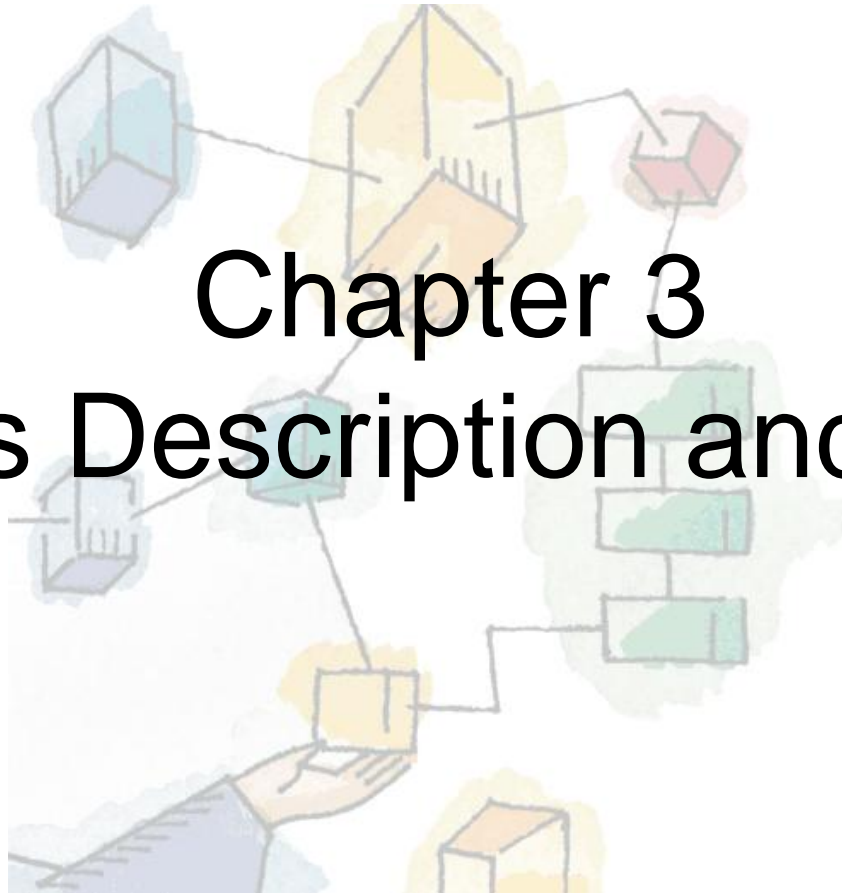


*Operating Systems:  
Internals and Design Principles, 6/E*  
William Stallings

# Chapter 3

## Process Description and Control





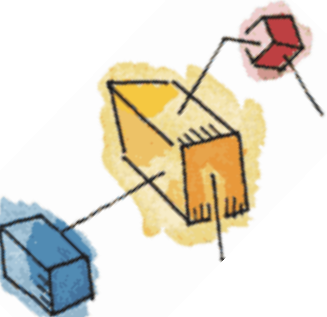
# Roadmap



How are processes represented and controlled by the OS.

- **Process states** which characterize the behaviour of processes.
- **Data structures** used to manage processes.
- Ways in which the OS uses these data structures to control process execution.
- Discuss process management in UNIX SVR4.





# Requirements of an Operating System

- *Fundamental Task: **Process Management***
- The Operating System must
  - **Interleave** the execution of multiple processes
  - Allocate **resources** to processes, and protect the resources of each process from other processes,
  - Enable processes to share and exchange **information**,
  - Enable **synchronization** among processes.

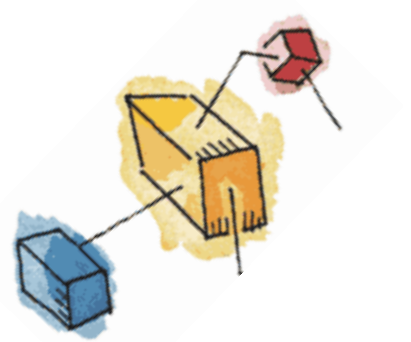




# What is a “*process*”?

- “*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”





# Process Elements

- A process is comprised of:
  - Program **code**
  - 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** – Unique ID to distinguish from others
  - **State** – Current status of process
  - **Priority** – Level relative to others
  - **Program counter** – Address of next instruction
  - **Memory pointers** – Pointers to code and data
  - **Context data** – Data present in CPU registers
  - **I/O status information** – Pending requests, list
  - **Accounting information** – Statistics
- These elements are stored in a data structure called PCB





# Process Control Block

- Contains the process elements
- **Created** and **managed** by the operating system
- Hence we can say that,
  - $PCB + Code + Data = Process$
- PCB enables the OS to support multiple processes
  - As it maintains the required information

