Chapter 3
Process Description and Control

(based on original slides by Pearson)

Req. of an OS wrt. Processes

- Interleave the execution of multiple processes to maximize processor utilization while providing reasonable response time
- Allocate resources to processes
- Support interprocess communication and user creation of processes

Previously Introduced Concepts

- Computer platform consists of a collection of hardware resources
  - Processor, main memory, timers, IOs
- Computer applications are developed to perform some task
  - Bespoke system vs general workstations
- Inefficient for applications to be written directly for a given hardware platform
  - Code reuse among applications
  - Need to manage multiprogramming (do you really want to put this into the application?)
  - Protect data, IO, etc (can you put this into the application?
- Some still believe otherwise
  - Consider porting assembly code?

Previously Introduced Concepts

- OS provides a convenient, feature rich, secure, and consistent interface for applications to use

- OS provides a uniform, abstract representation of resources that can be requested and accessed by application
  => a virtual machine

In This Set of Slides

- How does the OS manage execution of applications such that:
  - Resources made available to multiple applications
  - Processor is switched among multiple application
  - The processor and I/O devices can be used efficiently

Process (possible definitions)

- 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

Process (revisited)

- Consists of three components
  - An executable program
  - Associated data needed by the program
  - **Execution context of the program**
    - All information the operating system needs to manage the process

Process Elements

- Identifier
  - Usually abbreviated as PID
- State
  - E.g., running state
- Priority
  - relative to other processes
- Program counter
  - address of the next instruction

Process Elements

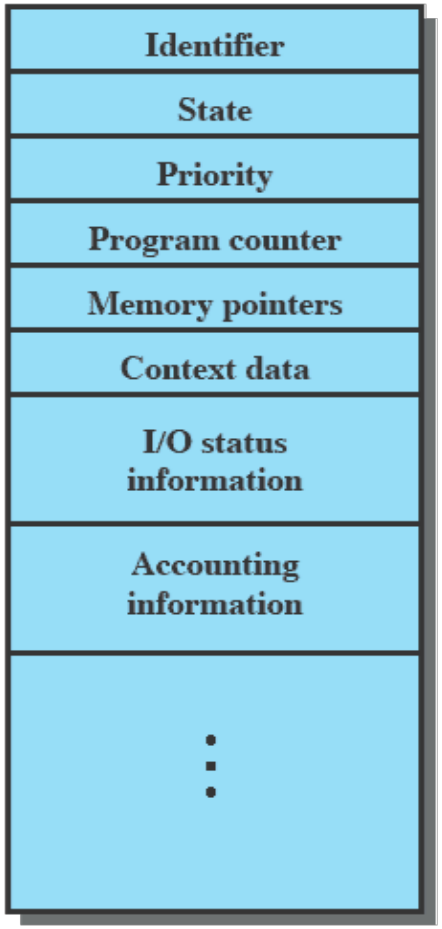
- Memory pointers
  - Pointers + shared memory blocks
- Context data
  - Registers, PSW
- I/O status information
  - Outstanding I/O requests, used I/O devices
- Accounting information
  - Amount of processor time, time limits, threads
  - Try out ‘ulimits’ and the ‘rabbit cage’

Process Control Block

- The data structure that contains the process elements
- Created and managed by the operating system
- Allows support for multiple processes

Question!

Process Control Block



Varies between Oss
=> Very small for embedded OS

Figure 3.1 Simplified Process Control Block

Trace of the Process

- Sequence of instruction that execute for a process
- Dispatcher switches the processor from one process to another
  - Compare to ‘dispatch table’

Example Execution (Setup)

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

(a) Trace of Process A **(b) Trace of Process B** **(c) Trace of Process C**

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

Example Execution (Memory Layout)

