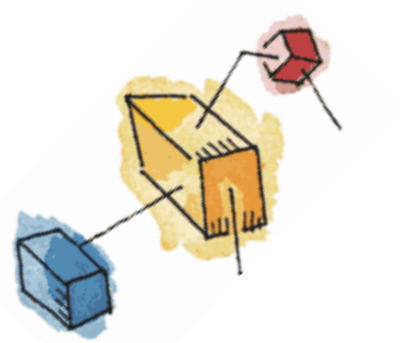
Operating Systems:
Internals and Design Principles, 6/E
William Stallings

Operating System Overview

Patricia Roy
Manatee Community College, Venice, FL



Operating System

- A program that controls the execution of application programs
- An interface between applications and hardware.
- Responsible for managing hardware resources.
- **Kernel** is a portion of operating system that is in main memory. It contains most frequently used functions.

Layers and Views

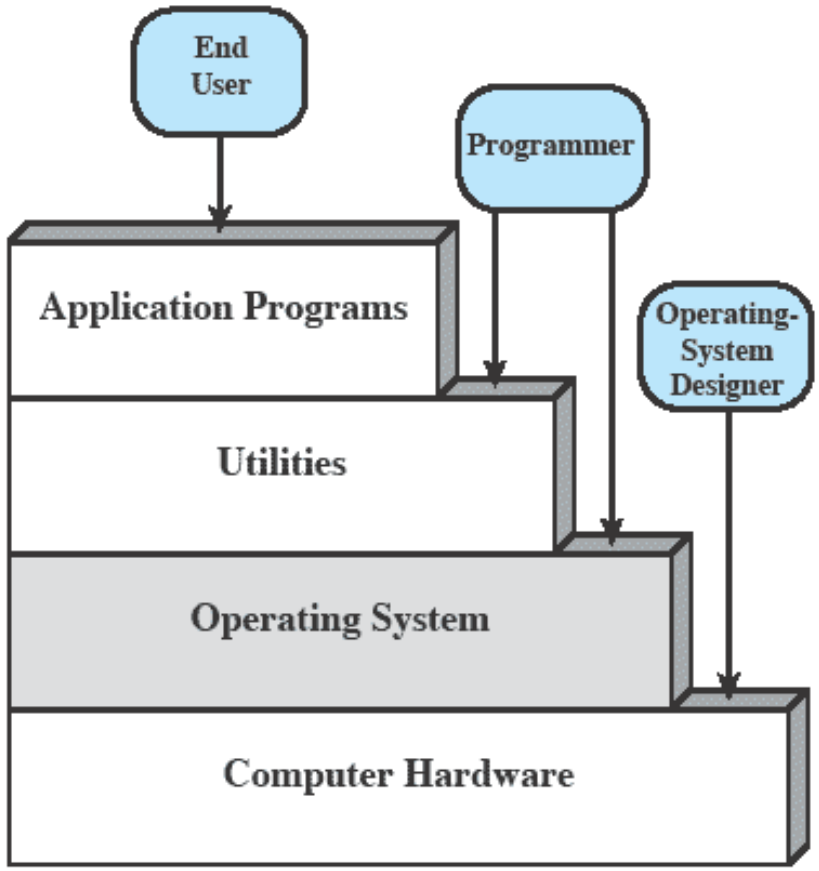
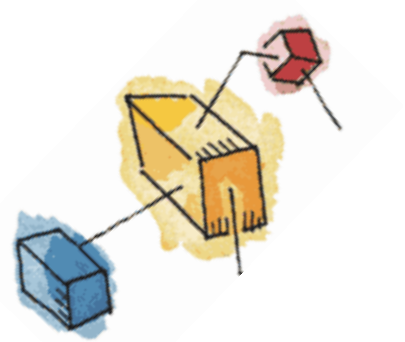


Figure 2.1 Layers and Views of a Computer System



OS Services

- Program development
- Program execution
- Access I/O devices
- Controlled access to files
- System access

Categories of Computer Systems

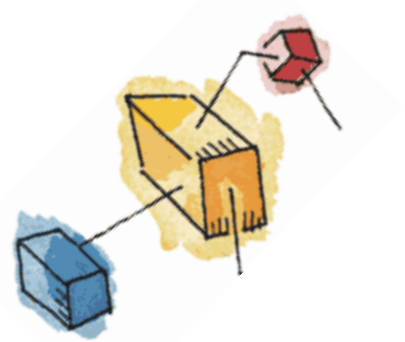
- Single Instruction Single Data (SISD) stream
  - Single processor executes a single instruction stream to operate on data stored in a single memory
- Single Instruction Multiple Data (SIMD) stream
  - Each instruction is executed on a different set of data by the different processors

Categories of Computer Systems

- Multiple Instruction Single Data (MISD) stream (Never implemented)
  - A sequence of data is transmitted to a set of processors, each of execute a different instruction sequence
- Multiple Instruction Multiple Data (MIMD)
  - A set of processors simultaneously execute different instruction sequences on different data sets

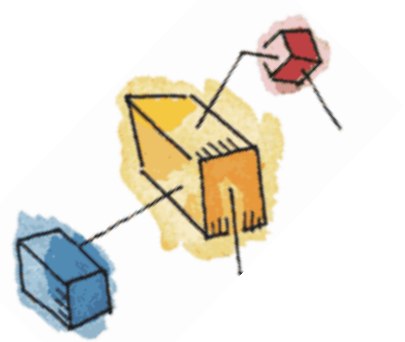
Evolution of Operating Systems

- Serial Processing
- Simple Batch Systems
- Multiprogrammed Batch Systems
- Time Sharing Systems
- Parallel Systems
- Distributed Systems



Serial Processing

- Programmer interacted directly with computer hardware with no operating system.
- Machines were run with a console consisting of display lights, toggle switches, some form of input device and a printer.
- Programs in machine code are loaded with the input device like card reader. If an error occur the program was halted and the error condition was indicated by lights. Programmers examine the registers and main memory to determine error. If the program is success, then output will appear on the printer.



Serial Processing

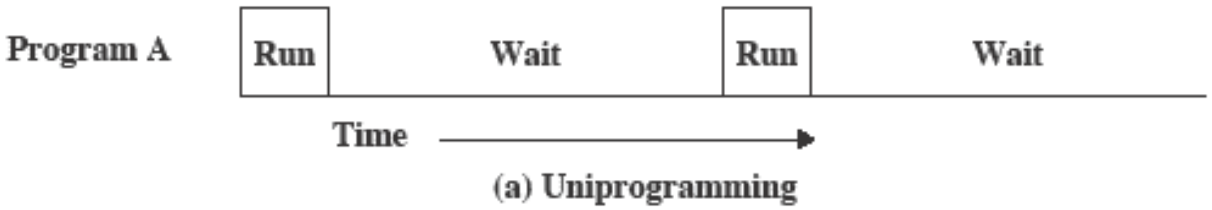
- Main problem here is the setup time. That is single program needs to load source program into memory, saving the compiled (object) program and then loading and linking together.

Simple Batch Systems

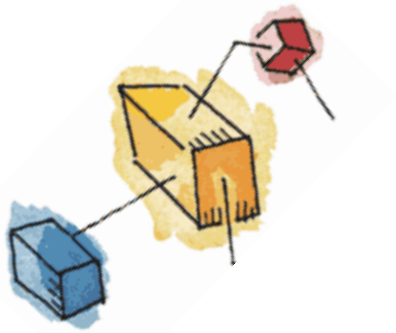
- To speed up processing, jobs with similar needs are batched together and run as a group.
- The programmers will leave their programs with the operator (monitor). The operator will sort programs into batches with similar requirements.
- The problems with Batch Systems are:
  - Lack of interaction between the user and job.
  - CPU is often idle, because the speeds of the mechanical I/O devices are slower than CPU.

