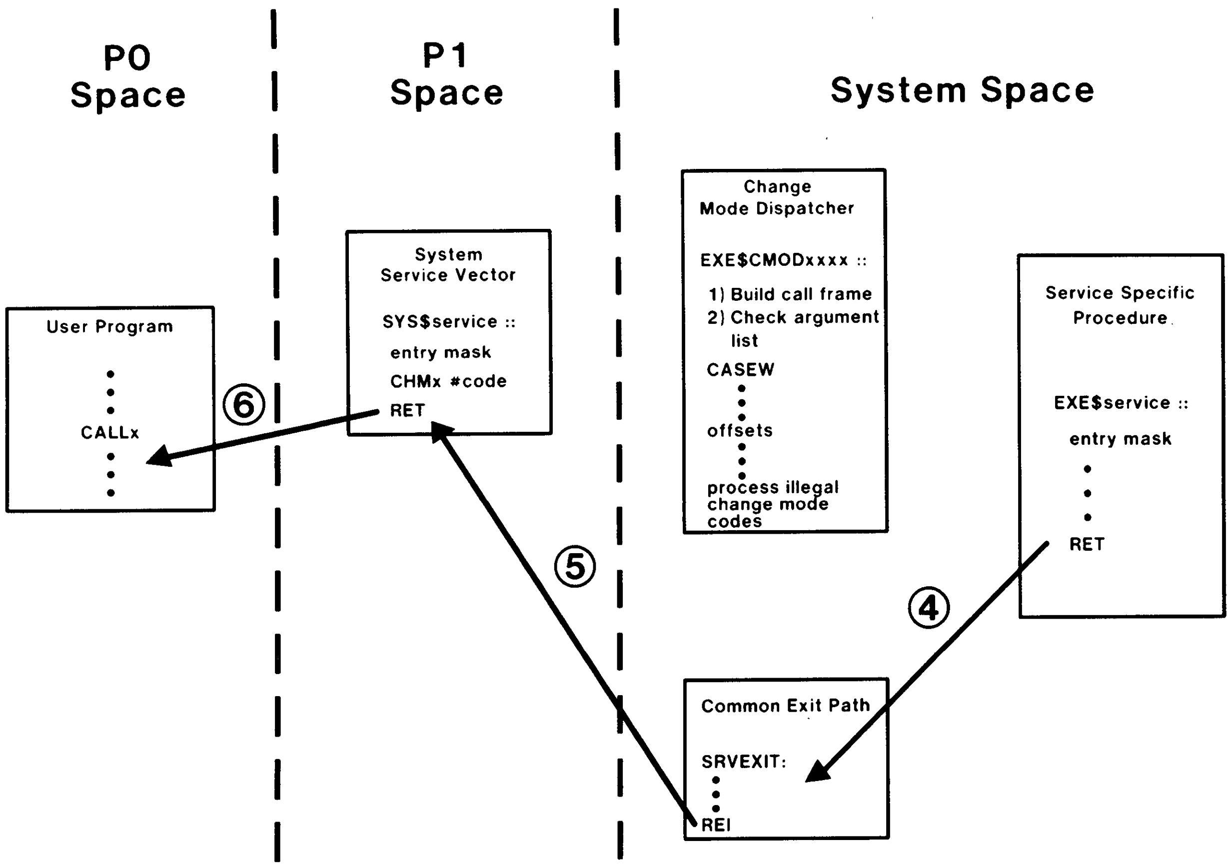


Figure 23 Return from System Service

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 original CALL

Nonprivileged System Service

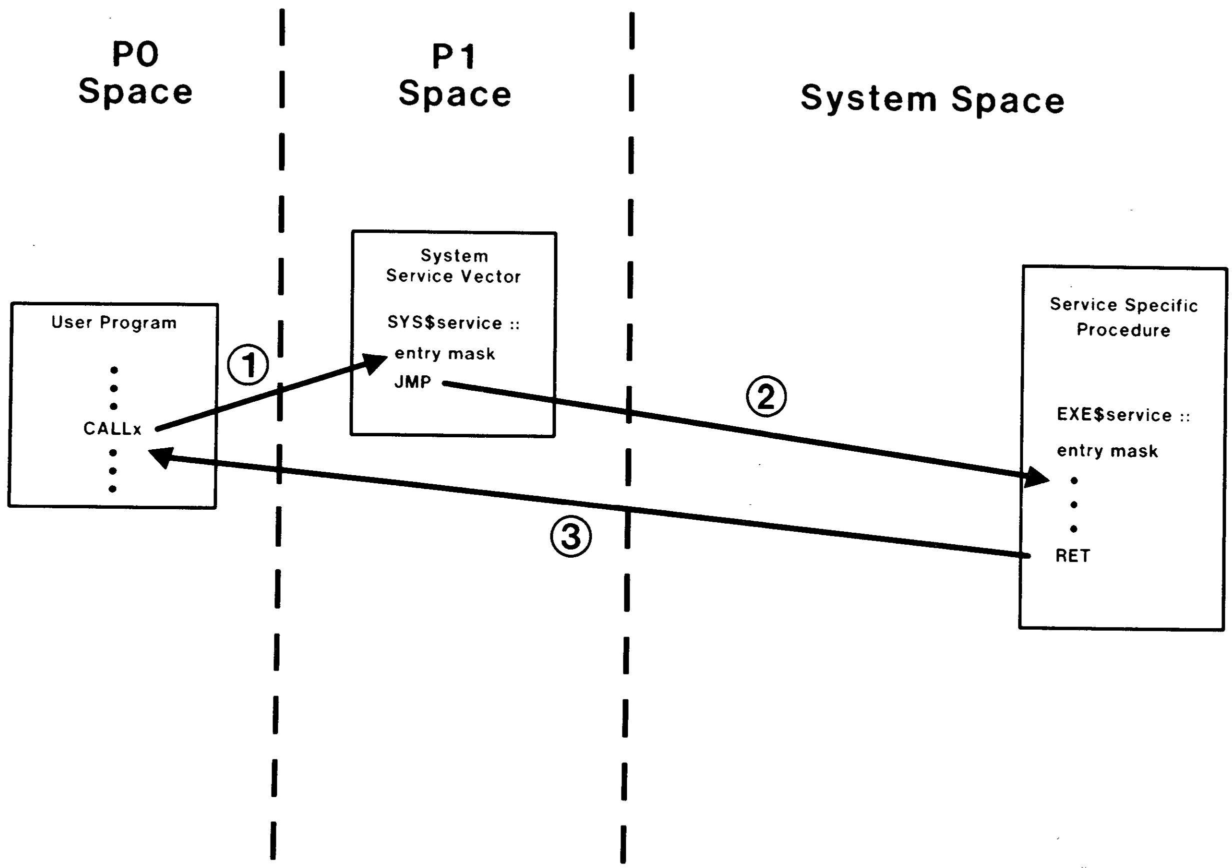


Figure 24 Nonprivileged System Service

1. Invoked with a CALL statement.

2. System services that do not require a change of access mode have a simpler control passing sequence.

   - $FAO
   - Timer conversion services

3. These services are not checked by SRVEXIT for error status codes.

Path to RMS

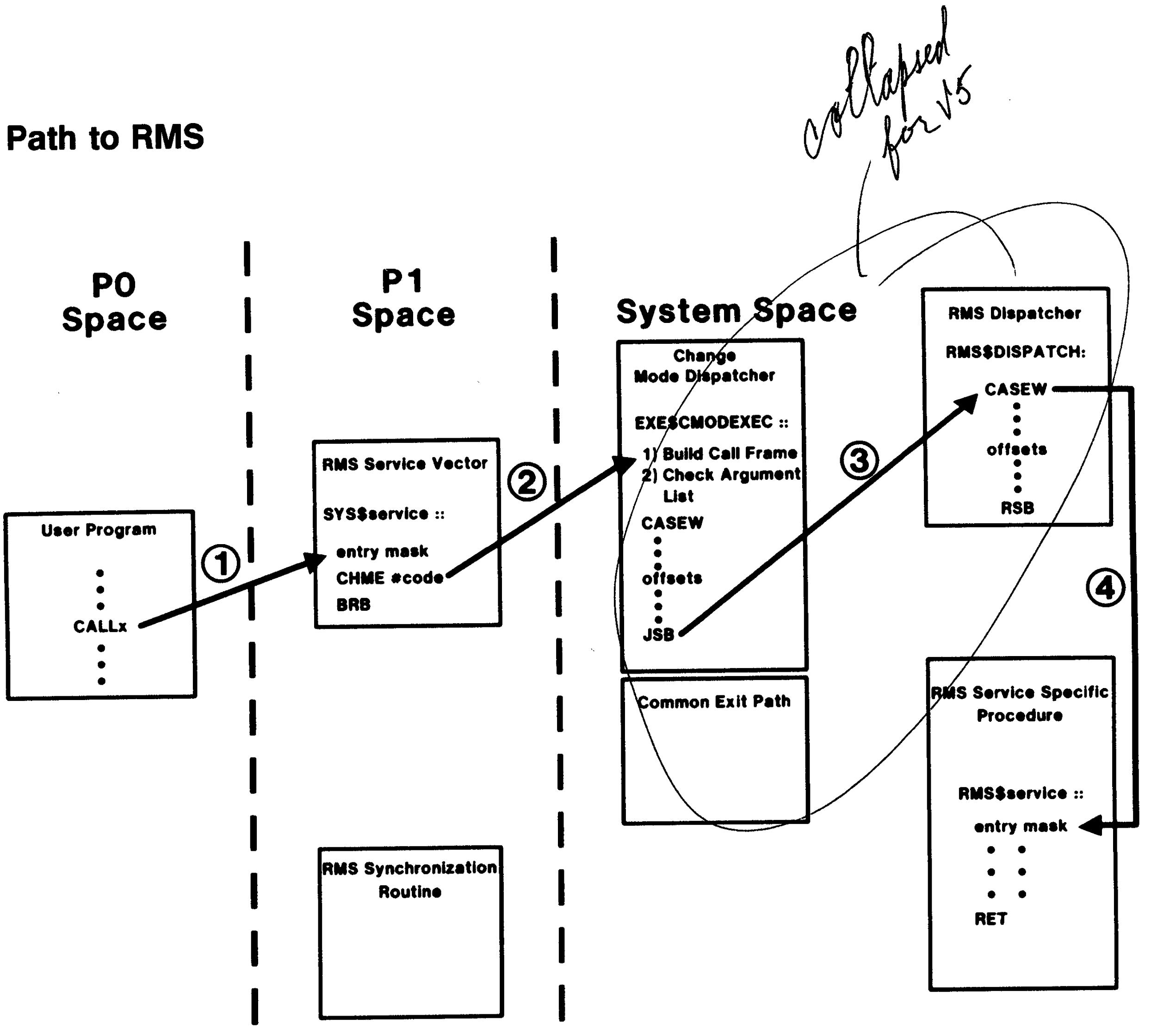


Figure 25 Path to RMS

1. Same path as executive mode system service

2. Same as 1

3. Falls off end of system service case table, so JSB to RMS case table

4. Dispatch to RMS procedure

Return from RMS

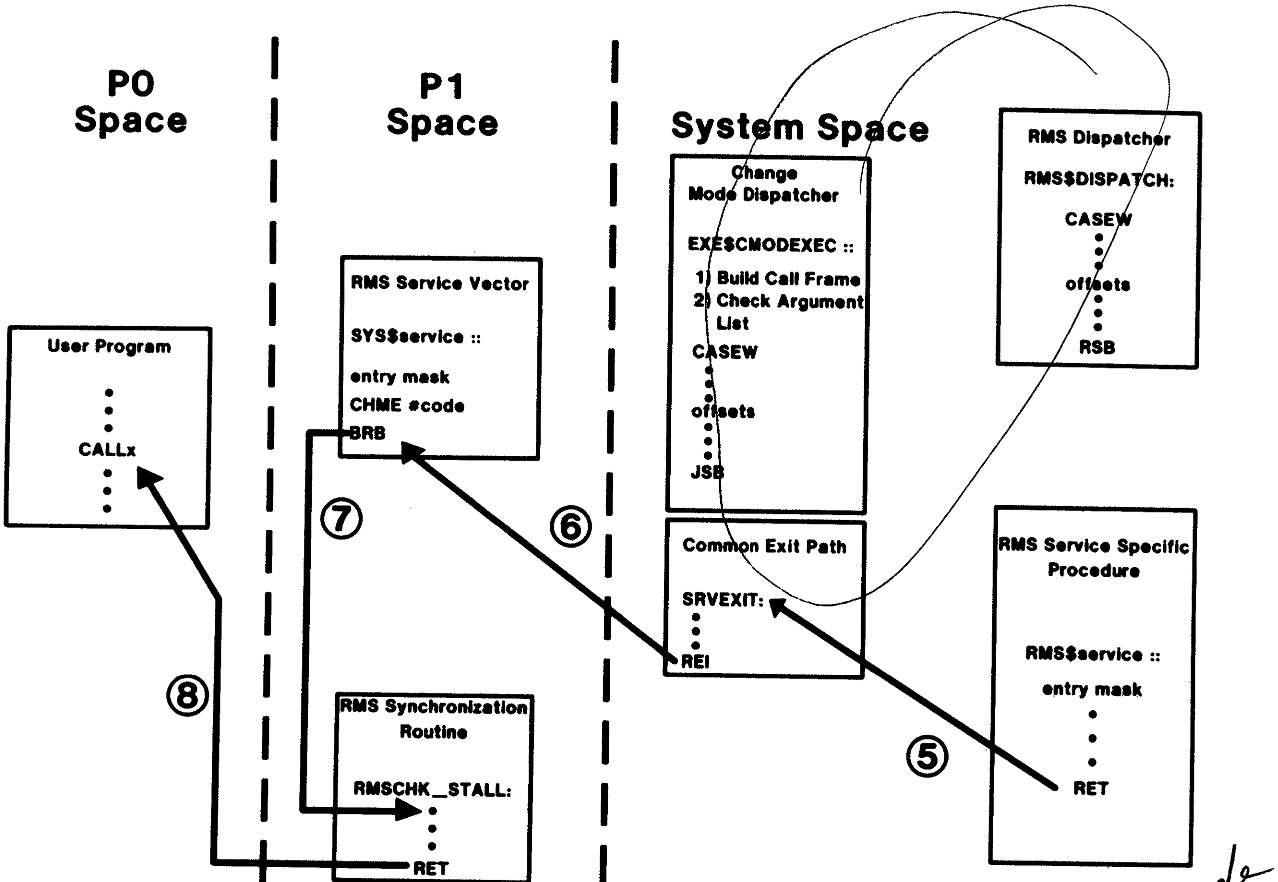


Figure 26 Return from RMS

5. Same path as system service

6. Same as 1

7. Extra step to manage the synchronous nature of most RMS I/O operations

8. RET for original CALL

Path to User-Written Service (1)

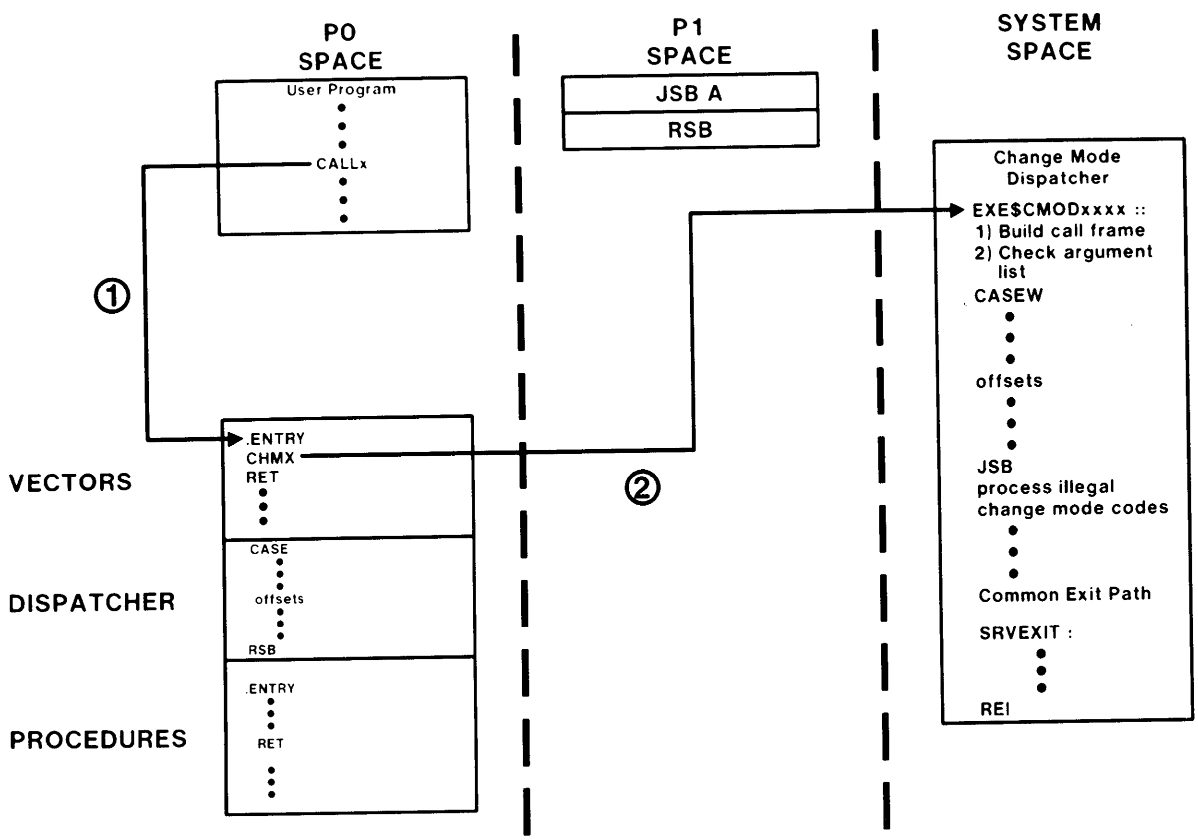


Figure 27 Path to User-Written System Service (Part 1)

1. To find the appropriate user-written service, a user program calls a global symbol defining a service entry vector.

2. A change mode instruction with a negative code causes the change mode dispatcher to look for system service dispatchers that were linked with the image.

Path to User-Written Service (2)

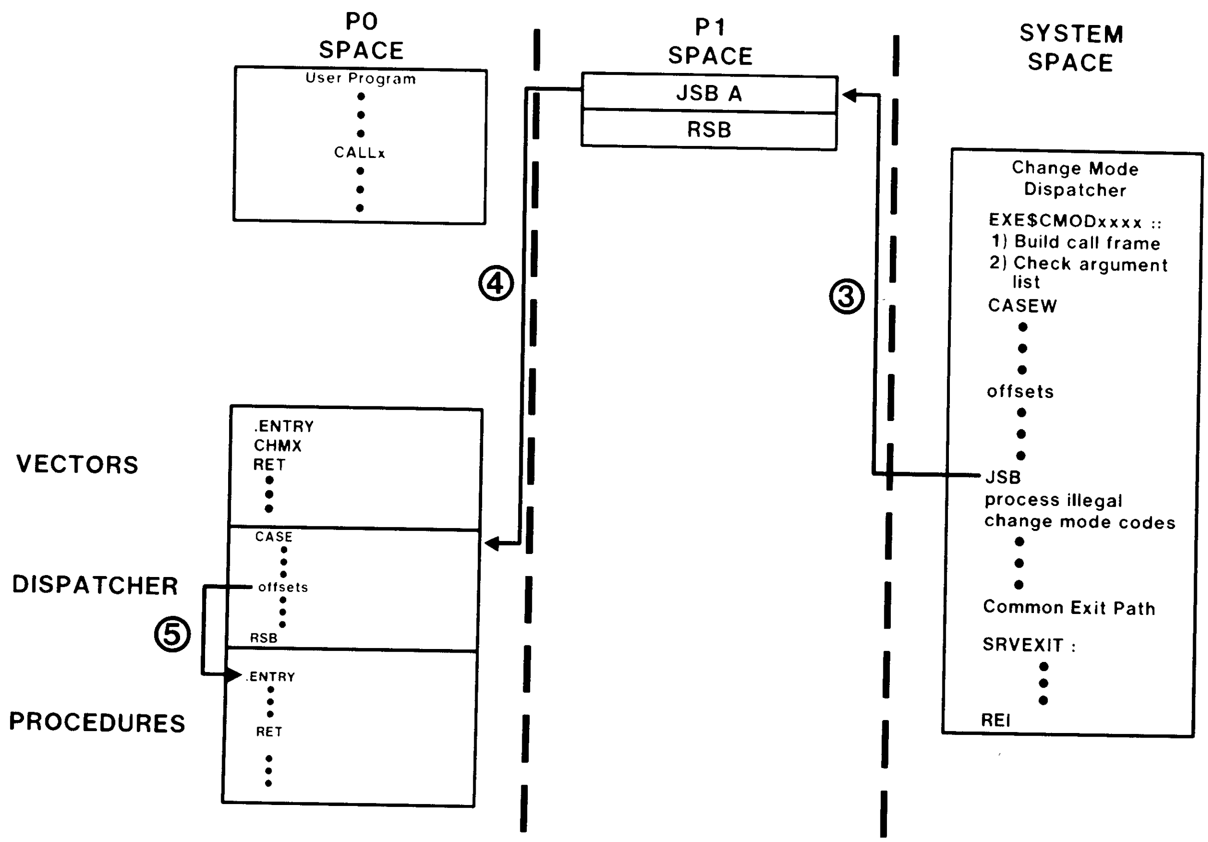


Figure 28 Path to User-Written System Service (Part 2)

3. Code for user-written system service causes JSB at end of case table to be executed.

4. When a request can be serviced, the user-written dispatcher passes control through a CASE instruction to the routine.

5. Same as 4.

Return from User-Written System Service

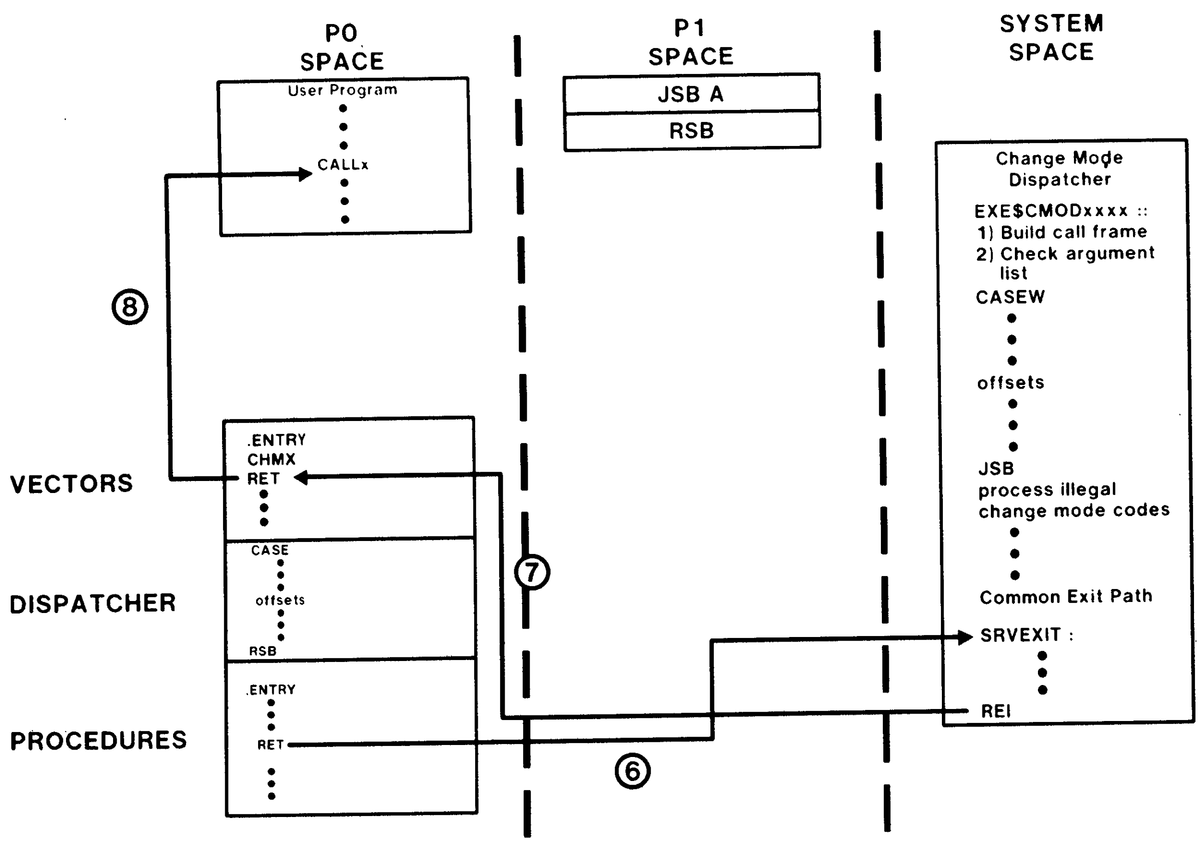


Figure 29 Return from User-Written System Service

6. When the user-written routine exits, it passes control to SRVEXIT, as the supplied system services do.

7. The rest of the return path to the user program is similar to the steps for the supplied system services.

8. Same as 7.

Two Dispatchers

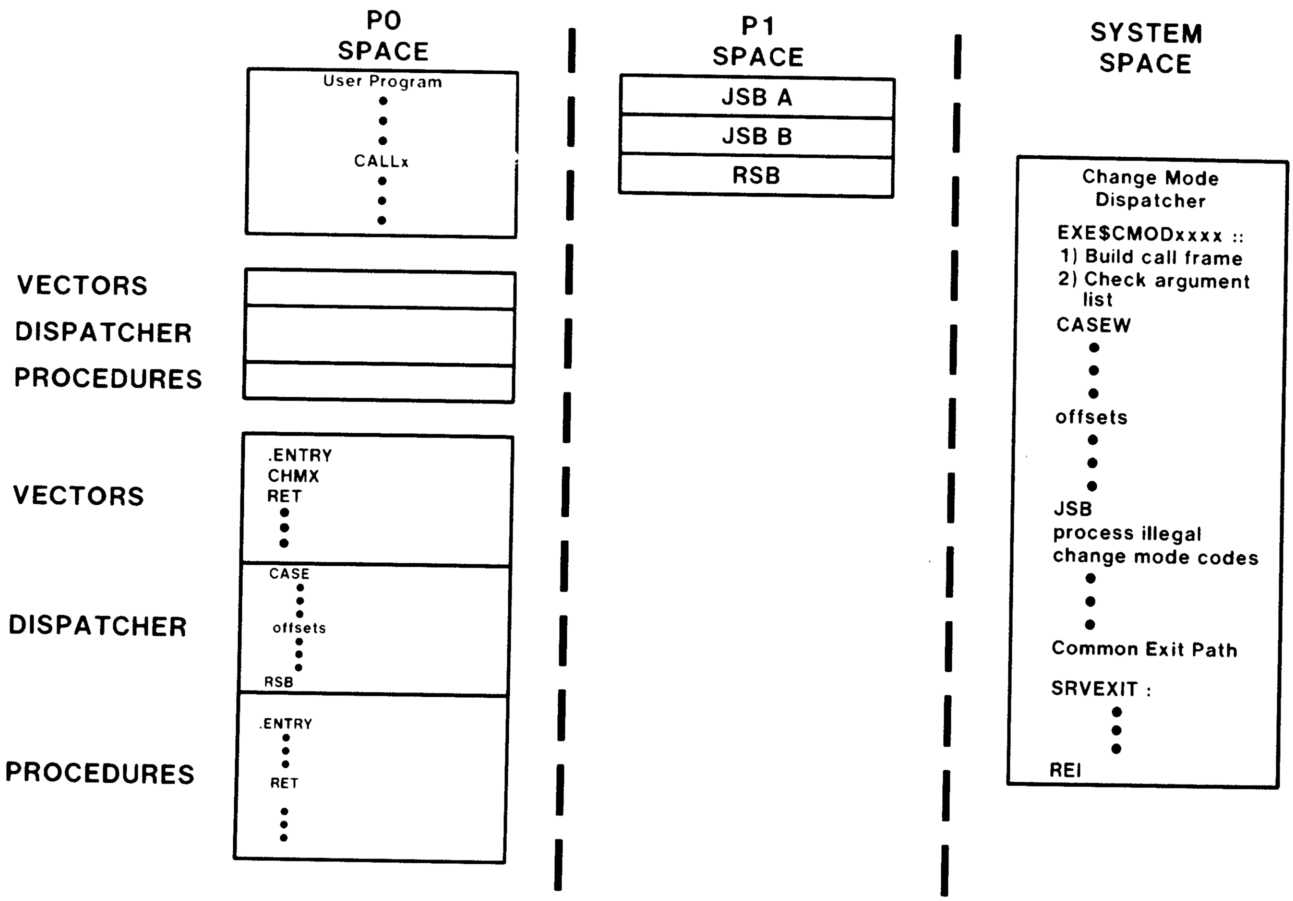


Figure 30 Two Dispatchers

- Multiple dispatchers can be linked to an image.
- Dispatchers are searched in order activated.
- Duplicate CHMx code numbers possible.
  - Only first occurrence recognized.

MISCELLANEOUS MECHANISMS

Dynamic Memory

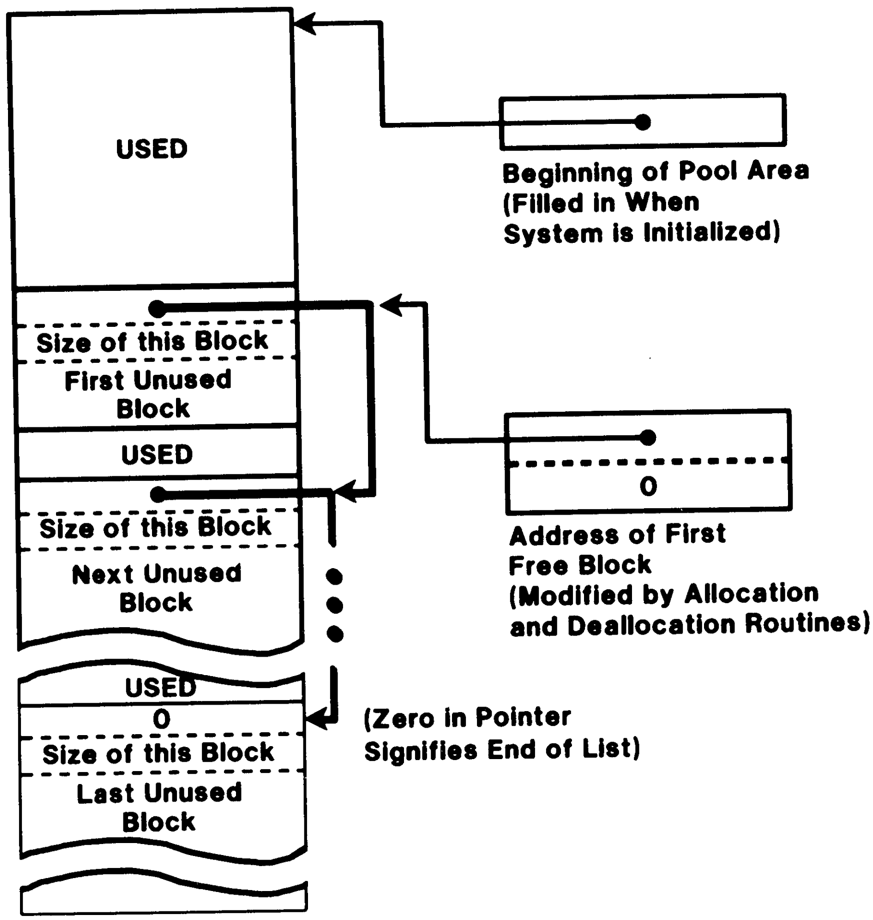


Figure 31 Paged Dynamic Memory

- 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

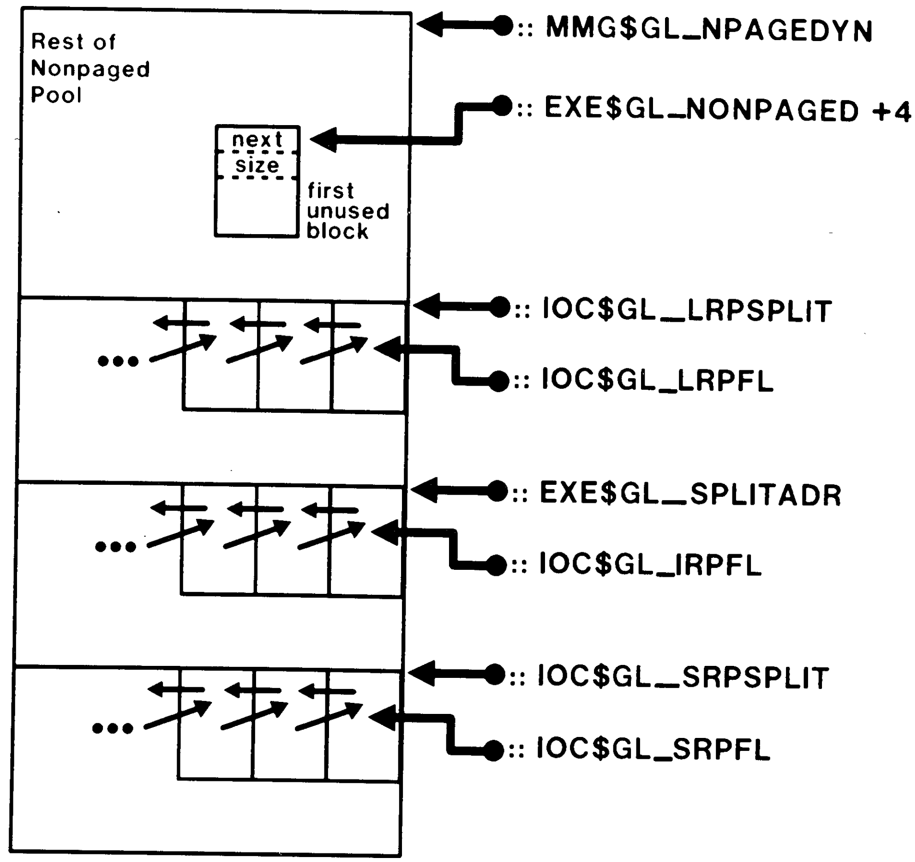


Figure 32 Allocating Nonpaged Pool

Relevant SYSGEN Parameters for Nonpaged Pool

Table 10 SYSGEN Parameters for Nonpaged Pool

Function	Parameter
Number of bytes preallocated for the nonpaged dynamic pool, exclusive of the lookaside lists	NPAGEDYN
Number of bytes to which the nonpaged pool may be extended.	NPAGEVIR
Number of large request packets preallocated for the LRP lookaside list.	LRPCOUNT
Number of LRPs to which the LRP list may be extended.	LRPCOUNTV
Number of bytes to allocate per LRP, exclusive of header. Number of bytes actually allocated per packet is LRPSIZE + 64.	LRPSIZE
Size of minimum allocation request for LRP (bytes)	LRPMIN
Number of I/O request packets preallocated for the IRP lookaside list.	IRPCOUNT
Number of IRPs to which the IRP list may be extended.	IRPCOUNTV
Number of small request packets preallocated for the SRP lookaside list.	SRPCOUNT
Number of SRPs to which the SRP list may be extended.	SRPCOUNTV
Number of bytes to allocate per SRP.	SRPSIZE

Notes on Table 10

- System page table entries are reserved and physical memory preallocated for NPAGEDYN, LRPCOUNT, IRPCOUNT, and SRPCOUNT.
- System page table entries are reserved but no physical memory preallocated for NPAGEVIR, LRPCOUNTV, IRPCOUNTV, and SRPCOUNTV. Physical memory is allocated on demand from the free page list if there is enough excess memory.
- Size of IRPs is 208 bytes.
- LRPMIN is a special parameter.

SUMMARY OF SYSTEM MECHANISMS

Table 11 Function and Implementation of System Mechanisms

Function	Implementation	Name
Keeping Track of CPU, Process State		
Store processor state	Register	Processor status longword (PSL)
Store, restore process state	Instruction	SVPCTX, LDPCTX
Handling and Uses of Interrupts		
Arbitrate interrupt requests	Hardware-maintained priority	Interrupt priority level (IPL)
Service interrupts and exceptions	Table of service routine addresses	System control block (SCB)
Synchronize execution of system routines	Interrupt service routines	Timer, SCHED, IOPOST..
Request an interrupt	MACRO	SOFTINT
Synchronize system's access to system data structures	MACRO-raise IPL to IPL$_SYNCH	SETIPL
Continue execution of code at lower-priority	Queue request, SOFTINT, REI	FORK
How User Executes Protected Code		
Protect memory from read/write	Hardware-maintained access modes	Kernel, Executive, Supervisor, User
Change access mode	Instruction	CHMx, REI
Enter system service, RMS, user-written system service	Call --> instruction	CALL_x --> CHMx

Table 11 Function and Implementation of System Mechanisms (Cont)

Function	Implementation	Name
Process Synchronization		
Synchronize certain system-level activities of processes	Adjusting IPL (SETIPL macro)	IPL
Allow process to request action at a specific time	Queue of requests and hardware and software timer interrupts	Timer queue
Synchronize access to data structures by processes	Semaphore	MUTEX
Allow process to execute procedure on completion of event	REI IPL2 interrupt service routine	Asynchronous system trap (AST)
Allow processes to synchronize access to various resources	$ENQ(W) and $DEQ system services	VMS lock manager

SYSGEN Parameters Related to System Mechanisms

Table 12 SYSGEN Parameters Related to System Mechanisms

Function	Parameter
Size of the interrupt stack (in pages)	INTSTKPAGES
Initial size of nonpaged pool (no lookaside lists)	NPAGEDYN
Maximum size of nonpaged pool	NPAGEVIR
Initial number of LRPs	LRPCOUNT
Maximum number of LRPs	LRPCOUNTV
Bytes in LRP (exclusive of header)	LRPSIZE
Size of minimum allocation request for LRP (bytes)	LRPMIN (*)
Initial number of IRPs	IRPCOUNT
Maximum number of IRPs	IRPCOUNTV
Initial number of SRPs	SRPCOUNT
Maximum number of SRPs	SRPCOUNTV
Number of bytes to allocate per SRP	SRPSIZE (*)
Initial size of Lock ID Table	LOCKIDTBL
Maximum size of Lock ID Table	LOCKIDTBL_MAX
Max. number of entries in Resource Hash Table	RESHASHTBL
Deadlock detection timeout period	DEADLOCK_WAIT
Number of retries for multiprocessor lock	LOCKRETRY (*)

(*) = special SYSGEN parameter

APPENDIX A
COMMONLY USED SYSTEM MACROS

IPL Control Macros

```
.MACRO   SETIPL IPL
         .IF NB IPL
         MTPR   IPL,S^#PR$_IPL
         .IFF
         MTPR   #31,S^#PR$_IPL
         .ENDC
.ENDM    SETIPL

.MACRO   DSBINT IPL,DST
         .IF B   DST
         MFPR    S^#PR$_IPL,-(SP)
         .IFF
         MFPR    S^#PR$_IPL,DST
         .ENDC
         .IF B   IPL
         MTPR    #31,S^#PR$_IPL
         .IFF
         MTPR    IPL,S^#PR$_IPL
         .ENDC
.ENDM    DSBINT

.MACRO   ENBINT  SRC
         .IF B   SRC
         MTPR    (SP)+,S^#PR$_IPL
         .IFF
         MTPR    SRC,S^#PR$_IPL
         .ENDC
.ENDM    ENBINT

.MACRO   SOFTINT IPL
         MTPR    IPL,S^#PR$_SIRR
.ENDM    SOFTINT
```

Example 1 IPL Control Macros

Argument Probing Macros

```
.MACRO    IFRD SIZ,ADR,DEST,MODE=#0
          PROBER     MODE,SIZ,ADR
          BNEQ       DEST
.ENDM     IFRD

.MACRO    IFNORD   SIZ,ADR,DEST,MODE=#0
          PROBER   MODE,SIZ,ADR
          BEQL     DEST
.ENDM     IFNORD

.MACRO    IFWRT    SIZ,ADR,DEST,MODE=#0
          PROBEW   MODE,SIZ,ADR
          BNEQ     DEST
.ENDM     IFWRT

.MACRO    IFNOWRT SIZ,ADR,DEST,MODE=#0
          PROBEW  MODE,SIZ,ADR
          BEQL    DEST
.ENDM     IFNOWRT
```

Example 2 Argument Probing Macros

Privilege Checking Macros

```
.MACRO   IFPRIV PRIV,DEST,PCBREG=R4
         .IF DIF <PRIV>,<R1>
         .IF DIF <PRIV>,<R2>
         BBS      #PRV$V_'PRIV,@PCB$L_PHD(PCBREG),DEST
         .IFF
         BBS      PRIV,@PCB$L_PHD(PCBREG),DEST
         .ENDC
         .IFF
         BBS      PRIV,@PCB$L_PHD(PCBREG),DEST
         .ENDC
.ENDM    IFPRIV

.MACRO   IFNPRIV PRIV,DEST,PCBREG=R4
         .IF DIF <PRIV>,<R1>
         .IF DIF <PRIV>,<R2>
         BBC      #PRV$V_'PRIV,@PCB$L_PHD(PCBREG),DEST
         .IFF
         BBC      PRIV,@PCB$L_PHD(PCBREG),DEST
         .ENDC
         .IFF
         BBC      PRIV,@PCB$L_PHD(PCBREG),DEST
         .ENDC
.ENDM    IFNPRIV
```

Example 3 Privilege Checking Macros

APPENDIX B

PRIVILEGE MASK LOCATIONS

Table 13 Privilege Mask Locations

Symbol Name	Use
CTL$GQ_PROCPRIV	Process permanent mask Altered by SET PROCESS/PRIV= command Used to reset current masks
PCB$Q_PRIV	Current mask, permanently resident Altered by known image activation Altered by $SETPRV system service Reset by image rundown
PHD$Q_PRIVMSK (PHD base address)	Current mask, swappable Altered by known image activation Altered by $SETPRV system service Reset by image rundown Used by IFPRIV, IFNPRIV macros
PHD$Q_IMAGPRIV	Mask of installed known image ORed with CTL$GQ_PROCPRIV to produce current masks
PHD$Q_AUTHPRIV	Mask defined in authorization file Not changed during life of process

APPENDIX C

THE REI INSTRUCTION

The REI instruction results in a reserved operand fault if any one of the following operations is attempted:

1. Decreasing the access mode value (to a more privileged access mode). (This is a comparison of the current mode fields of both the present PSL and the saved PSL on the stack.)

2. Switching to the interrupt stack from one of the four perprocess stacks.

3. Leaving the processor on the interrupt stack in other than kernel access mode.

4. Leaving the processor on the interrupt stack at IPL Ø.

5. Leaving the processor at elevated IPL (IPL > Ø) and not in kernel access mode.

6. Restoring a PSL in which the previous mode field is more privileged than the current mode field (previous mode < current mode).

7. Raising IPL.

8. Setting any of the following bits - PSL<29:28> or PSL<21> or PSL<15:8>.

When the processor attempts to enter compatibility mode, the following checks are made:

1. The first-part-done bit must be clear.

2. The interrupt stack bit must be clear.

3. All three arithmetic trap enables (DV, IV, and FU) must be clear.

4. The current mode field of the saved PSL must be user access mode.

If all the preceding checks are performed without error, the REI microcode continues by:

1. Saving the old stack pointer (SP register) in the appropriate processor register (KSP, ESP, SSP, or USP).

2. Setting the trace pending bit in the new PSL if the trace pending bit in the old PSL is set.

3. Moving the contents of the two temporaries (note 1 above) into the PC and PSL processor registers.

If the target stack is a perprocess stack:

1. Getting the new stack pointer from the corresponding processor register (KSP, ESP, SSP, or USP)

2. Checking for potential deliverability of pending ASTs.

Debugging Tools

$ define LIB$Debug Sys$library:Delta.exe
$ Run/Debug CHMK.Exe

SDA> Validate Queue <x>
(counts queue entries)

INTRODUCTION

Since VMS runs in executive and kernel modes and at elevated interrupt priority levels, any error is considered serious, and can cause a system crash.

VMS offers several tools to aid in debugging system level code. These tools are:

- SDA - a symbolic dump analyzer
- DELTA - a debugger for code running in operating modes from user to kernel.
- XDELTA - a debugger for kernel mode code running at elevated IPLs.

OBJECTIVES

1. To use various system-supplied debugging tools and utilities (for example, SDA, DELTA, XDELTA) to examine crash dumps and to observe a running system.
2. To use the system map file as an aid in reading source code, and identifying the source of system crashes.

