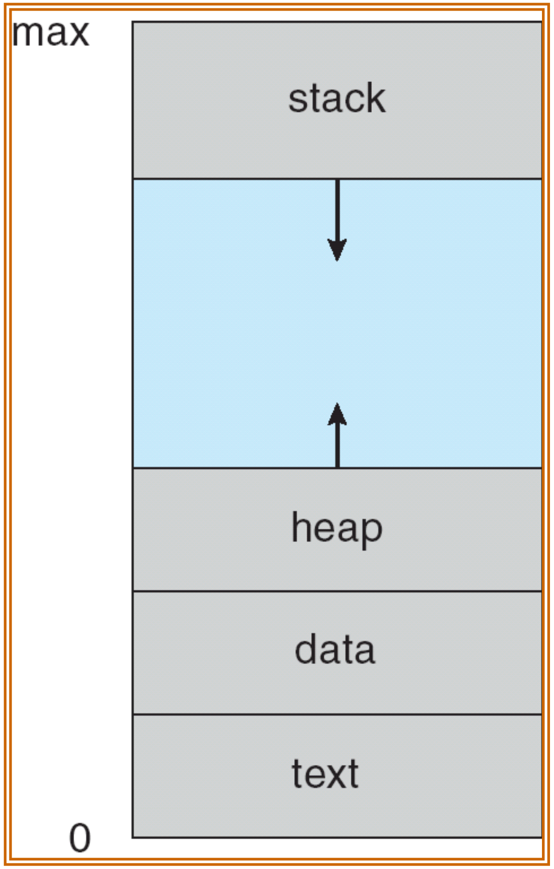
Process

What is a *"process"*?

- *A program in execution*
- An instance of a program running on a computer
- The entity that can be assigned to and executed on a processor
- A unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system instructions

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Textbook uses the terms *job* and *process* almost interchangeably
- Process – a program in execution; process execution must progress in sequential fashion
- A process includes:
  - program counter
  - stack
  - data section

Process in Memory



Process Elements

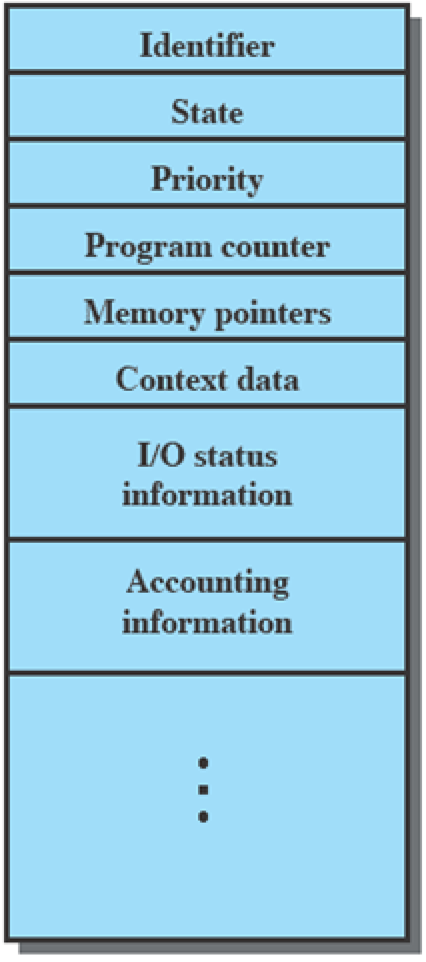
- A process is comprised of:
  - Program code (possibly shared)
  - A set of data
  - A number of attributes describing the state of the process

Process Elements

- While the process is running it has a number of elements including
  - Identifier
  - State
  - Priority
  - Program counter
  - Memory pointers
  - Context data
  - I/O status information
  - Accounting information

Process Control Block

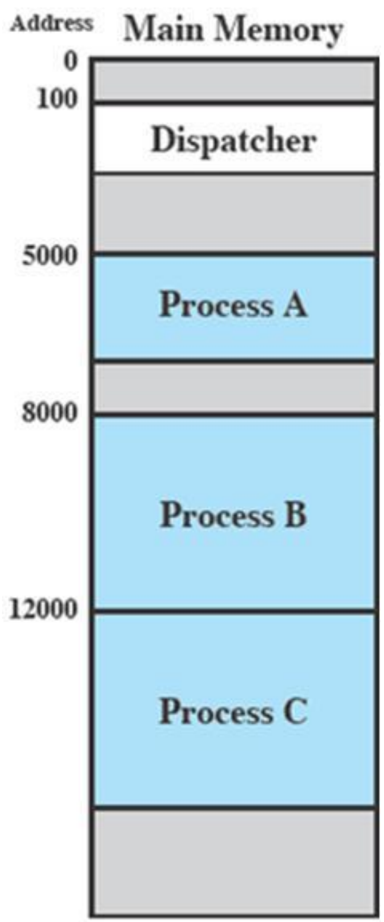
- Contains the process elements
- Created and manage by the operating system
- Allows support for multiple processes



Trace of the Process

- The behavior of an individual process is shown by listing the sequence of instructions that are executed
- This list is called a ***Trace***
- ***Dispatcher*** is a small program which switches the processor from one process to another

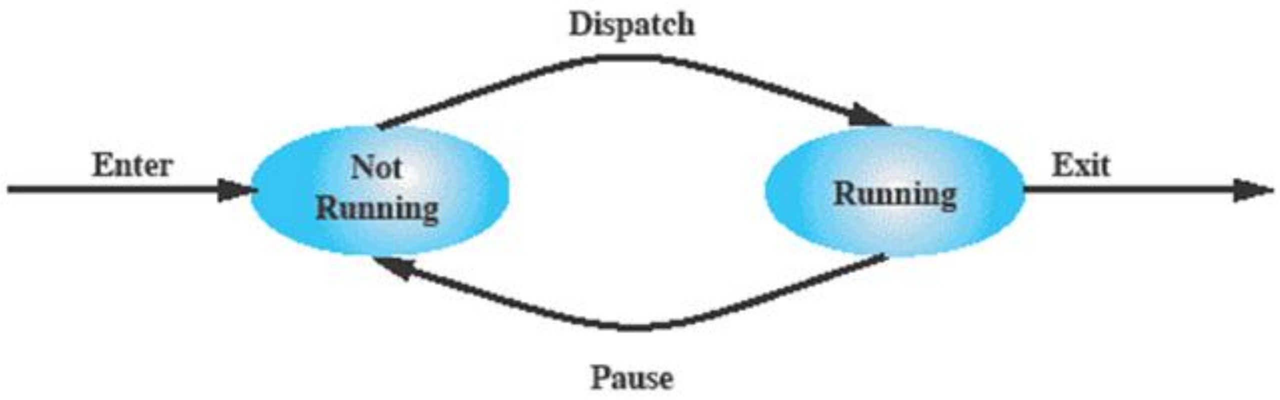
Process Execution



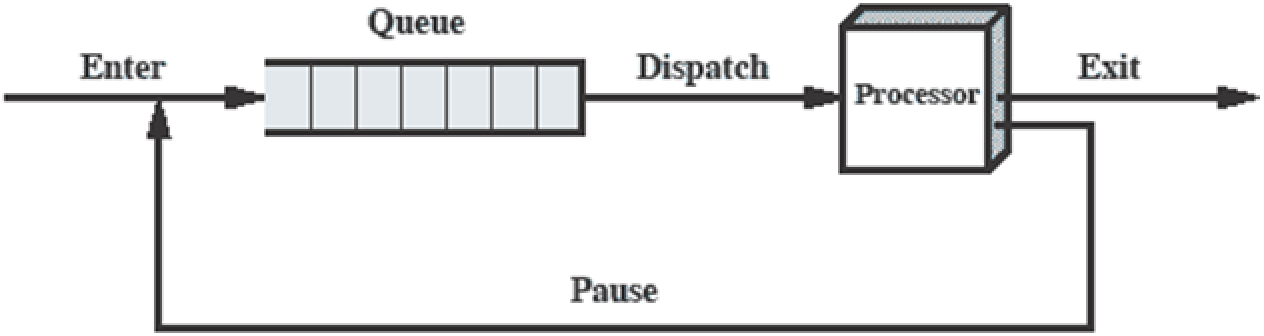
- Consider three processes being executed
- All are in memory (plus the dispatcher)
- Lets ignore virtual memory for this.

