CSC 553 Operating Systems

Lecture 3- Process Description and Control

Let's Review

- A computer platform consists of a collection of hardware resources
- Computer applications are developed to perform some task
- It is inefficient for applications to be written directly for a given hardware platform

Let’s Review

- The OS was developed to provide a convenient, feature-rich, secure, and consistent interface for applications to use
- We can think of the OS as providing a uniform, abstract representation of resources that can be requested and accessed by applications

OS Management of Application Execution

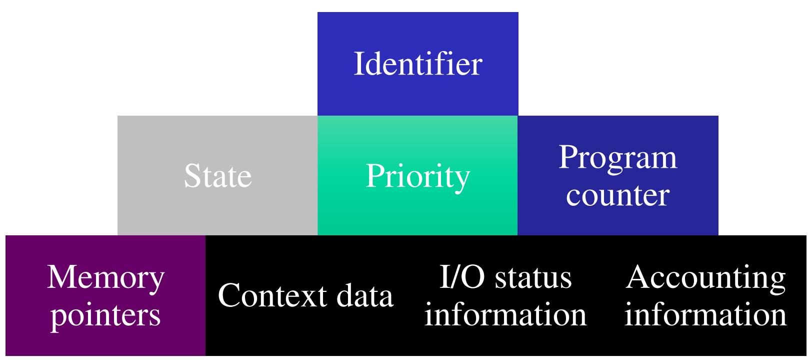
- Resources are made available to multiple applications
- The processor is switched among multiple applications so all will appear to be progressing
- The processor and I/O devices can be used efficiently

Process Elements

- Two essential elements of a process are:
  - Program code
    - which may be shared with other processes that are executing the same program
  - A set of data associated with that code
    - when the processor begins to execute the program code, we refer to this executing entity as a ***process***

Process Elements

- While the program is executing, this process can be uniquely characterized by a number of elements, including:



Process Control Block

- Contains the process elements
- It is possible to interrupt a running process and later resume execution as if the interruption had not occurred
- Created and managed by the operating system
- Key tool that allows support for multiple processes

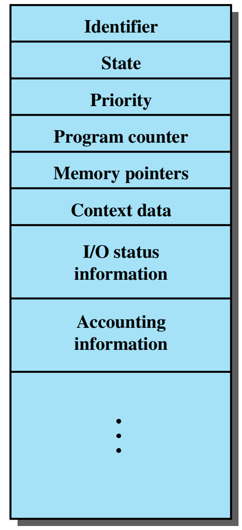


Figure 3.1 Simplified Process Control Block

Process States

Trace

- The behavior of an individual process by listing the sequence of instructions that execute for that process
- The behavior of the processor can be characterized by showing how the traces of the various processes are interleaved

Dispatcher

- Small program that switches the processor from one process to another

Process Execution

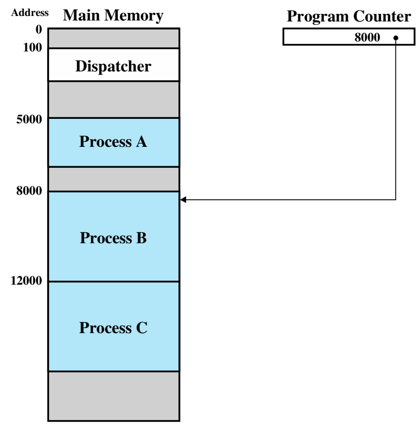


Figure 3.2 Snapshot of Example Execution (Figure 3.4) at Instruction Cycle 13

(a) Trace of Process A	(b) Trace of Process B	(c) Trace of Process C
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

5000 = Starting address of program of Process A
8000 = Starting address of program of Process B
12000 = Starting address of program of Process C

Figure 3.3 Traces of Processes of Figure 3.2

1	5000
2	5001
3	5002
4	5003
5	5004
6	5005
-------------------- Timeout	
7	100
8	101
9	102
10	103
11	104
12	105
13	8000
14	8001
15	8002
16	8003
----------------I/O Request	
17	100
18	101
19	102
20	103
21	104
22	105
23	12000
24	12001
25	12002
26	12003

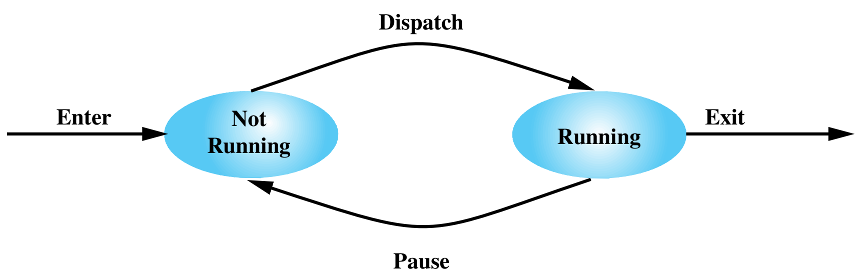
27	12004
28	12005
-------------------- Timeout	
29	100
30	101
31	102
32	103
33	104
34	105
35	5006
36	5007
37	5008
38	5009
39	5010
40	5011
-------------------- Timeout	
41	100
42	101
43	102
44	103
45	104
46	105
47	12006
48	12007
49	12008
50	12009
51	12010
52	12011
-------------------- Timeout	

