

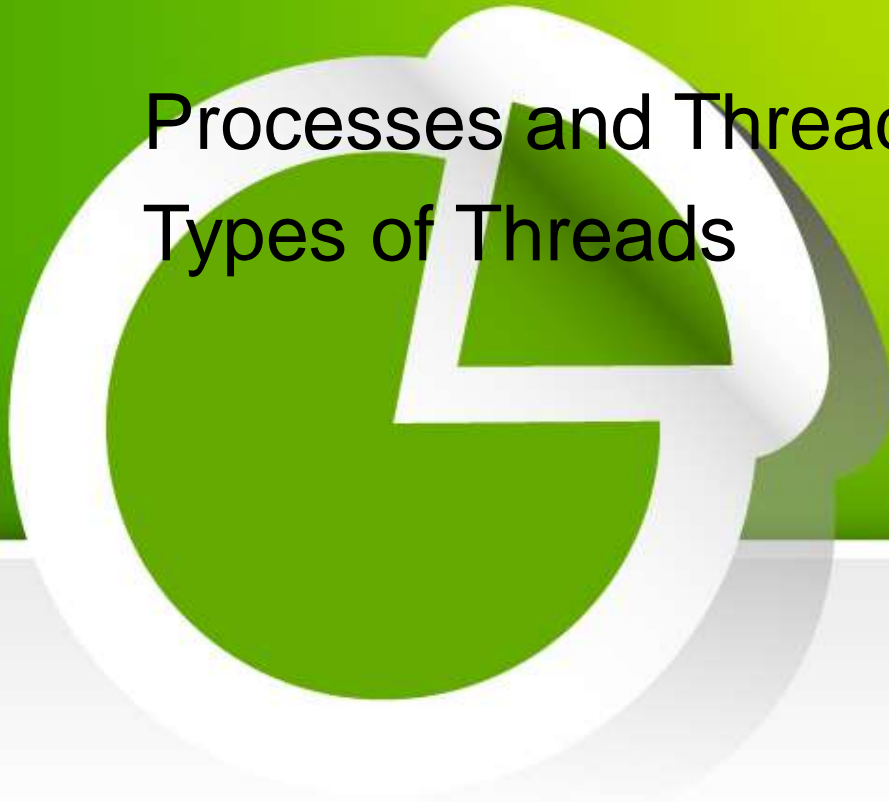
# UNIT II

Process States

Process Description and Process Control.

Processes and Threads

Types of Threads





- – 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.



- From earlier chapters we saw:
  - Computer platforms 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

# Concepts cont...



- OS provides an interface for applications to use
- OS provides a representation of resources that can be requested and accessed by application

# The OS Manages Execution of Applications



- Resources are made available to multiple applications
- The processor is switched among multiple application
- The processor and I/O devices can be used efficiently

# 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 (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

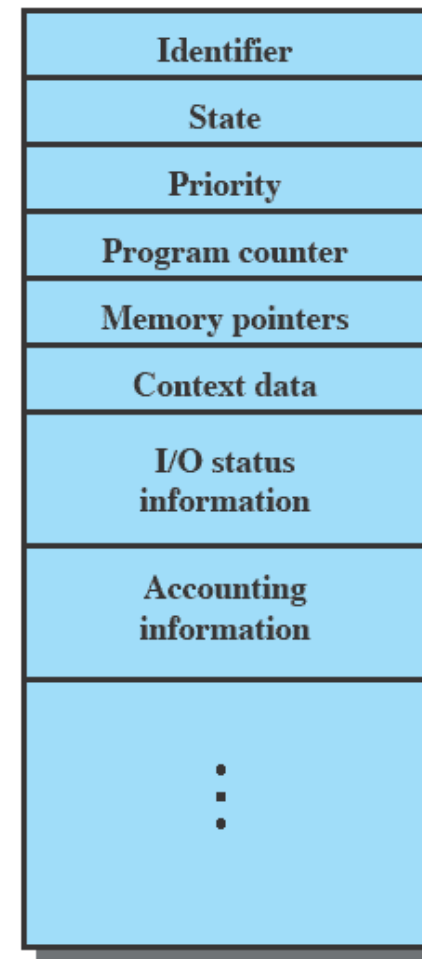


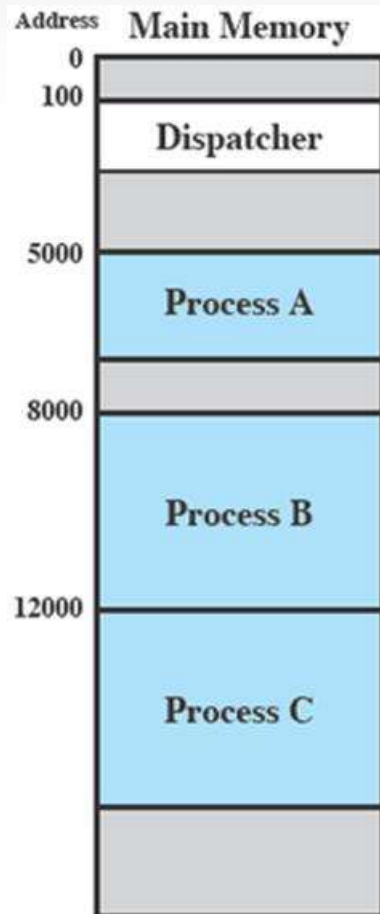
Figure 3.1 Simplified Process Control Block

# 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***
- ***Dispatcher*** is a small program which switches the processor from one process to another

# Process Execution

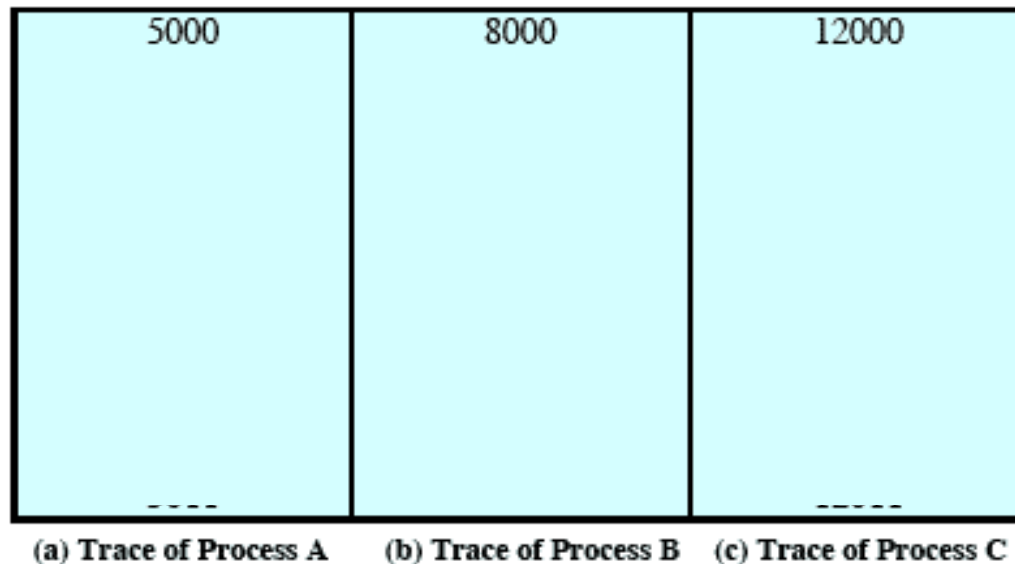


- Consider three processes being executed
- All are in memory (plus the dispatcher)
- Lets ignore virtual memory for this.