Uniprogramming

- Processor must wait for I/O instruction to complete before proceeding

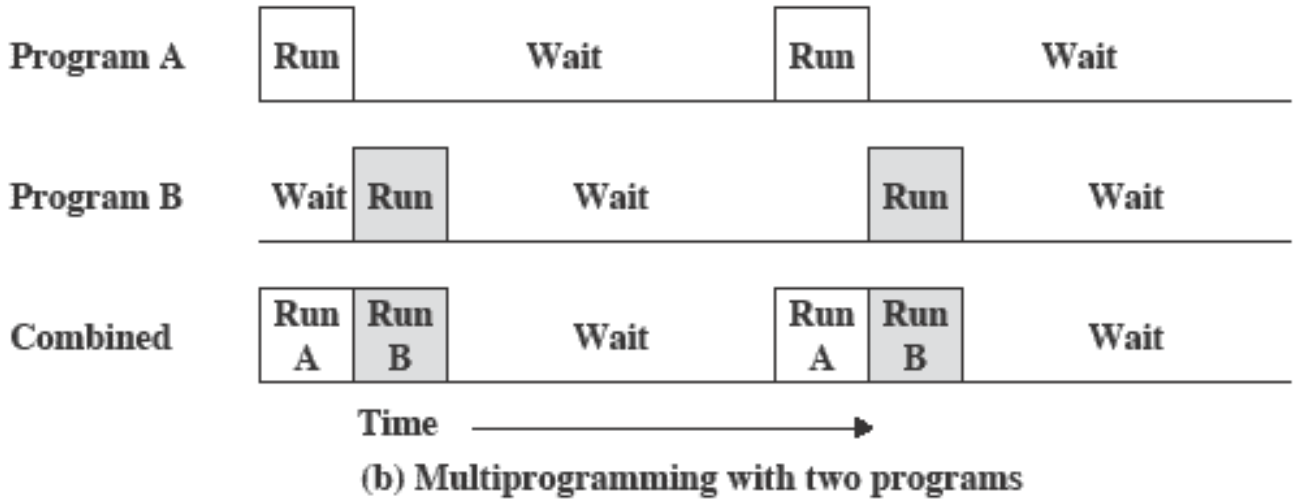


(a) Uniprogramming



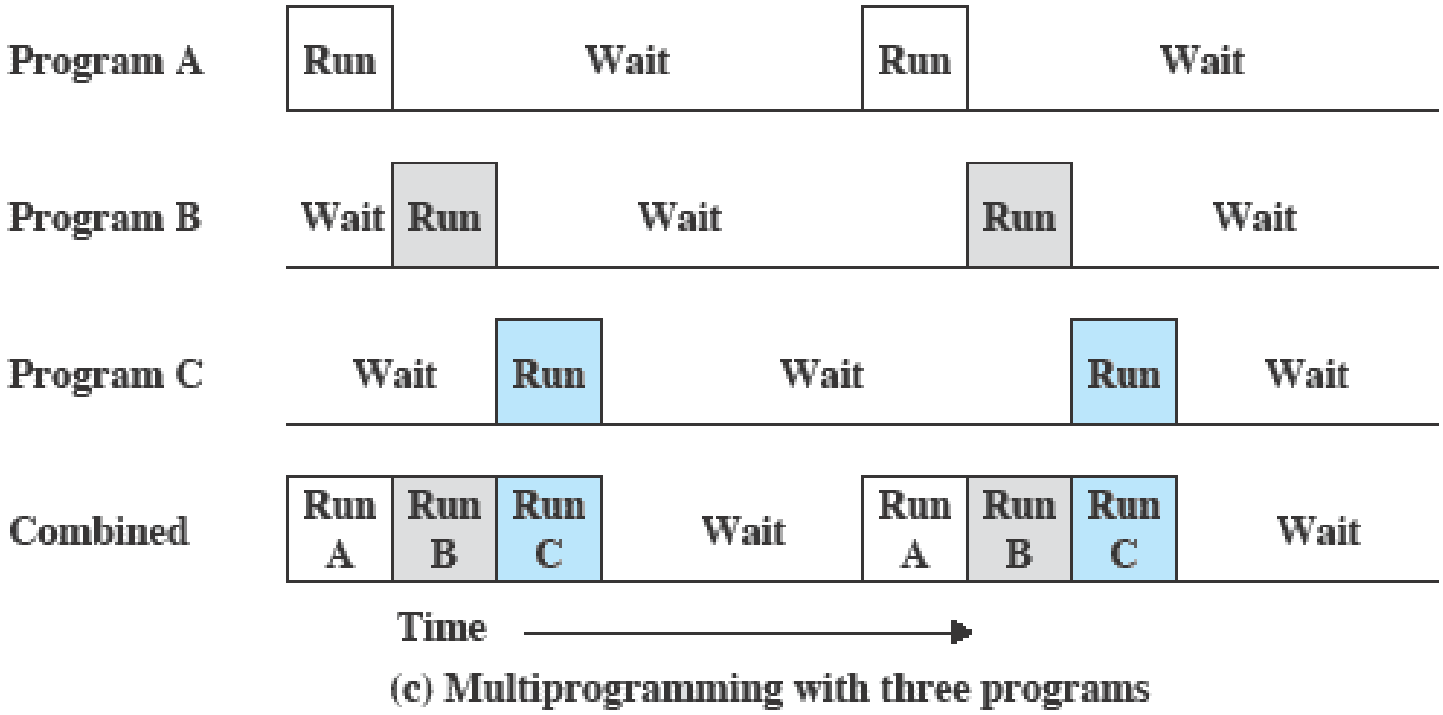
Multiprogramming

- When one job needs to wait for I/O, the processor can switch to the other job



(b) Multiprogramming with two programs

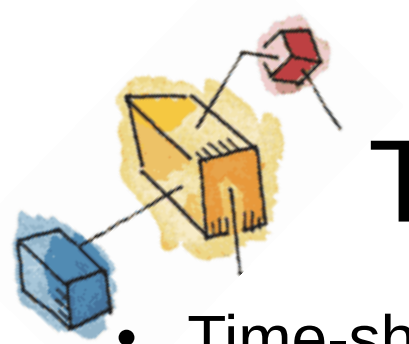
Multiprogramming



(c) Multiprogramming with three programs

Multiprogrammed Batch Systems

- Jobs must be run sequentially, on a first-come, first-served basis.
- However when several jobs are on a direct-access device like disk, job scheduling is possible. The main aspect of job scheduling is multiprogramming.
- Single user cannot keep the CPU or I/O devices busy at all times. Thus multiprogramming increases CPU utilization.
- when one job needs to wait, the CPU is switched to another job, and so on. Eventually, the first job finishes waiting and gets the CPU back.



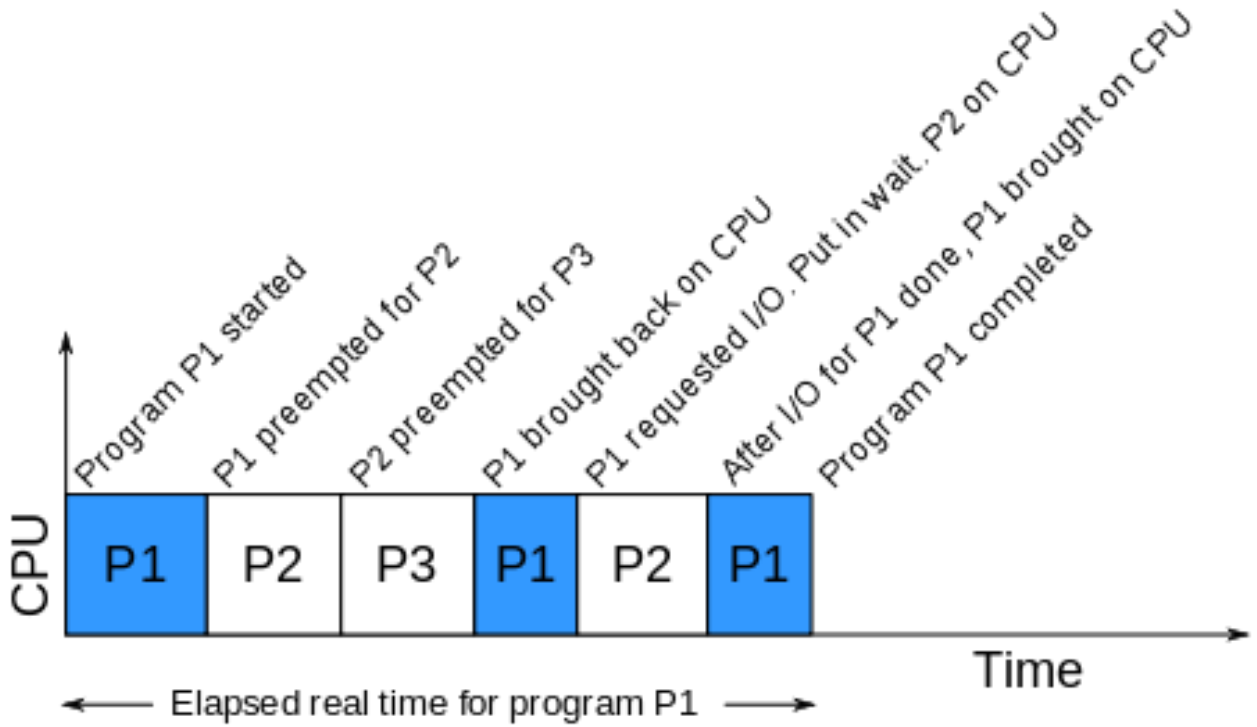
Time-Sharing Systems

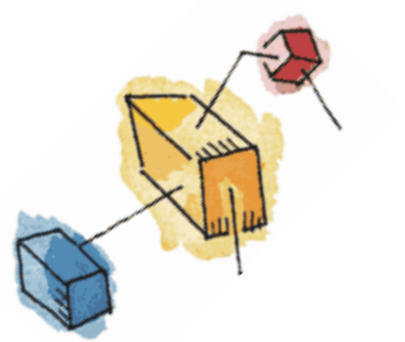
- Time-sharing or multitasking is a logical extension of multiprogramming. Multiple users simultaneously access the system through terminals.

- Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receives an immediate response.

- The main difference between Multi-programmed Batch Systems and Time-Sharing Systems is in Multi-programmed batch systems its objective is maximize processor use, whereas in Time-Sharing Systems its objective is minimize response time.

Time-Sharing Systems





Parallel Systems

- Multiprocessor systems have more than one processor.

- The advantages of parallel system are as follows
  - Throughput (Number of jobs to finish in a time period)
  - Save money by sharing peripherals and power supplies
  - Increase reliability
  - Fault-tolerant (Failure of one processor will not halt the system).