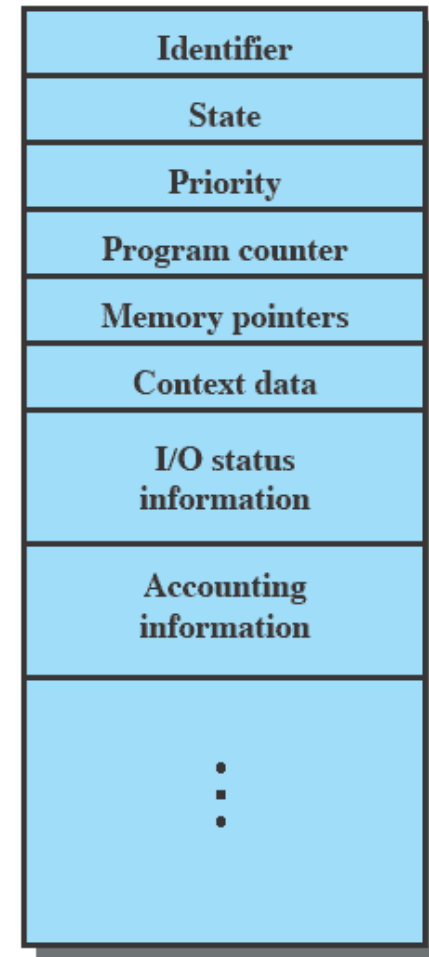


Figure 3.1 Simplified Process Control Block



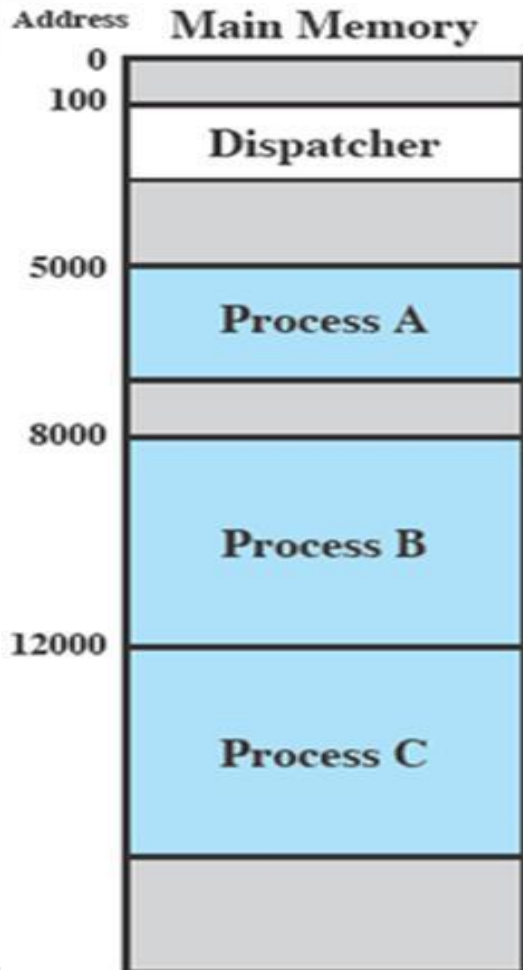
# Trace of the Process

- The behavior of an individual process is shown by listing the sequence of instructions that are executed
  - This list is called a **Trace**
- When there are multiple processes present in main memory, a separate program is required for process selection
  - **Dispatcher** is a small program which switches the processor from one process to another





# Process Execution



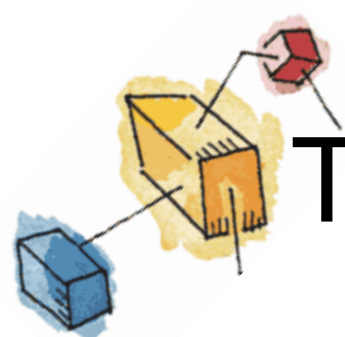
- Consider **three processes**
- All are **in memory** (plus the dispatcher)
- Suppose dispatcher has selected process A, it will run for fixed time
- After the fixed time is completed, dispatcher will run again and picks another process
- And so on...



# Roadmap

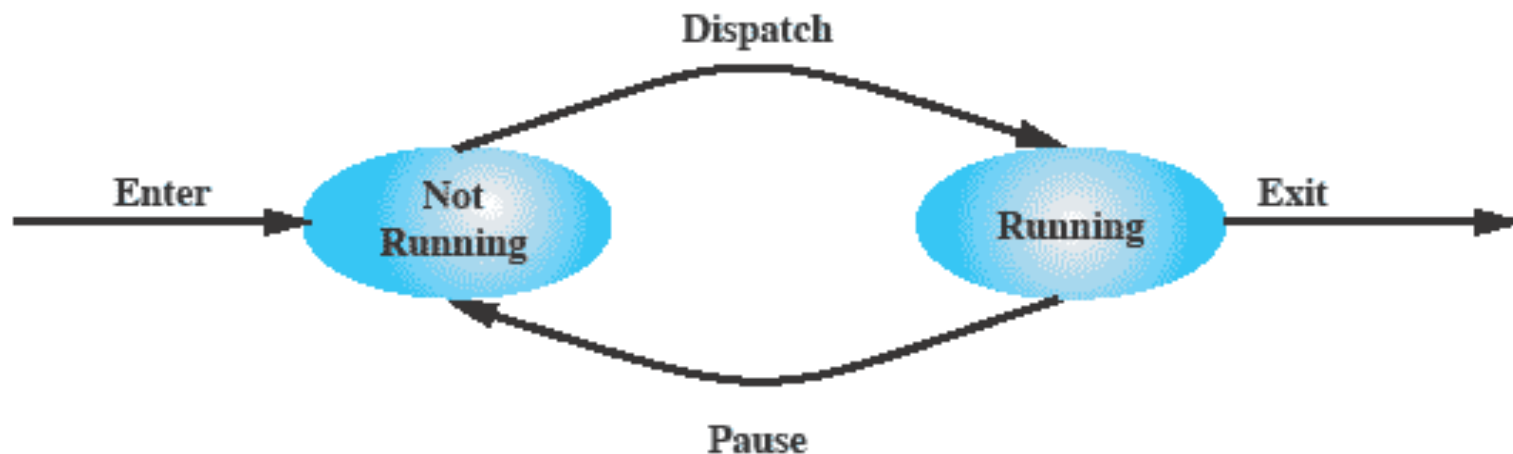
- How are processes represented and controlled by the OS.
- – **Process states** which characterize the behaviour of processes.
- **Data structures** used to manage processes.
- Ways in which the OS uses these data structures to control process execution.
- Discuss process management in UNIX SVR4.





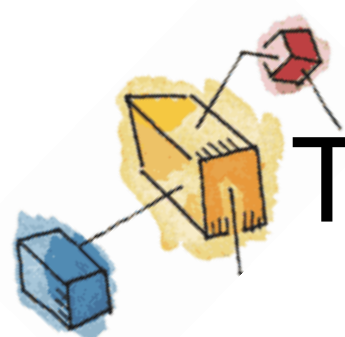
# Two-State Process Model

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



(a) State transition diagram



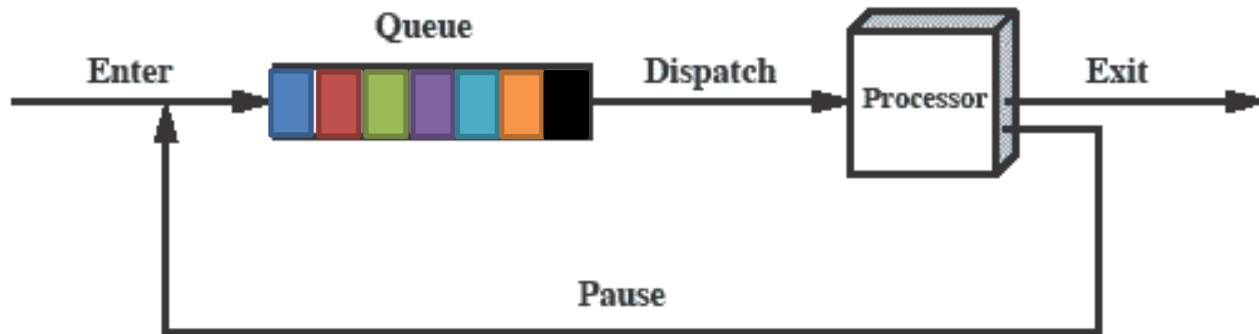


# Two-State Process Model

- When the OS creates a new process
  - **PCB** is created
  - Process is assigned “not running” **state**
  - Process **waits** for opportunity
- From **time to time**
  - Currently running process will be interrupted
  - Dispatcher selects another process
  - Currently running process moves from “Running” to “Not Running”



# Queuing Diagram



(b) Queuing diagram

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

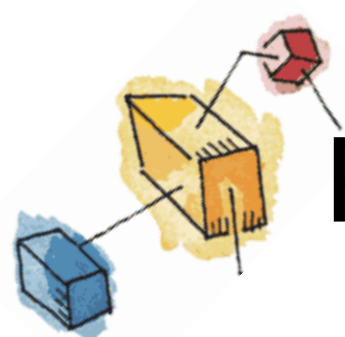




# Queuing Diagram

- Processes that are not running must be stored in a **queue**
  - Waiting for their turn
- Each **entry in queue** is a **pointer to PCB** for a process
- **Dispatcher's Role:**
  - **Transfer** the interrupted process to the queue
  - **Discard** the completed/aborted process
  - **Select** another process from queue to execute