100 = Starting address of dispatcher program

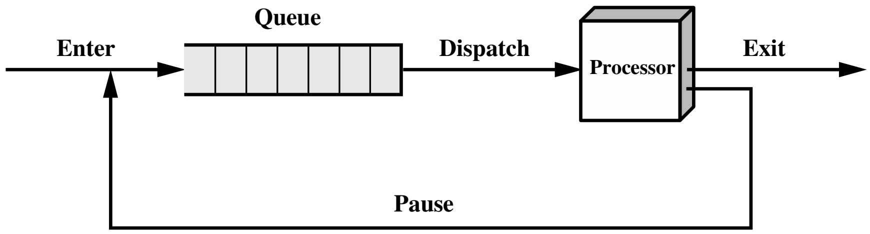
Shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

Figure 3.4 Combined Trace of Processes of Figure 3.2

Two-State Process Model



(a) State transition diagram



(b) Queuing diagram

Figure 3.5 Two-State Process Model

Reasons for Process Creation

New batch job	The OS is provided with a batch job control stream, usually on tape or disk. When the OS is prepared to take on new work, it will read the next sequence of job control commands.
Interactive logon	A user at a terminal logs on to the system.
Created by OS to provide a service	The OS can create a process to perform a function on behalf of a user program, without the user having to wait (e.g., a process to control printing).
Spawned by existing process	For purposes of modularity or to exploit parallelism, a user program can dictate the creation of a number of processes.

Process Creation

Process spawning

- When the OS creates a process at the explicit request of another process

Parent process

- Is the original, creating, process

Child process

- Is the new process

Process Termination

- There must be a means for a process to indicate its completion
- A batch job should include a HALT instruction or an explicit OS service call for termination
- For an interactive application, the action of the user will indicate when the process is completed (e.g. log off, quitting an application)

Reasons for Process Termination

Normal completion	The process executes an OS service call to indicate that it has completed running.
Time limit exceeded	The process has run longer than the specified total time limit. There are a number of possibilities for the type of time that is measured. These include total elapsed time ("wall clock time"), amount of time spent executing, and, in the case of an interactive process, the amount of time since the user last provided any input.
Memory unavailable	The process requires more memory than the system can provide.
Bounds violation	The process tries to access a memory location that it is not allowed to access.
Protection error	The process attempts to use a resource such as a file that it is not allowed to use, or it tries to use it in an improper fashion, such as writing to a read-only file.
Arithmetic error	The process tries a prohibited computation, such as division by zero, or tries to store numbers larger than the hardware can accommodate.
Time overrun	The process has waited longer than a specified maximum for a certain event to occur.
I/O failure	An error occurs during input or output, such as inability to find a file, failure to read or write after a specified maximum number of tries (when, for example, a defective area is encountered on a tape), or invalid operation (such as reading from the line printer).
Invalid instruction	The process attempts to execute a nonexistent instruction (often a result of branching into a data area and attempting to execute the data).
Privileged instruction	The process attempts to use an instruction reserved for the operating system.
Data misuse	A piece of data is of the wrong type or is not initialized.
Operator or OS intervention	For some reason, the operator or the operating system has terminated the process (e.g., if a deadlock exists).
Parent termination	When a parent terminates, the operating system may automatically terminate all of the offspring of that parent.
Parent request	A parent process typically has the authority to terminate any of its offspring.