Parallel Processor Architectures

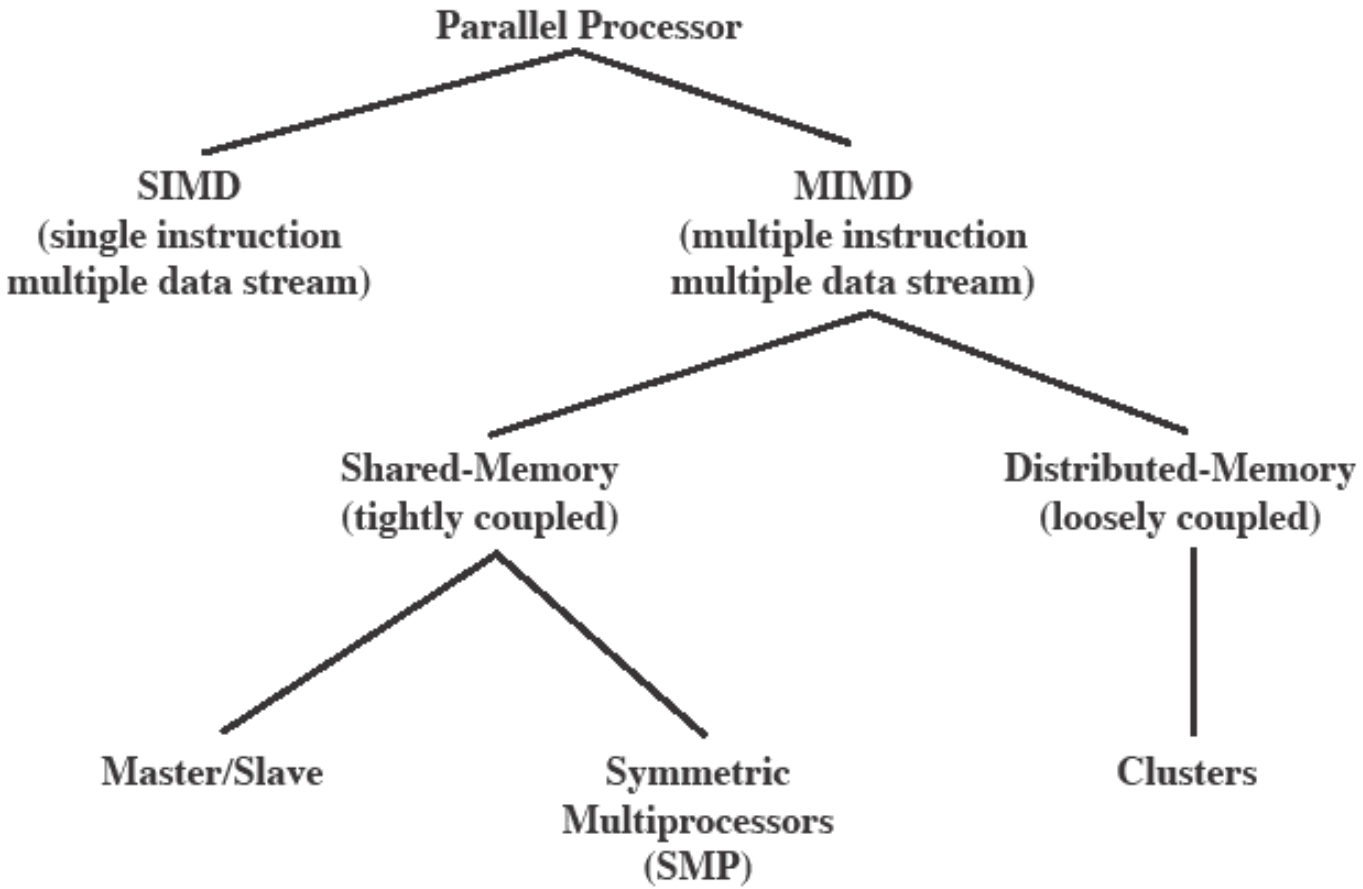


Figure 4.8 Parallel Processor Architectures

Multiprocessing Systems

- **Symmetric multiprocessing (SMP)**
  - There are multiple processors
  - These processors share same main memory and I/O facilities
  - All processors can perform the same functions

- **Asymmetric multiprocessing model**
  - Each processor is assigned a specific task. A master processor controls the system.

Multiprogramming and Multiprocessing

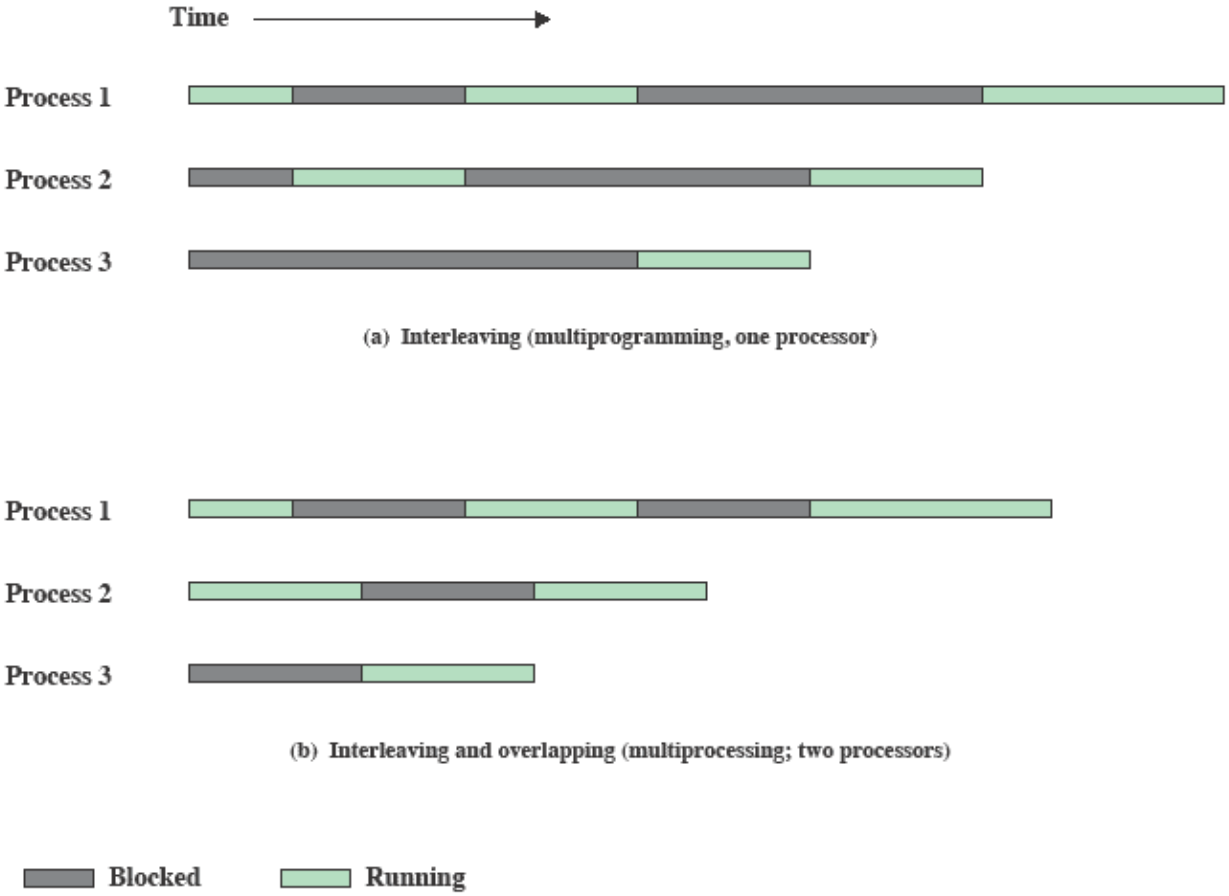
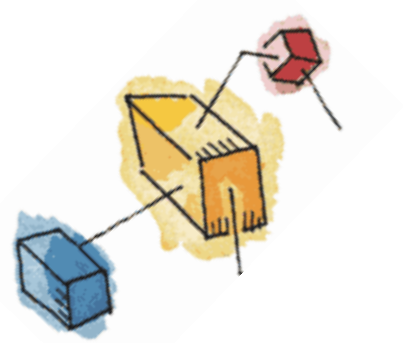


Figure 2.12 Multiprogramming and Multiprocessing

Symmetric Multiprocessing

- Kernel can execute on any processor
  - Allowing portions of the kernel to execute in parallel
- Typically each processor does self-scheduling from the pool of available process or threads



Microkernel

- keep the kernel as small as possible and other OS functions handled by separate processes
  - Address spaces management
  - Inter-process communication (IPC)
  - Basic scheduling
- this approach provides a high degree of flexibility and modularity.

Kernel Architecture

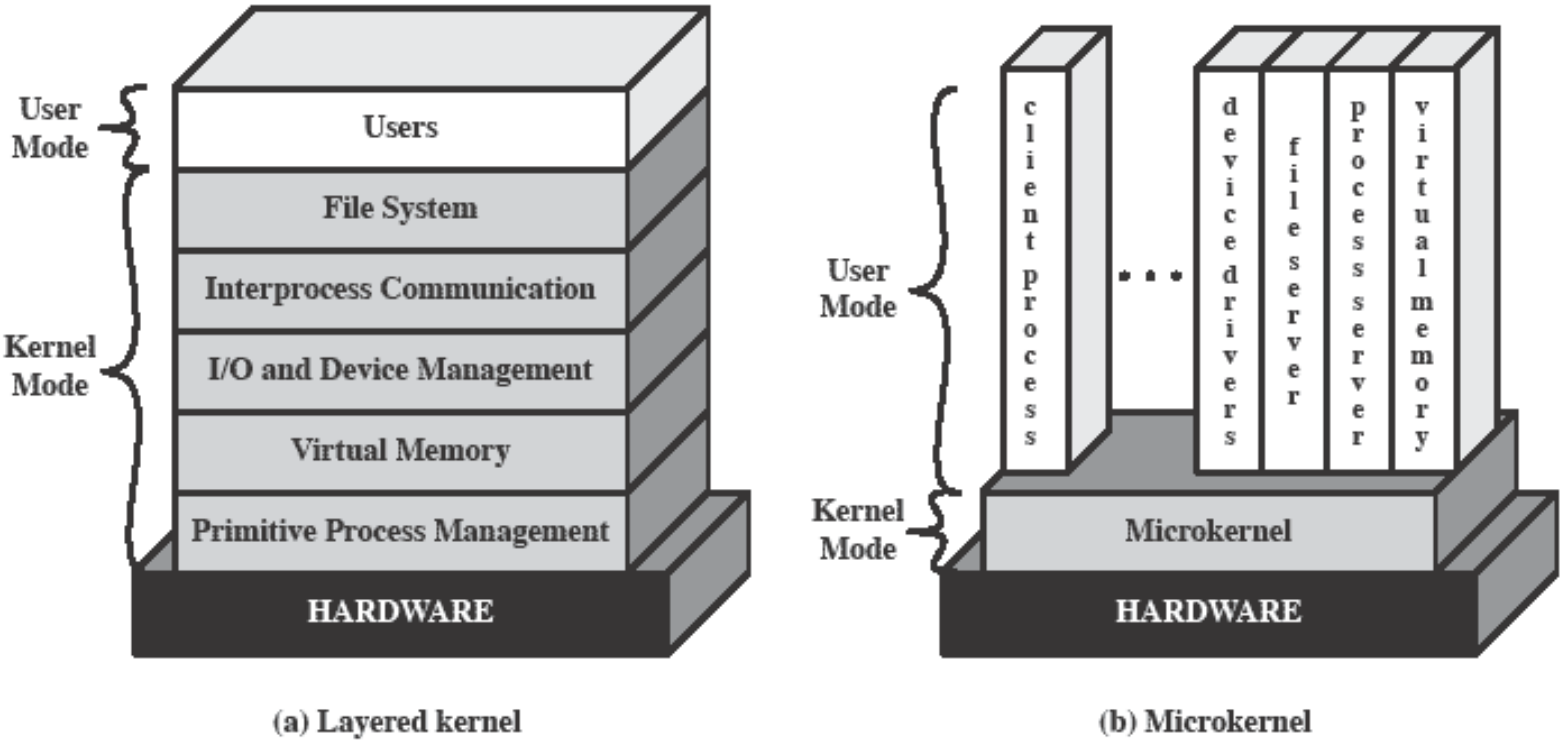


Figure 4.10 Kernel Architecture

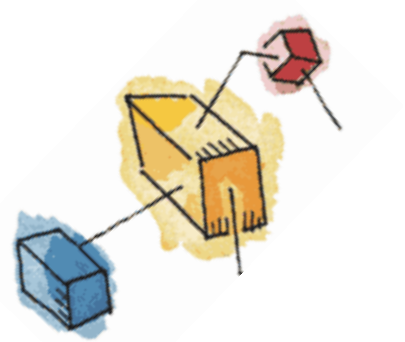
Distributed Systems

- Distributed systems distribute computation among several processors. In contrast to tightly coupled systems (i.e., parallel systems).