# 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 created 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



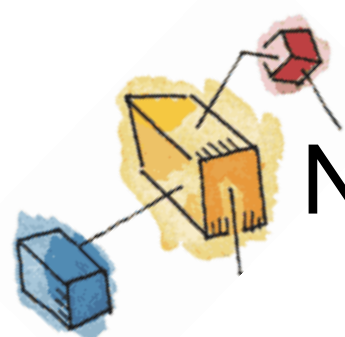




# Process Termination

- 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





# Need of 5-state Process Model

- In **2-state** process model, dispatcher selects process in **FIFO** manner
- But the process which is getting selected might be **blocked**
- Hence the **dispatcher** needs to find such a process which is
  - *Not blocked*
  - *Has been there for longest*
- So “Not Running” state is divided in
  - **Ready**
  - **Blocked**



# Five-State Process Model

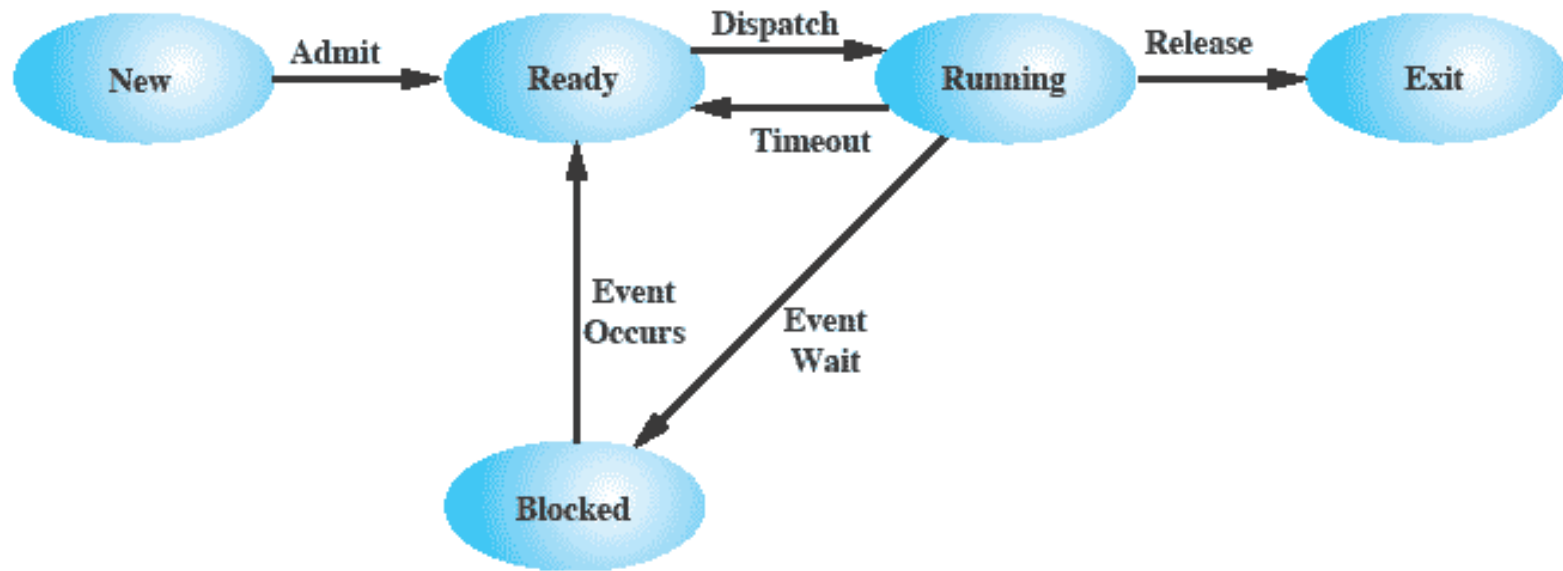
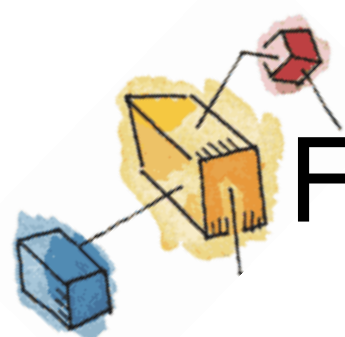


Figure 3.6 Five-State Process Model



# Five-State Process Model

- **New:**
  - Process is just created, but not yet admitted by the OS (in main memory)
- **Ready:**
  - Prepared to execute when given opportunity
- **Running:**
  - Currently being executed
- **Blocked / Waiting:**
  - Waiting for an event
- **Exit:**
  - Released from the pool of executable processes by OS





# Five-State Process Model : Transitions

- **Running -> Ready:**
  - Maximum time allocated is over or
  - Preemption on priority
  - Background process (automatically released CPU)
- **Running -> Blocked**
  - Waiting for an event (OS service, I/O, other process)
- **Blocked -> Ready**
  - When event occurs
- **Ready -> Exit and Blocked -> Exit**
  - Parent terminates child / Parent Terminated



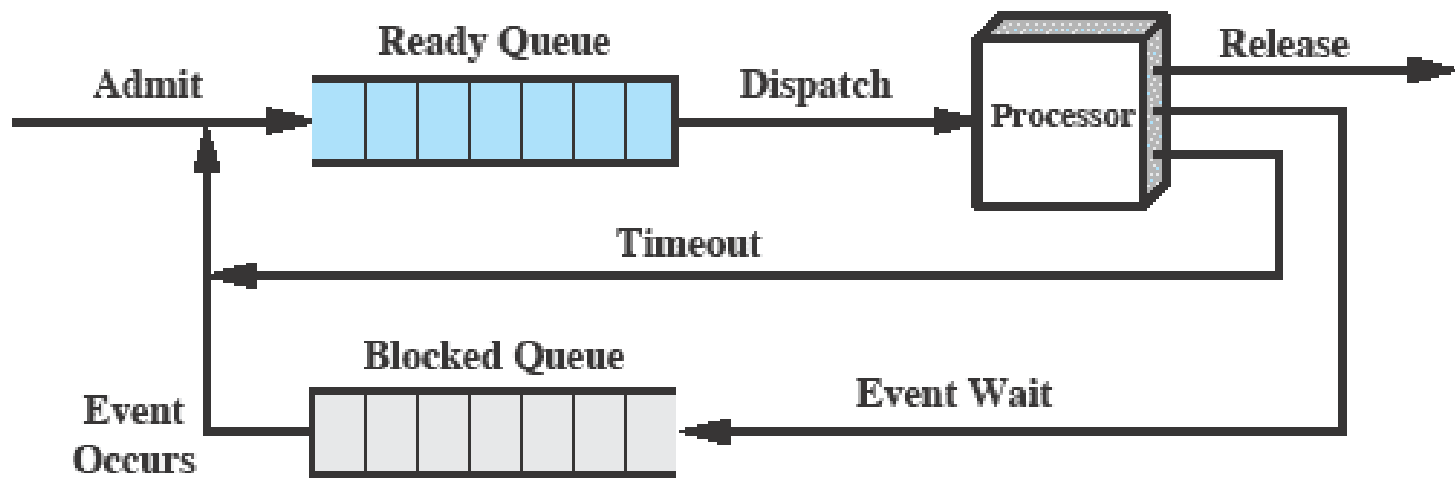


# Five-State Process Model : Transitions

- **Null -> New**
  - A new process is created to execute a program.
- **New -> Ready**
  - When OS is prepared to take additional process in main memory
- **Ready -> Running**
  - OS chooses one of the ready processes to run (Dispatcher)
- **Running -> Exit**
  - Process terminated on completed or aborted conditions