Five-State Process Model

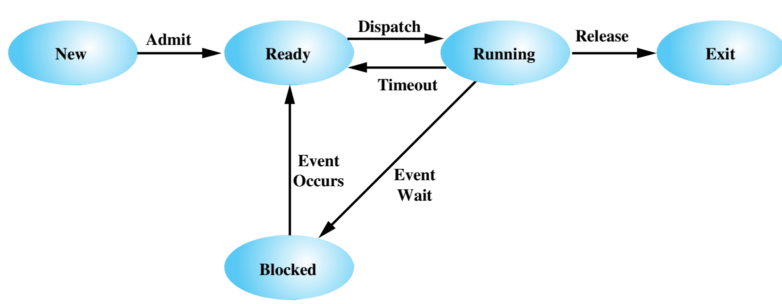


Figure 3.6 Five-State Process Model

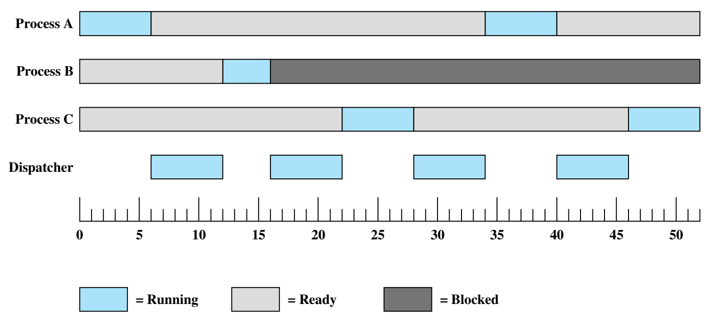


Figure 3.7 Process States for Trace of Figure 3.4

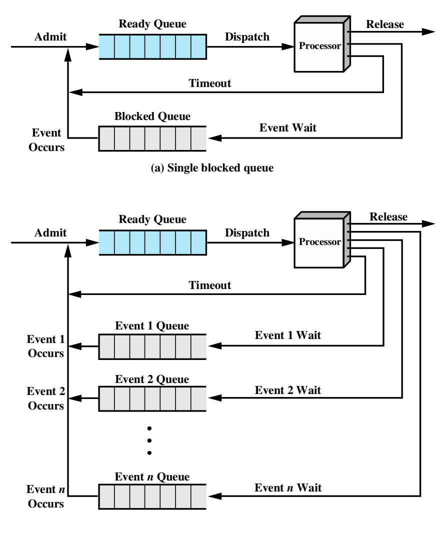


Figure 3.8 Queuing Model for Figure 3.6

Suspended Processes

- Swapping
  - Involves moving part of all of a process from main memory to disk

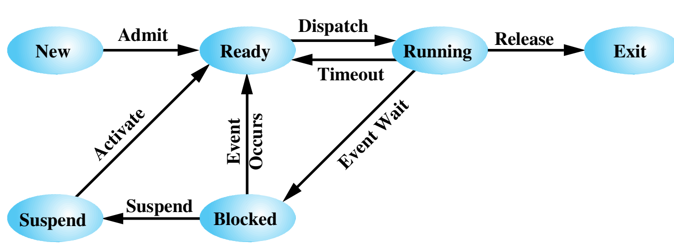
Suspended Processes

- Swapping
  - When none of the processes in main memory is in the Ready state, the OS swaps one of the blocked processes out on to disk into a suspend queue
    - This is a queue of existing processes that have been temporarily kicked out of main memory, or suspended
    - The OS then brings in another process from the suspend queue or it honors a new-process request
    - Execution then continues with the newly arrived process

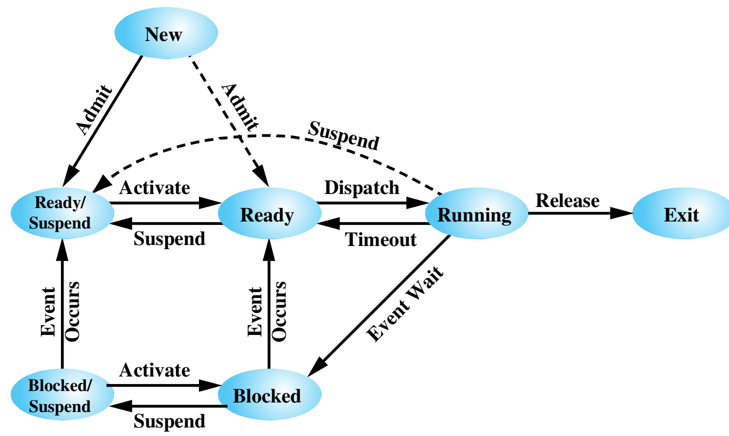
Suspended Processes

- Swapping
  - Swapping, however, is an I/O operation and therefore there is the potential for making the problem worse, not better. Because disk I/O is generally the fastest I/O on a system, swapping will usually enhance performance

Process State Transition Diagram with Suspend State



(a) With One Suspend State



(b) With Two Suspend States

Figure 3.9 Process State Transition Diagram with Suspend States

Characteristics of a Suspended Process

- The process is not immediately available for execution
- The process was placed in a suspended state by an agent: either itself, a parent process, or the OS, for the purpose of preventing its execution
- The process may or may not be waiting on an event
- The process may not be removed from this state until the agent explicitly orders the removal

Reasons for Process Suspension

Swapping	The OS needs to release sufficient main memory to bring in a process that is ready to execute.
Other OS reason	The OS may suspend a background or utility process or a process that is suspected of causing a problem.
Interactive user request	A user may wish to suspend execution of a program for purposes of debugging or in connection with the use of a resource.
Timing	A process may be executed periodically (e.g., an accounting or system monitoring process) and may be suspended while waiting for the next time interval.
Parent process request	A parent process may wish to suspend execution of a descendent to examine or modify the suspended process, or to coordinate the activity of various descendants.