Two-State Process Model

- Process may be in one of two states
  - Running
  - Not-running

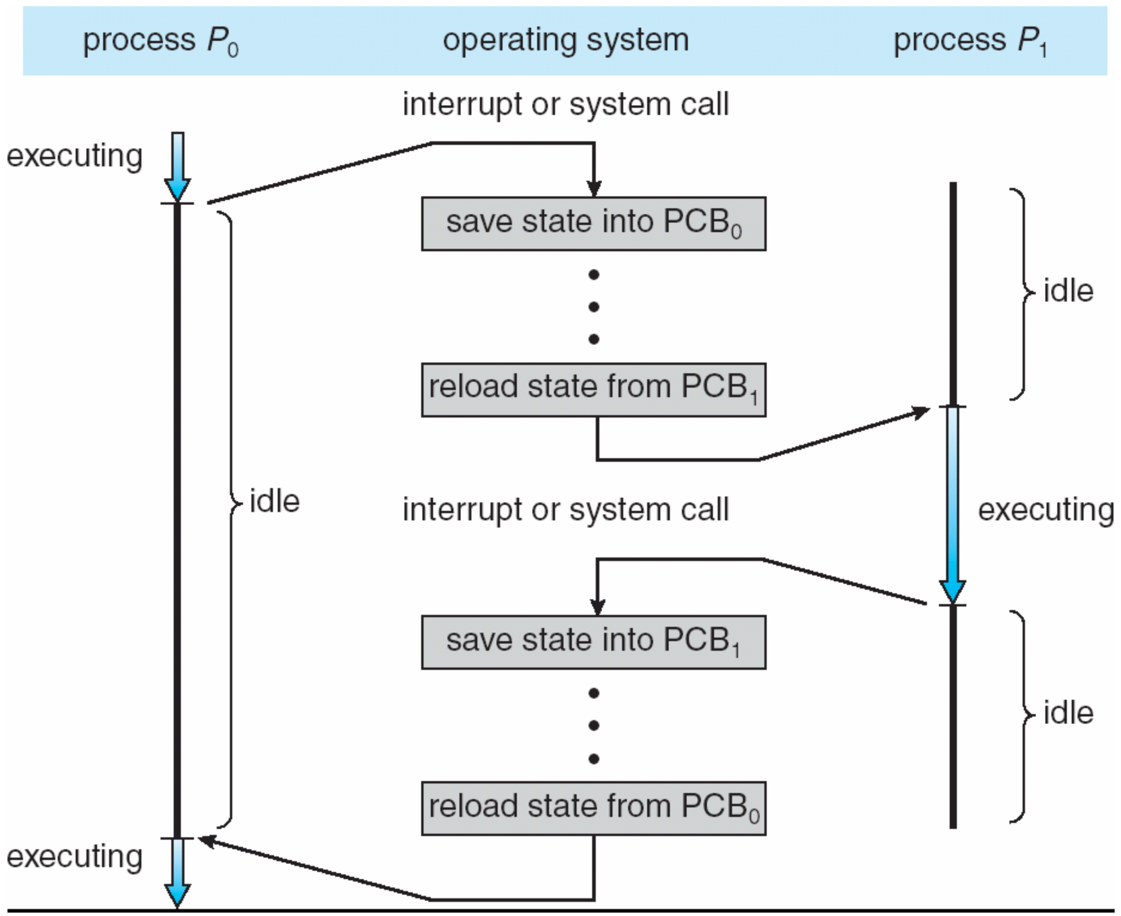


Queuing Diagram



Etc … processes moved by the dispatcher of the OS to the CPU then back to the queue until the task is competed