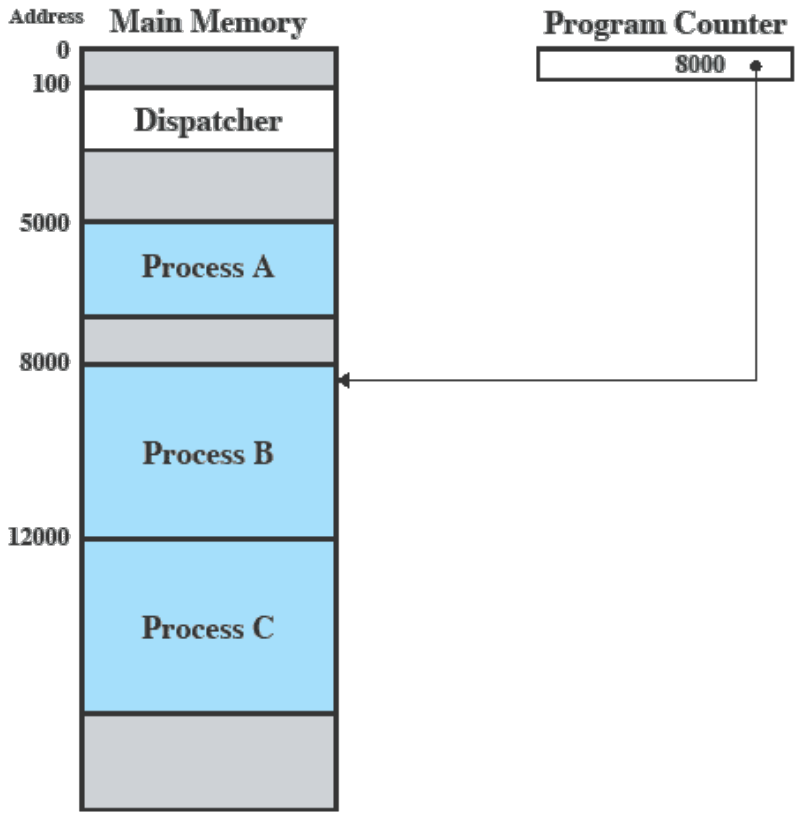


Figure 3.2 Snapshot of Example Execution (Figure 3.4) at Instruction Cycle 13

Combined Trace of All Processes

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

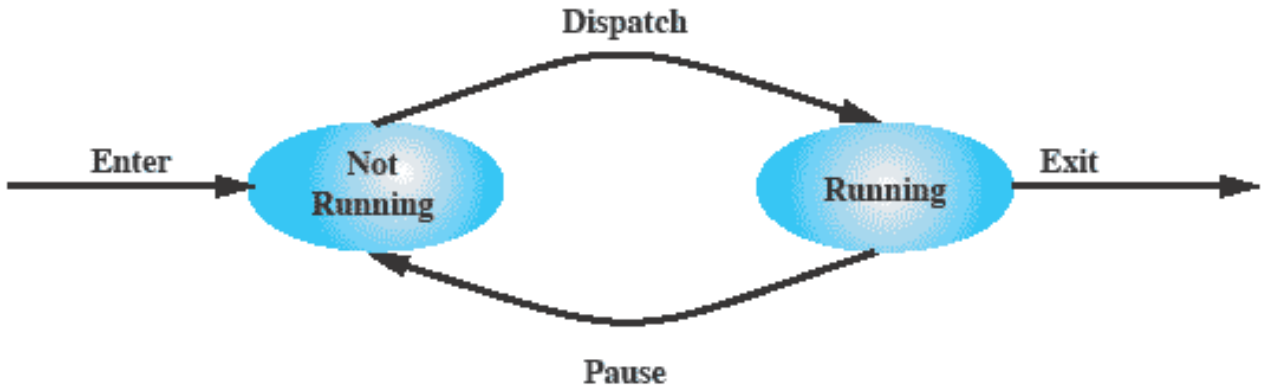
100 = Starting address of dispatcher program

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

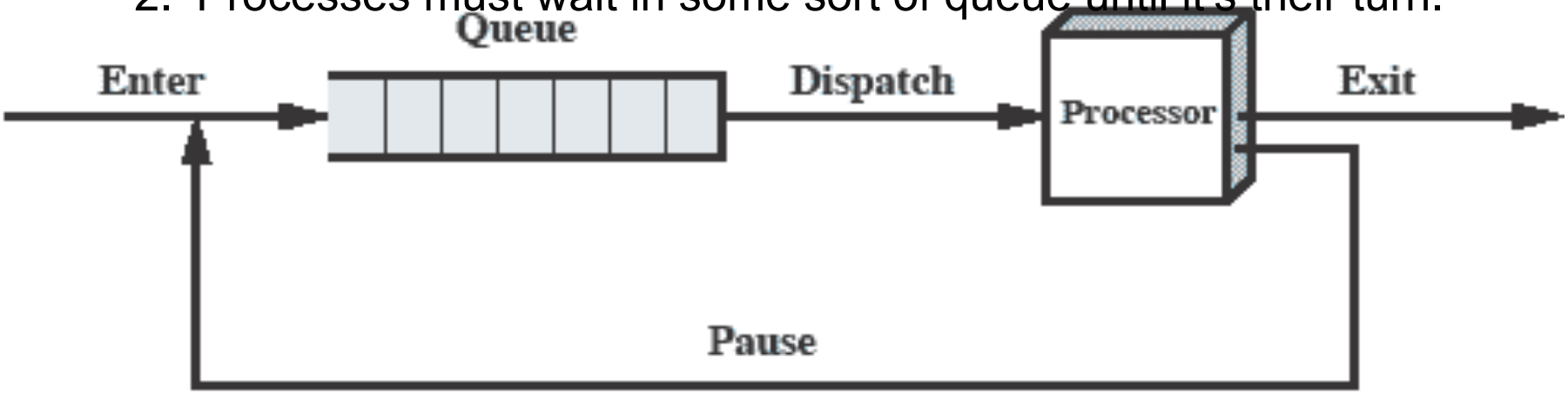
- Looking at the previous example: A process may be in one of two states
  - Running
  - Not-running



(a) State transition diagram

OS Structure (so far)

1. Store data about the process (process control block)
2. Processes must wait in some sort of queue until it's their turn.



(b) Queuing diagram

Process Creation

Table 3.1 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 spawning: a parent process explicitly creates a child process.
→ check this with 'ps'

Process Termination

Table 3.2 Reasons for Process Termination

Normal completion	The process executes an OS service call to indicate that it has completed running.
Time limit exceeded **ulimit**	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.

Explain: core dump

Process Termination

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 (for example, 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.

Explain: ‘kill –SIGTERM’ and others

Simple Queuing Mechanism is Inefficient

- ... because some processes are
  - Not-running but ready to execute
  - Not-running and block

- With a single queue: Dispatcher must scan list to find process not-running, ready, and in queue the longest
- With multiple queues: first pick the right queue and then go round robin.