# Trace from the *processes* point of view:



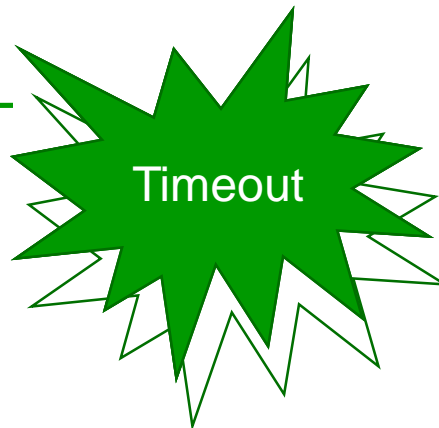
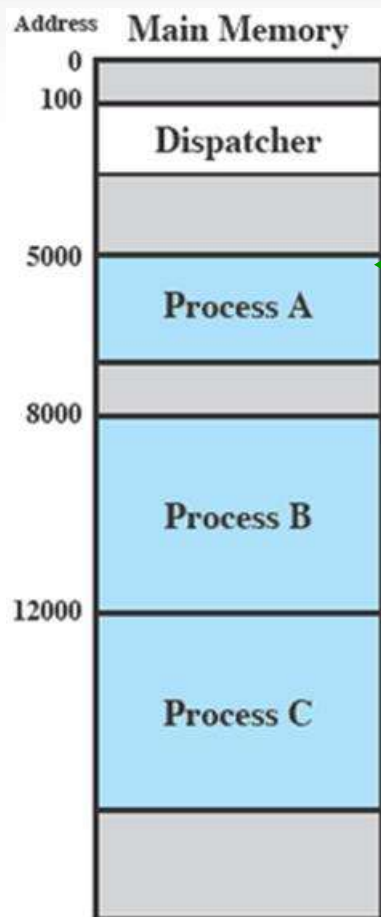
- Each process runs to completion



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

# Trace from Processors point of view



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

# Roadmap

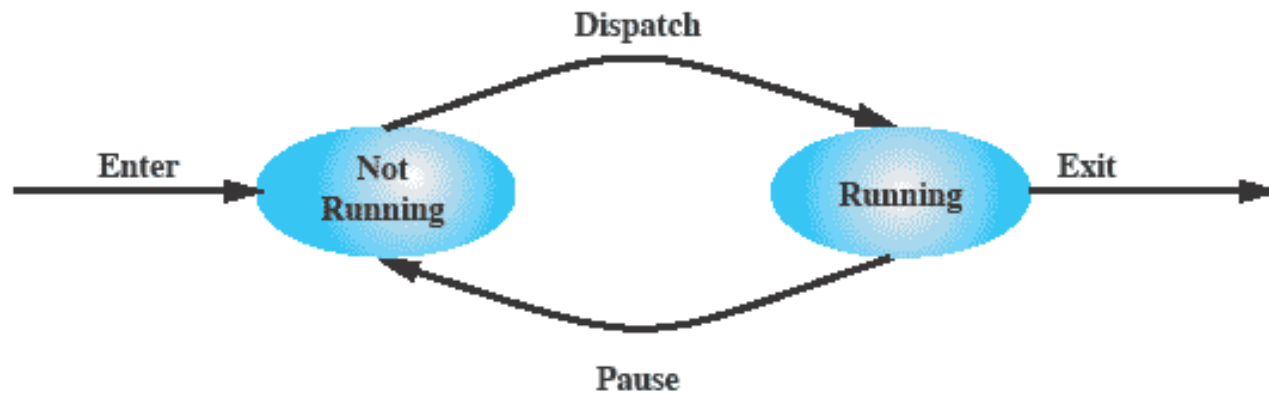


- How are processes represented and controlled by the OS.
- **Process states** which characterize the behaviour of processes.
- **Data structures** used to manage processes.
- Ways in which the OS uses these data structures to control process execution.
- Discuss process management in UNIX SVR4.

# Two-State Process Model



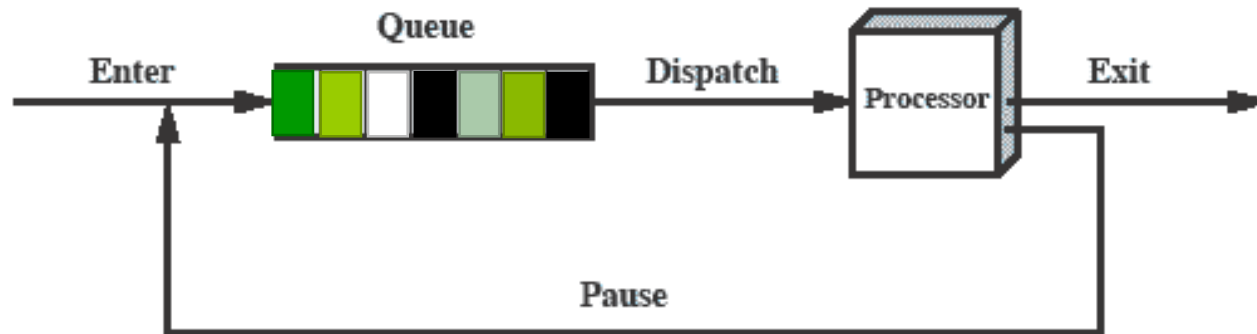
- Process may be in one of two states
  - Running
  - Not-running



(a) State transition diagram



# 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

# Process Birth and Death



<b>Creation</b>	<b>Termination</b>
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

See tables 3.1 and 3.2 for more

# 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

# Five-State Process Model

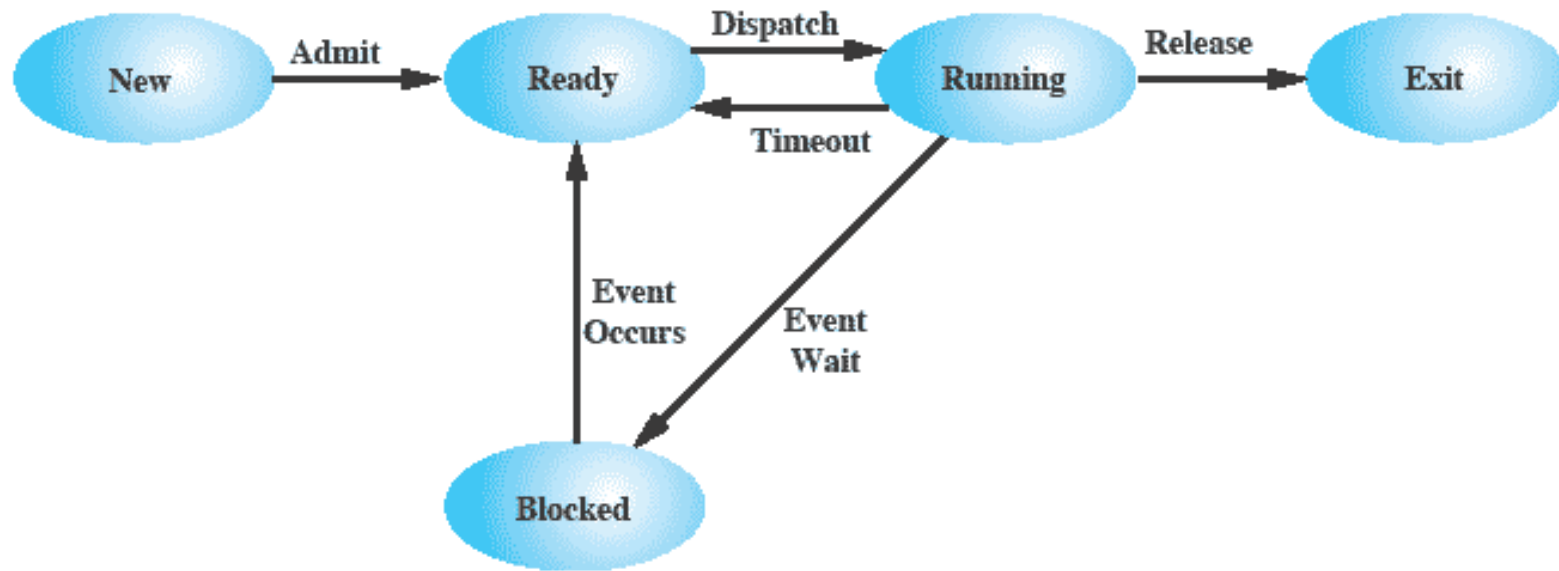
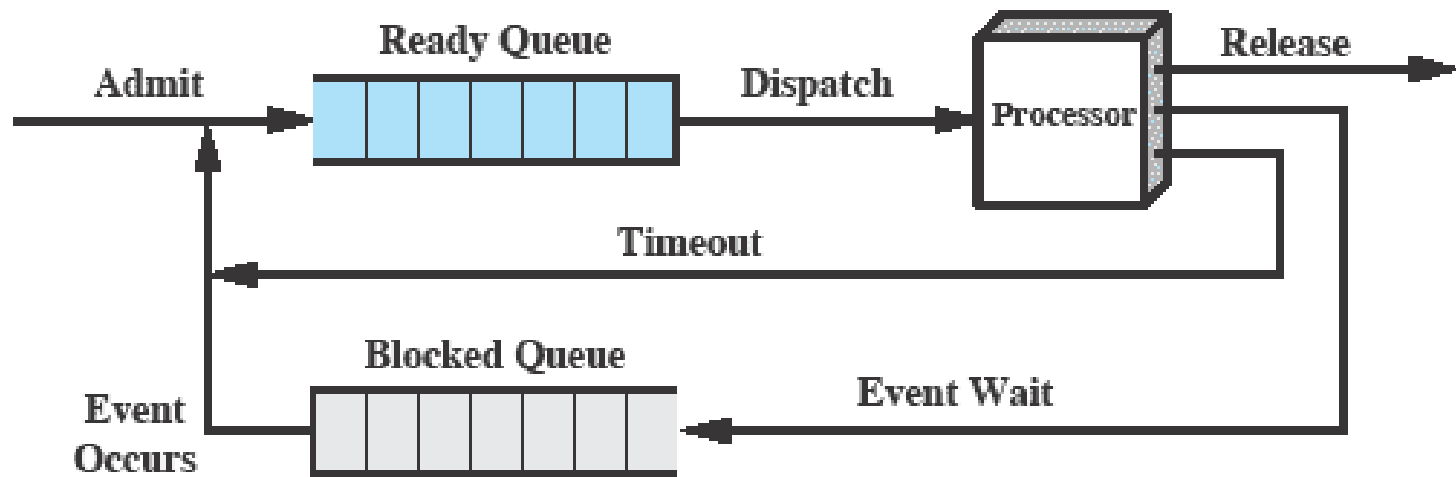