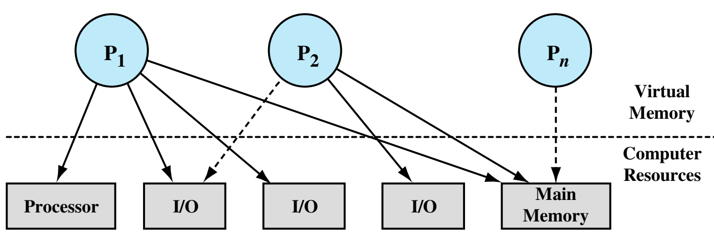


Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)

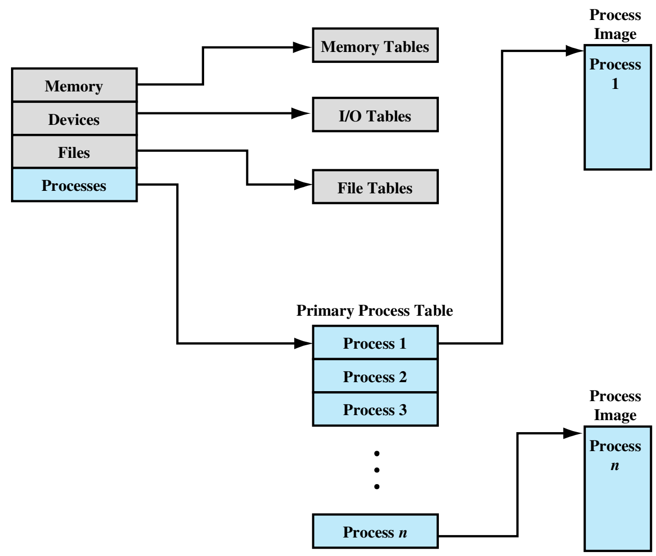
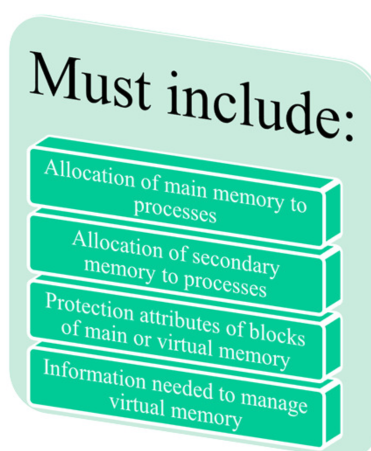


Figure 3.11 General Structure of Operating System Control Tables

Memory Tables

- Used to keep track of both main (real) and secondary (virtual) memory
- Processes are maintained on secondary memory using some sort of virtual memory or simple swapping mechanism



I/O Tables

- Used by the OS to manage the I/O devices and channels of the computer system
- At any given time, an I/O device may be available or assigned to a particular process

If an I/O operation is in progress, the OS needs to know:

- The status of the I/O operation
- The location in main memory being used as the source or destination of the I/O transfer

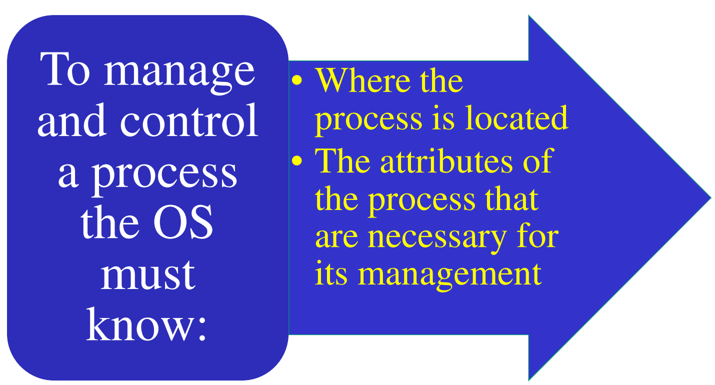
File Tables

- These tables provide information about:
  - Existence of files
  - Location on secondary memory
  - Current status
  - Other attributes
- Information may be maintained and used by a file management system
  - In which case the OS has little or no knowledge of files
- In other operating systems, much of the detail of file management is managed by the OS itself

Process Tables

- Must be maintained to manage processes
- There must be some reference to memory, I/O, and files, directly or indirectly
- The tables themselves must be accessible by the OS and therefore are subject to memory management

Process Control Structures



Process Control Structures - Process Location

- A process must include a program or set of programs to be executed
- A process will consist of at least sufficient memory to hold the programs and data of that process
- The execution of a program typically involves a stack that is used to keep track of procedure calls and parameter passing between procedures

Process Control Structures - Process Attributes

- Each process has associated with it a number of attributes that are used by the OS for process control
- The collection of program, data, stack, and attributes is referred to as the process image
- Process image location will depend on the memory management scheme being used

Typical Elements of a Process Image

User Data
The modifiable part of the user space. May include program data, a user stack area, and programs that may be modified.

User Program
The program to be executed.

Stack
Each process has one or more last-in-first-out (LIFO) stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls.