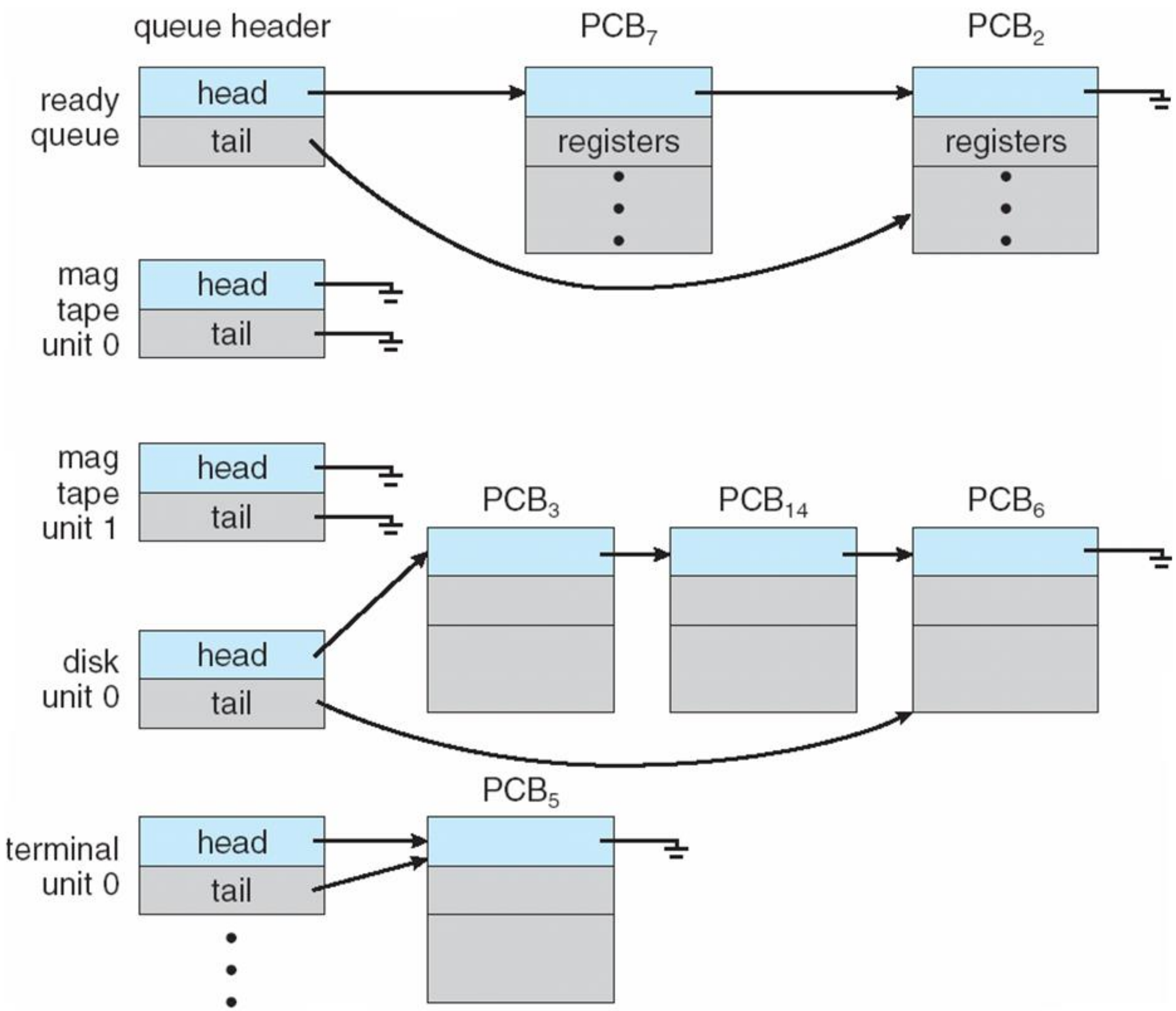
CPU Switch From Process to Process



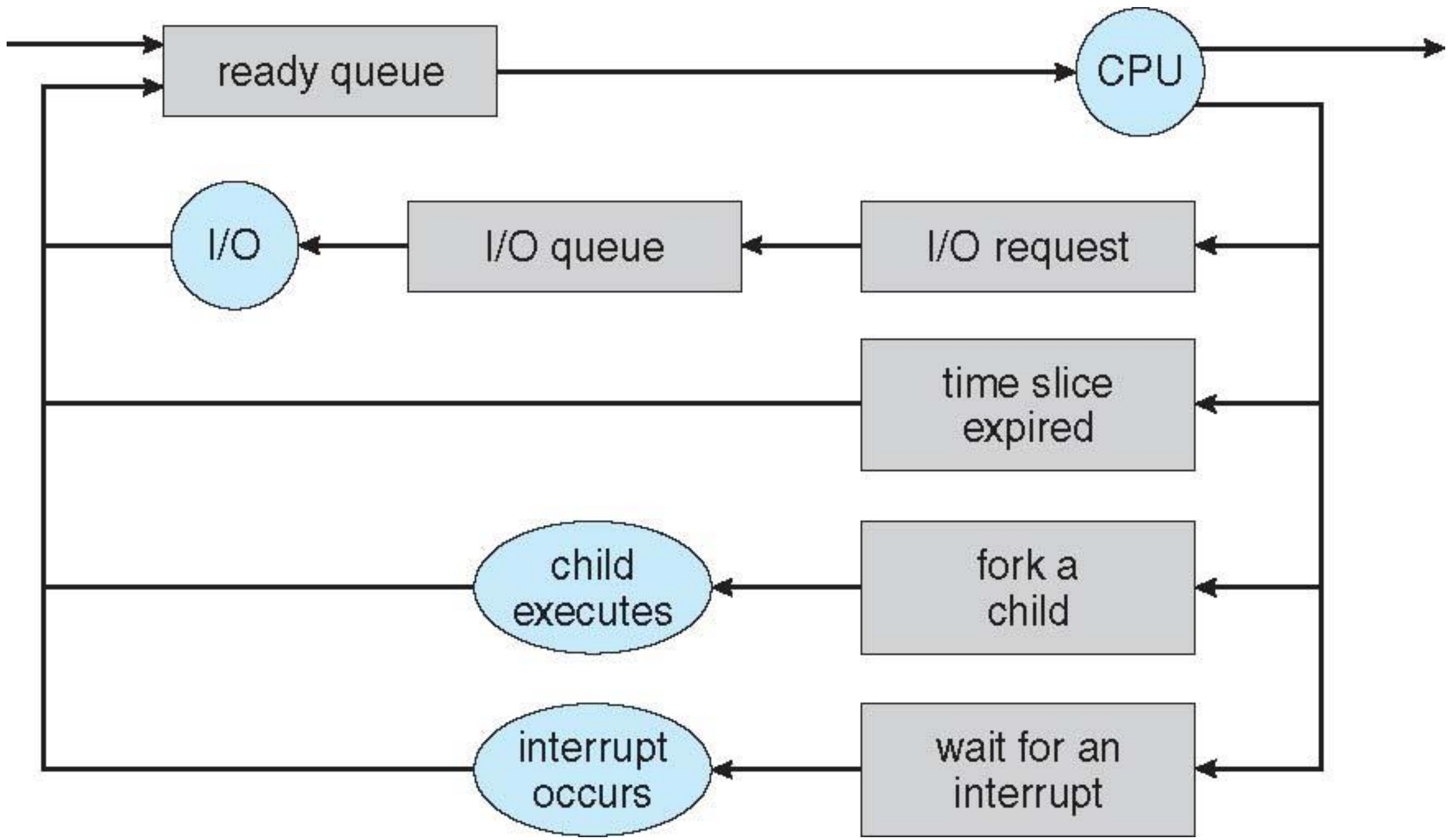
Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device
  - Processes migrate among the various queues

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling



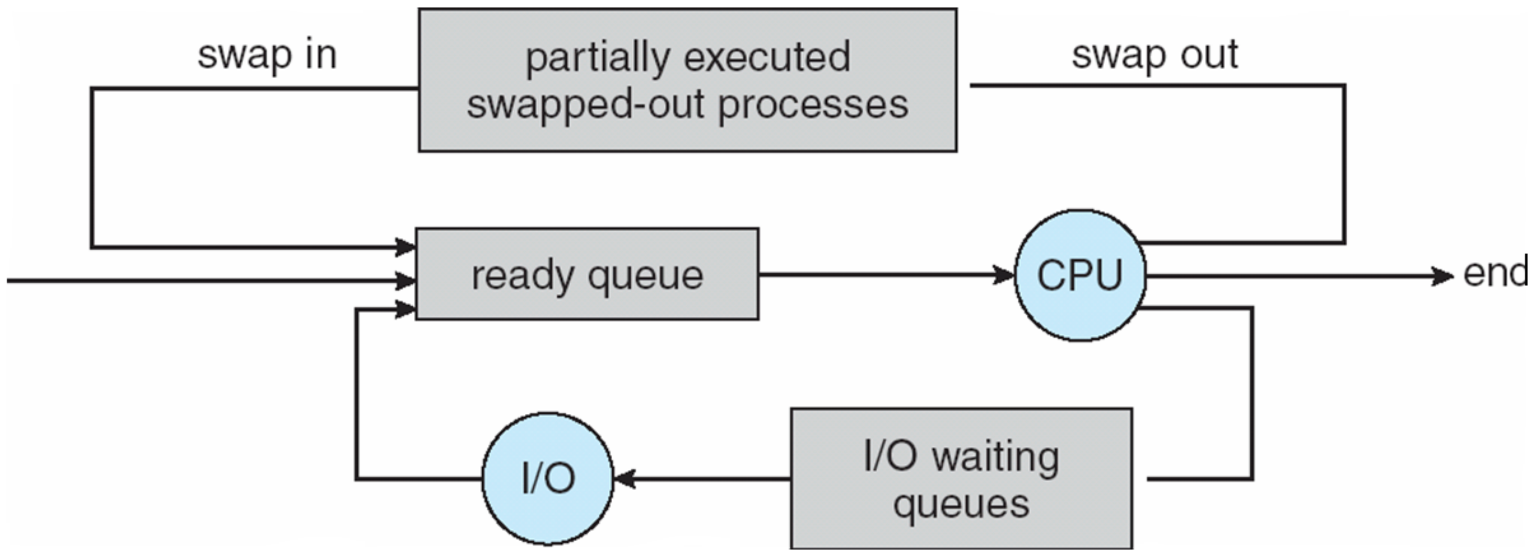
Schedulers

- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system

Schedulers (Cont.)