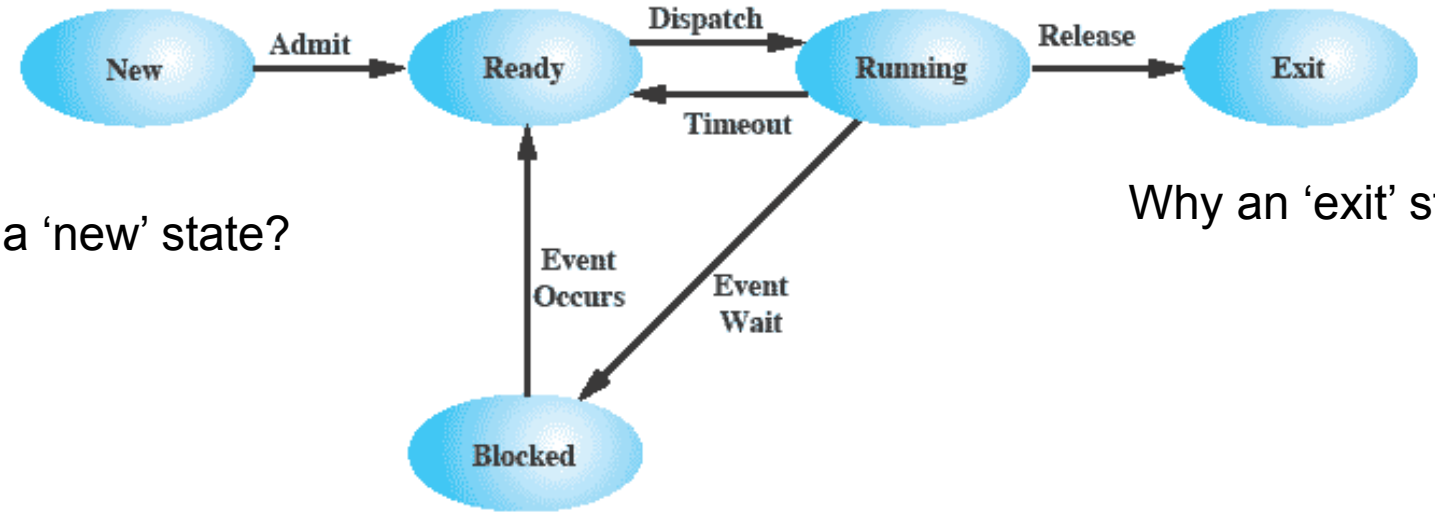
But what multiple queues should we have?

A Five-State Model

- Refine the not-running state:

- Running: the process currently executed
- Ready: a process that can be executed
- Blocked/Waiting: a process that cannot execute, because it waits for something
- New: a new process to enter the system
- Exit: a halted or aborted process

Five-State Process Model



Why a ‘new’ state?

Why an ‘exit’ state?

Figure 3.6 Five-State Process Model

Explain: preemption

Process States

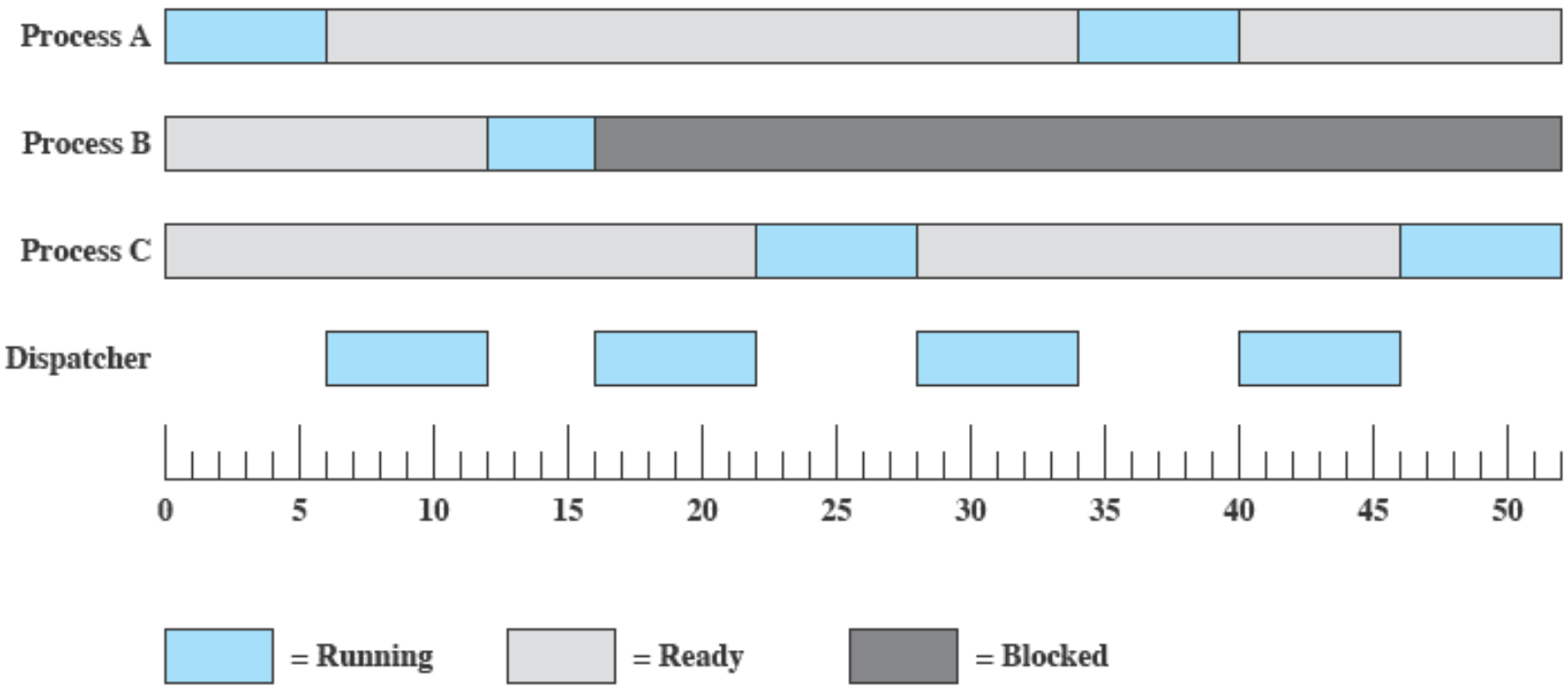
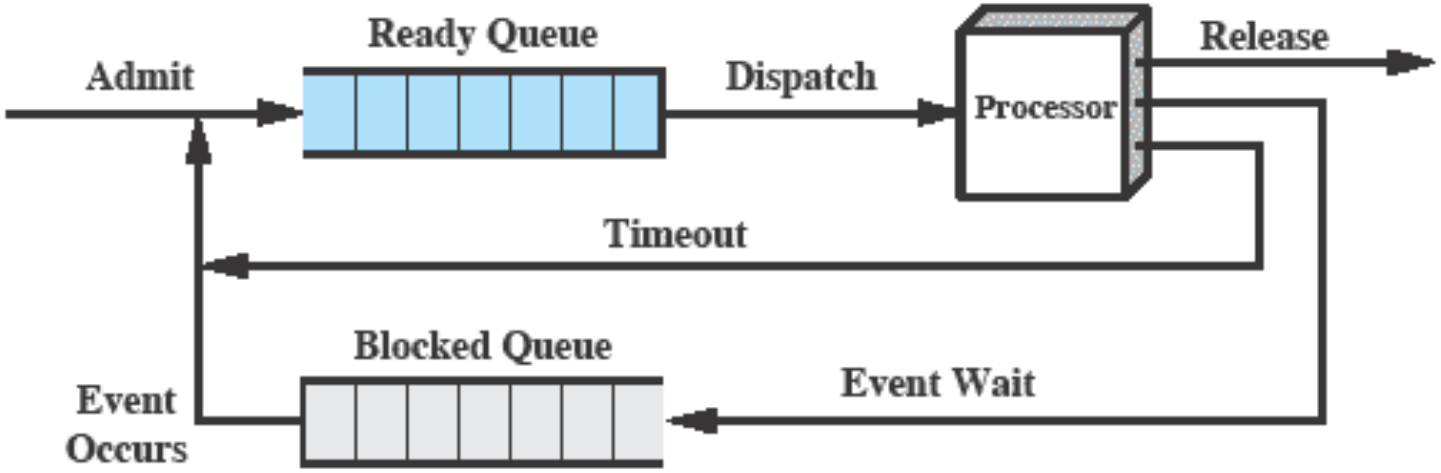