- The processors do not share memory or a clock. Instead, each processor has its own local memory.

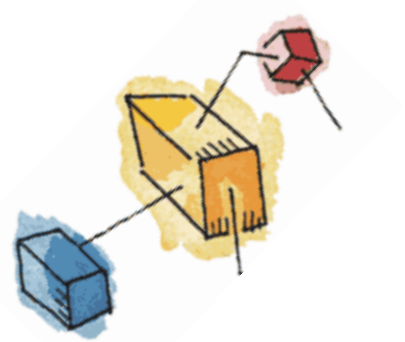
  The processors communicate with one another through various communication lines.

- Processors in a distributed system may vary in size and function.



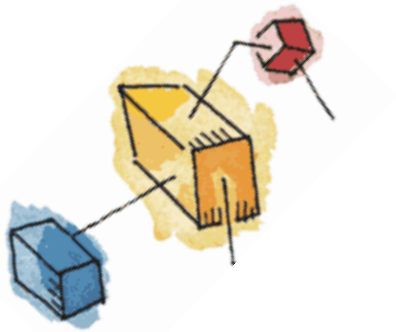
Distributed Systems

- The advantages of distributed systems are as follows:
  - Resource Sharing: With resource sharing facility user at one site may be able to use the resources available at another.
  - Communication Speedup: Speedup the exchange of data with one another via electronic mail.
  - Reliability: If one site fails in a distributed system, the remaining sites can potentially continue operating.



OS Achievements

- Processes
- Memory management
- Information protection and security
- Scheduling and resource management
- System structure



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

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

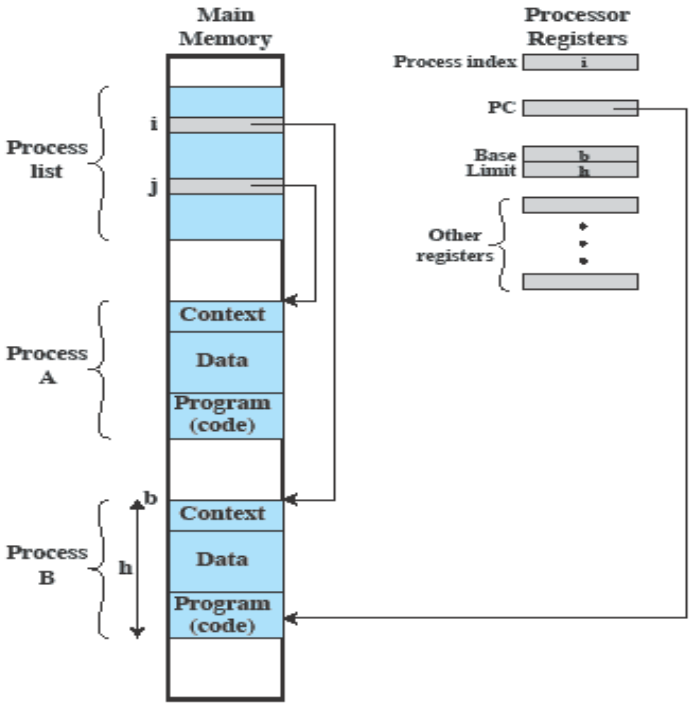


Figure 2.8 Typical Process Implementation

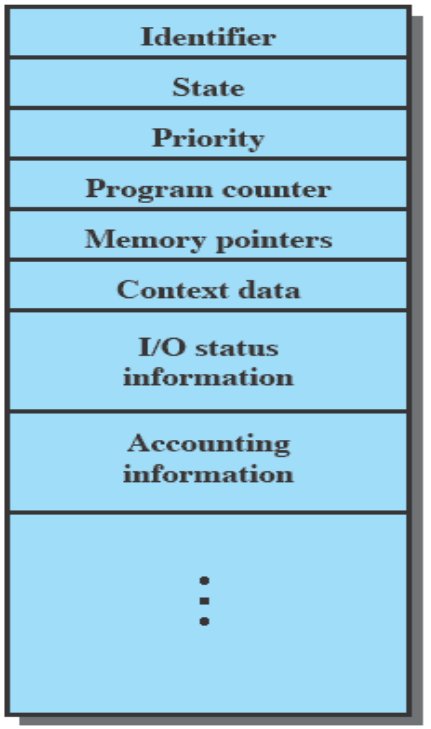
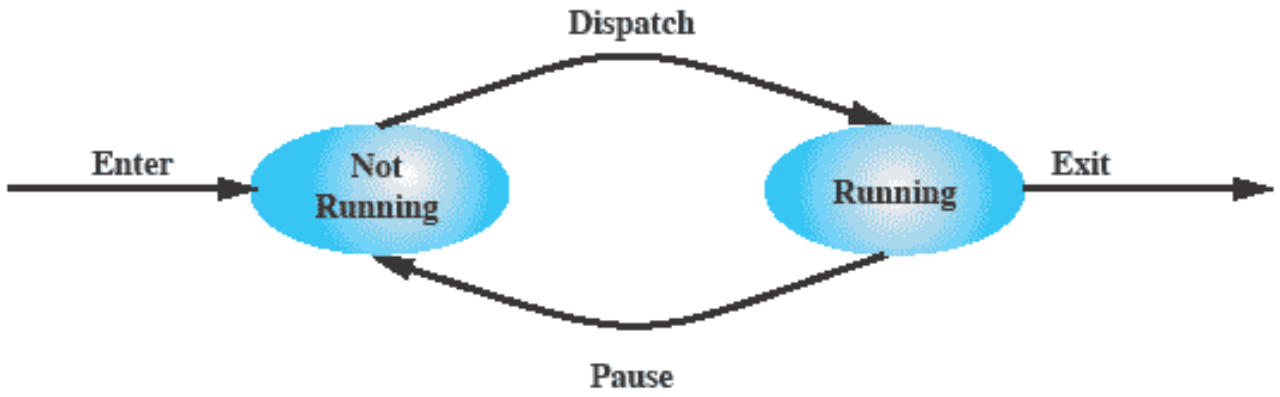


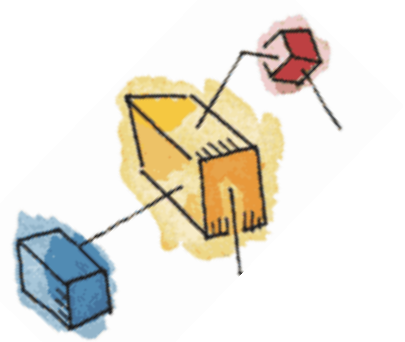
Figure 3.1 Simplified Process Control Block