RESOURCES

1. VAX/VMS System Dump Analyzer Reference Manual
2. VAX/VMS Internals and Data Structures, chapter on Error Handling
3. VAX/VMS PATCH Utility Reference Manual
4. VAX Hardware Handbook
5. Guide to Writing a Device Driver for VAX/VMS

TOPICS

I. VAX/VMS Debugging Tools

II. The System Dump Analyzer (SDA)

   A. Uses

   B. Requirements

   C. Commands

III. The System Map File

IV. Crash Dumps and Bugchecks

   A. How bugchecks are generated

   B. Sample stacks after bugchecks

   C. Sample crash dump analysis

V. The DELTA and XDELTA Debuggers

VAX/VMS DEBUGGING TOOLS

Table 1 Environment vs. Debugging Tools

Problem/Environment	Method of Analysis
Program IPL=0, User mode Examine perprocess memory	VAX/VMS Symbolic Debugger (Linked with image or included at run time)
Program IPL = 0, User to kernel mode Examine process and system memory	DELTA debugger (Linked with an image or included at run time) Nonsymbolic
Examine active system	System Dump Analyzer (SDA) Activated from DCL
Examine a Crash file	System Dump Analyzer (SDA) Activated from DCL
Program IPL > 0	XDELTA DEBUGGER (Linked with VMS, run from console terminal only) Nonsymbolic

- VAX/VMS provides several debugging tools
- Method of analysis depends on
  - Program environment
  - Nature of desired analysis

force crash: (find E PC)
777 D/0 F FFFFFFFF
777 D PSL 1F000000

THE SYSTEM DUMP ANALYZER (SDA)

- The System Dump Analyzer (SDA) is used to examine:
  - The system dump file (SYS$SYSTEM:SYSDUMP.DMP)
  - A copy of the dump file containing previous crash information
  - The active system
- Through the SDA, information can be:
  - Displayed on a video terminal
  - Printed on a hard-copy terminal
  - Sent to a file or line printer
- Requirements for running SDA
  - VIRTUALPGCNT must be size of SYSDUMP.DMP plus 3000 (pages)
  - PGFLQUOTA must be size of SYSDUMP.DMP plus 2000 (pages)
  - To examine the active system, the CMKRNL privilege is needed
  - To examine a dump file, read access to the file is needed

Table 2 Examining Crash Dump or Current System

To Examine	Command	Restrictions
Current System	$ ANALYZE/SYSTEM	CMKRNL priv needed
System Dump File or Other Dump File	$ ANALYZE/CRASH_DUMP	Read access to file needed

- SDA Functions
  - Examine locations by address or symbol
  - Displays process/system data
  - Formats and displays data structures
  - Assigns values to symbols as requested
- Command Format

```
SDA>  command   [parameter]  [/qualifier]
```

SDA Functions and Commands

Table 3 SDA Functions and Commands

Function	Command
Information	
Provides help using SDA	HELP
Displays specific data/information	SHOW
Formats and displays data structures	FORMAT
Displays contents of location(s)	EXAMINE
Manipulation	
Preserves second copy of dump file	COPY
Creates and defines symbols	DEFINE
Performs computations	EVALUATE
Sets/resets defaults	SET
Defines other VMS symbols	READ
Repeats last command	REPEAT or <Keypad 0>

Table 4 SDA Commands Used to Display Information

Function	Command	Comments
The last crash	SHOW CRASH	Dump file only
I/O data structure	SHOW DEVICE	Device_name parameter optional; /ADDRESS=n
Contents of dump file header	SHOW HEADER	
Resource locks	SHOW LOCK	/ALL
System page table	SHOW PAGE_TABLE	/GLOBAL, /SYSTEM /ALL (D)
PFN database	SHOW PFN_DATA	/FREE, /MODIFIED /SYSTEM, /BAD /ALL (D)
Dynamic pool	SHOW POOL	/IRP, /NONPAGED /PAGED, /SUMMARY, /ALL (D)
Process-specific information	SHOW PROCESS	/PCB (D), /ALL, /CHANNEL, /INDEX=n, /LOCKS, /P0, /P1, /PAGE_TABLES, /PHD, /PROCESS_SECTION_TABLE, /REGISTERS, /RMS, /SYSTEM, /WORKING_SET
Lock manager resource database	SHOW RESOURCE	/ALL, /LOCKID=nn
RMS display options	SHOW RMS	
Stacks	SHOW STACK	/INTERRUPT, /KERNEL /EXECUTIVE, /SUPER /USER
Summary of all processes	SHOW SUMMARY	/IMAGE
Symbol table	SHOW SYMBOL	Symbol-name parameter optional; /ALL

Table 5 Symbols and Operators

Function	Symbol or Operator	Example
Contents of location	@	Examine @8ØØØØ45A
Add 8ØØØØØØØ (SØ base) to address	G	G45A
Add 7FFEØØØØ (Pl stacks) to address	H	H7A4
Current location	.	Format .
Hexadecimal number radix	^H	^H1Ø
Octal number radix	^O	^O2Ø
Decimal number radix	^D	^D16
Register symbols	RØ-Rll, AP, FP, KSP, ESP, SSP, USP, PØBR, POLR, PlBR, PlLR, PC, PSL	

Table 6 Common Command Usage

Function	Command	Comment
Examine location(s)	EX . EX G14:G74	One location Several locations
Examine address at location	EX @USP	Examine address found contained in given location
Format data	Format addr Format @addr	Format at given location Format at contents addr
Define symbol	Define BEGIN = G58Ø	

Examining an Active System

```
$ ANALYZE/SYSTEM

VAX/VMS System Analyzer

SDA> EVALUATE G+(50*4)-(4/2)+^07
Hex = 80000145   Decimal = -2147483323
SDA>
SDA> EXAMINE G25C0
SCH$GL_NULLPCB+118:  0000E274   "tb.."
SDA>
SDA> EXAMINE
SCH$GL_NULLPCB+11C:  00000000   "...."
SDA>
SDA> EXAMINE      ! used keypad 0 to repeat last command
SCH$GL_NULLPCB+120:  FFFFFFFF   "...."
SDA>
SDA> EXAMINE      ! used keypad 0 to repeat last command
SCH$GL_NULLPCB+124:  FFFFFFFF   "...."
SDA>
SDA> EX IOC$GL_DEVLIST
IOC$GL_DEVLIST:  80000F5C   "\..."
SDA>
SDA> EX R0
R0:  00000020   " ..."
SDA>
SDA> EX/PSL PSL
        CMP TP FPD IS CURMOD PRVMOD IPL DV FU IV T N Z V C
         0   0  0   0  USER   USER   00  0  0  0 0 0 1 0 0
SDA>
SDA> EVALUATE/CONDITION C
%SYSTEM-F-ACCVIO, access violation, reason mask=!XB,
virtual address=!XL, PC=!XL, PSL=!XL
SDA>
SDA> EX G100:G140
00040019 8FBC00FC 00040018 8FBC003C  <.<.....|.<.....     80000100
0004001B 8FBC07FC 0004001A 8FBC00FC  |.<.....|.<.....     80000110
0004001D 8FBC0FFC 0004001C 8FBC00FC  |.<.....|.<.....     80000120
0004001F 8FBC003C 0004001E 8FBC01FC  |.<.....<.<.....     80000130
00040021 8FBC01FC 00040020 8FBC0010  ..<.....|.<.!...     80000140
```

Example 1 Examining an Active System (Sheet 1 of 5)

```
SDA> SHOW PROCESS
Process index: 0044   Name: HUNT   Extended PID: 00000144
----------------------------------------------------------
Process status:  02040001    RES,PHDRES

PCB address                 80126730    JIB address                 802001D0
PHD address                 80507800    Swapfile disk address       01001C81
Master internal PID         00020044    Subprocess count                   0
Internal PID                00020044    Creator internal PID        00000000
Extended PID                00000144    Creator extended PID        00000000
State                            CUR    Termination mailbox             0000
Current priority                   7    AST's enabled                   KESU
Base priority                      4    AST's active                    NONE
UIC                        [011,140]    AST's remaining                    7
Mutex count                        0    Buffered I/O count/limit         6/6
Waiting EF cluster                 0    Direct I/O count/limit           6/6
Starting wait time          1B001B1B    BUFIO byte count/limit     7840/7840
Event flag wait mask        DFFFFFFF    # open files allowed left         36
Local EF cluster 0          E0000023    Timer entries allowed left        10
Local EF cluster 1          D8000000    Active page table count            0
Global cluster 2 pointer 00000000    Process WS page count            250
Global cluster 3 pointer 00000000    Global WS page count              50

SDA>
SDA> SHOW LOCK
Lock database
-------------

Lock id:  00010001   PID:     00000000   Flags:    NOQUEUE SYNCSTS SYSTEM
Par. id:  00000000   Granted at    EX              CVTSYS
Sublocks:        0
LKB:      80257540
Resource:       5F535953 24535953    SYS$SYS_  Status:  NOQUOTA
 Length   16    00000000 00004449    ID......
 Exec. mode     00000000 00000000    ........
 System         00000000 00000000    ........
Local copy

Lock id:  00020002   PID:     00000000   Flags:    CONVERT NOQUEUE SYNCSTS
Par. id:  00000000   Granted at    CR              NOQUOTA CVTSYS
Sublocks:        0
LKB:      80257A80   BLKAST
Resource:       41566224 42313146    F11B$bVA  Status:  NOQUOTA
 Length   18    20334C52 534D5658    XVMSRL3
 Kernel mode    00000000 00002020      ......
 System         00000000 00000000    ........
Local copy

                          .
                          .
                          .
```

Example 1 Examining an Active System (Sheet 2 of 5)

```
SDA> READ OSI$LABS:GLOBALS
SDA>
SDA> FORMAT @EXE$GL_TQFL
80108524    TQE$L_TQFL                 8011B040
80108528    TQE$L_TQBL                 80002B58
8010852C    TQE$W_SIZE                     0030
8010852E    TQE$B_TYPE                   0F
8010852F    TQE$B_RQTYPE               05
80108530    TQE$L_FPC                  80107F36
            TQE$L_PID
80108534    TQE$L_AST                  802002B4
            TQE$L_FR3
80108538    TQE$L_ASTPRM               802002A0
            TQE$L_FR4
8010853C    TQE$Q_TIME                 90DED860
80108540                               008D1C99
80108544    TQE$Q_DELTA                00989680
80108548                               00000000
8010854C    TQE$B_RMOD                       00
8010854D    TQE$B_EFN                      00
8010854E                               0000
80108550    TQE$L_RQPID                00000000
            TQE$C_LENGTH
SDA>
SDA> FORMAT @.
8011B040    TQE$L_TQFL                 80106918
8011B044    TQE$L_TQBL                 80108524
8011B048    TQE$W_SIZE                     0000
8011B04A    TQE$B_TYPE                   0F
8011B04B    TQE$B_RQTYPE               05
8011B04C    TQE$L_FPC                  80118E11
            TQE$L_PID
8011B050    TQE$L_AST                  00000000
            TQE$L_FR3
8011B054    TQE$L_ASTPRM               8011AE10
            TQE$L_FR4
8011B058    TQE$Q_TIME                 924D0E60
8011B05C                               008D1C99
8011B060    TQE$Q_DELTA                00989680
8011B064                               00000000
8011B068    TQE$B_RMOD                       00
8011B069    TQE$B_EFN                      00
8011B06A                               0000
8011B06C    TQE$L_RQPID                00000000
            TQE$C_LENGTH
```

Example 1 Examining an Active System (Sheet 3 of 5)

```
SDA> SHOW POOL/IRP
                       Dump of blocks allocated from IRP lookaside list


CONF     801ED600    208
                       28106C00 0763009C 00000000 00000000 ..........c..l.(
                       80029200 00380000 00002020 00000000 ....    ....8.....
                       8002C800 80029800 80029600 80029400 .............H..
                       8002D000 8002CE00 8002CC00 8002CA00 .J...L...N...P..
                       8002F400 8002F200 8002F000 8002E000 .`...p...r...t..
                       00000000 80030A00 8002F800 8002F600 .v...x..........
                       00000000 00000028 00000010 0000006C l.......(.......
                       00000020 00000000 00000000 00000000 ............ ...
                       00000000 00000000 00000000 00000020  ...............
                       00000000 00000000 00000038 00000000 ....8...........
                       00000000 00000000 00000000 00000000 ................
                       00000000 00000000 00000000 00000000 ................
                       00000000 00000000 00000000 00000000 ................
FCB      801ED940    208
                       00000000 000700C0 801FC5B0 801EF5B0 Ou..OE..@.......
                       00010001 00010001 80259340 80259340 @.%.@.%.........
                       00000001 00000000 002E08ED 00000000 ....m...........
                       00000001 00000002 0003DAD9 00000000 ....YZ..........
                       000008ED 00000000 00000000 00000000 ............m...
                       00000000 00010004 00000000 00000000 ................
                       FFFFFFFF FFFFFFFF 00000000 05490058 X.I.............
                       00000000 00000000 00000000 0000EA00 .j..............
                       00000000 00000000 00000000 00000000 ................
                       00000000 00000000 00000000 00000000 ................
                       00000000 00000000 00000000 00000000 ................
                       00000000 00000000 00000000 00000000 ................
                       00000000 00000000 00000000 00000000 ................
IRP      801EDA10    208
                       00030029 410A00C4 80002A58 801F59A0 .Y..X*..D..A)...
                       8010AAE0 00000000 7FFC6928 800394E8 h...(i|.....`*..
                       80121BF0 0003FFB0 7FFC6934 1B1DC000 .@..4i|.O...p...
                       11010001 00000000 00000000 0100014F O...............
                       00000000 802575A0 00900820 00001200 ....  ....u%.....
                       4946204E 801159F4 244C4C41 0003520F .R..ALL$tY..N FI
                       0003002A 20020000 8011F470 0000454C LE..pt.....  *...
                       00000000 00000000 7FFB00C0 7FFB0CF8 x.{.@.{.........
                       2061206F 74206465 FA081603 03030000 .......zed to a
                       00000004 00000200 08020D54 4E495250 PRINT...........
                       00208001 00000000 02000000 00000003 .............. .
                       00000000 00000201 0000FFFF 64280100 ..(d............
                       00000000 00000000 00000000 00000000 ................
FCB      801EDC80    208
                       00000000 000700C0 80202B40 801EDEF0 p^..@+ .@.......
                       00000000 00010001 80261D40 80261D40 @.&.@.&.........
                       00000001 00000000 000600D7 00000001 ....W...........
                       00000040 00000040 0003D2C3 00000000 ....CR..@...@...
                       000000D7 00000000 00000000 00000000 ............W...
                       00000000 00010004 00000000 00000000 ................
                       FFFFFFFF FFFFFFFF 00000000 05490058 X.I.............
                       00000000 00000000 00000000 0000A000 ................
                       00000000 00000000 00000000 00000000 ................
                       00000000 00000000 00000000 00000000 ................
                       00000000 00000000 00000000 00000000 ................
                       00000000 00000000 00000000 00000000 ................
                       00000000 00000000 00000000 00000000 ................
JIB      801EDD50    208
                       4F56414C 002F0080 801EDD50 801EDD50 PJ..PJ..../.LAVO
                       20202020 20202020 20202020 20204549 IE
```

Example 1 Examining an Active System (Sheet 4 of 5)

```
SDA>
SDA> SHOW STACK/USER
Process stacks
--------------
Current operating stack (USER):

                  7FF31A44  00000000
                  7FF31A48  000011F8      SGN$C_MAXPGFL+1F8
                  7FF31A4C  00000001
                  7FF31A50  00000000
                  7FF31A54  00001D17      CTL$C_CLIDATASZ+773
                  7FF31A58  0001F5C2
                  7FF31A5C  00001D23      CTL$C_CLIDATASZ+77F
                  7FF31A60  0001ED74

           SP =>  7FF31A64  00001D1B      CTL$C_CLIDATASZ+777
                  7FF31A68  00000000
                  7FF31A6C  00000000
                  7FF31A70  2FFC0000
                  7FF31A74  7FF31AEB
                  7FF31A78  7FF31ACC
                  7FF31A7C  000070E3      SGN$C_NPAGEDYN+8E3
                  7FF31A80  000013AF      SGN$C_MAXPGFL+3AF
                  7FF31A84  00001D17      CTL$C_CLIDATASZ+773
                  7FF31A88  00000000
                  7FF31A8C  00000000
                  7FF31A90  0000000C
                  7FF31A94  00001D17      CTL$C_CLIDATASZ+773
                  7FF31A98  0001EE56
                  7FF31A9C  00001D23      CTL$C_CLIDATASZ+77F
                  7FF31AA0  7FFEDDD4
                  7FF31AA4  00001D2B      CTL$C_CLIDATASZ+787
                  7FF31AA8  00000003
                  7FF31AAC  00001D17      CTL$C_CLIDATASZ+773
                  7FF31AB0  0001EDD4
                  7FF31AB4  0001E926
                  7FF31AB8  0000000F
                  7FF31ABC  00000600      BUG$_NOHDJMT
                  7FF31AC0  00000000
                  7FF31AC4  00000000
                  7FF31AC8  00000000
                  7FF31ACC  0001FE56
                      .         .
                      .         .
                      .         .
```

Example 1 Examining an Active System (Sheet 5 of 5)

THE SYSTEM MAP FILE

Overview

- MAP of linked executive
- Available on every VMS system
  SYS$SYSTEM:SYS.MAP
- Useful in debugging crash dumps and when reading source code

Sections of SYS.MAP

1. Object module synopsis
   - Listed in order processed by linker
   - Includes creation data and source language
2. Image section synopsis
   - Lists base virtual address
3. Program section synopsis
   - Lists PSECTs by base virtual address
   - Includes PSECT size and attributes
4. Symbol cross-reference
   - Lists global symbols alphabetically
   - Includes symbol value, module(s) that define and reference it
5. Symbols by value
   - Lists global symbols by hexadecimal value
   - Multiple symbols have same value
6. Image synopsis
   - Miscellaneous information about the output image
7. Link run statistics
   - Miscellaneous information about the link run that produced the image.

SYS.MAP and Crash Dumps

1. Information in crash dumps given by value

   - Virtual address of code (PC)
   - Contents of data structures

     - Virtual address references
     - Symbolic references (for example, State of process)

2. SYS.MAP can be used to translate numbers to meaningful information.

   - Program section synopsis (virtual address to source code module)

   - Symbols by value (value to symbol name)

SYS.MAP and Source Code

1. Layout of linked executive in SØ space

   - Program section synopsis

2. Interrelationship of modules ("who references whom")

   - Symbol cross-reference

3. Module entry points and global data locations

CRASH DUMPS

- Generated when the system decides that it cannot continue normal flow of work
- System attempts to copy all the information in physical memory to a special file on a disk

Causes of Crash Dumps

- Fatal error or inconsistency (fatal bugcheck) recognized and declared by a component of the operating system
- Bugcheck is declared by referencing a central routine
- Some reasons for declaring a fatal bugcheck:
  - Exception at elevated IPL
  - Exception while on interrupt stack
  - Machine check in kernel mode
  - BUG_CHECK macro issued
  - HALT instruction restart
  - Interrupt stack invalid restart
  - Kernel or executive mode exception without exit handler

BUGCHECKS

The Two Types of Bugchecks

- Fatal - system must be taken down; no recovery possible
- Continue - nonfatal; the system may attempt recovery

How Crash Dumps Are Generated

- Written by the fatal bugcheck code
- For a dump to be written
  - Bugcheck must be fatal
  - If nonfatal bugcheck, all bugchecks must be declared fatal (done by setting BUGCHECKFATAL = 1)
  - DUMPBUG (a SYSGEN parameter) must be set (= 1). DUMPBUG is set by default.
  - SYS$SYSTEM:SYSDUMP.DMP must be the correct size file size = physical memory plus 4 (in pages)
  - Console must be allowed to finish printing the bugcheck output

How Bugchecks Are Generated

BUGCHECKS are generated using the BUG_CHECK macro.

```
BUG_CHECK     QUEUEMPTY,FATAL

generates

.WORD         ^XFEFF
.WORD         BUG$QUEUEMPTY!4
```

Bugchecks are generated by system components (EXEC, RMS, ACP, and so on) after detecting an internal (software) error.

Table 7 Sample BUGCHECKS

Name	Module	Type	Description
BADRSEIPL	RSE	Fatal	Bad IPL at entrance to RSE
FATALEXCPI	EXCEPTION	Fatal	Fatal executive or kernel mode exception
NOTPCB	MUTEX	Fatal	Structure is not a PCB
UNABLCREVA	EXCEPTION	Cont.	Unable to create virtual address space

NOTE

When looking at the crash dump, PC minus 4 is that address at which the BUG_CHECK macro is referenced.

```
**** FATAL BUG CHECK, VERSION = V4.0 SSRVEXCEPT, Unexpected system service exception

    CURRENT PROCESS = SYSTEM

    REGISTER DUMP

        R0 = 00000000
        R1 = 8000FDD2
        R2 = 00000040
        R3 = 7FFA50AF
        R4 = 80117F60
        R5 = 7FFE64B4
        R6 = 7FFED78A
        R7 = 7FFED78A
        R8 = 00000050
        R9 = 7FFED25A
        R10= 7FFEDDD4
        R11= 7FFE33DC
        AP = 7FFE7D8C
        FP = 7FFE7D74
        SP = 7FFE7D6C
        PC = 8000FDD8
        PSL= 00000000

    KERNEL/INTERRUPT STACK

        7FFE7D74  00000000
        7FFE7D78  00000000
        7FFE7D7C  00000000
        7FFE7D80  7FFE7DC8
        7FFE7D84  80000014
        7FFE7D88  80017F16
        7FFE7D8C  00000002
        7FFE7D90  7FFE7DB0
        7FFE7D94  7FFE7D98
        7FFE7D98  00000004  <-- MECHANISM ARRAY
        7FFE7D9C  7FF75360
        7FFE7DA0  FFFFFFFD
        7FFE7DA4  00000014
        7FFE7DA8  00000030
        7FFE7DAC  000008F8
        7FFE7DB0  00000005  <-- SIGNAL ARRAY
        7FFE7DB4  0000000C  <-- SS$_ACCVIO
        7FFE7DB8  00000000  <-- REASON MASK
        7FFE7DBC  00000014  <-- FAULTING V.A.
        7FFE7DC0  00000222  <-- PC
        7FFE7DC4  00C00000  <-- PSL
        7FFE7DC8  00000000
        7FFE7DCC  01040000
        7FFE7DD0  7FF75378
        7FFE7DD4  7FFE7DE4
        7FFE7DD8  8000940C
        7FFE7DDC  00000004
        7FFE7DE0  7FFED052
        7FFE7DE4  00000000
        7FFE7DE8  00000000
        7FFE7DEC  7FF75378
        7FFE7DF0  7FF75360
        7FFE7DF4  8000FDCE
        7FFE7DF8  7FFEDE96
        7FFE7DFC  03C00000
```

Example 2 Sample Console Output After Bugcheck

SAMPLE STACKS AFTER BUGCHECKS

Access Violation

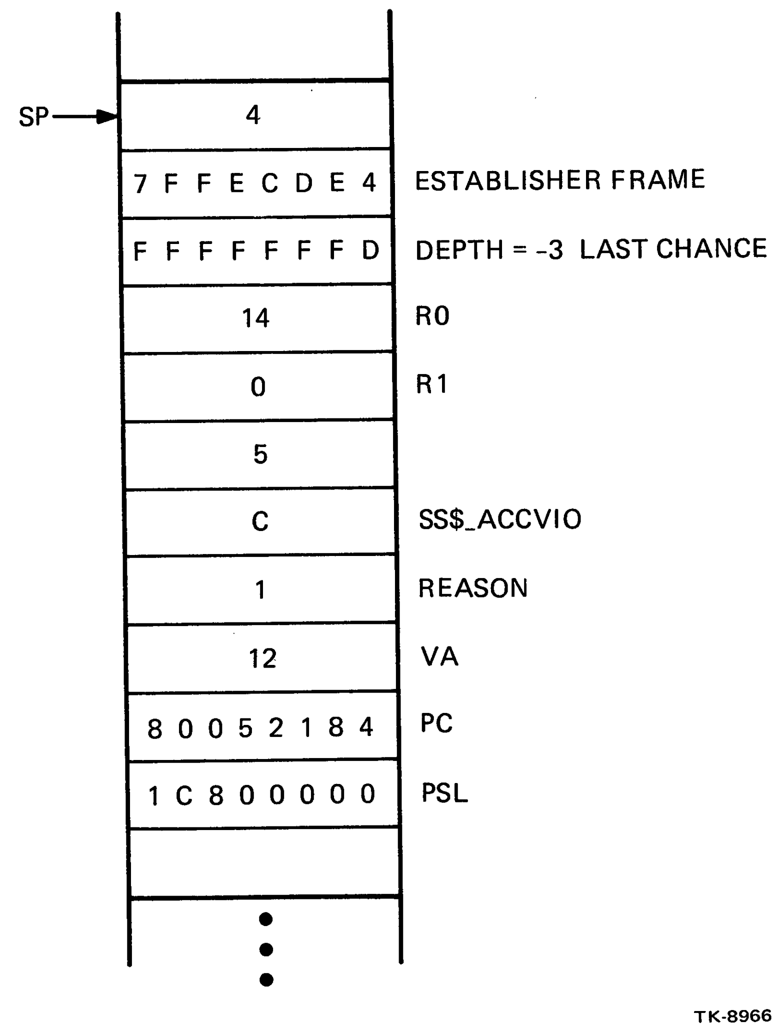


Figure 1 Stack After Access Violation Bugcheck

Probable Causes:

- Blown register
- Incorrect data structure field
- Improper synchronization

Page Fault Above IPL 2

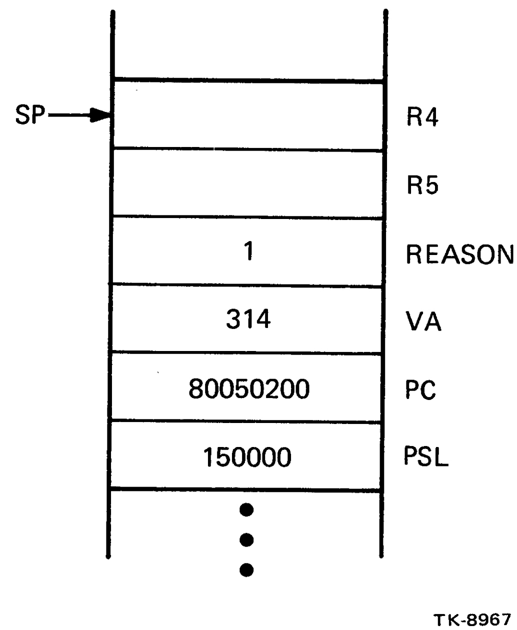


Figure 2 Stack After Page Fault Above IPL-2

Probable Causes:

- Blown register in fork interrupt routine
- Improper start I/O routine design

Reserved Operand Fault

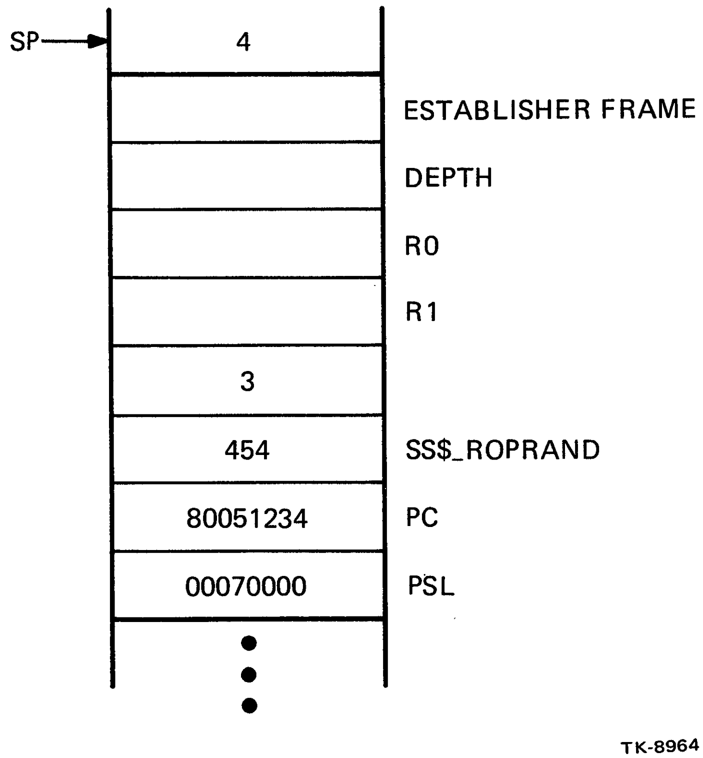


Figure 3 Stack After Reserved Operand Fault

Probable Causes:

- REI failure
  - IPL problems (allocate memory at wrong IPL)
  - Blown stack
- RET failure

Machine Check in Kernel Mode (CPU Timeout)

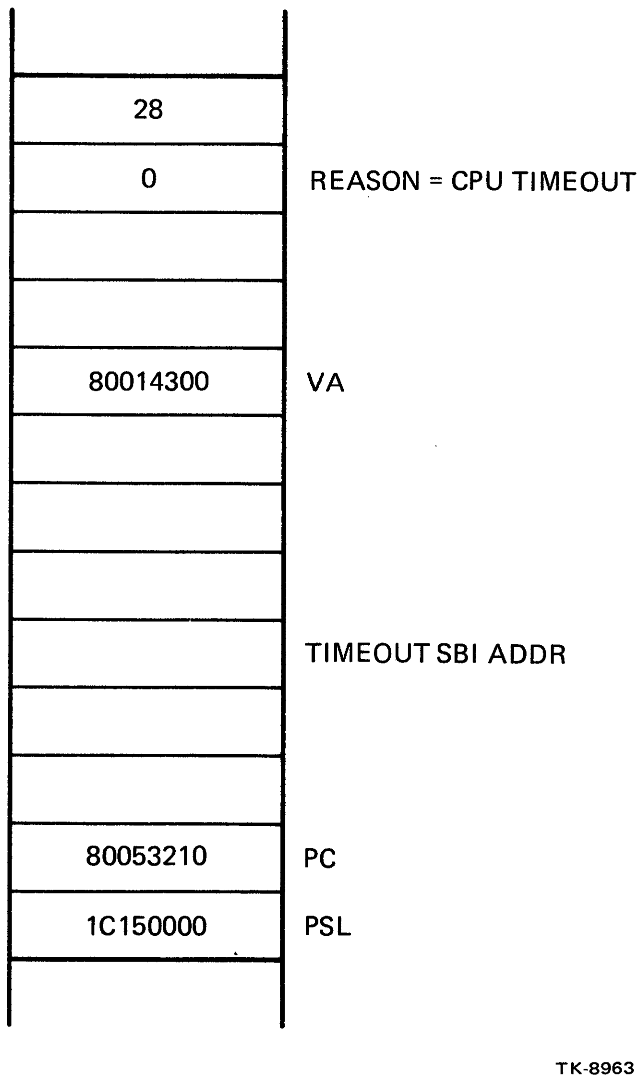


Figure 4 Stack After Machine Check in Kernel Mode

Reasons:

- Accessing nonexistent UBA or SBI address
- Corrupted page tables
- Processor device or bus failure

Sample Crash Dump Analysis

```
$ ANALYZE/CRASH SYS$SYSTEM:SYSDUMP.DMP
VAX/VMS System dump analyzer

Dump taken on 3-OCT-1984  12:26:20.27
SSRVEXCEPT, Unexpected system service exception

SDA> sho crash
System crash information
------------------------
Time of system crash:  3-OCT-1984 12:26:20.27


Version of system: VAX/VMS VERSION V4.0


Reason for BUGCHECK exception: SSRVEXCEPT, Unexpected system service exception


Process currently executing: SYSTEM


Current image file: DRA0:[SYS0.][SYSMGR]CRASHAST.EXE;3


Current IPL: 0  (decimal)


General registers:

        R0  = 00000000   R1  = 8000FDD2   R2  = 00000004   R3  = 7FFA50AF
        R4  = 80106EB0   R5  = 00000000   R6  = 7FFED78A   R7  = 7FFED78A
        R8  = 7FFED052   R9  = 7FFED25A   R10 = 7FFEDDD4   R11 = 7FFE33DC
        AP  = 7FFE7D88   FP  = 7FFE7D70   SP  = 7FFE7D70   PC  = 8000FDD8
        PSL = 00000000


Processor registers:

        P0BR   = 8024B600    PCBB   = 006CC478    ACCS   = 00000000
        P0LR   = 00000003    SCBB   = 007EFE00    SBIFS  = 00040000
        P1BR   = 7FA5E600    ASTLVL = 00000004    SBISC  = 00000000
        P1LR   = 001FFB96    SISR   = 00180000    SBIMT  = 00200400
        SBR    = 007F2000    ICCS   = 800000C1    SBIER  = 00008000
        SLR    = 00003800    ICR    = FFFFEC69    SBITA  = 20000000
                             TODR   = 9E670C51    SBIS   = 00000000

        ISP    = 8022EA00
        KSP    = 7FFE7D70
        ESP    = 7FFE9E00
        SSP    = 7FFED04E
        USP    = 7FF75360
```

Example 3 Sample Crash Dump Analysis (Sheet 1 of 4)

```
SDA> sho stack
Current operating stack
-----------------------
Current operating stack (KERNEL):

                  7FFE7D50  7FFED25A
                  7FFE7D54  7FFEDDD4
                  7FFE7D58  7FFE33DC    CTL$AG_CLIDATA+180
                  7FFE7D5C  7FFE7D88    CTL$GL_KSTKBAS+588
                  7FFE7D60  7FFE7D70    CTL$GL_KSTKBAS+570
                  7FFE7D64  7FFE7D68    CTL$GL_KSTKBAS+568
                  7FFE7D68  8000FDD8    EXE$EXCPTN+006
                  7FFE7D6C  00000000

           SP =>  7FFE7D70  00000000
                  7FFE7D74  00000000
                  7FFE7D78  00000000
                  7FFE7D7C  7FFE7DC8    CTL$GL_KSTKBAS+5C8
                  7FFE7D80  80000014    SYS$CALL_HANDL+004
                  7FFE7D84  80017F16    EXE$CONTSIGNAL+07C
                  7FFE7D88  00000002
                  7FFE7D8C  7FFE7DAC    CTL$GL_KSTKBAS+5AC
                  7FFE7D90  7FFE7D94    CTL$GL_KSTKBAS+594
                  7FFE7D94  00000004
                  7FFE7D98  7FF75360
                  7FFE7D9C  FFFFFFFD
                  7FFE7DA0  00C00009
                  7FFE7DA4  00000002
                  7FFE7DA8  000008F8    SS$_ENDOFFILE+088
                  7FFE7DAC  00000005
                  7FFE7DB0  0000000C
                  7FFE7DB4  00000000
                  7FFE7DB8  0000000C
                  7FFE7DBC  80009F68    MPH$QAST
                  7FFE7DC0  00C00004
                  7FFE7DC4  00000220    BUG$_MODRELNBAK
                  7FFE7DC8  00000000
                  7FFE7DCC  00240000
                  7FFE7DD0  7FF75378
                  7FFE7DD4  7FFE7DE4    CTL$GL_KSTKBAS+5E4
                  7FFE7DD8  8000940C    EXE$CMKRNL+00D
                  7FFE7DDC  00000004
                  7FFE7DE0  7FFE64B4    MMG$IMGHDRBUF+0B4
                  7FFE7DE4  00000000
                  7FFE7DE8  00000000
                  7FFE7DEC  7FF75378
                  7FFE7DF0  7FF75360
                  7FFE7DF4  8000FDCE    EXE$CMODEXEC+176
                  7FFE7DF8  7FFEDE96    SYS$CMKRNL+006
                  7FFE7DFC  03C00000
```

Example 3 Sample Crash Dump Analysis (Sheet 2 of 4)

```
_$255$DUA28:[SYS.OBJ]SYS.EXE;1                                     16-SEP-1984 04:00      VAX-11 Linker V04-00          Page    7

Psect Name      Module Name      Base     End              Length          Align              Attributes
----------      -----------      ----     ---              ------          -----              ----------

                                                     .
                                                     .
                                                     .

$OSWPSCHED                     800087CE 80008A76 000002A9 (       681.) BYTE 0 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,  WRT,NOVEC
                OSWPSCHED      800087CE 80008A76 000002A9 (       681.) BYTE 0

$ZBUGFATAL                     80008A78 80008A78 00000000 (         0.) WORD 1 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,  WRT,NOVEC
                BUGCHECK       80008A78 80008A78 00000000 (         0.) WORD 1

. BLANK .                      80008A78 80009D8D 00001316 (      4886.) BYTE 0 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,  WRT,NOVEC
                EXSUBROUT      80008A78 80008B10 00000099 (       153.) BYTE 0
                FORKCNTRL      80008B11 80008B1E 0000000E (        14.) BYTE 0
                NULLPROC       80008B1F 80008B20 00000002 (         2.) BYTE 0
                SYSACPFDT      80008B21 8000925B 0000073B (      1851.) BYTE 0
                SYSASCEFC      8000925C 8000927A 0000001F (        31.) BYTE 0
                SYSCANCEL      8000927B 800093B5 0000013B (       315.) BYTE 0
                SYSCANEVT      800093B6 800093EE 00000039 (        57.) BYTE 0
                SYSCHGMOD      800093EF 8000941F 00000031 (        49.) BYTE 0
                SYSDERLMB      80009420 8000945A 0000003B (        59.) BYTE 0
                SYSFORCEX      8000945B 8000949F 00000045 (        69.) BYTE 0
                SYSQIOFDT      800094A0 80009741 000002A2 (       674.) BYTE 0
                SYSSCHEVT      80009742 800098A9 00000168 (       360.) BYTE 0
                SYSQIOREQ      800098AA 80009CDB 00000432 (      1074.) BYTE 0
                SYSSETPRI      80009CDC 80009D6F 00000094 (       148.) BYTE 0
                SYSMTACCESS    80009D70 80009D79 0000000A (        10.) BYTE 0
                MTFDT          80009D7A 80009D8D 00000014 (        20.) BYTE 0

A$EXENONPAGED                  80009D90 8000A37C 000005ED (      1517.) LONG 2 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,  WRT,NOVEC
                ASTDEL         80009D90 8000A040 000002B1 (       689.) LONG 2
                FORKCNTRL      8000A044 8000A0C4 00000081 (       129.) LONG 2
                TIMESCHDL      8000A0C8 8000A37C 000002B5 (       693.) LONG 2

AES1                           8000A37D 8000A675 000002F9 (       761.) BYTE 0 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,  WRT,NOVEC
                RSE            8000A37D 8000A675 000002F9 (       761.) BYTE 0

AES2                           8000A676 8000A6A1 0000002C (        44.) BYTE 0 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,  WRT,NOVEC
```

Example 3 Sample Crash Dump Analysis (Sheet 3 of 4)

```
01C6         .SBTTL  SCH$QAST - ENQUEUE AST CONTROL BLOCK FOR P
01C6    ;++
01C6    ; FUNCTIONAL DESCRIPTION:
01C6    ;      SCH$QAST INSERTS THE AST CONTROL BLOCK SUPPLIED IN
01C6    ;      POSITION BY ACCESS MODE IN THE AST QUEUE OF THE PR
01C6    ;      BY THE PID FIELD OF THE AST CONTROL BLOCK.  AN AST
01C6    ;      IS THEN REPORTED FOR THE PROCESS TO REACTIVATE FRO
01C6    ;      IF APPROPRIATE.  THE AST CONTROL BLOCK WILL BE REL
01C6    ;      IF THE PID SPECIFIES A NON-EXISTENT PROCESS.
01C6    ;
01C6    ;      LOADABLE MULTI-PROCESSING CODE WILL REPLACE THIS R
01C6    ;      ENTIRELY NEW CODE, AT MPH$QAST.
01C6    ;
01C6    ; CALLING SEQUENCE:
01C6    ;      BSB/JSB SCH$QAST
01C6    ;
01C6    ; INPUT PARAMETERS:
01C6    ;      R2 - PRIORITY INCREMENT CLASS
01C6    ;      R5 - POINTER TO AST CONTROL BLOCK
01C6    ;
01C6    ; IMPLICIT INPUTS:
01C6    ;      PCB OF PROCESS IDENTIFIED BY PID FIELD
01C6    ;
01C6    ; OUTPUT PARAMETERS:
01C6    ;      R0 - COMPLETION STATUS CODE
01C6    ;      R4 - PCB ADDRESS OF PROCESS FOR WHICH AST WAS QUEU
01C6    ;
01C6    ; SIDE EFFECTS:
01C6    ;      THE PROCESS IDENTIFIED BY THE PID IN THE AST CONTR
01C6    ;      WILL BE MADE EXECUTABLE IF NOT SUSPENDED.
01C6    ;
01C6    ; COMPLETION CODES:
01C6    ;      SS$_NORMAL   -  NORMAL SUCCESSFUL COMPLETION STATUS
01C6    ;      SS$_NONEXPR  -  NON-EXISTENT PROCESS
01C6    ;--
01C6         .ENABL  LSB
01C6    QNONEXPR:
01C6         ; DEALLOCATE THE ACB AS LONG AS THE NODELETE BIT I
01C6         ; THIS REALLY SHOULDN'T HAPPEN, BUT IF IT DOES, WE
01C6         ; TO POSSIBLY LOSE POOL OVER POSSIBLY CORRUPTING I
01C6
01C6         BBS     #ACB$V_NODELETE,ACB$B_RMOD(R5),5$; BR IF N
01CB         MOVL    R5,R0                      ; RELEASE AST CONT
01CE         BSBW    EXE$DEANONPAGED            ; IF NO SUCH PROCE
01D1    5$:  MOVZWL  #SS$_NONEXPR,R0            ; SET ERROR STATUS
01D6         BRB     QEXIT                      ; AND EXIT
01D8
01D8    MPH$QAST::                              ; MULTI-PROCESSING
01D8    SCH$QAST::                              ; ENQUEUE AST FOR
01D8         MOVZWL  ACB$L_PID(R5),R0           ; GET PROCESS INDE
01DC         DSBINT  #IPL$_SYNCH                ; DISABLE SYSTEM E
01E2         MOVL    @W^SCH$GL_PCBVEC[R0],R4    ; LOOK UP PCB ADDR
01E8         CMPL    ACB$L_PID(R5),PCB$L_PID(R4) ; CHECK FOR MA
01ED         BNEQ    QNONEXPR                   ; PID MISMATCHES
01EF         CLRL    R0                         ; ASSUME KERNEL MO
```

Example 3 Sample Crash Dump Analysis (Sheet 4 of 4)

DELTA AND XDELTA

Table 8 Comparison of DELTA with XDELTA

Factors	DELTA	XDELTA
Usage	User images	Operating System Drivers
Terminal used for control	Any TTY	Console only (OPAØ:)
IPL	= Ø	≥Ø
How activated	Linked or included at run time	Included at boot time
Access mode	All modes	Kernel mode only

Both debuggers are:

- Nonsymbolic
- Use name command syntax
- No visible prompt
- Error message is "Eh?"

DELTA Debugger

To use the DELTA debugger, assemble and link a program in the following fashion:

```
1.  $ MACRO prog_nameSYS$LIBRARY:LIB/LIB
2.  $ LINK/DEBUG prog_name, SYS$SYSTEM:SYS.STB/SELECT
3.  $ DEFINE LIB$DEBUG DELTA
4.  $ RUN prog_name
```

Steps:

1. Assembles the program allowing system macros to be defined (SYS$LIBRARY:LIB/LIB).

2. Links the program with a debugger and resolving any system symbols (SYS$SYSTEM:SYS.STB).

3. Define the debugger used to be DELTA.

4. Activate the program mapping in DELTA.

CHMK Program

It is often convenient to observe data structures changing dynamically. One way to gain access to kernel mode data structures is to run the CHMK program. This program allows any privileged process (with CMKRNL privilege) to change mode to kernel, and enter DELTA commands (for example, to look at system data structures).

NOTE

Extreme caution should be exercised that data structures not be modified, since such modification could lead to a system crash.

Perform the following steps to use the CHMK program.

1. Assemble CHMK.
2. Link CHMK.
3. Indicate the DELTA debugger.
4. Run the CHMK program.
5. Enter a breakpoint in the program and tell it to proceed.

The Corresponding Commands are:

1. $ MACRO CHMK SYS$LIBRARY:LIB/LIB
2. $ LINK/DEBUG CHMK, SYS$SYSTEM:SYS.STB/SELECT
3. $ DEFINE LIB$DEBUG DELTA
4. $ RUN CHMK
5. 215;B;P

Note that at step 4, no prompt from DELTA is given.

