EY-0016E-SG-0001

VMS Internals

digital

EY-0016E-SG-0001

VMS Internals

Student Workbook

Prepared by Educational Services
of
Digital Equipment Corporation

Printed in U.S.A.

CONTENTS

SG STUDENT GUIDE

1 SYSTEM COMPONENTS

4 DEBUGGING TOOLS

5 SCHEDULING

6 PAGING

7 SWAPPING

8 PROCESS CREATION AND DELETION

9 SYSTEM INITIALIZATION AND SHUTDOWN

FIGURES

TABLES

EXAMPLES

SG

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 ("tuning" or "SYSGEN" 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 (e.g., 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. Specifically, the implementation of I/O and the method of writing a device driver for VAX/VMS are not discussed in this course.

COURSE GOALS

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

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

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

   - assigning priorities in a multi-process application,
   - controlling paging and swapping behavior for a process or an entire system, and
   - writing and installing a site-specific system service.

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

5. Describe the data structures and software components used when a process is created or deleted, an image is activated and terminated, and the operating system is initialized.

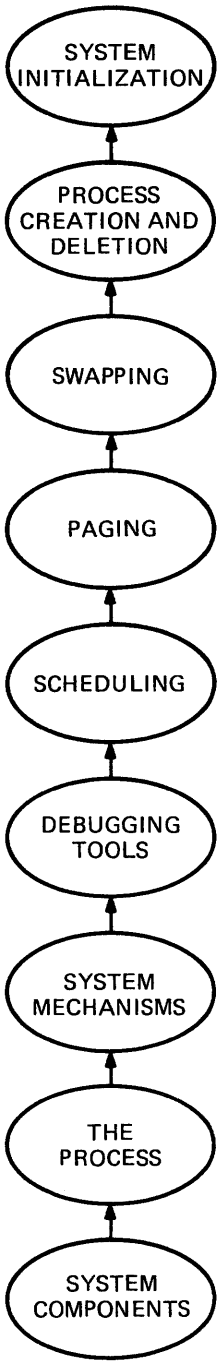
6. Describe how the following interrupt service routines are implemented:

   - AST delivery
   - Scheduling
   - Hardware clock
   - Software timers

RESOURCES

1. VAX/VMS Internals and Data Structures Manuals
2. VAX/VMS System Dump Analyzer Reference Manual
3. VAX/VMS Operating System Internals Supplemental Listings

COURSE MAP



TK-9007

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
6. DECNET communication

F. Examples of System processes

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

II. The Process

A. Overview - process data structures

1. Software context information
2. Hardware context information

B. Overview - virtual address space

1. SØ space (operating system code and data)
2. PØ space (user image code and data)
3. Pl space (command language interpreter, process data)

III. System Mechanisms

A. Processor and process state

B. Interrupts

C. Access modes and exceptions

D. Synchronization

IV. Debugging Tools

A. Crash dumps and bugchecks

B. The system map file (SYS.MAP)

C. The system dump analyzer (SDA)

D. Other debugging tools (DELTA, XDELTA, CCL, PATCH)

V. Scheduling

A. Process states

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

B. How process states are implemented in data structures

1. Queues
2. Process data structures

C. Operating system code that implements process state changes

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

D. Steps at quantum end

1. Automatic working set adjustment

E. Boosting software priority of normal processes

VI. Paging

A. Linker action in creating executable files

B. Image activator for setting up process header

C. Invoking pager routine

D. Memory management data structures

E. Following a process page faulted in and out of a process

F. Following a global page faulted in and out of a process

VII. Swapping

A. Comparison of paging and swapping

B. Swapper functions

1. Maintain free page count
   - Write modified pages to paging file
   - Shrink working sets
2. Outswap - rules and example
3. Inswap - rules and example

C. Selected events that wake swapper

D. Locating disk files for swap

E. How swapper's PØ page table is used to speed disk I/O

VIII. Process Creation and Deletion

A. Process creation and deletion

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

B. Initiating jobs

1. Interactive
2. Batch

C. DCL structure and function

D. Image activation and rundown

1. Mapping image file
2. Image startup
3. Termination handlers

IX. System Initialization and Shutdown

A. System initialization sequence

B. Functions of initialization programs

C. How memory is structured and loaded

D. Startup command procedures

E. Hardware differences in the 78Ø, 75Ø, and 73Ø and how they affect initialization

F. Front panel switches

G. Shutdown procedures and their functions

H. Auto-restart sequence

I. Power-fail recovery

1

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 may 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 residence characteristics (can be paged, swapped or shared).

OBJECTIVES

Upon completion of this module, for each selected VAX/VMS software component, you will be able to 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 Manual, overview

Additional Suggested Reading

- VAX/VMS Internals and Data Structures Manual, Chapters on I/O System Services, interactive and batch jobs, and miscellaneous system services.

Source Modules

Facility Name

SYS
DCL,MCR,CLIUTL
RSX
DEBUG
RTL
RMS
F11A,F11B,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

   F. DECNET communication

VI. Examples of System Processes

   A. Operator Communication (OPCOM)

   B. Error logger (ERRFMT)

   C. Job controller (JOB_CONTROL)

   D. Symbionts (PRTSYMBn)

THREE MAIN PARTS OF VMS

Scheduling and Process Control (IPL 3)

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
- Hardware clock and software timer interrupt service routines
- 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)

INVOKING SYSTEM CODE

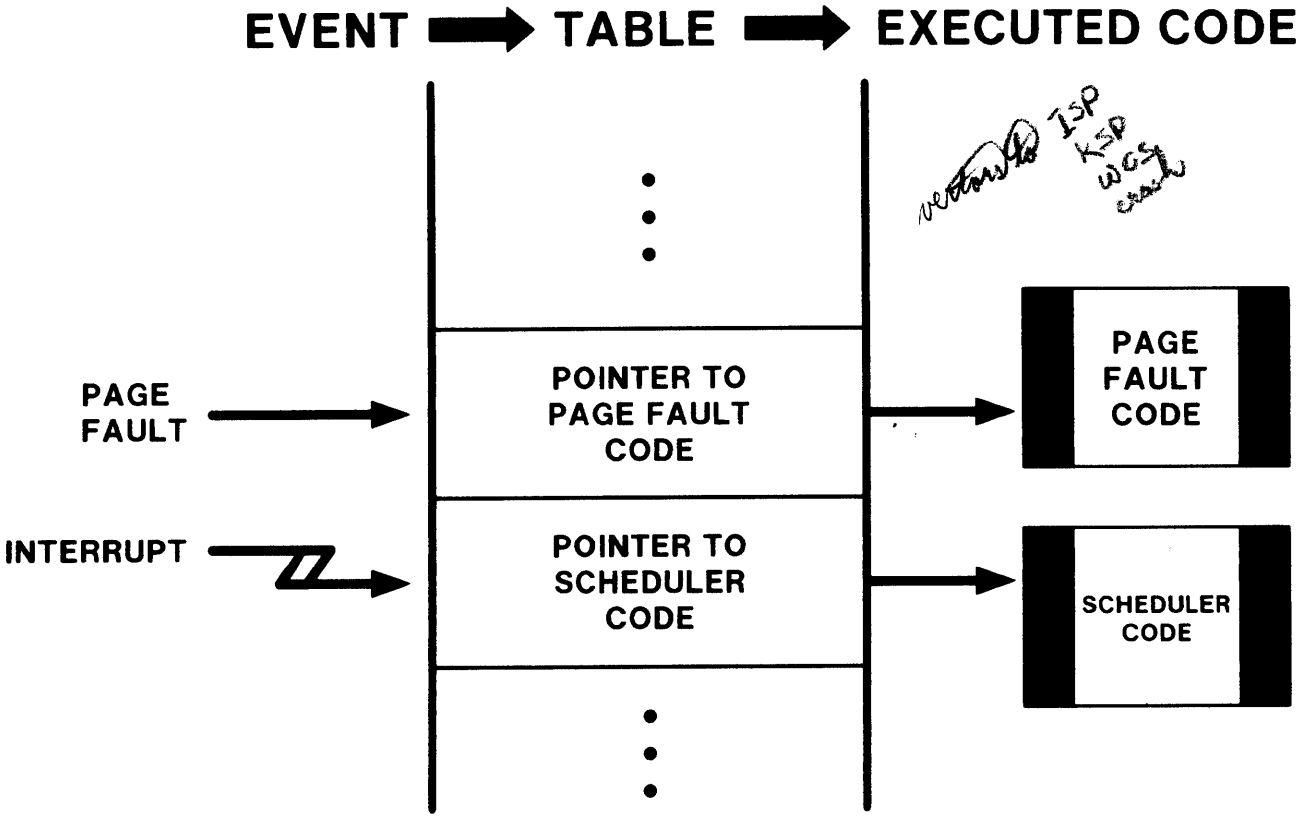


Figure 1-1 Invoking System Code

- 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

HARDWARE MAINTAINED PRIORITY LEVELS

- Processor is always operating at one of 32 possible hardware maintained priority levels (Ø - 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 Ø 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).

INTERRUPT SERVICING SEQUENCE

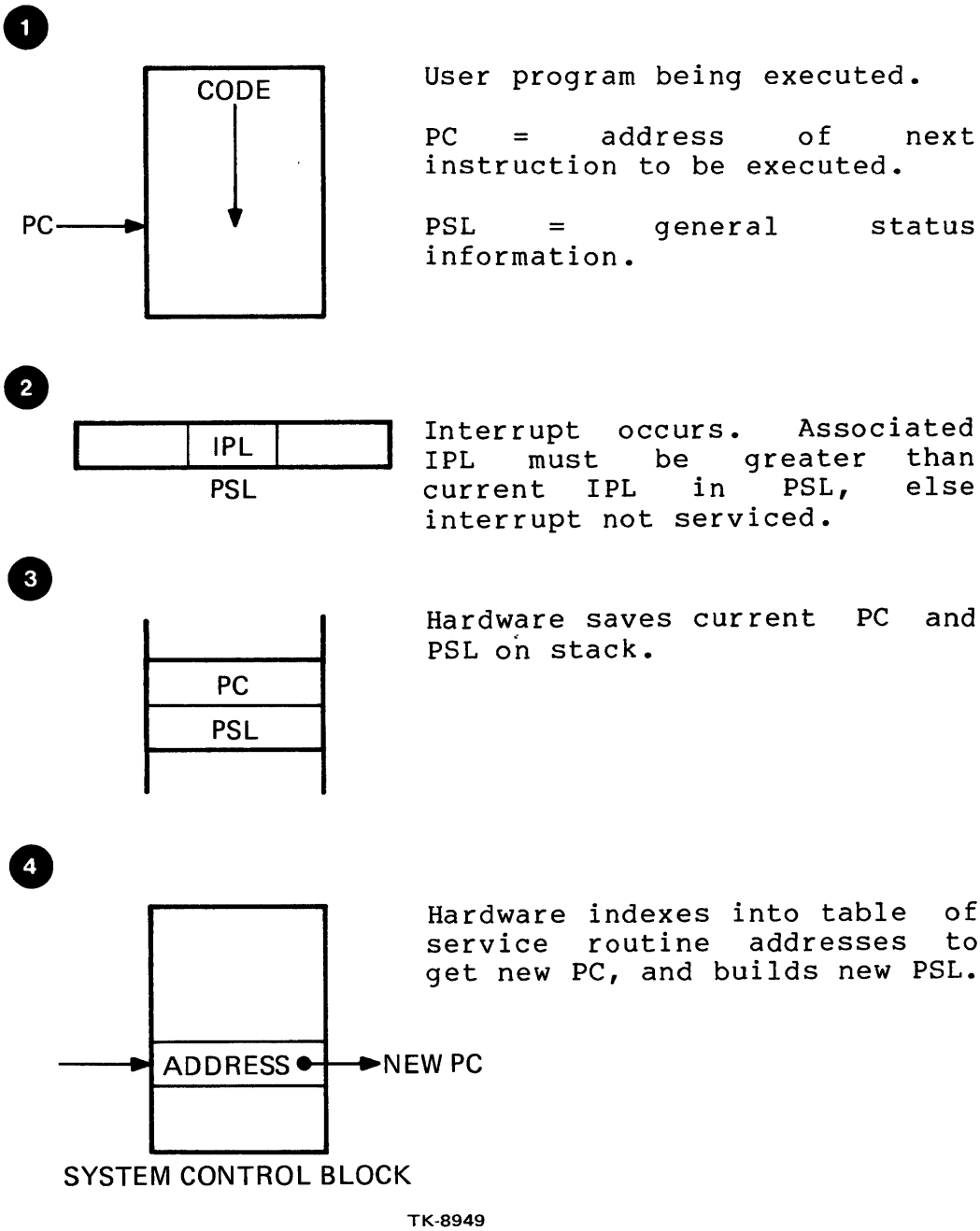
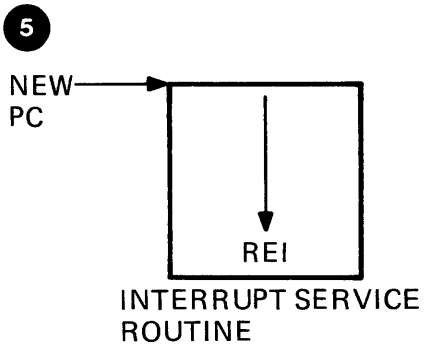
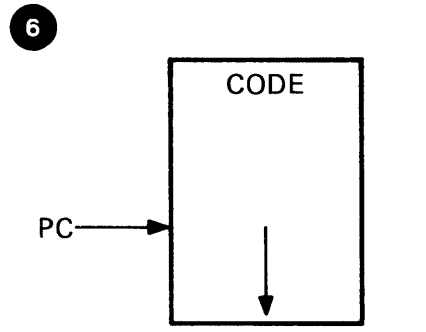


Figure 1-2 Example of Interrupt Servicing
(Sheet 1 of 2)



Interrupt service routine executes at new IPL.

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



REI

- Pops PC, PSL from stack
- Checks PSL
- Moves PC, PSL to CPU registers
- Transfers control to PC

Interrupted program continues execution.

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

TWO TYPES OF PRIORITY

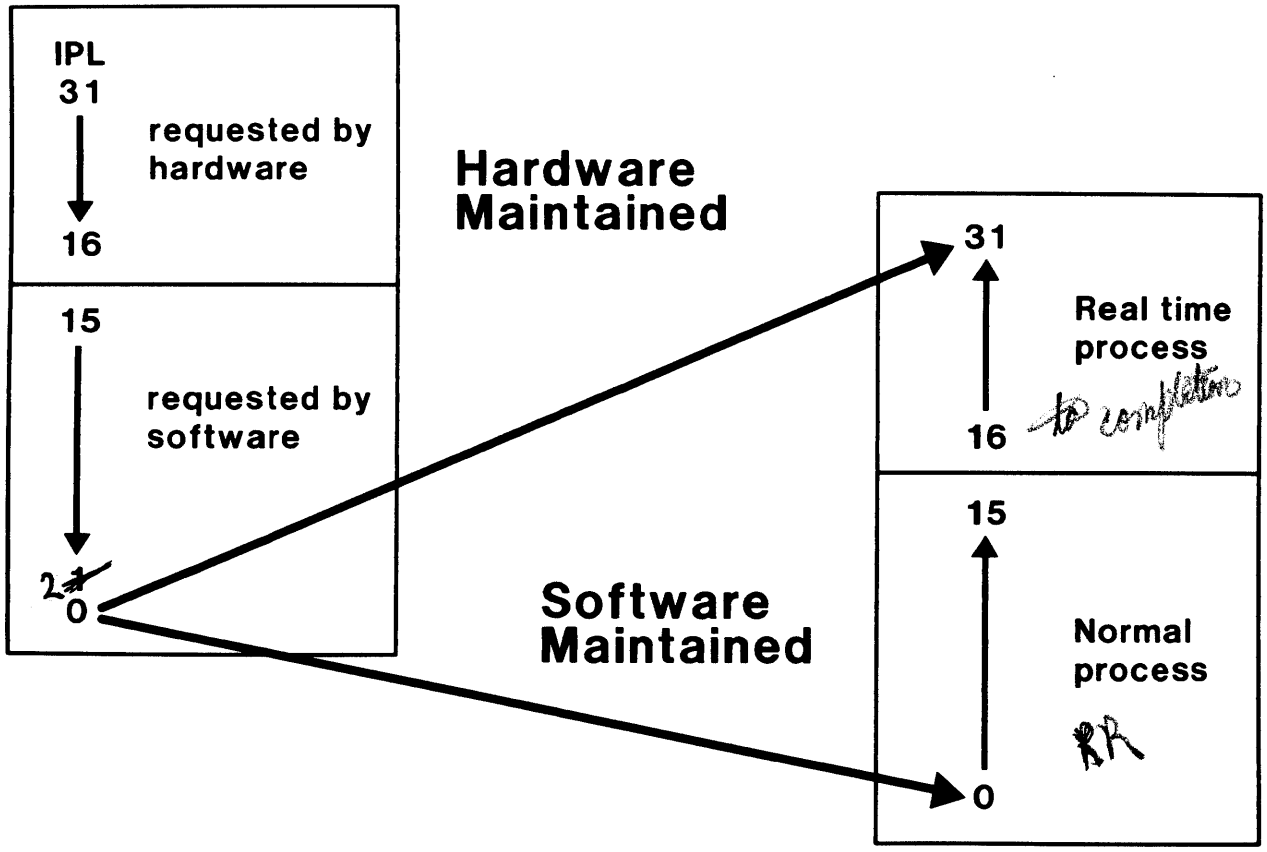


Figure 1-3 Two Types of Priority

dispatcher runs @ IPL 3

ACCESS MODES AND COMPONENTS

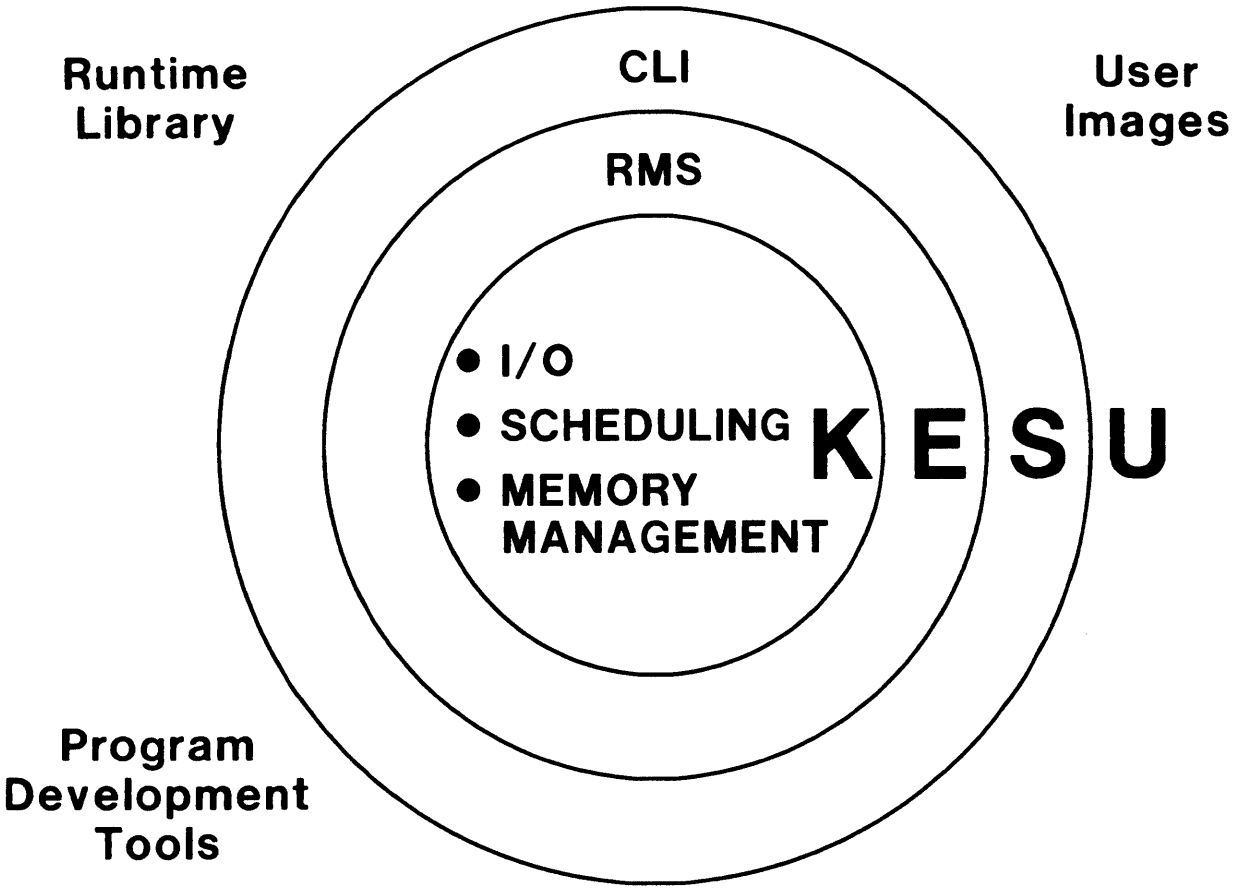


Figure 1-4 Access Modes and Components

1. The kernel of the operating system (shown here in the innermost circle) is protected from the user by several layers of access protection.

2. The user normally accesses this protected code and data through the Command Language Interpreter (CLI), Record Management Services (RMS), and the system services.

3. System services are routines in the operating system kernel which may be called by the user via a well defined interface.

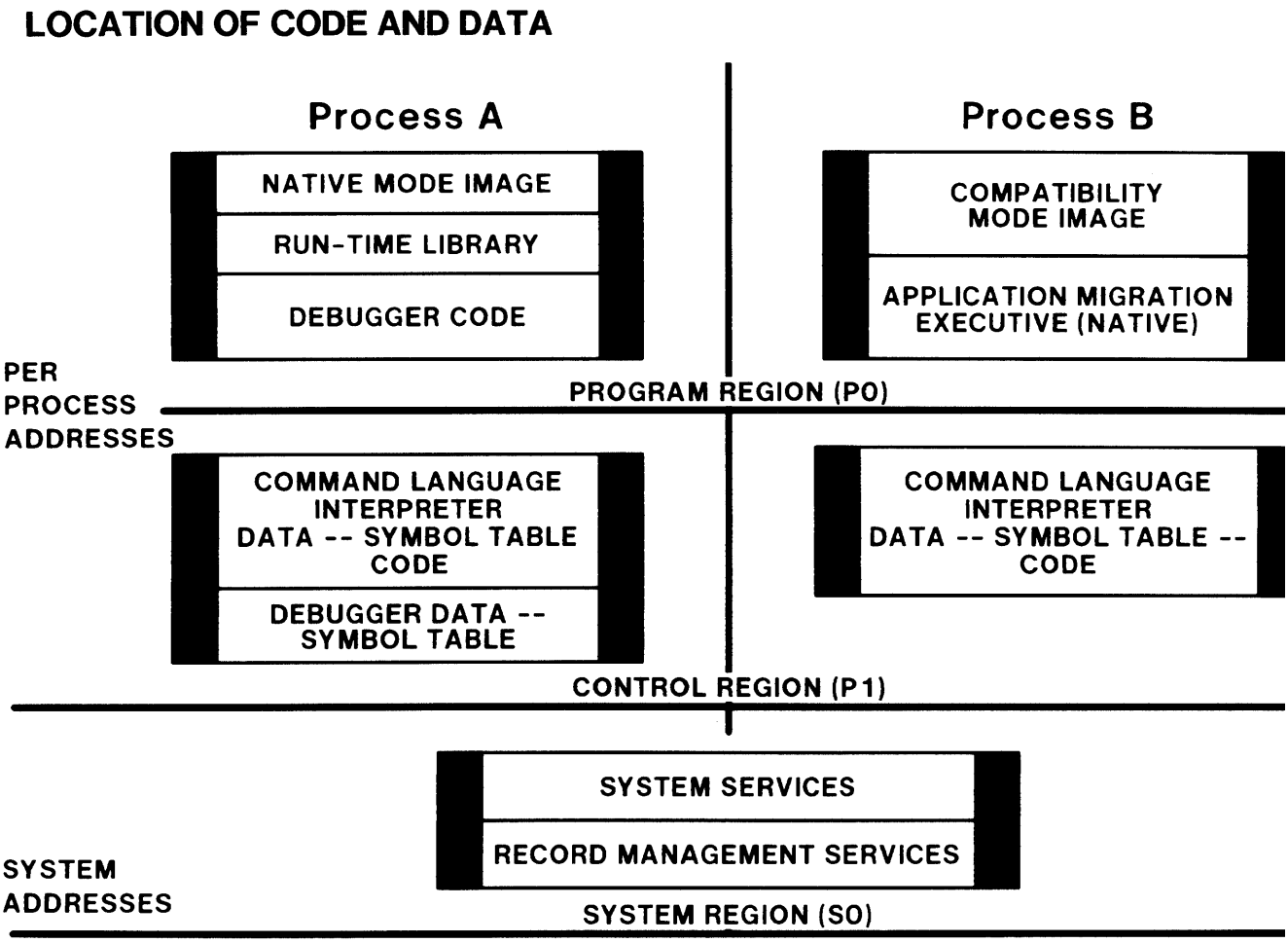


Figure 1-5 Location of Code and Data in Virtual Address Space

1. Images running within processes use several different types of software components. Many of these components are shareable sections, shareable images, or system code. PØ space (the program region) and P1 space (the control region) are mapped differently for native and compatibility mode images.

2. PØ space is where user's code and data are mapped.

3. P1 space is where process-specific information is stored by the operating system.

4. SØ space is where the operating system resides. One copy of the operating system is shared by all processes.

ENTRY PATHS INTO VMS KERNEL

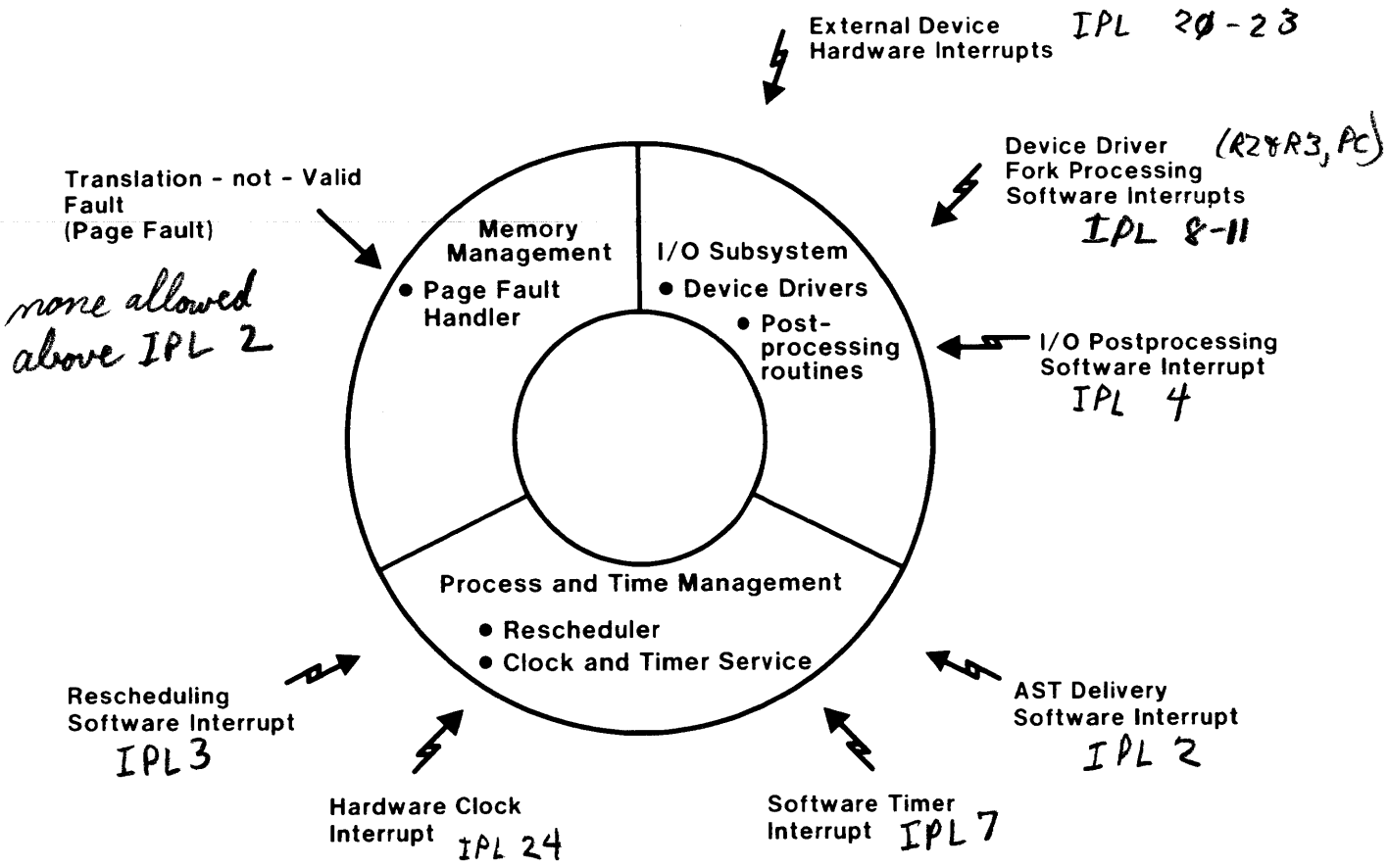


Figure 1-6 Entry Paths into VMS Kernel

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 1-1 Summary of System Components and Functions

Function	System Component
Assign CPU to highest priority computable process	SCHEDULER
Move working set between disk and memory	SWAPPER
Move pages from disk to memory	PAGER
Update system clock and quantum field, check for servicing at intervals	HARDWARE CLOCK ISR
Perform servicing at intervals • check for quantum end • cause events to be posted • check device timeout • wake swapper and error logger	SOFTWARE TIMER ISR
Handle requests to/replies from operator	OPCOM
Write errors to error log file	ERRFMT
Maintain volume structures for driver	ANCILLARY CONTROL PROCESS
Create processes for print jobs, batch jobs, interactive jobs	JOB CONTROLLER
Control devices, service device interrupts, check for and report device errors	DRIVERS
Handle printing of files	PRINT SYMBIONTS
Handle process state transitions resulting from event completion	REPORT SYSTEM EVENT

THREE TYPES OF SYSTEM COMPONENTS

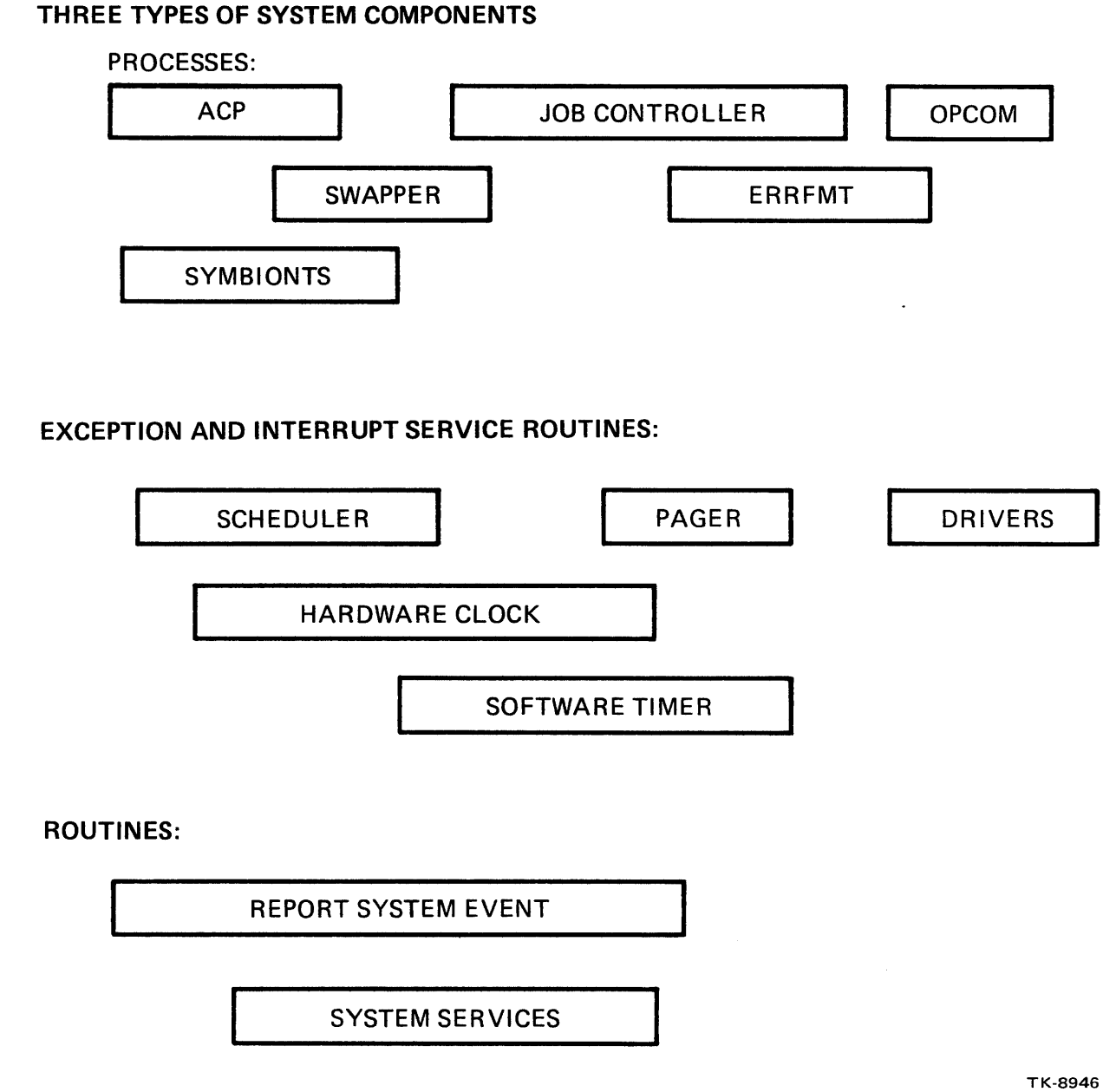


Figure 1-7 Three Types of System Components

HARDWARE CLOCK INTERRUPT

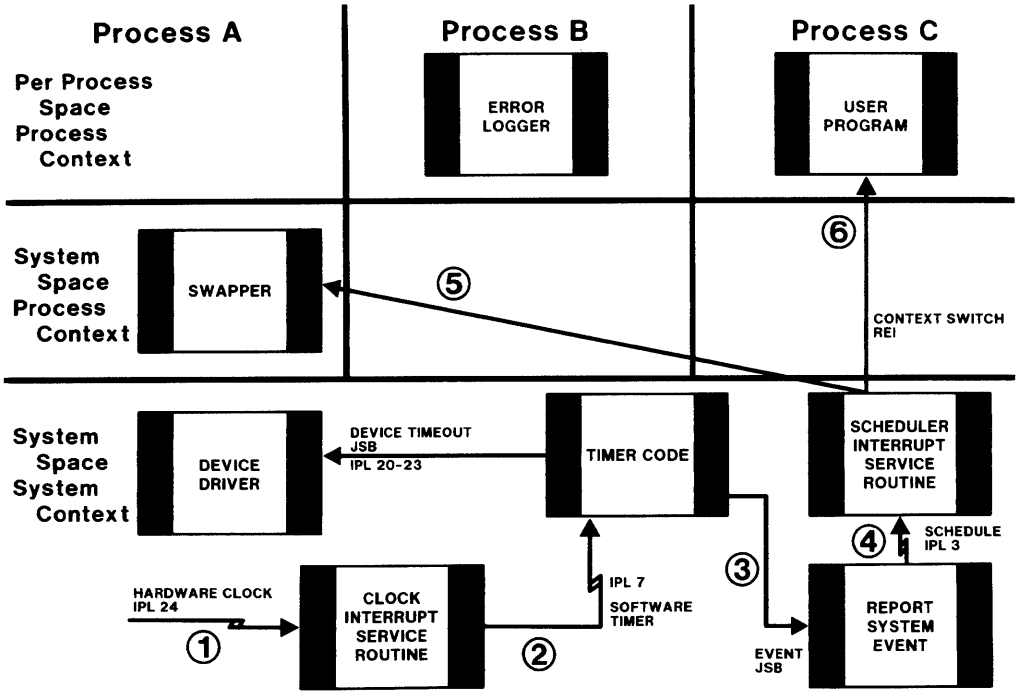


Figure 1-8 Hardware Clock Interrupt

❶ Clock
- Updates system time and quantum field
- Checks first timer queue entry

❷ Timer
- Checks for quantum end
- Causes events to be processed

❸ Report system event
- Changes process state
- May request scheduler interrupt

❹ Scheduler
- Current <----> Computable

❺ Swapper
- Inswaps computable process

❻ Scheduled user program runs

PERIODIC CHECK FOR DEVICE TIMEOUT

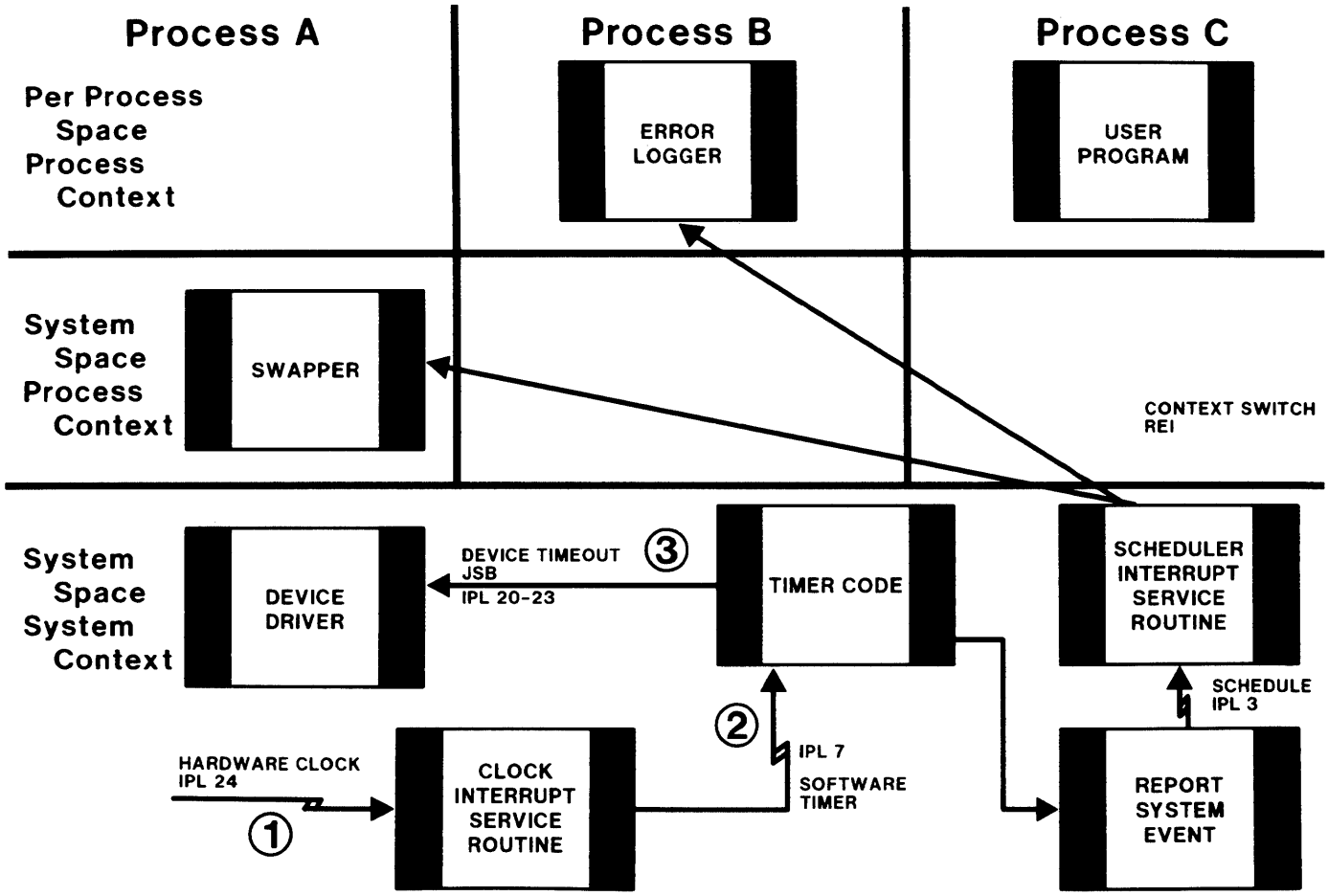


Figure 1-9 Periodic Check for Device Timeout

1.) JSB sch$_swpwake
2.) update in $GL_abstime
3.) checks wlog wake
4.) UCB TO
5.) deadlocks
6.) PIXSCAN

- Hardware clock interrupt
- Once every second, a timer queue entry becomes due which causes a system subroutine to execute.
- This system subroutine checks for device timeouts, calls drivers to handle timeouts.

PERIODIC WAKE OF SWAPPER, ERROR LOGGER

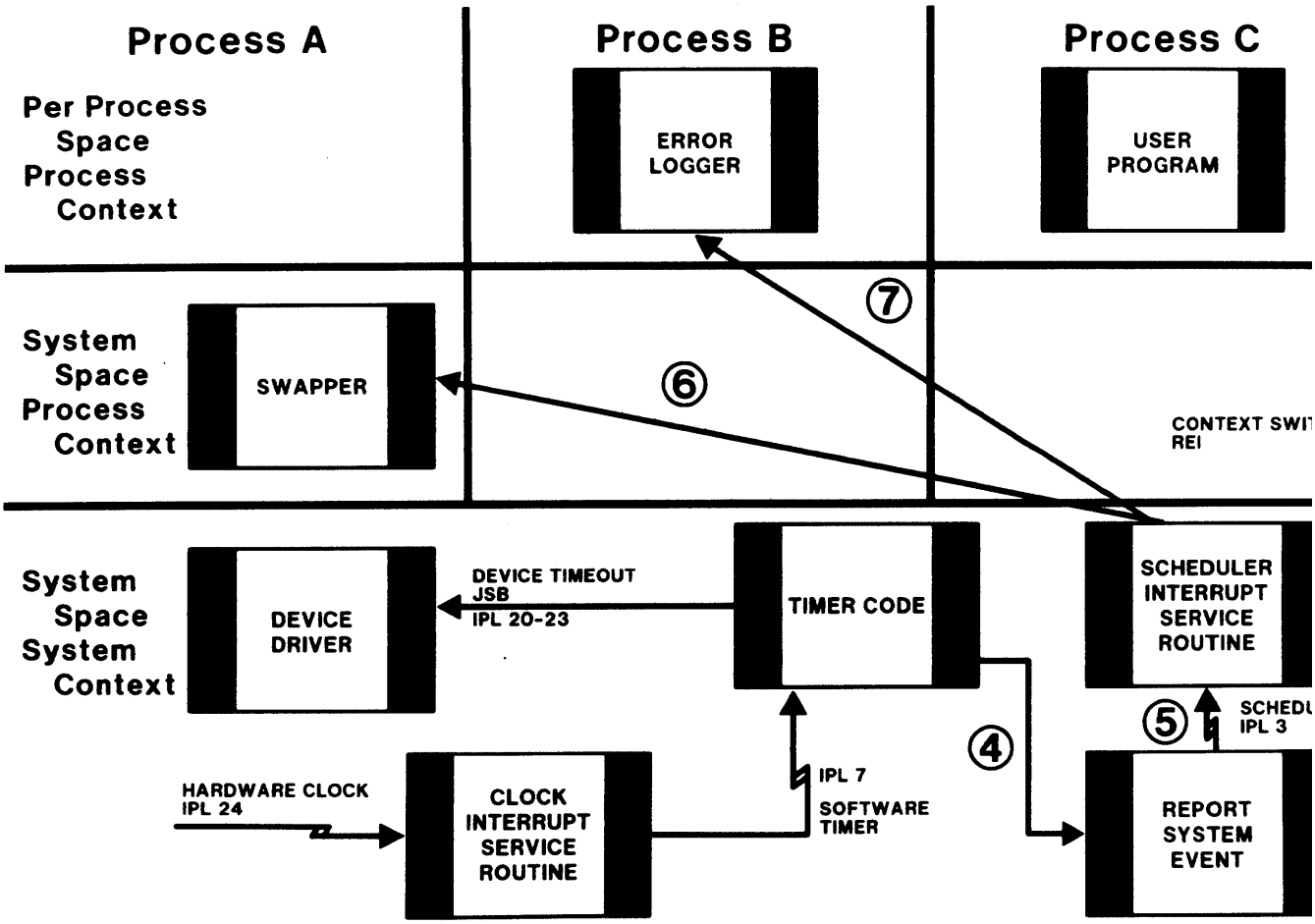


Figure 1-10 Periodic Wake of Swapper, Error Logger

★ SCH$CHSE

- The same system subroutine may wake the swapper process and the error logger processes.
- Scheduler interrupt is requested.
- Swapper and error logger will eventually run.

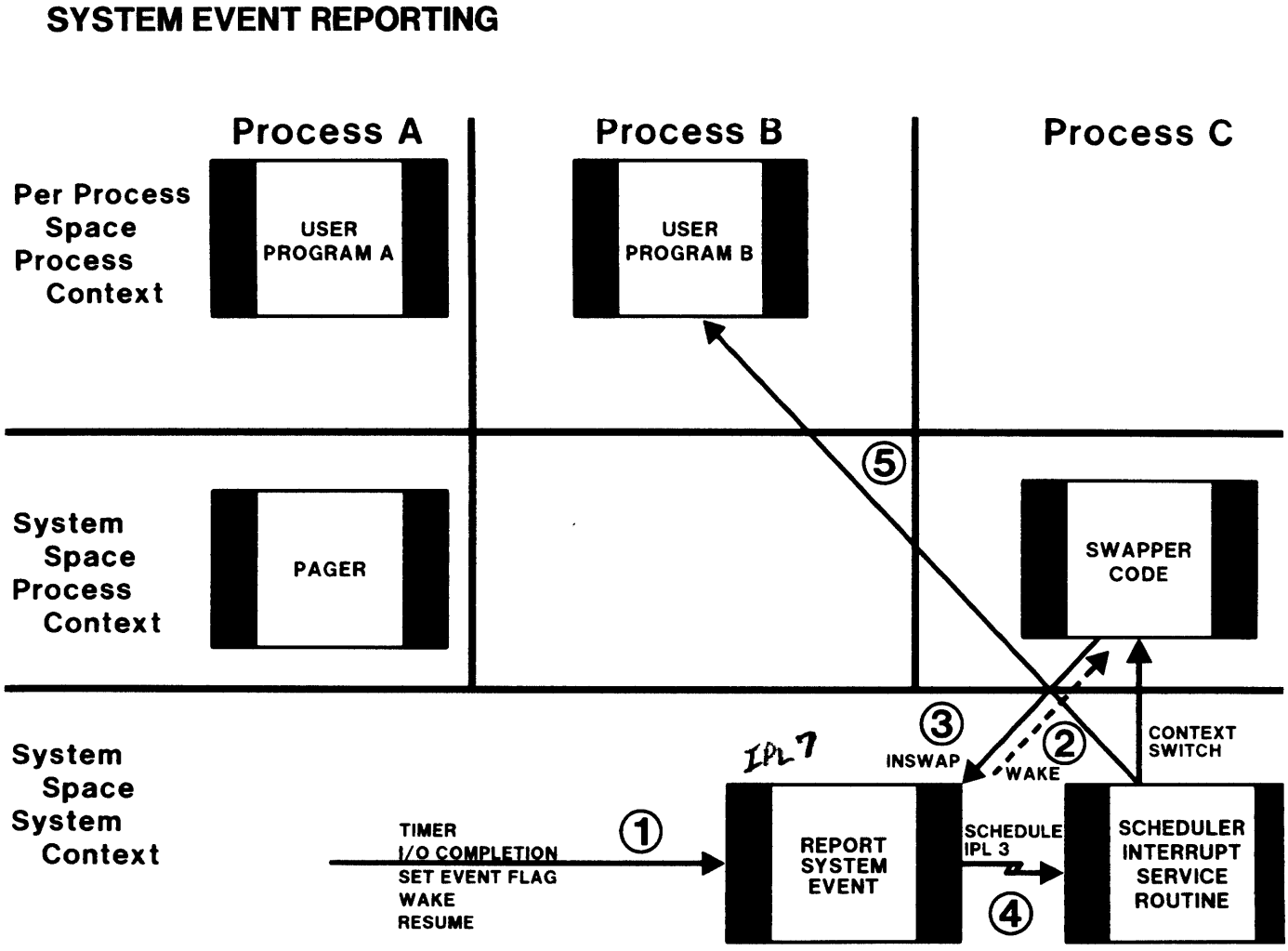


Figure 1-11 System Event Reporting

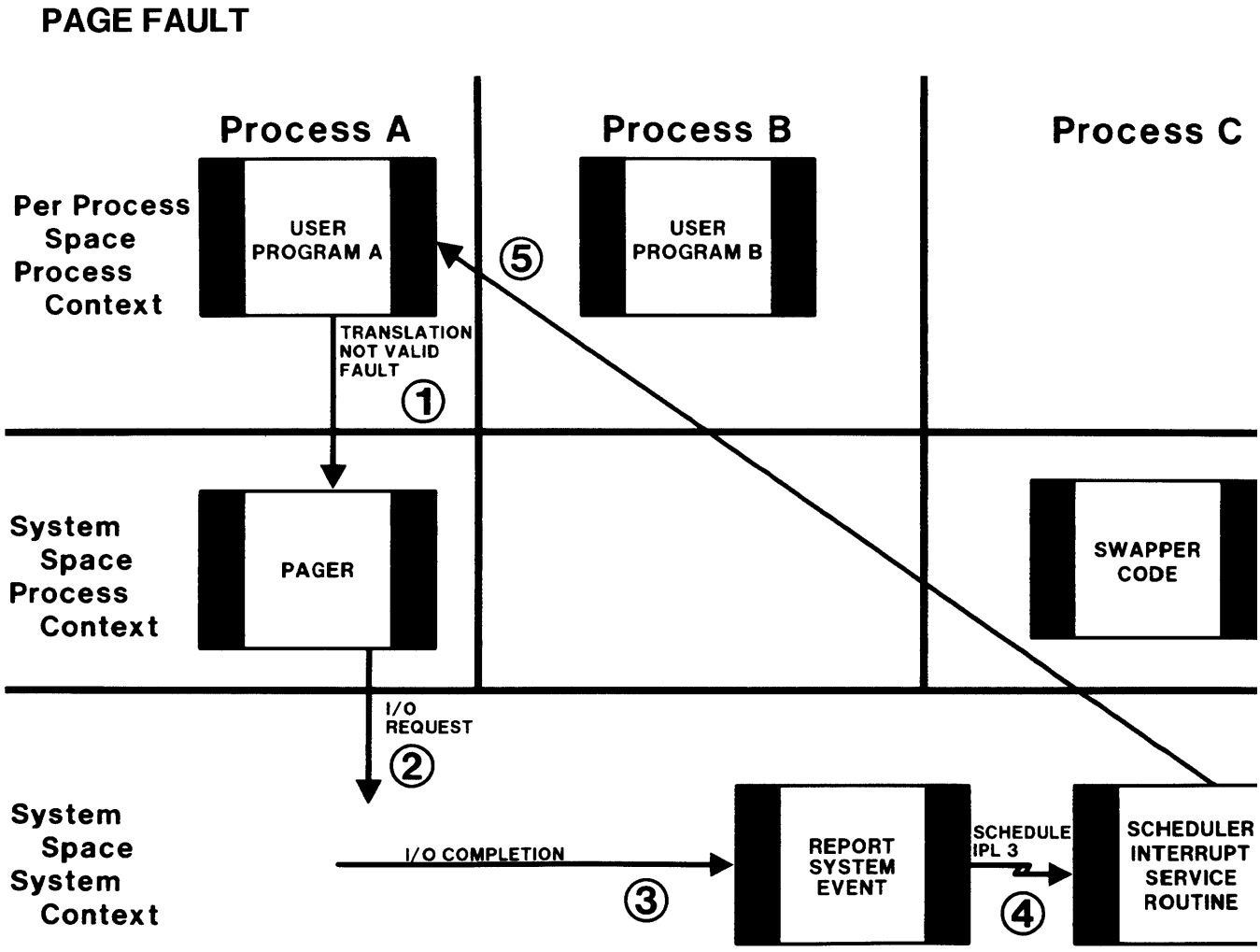


Figure 1-12 Page Fault

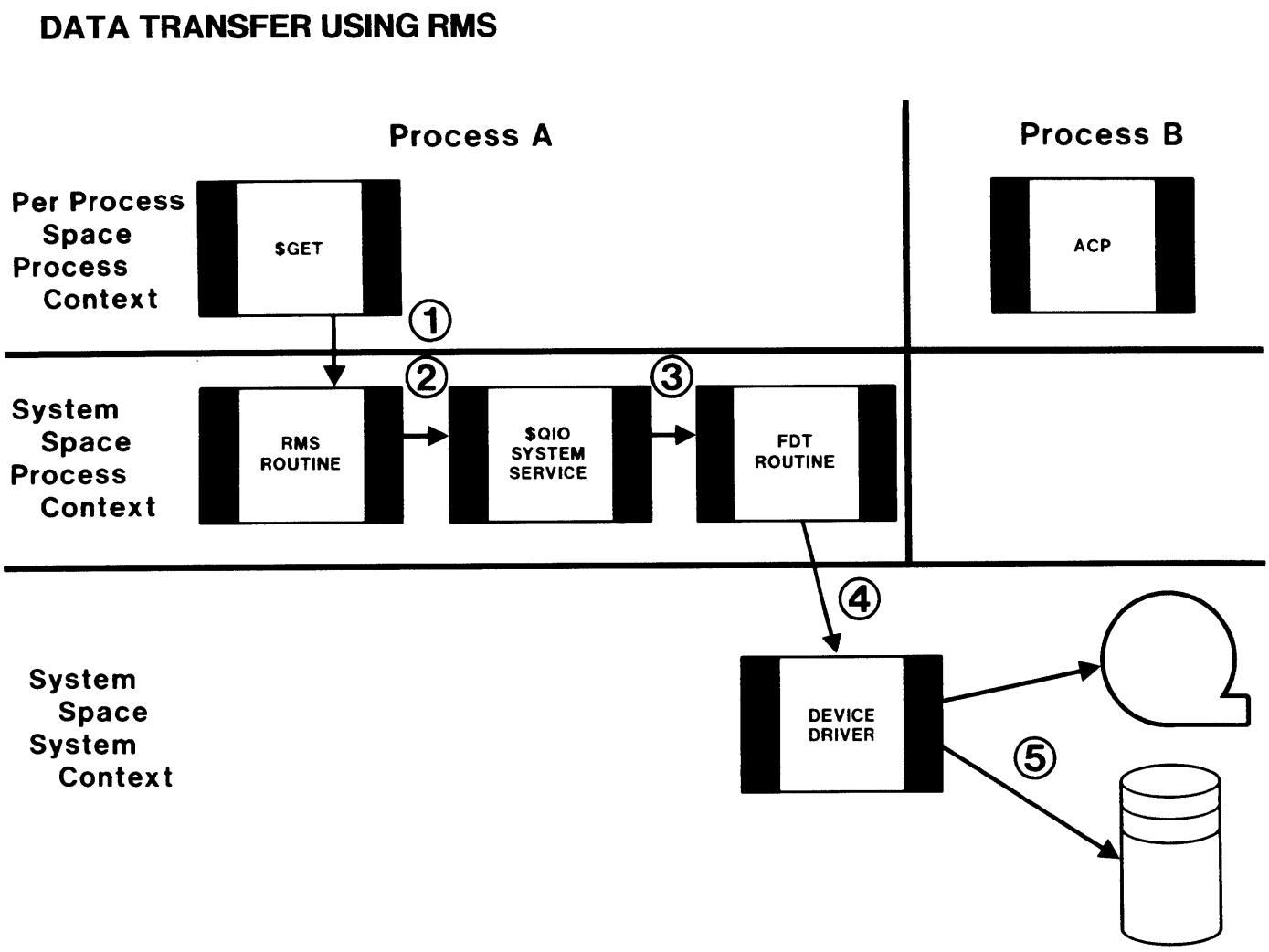


Figure 1-13 Data Transfer Using RMS

When record formats are defined within disk blocks (or clusters of blocks), then the Record Management Services (VAX-11 RMS) are used.

FILE MANIPULATION USING RMS

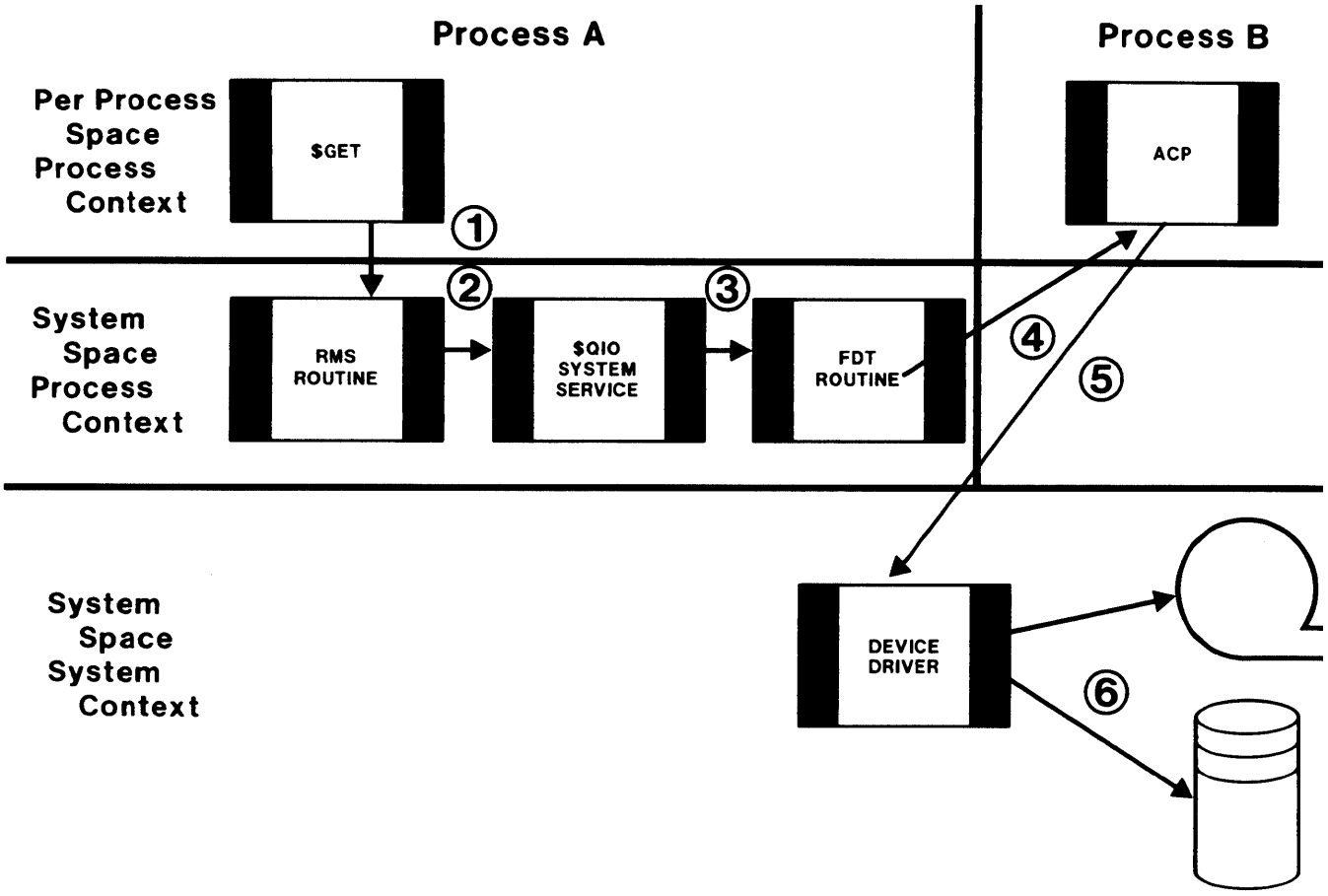


Figure 1-14 File Manipulation Using RMS

When a file structure is imposed on a volume, the following operations require the intervention of Ancillary Control Processes (ACPs) to interpret or manipulate the file structure.

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

"cathedral window" - chain of WCB (in dynamic non-pag pool

DATA TRANSFER USING $QIO

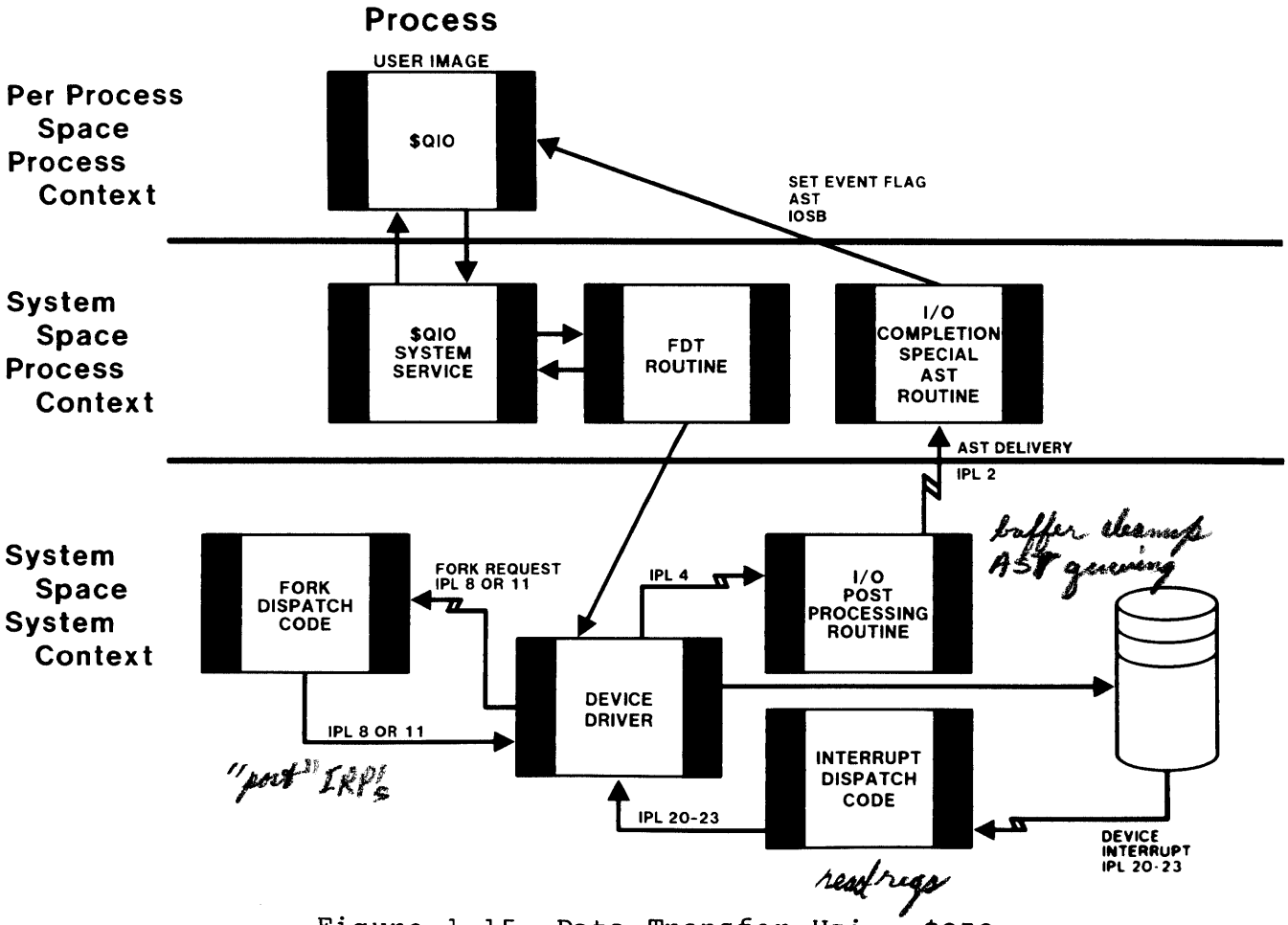


Figure 1-15 Data Transfer Using $QIO

$QIO SEQUENCE OF EVENTS

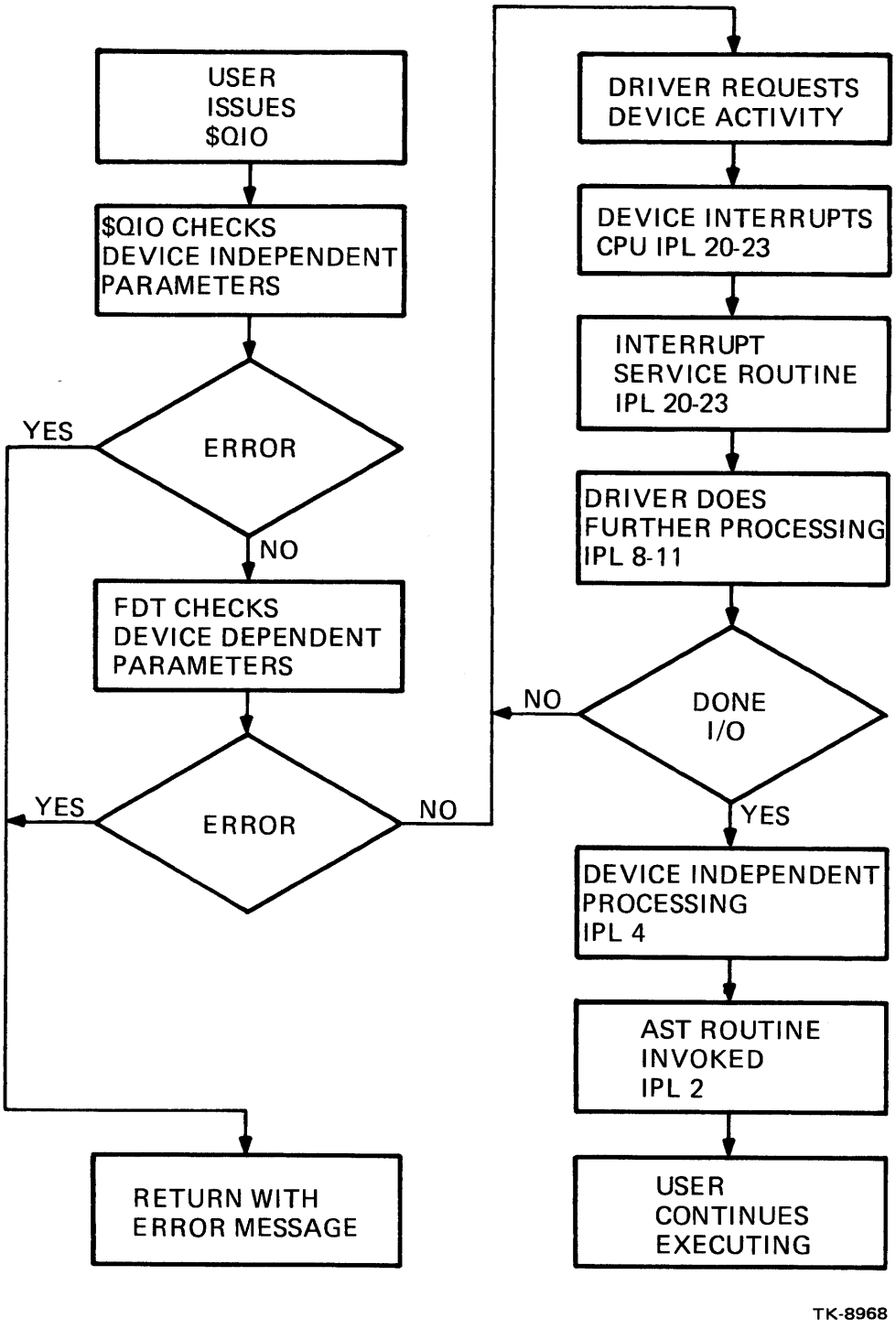


Figure 1-16 $QIO Sequence of Events

INTRODUCTION TO DECnet

DECnet Protocols

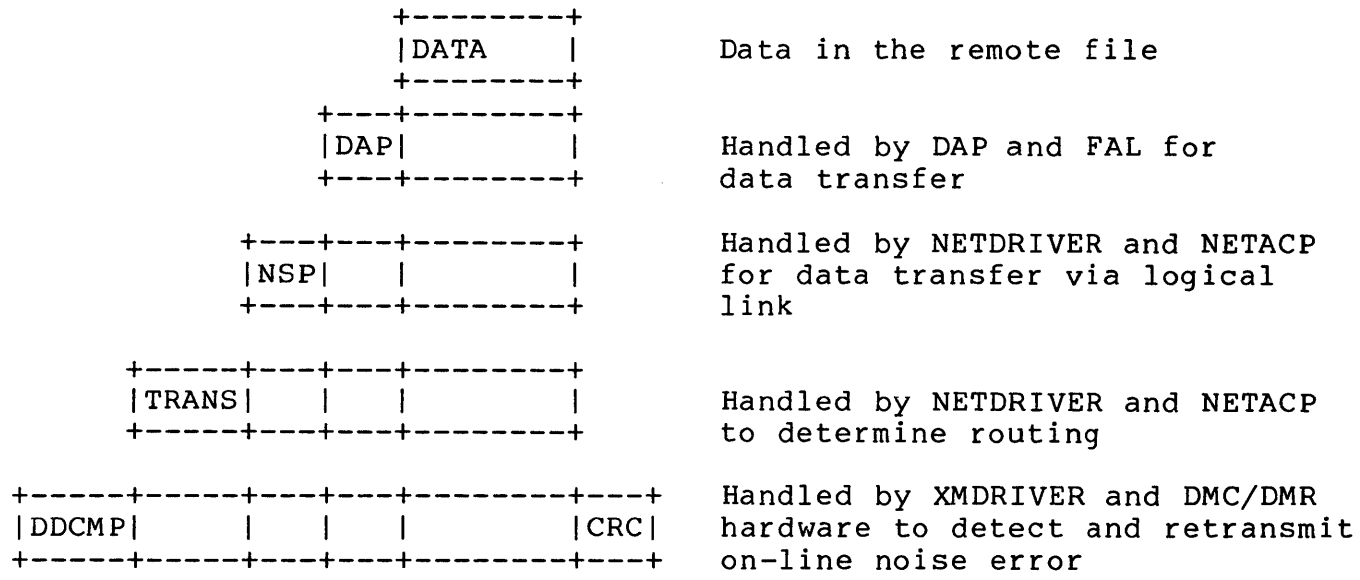


Figure 1-17 DECnet Protocol Layers

DECnet REMOTE FILE ACCESS

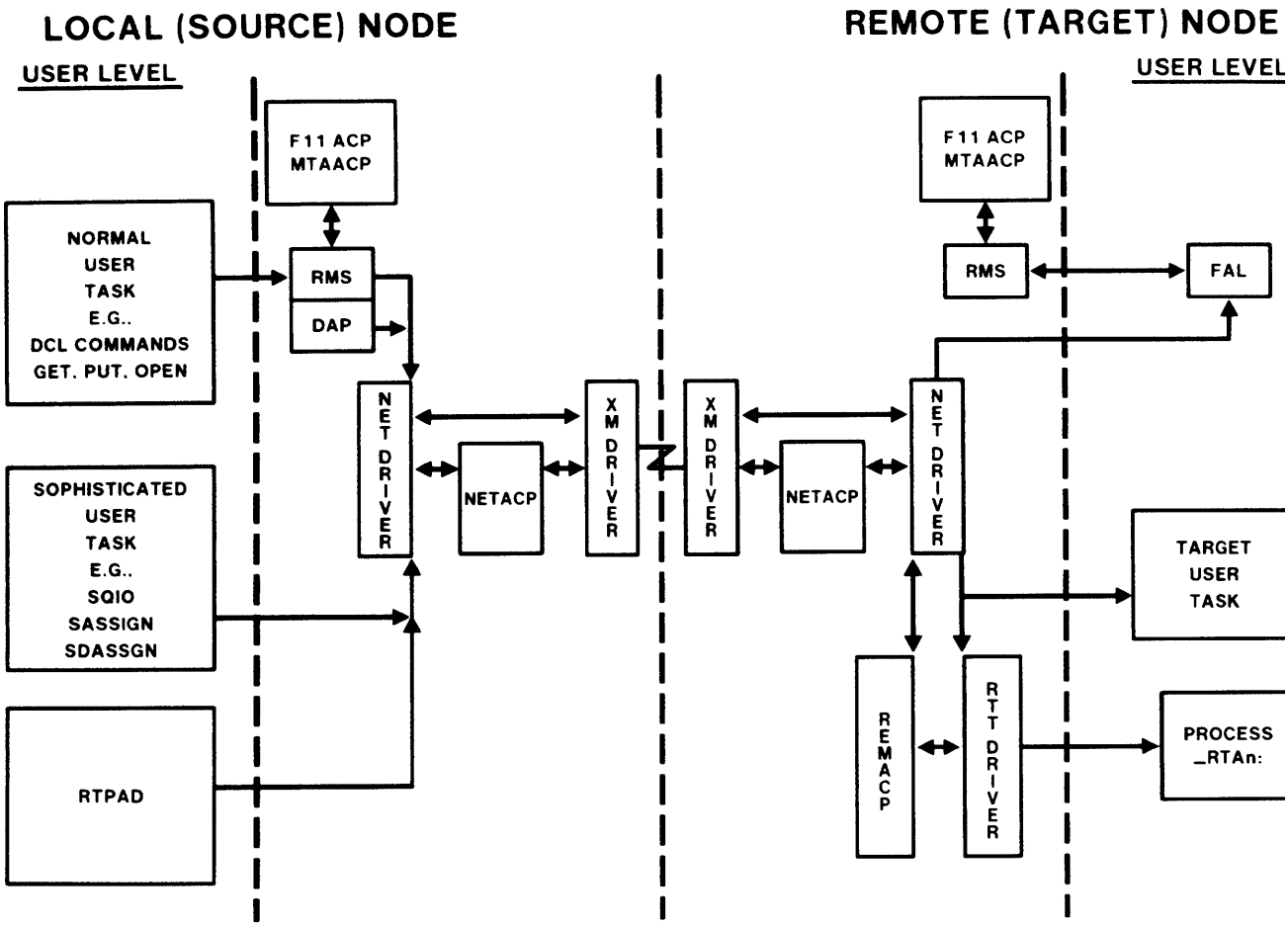


Figure 1-18 DECnet Remote File Access (e.g., Copy)

REMOTE FILE ACCESS-DATA FLOW

Setting Up Logical Link

On source node

1. User issues DCL command:

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

2. RMS detects ::, sets up logical link with process FAL on NODEB

3. RMS issues internal QIO to NETDRIVER.

4. NETDRIVER passes the I/O request to NETACP, wakes NETACP.

5. NETACP sets up data structures to support the logical link.

6. NETDRIVER builds packet and issues internal QIO to XMDRIVER.

7. XMDRIVER sends packet via DMC/DMR on physical line.

On remote node

1. XMDRIVER receives packet, passes to NETDRIVER.

2. NETDRIVER passes packet to NETACP; sends Connect Acknowledge back to source node.

3. NETACP sets up data structures, handles process creation for FAL.

Data Transfer Within Logical Link

1. Logical link has been established to FAL, which issues requests to RMS on remote node and sends packets back via NETDRIVER.

2. Protocol headers are added to packet at source and removed from packet at remote node.

DECnet TASK-TO-TASK COMMUNICATION

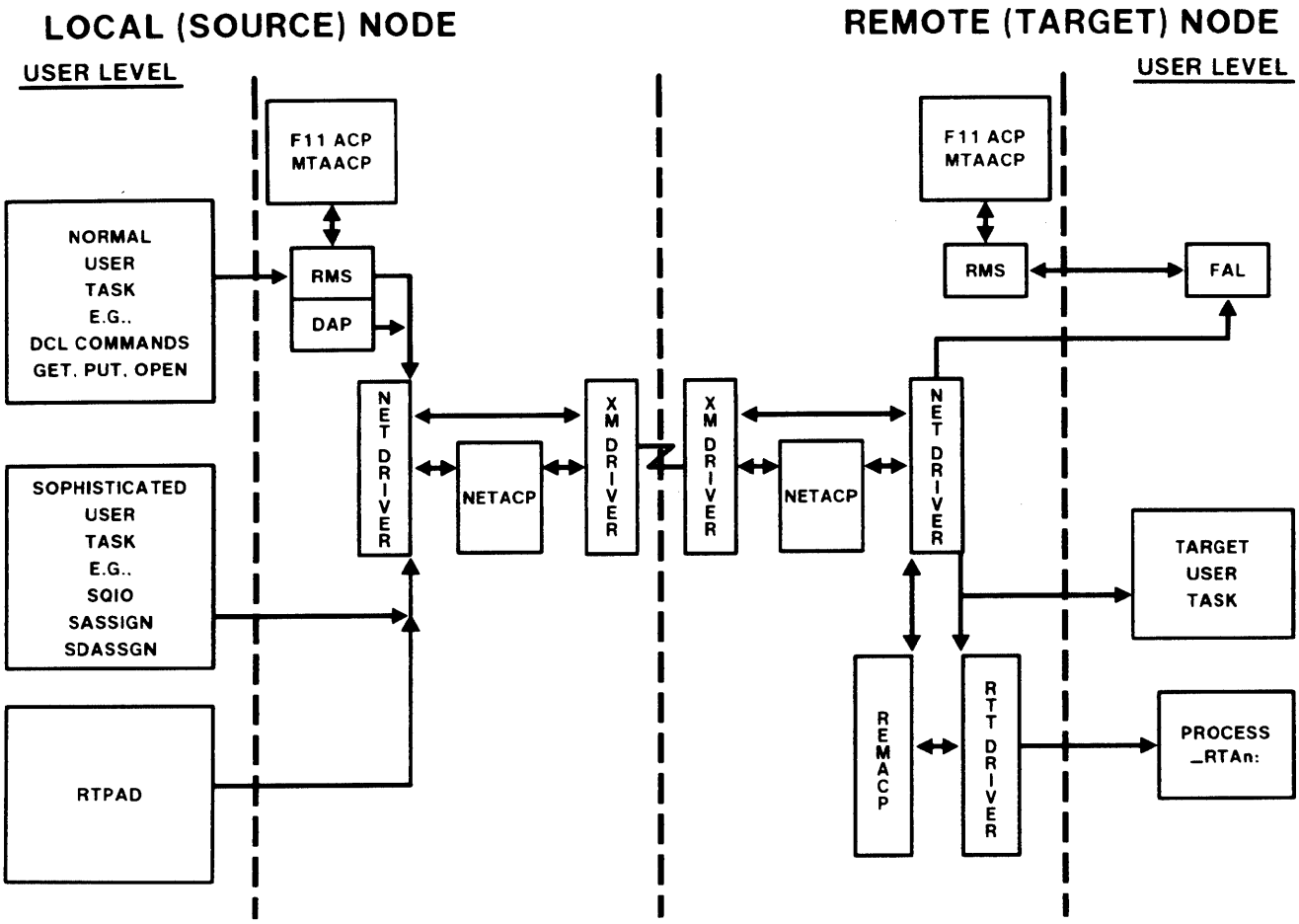


Figure 1-19 DECnet Task-to-Task Communication

TASK-TO-TASK COMMUNICATION DATA FLOW

1. For example, on the source node, the user issues:

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

   and in the program:

   OPEN (NAME=XXX ....)

2. The OPEN command will be passed to RMS.

3. RMS checks the translation and will set up a logical link with the remote program YYY.

4. The procedure is similar to remote file access with the following differences:

   a. The command procedure YYY.COM must reside on the directory of USERID on NODEB (SYS$LOGIN).

   b. The remote program will use the logical name SYS$NET to accept connection.

      e.g., OPEN (NAME=SYS$NET ....)

   c. The two programs must cooperate. For example, when one program issues a Read, the other issues a Write.

5. It is possible for a sophisticated user task to bypass RMS, by issuing QIO's directly to the NETDRIVER.

DECnet PERFORMING SET HOST OPERATION

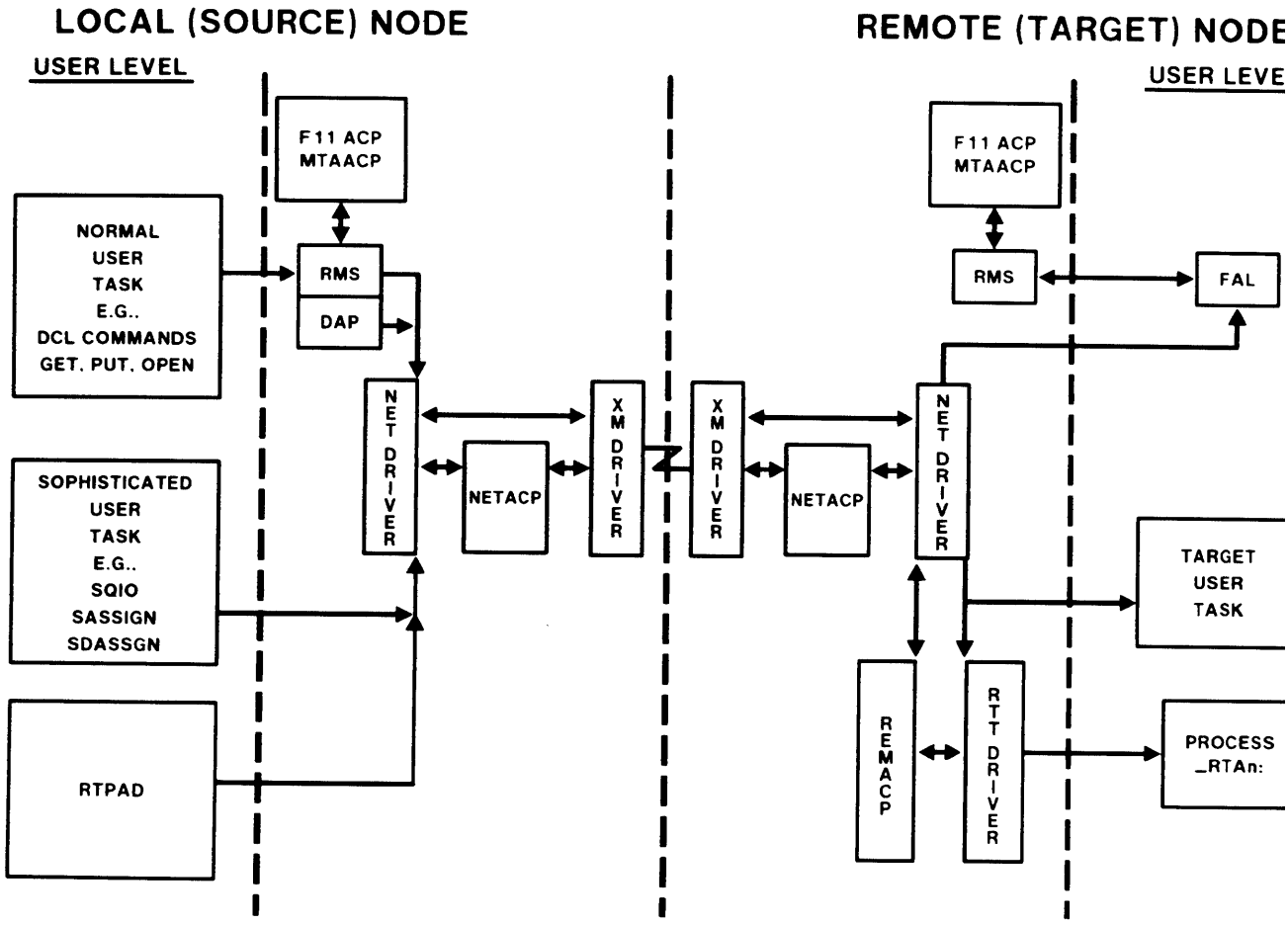


Figure 1-20 Performing Set Host Operation

SET HOST DATA FLOW

1. Similar to task-to-task communication with the following differences:

   a. The user task in source node is RTPAD. When user types 'SET HOST', RTPAD is initiated.

   b. A logical link is set up with REMACP. When the connection is established, REMACP creates an RT device and initializes it.

   c. The request from the user task on the source node is first given to REMACP via logical link. REMACP then gives it to RTTDRIVER. RTTDRIVER passes the request to VMS as if the request comes from a real terminal.

   d. The output from VMS goes directly from RTTDRIVER to NETDRIVER. The information is then transferred via the logical link to RTPAD on the source node.

2. REMACP is started at network startup time. It runs as a detached process and can handle more than one logical link. Each logical link will be associated with a process RTAn where n can go up to 15 by default.

DECnet Software Components

RMS

- Interprets command; issues internal QIO to NETDRIVER.

DAP

- Handles the protocol exchanges for file transfer

NETDRIVER and NETACP

- Handle the protocol exchanges for
  - NSP (to get to correct task)
  - Transport (to get to correct node)
- NETDRIVER handles the time critical functions (e.g., transmit or receiver data).
- NETACP handles the non-time critical functions (e.g., setting up logical link).

XMDRIVER

- Handles the DMC/DMR hardware which detects line noise errors and retransmits.

FAL

- Access files on remote system.