- Short-term scheduler is invoked very frequently (milliseconds) $\Rightarrow$ (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) $\Rightarrow$ (may be slow)
- The long-term scheduler controls the *degree of multiprogramming*
- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts

Addition of Medium Term Scheduling



Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a **context switch**.

- **Context** of a process represented in the PCB

- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB -> longer the context switch

- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU -> multiple contexts loaded at once

Process Birth and Death

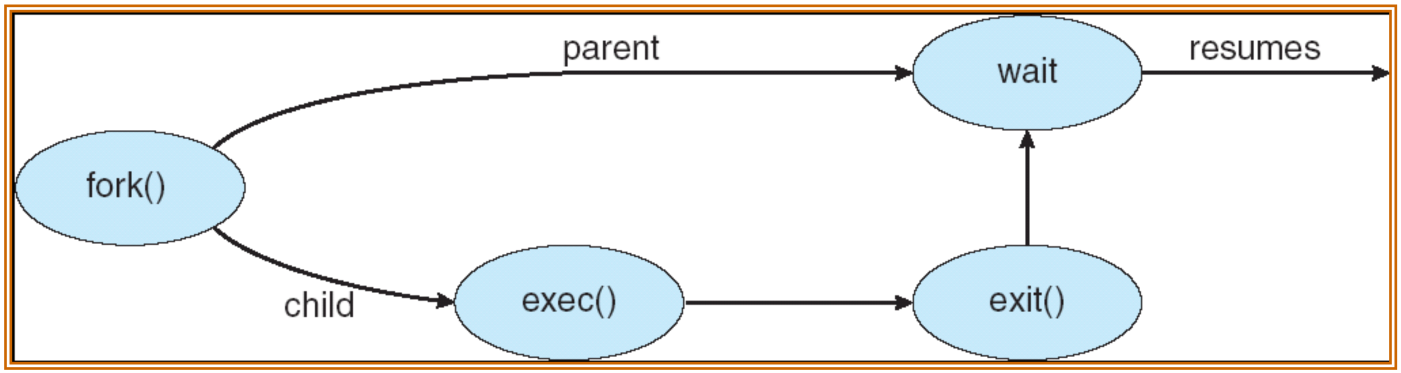
Creation	Termination
New batch job	Normal Completion
Interactive Login	Memory unavailable
Created by OS to provide a service	Protection error
Spawned by existing process	Operator or OS Intervention

Process Creation

- The OS builds a data structure to manage the process
- Traditionally, the OS creates all processes
  - But it can be useful to let a running process create another
- This action is called ***process spawning***
  - ***Parent Process*** is the original, creating, process
  - ***Child Process*** is the new process

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Resource sharing
  - Parent and children share all resources
  - Children share subset of parent’s resources
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently
  - Parent waits until children terminate

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **fork** system call creates new process
  - **exec** system call used after a **fork** to replace the process’ memory space with a new program


parent
wait
resumes
fork()
child
exec()
exit()

C Program Forking Separate Process

```
int main()
{
pid_t  pid;
	/* fork another process */
	pid = fork();
	if (pid < 0) { /* error occurred */
		fprintf(stderr, "Fork Failed");
		exit(-1);
	}
	else if (pid == 0) { /* child process */
		execlp("/bin/ls", "ls", NULL);
	}
	else { /* parent process */
		/* parent will wait for the child to complete */
		wait (NULL);
		printf ("Child Complete");
		exit(0);
	}
}
```

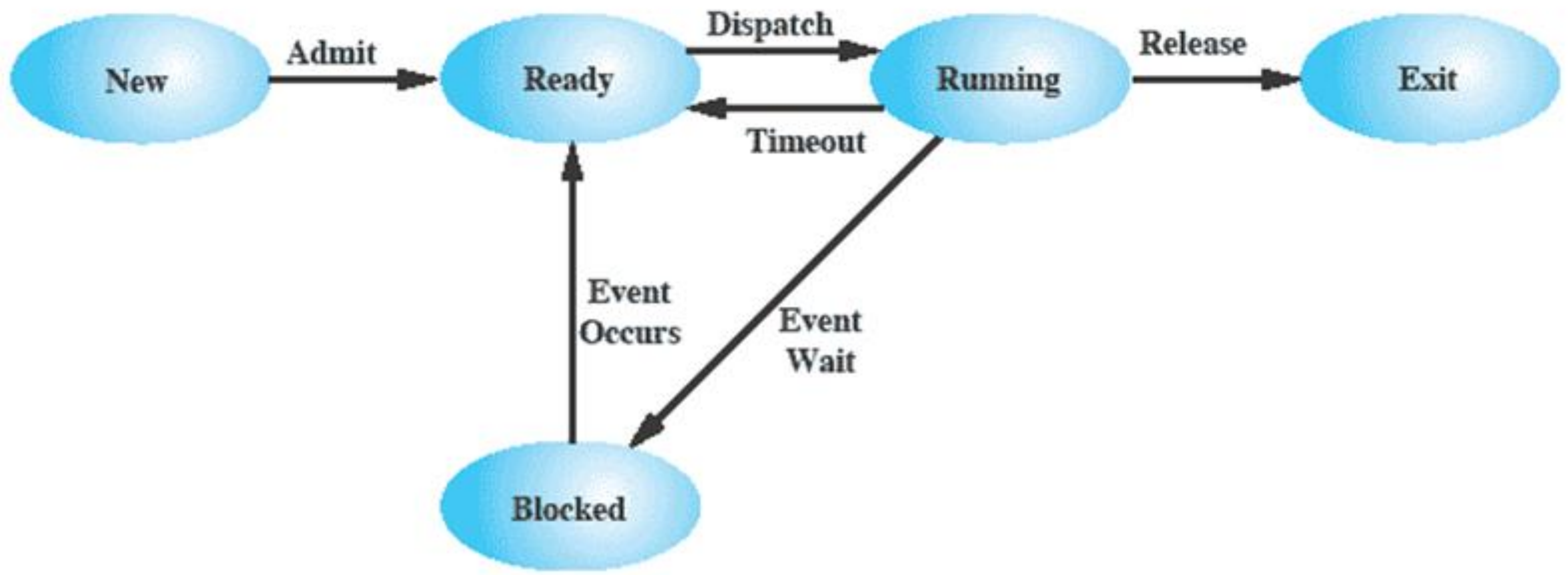
Process Termination

- There must be some way that a process can indicate completion.
- This indication may be:
  - A HALT instruction generating an interrupt alert to the OS.
  - A user action (e.g. log off, quitting an application)
  - A fault or error
  - Parent process terminating