Two-State Process Model

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

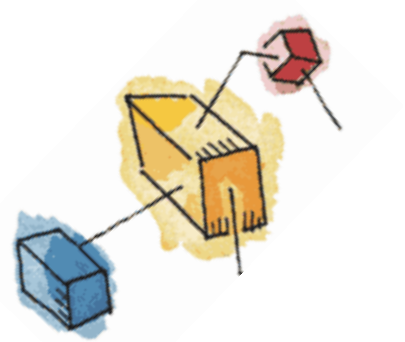


(a) State transition diagram



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

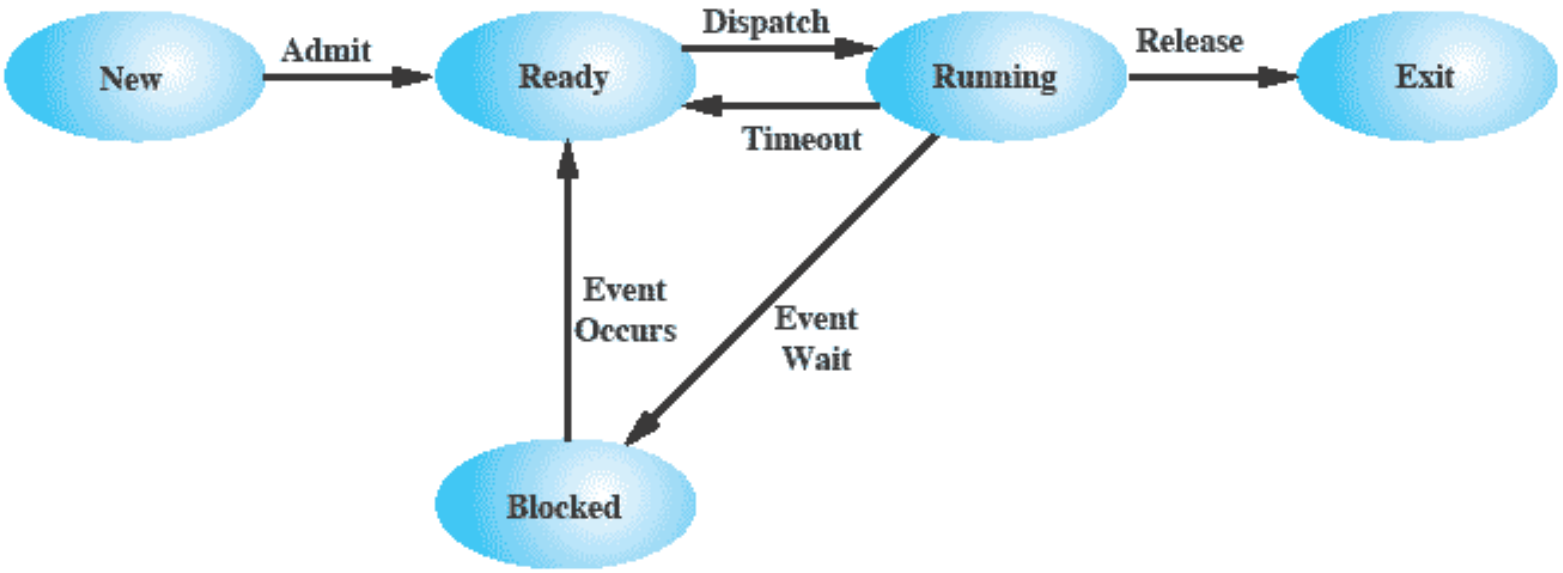
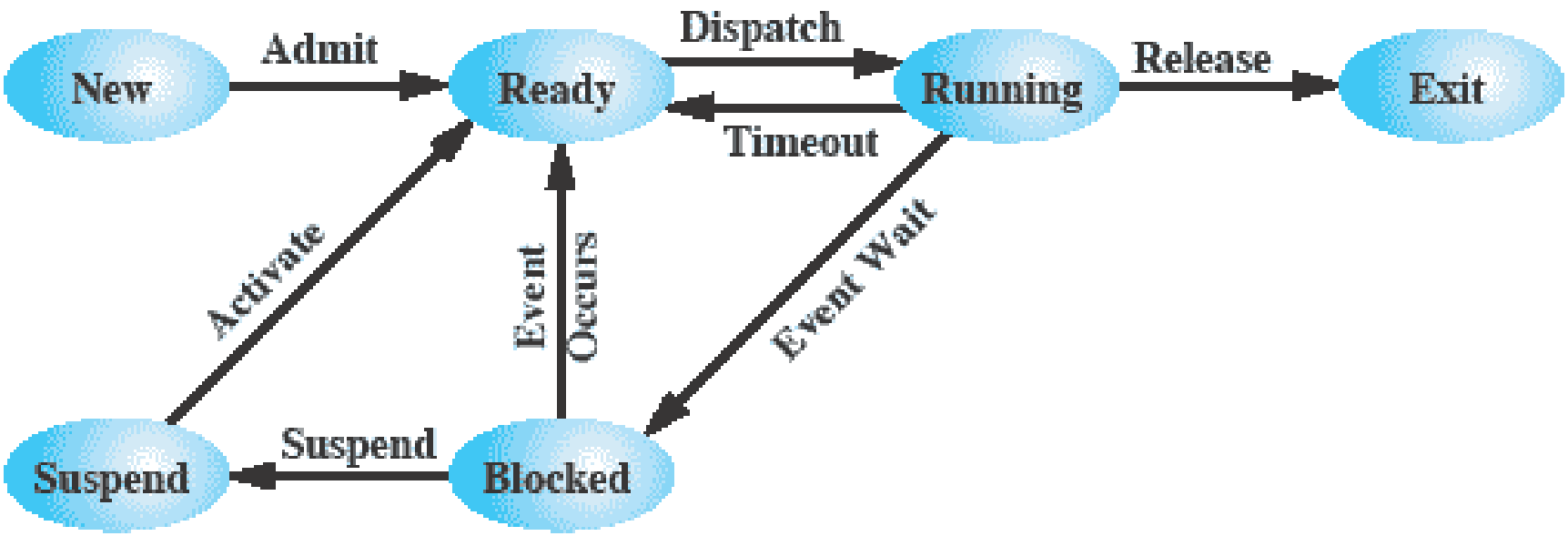


Figure 3.6 Five-State Process Model

Suspended Processes

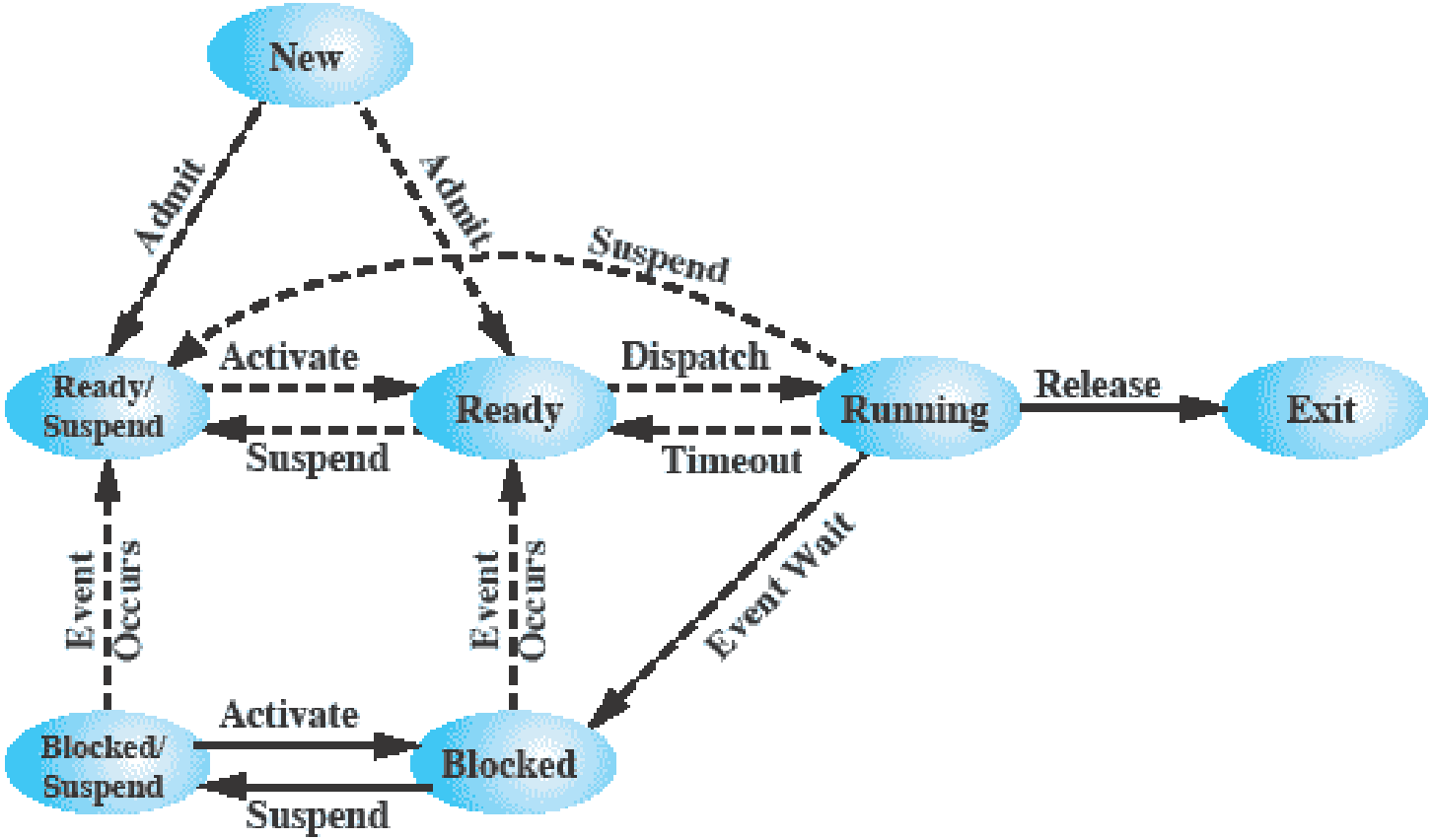
- Processor is faster than I/O so all processes could be waiting for I/O
  - Swap these processes to disk to free up more memory and use processor on more processes
- Blocked state becomes ***suspend*** state when swapped to disk
- Two new states
  - Blocked/Suspend
  - Ready/Suspend

One Suspend State



(a) With One Suspend State

Two Suspend States



(b) With Two Suspend States

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

Processes and Resources

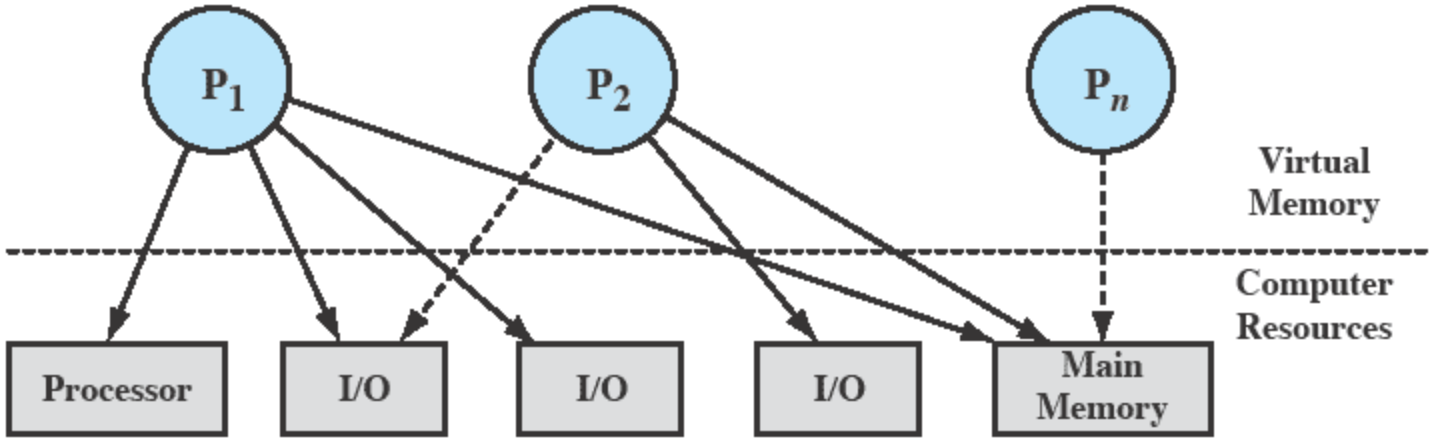
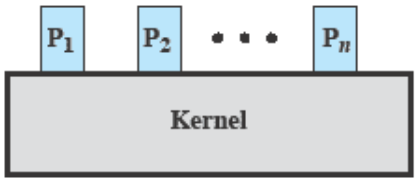


Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)

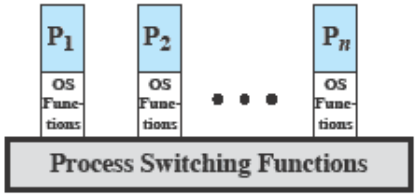
Process Modes of Execution

- 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

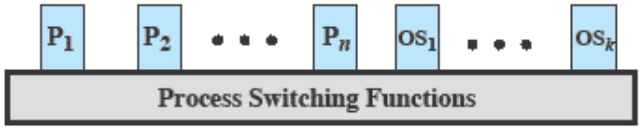
Execution of the Operating System



(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

(a) Operating system code is executed as a separate entity that operates in privileged mode.