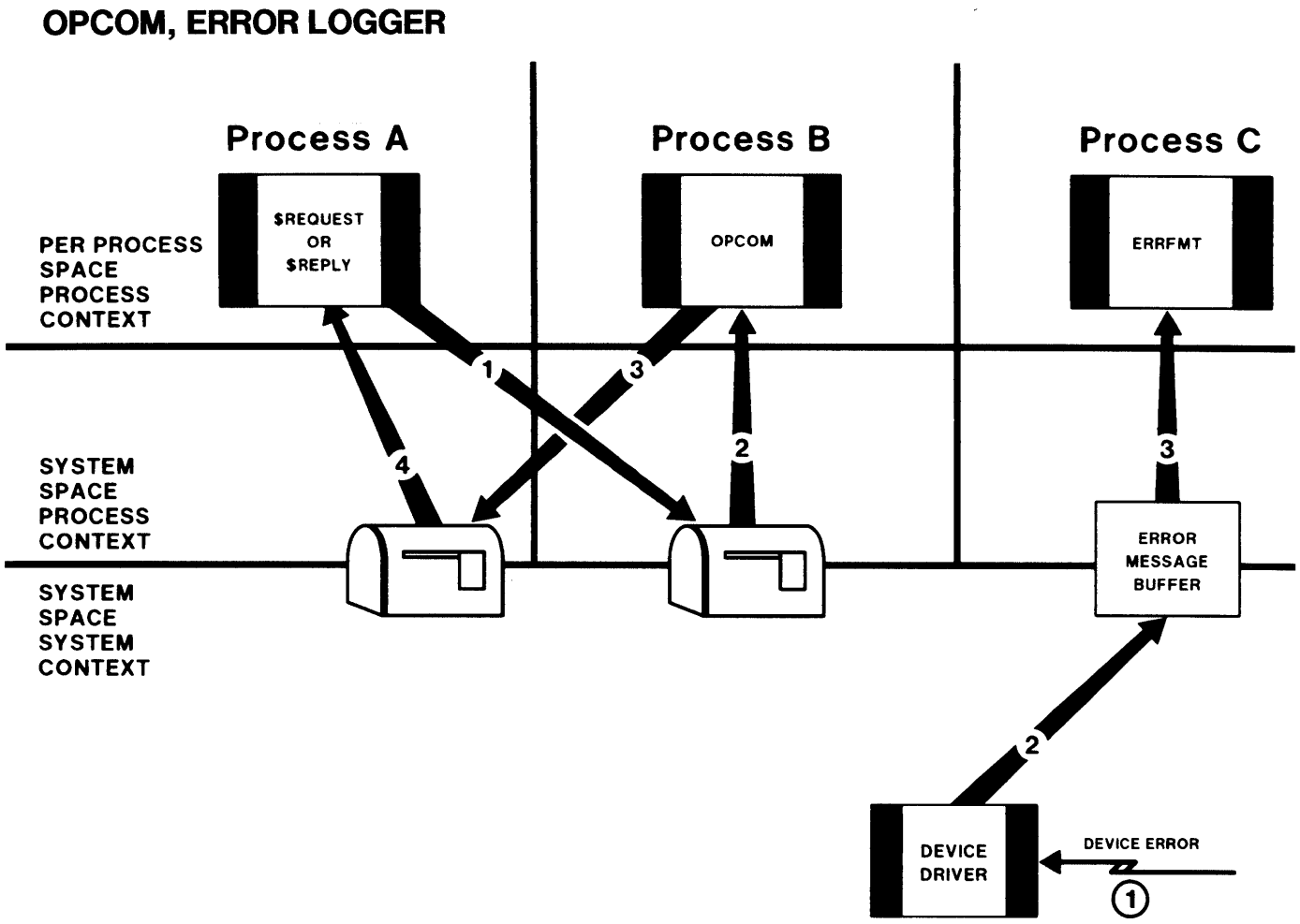


Figure 1-21 OPCOM, Error Logger

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

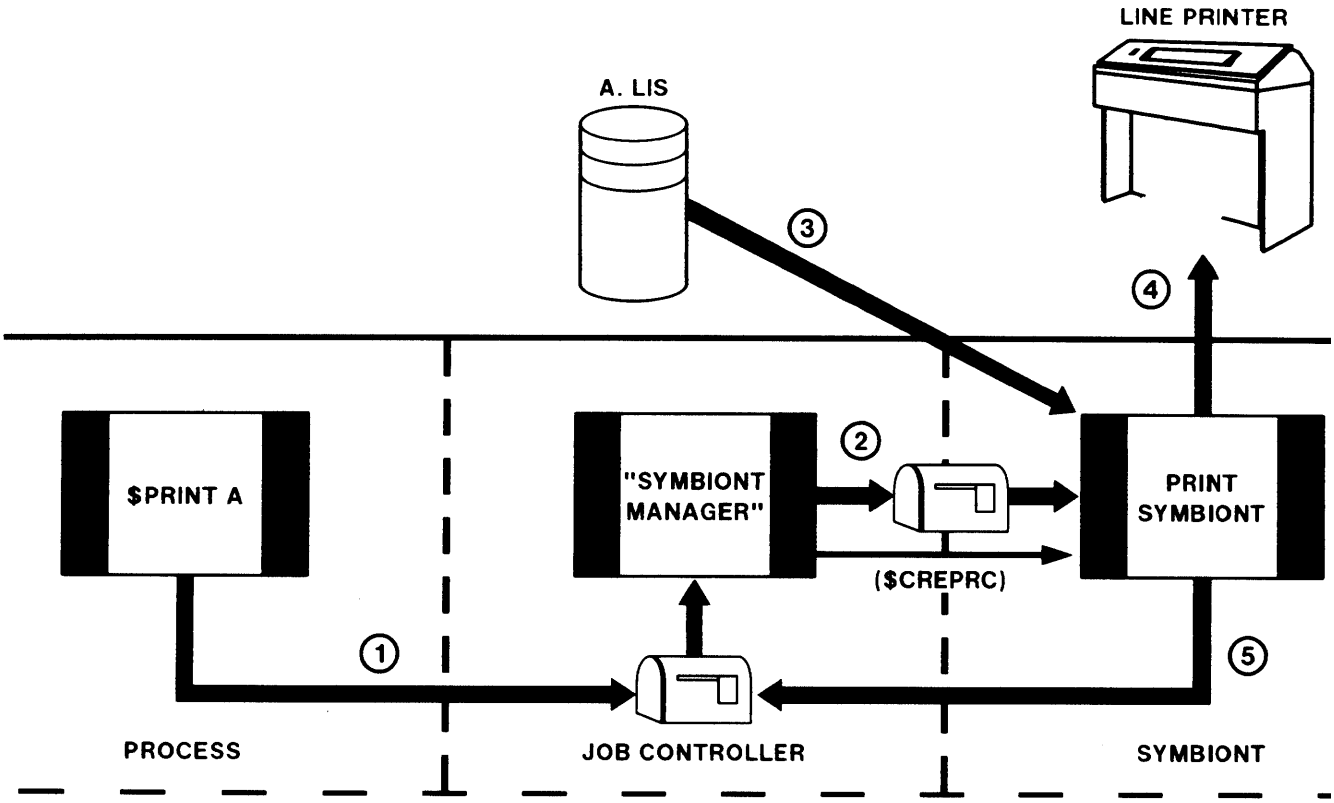


Figure 1-22 Print Jobs

JBC SYSQUE.EXE

printer is file ID number

BATCH JOBS

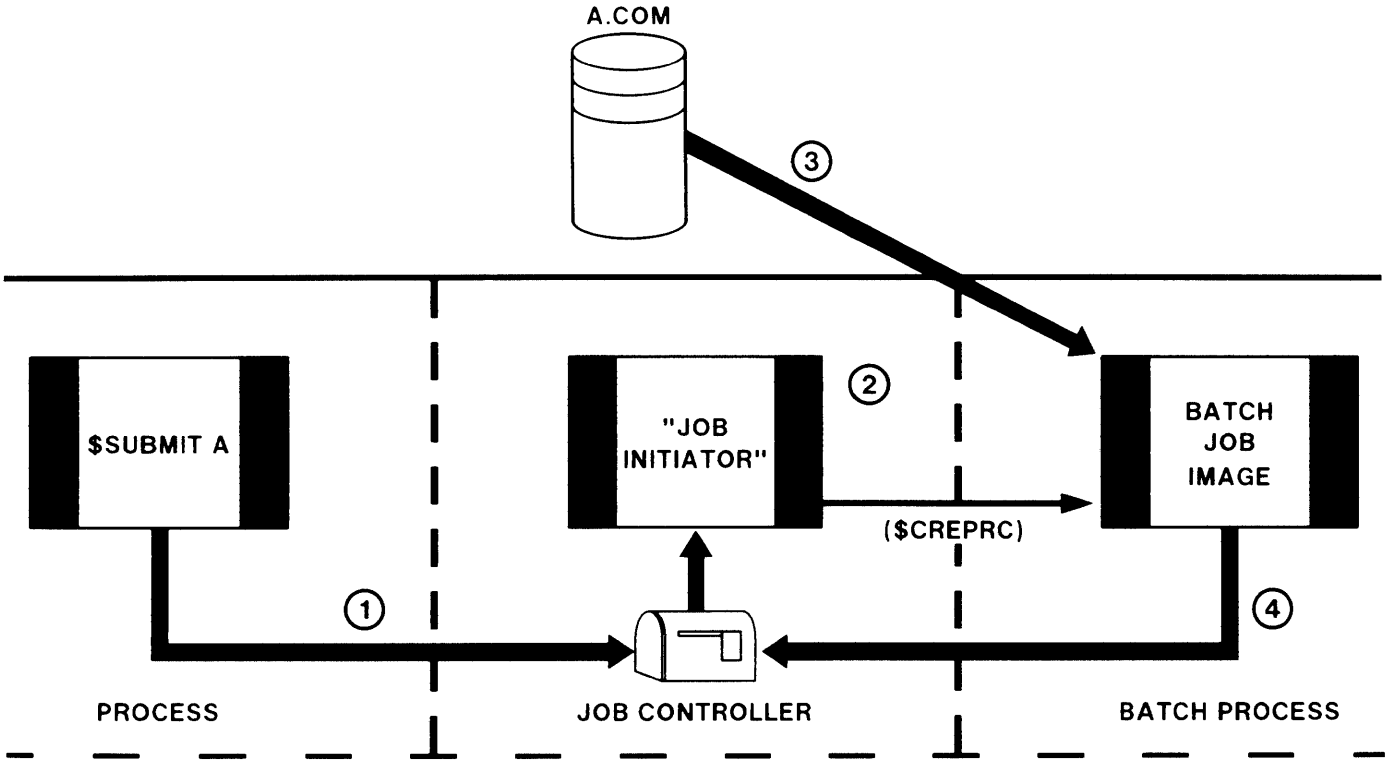


Figure 1-23 Batch Jobs

TERMINAL INPUT

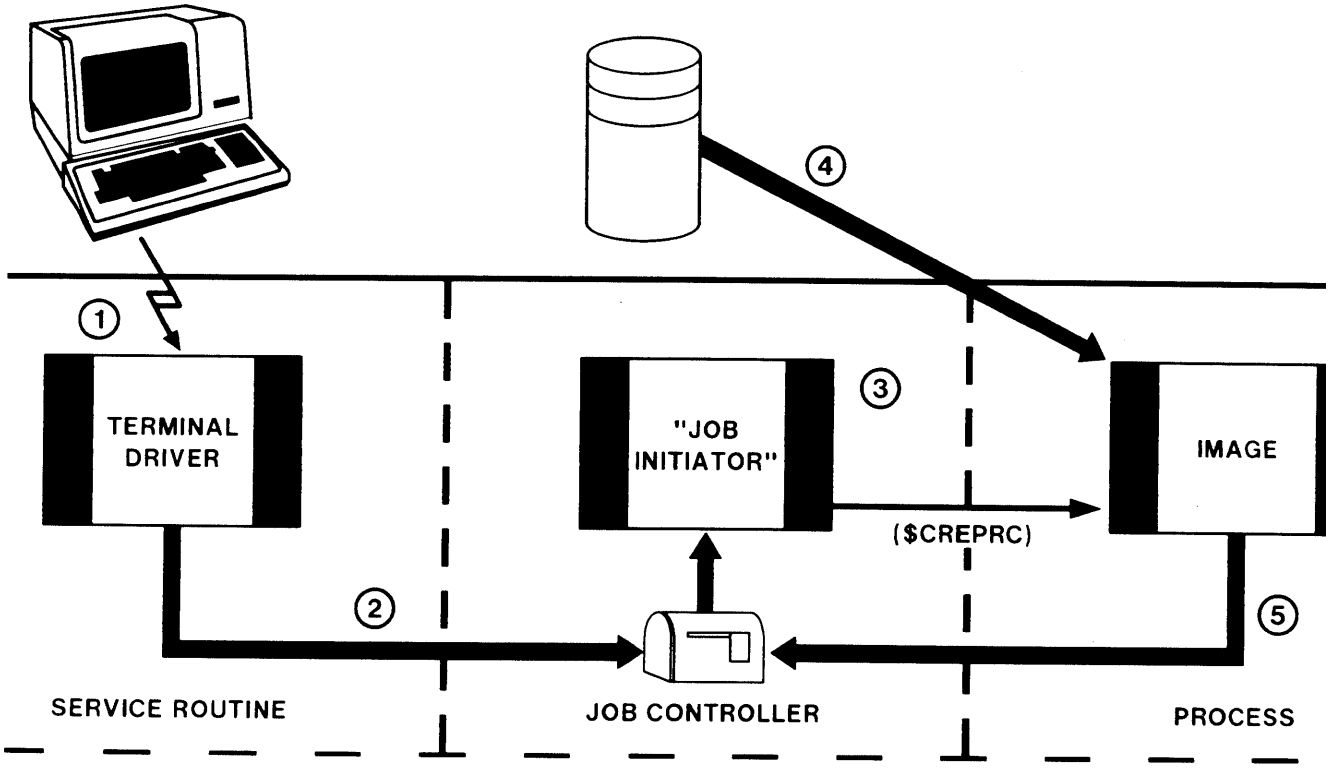


Figure 1-24 Terminal Input

CARD READER INPUT

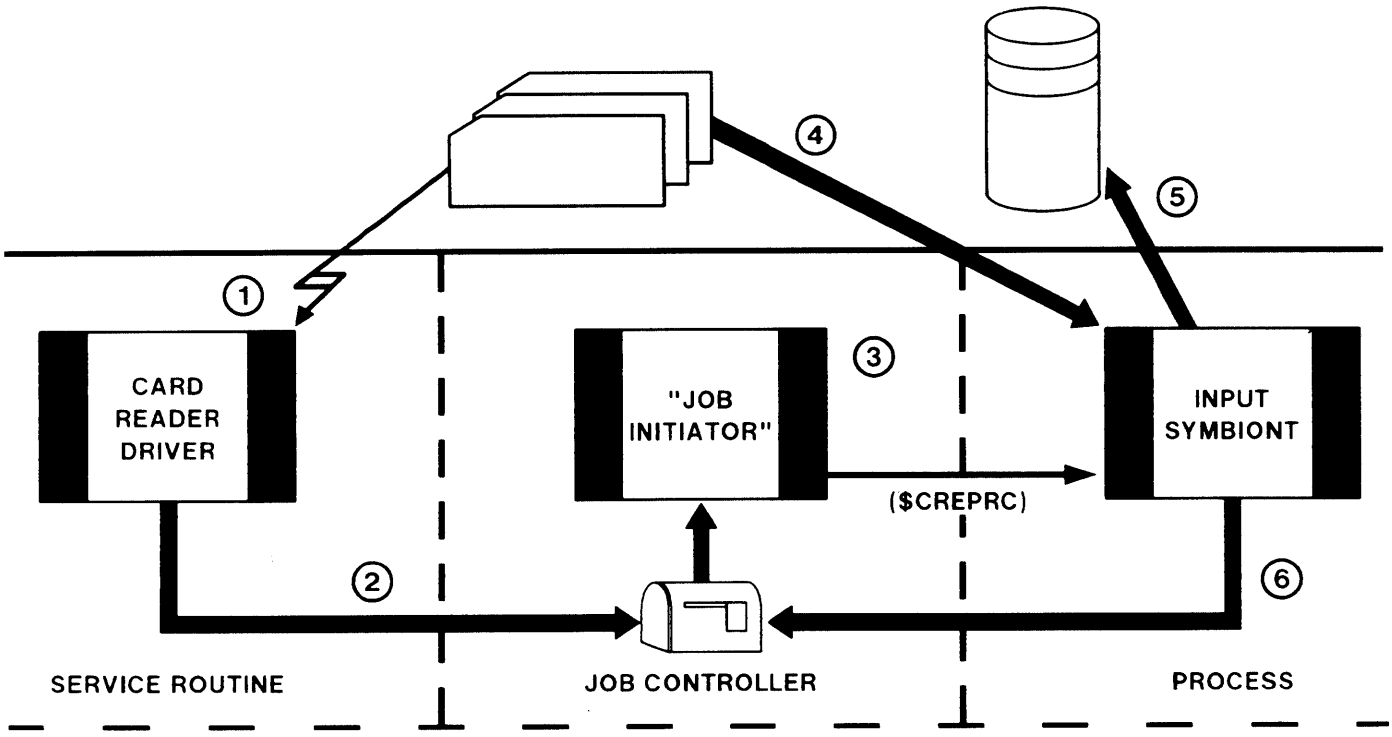


Figure 1-25 Card Reader Input

2

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. (See the "System Components" module.)

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 other resources and features 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
- Software process context
- 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 (RØ through R11)
- The frame pointer (FP), argument pointer (AP), the four perprocess stack pointers (KSP,ESP,SSP,USP), and the current stack pointer (SP)
- 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 page tables of the process.

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 username, 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 PØ) 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 inlcude interrupt service routines and device drivers.

OBJECTIVES

- Describe the similarities and differences of system context and process context.
- 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.
- Describe how the various process data structures are used, which of the structures are modified (and when this occurs), and which of the structures may be reset to default or initial values (and when this occurs).
- Discuss the SYSGEN parameters that relate to process characteristics, and the effects of altering those parameters.

RESOURCES

Reading

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

Additional Suggested Reading

- VAX/VMS Internals and Data Structures Manual, 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. Overview - Process Data Structures

   A. Software context information

   B. Hardware context information

II. Overview - Virtual Address Space

   A. SØ space (operating system code and data)

   B. PØ space (user image code and data)

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

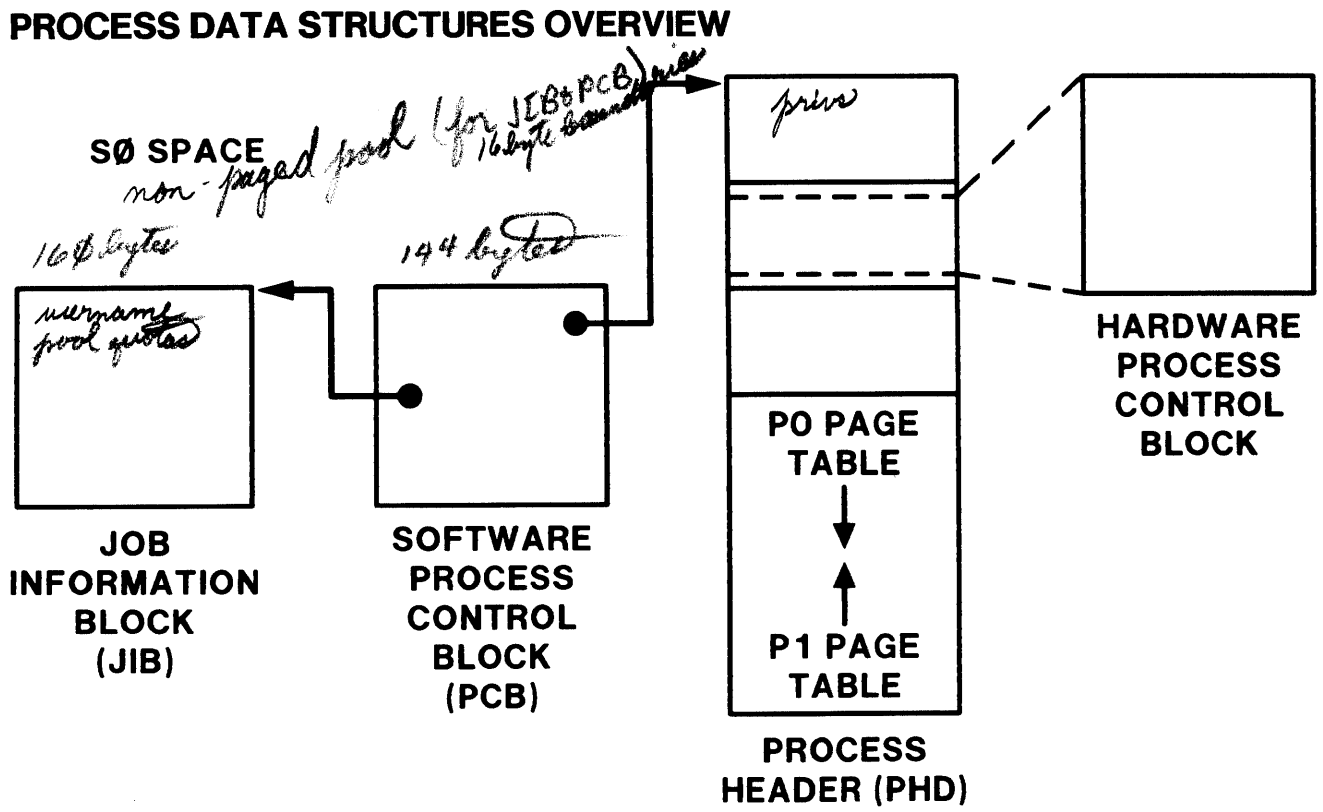
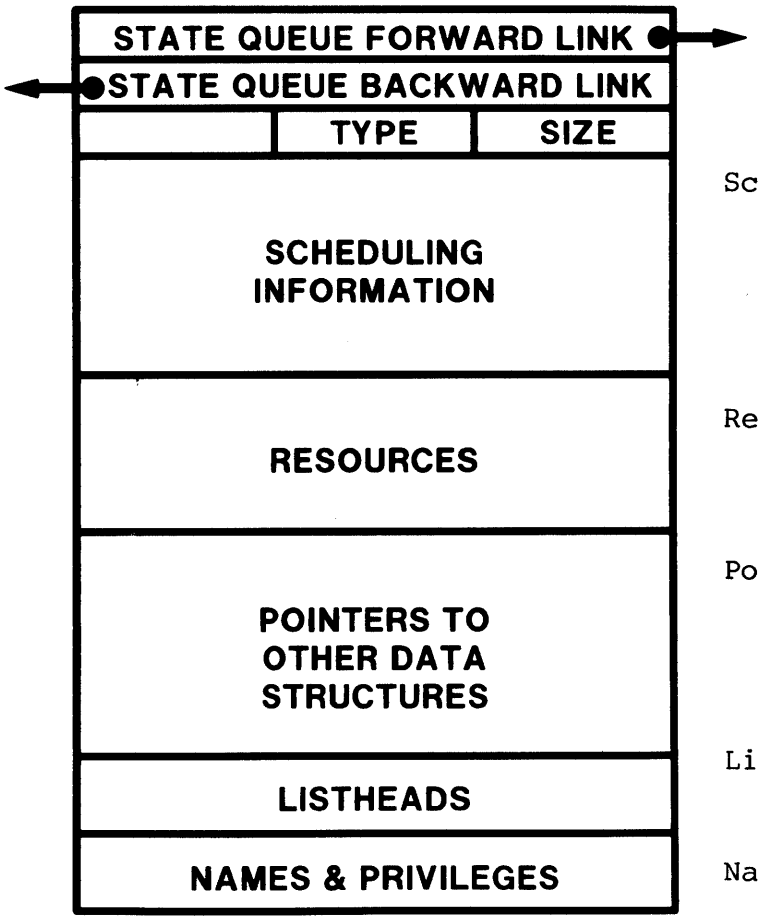


Figure 2-1 Process Data Structures

- Software Process Control Block (PCB) - Holds process specific data that must always be available, even if process outswapped (i.e., process state, pointer to process header).
- Process Header (PHD) - Contains process specific data needed only when process is memory resident (i.e., PØ and P1 page tables).
- Hardware Process Control Block - Contains saved contents of general purpose and memory management registers.
- Job Information Block (JIB) - Keeps track of resources for a detached process and all its subprocesses.

SOFTWARE PROCESS CONTROL BLOCK (PCB)



- VMS standard queue header
- Size of nonpaged pool allocation

Scheduling Information
- Priority
- Status
  Resident/outswapped
  Swap/noswap
- State

Resources
- I/O limits
- Subprocess count

Pointers to:
- Process header
- Hardware PCB
- JIB
- Event flag cluster

Listheads
- AST queue
- Lock queue

Names and Privileges
- Process ID (PID)
- Login UIC
- Privilege mask

Figure 2-2 Software Process Control Block (PCB)

PROCESS HEADER (PHD)

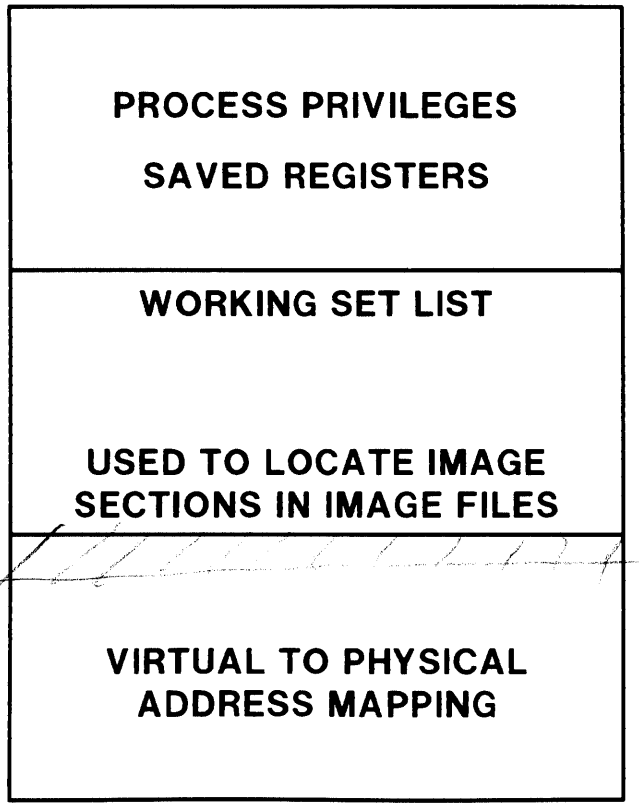


Figure 2-3 Process Header (PHD)

PRIVILEGED VS. GENERAL REGISTERS

Privileged

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

  Clock Registers

  Time of Year (PR$_TODR)
  Interval (PR$_ICR)

  Pointers to Data Structures

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

  Hardware Error Registers

  SBI Error on 78Ø (PR$_SBIER)
  Cache Error on 75Ø (PR$_CAER)

  Other Registers

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

General

- Can be accessed in any access mode using most instructions
- RØ-R11,AP,FP,SP,PC

HARDWARE PROCESS CONTROL BLOCK

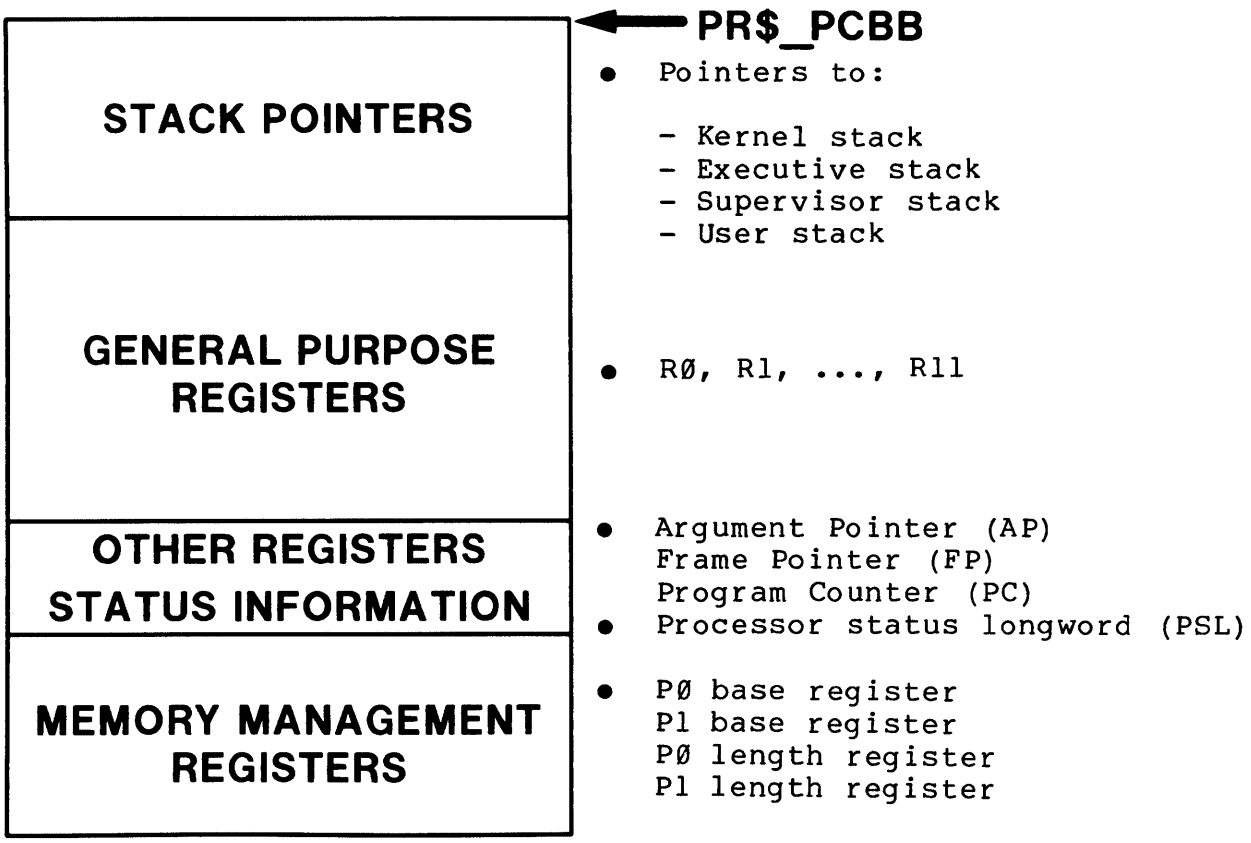


Figure 2-4 Hardware Process Control Block

JOB INFORMATION BLOCK

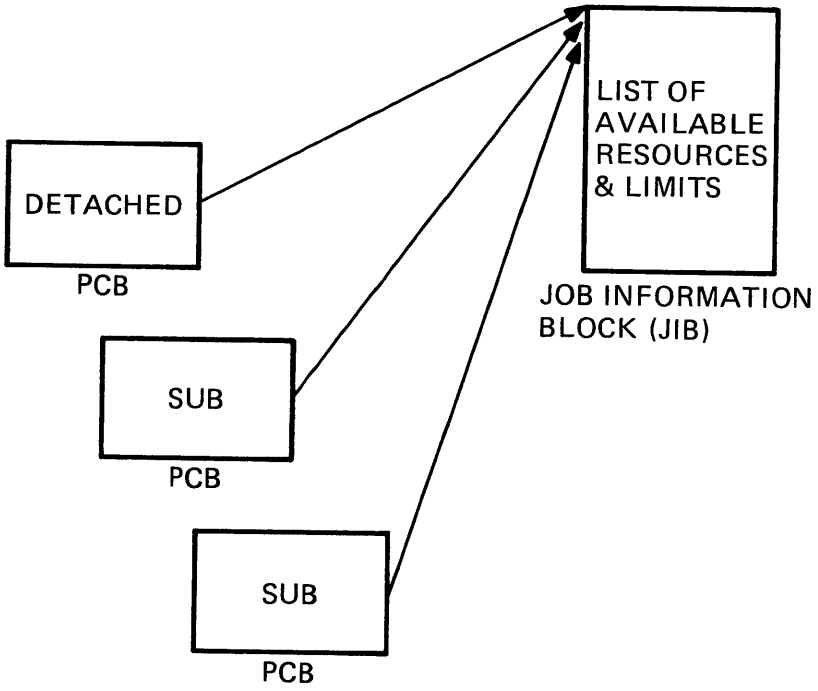


Figure 2-5 Job Information Block

A job consists of a detached process and its subprocesses. The 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

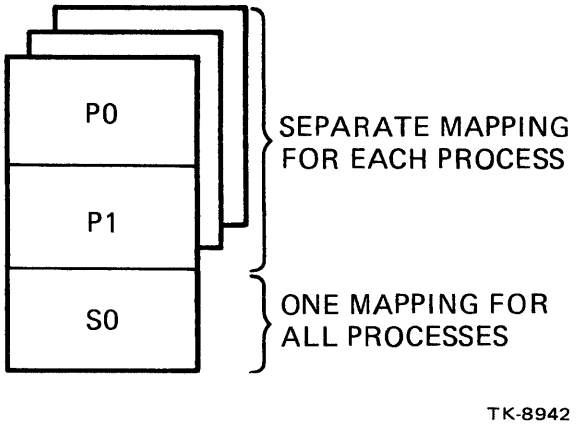


Figure 2-6 Virtual Address Space

Process Virtual Address Space

P0 - Image, Run Time Library, Debugger

P1 - Command Language Interpreter, stacks, I/O data areas

S0 - System services, Record Management Services, other executive code and data

S0 VIRTUAL ADDRESS SPACE

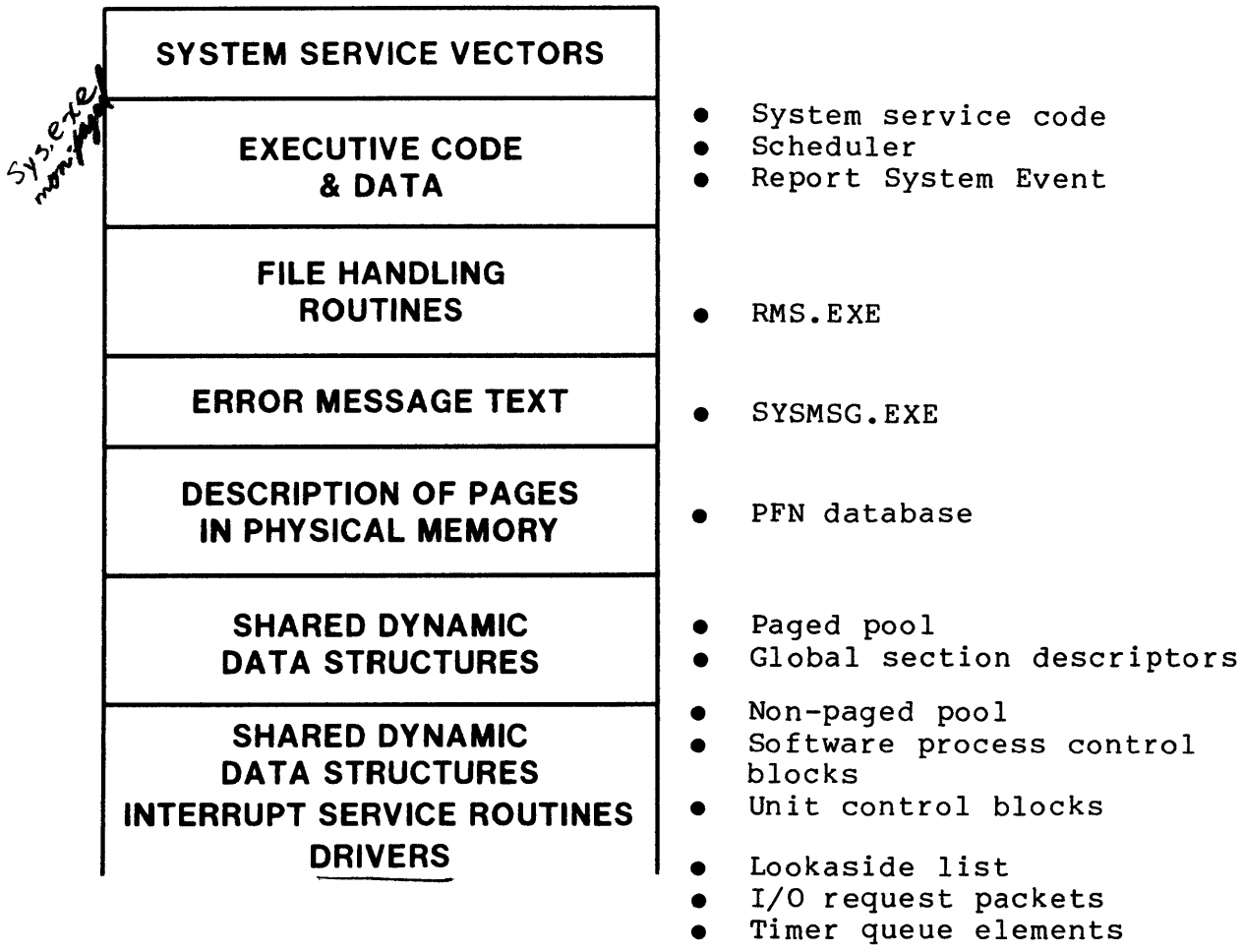


Figure 2-7 SØ Virtual Address Space
(Sheet 1 of 2)

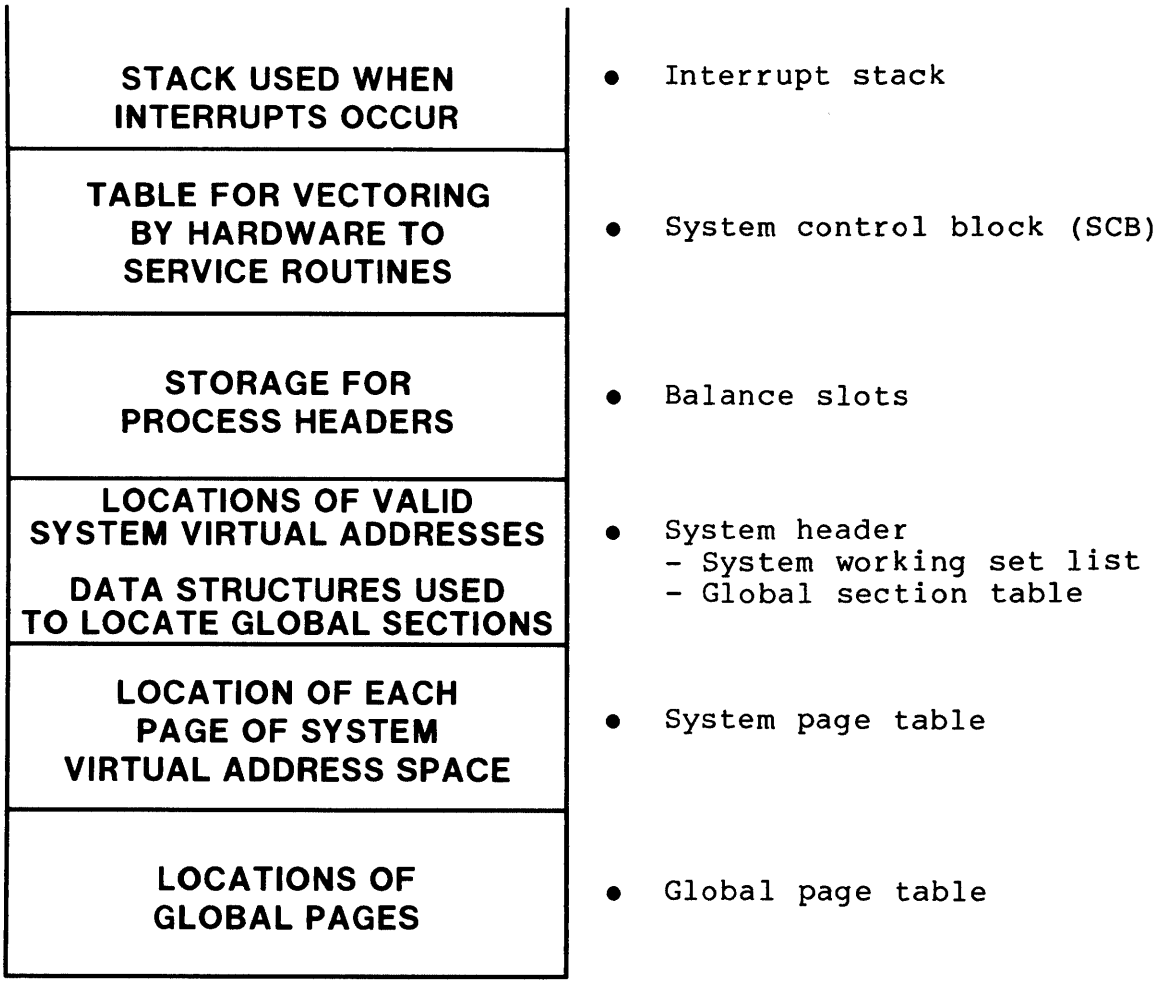


Figure 2-7 SØ Virtual Address Space
(Sheet 2 of 2)

P0 VIRTUAL ADDRESS SPACE

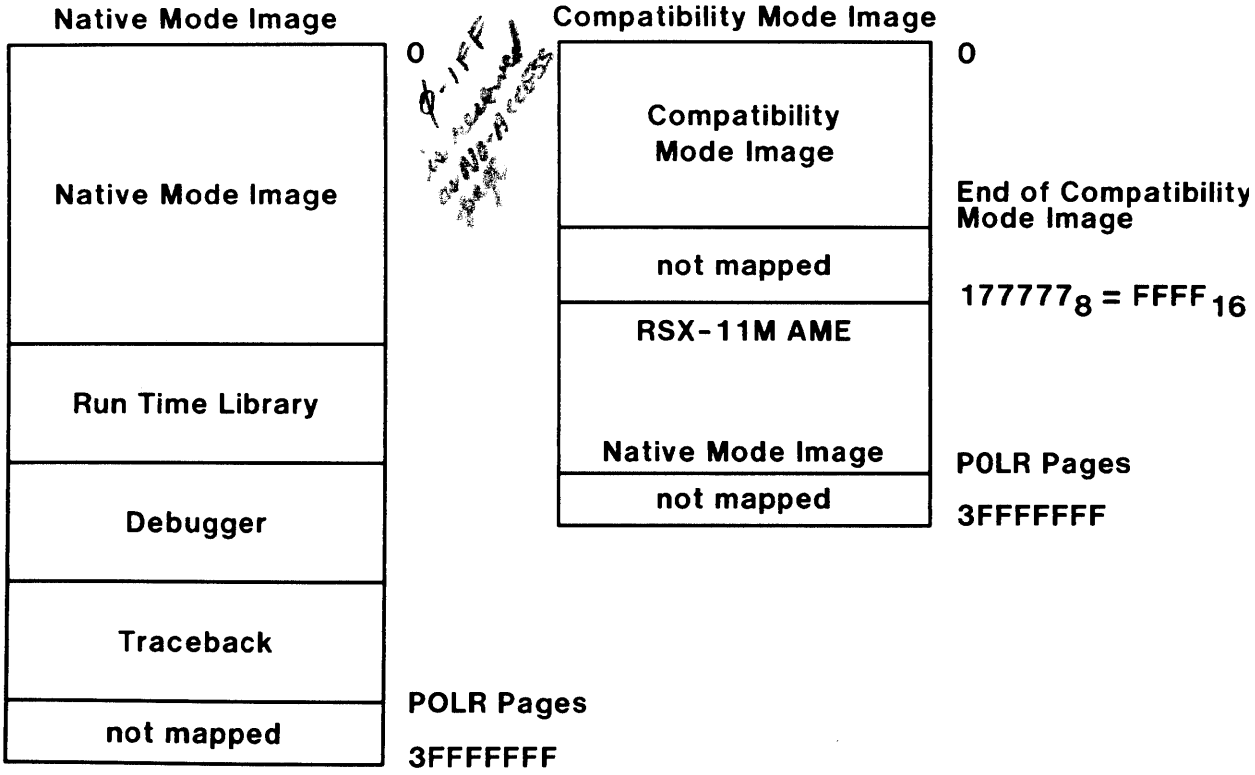


Figure 2-8 PØ Virtual Address Space

P1 VIRTUAL ADDRESS SPACE

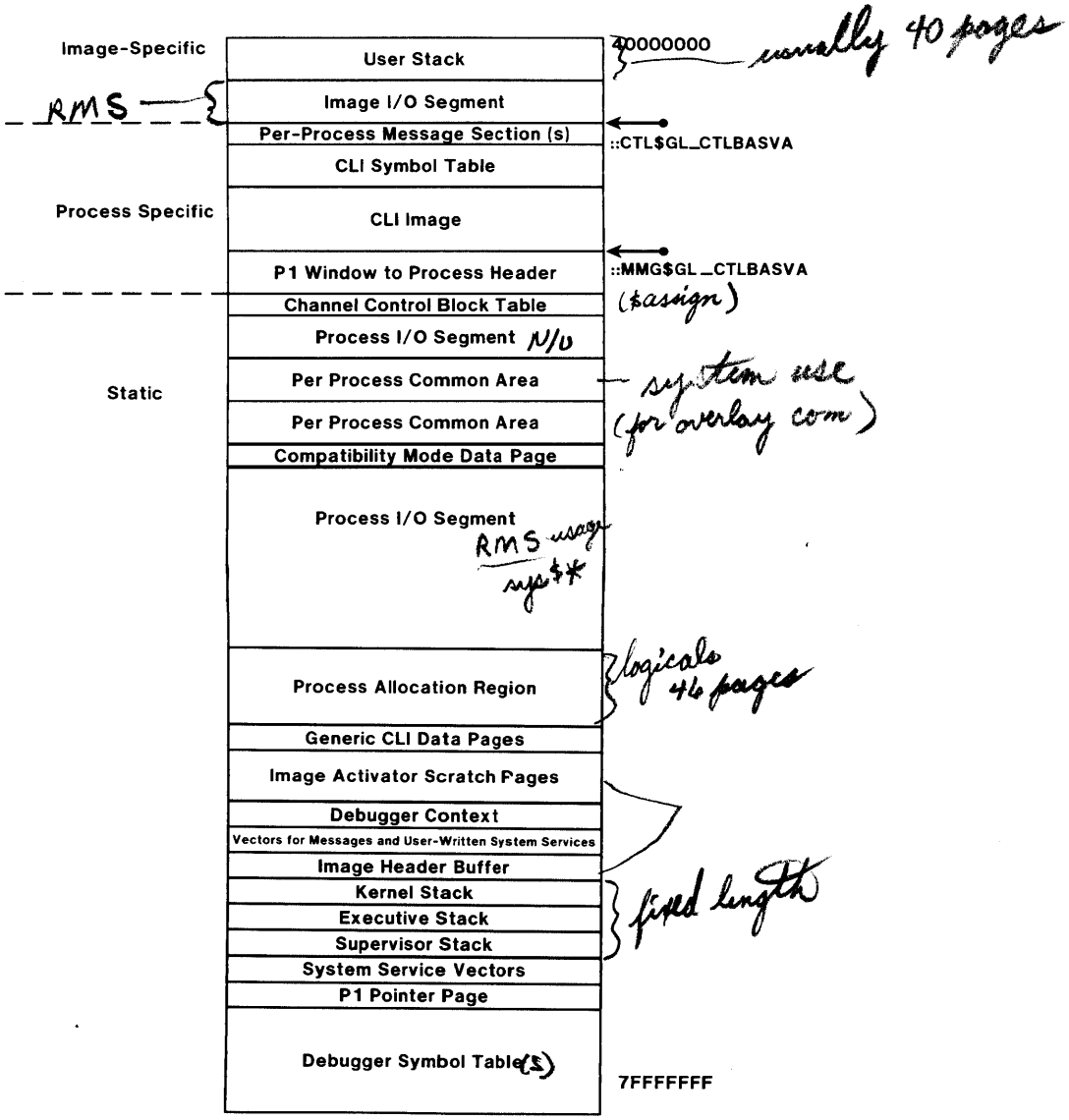


Figure 2-9 P1 Virtual Address Space

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

Table 2-1 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 SØ space)
	P1 pointer page (i.e., CTL$GL_CTLBASVA). (Address of primary, secondary, last change exception)
	Message vectors (per process messages)
Stacks	Kernel, executive, supervisor, user
RMS data • Record of open files	Image I/O segment Process I/O segment
Error message text	Per process message section
Storage area	
• Data stays around between images	Per Process Common Area (LIB$GET_COMMON)
	Process allocation region (i.e., logical names)
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)

SYSTEM MECHANISMS

3

INTRODUCTION

Most 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 case a system "deadlock." Several hardware instructions provide support for these software mechanisms. Additional mechanisms control the accessibility of data structures. The user identification code (UIC), file protection, and privileges are examples of qualitative controls. Quotas and limits are quantitative controls on system resources hat may have several synchronous allocations to a process.

The implementation of an interrupt priority strcture 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.

Several interrupt priority levels are used to protect data structures by assuming that IPL is raised to the required IPL value. There must be a mechanism for seralizing requests for some of these data structures when the requesting software routine is executing at or above the desired IPL. The fork process mechanism provides this ability by requiring that an interrupt service routine or driver first drop IPL below the desired value, then use common code to raise the IPL value. Because the common code only services one request from its queue at a time, competing requests are serialized into exclusive accesses.

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
- Compatibility mode fault.

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).

Privileges provide qualitative controls over classes of operations that may be requested by users. Privileges are granted through:

- The user authorization file
- The $SETPRV system service and the $SET PROCESS/PRIVILEGE command
- Installation of a known image with enhanced privileges
- User identification code (UIC)
- The file protection mechanism.

Quantitative control over user activities is also required, and is provided by several quota mechanisms.

- Process-specific quotas or limits, such as the number of I/O requests, timer requests, and ASTs outstanding
- Job-specific quotas, such as the number of subprocesses and the consumption of page file pages
- User-specific quotas, such as disk consumption (based on UIC and volume/volume set)
- System-wide limits based on system parameters, such as maximum process count.

Mutual exclusion semaphores (mutexes) are a system mechanism used to synchronize concurrent accesses to a protected data structure by more than one process. Mutexes allow one process to write to the protected data structure or multiple processes to read the data structure.

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.

Asynchronous system traps (ASTs) provide an asynchronous method of communication and synchronization between VMS and a process. ASTs are used to:

- Notify a process of the completion of an operation (I/O or a timer request, for example)
- Force execution within the context of a target process (for example, $GETJPI to another process, forced process deletion, and return of status information after an I/O request completes).

Timer requests can be issued by user processes to defer action until some future time. Also, periodic wakeup requests of user processes and periodic executions of system routines are provided by the software timer, with support from the hardware clocks.

OBJECTIVES

Upon completion of this module, you will 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 (e.g., memory), and the effects of altering those parameters.

RESOURCES

Reading

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

Additional Suggested Reading

1. VAX/VMS Internals and Data Structures Manual, chapters on hardware interrupts, and timer support

2. VAX-11 Architecture Handbook, Chapter 12

3. VAX-11/780 Hardware Handbook, Chapter 12

Source Modules

Facility Name	Module Name
SYS	ASTDEL,SCHED CMODSSDSP EXCEPTION,SYSUNWIND MEMORYALC MUTEX 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
RTL	LIBSIGNAL

TOPICS

I. Processor and Process State

II. Interrupts

III. Access Modes and Exceptions

IV. Synchronization

Table 3-1 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

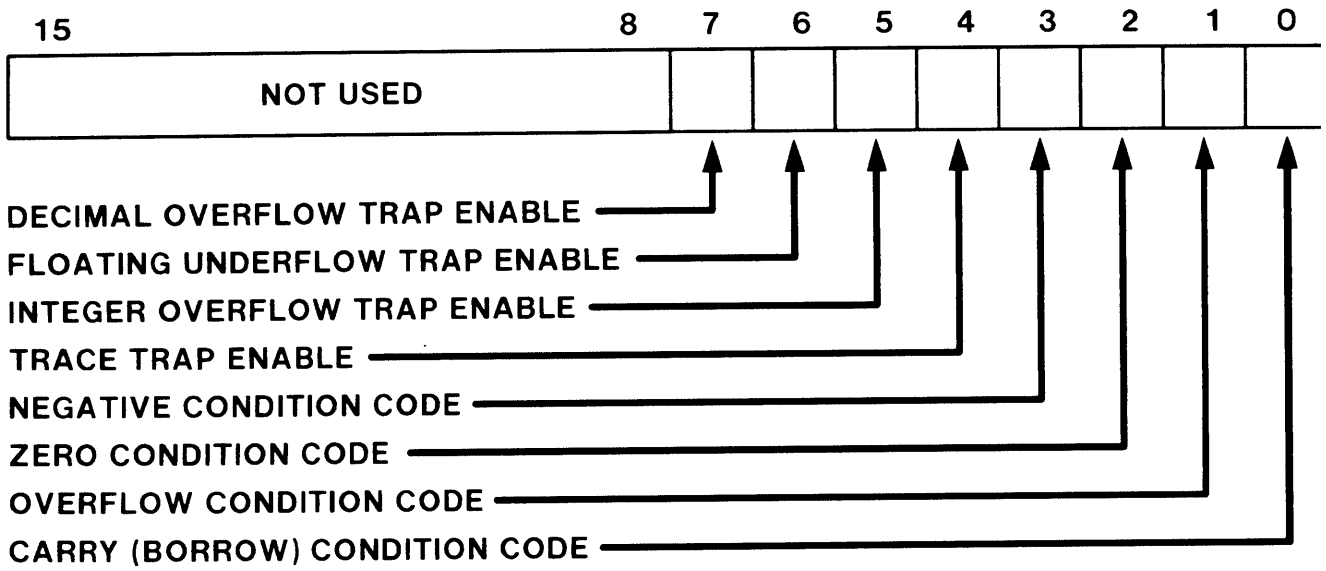


Figure 3-1 Processor Status Word

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

PROCESSOR STATUS LONGWORD

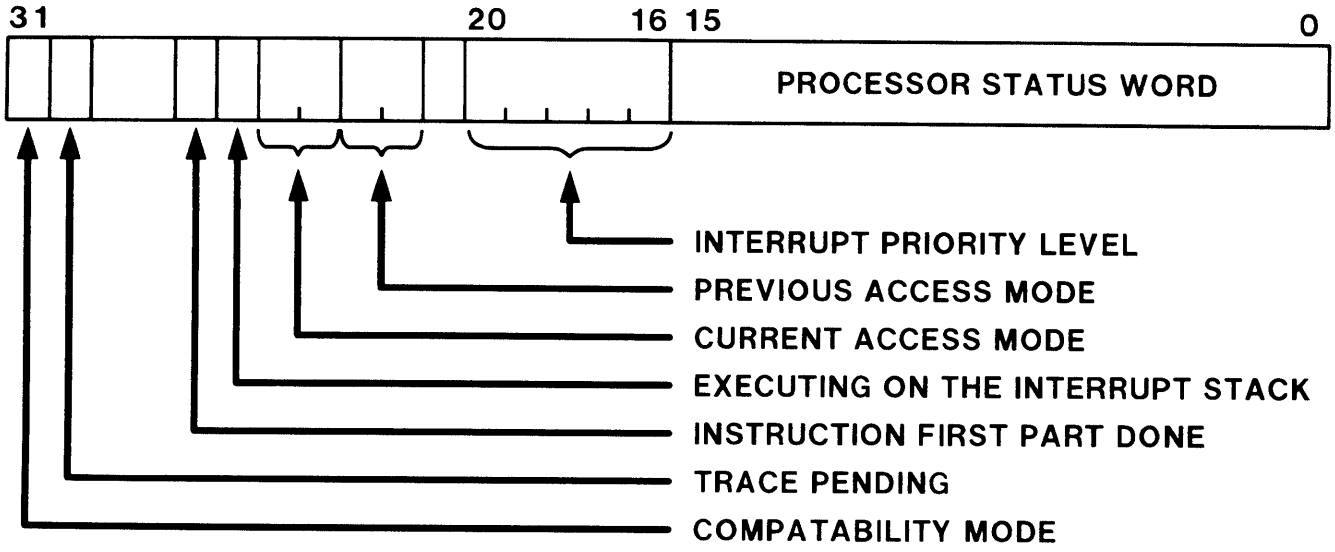


Figure 3-2 Processor Status Longword

- High-order word of Processor Status Longword (PSL)
- Read-only to nonprivileged users
- Changed as a result of CHMx, REI, MTPR instructions.

HARDWARE CONTEXT

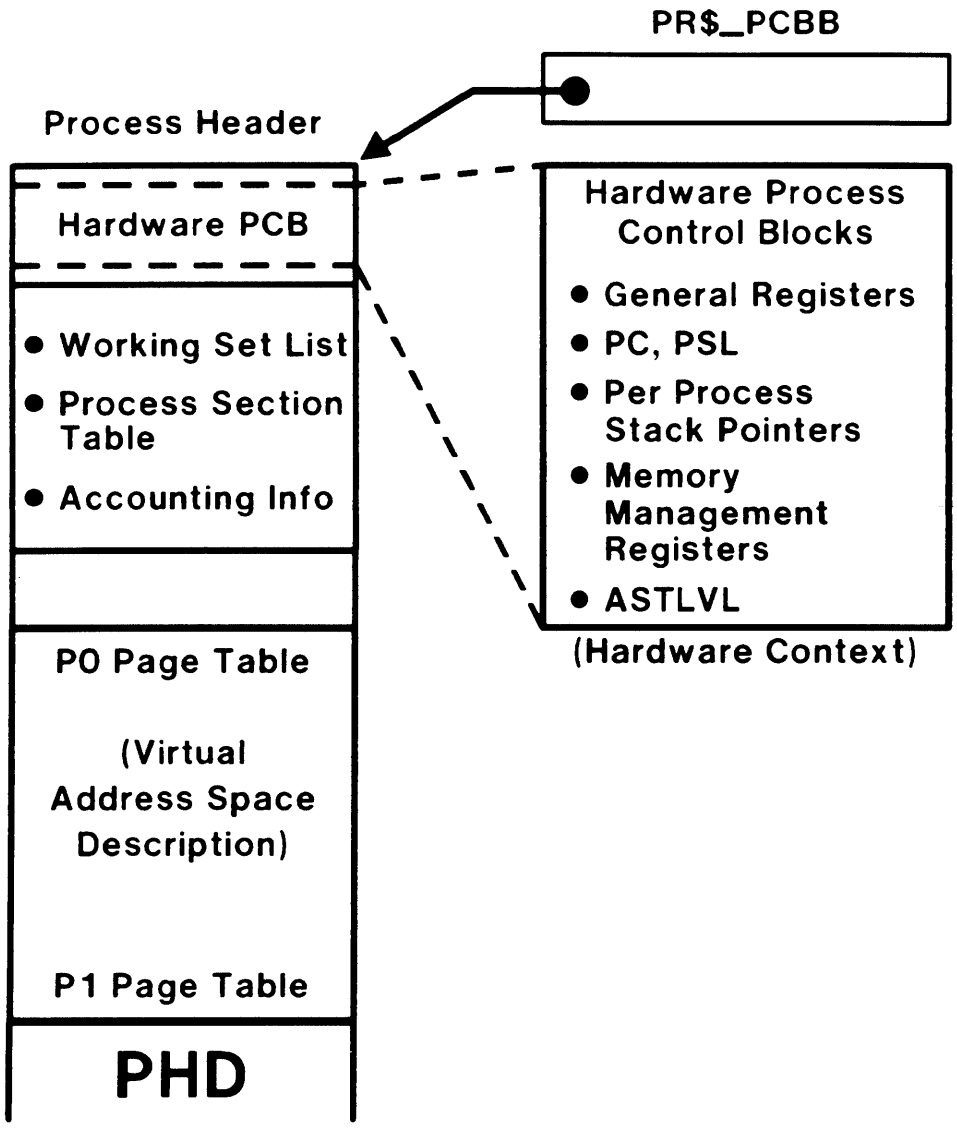


Figure 3-3 Hardware Context

The LDPCTX and SVPCTX instructions move the contents of the process-specific hardware registers to and from the hardware process control blocks of memory-resident processes.

The hardware PCB contains the processor register information for a process when it is not currently executing.

HANDLING AND USES OF INTERRUPTS

Table 3-2 Handling and Uses of Interrupts

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, IOPOST...
Request an interrupt	MACRO	SOFTINT
Synchronize system's access to system data structures	MACRO-raise IPL to 7	SETIPL
Continue execution of code at lower priority	Queue request, SOFTINT, REI	FORK

HARDWARE INTERRUPTS AND SCB

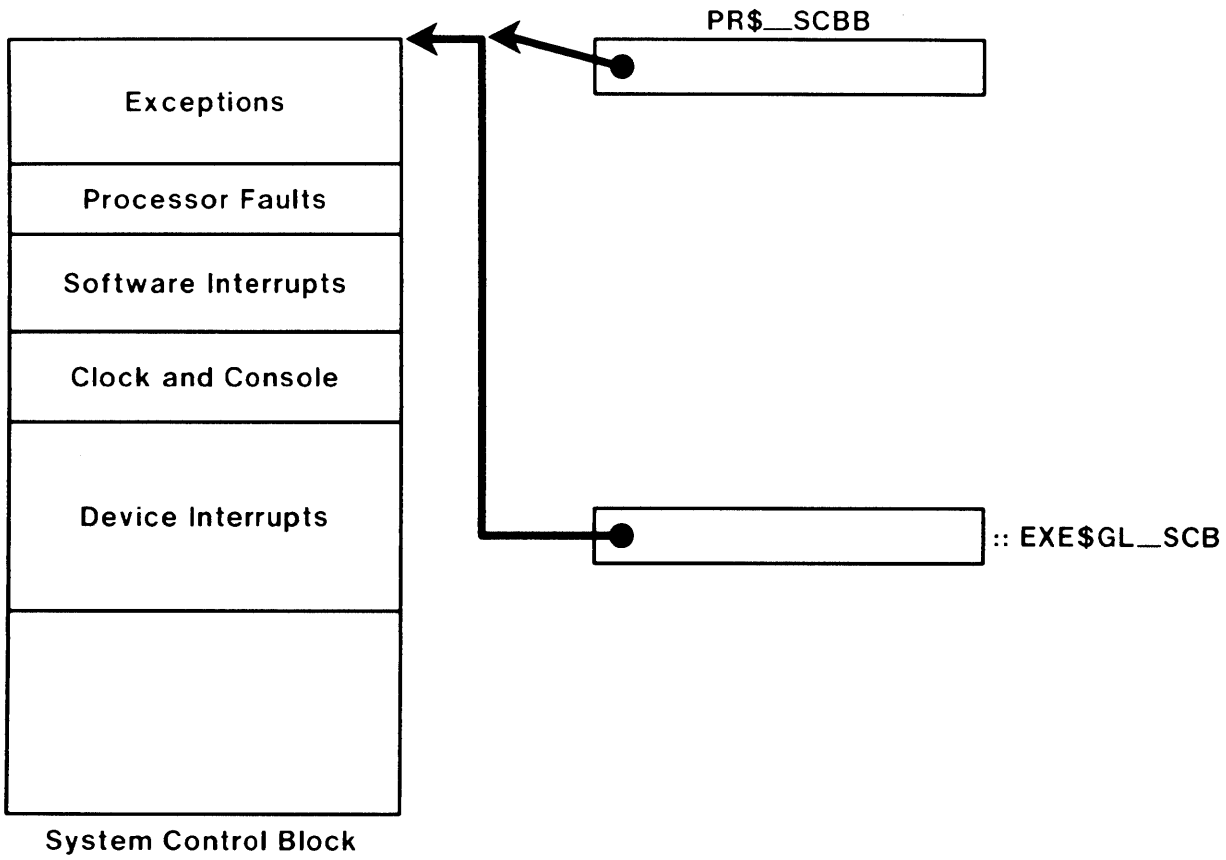


Figure 3-4 Hardware Interrupts and SCB

- 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 3-3 Hardware Interrupts and IPL

FUNCTION	VALUE (decimal)	NAME
Block all Interrupts	31	IPL$_POWER
Clock Interrupts	24	IPL$_HWCLK
Device Interrupts	20-23	UCB$B_DIPL*
Driver Fork Levels	8-11	UCB$B_FIPL*

* Offset into Device's Unit Control Block

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

  UCB$B_XXXX - Offset into UCB where device IPL and fork IPL are stored (see $UCBDEF).

SOFTWARE INTERRUPTS AND SCB

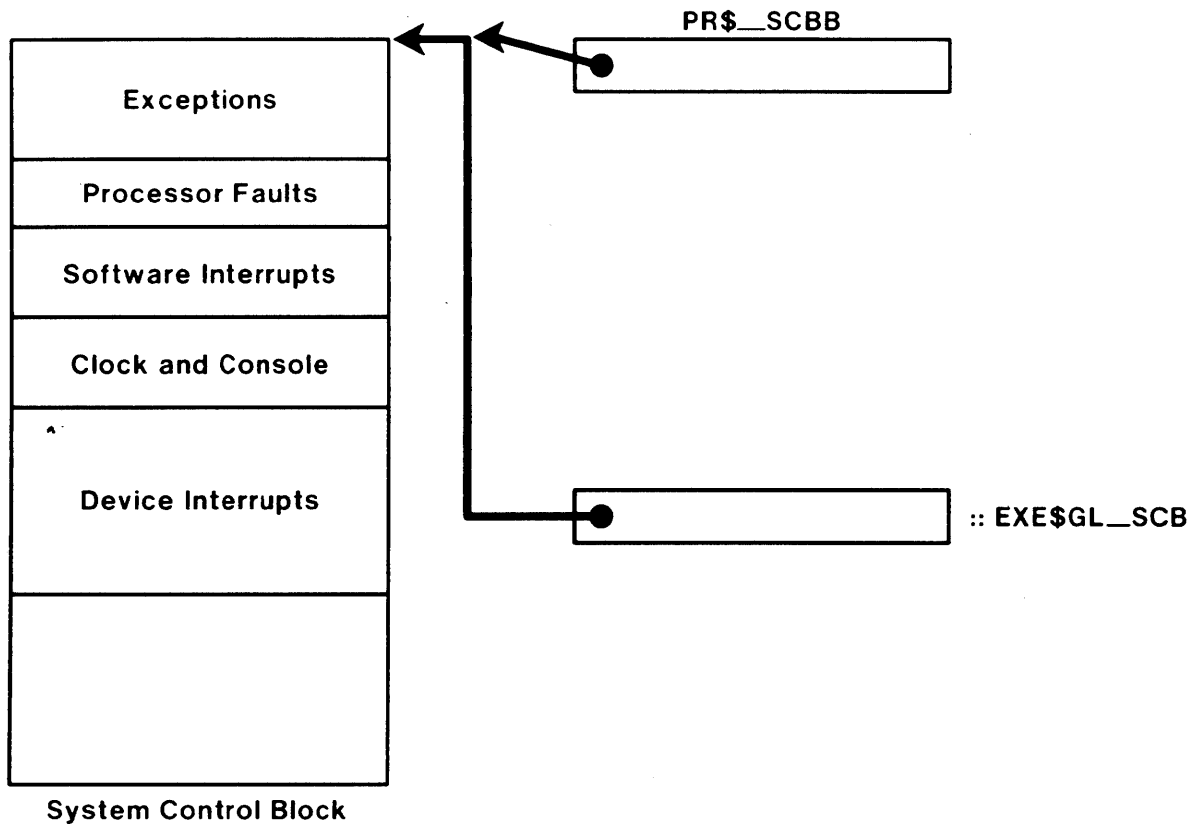


Figure 3-5 Software Interrupts and SCB

Hardware gets service routine address from longword in SCB.

SOFTWARE INTERRUPTS AND IPL

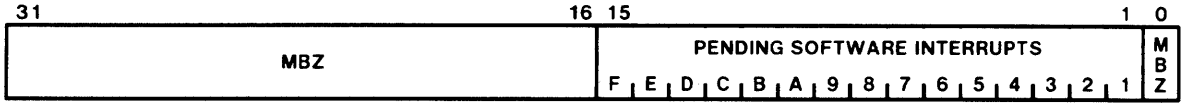
Table 3-4 SOFTWARE INTERRUPTS AND IPL

IPL	USE
15-12	Unused
11	IPL=11 Fork Dispatching
10	(IPL=10 Fork Dispatching)
9	(IPL=9 Fork Dispatching)
8	IPL=8 Fork Dispatching
7	Software Timer Service Routine
6	IPL=6 Fork Dispatching
5	Used to Enter XDELTA
4	I/O Post Processing
3	Rescheduling Interrupt
2	AST Delivery Interrupt
1	Unused

- Interrupt Priority Levels (IPL) 1 through 15, reserved for software interrupts
- IPLs 2-11 currently used.

SOFTWARE INTERRUPT REQUESTS

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



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

Figure 3-6 Software Interrupt Requests

Software Interrupt Summary Register

- Bits 1 through 15 correspond to IPL 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
```

LOWERING IPL

IPL 23 interrupt

Driver int. service routine

- Processing at IPL 23
- Queue UCB to fork dispatcher (PC in UCB)
- Request IPL 8 interrupt
- REI

IPL 8 interrupt

Fork dispatcher service routine

If queue empty, REI

Dequeue UCB

JSB to PC in UCB

Loop

PC: Driver
Processing @IPL 8

RSB

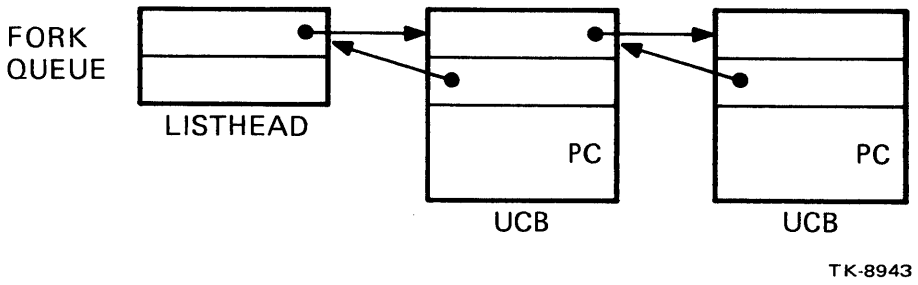


Figure 3-7 Fork Queue

BLOCKING INTERRUPTS

Table 3-5 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$B_DIPL*
Access to Scheduler's Data Structures	7	IPL$_SYNCH
Delivery of ASTs (Prevent Process Deletion)	2	IPL$_ASTDEL

* Offset into Device's Unit Control Block

IPL can also be used to block interrupt servicing. IPL$_SYNCH is used to coordinate accesses to critical VMS data structures, such as the software PCBs within the scheduler's database.

RAISING IPL TO SYNCH

Convention: raise to IPL$_SYNCH (IPL 7) to access scheduler data base (PCBs, PHDs, etc).

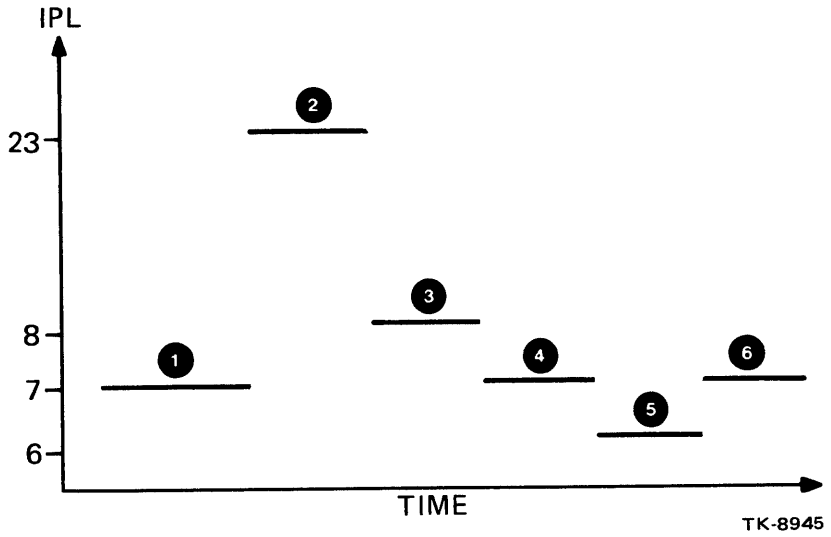


Figure 3-8 Raising IPL to SYNCH

1. Software timer at IPL 7
2. Device interrupt - driver code at IPL 23
3. Driver forks to IPL 8, then requests IPL 6 interrupt
4. Software timer at IPL 7
5. Driver fork to IPL 6
6. Driver raised to IPL 7

HOW USER EXECUTES PROTECTED CODE

Table 3-6 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

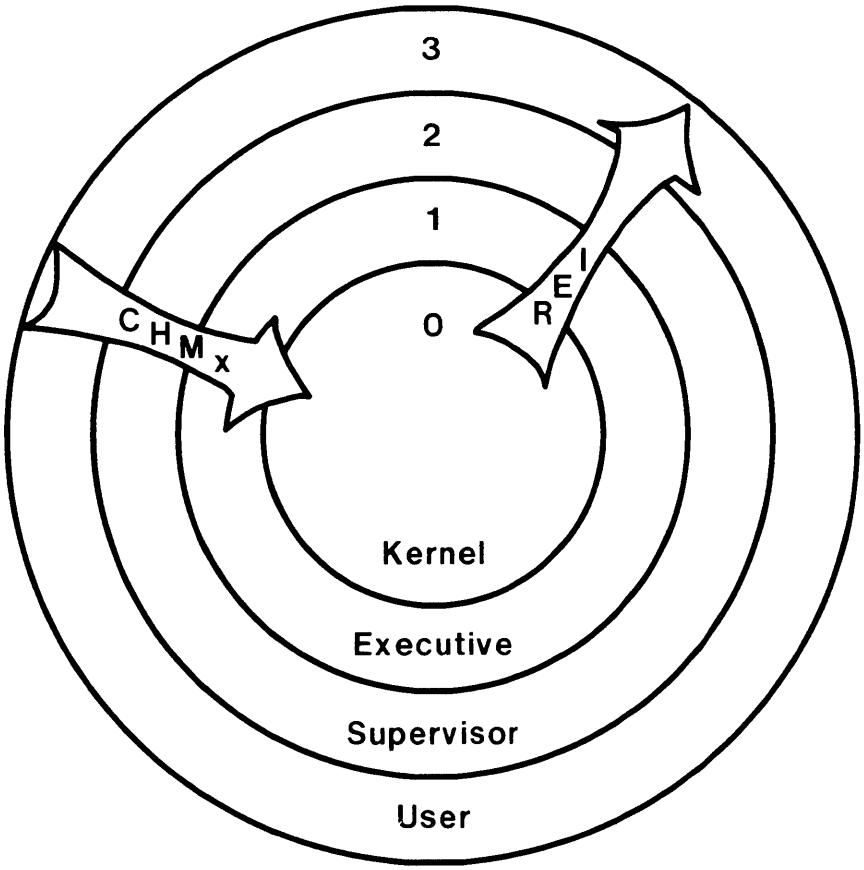


Figure 3-9 Access Mode Transitions

CHMx:

- Only way to move from less privileged to more privileged access modes

REI:

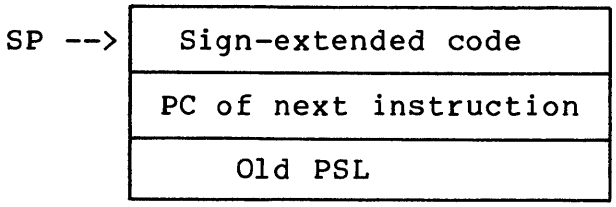
- Only way to move from more privileged to less privileged access modes
- Checks for illegal or unauthorized transitions.

CHMX AND REI INSTRUCTIONS

CHMX

CHMx CODE#

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



Sign-extended code
PC of next instruction
Old PSL

- PSL zeroed (except for IPL, CM, PM)
- Current mode of PSL moved to previous mode field
- Current mode changed to appropriate mode
- New PC taken from system control block (SCB).

REI

Replaces current PC and PSL with two longwords popped from the stack.

Before doing so,

1. Various checks are made to protect the integrity of the system.
2. Checks for pending ASTs.
3. Checks for pending software interrupts.
4. After placing the PC and PSL in temporary registers, the SP is switched to the appropriate access mode based on the current mode field of the PSL.

REI OCCURS IN FOUR DIFFERENT CONTEXTS

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