# Using Two Queues



(a) Single blocked queue



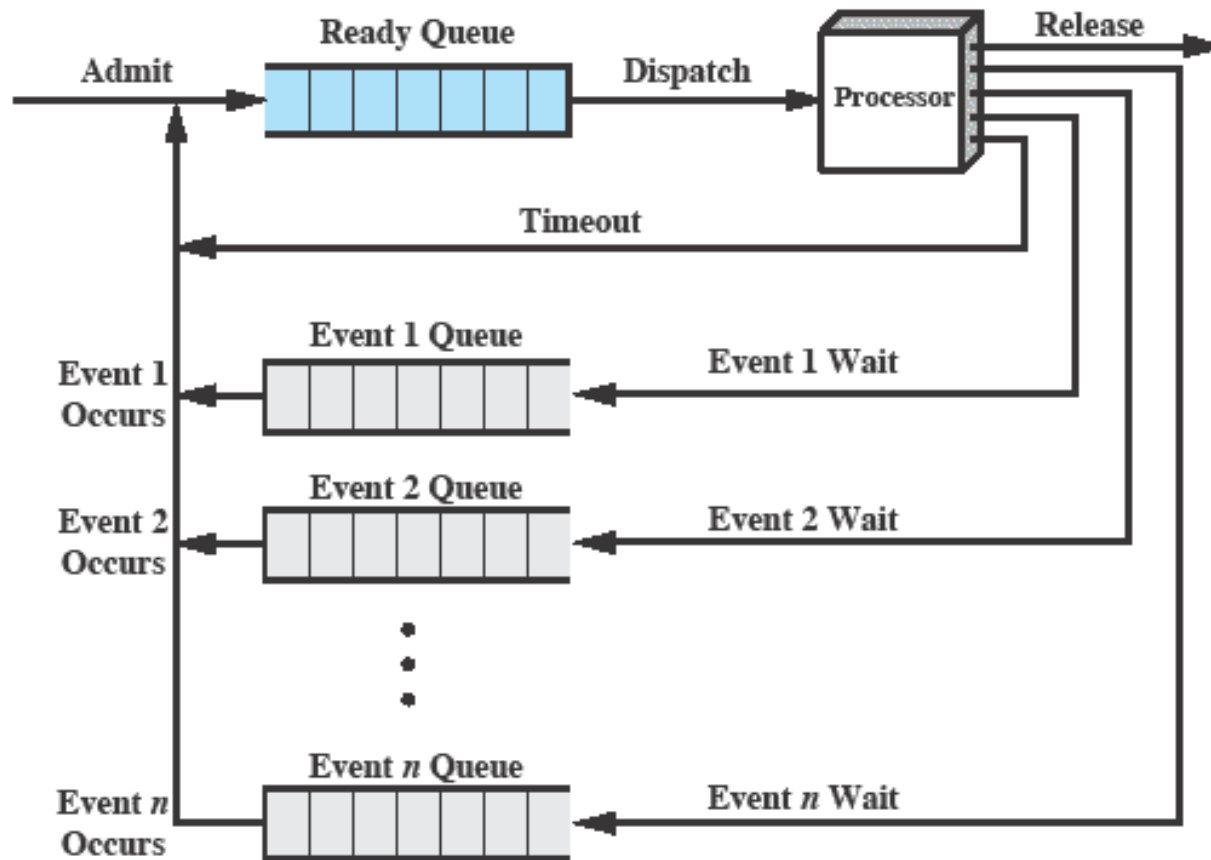
# Using Two Queues

- **Issue:**
  - When an event occurs, the OS has to scan the entire blocked queue
    - Which may have 100-1000 processes
- **Solution:**
  - Keep multiple blocked queues – one for each type of event
- If priority based scheme is being used, then multiple ready queues may be needed





# Multiple Blocked Queues



(b) Multiple blocked queues



# Suspended Processes

- Processor is **faster than** I/O so **all processes** could be **waiting for I/O**
  - **Idle Processor!!**
    - Swap these processes to disk to free up more memory -- Swapping
- **Two possibilities:**
  - One new state can be added
    - **Suspend**
  - Two new states
    - **Blocked/Suspend**
    - **Ready/Suspend**



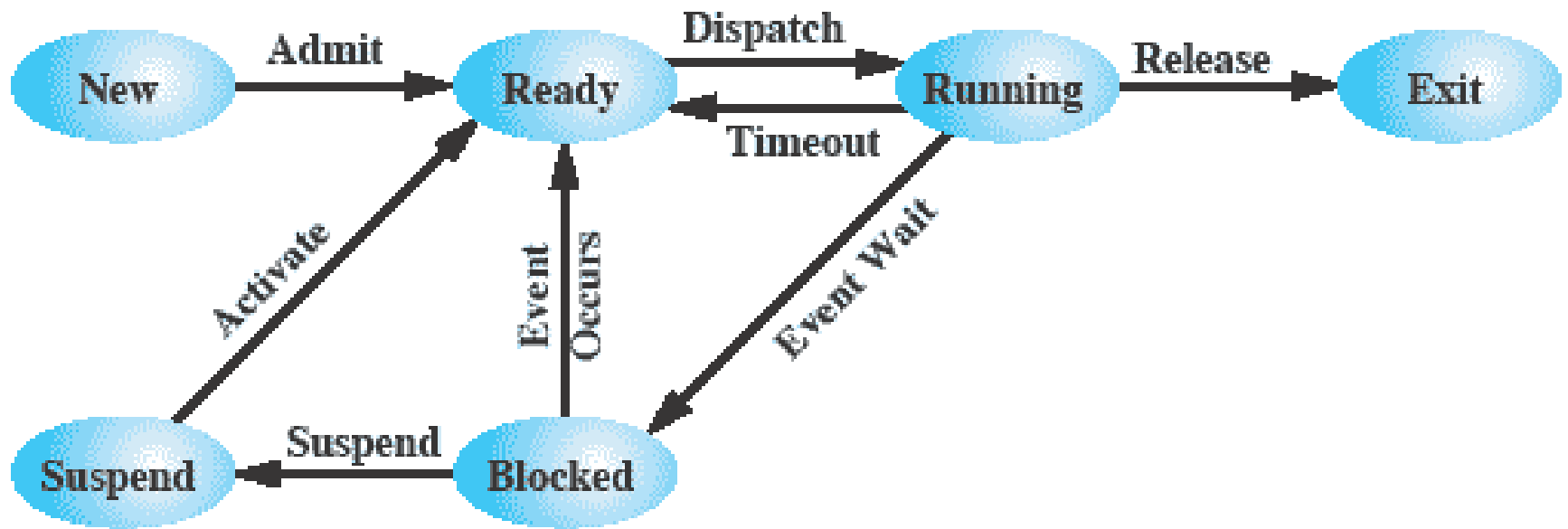


# Definition of States

- **Ready:**
  - Process in main memory, ready to execute
- **Blocked:**
  - Process in main memory, waiting
- **Ready/Suspend:**
  - Process in secondary memory, ready to execute (after loaded in main memory)
- **Blocked/Suspend:**
  - Process in secondary memory, waiting