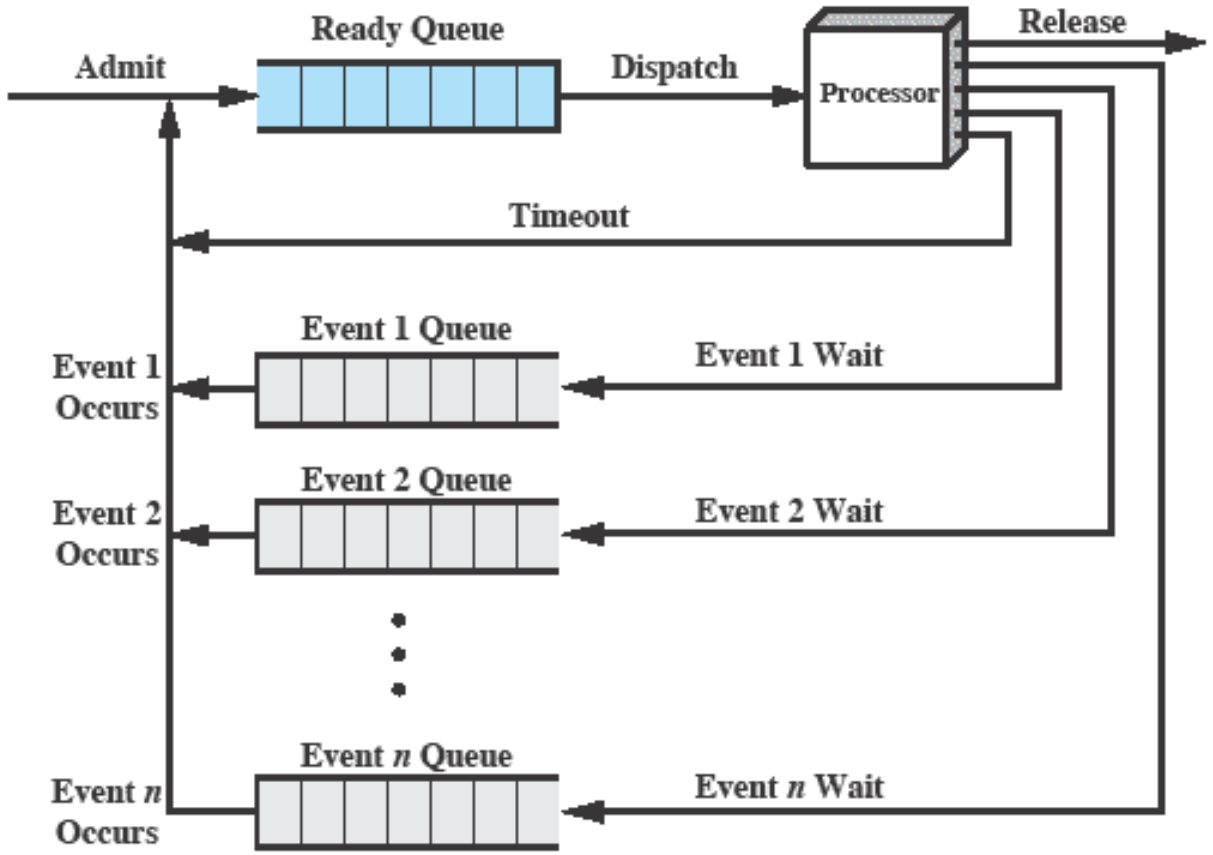
Figure 3.7 Process States for Trace of Figure 3.4