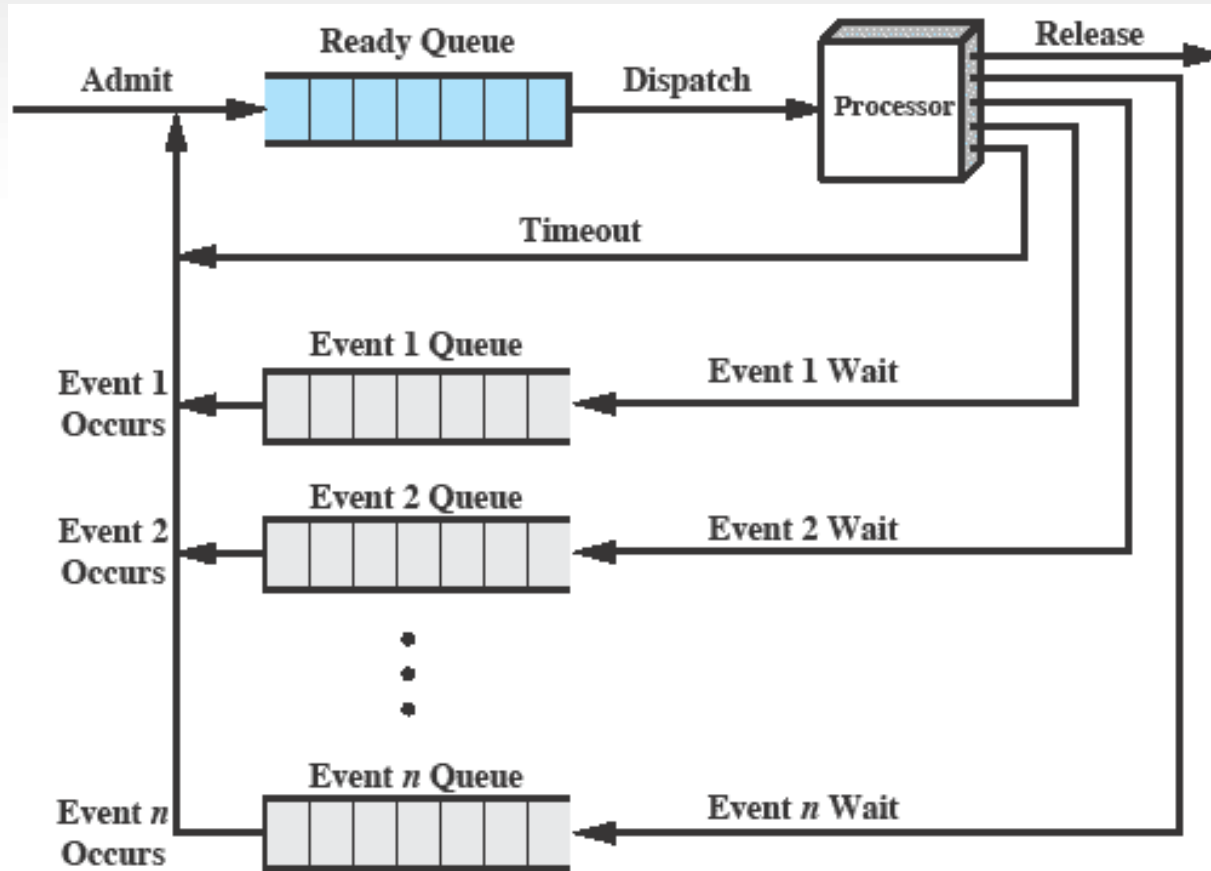
Figure 3.6 Five-State Process Model

# Using Two Queues



(a) Single blocked queue

# Multiple Blocked Queues



(b) Multiple blocked queues

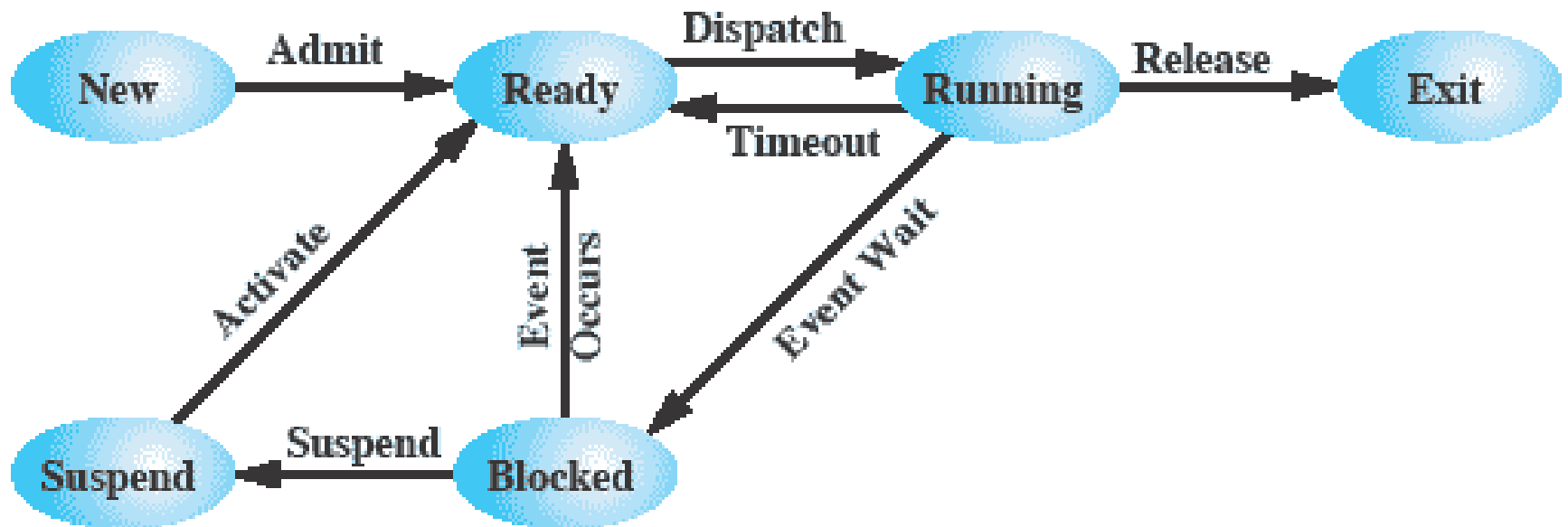
# Suspended Processes



- 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
- Blocked state becomes ***suspend*** state when swapped to disk
- Two new states
  - Blocked/Suspend
  - Ready/Suspend