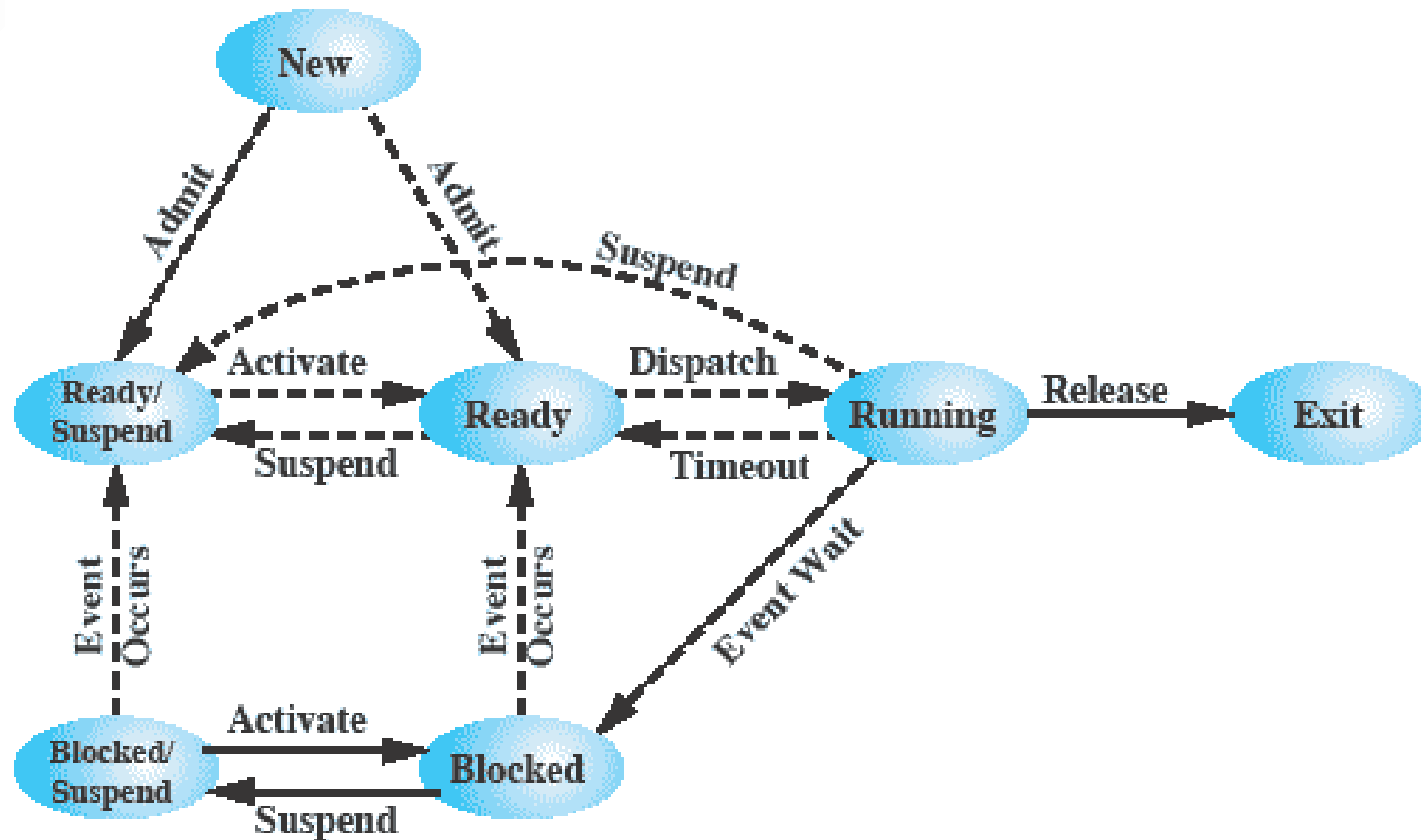


# One Suspend State



(a) With One Suspend State

# Two Suspend States



(b) With Two Suspend States



# Transitions

- **New -> Ready and New -> Ready/Suspend:**
  - First one applies in normal situation
  - Second one applies for house keeping
- **Blocked -> Blocked/Suspend:**
  - If no ready processes are available
- **Blocked/Suspend -> Ready/Suspend:**
  - When the event occurs
- **Ready/Suspend -> Ready:**
  - No ready process in main memory OR
  - The process has higher priority than any of the ready processes





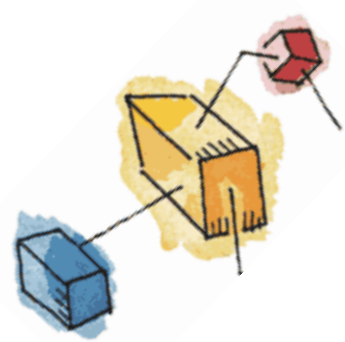
# Transitions

- **Ready -> Ready/Suspend:**
  - Normally OS would suspend a blocked process
  - But it may suspend a Ready process if it is needed to free memory for existing process
  - OS may suspend a lower priority ready process than higher priority blocked process
- **Blocked/Suspend -> Blocked:**
  - Already blocked – Why move?
  - Priority of Blocked/Suspended process is higher than Ready/Suspended processes



# Transitions

- Running -> Ready/Suspend:
  - Normally, Running -> Ready
  - But a ready process can be moved to secondary memory when a higher priority process on Blocked/Suspend queue is unblocked
- Any -> Exit:
  - Terminate on completion or fault







# Reasons 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.

**Table 3.3 Reasons for Process Suspension**





# Roadmap

- How are processes represented and controlled by the OS.
- **Process states** which characterize the behaviour of processes.
- **Data structures** used to manage processes.
- Ways in which the OS uses these data structures to control process execution.
- Discuss process management in UNIX SVR4.