After you receive the "stopped at breakpoint" message, you are in kernel mode, and may proceed to examine system data structures. To leave the program, type ';P', followed by EXIT. (If you just type EXIT, you will be logged off, since kernel mode exit implies process deletion.)

```
;       This program gets you into kernel mode.
;       Use with DELTA debugger to examine system locations.

GO:     .WORD   0                       ; Null entry mask
        $CMKRNL_S    ROUTIN = 10$       ; Enter kernel mode
        RET                             ; all done
10$:    .WORD   0                       ; Null entry mask
        NOP                             ; Where BPT instruction
        NOP                             ;  is placed (215;B)
        MOVZBL  #SS$_NORMAL, R0         ; Return success status
        RET                             ; All done in kernel mode
        .END    GO
```

Example 4   The CHMK Program

DELTA and XDELTA Functions and Commands

Table 9 DELTA and XDELTA Functions and Commands

Function	Command	Example
Display contents of given address	address/	GA88/00060034
Replace contents of given address	addr/contents new	GA88/00060034 GA88 GA88/00060034 'A' (Replace as ASCII)
Display contents of previous location	<ESC>	80000A88/80000BE4 <ESC> 80000A84/00000000
Display contents of next location	addr/contents <LF> addr/contents	80000004/8FBC0FFC 80000008/50E9002C
Display range of locations	addr,addr/contents	G4,GC/8FBC0FFC 80000008/50E9002C 8000000C/00000400
Display indirect	<TAB> or /	80000A88/80000BE4 <TAB> 80000BE4/80000078 80000A88/80000BE4/80000078
Single step command	S	1 brk at 8000B17D S 8000B17E/9A0FBB05
Set breakpoint	addr,N;B <RET> (N is a number 2-8)	800055F6,2;B
Display breakpoint	;B	;B 1 8000B17D 2 800055F6

Table 9 DELTA and XDELTA Functions and Commands (Cont)

Function	Command	Example
Clear breakpoint	Ø,N;B <RET>	Ø,2;B
Proceed from breakpoint	;P	;P
Set base register	'value',N;X	8ØØØØØØØ,Ø;X
Display base register	Xn <RET> or Xn=	XØ ØØØØØØØ3 XØ=ØØØØØØØ3
Display general register	Rn/ (n is in Hexadecimal)	RØ/ØØØØØØØ3
Show value	expression=	1+2+3+4=ØØØØØØØA (+,-,*,%{divide})
Executing stored command strings	addr;E <ret>	8ØØØØE58;E
Change display mode	[B [W [L ["	Byte width Word width Longword width ASCII display

! ⇒ show as instruction

RF+4 (PSL)

APPENDIX A
BUGCHECK FLOW OF CONTROL

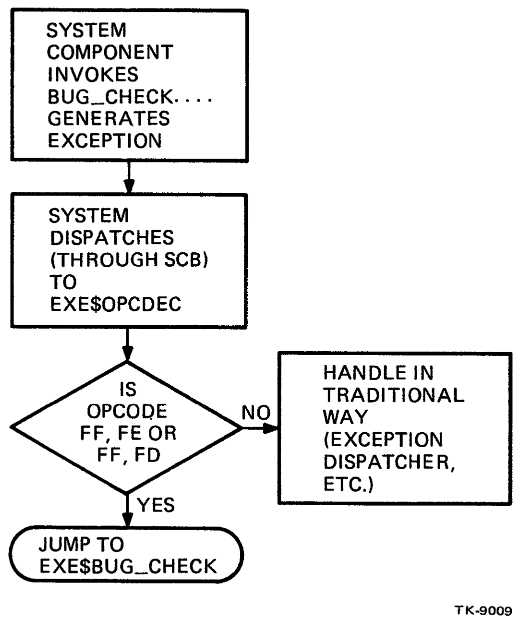


Figure 5 Bugcheck Flow of Control (Sheet 1 of 3)

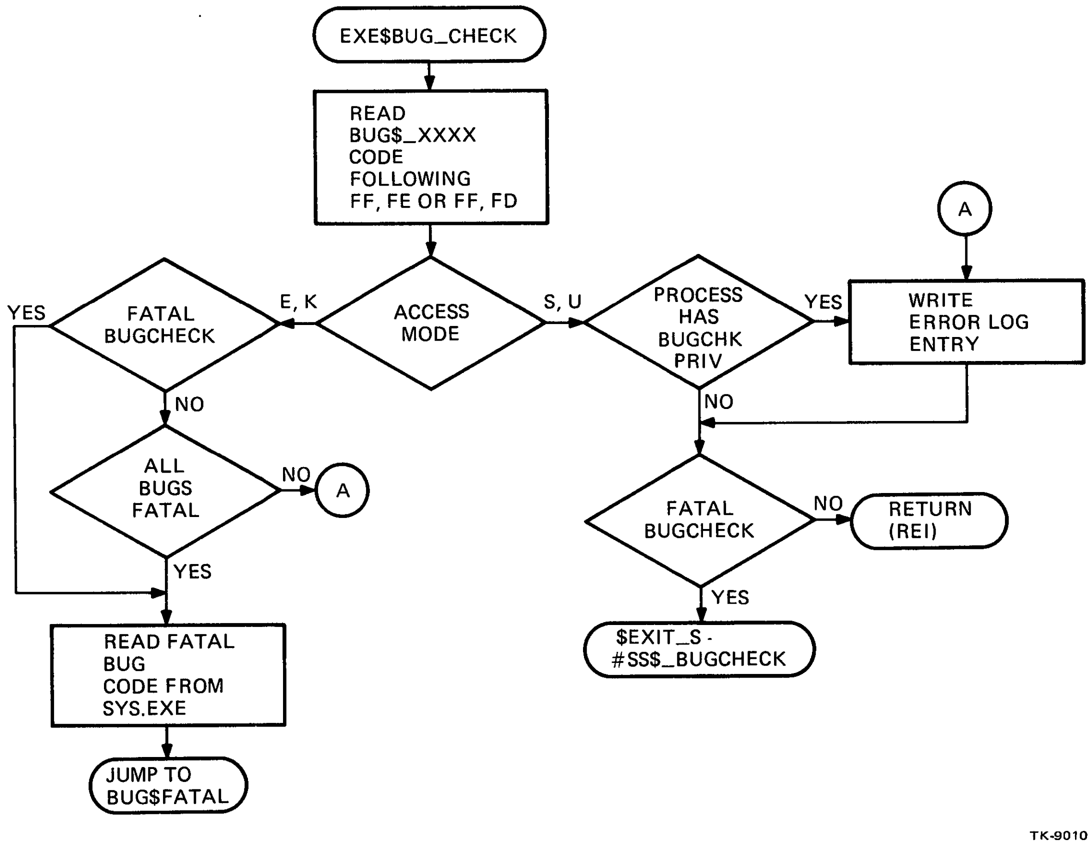


Figure 5 Bugcheck Flow of Control (Sheet 2 of 3)

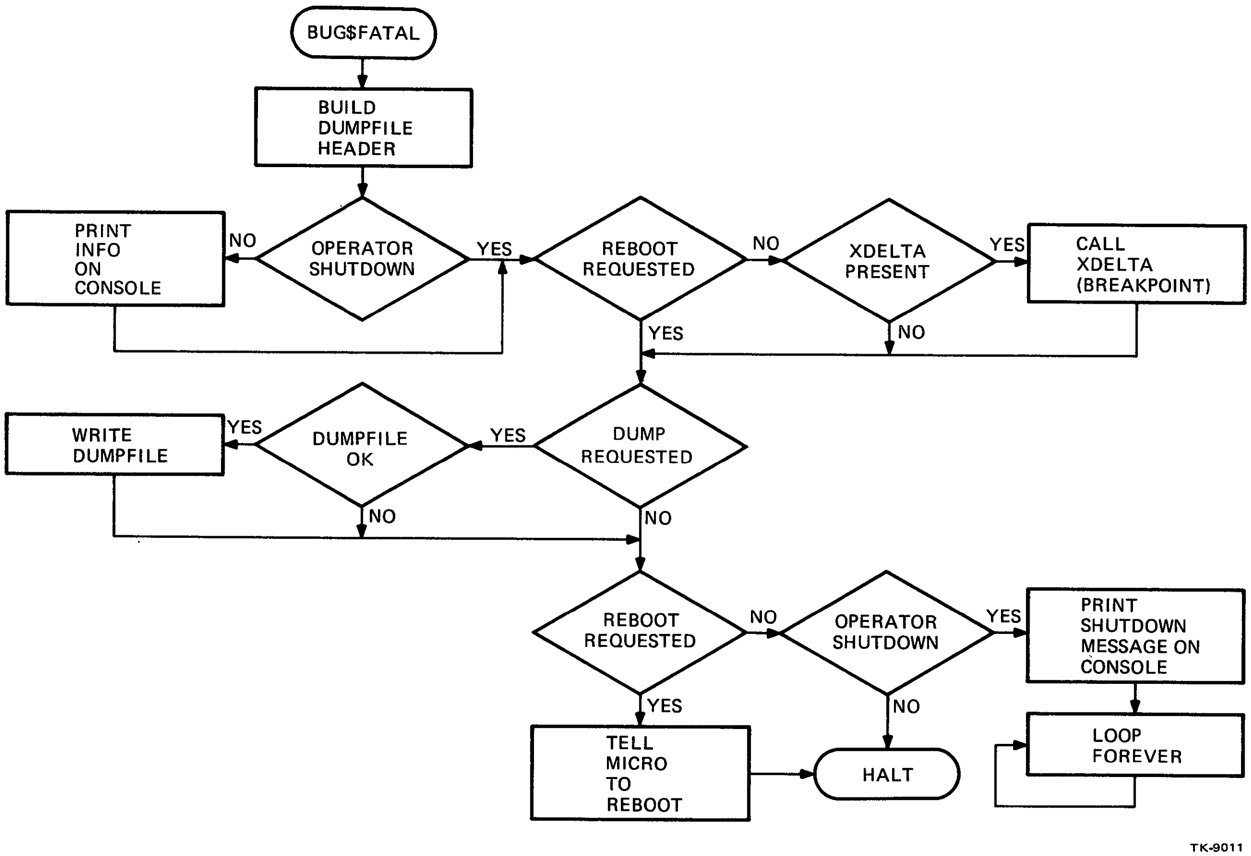


Figure 5 Bugcheck Flow of Control (Sheet 3 of 3)

APPENDIX B
PATCH

The patch utility enables a user to 'edit' an image file. Patch is intended to be used on non-DIGITAL software. Application of patches to DIGITAL software, other than those that are DIGITAL-supplied, invalidate the warranty.

Table 10 PATCH Commands

Function	Command
Display contents of one or more locations	Examine
Store new contents in one or more locations	Deposit
Insert one or more symbolic instructions	Insert
Verify the replace contents of location	Replace
Display various information (e.g., module names)	SHOW parameter
Alter default settings (e.g., module name referenced)	SET parameter

Scheduling

INTRODUCTION

Scheduling is the selection of a process for a particular action or event. The scheduler, a software interrupt service routine at IPL 3, is responsible for selecting which memory-resident, executable process will be the next one to use the CPU. The scheduler code performs the exchange of hardware process contexts between the set of resident, computable processes and the currently executing process.

The swapper, a system process, selects processes for removal from, or placement in, memory. Outswap operations move processes in memory-resident states to corresponding outswapped states. Inswap operations transform executable, nonresident processes into executable, resident ones.

Additional support routines provide the logic to establish and satisfy a range of conditions for which processes may wait. Examples of these conditions include system service requests (such as $HIBER, $RESUME, or $WAITFR) and resource waits (such as mutex wait or depleted system dynamic memory).

OBJECTIVES

1. For each process state, describe the properties of a process in the state, and how a process enters and leaves the state.
2. Given a set of initial conditions and a description of a system event, describe the operation of the scheduler.
3. Assign priorities for a multiprocess application.
4. Discuss the effects of altering SYSGEN parameters related to scheduling.

RESOURCES

Reading

- VAX/VMS Internals and Data Structures, the chapter on Scheduling.

Additional Suggested Reading

- VAX/VMS Internals and Data Structures, the chapters on Software Interrupts, Process Control and Communication, Timer Support, Swapping, and Synchronization Techniques.

Source Modules

Facility Name	Module Name
SYS	SCHED RSE SYSWAIT SDAT SWAPPER (local label SWAPSCHED) OSWPSCHED SYSPCNTRL

TOPICS

I. Process States

   A. What they are (current, computable, wait)

   B. How they are defined

   C. How they are related

II. How Process States are Implemented in Data Structures

   A. Queues

   B. Process data structures

III. The Scheduler (SCHED.MAR)

IV. Boosting Software Priority of Normal Processes

V. Operating System Code that Implements Process State Changes

   A. Context switch (SCHED.MAR)

   B. Result of system event (RSE.MAR)

VI. Steps at Quantum End

   A. Automatic working set adjustment

VII. Software Priority Levels of System Processes

THE PROCESS STATES

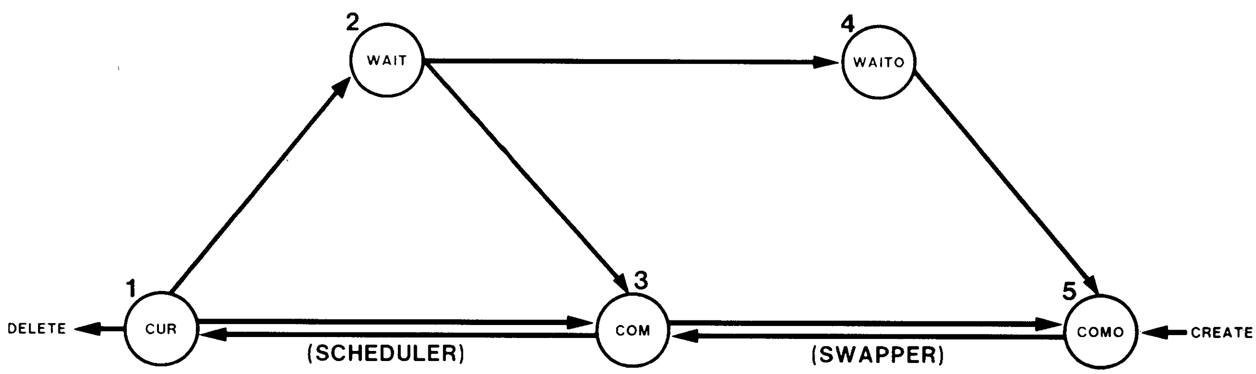


Figure 1 Process States

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

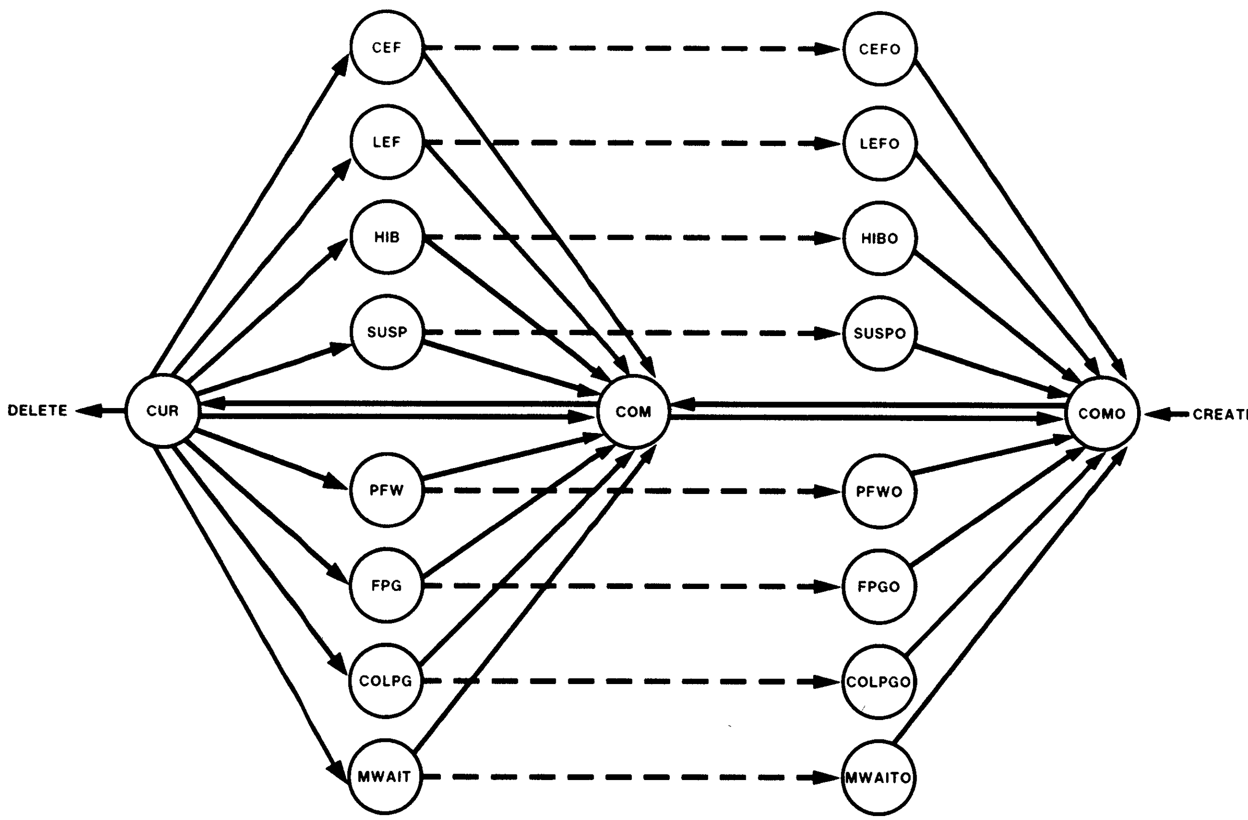


Figure 2 Process Wait States

Ways to Leave the Current State

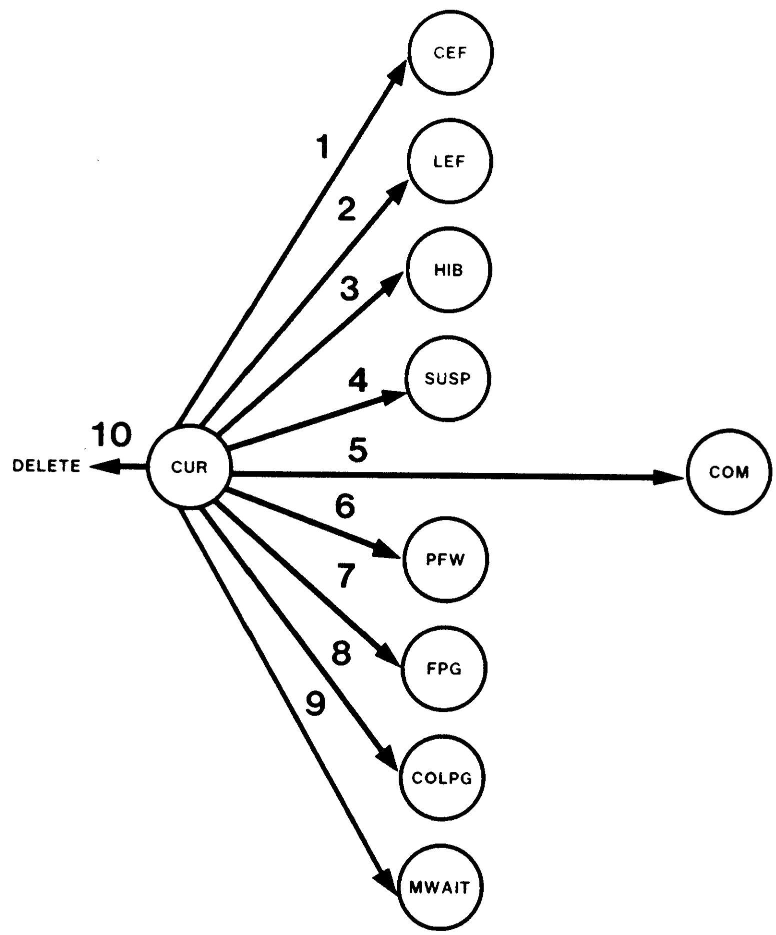


Figure 3 Ways to Leave Current State

1. Wait for common event flag(s) set ($WAITFR)
2. Wait for local event flag(s) set ($WAITFR)
3. Hibernate until wake-up ($HIBER)
4. Suspended until resume ($SUSPND)
5. Removed from execution-quantum end or preempted
6. Page read in progress
7. Wait for free page available
8. Wait for shared page to be read in by another process
9. Wait for miscellaneous resources or mutex
10. Deletion

Ways to Become Computable (Inswapped)

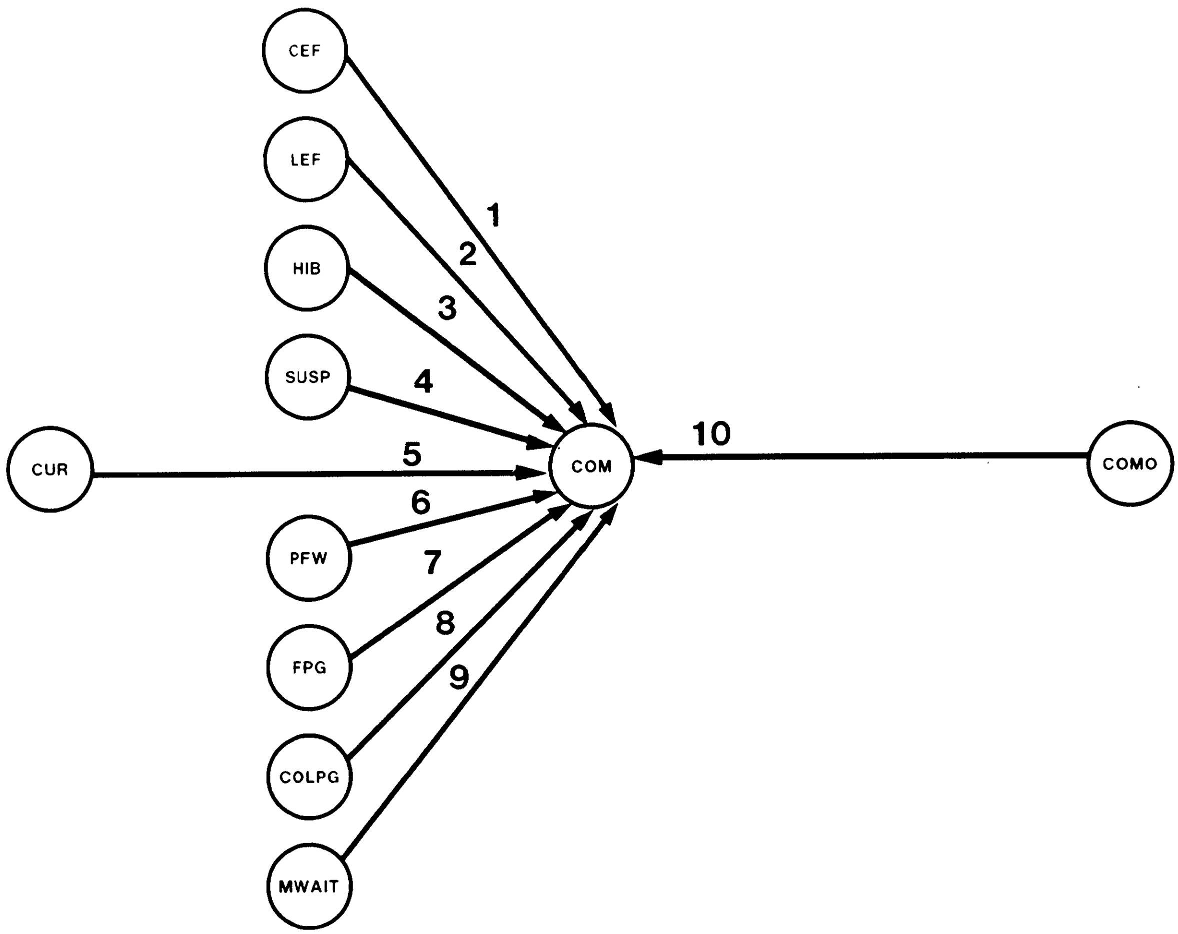


Figure 4 Ways to Become Computable (Inswapped)

1. Common event flag(s) set
2. Local event flag(s) set
3. Wake-up ($WAKE)
4. Resume ($RESUME)
5. Removed from execution-quantum end or preempt
6. Page read complete
7. Free page available
8. Shared page read complete
9. Miscellaneous resources available or mutex available
10. Outswapped computable process is inswapped

Inswapped to Outswapped Transitions

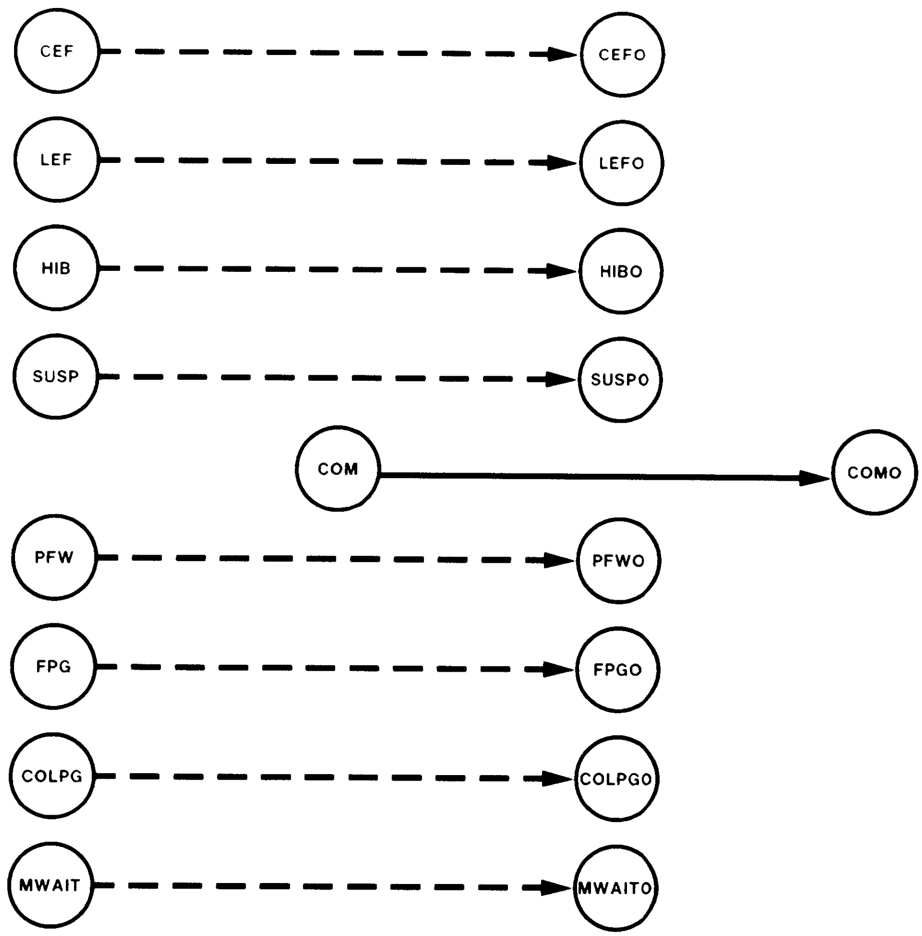


Figure 5 Inswapped to Outswapped Transitions

Ways to Become Computable (Outswapped)

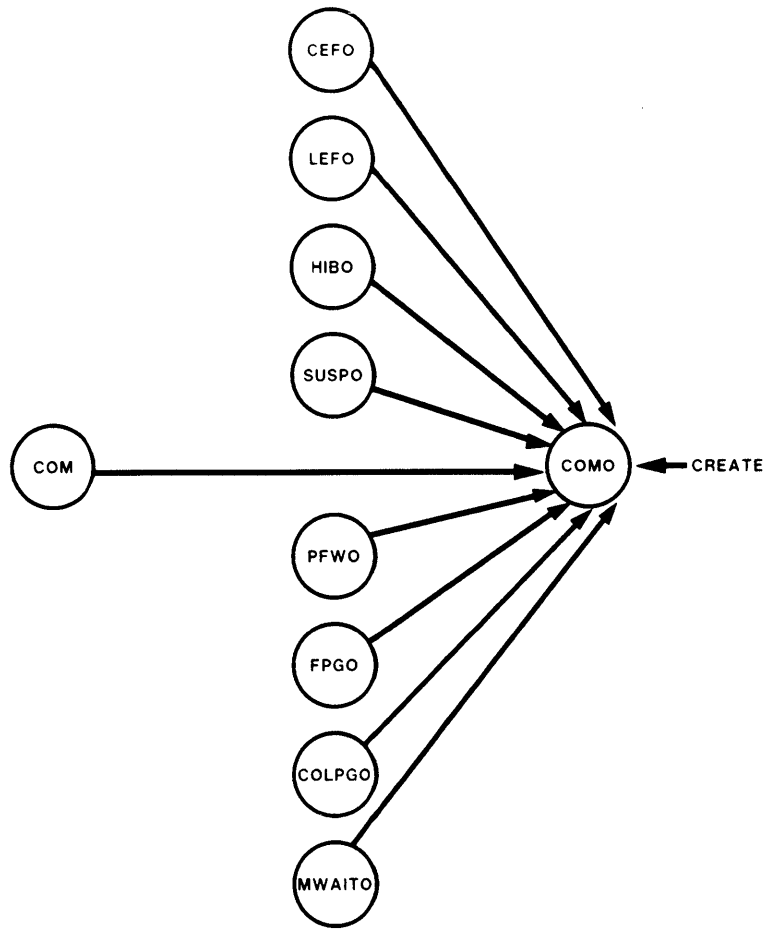


Figure 6 Ways to Become Computable (Outswapped)

HOW PROCESS STATES ARE IMPLEMENTED

Queues

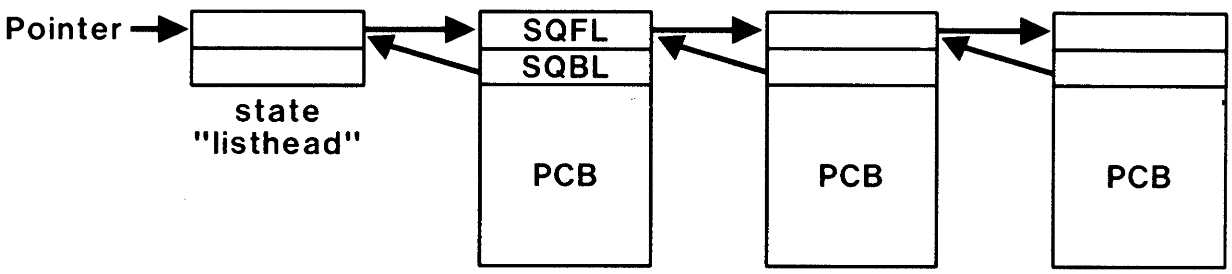


Figure 7 A State Implemented by a Queue

- 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

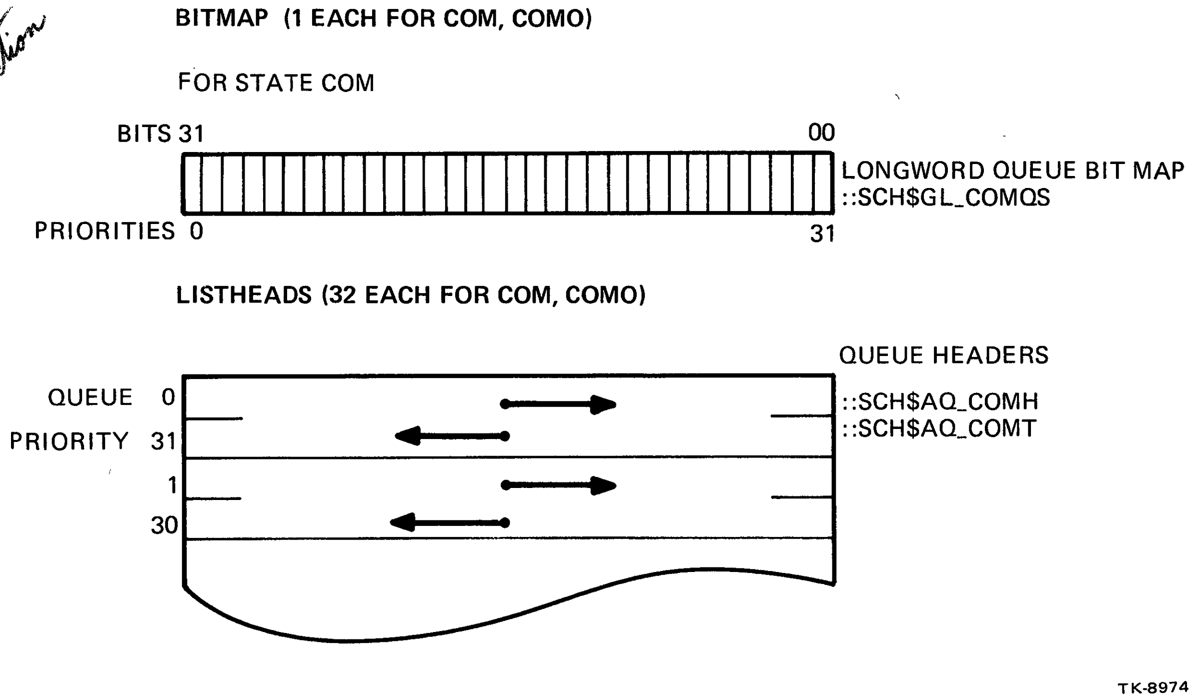


Figure 8 Implementation of COM and COMO States

- 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 (as 31 minus priority)
- COMO state is implemented like COM state
  - 32 more queues
  - Another summary longword

Example of Computable Queues

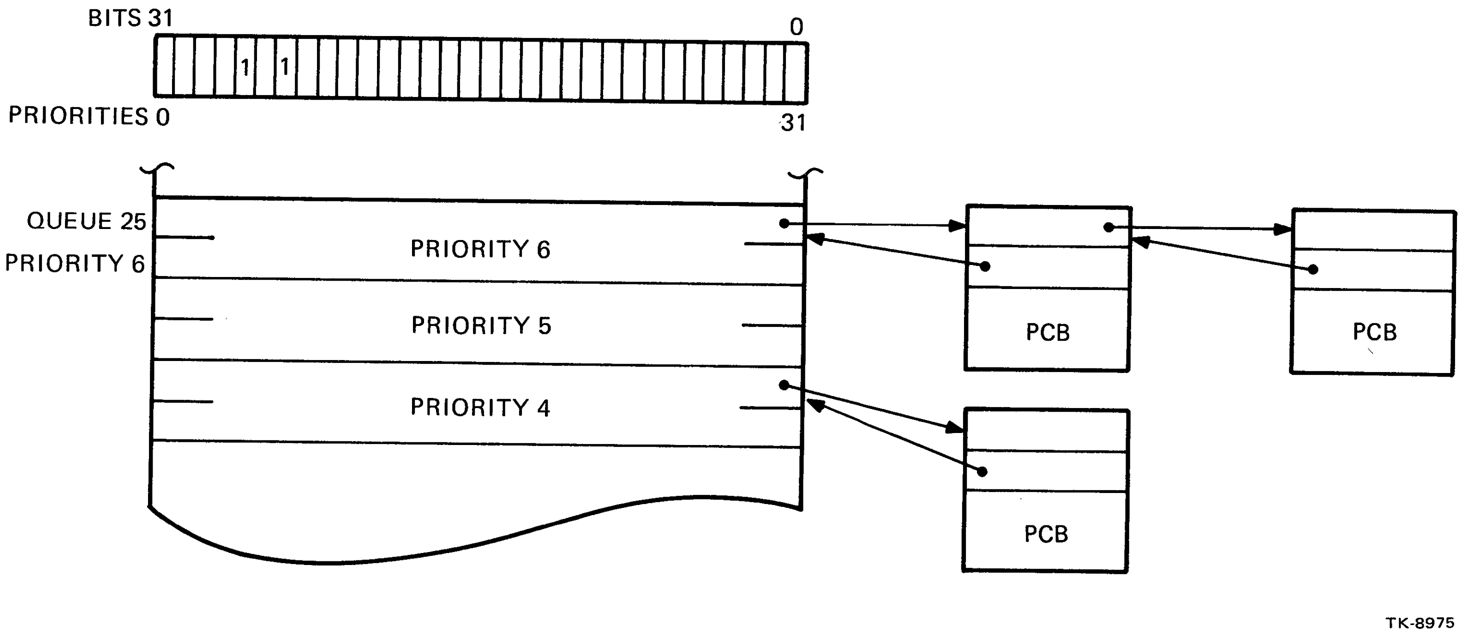


Figure 9 Example of Computable Queues

- COM processes at priorities 4 and 6
  - Bit 25 in summary longword is set
  - Queue for priority 6 has entries
  - Bit 27 in summary longword is set
  - Queue for priority 4 has an entry

Implementation of Wait States

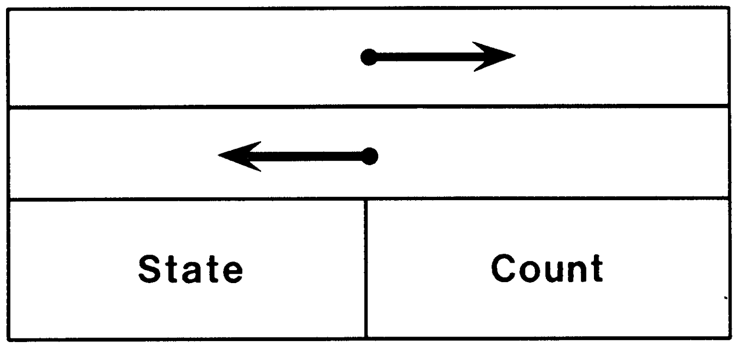


Figure 1Ø Wait State Listhead

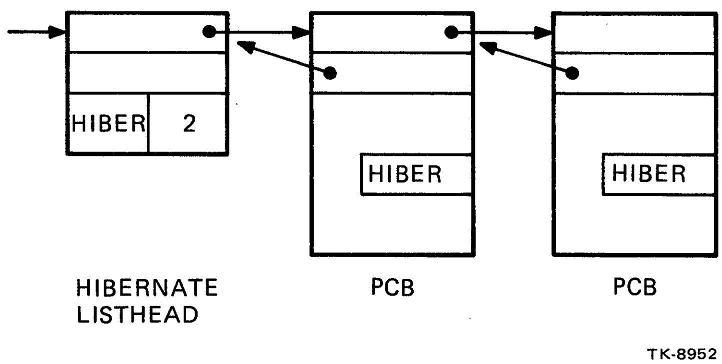


Figure 11 Implementation of Wait States

Implementation of CEF State

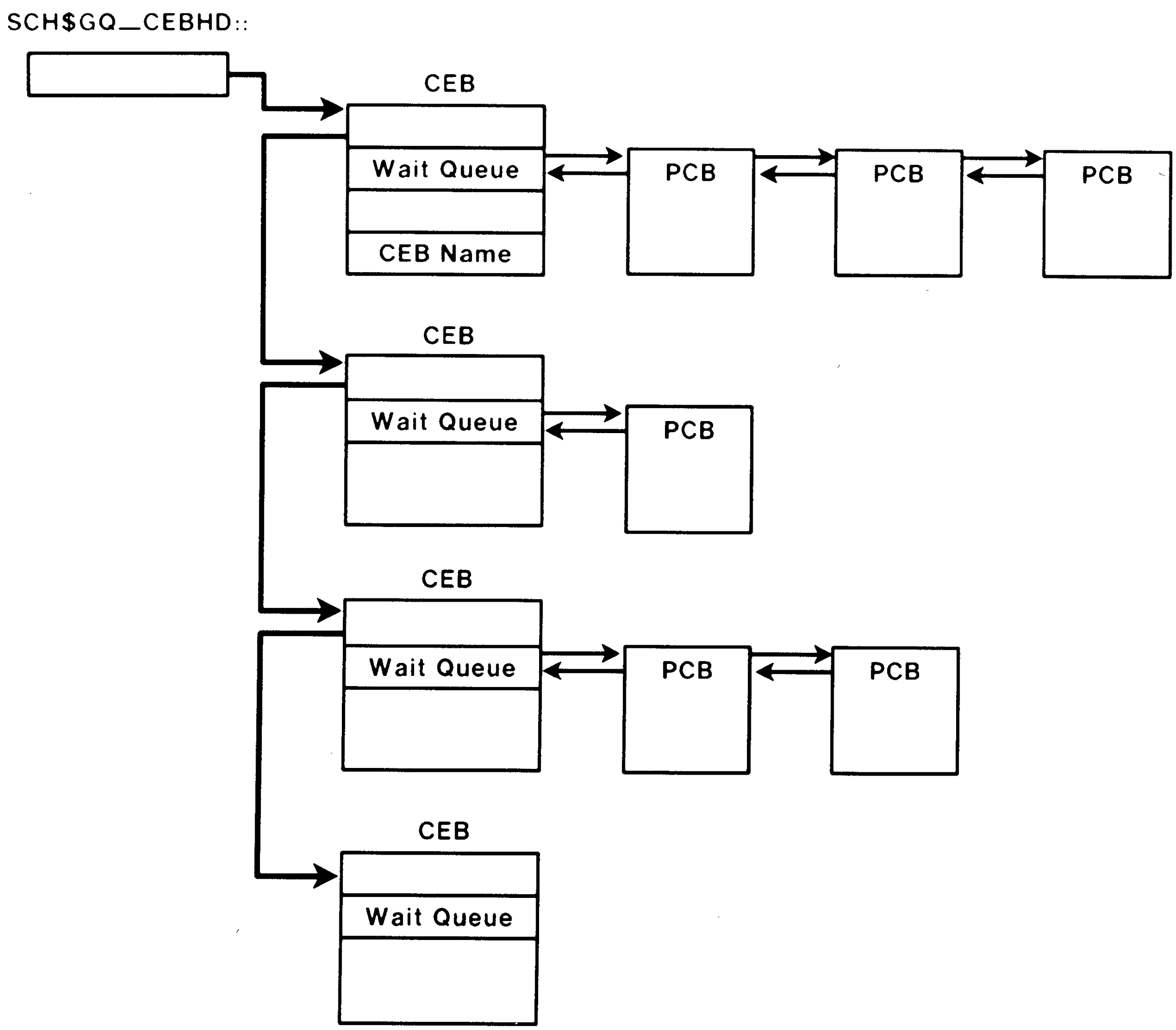


Figure 12 Implementation of CEF State

- CEB created when event flag cluster created
- CEB contains the cluster, CEF state queue listhead, and other information about the cluster
- One CEF state queue for each CEF cluster

Summary of Scheduling States

- Current
  - Implemented with one longword pointer
  - Contains at most one process
- Computable and computable-outswapped
  - Each consists of a summary longword, and 32 queues
- Voluntary wait (LEF, LEFO, SUSP, SUSPO, HIB, HIBO)
  - One queue for each state
- Involutary wait (PFW, PFWO, FPG, FPGO, COLPG, COLPGO, MWAIT, MWAITO)
  - In four queues
  - Resident and outswapped in same queue (differentiate with resident bit in PCB$L_STS)
  - Usually not in these states very often

Process Data Structures Related to Scheduling

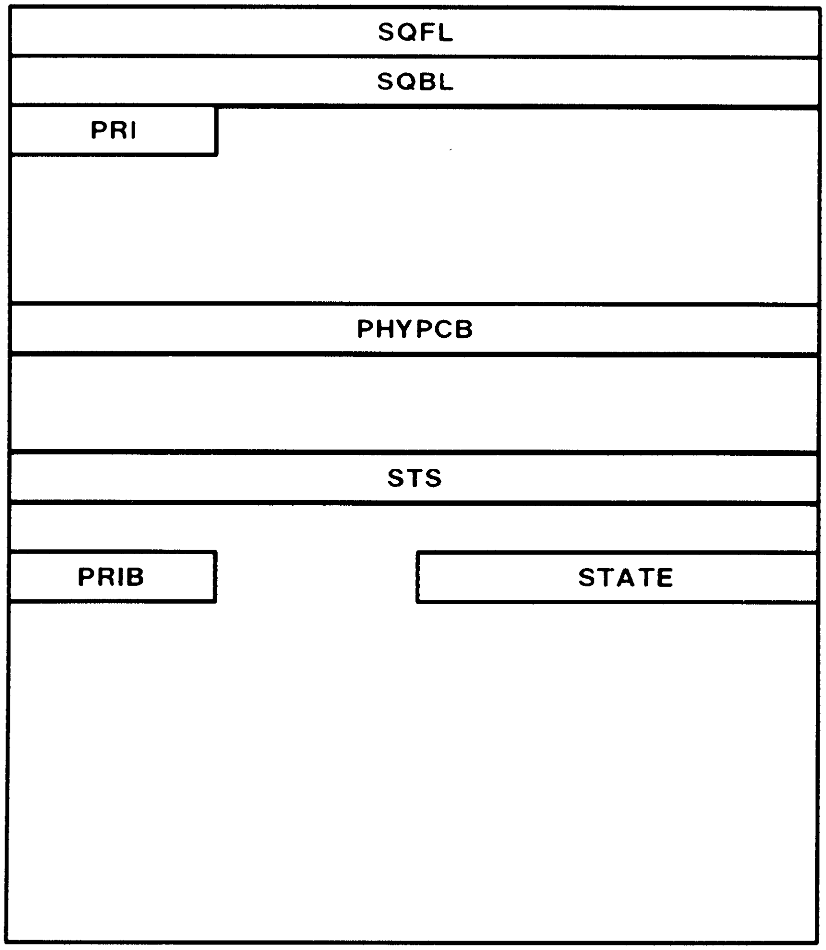


Figure 13 Scheduling Fields in Software PCB

- SQFL, SQBL - state queue forward, backward links, link PCBs in a given state
- STATE - process state
- PRI - current software priority
- PRIB - base software priority
- PHYPCB - physical address of hardware PCB
- STS - process status