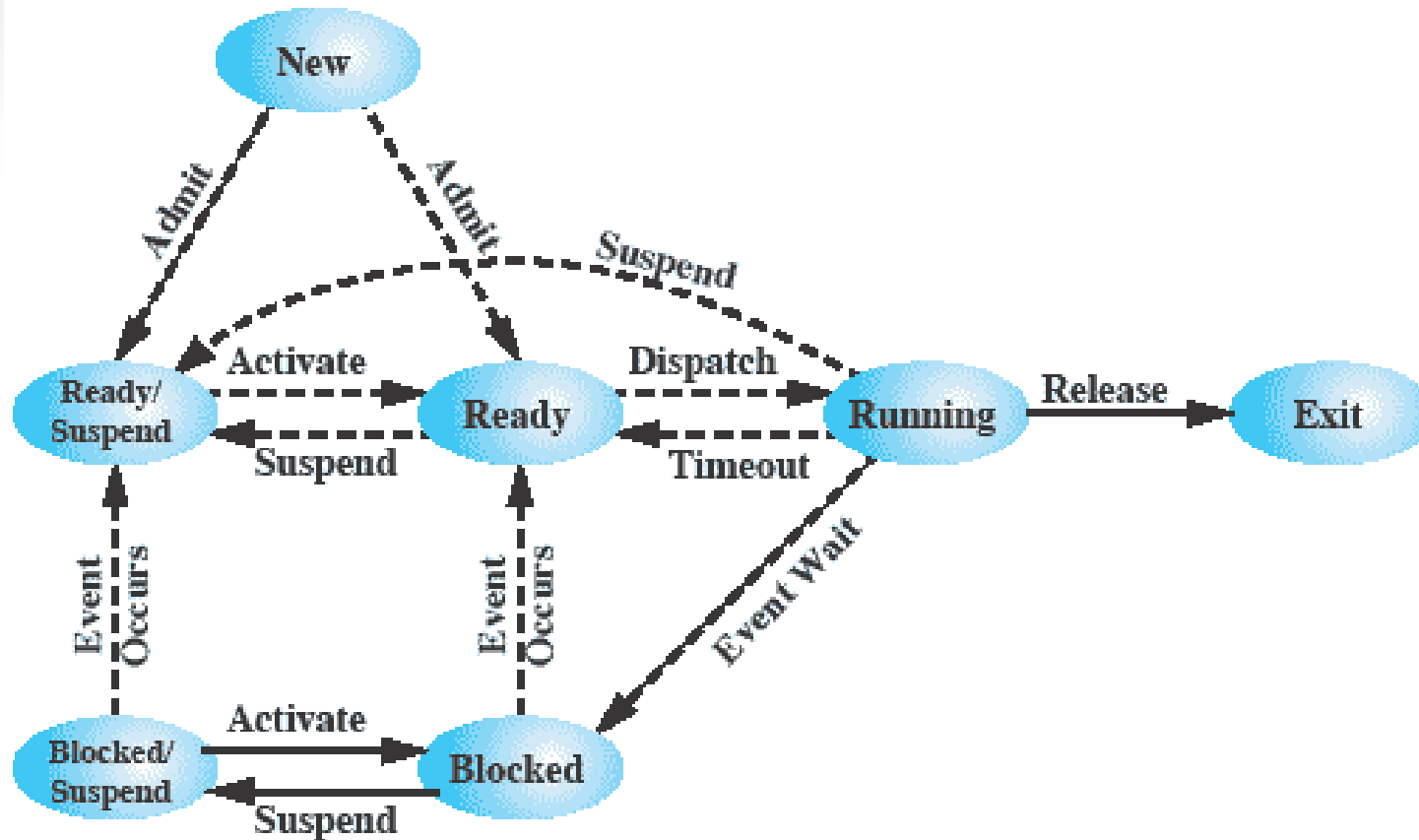


# One Suspend State



(a) With One Suspend State

# Two Suspend States



(b) With Two Suspend States

# 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.

# 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

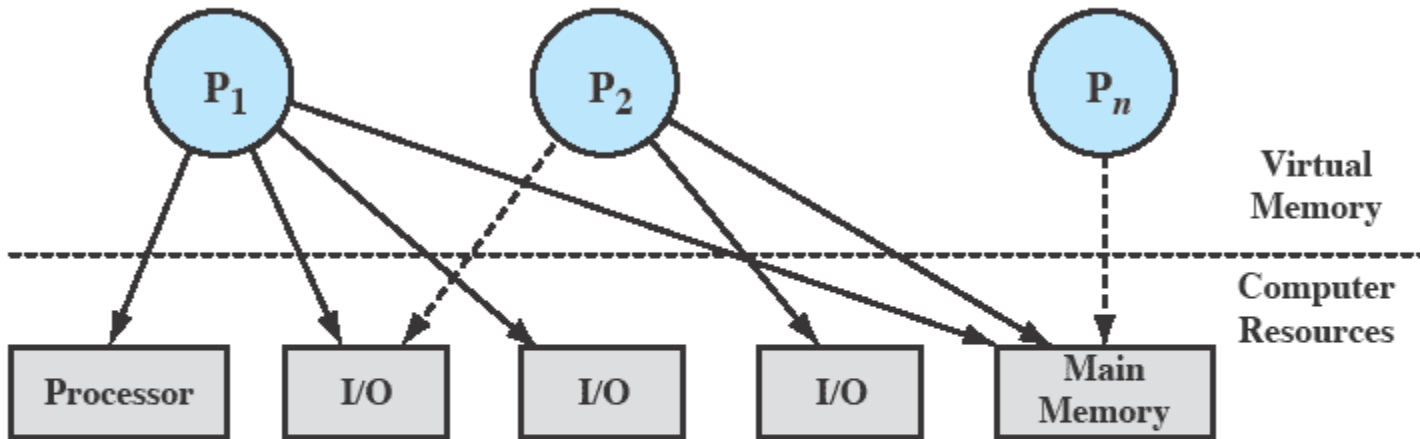


Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)