(b) Operating system software within context of a user process.

(C) Implement the OS as a collection of system process.

Processes VS. 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***
- Threads within a single process communicate via shared memory.
- Threads located in different processes use Remote Procedure Calls (RPCs)

Threads vs. processes

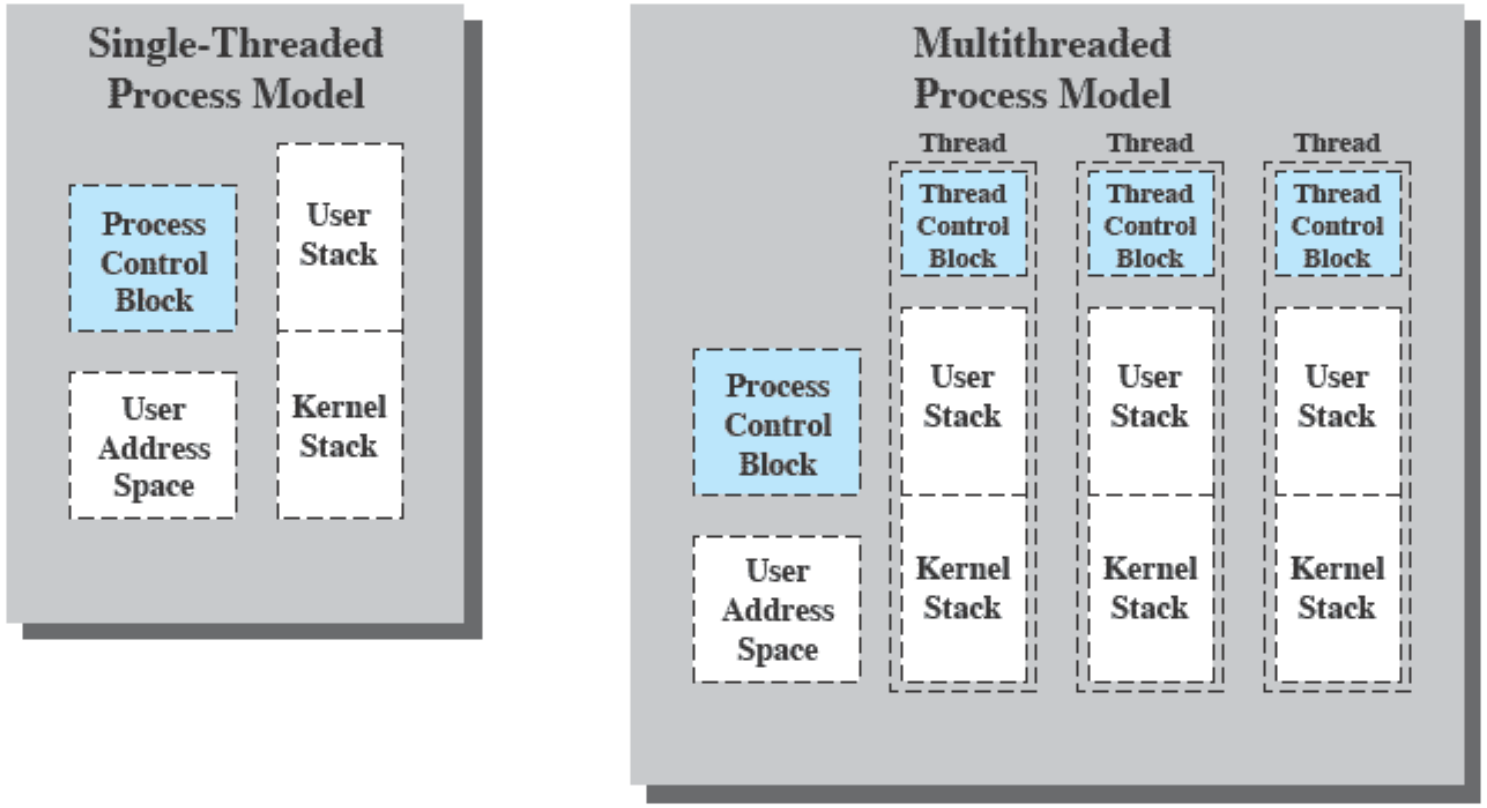


Figure 4.2 Single Threaded and Multithreaded Process Models

Multithreading

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

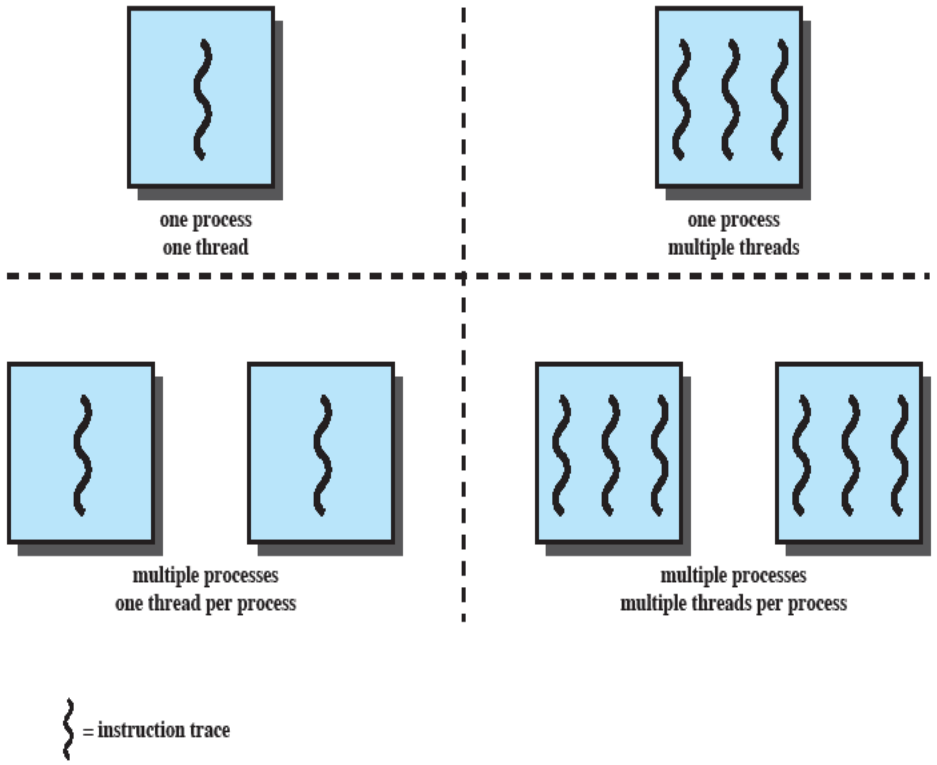


Figure 4.1 Threads and Processes [ANDE97]

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 that switching processes
- Threads can communicate with each other
  - without invoking the kernel

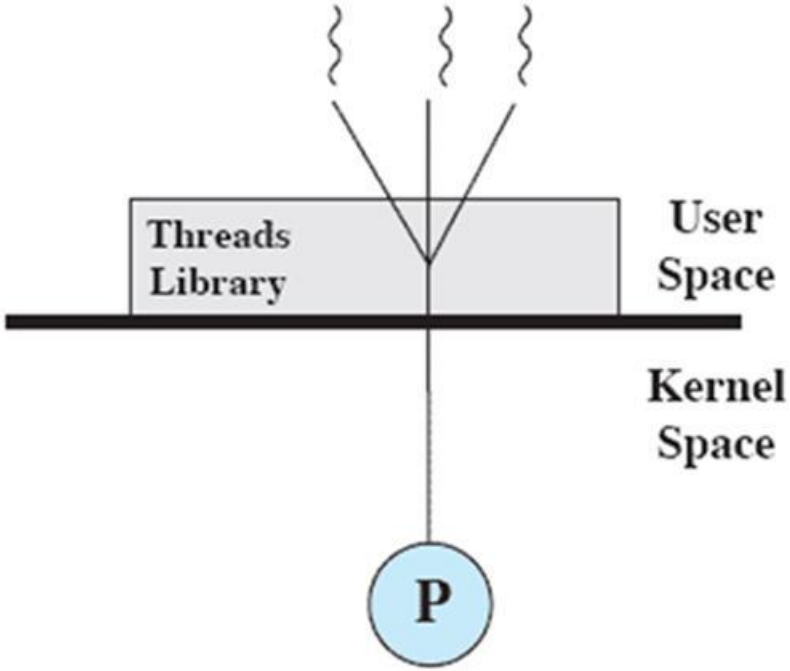
Categories of Thread Implementation

- User Level Thread (ULT)