Saving and Restoring CPU Registers

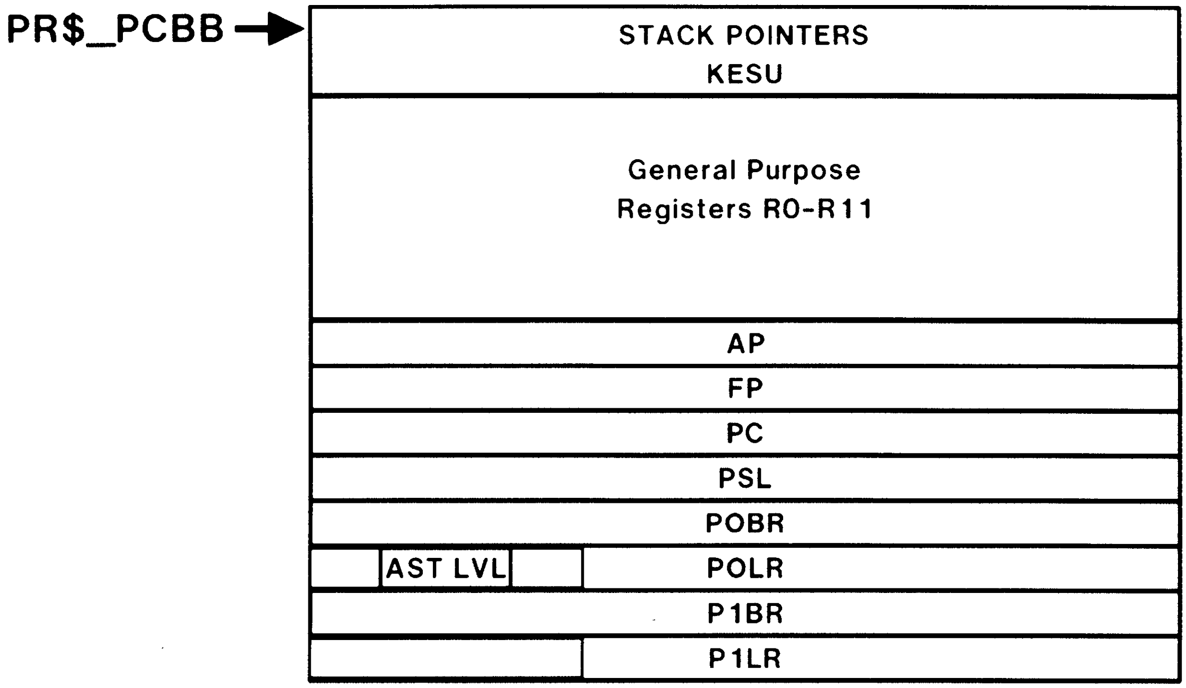


Figure 14 Saving and Restoring CPU Registers

- Process-specific CPU registers saved/restored during context switch

- SVPCTX instruction
  - pops PC, PSL from K-stk (or I-stk if IPL>3)
  - Copies registers to hardware PCB
  - Switches to Interrupt Stack
  - Does not save PØBR, PØLR, P1BR, P1LR, ASTLVL

- LDPCTX instruction
  - Restores registers (except PC, PSL) from hardware PCB
  - Pushes PC, PSL on kernel stack (REI removes them)

THE SCHEDULER (SCHED.MAR)

ensured by IPL 3

```
; SCH$RESCHED - RESCHEDULING INTERRUPT HANDLER
;
; THIS ROUTINE IS ENTERED VIA THE IPL 3 RESCHEDULING INTERRUPT.
; THE VECTOR FOR THIS INTERRUPT IS CODED TO CAUSE EXECUTION
; ON THE KERNEL STACK.
;
; ENVIRONMENT:      IPL=3 MODE=KERNEL IS=0
; INPUT:            00(SP)=PC AT RESCHEDULE INTERRUPT
;                   04(SP)=PSL AT INTERRUPT.
;--
        .ALIGN  LONG
MPH$RESCHED::                           ;MULTI-PROCESSING CODE HOOKS IN HERE
SCH$RESCHED::                           ;RESCHEDULE INTERRUPT HANDLER
        SETIPL  #IPL$_SYNCH             ;SYNCHRONIZE SCHEDULER WITH EVENT REPORTING
        SVPCTX                          ;SAVE CONTEXT OF PROCESS
        MOVL    L^SCH$GL_CURPCB,R1      ;GET ADDRESS OF CURRENT PCB
        MOVZBL  PCB$B_PRI(R1),R2        ;CURRENT PRIORITY
        BBSS    R2,L^SCH$GL_COMQS,10$   ;MARK QUEUE NON-EMPTY
10$:    MOVW    #SCH$C_COM,PCB$W_STATE(R1) ;SET STATE TO RES COMPUTE
        MOVAQ   SCH$AQ_COMT[R2],R3      ;COMPUTE ADDRESS OF QUEUE
        INSQUE  (R1),@(R3)+             ;INSERT AT TAIL OF QUEUE
;+
; SCH$SCHED - SCHEDULE NEW PROCESS FOR EXECUTION
;
; THIS ROUTINE SELECTS THE HIGHEST PRIORITY EXECUTABLE PROCESS
; AND PLACES IT IN EXECUTION.
;-
MPH$SCHED::                             ;MULTI-PROCESSING CODE HOOKS IN HERE
SCH$SCHED::                             ;SCHEDULE FOR EXECUTION
        SETIPL  #IPL$_SYNCH             ;SYNCHRONIZE SCHEDULER WITH EVENT REPORTING
        FFS     #0,#32,L^SCH$GL_COMQS,R2        ;FIND FIRST FULL STATE
        BEQL    SCH$IDLE                ;NO EXECUTABLE PROCESS??
        MOVAQ   SCH$AQ_COMH[R2],R3      ;COMPUTE QUEUE HEAD ADDRESS
        REMQUE  @(R3)+,R4               ;GET HEAD OF QUEUE
        BVS     QEMPTY                  ;BR IF QUEUE WAS EMPTY (BUG CHECK)
        BNEQ    20$                     ;QUEUE NOT EMPTY
        BBCC    R2,L^SCH$GL_COMQS,20$   ;SET QUEUE EMPTY
20$:
        CMPB    #DYN$C_PCB,PCB$B_TYPE(R4)  ;MUST BE A PROCESS CONTROL BLOCK
        BNEQ    QEMPTY                  ;OTHERWISE FATAL ERROR
        MOVW    #SCH$C_CUR,PCB$W_STATE(R4)      ;SET STATE TO CURRENT
        MOVL    R4,L^SCH$GL_CURPCB      ;NOTE CURRENT PCB LOC
        CMPB    PCB$B_PRIB(R4),PCB$B_PRI(R4)    ;CHECK FOR BASE
                                                ;PRIORITY=CURRENT
        BEQL    30$                     ;YES, DONT FLOAT PRIORITY
        BBC     #4,PCB$B_PRI(R4),30$    ;DONT FLOAT REAL TIME PRIORITY
        INCB    PCB$B_PRI(R4)           ;MOVE TOWARD BASE PRIO
30$:    MOVB    PCB$B_PRI(R4),L^SCH$GB_PRI      ;SET GLOBAL PRIORITY
        MTPR    PCB$L_PHYPCB(R4),#PR$_PCBB      ;SET PCB BASE PHYS ADDR
        LDPCTX                          ;RESTORE CONTEXT
        REI                             ;NORMAL RETURN

SCH$IDLE:                               ;NO ACTIVE, EXECUTABLE PROCESS
        SETIPL  #IPL$_SCHED             ;DROP IPL TO SCHEDULING LEVEL
        MOVB    #32,L^SCH$GB_PRI        ;SET PRIORITY TO -1(32) TO SIGNAL IDLE
        BRB     SCH$SCHED               ;AND TRY AGAIN

QEMPTY:         BUG_CHECK QUEUEMPTY,FATAL       ;SCHEDULING QUEUE EMPTY

        .END
```

Example 1 The Scheduler (SCHED.MAR)

Comments on SCHED.MAR:

1. Current process ---> computable resident

   a. Entry point

   b. Synchronize access to scheduler database

   c. Save hardware context of current process in hardware PCB

   d. Insert PCB at tail of COM queue

2. Highest-priority computable resident process ---> current

   a. Entry point

   b. Synchronize access to scheduler database

   c. Remove PCB from head of COM queue

   d. Restore hardware context, push PC and PSL onto stack

   e. Transfer control to current process

BOOSTING SOFTWARE PRIORITY OF NORMAL PROCESSES

- Usually normal interactive process has base priority 4
- To help interactive processes compete with compute-bound processes
  - Boosts applied upon certain events (I/O completion, resource available)
  - Different boosts for different events
  - Current priority equals greater of:
    - Current priority
    - Base priority plus boost
  - Lowering of priority
    - Each time process scheduled, decrement priority (until reach base priority)
    - Return to base priority at quantum end if COMO process exists
  - Not allowed to boost above normal priority range (Ø-15)

Example of Process Scheduling

Table 1 Initial Conditions for Scheduling Example

Process	Type	Base Priority	Priority	State
Swapper	System	16	16	HIB
Null	Compute Bound	0	0	COM
A	Compute Bound	4	9	CUR
B	I/O Bound	4	10	COMO
C	Real-Time	18	18	HIB

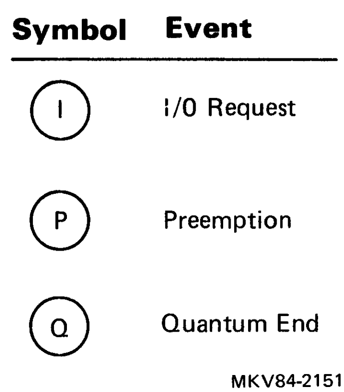


Figure 15 Scheduling Example Symbols

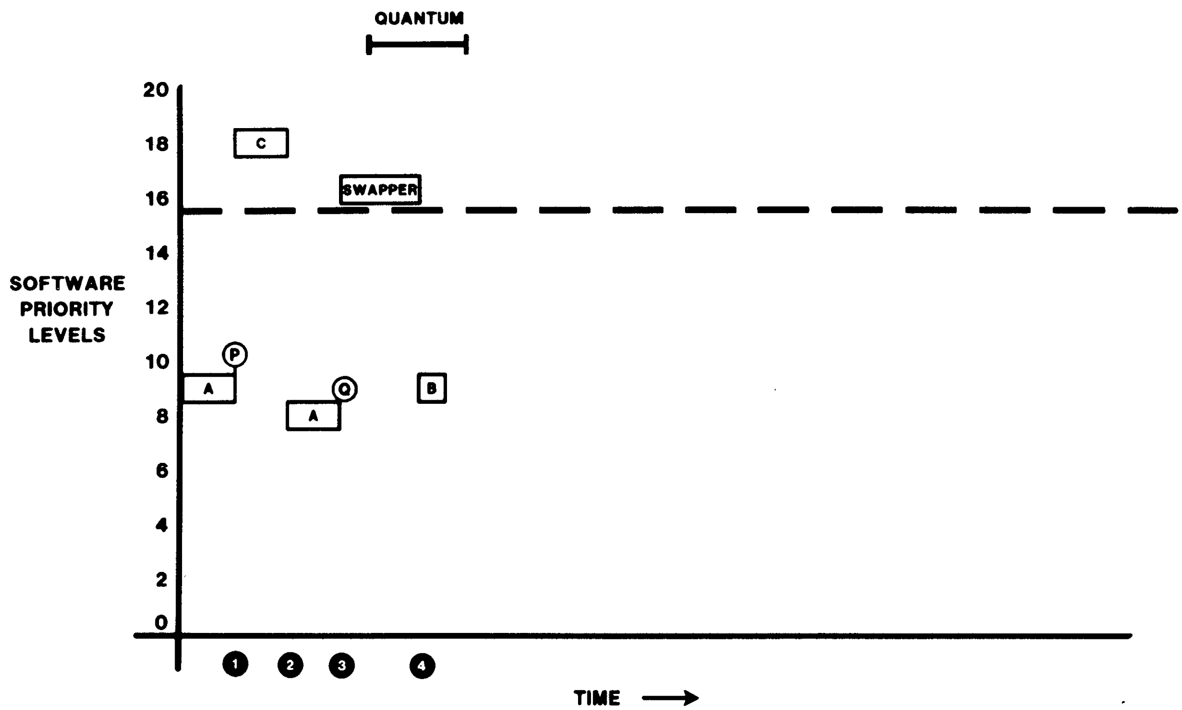


Figure 16 Example of Process Scheduling - Part 1

1. Process **C** becomes computable. Process **A** is preempted.

2. **C** hibernates. **A** executes again, one priority level lower.

3. **A** experiences quantum end and is rescheduled at its base priority. **B** is computable outswapped.

4. The swapper process executes to inswap **B**. **B** is scheduled for execution.

Only because of Inswap pending

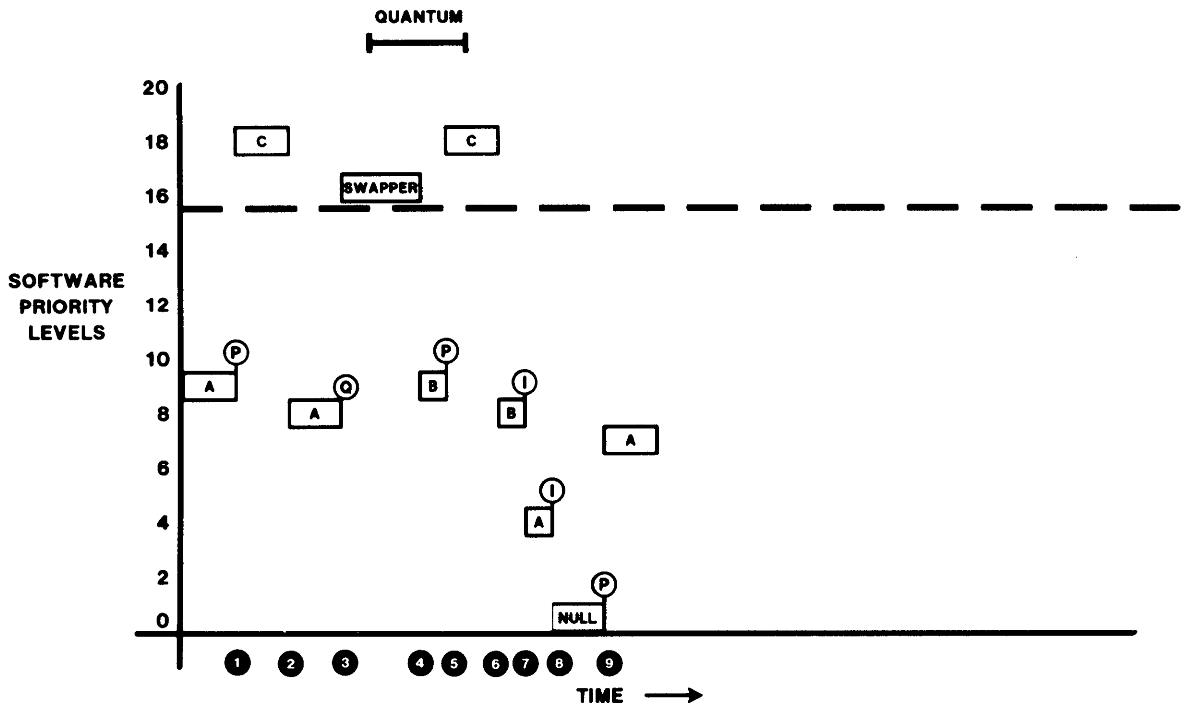


Figure 17 Example of Process Scheduling - Part 2

5. **B** is preempted by C.
6. **B** executes again, one priority level lower.
7. **B** requests an I/O operation (not terminal I/O). **A** executes at its base priority.
8. **A** requests a terminal output operation. The null process executes.
9. **A** executes following I/O completion at its base priority plus 3. (The applied boost was 4.)

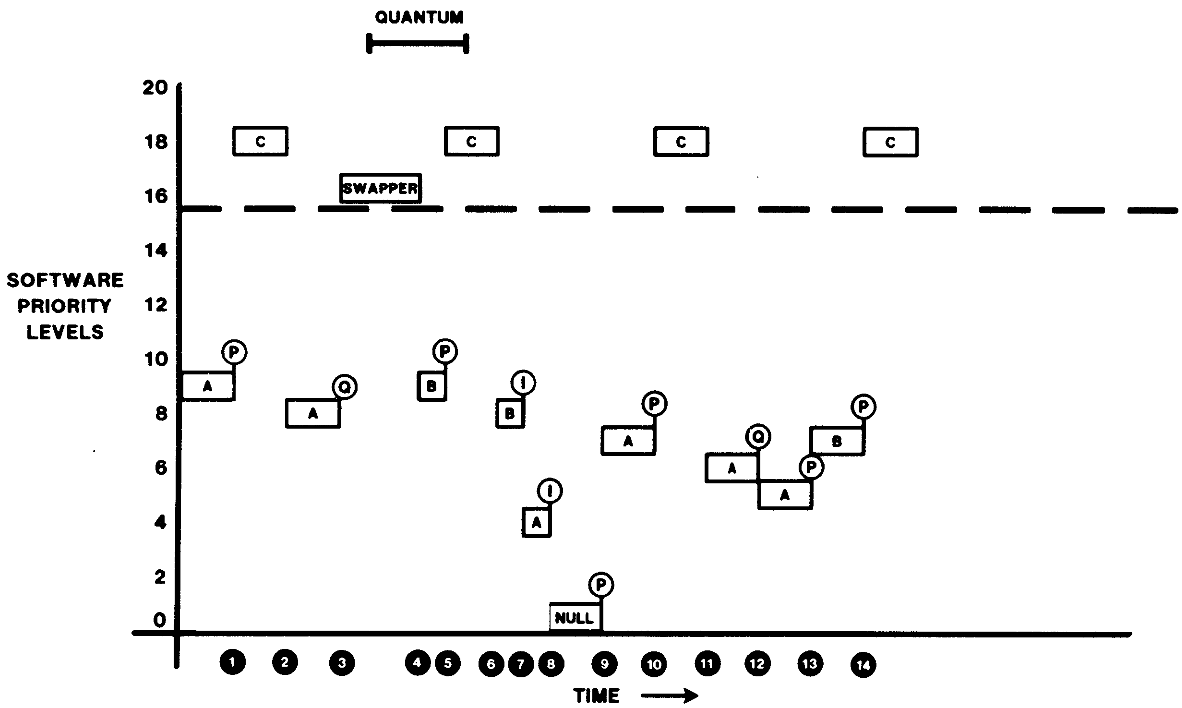


Figure 18 Example of Process Scheduling - Part 3

10. **A** is preempted by **C.**

11. **A** executes again, one priority level lower.

12. **A** experiences quantum end and is rescheduled at one priority level lower.

13. **A** is preempted by **B.** A priority boost of 2 is not applied to **B** because the result would be less than the current priority.

14. **B** is preempted by **C.**

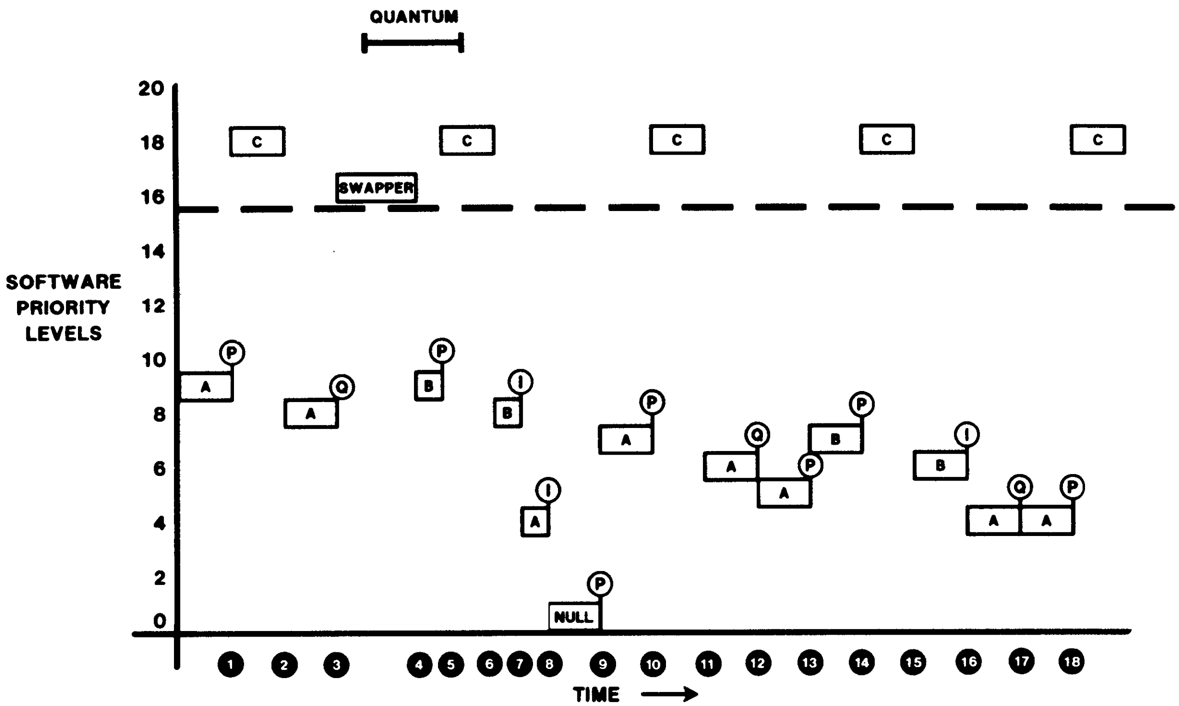


Figure 19 Example of Process Scheduling - Part 4

15. **B** executes again, one priority level lower.

16. **B** requests an I/O operation. **A** executes at its base priority.

17. **A** experiences quantum end and is rescheduled at the same priority (its base priority).

18. **A** is preempted by **C**.

IMPLEMENTATION OF PROCESS STATE CHANGES

Table 2 Operating System Code for Scheduling Functions

Function	Module	Routines
Change between CUR and COM	SCHED.MAR	SCH$RESCHED SCH$SCHED
Move between resident and outswapped	SWAPPER.MAR	SWAPSCHED INSWAP OUTSWAP
Move in and out of wait states	RSE.MAR	SCH$RSE SCH$UNWAIT (and others)
Quantum end processing	RSE.MAR	SCH$QEND

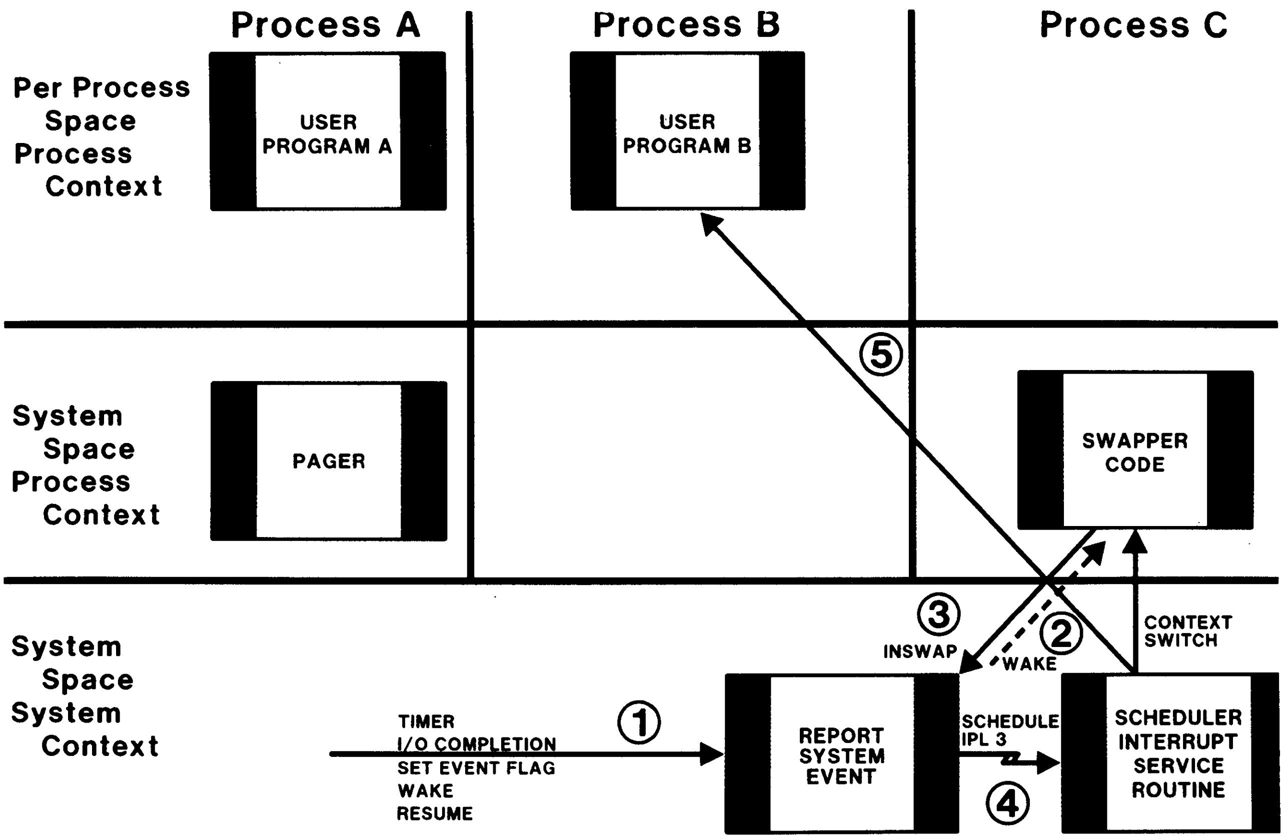


Figure 20 Interaction of Scheduling Components

Report System Event Component (RSE.MAR)

1. System events cause transitions between process states.
2. These transitions are accomplished by the code in RSE.MAR.
3. Inputs to RSE
   a. PCB address
   b. Event number (number for WAKE, CEF SET, and so on)
4. RSE flow
   a. Event checked for significance (for example, WAKE only if in HIBER state).
   b. PCB removed from wait queue and wait queue header count decremented.
   c. PCB inserted on COM or COMO state queue after priority adjustment, and summary bit set.
   d. Swapper process can be awakened (if PCB was inserted on COMO queue).
   e. Scheduler interrupt at IPL 3 requested if the new computable process has software priority greater than that of current process.

STEPS AT QUANTUM END

Real-Time Process

1. Reset PHD$B_QUANT to full quantum value.
2. Clear initial quantum bit PCB$V_INQUAN in PCB$L_STS.

Normal Process

1. Reset PHD$B_QUANT to full quantum value.
2. Clear initial quantum bit PCB$V_INQUAN in PCB$L_STS.
3. If any outswapped process computable, set current software priority PCB$B_PRI to base priority PCB$B_PRIB.
4. If SWAPPER needed, wake SWAPPER.
5. If CPU limit imposed, and limit has expired, queue AST to process for process deletion.
6. If not, then calculate automatic working set adjustment.
7. Request scheduling interrupt at IPL 3.

Automatic Working Set Adjustment

- Goal: optimal working set size
  - Large enough to allow good program performance
  - Small enough to optimize overall memory usage
- Adjustment calculated at quantum end
  - If high paging rate, want to increase working set size
  - If low paging rate, may want to decrease working set size (take back some physical memory)
- Usually gives large increases, small decreases
- Only affects the list size, not the number of entries in use
- No adjustment done for real-time processes
- Can disable adjustment for normal processes
  - Perprocess: $ SET WORKING_SET/NOADJUST
  - System-wide: SYSGEN> SET WSINC Ø

Automatic Working Set Adjustment

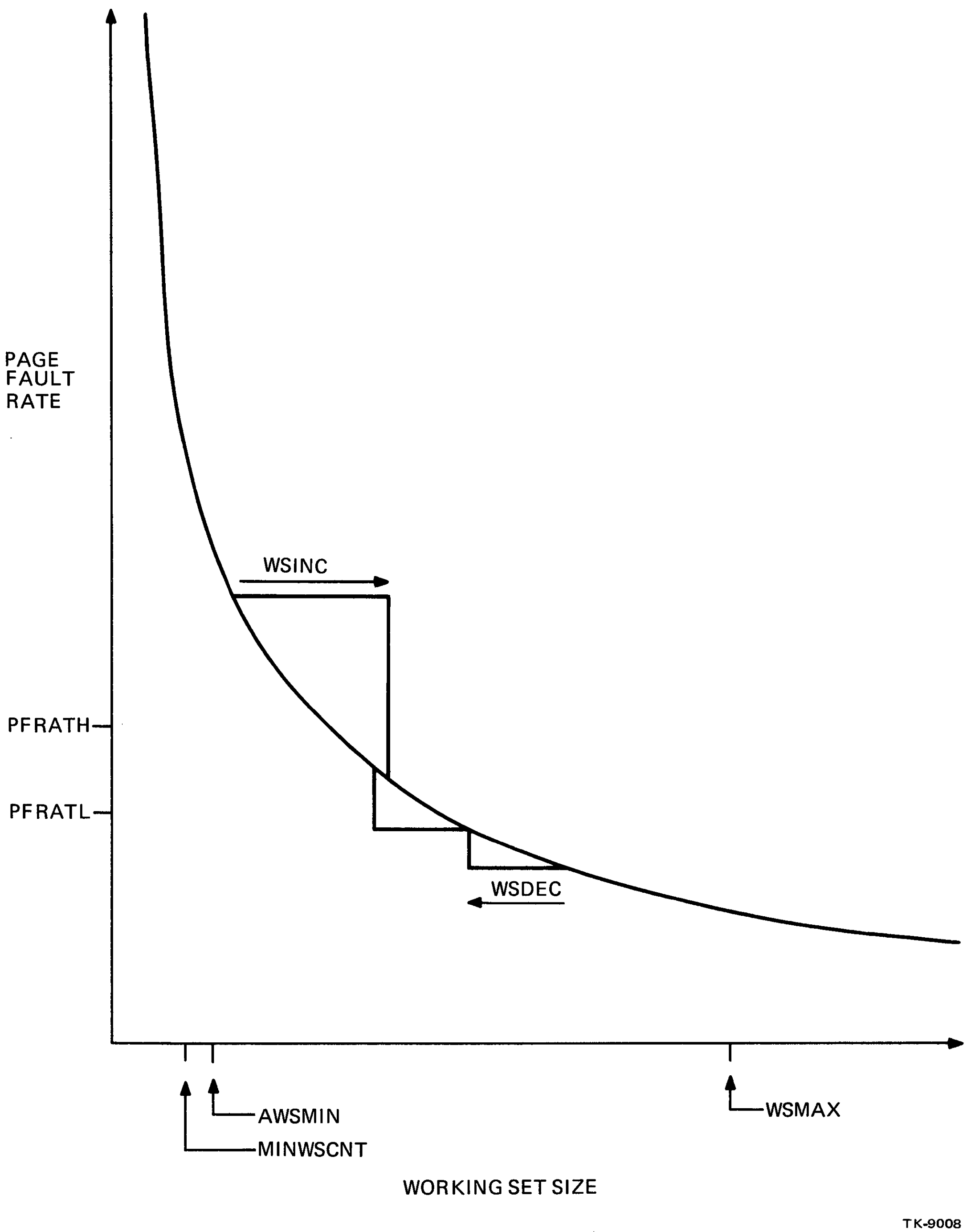


Figure 21 Automatic Working Set Adjustment

Rules for Working Set Adjustment

1. If PFRATL < PFRate < PFRATH, no adjustment is necessary.
2. If PFRate > PFRATH then perhaps WSSIZE = WSSIZE + WSINC.
   - WSSIZE can grow to WSQUOTA anytime
   - WSSIZE can grow to WSEXTENT if free pages > BORROWLIM
3. If PFRate < PFRATL then perhaps WSSIZE = WSSIZE - WSDEC.
   - WSSIZE can shrink to AWSMIN (no smaller)

Example 2 Working Set Adjustment Algorithm

Example of Working Set Size Variation

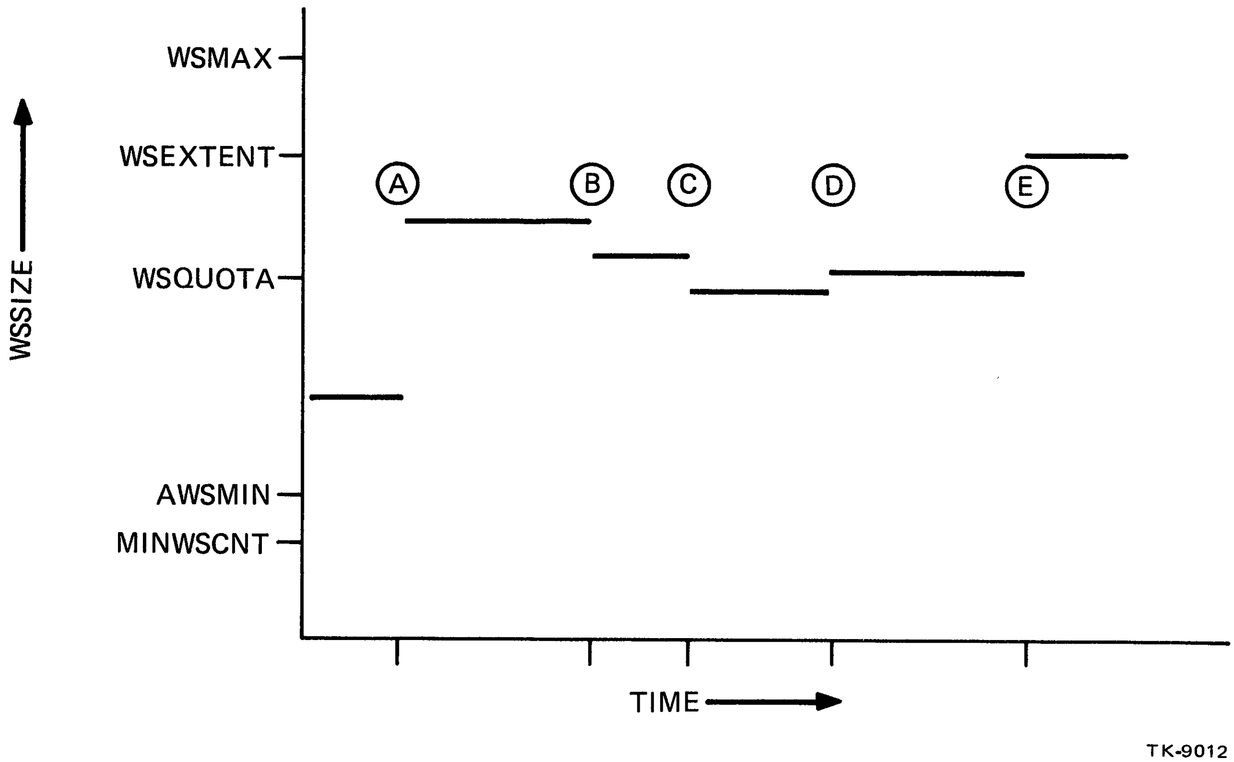


Figure 22 WSSIZE Variation Over Time

Table 3 Reasons for Working Set Size Variations

Time	Reason for WSSIZE Change
a	Page faults > PFRATH Free page count > BORROWLIM
b	Page faults < PFRATL
c	Page faults < PFRATL
d	Page faults > PFRATH Free page count < BORROWLIM
e	Page faults > PFRATH Free page count > BORROWLIM

Forcing Processes to Quantum End

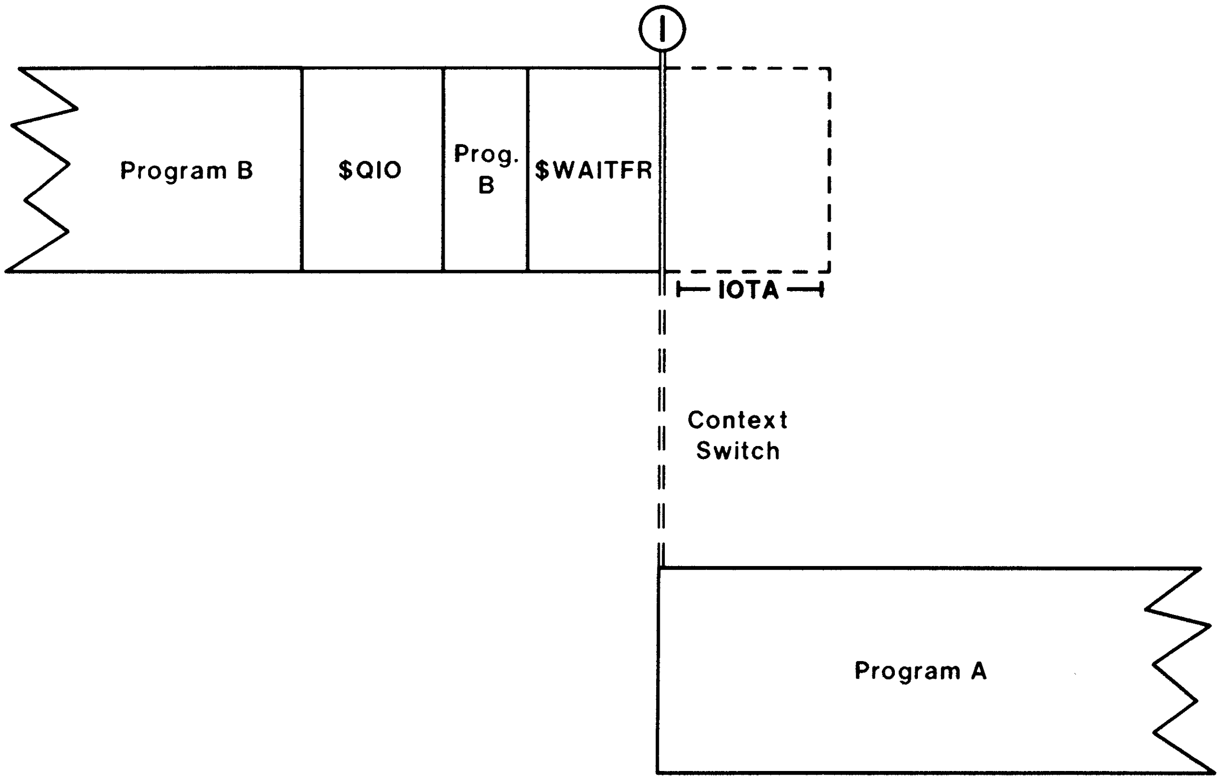


Figure 23 Use of the IOTA System Parameter

- IOTA - special system parameter (in 10 ms units)
- Deduct IOTA units from time quantum when process enters wait state
- Used to force processes to quantum end
- Not charged to process CPU limit

SOFTWARE PRIORITY LEVELS OF PROCESSES

Table 4 Software Priority Levels of Processes on VMS

Process	Base Priority	Purpose
NULL	0	Consume idle CPU time
default user	4	User activities
SYMBIONT_n	4	Input/output symbiont
OPCOM	6	Operator communications
ODS-1 disk ACPs	8	ODS-1 disk file structure
Tape ACPS	8	Tape file structure
ERRFMT	7	Write error log buffers
JOB_CONTROL	8	Queue and accounting manager
NETACP	8	DECnet ACP
REMACP	8	Remote ACP
SWAPPER	16	System-wide memory manager

- Base priority of process determined by argument to $CREPRC system service
- Base priority of system processes
  - Most are established during system initialization
  - Base priority of ACPs is controlled by ACP_BASEPRIO system parameter
- Normal processes receive priority boosts

SUMMARY

Table 5 SYSGEN Parameters Relevant to Scheduling

Function	Parameter
Base priority for Ancillary Control Processes	ACP_BASEPRIO
Minimum number of working set pages	AWSMIN
Minimum amount of time that must elapse for significant sample of a process page fault rate	AWSTIME
Minimum number of pages required on free page list before working sets are allowed to grow beyond WSQUOTA (checked at quantum end)	BORROWLIM
Base default priority for processes	DEFPRI
Time alloted to each of a process's exit handlers after CPU limit expires	EXTRACPU
Amount of time to deduct from process quantum for each voluntary wait	IOTA (*)
Minimum number of fluid working set pages	MINWSCNT
Page fault rate above which VMS attempts to increase the process working set size	PFRATH
Page fault rate below which VMS attempts to decrease the process working set size	PFRATL
Maximum amount of CPU time a normal process can receive before control passes to a computable process of equal priority	QUANTUM
Number of pages for working set size decrease	WSDEC
Number of pages for working set size increase	WSINC
Maximum number of pages for any working set	WSMAX

(*) = special SYSGEN parameter

Process Creation and Deletion

INTRODUCTION

This module discusses the operations required to create and delete processes under VAX/VMS.

Process creation and deletion involve several different components of VMS. Discussion in this module focuses on the process context of each component. Some operations execute in the context of the process that requests the particular action, while others execute in the context of the target process.

Interactive and batch processes involve additional components such as command language interpreters (CLIs), the job controller, and possibly the input symbiont process. In addition, interactive and batch processes may require execution of the LOGINOUT image for such functions as mapping the CLI.

The discussion of the life cycle of processes should contribute to a better understanding of the implications of multiprogramming application designs.

OBJECTIVES

1. To assist in the design of efficient multiprogramming applications, the student must understand how the following kinds of processes are created and deleted:

   - User-created processes
   - Interactive processes
   - Batch processes

2. To alter process characteristics (beyond the functionality provided by DCL), the student must know how process context is built.

3. To assist in managing processes, the student must understand the effects of altering SYSGEN parameters related to process creation and deletion.

RESOURCES

Reading

1. VAX/VMS Internals and Data Structures, chapters on process creation, process deletion, and interactive and batch jobs.