Using Two Queues



(a) Single blocked queue

Multiple Blocked Queues

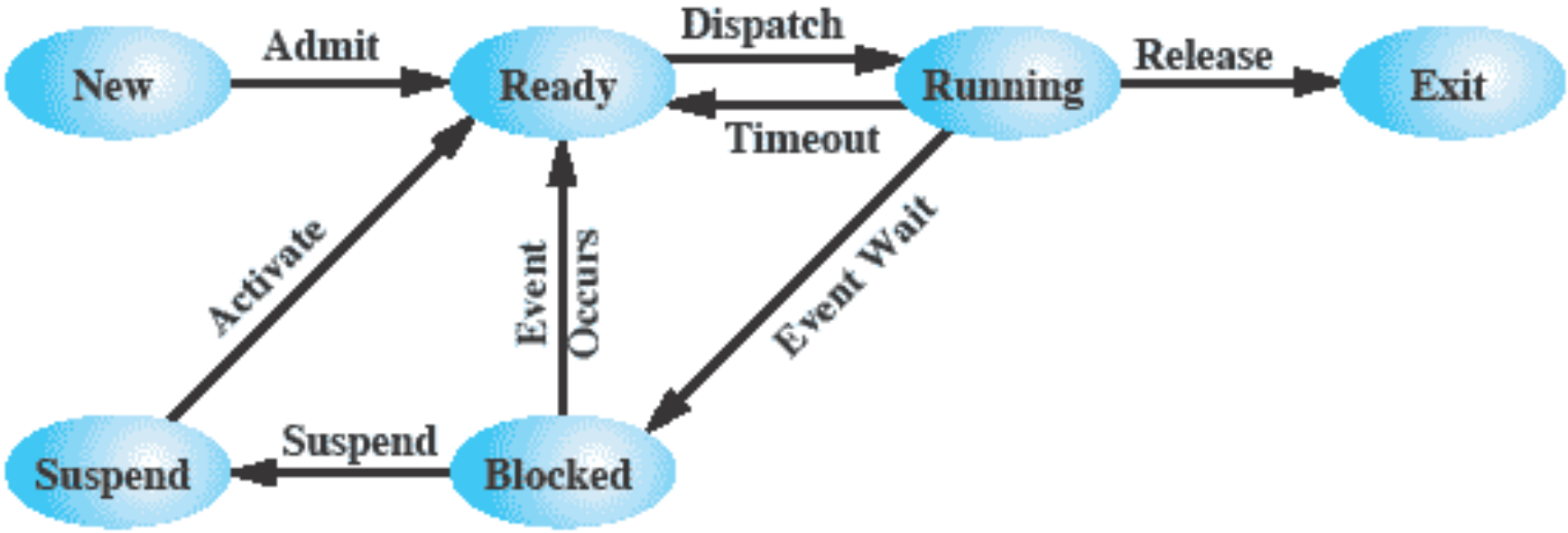


(b) Multiple blocked queues

Suspended Processes

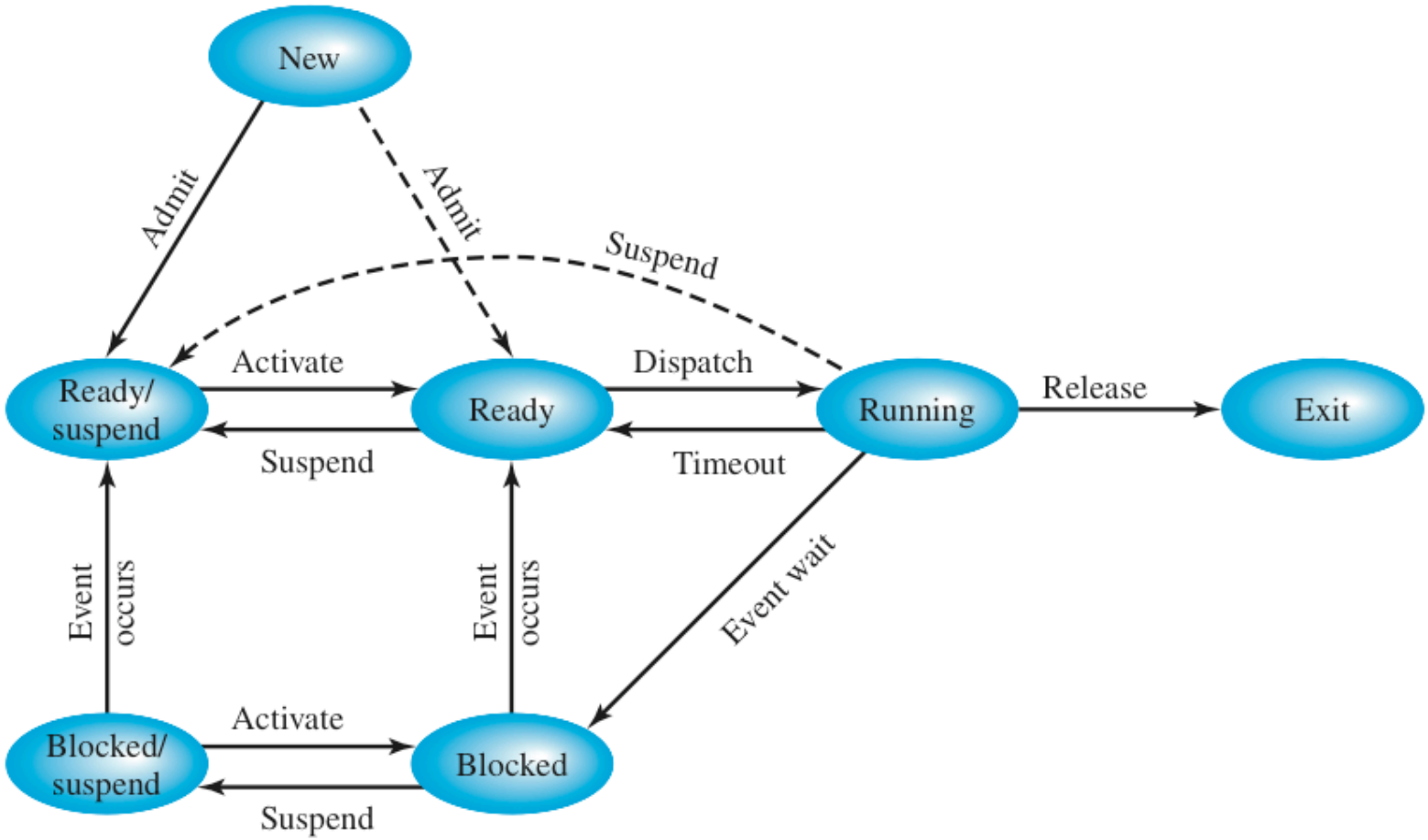
- Problem: Processor is faster than I/O so all processes could be waiting for I/O
- Solution: admit more processes
- Swap these processes to disk to free up more memory
- Blocked state becomes suspend state when swapped to disk
- Two new states
  - Blocked/Suspend
  - Ready/Suspend

One Suspend State



(a) With One Suspend State

Two Suspend States



Reason for Process Suspension

Table 3.3 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.