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

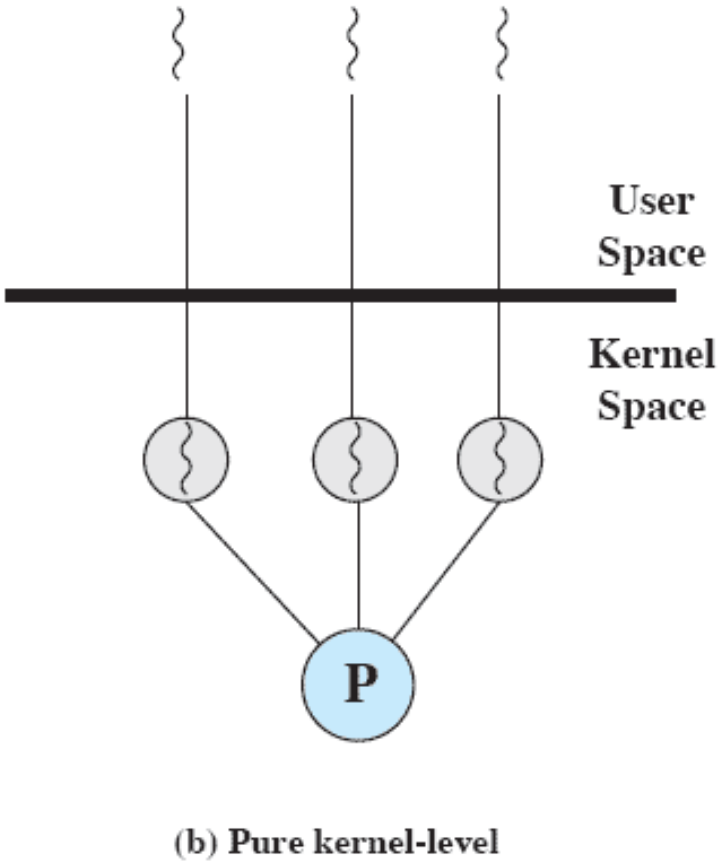
User-Level Threads

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



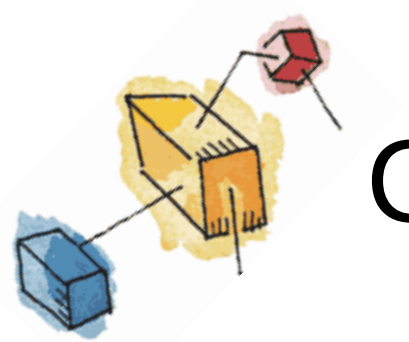
(a) Pure user-level

Kernel-Level Threads



(b) Pure kernel-level

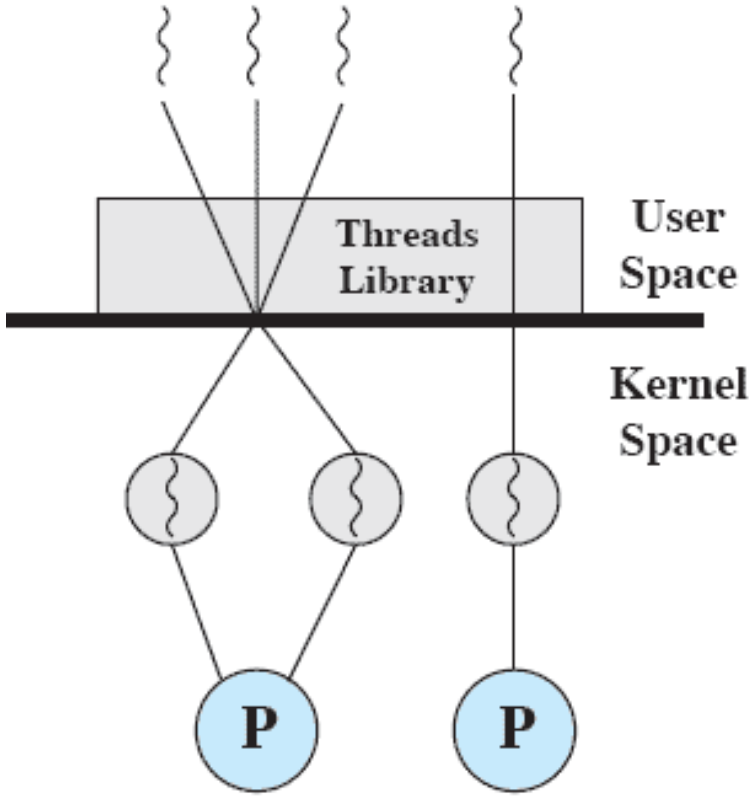
- Kernel maintains context information for the process and the threads
  - No thread management done by application
- Scheduling is done on a thread basis
- Windows is an example of this approach



Combined Approaches

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

- Example is Solaris

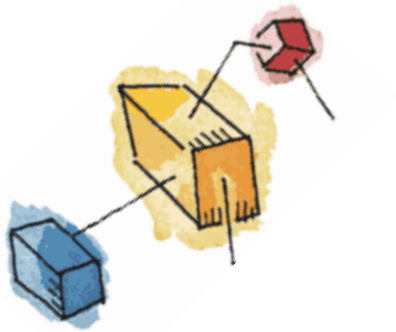


(c) Combined

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



Concurrency

Concurrency arises in:

- Multiple applications
  - Sharing time
- Structured applications
  - Extension of modular design
- Operating system structure
  - OS themselves implemented as a set of processes or threads

Difficulties of Concurrency

- Sharing of global resources
- Optimally managing the allocation of resources
- Difficult to locate programming errors as results are not deterministic and reproducible.

A Simple Example: On a Multiprocessor

Process P1

```
        .
chin = getchar();
        .
chout = chin;
putchar(chout);
        .
        .
```

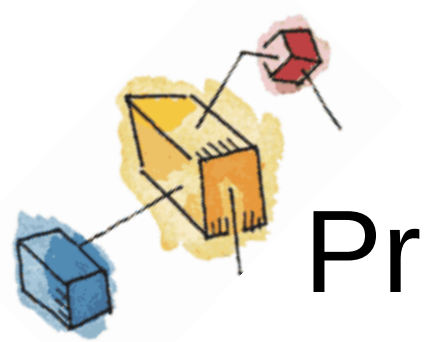
Process P2

```
        .
        .
chin = getchar();
chout = chin;
        .
putchar(chout);
        .
```

Process Interaction

Table 5.2 Process Interaction

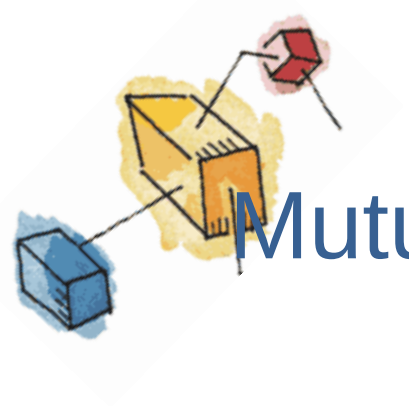
Degree of Awareness	Relationship	Influence That One Process Has on the Other	Potential Control Problems
Processes unaware of each other	Competition	• Results of one process independent of the action of others • Timing of process may be affected	• Mutual exclusion • Deadlock (renewable resource) • Starvation
Processes indirectly aware of each other (e.g., shared object)	Cooperation by sharing	• Results of one process may depend on information obtained from others • Timing of process may be affected	• Mutual exclusion • Deadlock (renewable resource) • Starvation • Data coherence
Processes directly aware of each other (have communication primitives available to them)	Cooperation by communication	• Results of one process may depend on information obtained from others • Timing of process may be affected	• Deadlock (consumable resource) • Starvation



Competition among Processes for Resources

Three main control problems:

- Need for Mutual Exclusion
  - Critical sections
- Deadlock
- Starvation



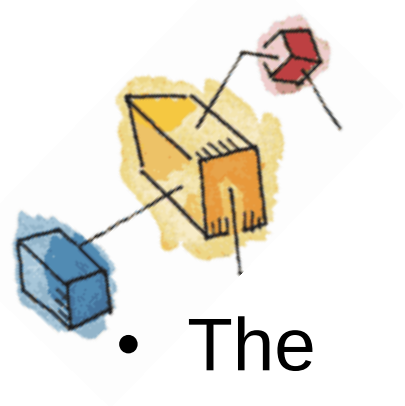
Mutual Exclusion: Hardware Support

Disabling Interrupts

- Uniprocessors only allow interleaving
- Interrupt Disabling
  - A process runs until it invokes an operating system service or until it is interrupted
  - Disabling interrupts guarantees mutual exclusion
  - Will not work in multiprocessor architecture

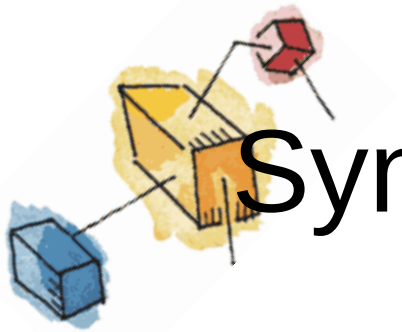
Semaphore

- Semaphore:
  - An integer value used for signalling among processes.
- Only three operations may be performed on a semaphore, all of which are atomic:
  - initialize,
  - Decrement (`semWait`)
  - increment. (`semSignal`)
- A queue is used to hold processes waiting on the semaphore
- ***Strong Semaphores*** use FIFO
- ***Weak Semaphores*** don't specify the order of removal from the queue



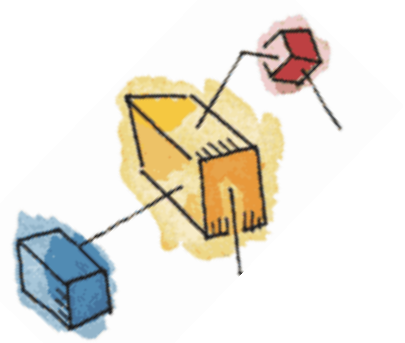
Monitors

- The monitor is a programming-language construct that provides equivalent functionality to that of semaphores and that is easier to control.
- Implemented in a number of programming languages.
- Local data variables are accessible only by the monitor
- Process enters monitor by invoking one of its procedures
- Only one process may be executing in the monitor at a time



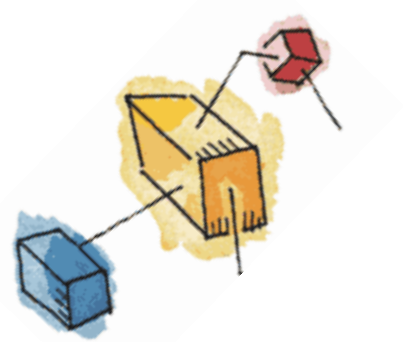
Synchronization in Monitors

- Synchronisation achieved by **condition variables** within a monitor
  - only accessible by the monitor.
- Monitor Functions:
  - Cwait(c): Suspend execution of the calling process on condition *c*
  - Csignal(c) Resume execution of some process blocked after a cwait on the same condition



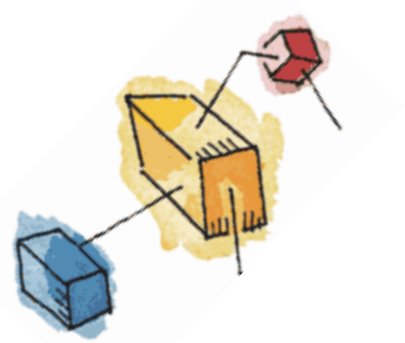
Process Interaction

- When processes interact with one another, two fundamental requirements must be satisfied:
  - synchronization and
  - communication.
- Message Passing is one solution to the second requirement
  - Added bonus: It works with shared memory *and* with distributed systems