# Processes and Resources

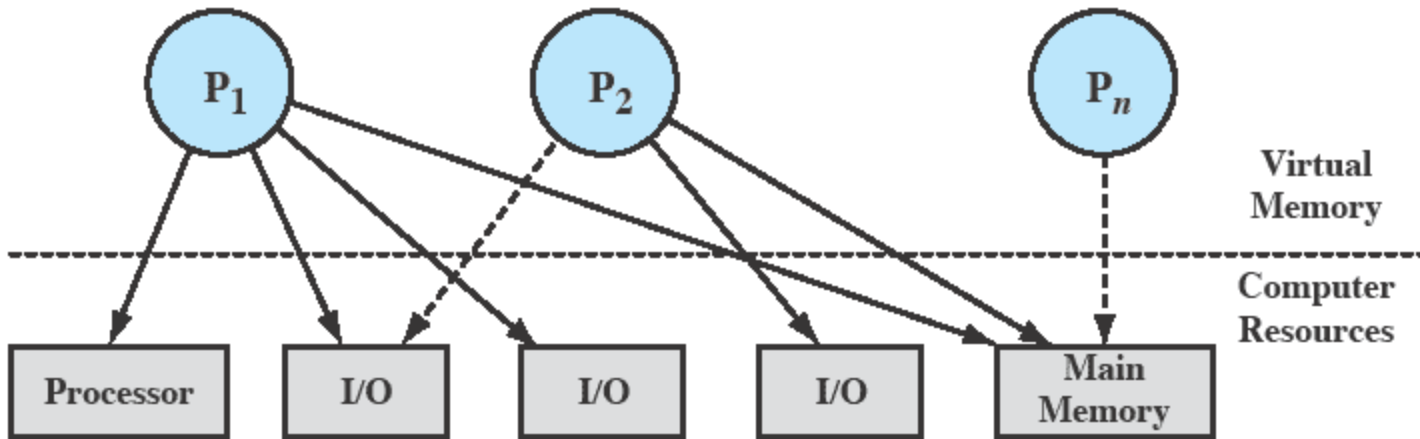


Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)



# 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.
  - Memory Tables
  - I/O Tables
  - File Tables
  - Process Tables



# OS Control Tables

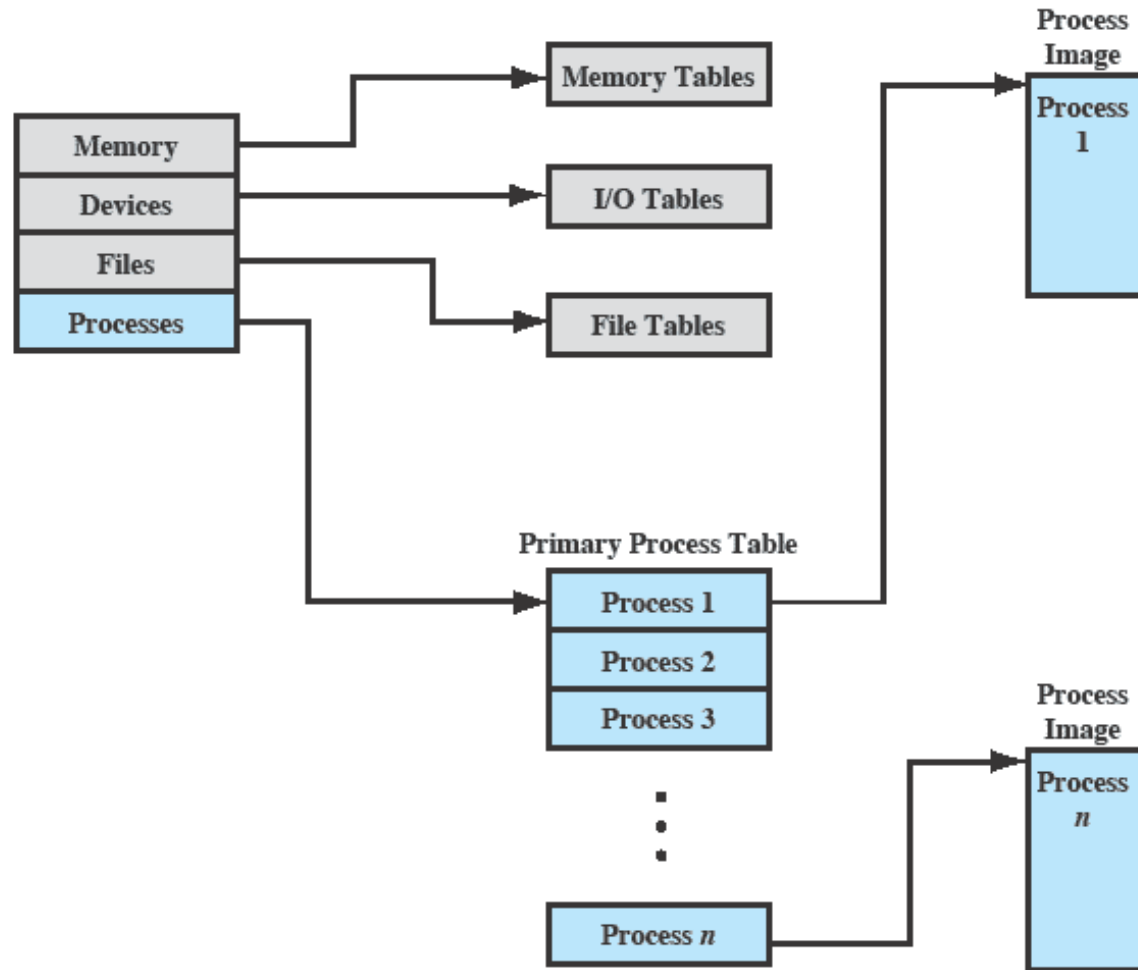


Figure 3.11 General Structure of Operating System Control Tables



# Memory Tables

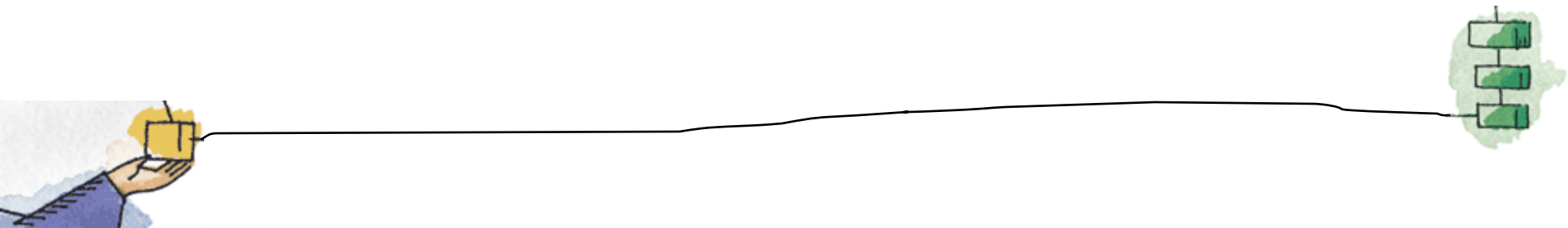
- Memory tables are used to keep track of both main and secondary memory.
- **Must include this information:**
  - Allocation of main memory to processes
  - Allocation of secondary memory to processes
  - Protection attributes for access to shared memory regions
  - Information needed to manage virtual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