Source Modules

Facility Name	Module Name
SYS	SHELL PROCSTRT SYSCREPRC, SYSDELPRC
LOGIN	
JOBCTL	
INPSMB	

TOPICS

I. Process Creation

   A. Roles of operating system programs

   B. Creation of process data structures

II. Types of Processes

III. Initiating Jobs

   A. Interactive

   B. Batch

IV. Process Deletion

V. SYSGEN Parameters Relating to Process Creation and Deletion

PROCESS CREATION

Table 1 Steps in Process Creation and Deletion

Action	Code
Creating process	SYS$CREPRC
Inswap a process	SWAPPER
Process startup	PROCSTRT
Process deletion	SYS$DELPRC

Table 2 Three Contexts Used in Process Creation

Creator's Context	Swapper's Context	New Process's Context
$CREPRC	From SHELL	PC= EXE$PROCSTRT
• PCB	PHD filled in	PSL= K mode, IPL=2
• JIB	COMO --> COM	Sets up:
• PQB (temp)		- logical names (sys$input...) - Catch-all cond. handler
SW priority boost		- RMS dispatcher - XQP merged in - Image name moved to PHD - Image activated
Process returned COMO		

Creation of PCB, JIB, and PQB

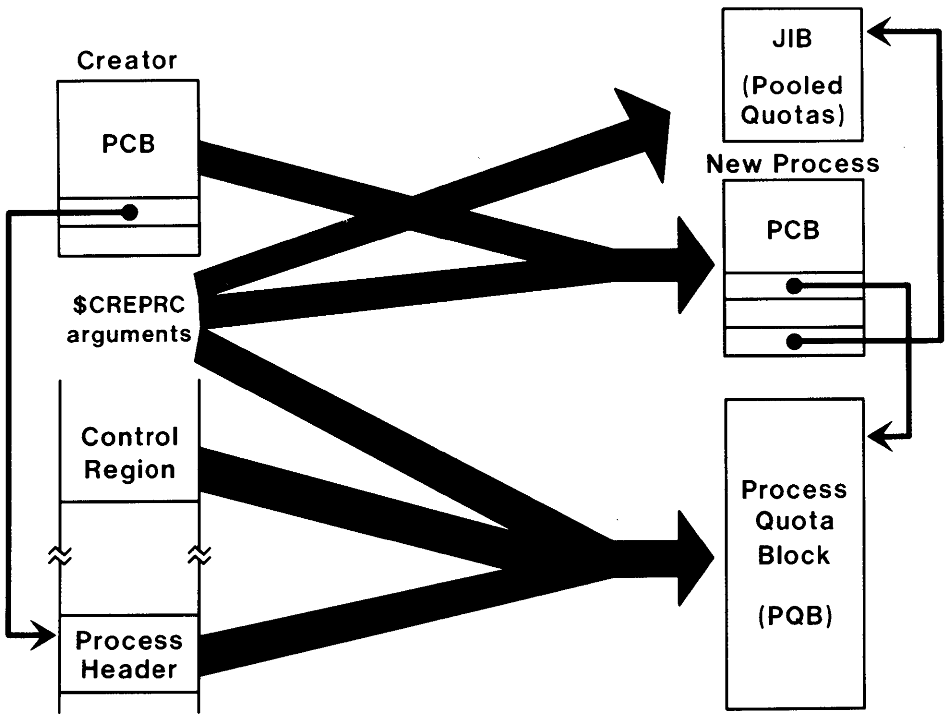


Figure 1 Creation of PCB, JIB and PQB

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

*SYSGEN -

PQL_xxxx parameters

Relationships Between PCBs and JIB

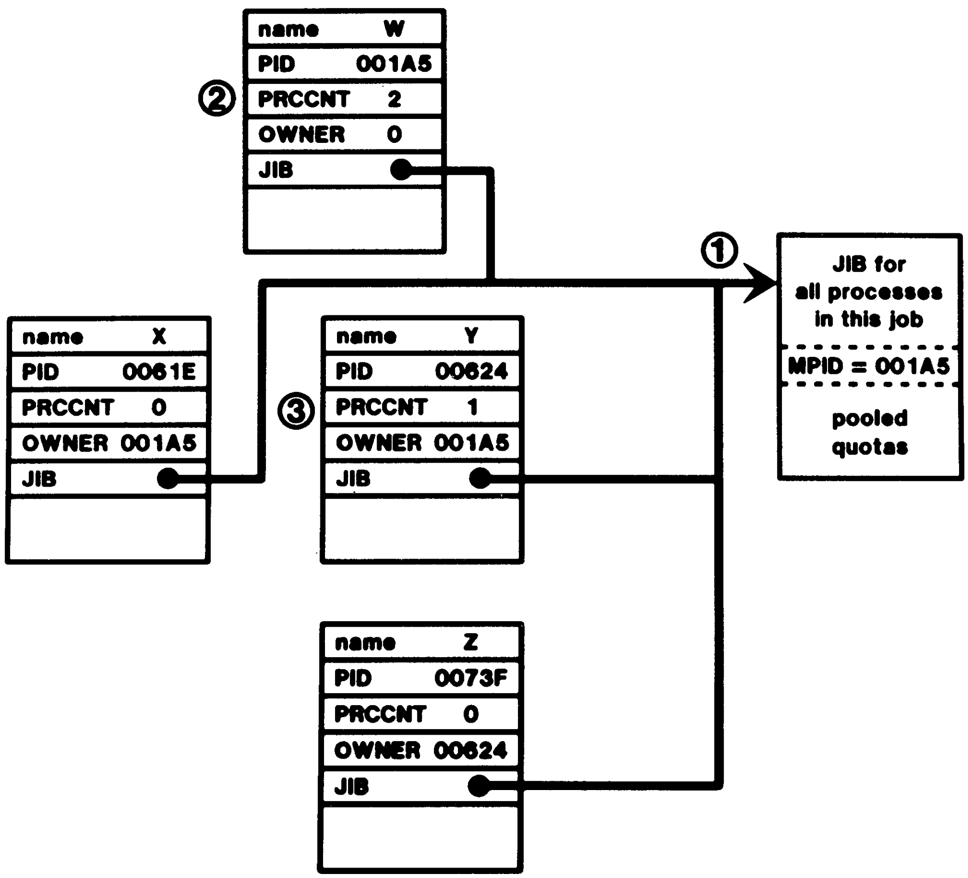


Figure 2 Relationships Between PCBs and JIB

1. All PCBs point to JIB

   W created X and Y

2. W's PRCCNT is 2

3. X and Y owner PID is W PID

   Y created Z

   No pointers from creator to subprocess

PCB Vector

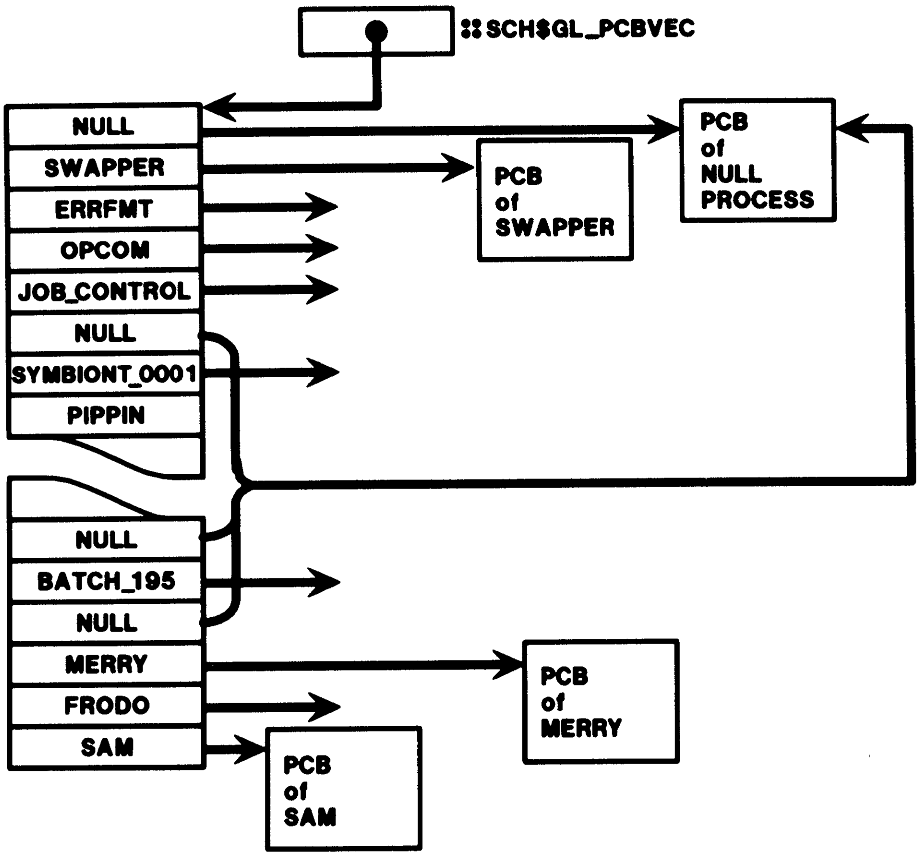


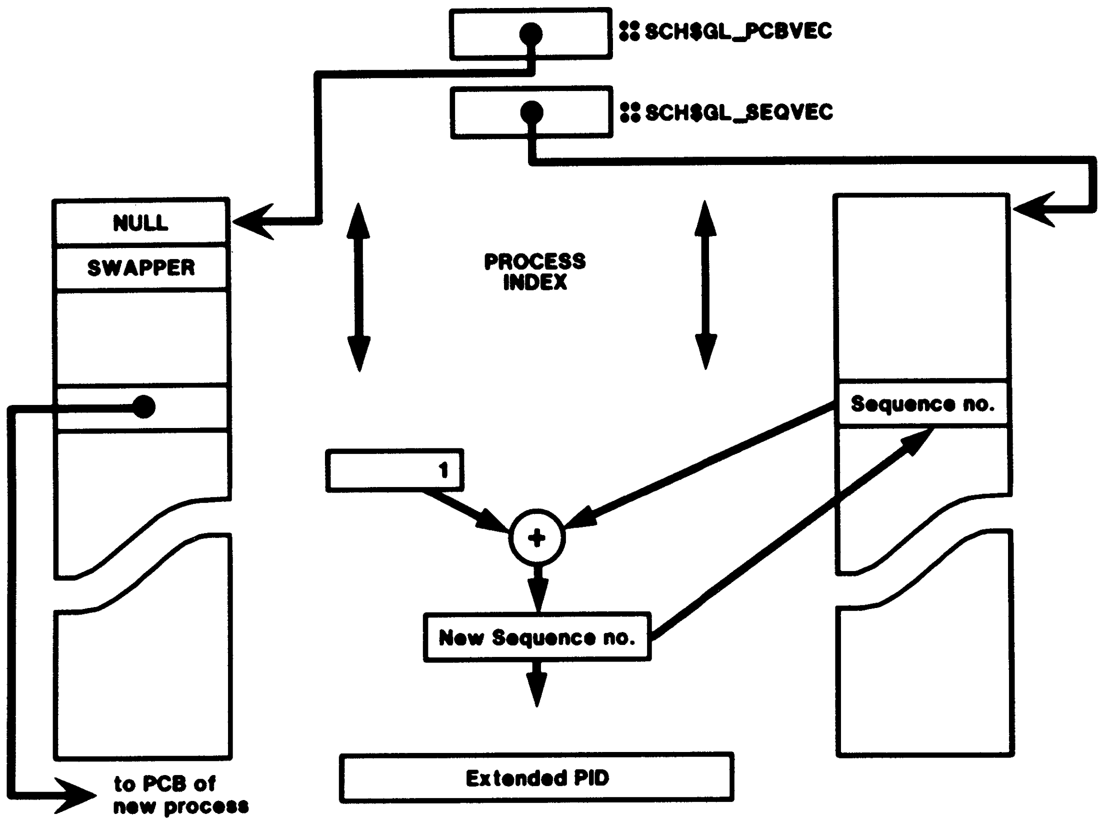
Figure 3 PCB Vector

- On process creation, search for 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 points to start of table

*SYSGEN -

MAXPROCESSCNT

PID and PCB, Sequence Vectors



PCB$L_EPID Figure 4 PID and PCB, Sequence Vectors

unique across cluster (changes from node to node)

- Extended PID contains four parts:
  - Process index into PCB and sequence vectors <13:5>
  - Process sequence number <20:14>
  - Cluster node index <28:21>
  - Node sequence number <30:29>
- PID formed at process creation
- Sequence number incremented each time vector slot re-used
- SCH$GL_SEQVEC points to start of sequence vector

negative sequence # => system I/O (like pager)

Process IDs

- There are actually two PIDs for a process
- Extended PID
  - Visible at the user level
  - Uniquely identifies a process on a single system, and on a VAXcluster
  - Displayed by VMS utilities and system services
  - Stored in PCB at offset PCB$L_EPID
  - Format is very subject to change
- Internal PID
  - Only visible through SDA, and in VMS source code
  - Stored in PCB at offset PCB$L_PID
  - Only contains process index and sequence number (original pre-v4 PID)
  - Used by most kernel-mode code
  - Some privileged data structures contain internal PIDs (for example TQEL_PID, ACBL_PID, and LKB$L_PID)
- Several routines available for manipulating PIDs

Table 3 Routines for Manipulating PIDs

Operation	Mechanism
Convert an extended PID to an internal PID	EXE$EPID_TO_IPID
Convert an internal PID to an extended PID	EXE$IPID_TO_EPID
Return the PCB address given an extended PID	EXE$EPID_TO_PCB
Return the PCB address given an internal PID	EXE$IPID_TO_PCB

Swapper's Role in Process Creation

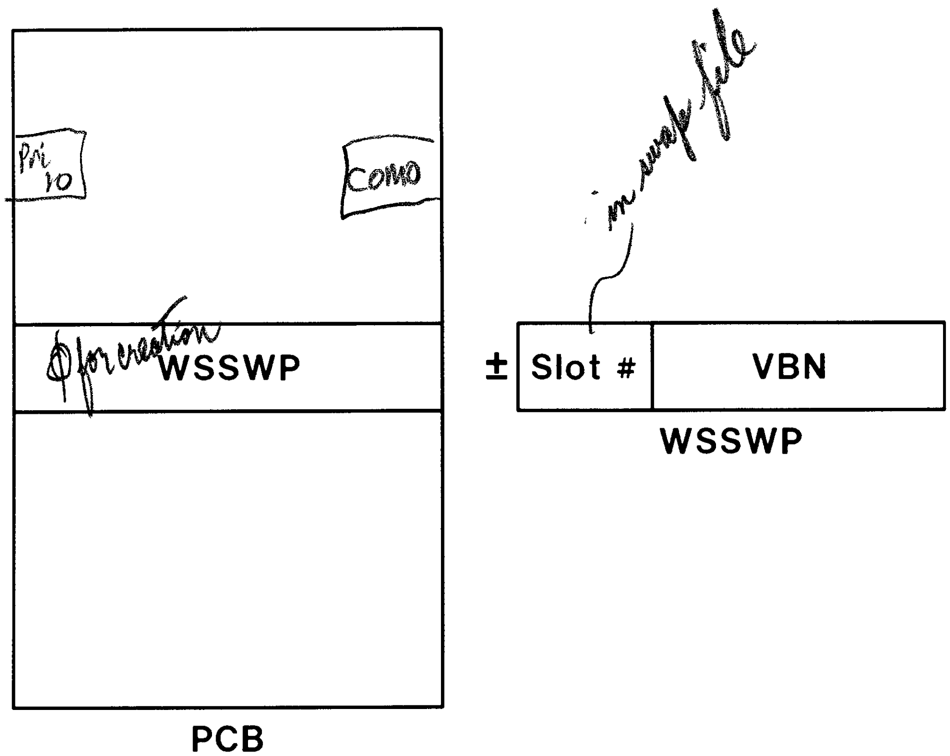


Figure 5 Swapper's Role in Process Creation

- For new process, WSSWP is less than or equal to zero
- WSSWP less than or equal to zero causes SHELL to be copied
- Swapper
  - Stores SYSGEN parameters in PHD
  - Initializes pointers, counters in PHD
  - Initializes system page table entries

PROCSTRT's Role in Process Creation

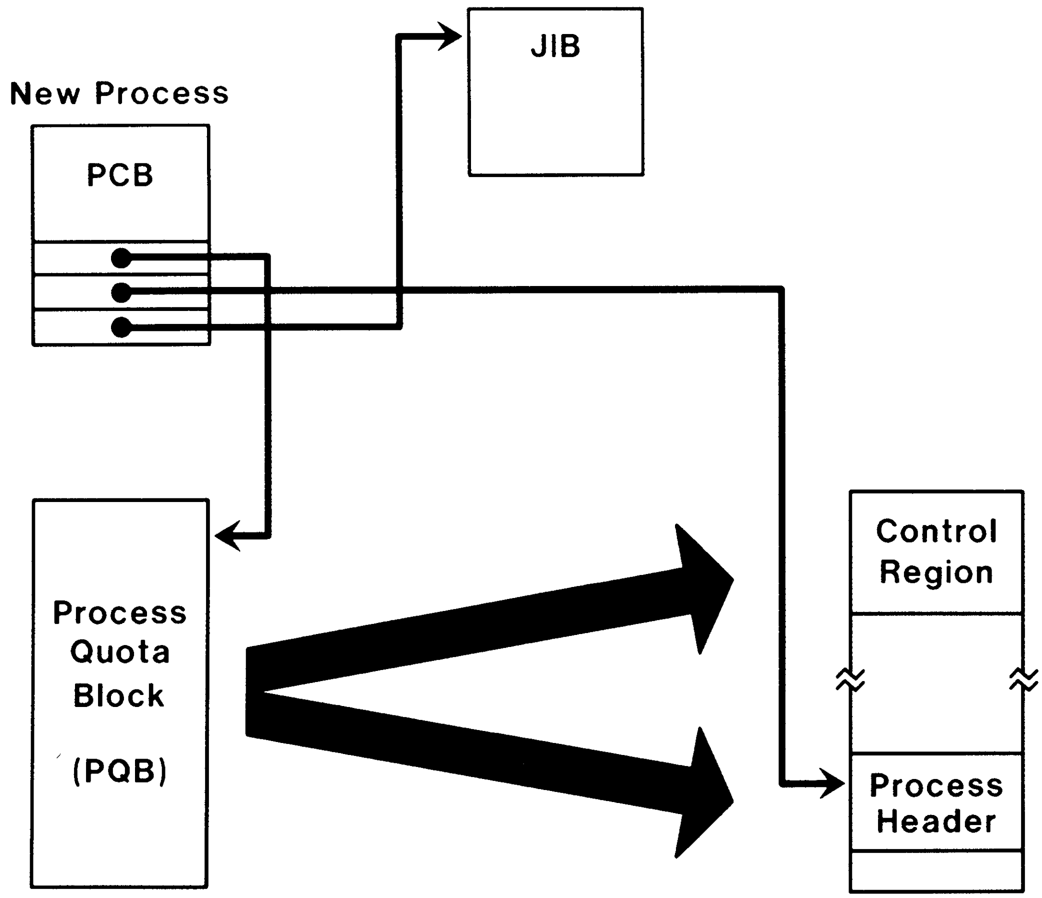


Figure 6 PROCSTRT's Role in Process Creation

- Hardware PCB defined in SHELL
- PC and IPL invoke PROCSTRT at IPL 2
- Code located in SYS.EXE
- Functions
  - PQB information moved to PHD and P1
  - Create logical name tables
  - Change to user mode, IPL 0
  - Map in F11BXQP
  - Call SYS$IMGACT
  - Call image at transfer vector

TYPES OF PROCESSES

Table 4 Types of Processes

	Created By	Creating Code	Special Properties
Batch	Job Controller	SUBMIT, $SNDJBC, $CREPRC	- Deleted upon logout, or at end of command stream - No password check
Detached	Another process	RUN, $CREPRC	- Survives deletion of its creator - May be interactive or not
Network	Network ACP (result of DCL command with node name)	$CREPRC	- Deleted when no more logical links to service
Subprocess	Another process (the owner)	RUN, SPAWN, LIB$SPAWN, $CREPRC	- Cannot survive deletion of owner - Quotas are pooled with owner - May be interactive or not

- RUN and SPAWN call $CREPRC
- After system initialization
  - A process is created by another process
  - Process creation is done by $CREPRC
- An interactive process has:
  - PCB$V_INTER bit set in PCB$L_STS field
  - Non-file-oriented SYS$INPUT

Table 5 PCB Fields Defining Process Types

	PCB$V_BATCH	PCB$V_NETWRK	PCB$V_INTER	PCB$L_OWNER
Network	Ø	1	Ø	Ø
Batch	1	Ø	Ø	Ø
Detached	Ø	Ø	Ø or 1	Ø
Subprocess	Ø	Ø	Ø or 1	non-zero

- PCB$V_xxx symbols represent bits in PCB$L_STS longword
- These bits in the status longword
  - Are intended ONLY for use by the system (for example, the job controller or SPAWN)
  - Can be set using STSFLG argument to $CREPRC
- Interactive processes have the PCB$V_INTER bit set

Table 6 Restrictions on Process Creation

Quota/Limit	Meaning
MAXJOBS	Maximum number of interactive, detached, and batch processes a user may create
MAXDETACH	Maximum number of detached processes a process may create
PRCLM	Limit on number of subprocesses a process may create
Privilege	**Required for**
DETACH or CMKRNL	Creation of a detached process with a different UIC than the creator

The LOGINOUT Image

- Initialize the process permanent data region (store SYS$INPUT value, etc.)
- Perform initializations specific to the type of process
  - Network process

    Validate user name and password
    Map CLI if necessary
  - Batch process

    Obtain job parameters from job controller
  - Subprocess

    No special initialization
  - Interactive process (only if initiated by unsolicited terminal input)

    Ensure that SYS$INPUT is non-file-oriented
    Process system password (if necessary)
    Write SYS$ANNOUNCE (to sys $output)
    Verify user name and password
    Check for re-connections
    Ensure that interactive job quota not exceeded
  - Detached process

    Store user name (no need to verify password)
- Check job limits, account and password expiration, and hourly restrictions
- If interactive process, write welcome message to sys $output
- Initialize CLI if not activating a single image
- Alters process characteristics to match UAF record
  - privileges
  - quotas
- Pass control to CLI or to image

INITIATING JOBS

Initiating an Interactive Job

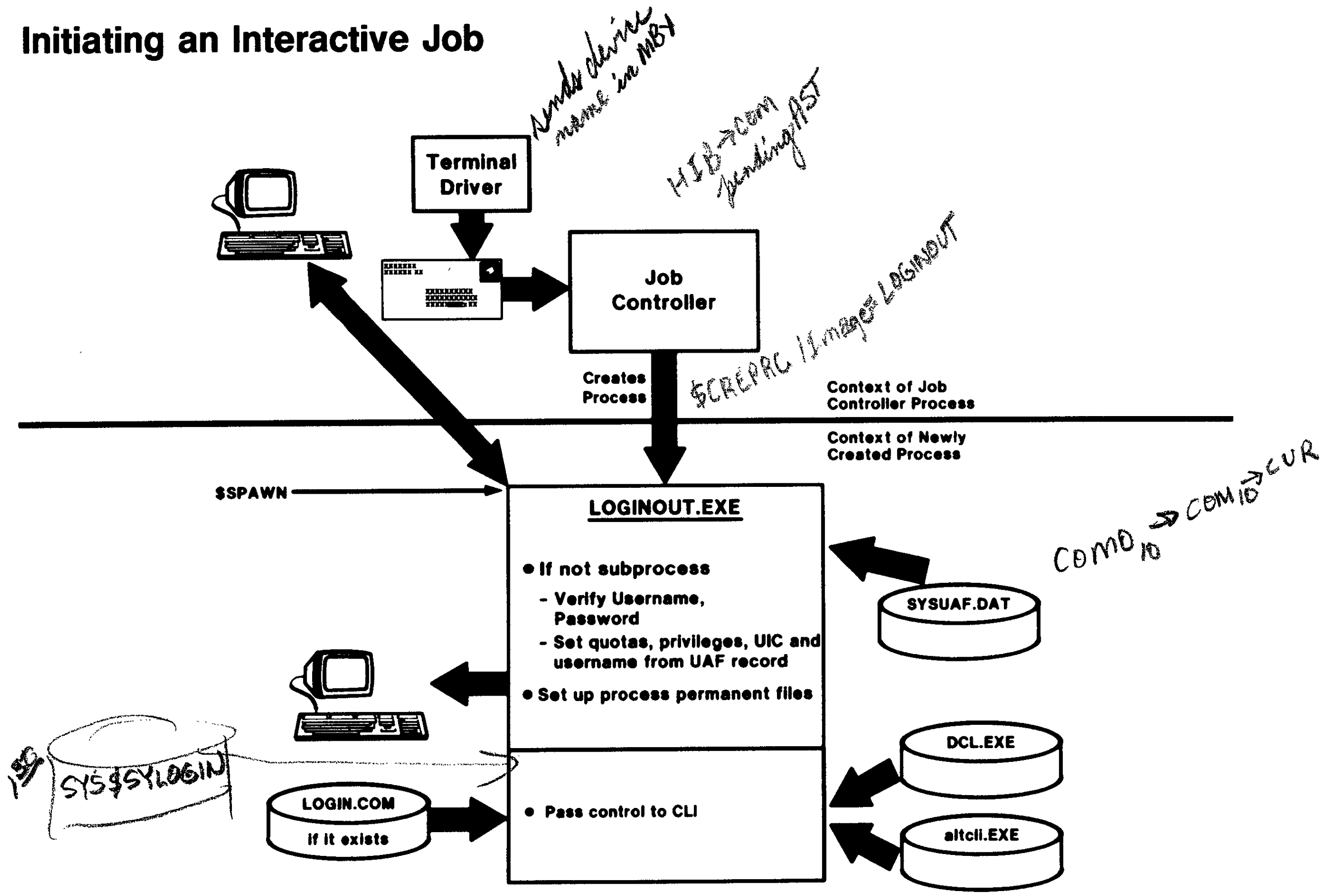


Figure 7 Initiating an Interactive Job

- Initiated by unsolicited input at a free terminal
  - Job controller notified by driver
  - Creates process with user name equal to terminal name
- LOGINOUT runs
- DCL mapped (or alternate CLI)
- SPAWN creates an interactive or non-interactive subprocess (no need to verify user name, etc.)

Initiating Job Using $SUBMIT

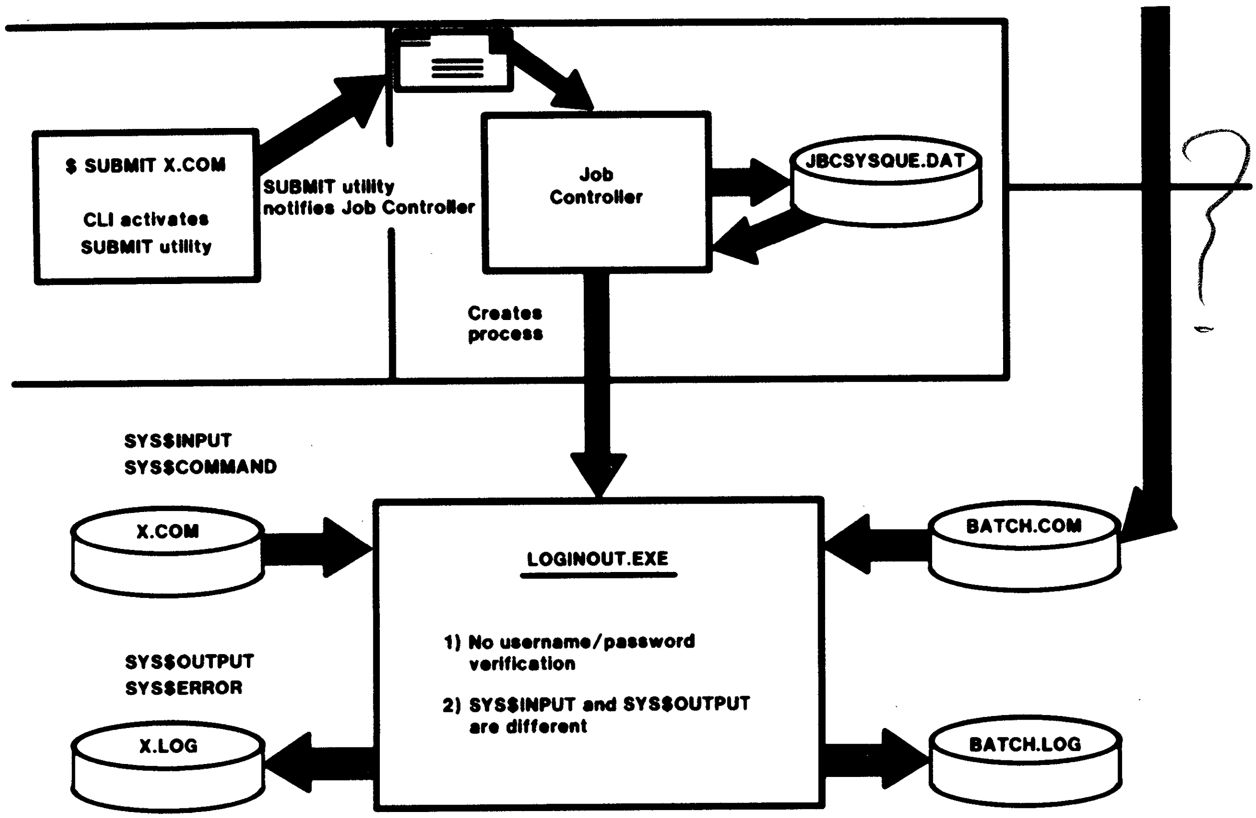


Figure 8 Initiating Job Using $SUBMIT

- o Similar to interactive process, except
  - Job controller notified by DCL ($SUBMIT)
  - User already validated
  - Files are assigned:

    SYS$INPUT to batch stream
    SYS$OUTPUT to log file

Initiating Job Through Card Reader

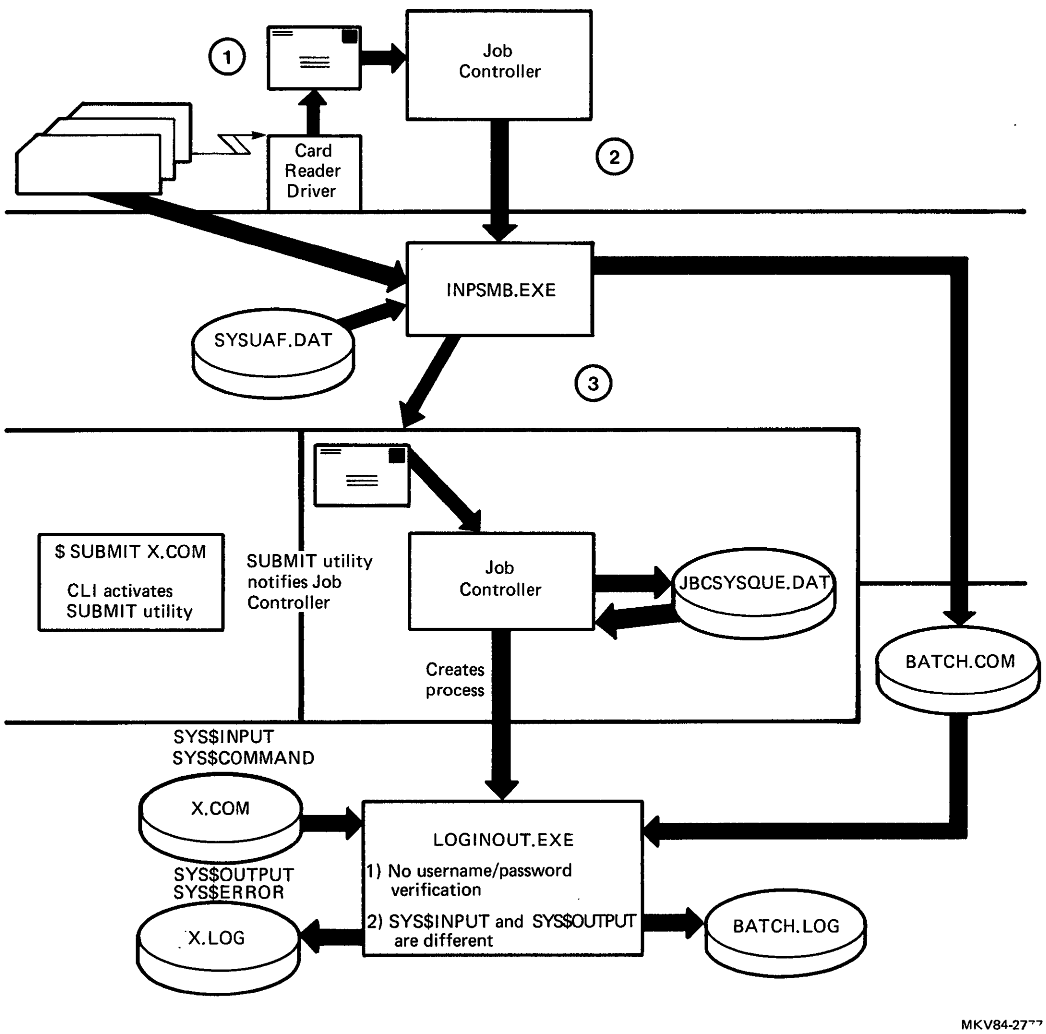


Figure 9 Initiating Job Through Card Reader

1. Job controller notified by card reader driver

2. Job controller creates input symbiont process

   - User authorization
   - Read cards into command file
   - Submit as batch job

3. Same as for $SUBMIT

PROCESS DELETION

- After image runs and exits, process deleted
  - Unless running with a CLI
- All traces of process removed from system
- All system resources returned
- Accounting information passed to job controller
- For subprocess, all quotas and limits returned to creator
- Creator notified of deletion

Process Deletion Sequence

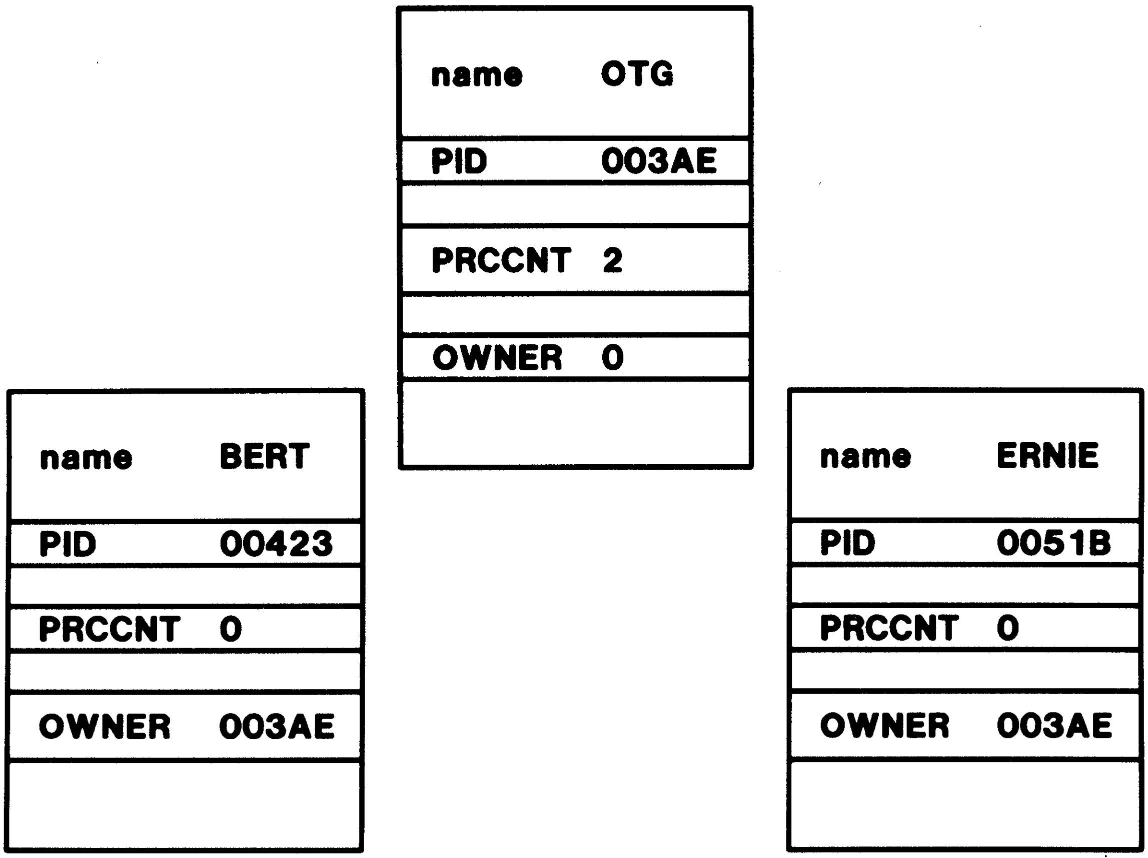


Figure 10 Process Deletion

- o Deleted by kernel AST while CURRENT
- o Sequence
  - Delete any subprocesses
  - Accounting information to job controller
  - Call SYS$RUNDOWN
  - Delete P1 space
  - Free PCBVEC and SWAP slots, page file space
  - Decrement counts

    Balance set
    Total processes

  - Jump to SCH$SCHED

SUMMARY

Table 7 Steps in Process Creation and Deletion

Action	Code
Creating process	SYS$CREPRC
Inswap a process	SWAPPER
Process startup	PROCSTRT
Process deletion	SYS$DELPRC

Table 8 SYSGEN Parameters Relating to Process Creation and Deletion

Function	Parameter
Maximum number of processes allowed on the system	MAXPROCESSCNT
System default values for some process limits and quotas	PQL_Dxxx
System minimum values for some process limits and quotas	PQL_Mxxx

System Initialization and Shutdown

INTRODUCTION

The study of the initialization of a VAX/VMS system provides a convenient summary of many of the topics previously discussed in this course. It is during initialization that the structures, mechanisms, and other features of the VMS environment are established.

Each component of the initialization sequence is discussed from turning on the power to the final start-up command procedure and the enabling of logins. Included is an explanation of:

- Why each component executes in its particular environment
- Why it executes at its position in the overall initialization sequence.

Hardware differences between VAX systems, especially the components of the console subsystem, have an effect on the initial stages of system initialization. The basic configurations of the VAX-11/730, VAX-11/750 and VAX-11/780 are described, highlighting the effects of the differences on the initialization sequence.

In addition, some time is spent discussing the shutdown and recovery sequences involved in power failure and bugcheck.

OBJECTIVES

1. Describe, in general terms, the sequence of operations involved in:
   - Initial bootstrap
   - Powerfail and recovery
   - Bugcheck and reinitialization
2. Describe the differences between console subsystems of the VAX family systems, and the effects on system initialization.
3. Discuss the effects of altering SYSGEN parameters relating to system initialization.

RESOURCES

Reading

1. VAX/VMS Internals and Data Structures, chapters on error handling, bootstrap procedures, operating system initialization, and powerfail recovery.

Source Modules

Facility Name	Module Name
BOOTS	SYSBOOT, SYSGEN
	VMB
SYS	INIT
	SYSPARAM
	POWERFAIL
	BUGCHECK, BUGCHKMSG
SYSINI	SYSINIT
Hardware Microfiche	CONSOLE.SYS
	Memory ROM program

TOPICS

I. Initialization

A. System initialization sequence

B. Functions of initialization programs

C. How memory is structured and loaded

D. Start-up command procedures

E. SYSBOOT, SYSGEN

F. VAX-11/780, VAX-11/750, and VAX-11/730 hardware differences and how they affect initialization

II. Shutdown and Restart

A. Front panel switches

B. Shutdown procedures and their functions

C. Autorestart sequence

D. Powerfail recovery

VAX-11/780, 11/750, 11/730 CONSOLE DIFFERENCES

780 and 730

- Contain a console microprocessor

  780 - LSI-11

  730 - 8085

- Boot/restart information available on console media

  780 - floppy
  730 - TU58

750

- No console microprocessor
- Boot/restart information in ROM (normally) or on disk

SYSTEM INITIALIZATION

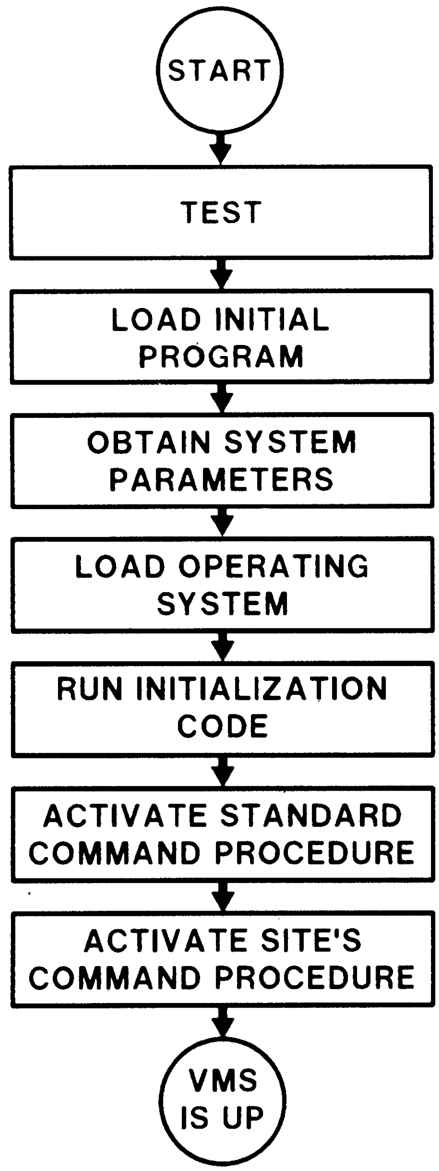


Figure 1 System Initialization

SYSTEM INITIALIZATION SEQUENCE

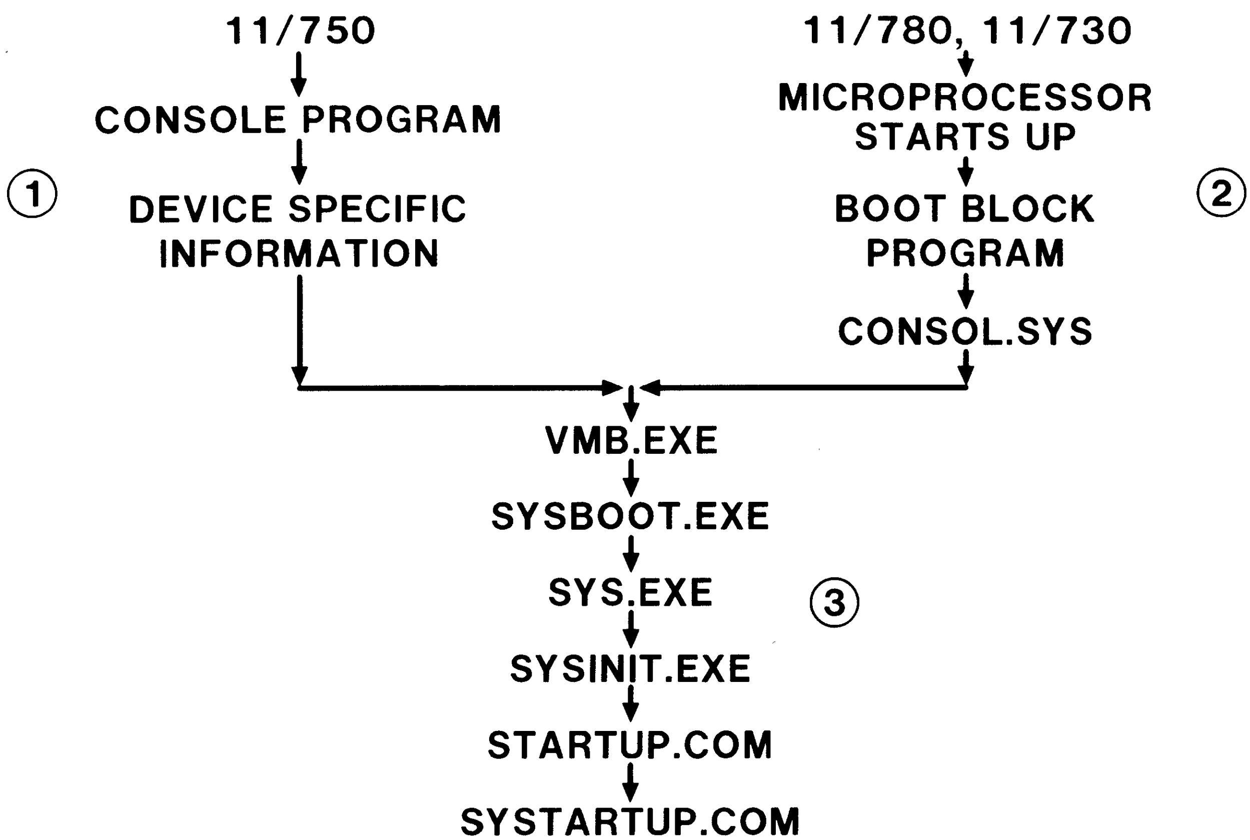


Figure 2 System Initialization Sequence

1. Bootstrap computer using ROMs in CPU
2. Bootstrap computer using LSI-11 (780) or 8085 (730)
3. Finish system initialization
   - Finish preparing system
   - Load operating system
   - Run operating system initialization code
   - Activate VMS standard and site-specific DCL procedures

INITIALIZATION PROGRAMS

Table 1 Initialization Programs

Program	Function	Environment
CONSOLE.SYS (CONSOLE.EXE on 730)	Loads VAX writable diagnostic control store Acts as monitor for console terminal commands On boot command loads, passes control to VMB.EXE	LSI (780) 8085 (730) CPU (750)
VMB.EXE	Sizes and tests physical memory, discovers external adapters Sets up primitive SCB Locates, loads, and passes control to SYSBOOT.EXE	VAX memory Physical address
SYSBOOT.EXE	Locates and loads SYS.EXE Loads SYSBOOT parameters Opens and stores location of dump file Sets up full SCB Sizes system space, sets up system page table Maps nonpaged pool into high end of physical memory Loads terminal driver and system disk driver Sets up P0 page table Passes control to INIT in SYS.EXE	VAX memory Physical address
INIT (in SYS.EXE)	Turns on memory management Maps and initializes the I/O adapter Maps paged pool Initializes several scheduling and memory management data structures Invokes SCHED.MAR	VAX memory Physcial address/ Virtual address
SYSINIT	Opens and stores locations of page files and swap files Maps RMS and system message file as system sections Mounts system disk	Process

Table 1 Initialization Programs (Cont)

Program	Function	Environment
STARTUP.COM	Creates several system logical names Creates job controller, error log formatter, OPCOM processes Invokes INSTALL Invokes SYSGEN for autoconfigure Invokes SYSTARTUP.COM	Process
SYSTARTUP.COM	Site-specific, such as: • Create logical names • Load user-written device drivers • Install privileged and shareable images • Set up queues and terminal characteristics	Process

PHYSICAL MEMORY DURING INITIALIZATION

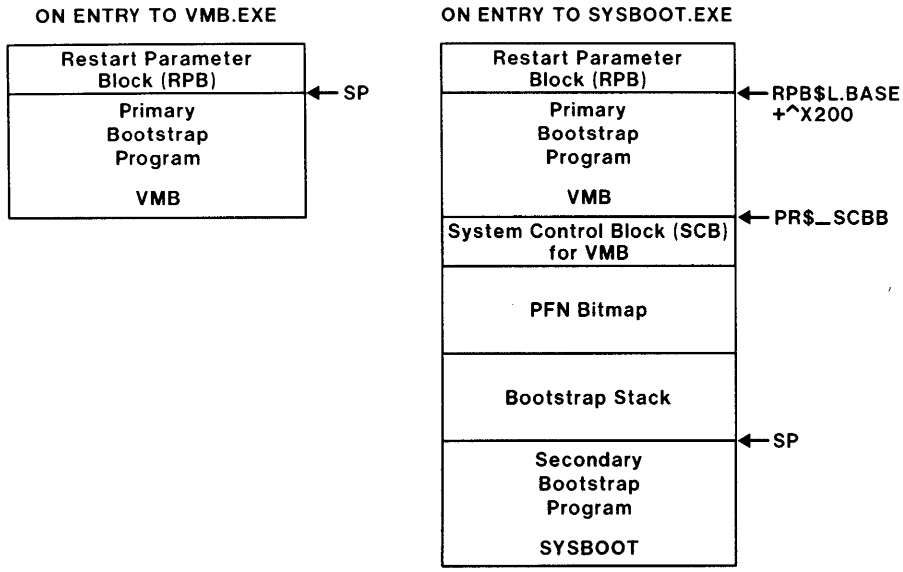


Figure 3 Physical Memory During Initialization

- Console or ROM programs have located 64K bytes of good contiguous memory.

- On entry to VMB.EXE

  Console program has loaded VMB into the known good memory, leaving 512 bytes for the Restart Parameter Block.

- On entry to SYSBOOT.EXE

  VMB has loaded

  - Restart Parameter Block with values from RØ-R5

  - System Control Block with vectors pointing to one routine

  - PFN Bitmap with map of error-free pages in physical memory

  - SYSBOOT.EXE

  VMB has also allocated Bootstrap Stack, used by VMB and SYSBOOT.