# 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

# OS Control Tables

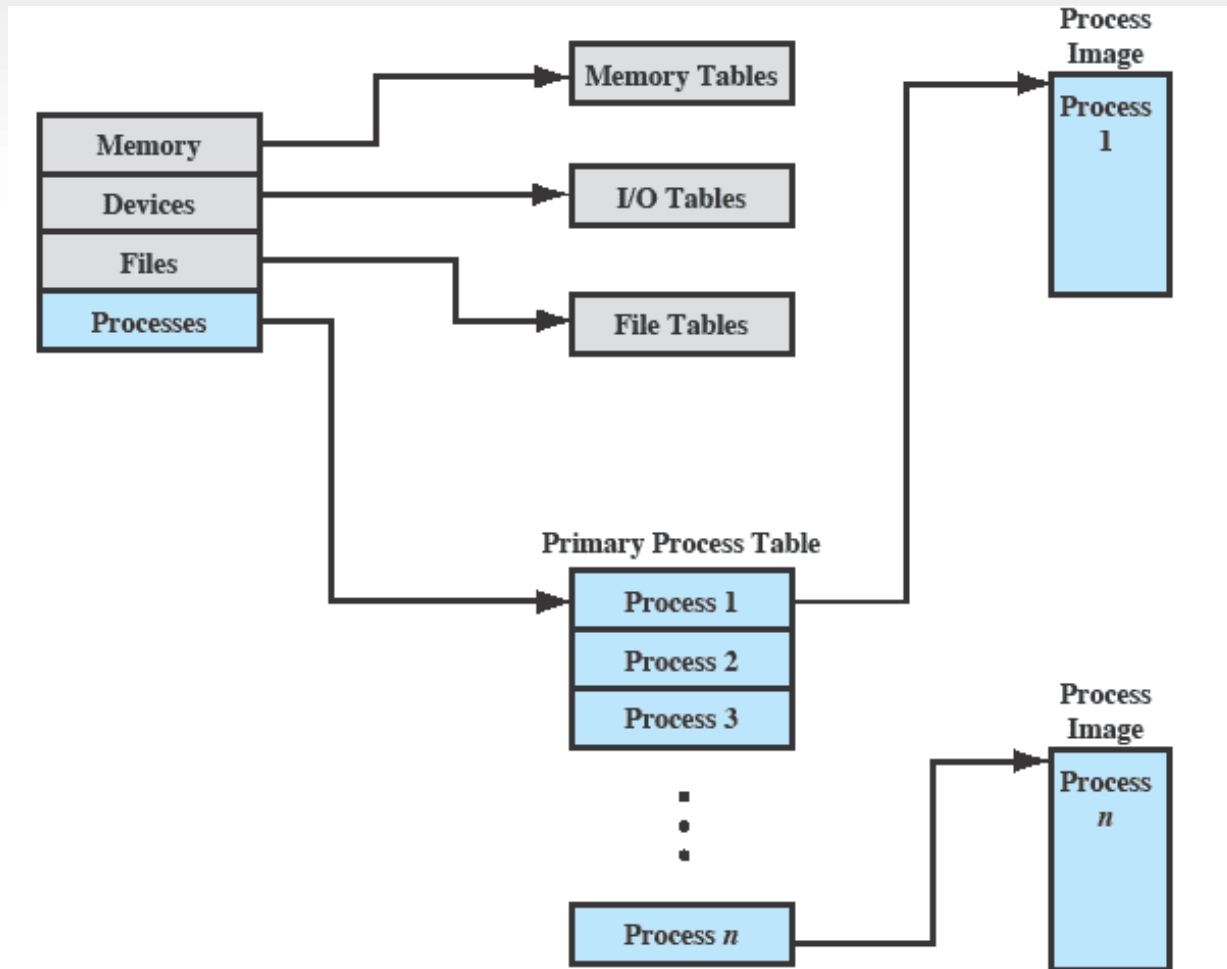


Figure 3.11 General Structure of Operating System Control Tables

# 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





- 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.
  - See table 3.5 for scope of information

# 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
- 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

**Table 3.8 Mechanisms for Interrupting the Execution of a Process**

# 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

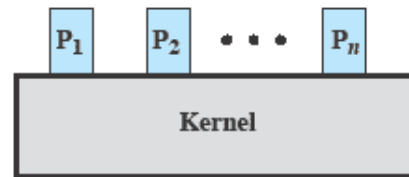
# Is the OS a Process?



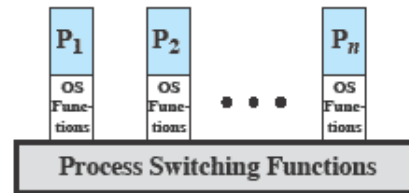
- If the OS is just a collection of programs and if it is executed by the processor just like any other program, is the OS a process?
- If so, how is it controlled?
  - Who (what) controls it?



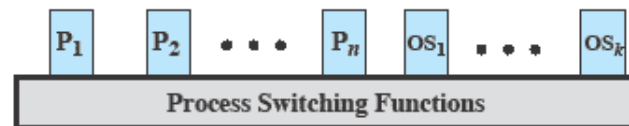
# Execution of the Operating System



(a) Separate kernel



(b) OS functions execute within user processes



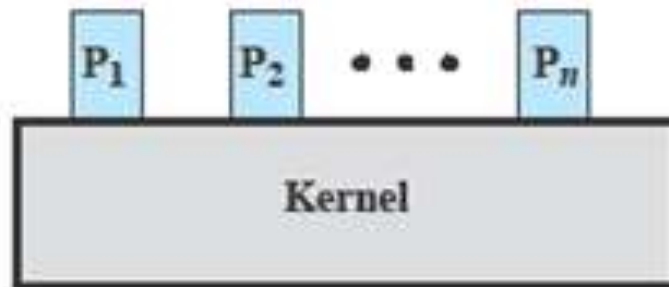
(c) OS functions execute as separate processes

**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

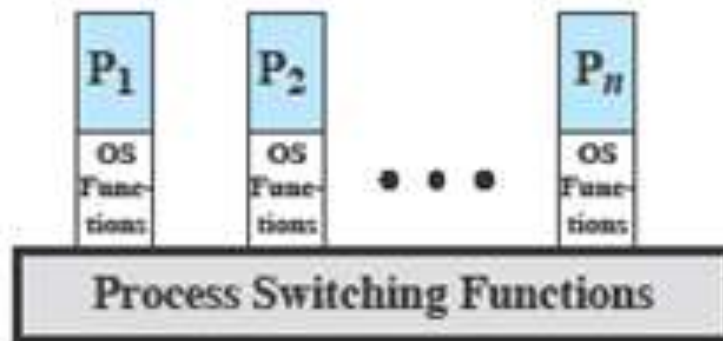


(a) Separate kernel

# 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

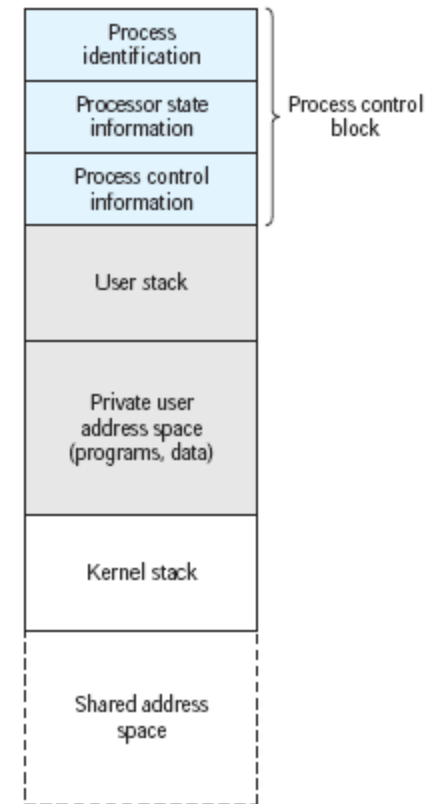
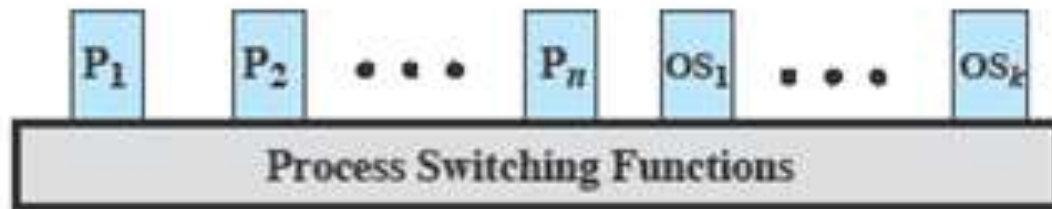


Figure 3.16 Process Image: Operating System Executes within User Space

# Process-based Operating System



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



(c) OS functions execute as separate processes

# System access threats



- Intruders
  - Masquerader (outsider)
  - Misfeasor (insider)
  - Clandestine user (outside or insider)
- Malicious software (malware)

# Roadmap

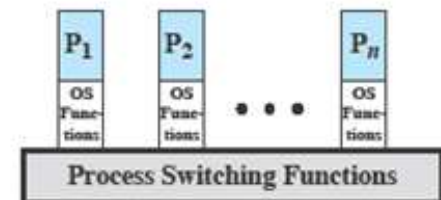


- How are processes represented and controlled by the OS.
- **Process states** which characterize the behaviour of processes.
- **Data structures** used to manage processes.
- Ways in which the OS uses these data structures to control process execution.
- Discuss process management in UNIX SVR4.