```
CHMx CODE
     .
     .
     .
REI
```

2. To service and dismiss a hardware interrupt:

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

3. To switch to compatibility mode:

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

4. To service and dismiss a software interrupt:

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

INTERRUPTS VS. EXCEPTIONS

Table 3-7 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.

3 exceptions:
abort ??? god knows!
trap (PC)
fault (PC-1)

EXCEPTIONS AND SCB

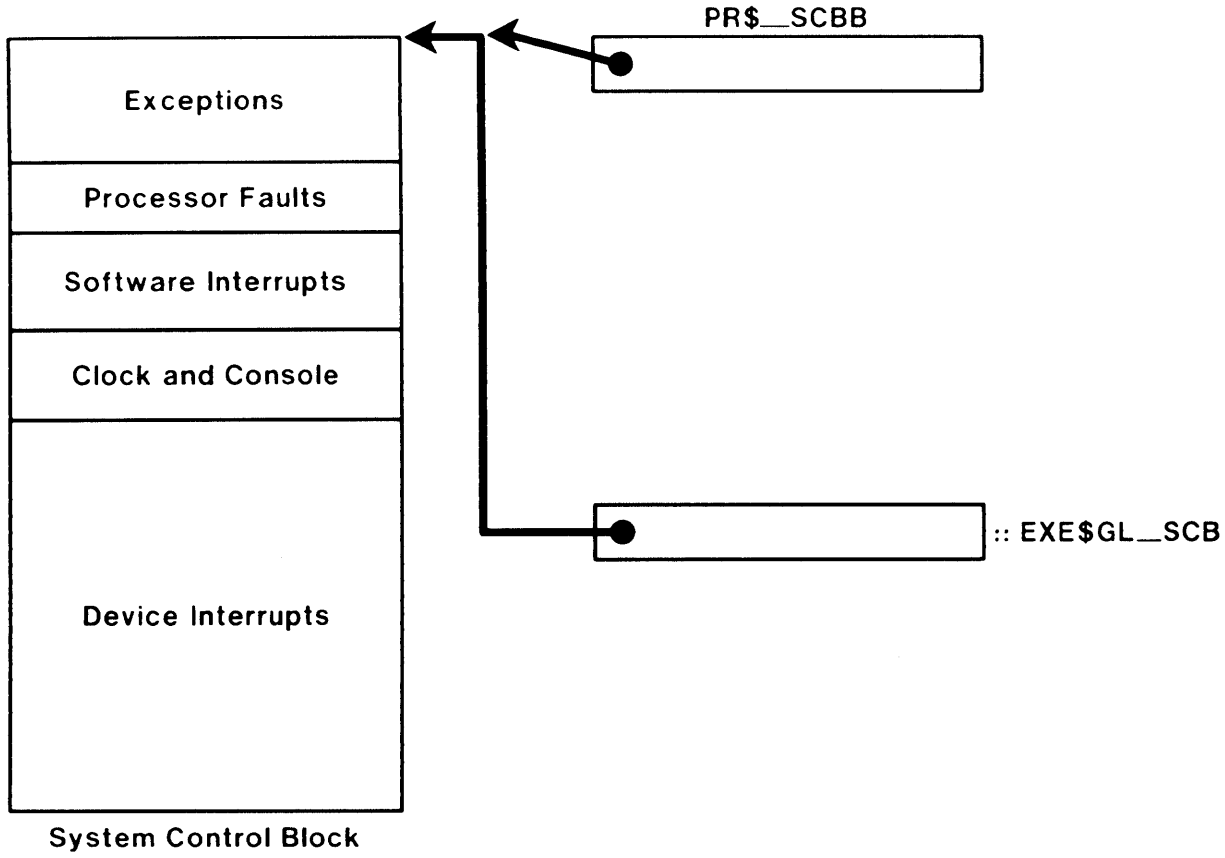


Figure 3-10 Exceptions and SCB

Exceptions are serviced by system routines that are initially dispatched through the system control block.

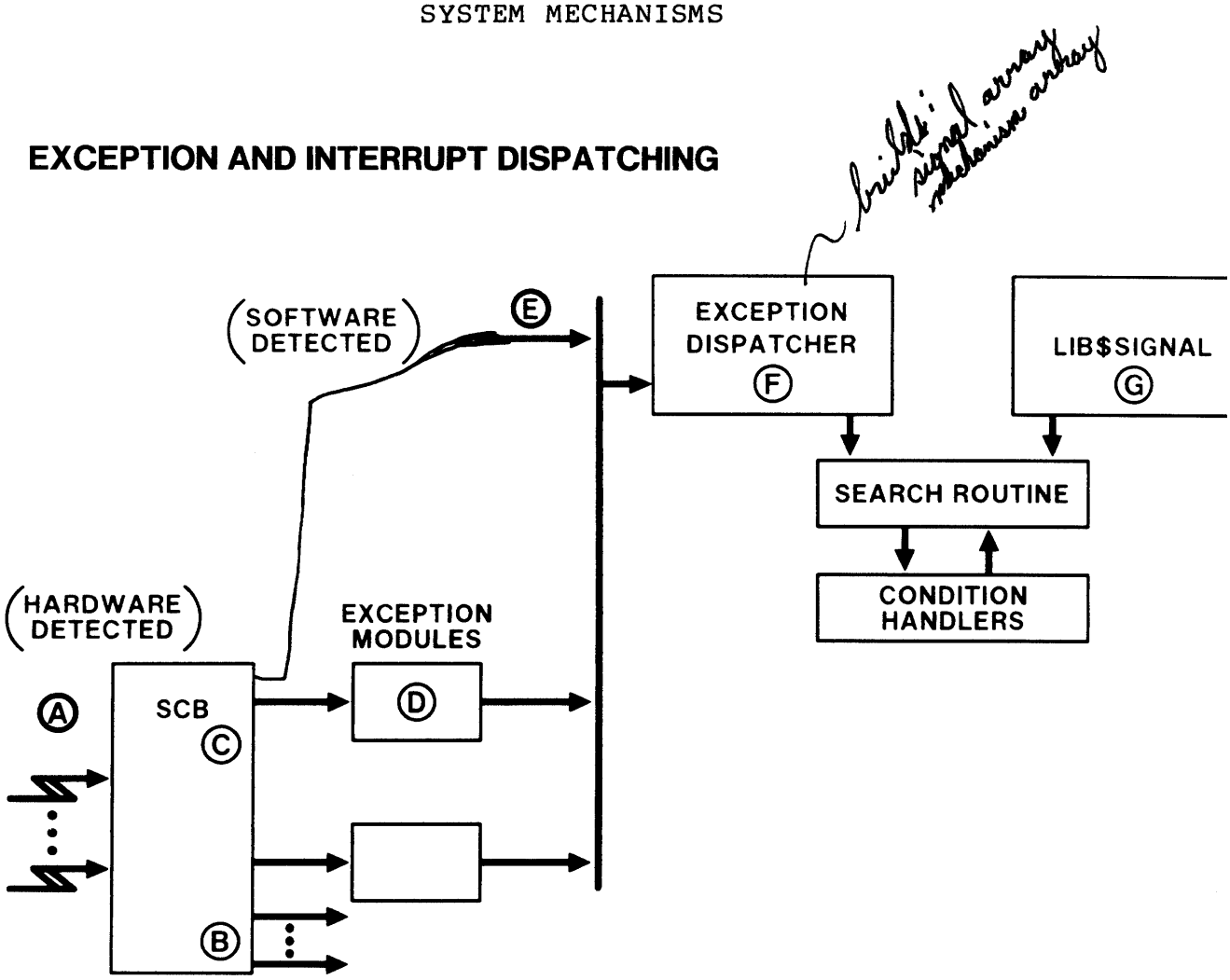


Figure 3-11 Exception and Interrupt Dispatching

Notes to Figure 3-11:

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

2. Exceptions/interrupts treated by VMS

3. Exceptions not treated by VMS

4. Routines that push "SS$exception_name" and "N" (total of 1w now in completed signal array) onto stack. Notes 1 + 2 = signal array.

5. Detected and signaled by Executive

6.

    a. Builds mechanism array and argument

    b. Search order

        1. Primary exception
        2. Secondary exception
        3. Call frames (maximum 64K)
        4. Last chance

7. Alternate condition handling mechanism

    a. Signaled by RTL or a user

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

PATH TO SYSTEM SERVICE

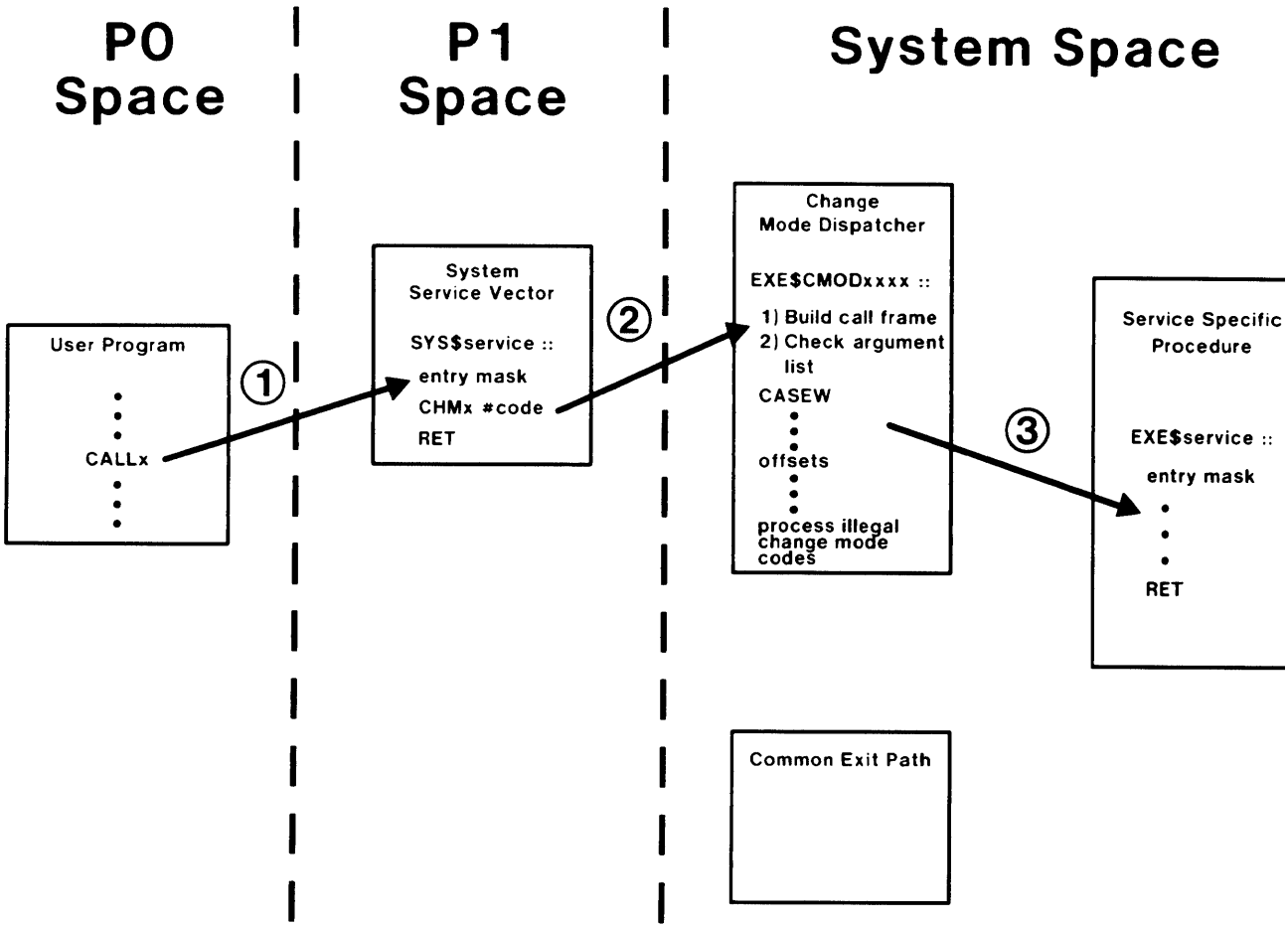


Figure 3-12 Path to System Service

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

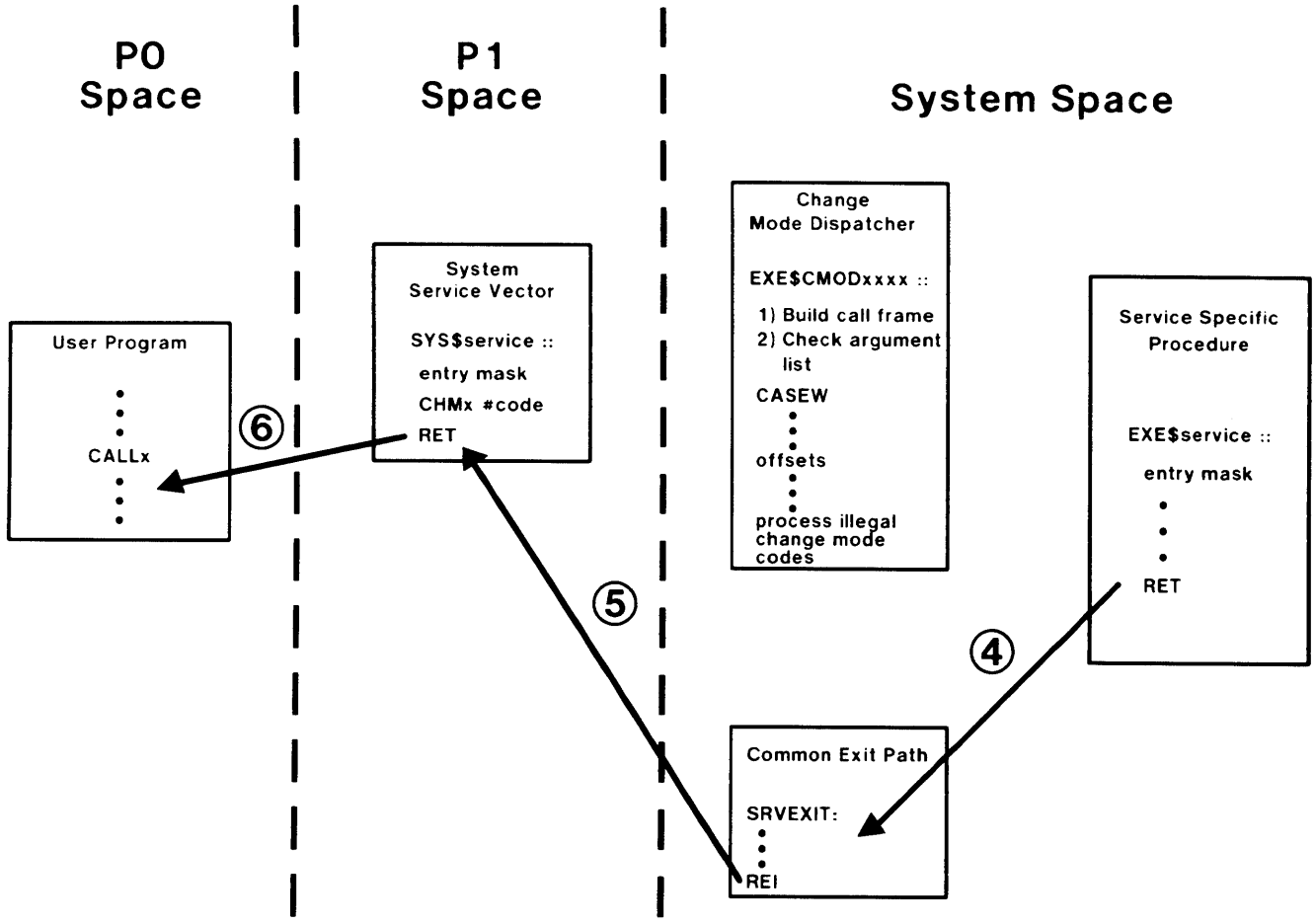


Figure 3-13 Return from System Service

4. System services return through a common code sequence, SRVEXIT, that checks the return status code and causes a system service failure exception if that feature has been enabled and the service has failed.

5. REI from CHMx exception service routine.

6. RET for original CALL.

NONPRIVILEGED SYSTEM SERVICE

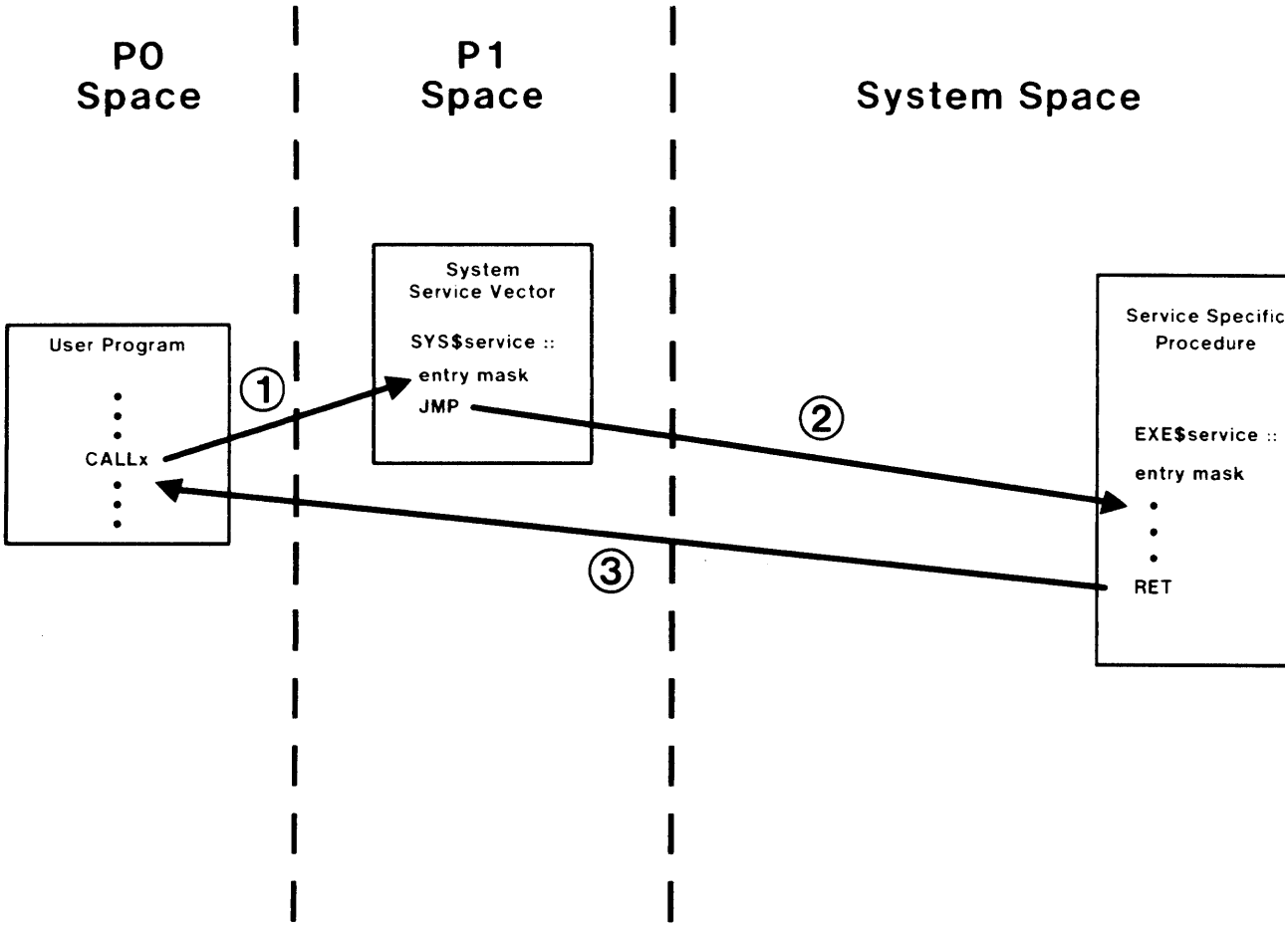


Figure 3-14 Nonprivileged System Service

1. System services that do not require a change of access mode have a simpler control passing sequence.
   - $FAO
   - Timer conversion services
2. Same as 1.
3. These services are not checked by SRVEXIT for error status codes.

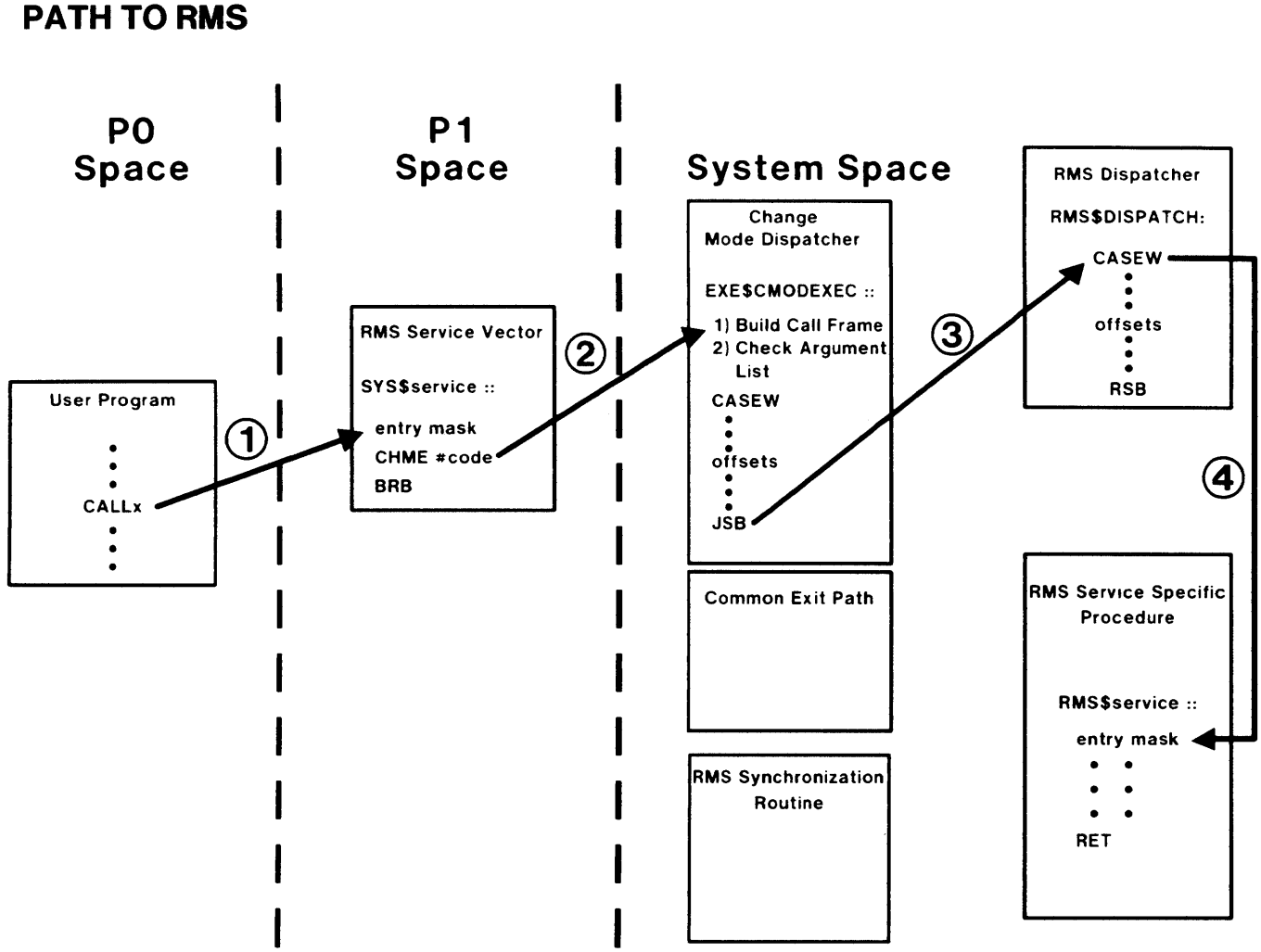


Figure 3-15 Path to RMS

1. Same path as executive mode system service.
2. Same as 1.
3. Falls off end of system service case table JSB to RMS case table.
4. Dispatch to RMS procedure.

RETURN FROM RMS

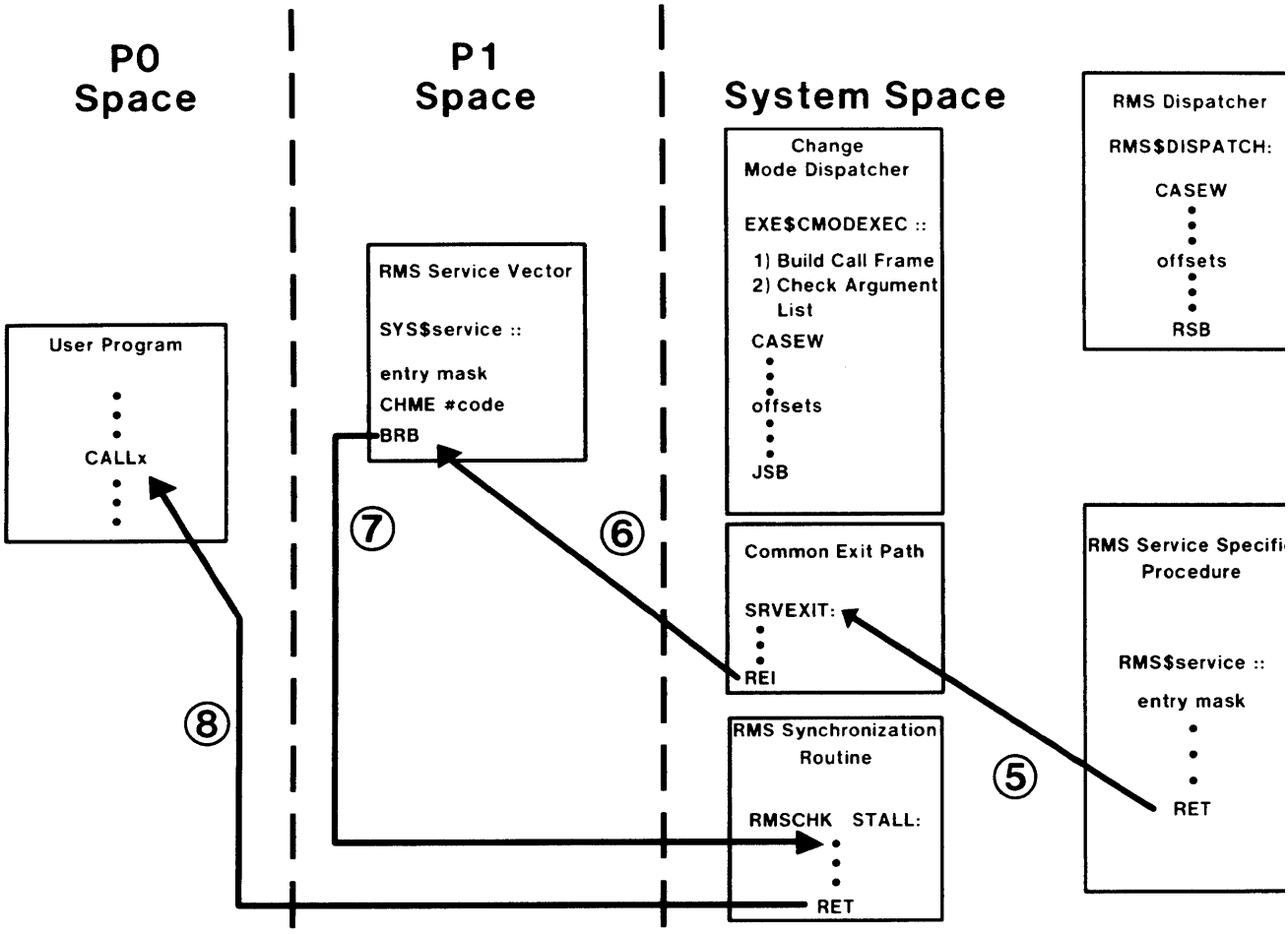


Figure 3-16 Return from RMS

❺ Same path as system service.

❻ Same as 1.

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

❽ RET for original CALL.

PATH TO USER WRITTEN SERVICE (1)

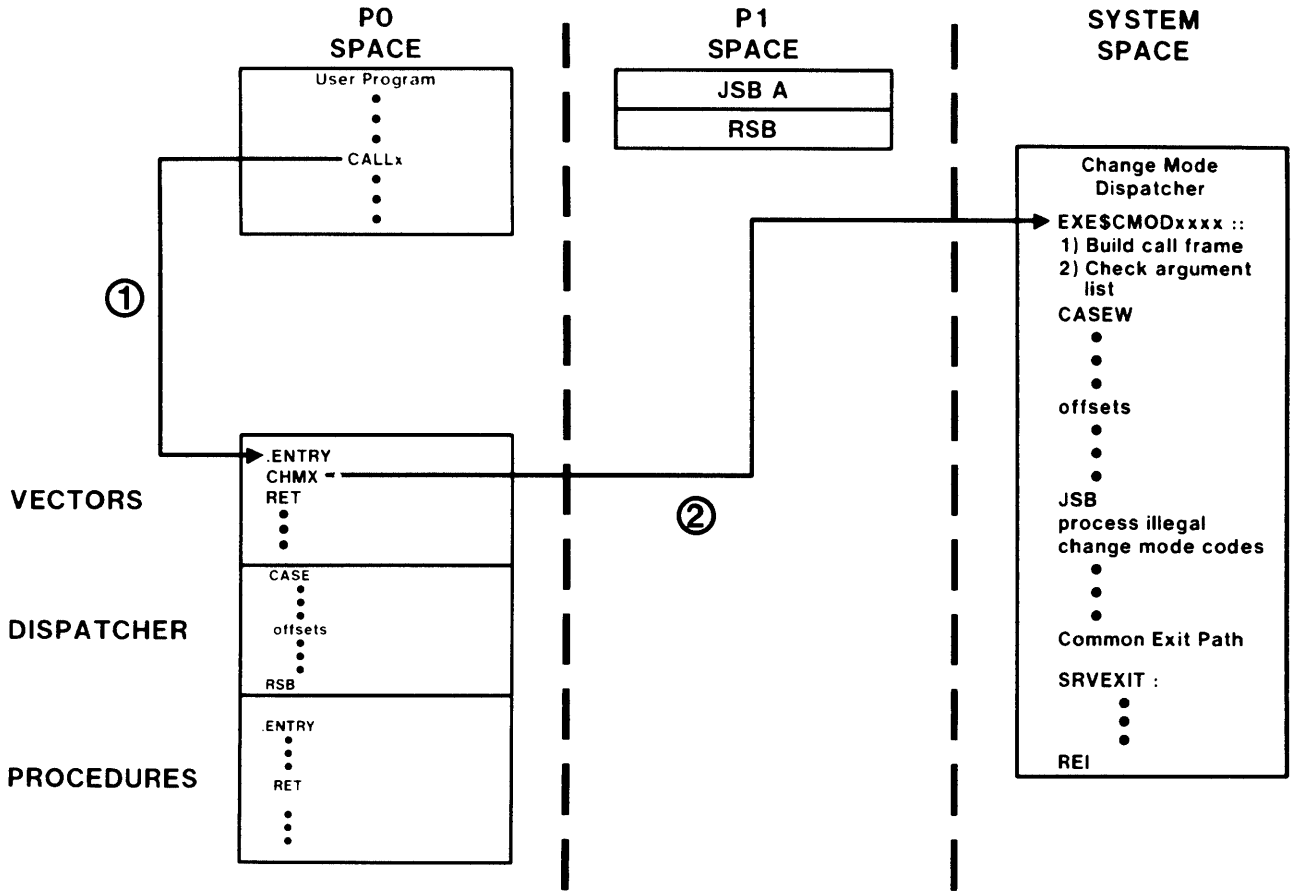


Figure 3-17 Path to User Written System Service (Sheet 1 of 2)

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

❷ A change mode instruction with a negative code causes the change mode dispatcher to look for user-written system service dispatchers in the image.

PATH TO USER WRITTEN SERVICE (2)

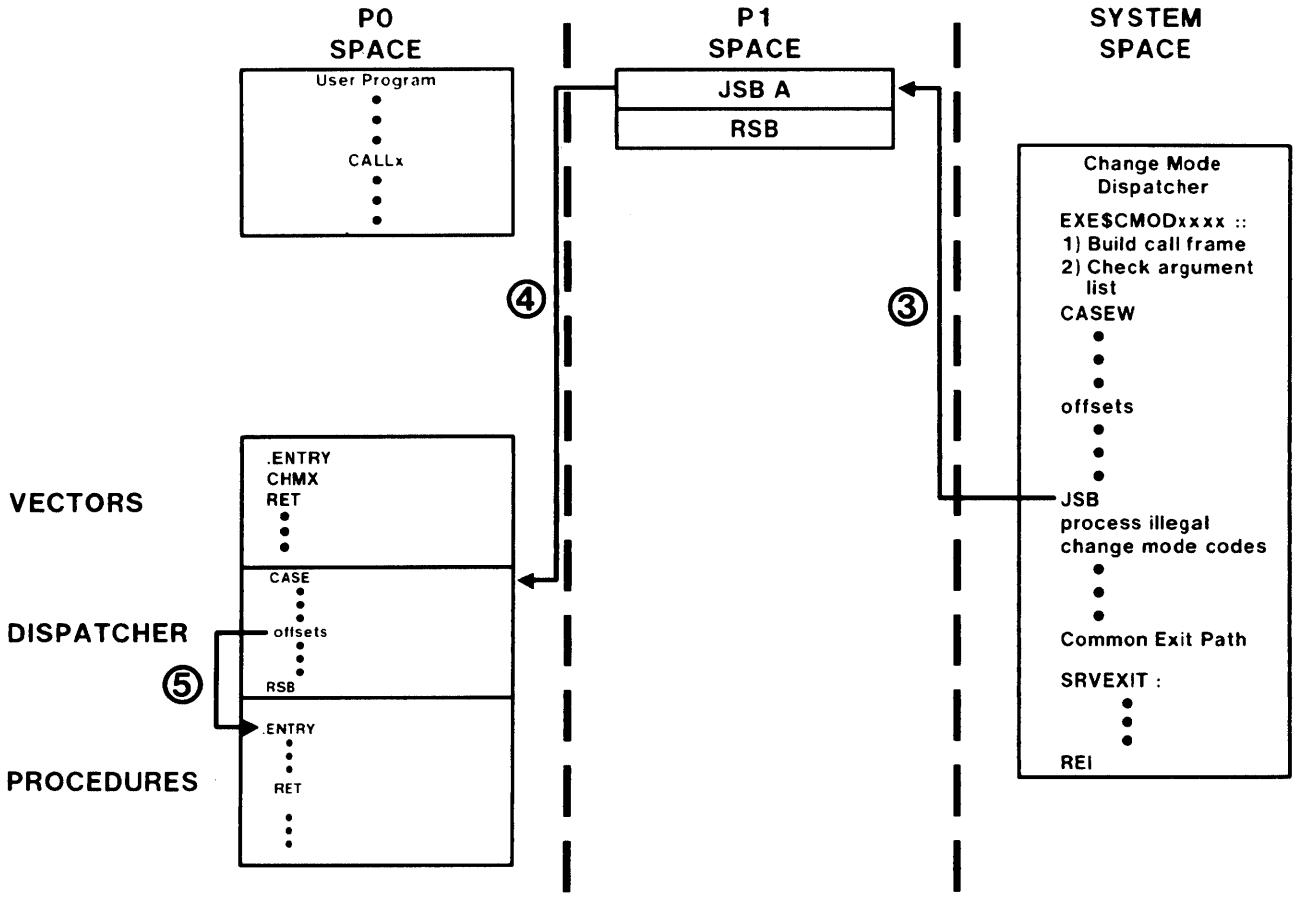


Figure 3-17 Path to User Written System Service
(Sheet 2 of 2)

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

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

❺ Same as 4.

RETURN FROM USER WRITTEN SERVICE

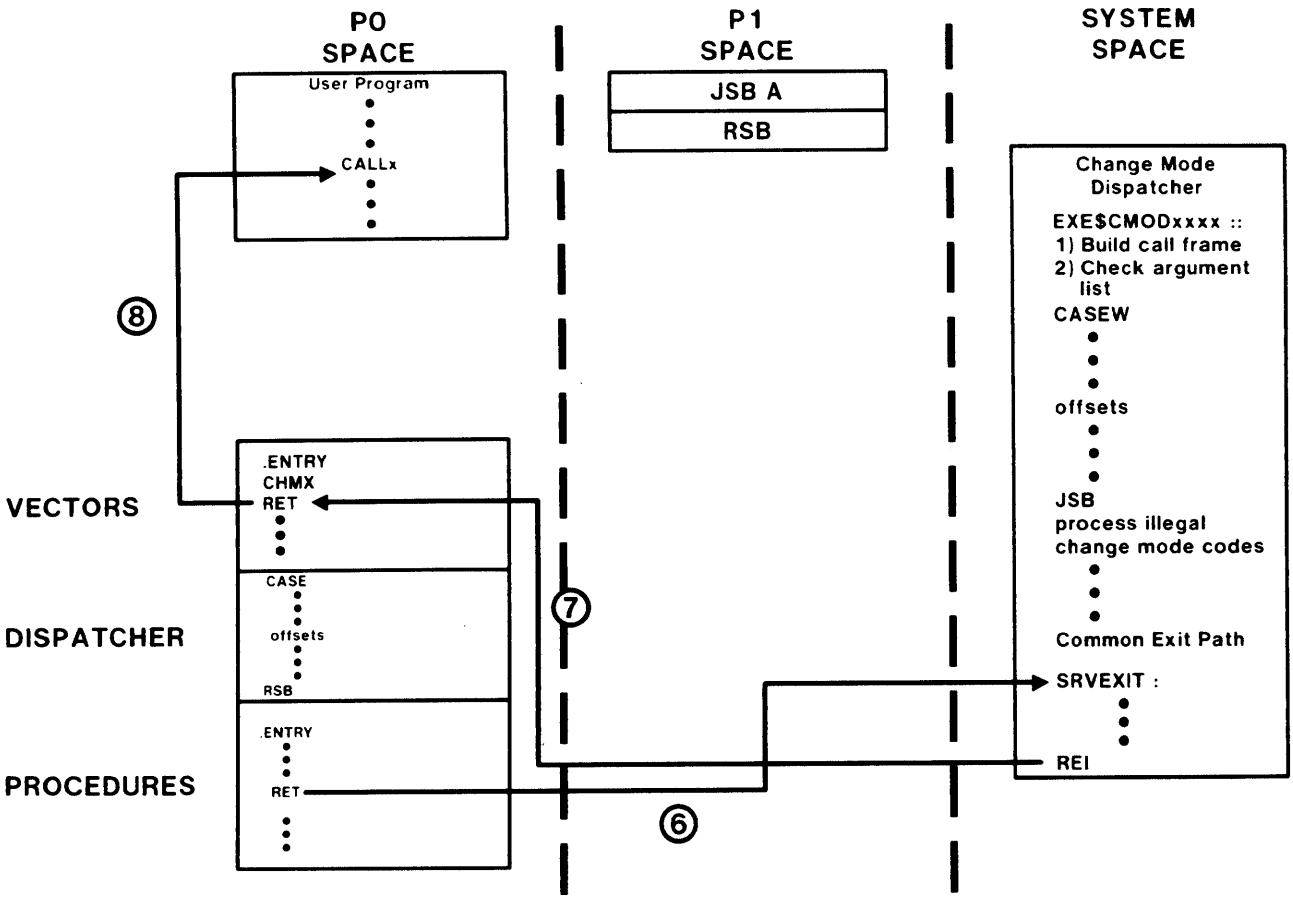


Figure 3-18 Return from User Written System Service

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

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

❽ Same as 7.

TWO DISPATCHERS

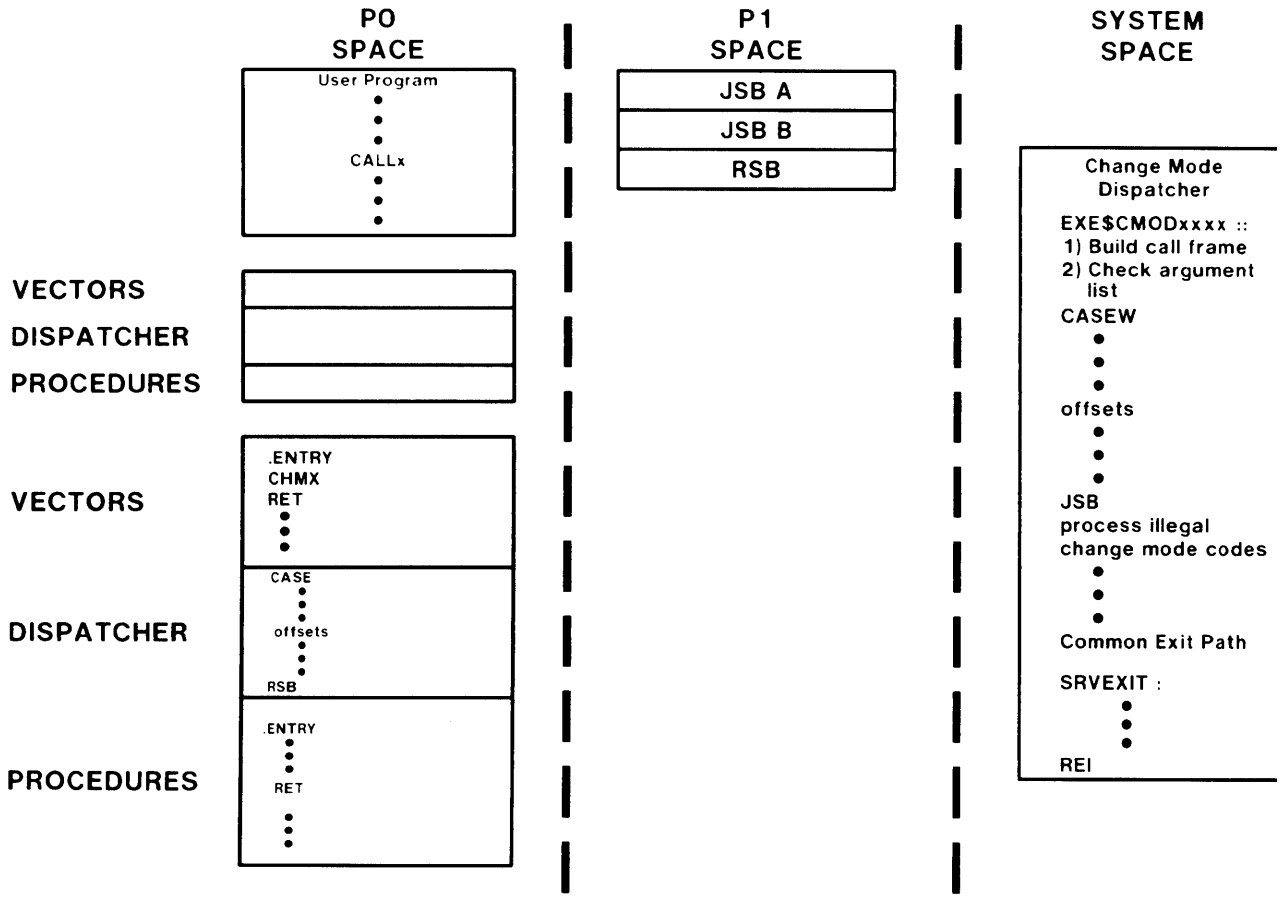


Figure 3-19 Two Dispatchers

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

PROCESS SYNCHRONIZATION

Table 3-8 Process Synchronization

Function	Implementation	Name
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 process to request action at a certain time	Hardware clock interrupt Queue of requests	Timer queue

MUTEX

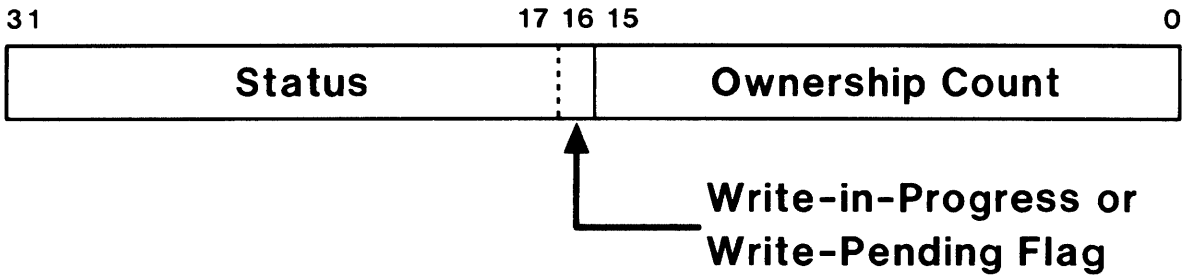


Figure 3-2Ø A Mutex

- Mutexes protect lists of data structures against conflicting accesses by multiple processes.
- One writer or multiple readers are allowed.
- Examples:
  - Group logical name tables
  - System logical name table

OBTAINING AND RELEASING MUTEXES

To Obtain a Mutex

```
MOVAL G^IOC$GL_MUTEX,RØ
MOVL  G^SCH$GL_CURPCB,R4
JSB   G^SCH$LOCKR     ;read
      G^SCH$LOCKW     ;write

When returns, have mutex,
but R2-R5 altered, at
IPL=2
```

To Release a Mutex

```
MOVAL G^IOC$GL_MUTEX,RØ
MOVL G^SCH$GL_CURPCB,R4
JSB G^SCH$UNLOCK
SETIPL #Ø

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

DYNAMIC MEMORY

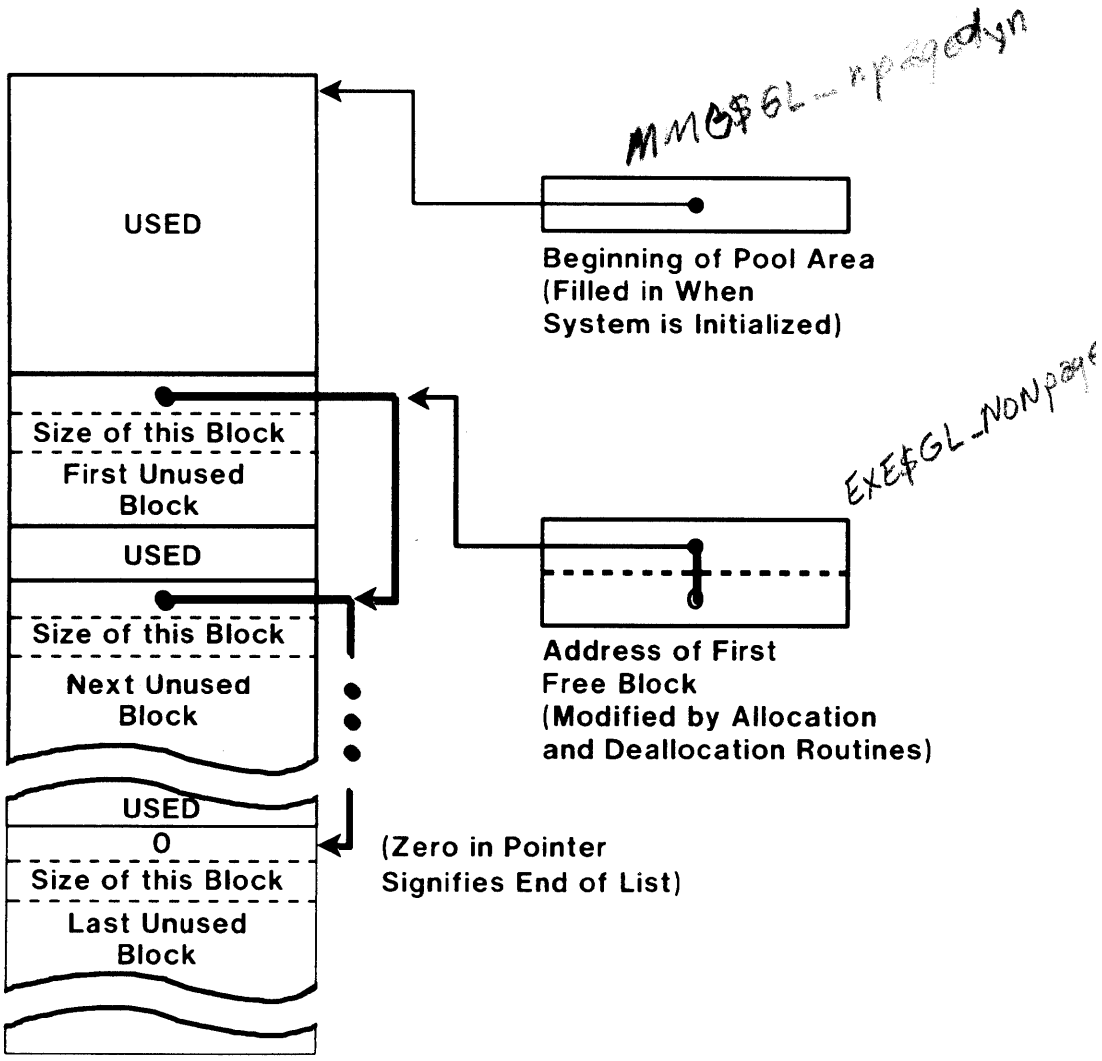


Figure 3-21 Dynamic Memory

Dynamic memory regions, or pools, are 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.

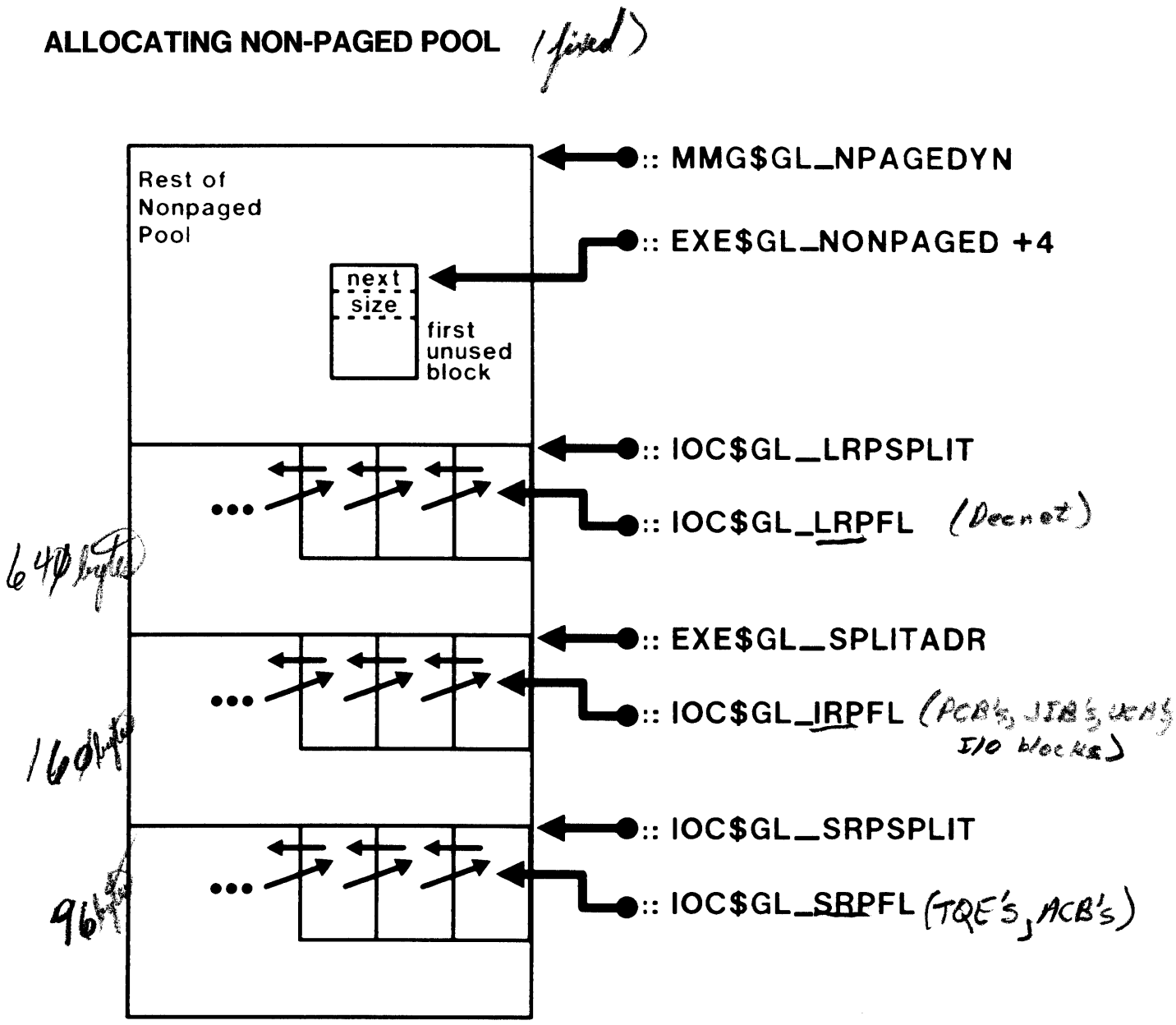


Figure 3-22 Allocating Non-Paged Pool

RELEVANT SYSGEN PARAMETERS FOR NON-PAGED POOL

NPAGEDYN - number of bytes pre-allocated for the non-paged dynamic pool, exclusive of the lookaside lists

NPAGEVIR - number of bytes to which the non-paged pool may be extended.

LRPCOUNT - number of large request packets pre-allocated for the LRP lookaside list.

LRPCOUNTV - number of LRPs to which the LRP list may be extended.

LRPSIZE - number of bytes to allocate per LRP, exclusive of header. Number of bytes actually allocated per packet is LRPSIZE + 64.

IRPCOUNT - number of I/O request packets pre-allocated for the IRP lookaside list.

IRPCOUNTV - number of IRPs to which the IRP list may be extended.

SRPCOUNT - number of small request packets pre-allocated for the SRP lookaside list.

SRPCOUNTV - number of SRPs to which the SRP list may be extended.

SRPSIZE - number of bytes to allocate per SRP.

Notes:

1. System page table entries are reserved and physical memory pre-allocated for NPAGEDYN, LRPCOUNT, IRPCOUNT, and SRPCOUNT.

2. System page table entries are reserved but no physical memory pre-allocated for NPAGEVIR, LRPCOUNTV, IRPCOUNTV, and SRPCOUNTV. Physical memory will be allocated on demand from the free page list.

3. Size of IRP's is 160 bytes.

ASTs

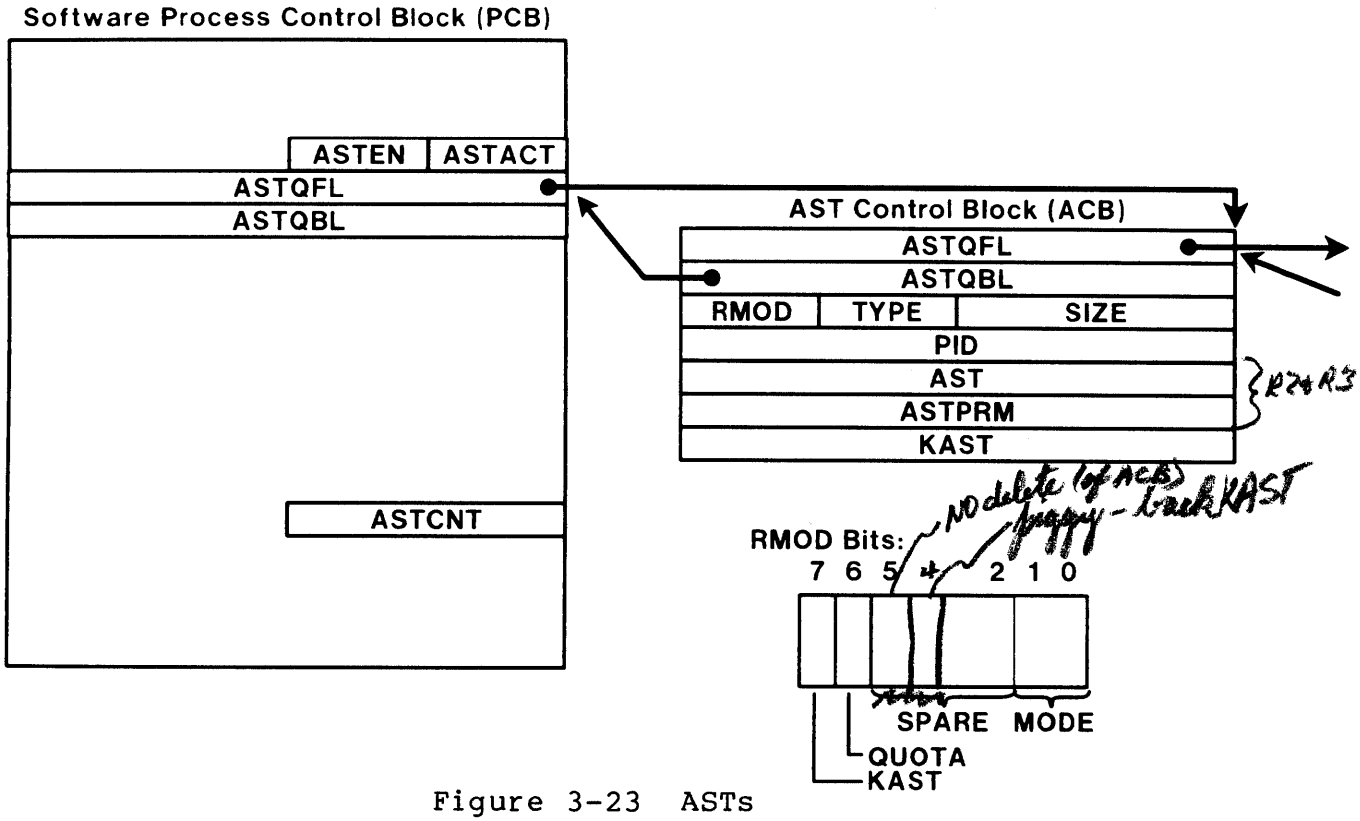


Figure 3-23 ASTs

AST control blocks are queued to the software process control block based on:

- The access mode in which the associated request was made
- The order in which the ASTs are reported to the system.

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.

Rules for AST Delivery

AST DELIVERY ORDER:

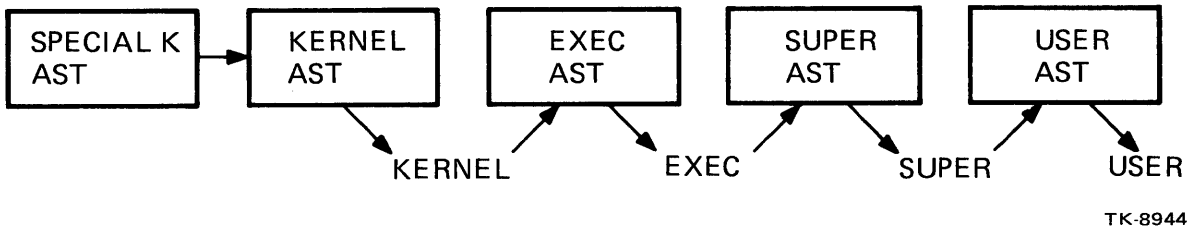


Figure 3-24 AST Delivery Order

- AST control blocks (ACBs) are queued to the PCB in order (Special K, Kernel, Exec, Super, User).
- ASTLVL = 4 if no ACBs, AST delivery is disabled or an AST is Active

  ASTLVL = Ø if AST is Special K AST;

  ASTLVL = level of first AST

Note:

ASTLVL	Deliverable ASTs	Access Mode	Code
Ø	Kernel	Kernel	Ø
1	Exec	Exec	1
2	Super	Super	2
3	User	User	3
4			

Some Special K ASTs

- I/O completion
- Suspend another process
- GETJPI on another process.

AST Delivery Sequence

Sch$QAST

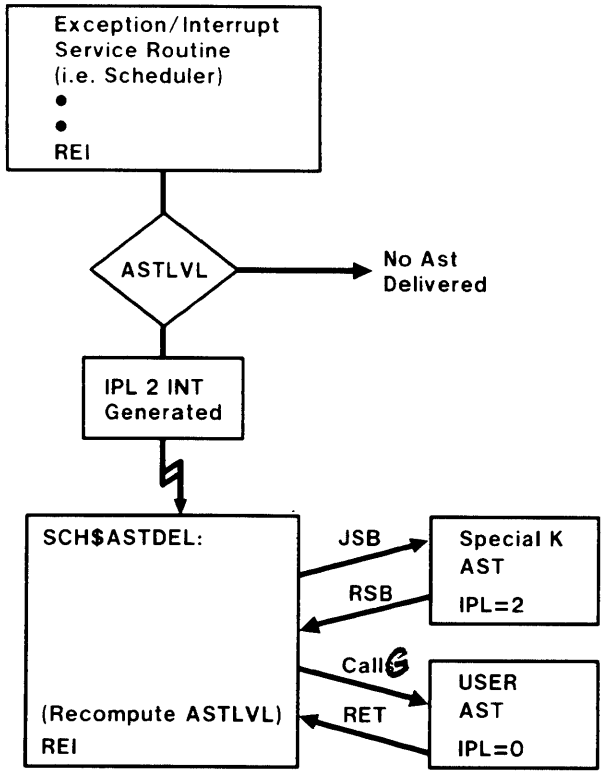


Figure 3-25 AST Delivery Sequence

Table 3-9 Rules for Selection of ASTs

Rule	Example
a) ASTLVL > new access mode	User AST (3) > kernel access mode (Ø)
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	IPL = 3 scheduler

TIMER QUEUE ELEMENT

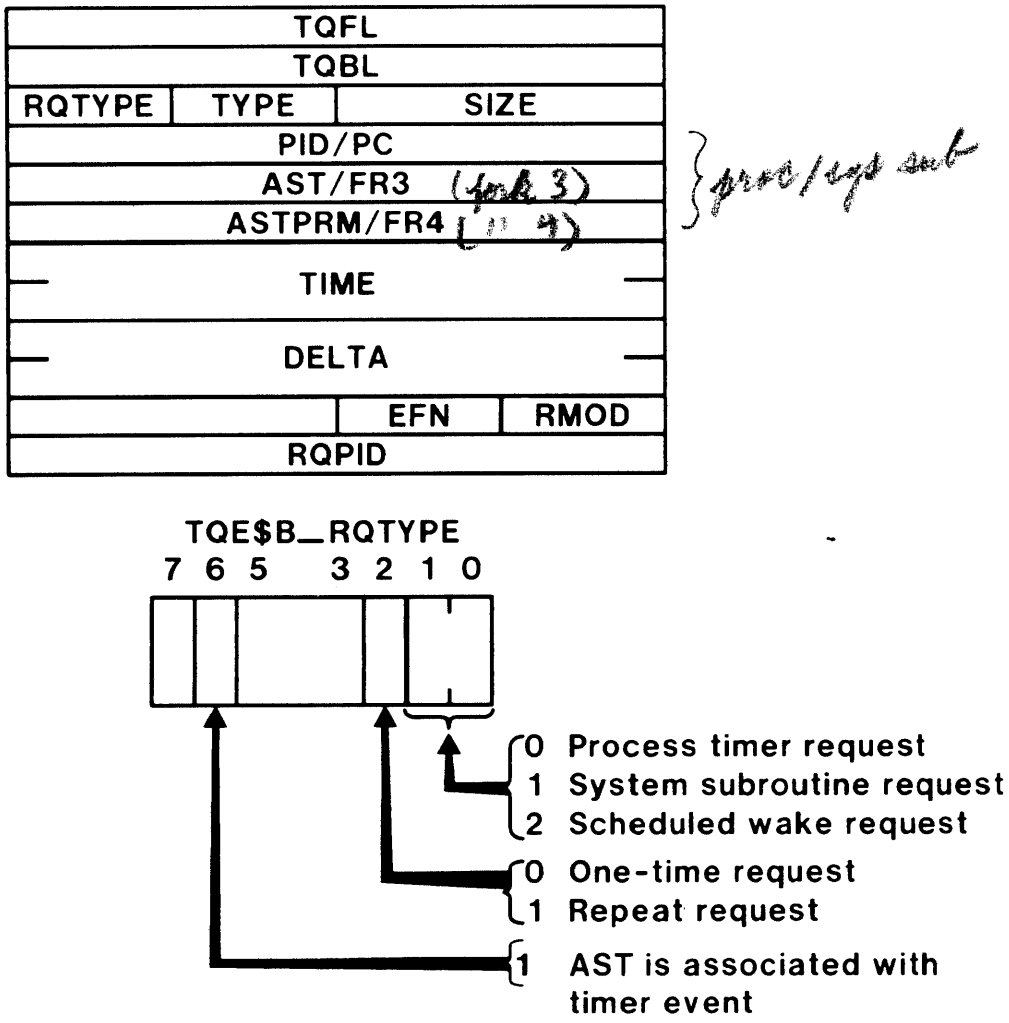


Figure 3-26 Timer Queue Element

- The timer queue element is inserted in the timer queue based upon the absolute expiration time of the request.
- 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.

CLOCKS AND TIMER SERVICES

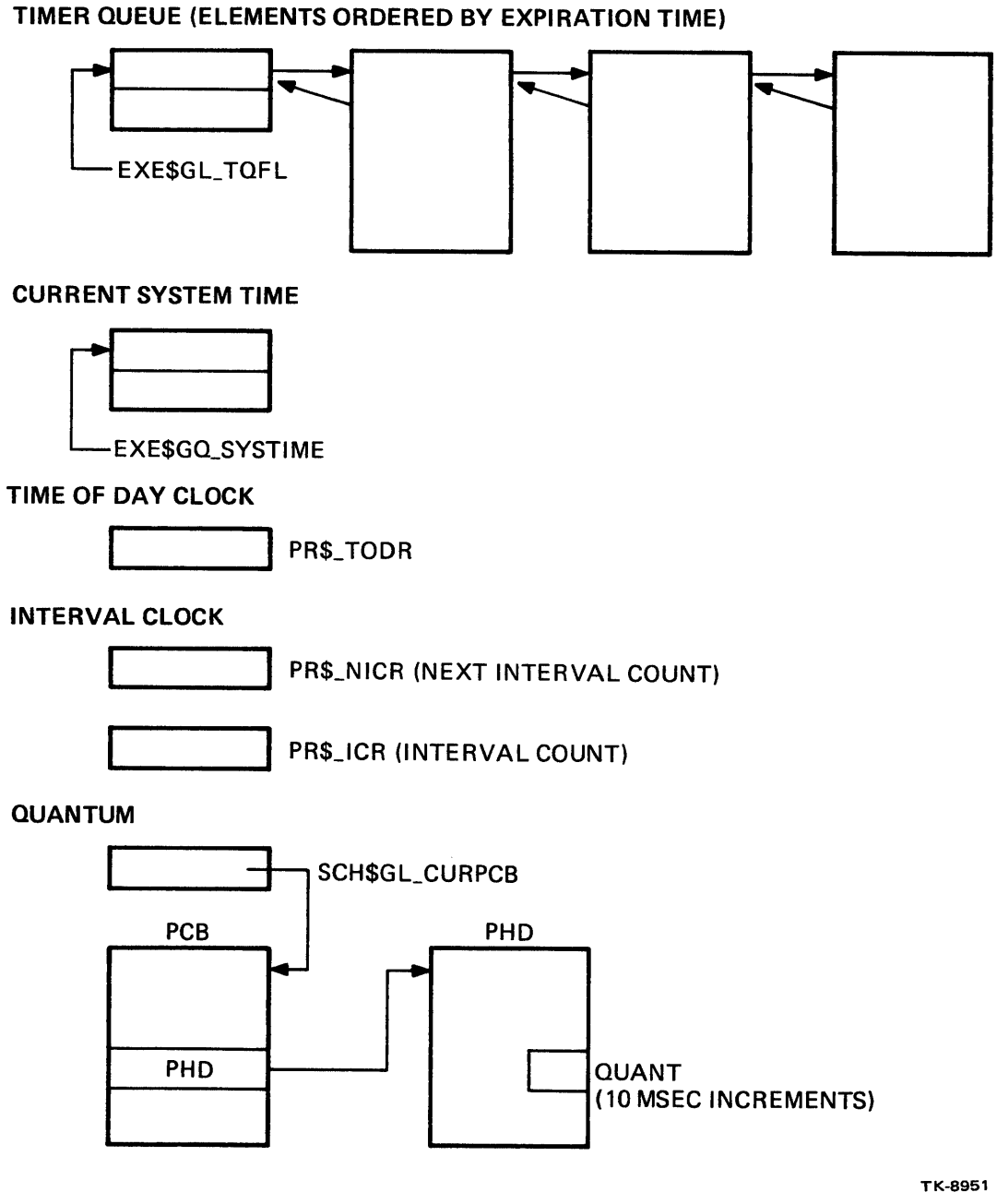


Figure 3-27 Clocks and Timer Services

SUMMARY OF SYSTEM MECHANISMS

Table 3-1Ø 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 7	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 3-10 Function and Implementation of System Mechanisms (Cont)

Function	Implementation	Name
Process Synchronization		
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 process to request action at a specific time	Hardware clock interrupt Queue of requests	Timer queue

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 3-1 IPL Control Macros

Argument Probing Macros

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

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

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

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

Example 3-2 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 3-3 Privilege Checking Macros

Privilege Mask Locations

Table 3-11 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 B

THE REI INSTRUCTION

The REI instruction will result 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

4

INTRODUCTION

Since VMS runs in Executive and Kernel modes and at elevated interrupt priority levels, any error is considered serious, and may 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

Upon completion of this module, given the lecture notes, you will be able to:

1. Use various system-supplied debugging tools and utilities (e.g., SDA, DELTA, XDELTA) to examine crash dumps and to observe a running system.
2. 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 PATCH Utility Reference Manual
3. VAX Hardware Handbook
4. VAX/VMS Guide to Writing a Device Driver

TOPICS

I. Crash Dumps and Bugchecks (System Errors)

   A. What they are

   B. How they are generated

   C. Fatal and nonfatal system errors

   D. Sample system errors and stack output

II. The System Map File (SYS.MAP)

   A. Contents and format

   B. How to use it

      1. For bugchecks
      2. For source code

III. The System Dump Analyzer (SDA)

   A. Commands

   B. Crash file analysis

   C. Active system analysis

IV. Other Debugging Tools

   A. DELTA

   B. XDELTA

   C. CCL (console command language)

   D. PATCH

CRASH DUMPS

A crash dump is generated when the system decides that it can not continue functioning. The system stops the normal flow of work and attempts to copy all the information in physical memory to a special file on a disk.

Causes of Crash Dumps

A crash dump is a result of a fatal error or inconsistency (fatal bugcheck) being recognized and declared by a component of the operating system. A bugcheck is declared by that component by referencing a central routine. Some reasons for declaring a fatal bugcheck are listed below.

- 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 - non-fatal; 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 non-fatal bugcheck, all bugchecks must be declared fatal (done by setting BUGCHECKFATAL = 1)
  - DUMPBUG = 1 (requests dump on fatal bugcheck)
  - SYS$SYSTEM:SYSDUMP.DMP must be the correct size file size (in blocks) = physical memory (in pages) plus 4 pages

How Bugchecks Are Generated

BUGCHECKS are generated by 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, etc.) after detecting an internal (software) error.

Table 4-1 Sample BUGCHECKS

Name	Module	Type	Description
BADRSEIPL	RSE	Fatal	Bad IPL at entrance to RSE
FATALEXCPI	EXCEPTION	Fatal	Fatal Exec or Kernel mode exception
NOTPCB	MUTEX	Fatal	Structure not PCB
UNABLCREVA	EXCEPTION	Cont.	Unable to create V.A. space

NOTE

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

```
>>>E P
        00000000
>>>E/G F
      G       0000000F        80007B06
>>>E/I 1
        I     00000001        00000000
>>>E/I 2
        I     00000002        00000000
>>>E/I 3
        I     00000003        00000000
>>>E/I 4
        I     00000004        8011DA00
>>>D/G F FFFFFFFF
>>>D P 001F0000
>>>C

**** FATAL BUG CHECK, VERSION = V3.0 INVEXCEPTN, Exception while above ASTDEL or on
    CURRENT PROCESS = NULL
    REGISTER DUMP
        R0 = 0000001F
        R1 = 001F0000
        R2 = 00000000
        R3 = 00000000
        R4 = 00000000
        R5 = 00000000
        R6 = 00000000
        R7 = 00000000
        R8 = 00000000
        R9 = 00000000
        R10= 00000000
        R11= 00000000
        AP = 00000009
        FP = 00000000
        SP = 80000998
        PC = 800030C6
        PSL= 001F0009
    KERNEL/INTERRUPT STACK
        800009A0  00000004   <- MECHANISM ARRAY
        800009A4  00000000
        800009A8  FFFFFFF0
        800009AC  00000000
        800009B0  00000000
        800009B4  00000001   <- SIGNAL ARRAY
        800009B8  00000005
        800009BC  0000000C   <- SS$_ACCVIO
        800009C0  00000001   <- REASON MASK
        800009C4  FFFFFFFF   <- FAULTING V.A.
        800009C8  FFFFFFFF   <- PC
        800009CC  001F0000   <- PSL
        800009D0  00000000
        800009D4  00000000
        800009D8  00000000
        800009DC  00000000
        800009E0  00000000
        800009E4  00000000
        800009E8  00000000
        800009EC  00000000
        800009F0  00000000
        800009F4  00000000
        800009F8  00000000
```

Example 4-1 Sample Console Output After Bugcheck

SAMPLE STACKS AFTER BUGCHECKS

Access Violation

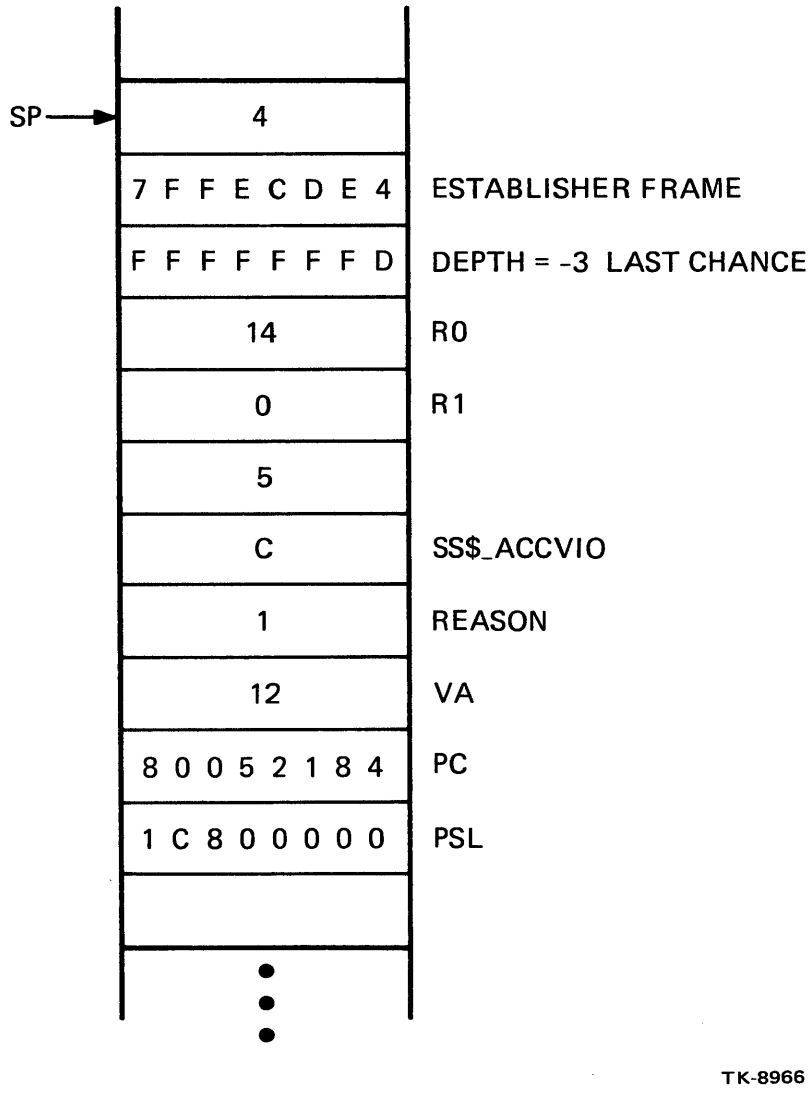


Figure 4-1 Stack After Access Violation Bugcheck

Probable Causes:

- Blown register
- Incorrect data structure field
- Improper synchronization

Page Fault Above IPL 2

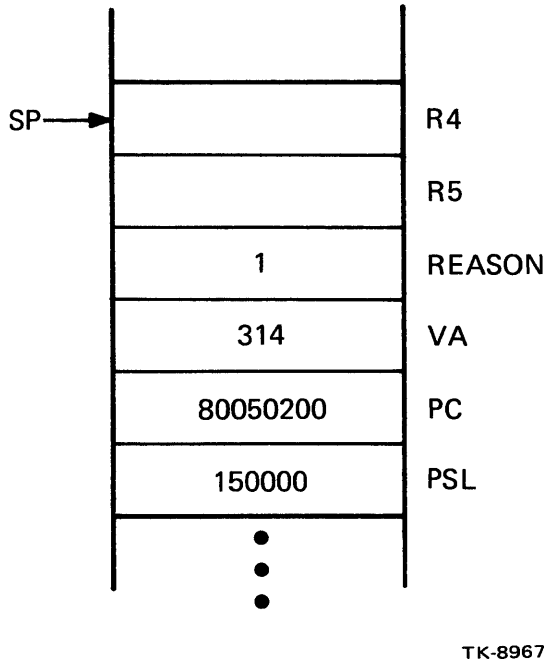


Figure 4-2 Stack After Page Fault Above IPL-2

Probable Causes:

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

Reserved Operand Fault

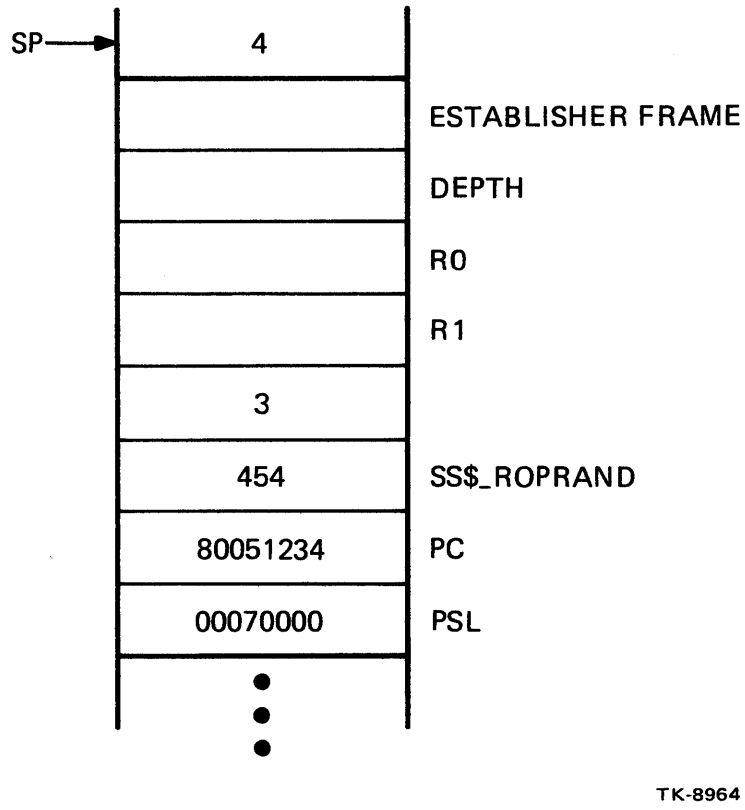


Figure 4-3 Stack After Reserved Operand Fault

Probable Causes:

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

Machine Check in Kernel Mode (CPU Timeout)

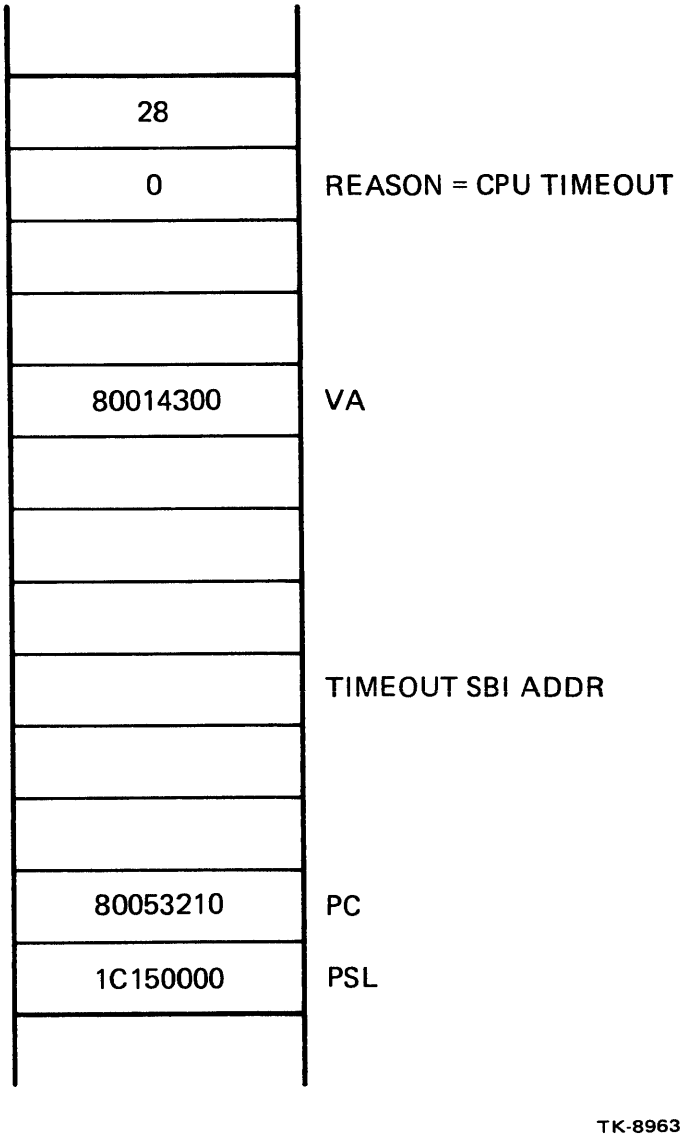


Figure 4-4 Stack After Machine Check in Kernel Mode

Reasons:

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

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 (i.e., State of process)

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

   - Program section synopsis (V.A. --> source code module)

   - Symbols by value (value --> symbol name)

SYS.MAP and Source Code

1. Layout of linked Executive in SØ space

   - Program section synopsis

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

   - Symbol cross reference

3. Module entry points and global data locations.

VAX/VMS DEBUGGING TOOLS

Under the VAX/VMS operating system, there are several ways to debug or analyze a problem that is occurring in a program. The method of analysis depends on the environment the program is running under and the nature of the analysis you wish to do (i.e., monitoring only, stepping through a program). The table below describes the type of analysis, program environment, and the suggested method of analysis (the debugger).

Table 4-2 Environment Vs. Debugging Tools

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

THE SYSTEM DUMP ANALYZER (SDA)

Uses

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.

Information Handling

Through the SDA, information can be:

- Displayed on a video terminal
- Printed on a hardcopy terminal
- Sent to a file/line printer.

Requirements

- SYS SYSTEM:SYSDUMP.DMP file must be large enough
  Size of file = physical memory plus 4 (in pages)
- Dumpbug (a SYSGEN parameter) must be set (=1) by default, it is set.
- Console must be allowed to finish printing the bugcheck output.
- To run SDA:
  - VIRTUALPGCNT > SYSDUMP.DMP + 1000 (pages)
  - PGFLQUOTA > SYSDUMP.DMP + 1000 (pages)
- To examine the active system, the CMKRNL privilege is needed.
- To examine a dump file, read access to the file is needed.

Activation of SDA

```
$ ANALYZE/qualifier
```

What you use as a qualifer will determine what you will be examining.

Table 4-3 Examining Crash Dump or Current System

To Examine	Qualifier	Comment
Current System	/SYSTEM	CMKRNL priv needed
System Dump File or Other Dump File	/CRASH_DUMP	Read access to file needed

SDA Functions

- Examine locations by address or symbol
- Displays process/system data
- Formats and displays data-known data structures
- Assigns values to symbols as requested.

Command Format

```
command     [parameter]     [/qualifier]
```

Functions

Table 4-4 SDA Functions and Commands

Information

Function	Command
Provides help using SDA	HELP
Displays specific data/information	SHOW
Format and display data structures	FORMAT
Display contents of location(s)	EXAMINE

Manipulation

Preserve second copy of dump file	COPY
Create and define symbols	DEFINE
Perform computations	EVALUATE
Set/reset defaults	SET
Define other VMS symbols	READ
Repeat last command	REPEAT or <ESC>

Table 4-5 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
System page table	SHOW PAGE_TABLE	/GLOBAL, /SYSTEM /ALL (D)
PFN data base	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), /PHD /WORKING_SET, /ALL /REGISTERS, /PAGE_TABLES /PROCESS_SECTION_TABLE
Stacks	SHOW STACK	/INTERRUPT, /KERNEL /EXECUTIVE, /SUPER /USER
Summary of all processes	SHOW SUMMARY	
Symbol table	SHOW SYMBOL	Symbol-name parameter optional, /ALL

Table 4-6 Symbols and Operators

Function	Symbol or Operator	Example
Contents of location	@	Examine @80000045A
Add 80000000 (S0 base) to address	G	G45A
Add 7FFE0000 (P1 stacks) to address	H	H7A4
Current location	.	Format
Hexadecimal number radix	^H	^H10
Octal number radix	^O	^020
Decimal number radix	^D	^D16
Register symbols	R0-R11, AP, FP, KSP, ESP, SSP, USP, P0BR, P0LR, P1BR, P1LR, PC, PSL	

Table 4-7 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 = G580	

Examining an Active System

```
$ ANALYZE/SYSTEM (1)

VAX/VMS System analyzer


SDA> EVALUATE G+(50*4)-(4/2)+^07 (2)
Hex = 80000145   Decimal = -2147483323

SDA> EXAMINE G25C0 (3)
SCH$GL_SWPPCB+058:  00000000    '....'

SDA> EXAMINE (4)
SCH$GL_SWPPCB+05C:  00000000    '....'

SDA> $<ESC>
SCH$GL_SWPPID:  00010001    '....'

SDA> $<ESC>
SCH$GL_SWPPID+004:  800023F0    '.#..'

SDA> EX IOC$GL_DEVLIST
IOC$GL_DEVLIST:  80000C6C    'l...'

SDA> EX R0
R0:  00000001    '....'

SDA> EX PSL
PSL:  03C00000    '....'

SDA> EX G100:G140
0004001B 8FBC00FC 0004001A 8FBC003C  <...............     80000100
0004001D 8FBC07FC 0004001C 8FBC00FC  ................      80000110
0004001F 8FBC0FFC 0004001E 8FBC00FC  ................      80000120
00040021 8FBC003C 00040020 8FBC01FC  ....  ...<...!...     80000130
00040023 8FBC01FC 00040022 8FBC0010  ....'.......#...     80000140
```

Example 4-2 Examining an Active System (Sheet 1 of 5)

```
SDA> SHOW PROCESS                                    (1)
Process status:  00040001   RES,PHDRES

PCB address                 800E78C0    JIB address                 800E7780
Master PID                  00050048    Creator PID                 00000000
PID                         00050048    Subprocess count                 0
PHD address                 80149C00    Swapfile disk address       01000421
State                          CUR      Termination mailbox             0000
Current priority                 9      AST's enabled                   KESU
Base priority                    4      AST's active                    NONE
UIC                        [011,110]    AST's remaining                  9
Mutex count                      0      Buffered I/O count/limit         6/6
Waiting EF cluster               0      Direct I/O count/limit           6/6
Starting wait time          00001B1B    BUFIO byte count/limit      8128/8192
Event flag wait mask        F7FFFFFF    # open files allowed left       36
Local EF cluster 0          CC000001    Timer entries allowed left      10
Local EF cluster 1          00000000    Active page table count          0
Global cluster 2 pointer    00000000    Process WS page count          188
Global cluster 3 pointer    00000000    Global WS page count            12


SDA> SHOW PFN_DATA/MODIFIED                          (2)

Count:                  29
Lolimit:               140
High limit:             44

PFN    PTE ADDRESS    BAK       REFCNT  FLINK BLINK     TYPE          STATE
----   -----------    --------  ------  ----- -----     ----------    ----------

0199     8014CC84   04000000        0    011F 0000    00 PROCESS     81 MDFYLST
011F     8014CAC8   04000000        0    01D0 0199    00 PROCESS     81 MDFYLST
01D0     8015F0D4   04000000        0    066F 011F    00 PROCESS     81 MDFYLST
066F     8015F534   04000000        0    0297 01D0    00 PROCESS     81 MDFYLST
0297     8014C8B8   04000000        0    01E9 066F    00 PROCESS     81 MDFYLST
01E9     8014C9FC   04000000        0    01E7 0297    00 PROCESS     81 MDFYLST
01E7     8014CA04   04000000        0    01E5 01E9    00 PROCESS     81 MDFYLST
01E5     8014CA0C   04000000        0    01E6 01E7    00 PROCESS     81 MDFYLST


.TITLE GLOBALS

$PHDDEF GLOBAL                  ; PROCESS HEADER
$DPTDEF GLOBAL                  ; DRIVER PROLOGUE TABLE
$IDBDEF GLOBAL                  ; INTERRUPT DATA BLOCK
$CRBDEF GLOBAL                  ; CHANNEL REQUEST BLOCK
$ADPDEF GLOBAL                  ; ADAPTER CONTROL BLOCK
$DDBDEF GLOBAL                  ; DEVICE DATA BLOCK
$UCBDEF GLOBAL                  ; UNIT CONTROL BLOCK
$VCBDEF GLOBAL                  ; VOLUME CONTROL BLOCK
$ACBDEF GLOBAL                  ; AST CONTROL BLOCK
$IRPDEF GLOBAL                  ; I/O REQUEST PACKET

.END
```

Example 4-2 Examining an Active System (Sheet 2 of 5)

```
SDA> READ   [HELLER]GLOBALS ❶

SDA> FORMAT @IOC$GL_DEVLIST ❷
80000C6C    DDB$L_LINK                  80000D9C
80000C70    DDB$L_UCB                   80000CA0
80000C74    DDB$W_SIZE                      0034
80000C76    DDB$B_TYPE                    06
80000C77                                00
80000C78    DDB$L_DDT                   800B4AFC
80000C7C    DDB$L_ACPD                  01313146
            DDB$B_ACPCLASS
80000C80    DDB$T_NAME                  'DRA'
80000C84                                00000000
80000C88                                00000000
80000C8C                                00000000
80000C90    DDB$T_DRVNAME               'DRDRIVER'
80000C99                                000000
80000C9C                                00000000
            DDB$C_LENGTH

SDA> FORMAT @. ❸
80000D9C    DDB$L_LINK                  80000F84
80000DA0    DDB$L_UCB                   80000DD0
80000DA4    DDB$W_SIZE                      0034
80000DA6    DDB$B_TYPE                    06
80000DA7                                00
80000DA8    DDB$L_DDT                   800B29B3
80000DAC    DDB$L_ACPD                  00000000
            DDB$B_ACPCLASS
80000DB0    DDB$T_NAME                  'OPA'
80000DB4                                00000000
80000DB8                                00000000
80000DBC                                00000000
80000DC0    DDB$T_DRVNAME               'OPERATOR'
80000DC9                                000000
80000DCC                                00000000
            DDB$C_LENGTH

SDA> $<ESC> ❹
80000F84    DDB$L_LINK                  80001144
80000F88    DDB$L_UCB                   8000103C
80000F8C    DDB$W_SIZE                      0034
80000F8E    DDB$B_TYPE                    06
80000F8F                                00
80000F90    DDB$L_DDT                   800013E8
80000F94    DDB$L_ACPD                  00000000
            DDB$B_ACPCLASS
80000F98    DDB$T_NAME                  'MBA'
80000F9C                                00000000
80000FA0                                00000000
80000FA4                                00000000
80000FA8    DDB$T_DRVNAME               'MBDRIVER'
80000FB1                                000000
80000FB4                                00000000
            DDB$C_LENGTH
SDA>
```

Example 4-2 Examining an Active System (Sheet 3 of 5)

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

BUFIO    800DDA00   160
                    00000000 0013009E 800E9440 800DDAA0 ....@.........
                    80000CA0 800D203E 4144530A 8002DB28 (....SDA> ......
                    801015C0 101BFF70 7FFB9A14 1B1B0034 4.......p.......
                    00000000 00000001 00000000 0005005D ]...............
                    00000041 00000002 00000020 00000600 .... ......A...
                    00000000 800E793C 00000000 00001041 A.......<y......
                    00000000 00000000 00000000 00000000 ................
                    00000000 00000000 00000000 00000000 ................
                    00000000 00000000 00000000 00000000 ................
                    00000000 00000000 00000000 00000000 ................
IRP      800DE680   160
                    00010047 430A009C 80002790 800E6D80 .m...'.....CG...
                    8000103C 00000000 00000280 00002FEC ./..........<...
                    00000000 0403FFE0 00000288 17000021 !...............
                    00000000 00000290 00000000 03200000 .. ............
                    8014CCF0 00000001 00000200 00000000 ................
                    30203635 800E1E9C 20203020 000010CA .... 0  ....56 0
                    80000000 00000000 00000000 FFFFFFFF ................
                    54534C59 46444D20 31382020 20202053 S     81 MDFYLST
                    800DE70C 00090048 800DE6FC 0000000D ........H........
                    00000000 00000000 00000000 800DE70C ................
UCB      800DF760   160
                    800DF76C 081000A0 00000000 00000000 ............l...
                    00000000 00000000 00000000 800DF76C l...............
                    00000000 00010047 80106540 801065A0 .e..@e..G.......
                    40000082 00840143 08840043 800E64C0 .d..C...C......@
                    00000000 00000000 800DF7A0 800DF7A0 ................
                    00000000 00000010 80000CA0 03140001 ................
                    00000000 00000000 00000000 00000001 ................
                    00000000 8008C958 00000000 00000000 ........X........
                    00000000 0000000F 0000FFFF 00000000 ................
                    00000000 00000000 00000000 00000000 ................
PCB      800DF800   160
                    00000F00 180C00A0 800E1E20 800E7D20  }.. ............
                    00000000 000C4674 800DF810 800DF810 ........tF........
                    18000007 00001817 00040001 01000181 ................
                    000C000C 000C003F 003B0000 00000000 ......;.?.........
                    F7FFFFFF 00000000 00000000 0000000C ................
                    00000000 00000000 00000000 4C000000 ...L............
                    00544D46 52524506 8011E000 0002004A J........ERRFMT.
                    F3FFFDFF 800DF8A0 00000000 00000000 ................
                    800DF88C 00010006 800DF87C FFFFFFFF ....|............
                    00000000 00000000 00000000 800DF88C ................
JIB      800DF8A0   160
                    00000000 002F0070 00000000 00000000 ........p./......
                    00000000 00000000 0000A000 00009FC0 ................
                    00004FB9 00005000 0040003F 0040003F ?.@.?.@..P...O..
                    000A000A 00000000 00400000 00000000 ......@.........
                    54535953 00000000 00000000 0002004A J...........SYST
                    00000000 00000000 20202020 20204D45 EM        ........
                    00000000 00000000 00000000 00000000 ................
                    00000000 00000000 00000000 00000000 ................
                    00000000 00000000 00000000 00000000 ................
                    00000000 00000000 00000000 00000000 ................
CRB      800DFE40   160
                    00000008 42050070 800DFE40 800DFE40 @...@...p...B....
                    001F0000 00000000 00000000 00000000 ................
```

Example 4-2 Examining an Active System (Sheet 4 of 5)

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

                 7FFB624C  000125E8
                 7FFB6250  000011C0      SGN$C_MAXPGFL+1C0
                 7FFB6254  00012C04
                 7FFB6258  00000000
                 7FFB625C  00000010
                 7FFB6260  7FFB62E0
                 7FFB6264  7FFB62E0
                 7FFB6268  7FFB626C

           SP => 7FFB626C  00000001
                 7FFB6270  7FFB6270
                 7FFB6274  00000000
                 7FFB6278  200C0000
                 7FFB627C  000011C0      SGN$C_MAXPGFL+1C0
                 7FFB6280  7FFB62F0
                 7FFB6284  0001161E
                 7FFB6288  00000000
                 7FFB628C  000125E8
                 7FFB6290  00000003
                 7FFB6294  7FFB6800
                 7FFB6298  7FFB6800
                 7FFB629C  7FFB626C
                 7FFB62A0  00000000
                 7FFB62A4  8011DA00
                 7FFB62A8  00000400      BUG$_UNEXUBAINT
                 7FFB62AC  000114AA
                 7FFB62B0  7FFEAE00      CTL$GL_KSPINI
                 7FFB62B4  7FFEAE00      CTL$GL_KSPINI
                 7FFB62B8  00000600      SWP$AL_PTRPAG
                 7FFB62BC  000114B4
                 7FFB62C0  7FFEBE00
                 7FFB62C4  7FFEBE00
                 7FFB62C8  00001000      SGN$C_MAXPGFL
                 7FFB62CC  000114BB
                 7FFB62D0  7FFED54C
                 7FFB62D4  7FFEDE00      SYS$QIOW
                 7FFB62D8  00002000      SGN$C_MAXVPGCNT
                 7FFB62DC  000114C5
                 7FFB62E0  7FFB626C
                 7FFB62E4  7FFB6800
                 7FFB62E8  7FFB6800
                 7FFB62EC  000114D0
                 7FFB62F0  00000000
                 7FFB62F4  07FC0000
                 7FFB62F8  7FFB636C
                 7FFB62FC  7FFB6330
                 7FFB6300  00036C8E
                 7FFB6304  00000000
                 7FFB6308  000125E8
                 7FFB630C  000011C0      SGN$C_MAXPGFL+1C0
                 7FFB6310  00012C04
                 7FFB6314  00000000
                 7FFB6318  000011CC      SGN$C_MAXPGFL+1CC
                 7FFB631C  000125E4
                 7FFB6320  000011D4      SGN$C_MAXPGFL+1D4
                 7FFB6324  000011F7      SGN$C_MAXPGFL+1F7
                 7FFB6328  00000000
                 7FFB632C  00000000
                 7FFB6330  00000000
                 7FFB6334  2FFC0000
                 7FFB6338  7FFB63AC
                 7FFB633C  7FFB6390
                 7FFB6340  00002DDA      SS$_EXENQLM+396
                 7FFB6344  00000FF8      SS$_EXDEPTH+1DF
```

Example 4-2 Examining an Active System (Sheet 5 of 5)

Examining a Crash Dump File

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

Dump taken on  7-JUN-1982 17:15:54.12
INVEXCEPTN, Exception while above ASTDEL or on interrupt stack

SDA> SHOW CRASH
Time of system crash:  7-JUN-1982 17:15:54.12

Version of system: VAX/VMS VERSION V3.0

Reason for BUGCHECK exception: INVEXCEPTN, Exception while above ASTDEL or on interr

Process currently executing: NULL

Current IPL: 31  (decimal)



General registers:

        R0  = 0000001F   R1  = 001F0000   R2  = 00000000   R3  = 00000000
        R4  = 00000000   R5  = 00000000   R6  = 00000000   R7  = 00000000
        R8  = 00000000   R9  = 00000000   R10 = 00000000   R11 = 00000000
        AP  = 00000000   FP  = 00000000   SP  = 800009A0   PC  = 80003DC6
        PSL = 001F0009



Processor registers:

        P0BR   = 80000000      P0LR   = 00000000      P1BR   = 7F802000
        P1LR   = 00200000      SBR    = 00171A00      SLR    = 00003980

        ISP    = 8011DA00      PCBB   = 0012184C      SCBB   = 0016FE00
        KSP    = 800009A0      ASTLVL = 00000004      SISR   = 00000000
        ESP    = 00000000      ICCS   = 800000C1      ICR    = FFFFF9DE
        SSP    = 00000000      TODR   = 61392F29
        USP    = 00000000

        ACCS   = 00000000      TBDR   = 00000000      CADR   = 00000000
        MCESR  = 00000000      CAER   = 00000000      CMIERR = 00080010
SDA>
```

Example 4-3 Examining a Crash Dump File

DELTA AND XDELTA

Table 4-8 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:

- Non-symbolic
- 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_name+SYS$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 (e.g., to look at system data structures). Extreme caution should be exercised so that data structures are not 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.)