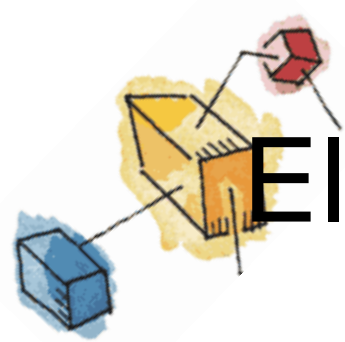




# Process Tables

- To **manage processes** the OS needs to know details of the processes
  - Current state
  - Process ID
  - Location in memory
  - Etc
- The required information is maintained using process image





# Elements of Process Image

- **User Data:**
  - The modifiable part of user space.
- **User Program:**
- The program to be executed
- **Stack:**
- Each process has one or more stack associated with it
- **Process Control Block**





# 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





# Roadmap

- How are processes represented and controlled by the OS.
- **Process states** which characterize the behaviour of processes.
- **Data structures** used to manage processes.
- Ways in which the OS uses these data structures to control process execution.
- Discuss process management in UNIX SVR4.

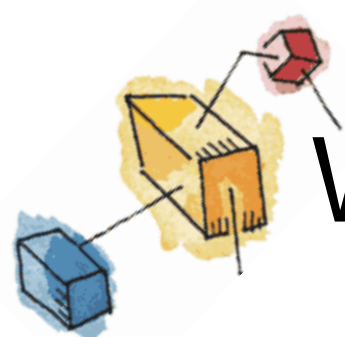




# Modes of Execution

- Most processors support at least two modes of execution for protection of OS and other key tables
- **User mode**
  - Less-privileged mode
  - User programs typically execute in this mode
- **System mode / Kernel Mode / Control Mode**
  - More-privileged mode
  - Certain instructions can only be executed in this mode only ( Process, memory, I/O Management etc)





# When mode is changed?

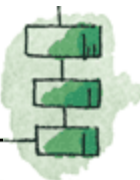
- When a **user program** requests OS service
- An **interrupt** needs to execute OS routine
- On return, mode is set back to User
- A bit in **PSW** indicates the current mode





# Process Creation

- Once the OS decides to create a new process, it:
  - **Assigns a unique process identifier**
    - An entry is added in Primary Process Table
  - **Allocates space for the process**
    - For all elements of process image
  - **Initializes process control block**
    - Pointers, state etc.
  - **Sets up appropriate linkages**
    - Puts the process in appropriate list (queue)
  - **Creates or expand other data structures**
    - Creates additional data structures (e.g. for accounting)





# 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 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

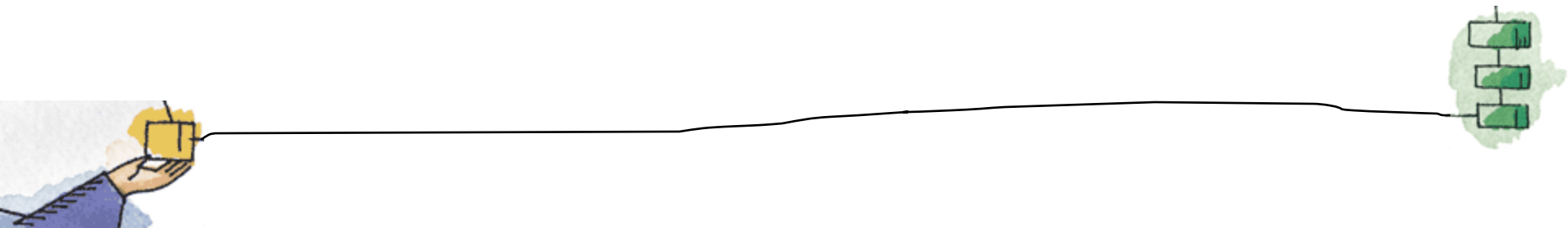






# Change of Process State ...

- 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?

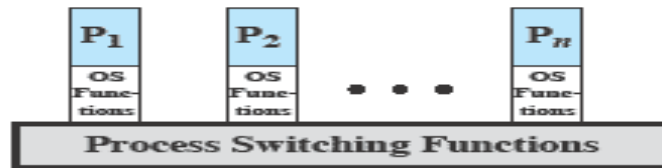
- If the OS is just a **collection of programs** and if it is **executed by the processor** just like any other program, is the OS a process?
- If so, how is it controlled?
  - Who (what) controls it?



# Execution of the Operating System



(a) Separate kernel



(b) OS functions execute within user processes



(c) OS functions execute as separate processes

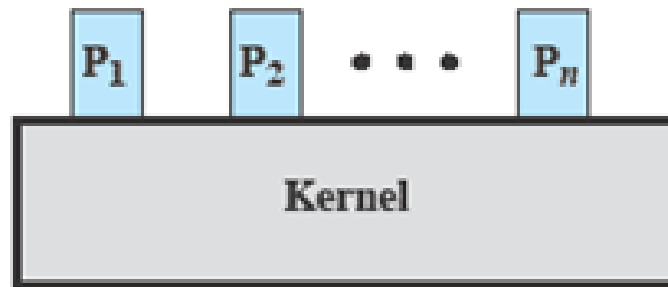
**Figure 3.15 Relationship Between Operating System and User Processes**





# 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





# Non-process Kernel

- When the currently running process is interrupted or issues a supervisor call
  - Context of process is saved and control is passed to kernel
- As OS executes as separate entity, it has own region of memory and system stack
- This approach was used by **older OS**



# Execution *Within* User Processes

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

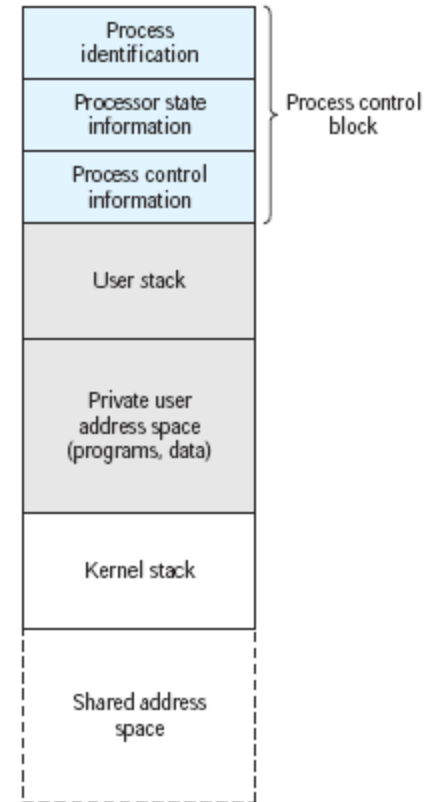
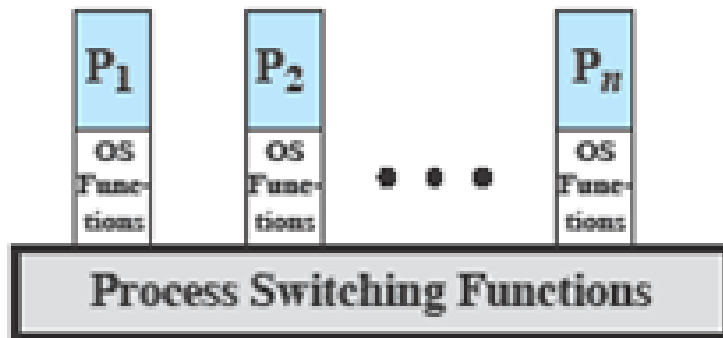