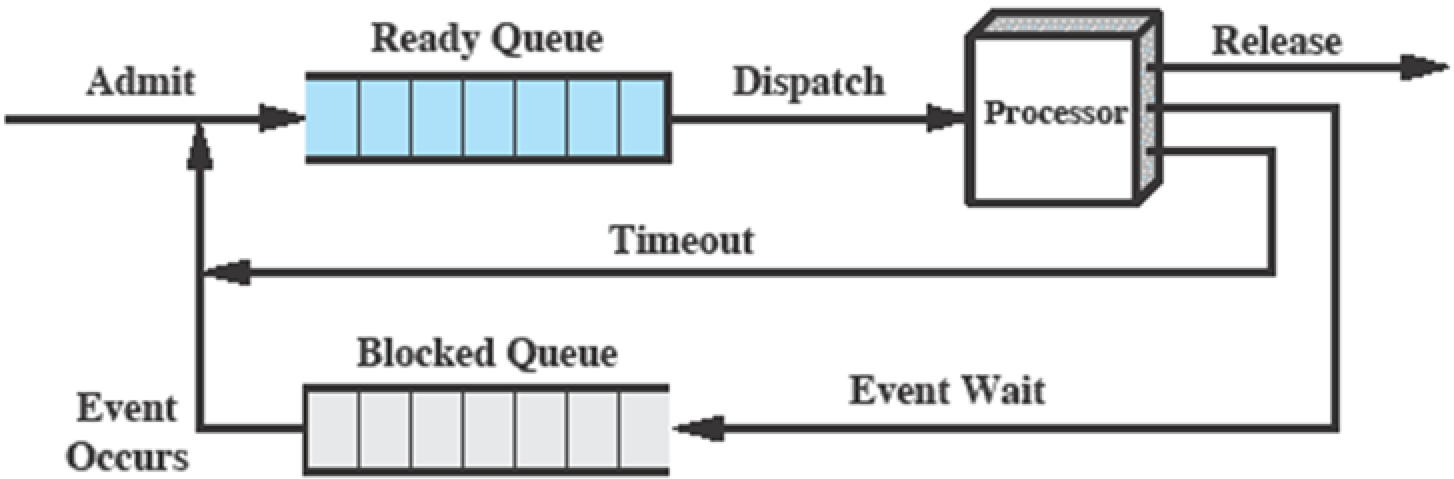
Process Termination

- Process executes last statement and asks the operating system to delete it (**exit**)
  - Output data from child to parent (via **wait**)
  - Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (**abort**)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating system do not allow child to continue if its parent terminates
      - All children terminated - *cascading termination*

Five-State Process Model

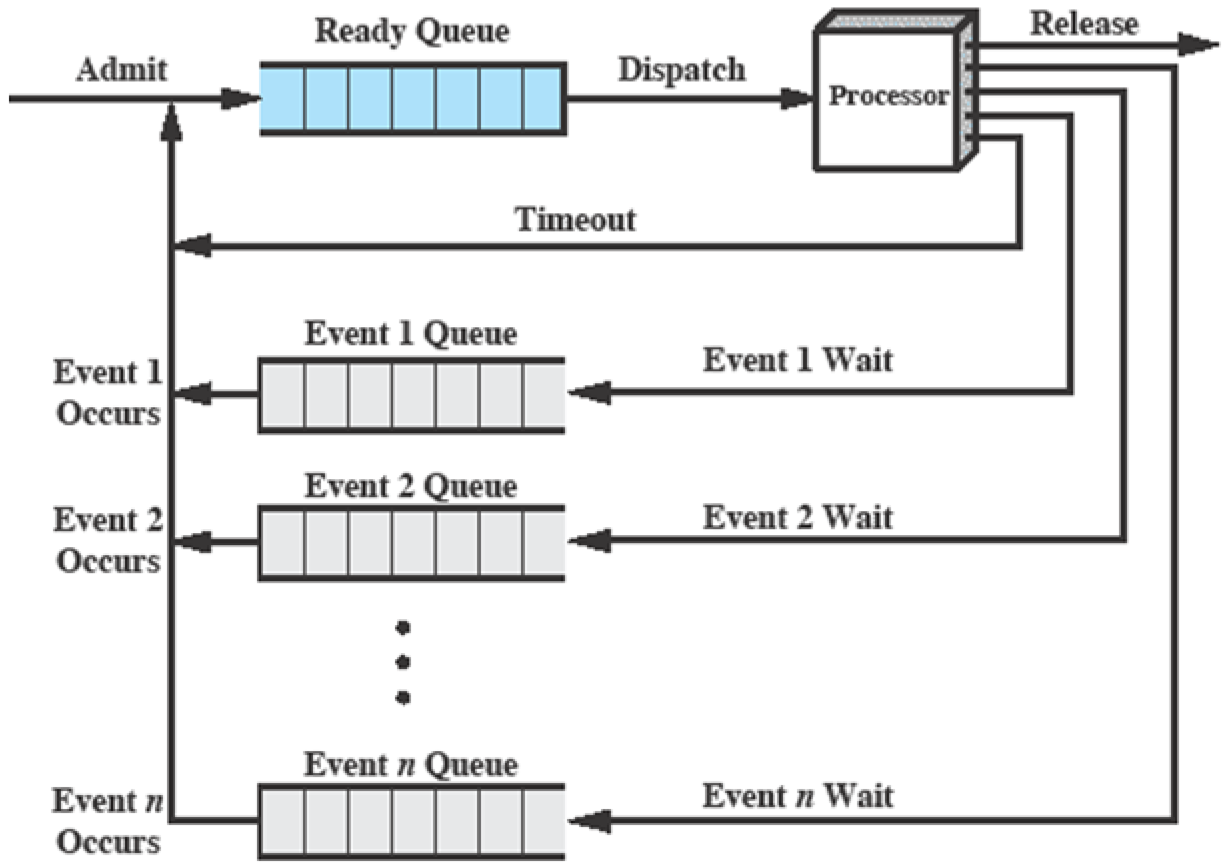


Using Two Queues



- Problem??