Process Control Block
Data needed by the OS to control the process (see Table 3.5).

Typical Elements of a Process Control Block

Process Identification

Identifiers
Numeric identifiers that may be stored with the process control block include
- Identifier of this process
- Identifier of the process that created this process (parent process)
- User identifier

Processor State Information

User-Visible Registers
A user-visible register is one that may be referenced by means of the machine language that the processor executes while in user mode. Typically, there are from 8 to 32 of these registers, although some RISC implementations have over 100.

Control and Status Registers
These are a variety of processor registers that are employed to control the operation of the processor. These include
- **Program counter:** Contains the address of the next instruction to be fetched
- **Condition codes:** Result of the most recent arithmetic or logical operation (e.g., sign, zero, carry, equal, overflow)
- **Status information:** Includes interrupt enabled/disabled flags, execution mode

Stack Pointers
Each process has one or more last-in-first-out (LIFO) system stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls. The stack pointer points to the top of the stack.

Process Control Information

Scheduling and State Information

This is information that is needed by the operating system to perform its scheduling function. Typical items of information:

- **Process state**: Defines the readiness of the process to be scheduled for execution (e.g., running, ready, waiting, halted).
- **Priority:** One or more fields may be used to describe the scheduling priority of the process. In some systems, several values are required (e.g., default, current, highest-allowable)
- **Scheduling-related information:** This will depend on the scheduling algorithm used. Examples are the amount of time that the process has been waiting and the amount of time that the process executed the last time it was running.
- **Event:** Identity of event the process is awaiting before it can be resumed.

Data Structuring

A process may be linked to other process in a queue, ring, or some other structure. For example, all processes in a waiting state for a particular priority level may be linked in a queue. A process may exhibit a parent-child (creator-created) relationship with another process. The process control block may contain pointers to other processes to support these structures.

Interprocess Communication

Various flags, signals, and messages may be associated with communication between two independent processes. Some or all of this information may be maintained in the process control block.

Process Privileges

Processes are granted privileges in terms of the memory that may be accessed and the types of instructions that may be executed. In addition, privileges may apply to the use of system utilities and services.

Memory Management

This section may include pointers to segment and/or page tables that describe the virtual memory assigned to this process.

Resource Ownership and Utilization

Resources controlled by the process may be indicated, such as opened files. A history of utilization of the processor or other resources may also be included; this information may be needed by the scheduler.

Typical Elements of a Process Control Block

Process Identification

- Each process is assigned a unique numeric identifier
  - Otherwise there must be a mapping that allows the OS to locate the appropriate tables based on the process identifier
- Many of the tables controlled by the OS may use process identifiers to cross-reference process tables

Process Identification

- Memory tables may be organized to provide a map of main memory with an indication of which process is assigned to each region
  - Similar references will appear in I/O and file tables
- When processes communicate with one another, the process identifier informs the OS of the destination of a particular communication
- When processes are allowed to create other processes, identifiers indicate the parent and descendents of each process

Processor State Information

Consists of the contents of processor registers

- User-visible registers
- Control and status registers
- Stack pointers

Program status word (PSW)

- Contains condition codes plus other status information
- EFLAGS register is an example of a PSW used by any OS running on an x86 processor

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	ID	VIP	VIF	AC	VM	RF	0	NT	IOPL		OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

X ID = Identification flag
X VIP = Virtual interrupt pending
X VIF = Virtual interrupt flag
X AC = Alignment check
X VM = Virtual 8086 mode
X RF = Resume flag
X NT = Nested task flag
X IOPL = I/O privilege level
S OF = Overflow flag
C DF = Direction flag
X IF = Interrupt enable flag
X TF = Trap flag
S SF = Sign flag
S ZF = Zero flag
S AF = Auxiliary carry flag
S PF = Parity flag
S CF = Carry flag

S Indicates a Status Flag
C Indicates a Control Flag
X Indicates a System Flag
Shaded bits are reserved

Figure 3.12 x86 EFLAGS Register

x86 EFLAGS Register Bits

Status Flags (condition codes)

AF (Auxiliary carry flag)
Represents carrying or borrowing between half-bytes of an 8-bit arithmetic or logic operation using the AL register.

CF (Carry flag)
Indicates carrying out or borrowing into the leftmost bit position following an arithmetic operation. Also modified by some of the shift and rotate operations.

OF (Overflow flag)
Indicates an arithmetic overflow after an addition or subtraction.

PF (Parity flag)
Parity of the result of an arithmetic or logic operation. 1 indicates even parity; 0 indicates odd parity.

SF (Sign flag)
Indicates the sign of the result of an arithmetic or logic operation.

ZF (Zero flag)
Indicates that the result of an arithmetic or logic operation is 0.

Control Flag

DF (Direction flag)
Determines whether string processing instructions increment or decrement the 16-bit half-registers SI and DI (for 16-bit operations) or the 32-bit registers ESI and EDI (for 32-bit operations).