- Uses the model of fig3.15b where most of the OS executes in the user process
- 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

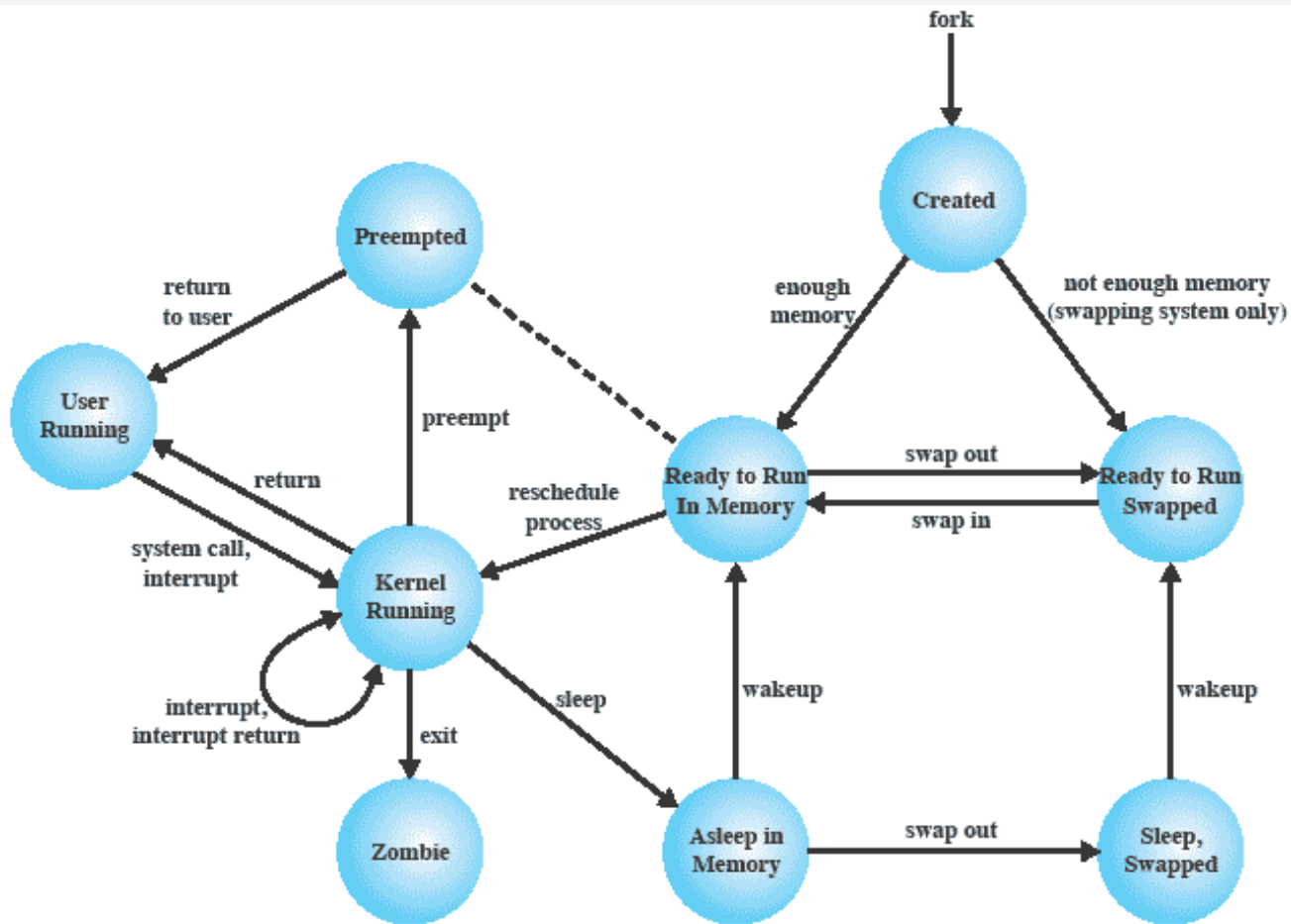


Figure 3.17 UNIX Process State Transition Diagram



# 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.

# 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.

# THREADS, SMP



- Threads: Resource ownership and execution
- Symmetric multiprocessing (SMP).
- Microkernel
- Case Studies of threads and SMP:
  - Windows
  - Solaris
  - Linux

# Processes and Threads



- Processes have two characteristics:
  - **Resource ownership** - process includes a virtual address space to hold the process image
  - **Scheduling/execution** - follows an execution path that may be interleaved with other processes
- These two characteristics are treated independently by the operating system

# Processes and Threads



- The unit of dispatching is referred to as a ***thread*** or lightweight process
- The unit of resource ownership is referred to as a process or ***task***



# Multithreading



- The ability of an OS to support multiple, concurrent paths of execution within a single process.

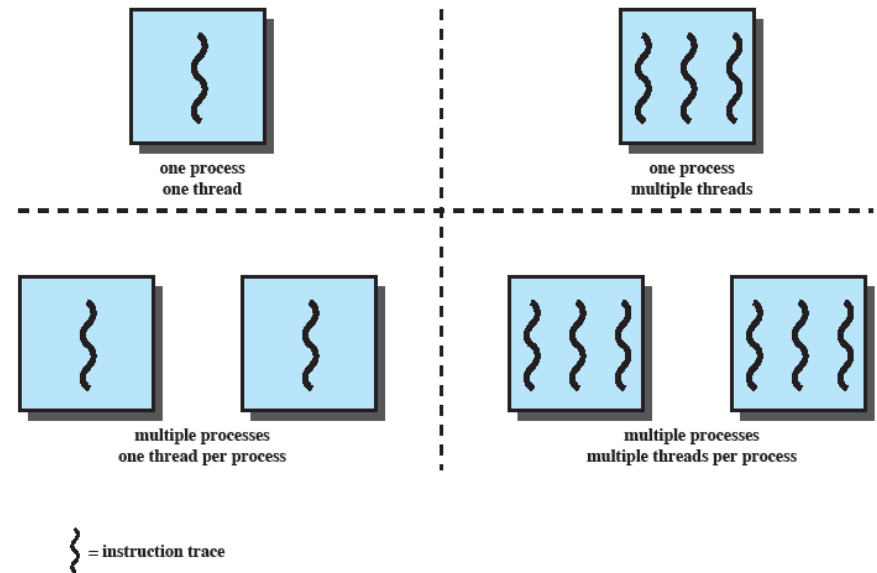


Figure 4.1 Threads and Processes [ANDE97]

# Single Thread Approaches



- MS-DOS supports a single user process and a single thread.
- Some UNIX, support multiple user processes but only support one thread per process

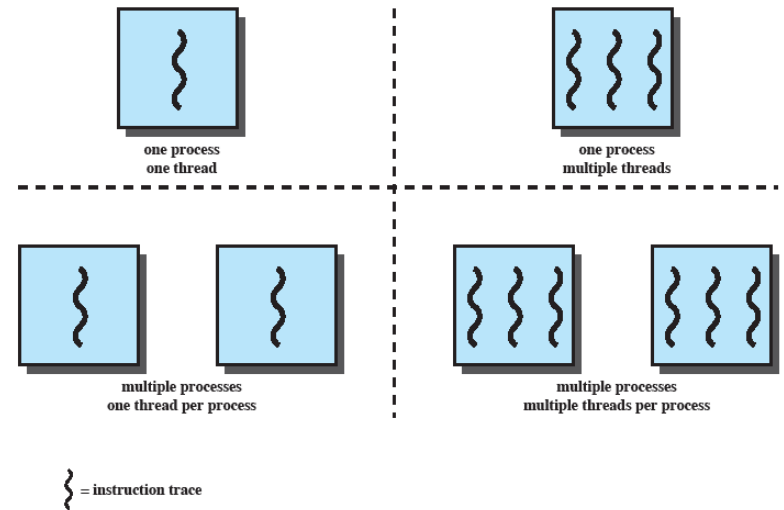


Figure 4.1 Threads and Processes [ANDE97]

# Multithreading



- Java run-time environment is a single process with multiple threads
- Multiple processes **and** threads are found in Windows, Solaris, and many modern versions of UNIX