```
GO:     .WORD   0                       ; NULL ENTRY MASK
        $CMKRNL_S  ROUTIN = 10$ ; ENTER KERNEL MODE
        RET                             ; ALL DONE
10$:    .WORD   0                       ; NULL ENTRY MASK
        NOP                             ; WHERE BPT INSTRUCTION
        NOP                             ; IS PLACED, BY 215;B
                                        ; ISSUED BY USER
        MOVZBL  #SS$_NORMAL,R0  ; RETURN SUCCESS STATUS
        RET                             ; ALL DONE
        .END GO
```

Example 4-4 The CHMK Program, Run with DELTA

DELTA and XDELTA Functions and Commands

Table 4-9 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 80000005/50E9002C
Display range of locations	addr,addr'/contents	G4,GC/8FBC0FFC 80000008/50E9002C 8000000C/00000400
Display indirect	<TAB> or /	80000A88/80000BE4 <TAB> 80000BE4/80000078 80000A88/80000BE4/800000
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 4-9 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

CONSOLE COMMANDS

Table 4-1Ø Console Commands

Function	Command*
Look at Physical memory General and processor registers	Examine
Place information into Physical memory General and processor registers	Deposit
Put processor into known state	Initialize
Stop the CPU (necessary for 78Ø only)	Halt
Restart halted CPU	Continue
Execute one instruction at a time	Single step

* Exact form of command is CPU-specific. Consult the VAX Hardware Handbook.

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 4-11 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

APPENDIX

BUGCHECK FLOW OF CONTROL

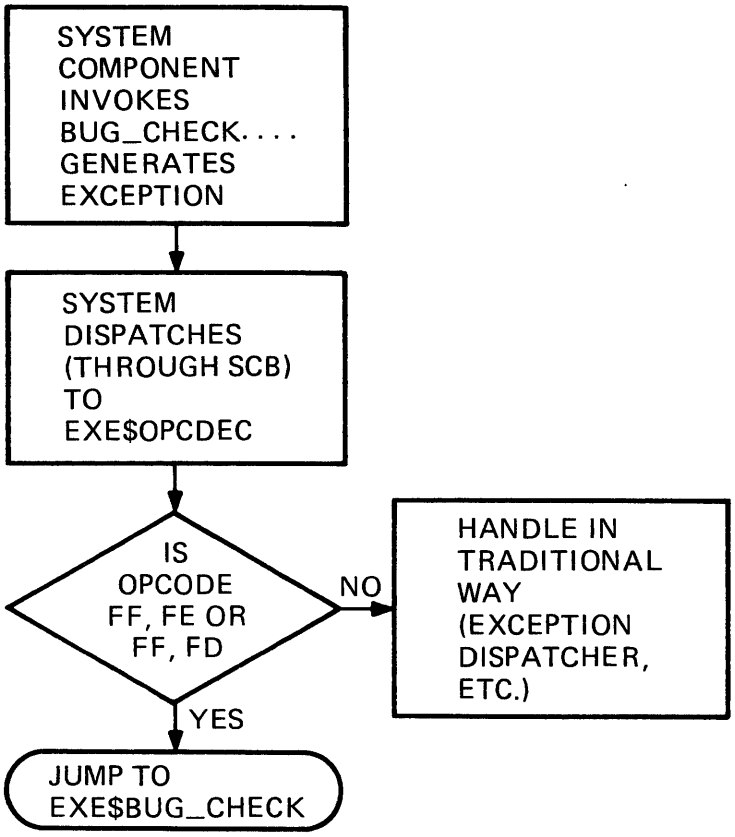


Figure 4-5 Bugcheck Flow of Control (Sheet 1 of 3)

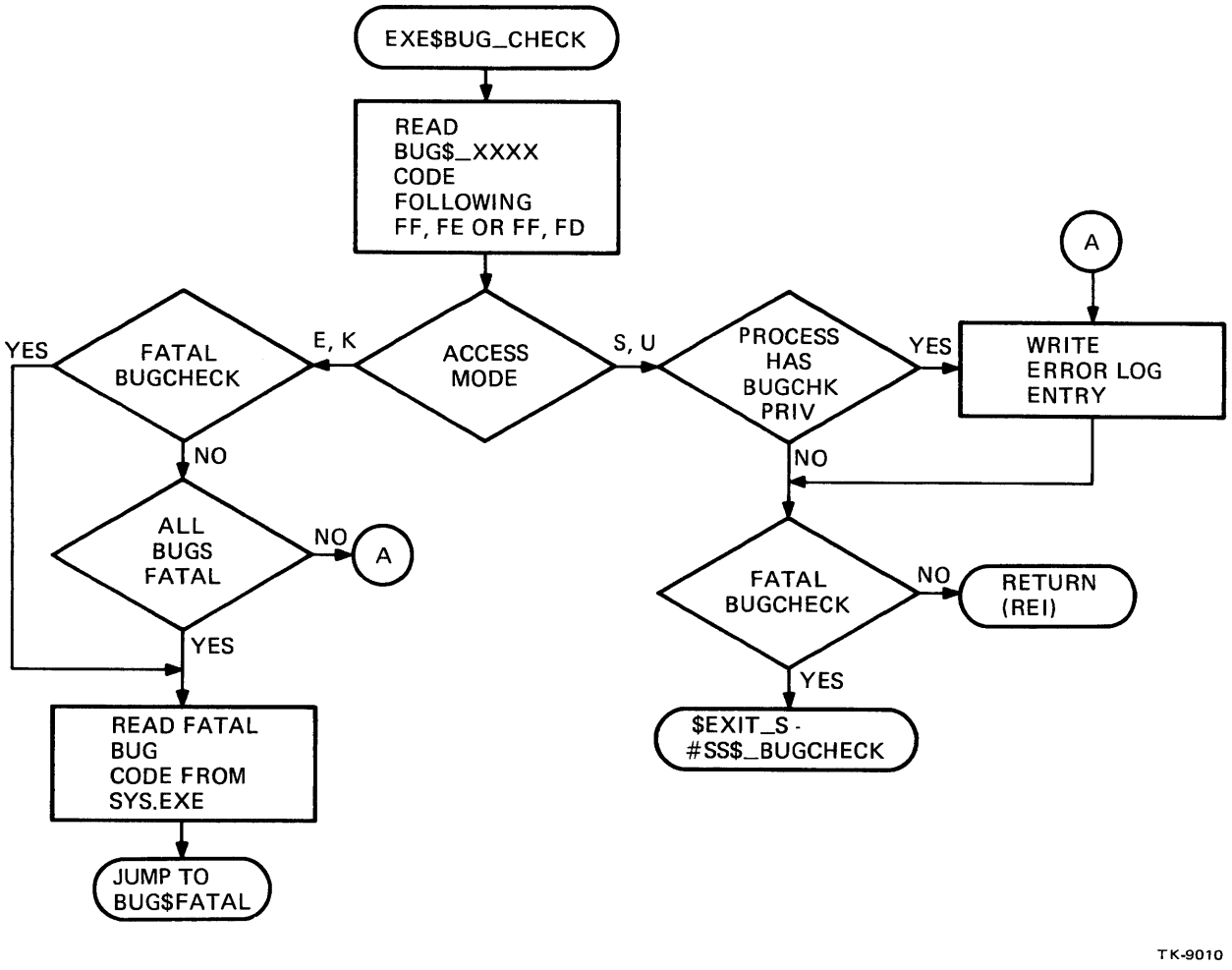


Figure 4-5 Bugcheck Flow of Control (Sheet 2 of 3)

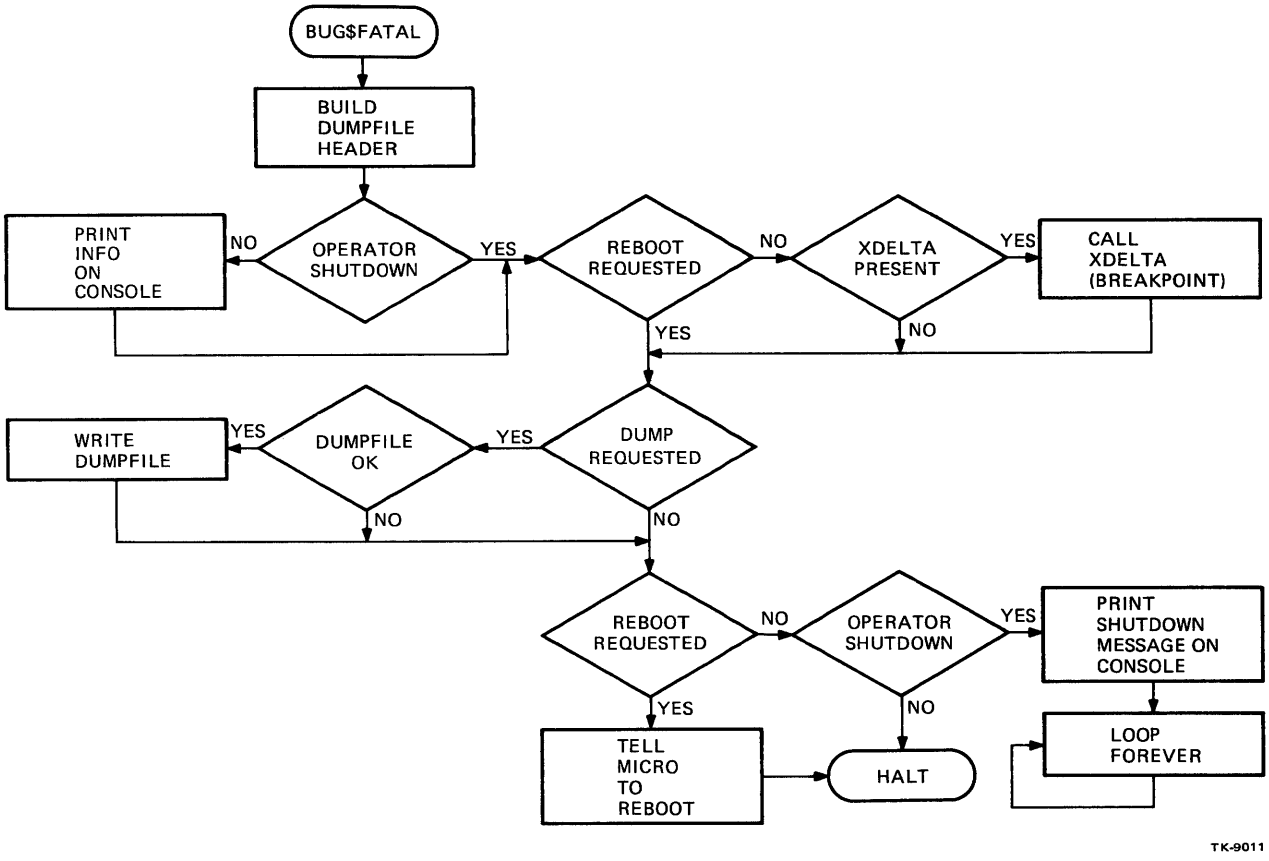


Figure 4-5 Bugcheck Flow of Control (Sheet 3 of 3)

SAMPLE CRASH DUMP ANALYSIS

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

Dump taken on 10-JUN-1982 12:54:02.76
SSRVEXCEPT, Unexpected system service exception

SDA> SHOW STACK
Current operating stack (KERNEL):

                7FFEAD58  7FFED778
                7FFEAD5C  7FFEDDD4
                7FFEAD60  7FFE7C90      CTL$AG_CLIDATA+080
                7FFEAD64  7FFEAD90      CTL$GL_KSTKBAS+590
                7FFEAD68  7FFEAD78      CTL$GL_KSTKBAS+578
                7FFEAD6C  7FFEAD70      CTL$GL_KSTKBAS+570
                7FFEAD70  8000CFF0      EXE$EXCPTN+006
                7FFEAD74  00000000

          SP => 7FFEAD78  00000000
                7FFEAD7C  00000000
                7FFEAD80  00000000
                7FFEAD84  7FFEADD0      CTL$GL_KSTKBAS+5D0
                7FFEAD88  80000014      SYS$CALL_HANDL+004
                7FFEAD8C  800130D1      EXE$SRCHANDLER+0C5
                7FFEAD90  00000002 (1)
                7FFEAD94  7FFEADB4      CTL$GL_KSTKBAS+5B4
                7FFEAD98  7FFEAD9C (2)  CTL$GL_KSTKBAS+59C
                7FFEAD9C  00000004
                7FFEADA0  7FFB6360
                7FFEADA4  FFFFFFFD      VA$M_VPG+1FD
                7FFEADA8  00C00009      PSL$M_PRVMOD+009
                7FFEADAC  00000002
                7FFEADB0  000008F8      SS$_ENDOFFILE+088
                7FFEADB4  00000005 (3)
                7FFEADB8  0000000C
                7FFEADBC  00000000
                7FFEADC0  0000000C
                7FFEADC4  80008BE8 (4)  MPH$QAST
                7FFEADC8  00C00004      PSL$M_PRVMOD+004
                7FFEADCC  0000021D      FIL$C_SIZE+005
                7FFEADD0  00000000
                7FFEADD4  00000000
                7FFEADD8  7FFB6378
                7FFEADDC  7FFEADE4      CTL$GL_KSTKBAS+5E4
                7FFEADE0  80008089      EXE$CMKRNL+00D
                7FFEADE4  00000000
                7FFEADE8  00000000
                7FFEADEC  7FFB6378
                7FFEADF0  7FFB6360
                7FFEADF4  8000CFE6      EXE$CMODEXEC+18E
                7FFEADF8  7FFEDE96      SYS$CMKRNL+006
                7FFEADFC  03C00000
```

Example 4-5 Sample Crash Dump Analysis (Sheet 1 of 4)

```
_DRA7:CSYS.OBJ]SYS.EXE;1                    27-APR-1982 03:46        VAX-11 Linker V03-16                Page   10

Psect Name      Module Name      Base      End         Length            Align                Attributes
----------      -----------      ----      ---         ------            -----                ----------
. BLANK .                     80007A6E 80008A2B 00000FBE (      4030.) BYTE 0 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,  WRT,NOVEC
                NLDRIVER      80008A0B 80008A0B 00000000 (         0.) BYTE 0
                MTFDT         80008A0B 80008A2B 00000021 (        33.) BYTE 0
                CONINTDSP     80008A2C 80008A2C 00000000 (         0.) BYTE 0
                CCMDRVSUB     80008A2C 80008A2C 00000000 (         0.) BYTE 0
                MAHANDLER     80008A2C 80008A2C 00000000 (         0.) BYTE 0
                XDSTRING      80008A2C 80008A2C 00000000 (         0.) BYTE 0
                VERSION       80008A2C 80008A2C 00000000 (         0.) BYTE 0
                DEVICEDAT     80008A2C 80008A2C 00000000 (         0.) BYTE 0
                MDAT_END      80008A2C 80008A2C 00000000 (         0.) BYTE 0
                SYS$IODEF     80008A2C 80008A2C 00000000 (         0.) BYTE 0
                LIB$MSGDEF    80008A2C 80008A2C 00000000 (         0.) BYTE 0
                SYS$PRDEF     80008A2C 80008A2C 00000000 (         0.) BYTE 0
                RMS$GLOBALS   80008A2C 80008A2C 00000000 (         0.) BYTE 0
                SYS$SSDEF     80008A2C 80008A2C 00000000 (         0.) BYTE 0
                SYS$P1_VECTOR 80008A2C 80008A2C 00000000 (         0.) BYTE 0

A$EXENONPAGED                 80009A2C 80008F9C 00000571 (      1393.) LONG 2 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,  WRT,NOVEC
                ASTDEL     (1)80008A2C 80008CC0 00000295 (       661.) LONG 2
                FORKCNTRL     80008CC4 80008D3F 0000007C (       124.) LONG 2
                TIMESCHDL     80008D40 80008F9C 0000025D (       605.) LONG 2

AES1                          80008F9D 8000925A 000002BE (       702.) BYTE 0 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,  WRT,NOVEC
                RSE           80009F9D 8000925A 000002BE (       702.) BYTE 0

AES2                          80009259 80009286 0000002C (        44.) BYTE 0 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,  WRT,NOVEC
                RSE           80009259 80009286 0000002C (        44.) BYTE 0

AEXENONPAGED                  80009288 8000A32A 000010A3 (      4259.) LONG 2 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,  WRT,NOVEC
                SHELL         80009288 80009289 00000004 (         4.) BYTE 0
                EXSUBROUT     8000928C 8000936D 000000E2 (       226.) BYTE 0
                MEMORYALC     8000936E 800097C0 00000453 (      1107.) BYTE 0
                MUTEX         800097C1 800098DC 0000011C (       284.) BYTE 0
                POSTEF        800098DD 80009A3F 00000163 (       355.) BYTE 0
                PROCSTRT      80009A40 80009A50 00000011 (        17.) BYTE 0
                SCHED         80009A54 80009ACA 00000077 (       119.) LONG 2
                SYSASCEFC     80009ACB 80009B17 0000004D (        77.) BYTE 0
                SYSBRDCST     80009B18 80009EF8 000003E4 (       996.) BYTE 0
                SYSDELPRC     80009EFC 80009F49 0000004E (        78.) BYTE 0
                SYSEVTSRV     80009F4A 80009FE8 0000009F (       159.) BYTE 0
                SYSGETJPI     80009FE9 80009FEF 00000007 (         7.) BYTE 0
                SYSPCNTRL     80009FF0 8000A18C 000001CD (       461.) BYTE 0
                SYSSETPFM     8000A1C0 8000A246 00000087 (       135.) LONG 2
                SYSWAIT       8000A247 8000A32A 000000E4 (       228.) BYTE 0

LOCKMGR                       8000A32C 8000AC96 0000096B (      2411.) LONG 2 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,  WRT,NOVEC
                DEADLOCK      8000A32C 8000A53F 00000214 (       532.) BYTE 0
                SYSENQDEQ     8000A540 8000AC96 00000757 (      1879.) LONG 2

WIONONPAGED                   8000AC98 8000C2A9 00001612 (      5650.) LONG 2 NOPIC,USR,CON,REL,LCL,NOSHR,  EXE,  RD,  WRT,NOVEC
                BUFFERCTL     8000AC98 8000AD18 00000081 (       129.) BYTE 0
                ERRORLOG      8000AD19 8000AFF4 000002DC (       732.) BYTE 0
```

Example 4-5 Sample Crash Dump Analysis (Sheet 2 of 4)

```
01AF    455          .SBTTL  SCH$QAST - ENQUEUE AST CONTROL BLOCK FOR P
01AF    456 ;++
01AF    457 ; FUNCTIONAL DESCRIPTION:
01AF    458 ;        SCH$QAST INSERTS THE AST CONTROL BLOCK SUPPLIED IN
01AF    459 ;        POSITION BY ACCESS MODE IN THE AST QUEUE OF THE PR
01AF    460 ;        BY THE PID FIELD OF THE AST CONTROL BLOCK.  AN AST
01AF    461 ;        IS THEN REPORTED FOR THE PROCESS TO REACTIVATE FRO
01AF    462 ;        IF APPROPRIATE.  THE AST CONTROL BLOCK WILL BE REL
01AF    463 ;        IF THE PID SPECIFIES A NON-EXISTENT PROCESS.
01AF    464 ;
01AF    465 ;        LOADABLE MULTI-PROCESSING CODE WILL REPLACE THIS R
01AF    466 ;        ENTIRELY NEW CODE, AT MPH$QAST.
01AF    467 ;
01AF    468 ; CALLING SEQUENCE:
01AF    469 ;        BSB/JSB SCH$QAST
01AF    470 ;
01AF    471 ; INPUT PARAMETERS:
01AF    472 ;        R2 - PRIORITY INCREMENT CLASS
01AF    473 ;        R5 - POINTER TO AST CONTROL BLOCK
01AF    474 ;
01AF    475 ; IMPLICIT INPUTS:
01AF    476 ;        PCB OF PROCESS IDENTIFIED BY PID FIELD
01AF    477 ;
01AF    478 ; OUTPUT PARAMETERS:
01AF    479 ;        R0 - COMPLETION STATUS CODE
01AF    480 ;        R4 - PCB ADDRESS OF PROCESS FOR WHICH AST WAS QUEL
01AF    481 ;
01AF    482 ; SIDE EFFECTS:
01AF    483 ;        THE PROCESS IDENTIFIED BY THE PID IN THE AST CONTR
01AF    484 ;        WILL BE MADE EXECUTABLE IF NOT SUSPENDED.
01AF    485 ;
01AF    486 ; COMPLETION CODES:
01AF    487 ;        SS$_NORMAL  -  NORMAL SUCCESSFUL COMPLETION STATUS
01AF    488 ;        SS$_NONEXPR -  NON-EXISTENT PROCESS
01AF    489 ;--
01AF    490          .ENABL  LSB
01AF    491 QNONEXPR:                                   ;
01AF    492          MOVL    R5,R0                      ; RELEASE AST CONT
01B2    493          BSBW    EXE$DEANONPAGED            ; IF NO SUCH PROCE
01B5    494          MOVZWL  #SS$_NONEXPR,R0            ; SET ERROR STATUS
01BA    495          BRB     QEXIT                      ; AND EXIT
01BC    496
01BC    497 MPH$QAST::                                  ; MULTI-PROCESSIN(
01BC    498 SCH$QAST::                                  ; ENQUEUE AST FOR
01BC (1) 499       (2)MOVZWL  ACB$L_PID(R5),R0          ; GET PROCESS INDE
01C0    500          DSBINT  #IPL$_SYNCH                ; DISABLE SYSTEM E
01C6    501          MOVL    @W^SCH$GL_PCBVEC[R0],R4    ; LOOK UP PCB ADDF
01CC    502          CMPL    ACB$L_PID(R5),PCB$L_PID(R4) ; CHECK FOR M/
01D1    503          BNEQ    QNONEXPR                   ; PID MISMATCHES
01D3    504          CLRL    R0                         ; ASSUME KERNEL M(
```

Example 4-5 Sample Crash Dump Analysis (Sheet 3 of 4)

```
SDA> SHOW CRASH
Time of system crash: 10-JUN-1982 12:54:02.76

Version of system: VAX/VMS VERSION V3.0

Reason for BUGCHECK exception: SSRVEXCEPT, Unexpected system service exception

Process currently executing: _OPA0:

Current image file name: DRA0:[HELLER]CRASHAST.EXE;1

Current IPL: 0  (decimal)


General registers:

        R0  = 00000000   R1  = 8000CFEA   R2  = 00000004   R3  = 7FFC4B3B
        R4  = 800E53E0   R5  = 00000000 (1) R6  = 31000408   R7  = 7FFED988
        R8  = 7FFED570   R9  = 7FFED778   R10 = 7FFEDDD4   R11 = 7FFE7C90
        AP  = 7FFEAD90   FP  = 7FFEAD78   SP  = 7FFEAD78   PC  = 8000CFF0
        PSL = 00000000


Processor registers:

        P0BR   = 801A3800      P0BB   = 0004A874      ACCS   = 00000000
        P0LR   = 00000003      SCBB   = 0016FE00      TBDR   = 00000000
        P1BR   = 7F9B7200      ASTLVL = 00000004      CADR   = 00000000
        P1LR   = 001FFD9E      SISR   = 00000000      MCESR  = 00000000
        SBR    = 00171A00      ICCS   = 800000C1      CAER   = 00000000
        SLR    = 00003980      ICR    = FFFFFDAA      CMIERR = 00080010
                               TODR   = 62AC9DEC

        ISP    = 8011DA00
        KSP    = 7FFEAD78
        ESP    = 7FFEBE00
        SSP    = 7FFED56C
        USP    = 7FFB6360
SDA>
```

Example 4-5 Sample Crash Dump Analysis (Sheet 4 of 4)

SCHEDULING

5

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 control the CPU. The scheduler code performs the exchange of hardware process contexts between the set of resident, computable processes and the currently executing process.

Two portions of the swapper, a system process, select candidates for removal from, or placement in, the set of memory-resident processes (the balance set). 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 Manual, chapters on scheduling and swap scheduling.

Additional Suggested Reading

- VAX/VMS Unternals and Data Structures Manual, chapters on software interrupts, process control and communication, timer support, 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. Operating System Code that Implements Process State Changes

   A. Context switch (SCHED.MAR)

   B. Result of system event (RSE.MAR)

IV. Steps at Quantum End

   A. Automatic working set adjustment

V. Boosting software Priority of Normal Processes.

PROCESS STATES

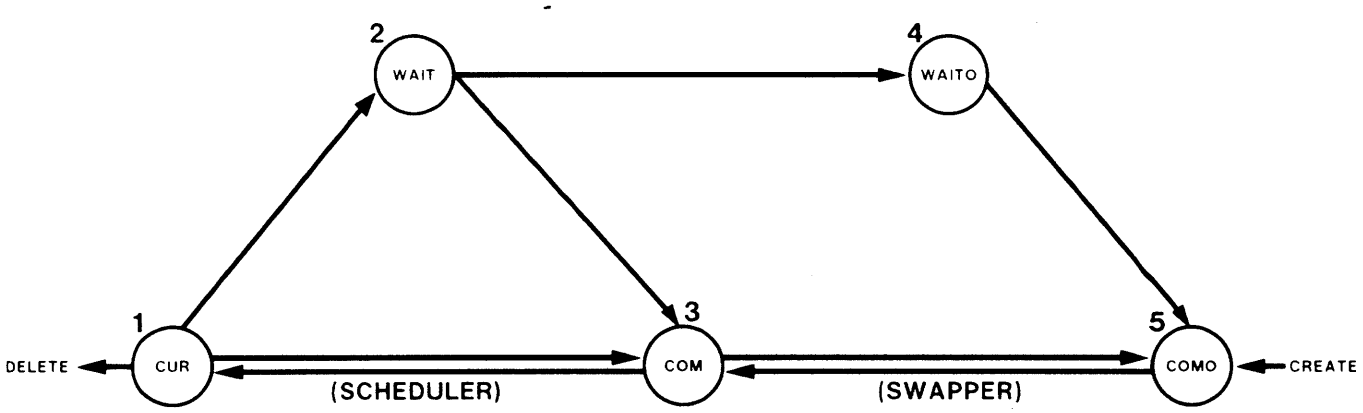


Figure 5-1 Process State Diagram

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 STATE DIAGRAM

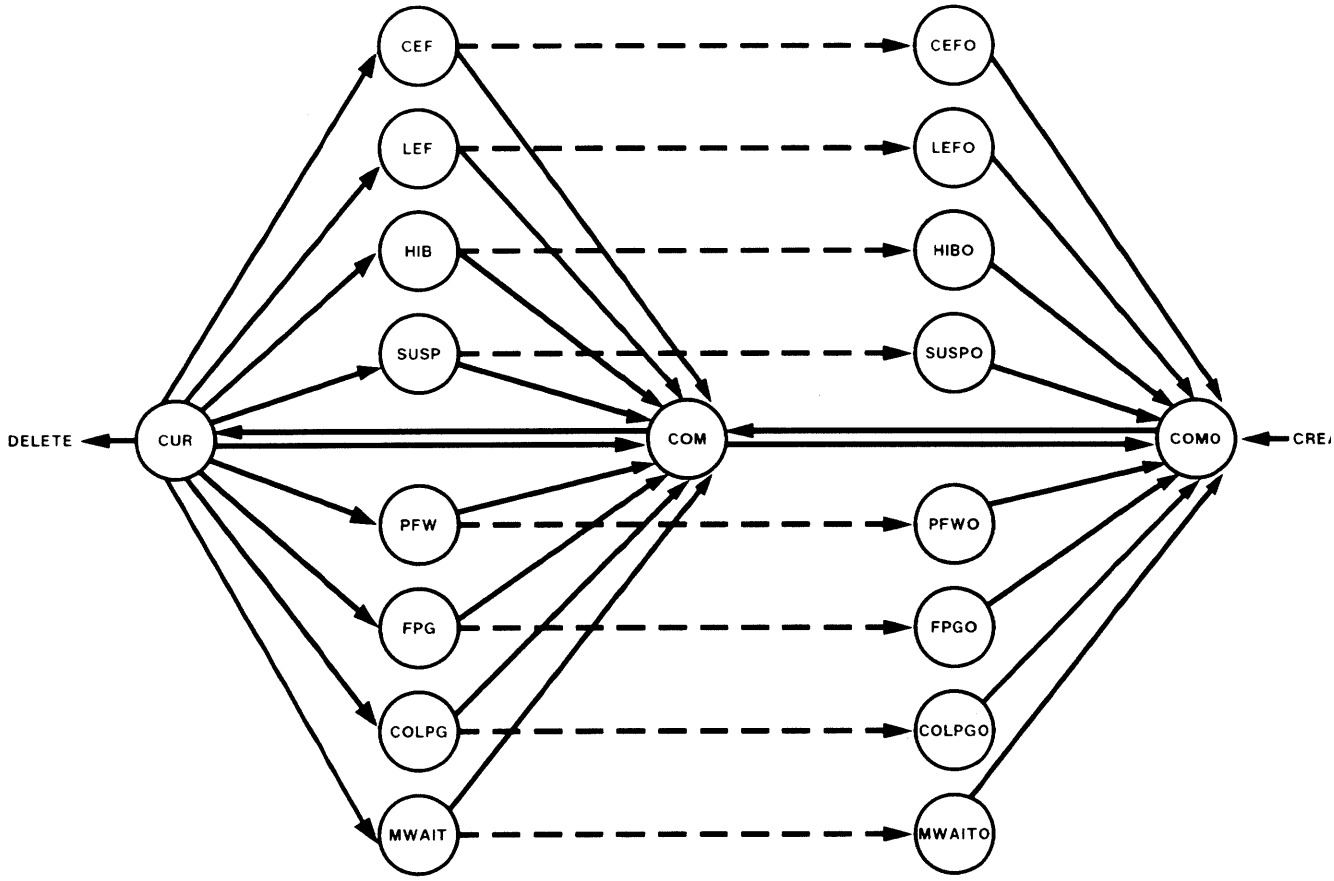


Figure 5-2 Process Wait State Diagram

WAYS TO LEAVE CURRENT STATE

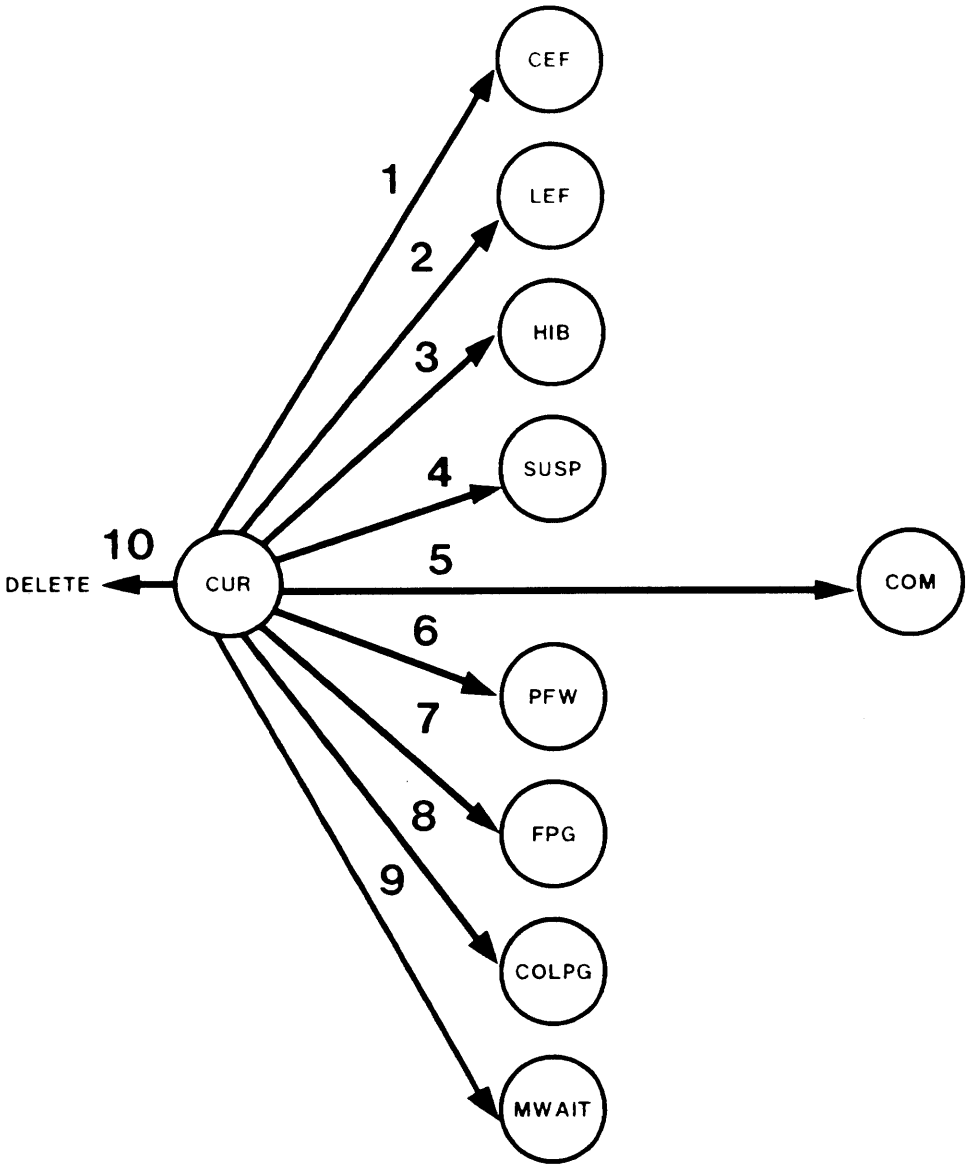


Figure 5-3 Ways to Leave Current State

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)

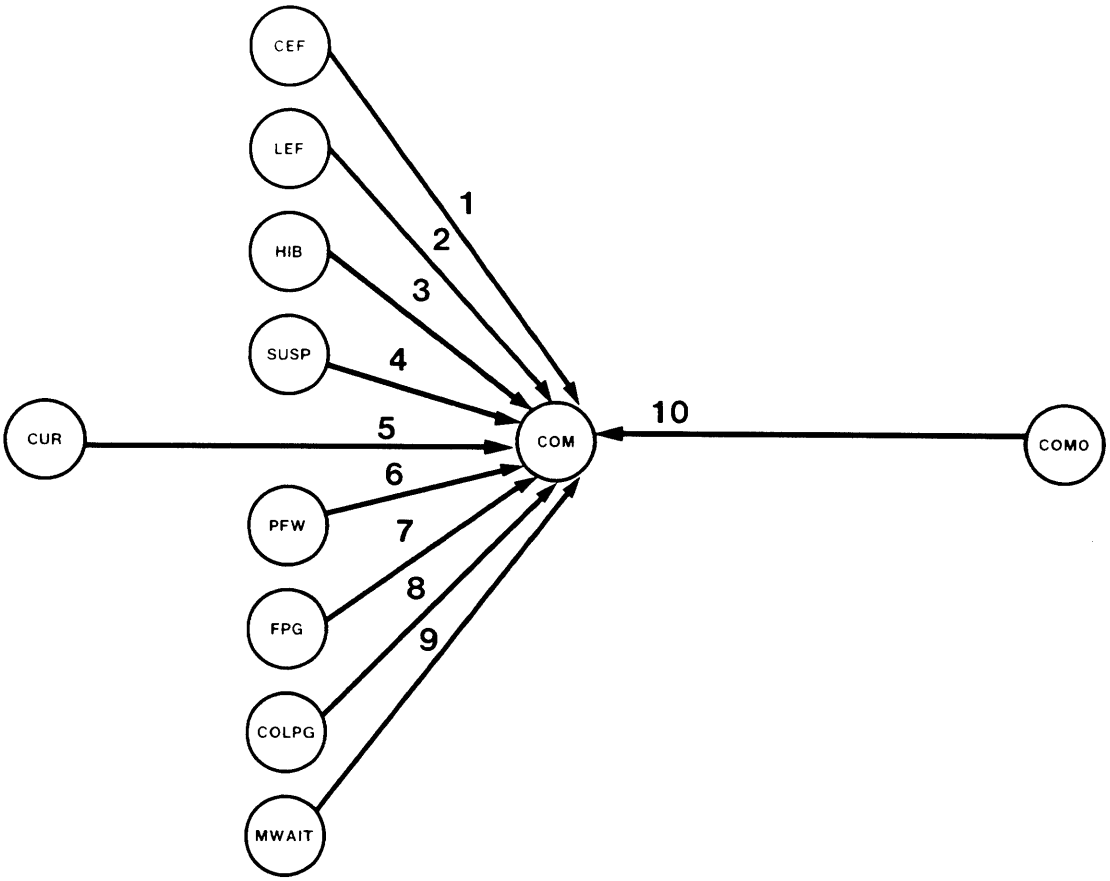


Figure 5-4 Ways to Become Computable (Inswapped)

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

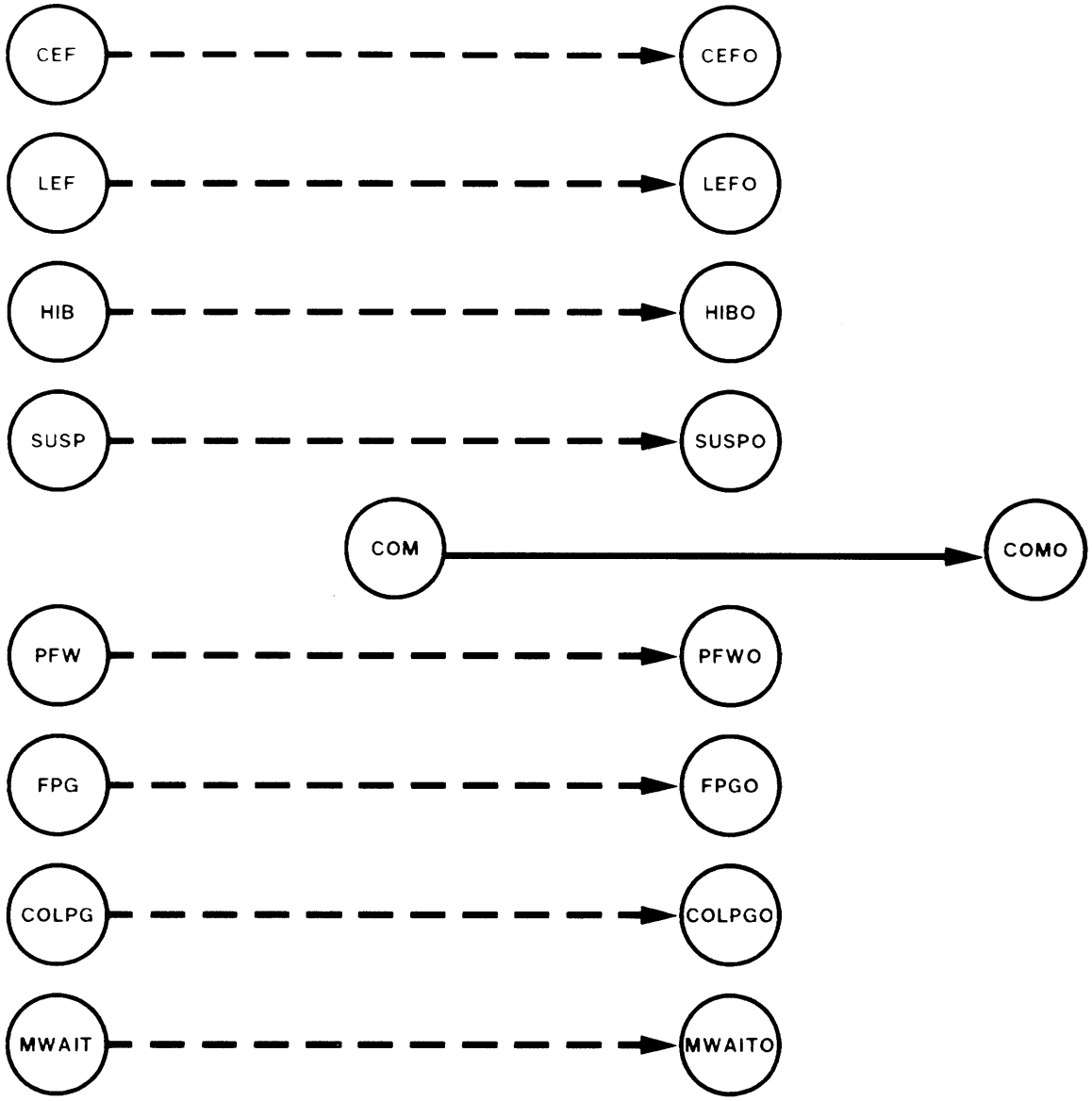


Figure 5-5 Inswapped to Outswapped Transitions

WAYS TO BECOME COMPUTABLE (OUTSWAPPED)

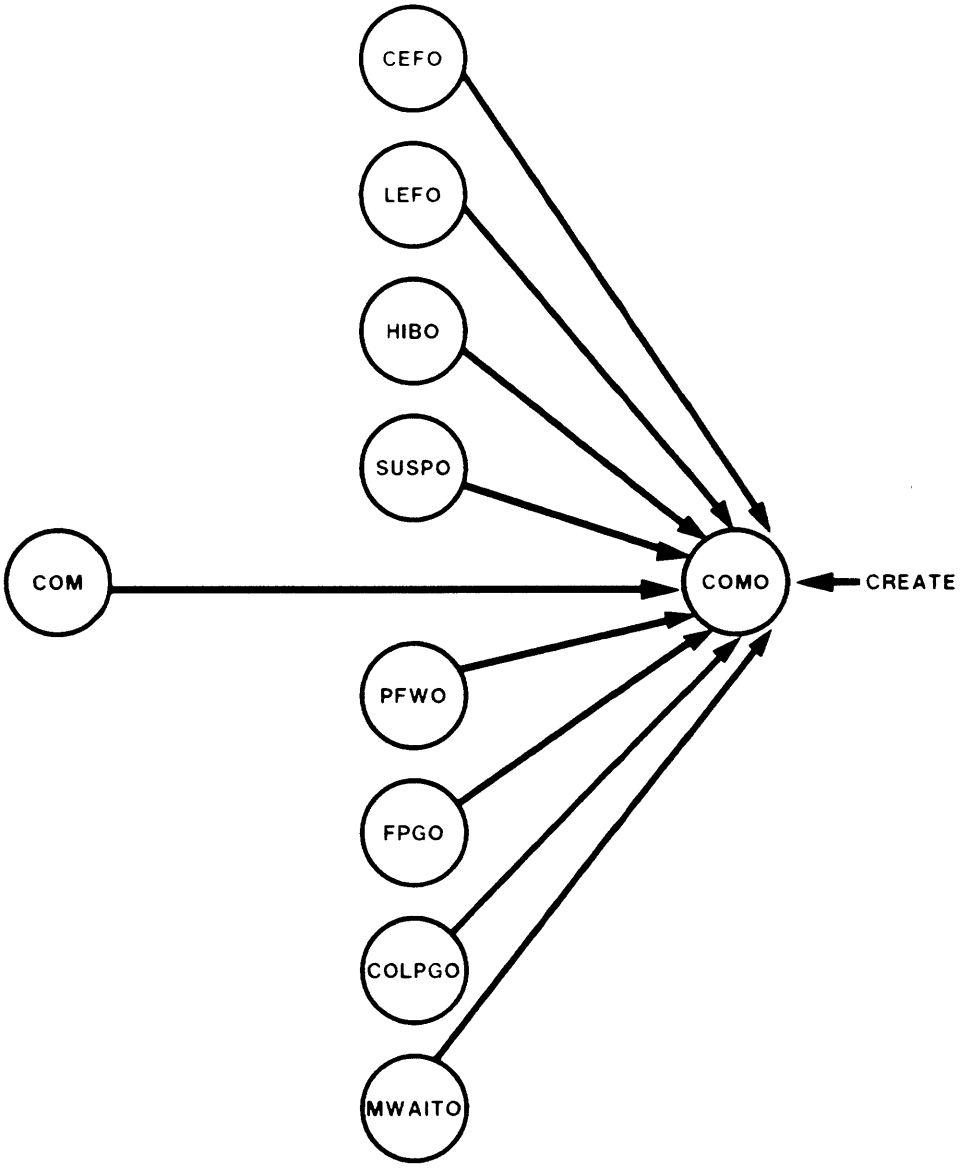
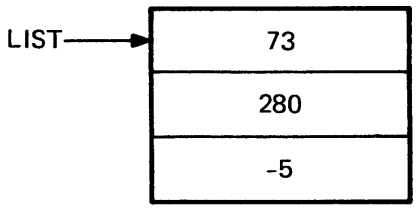


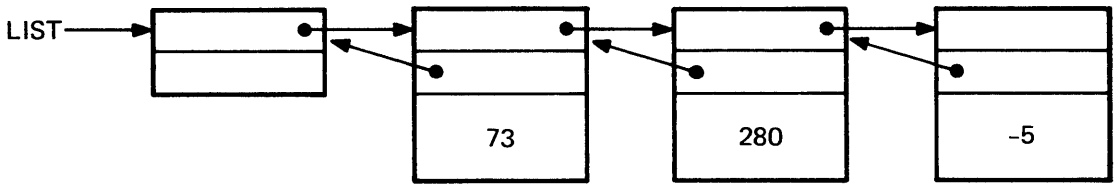
Figure 5-6 Ways to Become Computable (Outswapped)

QUEUES

Information in a table:



Information in a queue:



In General,

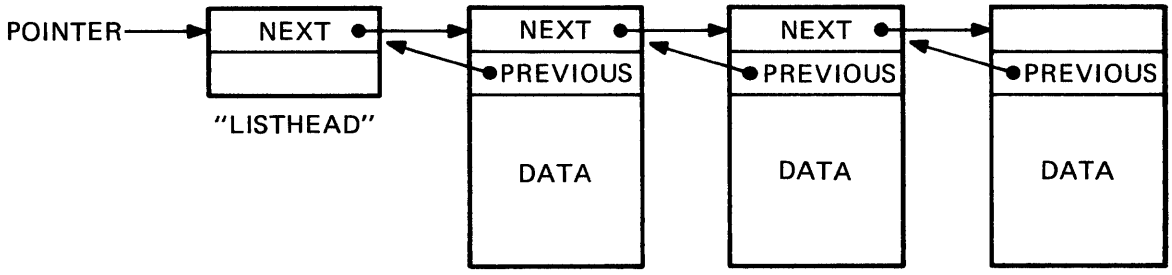


Figure 5-7 Queues

IMPLEMENTATION OF STATES BY QUEUES

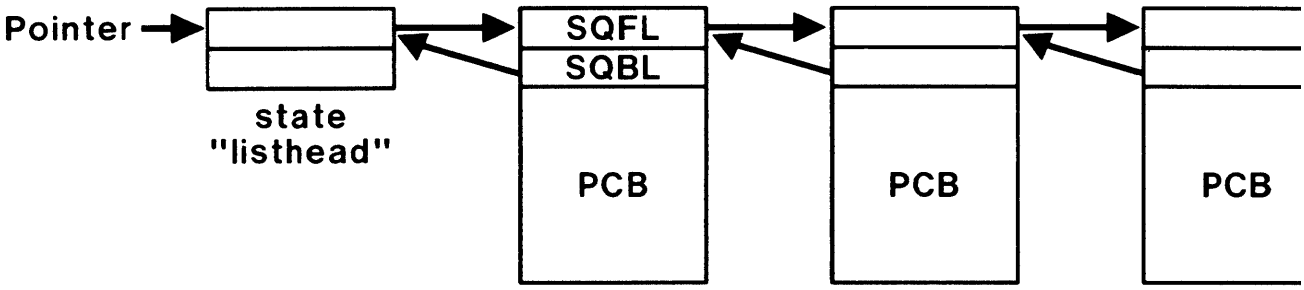


Figure 5-8 A State Implemented in Queues

INSQUE Instruction

The INSQUE instruction changes forward and backward links so that data structure (here the PCB) is inserted in a queue. For example, when a process is removed from execution by the scheduler, its PCB is inserted on a COM queue.

REMQUE Instruction

The REMQUE instruction changes forward and backward links so that data structure is removed from a queue. For example, when a process becomes current, its PCB is removed from a COM queue.

Implementation of COM and COMO States

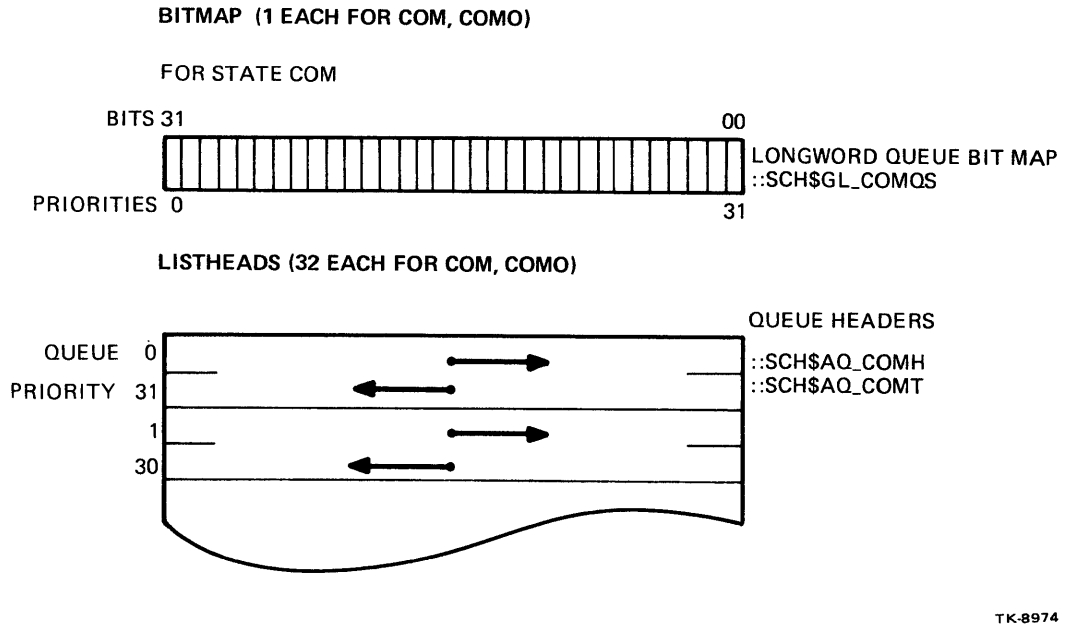


Figure 5-9 Implementation of COM and COMO States

Each bit and listhead corresponds to a scheduling queue at a particular software priority level. Figure 5-10 shows an example of this for computable processes at priority levels 4 and 6.

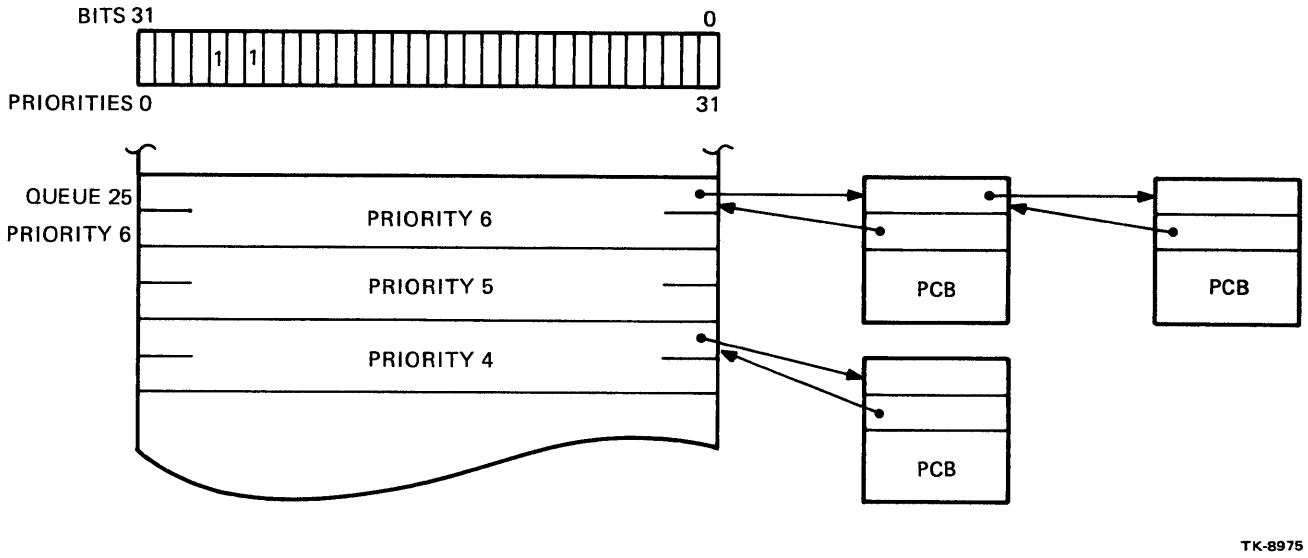


Figure 5-10 Example of Computable Queues

Implementation of Wait States

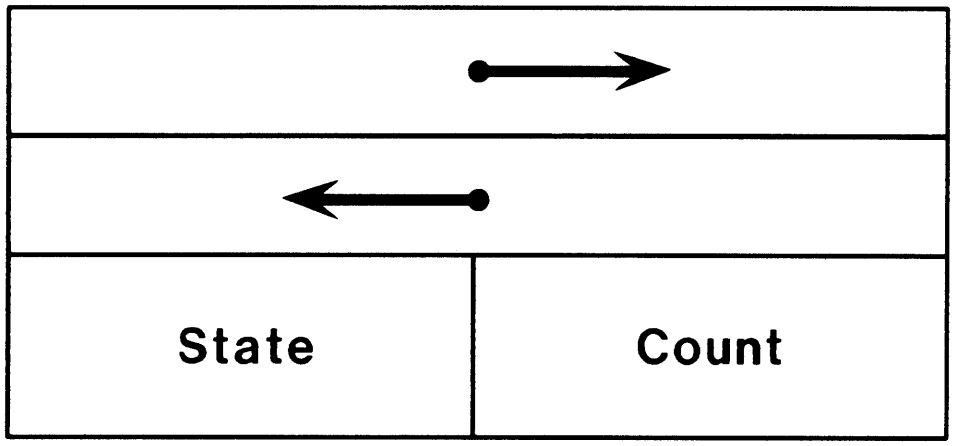


Figure 5-11 Wait State Listhead

The listhead for a wait state contains forward and backward links, a count of the PCBs in the wait queue, and an integer value corresponding to the wait state. Figure 5-12 shows what two processes in hibernate state would look like.

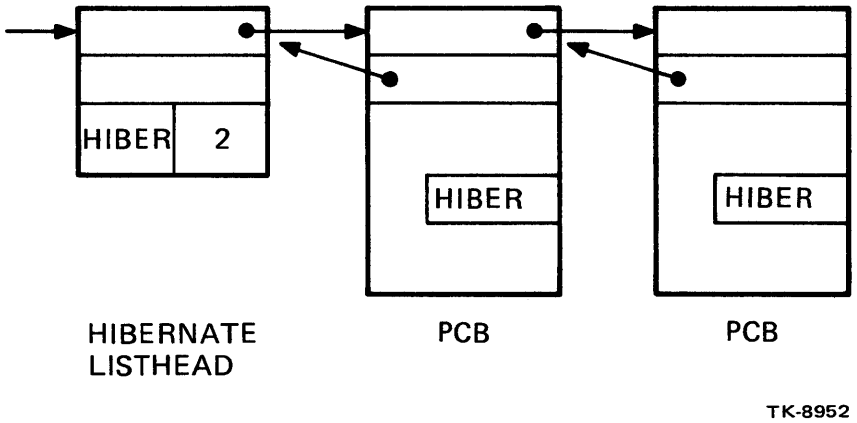


Figure 5-12 Implementation of Wait States

Implementation of CEF State

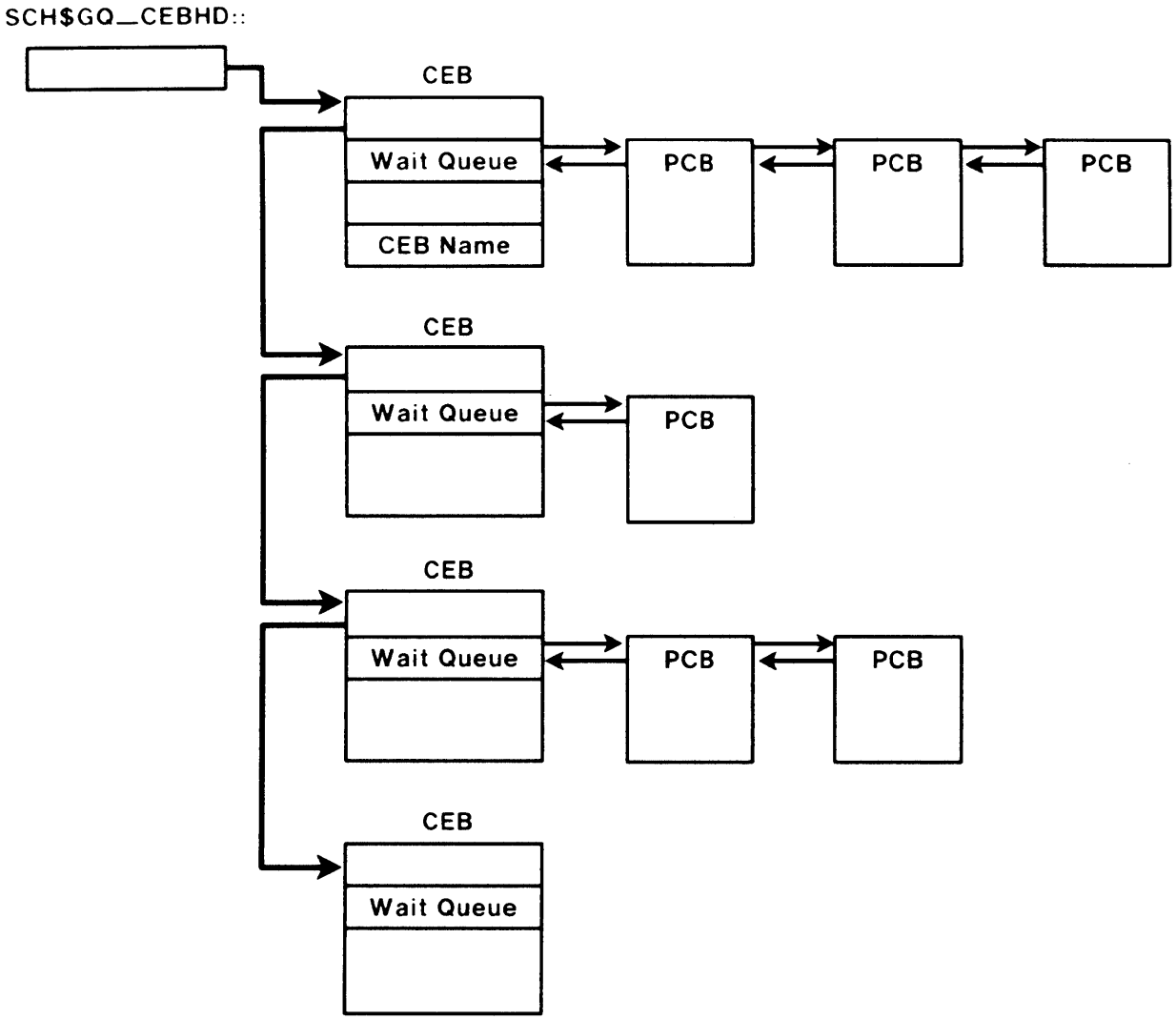


Figure 5-13 Implementation of CEF State

When a common event flag cluster is created, a data structure contains both the cluster and listhead for the PCBs of processes waiting for event flags within the cluster. Therefore, there are as many CEF state queues as there are clusters.

SCHEDULING FIELDS IN SOFTWARE PCB

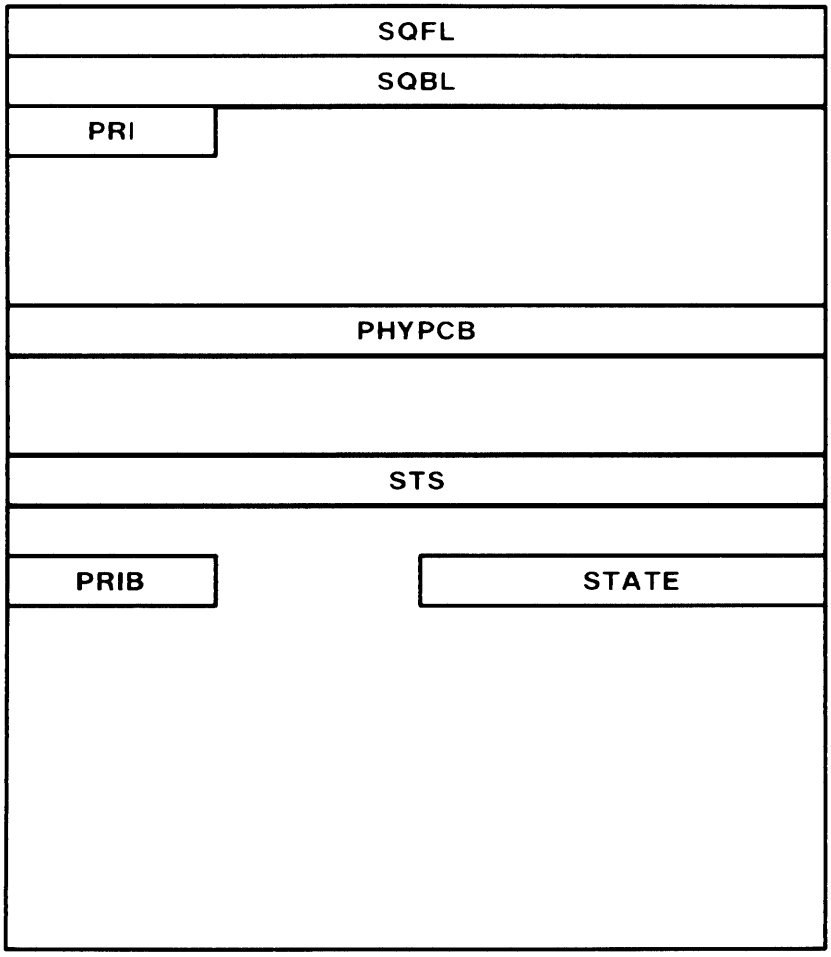


Figure 5-14 Scheduling Fields in Software PCB

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

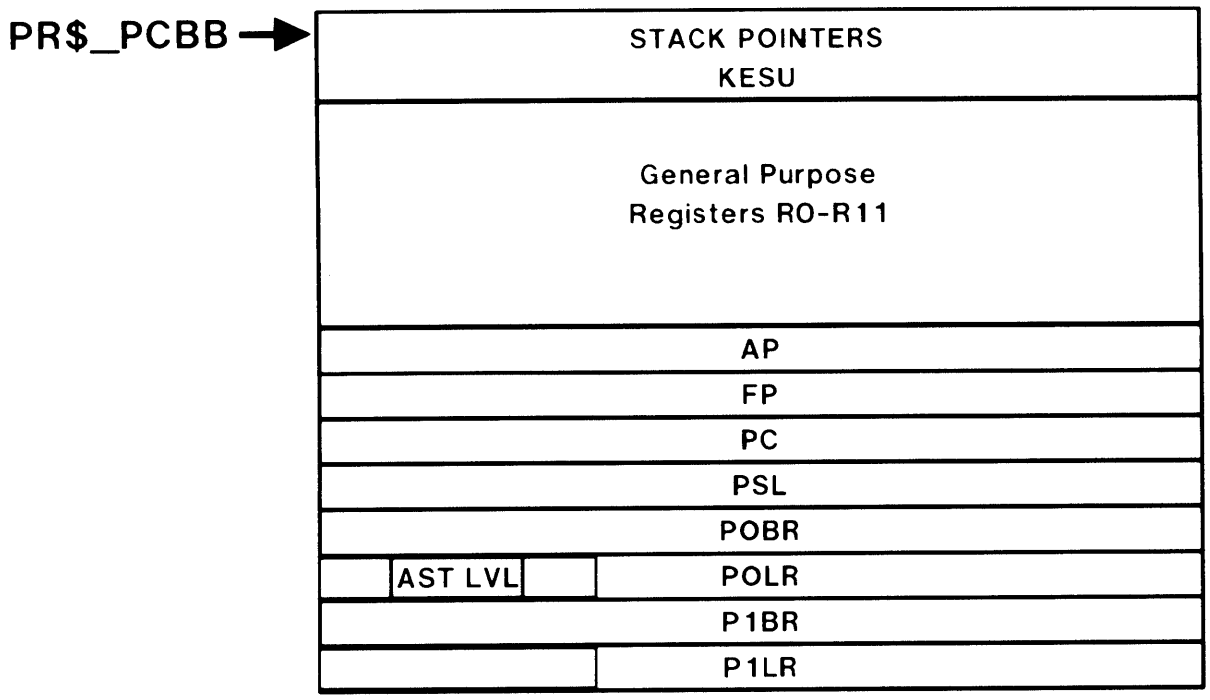


Figure 5-15 Saving and Restoring CPU Registers

Process-specific CPU registers are saved to/restored from hardware PCB during context switch (state change current <--> computable).

SVPCTX - copies stack pointers, general purpose registers, argument pointer, frame pointer, PC, and PSL to current process hardware PCB, then switches to system-wide interrupt stack.

LDPCTX - restores stack pointers, general purpose registers, RP, FP, AST LVL, P0LR, R0BR, P1LR, P1BR from hardware PCB pointed by by PR$_PCBB. Pushes saved PC and PSL on stack, since REI which follows pops them.

SCHEDULER (SCHED.MAR)

```
 1 ; SCH$RESCHED - RESCHEDULING INTERRUPT HANDLER   (1)
 2 ;
 3 ; ENVIRONMENT:
 4 ;   IPL=3 MODE=KERNEL IS=0
 5 ; INPUT:
 6 ;   00(SP)=PC AT RESCHEDULE INTERRUPT
 7 ;   04(SP)=PSL AT INTERRUPT.
 8 ;--
 9     .ALIGN  LONG
10 MPH$RESCHED::    (1)a
11 SCH$RESCHED::
12     SETIPL  #IPL$_SYNCH    (1)b
13     SVPCTX  (1)c
14     MOVL    W^SCH$GL_CURPCB,R1
15     MOVZBL  PCB$B_PRI(R1),R2
16     BBSS    R2,W^SCH$GL_COMQS,10$
17 10$:        MOVW    #SCH$C_COM,PCB$W_STATE(R1)
18     MOVAQ   W^SCH$AQ_COMTD[R2],R3
19     INSQUE  (R1),@(R3)+    (1)d
20 ;+
21 ; SCH$SCHED - SCHEDULE NEW PROCESS FOR EXECUTION   (2)
22 ;
23 MPH$SCHED::
24 SCH$SCHED::      (2)a
25     SETIPL  #IPL$_SYNCH    (2)b
26     FFS     #0,#32,W^SCH$GL_COMQS,R2
27     BEQL    SCH$IDLE
28     MOVAQ   W^SCH$AQ_COMHD[R2],R3
29     REMQUE  @(R3)+,R4    (2)c
30     BVS     QEMPTY
31     BNEQ    20$
32     BBCC    R2,W^SCH$GL_COMQS,20$
33 20$:
34     CMPB    #DYN$C_PCB,PCB$B_TYPE(R4)
35     BNEQ    QEMPTY
36     MOVW    #SCH$C_CUR,PCB$W_STATE(R4)
37     MOVL    R4,W^SCH$GL_CURPCB
38     CMPB    PCB$B_PRIB(R4),PCB$B_PRI(R4)
39     BEQL    30$
40     BBC     #4,PCB$B_PRI(R4),30$
41     INCB    PCB$B_PRI(R4)
42 30$:        MOVB    PCB$B_PRI(R4),W^SCH$GB_PRI
43     MTPR    PCB$L_PHYPCB(R4),#PR$_PCBB
44     LDPCTX    (2)d
45     REI     (2)e
46
47 SCH$IDLE:
48     SETIPL  #IPL$_SCHED
49     MOVB    #32,W^SCH$GB_PRI
50     BRB     SCH$SCHED
51
52 QEMPTY:     BUG_CHECK QUEUEMPTY,FATAL
53
54     .END
```

Example 5-1 Scheduler (SCHED.MAR)

Comments on SCHED.MAR:

1. Current process ---> computable resident
    a. Entry point
    b. Synchronize access to scheduler data base
    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 data base
    c. Remove PCB from head of COM queue
    d. Restore hardware context, push PC and PSL onto stack
    e. Transfer control to current process.

SOFTWARE PRIORITIES AND PRIORITY ADJUSTMENTS

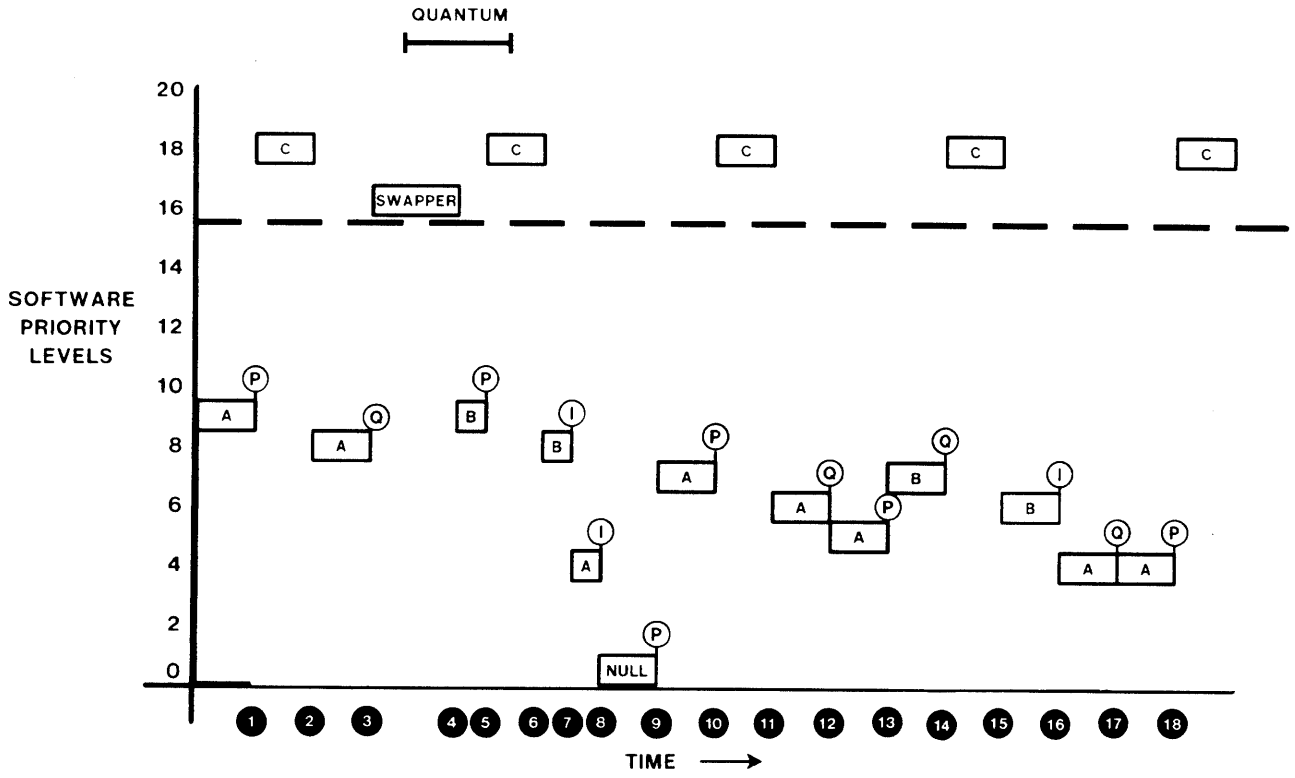


Figure 5-16 Software Priorities and Priority Adjustments

Notes on Software Priorities and Priority Adjustments:

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.
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.)
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'.
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'.

STEPS AT QUANTUM END

For Real Time Process

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

For 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.

WSSIZE VARIATION OVER TIME

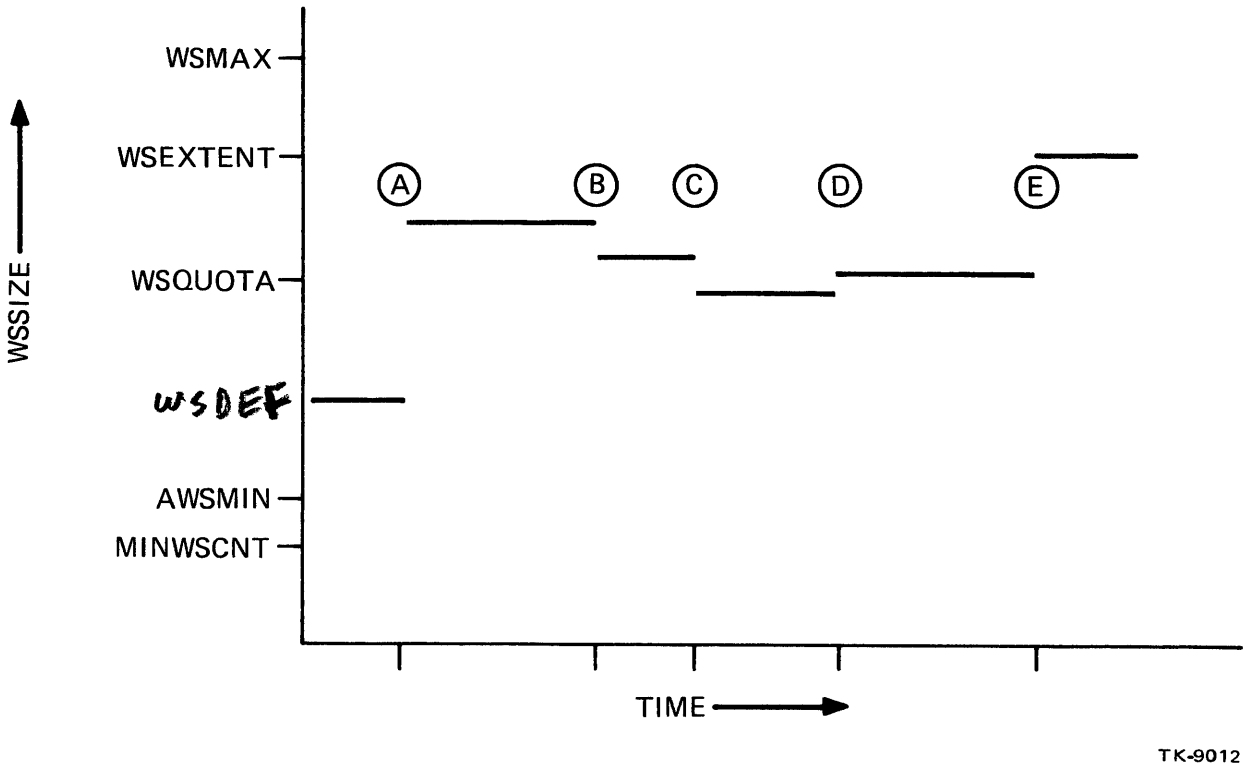


Figure 5-17 WSSIZE Variation Over Time

Table 5-1 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

AUTOMATIC WORKING SET ADJUSTMENT

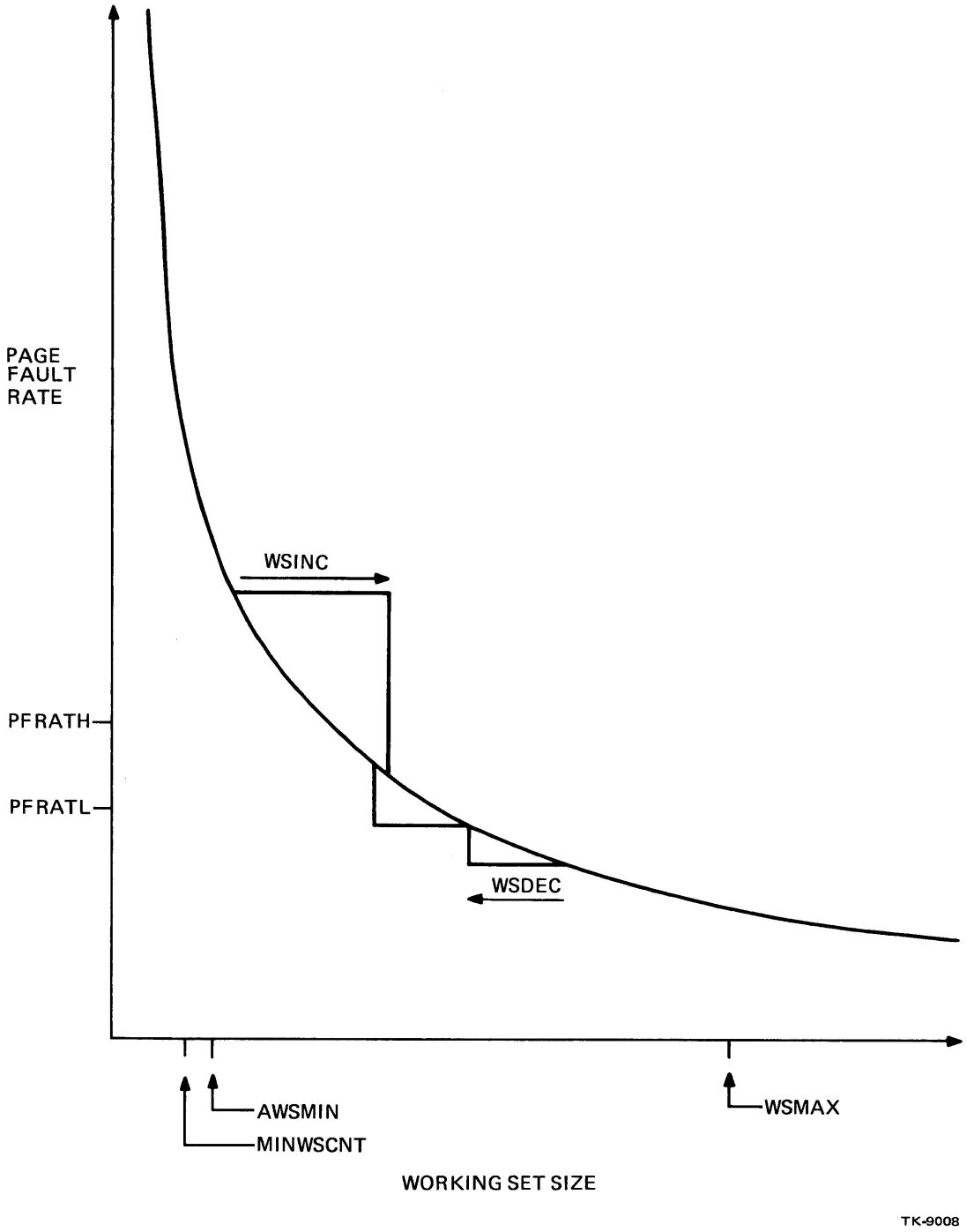


Figure 5-18 Automatic Working Set Adjustment

IOTA

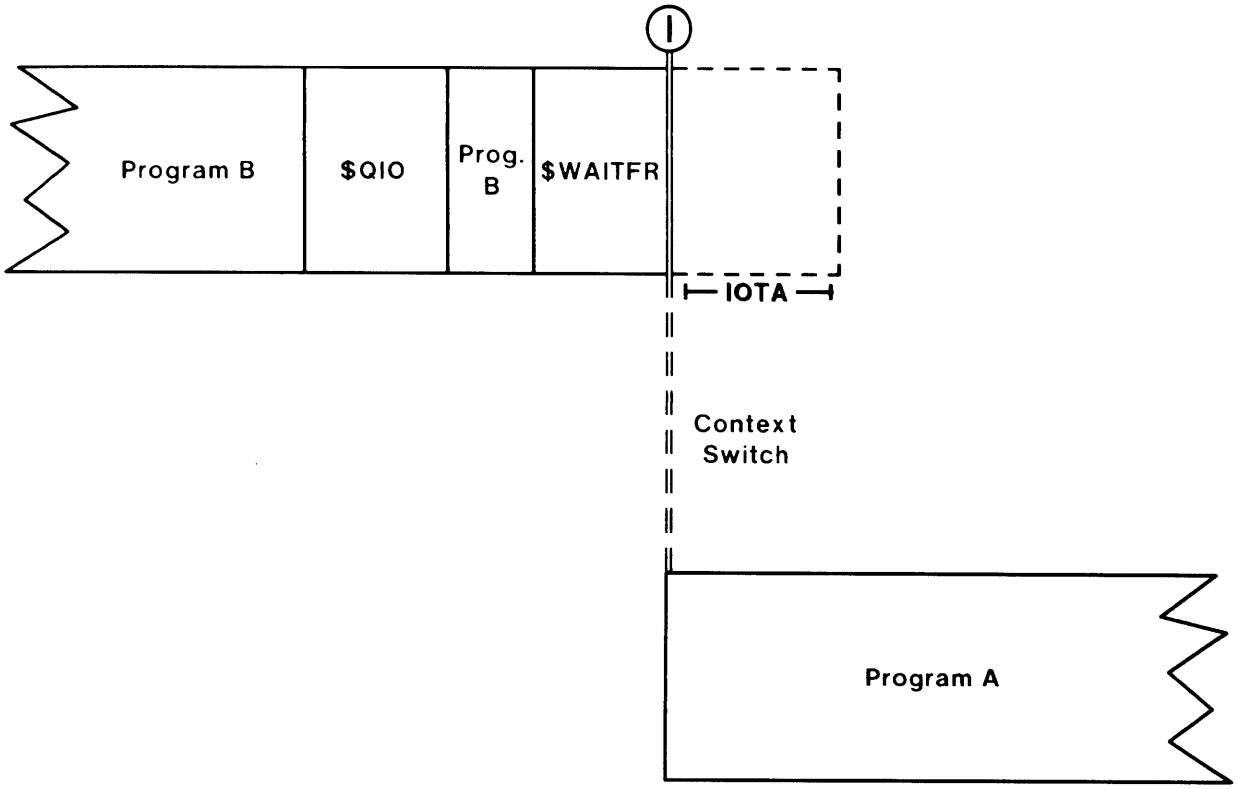


Figure 5-19 IOTA

The special system parameter IOTA deducts a fixed amount from the time remaining in the quantum of a process whenever a process enters a wait state. This surcharge is not applied when the process is preempted by a process of higher priority. It is used to force processes to reach quantum end.