System Flags (should not be modified by application programs)

AC (Alignment check)
Set if a word or doubleword is addressed on a nonword or nondoubleword boundary.

ID (Identification flag)
If this bit can be set and cleared, this processor supports the CPUID instruction. This instruction provides information about the vendor, family, and model.

RF (Resume flag)
Allows the programmer to disable debug exceptions so that the instruction can be restarted after a debug exception without immediately causing another debug exception.

IOPL (I/O privilege level)
When set, causes the processor to generate an exception on all accesses to I/O devices during protected mode operation.

IF (Interrupt enable flag)
When set, the processor will recognize external interrupts.

TF (Trap flag)
When set, causes an interrupt after the execution of each instruction. This is used for debugging.

NT (Nested task flag)
Indicates that the current task is nested within another task in protected mode operation.

VM (Virtual 8086 mode)
Allows the programmer to enable or disable virtual 8086 mode, which determines whether the processor runs as an 8086 machine.

VIP (Virtual interrupt pending)
Used in virtual 8086 mode to indicate that one or more interrupts are awaiting service.

VIF (Virtual interrupt flag)
Used in virtual 8086 mode instead of IF.

Process Control Information

- The additional information needed by the OS to control and coordinate the various active processes

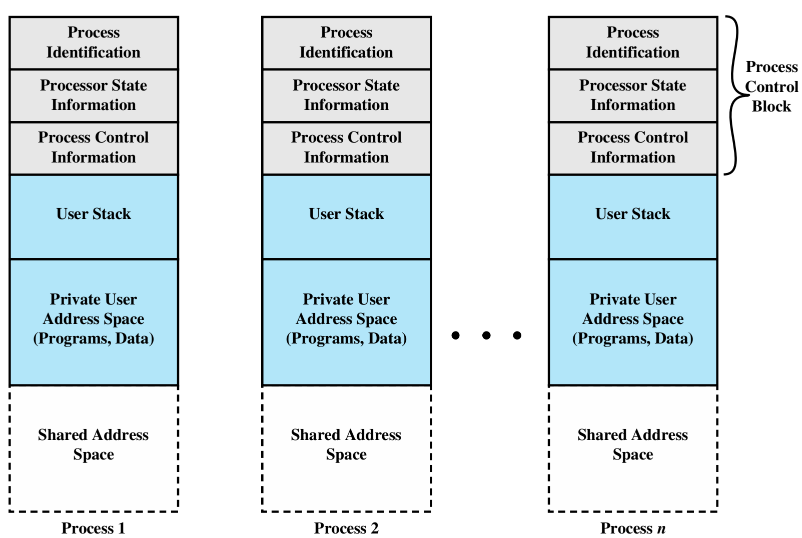


Figure 3.13 User Processes in Virtual Memory

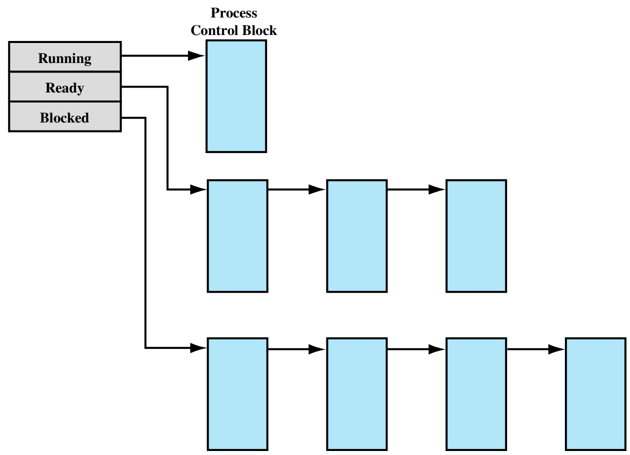


Figure 3.14 Process List Structures

Role of the Process Control Block

- The most important data structure in an OS
  - Contains all of the information about a process that is needed by the OS
  - Blocks are read and/or modified by virtually every module in the OS
  - Defines the state of the OS

Role of the Process Control Block

- Difficulty is not access, but protection
  - A bug in a single routine could damage process control blocks, which could destroy the system's ability to manage the affected processes
  - A design change in the structure or semantics of the process control block could affect a number of modules in the OS

Modes of Execution

- User Mode
  - Less-privileged mode
  - User programs typically execute in this mode
- System Mode
  - More-privileged mode
  - Also referred to as control mode or kernel mode
  - Kernel of the operating system

Typical Functions of an Operating System Kernel

Process Management

- Process creation and termination
- Process scheduling and dispatching
- Process switching
- Process synchronization and support for interprocess communication
- Management of process control blocks

Memory Management

- Allocation of address space to processes
- Swapping
- Page and segment management

I/O Management

- Buffer management
- Allocation of I/O channels and devices to processes

Support Functions

- Interrupt handling
- Accounting
- Monitoring

Process Creation

- Once the OS decides to create a new process it:

1. Assigns a unique process identifier to the new process
2. Allocates space for the process
3. Initializes the process control block
4. Sets the appropriate linkages
5. Creates or expands other data structures

Mechanisms for Interrupting the Execution of a Process

Mechanism	Cause	Use
Interrupt	External to the execution of the current instruction	Reaction to an asynchronous external event
Trap	Associated with the execution of the current instruction	Handling of an error or an exception condition
Supervisor call	Explicit request	Call to an operating system function

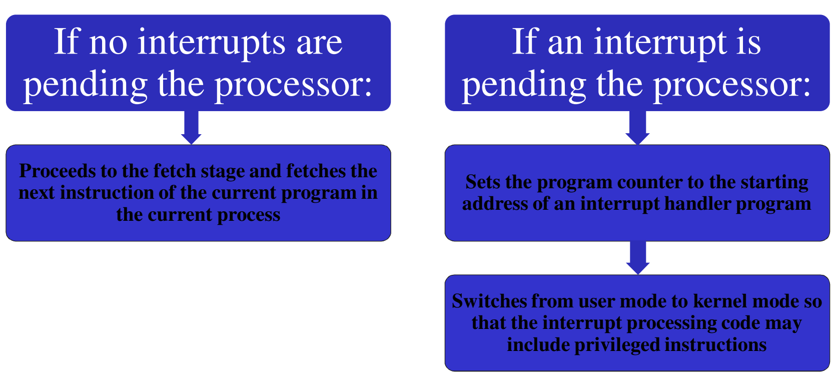
System Interrupts

- Due to some sort of event that is external to and independent of the currently running process
  - Clock interrupt
  - I/O interrupt
  - Memory fault
- Time slice
  - The maximum amount of time that a process can execute before being interrupted

System Interrupts - Traps

- An error or exception condition generated within the currently running process
- OS determines if the condition is fatal
  - Moved to the Exit state and a process switch occurs
  - Action will depend on the nature of the error the design of the OS

Mode Switching



Change of Process State

- The steps in a full process switch are:

1. Save the context of the processor
2. Update the process control block of the process currently in the Running state
3. Move the process control block of this process to the appropriate queue
4. Select another process for execution
5. Update the process control block of the process selected
6. Update memory management data structures
7. Restore the context of the processor to that which existed at the time the selected process was last switched out

Execution of the Operating System

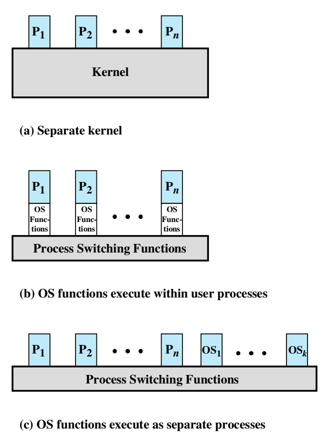


Figure 3.15 Relationship Between Operating System and User Processes

Execution Within User Processes

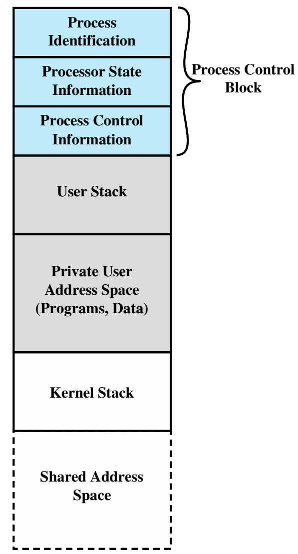


Figure 3.16 Process Image: Operating System Executes Within User Space

Unix SVR4

- Uses the model where most of the OS executes within the environment of a user process
- System processes run in kernel mode
  - Executes operating system code to perform administrative and housekeeping functions
- User Processes
  - Operate in user mode to execute user programs and utilities
  - Operate in kernel mode to execute instructions that belong to the kernel
  - Enter kernel mode by issuing a system call, when an exception is generated, or when an interrupt occurs

UNIX Process States

User Running	Executing in user mode.
Kernel Running	Executing in kernel mode.
Ready to Run, in Memory	Ready to run as soon as the kernel schedules it.
Asleep in Memory	Unable to execute until an event occurs; process is in main memory (a blocked state).
Ready to Run, Swapped	Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
Sleeping, Swapped	The process is awaiting an event and has been swapped to secondary storage (a blocked state).
Preempted	Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
Created	Process is newly created and not yet ready to run.
Zombie	Process no longer exists, but it leaves a record for its parent process to collect.