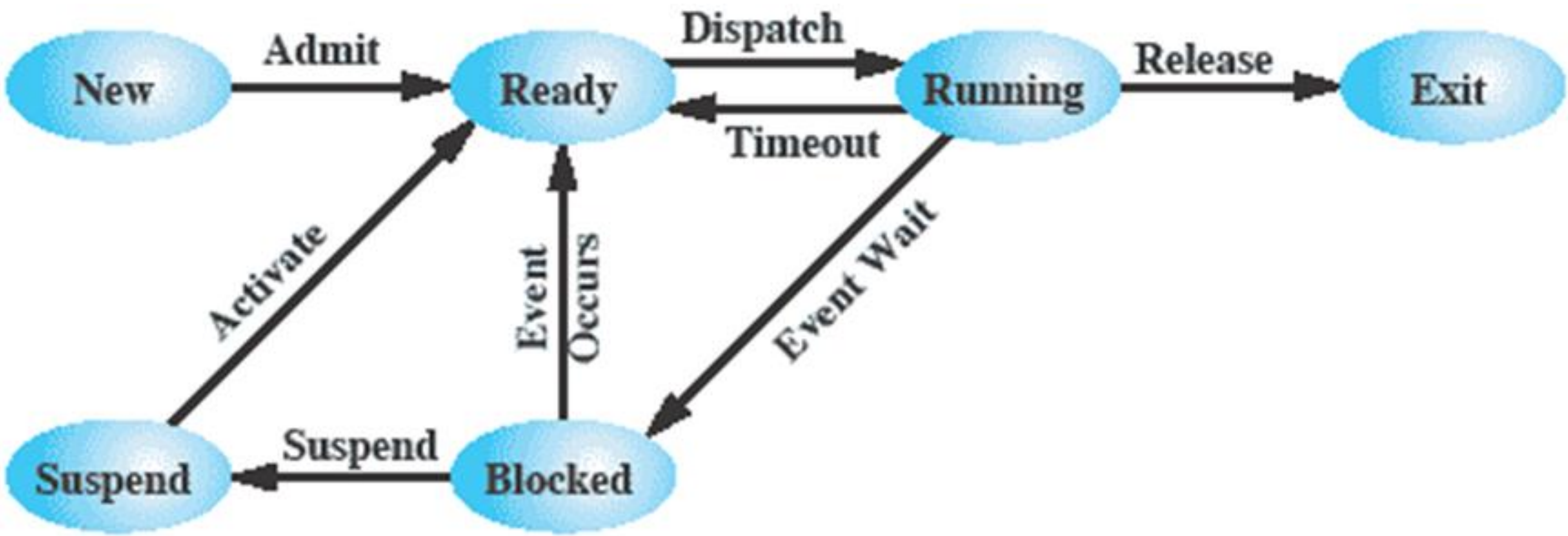
Multiple Blocked Queues



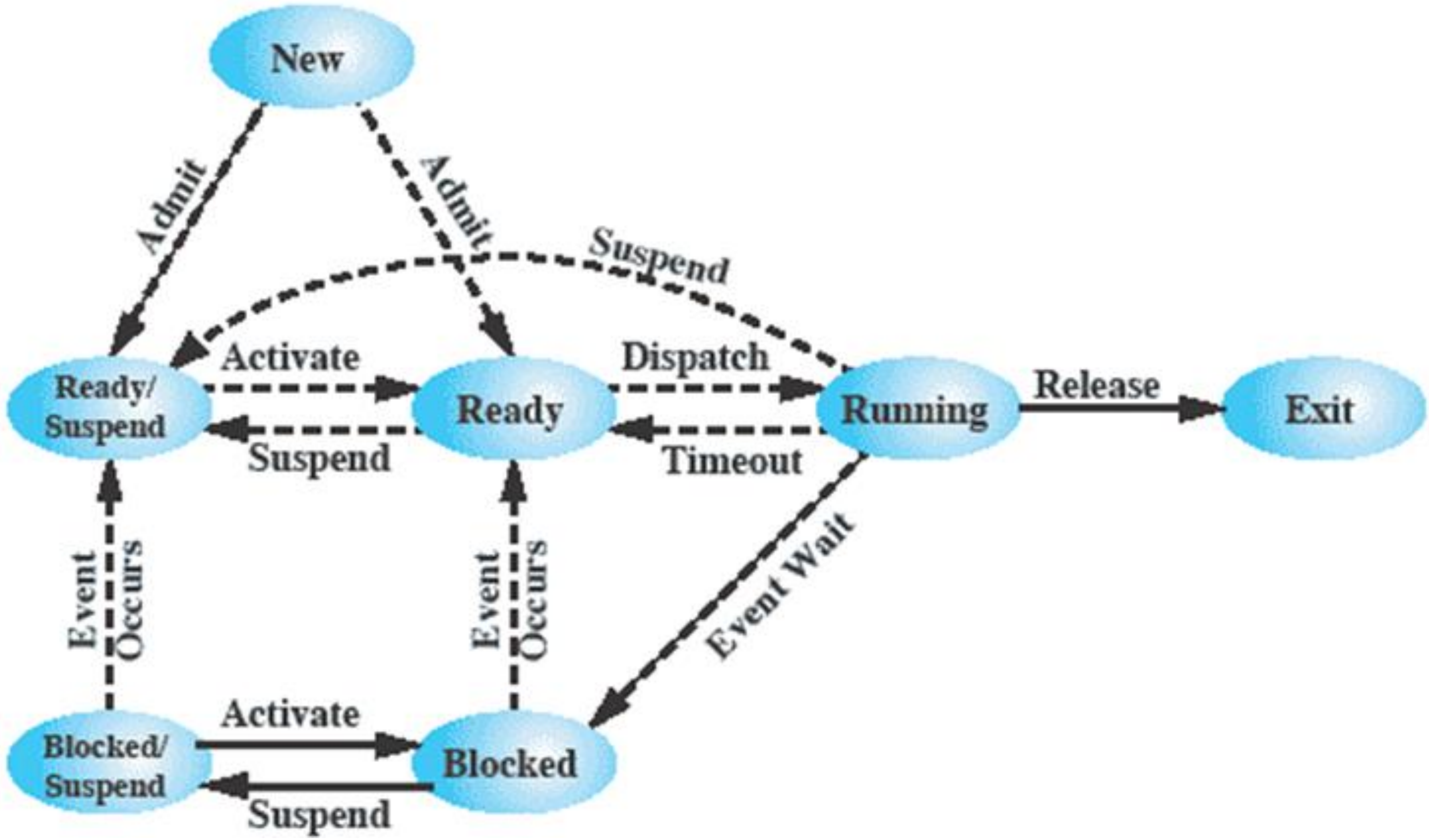
Suspended Processes

- Processor is faster than I/O so all processes could be waiting for I/O
  - Swap these processes to disk to free up more memory and use processor on more processes(aim??)
- Blocked state becomes ***suspend*** state when swapped to disk
- Two new states
  - Blocked/Suspend
  - Ready/Suspend

One Suspend State



Seven state model of process



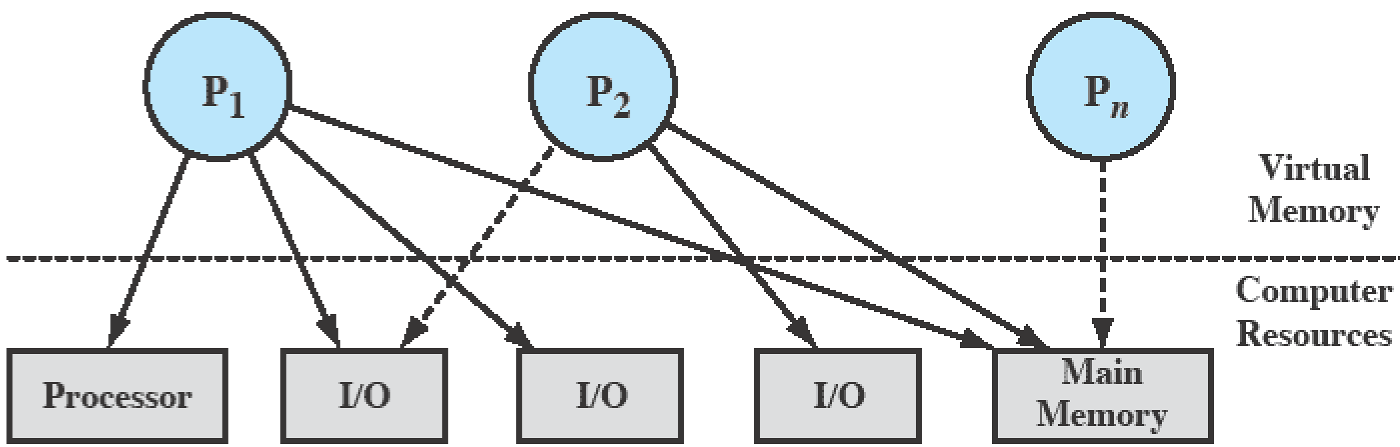
- **Ready:** The process is in main memory and available for execution.
- **Blocked:** The process is in main memory and awaiting an event.
- **Blocked/Suspend:** The process is in secondary memory and awaiting an event.
- **Ready/Suspend:** The process is in secondary memory but is available for execution as soon as it is loaded into main memory.

Reason for Process Suspension

Reason	Comment
Swapping	The OS needs to release sufficient main memory to bring in a process that is ready to execute.
Other OS Reason	OS suspects process of causing a problem.
Interactive User Request	e.g. 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.
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.