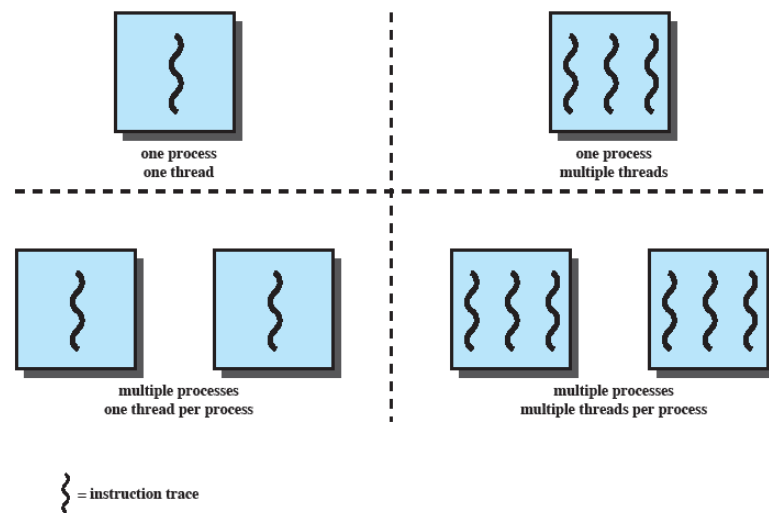


Figure 4.1 Threads and Processes [ANDE97]

# Processes



- A virtual address space which holds the process image
- Protected access to
  - Processors,
  - Other processes,
  - Files,
  - I/O resources

# One or More Threads in Process



- Each thread has
  - An execution state (running, ready, etc.)
  - Saved thread context when not running
  - An execution stack
  - Some per-thread static storage for local variables
  - Access to the memory and resources of its process (all threads of a process share this)



- *One way to view a thread is as an independent program counter operating within a process.*

# Threads vs. processes

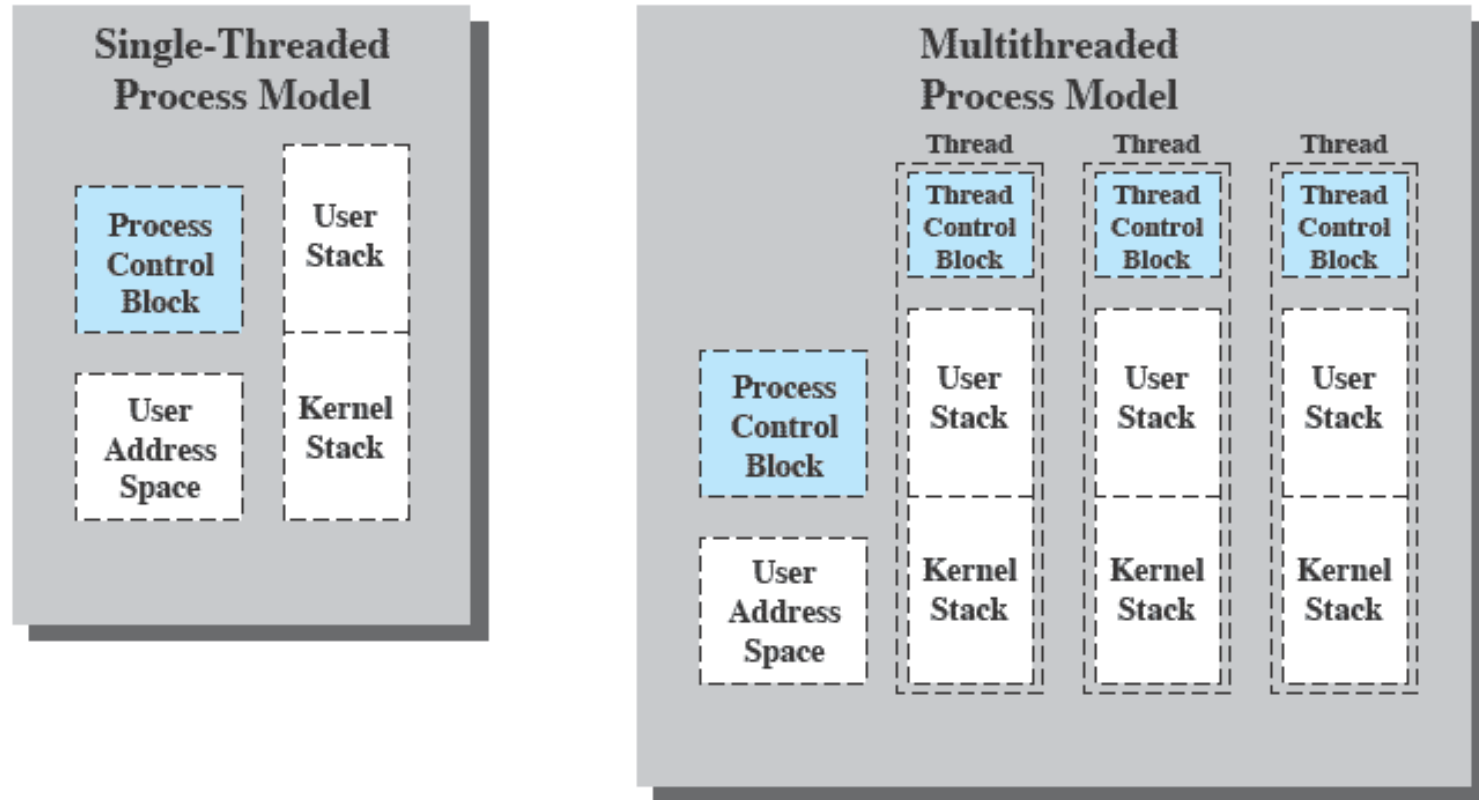


Figure 4.2 Single Threaded and Multithreaded Process Models

# Benefits of Threads



- Takes less time to create a new thread than a process
- Less time to terminate a thread than a process
- Switching between two threads takes less time than switching processes
- Threads can communicate with each other
  - without invoking the kernel



# Thread use in a Single-User System



- Foreground and background work
- Asynchronous processing
- Speed of execution
- Modular program structure

# Threads



- Several actions that affect all of the threads in a process
  - The OS must manage these at the process level.
- Examples:
  - Suspending a process involves suspending all threads of the process
  - Termination of a process, terminates all threads within the process

# Activities similar to Processes



- Threads have execution states and may synchronize with one another.
  - Similar to processes
- We look at these two aspects of thread functionality in turn.
  - States
  - Synchronisation

# Thread Execution States



- States associated with a change in thread state
  - Spawn (another thread)
  - Block
    - Issue: will blocking a thread block other, or *all*, threads
  - Unblock
  - Finish (thread)
    - Deallocate register context and stacks

# Categories of Thread Implementation

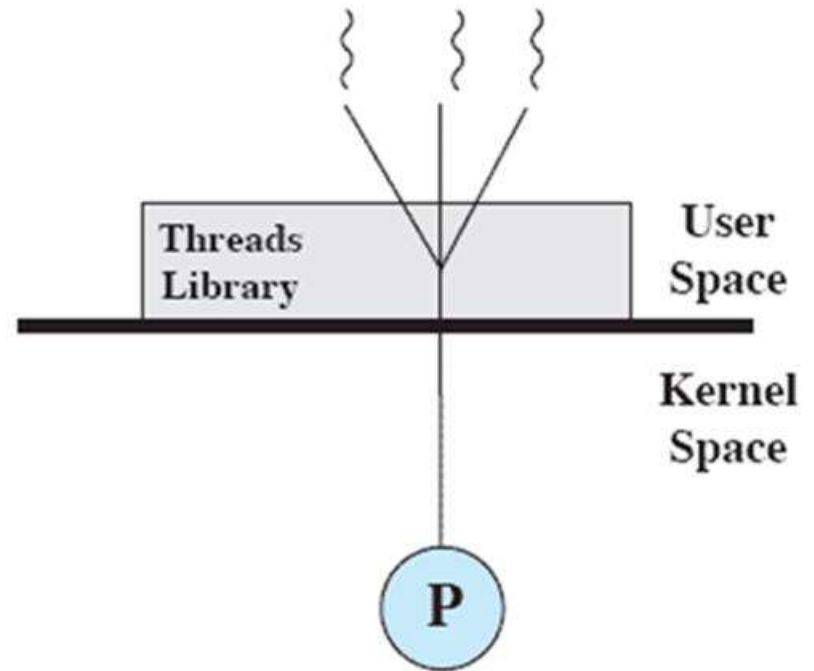


- User Level Thread (ULT)
- Kernel level Thread (KLT) also called:
  - kernel-supported threads
  - lightweight processes.