SOFTWARE PRIORITY LEVELS OF SYSTEM PROCESSES

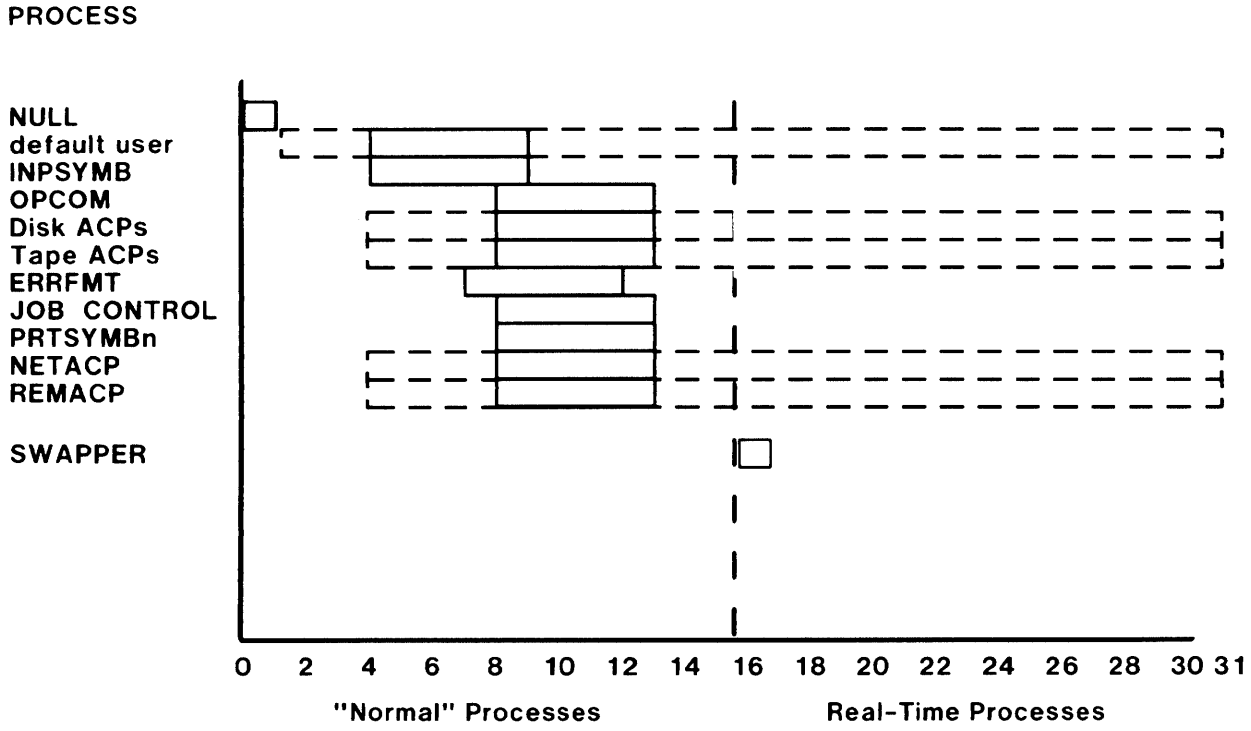


Figure 5-20 Software Priority Levels of System Processes

Most system processes have a fixed base priority established during system initialization. The base priority of disk and tape ACPs is controlled by the system parameter ACP_BASEPRIO. The initial base priority of a process is controlled by an argument to the $CREPRC system service.

MISCELLANEOUS RESOURCE WAITSTATES (MWAIT)

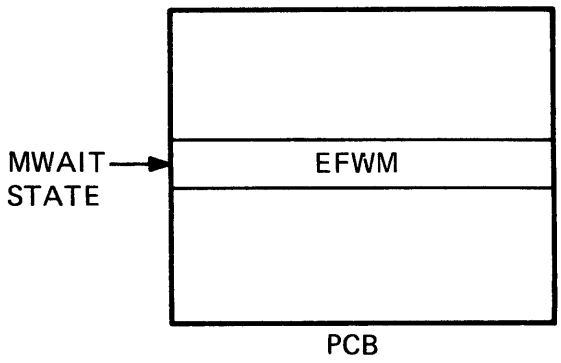


Figure 5-21 Miscellaneous Resource Wait States (MWAIT)

- Nonpaged Dynamic Memory
- Modified Page List Too Large
- Paged Dynamic Memory

  See $RSNDEF.

REPORT SYSTEM EVENT (RSE.MAR)

1. System events cause transitions between process states.

2. These transitions are effected by the routine RSE.MAR.

3. Inputs to RSE

   a. PCB address

   b. Event number (i.e., number for WAKE, CEF SET, etc.)

4. RSE flow

   a. Event checked for significance (e.g., WAKE only if in HIBER state).

   b. PCB removed from wait queue and wait queue header count decremented.

   c. PCB inserted on COM or COM0 state queue after priority adjustment, and summary bit set.

   d. Swapper process may be awakened (if PCB was inserted on COM0 queue).

   e. Scheduler interrupt at IPL 3 requested if the new computable process has software priority greater than that of current process.

PAGING

6

INTRODUCTION

There are two functions required of the memory management subsystem of the operating system. The first gives each user program the impression that it is running in contiguous physical memory, starting at address zero. The second function divides the available physical memory equitably among the users of the system.

The first function requires that the user's virtual address be translated to a physical address. If the data is already in memory, the translation is done by hardware. When a program refers to data that is on disk, software is invoked to bring the data into memory. This software is an exception service routine called the pager.

VMS implements the second function by using working sets and paging. Each process is required to execute with a limited amount of its data in memory. To avoid fragmentation of physical memory, this data is divided into 512 byte pieces, called pages. The number of valid pages a process has in memory at any time is called the working set.

Because the working set limit represents the amount of physical memory "owned" by a process, processes at their working set limits must replace pages in the working set with newly demanded ones (rather than simply acquiring more physical memory). This replacement is performed by the pager.

OBJECTIVES

Upon completion of this module, you will be able to:

- Describe the effects of changing working set size, creating/deleting virtual address space, and creating/mapping a global section.
- Discuss the programming considerations that affect paging overhead.
- Given a set of initial conditions and a page request, describe the changes in the status and locations of pages and the changes in process states.
- Discuss the effects of altering SYSGEN parameters governing paging.

RESOURCES

Reading

1. VAX/VMS Internals and Data Structures Manual, chapters on memory management data structures, paging dynamics, and memory management system services.

Additional Suggested Reading

1. VAX/VMS Internals and Data Structures Manual, chapter on image activation and termination.

Source Modules

Facility Name	Module Name
SYS	PAGEFAULT ALLOCPFN SVAPTE
	SYSADJWSL,SYSLKWSET, SYSPURGWS SYSCRMPSC,SYSDGBLSC SYSCREDEL RSE IOCIOPOST
RTL	LIBVM

TOPICS

Paging: Per process memory management

Order of topics to be discussed:

1. Linker action in creating executable files
2. Image activator setting up Process Header
3. Invoking pager routine
4. Memory management data structures
5. Following a process page faulted in and out of a process
6. Following a global page faulted in and out of a process

Reason for study:

- To understand how various sections of VMS interrelate for memory management
- How SYSGEN parameters relate to memory management code and data structures
- To be able to read the paging code

ADDRESS TRANSLATION

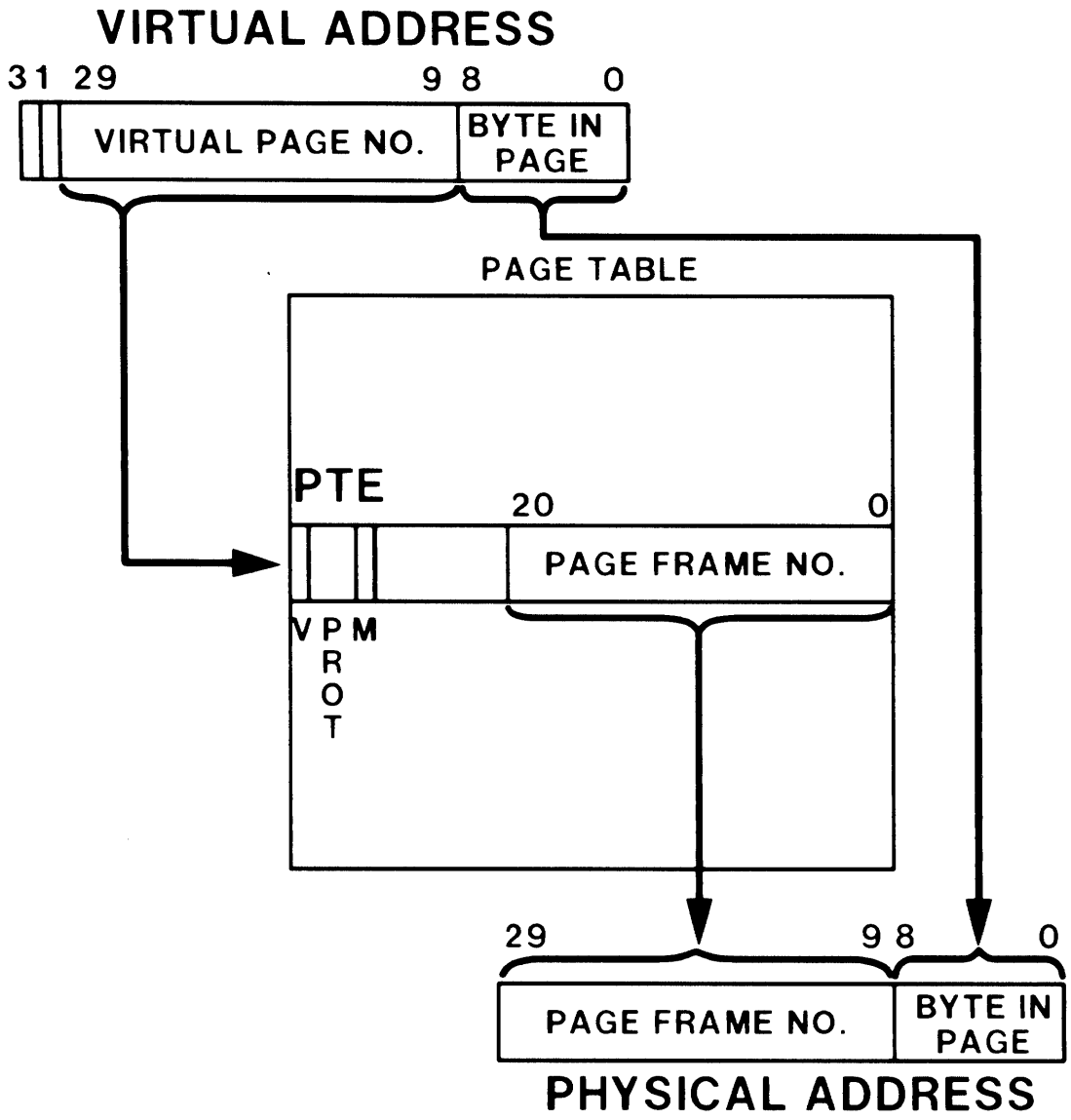


Figure 6-1 Address Translation

Address translation is the hardware operation of converting a virtual address into a physical address for actual execution of an instruction. The conversion, or mapping, information is located in an entry in the appropriate page table.

RESOLVING PAGE FAULTS

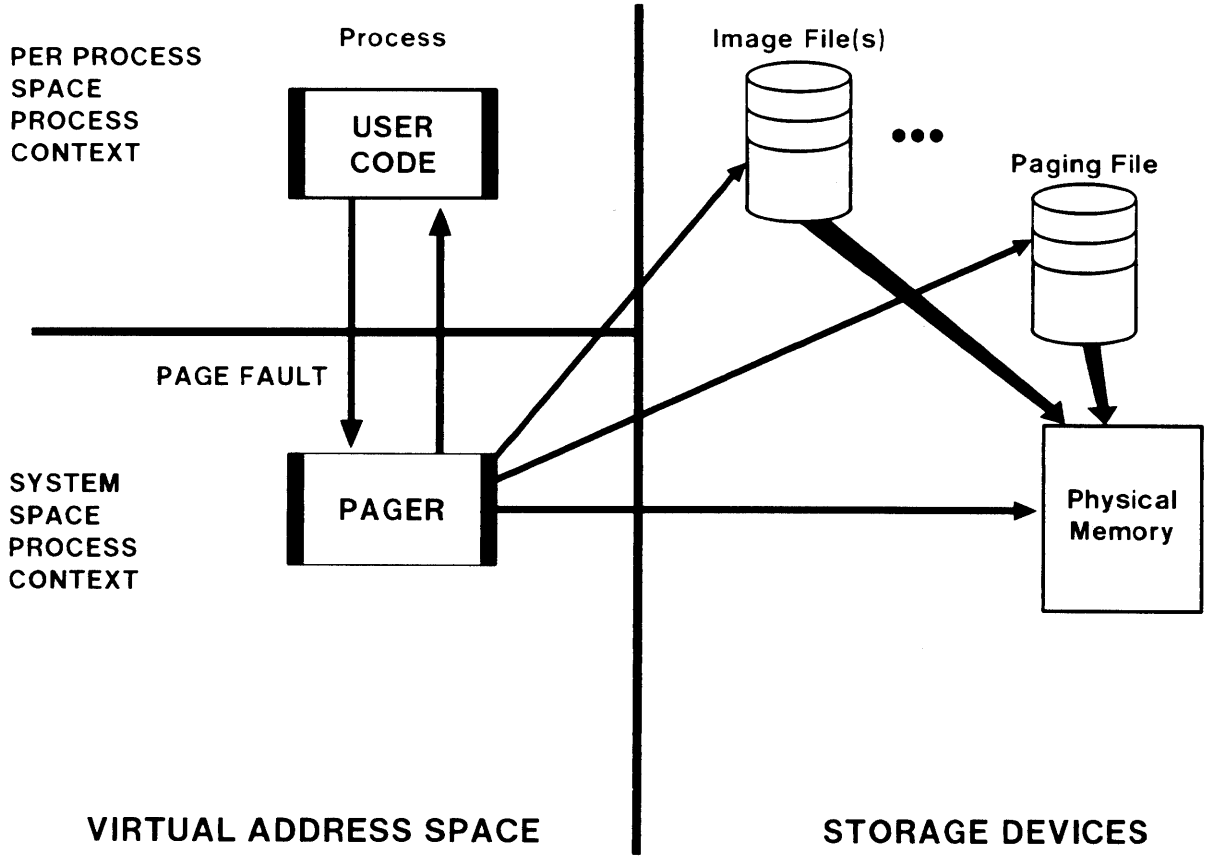


Figure 6-2 Resolving Page Faults

- Pager is an exception service routine executing within the context of the process that incurred the page fault
- Not already in memory - read I/O issued to image file or page file
- Already in memory - taken from free or modified page list, or valid global page

PROCESS SECTIONS AND IMAGE FILE

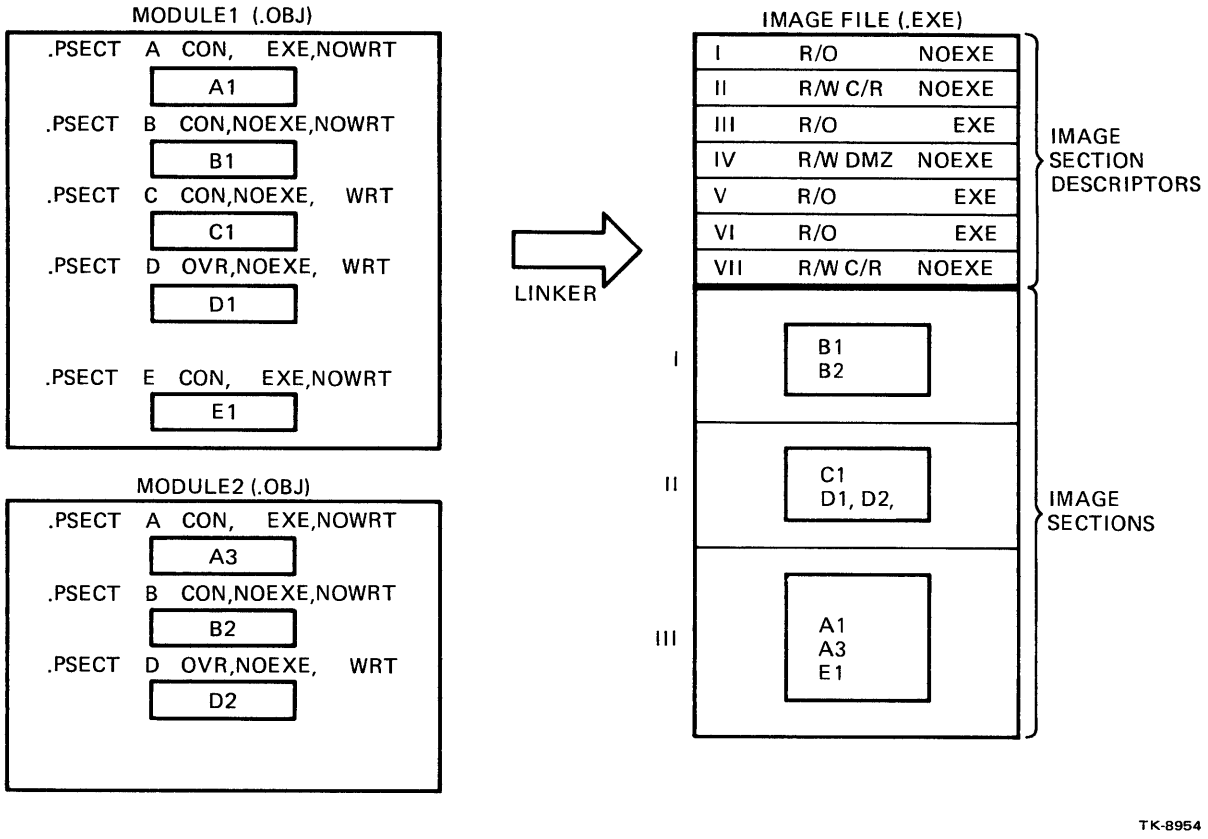


Figure 6-3 Process Sections and Image File

Image Sections Stored in Image File

I. Read Only Data
II. Read/Write Data
III. Code
IV. User Stack

If Run Time Library Referred to -
V. RTL Transfer Vectors
VI. RTL Code
VII. RTL Private Impure Data

IMAGE FILE AND PROCESS HEADER

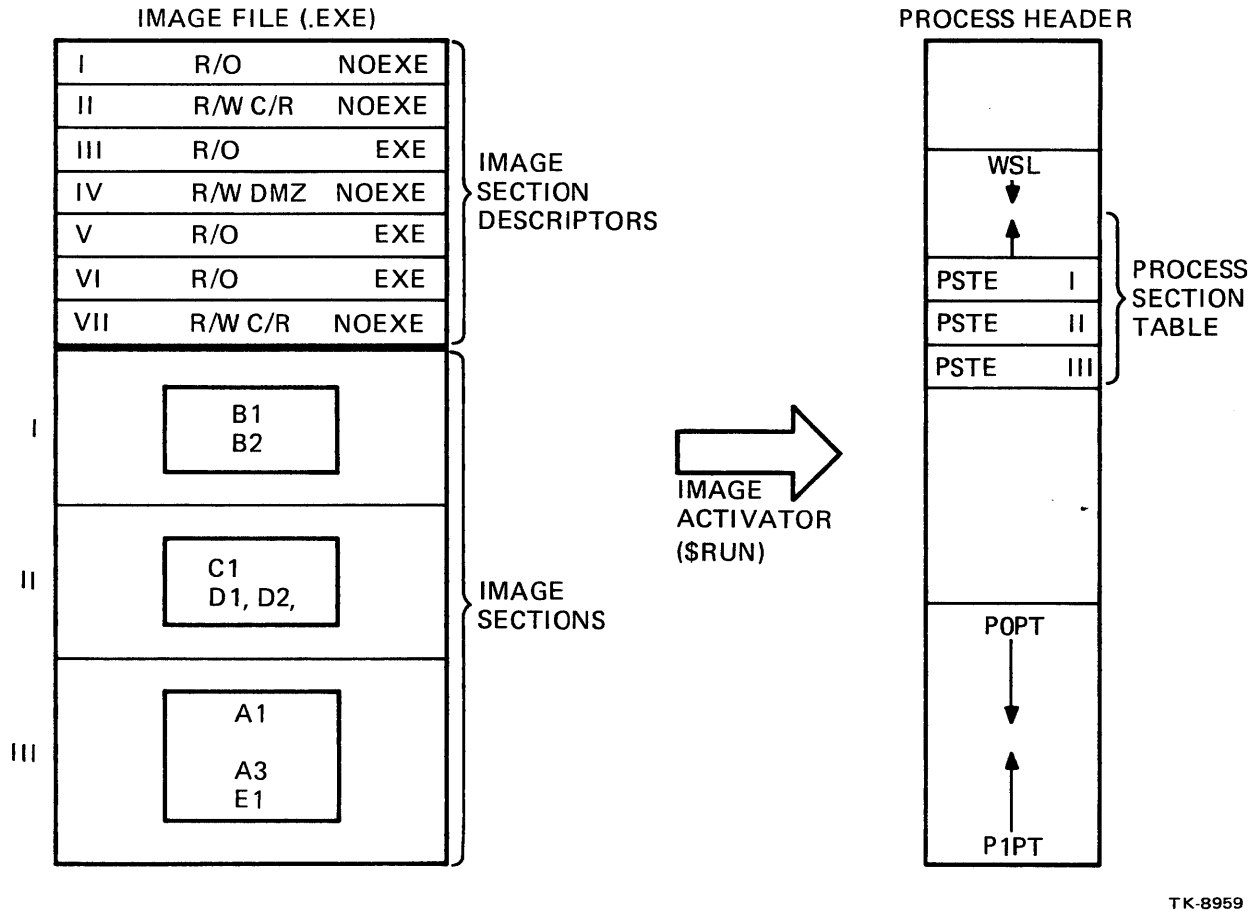


Figure 6-4 Image File and Process Header

Image Activator

- Fills in Process Section Table entries from the image section descriptors.
- Fills in the Page table entries from DMZ, private and global pages.
- Resolves any shared addresses.

IMAGE SECTION DESCRIPTOR FORMATS

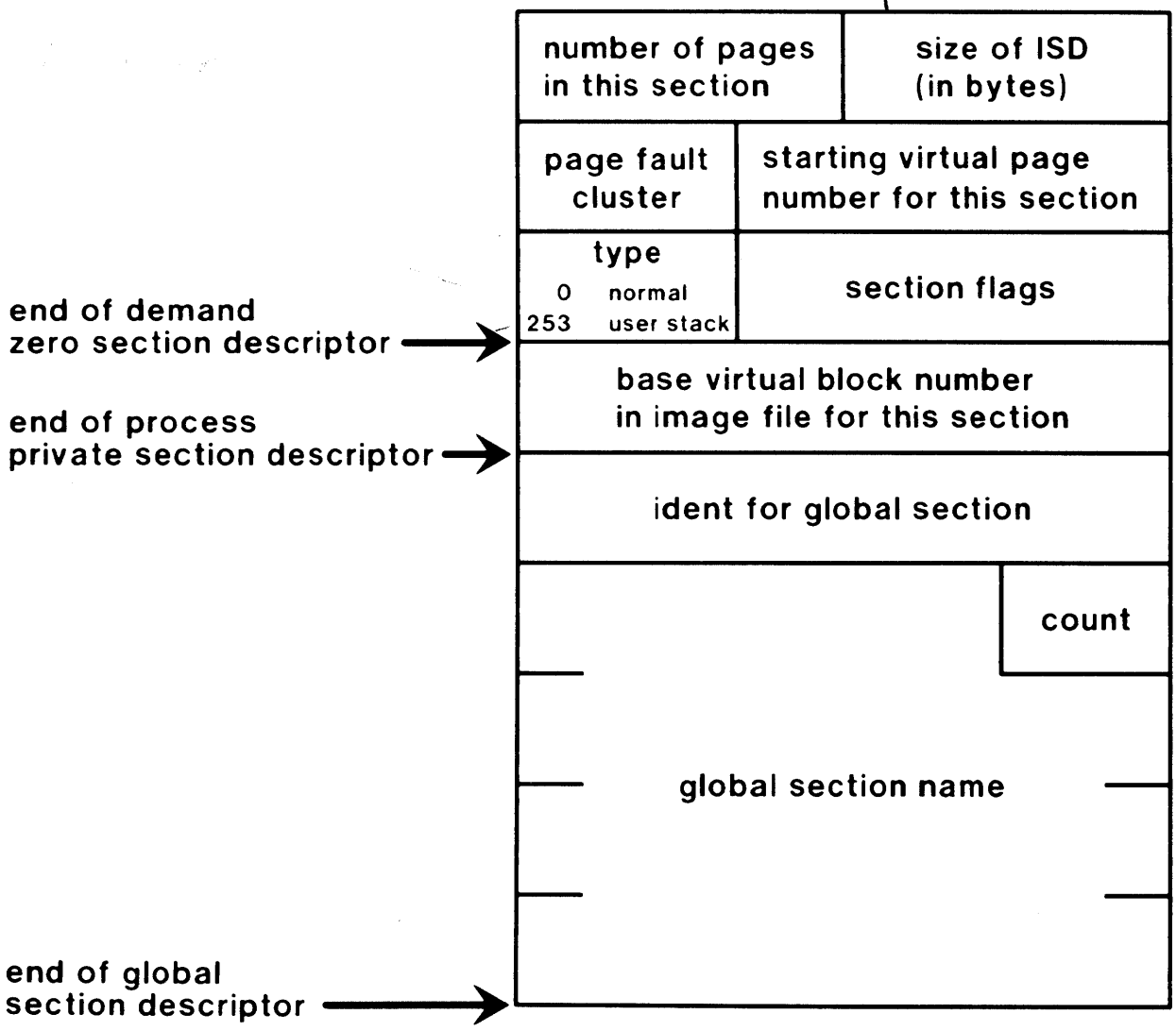


Figure 6-5 Image Section Descriptor Formats

HOW PTEs, PSTEs ARE FILLED IN

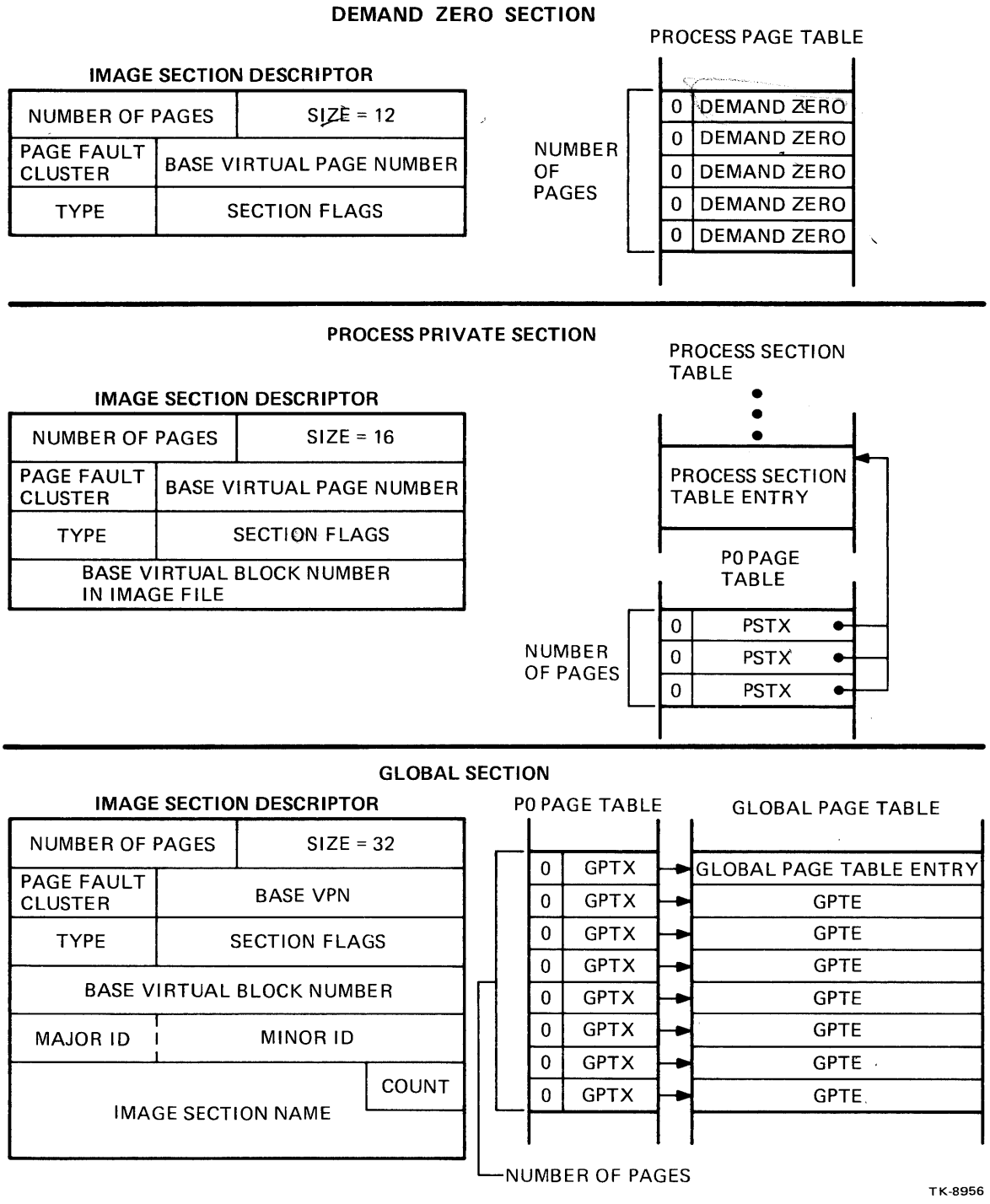


Figure 6-6 How PTEs, PSTEs Are Filled In

PAGE TABLES MAP VIRTUAL ADDRESS SPACE

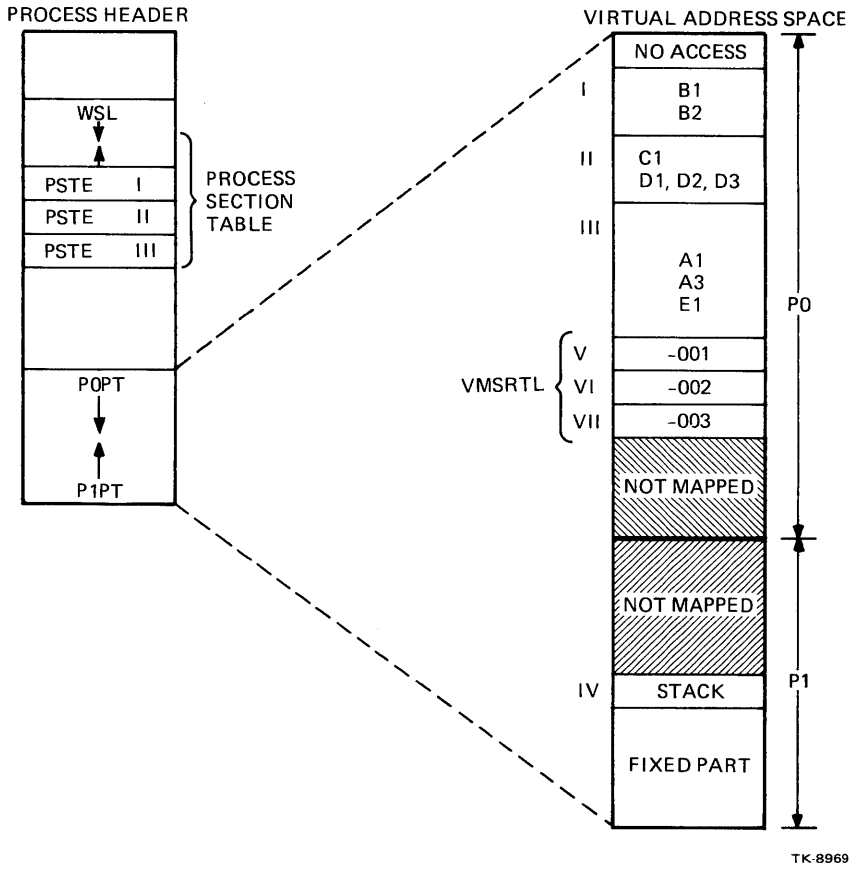


Figure 6-7 Page Tables Map Virtual Address Space

P0 Space

- No Access Page
- Image and Code
- VMSRTL (Run Time Library)

P1 Space

- User Stack
- Image Specific Portion

DATA STRUCTURES USED BY THE PAGER

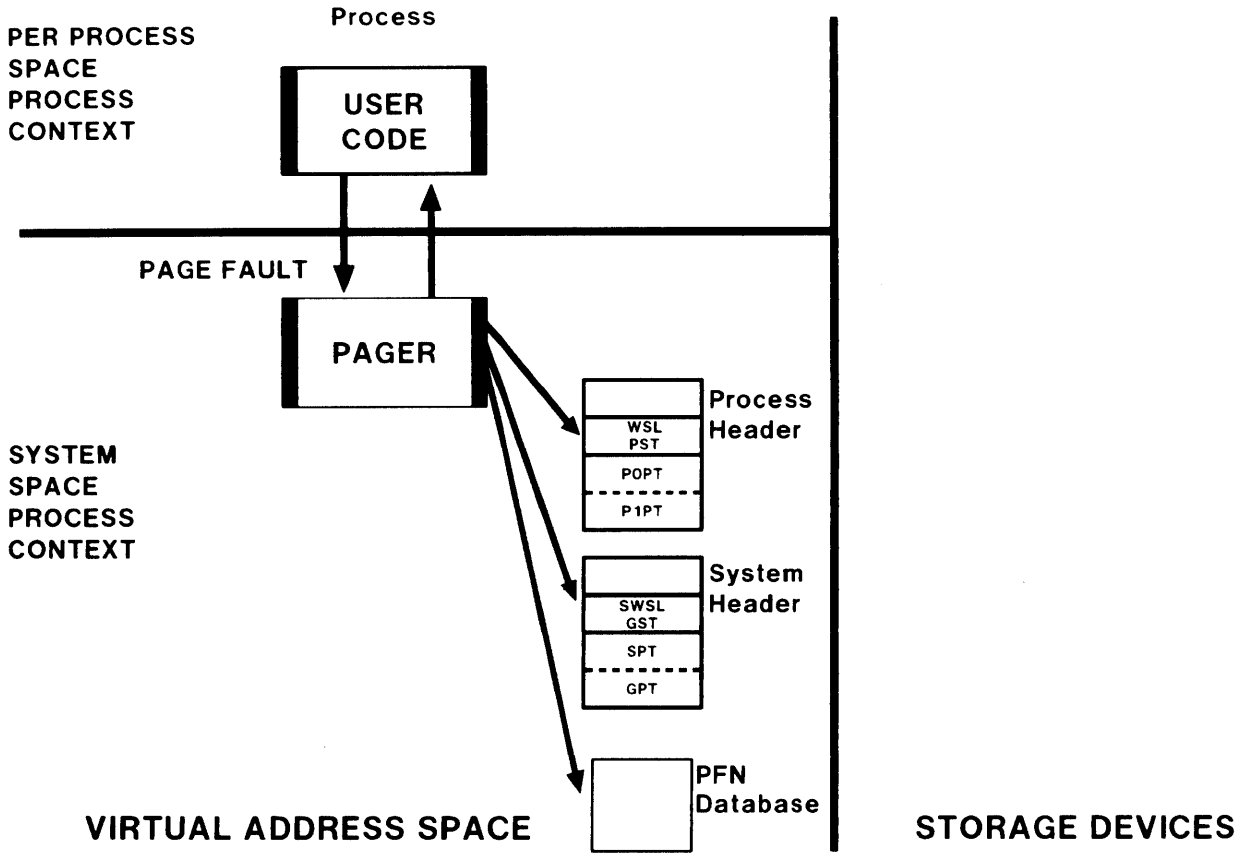


Figure 6-8 Data Structures Used by the Pager

Table 6-1 Where Memory Management Information is Stored

Memory Management Information	Data Structure
Process (PØ and Pl space)	Process Header - Process Section Table - Page Table
System (SØ space)	System Header - System Page Table
Global Sections	System Header - Global Page Tables
Physical Memory	PFN Data base (Page Frame Number)

PHYSICAL ADDRESS SPACE

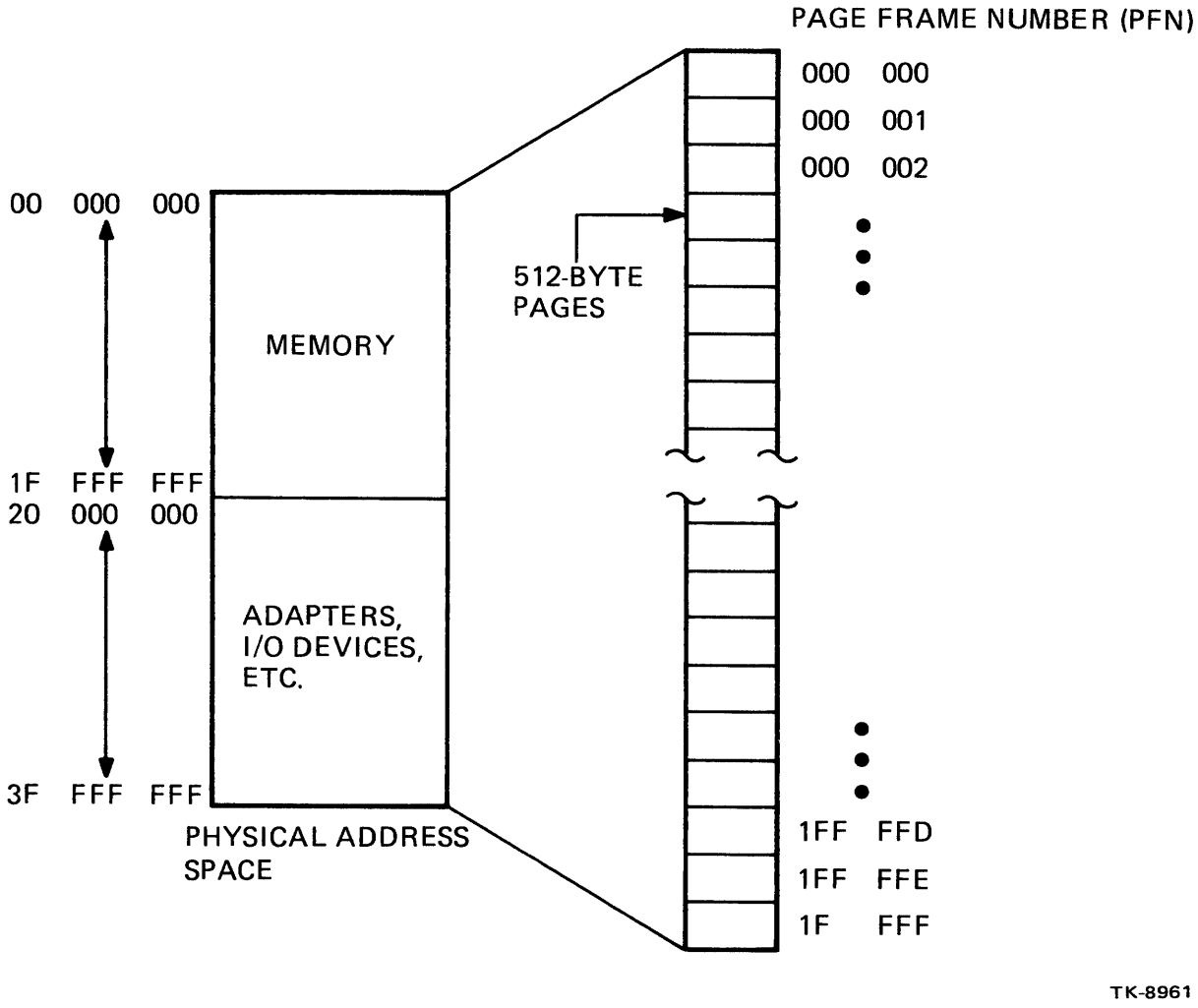


Figure 6-9 Physical Address Space

*SYSGEN
PHYSICALPAGES

VIRTUAL AND PHYSICAL MEMORY

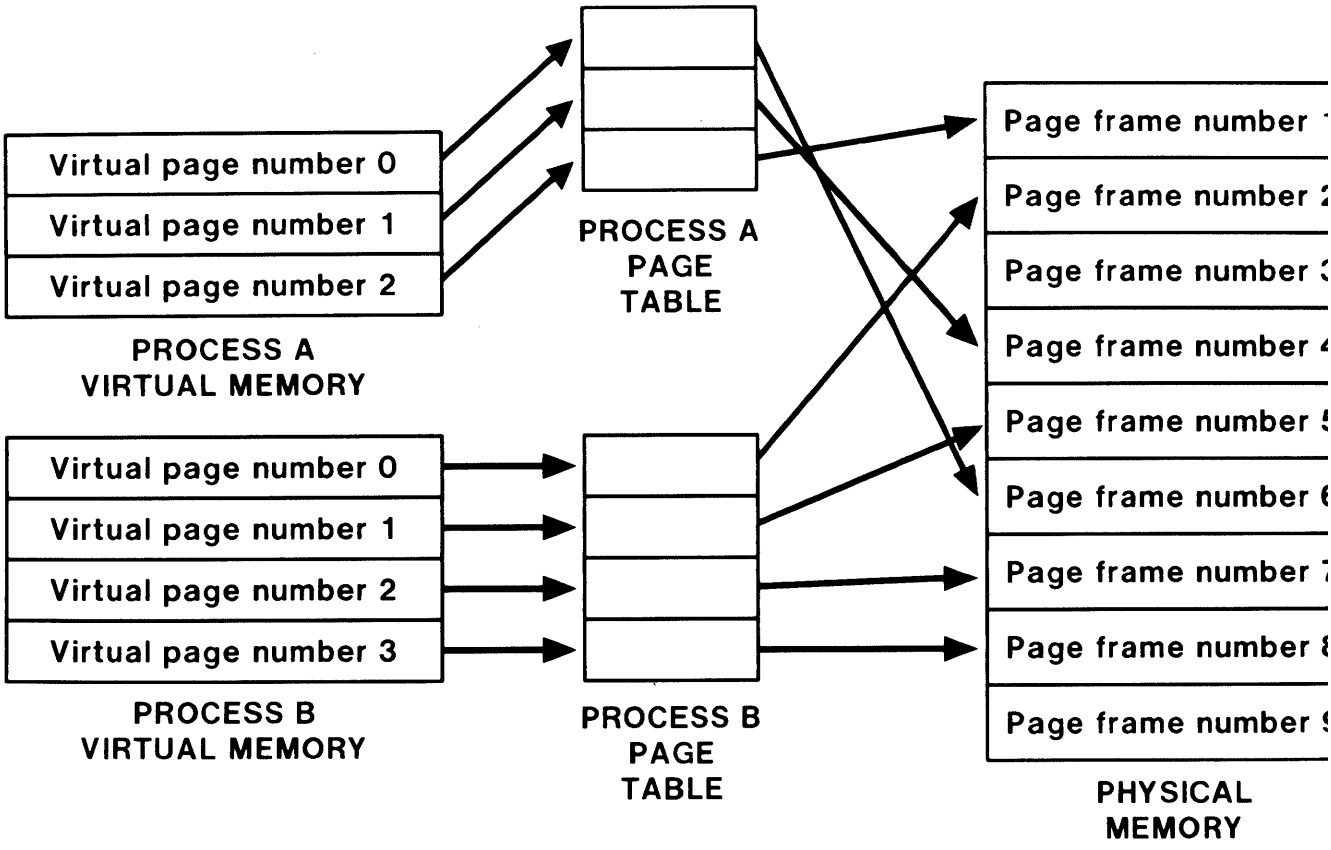


Figure 6-1Ø Virtual and Physical Memory

- Process A has Allocated
  PFN 1
  PFN 4
  PFN 6

- Process B has Allocated
  PFN 2
  PFN 5
  PFN 7
  PFN 8

- Translation from virtual to physical address done using Page Tables (see Appendix)

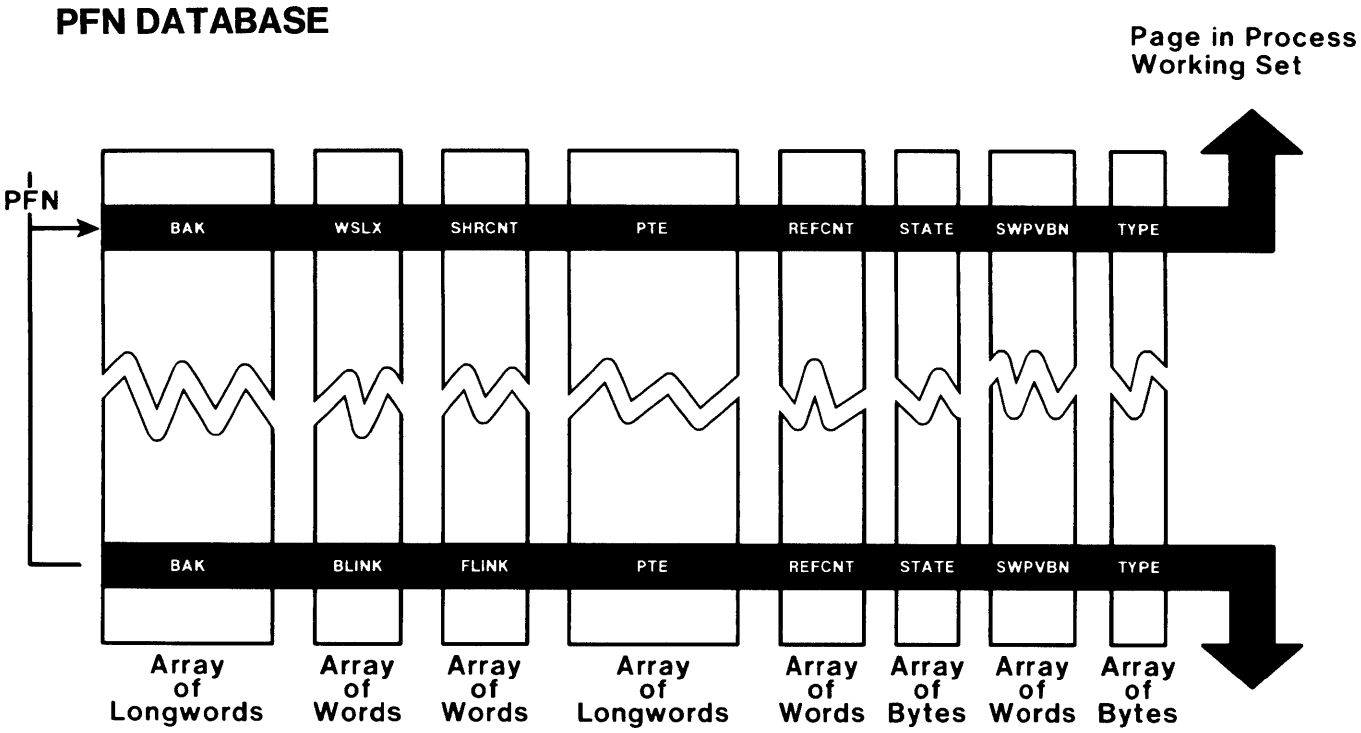


Figure 6-11 PFN Database

BAK	where page should go if it must leave memory
WSLX, BLINK	index into working set list, backward link
SHRCNT,FLINK	number of processes sharing page, forward link
PTE	virtual address of PTE that maps this page
REFCNT	number of reasons not to put page on free or modified page list
STATE	specifies list or activity
SWAPVBN	virtual block number in swap file or page file
TYPE	type of page - e.g., process, system global

Note: PFN is index into arrays.
FLINK, BLINK arrays may be longwords for large physical memory.

PROCESS HEADER

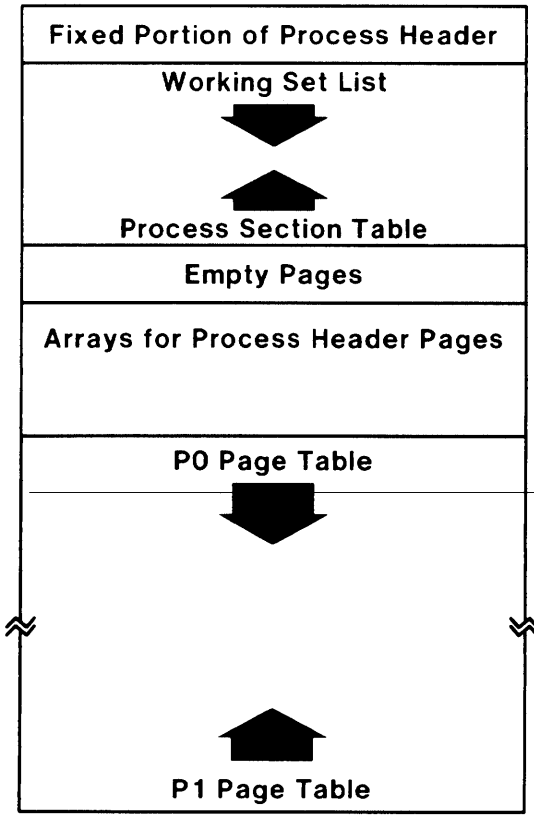


Figure 6-12 Process Header

The process header contains all of the memory management information about a process.

Four major areas of the process header are used in paging operations:

Area	SYSGEN Parameters
• PØ page table	VIRTUALPGCNT
• Pl page table	
• Process Section Table	PROCSECTCNT
• Working set list	WSMAX

*SYSGEN -

VIRTUALPAGECNT

WORKING SET LIST

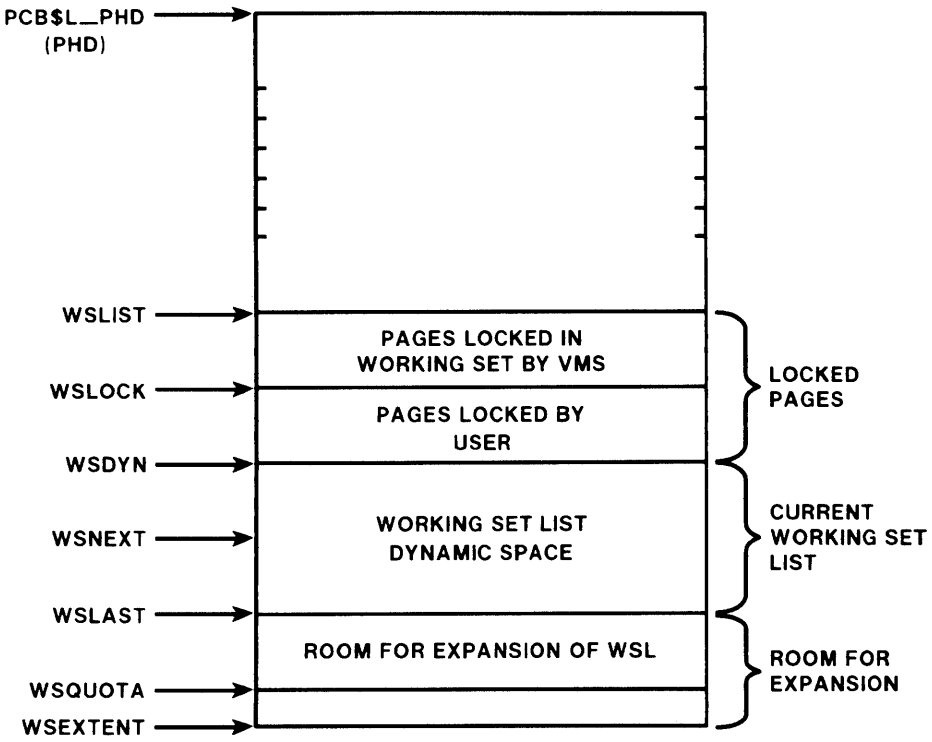


Figure 6-13 Working Set List

- WSLAST can move to

  WSQUOTA if few free pages (free page count <BORROWLIM)
  WSEXTENT if many free pages (free page count >BORROWLIM)

- WSNEXT - latest entry put in working set list

- Page replacement scheme is First In, First Out

*SYSGEN -

BORROWLIM

WSMAX

PQL_DWSDEFAULT, PQL_MWSDEFAULT

PQL_DWSEXTENT, PQL_MWSEXTENT

PQL_DWSQUOTA, PLQ_MWSQUOTA

MINWSCNT

PROCESS SECTION TABLE

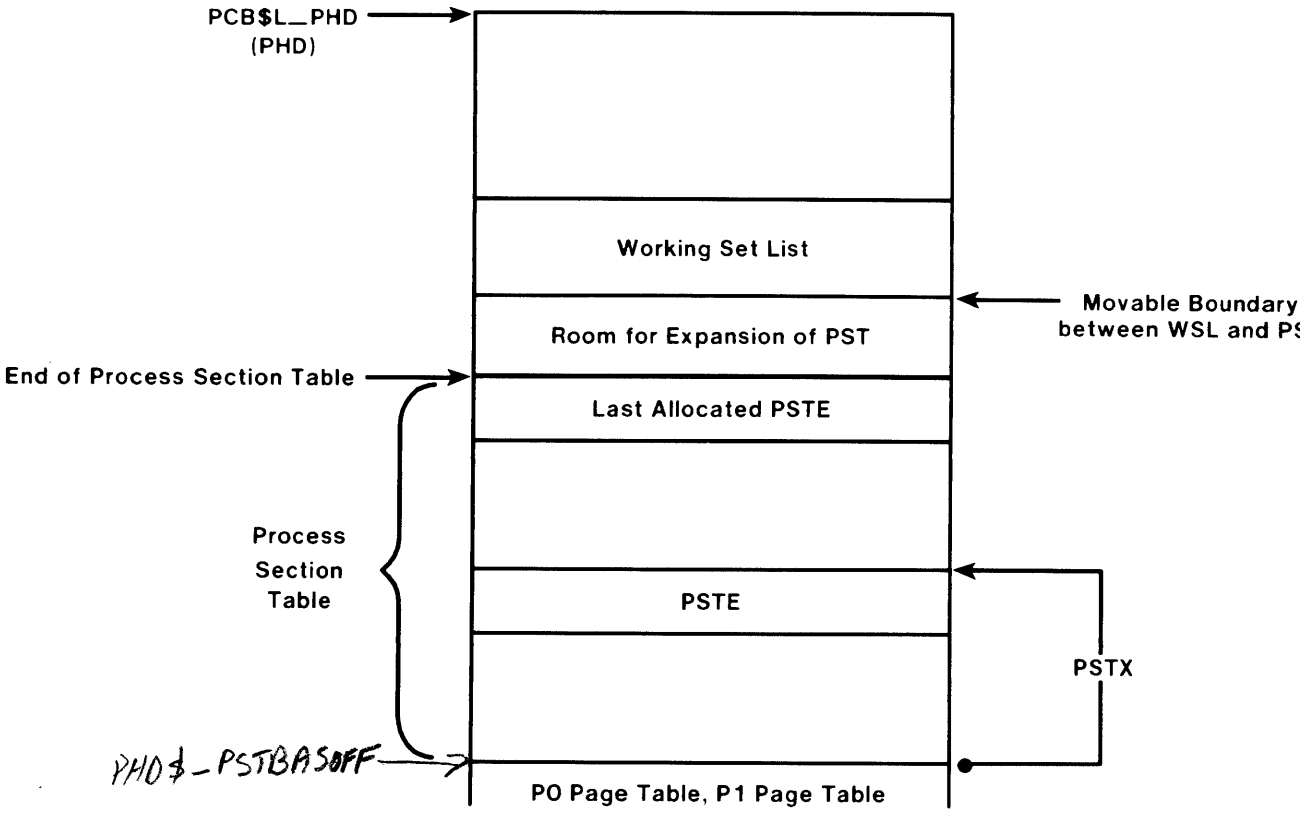


Figure 6-14 Process Section Table

The process section table:

- Contains entries that locate image sections on disk
- Grows toward lower offsets in the variable portion of the process header

*SYSGEN -

PROCSECTCNT

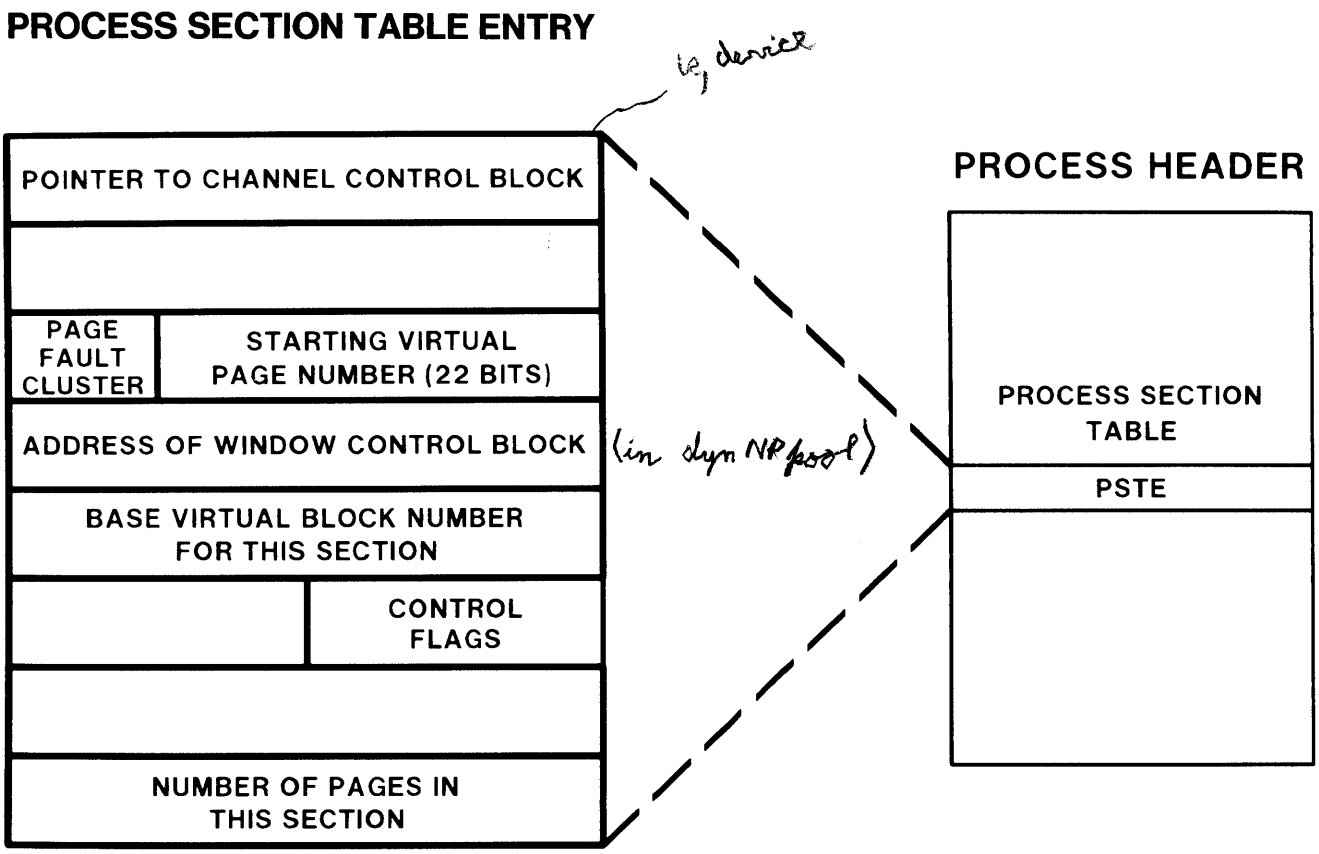


Figure 6-15 Process Section Table Entry

The process section table entry establishes the relationship between virtual pages in the address space and virtual blocks in an image file.

This information is constructed from information provided by the linker in the image section descriptor.

Control flags describe attributes of the section, such as:

Global
Copy on Reference
Demand zero
Writable

PAGE FILE CONTROL BLOCK

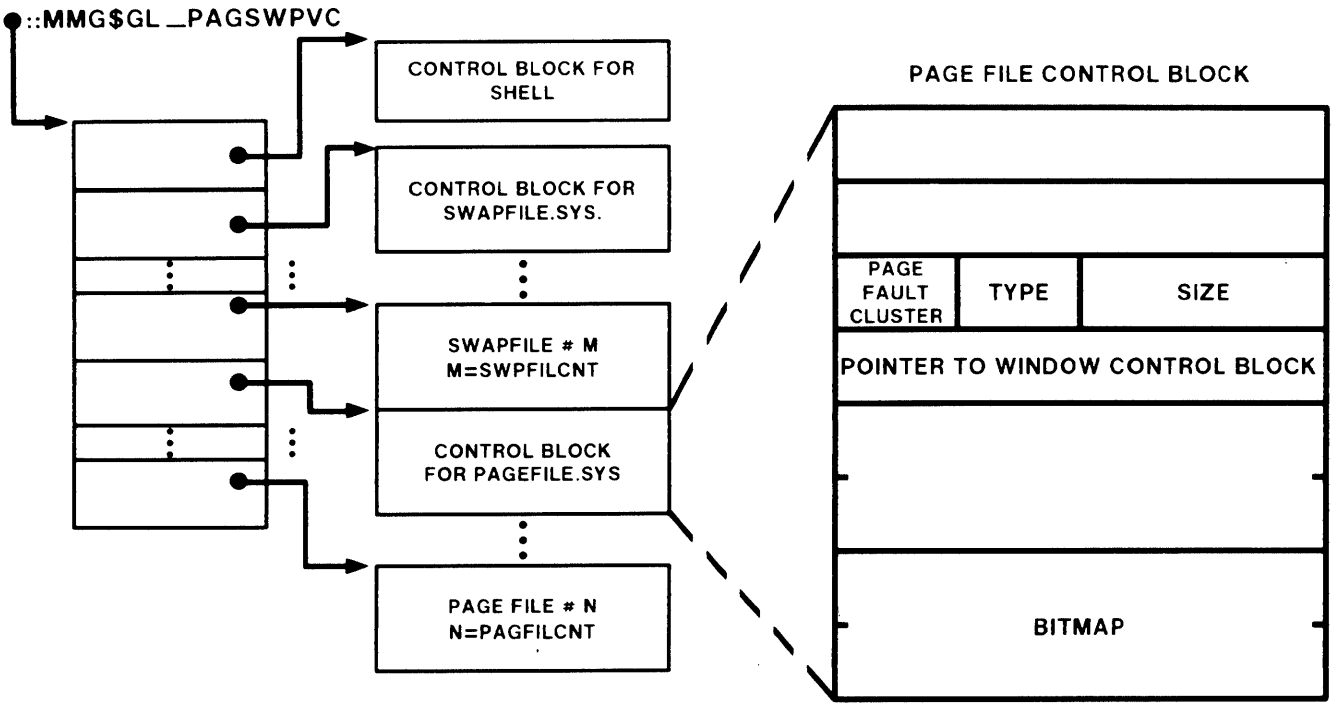


Figure 6-16 Page File Control Block

Control Block

- address of bitmap
- page fault cluster
- pointer to window control block
- base virtual block number
- pages which may be allocated or reserved

Bitmap

- one bit per block in the page file
- bit set implies block available

*SYSGEN -

SWPFILCNT

PAGFILCNT

DIFFERENT FORMS OF PAGE TABLE ENTRY

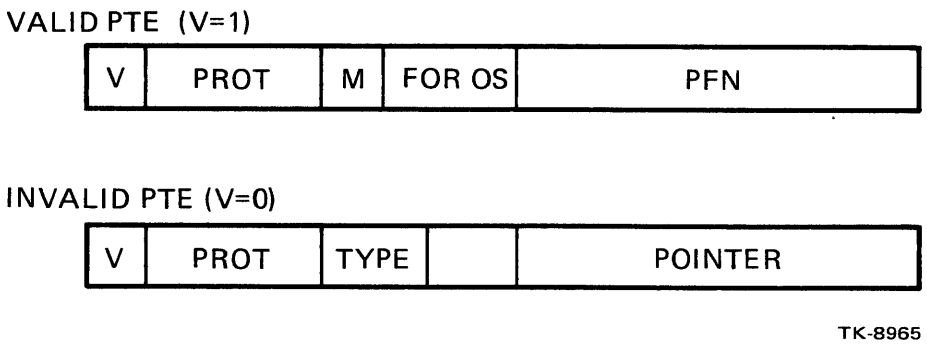


Figure 6-17 Different Forms of Page Table Entry

Page Table Entry Field PROT

Read/Write protection in Kernel, Executive, Supervisor, and User Access Modes.

Table 6-2 Fields Pager Uses to Determine Location of Page

Type	Pointer
Demand zero page	Ø
Page in transition	Page Frame Number (PFN)
Invalid global page	Global Section Table Index
Page in paging file	Paging File Virtual Block Number
Page in image file	Process Section Table Index

PROCESS PTEs MAP TO GLOBAL PTEs

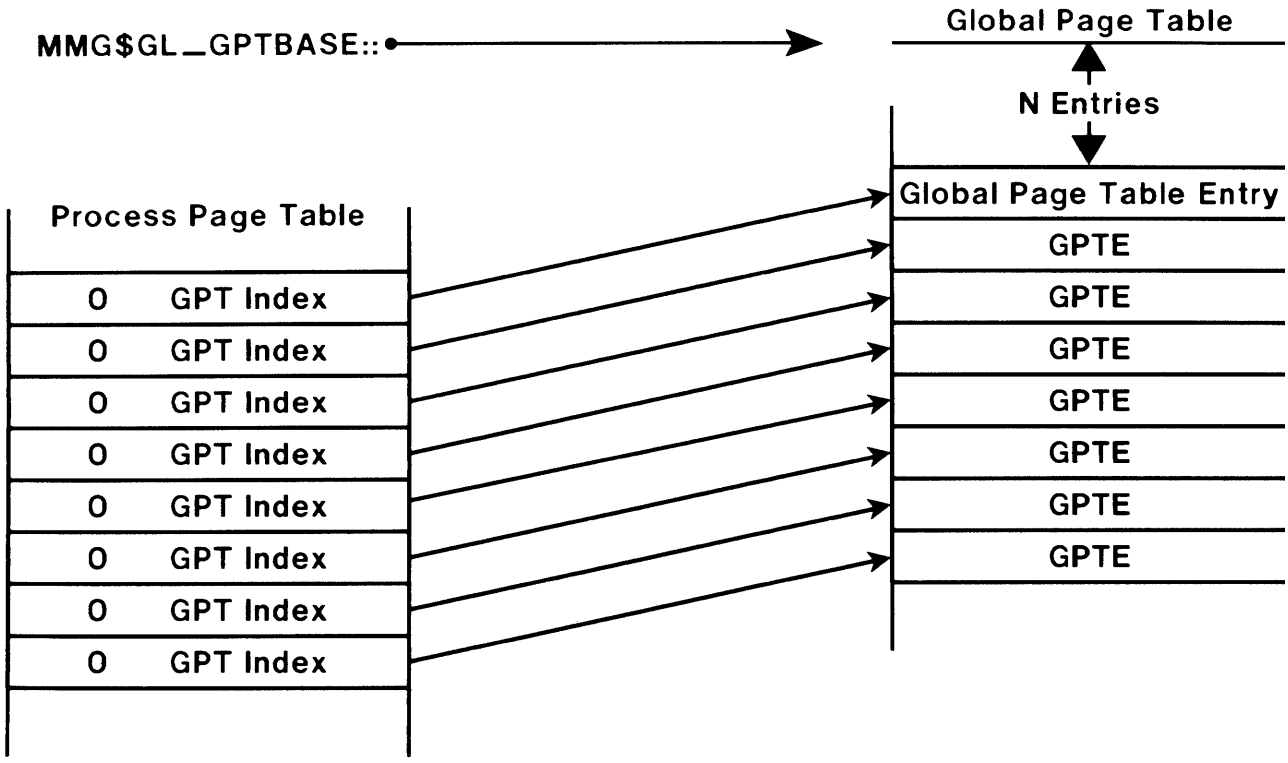


Figure 6-18 Process PTEs Map to Global PTEs

The page table entries associated with a global section table entry are in the global page table.

When processes map a global section, correspondence is established between process PTEs and global PTEs.

*SYSGEN -

GBLPAGES

GBLPAGFIL

RELATIONSHIP AMONG GLOBAL SECTION DATA STRUCTURES

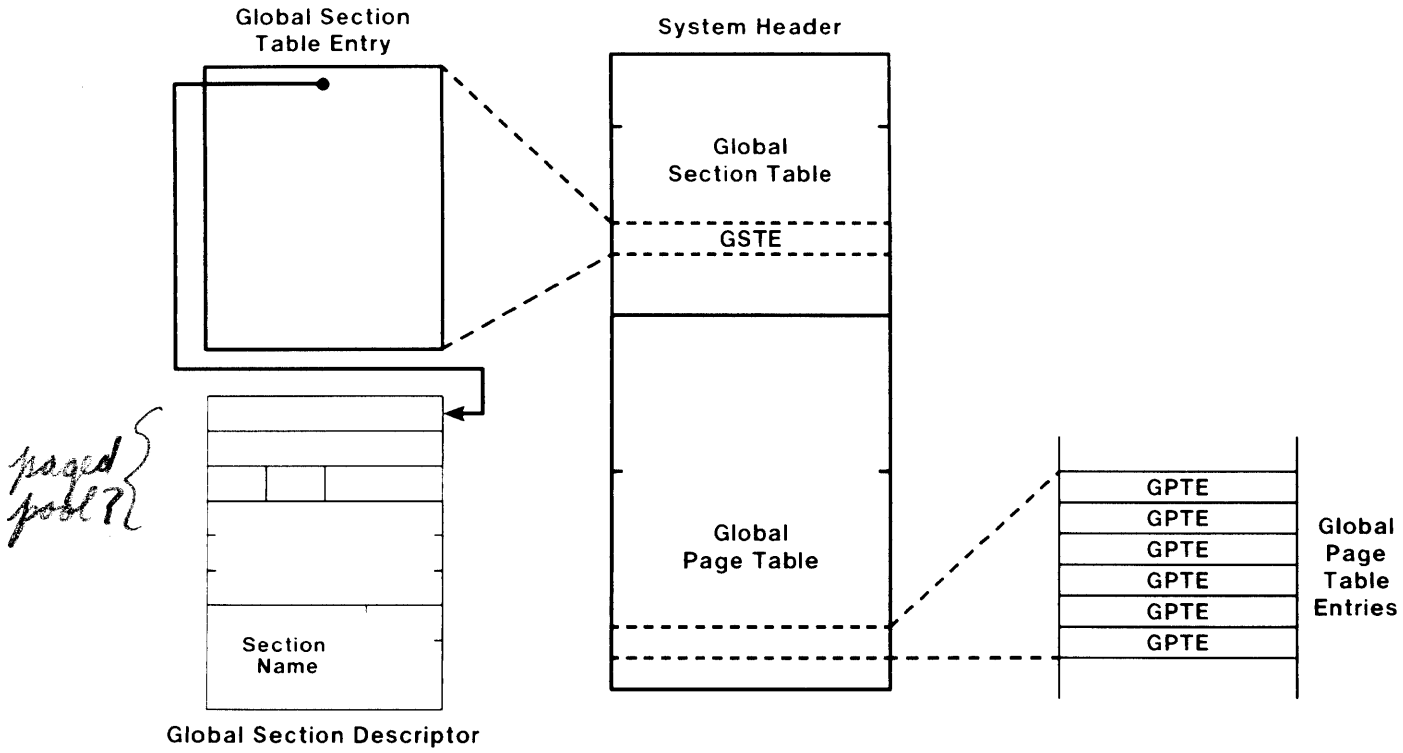


Figure 6-19 Relationship Among Global Section Data Structures

Three data structures contain global section information.

- global page table and
- global section table perform similar roles to those of the process page tables and process section table
- global section descriptors allow the location of global section information by name
- GSDs are placed in either a system queue or a group queue

*SYSGEN -

GBLSECTIONS

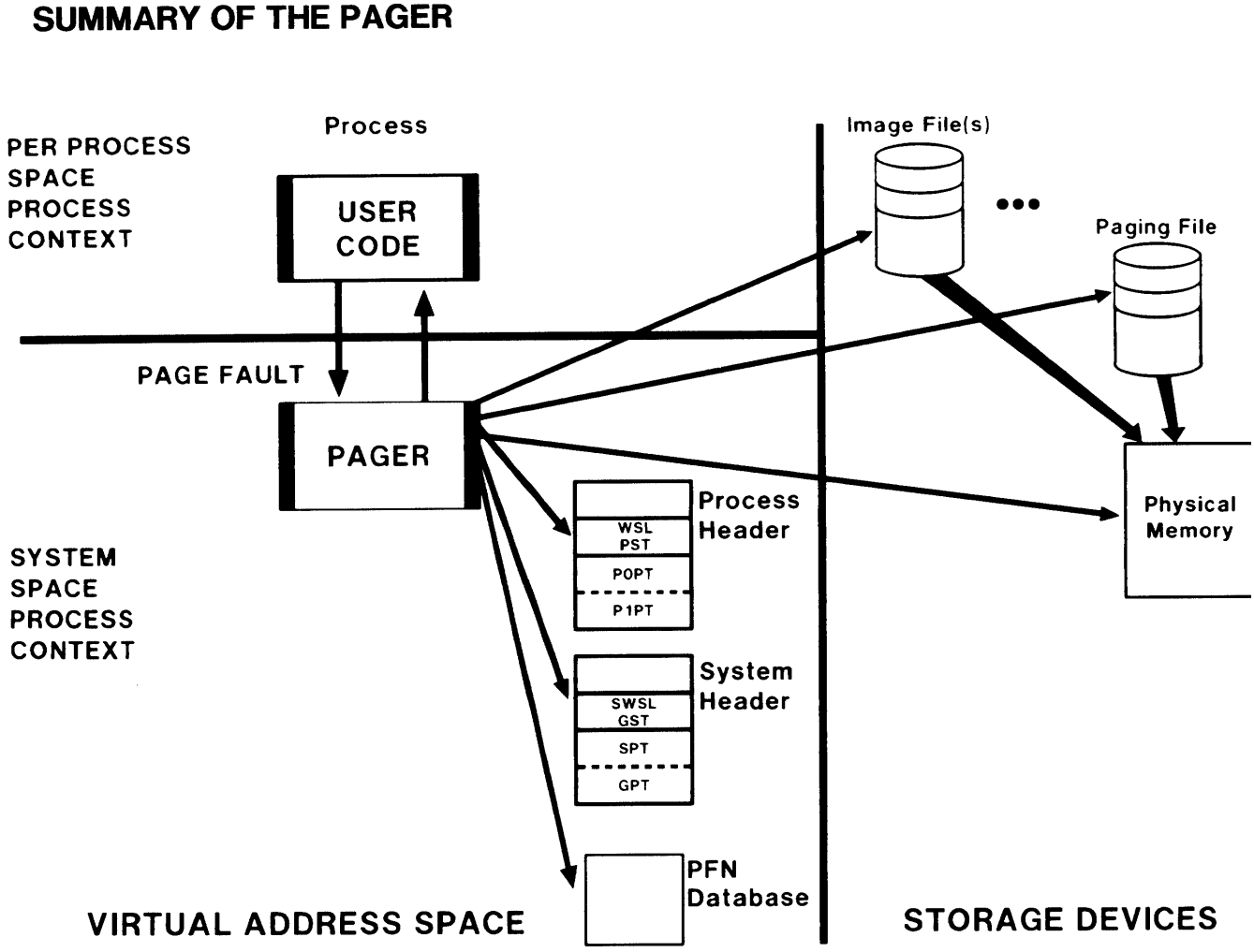


Figure 6-20 Summary of the Pager

INTRODUCTION – PAGING DYNAMICS EXAMPLES

- Process Read/Write Section Page
  Result of $CRMPSC for private section
- Process Copy-on-Reference Page
  For example, read/write data
- Global Read/Write Section Page
  Result of $CRMPSC for global section

INITIAL STATUS OF PROCESS READ/WRITE SECTION PAGE

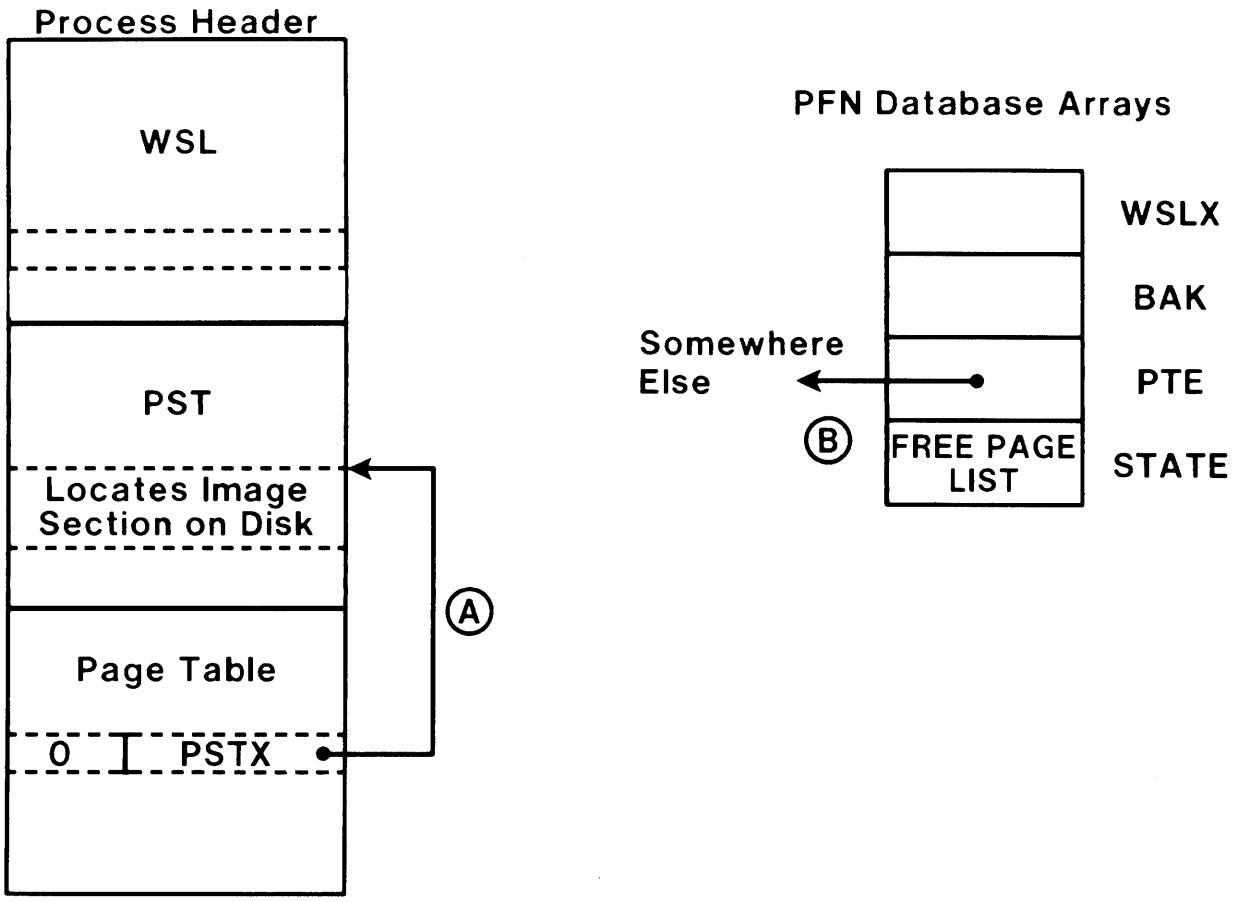


Figure 6-21 Initial Status of Process Read/Write Section Page

A. Page must be found through the process section table entry pointing to portion of an image file.

B. No connection between the process header structures and the PFN data base.

ADDING PROCESS READ/WRITE SECTION TO WORKING SET

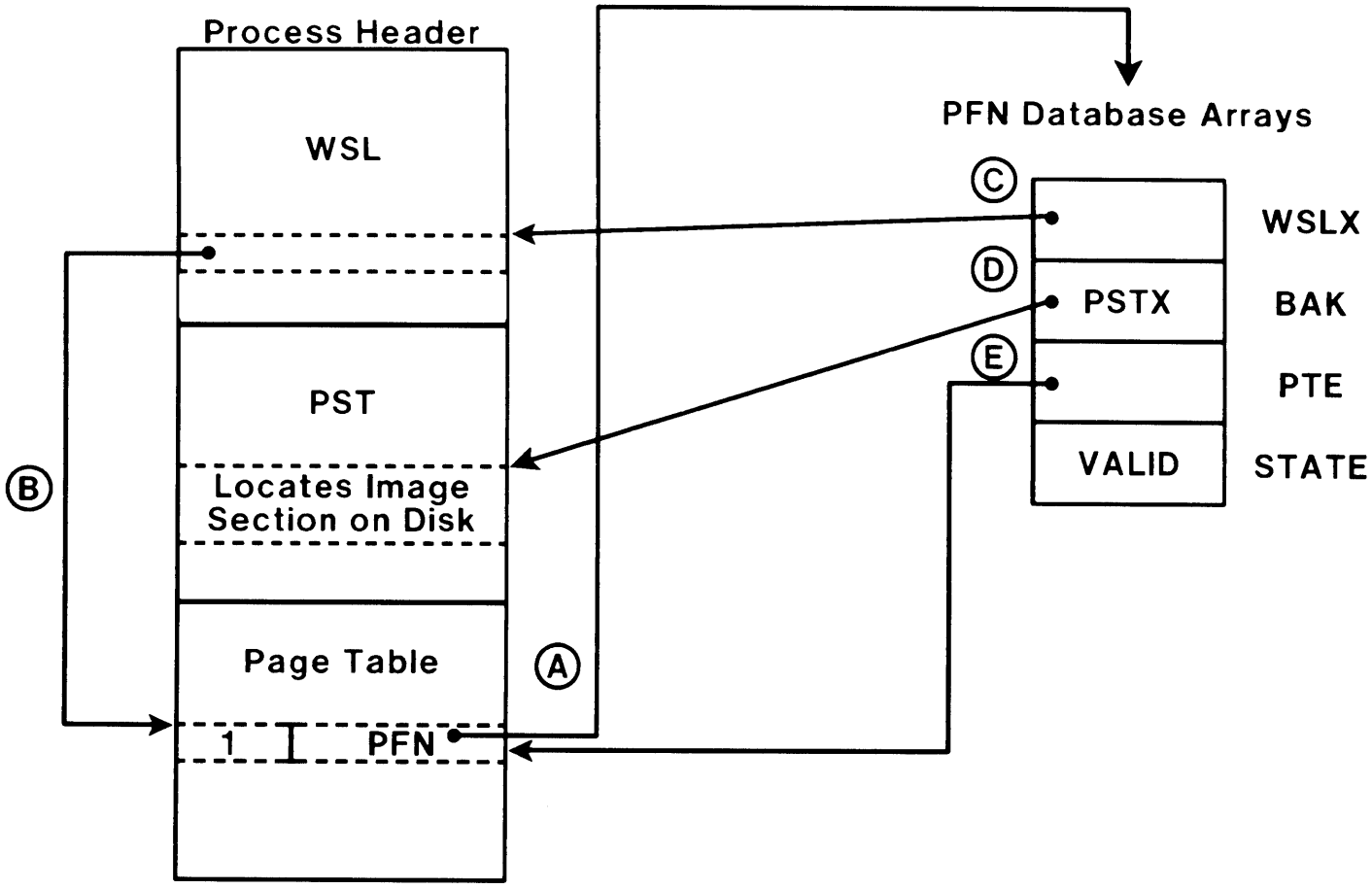


Figure 6-22 Adding Process Read/Write Section to Working Set

Page read complete

A. PTE contains PFN

B. Working set list entry points to PTE

C. Index to working set list entry

D. Index to process section table entry (was filled in from PTE)

E. Backward pointer to system virtual address of PTE

REMOVING MODIFIED PROCESS READ/WRITE SECTION PAGE FROM WORKING SET

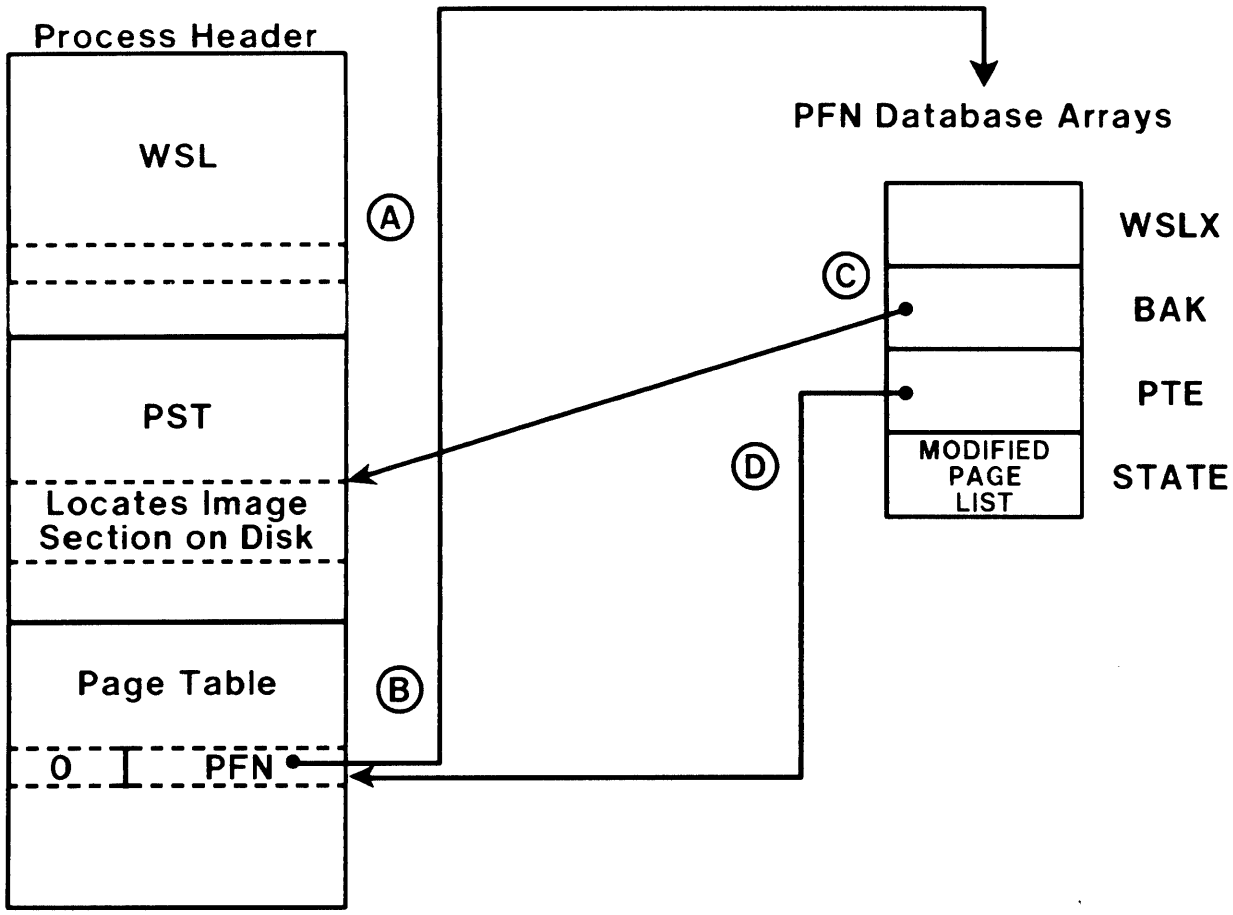


Figure 6-23 Removing Modified Process Read/Write Section Page from Working Set

A. Links to WSL have been broken

B. PFN still in PTE allows page to be faulted without disk I/O

C. Index to PSTE still on BAK - where to save now modified page

D. Backward pointer to PTE for modified page writer

MOVING PAGE FROM MODIFIED PAGE LIST TO FREE PAGE LIST

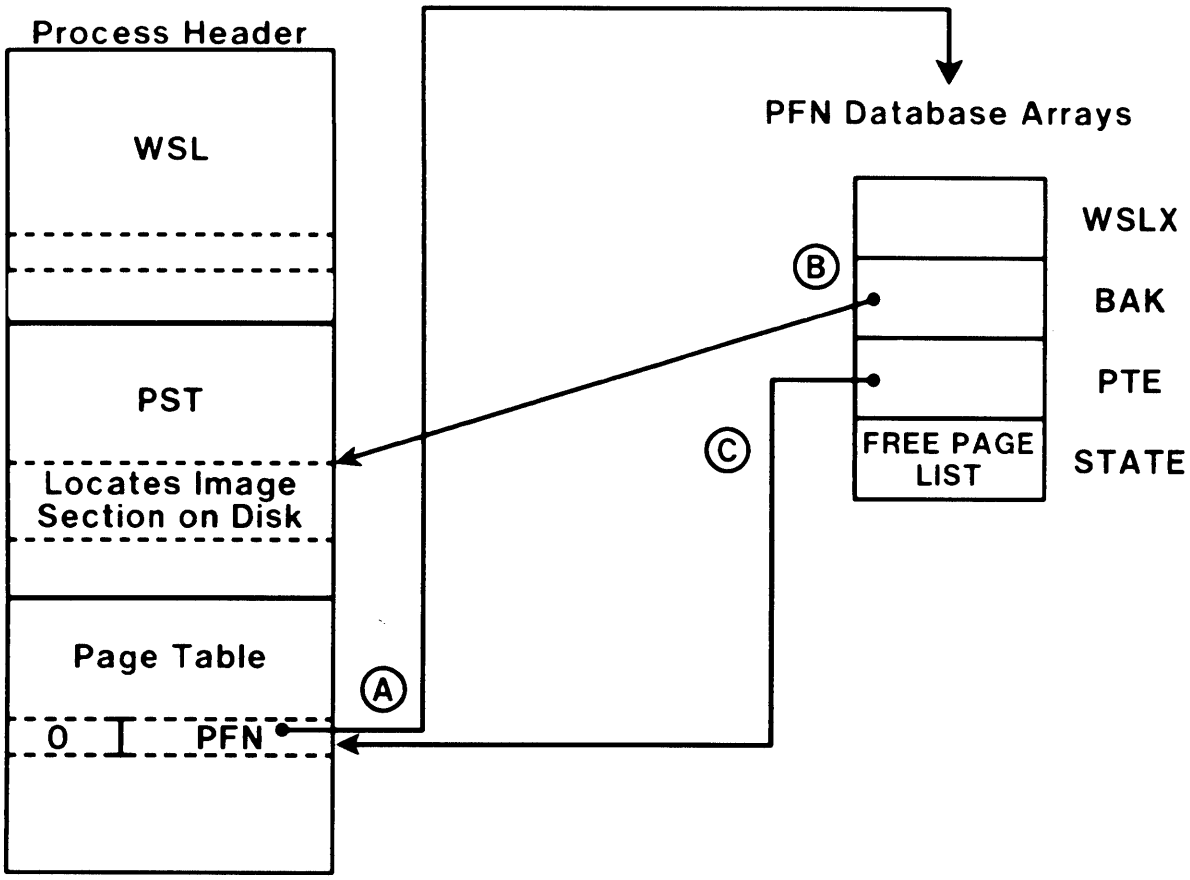


Figure 6-24 Moving Page from Modified Page List to Free Page List

(Page has been written back to image file on disk by modified page writer.)