Processes and Resources

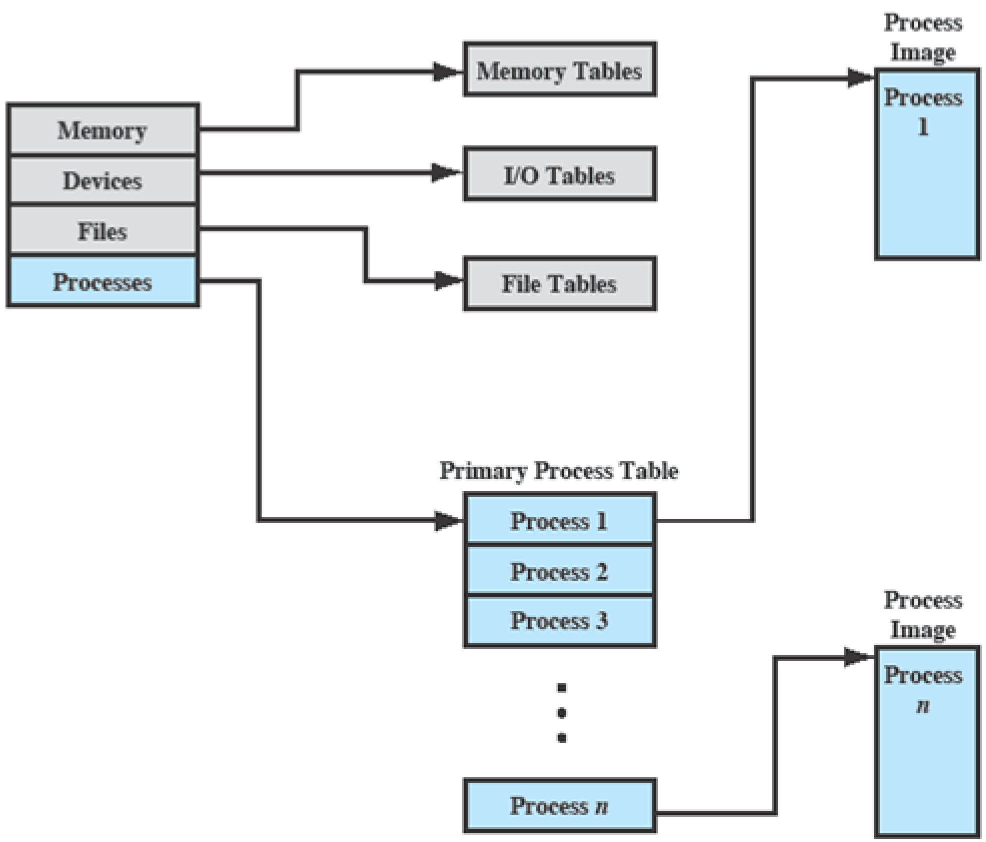
- OS controls events within the computer system. It schedules and dispatches processes for execution by the processor, allocates resources to processes, and responds to requests by user processes for basic services.



Operating System Control Structures

- For the OS is to manage processes and resources, it must have information about the current status of each process and resource.
- Tables are constructed for each entity the operating system manages

OS Control Tables



Memory Tables

- Memory tables are used to keep track of both main and secondary memory.
- Must include this information:
  - Allocation of main memory to processes
  - Allocation of secondary memory to processes
  - Protection attributes for access to shared memory regions
  - Information needed to manage virtual memory

I/O Tables

- Used by the OS to manage the I/O devices and channels of the computer.
- The OS needs to know
  - Whether the I/O device is available or assigned
  - The status of 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.
- Sometimes this information is maintained by a file management system

Process Tables

- To manage processes the OS needs to know details of the processes
  - Current state
  - Process ID
  - Location in memory
  - etc
- Process control block
  - ***Process image*** is the collection of program. Data, stack, and attributes

Process Attributes

- We can group the process control block information into three general categories:
  - Process identification
  - Processor state information
  - Process control information

Process Identification

- Each process is assigned a unique numeric identifier.
- Many of the other tables controlled by the OS may use process identifiers to cross-reference process tables

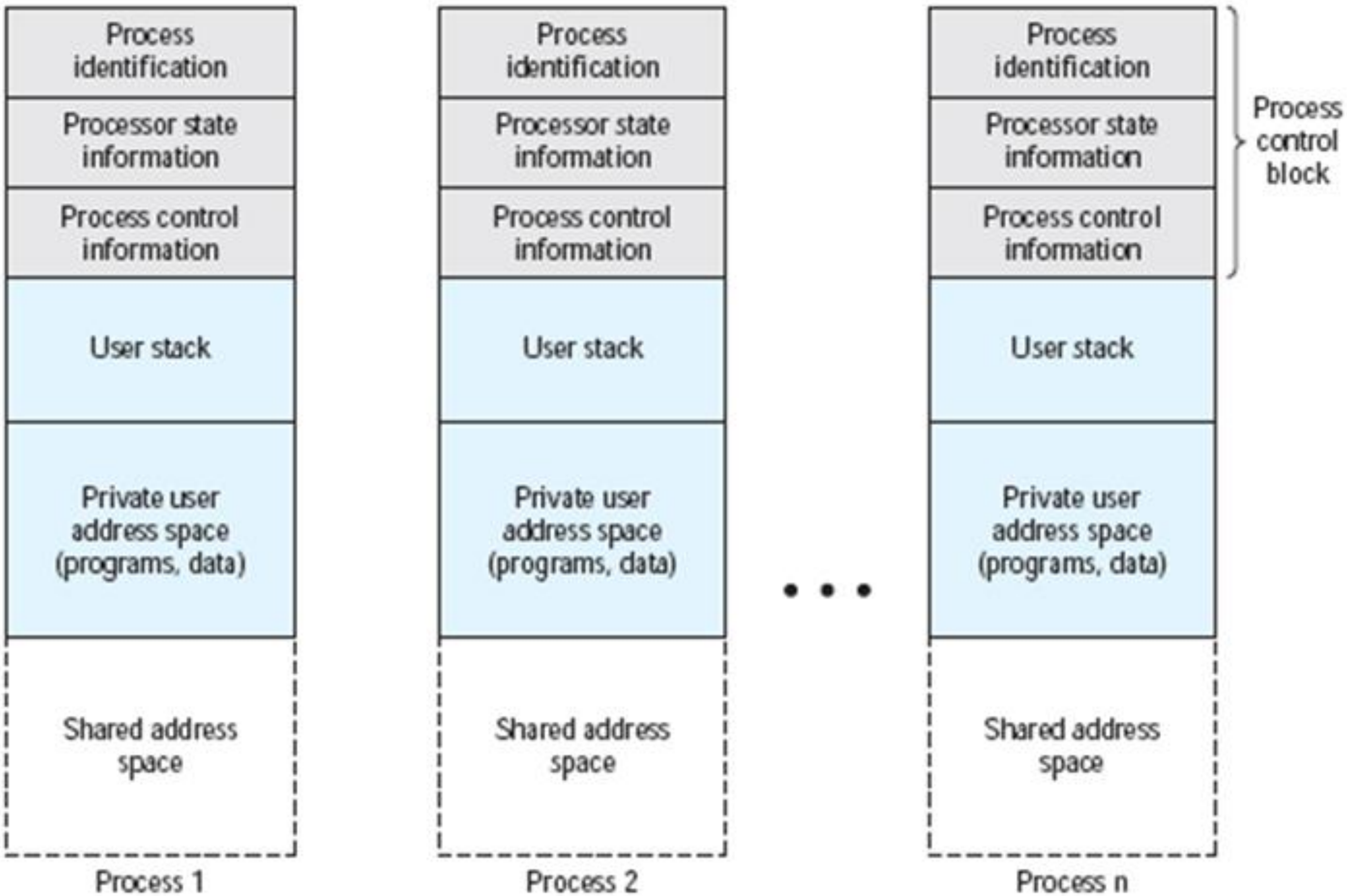
Processor State Information

- This consists of the contents of processor registers.
  - User-visible registers
  - Control and status registers
  - Stack pointers
- Program status word (PSW)
  - contains status information
  - Example: the EFLAGS register on Pentium processors

Process Control Information

- This is the additional information needed by the OS to control and coordinate the various active processes.