Message Passing

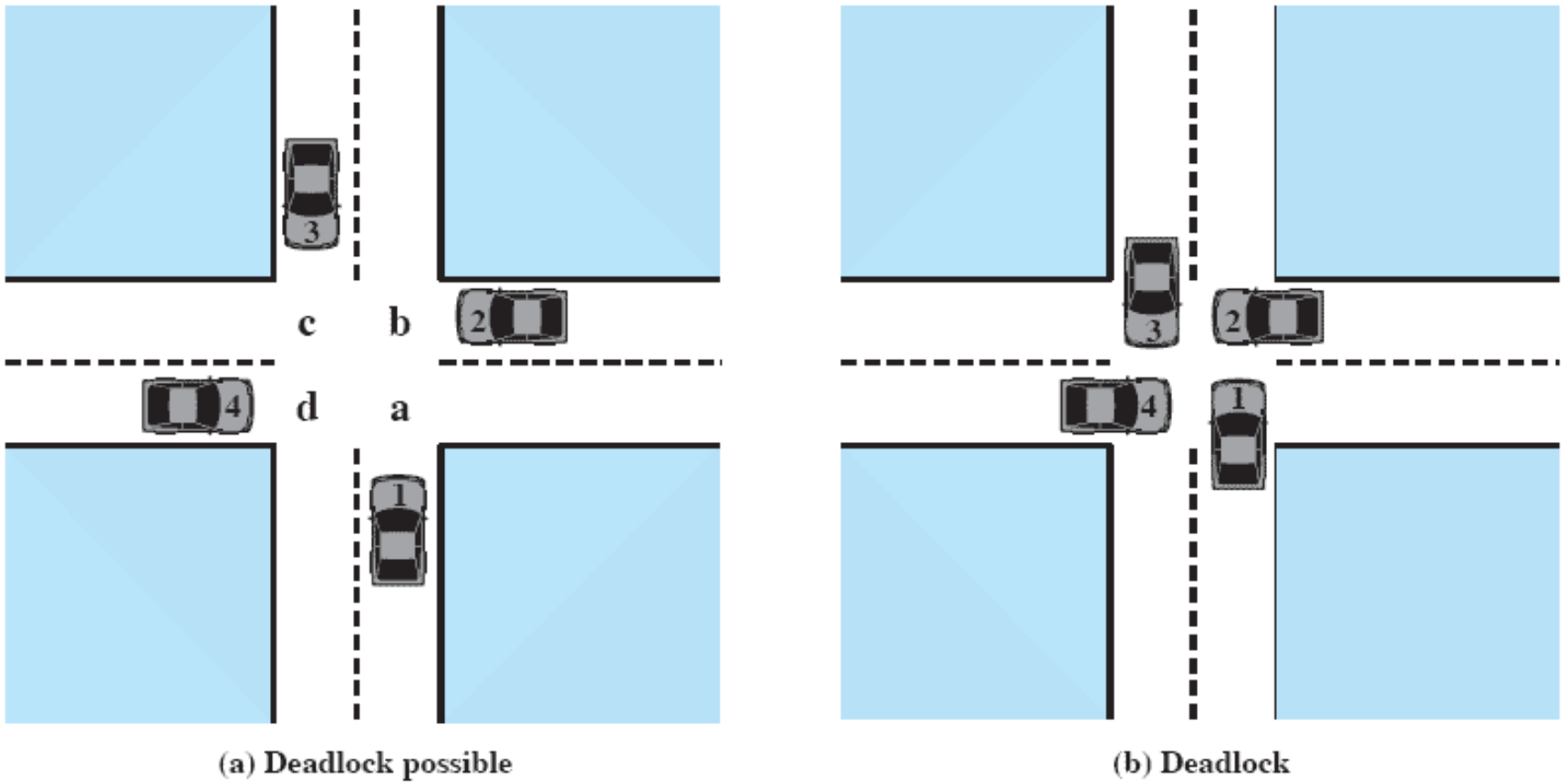
- The actual function of message passing is normally provided in the form of a pair of primitives:
- send (destination, message)
- receive (source, message)
- Communication requires synchronization
  - Sender must send before receiver can receive



Deadlock

- Permanent blocking of a set of processes that either compete for system resources or communicate with each other
- No efficient solution
- Involve conflicting needs for resources by two or more processes

Deadlock



(a) Deadlock possible

(b) Deadlock

Figure 6.1 Illustration of Deadlock

Conditions for Deadlock

- Mutual exclusion
  - Only one process may use a resource at a time
- Hold-and-wait
  - A process may hold allocated resources while awaiting assignment of others

Conditions for Deadlock

- No preemption
  - No resource can be forcibly removed form a process holding it
- Circular wait
  - A closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain

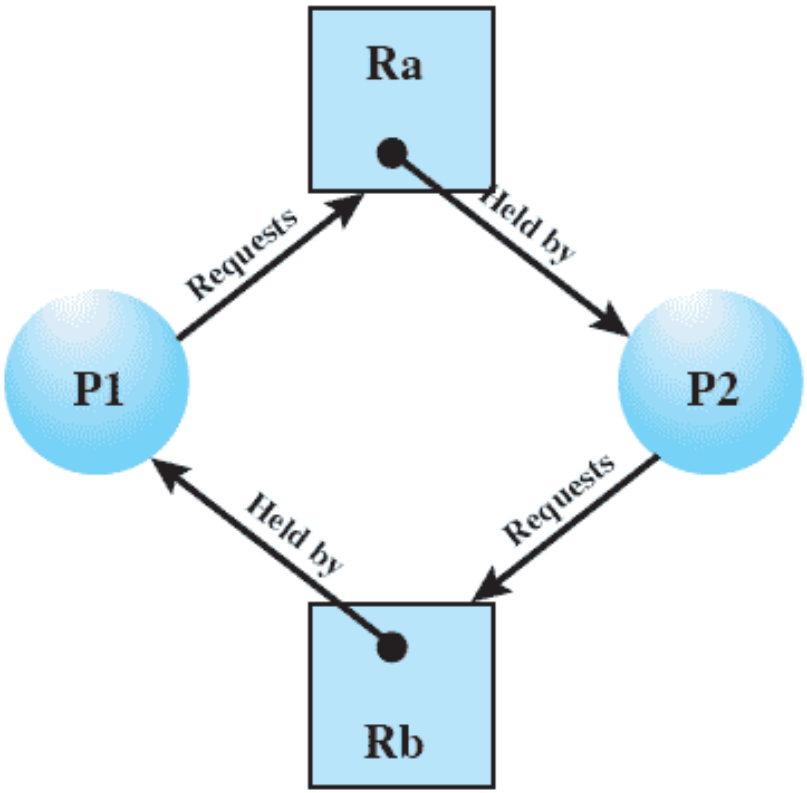
Resource Allocation Graphs

- Directed graph that depicts a state of the system of resources and processes

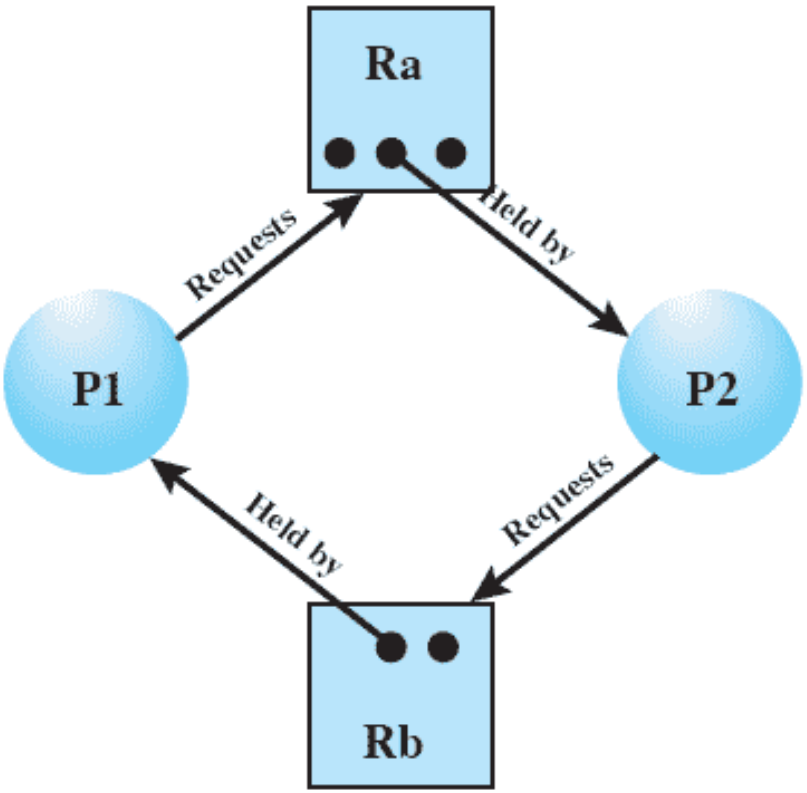
(a) Resouce is requested

(b) Resource is held

Resource Allocation Graphs



(c) Circular wait



(d) No deadlock

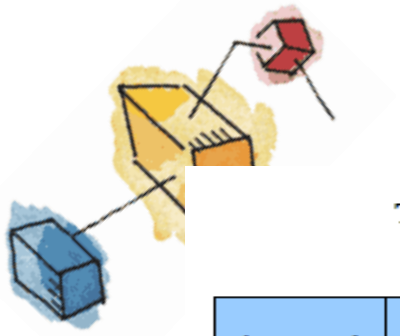


Table 6.1 Summary of Deadlock Detection, Prevention, and Avoidance Approaches for Operating Systems [ISLO80]

Approach	Resource Allocation Policy	Different Schemes	Major Advantages	Major Disadvantages
Prevention	Conservative; undercommits resources	Requesting all resources at once	•Works well for processes that perform a single burst of activity •No preemption necessary	•Inefficient •Delays process initiation •Future resource requirements must be known by processes
		Preemption	•Convenient when applied to resources whose state can be saved and restored easily	•Preempts more often than necessary
		Resource ordering	•Feasible to enforce via compile-time checks •Needs no run-time computation since problem is solved in system design	•Disallows incremental resource requests
Avoidance	Midway between that of detection and prevention	Manipulate to find at least one safe path	•No preemption necessary	•Future resource requirements must be known by OS •Processes can be blocked for long periods
Detection	Very liberal; requested resources are granted where possible	Invoke periodically to test for deadlock	•Never delays process initiation •Facilitates online handling	•Inherent preemption losses

Scheduling and Resource Management

- Fairness
  - Give equal and fair access to resources
- Differential responsiveness
  - Discriminate among different classes of jobs
- Efficiency
  - Maximize throughput, minimize response time, and adjust as many uses as possible