# User-Level Threads

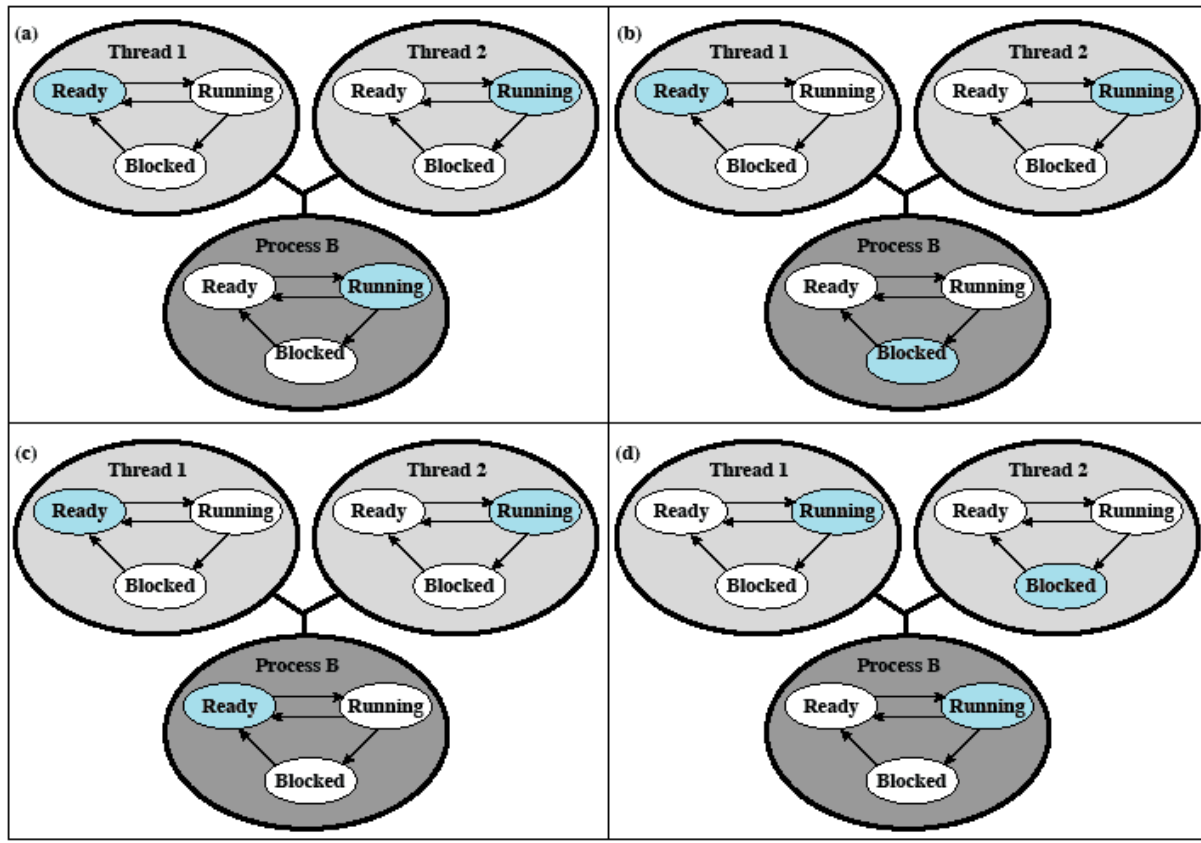


- All thread management is done by the application
- The kernel is not aware of the existence of threads



(a) Pure user-level

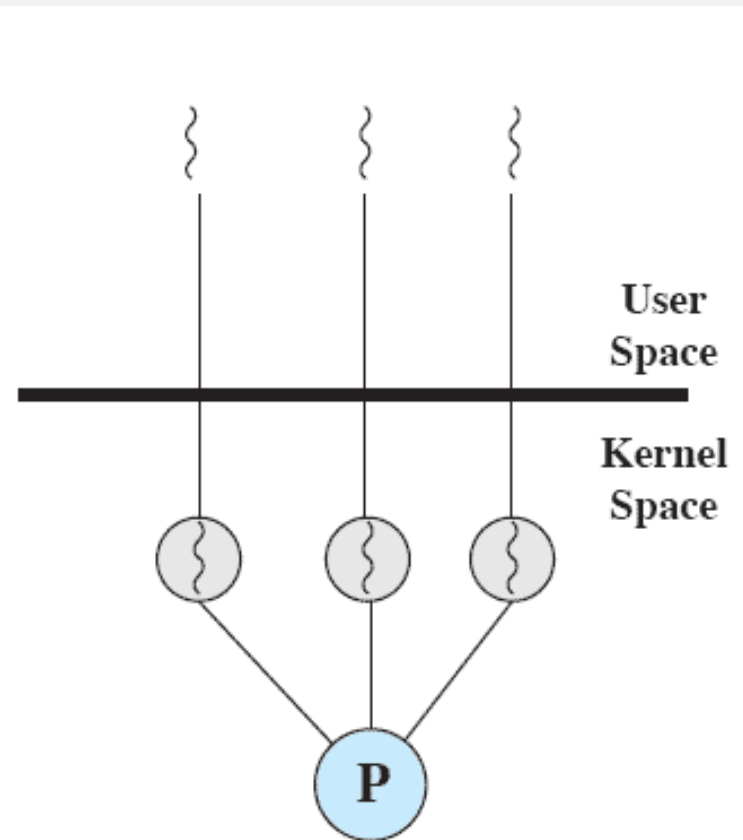
# Relationships between ULT Thread and Process States



Colored state  
is current state

Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States

# Kernel-Level Threads



(b) Pure kernel-level

- Kernel maintains context information for the process and the threads
  - No thread management done by application
- Scheduling is done on a thread basis
- Windows is an example of this approach



# Advantages of KLT



- The kernel can simultaneously schedule multiple threads from the same process on multiple processors.
- If one thread in a process is blocked, the kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.

# Disadvantage of KLT

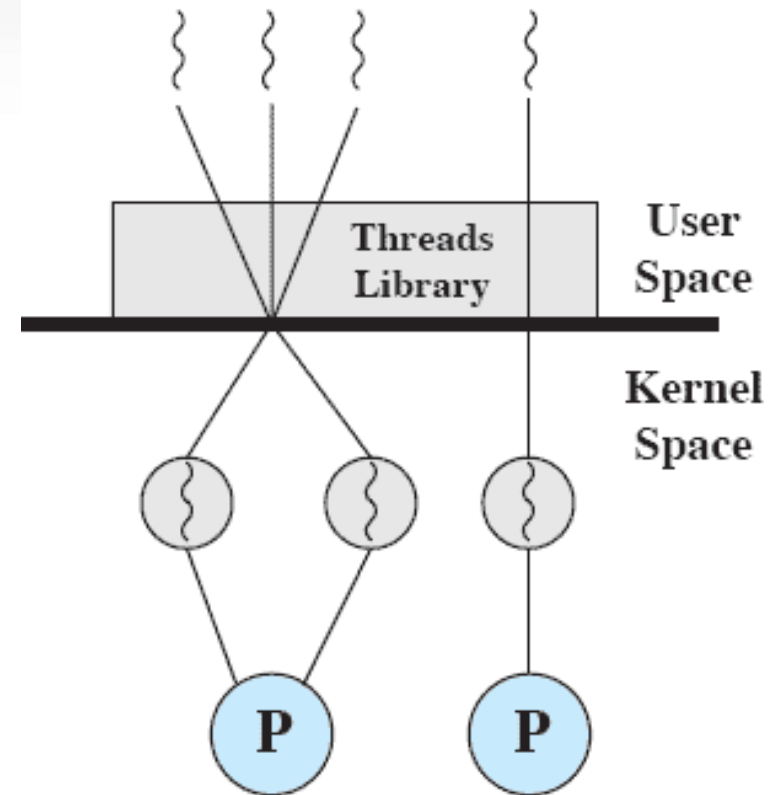


- The transfer of control from one thread to another within the same process requires a mode switch to the kernel

# Combined Approaches



- Thread creation done in the user space
- Bulk of scheduling and synchronization of threads by the application
- Example is Solaris



(c) Combined

# Relationship Between Thread and Processes



**Table 4.2 Relationship Between Threads and Processes**

Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux, OS/2, OS/390, MACH
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:N	Combines attributes of M:1 and 1:M cases.	TRIX