PHYSICAL MEMORY LAYOUT AFTER SYSBOOT ENDS

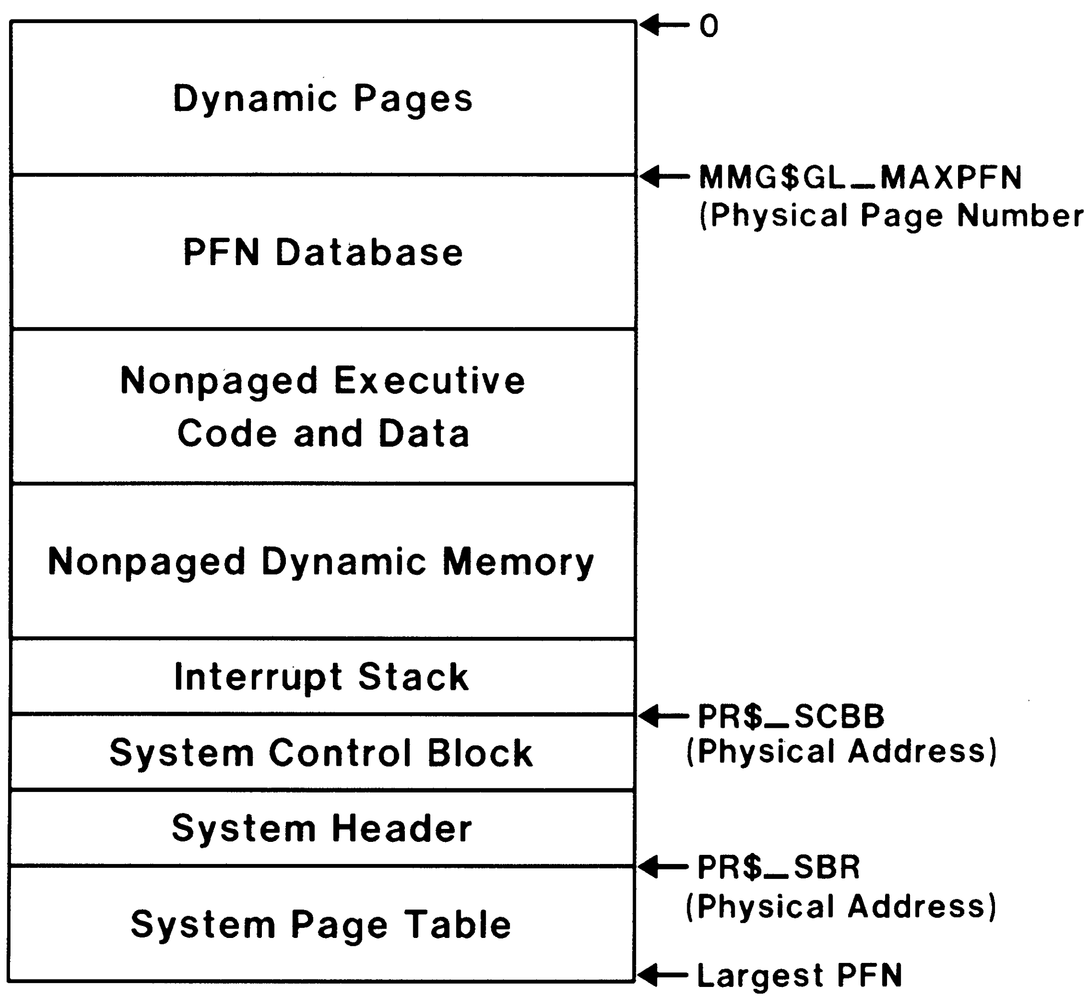


Figure 4 Physical Memory After SYSBOOT

SYSBOOT has

- Sized the pieces of memory shown above
- Filled in the SCB and part of the system header
- Mapped and read in SYS.EXE (Executive code)

TURNING ON MEMORY MANAGEMENT

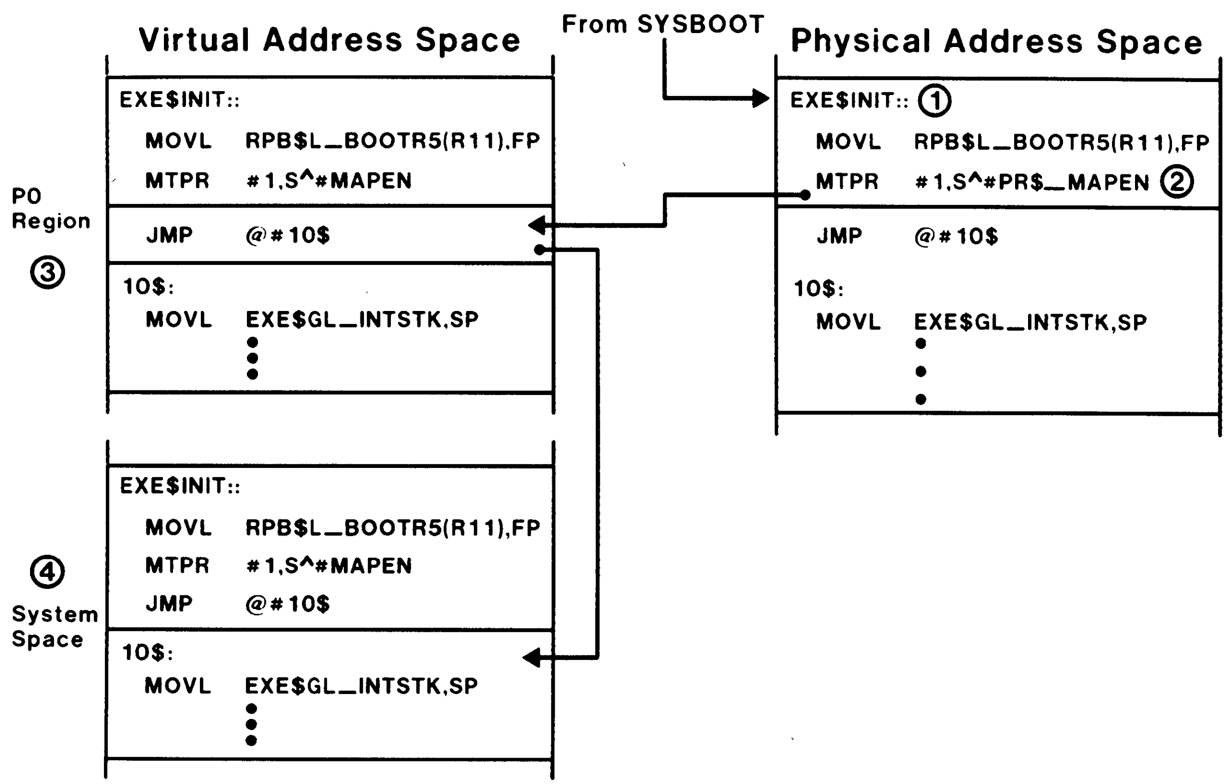


Figure 5 Turning on Memory Management

Turning on Memory Management

- Done by INIT in SYS.EXE

- Physical to virtual transition:

  1.
     - All address references treated as physical addresses
     - INIT page table entries set up so PØ virtual address
       equals physical address
     - SØ and PØ page table entries for INIT contain same PFNs

  2. Writing a 1 to processor register MAPEN causes following address references to be treated as virtual addresses

  3. Next instruction is found in PØ space

  4. When INIT was linked, base was in SØ space, so JMP @#1Ø$ causes jump to address in SØ space

SYSINIT

- Created by swapper as part of one-time initialization routine
- Selected from COM queue after SWAPPER goes into normal HIB
- Major functions:
  - Opens and records locations of page and swap files
  - Maps RMS and system message files
  - Creates XQP global section
  - Mounts system disk
  - Creates start-up process

START-UP

Start-Up Process

- Runs as final part of initialization
- Runs using DCL command procedures
  - STARTUP.COM
  - SYSTARTUP.COM

STARTUP.COM

- Assigns logical names
- Installs VMS images
- Creates system processes
  - ERRFMT
  - JOB_CONTROL
  - OPCOM
- Autoconfigures all devices

SYSTARTUP.COM

- Mounts volumes other than the system disk
- Assigns site-specific logical names
- Sets up site-specific
  - Terminal characteristics
  - Print and batch queues
- Installs site-specific images
- Starts DECnet
- Loads user-written device drivers

SYSBOOT AND SYSTEM PARAMETERS

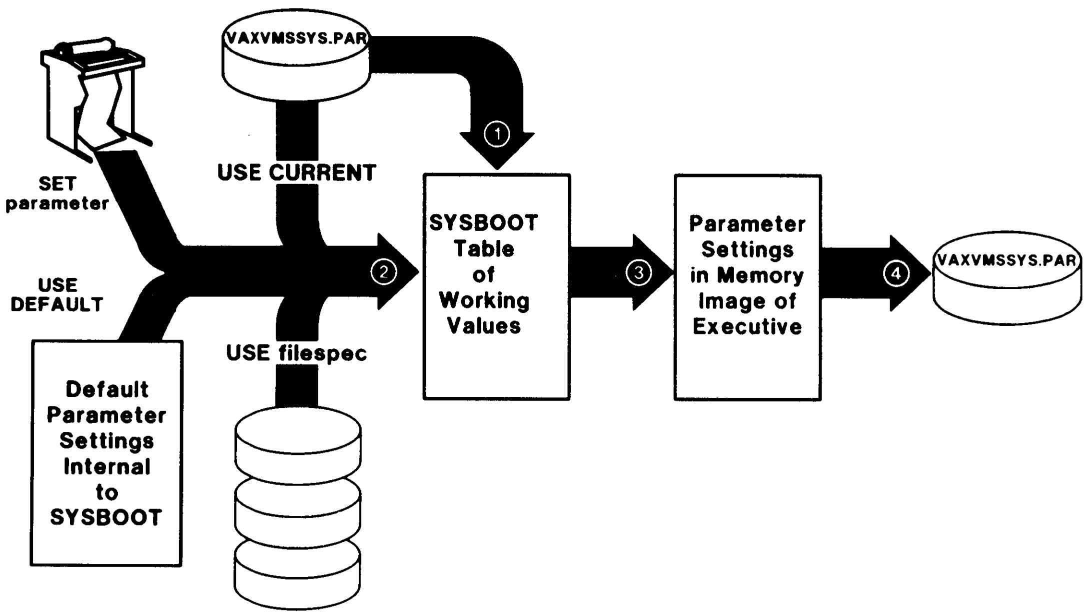


Figure 6 SYSBOOT and System Parameters

SYSBOOT executes as part of system initialization.

1. Automatically brings in current parameters
2. Allows changes if conversational boot requested
   - Valid commands are USE, SET, CONTINUE, EXIT
   - Can alter all parameters used in present system
   - Cannot create alternate parameter files
3. Writes parameters to copy of SYS.EXE in memory
4. Later in initialization sequence, parameter values are copied to VAXVMSSYS.PAR for subsequent boots

SYSGEN AND SYSTEM PARAMETERS

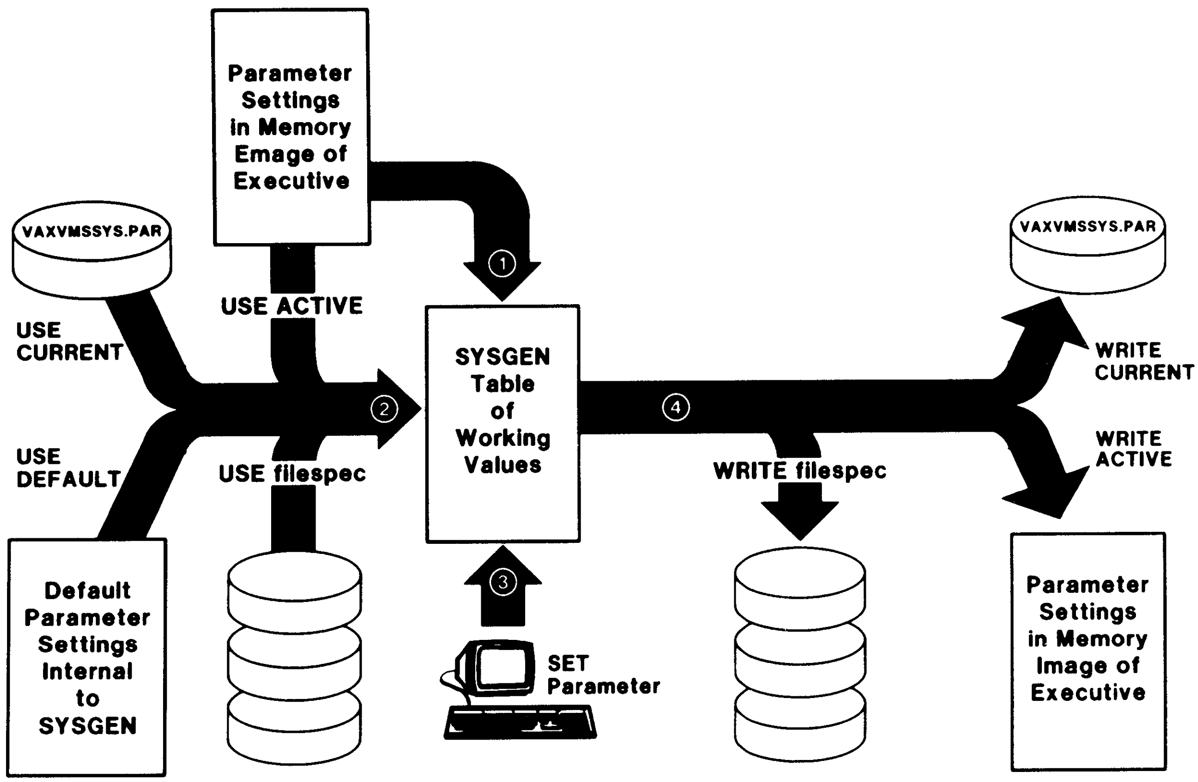


Figure 7 SYSGEN and System Parameters

SYSGEN runs as an editor-like utility under VMS

1. SYSGEN copies active system parameters into its buffer

2. Can replace all values with **current**, **default** or **active** values, or with values in an alternate file

3. Can alter individual parameters in SYSGEN buffer

4. Use WRITE command to record new values:

   - Can create alternate parameter files
   - Can alter dynamic parameters on present system
   - Can alter parameters used on **next** system boot

VAX-11/780 PROCESSOR

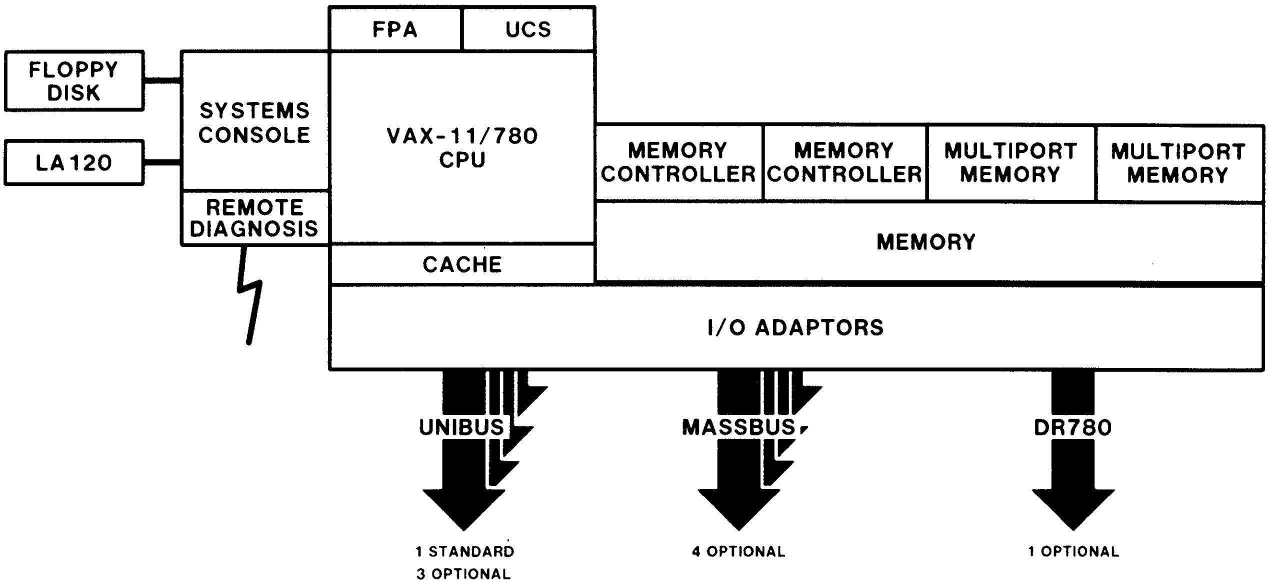


Figure 8 VAX-11/780 Processor

- Program on ROM causes CONSOLE.SYS to be loaded from floppy into LSI-11 memory
- CONSOLE.SYS runs on LSI-11
  - Loads diagnostic control store
  - Causes ROM in memory controller to find 64K good bytes
  - Loads VMB.EXE from floppy disk to VAX memory

VAX-11/750 PROCESSOR

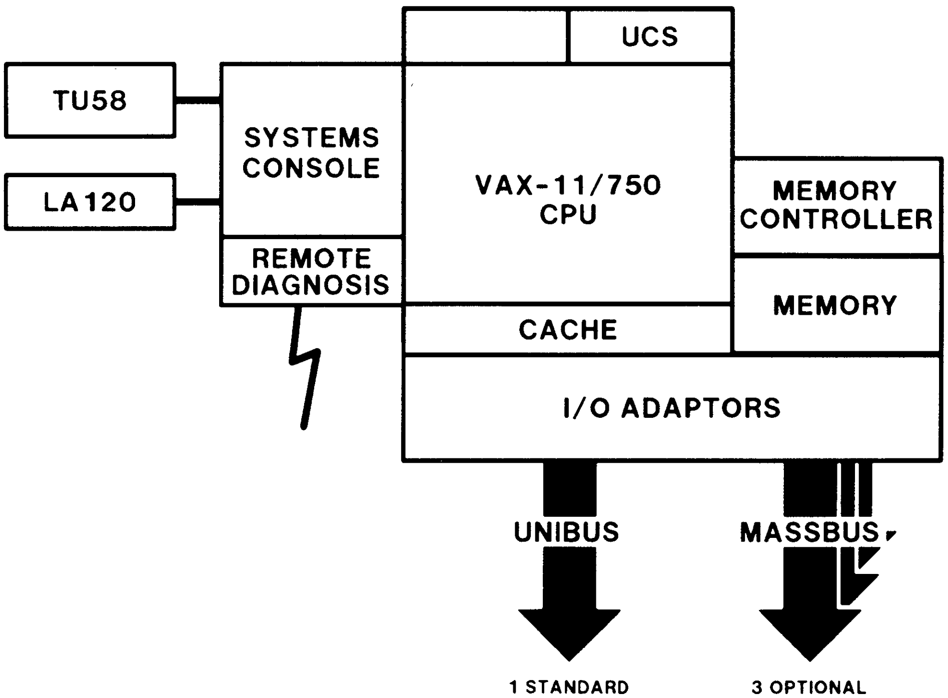


Figure 9 VAX-11/750 Processor

- Console program stored in ROM with CPU
  - Locates 64K good bytes
  - Passes control to device ROM
- Device ROM
  - Reads boot block from device
- Boot block program
  - Loads VMB.EXE from specified system device

VAX-11/730 PROCESSOR

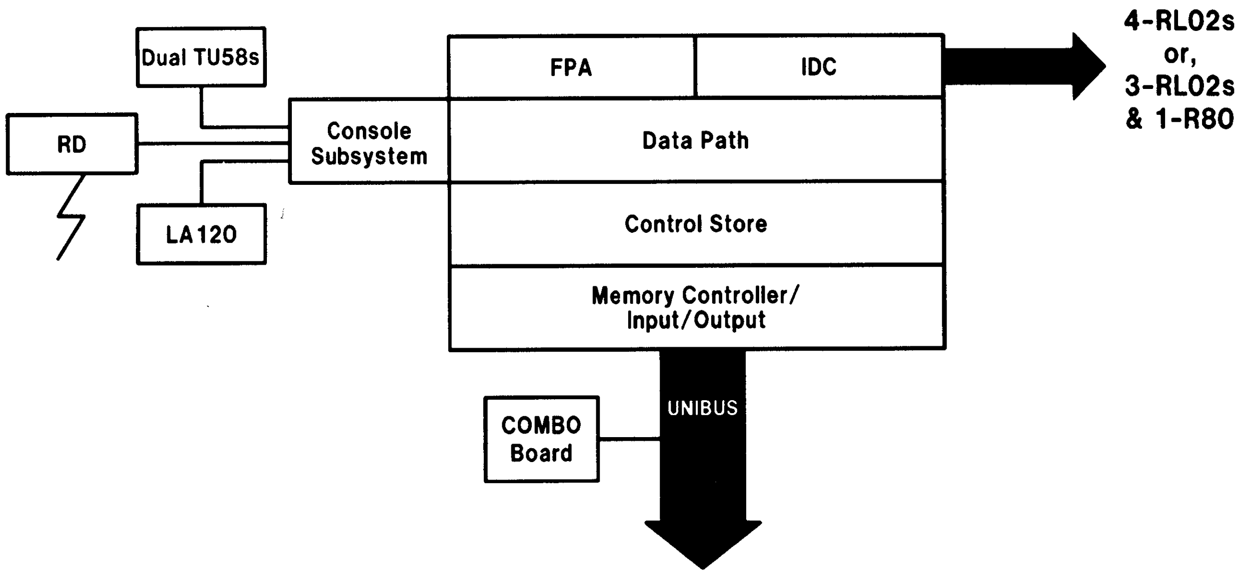
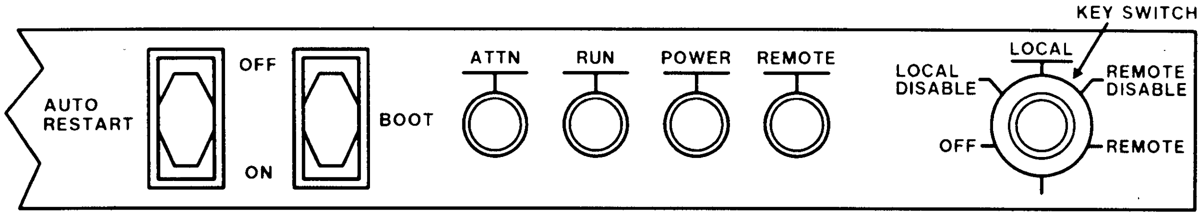


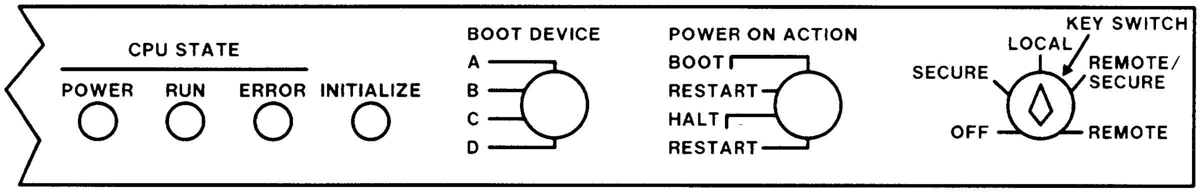
Figure 10 VAX-11/730 Processor

- Program on ROM causes CONSOLE.EXE to be loaded from TU58 into 8085 memory
- CONSOLE.EXE runs on 8085
  - Loads microcode into CPU from TU58
  - Executes DEFBOO - loads registers of CPU, finds 64K good bytes
  - Loads VMB.EXE from TU58

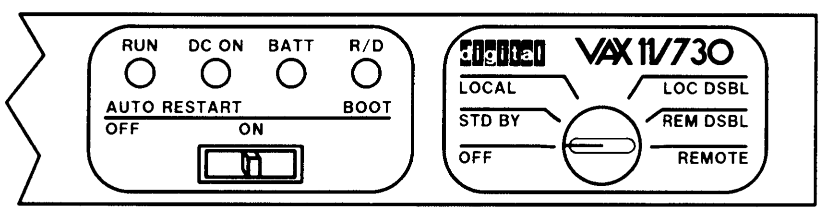
VAX FRONT PANELS



VAX-11/780 Panel



VAX-11/750 Panel



VAX-11/730 Panel

Figure 11 VAX Front Panels

Table 2 Switches on the VAX-11/780, /730, /750

11/780	11/750	11/730	Effects on Console Terminal and System
OFF	OFF	STANDBY	Power partially off
LOCAL/DISABLE	SECURE	LOCAL/DISABLE	Local terminal-program I/O mode only. Remote disabled.
LOCAL	LOCAL	LOCAL	Local terminal-program I/O mode and console I/O mode. Remote disabled.
REMOTE	REMOTE	REMOTE	Local terminal disabled. Remote-console I/O mode and program I/O mode.
REMOTE/DISABLE	REMOTE/SECURE	REMOTE/DISABLE	Local terminal disabled. Remote-program I/O mode only.
—	—	OFF	Power completely off

SHUTDOWN OPERATIONS

Table 3 Shutdown Operations

Action	Operation	
Clean shutdown	$ @SYS$SYSROOT:[SYSEXE]SHUTDOWN	
Quick shutdown	$ RUN SYS$SYSTEM:OPCCRASH	
Forced crash	Control/P >>>@CRASH >>>E P >>>E/G F >>>E/I 0 >>>E/I 1 >>>E/I 2 >>>E/I 3 >>>E/I 4 >>>D/G F FFFFFFFF >>>D P 001F0000 >>>C	(on OPA0:) (780/730 only) (750 only)
Halt system	Control/P >>>H	(on OPA0:) (780/730 only)

SHUTDOWN PROCEDURES

Table 4 Shutdown Procedures

Procedure	Function
SHUTDOWN.COM	- Warns users of shutdown - Stops queues - Removes installed images - Stops processes - Dismounts disks - Runs OPCCRASH
OPCCRASH	- Marks system disk for dismount (to force cache flushing) - Flushes modified page list - Requests "operator" BUGCHECK
CRASH.CMD	- Halts CPU - Examines PSL and all SPs - Deposits -1 in PC 1FØØØ in PSL - Continues

AUTORESTARTING THE SYSTEM

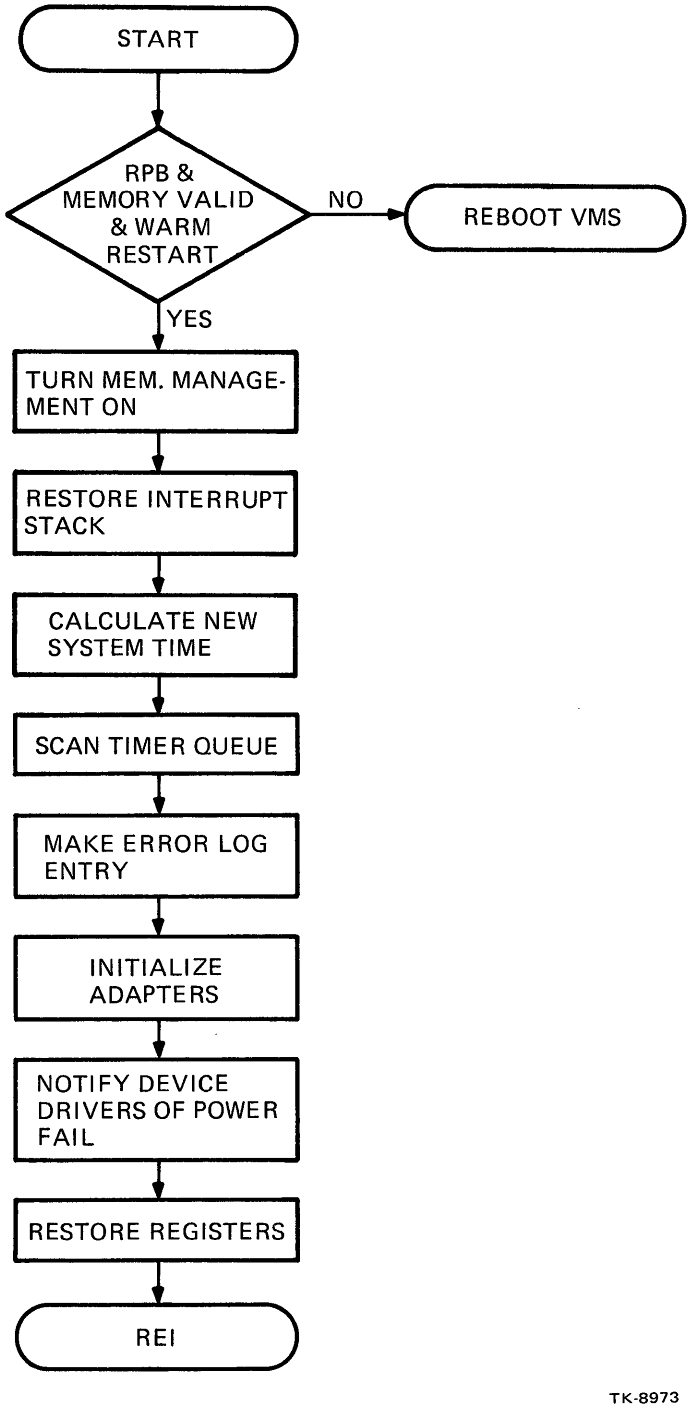


Figure 12 Autorestarting the System

REQUIREMENTS FOR RECOVERY AFTER POWER-FAIL

- Battery backup
- Memory valid (battery not run down)
- RPB and memory valid and warm restart flag cleared
- VAX-11/780 - Autorestart On
  - RESTART.CMD on console floppy
  - RESTART.CMD contains right TR number for system disk adapter
- VAX-11/750 - Power action SW on 'Restart/Boot' or 'Restart/Halt'
- VAX-11/730 - Enable restart

SUMMARY

- Initialization
  - System initialization sequence
  - Functions of initialization programs
  - How memory is structured and loaded
  - Start-up command procedures
  - SYSBOOT, SYSGEN
  - VAX-11/780, VAX-11/750, and VAX-11/730 hardware differences and how they affect initialization

- Shutdown and Restart
  - Front panel switches
  - Shutdown procedures and their functions
  - Autorestart sequence
  - Powerfail recovery

Using The Linker

Introduction

The linker binds object modules, together with any other necessary information, into executable and shareable images. Most linker operations are transparent to the user, but a basic understanding of these operations allows a user to write programs that execute more efficiently.

An optional output file produced by the linker, called a linker map, can be particularly helpful in locating and debugging run-time errors.

This module provides an overview of the linker's processing of input files, along with the qualifiers available with the LINK command. These qualifiers and options control the execution characteristics of the images produced.

Objectives

1. To build images that execute efficiently, a programmer must be able to:

   - Describe the manner in which the linker arranges the contents of object modules to form images.
   - Use the qualifiers and options available with the LINK command.

2. To locate certain types of run-time errors, a programmer must be able to produce and read a linker map.

Resources

1. *VAX/VMS Linker Utility Reference Manual*
2. *VAX/VMS DCL Dictionary*

1 Linking Object Modules to Form an Image

The linker accepts object modules, shareable images, and libraries as input, and creates executable and shareable images. When an image is executed, the image activator uses information placed in the image file by the linker to map the image into the virtual address space of a process.

1.1 Using the LINK Command

The *VAX/VMS DCL Dictionary* describes the LINK command and its command and file qualifiers. The LINK command has the following format:

$ LINK file-spec [,file-spec...]

The default file type for input object files is .OBJ. Input files that are not object files (shareable images), are indicated by a file qualifier, and have different default file types.

Tables 1 and 2 list some of the most frequently used qualifiers. The default qualifiers are labeled with a (D).

Table 1 Commonly Used Qualifiers for the LINK Command

Operation	Qualifier
Create an executable image	/EXECUTABLE (D)
Include a debugging module	/DEBUG
Create a full linker map	/FULL and /MAP
Create a shareable image	/SHAREABLE
Search the default system libraries to resolve undefined references	/SYSLIB (D)

Table 2 File Qualifiers Commonly Used with the LINK Command

Operation	Qualifier
Include one or more modules from a library	/INCLUDE
Specify that the input file is a library	/LIBRARY
Specify that the input file is an options file	/OPTIONS

1.2 Program Sections

The VAX-11 MACRO assembler and high-level language compilers translate source code into object code. Different parts of a source file have different properties (for example, code is executable; data is not). Table 3 lists some of the properties that might describe different segments of source code. In creating an object file, the compiler (or assembler) divides the code into **program sections** (PSECTs). Each PSECT contains code with similar properties; the properties of a particular PSECT are called its PSECT attributes.

Table 3 PSECT Attributes

WRT	Writeable	NOWRT	Not Writeable
RD	Readable	NORD	Not Readable
EXE	Executable	NOEXE	Not Executable
PIC	Position-Independent	NOPIC	Not Position-Independent
LCL	Local	GBL	Global
CON	Concatenated	OVR	Overlaid
SHR	Potentially Shareable	NOSHR	Not Shareable
VEC	Protected (vector)	NOVEC	Nonprotected (vector)

Figure 1 shows the organization of a sample source file into various program sections. All executable code is gathered into a PSECT named CODE, which has the attributes EXE and NOWRT.

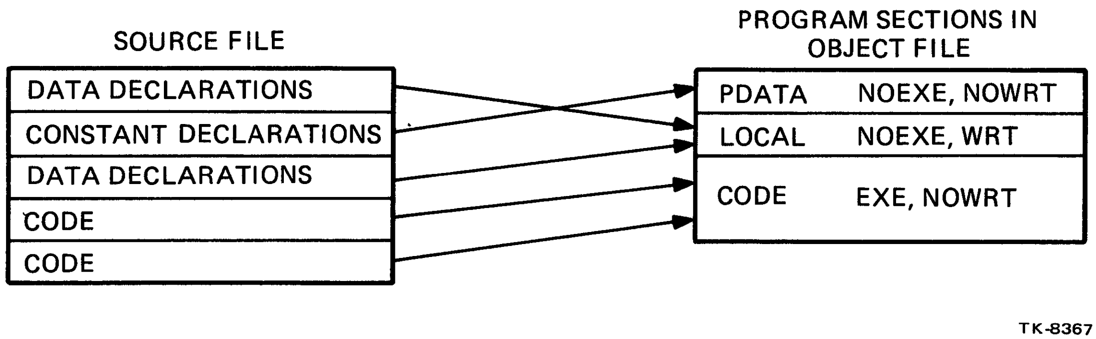


Figure 1 Organization of Source Files into Program Sections

MACRO programmers can assign attributes to different sections of a program. PSECT attributes for high-level language programs, however, are assigned by the compiler. High-level language programmers can determine the PSECT attributes given to a program by examining the listing file produced when the program is compiled using both the /MACHINE_CODE and /LIST qualifiers. Any programmer can alter the attributes of a PSECT using a linker options file, discussed later in the module.

1.3 Linker Clusters

The linker must first collect all files specified as input for an image. As the linker collects the input files, it organizes them into **clusters**, and stores the clusters in a buffer. A cluster is the unit in which the linker handles your program. The input is processed and written to the image file, cluster-by-cluster.

It is sometimes beneficial to have certain segments of code close to each other in an executable image. Since the placement of input modules in clusters defines the order of the code in an image, it is useful to know how the linker clusters input modules.

An executable image is mapped into the virtual address space of a process at run time, but may not fit into the physical memory allocated to the process (the process working set). In this case, segments of the program are paged into the working set as needed. If related segments of the program are close to each other in an executable image, they will be paged into the working set together, which can improve program performance. You can ensure that related segments of code are near each other in an executable image by controlling their placement in clusters.

By default, the linker places all input object modules in a default cluster. Even if the object modules are stored in different files, they are placed in the same default cluster. In addition, a separate cluster is created for each shareable image referenced by the program, as in Figure 2. The code for a shareable image is not copied into the image file (to conserve disk space), rather, a descriptor for the shareable image is included in the executable image file.

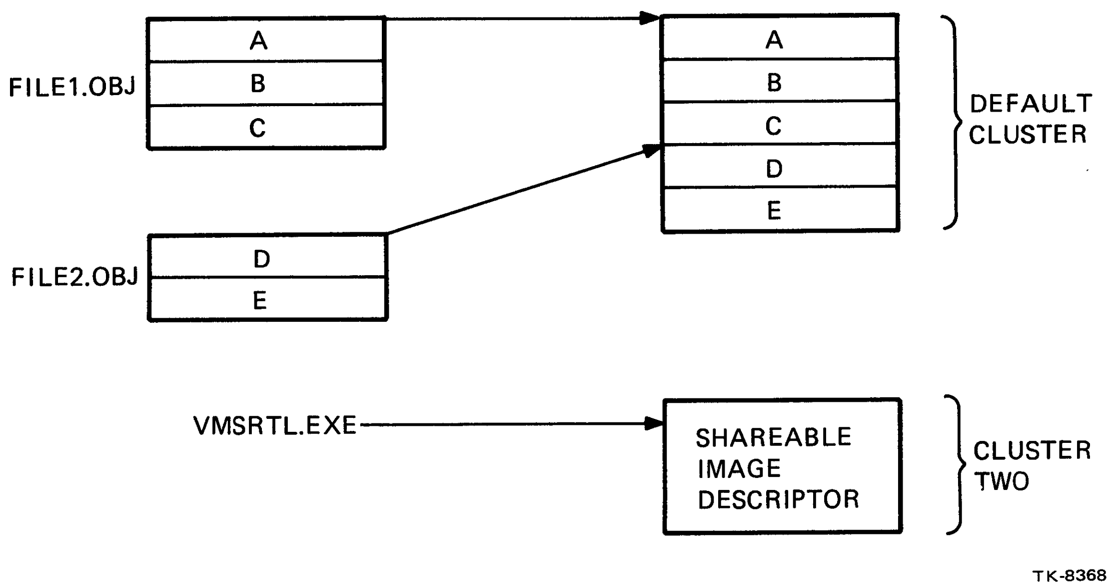


Figure 2 Organization of Input Files into Clusters

Options on the LINK command allow you to control placement of program sections within clusters. (Linker options are discussed in more detail later in the module.) Take, for example, a transaction processing application that collects and processes data input to a terminal. One set of routines displays three different forms on the terminal, and another set collects and processes the data input for each form. Because the screen formatting routines are similar to each other, they are stored in the same subdirectory. Similarly, the processing routines are stored together in another subdirectory, as in Figure 3. To place the form and processing routines for each screen next to each other in the final image, you might specify the files on the LINK command in the following order:

```
$ LINK MAIN, [.FORMS]FORM1, [.PROCESSRS]PROCESS1, -
             [.FORMS]FORM2, [.PROCESSRS]PROCESS2, -
             [.FORMS]FORM3, [.PROCESSRS]PROCESS3
```

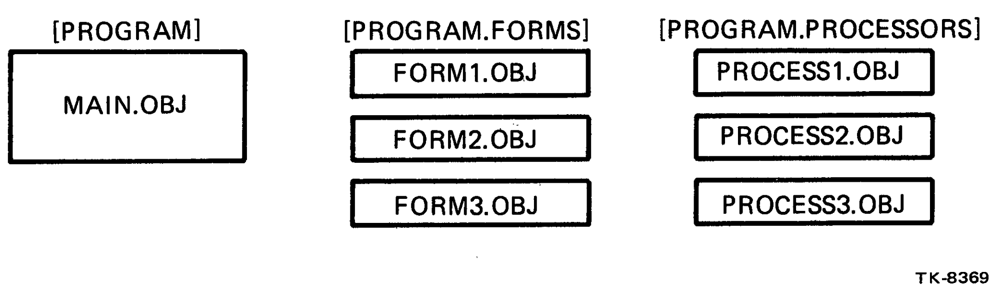


Figure 3 Routines for Transaction Processing Application

The organization of input files into clusters, however, is not defined by the order of the files on the LINK command. Rather, the linker gathers similar PSECTS from the input files, so the routines are ordered in the final image, as shown in section A of Figure 4. To ensure that the related routines are near each other in the final image (as shown in section B of Figure 4), use the CLUSTER option of the linker. This is discussed later in this module.

Small programs that fit into the working set of a process need not be too concerned with the location of related code in an image. For large programs, the advantages of clustering are three-fold:

- Faster image activation
- Improved program performance (less paging I/O)
- Improved system performance (decreased paging activity)

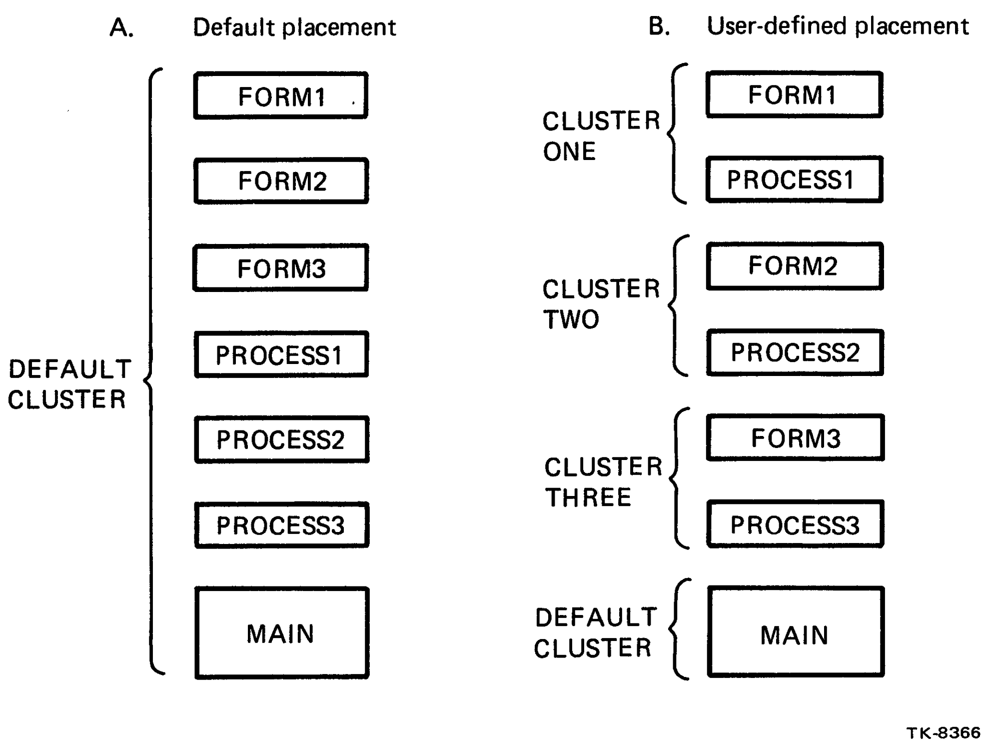


Figure 4 Placement of Program Sections in Clusters

LEARNING ACTIVITY

1. (OPTIONAL) See the *VAX/VMS Linker Utility Reference Manual* for a more complete description of the way the linker organizes input into clusters.

1.4 Image Sections

Once the linker has located all modules needed to create an image, and has organized them into clusters, the modules are processed on a cluster-by-cluster basis to form the final image. This processing has three parts:

1. Organize the PSECTS into image sections.
2. Assign virtual addresses to the image sections.
3. Write image sections to the image file.

The linker must organize your image into image sections because that is the unit in which the image activator handles your program. Your image is mapped to your virtual address space an image section at a time.

The following paragraphs describe the creation of image sections by the linker. The allocation of virtual memory is discussed in the next section.

For each cluster, the linker gathers PSECTs with similar attributes and organizes them into **image sections**. When creating image sections, the linker only looks at certain relevant PSECT attributes. For all images, the WRT/NOWRT, EXE/NOEXE, and VEC/NOVEC attributes are considered. When creating shareable images, the PIC/NOPIC and SHR/NOSHR attributes are also considered.

Figure 5 shows the creation of image sections for a typical default cluster. This default cluster contains object modules from three separate input files. All PSECTS with both the NOEXE and NOWRT attributes are collected into the first image section. The rest of the image sections are created similarly.

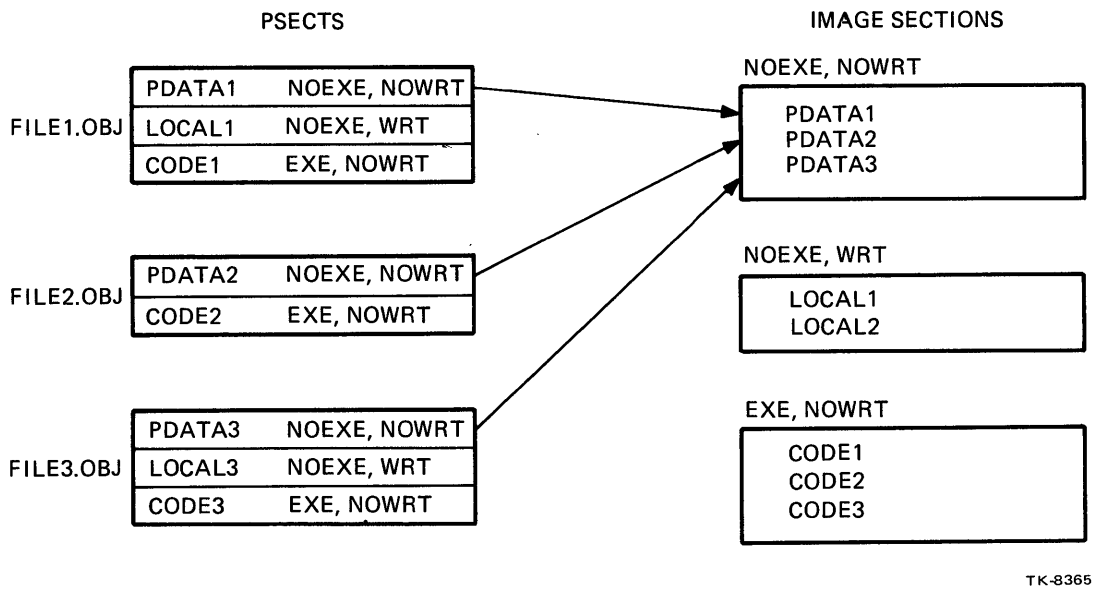


Figure 5 Organization of PSECTs into Image Sections

When the linker creates image sections:

- PSECTs are alphabetized by name within each image section.
- Image sections are organized within a cluster in a predefined order (see the *VAX/VMS Linker Utility Reference Manual*).