Notes!

Processes and Resources

Dashed line = Waiting

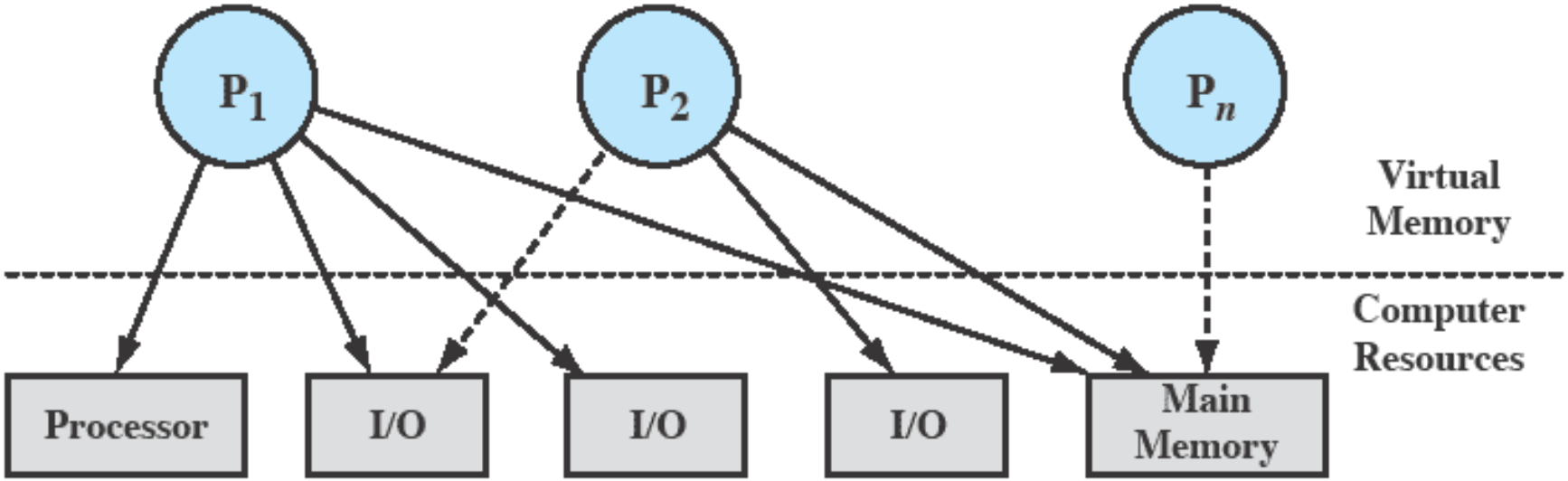


Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)

Operating System Control Structures

- Information about the current status of each process and resource
- Tables are constructed for each entity the operating system manages

Memory Tables

- … used to keep track of main & secondary (virtual) memory
- 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 I/O devices (see /proc directory)
- I/O device is available or assigned
- Status of I/O operation
- Location in main memory being used as the source or destination of the I/O transfer

File Tables

- Existence of files
- Location on secondary memory
- Current Status
- Attributes (e.g., rwxr--r--)
- Sometimes this information is maintained by a file management system

Process Table

- Where process is located in memory
- Attributes in the process image
  - Program
  - Data
  - Stack
  - Process control block (aka task control block, process descriptor, task descriptor)

  - Question: How/when does the OS create these tables?

Process Image

Table 3.4 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

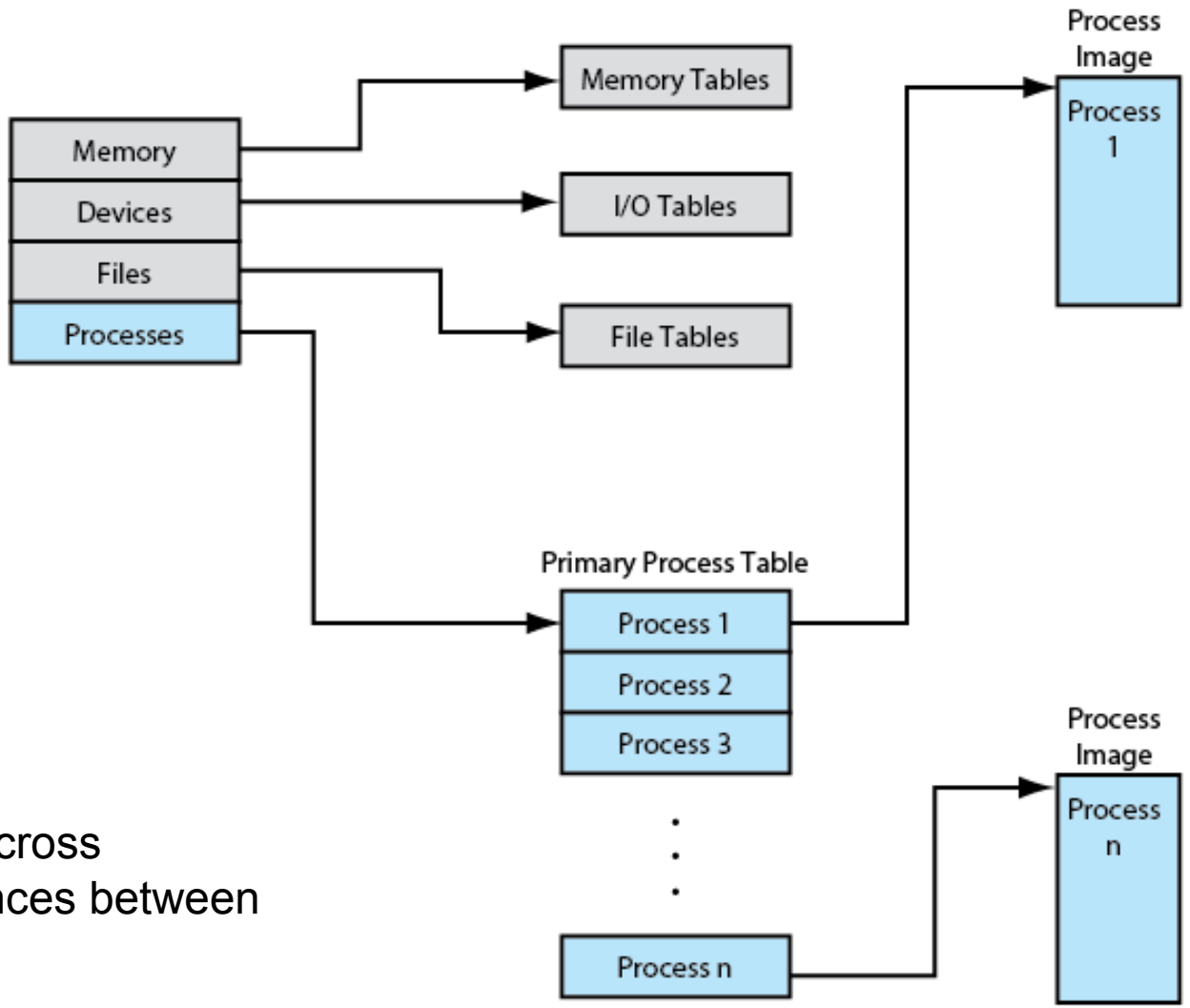
The program to be executed.