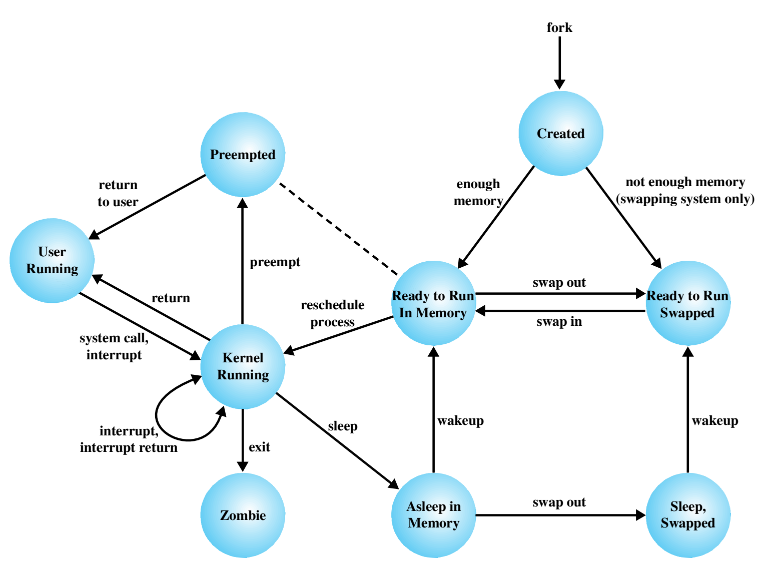


Figure 3.17 UNIX Process State Transition Diagram

UNIX Process Image

	User-Level Context
Process text	Executable machine instructions of the program
Process data	Data accessible by the program of this process
User stack	Contains the arguments, local variables, and pointers for functions executing in user mode
Shared memory	Memory shared with other processes, used for interprocess communication
	Register Context
Program counter	Address of next instruction to be executed; may be in kernel or user memory space of this process
Processor status register	Contains the hardware status at the time of preemption; contents and format are hardware dependent
Stack pointer	Points to the top of the kernel or user stack, depending on the mode of operation at the time or preemption
General-purpose registers	Hardware dependent
	System-Level Context
Process table entry	Defines state of a process; this information is always accessible to the operating system
U (user) area	Process control information that needs to be accessed only in the context of the process
Per process region table	Defines the mapping from virtual to physical addresses; also contains a permission field that indicates the type of access allowed the process: read-only, read-write, or read-execute
Kernel stack	Contains the stack frame of kernel procedures as the process executes in kernel mode

UNIX Process Table Entry

Process status	Current state of process.
Pointers	To U area and process memory area (text, data, stack).
Process size	Enables the operating system to know how much space to allocate the process.
User identifiers	The **real user ID** identifies the user who is responsible for the running process. The **effective user ID** may be used by a process to gain temporary privileges associated with a particular program; while that program is being executed as part of the process, the process operates with the effective user ID.
Process identifiers	ID of this process; ID of parent process. These are set up when the process enters the Created state during the fork system call.
Event descriptor	Valid when a process is in a sleeping state; when the event occurs, the process is transferred to a ready-to-run state.
Priority	Used for process scheduling.
Signal	Enumerates signals sent to a process but not yet handled.
Timers	Include process execution time, kernel resource utilization, and user-set timer used to send alarm signal to a process.
P_link	Pointer to the next link in the ready queue (valid if process is ready to execute).
Memory status	Indicates whether process image is in main memory or swapped out. If it is in memory, this field also indicates whether it may be swapped out or is temporarily locked into main memory.

UNIX U(for User) Area

Process table pointer	Indicates entry that corresponds to the U area.
User identifiers	Real and effective user IDs. Used to determine user privileges.
Timers	Record time that the process (and its descendants) spent executing in user mode and in kernel mode.
Signal-handler array	For each type of signal defined in the system, indicates how the process will react to receipt of that signal (exit, ignore, execute specified user function).
Control terminal	Indicates login terminal for this process, if one exists.
Error field	Records errors encountered during a system call.
Return value	Contains the result of system calls.
I/O parameters	Describe the amount of data to transfer, the address of the source (or target) data array in user space, and file offsets for I/O.
File parameters	Current directory and current root describe the file system environment of the process.
User file descriptor table	Records the files the process has opened.
Limit fields	Restrict the size of the process and the size of a file it can write.
Permission modes fields	Mask mode settings on files the process creates.

Process Control

- Process creation is by means of the kernel system call, `fork()`
- When a process issues a fork request, the OS performs the functions on the following slide:

Process Control

1. Allocates a slot in the process table for the new process
2. Assigns a unique process ID to the child process
3. Makes a copy of the process image of the parent, with the exception of any shared memory
4. Increments counters for any files owned by the parent, to reflect that an additional process now also owns those files
5. Assigns the child process to the Ready to Run state
6. Returns the ID number of the child to the parent process, and a 0 value to the child process

After Creation

- After creating the process the Kernel can do one of the following, as part of the dispatcher routine:
  - Stay in the parent process. Control returns to user mode at the point of the fork call of the parent.
  - Transfer control to the child process. The child process begins executing at the same point in the code as the parent, namely at the return from the fork call.
  - Transfer control to another process. Both parent and child are left in the Ready to Run state.