A. PFN still in PTE allows page to be faulted without I/O

B. Index to PSTE still in BAK - must save for PTE

C. Backward pointer to PTE for pager

REMOVING PAGE FROM FREE PAGE LIST

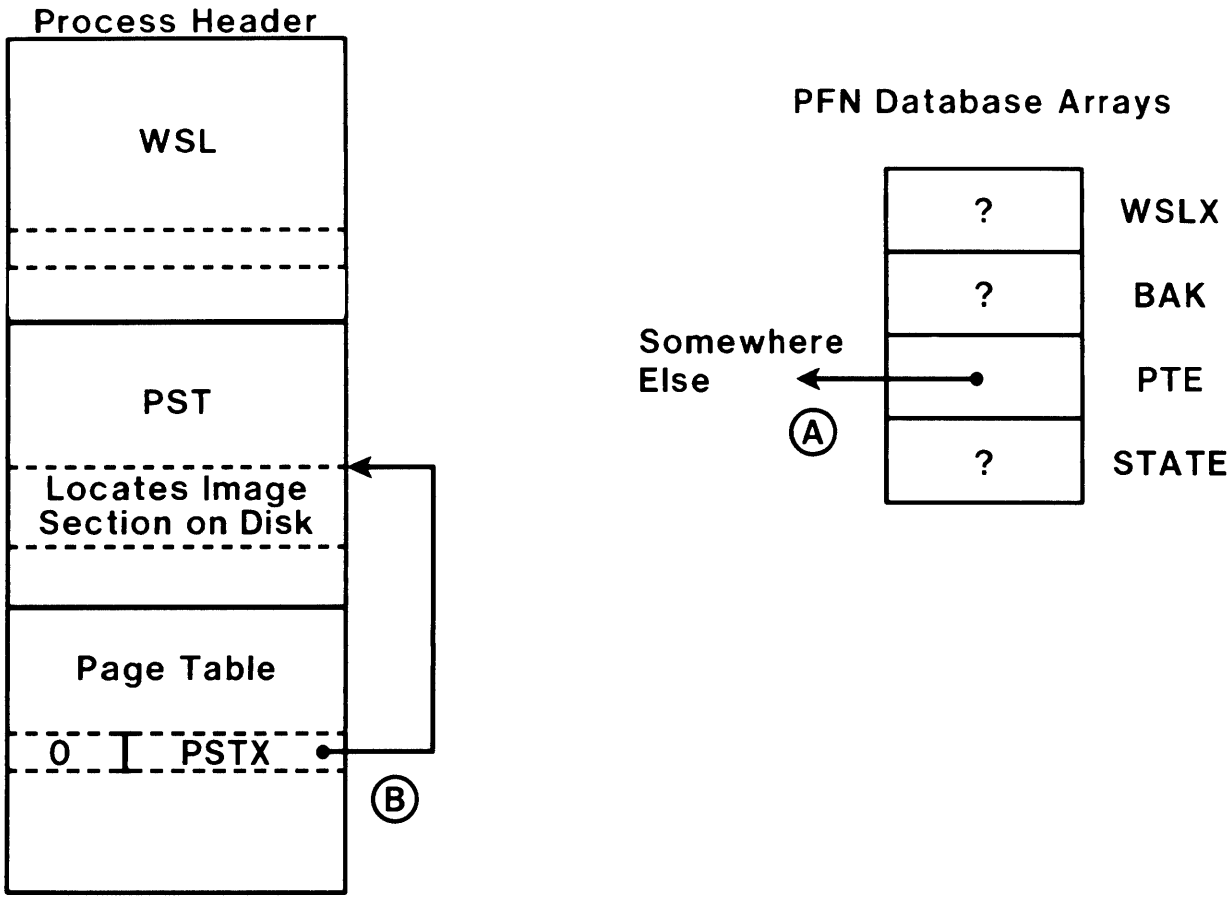


Figure 6-25 Removing Page from Free Page List

If page not faulted from free page list:

A. PFN data base entries point somewhere else

B. Process section table index has been put back into PTE

Data structures back to original states.

INITIAL STATUS OF PROCESS COPY-ON-REFERENCE PAGE

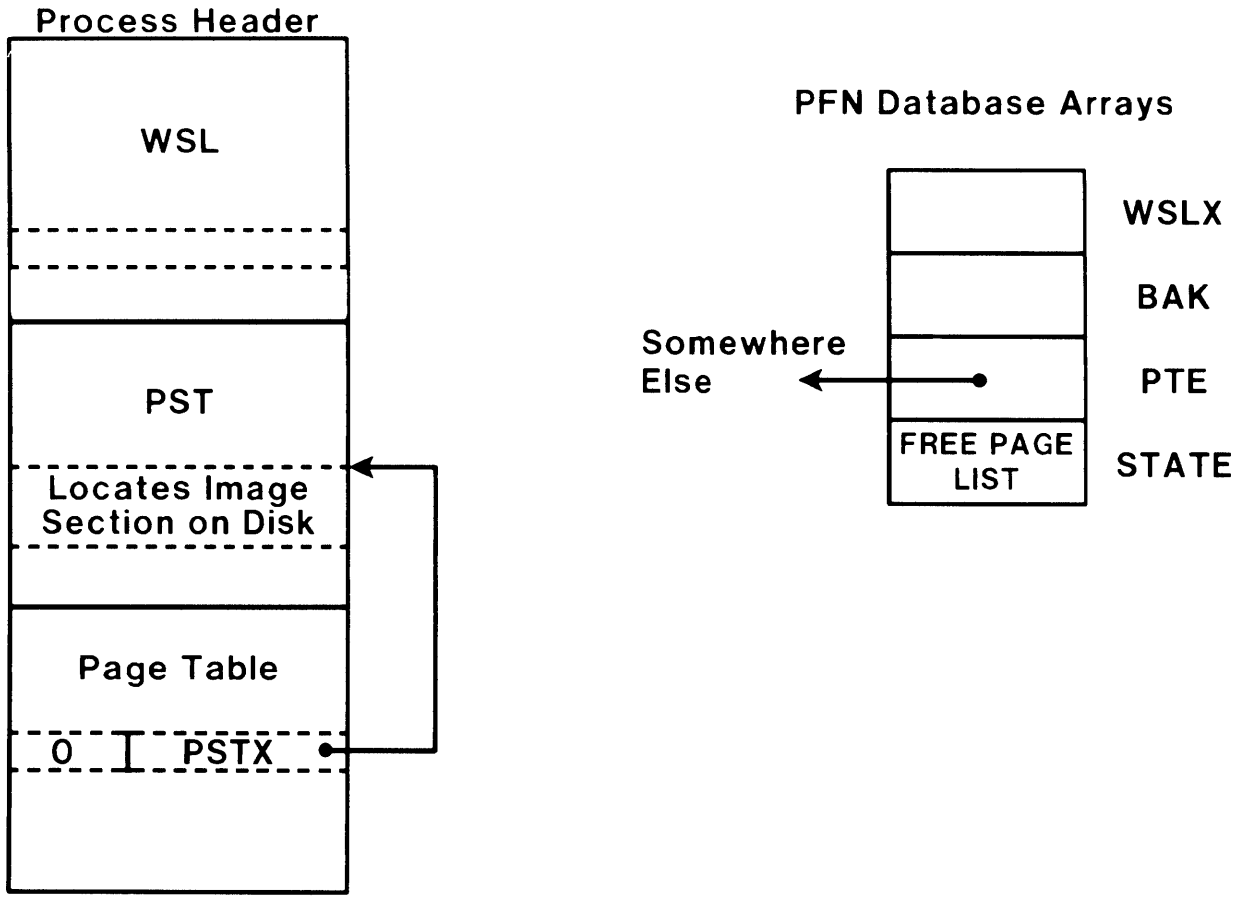


Figure 6-26 Initial Status of Process Copy-on-Reference Page

The initial state of this example is identical to that of a process read/write section page, except that copy-on-reference bits are set in the process section table entry and the PTE.

ADDING PROCESS COPY-ON-REFERENCE PAGE TO WORKING SET

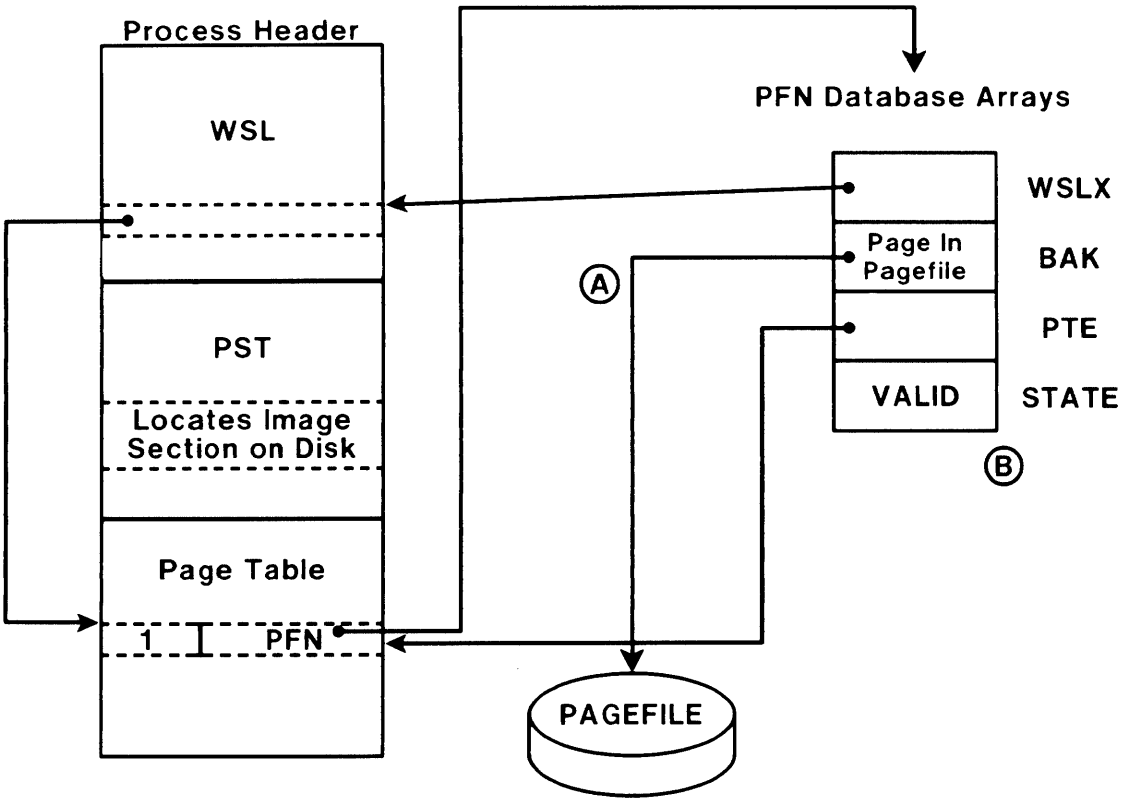


Figure 6-27 Adding Process Copy-on-Reference Page to Working Set

When the page is faulted into the working set, the same basic operations are performed as for a process read/write section page, except that

A. BAK indicates a reserved page in the page file

B. the modify bit in the STATE array is automatically set

This page is never associated with the original image file again.

REMOVING PROCESS COPY-ON-REFERENCE SECTION PAGE FROM WORKING SET

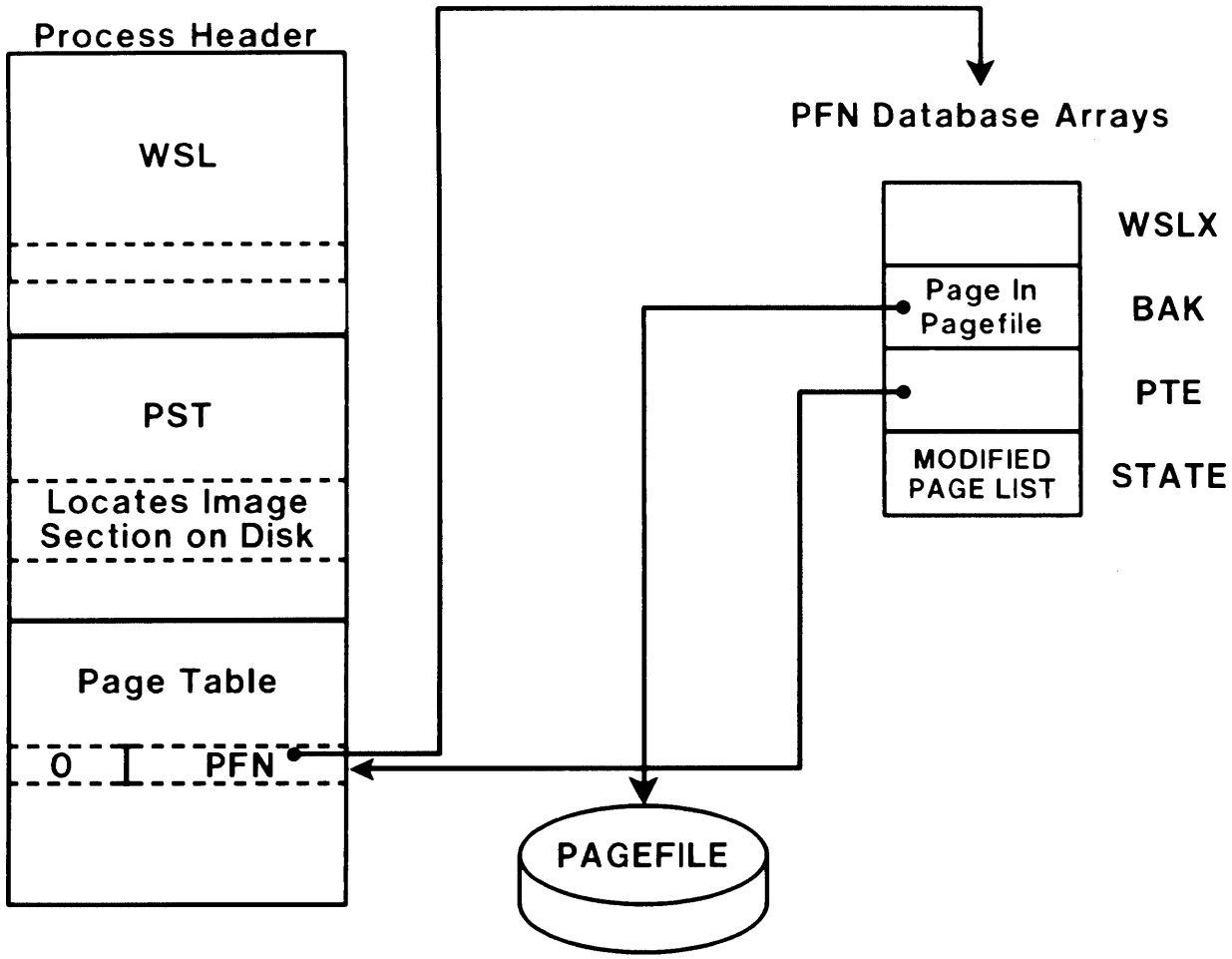


Figure 6-28 Removing Process Copy-on-Reference Section Page from Working Set

When this page is removed from the working set, it is placed in the modified list because the modify bit had been set in the previous step.

REMOVING PROCESS COPY-ON-REFERENCE PAGE FROM MODIFIED PAGE LIST

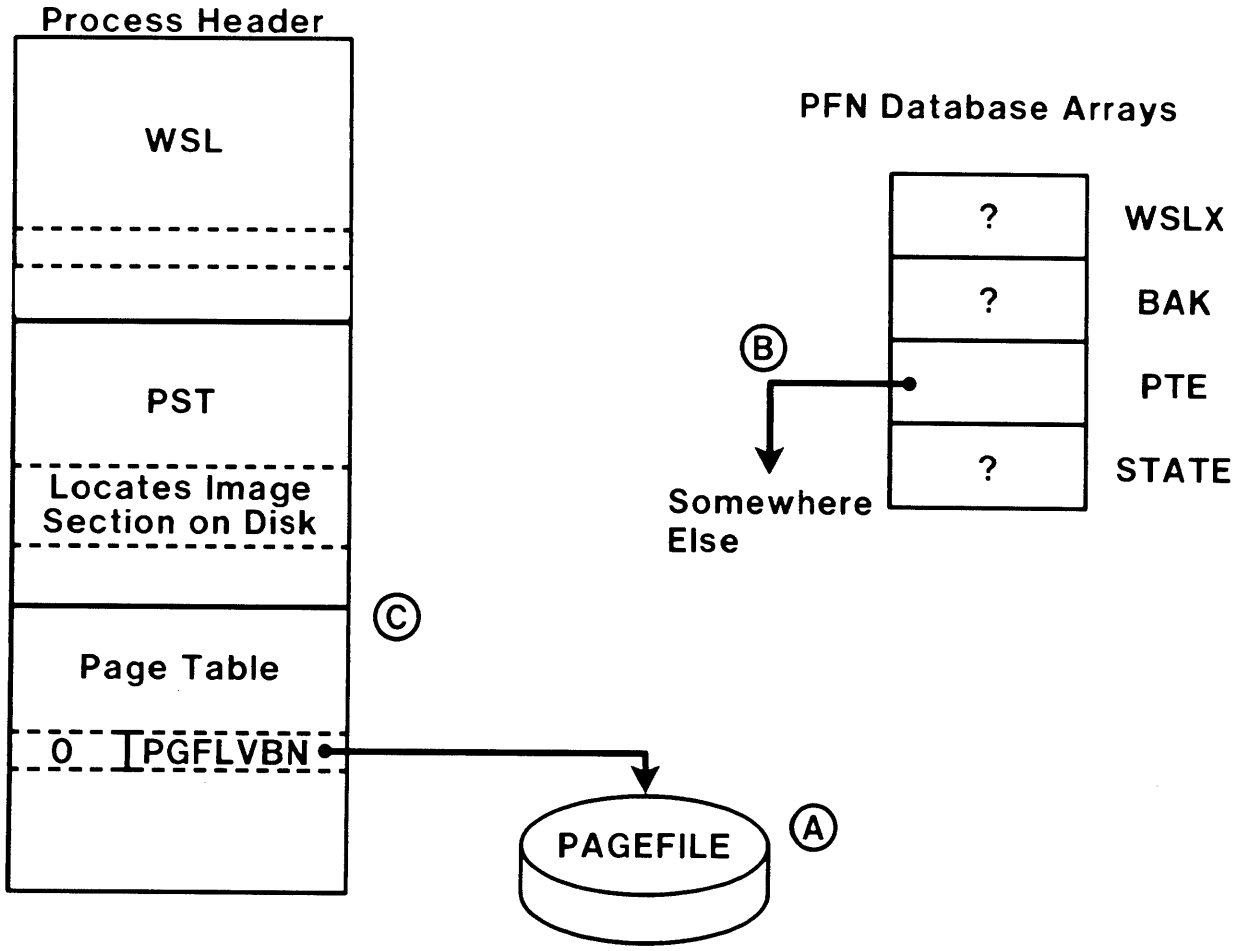


Figure 6-29 Removing Process Copy-on-Reference Page from Modified Page List

A. When this page filters through the modified list, the contents of the page are written to the page file.

B. When the page is allocated to another process from the head of the free list, all links between the process header and the PFN data base are broken.

C. The PTE points to the location of the page in the page file, from which it can later be faulted.

INITIAL STATUS OF GLOBAL READ/WRITE SECTION PAGE

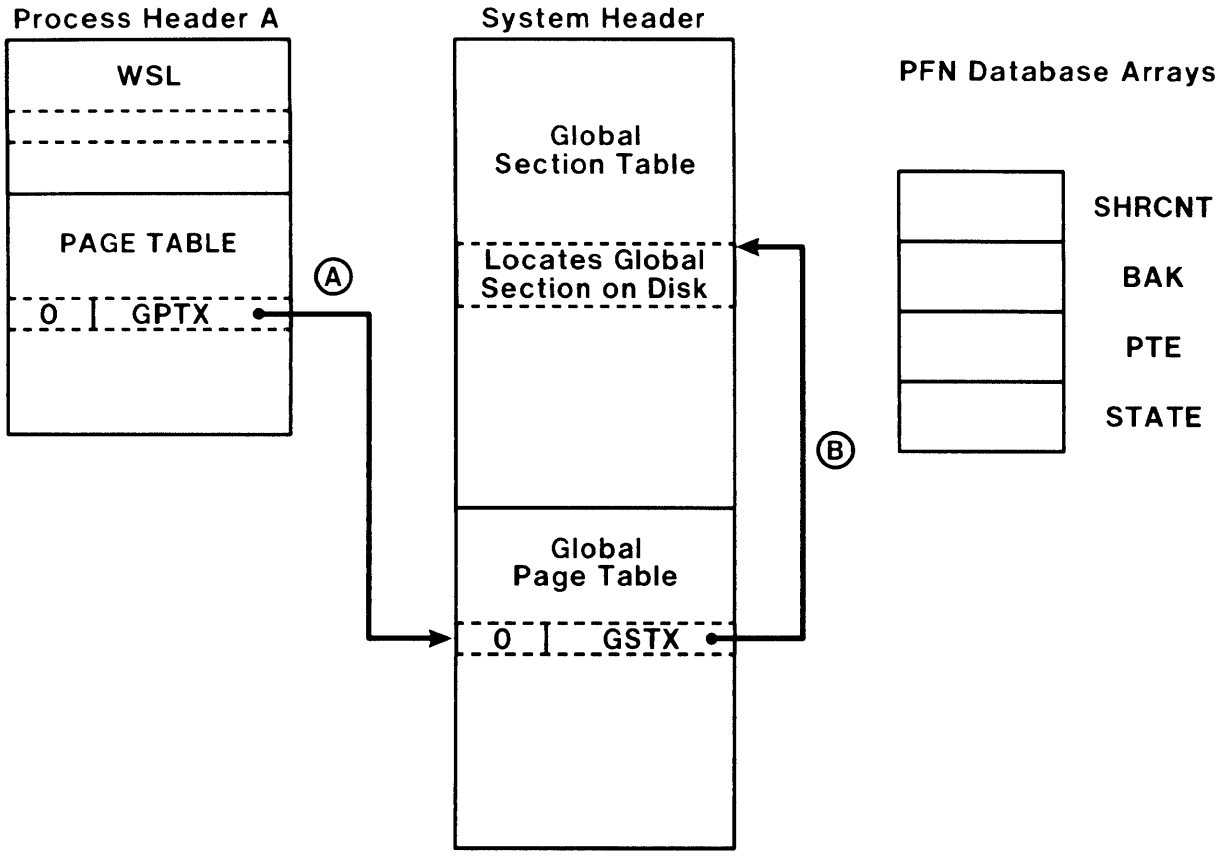


Figure 6-30 Initial Status of Global Read/Write Section Page

A. When Process A maps the global section, the PTEs point to the corresponding global page table entries for the section.

B. Each of the GPTEs points to the global section table entry which, in turn, points to the section file.

ADDING GLOBAL READ/WRITE SECTION PAGE TO WORKING SET

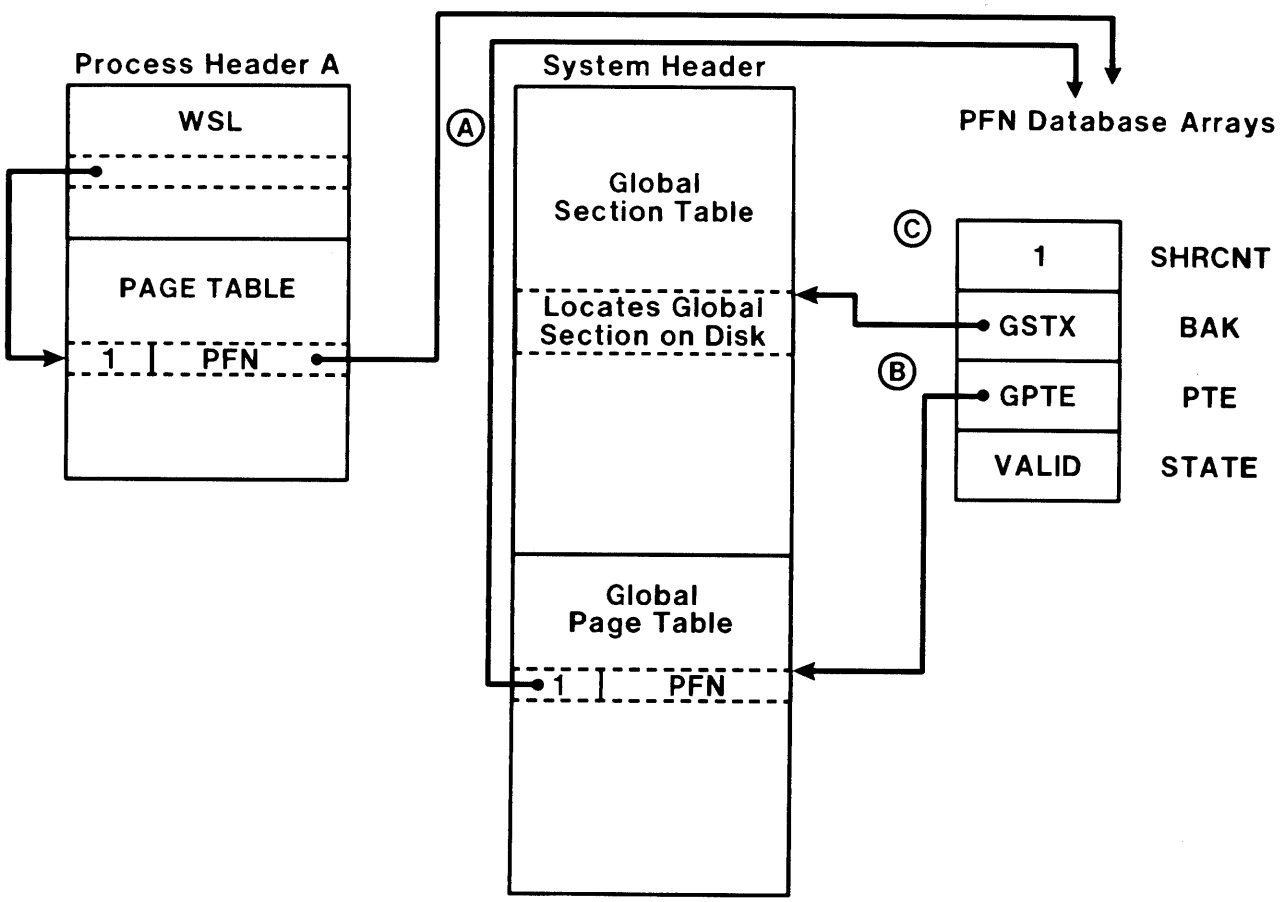


Figure 6-31 Adding Global Read/~~Write~~-Section Page to Working Set

When Process A faults the global page

A. both the process PTE and the GPTE contain the page frame number

B. the PFN data base points only to the system header data structures (GSTE and GPTE)

C. the SHRCNT is initialized to 1

INITIAL STATUS OF PTE OF SECOND PROCESS MAPPING THE SAME GLOBAL SECTION

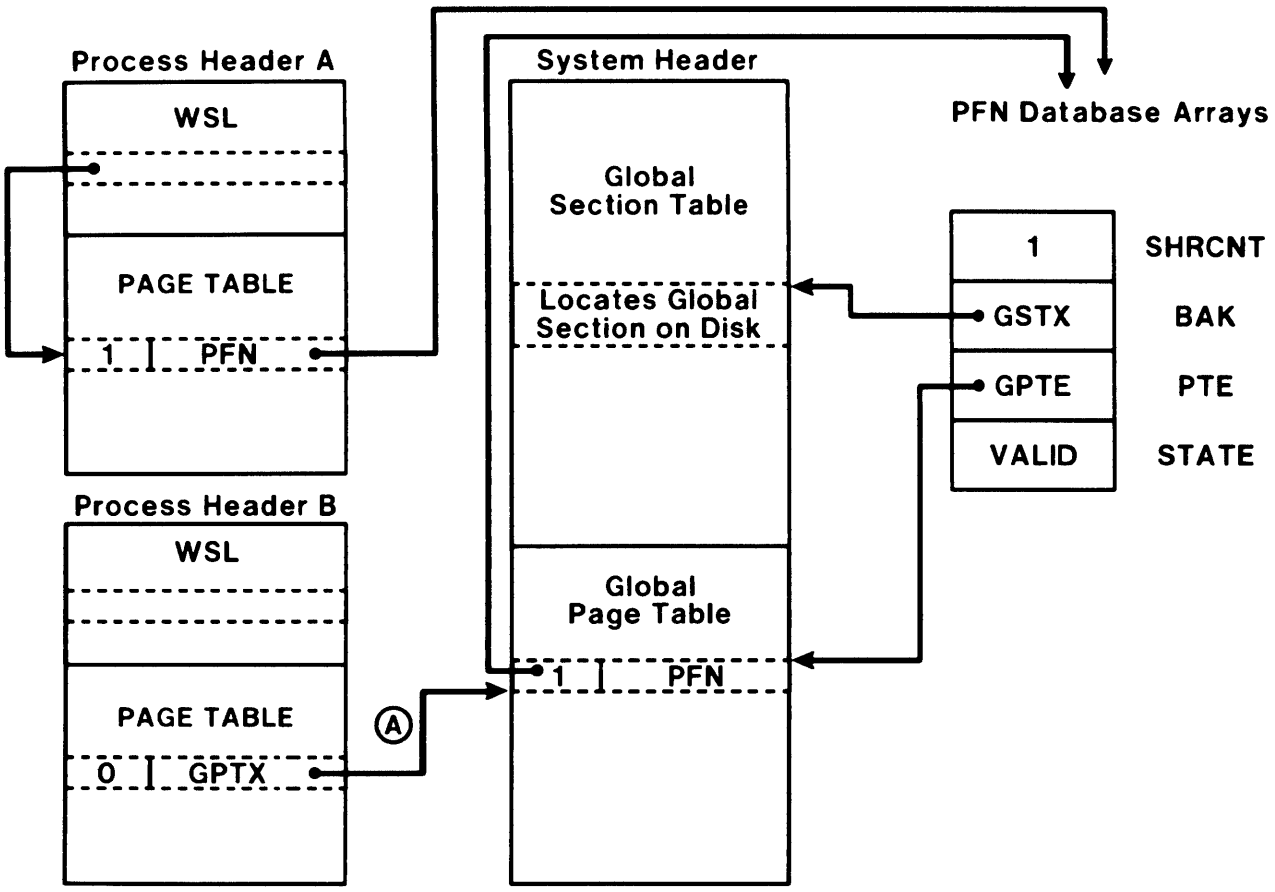


Figure 6-32 Initial Status of PTE of Second Process Mapping the Same Global Section

A. When Process B maps the same global section, its PTE contains the GPTX.

ADDING GLOBAL READ/WRITE SECTION PAGE TO SECOND WORKING SET

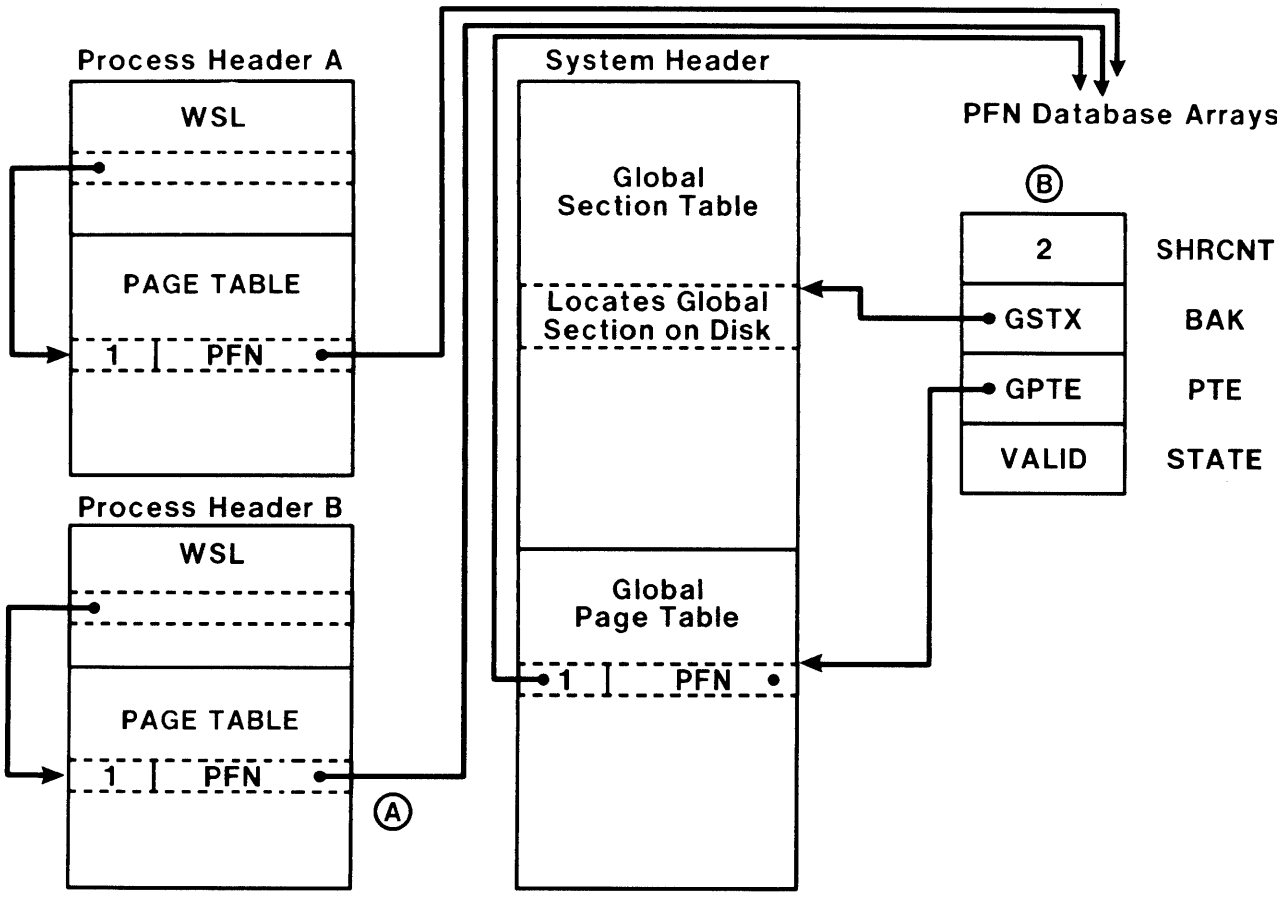


Figure 6-33 Adding Global Read/~~Write~~ Section Page to Second Working Set

A. When Process B faults the same global page as Process A, the PTE of Process B also points to the page frame.

B. The only change in the system data structures is the incrementing of the SHRCNT value to two.

REMOVING GLOBAL READ/WRITE SECTION PAGE FROM WORKING SET

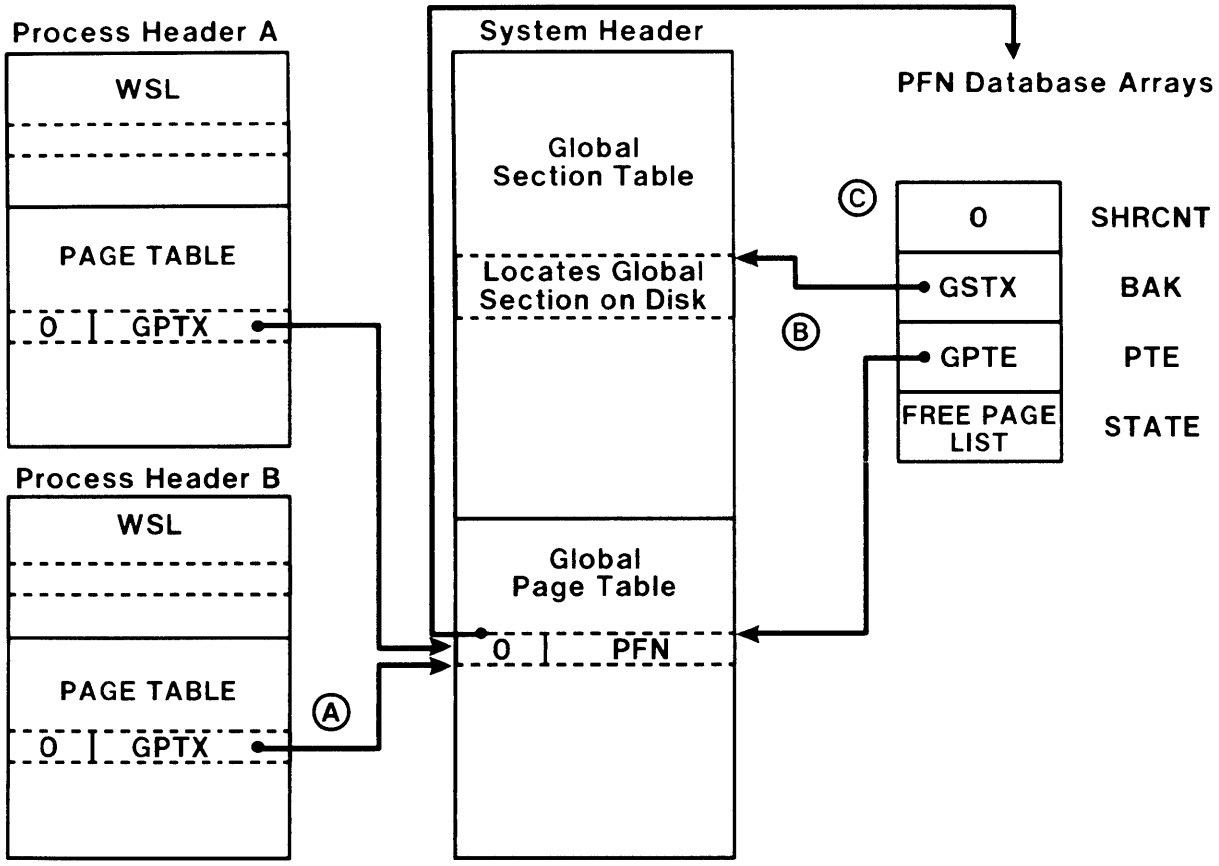


Figure 6-34 Removing Global Read/~~Write~~ Section Page from Working Set

Eventually both processes release the global pages from their working sets.

A. As each process loses page from working set, the PFN in the process PTE is overwritten by GPTX.

B. The relationships between the system header data structures and the PFN data base are similar to those for a process private page on the free list.

C. The global page is placed on the page lists only after SHRCNT is decremented to zero.

REMOVING GLOBAL READ/WRITE SECTION PAGE FROM LIST

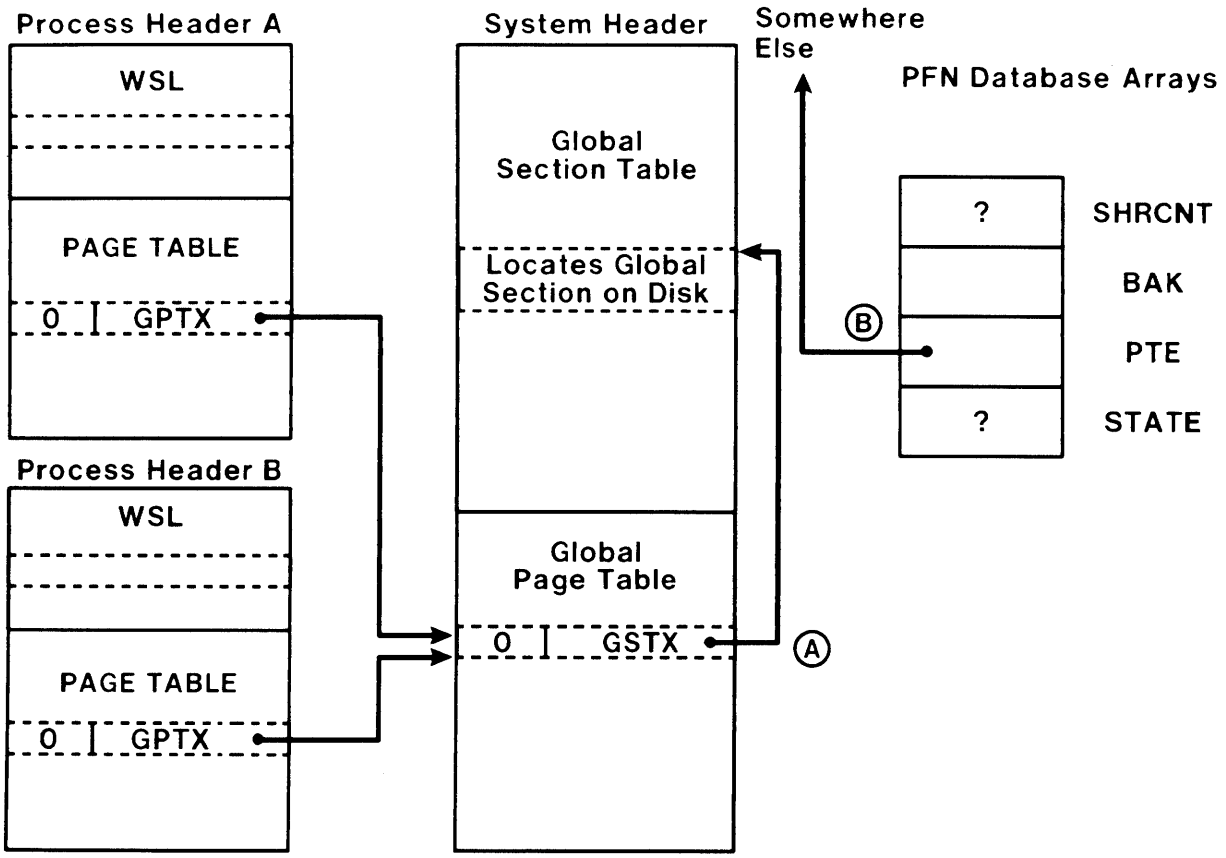


Figure 6-35 Removing Global Read/Write Section Page from List

When the page is allocated to another process from the head of the free list

A. The system header data structures are returned to their initial states.

B. All links to the PFN data base are destroyed.

APPENDIX

PAGE READ CLUSTERING

Why Cluster Pages

- More efficient QIO.
- Makes image activation easier.
- Bring into working set pages that potentially will be referenced.

How a Cluster Is Made

Pager scans successive PTE's for the same backing store address.

Examples:

- PSTX
- Consecutive pagefile VBN's
- Consecutive GPTE's with same GSTX

Pager scans until:

- No more WSLE's are available
- No physical pages available
- Page table page for PTE not valid
- Maximum cluster size reached

If no page can be clustered, previous PTE's are scanned using above rules.

Maximum Cluster Size Determination

Table 6-3 Cluster Sizes and Where They Are Stored

Page	Cluster Size
Global Page Tables	1
Process Page Tables	PAGTBLPFC
Paging File Pages	PFL$B_PFC/PFCDEFAULT
Process, Global Sections	SEC$B_PFC/PFCDEFAULT

Changing/Controlling Cluster Size

- SYSGEN parameters
  - PFCDEFAULT
  - PAGTBLPFC
- PFC argument in $CREPRC
- Linker option
  (cluster=clustername,[base_adr],[pfc]file spec[,...]

PROGRAM SECTIONS (.PSECTs)

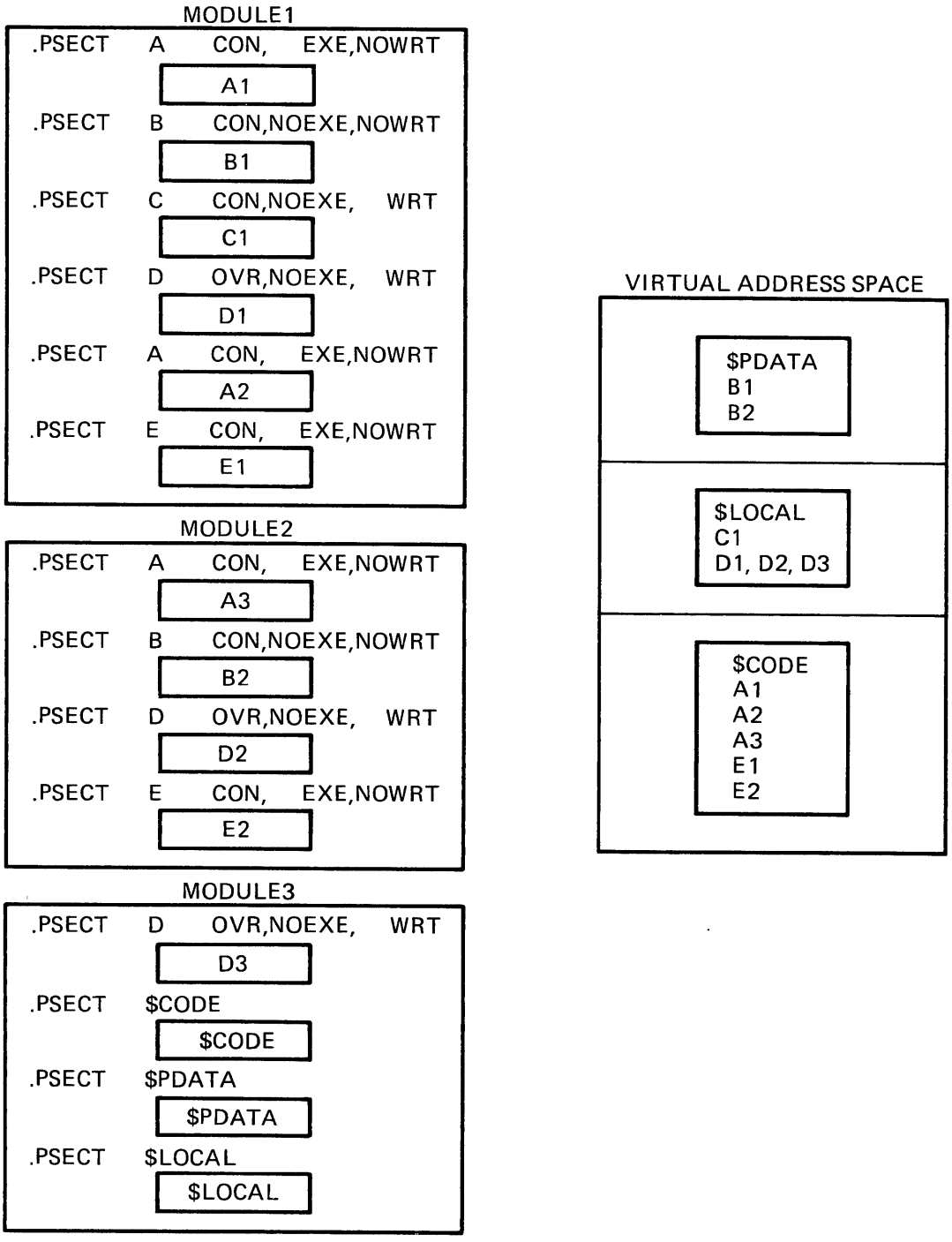


Figure 6-36 Program Sections (.PSECTs)

LINKER CLUSTERS

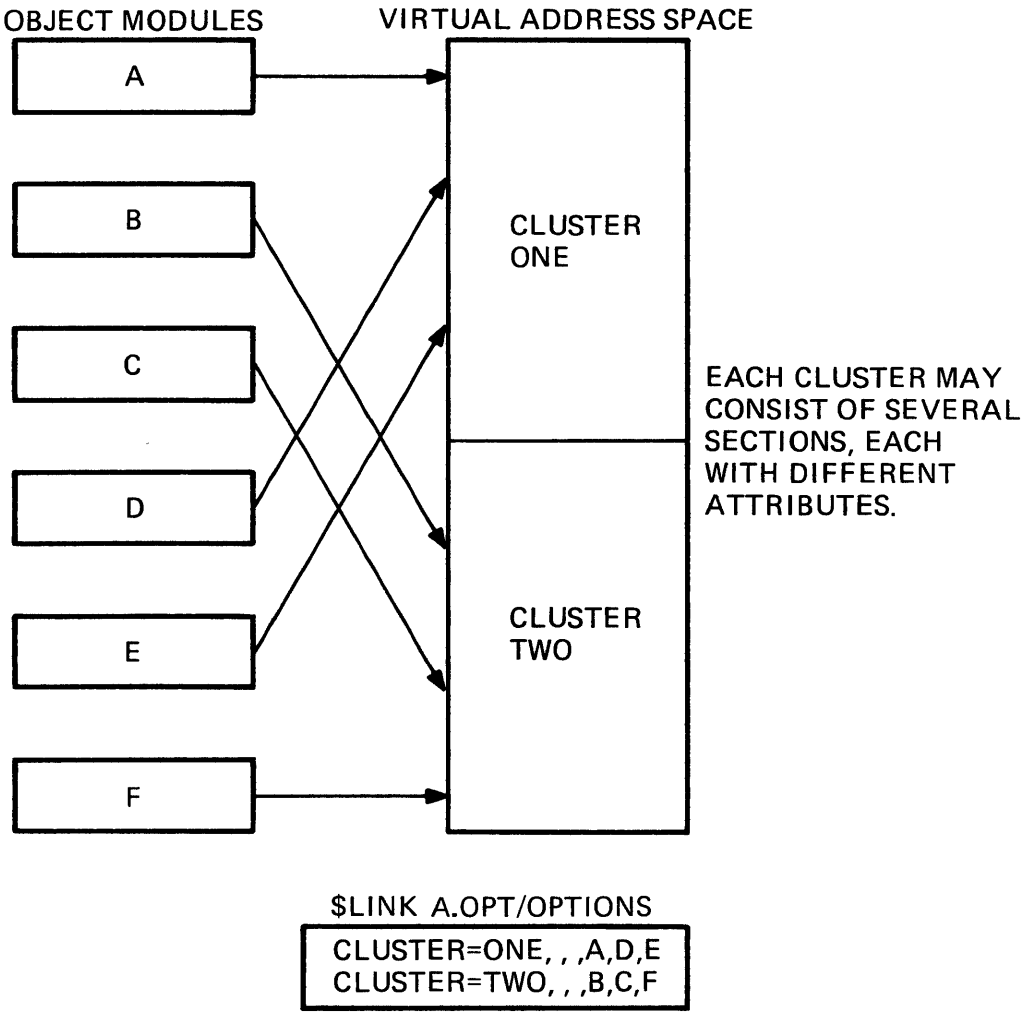


Figure 6-37 Linker Clusters

PROGRAM SECTION ATTRIBUTES GBL/LCL

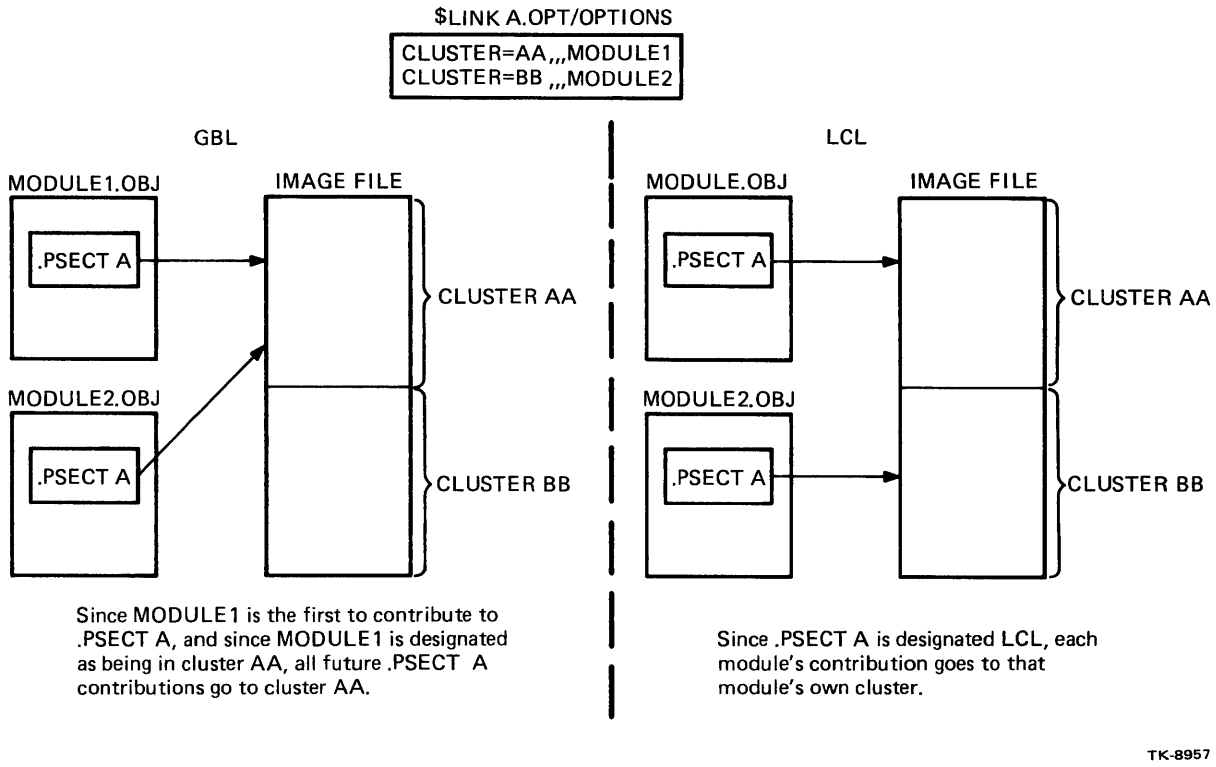


Figure 6-38 Program Section Attributes GBL/LCL

HARDWARE CHECKS

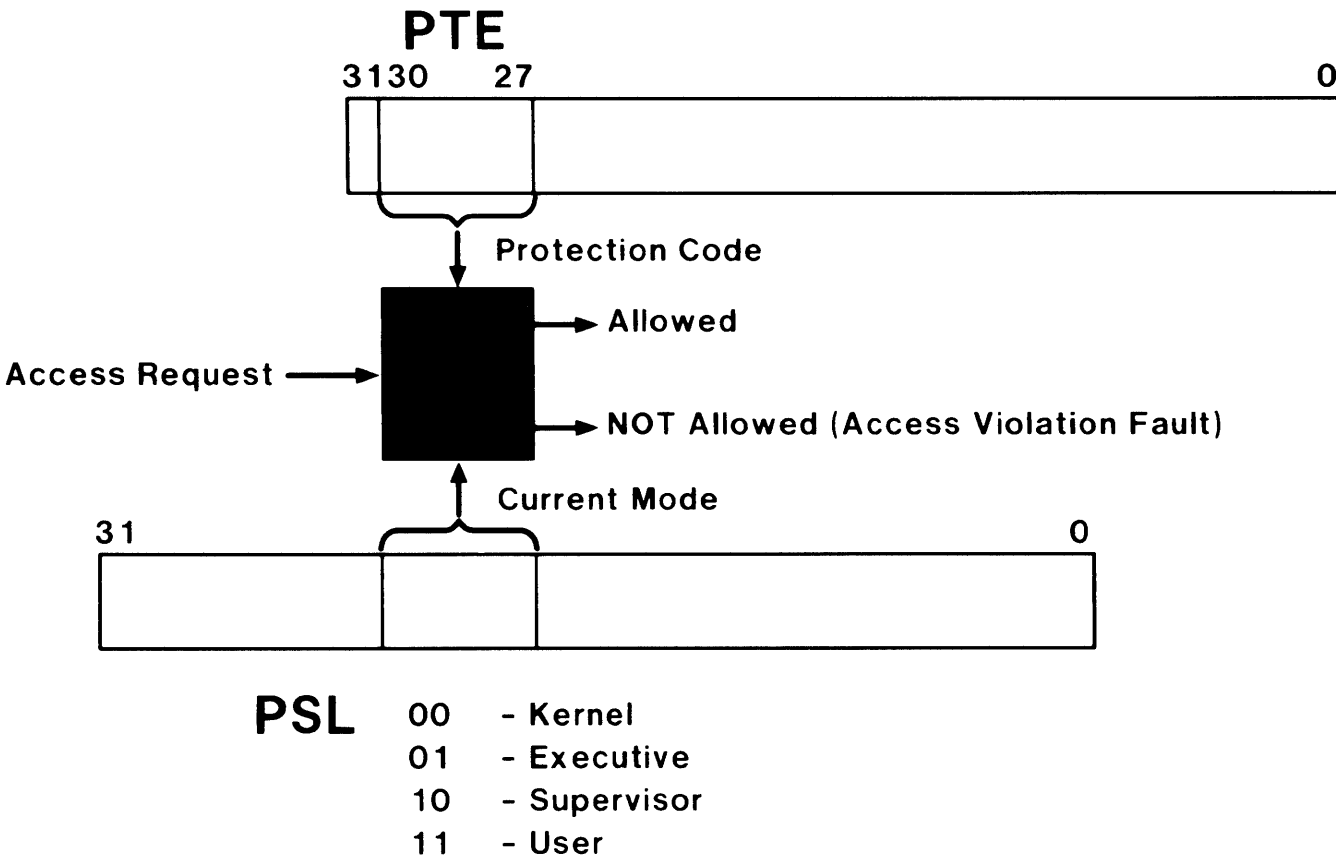


Figure 6-39 Hardware Checks

Before address translation occurs, the hardware checks the type of request (read or modify/write) against

- The protection field of the corresponding page table entry (PTE).
- The current access mode field of the processor status longword (PSL)

If access is denied, no address translation occurs and an access violation condition is signaled.

VIRTUAL ADDRESS SPACE

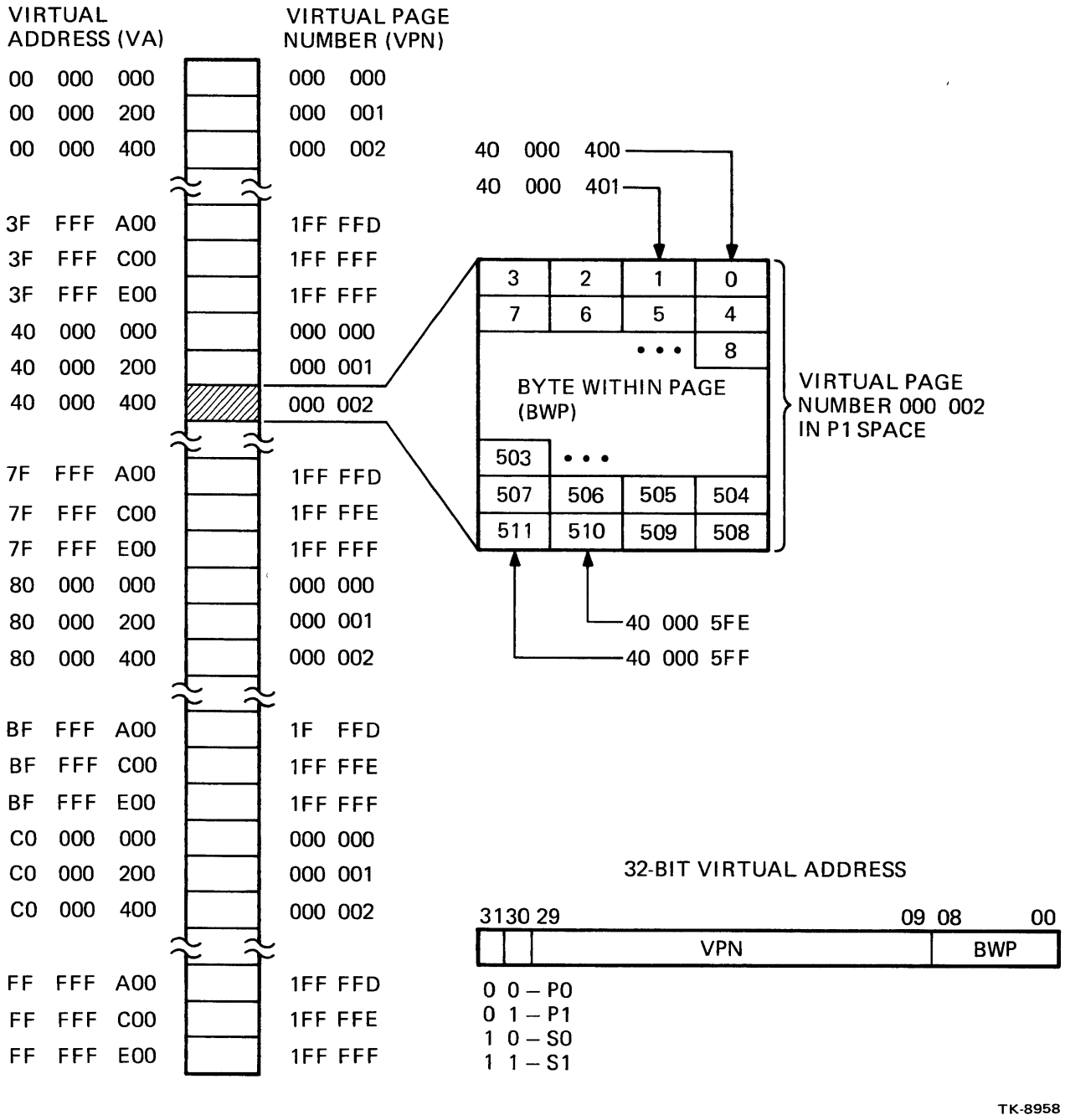


Figure 6-40 Virtual Address Space

PAGE TABLE MAPPING

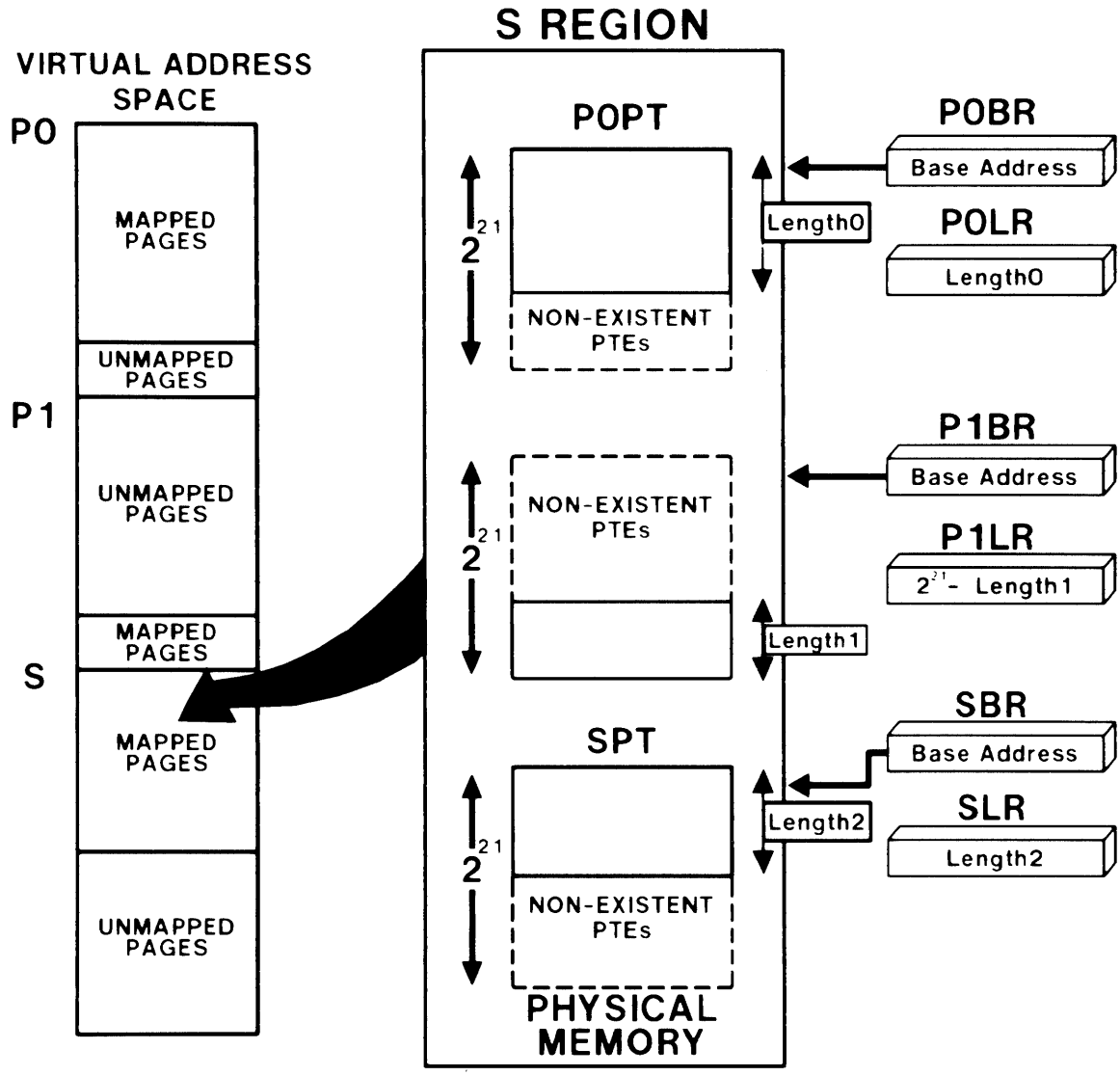


Figure 6-41 Page Table Mapping

All page tables are mapped in system space. Each page table is located through the corresponding processor base register, and its length is indicated by the corresponding processor length register. The system page table is permanently resident in memory and located by physical address.

SYSTEM SPACE ADDRESS TRANSLATION

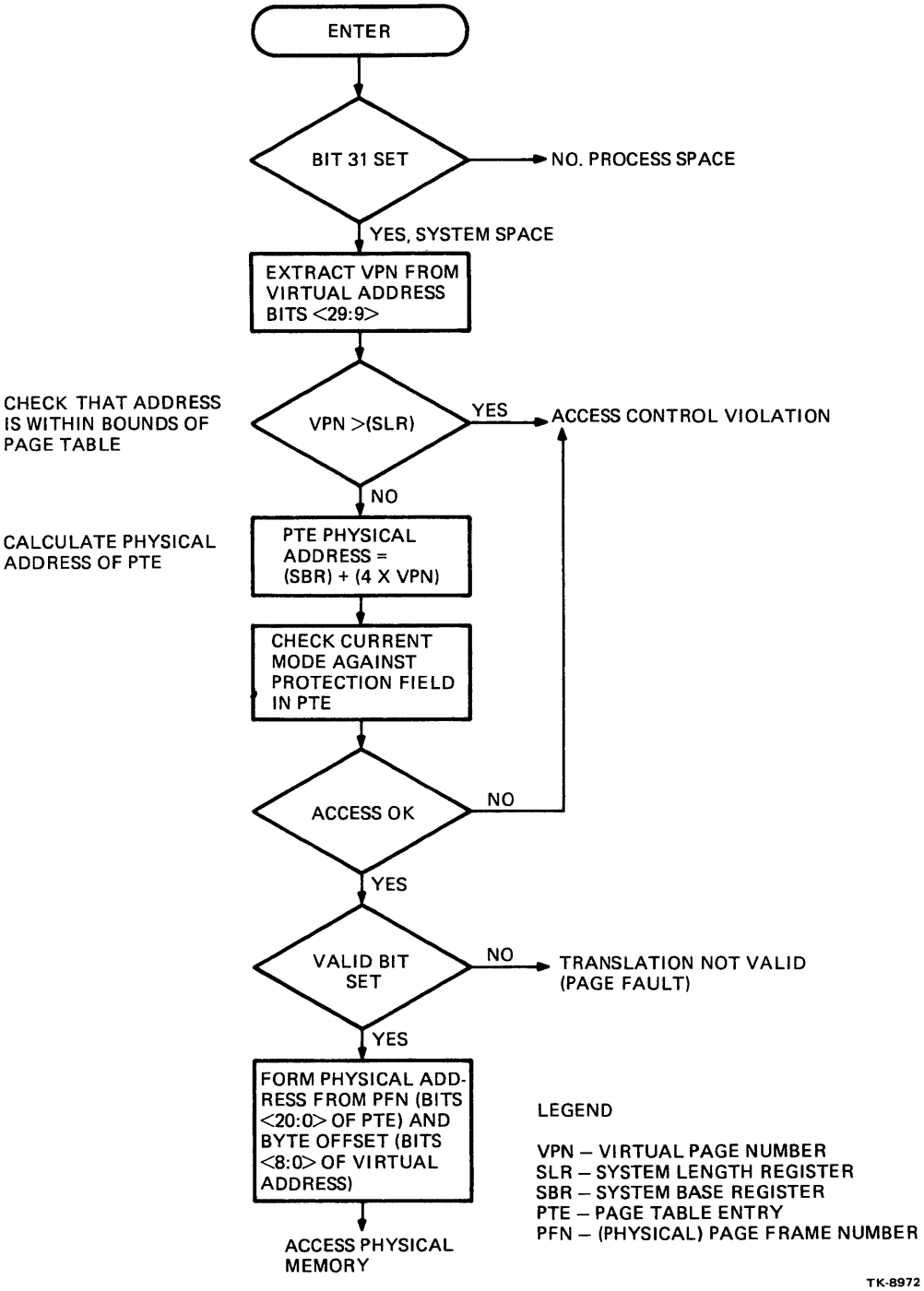


Figure 6-42 System Space Address Translation

LEGEND
VPN - virtual page number
SLR - system length register
SBR - system base register
PTE - page table entry
PFN - (physical) page frame number

PROCESS SPACE ADDRESS TRANSLATION

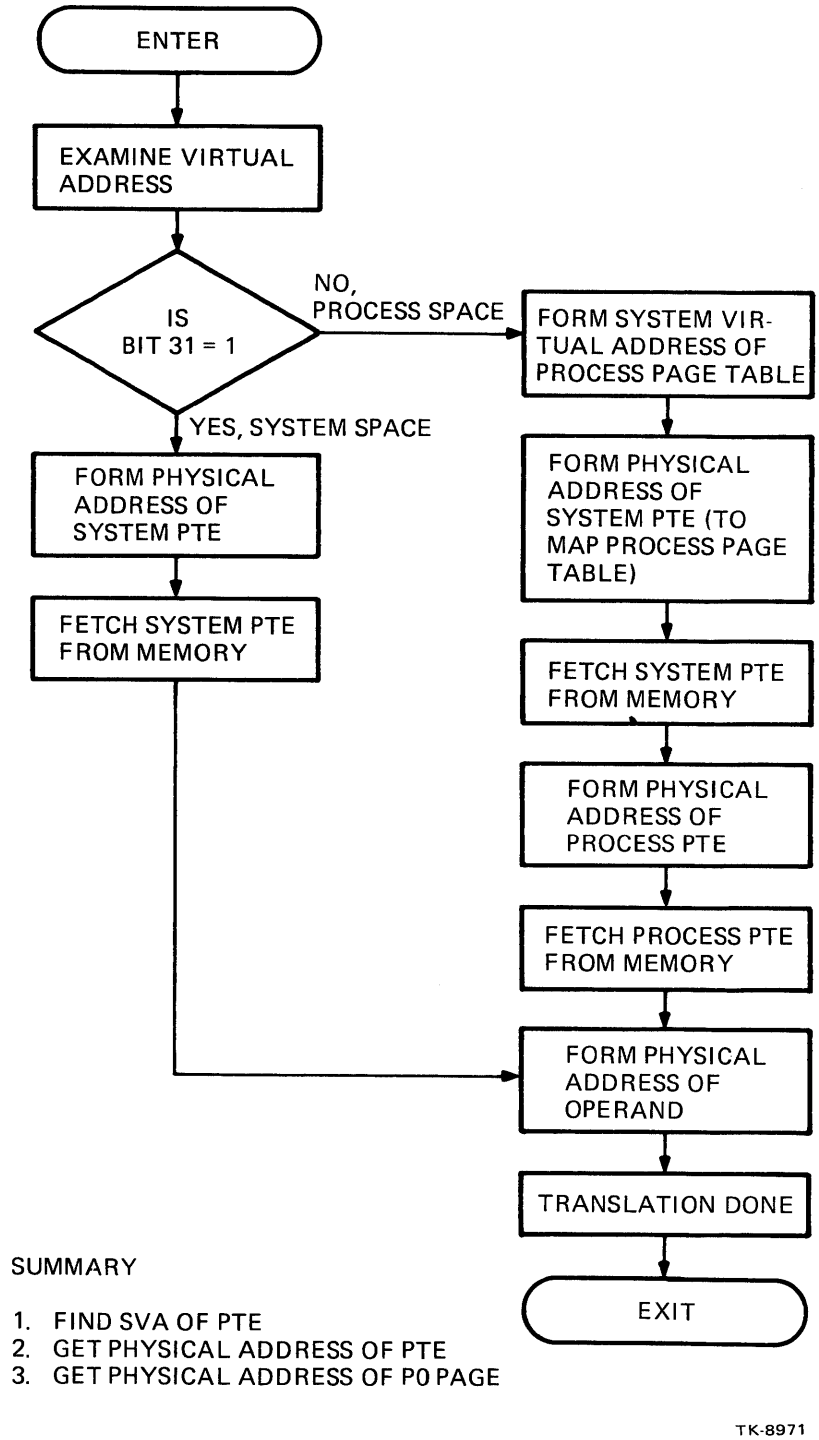


Figure 6-43 Process Space Address Translation Summary

1. Find SVA of PTE
2. Get physical address of PTE
3. Get physical address of PØ page

VIRTUAL TO PHYSICAL ADDRESS TRANSLATION

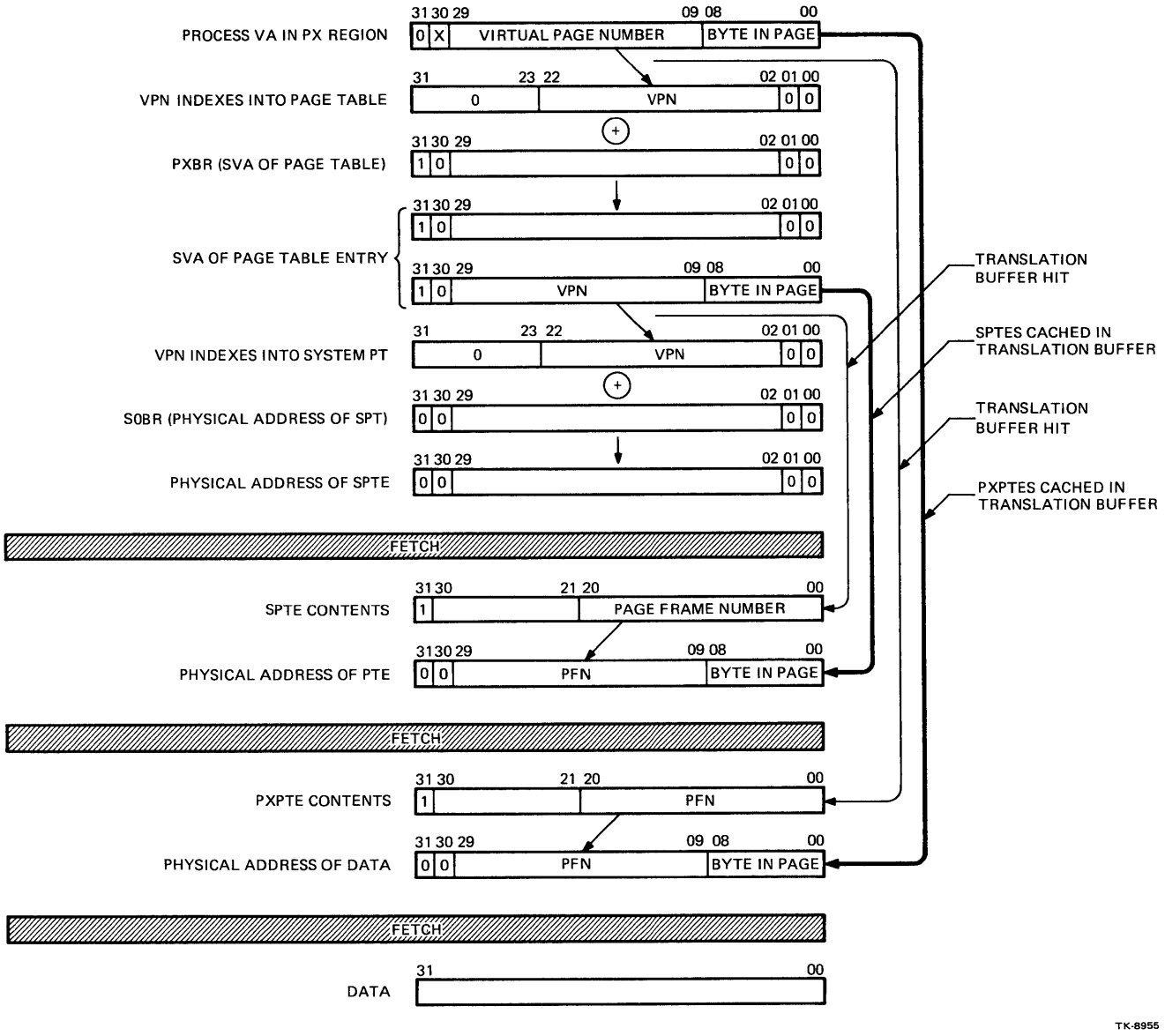


Figure 6-44 Virtual to Physical Address Translation

Note: This figure assumes that all SPTEs are valid.

SWAPPING

7

INTRODUCTION

The swapper is a process. The code of the swapper is part of the system image and executes in kernel access mode in SØ space.

The swapper is responsible for memory management on a system-wide basis. While the pager is the component servicing the demands within a process, the swapper balances the demands for physical memory by all of the processes in the system and the pageable portion of the operating system. To accomplish this purpose, three operations are performed by the swapper:

- inswap/outswap
- modified page writing
- shrinking working sets

Inswap/outswap operations are transfers of working sets between memory and disk.

Outswapping operations typically release over 1ØØ pages at a time, and provide a rapid way to replenish the free page list. Included in these transfers are:

- the PØ and Pl space pages that are memory-resident and valid, and
- the process headers (including the hardware context, accounting information, and all of the memory management data structures of the process).

The only information normally retained in physical memory after a process has been outswapped is found in data structures allocated from nonpaged dynamic memory, particularly the software process control block (PCB) and the job information block (JIB).

Modified page writing also is performed by the swapper process. When pages are needed, they always are allocated from the free page list.