System Stack

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.

Process Control Block

Data needed by the operating system to control the process (see Table 3.5).



Lacks cross references between tables.

Figure 3.11 General Structure of Operating System Control Tables

Process Control Block

- Process identification
  - Identifiers
    - Numeric identifiers that may be stored with the process control block include
      - Identifier of this process (= unique key)
      - Identifier of the process that created this process (parent process)
      - User identifier
      - Defined by pid_t

Process Control Block

- 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.
    - Might be as low as only 1 working register

Process Control Block

- Processor State Information
  - 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

Process Control Block

- Processor State Information
  - 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 Block

- Process Control Information
  (… meta info for handling processes, differs for each OS → design criteria)
  - 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

    • **Deadline & execution time !!**

Process Control Block

- Process Control Information
  - 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.

Process Control Block

- Process Control Information
  - 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.

Process Control Block

- Process Control Information
  - 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.

Processor State Information

- 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 machines

Modes of Execution

- User mode
  - Less-privileged mode
  - User programs typically execute in this mode
- System mode, control mode, or kernel mode
  - More-privileged mode
  - Kernel of the operating system

- Can you think of why you want more modes?
- → granular access, virtual machines, sandboxing downloaded programs

Steps in Process Creation

1. Assign a unique process identifier
2. Allocate space for the process
3. Initialize process control block
4. Set up appropriate linkages
   - Ex: add new process to linked list used for scheduling queue
5. Create of expand other data structures
   - Ex: maintain an accounting file

When to Switch a Process

- Looks easy, but is quite tricky if you want to provide guarantees (e.g., bandwidth, performance, etc)
- Clock interrupt
  - process has executed for the maximum allowable time slice
- I/O interrupt (= I/O completed)
- Memory fault
  - memory address is in virtual memory so it must be brought into main memory

When to Switch a Process

- Trap
  - error or exception occurred
  - may cause process to be moved to Exit state
  - Used for debugging (ICD)
- Supervisor call
  - such as file open

Steps for a Process Switch

- Save context of processor including program counter and other registers
- Update the process control block of the process that is currently in the Running state
- Move process control block to appropriate queue – ready; blocked; ready/suspend
- Select another process for execution

Change of Process State

- Update the process control block of the process selected
- Update memory-management data structures
- Restore context of the selected process

Execution of the Operating System

- Non-process Kernel
  - Execute kernel outside of any process
  - Operating system code is executed as a separate entity that operates in privileged mode
  - Own memory, own call stack
  - The concept of processes only applies to user programs

Execution of the Operating System

- Execution Within User Processes
  - OS is primarily a collection of routines
  - Operating system software within context of a user process
  - Process executes in privileged mode when executing operating system code
  - Perform a "context switch" (mode switch) when entering a system call, but continue with the same process.

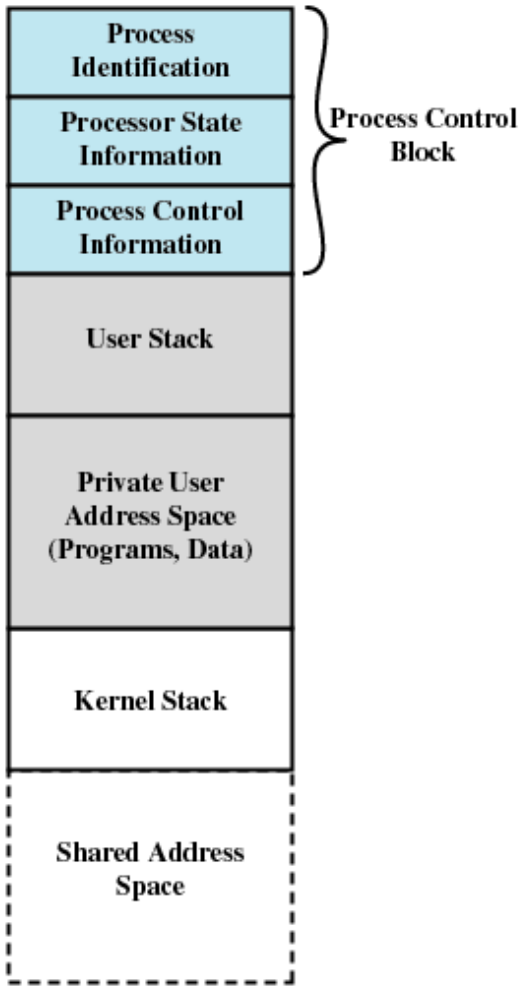
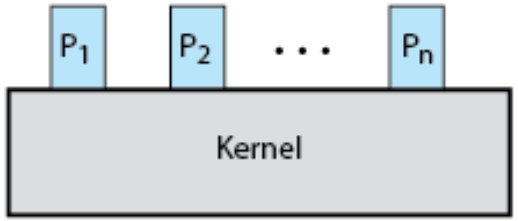