2 Mapping an Image to the Virtual Address Space of a Process

The linker and the image activator work together to assign virtual addresses to executable code. The code is mapped to these addresses in the virtual address space of a process at run time.

2.1 Linker Assigns Virtual Addresses

On a cluster by cluster basis, the linker assigns virtual addresses to the image sections. The image file is mapped to these addresses in process virtual address space when the RUN command is issued. An executable image file is always mapped to the same virtual addresses each time it is run.

In most cases, virtual addresses are assigned to shareable images at run time, rather than when they are created by the linker. This avoids addressing conflicts. If, for example, virtual addresses are assigned at creation, then two shareable images could both be assigned to start at address 200. They could not both be included in the same program. To avoid such addressing conflicts, the image activator assigns virtual addresses to position-independent shareable images at run time.

Sometimes it is necessary to include data definitions which contain virtual addresses in a shareable image (for example, a character string descriptor). An address must be assigned to this code for it to link successfully. The correct address will not be known until run time, when addresses are assigned to the rest of the image. To satisfy the need for an address and preserve the position independence of the shareable image, the linker assigns an offset to the code. The offset is translated to the correct address at run time by the image activator.

The linker performs this special action for:

- .ADDRESS and .ASCID directives in a shareable image.
- General addressing mode (G^) references to a location in a shareable image.

General addressing mode and .ADDRESS directives are used in MACRO; high-level language compilers generate the object language equivalent. Some knowledge of MACRO is helpful in understanding this discussion, but the concept relates to all languages.

To illustrate handling a general addressing mode reference to a routine in a shareable image, consider a call to MTH$SQRT. This mathematical Run-time Library routine is part of the shareable image MTHRTL.EXE. A program written in a high-level language references the MTH$SQRT routine as follows:

CALL MTH$SQRT(number)

The compiler translates this to:

CALLG ARGLIST, G^MTH$SQRT

which is how the call appears in a MACRO program. (Note that some compilers may translate this call to a CALLS instead.) When the program is linked, the linker calculates the location of MTH$SQRT in MTHRTL, and stores the offset in a symbol named SQRT.

CALLG ARGLIST, @L^SQRT
SQRT: .LONG X

At run time, virtual address space is assigned to MTHRTL, and the image activator can translate the offset to a true virtual address:

SQRT + (MTHRTL-base-address) = address for routine

The linker handles .ADDRESS and .ASCID directives in an object module in much the same way as G^references. These directives are often used by MACRO programmers. The equivalent object language commands are generated by high-level language compilers when building argument lists with arguments passed by reference or descriptor.

The linker resolves the .ADDRESS reference to an offset, rather than an address. The offset represents the location of the target within the shareable image. After assigning virtual addresses to the shareable image, the image activator calculates the correct virtual address of the instruction:

Offset + SHIMG-base-address = address of instruction

This treatment of G^ references and .ADDRESS directives preserve the position independence of shareable images.

To conserve disk space, the linker does not allocate memory for large arrays that do not contain data before the program is run. Instead, a descriptor for the array is placed in a special type of image section, a demand-zero section. At run time, the image activator allocates memory for these large arrays. This special treatment of large arrays only applies to executable images, not shareable images.

2.2 Image Activator Maps Image to Virtual Address Space

At run time, image sections are mapped to their assigned virtual addresses by the image activator. Figure 6 illustrates mapping an image composed of four image sections: three containing PSECTs and one with a pointer to the Run-Time Library shareable image.

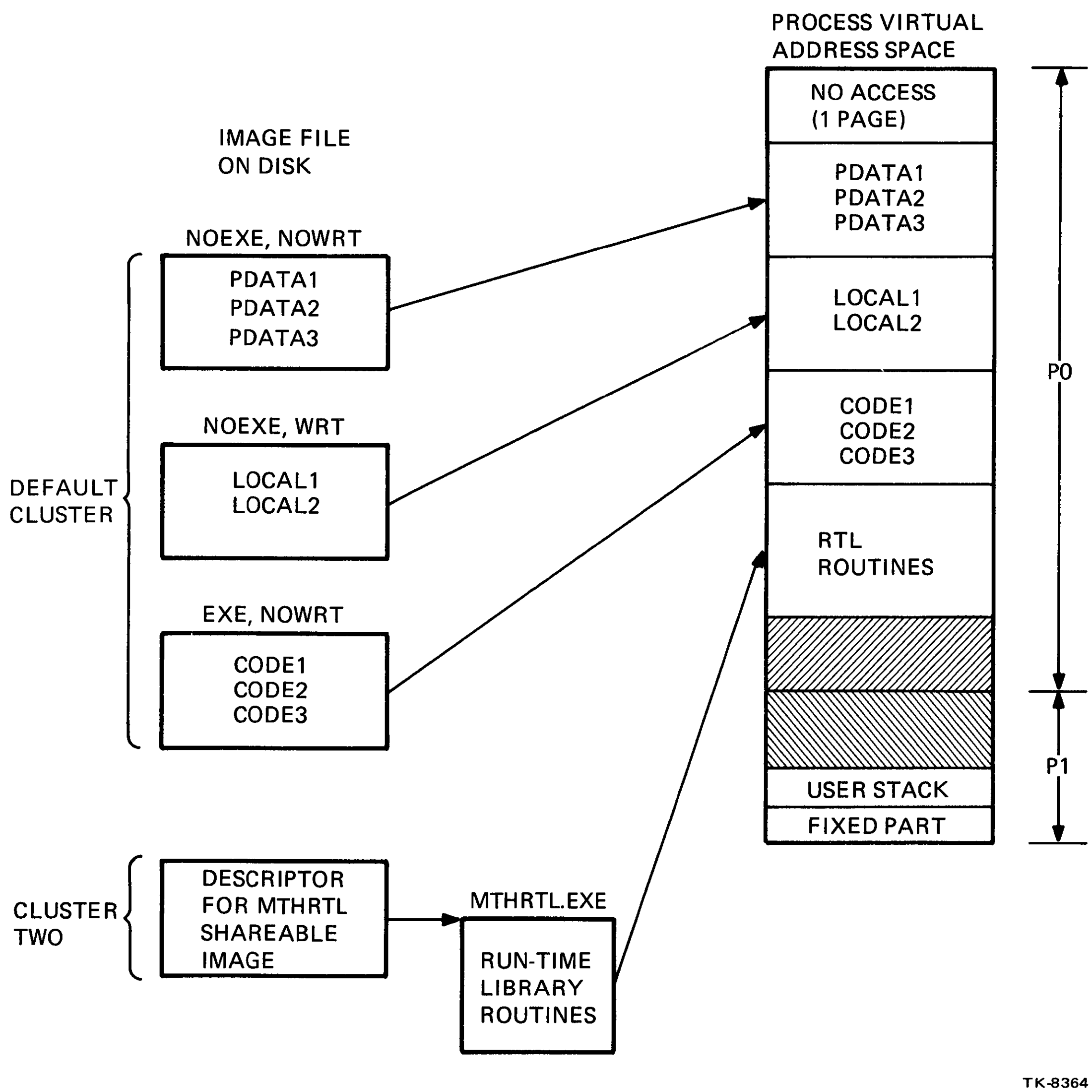


Figure 6 Mapping an Image into Process Virtual Address Space

Notice that the first page of virtual address space is inaccessible to catch common programming errors (for example, using data as addresses). Since this program references MTHRTL routines, the image activator uses the descriptor to locate MTHRTL.EXE, and maps the entire shareable image into the virtual address space. Any other referenced shareable images would be handled the same way.

3 Creating and Reading a Linker Map

The linker optionally creates a listing containing information about a program and the link operation. This listing, called a **linker map**, is often helpful when debugging run-time errors.

3.1 Creating a Linker Map

Including an optional qualifier on the LINK command directs the linker to create a linker map. The map can be in one of three formats:

- Brief Map
- Default Map
- Full Map

A full map contains the following sections of information, of which the brief and default maps contain subsets:

- Object Module Synopsis
- Image Section Synopsis
- Program Section Synopsis
- Symbols by Name (or Symbol Cross-Reference)
- Symbols by Value
- Image Synopsis
- Link Run Statistics

3.2 Using a Linker Map to Debug Run-Time Errors

A linker map, especially a full map, can be useful in debugging run-time errors and reading large listing files. Some of the uses for a linker map include:

- Locating an instruction that caused a run-time error.
- Translating a number displayed by the debugger to its related symbol or address.
- Locating symbol definitions.

The Program Section Synopsis is used with a listing file to determine the instruction that caused a run-time error:

1. **Obtain PC** — The error message and traceback should provide you with the program counter (PC). The PC indicates the virtual address of the instruction that caused the error. Alternately, the PC could be output by a user-written condition-handling routine.
2. **Locate PSECT** — The Program Section Synopsis lists the beginning and ending addresses of each program section in the image (the virtual addresses that each program section was mapped into). Locate the program section that contains the problem instruction by locating the PSECT that contains the PC.
3. **Calculate Offset** — Subtract the base address of the program section (from step 2) from the PC to obtain the offset into the PSECT of the erroneous instruction.
4. **Locate Instruction** — Consult the listing file for the program to obtain the instruction associated with that offset.

The Symbols by Reference section can be used to translate a number to its related symbol or address. For example, the debugger refers to most entities by number, but you usually want to know what symbol or address the numbers represent.

If you encounter a symbol in a large listing and need to know where it is defined, consult the Symbol Cross-Reference section of a full or default map. Note that this section is included instead of the Symbols by Name section only if the /CROSS_REFERENCE qualifier is included on the LINK command.

If you need to change a routine, you can consult the Symbol Cross-Reference section to determine all modules that reference that routine. This allows you to easily locate all codes that might be affected by your change, preventing future problems.

4 Linker Options Files

You may need to specify additional input and/or directions to the linker when you invoke the LINK command. Sometimes this additional information cannot be included on the command line. A linker **options file** includes this extra information. An options file is created using the DCL CREATE command, or a text editor.

Options files, which have the default file type .OPT, are used to:

- Store frequently used input file specifications.
- Enter large input specifications.
- Specify a shareable image as input.
- Alter program section attributes.
- Define clusters.
- Specify special instructions (options) to the linker.

The **Sharing Code and Data** module illustrates the use of an options file to specify a shareable image as input to the linker.

4.1 Creating and Using Linker Options Files

Linker options, like CLUSTER and PSECT_ATTR, cannot be included on the command line because DCL cannot recognize them. They are included in an options file.

An options file is specified as input to the linker by placing the name of the file on the command line, followed by the /OPTIONS qualifier:

$ LINK FILE, FILE2, OPTFILE/OPTIONS

It is sometimes convenient to enter the additional input to the linker directly from the terminal, rather than specifying a separate disk file. This can be done by specifying SYS$INPUT as the options file. The system will wait for you to enter the additional input, the end of which is signaled by entering CTRL/Z. For example:

$ LINK EMILIE, LIZ, SYS$INPUT/OPTIONS
HELPING/SHARE
ANOTHER/SHARE
<CTRL/Z>

If you frequently use the same options file as input to the linker, you may want to put the LINK command and the options file contents in a command procedure. Then you need only execute one command (invoking your command procedure) to execute the link operation:

$ @DOLINK

where DOLINK.COM contains the following:

$ LINK/FULL/MAP EMILIE, LIZ, SYS$INPUT/OPTIONS
HELPING/SHARE
ANOTHER/SHARE
<CTRL/Z>

4.2 Linker Options Records

Linker options records are available in MACRO only. These object code records allow the specification of additional files to the linking operation. See the *Guide to Programming in VAX MACRO* for more information about linker options records.

4.3 Using the Cluster Option to Create More Efficient Images

The order of the clusters, and the image sections within those clusters, determines the order in which the modules appear in the final image. The order in which files appear on the LINK command line does **not** necessarily reflect their order in the final image.

To increase program performance, especially for large applications, you may want to control the placement of object modules within clusters. Segments of code that frequently refer to each other should be close together in the executable image. Take, for example, the transaction processing application presented in Section 1.3, Linker Clusters. To ensure that the related routines are near each other in the final image, use the CLUSTER option of the LINK command:

CLUSTER= cluster-name, [base-adr], [pfc], [file-spec,...]

For this example, the option should be used as follows:

$ LINK MAIN, OTHERS/OPTIONS

where the file OTHERS.OPT contains:

CLUSTER=ONE,,,FORM1,PROCESS1
CLUSTER=TWO,,,FORM2,PROCESS2
CLUSTER=THREE,,,FORM3,PROCESS3

This command creates three clusters in addition to the default cluster, as shown in Figure 7. Note that the optional arguments may be omitted, but the commas may not. Refer to the *VAX/VMS Linker Utility Reference Manual* for a description of the arguments omitted from this example.

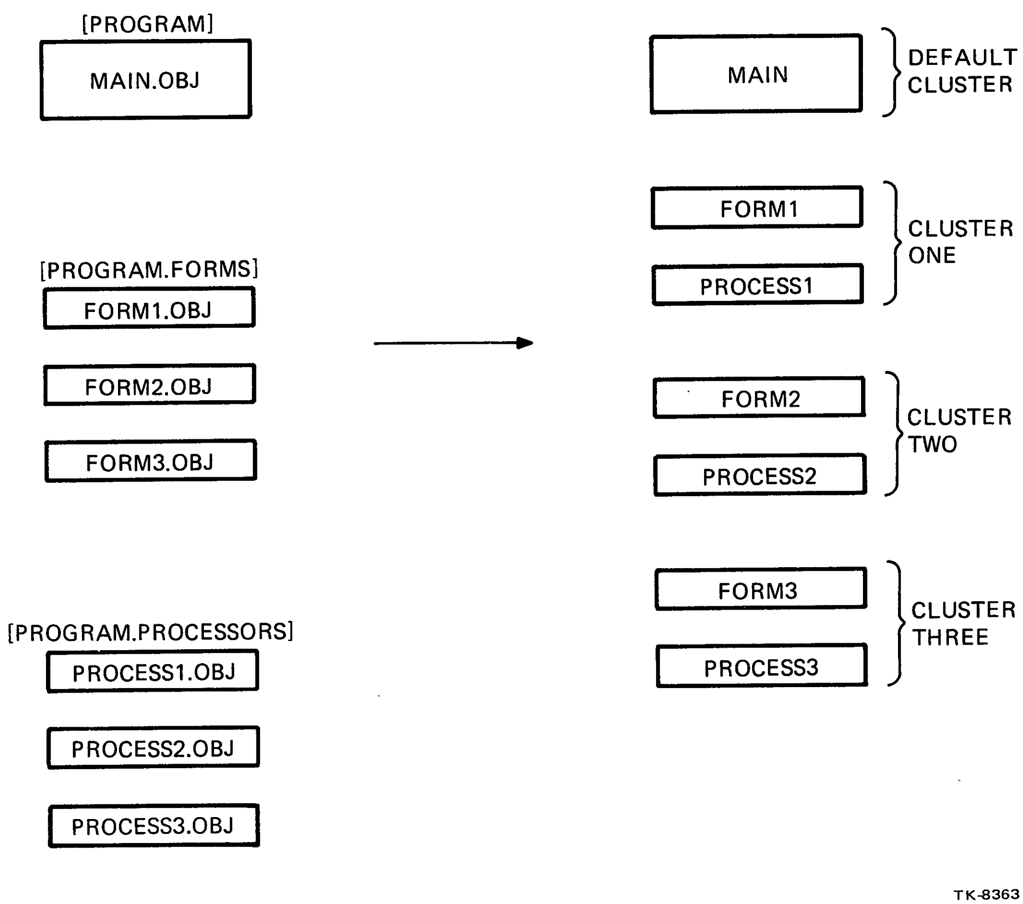


Figure 7 Clustering Related Code in an Executable Image

When the image is executed, the related routines are mapped consecutively into the physical memory allocated to the process. This decreases the amount of paging needed to execute the image, and causes the image to run faster. The system also runs faster, because paging activity is decreased.

In addition, MACRO programmers can collect modules into specified clusters at the PSECT level, not just on a file basis. This is done using the COLLECT option, referring to the PSECTs by name. High-level language programmers do not have control over PSECT names, and, therefore, cannot exercise the COLLECT option.

LEARNING ACTIVITY

1. Do the written exercises for this module.

Written Exercises

1. Multiple choice: The linker can create:

   a. Executable images

   b. Shareable images

   c. Linker maps

   d. All of the above

2. Match each term with its description by placing the appropriate number in each blank.

   Terms

   1. PSECT
   2. Object module
   3. Linker cluster
   4. Image section

   Descriptions

   _____ Contains code with similar properties

   _____ The unit in which the linker handles a program

   _____ The unit in which the image activator handles a program

   _____ Input for the linker

3. What is the advantage of clustering related code in a large image?

   a. Faster image activation

   b. Improved program performance

   c. Improved system performance

   d. All of the above

4. Specify which VMS component performs each activity by placing the appropriate number in each blank.

   VMS Components

   1. Linker
   2. Image activator

   Activities

   _____ Organize PSECTS into image sections

   _____ Map an image file to addresses in process virtual address space

   _____ Assign virtual addresses to image sections

   _____ Write image sections to an image file

   _____ Assign virtual addresses to position-independent shareable images

5. Specify which file would be used for each activity by placing the appropriate number in each blank.

   Files

   1. Linker map
   2. Linker options file

   Activities

   _____ Specify additional input and/or directions to the linker

   _____ Locate an instruction that caused a run-time error

   _____ Alter PSECT attributes

   _____ Translate a number displayed by the debugger to its related symbol or address

   _____ Define linker clusters

   _____ Locate symbol definitions

Solutions

1. The linker can create:

   a. Executable images

   b. Shareable images

   c. Linker maps

   ** d. All of the above

2. Match each term with its description by placing the appropriate number in each blank.

   Terms

   1. PSECT
   2. Object module
   3. Linker cluster
   4. Image section

   Descriptions

   __1__ Contains code with similar properties

   __3__ The unit in which the linker handles a program

   __4__ The unit in which the image activator handles a program

   __2__ Input for the linker

3. What is the advantage of clustering related code in a large image?

   a. Faster image activation

   b. Improved program performance

   c. Improved system performance

   ** d. All of the above

4. Specify which VMS component performs each activity by placing the appropriate number in each blank.

   VMS Components

   1. Linker
   2. Image activator

   Activities

   __1__ Organize PSECTS into image sections

   __2__ Map an image file to addresses in process virtual address space

   __1__ Assign virtual addresses to image sections

   __1__ Write image sections to an image file

   __2__ Assign virtual addresses to position-independent shareable images

5. Specify which file would be used for each activity by placing the appropriate number in each blank.

   Files

   1. Linker map
   2. Linker options file

   Activities

   __2__ Specify additional input and/or directions to the linker

   __1__ Locate an instruction which caused a run-time error

   __2__ Alter PSECT attributes

   __1__ Translate a number displayed by the debugger to its related symbol or address

   __2__ Define linker clusters

   __1__ Locate symbol definitions

EXERCISES

System Components

EXERCISES

For each system component named below, fill in the required information.

- Under **Implementation**, specify system process **(PCS)**, procedure **(PCR)**, exception service routine **(EXC)**, interrupt service routine **(INT)**, or shared image **(SHR)**.
- Under **Context**, indicate system **(SYS)** or process **(PCS)**.
- Under **Address Region**, specify program **(PGM)**, control **(CTL)**, or system **(SYS)**.
- Under **Purpose**, briefly describe the primary function of the component.

Component Name	Implementation	Context	Address Region	Purpose
system service	PCR	PCS	SYS	common internal function
1. scheduler	INT	SYS	SYS	context switch
2. swapper	PCS	PCS	SYS	swap
3. symbiont	PCS	PCS	~~[illegible]~~ PGM	device driver
4. AME	SHR	PCS	PGM	compatibility
5. XQP	PCR	PCS	CTL ~~[illegible]~~	file mapping
6. run-time library	SHR	PCS	PGM	HLL support
7. error logger	PCS	PCS	PGM	—
8. pager	EXC	PCS	SYS	paging
9. CLI	SHR	PCS	CTL	DCL
10. RMS	SHR	PCS	SYS	context mgt

System Components

SOLUTIONS

Component Name	Implementation	Context	Address Region	Purpose
system service	PCR	PCS	SYS	common internal function
1. scheduler	INT	SYS	SYS	chooses next process to execute
2. swapper	PCS	PCS	SYS	system-wide mem.management
3. symbiont	PCS	PCS	PGM	input/output spooling
4. AME	EXC	PCS	PGM	implements compatibility mode
5. XQP	PCR	PCS	CTL	implements ODS-2 file structure
6. run-time library	PCR	PCS	PGM	common subroutines and functions
7. error logger	PCS	PCS	PGM	records hardware errors
8. pager	EXC	PCS	SYS	process memory management
9. CLI	SHR	PCS	CTL	command language processing
10. RMS	PCR	PCS	SYS	record/file management

System Components

EXERCISES

1. <u>Using the System Dump Anaylzer (SDA)</u>

Throughout this week you will be encountering data structures and concepts that will require further explanation. One way to assist in this is to examine the contents of a VMS system's memory (or a copy of it). The System Dump Analyzer (SDA) allows you to do just that. SDA is an interactive utility enabling you to examine:

- the system dump file, SYS$SYSTEM:SYSDUMP.DMP (read access required)
- a copy of the system dump file (read access required)
- the actively running system (CMKRNL privilege required)

This exercise will "walk" you through an examination of a system dump file. Do not attempt to examine the actively running system until you have completed this lab and have the permission of your instructor.

a. Activate the System Dump Analyzer (SDA) using the command

```
$ ANALYZE/CRASH OSI$LABS:CRASH1.DMP
```

b. The basic crash information will be displayed on your terminal:

- date of crash
- reason for crash

EXERCISES

c. At the SDA prompt (SDA>), enter the command "HELP". The commands available are displayed on the terminal. To find out more information about a command, enter:

```
SDA> HELP 'command'
```

d. Using the HELP command, find out about each of the following commands:

- SET
- SHOW
- FORMAT
- READ

e. Once you feel comfortable with the definition and purpose of the above SDA commands, issue the following commands to see what information each provides.

- SHOW SUMMARY
- SHOW PROCESS
- SHOW SYMBOL/ALL
- SHOW POOL/IRP

f. Use the following commands to display the message text associated with some common condition codes:

- EVALUATE/CONDITION 1
- EVALUATE/CONDITION C

EXERCISES

g. Some locations in P1 and S0 virtual address space store pointers to code and data used by the operating system. VMS defines global symbols for these virtual addresses.

Consult the Naming Conventions chapter in VAX/VMS Internals and Data Structures for information on the syntax of VMS global symbols.

For example, the global symbol EXE$GL_SCB equates to an S0 address that contains the address of the System Control Block (SCB), as shown in Figure 1.

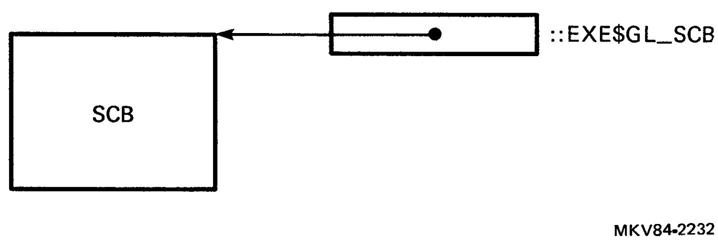


Figure 1 Global Symbol Locating Pointer to SCB

- Determine the value of the symbol EXE$GL_SCB using the EVALUATE command in SDA. Record the hexadecimal and decimal values below.

- Determine the contents of the address EXE$GL_SCB using the EXAMINE command. Record the contents below, in hexadecimal and ASCII formats.

- Determine the contents of the first longword of the SCB using the following command:

```
SDA> EXAMINE @EXE$GL_SCB
```

The unary operator "@" is used in SDA to provide a level of indirection.

EXERCISES

A summary of the above commands and another example are provided in Figure 2 and Table 1.

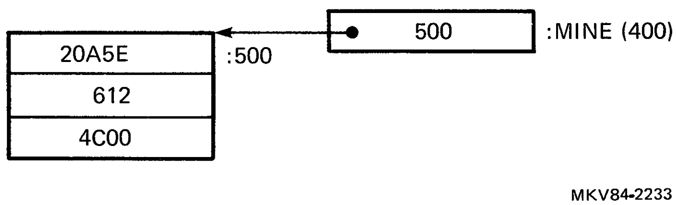


Figure 2 Sample Addresses and Symbols

Table 1 Using Symbols in SDA

SDA Commands and Output	Notes
SDA> evaluate MINE Hex = 00000400 Decimal = 1024	Value of symbol is displayed in hex and ASCII formats
SDA> examine MINE MINE: 00000500 "...."	Contents at address 400 are displayed
SDA> show symbol MINE MINE = 00000400 : 00000500	Value of symbol and contents at that address are displayed
SDA> examine @MINE 0000500: 00020A5E	Symbol equals address 400 which contains a 500; contents at address 500 are shown

h. To provide the additional symbolic definitions necessary in the following questions, use the SDA READ command to read in the file OSI$LABS:GLOBALS.STB.

EXERCISES

i. The list below contains some of the system-defined symbols you will be seeing throughout the course. These particular symbols equate to addresses.

Choose five symbols and determine and record, for each:

1. Its value
2. The contents at that address
3. The contents at the address obtained in step (2)

The symbols are:

- SCH$GL_CURPCB 80002Lf8 / 801B3390 / 8000204C
- CTL$GL_PHD 7ffefe88 / 7ffd8860 / ffffffff
- CTL$GL_PCB
- CTL$GQ_PROCPRIV
- EXE$GL_RPB 80003F7C / 8011e460 / 0
- IOC$GL_IRPBL 80002A58 / 802D8D60
- IOC$GL_IRPFL
- SCH$GL_COMQS
- SCH$GL_PCBVEC
- SCH$GQ_HIBWQ (mask)
- SCH$GQ_LEFWQ Fl Bl | 80002 1A4 rpt | 5000

j. Format the data structures pointed to by the following symbols:

- SCH$GL_CURPCB
- IOC$GL_IRPFL (SIB)

EXERCISES

k. Issue the SHOW CRASH command, and use the output to answer the following questions:

- What was the current process at the time of the crash?

  Martin

- What image (if any) was executing?

- What was the reason for the crash (according to SDA)?

  bugcheck SSRVEXCEPT
  unexpected sysservice

l. Exit SDA and return to the DCL prompt.

2. Read the following chapters in the <u>VAX/VMS System Dump Analyzer Reference Manual</u>:

   a. Introduction

   b. Using SDA

   c. Reading the System Dump File

   d. SDA Command Format

   The last section of the manual contains descriptions of the SDA commands. Keep this manual handy for quick reference while working on other lab exercises.

3. Throughout the course you will see system symbols referencing S0 addresses. The contents at these addresses change over the life of the system. Examining these addresses allows you to observe various system activities. This is the purpose of the MONITOR utility.

   Write a MACRO program that examines the word in S0 space that records the maximum number of processes that are allowed on the system. This location is referenced by the symbol SGN$GW_MAXPRCCT.

   You can use the template program in OSI$LABS:COMPTEMP.MAR.

SOLUTIONS

1. Consult your instructor for the solutions to these exercises.

2. Consult your instructor for the solutions to these exercises.

3. The program in Example 1 examines and displays the contents referenced by SGN$GW_MAXPRCCT.

```
        .TITLE  COMPLAB3
;++
;
; ABSTRACT:
;
;       This program examines and displays the maximum
;       process count, at SGN$GW_MAXPRCCT.
;
; ENVIRONMENT:
;
;       Changes mode to executive.  CMEXEC privilege required.
;
;       Linked with SYS.STB:
;       $ LINK  COMPLAB3, SYS$SYSTEM:SYS.STB/SELECTIVE
;--
;       Declare macros
        .MACRO  CHECK_STATUS    CODE=R0, ?GO
        BLBS    R0, GO
        PUSHL   R0
        CALLS   #1,G^LIB$STOP
        RET
GO:
        .ENDM   CHECK_STATUS

        .MACRO  CONVERT1        BINARY, TEXT
        PUSHAL  TEXT
        PUSHAL  BINARY
        CALLS   #2, G^OTS$CVT_L_TZ
        CHECK_STATUS
        .ENDM   CONVERT1
```

Example 1 Examining an S0 Location (Sheet 1 of 3)

SOLUTIONS

```
        .MACRO   CONCAT2 BUFFER,ARG1,ARG2
        PUSHAL   ARG2
        PUSHAL   ARG1
        PUSHAL   BUFFER
        CALLS    #3,G^STR$CONCAT
        CHECK_STATUS
        .ENDM    CONCAT2

        .MACRO   DISPLAY          MESSAGE
        PUSHAL   MESSAGE
        CALLS    #1,G^LIB$PUT_OUTPUT
        CHECK_STATUS
        .ENDM    DISPLAY

; **************************************************************
        .PSECT   DATA     NOEXE,WRT,NOSHR

E_ARG_LIST:
                .LONG    1                  ; for $cmexec call
                .ADDRESS MAX_PROC_CNT       ;  passed by reference
MAX_PROC_CNT:   .BLKW    1                  ; word for max proc cnt
LWORD_MAX:      .BLKL    1                  ; for lw form of max cnt
; declare ascii formats of version longwords, and descriptors
CNT_ASCII:      .BLKB    8                ; 4 bytes x 2 chars = 8 max
CNT_DESC:       .LONG    8
                .ADDRESS CNT_ASCII
HDR_DESC: .ASCID  /Current maximum process count, in hex, is:  /
BIG_STRING:     .LONG    80                 ; for concatenated string
                .ADDRESS BYTES
BYTES:          .BLKB    80
```

Example 1 Examining an S0 Location (Sheet 2 of 3)

SOLUTIONS

```
; ****************************************************************
        .PSECT   CODE     EXE,NOWRT,PIC,SHR
START:  .WORD    ^M<>

;       read max process count... need to be in exec mode
        $CMEXEC_S        routin= 100$, arglst= E_ARG_LIST
        CHECK_STATUS

        MOVZWL     MAX_PROC_CNT, LWORD_MAX  ; need lw for convertl

;       convert longwords to ascii, concatenate, and output
        CONVERTl   LWORD_MAX, CNT_DESC
        CONCAT2    BIG_STRING, HDR_DESC, CNT_DESC
        DISPLAY    BIG_STRING

        MOVL     #SS$_NORMAL, R0             ; set normal completion
        RET                                  ; all done

; **************  executive mode code  *******************
100$:   .WORD    ^M<>
;
;       move version number into argument list
        MOVW     G^SGN$GW_MAXPRCCT, @4(AP)
        MOVL     #SS$_NORMAL, R0
        RET                                  ; finished in exec. mode

        .END     START
```

Example 1 Examining an S0 Location (Sheet 3 of 3)

EXERCISES

For each resource associated with, or used by, a process and listed on the following page:

- Name the data structure or component that implements or controls it.
- State the region (program, control, or system) in which the data structure or component resides.
- State whether the data structure or component is paged.
- State whether the data structure or component is included in the working set of the process and swapped.

For resources that are **not** part of a larger data structure (for example, the user stack), simply copy the name into the data structure column. For resources that occur in multiple locations, answer for each location.

EXERCISES

Resource	Data Structure	Region	Paged?	Swapped?
user stack	user stack	control	yes	yes
page tables	PHD	Sys	yes	yes
privilege mask	PHD	Sys	yes	yes
CLI data areas	CLI	CTL	No	No
run-time library	RTL	P0	Yes	Yes
general-purpose regs. when process is not the current one	PHD Hdwe PCB	Sys	No	Yes
process priority	PCB	Sys	No	No
quotas/limits on system resources	PHD	Sys	Yes	Yes
VAX-11 RMS code	RMS	Sys		
image of user program	P0	P0	Yes	Yes
working set list	PHD	Sys	No	Yes
kernel stack	—	P1	No	Yes
process I/O data structures	—	CTL	Yes	Y
process ID	PCB	Sys	N	N
CLI code	P1	CTL	Y	Y
interrupt stack	—	Sys	No	No

SOLUTIONS

Resource	Data Structure	Region	Paged?	Swapped?
user stack	user stack	control	yes	yes
page tables	process header	system	yes	yes
privilege mask	process header software PCB pointer page	system* system control	no no no	yes no yes
CLI data areas	CLI data areas	control	yes	yes
run-time library	run-time library	program	yes	yes**
general-purpose registers when process is not the current one	hardware PCB	system*	no	yes
process priority	software PCB	system	no	no
quotas/limits on system resources	software PCB JIB	system system	no no	no no
VAX-11 RMS code	RMS code	system	yes	no
image of user program	image	program	yes	yes**
working set list	process header	system*	no	yes
kernel stack	kernel stack	control	no	yes
process I/O data structures	process I/O data structures	control	yes	yes
process ID	software PCB	system	no	no
CLI code	CLI code	control	yes	yes**
interrupt stack	interrupt stack	system	no	no

*These portions of the PHD are also mapped by the P1 "window."
**These software components are or may be global read-only sections. As such, they are included in the process working set, but may not be outswapped with the rest of the working set. (See VAX/VMS Internals and Data Structures for details.)

The Process

EXERCISES

1. The System Dump Analyzer can be used to obtain information about the processes on a system at the time of a crash.

   Enter the SDA with the following command:

   ```
   $ ANALYZE/CRASH  OSI$LABS:CRASH1.DMP
   ```

   Issue the following SDA commands and observe the information they provide about VMS processes.

   a. Issue the SDA command SHOW SUMMARY/IMAGE and note the information it provides.

      An external process ID (EPID) uniquely identifies a process on a single system, or on a VAXcluster. Process IDs are discussed in more detail later in the course.

      This listing also shows the addresses of the software PCB and the process header for each process.

   b. Issue the SDA command SHOW PROCESS.

      By default, this command displays information from the process software PCB.

      - Record the name of the process. Martin

      - Record the address of the software PCB for the process. 801B3390

   c. Read the symbol table file OSI$LABS:GLOBALS.STB into your SDA session to provide the symbolic definitions required for some later questions.

   d. SHOW PROCESS does not display all the information from the software PCB. Use the FORMAT command, and the address you recorded in question (b), to display the contents of the process's software PCB.

EXERCISES

e. When SDA is invoked, it chooses a process to be its current process, and thus the target of any process-specific SDA commands. When analyzing a dump file, SDA's initial current process is the process that was executing when the system failed. If you invoke SDA to examine the running system, the current process is your process.

   The SET PROCESS command is used to change process context in SDA.

   - Use the SET PROCESS command to make OPCOM SDA's current process.

   - Issue the SHOW PROCESS command to display information about the OPCOM process.

   - Use the SET PROCESS command to restore the initial current process.

f. Using the SDA manual, or the HELP command in SDA, read about the qualifiers to the SDA SHOW PROCESS command.

g. Issue the appropriate form of the SHOW PROCESS command to display data from the process data structure that maintains process memory management information.

   /PPT/WSL

h. Issue the appropriate form of the SHOW PROCESS command to display the values of the process registers.

   /Reg

EXERCISES

i. The EXAMINE/PSL command can be used to produce a formatted display of a processor status longword. This is often easier than deciphering the fields manually.

Issue the following command to format the PSL for SDA's current process.

```
SDA> EXAMINE/PSL  PSL
```

What is the current IPL for this process? Ø

j. Determine the address of the process header for the OPCOM process.

k. Format the process header for OPCOM.

Remember that the process header does not have a TYPE field. You must, therefore, use a qualifier on the FORMAT command to tell SDA you are referencing a process header.

l. Read the description of the READ command in the VAX/VMS System Dump Analyzer Reference Manual. Which system-supplied symbol table contains symbols for the I/O database?

EXERCISES

2.

a. At DCL level, issue the following command to list the modules of the STARLET macro library at your terminal:

```
$ LIBRARY/LIST  SYS$LIBRARY:STARLET.MLB
```

Do you recognize any of the modules in this library?

b. List the modules of SYS$LIBRARY:LIB.MLB on your terminal.

Do you recognize any of the modules in this library?

You may want to make a hard copy of this listing for future reference.

c. What kind of programmer would reference the modules in STARLET.MLB? in LIB.MLB?

The Process

SOLUTIONS

1. Enter SDA with the command shown.

   a. Issue the SHOW SUMMARY/IMAGE command as shown.

   b.

   - The name of the process is shown at the top of the display.

   - The address of the software PCB is at the top of the first column of the SHOW PROCESS display. Note that the address is in system virtual address space (S0).

   c. SDA> READ OSI$LABS:GLOBALS.STB

   d. SDA> FORMAT pcb_address_from_lb

   e.

   - SDA> SET PROCESS OPCOM

   - SDA> SHOW PROCESS

   - SDA> SET PROCESS initial_process_name

   f. Use the SDA manual or the on-line help to find out about the qualifiers for the SHOW PROCESS command.

   g. SDA> SHOW PROCESS/PHD

   h. SDA> SHOW PROCESS/REGISTERS

   i. The current IPL for the process is in bits 16-20 of the PSL, and is labeled with "IPL" in the EXAMINE/PSL display.

   j. SHOW PROCESS OPCOM will display the address of the process header for OPCOM.

   k. FORMAT/TYPE=PHD address_from_lj

   l. SYSDEF.STB contains symbols for the I/O database.

SOLUTIONS

2.

a. The modules in STARLET.MLB include macros for calling system services, calling RMS routines, and defining user-level RMS data structures.

b. The modules in LIB.MLB include macros defining offsets into many system-level data structures, and macros for common VMS activities.

c. Nonprivileged programmers might make use of the modules in STARLET, whereas LIB is used primarily by privileged, system-level programmers.

EXERCISES

1. VMS uses a variety of mechanisms to synchronize its activities.

   a. To synchronize access to the scheduler's data structures, a program raises IPL to IPL$_SYNCH. Why does the program raise IPL, rather than request an interrupt at IPL 8?

   b. Why can't a mutex be used to lock the scheduler's data structures?

   c. Which VMS mechanism is used to synchronize access to the system logical name table?

2. When an exception or interrupt occurs, the PSL and the PC are pushed onto the stack, and a new PC and PSL are created.

   a. Which stack is used?

   b. How is the new PC value formed?

EXERCISES

c. What are the contents of the **current mode** and **previous mode** fields of the new PSL?

d. What is the new IPL?

e. When an REI instruction is executed, is the **previous mode** field of the PSL significant? Explain.

3.

a. The following table illustrates a hypothetical sequence of hardware and software interrupts. At each step, fill in the contents of the indicated items. In the "Saved IPL" column, indicate the stack that contains the saved IPL. Indicate where control is passed after each REI instruction. All numbers are decimal. Assume that software interrupts above IPL 6 are handled on the interrupt stack, and that those at IPL 1 through IPL 6 are handled on the kernel stack. Further assume that all device interrupts are handled on the interrupt stack.

EXERCISES

Note that this example is hypothetical and bears little resemblance to the VAX/VMS operating system. Its purpose is to explore the workings of interrupts, especially software interrupts.

Event	Stack	IPL	SISR(hex)	Saved IPL
1. Executing user image				
2. Device int. at IPL 21				
3. SOFTINT 8				
4. REI to ______				
5. SOFTINT 5				
6. SOFTINT 3				
7. REI to ______				
8. Device int. at IPL 20				
9. SOFTINT 8				
10. REI to ______				
11. SOFTINT 4				
12. REI to ______				
13. REI to ______				
14. REI to ______				
15. REI to ______				

EXERCISES

b. In steps 7 and 12, a switch is made from the interrupt stack to the kernel stack. Why?

4.

a. Briefly describe how system services are dispatched. Assume that **no** errors occur. Include all steps from the program's initial call until control is passed back to that program.

b. Why does the routine SRVEXIT issue an REI instruction?

c. Several system services have access mode as one of their arguments. The service routines that perform these requests first call a routine called Maximize Access Mode that chooses the **least** privileged access mode of the one requested and the access mode of the caller. Describe how this might be done. Why is it done?

EXERCISES

5. List two differences between the exception dispatching within the executive and the Common Run-Time Library procedure LIB$SIGNAL.

SOLUTIONS

1.

a. An IPL 8 interrupt would invoke the IPL 8 fork dispatcher, which is not the desired result. Remember the difference between using IPLs for blocking and synchronization, and using IPLs to determine how to service an interrupt.

b. Mutexes are a synchronization technique available to processes. When on the interrupt stack, the system is not in any process context. Hence the method of elevating IPL is the only synchronization technique available.

c. A mutex is used to synchronize access to the system logical name table.

2.

a. The entry to an exception or interrupt service routine must be longword aligned. Thus, the two low bits in the SCB can be used for other purposes. Bit 0 determines whether the interrupt is handled on the kernel stack (bit 0 clear) or on the interrupt stack (bit 0 set).

SOLUTIONS

All device interrupts are handled on the interrupt stack. All software interrupts (except ASTDEL at IPL 2 and RESCHED at IPL 3) are handled on the interrupt stack.

CHMx exceptions are placed on the resultant perprocess stack. Machine Check, Power Fail, and Kernel Stack Not Valid exceptions are handled on the interrupt stack. The rest of the exceptions are handled on the kernel stack.

b. The new PC value is the address found in bits<31:2> of the SCB entry for this particular exception or interrupt. (PC bits<1:0> are always cleared.)

c. For all exceptions **except** CHMU, CHMS and CHME, the current mode will be zero, kernel access mode.

For exceptions, the previous mode field will be the access mode that the CPU was in when the exception occurred. In fact, PSL<previous mode> is the same as the current mode field of the saved PSL on the stack.

The previous mode field of the PSL is set to 0 (kernel mode) following an interrupt.

d. The new IPL depends upon the interrupt or exception:

Exceptions	IPL (decimal)
Machine check	31
Kernel stack not valid	31
All other exceptions	unchanged!
Software Interrupts	IPL raised to corresponding level
Hardware Interrupts	
Interval timer	24
Console	20
Other devices	20-23
Power fail	30

SOLUTIONS

e. No, the previous mode field of the PSL is not significant when an REI executes. The previous mode field is an historical parameter, recording where the processor came from. The previous mode field is used by the PROBEx instructions.

The relevant field (and the one checked by the REI instruction microcode) is the current mode field of the PSL on the stack. If privileged software wishes to alter its destination, IPL, or mode, then this longword is what should be changed.

3.

a.

Event	Stack	IPL	SISR(hex)	Saved IPL
1. Executing user image	user	0	0	--
2. Device int. at IPL 21	interrupt	21	0	0(I)
3. SOFTINT #8	interrupt	21	100	0(I)
4. REI to IPL 8 serv. routine	interrupt	8	0	0(I)
5. SOFTINT #5	interrupt	8	20	0(I)
6. SOFTINT #3	interrupt	8	28	0(I)
7. REI to IPL 5 serv. routine	kernel	5	8	0(K)
8. Device int. at IPL 20	interrupt	20	8	5(I),0(K)
9. SOFTINT #8	interrupt	20	108	5(I),0(K)
10. REI to IPL 8 serv. routine	interrupt	8	8	5(I),0(K)

SOLUTIONS

3.a. (Cont)

11. SOFTINT #4	interrupt	8	18	5(I),0(K)
12. REI to interrupted IPL 5 serv. routine	kernel	5	18	0(K)
13. REI to IPL 4 serv. routine	kernel	4	8	0(K)
14. REI to IPL 3 serv. routine	kernel	3	0	0(K)
15. REI to interrupted user image	user	0	0	--

b. At step 7, the REI triggers a software interrupt at IPL 5. One of the assumptions was that IPL 5 (actually IPL 6 and below) interrupts were to be handled on the kernel stack.

At step 12, the restored PSL requires IPL 5 but also PSL<IS> is clear. The REI instruction microcode then switches stacks, in this case to the kernel stack.

4.

a. The user program issues a CALLx instruction to the vector area of system virtual address space. A CHMK or CHME instruction transfers control to a change mode dispatcher that builds a call frame and then executes a CASE instruction to dispatch to the service specific procedure.

When that procedure completes its operations, it executes an RET instruction which returns control to a routine SRVEXIT. Because no error occurred (as assumed), an REI instruction is executed to pass control back to the vector area where another RET instruction returns control to the user program.

SOLUTIONS

b. The CHMK and CHME instructions cause corresponding exceptions that push a PSL and PC pair plus a service code used in dispatching and change access mode to the required mode. The exit from the exception service routine must be an REI instruction to restore the previous access mode and reset the PC and PSL.

c. The caller's access mode can be obtained from either the previous mode field from the **current** PSL or from the current mode field of the **saved** PSL.