Pages are provided for allocation by writing modified pages to their backing storage locations and then inserting the pages on the free page list.

Before the swapper outswaps a process, it will attempt to replenish the free page list by taking pages from the process working set (shrinking the working set).

The swapper also is involved in both process creation and system initialization.

This is discussed in the course modules "Process Creation and Deletion" and "System Initialization and Shutdown."

OBJECTIVES

1. Describe the swapping operation (inswap/outswap, handling I/O in progress, and global pages).
2. Determine the best size and placement for a system's swap file or files.
3. Explain why swapping is performed when paging also is implemented in VAX/VMS.
4. Discuss the effects of altering SYSGEN parameters relating to swapping.

RESOURCES

Reading

1. VAX/VMS Internals and Data Structures Manual, chapter on swapping

Additional Suggested Reading

1. VAX/VMS Internals and Data Structures Manual, chapters on memory management data structures, paging dynamics, and memory management system services

Source Modules

Facility Name	Module Name
SYS	PAGEFILE
	SWAPPER
	OSWPSCHED
	SDAT
	SHELL
	WRTMFYPAG
	IOCIOPOST
	SYSUPDSEC

TOPICS

I. Comparison of Paging and Swapping

II. Swapper functions

A. Maintain free page count

Write modified pages to paging file
Shrink working sets

B. Outswap - rules and example

C. Inswap - Rules and Example

III. Selected Events that Wake Swapper

IV. Locating Disk Files for Swap

V. How Swapper's PØ Page Table Is Used to Speed Disk I/O

SWAPPER

- Description of Code
  - located in SØ space
  - separate process
  - part of system image
  - executes in kernel mode only

- Function

  To control memory for the entire system through:
  - modified page writing
  - shrinking of working sets
  - inswapping/outswapping of working sets

COMPARISON OF PAGING AND SWAPPING

Table 7-1 Comparison of Paging and Swapping

Differences

	Paging	Swapping
Function	Supports programs with very large address spaces.	Supports a large number of concurrently active processes.
Implementation	Moves pages into and out of process working sets.	Moves entire processes into and out of physical memory.
How Invoked	Exception service routine that executes in the context of the process that incurred the page fault.	Separate process that is awakened from its hibernating state by components that detect a need for swapper activity.
Unit	the page	The process working set

Similarities

1. The pager and swapper work from a common data base.

2. The pager and swapper do conventional I/O.

3. Both components attempt to maximize the number of blocks read or written with a given I/O request.

*SYSGEN -
MPWPRIO
SWPPRIO

SWAPPER MAIN LOOP

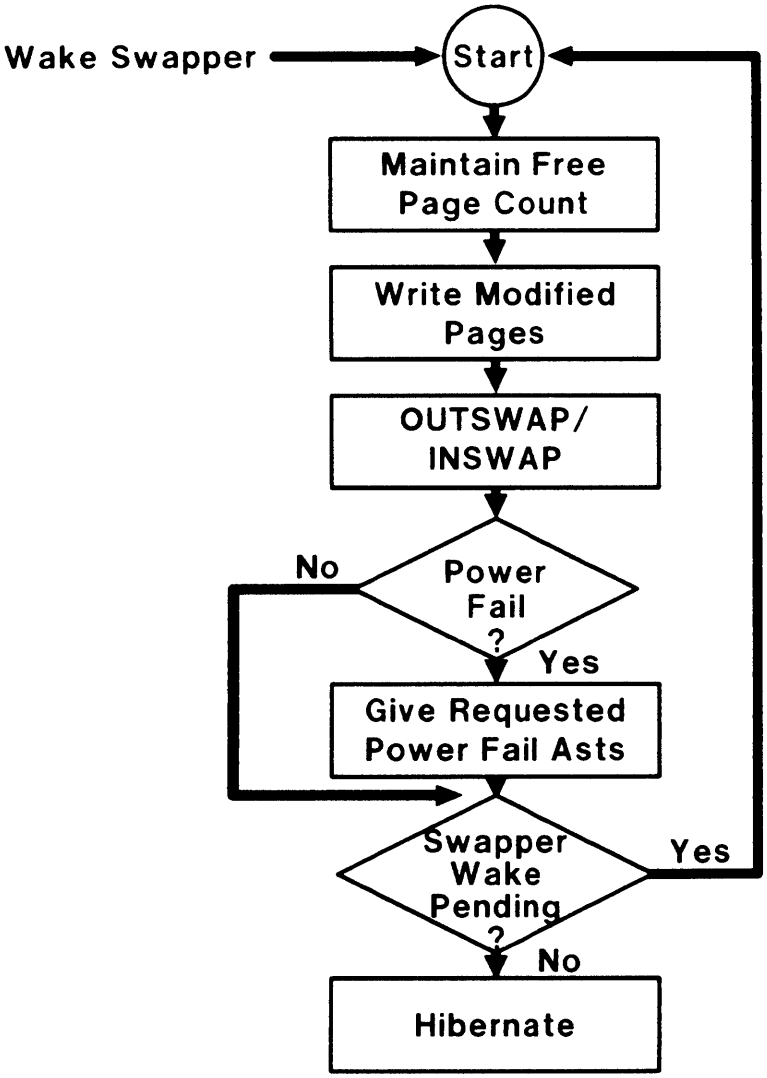


Figure 7-1 Swapper Main Loop

```
*SYSGEN -

MPW_WRTCLUSTER

FREELIM

MPW_HILIMIT

MPW_LOLIMIT
```

MAINTAINING FREE PAGE COUNT

To maintain at least FREELIM free pages, swapper will attempt to:

1. Reclaim pages from deleted process headers
2. Write modified pages
   - if (FREEGOAL minus number on free list) $\leq$ (number on modified list minus MPW_LOLIMIT)
   - will stop writing at MPW_LOLIMIT
3. Shrink working sets to SWPOUTPGCNT pages
4. Outswap processes

*SYSGEN -

SWPOUTPGCNT

ORDER OF SEARCH FOR POTENTIAL OUTSWAP CANDIDATES

Table 7-2 Order of Search for Potential Outswap Candidates

Process State (Mnemonic)	Priority Important	Initial Quantum	Additional Notes
Suspended (SUSP)	No	No	
Local Event Flag Wait (LEF)	No	No	Direct I/O count must be zero, longwait
Hibernating (HIB)	No	No	Longwait
Common Event Flag Wait (CEF)	No	No	Direct I/O count must be zero
Local Event Flag Wait (LEF)	No	No	Direct I/O count must be zero, not longwait
Hibernating (HIB)	No	No	Not longwait
Free Page Wait (FPG)	Yes	No	
Collided Page Wait (COLPG)	Yes	No	
Miscellaneous Wait (MWAIT)	No	No	
Common Event Flag Wait (CEF)	Yes	Yes	Direct I/O count cannot be zero
Local Event Flag Wait (LEF)	Yes	Yes	Direct I/O count cannot be zero
Page Fault Wait (PFW)	Yes	Yes	
Computable (COM)	Yes	Yes	

For an Outswap Table Section

- Shrink candidates' working sets to WSQUOTA
- Shrink candidates' working sets to SWPOUTPGCNT (initial quantum check)
- Select outswap candidate (initial quantum check)

If free page deficit not balanced and no outswap candidate, go to next outswap table section.

*SYSGEN

LONGWAIT

EXPANDING AND SHRINKING WORKING SETS

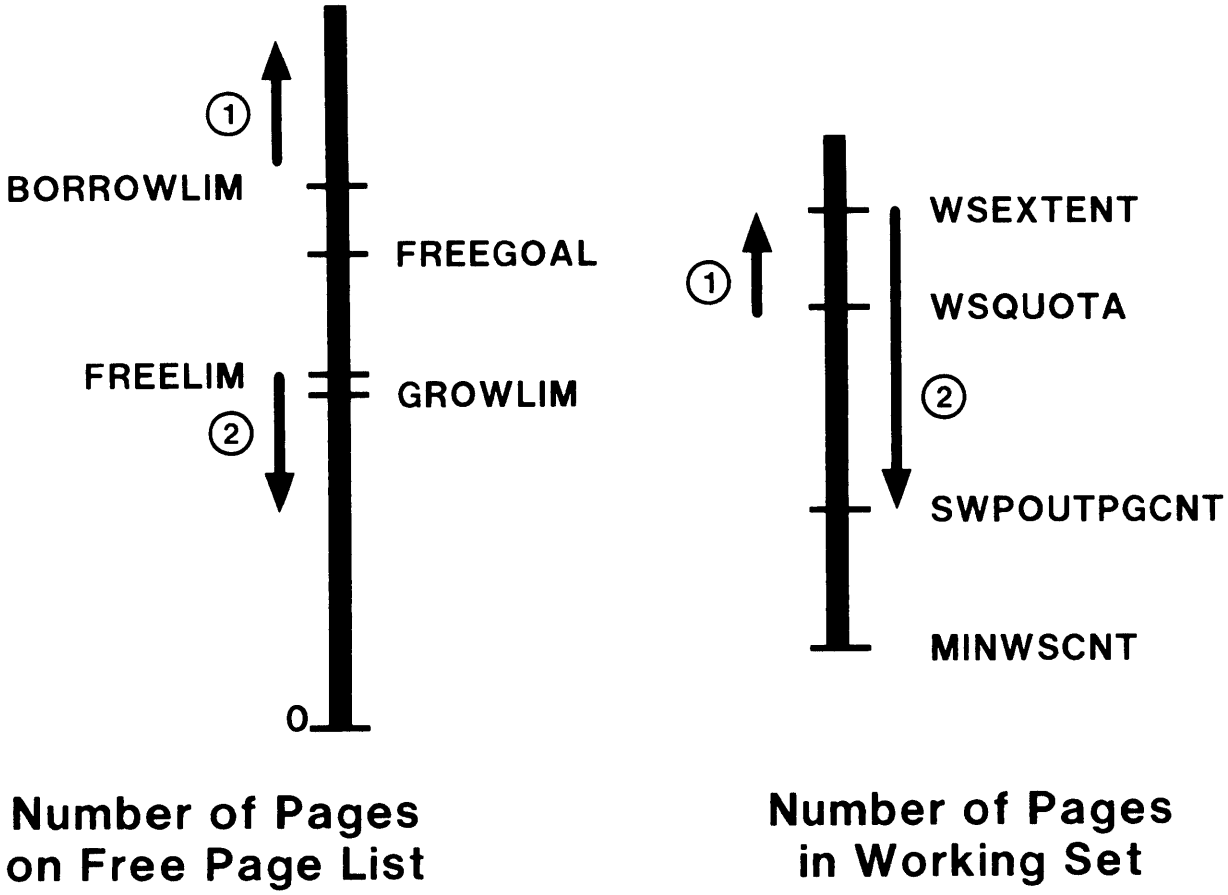


Figure 7-2 Expanding and Shrinking Working Sets

1. If free page count > BORROWLIM, working set may grow past WSQUOTA to WSEXTENT.
2. If free page count < FREELIM, swapper will attempt to
   - Shrink working sets from WSEXTENT to WSQUOTA
   - Shrink working sets from WSQUOTA to SWPOUTPGCNT

WAKING THE SWAPPER OR MODIFIED PAGE WRITER

Table 7-3 Selected Events that Cause the Swapper or Modified Page Writer to Be Awakened

Event	Module	Additional Comments
Process that is outswapped becomes computable	RSE	Swapper will attempt to make this process resident
Quantum end	RSE	Outswap previously blocked by initial quantum flag setting may now be possible
CPU time expiration	RSE	Process may be deleted, allowing previously blocked inswap to occur
Process enters wait state	SYSWAIT	Process that entered wait state may be suitable outswap candidate
Modified page list exceeds upper limit threshold	ALLOCPFN	Modified page writing is performed by swapper
Free page list drops below low limit threshold	ALLOCPFN	Swapper must balance free page cour by: 1. Writing modified pages 2. Swapping headers of previously outswapped process bodies Shrinking working sets 3. Swapping more processes
Balance slot of deleted process becomes available	SYSDELPRC	Previously blocked inswap may now be possible
Process header reference count goes to zero	PAGEFAULT	Process header can now be outswappe to join previously outswapped process body
System timer subroutine executes	TIMESCHDL	The swapper is awakened every second to check if there is any work to be done

OVERVIEW OF SWAPPER FUNCTIONS

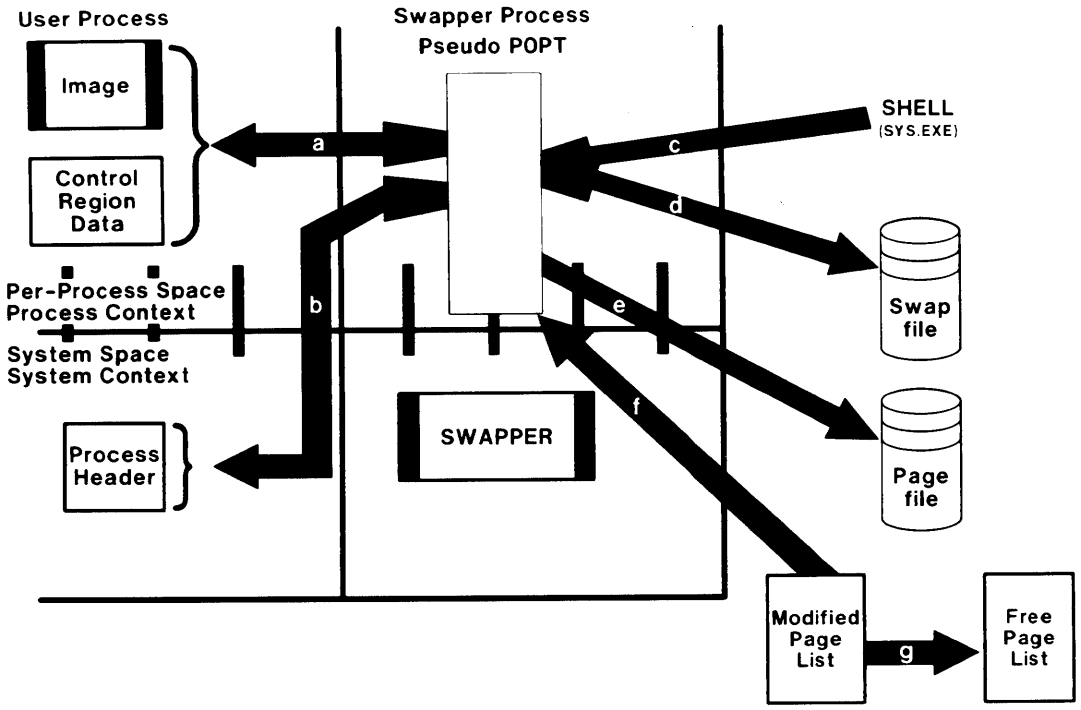


Figure 7-3 Overview of Swapper Functions

Outswap

a	P0 and P1 pages are "adopted" into swappers P0 space
d,e	Process outswapped to swap file/page file
b	PHD pages are "adopted" into swappers P0 space
d,e	PHD outswapped to swap file/page file

Inswap

Reverse of outswap

Modified page Writing

f	Selected modified pages "adopted" to swappers P0 space
e	Modified pages written to page file
g	"Modified" pages transferred to free page list

Process Creation

c	SHELL copied to swappers P0 space
a,b	SHELL code and data transferred to P1 and PHD of new process

LOCATING DISK FILES FOR SWAP

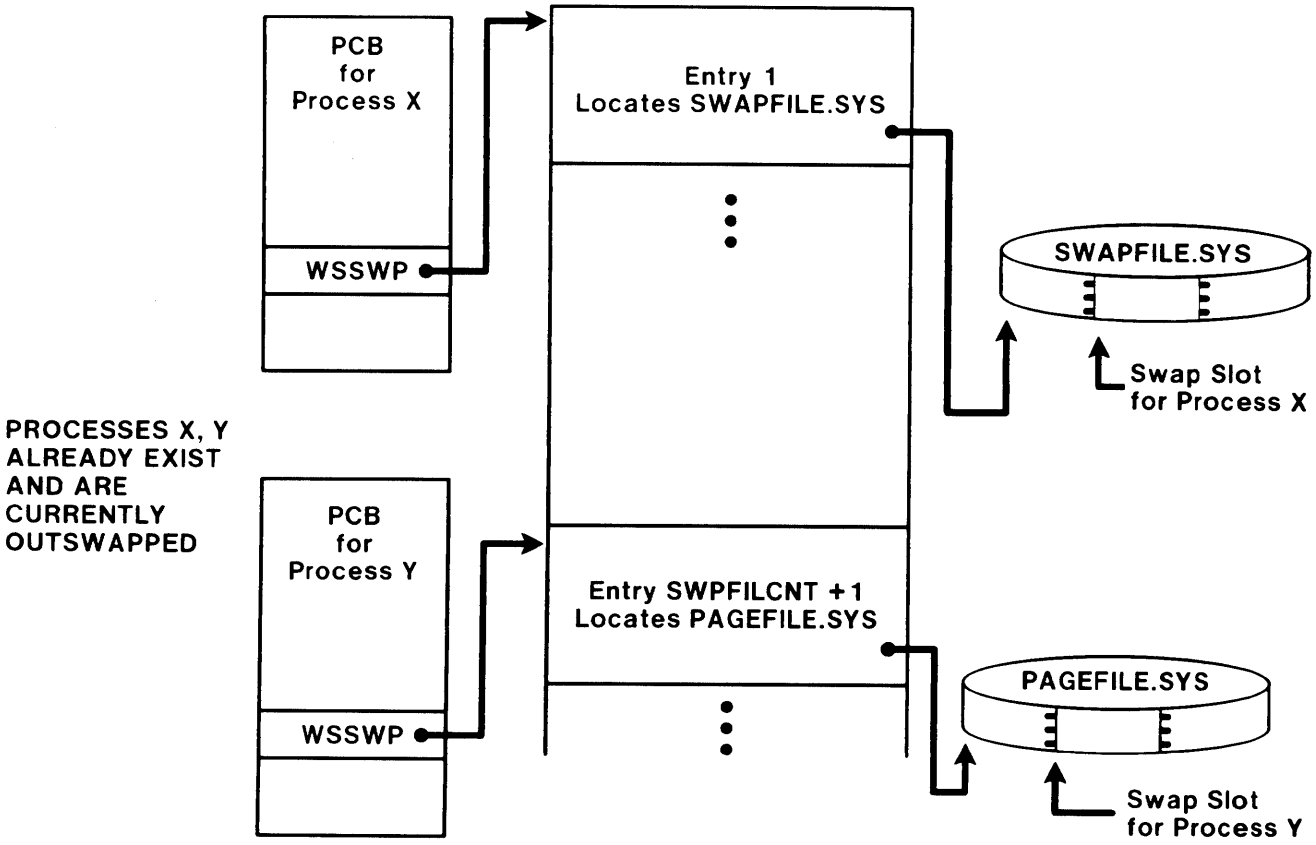


Figure 7-4 Locating Disk Files for Swap

The choice of swap file or page file is determined by a field in the PCB called WSSWP.

The use of a swap file is optional; however, using swap files should improve performance.

Swap slots are assigned dynamically in increments of SWAPALLOCINC, up to WSQUOTA pages.

*SYSGEN -

SWPFILCNT

PAGFILCNT

HOW SWAPPER'S P0 PAGE TABLE IS USED TO SPEED SWAP I/O

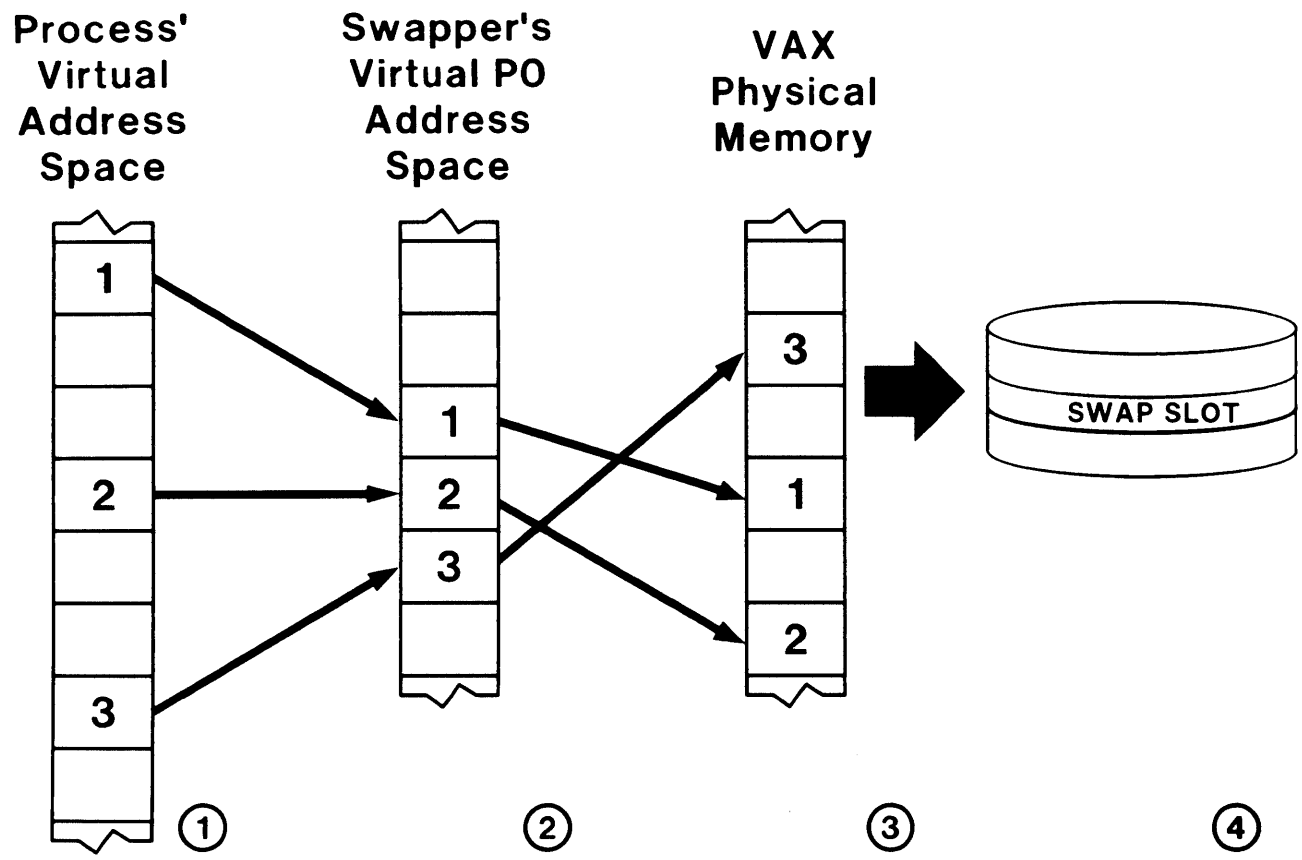


Figure 7-5 How Swapper's PØ Table Is Used to Speed Swap I/O

1. Working set pages usually virtually discontiguous in process address space.
2. Mapped to virtually contiguous addresses in swapper's PØ space.
3. Both virtual pages correspond to same PFNs in physical memory.
4. $QIO on swapper's contiguous virtual addresses --> one I/O to disk (QIO issued with base virtual address and byte count).

SWAPPER'S PSEUDO PAGE TABLES

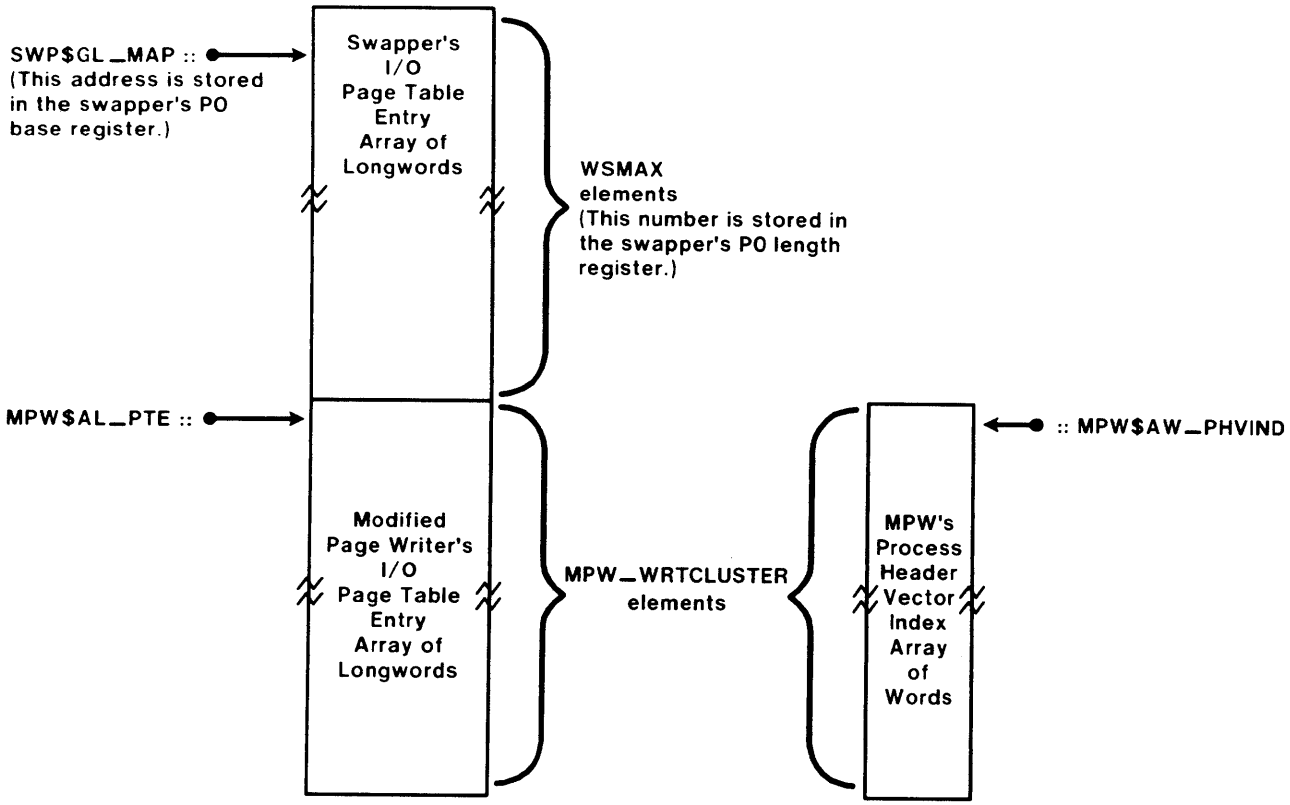


Figure 7-6 Swapper's Pseudo Page Tables

Swapper can have one swap I/O and one modified page write I/O in progress at the same time.

PARTIAL OUTSWAPS AND THE PROCESS HEADER

- In partial outswap, process body outswapped, process header remains resident
- Reason for partial outswap - pages locked in memory
  - $LCKPAG or direct I/O
  - Locked pages and PHD stay in memory
- Note that $LCKWSET has no effect on PHD being outswapped
- Effects of partial outswap
  - Balance slot still occupied, preventing another process getting inswapped (BALSETCNT = maximum number of resident processes, MAXPROCCNT >= BALSETCNT)
  - On inswap, if PHD still resident, only process body is inswapped, so process page tables are rebuilt, but not system page table entries mapping PHD.
- PHD size = f(PHD$K_LENGTH, WSMAX, PROCSECTCNT, VIRTUALPAGECNT)

OUTSWAP RULES

Table 7-4 Rules for Scan of Working Set List on Outswap

Type of Page	Valid	Action of Swapper for this Page
1. Process Page	Valid	Outswap page. If there is outstanding I/O and the page is modified, load SWPVBN array element with block in swap/page file where the updated page contents should be written when the I/O completes.
2. System Page		Impossible for system page to be in process working set. Swapper generates an error.
3. Global Read Only	Valid	a. If SHRCNT = 1, then outswap. b. If SHRCNT > 1, DROP from working set. It is highly likely that process can fault page later without I/O. This check avoids multiple copies of same page in swap page file.
4. Global Read/Write		DROP from working set. It is extremely difficult to determine whethe page in memory was modified after this copy was written to the swap pag file.
5. Page Table Page		Not part of process body. However, while scanning process body, VPN field in WSL is modified to reflect offset from beginning process header because page table pages will probably be located at different virtual addresses following inswap.

The scan of the working set list on outswap is keyed off a combination of the physical page type (WSL<3:1>) and the valid bit (PTE<31>).

OUTSWAP – WORKING SET LIST BEFORE OUTSWAP SCAN

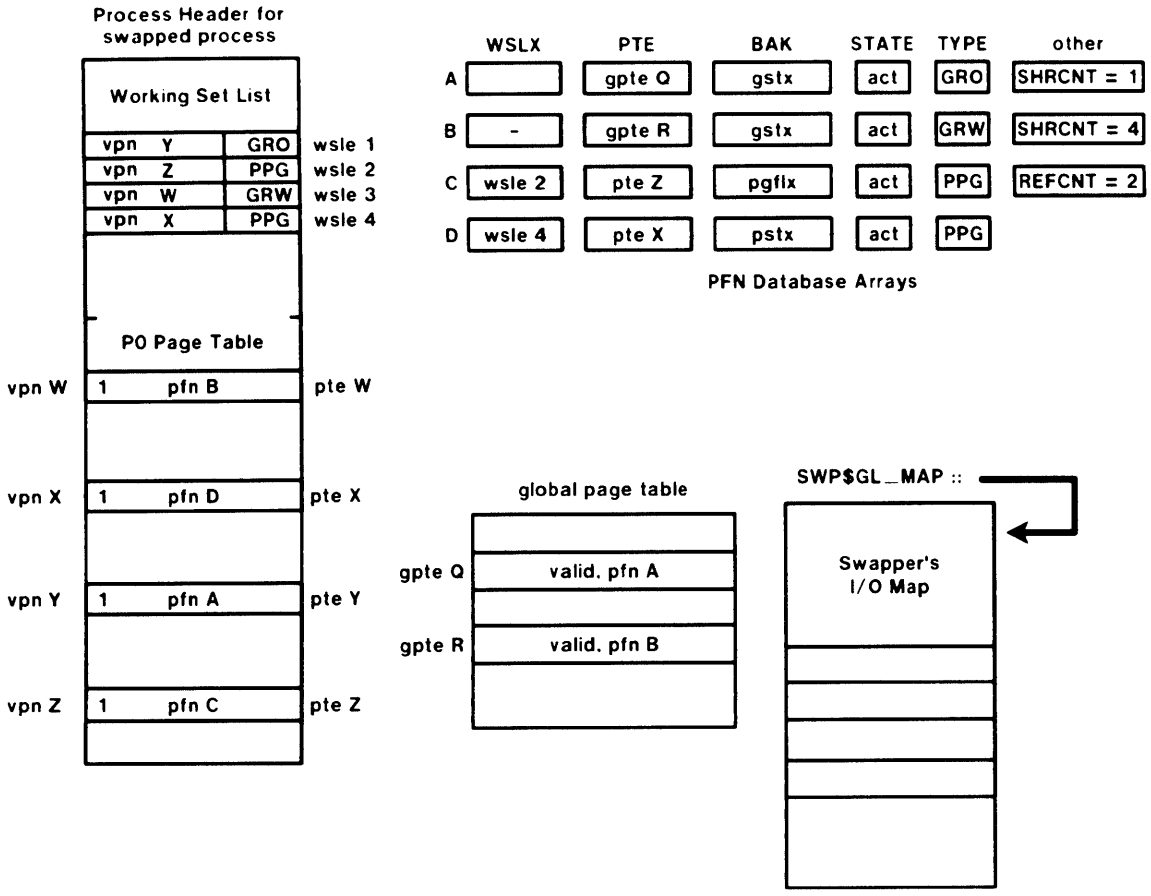


Figure 7-7 Outswap - Working Set List Before Outswap Scan

Y - Global read only, in only this process working set
Z - Process page, direct I/O in progress
W - Global read/write, in four process working sets
X - Process page

This is the state of the data structures before the swapper takes any action.

OUTSWAP – WORKING SET LIST AFTER OUTSWAP SCAN

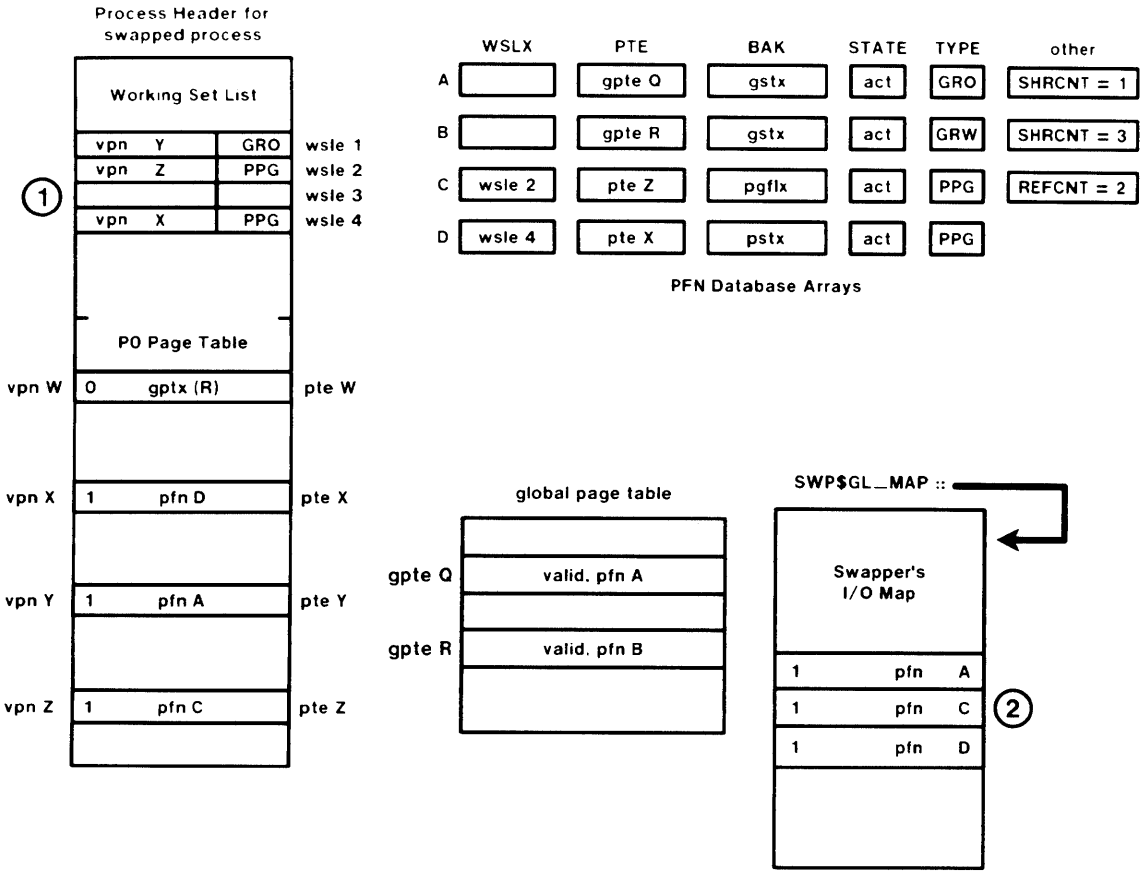


Figure 7-8 Outswap - Working Set List After Outswap Scan

1. The global read/write page is removed from the working set.

2. The remaining elements of the working set are mapped by the I/O map, and then the I/O request is made.

OUTSWAP – PROCESS PAGE TABLE CHANGES AFTER SWAPPER'S WRITE COMPLETES

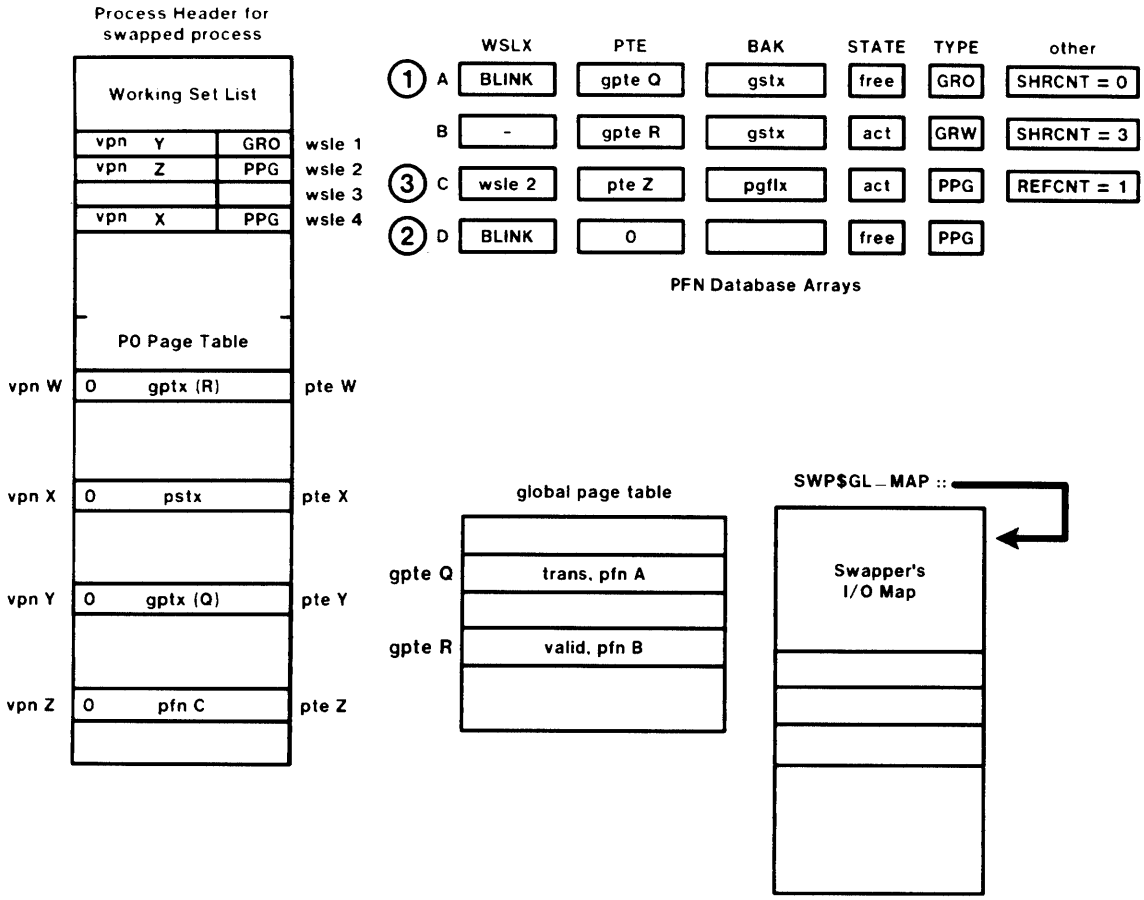


Figure 7-9 Outswap - Process Table Changes After Swapper's Write Completes

1. The global read-only page and the process page without I/O are placed on the free list.
2. Same as one.
3. The remaining process page (with I/O) has its REFCNT decremented by one.

INSWAP RULES

Table 7-5 Rules for Rebuilding the Working Set List and the Process Page Tables at Inswap

Type of Page Table Entry	Action of Swapper for this Page
1. PTE is valid	Page is locked into memory and was never outswapped.
2. PTE indicates a transition page (probably due to outstanding I/O when process was outswapped)	Fault transition page into process working set. Release duplicate page that was just swapped in.
3. PTE contains a global page table index (GPTX)	Swapper action is based on the contents of the global page table entry (GPTE).
(Page must be global read-only because global read/-write pages were dropped from the working set at outswap time)	a. If the global page table entry is valid, add the PFN in the GPTE to the process working set and release the duplicate page.
	b. If the global page table entry indicates a transition page, make the global page table entry valid, add that physical page to the process working set, and release the duplicate page.
	c. If the global page table entry indicates a global section table index, then keep the page just swapped in, and make that the master page in the global page table entry as well as the slave page in the process page table entry.

Table 7-5 Rules for Rebuilding the Working Set List and the Process Page Tables at Inswap (Cont)

Type of Page Table Entry	Action of Swapper for this Page
4. PTE contains a page file index or a process section table index	This is the usual content for pages that did not have outstanding I/O or other page references when the process was outswapped. The PFN in the swapper map is inserted into the process page table. The PFN arrays are initialized for that page.

At inswap time the swapper uses the contents of the page table entry to determine what action to take for each particular page.

INSWAP – WORKING SET LIST AND SWAPPER MAP BEFORE PHYSICAL PAGE ALLOCATION

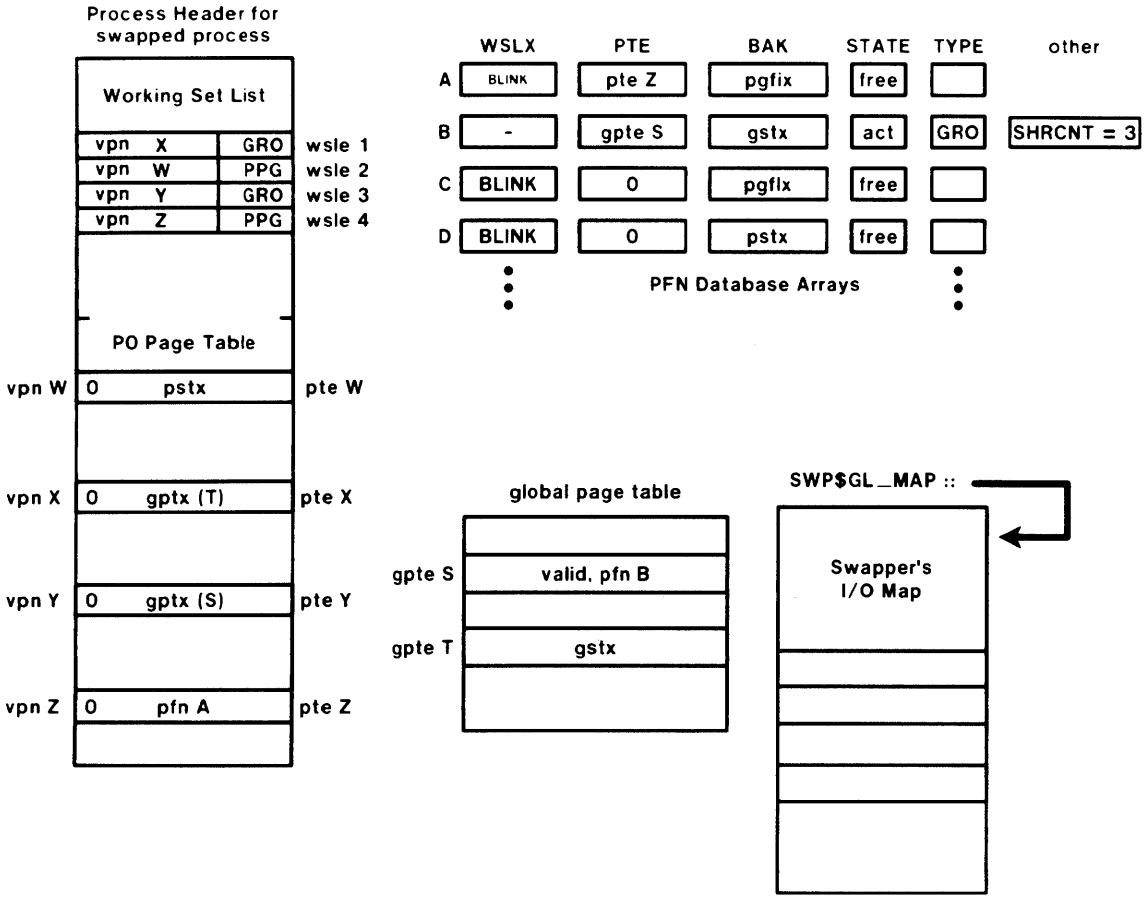


Figure 7-1Ø Inswap - Working Set List and Swapper Map Before Physical Page Allocation

X - Global read only, not in memory
W - Process page, not in memory
Y - Global read only, copy in memory (valid GPTE)
Z - Process page, on free page list

This is the state of the data structures before the swapper takes any action.

NOTE
Process header was not outswapped so it does not need to be inswapped.

INSWAP – WORKING SET LIST AND SWAPPER MAP AFTER PHYSICAL PAGE ALLOCATION

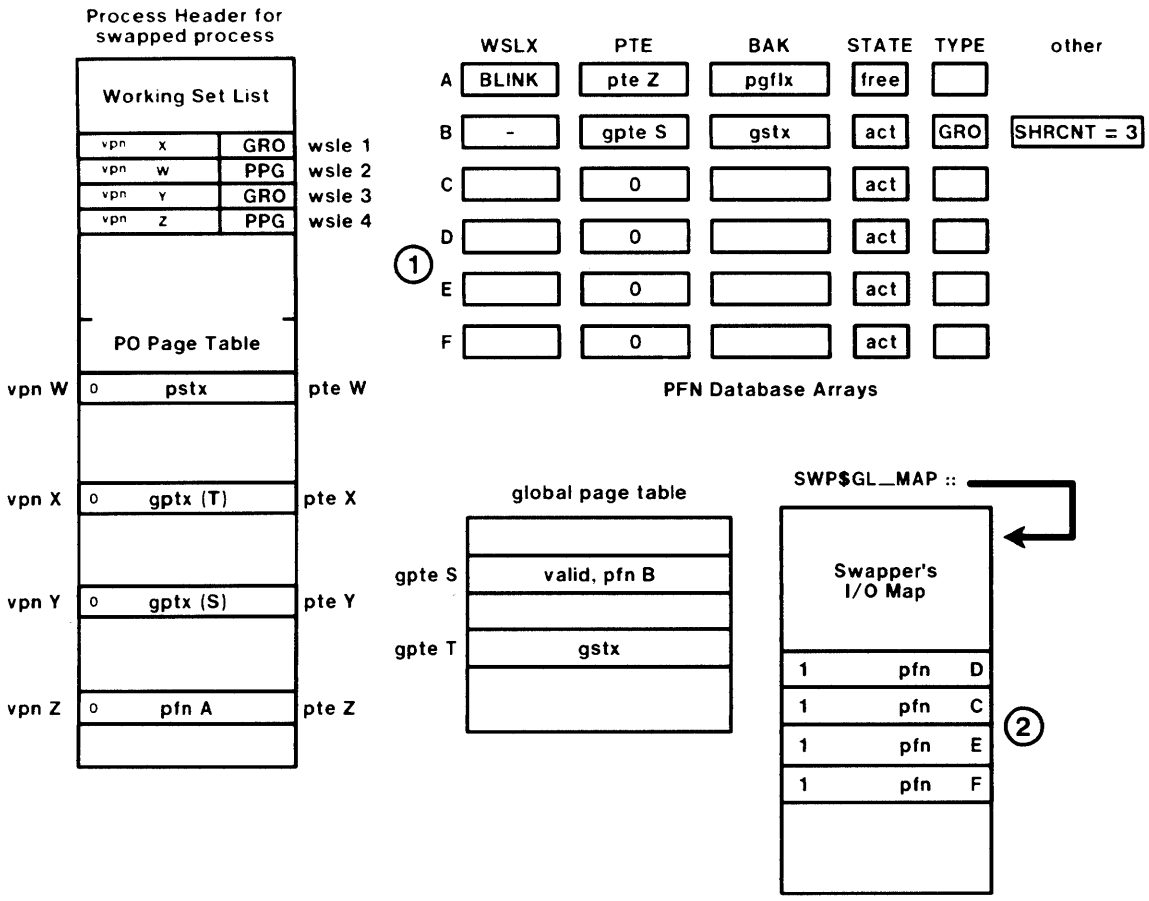


Figure 7-11 Inswap - Working Set List and Swapper Map After Physical Page Allocation

1. Swapper allocates pages from free page list for every page in the process working set.

2. Swapper copies PFNs into its PØ space.

   Swapper issues read from disk which copies swapped working set into physical memory.

INSWAP – WORKING SET LIST AND REBUILT PAGE TABLES

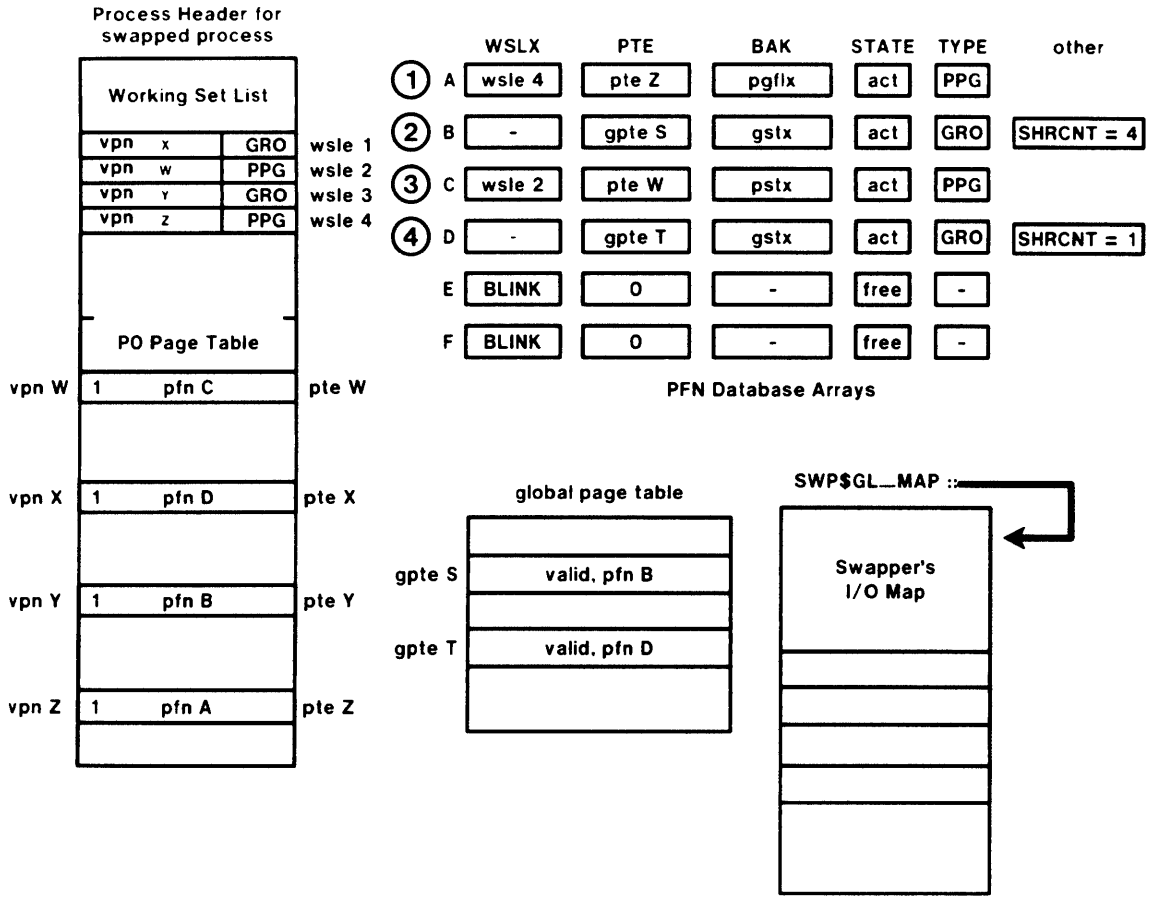


Figure 7-12 Inswap - Working Set List and Rebuilt Page Tables

❶ - pfn A still on free list so made valid.
- pfn E --> free page list

❷ - pfn B still valid so SHRCNT upped to 4
- pfn copied to PTE Y
- pfn F --> free page list

❸ pfn C copied to PTE W

❹ - pfn D copies to PTE X
- SHRCNT = 1

The actual order of operations is 4,3,1,2.

HOW MODIFIED PAGE WRITER GATHERS PAGES

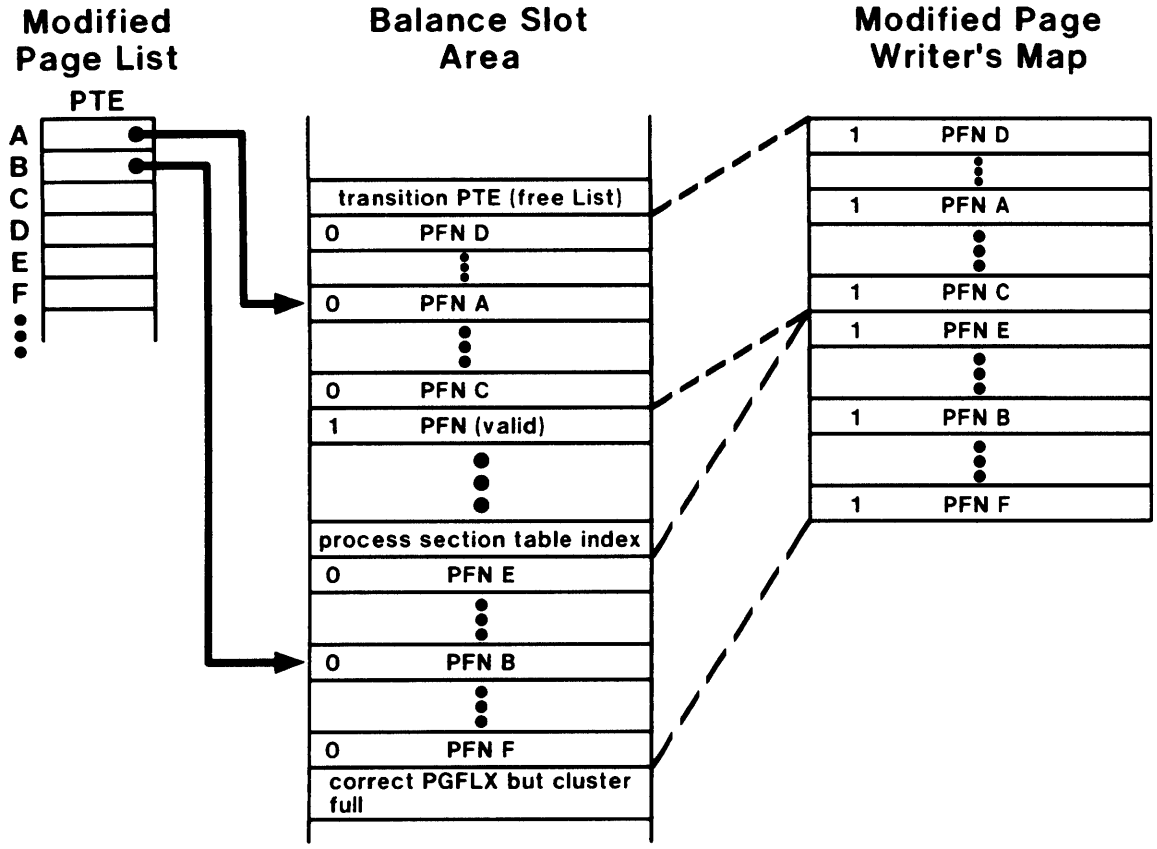


Figure 7-13 How Modified Page Writer Gathers Pages

Gather pages around selected PTE from modified pages list until PTE is:

- free page PTE
- valid PTE
- PSTX in PTE
- PGFLX in PTE but cluster is full

*SYSGEN -
MPW_CLUSTER

APPENDIX

```
 1      .SBTTL  SWAPPER - MAIN LOOP
 2  ;++
 3  ; FUNCTIONAL DESCRIPTION:
 4  ;    THE MAIN LOOP OF THE SWAPPER IS EXECUTED WHENEVER THE SWAPPER IS AWAKEN
 5  ;    FOR ANY REASON.  EACH OF THE FUNCTIONAL ROUTINES WILL CHECK TO SEE IF
 6  ;    THEY HAVE ANY ACTION TO PERFORM.
 7  ;--
 8      .PSECT  $AEXENONPAGED           ; NON-PAGED PSECT
 9  LOOP:       BSBB    BALANCE         ; BALANCE FREE PAGE COUNT
10      BSBW    MMG$WRTMFYPAG           ; WRITE MODIFIED PAGES
11      BSBB    SWAPSCHED               ; SCHEDULE SWAP
12      TSTL    W^EXE$GL_PFATIM         ; CHECK FOR POWER FAIL TIME
13      BEQL    15$                     ; BRANCH IF NO POWERFAIL
14      JSB     EXE$POWERAST            ; GIVE ANY REQUIRED POWER FAIL ASTS
15  15$:        MOVL    W^SCH$GL_CURPCB,R4 ; GET PROPER PCB ADDRESS
16      MOVAQ   W^SCH$GQ_HIBWQ,R2       ; AND ADDRESS OF WAIT QUEUE HEADER
17      SETIPL  #IPL$_SYNCH             ; BLOCK SYSTEM EVENTS WHILE CHECKING
18      BBSC    #PCB$V_WAKEPEN,PCB$L_STS(R4),20$ ; TEST AND CLEAR WAKE PENDING
19      PUSHL   #0                      ; NULL PSL
20      BSBW    SCH$WAITK               ; WAIT WITH STACK CLEAN
21  20$:        SETIPL  #0              ; DROP IPL
22      BRB     LOOP                    ; CHECK FOR WORK TO DO
23      .DISABLE LSB
```

Example 7-1 Swapper - Main Loop

MODIFIED PAGE WRITE CLUSTERING

- Scans PTE's in reverse order from page read clustering
- Can write clusters to
  - page file
  - image file
- If SWPVBN=Ø, page going to swap file, no clustering.
- When building clusters
  - cluster size determined by SYSGEN parameter MPW_WRTCLUSTER
  - scan terminated if
    1. PTE indicates page not on modified page list
    2. PTE points to page in shared memory, or page mapped by PFN
    3. PSTX or GSTX doesn't match that of original PTE
- When writing to page file
  - build up several mini-clusters into one larger cluster
  - use one I/O to write larger cluster to disk
  - note that on later page read, mini-clusters may be read separately.

PROCESS CREATION AND DELETION

8

INTRODUCTION

This module discusses the operations required

- to create and delete processes under VAX/VMS, and
- to activate and rundown images within existing processes.

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.

Image activation and rundown involve construction and removal of the data structures and the virtual memory that are defined for a specific image rather than for the 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 require execution of the LOGINOUT image for authorization (only in the case of interactive processes), and also to map the CLI.

The discussion of the life cycles of processes and images should contribute to a better understanding of

- the implications of multiprogramming application designs, and
- the more fundamental concepts of process and image themselves.

OBJECTIVES

Upon completion of this module, you will be able to:

1. List and explain several differences between user-created processes, interactive processes, and batch processes, in terms of:

   - how the processes are created and deleted
   - how images are activated and exited

2. Discuss the effects of altering SYSGEN parameters related to process creation and deletion, as well as image activation.

RESOURCES

Reading

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

Source Modules

Facility Name	Module Name
SYS	SHELL PROCSTRT SYSCREPRC, SYSDELPRC SYSEXIT SYSIMGACT, SYSIMGSTA, SYSRUNDWN
DCL	HANDLE, IMAGECTRL, IMAGEXECT, COMMAND
MCR	MCRHANDLE, MCRIMGCTL, MCRIMGEXE, MCRCOMD
LOGIN	
JOBCTL	
INPSMB	

TOPICS

I. Process Creation and Deletion

   A. Roles of operating system programs

   B. Creation of process data structures

   C. Deletion sequence

II. Initiating Jobs

   A. Interactive

   B. Batch

III. DCL Structure and Function

IV. Image Activation and Rundown

   A. Mapping image file

   B. Image startup

   C. Termination handlers

LIFE OF A PROCESS

Table 8-1 Steps in Process Creation and Deletion

Action	Code
Creating process	SYS$CREPRC
Inswapped	SWAPPER
Process startup	PROCSTRT
Process deleted	SYS$DELPRC

Table 8-2 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)	SW priority boost	- logical names (sys$input...)
Process returned COMO		- Catch-all cond. hand. - RMS dispatcher - Image name moved to PHD - Image activated

CREATION OF PCB, JIB, AND PQB

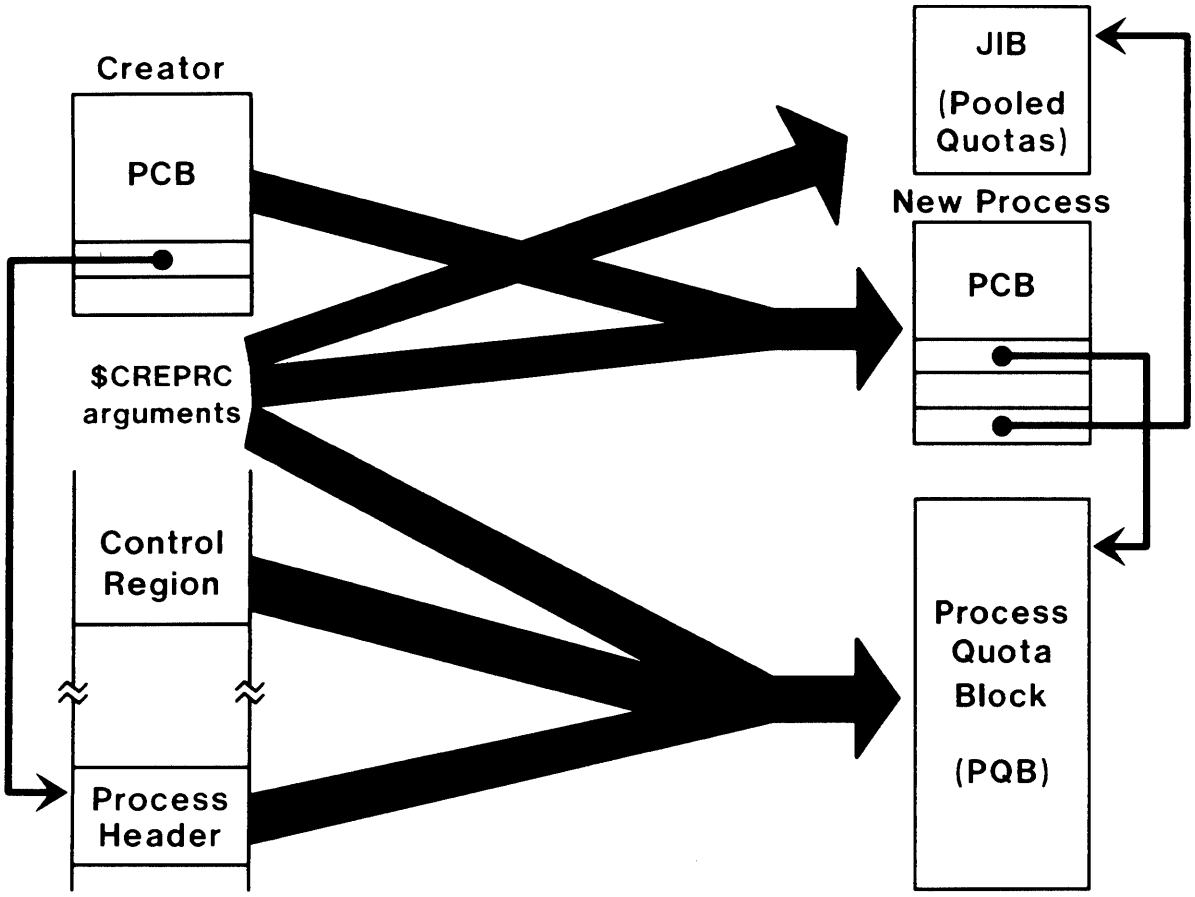


Figure 8-1 Creation of PCB, JIB and PQB

1. $CREPRC allocates from nonpaged pool.

   - PCB
   - JIB (if new process is detached)
   - PQB (temporary)

2. These new data structures are filled from:

   - $CREPRC arguments (e.g., process name)
   - Creator's PCB (e.g., owner PID)
   - Creator's control region (e.g., user name)
   - Creator's process header (e.g., default privileges)

RELATIONSHIPS – PCBs AND JIB

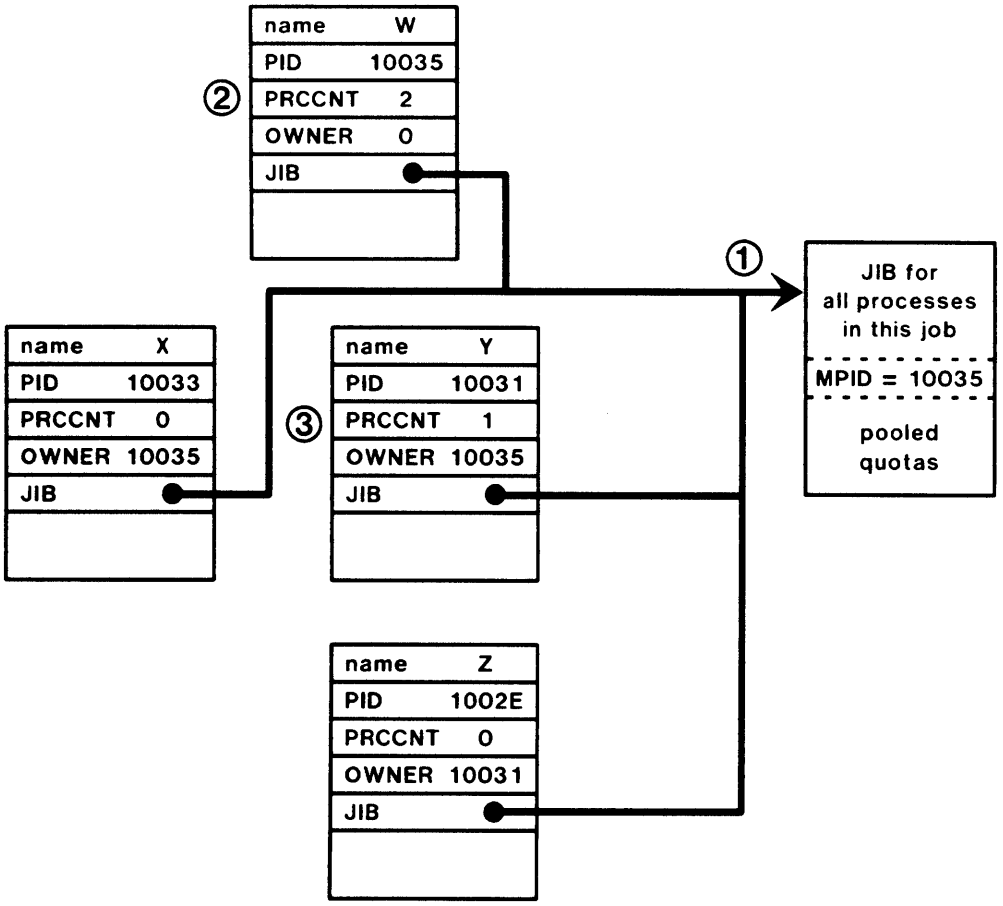


Figure 8-2 Relationships - PCBs and JIB

❶ All PCBs point to JIB

W created X and Y

❷ W's PRCCNT is 2

❸ X and Y owner PID is W PID

Y created Z

No pointers from creator to subprocess

PCB VECTOR

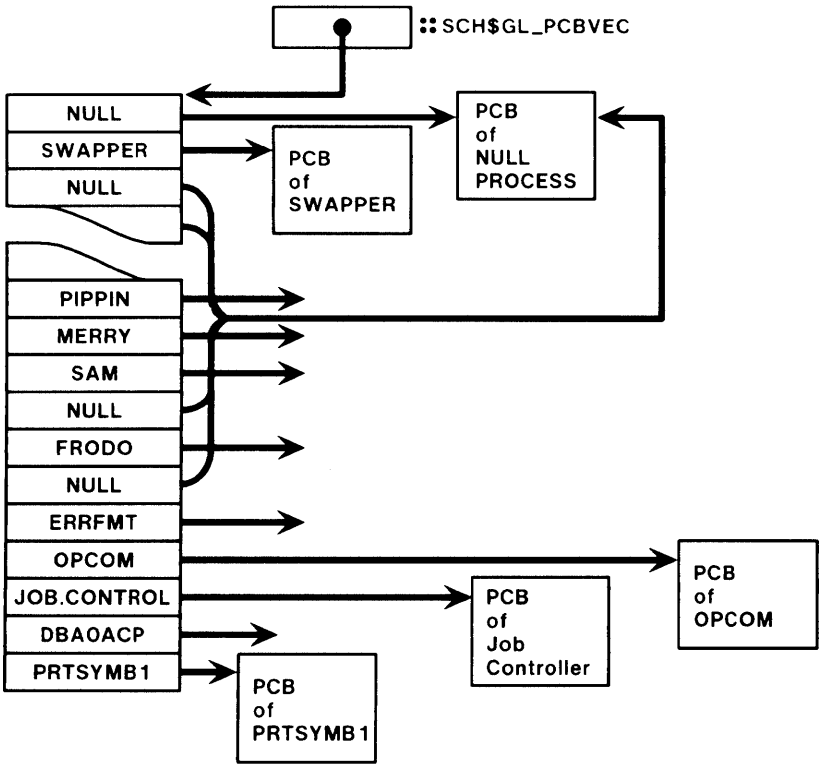


Figure 8-3 PCB Vector

- On process creation, search for unused vector
- Unused vectors point to Null's PCB
- Table of pointers to all PCBs
- Index into table is PID
- SCH$GL_PCBVEC points to start of table

*SYSGEN

MAXPROCESSCNT

Maximum number of processes allowed on the system.

PID AND PCB, SEQUENCE VECTORS

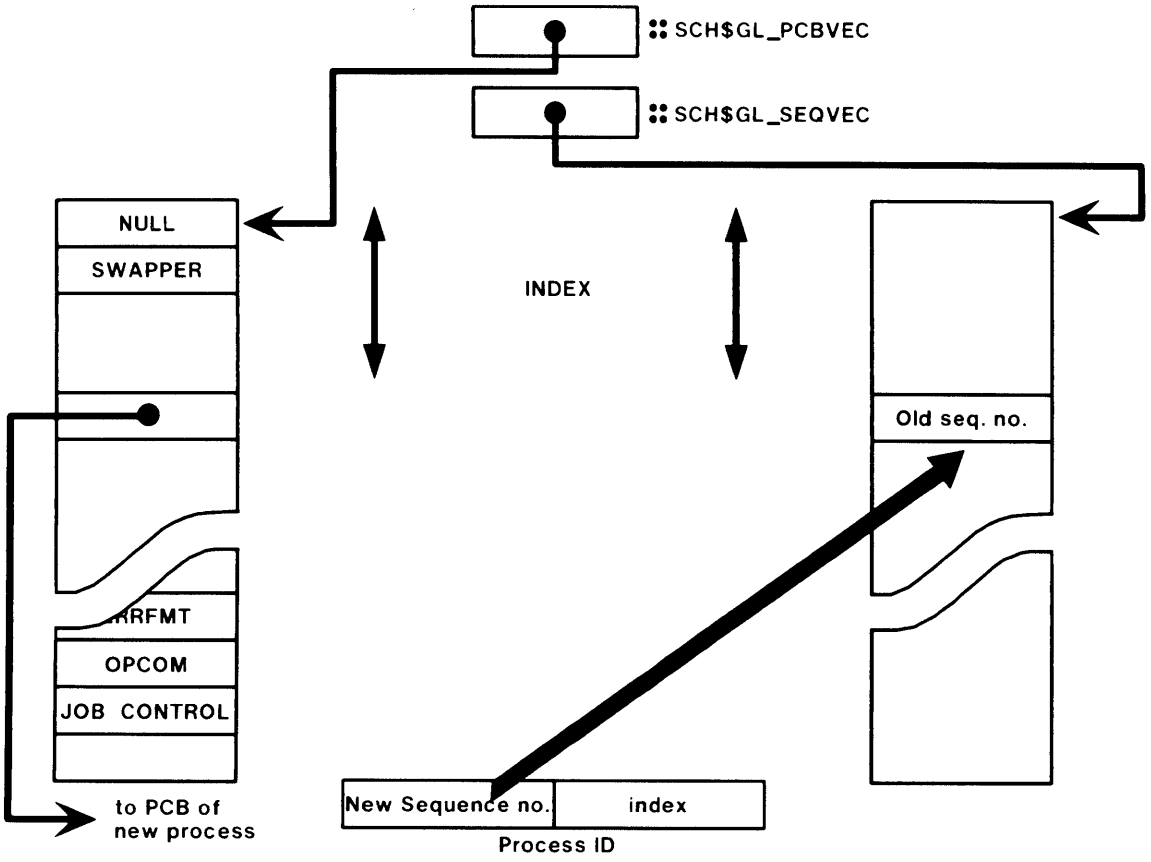


Figure 8-4 PID and PCB, Sequence Vectors

- PID contains two parts:
  - Index (into PCB vector and sequence vector)
  - Count (sequence number)
- PID formed at process creation.
- Old sequence number + 1 = new sequence number.
- SCH$GL_SEQVEC points to start of sequence vector.

SWAPPER'S ROLE IN PROCESS CREATION

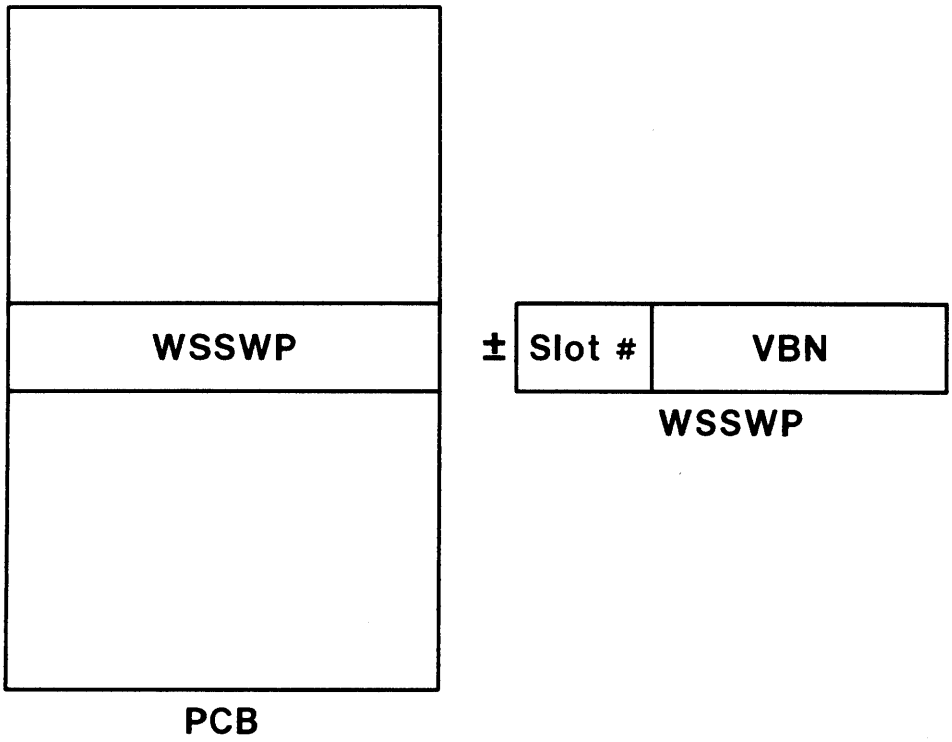


Figure 8-5 Swapper's Role in Process Creation

- For new process, WSSWP is negative
- Negative WSSWP --> SHELL copied
- Swapper
  - Stores SYSGEN parameters in PHD
  - Initializes pointers, counters in PHD
  - Initializes system page table entries

PROCSTRT'S ROLE IN PROCESS CREATION

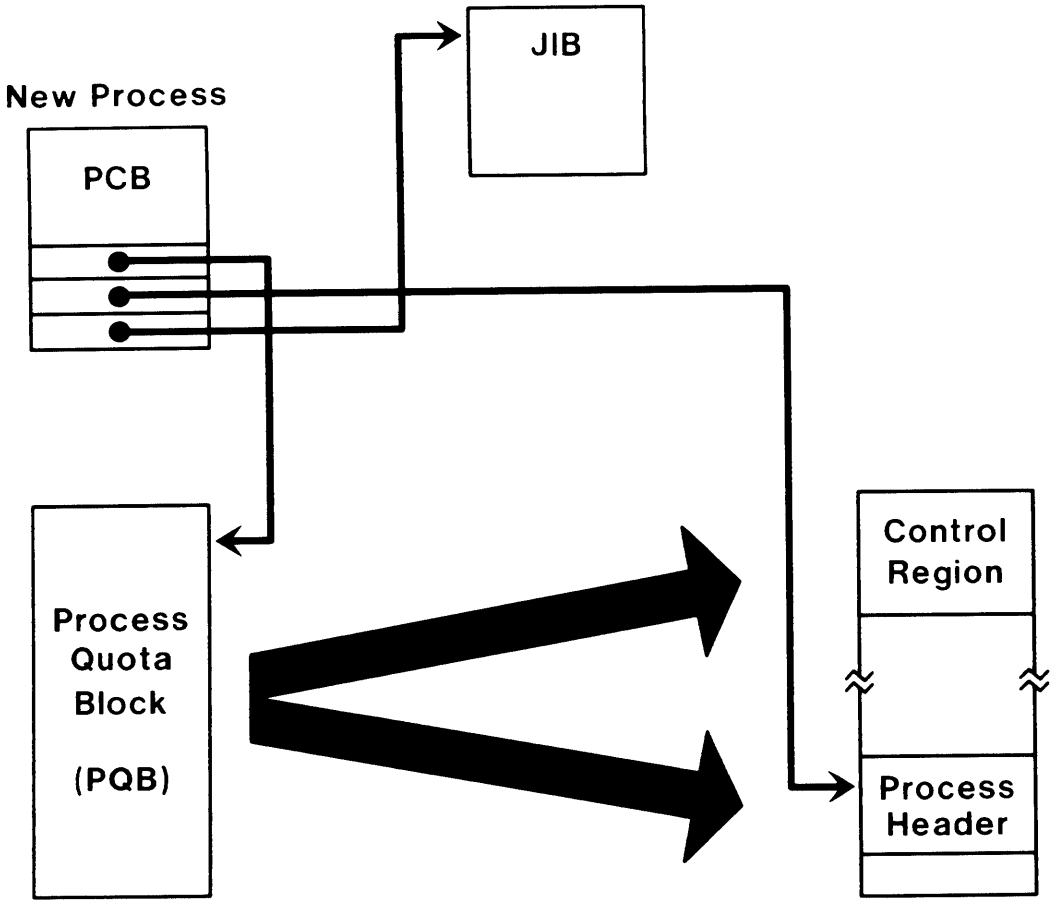


Figure 8-6 PROCSTRT's Role in Process Creation

- Hardware PCB defined in SHELL
- PC and IPL invoke PROCSTRT at IPL 2
- Code located in SYS.EXE
- Functions
  - PQB --> PHD and P1
  - Change to user mode, IPL Ø
  - Call SYS$IMGACT
  - Call image at transfer vector

AFTER PROCESS CREATION, IMAGE RUNS AND EXITS

Table 8-3 Steps in Process Creation and Deletion

Action	Code
Creating process	SYS$CREPRC
Inswapped	SWAPPER
Process startup	PROCSTRT
Process deleted	SYS$DELPRC

INTRODUCTION – PROCESS DELETION

- 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

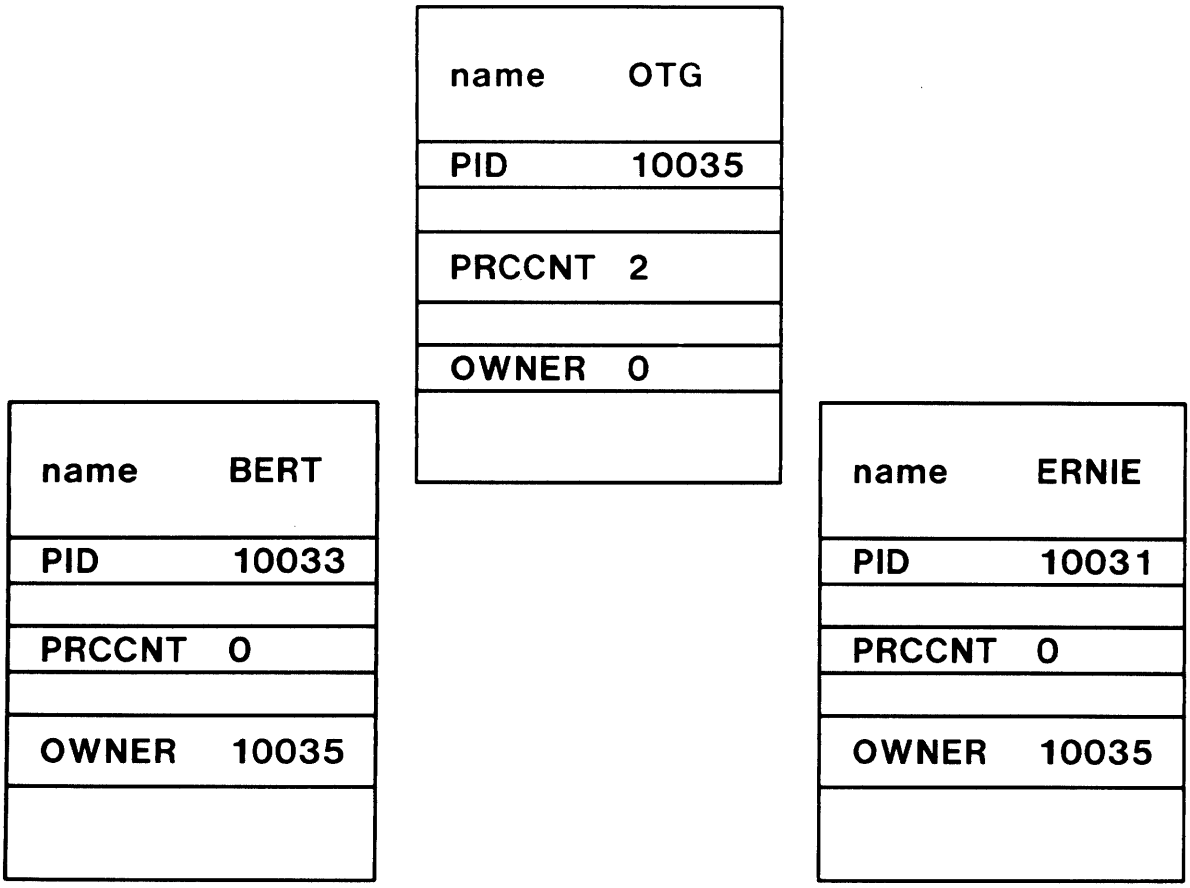


Figure 8-7 Process Deletion

- Deleted by special KAST while CURRENT.
- Sequence
  - $DELPRC (subprocesss)
  - Accounting information to job controller
  - Call SYS$RUNDOWN
  - Delete P1 space
  - Free PCBVEC and SWAP slots, page file space
  - Decrement counts
    - Balance set
    - Total processes
  - Jump to SCH$SCHED

PROCESS TYPES AND CREATORS

Table 8-4 Process Types and Creators

Process Type	Code	Created By
Interactive	$CREPRC	Job Controller
Batch	$CREPRC	Job Controller
Subprocess	$CREPRC= $RUN $SPAWN (DCL)	Owner
Detached	$CREPRC $RUN (DCL)	Owner

Note: RUN and SPAWN call $CREPRC

After system initialization

- A process is created by another process
- Process creation is done by $CREPRC

DCL BASED PROCESSES

- Image run is LOGINOUT.EXE
- LOGINOUT functions:
  - Interactive
    prompts for username, password
    checks SYSUAF.DAT
  - Batch
    no prompting
  - Alters process characteristics to match UAF record
    privileges
    quotas
  - Activates login command procedure

INITIATING INTERACTIVE JOB

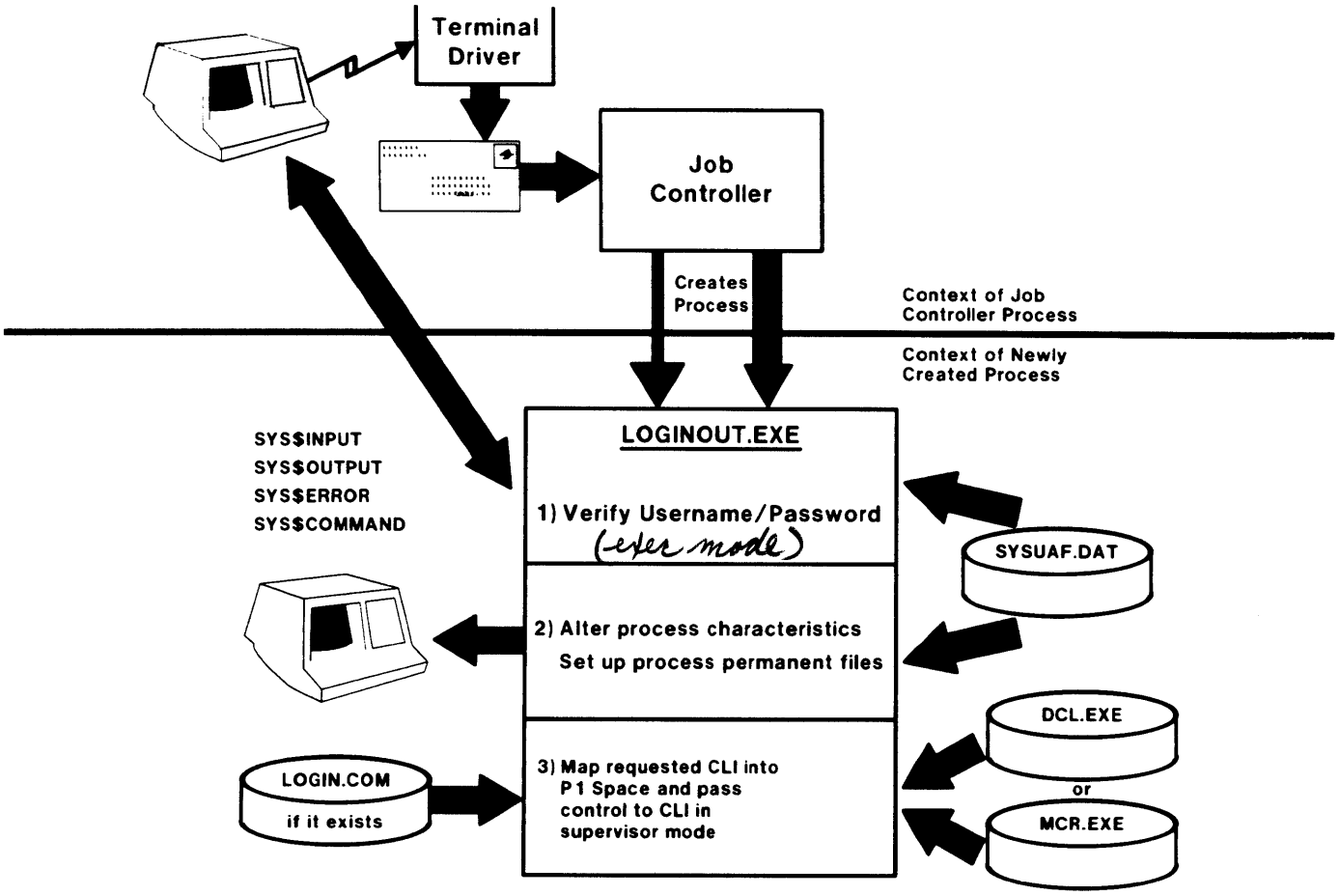


Figure 8-8 Initiating Interactive Job

- Job controller notified by driver
- Job controller creates process _TTcu:
- LOGINOUT runs
- DCL, MCR mapped