Figure 3.16 Process Image: Operating System Executes within User Space

(b) OS functions execute within user processes





# Execution *Within* User Processes

- OS works as a collection of routines, within the environment of user process
  - Suitable for PCs
- A **separate kernel stack** is given to the process while it is in **kernel mode**
- When an **interrupt or supervisor call** happens
  - Processor is placed in kernel mode and control is passed to OS
  - But the execution continues in current user process only
  - Hence only **mode switch** is needed







# Process-based Operating System

- **Process-based operating system**
  - Implement the OS as a collection of system process
- **Advantage:**
  - Modularity



(c) OS functions execute as separate processes





# Security Issues

- An OS associates a set of privileges with each process.
- A key security issue in the design of any OS is to prevent anything (user or process) from gaining unauthorized privileges on the system
  - Especially - from gaining root access.





# System access threats

- **Intruders**
  - Masquerader (outsider) – Unauthorized access
  - Misfeasor (insider) – Accesses restricted aspects
  - Clandestine user (outside or insider) – Gains supervisory control
- **Malicious software (malware)**
  - Parasitic (needs host)
  - Independent





# Roadmap

- How are processes represented and controlled by the OS.
- **Process states** which characterize the behaviour of processes.
- **Data structures** used to manage processes.
- Ways in which the OS uses these data structures to control process execution.
- – Discuss process management in UNIX SVR4.

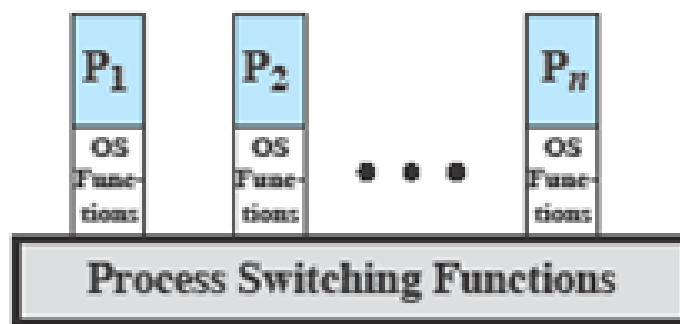




# Unix SVR4

## System V Release 4

- **System Processes** –
- Kernel mode only
- **User Processes**
  - User mode to execute user programs and utilities
  - Kernel mode to execute instructions that belong to the kernel.



(b) OS functions execute within user processes



# UNIX Process State Transition Diagram

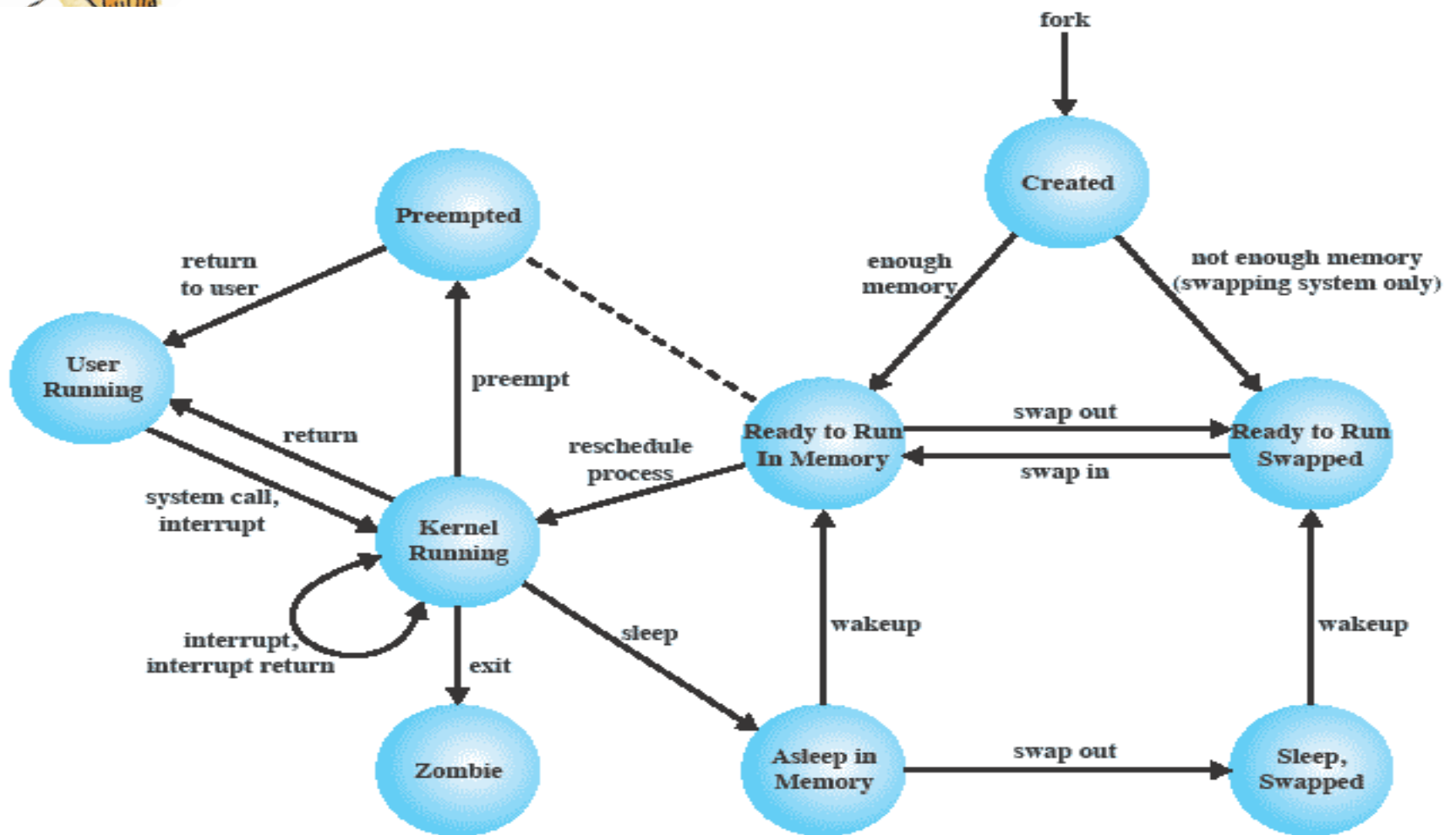


Figure 3.17 UNIX Process State Transition Diagram



# UNIX Process States

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