Structure of Process Images in Virtual Memory



Role of the Process Control Block

- The most important data structure in an OS
  - It defines the state of the OS
- Process Control Block requires protection
  - A faulty routine could cause damage to the block destroying the OS's ability to manage the process
  - Any design change to the block could affect many modules of the OS

Modes of Execution

- Most processors support at least two modes of execution
- User mode
  - Less-privileged mode
  - User programs typically execute in this mode
- System mode
  - More-privileged mode
  - Kernel of the operating system

Process Creation

- Once the OS decides to create a new process it:
  - Assigns a unique process identifier
  - Allocates space for the process
  - Initializes process control block
  - Sets up appropriate linkages
  - Creates or expand other data structures

Switching Processes

- Several design issues are raised regarding process switching
  - What events trigger a process switch?
  - We must distinguish between mode switching and process switching.
  - What must the OS do to the various data structures under its control to achieve a process switch?

When to switch processes

A process switch may occur any time that the OS has gained control from the currently running process. Possible events giving OS control are:

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

Change of Process State …

- The steps in a process switch are:
  1. Save context of processor including program counter and other registers
  2. Update the process control block of the process that is currently in the Running state
  3. Move process control block to appropriate queue – ready; blocked; ready/suspend

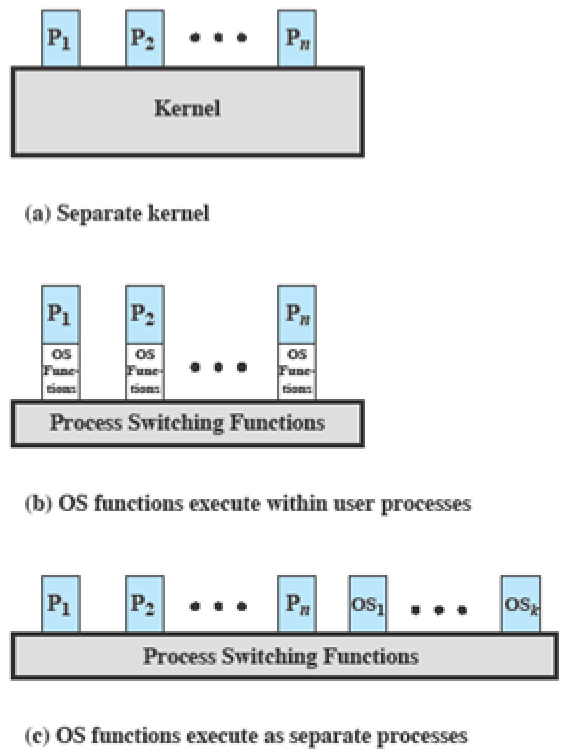
Change of Process State cont…

4. Select another process for execution
5. Update the process control block of the process selected
6. Update memory-management data structures
7. Restore context of the selected process

Is the OS a Process?

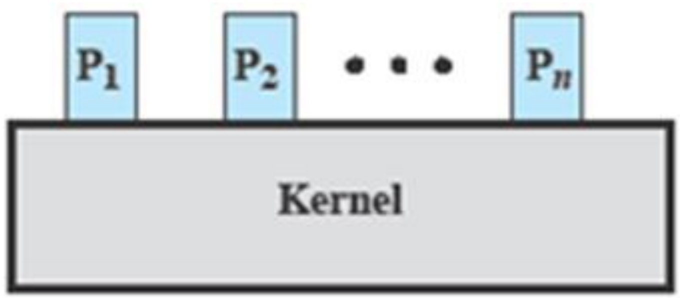
- If the OS is just a collection of programs and if it is executed by the processor just like any other program, is the OS a process?
- If so, how is it controlled?
  - Who (what) controls it?

Execution of the Operating System



Non-process Kernel

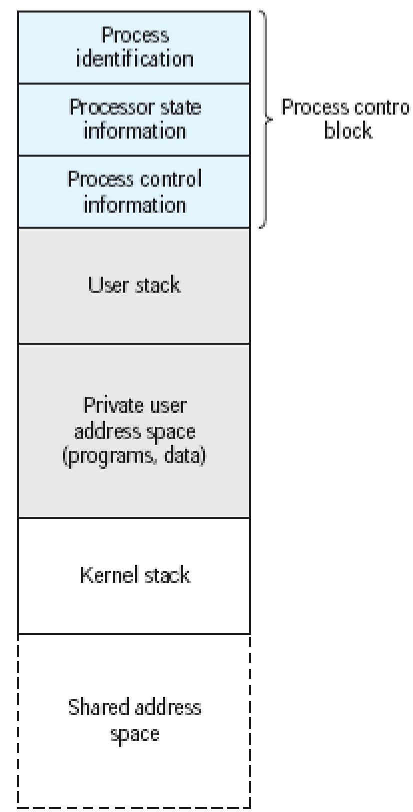
- Execute kernel outside of any process
- The concept of process is considered to apply only to user programs
  - Operating system code is executed as a separate entity that operates in privileged mode

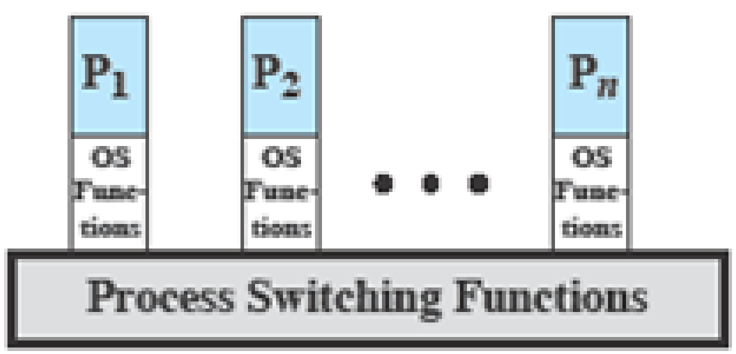


(a) Separate kernel

Execution *Within* User Processes

- Execution Within User Processes
  - Operating system software within context of a user process
  - No need for Process Switch to run OS routine





(b) OS functions execute within user processes

Process-based Operating System

- Process-based operating system
  - Implement the OS as a collection of system process

(c) OS functions execute as separate processes

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.

A Unix Process

- A process in UNIX is a set of data structures that provide the OS with all of the information necessary to manage and dispatch processes.
- See Table 3.10 which organizes the elements into three parts:
  - user-level context,
  - register context, and
  - system-level context.

Process Creation

- Process creation is by means of the kernel system call,fork( ).
- This causes the OS, in Kernel Mode, to:
  1. Allocate a slot in the process table for the new process.
  2. Assign a unique process ID to the child process.
  3. Copy of process image of the parent, with the exception of any shared memory.

Process Creation cont…

4. Increment the counters for any files owned by the parent, to reflect that an additional process now also owns those files.
5. Assign 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.
  - Transfer control to the child process
  - Transfer control to another process.