INITIATING JOB USING $SUBMIT

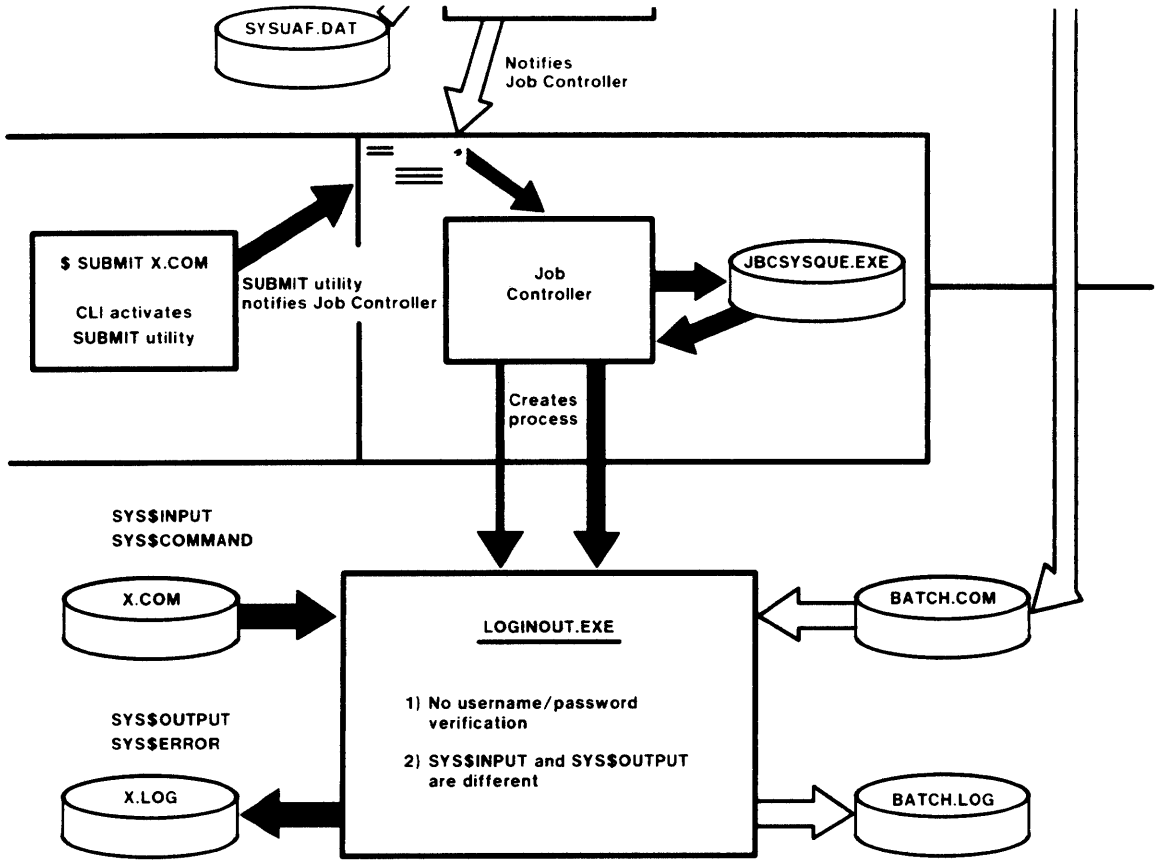


Figure 8-9 Initiating Job Using $SUBMIT

- Similar to interactive process but -
  - 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

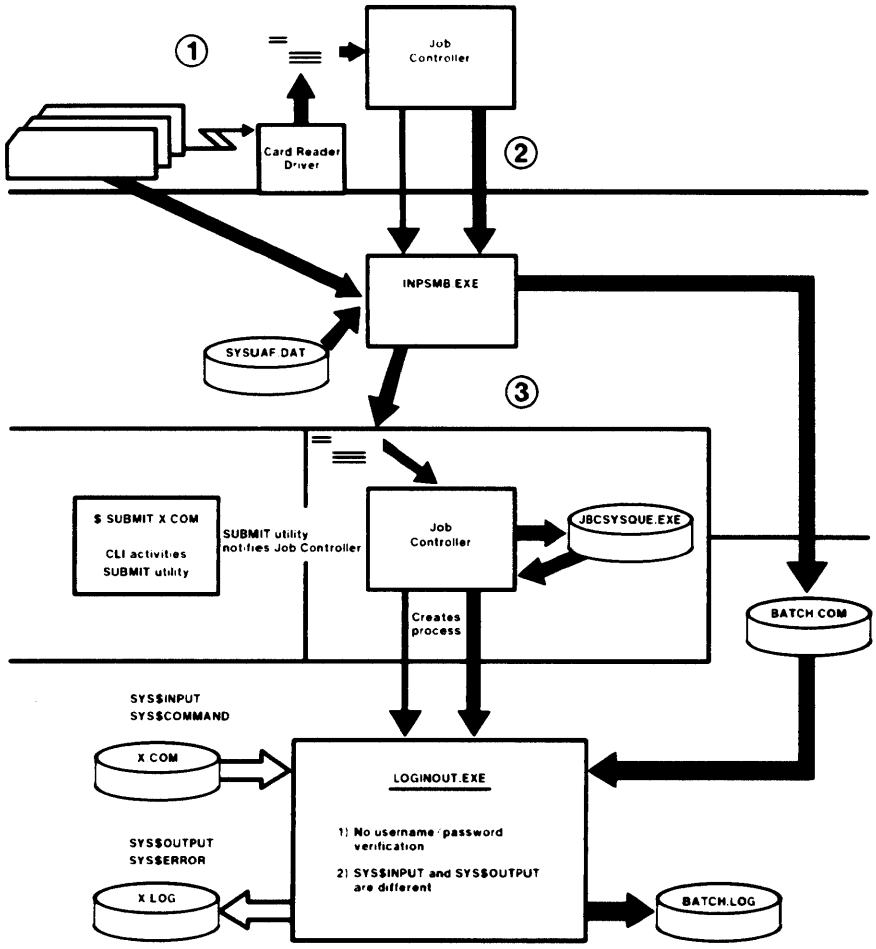


Figure 8-1Ø Initiating Job Through Card Reader

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

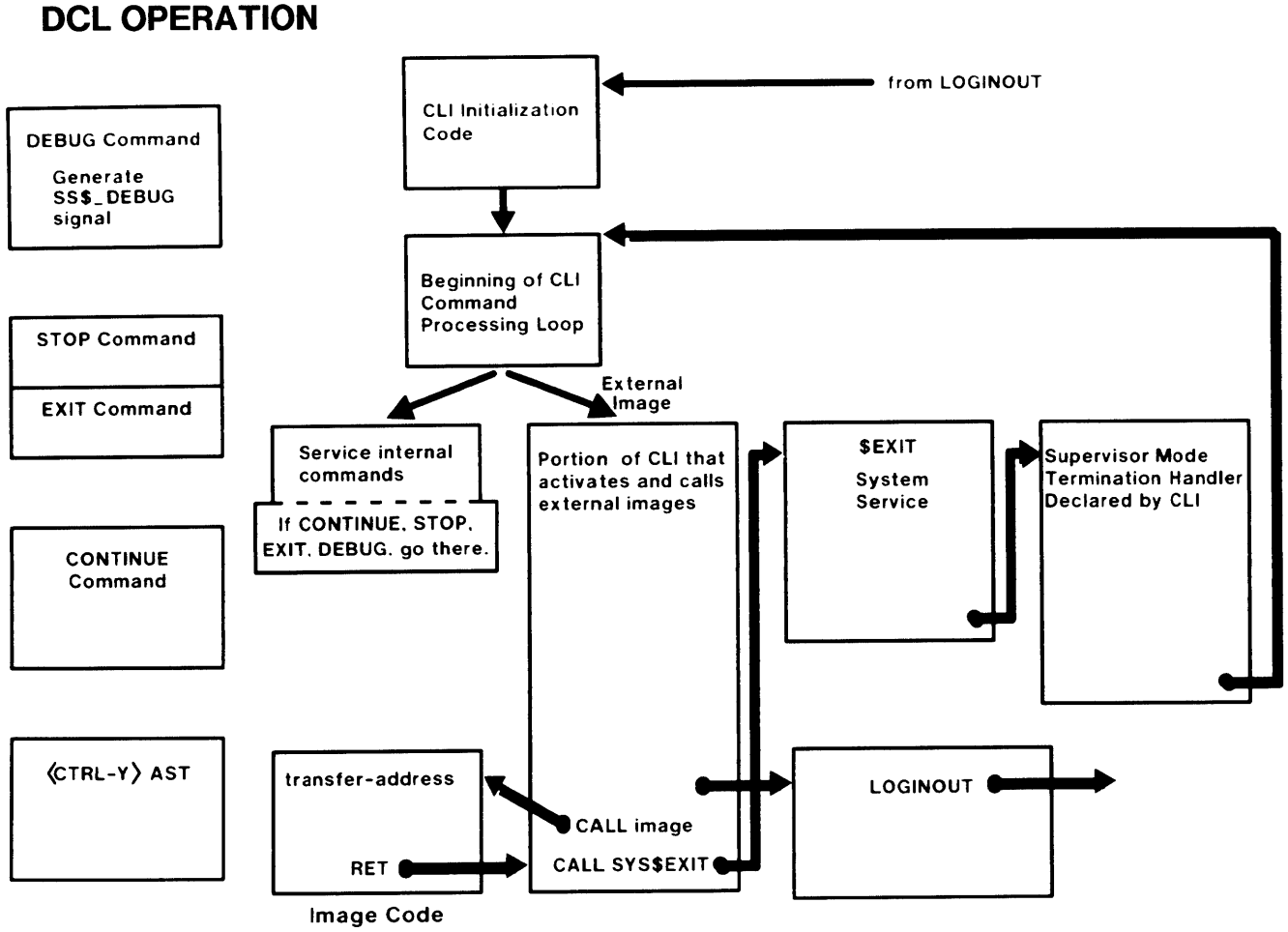


Figure 8-11 DCL Operation

- Glorified exit handler
- Main command loop
  - prompts for command
  - uses DCL tabes to decide
    - image
    - internal routine
- Command code
  - image, runs in PØ space
  - internal routine, runs in Pl space
- Control-Y AST
- $EXIT (image)
- LOGOUT (LOGINOUT.EXE)

STEPS IN IMAGE ACTIVATION AND TERMINATION

Table 8-5 Steps in Image Activation and Termination

Action	Code
1. Image activation	SYS$IMGACT
2. Image startup	SYS$IMGSTA
3. Image executes	
4. Image exit	SYS$EXIT

IMAGE ACTIVATION

SYS$IMGACT:

- System service, executive mode
- Called by DCL, INSTALL
- Sequence
  - Open image file
  - Read image header
  - Map image

IMAGE FILE MAPPED TO VIRTUAL ADDRESS SPACE

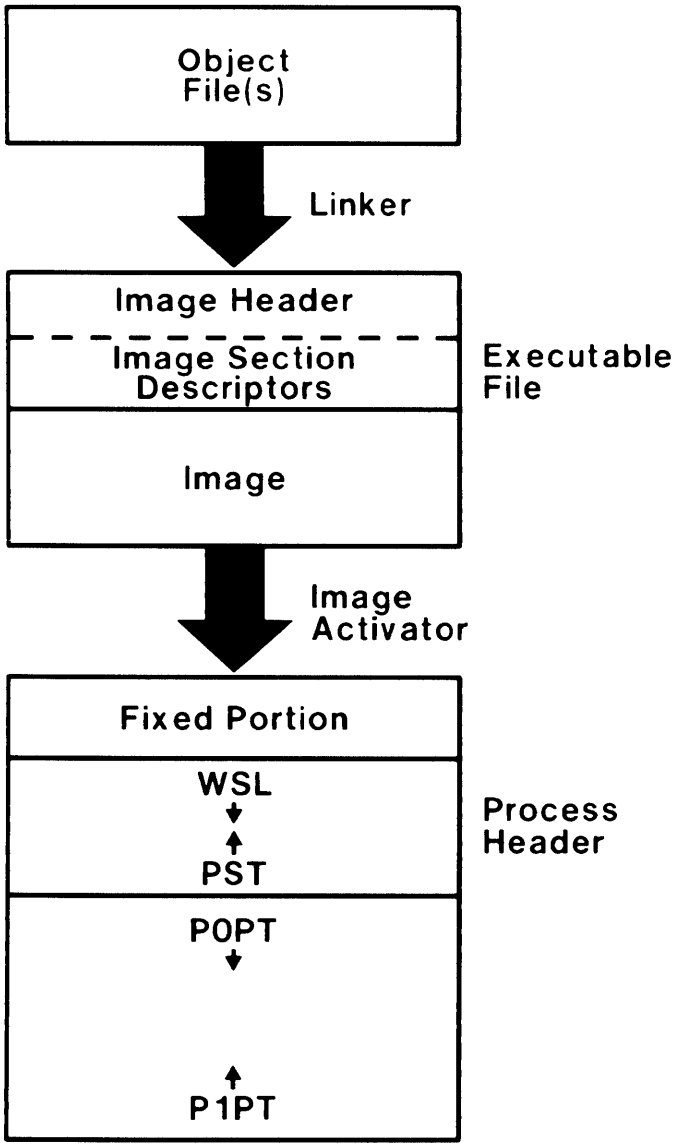


Figure 8-12 Image File Mapped to Virtual Address Space

IMAGE HEADER

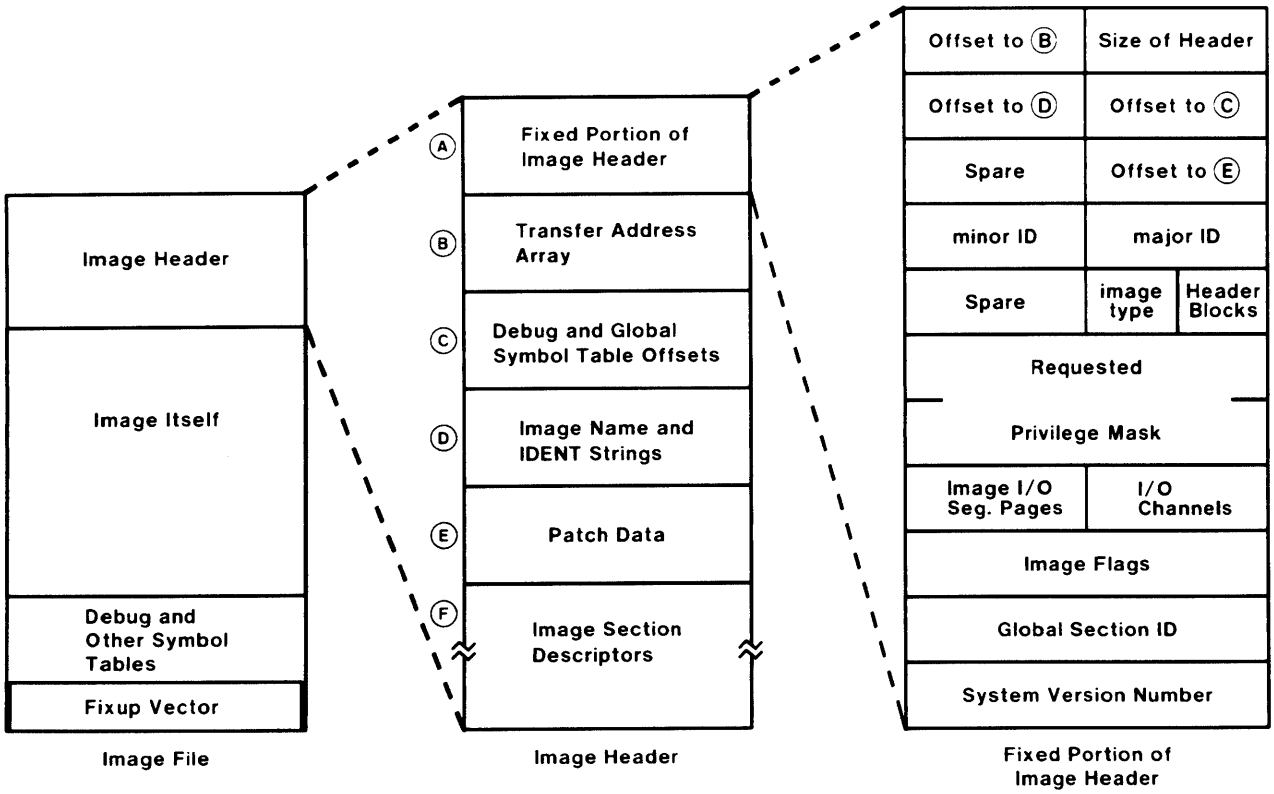


Figure 8-13 Image Header

- Image file contains
  - Image header
  - Image
  - Symbol tables
  - Fixup vector

IMAGE SECTION DESCRIPTOR

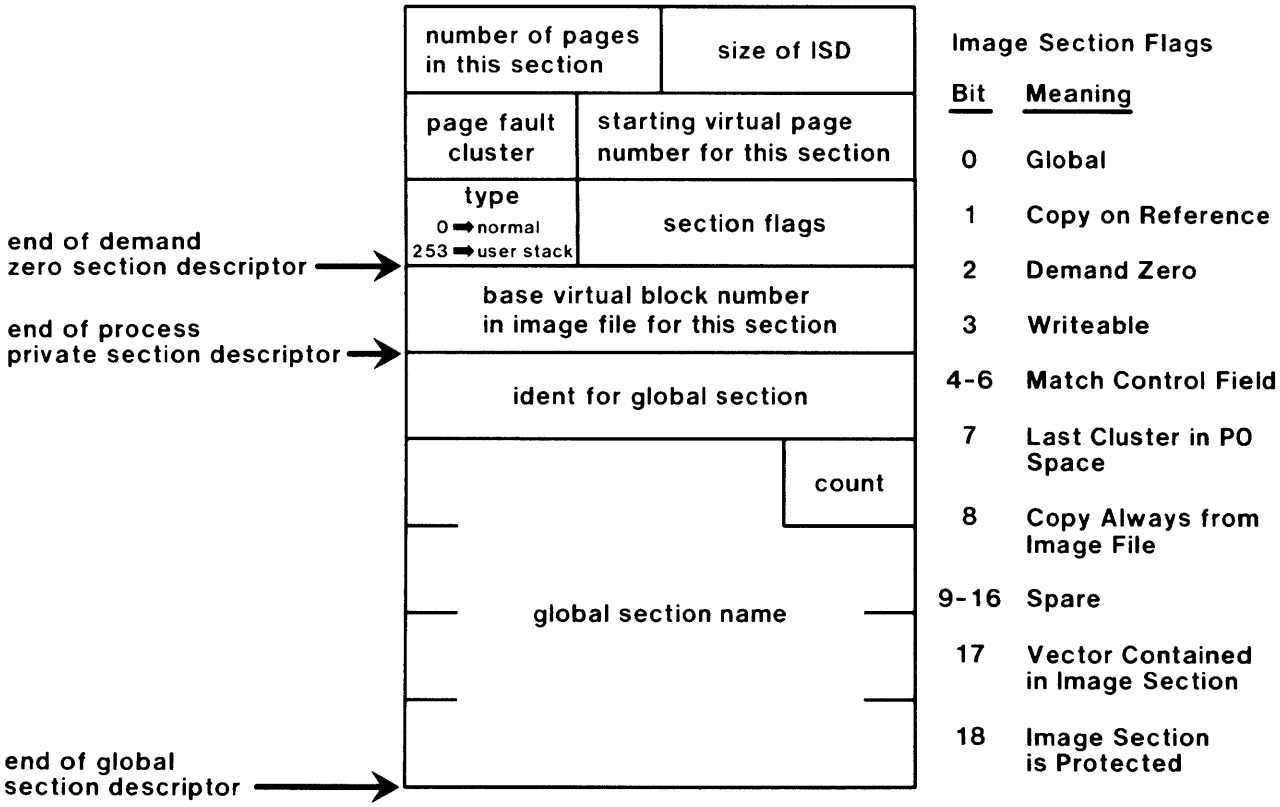


Figure 8-14 Image Section Descriptor

- Image section descriptors for
  - Demand zero sections
  - Process private sections
  - Global sections

KNOWN FILE ENTRY, HEADER

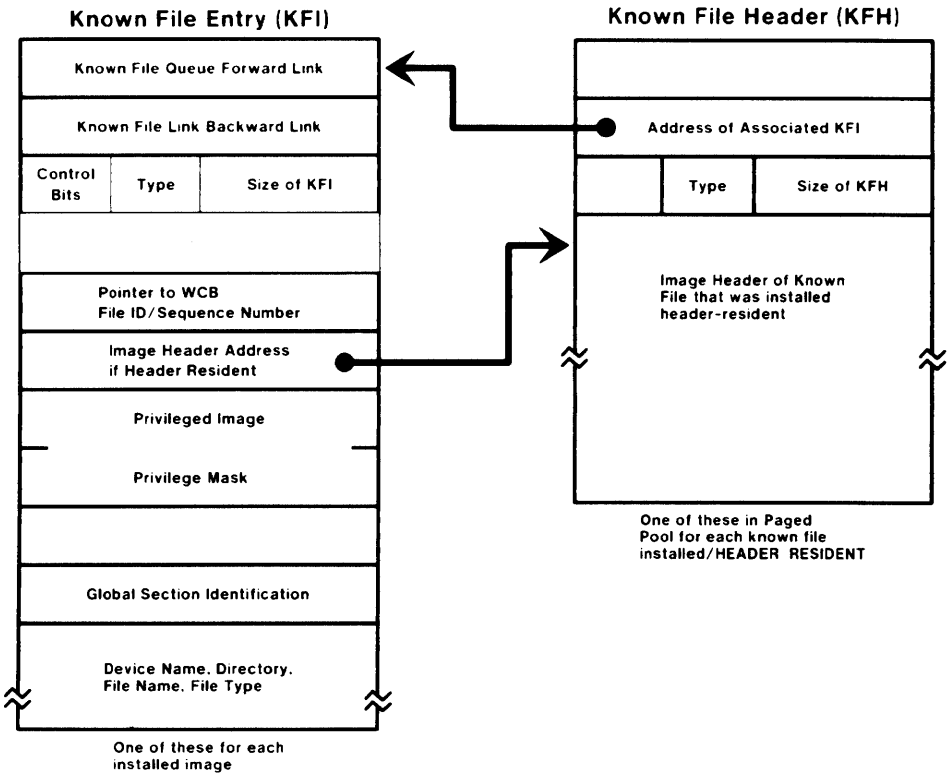


Figure 8-15 Known File Entry, Header

- Image file marked as known in image header
- Known file lists searched
- Created by INSTALL utility

*SYSGEN

KFILSTCNT

IMAGE START UP

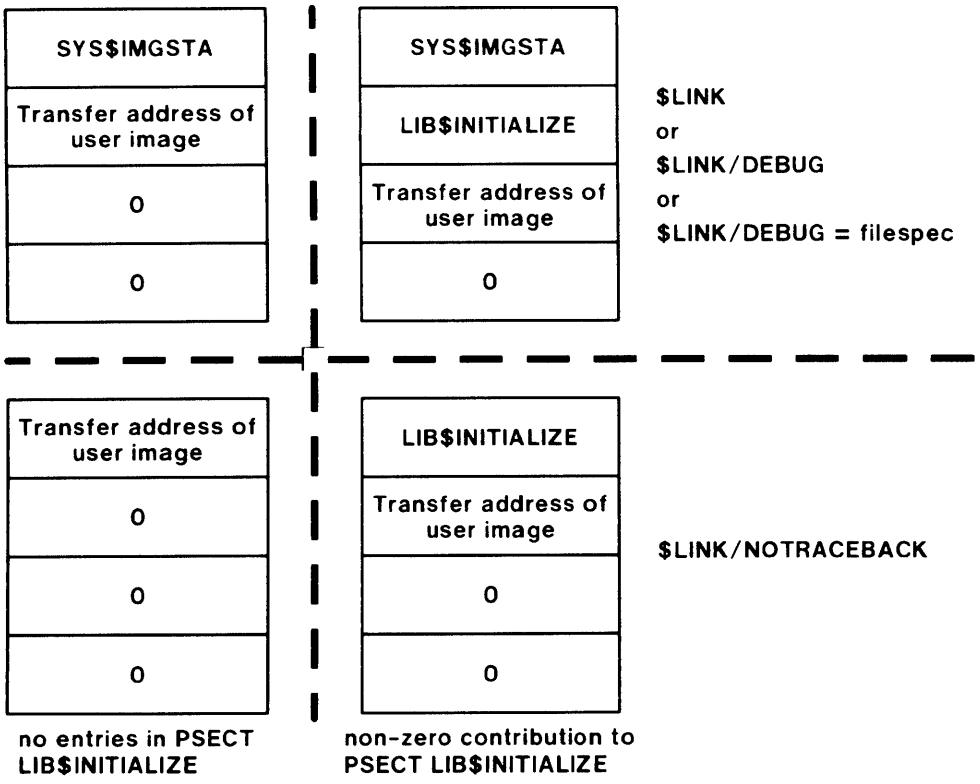


Figure 8-16 Image Startup

- SYS$IMGSTA
  - Functions
    Purge working set
    Map debugger
    Establish traceback handler
    Alter argument list - point to next transfer vector address
  - LIB$INITIALIZE
  - Transfer address obtained from image header.

EXIT SYSTEM SERVICE

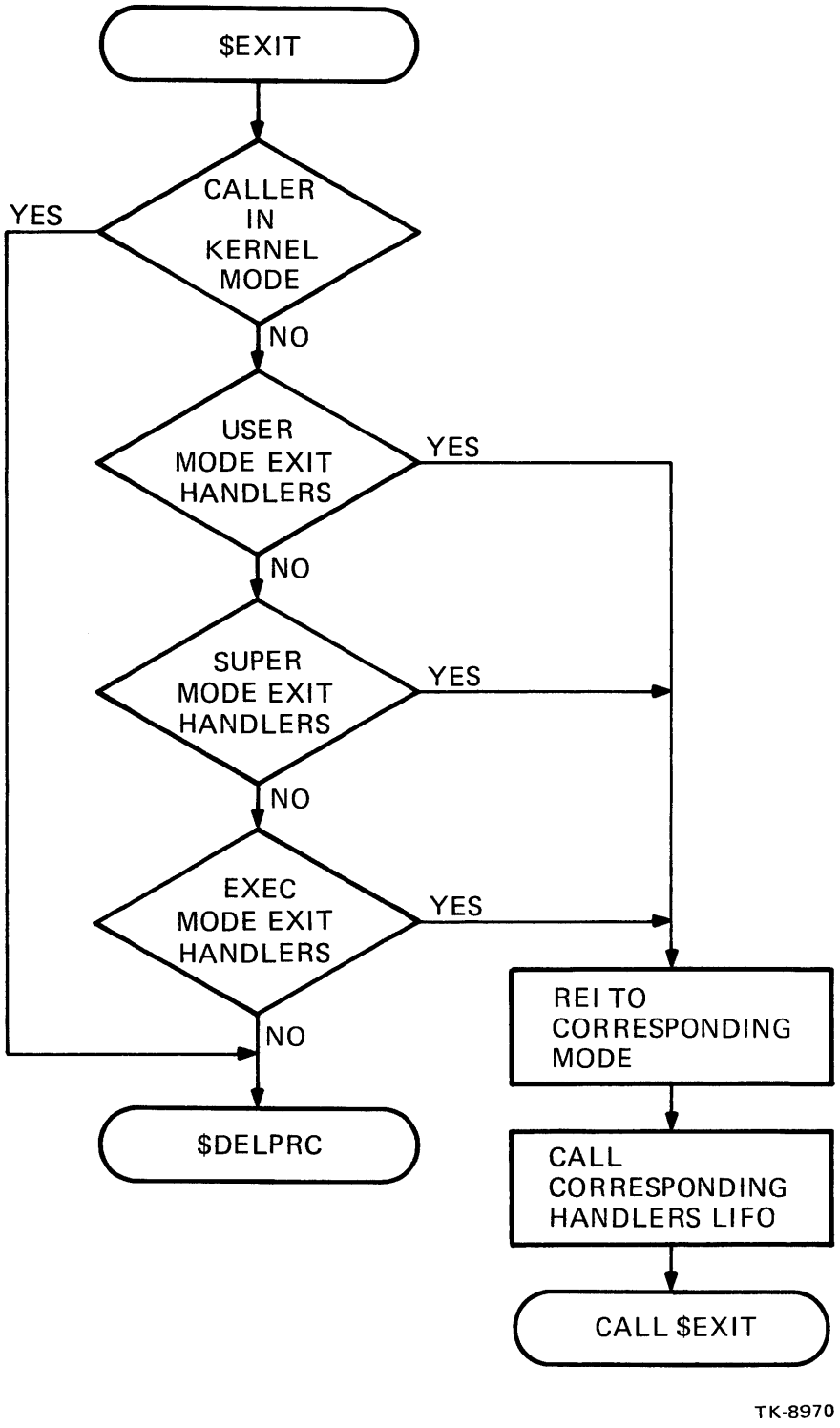


Figure 8-17 Exit System Service

TERMINATION HANDLERS

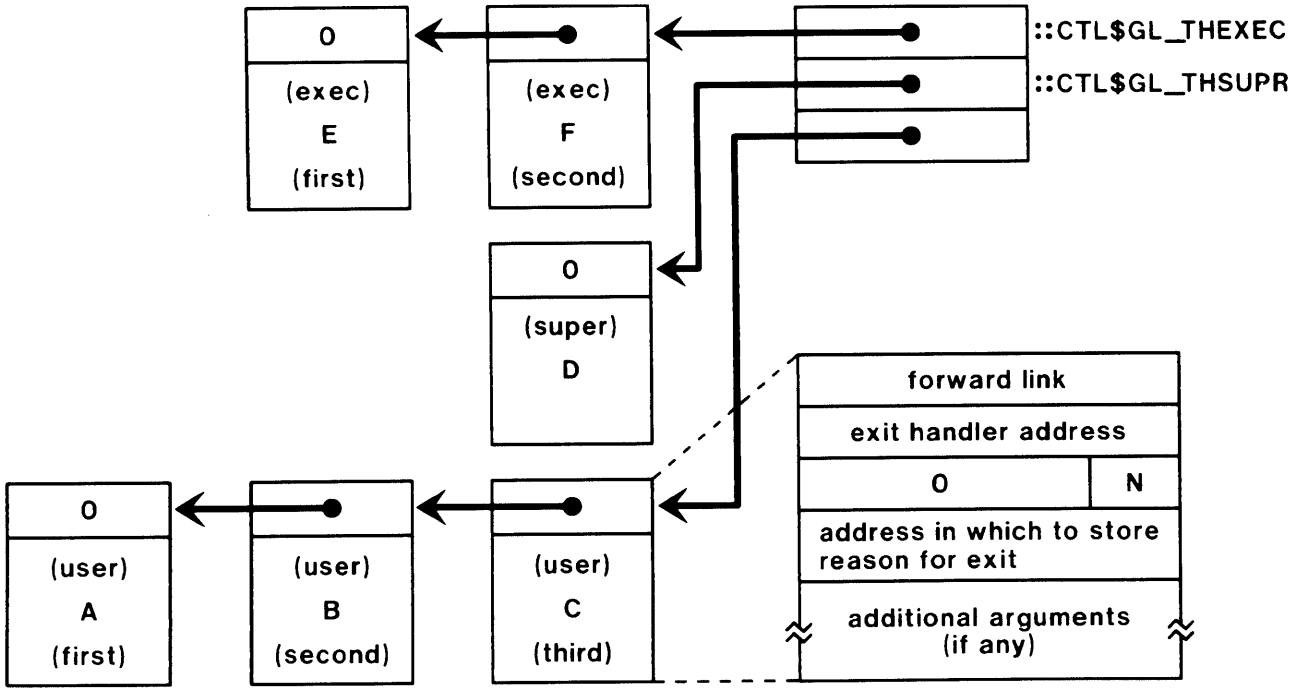


Figure 8-18 Termination Handlers

Table 8-6 How Termination Handlers Are Set Up For Different Access Modes

Mode	Set Up By
User	User image
Supervisor	DCL
Executive	PROCSTRT for RMS rundown
• Exit in kernel mode causes process deletion.	

SYSTEM INITIALIZATION AND SHUTDOWN

9

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 startup command procedure and the enabling of logins. Included is an explanation of:

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

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

OBJECTIVES

Upon completion of this module, you will be able to describe, in general terms the sequence of operations involved in:

1. initial bootstrap
2. powerfail and recovery
3. bugcheck and reinitialization

Differences between 780/750/730

Discuss the effects of altering SYSGEN parameters relating to system initialization.

RESOURCES

Reading

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

Source Modules

Facility Name	Module Name
BOOTS	SYSBOOT, SYSGEN
	VMB
SYS	INIT
	INILOA
	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. Startup command procedures

   E. SYSBOOT, SYSGEN

   F. 78Ø, 75Ø, 73Ø 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. Power-fail recovery

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

780 and 730

- contain a console microprocessor

  780 - LSI11
  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

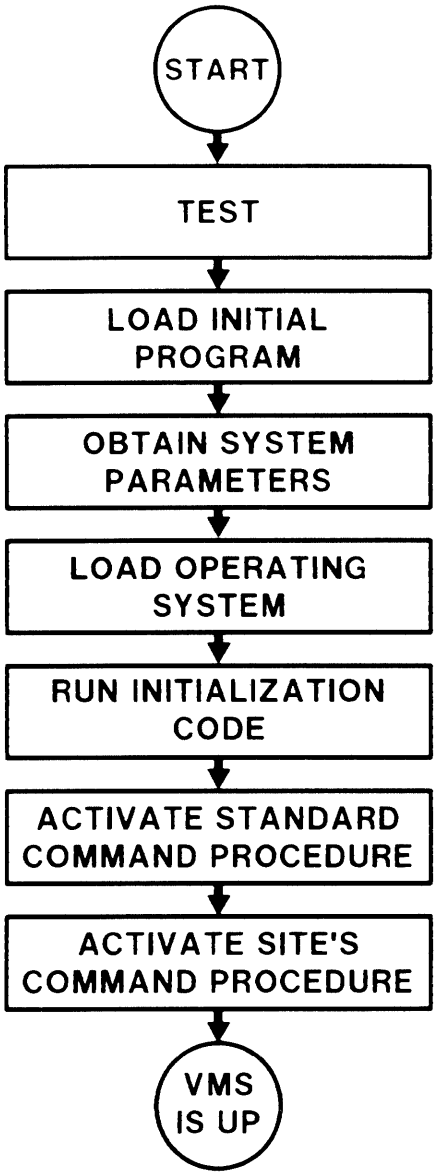


Figure 9-1 System Initialization

SYSTEM INITIALIZATION SEQUENCE

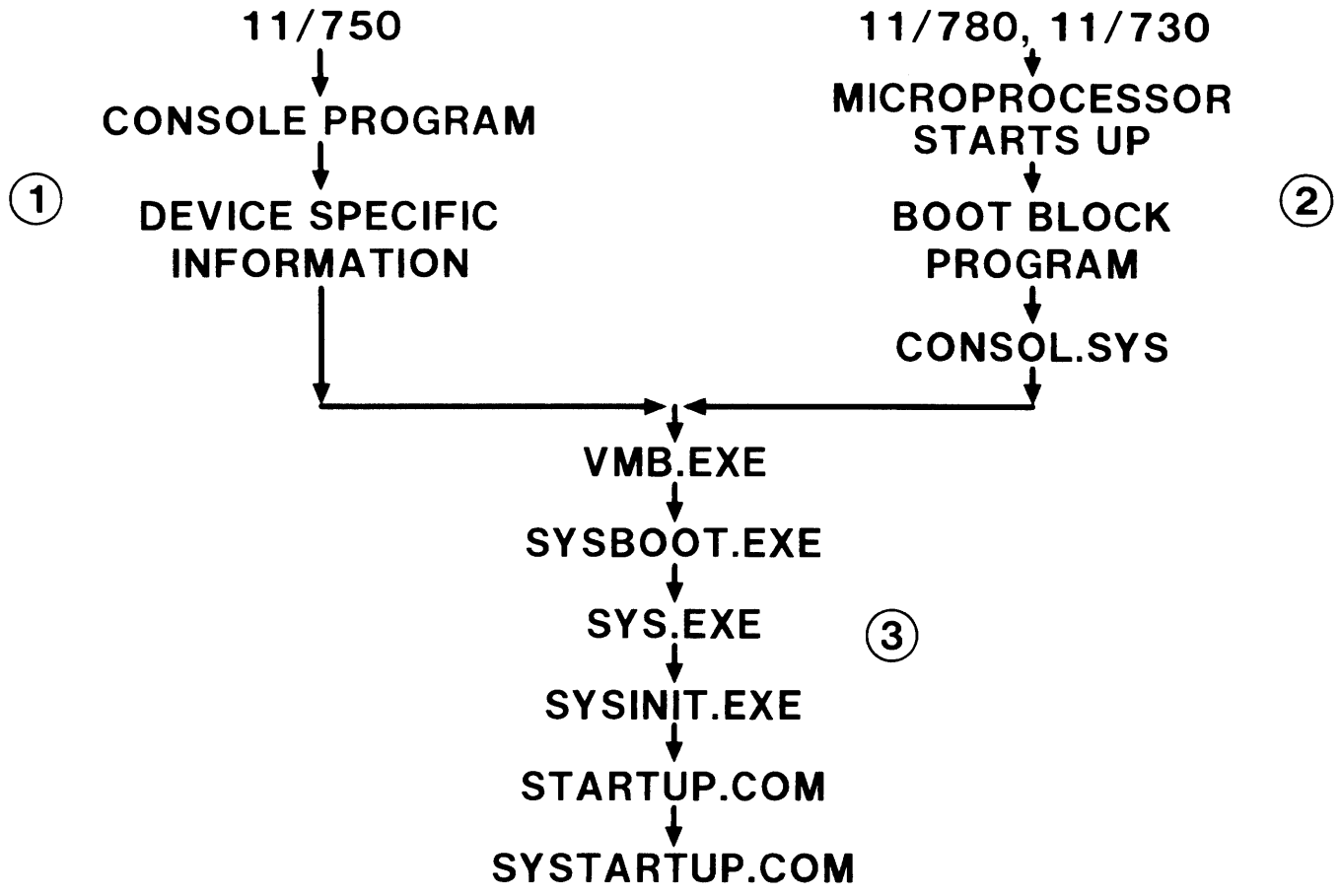


Figure 9-2 System Initialization Sequence

1. Bootstrap computer using ROMs in CPU.
2. Bootstrap computer using LSI-11 (78Ø) or 8Ø85 (73Ø)
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 9-1 Initialization Programs

Program	Function	Environment
CONSOLE.SYS (CONSOLE.EXE on 73Ø)	Loads VAX writeable diagnostic control store Acts as monitor for console terminal commands On boot command loads, passes control to VMB.EXE	LSI (78Ø) 8Ø85 (73Ø) CPU (75Ø)
VMB.EXE	Sizes physical memory, discovers external adapters Sets up primitive SCB Locates, loads, and passes control to SYSBOOT.EXE	VAX memory Physical address
SYSBOOT.EXE	Loads SYSBOOT parameters Sizes system space, sets up system page table Locates and loads SYS.EXE Sets up full SCB Maps nonpaged pool into high end of physical memory Sets up PØ 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 Loads terminal driver and system disk driver 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, files, swap files, and dump file Maps RMS and system message file as system sections Creates disk ACP process	Process

Table 9-1 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 RMSSHARE 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

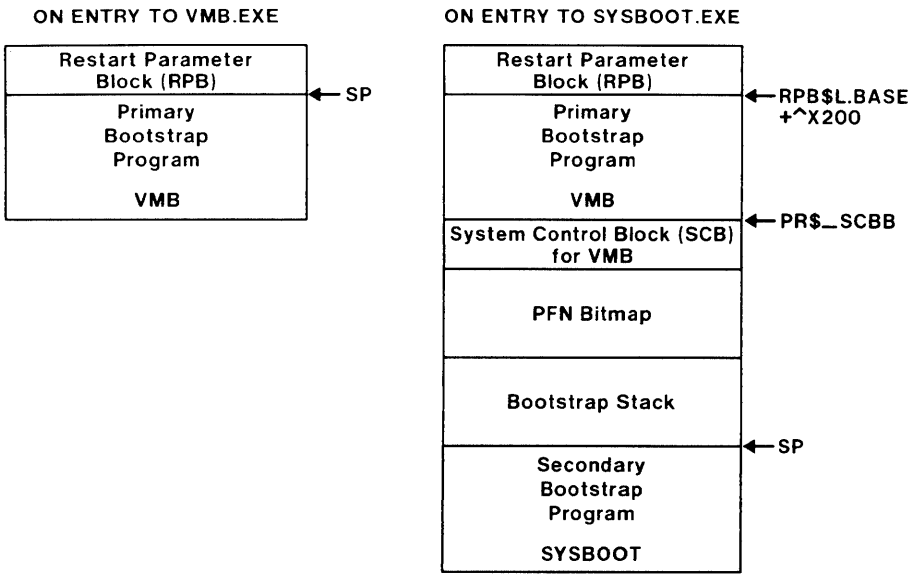


Figure 9-3 Physical Memory During Initialization

- 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

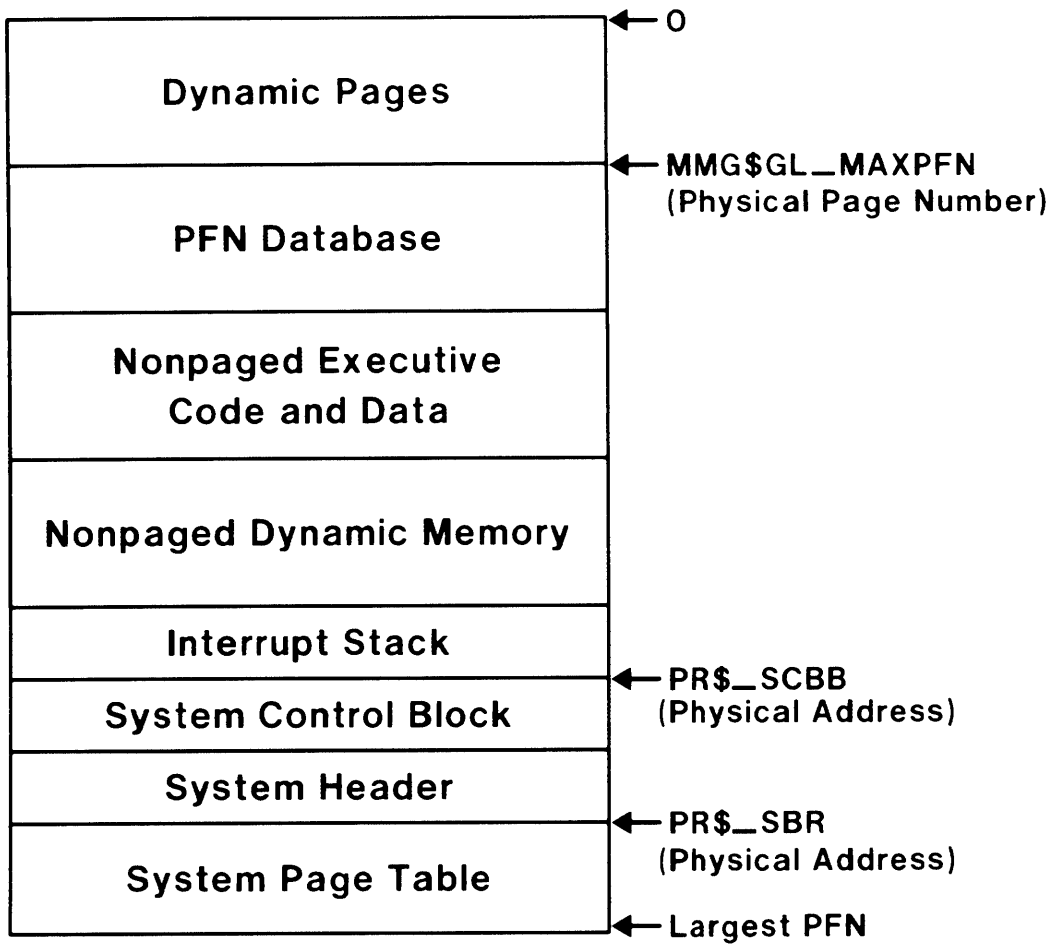


Figure 9-4 Physical Memory After SYSBOOT Ends

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

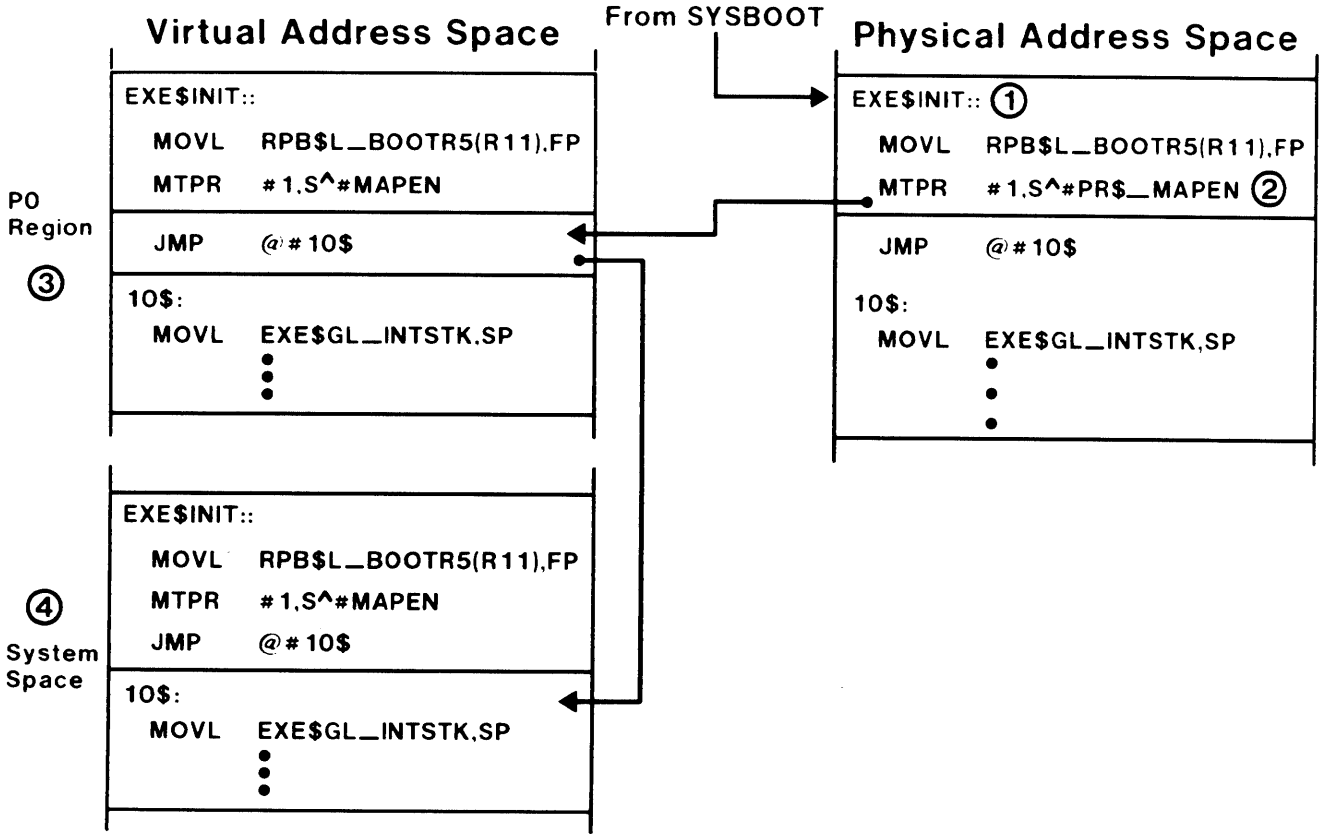


Figure 9-5 Turning on Memory Management

- Done by INIT in SYS.EXE
- Physical to virtual transition:

❶ - 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

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

❸ Next instruction is found in PØ space

❹ 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
  - Open and record locations of page, swap and dump files
  - Map RMS and system message files
  - Mount system disk, creating disk ACP
  - Create STARTUP process

STARTUP

Startup Process

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

STARTUP.COM

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

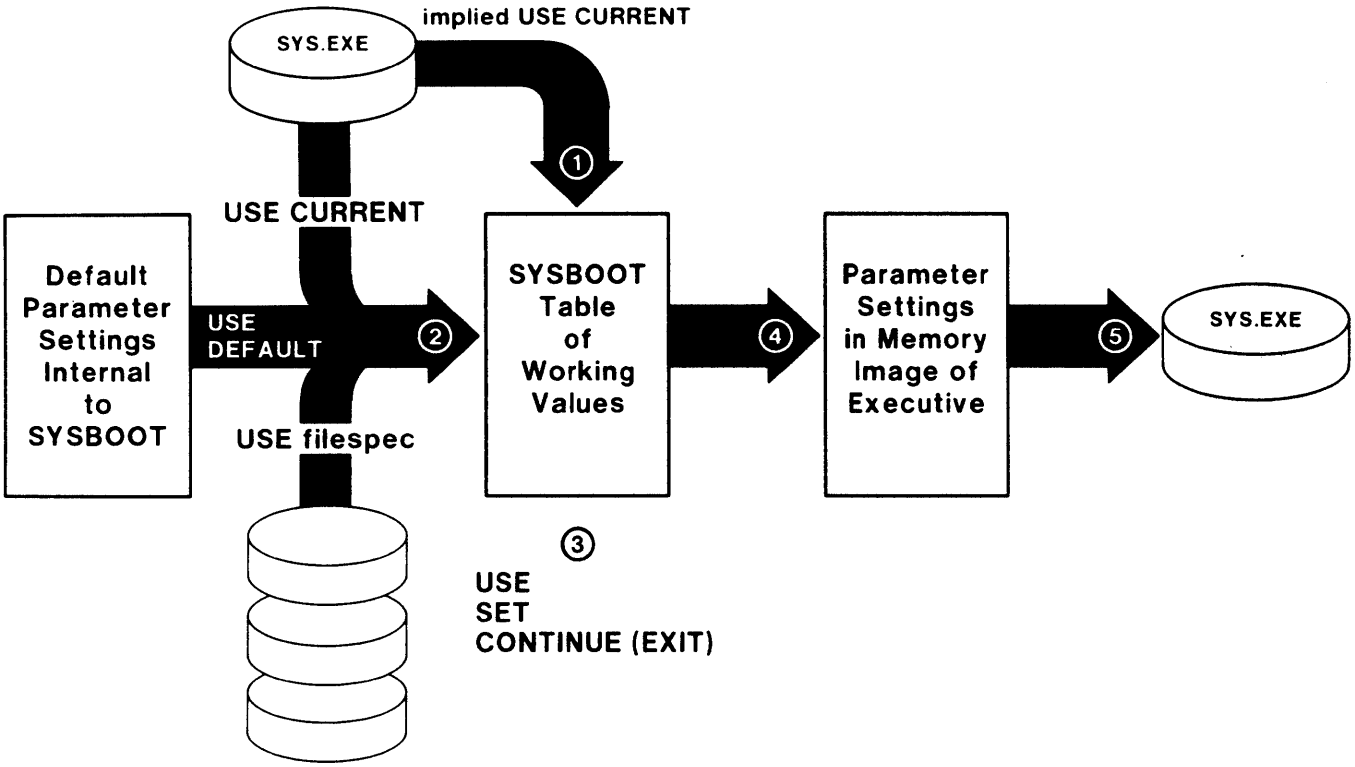


Figure 9-6 SYSBOOT and System Parameters

SYSBOOT:

- Brings in current parameters
- Allows changes if conversational boot requested
- Writes parameters to loaded SYS.EXE
- Runs as part of system initialization
- Can alter all parameters used in present system
- Cannot create alternate parameter files

SYSGEN AND SYSTEM PARAMETERS

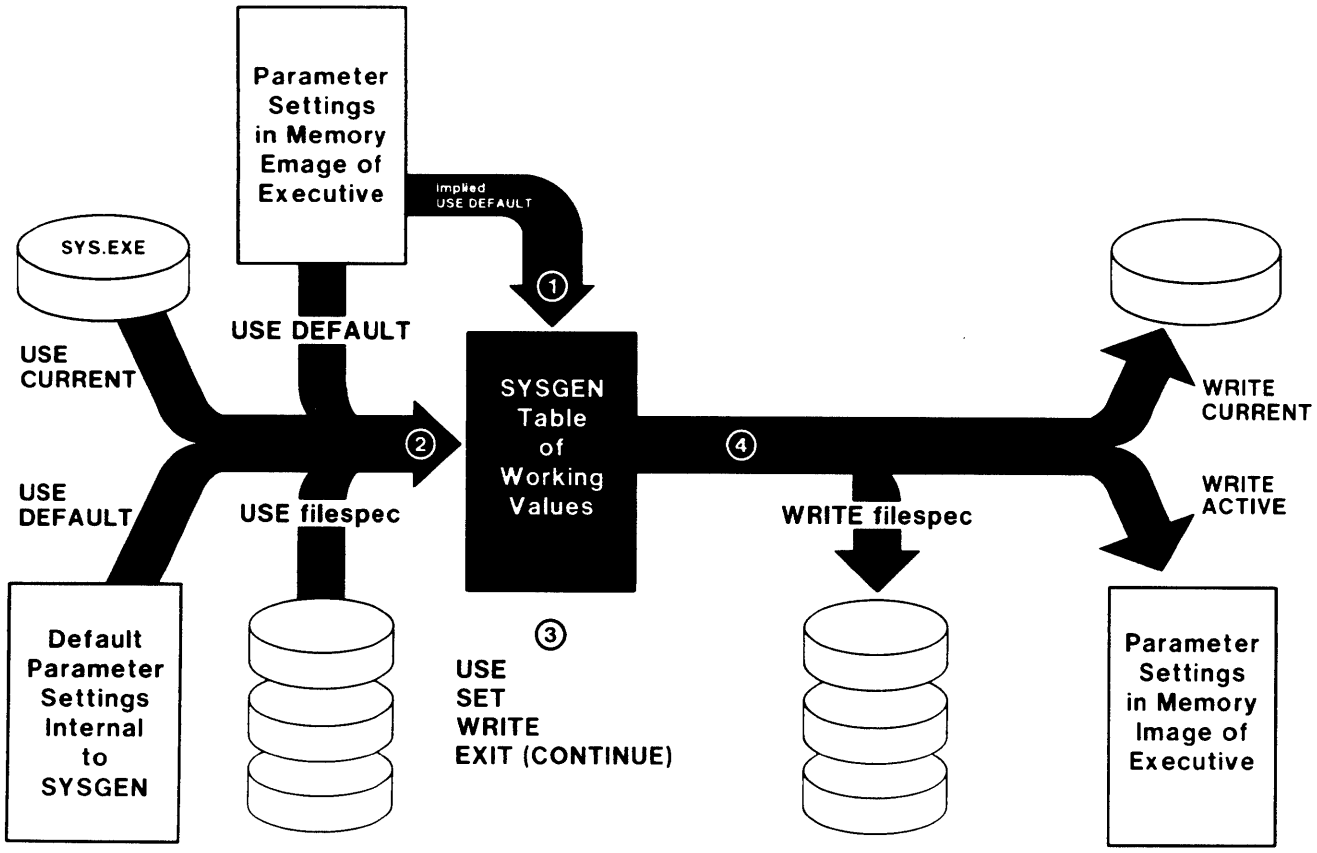


Figure 9-7 SYSGEN and System Parameters

SYSGEN:

- Runs as an editor-like utility under VMS
- Can alter dynamic parameters on present system
- Can create alternate parameter files
- Can alter parameters used on next system initialization

VAX-11/780 PROCESSOR

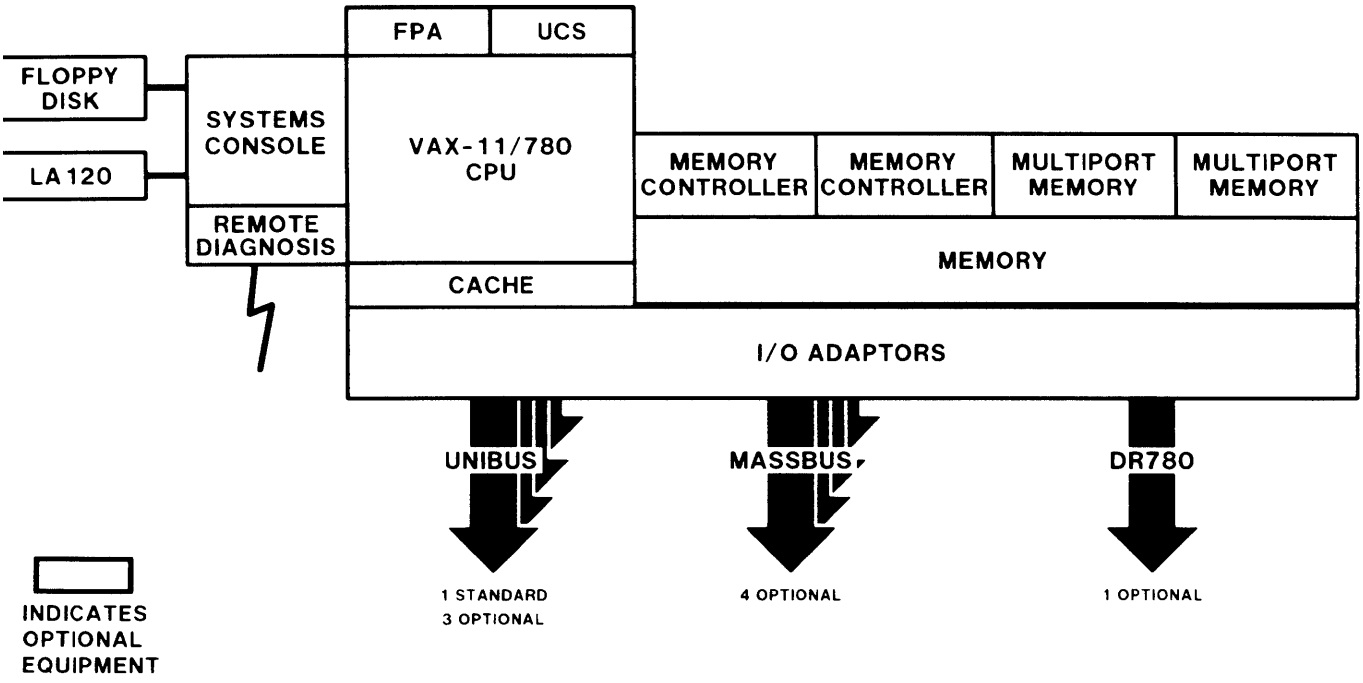


Figure 9-8 VAX-11/780 Processor

- 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

  cause ROM in memory controller to find 64K good bytes

  loads VMB.EXE from floppy disk to VAX memory

VAX-11/750 PROCESSOR

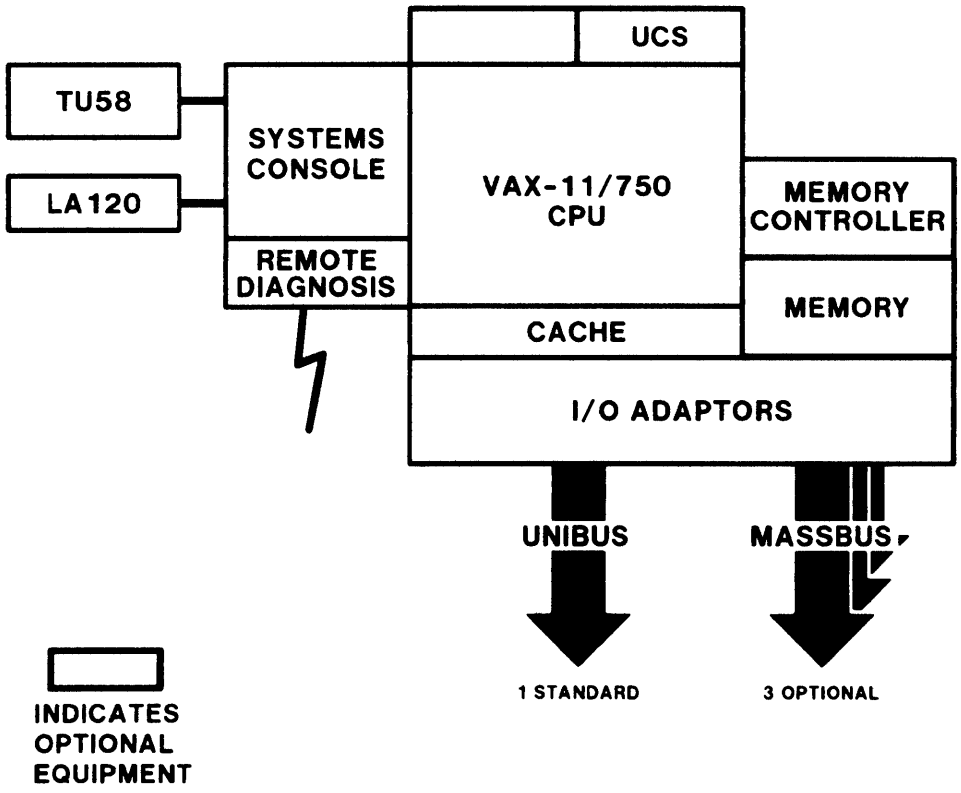


Figure 9-9 VAX-11/750 Processor

- 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

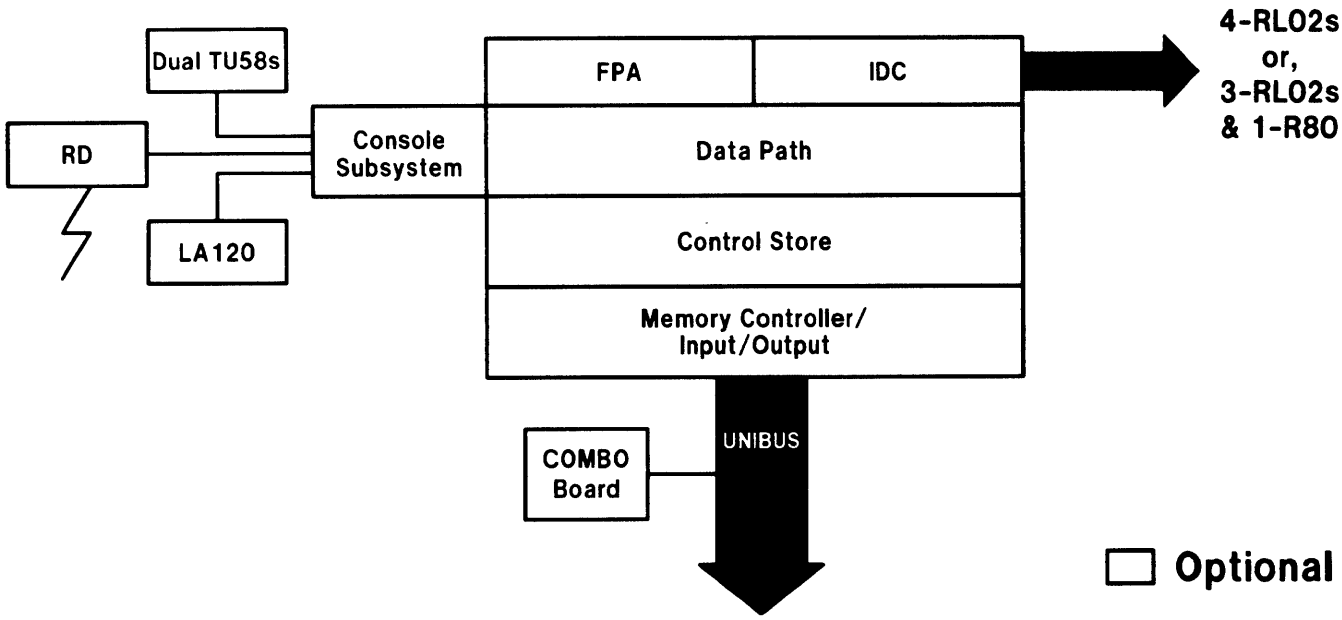


Figure 9-10 VAX-11/730 Processor

- Program on ROM causes CONSOLE.EXE to be loaded from TU58 into 8085 memory

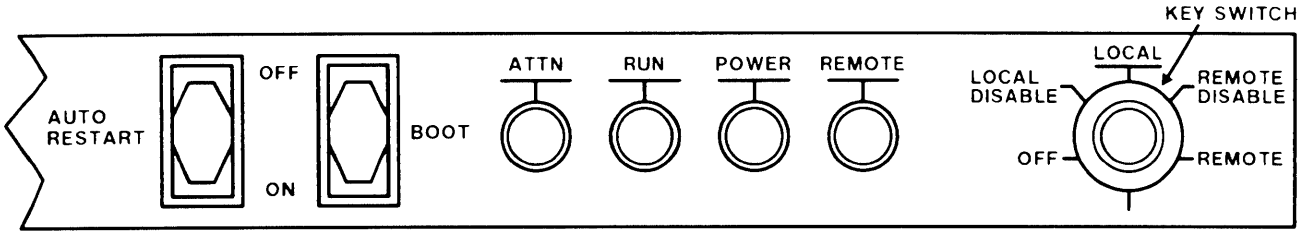
- CONSOLE.EXE runs on 8085

  loads microcode into CPU from TU58

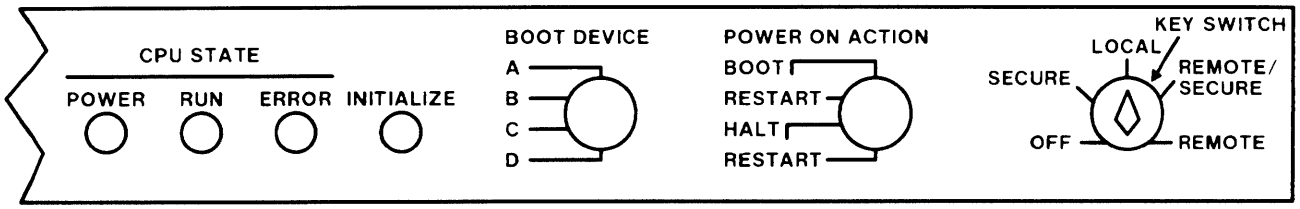
  executes DEFBOO - loads registers of CPU, finds 64K good bytes

  loads VMB.EXE from TU58

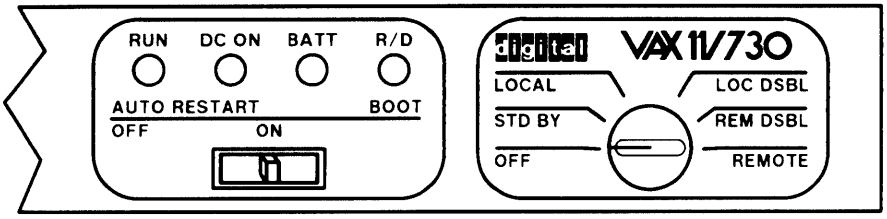
VAX FRONT PANELS



VAX 11/780 Front Panel



VAX 11/750 Front Panel



VAX 11/730 FRONT PANEL

Figure 9-11 VAX Front Panels

Table 9-2 Switches on 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 9-3 Shutdown Operations

Action	Operation
Clean shutdown	$ @SYS$SYSROOT:[SYSEXE]SHUTDOWN
Quick shutdown	$ RUN SYS$SYSTEM:OPCCRASH
Forced crash	Control/P (on OPA0:) >>>@CRASH (780/730 only) >>>E P (750 only) >>>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
Halt system	Control/P (on OPA0:) >>>H (780/730 only)

SHUTDOWN PROCEDURES

Table 9-4 Shutdown Procedures

Procedure	Function
SHUTDOWN.COM	- Warns users of shutdown - Stops queues - Removes installed images - Stops processes - 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 1F000 in PSL - Continues

AUTORESTARTING THE SYSTEM

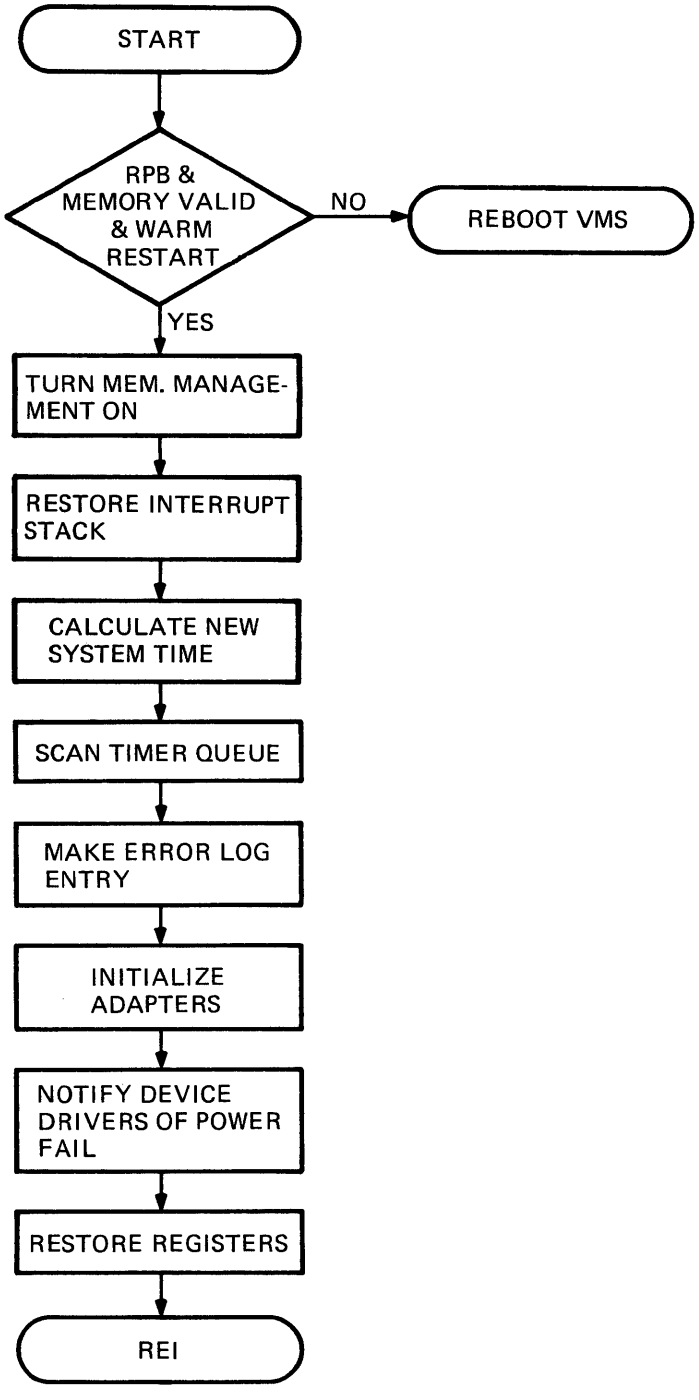


Figure 9-12 Autorestarting the System

REQUIREMENTS FOR RECOVERY AFTER POWER-FAIL

- Battery backup
- Memory Valid (battery not run down)
- RPB & 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

APPENDIX

VAX-11/782

Definitions

Loosely Coupled:
: Each processor executes a separate copy of the operating system. This is good for high-availability.

Tightly Coupled:
: Both processors share the same copy of the operating system.

Symmetric:
: All processors execute all the operating system code.

Assymmetric:
: All processors cannot execute all the operating system code.

Primary Processor:
: The CPU that executes Kernel-mode code as well as Executive, Supervisor, and User.

Secondary Processor:
: The CPU that cannot execute Kernel-mode code. It executes Executive, Supervisor, and User mode code.

The VAX-11/782 is a tightly-coupled multi-processing system that is assymmetric for kernel-mode and symmetric for the other modes.

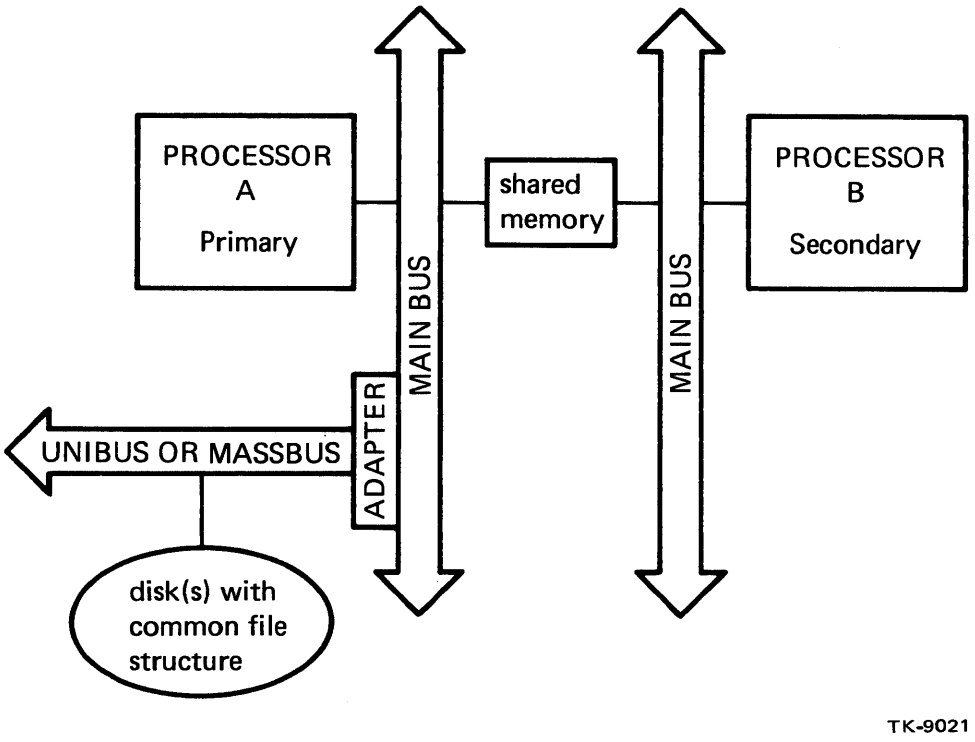


Figure 9-13 Sample VAX-11/782 Configuration

- Two VAX-11/780s connected to the same shared memory.
- The primary (on the left) has the I/O devices. The secondary or attached processor (on the right) has just the CPU.
- Minimum local physical memory (256Kb) on each CPU for diagnostics only.
- All information in the shared memory.
- Eight Meg maximum physical memory for the shared memory.
- Primary Processor runs all interrupt and Kernel-mode code. Both processors run Executive, supervisor, and user mode code.
- Multi-processing code takes approximately 8K bytes (16 pages) in non-paged pool.

1. Initialization

   a. Start primary processor.

      The DEFBOO.CMD file used to boot the primary processor "request" that the MA78Ø memory be used instead of the local physical memory. The memory on MA78Ø #1 starts at physical address Ø.

   b. After the normal system is booted, a privileged program (MP.EXE) is run in the site-specific command file. MP.EXE is activated by the START/CPU DCL command. MP.EXE does the following:

      - Allocates nonpaged pool and loads in the MP code.

      - Connects the 'hooks' into the VMS code (discussed later).

      - New SCB initialized for the secondary CPU.

      - Primary SCB slightly modified to handle

      • scheduling code for secondary processor
      • MA78Ø interrupt communication

   c. Start Secondary processor. Accomplished by booting with an abbreviated command file in CSA1. This results in:

      - initialization of memory configuration
      - start executing at address in RPB.

2. Hooks into VMS

   Naming Conventions

   - MPH$samename

     indicates a routine that will be entirely replaced by a MP routine of the same name.

   - MPH$newnameHK

     indicates a location of a hook to additional MP code (instead of a replacement).

   - MPH$newnameCONT

     indicates a location where additional MP code will return to normal flow of code.

     Locations of Hoods (Executive Module Names)

     - ASTDEL AST delivery and queueing
     - BUGCHECK BUGCHECK for both processors
     - PAGEFAULT Translation buffer invalidations
     - SCHED Process scheduling and rescheduling

3. SCB changes

   - CPU2

     New SCB created for the secondary processor in nonpaged pool. This SCB points to different routines than those used by the primary CPU.

   - CPU1

     - MA78Ø vectors redirected to point to new MP primary CPU interrupt routine.

     - IPL=5 SCB interrupt vector now contains address of MP secondary scheduling routine.

     - XDELTA interrupt is moved from IPL=5 to IPL=F.

4. Secondary Processor States

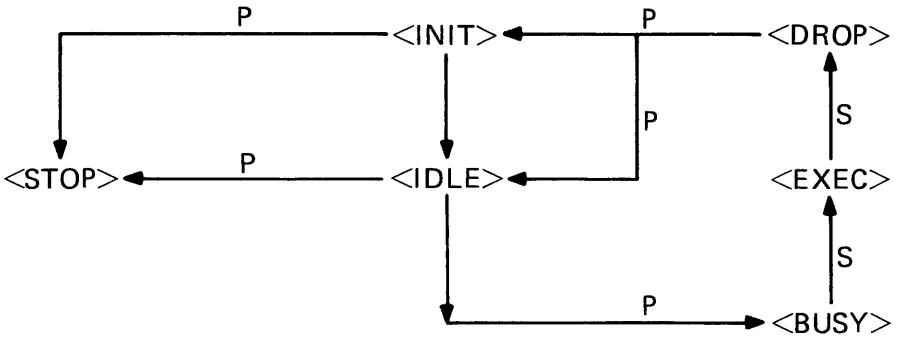


Figure 9-14 Secondary Processor States

The current state of the secondary processor is recorded in the state variable in nonpaged pool. The contents of this variable (the state) is used to by the primary processor to determine whether to schedule word for the secondary processor or not.

<INIT>	Processor state when MP.EXE runs.
v	
<IDLE>	After MP.EXE (initialization code runs)
v	
<BUSY>	when a process is found for CPU2
v	
<EXECUTE>	After a LDPCTX instruction is issued
v	
<DROP>	At quantum end or kernel mode request by CPU2, a SVPCTX issued, the state changed to DROP, interrupt primary.
v	
<IDLE>	After CPU1 takes back process
<STOP>	Requested by system manager ($STOP/CPU) requested by CPU1

If there is not process for CPU2 to run, an AST is used to indicate when a process falls below the kernel mode level. The AST delivery is turned into a rescheduling interrupt.

5. Exceptions for CPU2

   If there is a transition to Kernel mode:

   A SVPCTX is issued
   Process is "handed" to CPU1

   AST Delivery and quantum End both execute have special code.

   A separate SCB for the secondary processor allows the enforcement of the rules.

6. MA78Ø

   Has the ability to interrupt either processor.

   Reasons for CPU1 to interrupt CPU2

   1. To request an invalidation of a System Virtual Address.

   2. AST has arrived for process on CPU2

   3. BUGCHECK

   Reasons for CPU2 to interrupt CPU1

   1. To request rescheduling

   2. Log an error

   3. Request a BUGCHECK

7. Faults

- POWERFAIL

  If CPU2 goes, CPU1 continues
  If CPU1 goes, CPU2 waits

- BUGCHECK

  If a BUGCHECK occurs, CPU2 goes IDLE while CPU1 writes the sysdump file and reboots.

- MACHINE CHECK

  Like normal VMS

8. Restrictions

- The processors must be "twins" (ECO,WCS level, FPA)
- First MA78Ø must be at physical address Ø
- Same TR # for the MA78Øs on both CPUs
- No DR78Ø
- No high speed RPØ7s (2.2mb), 1.3mb allowed

NOTE

Before MP.EXE runs, the RPB contains a self-jump loop. After MP.EXE runs, RPB contains the address of the secondary CPU startup code. In this way the secondary CPU (CPU2) can be started before the primary CPU (CPU1) is finished booting.

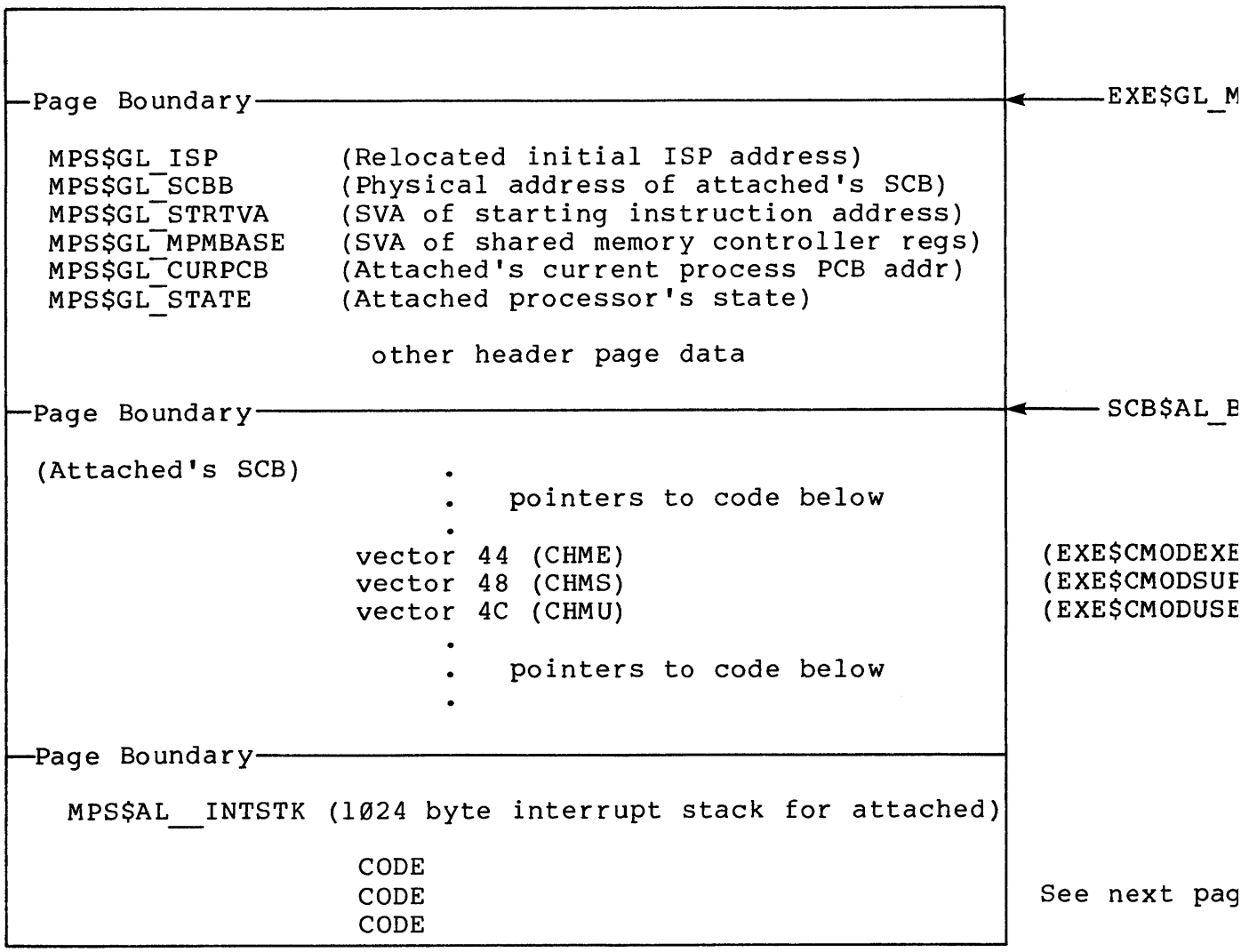


Figure 9-15 MP.EXE Loaded into Nonpaged Pool

CODE Section from diagram on previous page contains:

1. Addresses being pointed to by almost all of the SCB vectors

2. System locations that are jumped to as a result of being modified by the MP.EXE code. See Table 9-5.

Table 9-5 System Locations and the Resulting MP Locations

System Locations	MP Locations
SCH$SCHED	SCH$MSCHED
SCH$RESCHED	SCH$MRESCHED
MPH$QAST	MPS$QAST
MMG$INVALIDATE	MPS$INVALID
MPH$BUGCHKH	MPS$BUGVHECK
MPH$ASTDELHK	MPS$ASTSCHEDCHK
MPH$NEWLVLHK	MPS$ASTNEWLVL