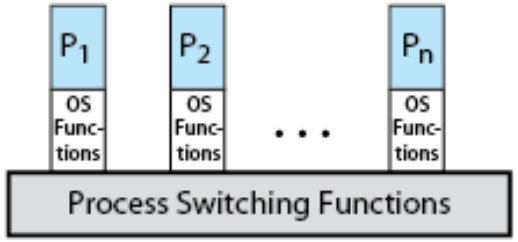
Figure 3.16 Process Image: Operating System Executes Within User Space

Execution of the Operating System

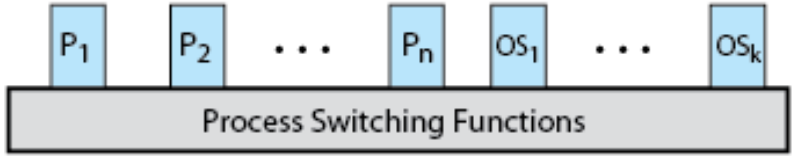
- Process-Based Operating System
  - Implement operating system as a collection of system processes
  - Useful in multi-processor or multi-computer environment
  - Good choice for real-time systems



(a) Separate kernel



(b) OS functions execute within user processes

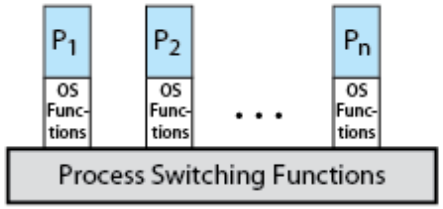


(c) OS functions execute as separate processes

Figure 3.15 Relationship Between Operating System and User Processes

UNIX SVR4 Process Management

- Most of the operating system executes within the environment of a user process



(b) OS functions execute within user processes

UNIX Process States

Table 3.9 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.

UNIX Process Image

Table 3.10 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

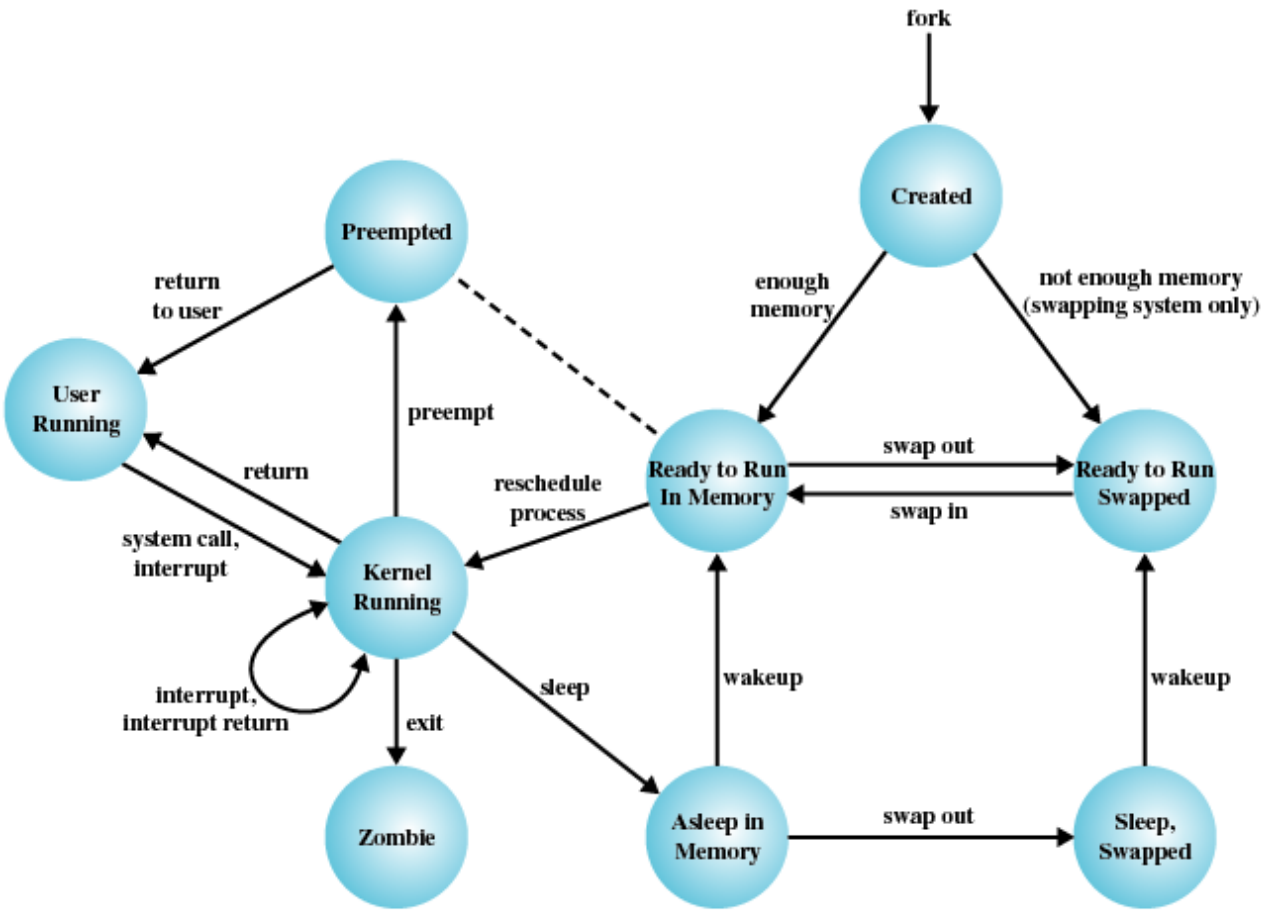


Figure 3.17 UNIX Process State Transition Diagram