   Because the saved PSL may be at an unspecified offset from the top of the stack, the previous mode field of the current PSL is simply compared to the access mode passed as an argument to the system service. The larger (less privileged) access mode is the one used by the system service.

   This operation is performed to ensure that a nonprivileged image does not gain access rights by, for example, queuing an executive or kernel mode AST to itself.

5. LIB$SIGNAL may be invoked by any code on detection of an error that is to be treated as an exception. Software makes the decision.

   The exception dispatcher is entered as a result of hardware exceptions and a small set of software exceptions.

   LIB$SIGNAL, through its alternate entry point LIB$STOP, can force an image to exit. The exception dispatcher has no such feature, although a condition handler could issue a $EXIT system service.

EXERCISES

1. Using the System Dump Analyzer, obtain the following information about the system recorded in the dump file named OSI$LABS:CRASH1.DMP.

   It will be helpful to read in the file OSI$LABS:GLOBALS.STB.

   a. Locate the listhead for the system timer queue.

      (HINT: The listhead consists of two longword pointers, each of which can be located using a global system symbol (EXE$GL_xxxx).)

   b. Locate a timer queue entry for a system subroutine request.

      (HINT: One of the bits in the TQE$B_RQTYPE field indicates whether or not the TQE represents a system subroutine request. Consult Internals and Data Structures for information on the use of system subroutine requests.)

   c. What is the PC of the routine that will be invoked by the software timer when this TQE expires?

   d. Scan some other entries in the timer queue. Note the kinds of requests that are being made.

EXERCISES

2. [Optional] VMS allows privileged users to write and implement their own system services.

   a. User-written system services are implemented as privileged shareable images. Read about privileged shareable images in the VAX/VMS Release Notes for version 4.0.

   b. Install and test the sample user-written system services in the SYS$EXAMPLES directory.

      - Obtain a copy of the files from SYS$EXAMPLES:

        - USSDISP.MAR
        - USSLINK.COM
        - USTEST.MAR
        - USSTSTLNK.COM

      - Assemble the .MAR files.

        You may want to include the debugger with USSTEST. That will make it easier to verify whether or not the program works since it does not do any output.

      - Link the privileged shareable image containing the user-written system services using USSLNK.COM.

        To avoid conflicts with other students in the class, rename the resulting shareable image file to a unique name (for example, using your initials).

      - Link the USSTEST object module with the shareable image file. Follow the format used in USSTSTLNK.COM, replacing USS.EXE with the name of your shareable image file.

        Link USSTEST with the debugger if you like.

      - By default, the image activator expects all shareable image files to be in SYS$SHARE.

        Therefore, you should define a logical name for your shareable image file. Equate the file name to the full file specification.

EXERCISES

For example, if your shareable image were named

```
WORK1:[HUNT.LABS]USSLH.EXE;1
```

you would make the following logical name assignment:

```
$ DEFINE  USSLH  WORK1:[HUNT.LABS]USSLH.EXE
```

- Install the shareable image with the /PROTECT and /SHARE attributes. Be sure to specify the full file specification.

  You will need CMKRNL privilege to do this.

- Run the USSTEST program to ensure that it works. If you included the debugger, examine R0 and location BUF after the call to USER_GET_TODR.

- Remember to deINSTALL the shareable image when you are done.

SOLUTIONS

1.

a. First locate the listhead for the timer queue using the symbol EXE$GL_TQFL. Examine the TQE$B_RQTYPE field of each timer queue entry, looking for an entry with an odd value in this field. If the low bit in the TQE$GL_RQTYPE field is set, then the request is for a system subroutine.

b. The PC of the routine to be invoked by the software timer is at offset TQE$L_FPC in the timer queue entry.

c. To locate successive entries in the queue, use the value at offset TQE$L_TQFL in each entry. You can scan backwards using the value at offset TQE$L_TQBL.

2.

a. In addition to the information in the <u>VAX/VMS Release Notes</u>, you will find an overview of user-written system services in the comments of the template files in SYS$EXAMPLES.

b.

```
$ COPY SYS$EXAMPLES:USS*.* your_directory
$
$ ! assemble the files
$ MACRO USSDISP
$ ! include debugger with USSTEST if desired
$ MACRO USSTEST
$
$ ! link shareable image, and rename to unique name
$ @USSLINK.COM
$ RENAME USS.EXE your_file_name.EXE
$
$ ! link the main program; include debugger if desired
$ LINK/MAP/FULL USSTEST, SYS$INPUT/OPTIONS
  your_file_name.EXE/SHARE
  ^Z
$
$ ! continued on next page....
```

SYSTEM MECHANISMS

SOLUTIONS

```
$ ! define logical name for shareable image so
$ !  image activator will locate it properly
$ DEFINE  your_file_name  your_full_file_spec
$
$ ! get privileges for install
$ SET PROCESS/PRIV=(CMKRNL)
$ ! install the shareable image
$ RUN SYS$SYSTEM:INSTALL
INSTALL> your_full_file_spec/SHARE/PROTECT
INSTALL> your_full_file_spec/LIST
INSTALL> ^Z
$ SET PROCESS/PRIV=(NOCMKRNL)
$
$ ! test the program, and then deinstall
$ RUN USSTEST
$ SET PROCESS/PRIV=(CMKRNL)
$ RUN SYS$SYSTEM:INSTALL
INSTALL> your_full_file_spec/DELETE
INSTALL> ^Z
$ SET PROCESS/PRIV=(NOCMKRNL)
```

Debugging Tools

EXERCISES

1. Which debugger would you use under the following conditions?

   a. Examine the current system

   b. Examine a crash dump

   c. Debug a user mode image at IPL 0

   d. Debug a driver

2. Which is NOT a reason for a crash dump to occur?

   a. Exception at elevated IPL

   b. User mode image error

   c. Machine check in kernel mode

EXERCISES

3. Use SYS.MAP and the other listings in your Source Listings book to answer the following questions about the $SUSPND system service and AST delivery.

$SUSPND System Service

a. Which module contains the code that implements the $SUSPND system service? (Remember that all system services have two entry points, one of the form SYS$name that is the starting address of the vector entry, and one of the form EXE$name that is the starting point of the actual code.)

b. What other routines are defined in this module?

c. How long (in bytes) is this module?

d. Which system mechanism is used to suspend a process?

EXERCISES

e. List all of the system subroutines that are called by the $SUSPND system service.

f. A process can suspend another process only if it is in the same group **and** the issuing process has GROUP privilege, **or** if the issuing process has WORLD privilege. Where in the code is this check made? What other system services need to make this check?

g. The $HIBER system service does not make the same UIC and privilege check that $SUSPND does (see question (f)). Why?

AST Delivery

h. What line of the $SUSPND system service actually queues the AST?

EXERCISES

i. What section of code in the routine SCH$NEWLVL computes the ASTLVL value and stores the value in the hardware PCB and ASTLVL processor register?

j. Assume that the current process is issuing a $SUSPND for itself, and that it will be able to complete the $SUSPND system service without interruption. At what point in the system service dispatching sequence will the AST delivery code (the IPL 2 interrupt service routine) be entered? (This is the code that will eventually transfer control to the AST routine.)

Debugging Tools

SOLUTIONS

1.

   a. To examine the current system, use the System Dump Analyzer.

   b. To examine a crash dump, use the System Dump Analyzer.

   c. The symbolic debugger is used to debug user mode images at IPL 0. For other access modes at IPL 0, use the DELTA debugger.

   d. Use XDELTA to debug a driver, which operates at elevated IPL in kernel access mode.

2. A user mode image error will not cause a crash dump to occur. What will occur is a traceback, and any condition handling that has been set up.

Debugging Tools

SOLUTIONS

3.

$SUSPND System Service

a. SYSPCNTRL is the module that defines the symbol EXE$SUSPND.

b. There are two ways to find the routines defined in SYSPCNTRL. The easiest way is to look at the table of contents of the SYSPCNTRL module listing. This lists all the entry points:

EXE$SUSPND	EXE$NAMPID
EXE$RESUME	EXE$xPID_TO_xxx
EXE$HIBER	EXE$SETPRN
EXE$WAKE	

Another way to answer this question is to first find the PSECT in which the SYSPCNTRL module resides. This is accomplished by searching sequentially through the Program Section Synopsis of SYS.MAP until SYSPCNTRL is found. Ignore any reference that shows identical base and end virtual addresses.

SYSPCNTRL appears on page 8 under the AEXENONPAGED PSECT with a base of 8000B2B5 and an end of 8000B54A. Note that the length of 296 also appears here, which answers question (c) as well. Any routines defined by SYSPCNTRL must have entry points that fall between the base and end addresses.

All symbols are listed in numerical order in the Symbols By Value section of SYS.MAP. On page 98 you will find the following entry points:

8000B2B5	EXE$SUSPND
8000B32B	EXE$RESUME
8000B340	EXE$HIBER
8000B356	EXE$WAKE
8000B367	EXE$NAMPID
8000B44E	EXE$EPID_TO_PCB
8000B455	EXE$IPID_TO_PCB
8000B477	EXE$EPID_TO_IPID
8000B4AA	EXE$IPID_TO_EPID
8000B4D7	EXE$SETPRN

c. The length of the module is 296 bytes hexadecimal or 662 bytes decimal. This can be found on page 8 of SYS.MAP as described in question (b), or by looking at the last line of code in the SYSPCNTRL module.

SOLUTIONS

d. The system suspends a process by queuing a kernel mode AST to the target process, as mentioned in the comments on page 4 of SYSPCNTRL (under Functional Description).

e. The following system subroutines are used:

```
EXE$NAMPID
EXE$ALLOCIRP
SCH$QAST
```

f. The UIC and privilege check is made in the EXE$NAMPID routine. The actual check occurs in line 497 for group privilege and line 496 for world privilege.

The other system services that need to make this check are:

```
$DELPRC          $SCHDWK
$RESUME          $FORCEX
$WAKE            $SETPRI
$CANWAK          $GETJPI
```

Most of these services can be deduced from the names of the modules that reference EXE$NAMPID, found on page 35 of SYS.MAP:

```
SYSPCNTRL            SYSFORCEX
    $SUSPND              $FORCEX
    $RESUME          SYSGETJPI
    $WAKE                $GETJPI
SYSCANEVT            SYSRTSLST
     $CANWAK              $GRANTID
SYSDELPRC            SYSSCHEVT
     $DELPRC              $SCHDWK
                      SYSSETPRI
                          $SETPRI
```

To verify the check in each case, locate the call to EXE$NAMPID in the code for each service. (Merely understanding the process and perhaps doing it in the case of the SYSPCNTRL module, is sufficient for this exercise.)

SOLUTIONS

g. $HIBER makes no privilege check because a process is only allowed to hibernate itself (not others), although it can be awakened by other processes. This is not mentioned explicitly in the code comments, but could perhaps be deduced from the absence of the privilege check or from the fact that the $HIBER system service does not have any arguments.

AST Delivery

h. Line 173 of SYSPCNTRL invokes SCH$QAST to actually queue the kernel mode AST to the target process. The routine SCH$QAST is located in the module ASTDEL, as indicated in SYS.MAP.

i. Lines 622-644 of module ASTDEL calculate the ASTLVL value and store it. Line 632 extracts the access mode of the first AST in the queue. Line 637 stores the ASTLVL value in the hardware PCB field, while line 638 performs the same operation for the ASTLVL processor register.

j. The AST delivery mechanism begins with an REI instruction detecting the deliverability of an AST and causing a software interrupt at IPL 2. If the process is not interrupted between the queuing of the AST in SCH$QAST and the REI instruction in the SRVEXIT routine, then the first REI instruction encountered will be that one.

EXERCISES

1. Consult your instructor for a list of the crash dump files on your system.

   For each crash dump

   - Determine the current process (and image, if applicable).
   - Determine the current IPL.
   - Determine the reason for the crash. In addition to the reason displayed by SDA, explain why that crash occurred.

Debugging Tools

SOLUTIONS

1. Consult your instructor for the solutions to this exercise.

Scheduling

EXERCISES

1. For each state described below, briefly discuss the properties of a process in the state (for example, memory-resident, or executable), what event or system service placed the process in the state, what system events must occur before the process can leave the present state, and what the next process state can be.

   a. CUR

   b. HIB

   c. SUSPO

   d. CEF

   e. COLPG

   f. PFW

   g. COMO

EXERCISES

2. Assuming the **same** initial conditions (stated below) for **each** question, state

   - What happens to the currently executing process
   - Which process is next selected for execution
   - At what software priority that process executes

   Initial Conditions:

Process Name	Software Priority	Process State
A	5	COM
B	7	LEF
C	17	HIB
D	5	CUR

   a. System event: quantum end for Process D.

   b. System event: post event flag (terminal output completed) for Process B.

   c. System event: scheduled wakeup (from software timer) for Process C.

EXERCISES

3. Describe how processes in the categories below may be included in multiprocess applications. Indicate any possible interactions with system processes that must be considered in assigning processes to these categories and the expected execution behavior of processes in the category.

   a. Time-critical processes

   b. Normal processes with elevated base priorities

   c. Normal processes with normal (default) base priorities

   d. Normal processes with lowered base priorities

Scheduling

SOLUTIONS

1.

a. CUR -- The process is the current executing process and is memory-resident. The state is only entered from the computable, memory-resident state (COM) as a result of a scheduling operation. A process leaves the CUR state as a result of quantum end, process deletion, a wait condition, or preemption by a higher-priority COM process.

b. HIB -- The process is memory-resident, but not computable. The hibernate state is entered by issuing a request to the $HIBER system service (from the CUR state) or requesting the action as part of a create process request ($CREPRC). A process outswapped while hibernating is placed in the HIBO wait state. A process can be made computable (COM) by receiving an AST, a $WAKE request, or a process deletion request.

c. SUSPO -- The process is neither memory-resident nor computable. The state is entered from the CUR state as a result of a $SUSPND system service request, followed at some point by an outswap operation. A process leaves this state only after a $RESUME system service request issued by **another** process, or as a result of a process deletion request. In each case, the process is next placed in the appropriate COMO queue.

d. CEF -- The process is waiting for one or more event flags in a common event flag cluster. Memory-resident and outswapped CEF processes share the same wait state and queue (for a particular common event flag cluster). When the combination of event flags is satisfied, the process is placed into either the computable, resident (COM) or computable, outswapped (COMO) state depending on the memory-resident status bit in the software PCB. The process can also be made computable as a result of AST delivery and process deletion.

e. COLPG -- The process referenced a page already being read into memory as a result of other activity in the system. When the page is available, the process will be made computable or computable outswapped, depending upon its memory-resident status when the page becomes available. AST delivery and process deletion also make COLPG processes computable.

f. PFW -- The process is waiting for a paging operation (page read I/O) to complete. When the page becomes available, the process enters the COM or COMO state, depending upon the memory-resident status. A PFW process can also be made computable as a result of either AST delivery or process deletion.

g. COMO -- The process is computable but not resident in memory. The state may be entered from the various outswapped wait states after any of the system events that make such a process computable. The COMO state is also the initial state of a newly created process. The only transition is to the computable, resident (COM) state after an inswap operation, the event for which the process is waiting.

2.

a. Process D will be rescheduled into the tail of the priority 5 COM state queue. Process A will be scheduled by removing it from the head of the priority 5 state queue and executing it at priority 4.

b. Process D will be rescheduled as in answer a. above. The event flag service will make Process B computable at priority 11 (after the terminal input boost is applied). The scheduler brings Process B into execution at priority 10.

c. Process D will be rescheduled as in answer a. above. Awakening Process C makes it computable at priority 17, and it will be scheduled at priority 17.

Scheduling

SOLUTIONS

3.

a. Time-critical processes are useful for the traditional real-time type of application. They are characterized by fast response times, fixed execution priorities, and invulnerability to quantum end events. For predictable scheduling, time-critical processes should be assigned unique priorities. Otherwise, there is a potential for round robin scheduling of computable real-time processes. In addition, these processes should disable swapping to prevent scheduling conflicts with the swapper, a time-critical process at priority 16.

b. Normal processes with elevated base priorities are characterized by fast response times, but they are susceptible to quantum end events, including the working set adjustment and CPU time expiration operations. As the base priority approaches 15, the current priority level tends to remain more constant than for default processes. Normally, interaction with the system processes (which are mostly implemented as processes of this type) is not a serious concern, because their normal process states are either HIB or LEF. A process such as an active magtape ACP may, however, cause some contention for CPU time.

c. Normal processes with default or normal base priorities typically represent the majority of the processes on a system. The full range of scheduling-related operations apply -- round robin scheduling, dynamic priority recomputation, and quantum end (with working set adjustment and CPU time limit checking). Interactive processes in this category tend to be favored over compute-bound processes because of the priority boost mechanism.

d. Normal processes with lowered base priorities are, effectively, background processes. On a busy system, these processes will only experience occasional scheduling. This category, if used at all, is typically reserved for batch streams, where response time is less critical.

EXERCISES

1. Obtain the following information about the system recorded in the dump file named OSI$LABS:CRASH1.DMP.

   a. Locate the listhead for the HIB state queue.

   (HINTS: Recall there is a system symbol pointing to each state queue. If you do not recall the name of the symbol, you can probably find it in the Symbols Cross-Reference section of the system image map. These symbols begin with the code SCH$.)

   b. How many processes were in the HIB state when the system crashed?

   c. List the software priority (base and current) of each process in the HIB state at the time of the crash.

2. Read the following information on the MWAIT state, and then answer the questions.

   <u>The MWAIT State</u>

   Any process waiting for a mutex or a system resource is placed in the MWAIT (miscellaneous wait) state. There are a few different methods for discovering which mutex or resource the process is waiting for.

   If SHOW SYSTEM lists the process state as RWxxx, then the process is waiting for a resource (xxx represents the desired resource). SHOW SYSTEM displays a mnemonic specifying the specific resource wait, rather than simply notifying you the process is in the MWAIT state. Table 1 lists the RWxxx codes used by SHOW SYSTEM.

   These mnemonics are also used in the MONITOR STATES display to provide you with more information about processes in the MWAIT state.

EXERCISES

When a process is waiting for a resource, a number representing the resource is placed in the EFWM field of the PCB. These numbers are listed with the resource waits in Table 1. VMS defines symbols to represent the resource numbers (in the $RSNDEF macro).

You can use SDA to determine which resource a process is waiting for, but SHOW SYSTEM is usually easier.

Remember The EFWM field normally contains the process event flag wait mask. The multiple use of this field does not cause a conflict, however, because a process in the MWAIT state cannot also be waiting for event flags.

Table 1 Resource Waits

Resource Wait	Mnemonic	Symbol	Numeric
AST Wait (for system AST)	RWAST	RSN$_ASTWAIT	1
Mailbox Full	RWMBX	RSN$_MAILBOX	2
Nonpaged Dynamic Memory	RWNDY	RSN$_NPDYNMEM	3
Page File Full	RWPGF	RSN$_PGFILE	4
Paged Dynamic Memory	RWPDY	RSN$_PGDYNMEM	5
Breakthrough (Wait for broadcast message)	RWBRO	RSN$_BRKTHRU	6
Image Activation Lock	RWIAC	RSN$_IACLOCK	7
Job Pooled Quota (unused)	RWJQO	RSN$_JQUOTA	8
Lock ID Database	RWLKI	RSN$_LOCKID	9
Swap File Space	RWSWP	RSN$_SWPFILE	A
Modified Page List Empty	RWMPE	RSN$_MPLEMPTY	B
Modified Page Writer Busy	RWMPB	RSN$_MPWBUSY	C
System Control Services	RWSCS	RSN$_SCS	D
Cluster State Transition	RWCLU	RSN$_CLUSTRAN	E

If SHOW SYSTEM lists the process state as MUTEX, then the process is waiting for a mutex. In this case, use SDA to determine which mutex. The system virtual address of the particular mutex is in the PCB$L_EFWM field of the software PCB. The symbolic names of these addresses are listed in Table 2.

EXERCISES

Table 2 Mutexes

Mutex	Symbol	Address (*)
Logical Name Table	LNM$AL_MUTEX	
I/O Database	IOC$GL_MUTEX	
(Not used)	CIA$GL_MUTEX	
Common Event Block List	EXE$GL_CEBMTX	
Paged Dynamic Memory	EXE$GL_PGDYNMTX	
Global Section Descriptor List	EXE$GL_GSDMTX	
Shared Memory Global Section Descriptor Table	EXE$GL_SHMGSMTX	
Shared Memory Mailboxes	EXE$GL_SHMMBMTX	
(Not used)	EXE$GL_ENQMTX	
(Not used)	EXE$GL_ACLMTX	
Line Printer Unit Control Block	UCB$L_LP_MUTEX	(**)

(*) See question (2a)

(**) The mutex associated with each line printer unit does not have a fixed address like the other mutexes. Its value depends on where the UCB for that unit is located.

In summary, there are two categories of MWAIT, resource waits and mutex waits. A process is waiting for a mutex if SHOW SYSTEM lists its state as MUTEX, and the PCB$L_EFWM field contains an address greater than 80 million (hex).

A process is in a resource wait if SHOW SYSTEM lists RWxxx as its state, and the PCB$L_EFWM field contains a small number representing the particular resource.

a. Determine the system virtual addresses of the mutexes listed in Table 2. Add them to the table.

   (HINT: you can find these values in SYS$SYSTEM:SYS.MAP)

EXERCISES

b. A process on your system named GONZO seems to be 'hung'. The display from SHOW SYSTEM tells you that its state is RWAST, which you know is a subdivision of the MWAIT state.

Analyze the resulting crash dump in OSI$LABS:MWAIT.DMP to verify that GONZO was

- In the MWAIT state
- Waiting for an AST

Scheduling

SOLUTIONS

1.

a. The listhead for the HIB wait state queue is at location SCH$GQ_HIBWQ.

b. The count of processes in the HIB state is stored at offset WQH$W_WQCNT in the wait queue listhead. (The $WQHDEF macro is in SYS$LIBRARY:LIB.MLB.)

   On most systems, the following processes are often in the HIB state: SWAPPER, ERRFMT, JOB_CONTROL, and REMACP and NETACP if DECnet is installed.

c. To find the software priority (base and current) of each process in the HIB state, trace through the the software PCBs in the queue.

   The base priority is at offset PCB$B_PRIB, and the current priority is at offset PCB$B_PRI.

2.

a. The system virtual addresses of the mutexes can be determined by examining the output produced by the following DCL commands:

```
$ SEARCH  SYS$SYSTEM:SYS.MAP  MTX
$ SEARCH  SYS$SYSTEM:SYS.MAP  MUTEX
```

b. The PCB$W_STATE field of GONZO's software PCB contains the value 2 (SCH$C_MWAIT) which means that GONZO was in the MWAIT state.

   The PCB$L_EFWM field contains a 1, which means that GONZO was waiting for a resource. The resource was an AST (see Table 1).

Process Creation and Deletion

EXERCISES

1. List two advantages to performing process deletion in the context of the process being deleted.

2. Name two errors that can result from process creation. One of the errors should be returned from the $CREPRC system service request and the other only through a termination mailbox. Explain why the $CREPRC system service is not capable of detecting the second type of error.

3. Explain why a process with a CLI mapped in is not deleted when an image exits.

Process Creation and Deletion

SOLUTIONS

1. When executing in the context of the process being deleted, all the virtual address space of that process is accessible. In particular, the contents of the control region (P1 space) that describe the state of the process at the time of deletion is readily available.

   In addition, the full support of VAX/VMS (including RMS and all the system services) is available to aid in the process deletion. Much of this support is not available to code executing outside of process context.

2. The complete list of errors that can be detected by the $CREPRC system service is listed in the description of $CREPRC in the VAX/VMS System Services Reference Manual. Possible errors include privilege violation, insufficient quota, and process name errors.

   Several errors can be detected only when the newly created process executes. These errors include the specification of an image that does not exist or bad equivalence strings for SYS$INPUT, SYS$OUTPUT, or SYS$ERROR.

   By the time the new process is placed into execution, the $CREPRC system service has already completed its work for the creator and returned a status code. All errors that cannot be detected except in the context of the newly created process can only be reported to the creator through a termination mailbox.

3. Image exit results in all previously declared termination handlers being called. The command language interpreter has declared a handler that runs the image down (if necessary), restores the supervisor stack to its state before the image was initially called, and looks for the next command from SYS$INPUT. This allows multiple images to execute sequentially in the same process. Only a special action, such as a LOGOUT command within the process, or an external STOP/ID= command, can cause such a process to be deleted.

Process Creation and Deletion

EXERCISES

1. Write a program that will:

   a. Prompt the user for a Process ID.

   b. Use a routine (or routines) in the SYSPCNTRL module of VMS to locate the software PCB for the specified process.

   c. Display the event flag wait mask and current priority of the process.

   Things to remember when writing your program:

   - Read through the routine(s) in SYSPCNTRL that you will call. Note the inputs and outputs, calling sequence, environment (access mode, IPL) and side effects of the routine(s).

   - Remember that the software priority of a process is stored in the software PCB as 31 minus the priority (to simplify the scheduler code).

   Run the program to gather the information about your process and some of the system processes (ERRFMT, OPCOM, etc.). Compare the software priorities provided by your program with those listed by SHOW SYSTEM.

2.

   a. Write a program to output, and then change, your account name. This must be done in elevated access mode. (Your account name is stored in your P1 space.)

   b. Use the system dump analyzer (SDA) on the current system to verify that you have changed your account name.

      You may also want to log out after changing the account name, then log in again and enter:

      $ ACCOUNTING/FULL/ACCOUNT=new-name

      You should see an accounting record that has your CHANGED account name.

SOLUTIONS

1. The program in Example 1 uses EXE$EPID_TO_PCB (in VMS module SYSPCNTRL) to locate a software PCB. It then displays the event flag wait mask and current priority of the process.

```
        .TITLE   PCDLAB1                        ; for process cre/delete
;++
; ABSTRACT:
;
;  This program accepts a PID and displays the event flag
;  wait mask and current priority of the specified process.
;  It uses EXE$EPID_TO_PCB to locate the PCB.
;
; ENVIRONMENT:
;  Begins execution in user mode, changes mode to kernel.
;  Raises IPL to IPL$_SYNCH to synchronize.
;  Requires: CMKRNL privilege; link with SYS.STB
;
; SIDE EFFECTS:
;  none known
;--
        .LIBRARY   /OSI$LABS:OSIMACROS/        ; for I/O
        .LIBRARY   /SYS$LIBRARY:LIB/           ; system def's
        $IPLDEF                                ; IPL symbol def's
        $PCBDEF                                ; pcb offsets
;
; *********************** data ******************************
        .PSECT     NOSHARED_DATA   PIC, NOEXE, LONG
PID_ASC: .LONG  8
        .ADDRESS       ASC_BUF
ASC_BUF: .BLKB  8
EFWM_ASC:.LONG  8
        .ADDRESS       EFWM_BUF
EFWM_BUF: .BLKB   8
CURPRI_ASC:
        .LONG  8
        .ADDRESS       CURPRI_BUF
CURPRI_BUF:
        .BLKB   8
BIG_STRING:
        .LONG   80
        .ADDRESS       BYTES
BYTES:  .BLKB   80
```

Example 1 Program to Locate and Read PCB
(Sheet 1 of 3)

SOLUTIONS

```
PROMPT: .ASCID  /Enter a Process ID (all 8 digits):  /
HDR1:   .ASCID  /Event Flag Wait Mask is: /
HDR2:   .ASCID  /Current Priority is:     /
ERRMSG: .ASCID  /Error finding PCB./
K_ARG_LIST:                                     ; for $CMKRNL call
        .LONG       3
PROCESS_ID:
        .LONG       0.                          ; passed by value
        .ADDRESS    EFWM                ; passed by reference
        .ADDRESS    CURPRI              ; passed by reference
EFWM:      .LONG    0
CURPRI: .LONG       0
; *********************** main code ***************************
        .PSECT      CODE                EXE,NOWRT,PIC,SHR
        .ENTRY      BEGIN    ^M<>
;
        PUSHAL  PROMPT
        PUSHAL  PID_ASC
        CALLS   #2, G^LIB$GET_INPUT
        CHECK_STATUS
        CONV_HEX_BIN  PID_ASC, PROCESS_ID

;       Invoke kernel mode routine.  It returns EFWM and
;       current priority.  EFWM remains = 0 if any errors.
        $CMKRNL_S  routin= KERNEL1, arglst= K_ARG_LIST
        CHECK_STATUS

30$:       TSTL     EFWM                ; error finding pcb?
        BNEQ        40$                 ; (BEQL 63$: will not reach)
        BRW         63$                 ; if yes, branch to error rtn

40$:       CONV_BIN_HEX     EFWM, EFWM_ASC
        CONCAT2         BIG_STRING, HDR1, EFWM_ASC
        DISPLAY         BIG_STRING

;  adjust priority from internal format
        SUBB3       CURPRI, #31, CURPRI
        CONV_BIN_HEX    CURPRI, CURPRI_ASC
        CONCAT2         BIG_STRING, HDR2, CURPRI_ASC
        DISPLAY         BIG_STRING
```

Example 1 Program to Locate and Read PCB
(Sheet 2 of 3)

SOLUTIONS

```
50$:        MOVL     #SS$_NORMAL, R0
        RET

;  error routines
63$:        DISPLAY         ERRMSG
        BRW       50$

; *********************** kernel mode code ********************
;  returns EFWM and current priority
        .ENTRY     KERNEL1          ^M<R5,R6>

;       get input argument (PID) off user stack before raise IPL
        MOVL       4(AP), R0                   ; PID is first argument
        CLRL    R6                  ; cuz we only move a byte into it
;
;       save old IPL on stack, raise IPL.  Reference SYNCH
;       variable to lock down elevated IPL code.
        DSBINT     SYNCH

;  PID is in R0 (required by epid routine), jsb to EPID_TO_PCB
        JSB        G^EXE$EPID_TO_PCB         ; returns PCB addr. in R0
        BEQL       140$                      ;  and sets cond. codes

        MOVL       PCB$L_EFWM(R0), R5        ; save EFWM for main code
        MOVB       PCB$B_PRI(R0), R6         ; save current priority

   ENBINT                               ; IPL back to zero
;       can touch the user stack now because back at IPL 0
        MOVL    R5, @8(AP)                  ; store EFWM in arg list
        MOVL    R6, @12(AP)                 ; store cur. pri in arg list
;       branch here if could not find PCB.  Leave zeros in arg list
140$:   SETIPL  #0
        MOVL       #SS$_NORMAL, R0
        RET                                  ; all done in kernel mode

SYNCH:  .LONG      IPL$_SYNCH
        .END       BEGIN
```

Example 1 Program to Locate and Read PCB
(Sheet 3 of 3)

SOLUTIONS

2. The program in Example 2 displays and changes the account name for the process.

```
   .TITLE  PCDLAB2
;++
; ABSTRACT:
;       Program to change P1 control information (account name)
;
; ENVIRONMENT:
;       Changes mode to exec to read P1 space, and to kernel
;       to write P1 space.
;
;       Linked with SYS.STB:
;       $ LINK PCDLAB2, SYS$SYSTEM:SYS.STB/SELECTIVE
;
; SIDE EFFECTS:
;       Process account name is changed.
;--
        .MACRO  CHECK_STATUS    CODE=R0, ?GO
        BLBS    R0, GO
        PUSHL   R0
        CALLS   #1, G^LIB$STOP
        RET
GO:
        .ENDM   CHECK_STATUS

; ************************** data ****************************
        .PSECT  NOSHARED_DATA   PIC,NOEXE,LONG

MESS1:  .ASCID  /Account name: /
PROMPT: .ASCID  /Enter account name (1-8 characters): /
ACC_NAME:                               ; descriptor for
        .LONG   8                       ;  account name
        .ADDRESS        ACC_BUF
ACC_BUF:
        .BLKB   8
E_ARG_LIST:                             ; argument list for CHME
        .LONG 1
        .ADDRESS        ACC_BUF
```

Example 2 Program to Display and Change Account Name (Sheet 1 of 3)

SOLUTIONS

```
K_ARG_LIST:                                  ; argument list for CHMK
        .LONG 2
        .ADDRESS        ACC_BUF
        .ADDRESS        LENGTH
BUFFER: .LONG   80                           ; descriptor for
        .ADDRESS        BUF                  ;  string concats
BUF:    .BLKB   80
LENGTH: .BLKW   1                            ; storage for prompt

; ************************** code ******************************
        .PSECT  CODE    EXE,NOWRT,PIC,SHR
        .ENTRY  ACCNAME         ^M<>
;       change mode to executive to read account name in P1 space
        $CMEXEC_S       routin=EXEC_RTN, arglst=E_ARG_LIST
        CHECK_STATUS

        PUSHAL  ACC_NAME
        PUSHAL  MESS1
        PUSHAL  BUFFER
        CALLS   #3, G^STR$CONCAT            ; put string together
        CHECK_STATUS

        PUSHAL  BUFFER
        CALLS   #1, G^LIB$PUT_OUTPUT        ; ...and show it
        CHECK_STATUS

        PUSHAW  LENGTH                      ; prompt for "new"
        PUSHAL  PROMPT                      ;  account name
        PUSHAL  ACC_NAME
        CALLS   #3, G^LIB$GET_INPUT
        CHECK_STATUS
```

Example 2 Program to Display and Change Account Name
(Sheet 2 of 3)

```
;       change mode to kernel to write new account name
        $CMKRNL_S       routin=KERNEL_RTN, arglst=K_ARG_LIST
        CHECK_STATUS
        MOVL    #SS$_NORMAL, R0
        RET

; *******************  exec mode code  ***********************
        .ENTRY  EXEC_RTN        ^M<R2,R3,R4,R5>
;       save r2-r5 because destroyed by MOVC

;       put account name from Pl in argument list
        MOVC3   #8, G^CTL$T_ACCOUNT, @4(AP)
        MOVL    #SS$_NORMAL, R0             ; set normal completion
        RET

; *******************  kernel mode code  ***********************
        .ENTRY  KERNEL_RTN      ^M<R2,R3,R4,R5>
;       save r2-r5 because destroyed by MOVC

        MOVC5   @8(AP), @4(AP),     -; src len and addr in arglst
                #^A/ /,             -; fill with blanks
                #8, G^CTL$T_ACCOUNT  ; dest is 8 bytes in Pl
        MOVL    #SS$_NORMAL, R0             ; set normal completion
        RET

        .END    ACCNAME
```

Example 2 Program to Display and Change Account Name
(Sheet 3 of 3)

EXERCISES

Differentiate the two programs SYSBOOT and SYSGEN, including their

- Purposes
- Environments
- Command syntax

SOLUTIONS

SYSBOOT

- **Purpose:** SYSBOOT is the program that performs the secondary phase of the bootstrap sequence. It reads parameters from the system image and, optionally, from a parameter file. All adjustable parameters are calculated. The system page table is set up. The system image is read into memory.

  SYSBOOT is **not** involved in determining which devices are present or in loading the drivers and associated data structures for these devices.

- **Environment:** SYSBOOT executes in a stand-alone environment with memory management turned off. All communication with the console terminal and all file operations must be performed by code contained in the SYSBOOT image, because there is no RMS or ACP to provide these services.

- **Command Syntax:** SYSBOOT does **not** recognize those commands associated with loading device drivers. The WRITE command is also ignored by SYSBOOT.

  SYSBOOT begins its operation by reading the values of adjustable parameters from the system image file. This is an implied USE CURRENT command.

SYSGEN

- **Purpose:** SYSGEN is not directly involved in the bootstrap operation. Its primary purpose is to create a parameter file that will be used by SYSBOOT during future bootstrap operations.

  SYSGEN also loads device drivers for all devices that it finds on the system or in response to explicit commands. The data structures required by the driver are allocated and initialized by SYSGEN.

- **Environment:** SYSGEN is a normal image that executes in full process context. This means that services of the VAX/VMS operating system are available for file operations including terminal communication.

SOLUTIONS

- **Command Syntax:** All commands can be performed by SYSGEN. However, SET commands do **not** normally affect the current system, but merely change the values in a table that will be written to a parameter file. A WRITE CURRENT command will establish the parameter values used in the **next** system initialization. A WRITE ACTIVE command can change the values of dynamic system parameters on the running system.

Tests

VMS Internals I

PRE-TEST

Circle the letter that best answers each of the following questions.

1. Which utility is used to make shareable files available to all users?

   a. SYSGEN
   b. SDA
   c. SYE
   d. INSTALL

2. If you have an existing file, and would like to produce a statistical report summarizing file characteristics, which RMS utility would you use?

   a. CREATE/FDL
   b. EDIT/FDL
   c. CONVERT
   d. ANALYZE/RMS_FILE

3. Which address region contains the user stack?

   a. Program region (P0)
   b. Control region (P1)
   c. System region (S0)
   d. Reserved region (S1)

4. If, after calling a system service, the status code equals one, the system service has completed:

   a. With a warning
   b. Successfully
   c. With an error
   d. With a severe error

5. Which of the following must be done before an I/O operation can be requested on a device?

   a. The device must be allocated
   b. The device must be mounted
   c. A channel must be assigned to the device
   d. The device must be initialized

PRE-TEST

6. Which of the following is the fastest interprocess communication mechanism?

   a. Mailbox
   b. Global section
   c. DECnet
   d. Shared file

7. Which of the following is true for a hibernating process, but not true for a suspended process?

   a. ASTs can be queued
   b. ASTs can be delivered
   c. ASTs are disabled
   d. ASTs cannot awaken main-line code

8. What type of condition occurs as the result of an external hardware event?

   a. Exception
   b. Interrupt
   c. Trap
   d. Fault

9. Which condition handler is looked for first when an exception occurs?

   a. Primary handler
   b. Secondary handler
   c. User-defined handler in current call frame
   d. Last chance handler

10. In designing an application interface, VMS provides assistance in implementing which of the following features?

    a. A HELP facility
    b. Application-specific error messages
    c. Parsing user input
    d. All of the above

PRE-TEST

11. The linker places information into an executable or shareable image file for later use by:

    a. A compiler or assembler
    b. The image activator
    c. The scheduler
    d. The disk ACP

12. A MAP file is produced by:

    a. An assembler or compiler
    b. The linker
    c. The librarian
    d. The Message utility

13. Which of the following types of files can be used to group image sections into clusters?

    a. Options file
    b. Library file
    c. Shared image file
    d. Transfer vector file

14. Which utility can be used to determine the cause of an operating system failure, and also to examine the characteristics of the currently executing process?

    a. SDA
    b. Accounting
    c. Monitor
    d. SPM

15. To decrease paging activity, system services can be used to:

    a. Adjust the size of the working set
    b. Lock pages in the working set and/or in physical memory
    c. Disable the swapping of a process
    d. All of the above

16. In the following instruction, which of the operands is in Register Deferred mode?

```
ADDL3   #100, (R3), SUMS
```

a. #100
b. (R3)
c. SUMS
d. None of the above.

17. What would be the contents of the destination after the execution of the MOVW (R5)+, R3 instruction?

where R5 = 0600
R3 = 3F90
0600 = 0700
0602 = 0702

a. 0600
b. 0602
c. 0700
d. 0702

18. What would be the contents of the source after the execution of the MOVW (R5)+, R3 instruction?

where R5 = 0600
R3 = 3F90
0600 = 0700
0602 = 0702

a. 0600
b. 0602
c. 0700
d. 0702

PRE-TEST

19. What hexadecimal value will be in R3 after the MOVL #^B1011, R3 instruction executes?

    a. A
    b. B
    c. C
    d. D

20. Which instruction is used to divide the longword QUARTS by 4, placing the result in the longword GALLONS?

    a. DIVL #4,QUARTS,GALLONS
    b. DIVL3 4,QUARTS,GALLONS
    c. DIVL3 #4,QUARTS,GALLONS
    d. DIVL3 #4,GALLONS,QUARTS

21. Which register mask saves R2 and R5 on the stack?

    a. PUSHR #^M<R2,R5>
    b. PUSH #^M<R2,R5>
    c. PUSHL #^M<R2,R5>

22. Which set of instructions are used to invoke a subroutine?

    a. CALLS, CALLG, RET
    b. CALLS, CALLG, RSB
    c. JSB, BSBx, RET
    d. JSB, BSBx, RSB

23. The Command Language Interpreter runs primarily in what mode?

    a. Kernel mode
    b. Executive mode
    c. Supevisor mode
    d. User mode

24. What is the name of the control block created when the Save Process Context instruction is executed?

    a. Software PCB
    b. Hardware PCB
    c. Process header
    d. AST control block

25. When a typical user logs in to a VAX system, what kind of process is created?

    a. Owner process
    b. Detached process
    c. Subprocess
    d. Privileged process

VMS Internals I

SOLUTIONS TO PRE-TEST

Circle the letter that best answers each of the following questions.

1. Which utility is used to make shareable files available to all users?

    a. SYSGEN
    b. SDA
    c. SYE
    (d.) INSTALL

2. If you have an existing file, and would like to produce a statistical report summarizing file characteristics, which RMS utility would you use?

    a. CREATE/FDL
    b. EDIT/FDL
    c. CONVERT
    (d.) ANALYZE/RMS_FILE

3. Which address region contains the user stack?

    a. Program region (P0)
    (b.) Control region (P1)
    c. System region (S0)
    d. Reserved region (S1)

4. If, after calling a system service, the status code equals one, the system service has completed:

    a. With a warning
    (b.) Successfully
    c. With an error
    d. With a severe error

5. Which of the following must be done before an I/O operation can be requested on a device?

    a. The device must be allocated
    b. The device must be mounted
    (c.) A channel must be assigned to the device
    d. The device must be initialized

SOLUTIONS TO PRE-TEST

6. Which of the following is the fastest interprocess communication mechanism?

   a. Mailbox
   (b.) Global section
   c. DECnet
   d. Shared file

7. Which of the following is true for a hibernating process, but not true for a suspended process?

   a. ASTs can be queued
   (b.) ASTs can be delivered
   c. ASTs are disabled
   d. ASTs cannot awaken main-line code

8. What type of condition occurs as the result of an external hardware event?

   a. Exception
   (b.) Interrupt
   c. Trap
   d. Fault

9. Which condition handler is looked for first when an exception occurs?

   (a.) Primary handler
   b. Secondary handler
   c. User-defined handler in current call frame
   d. Last-chance handler

10. In designing an application interface, VMS provides assistance in implementing which of the following features?

   a. A HELP facility
   b. Application-specific error messages
   c. Parsing user input
   (d.) All of the above

SOLUTIONS TO PRE-TEST

11. The linker places information into an executable or shareable image file for later use by:

    a. A compiler or assembler
    (b.) The image activator
    c. The scheduler
    d. The disk ACP

12. A MAP file is produced by:

    a. An assembler or compiler
    (b.) The linker
    c. The librarian
    d. The Message utility

13. Which of the following types of files can be used to group image sections into clusters?

    (a.) Options file
    b. Library file
    c. Shared image file
    d. Transfer vector file

14. Which utility can be used to determine the cause of an operating system failure, and also to examine the characteristics of the currently executing process?

    (a.) SDA
    b. Accounting
    c. Monitor
    d. SPM

15. To decrease paging activity, system services can be used to:

    a. Adjust the size of the working set
    b. Lock pages in the working set and/or in physical memory
    c. Disable the swapping of a process
    (d.) All of the above

SOLUTIONS TO PRE-TEST

16. In the following instruction, which of the operands is in Register Deferred mode?

```
ADDL3   #100, (R3), SUMS
```

a. #100
(b.) (R3)
c. SUMS
d. None of the above.

17. What would be the contents of the destination after the execution of the MOVW (R5)+, R3 instruction?

```
where  R5 = 0600
       R3 = 3F90
       0600 = 0700
       0602 = 0702
```

a. 0600
b. 0602
(c.) 0700
d. 0702

18. What would be the contents of the source after the execution of the MOVW (R5)+, R3 instruction?

```
where R5 = 0600
      R3 = 3F90
      0600 = 0700
      0602 = 0702
```

a. 0600
(b.) 0602
c. 0700
d. 0702

SOLUTIONS TO PRE-TEST

19. What hexadecimal value will be in R3 after the MOVL #^B1011, R3 instruction executes?

    a. A
    (b.) B
    c. C
    d. D

20. Which instruction is used to divide the longword QUARTS by 4, placing the result in the longword GALLONS?

    a. DIVL #4,QUARTS,GALLONS
    b. DIVL3 4,QUARTS,GALLONS
    (c.) DIVL3 #4,QUARTS,GALLONS
    d. DIVL3 #4,GALLONS,QUARTS

21. Which register mask saves R2 and R5 on the stack?

    (a.) PUSHR #^M<R2,R5>
    b. PUSH #^M<R2,R5>
    c. PUSHL #^M<R2,R5>

22. Which set of instructions are used to invoke a subroutine?

    a. CALLS, CALLG, RET
    b. CALLS, CALLG, RSB
    c. JSB, BSBx, RET
    (d.) JSB, BSBx, RSB

23. The Command Language Interpreter runs primarily in what mode?

    a. Kernel mode
    b. Executive mode
    (c.) Supevisor mode
    d. User mode

SOLUTIONS TO PRE-TEST

24. What is the name of the control block created when the Save Process Context instruction is executed?

    a. Software PCB
    (b.) Hardware PCB
    c. Process header
    d. AST control block

25. When a typical user logs in to a VAX system, what kind of process is created?

    a. Owner process
    (b.) Detached process
    c. Subprocess
    d. Privileged process