Secure Operating Systems

Nikos Tziritas

What is an Operating System?

- A program managing the computer hardware
- Provides a basis for application programs
- Acts as an intermediary between the computer user and the computer hardware

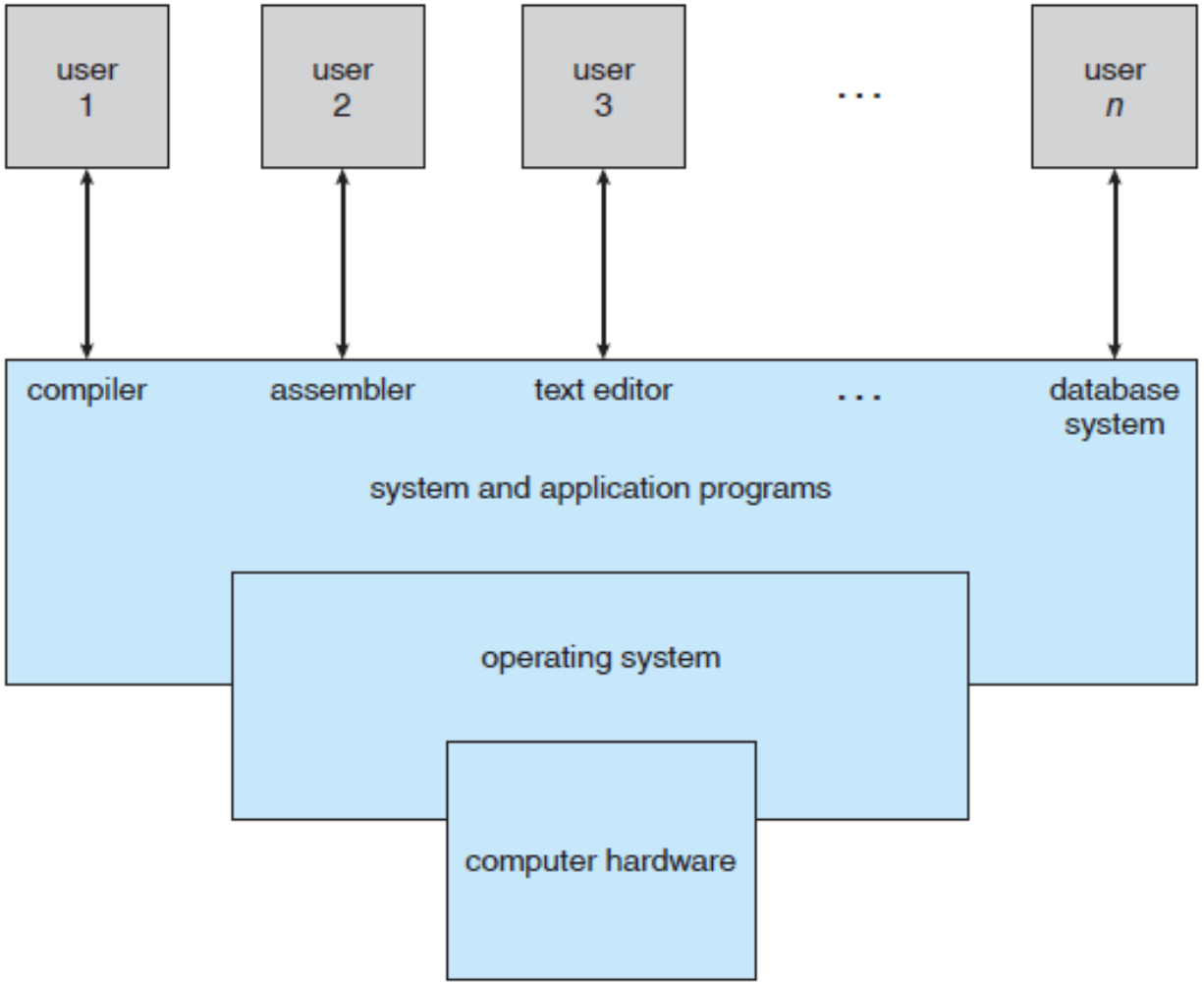
Operating System Goals

- Execute user programs and make solving user problems easier.
- Make the computer system convenient to use
- Use the computer hardware in an efficient way.

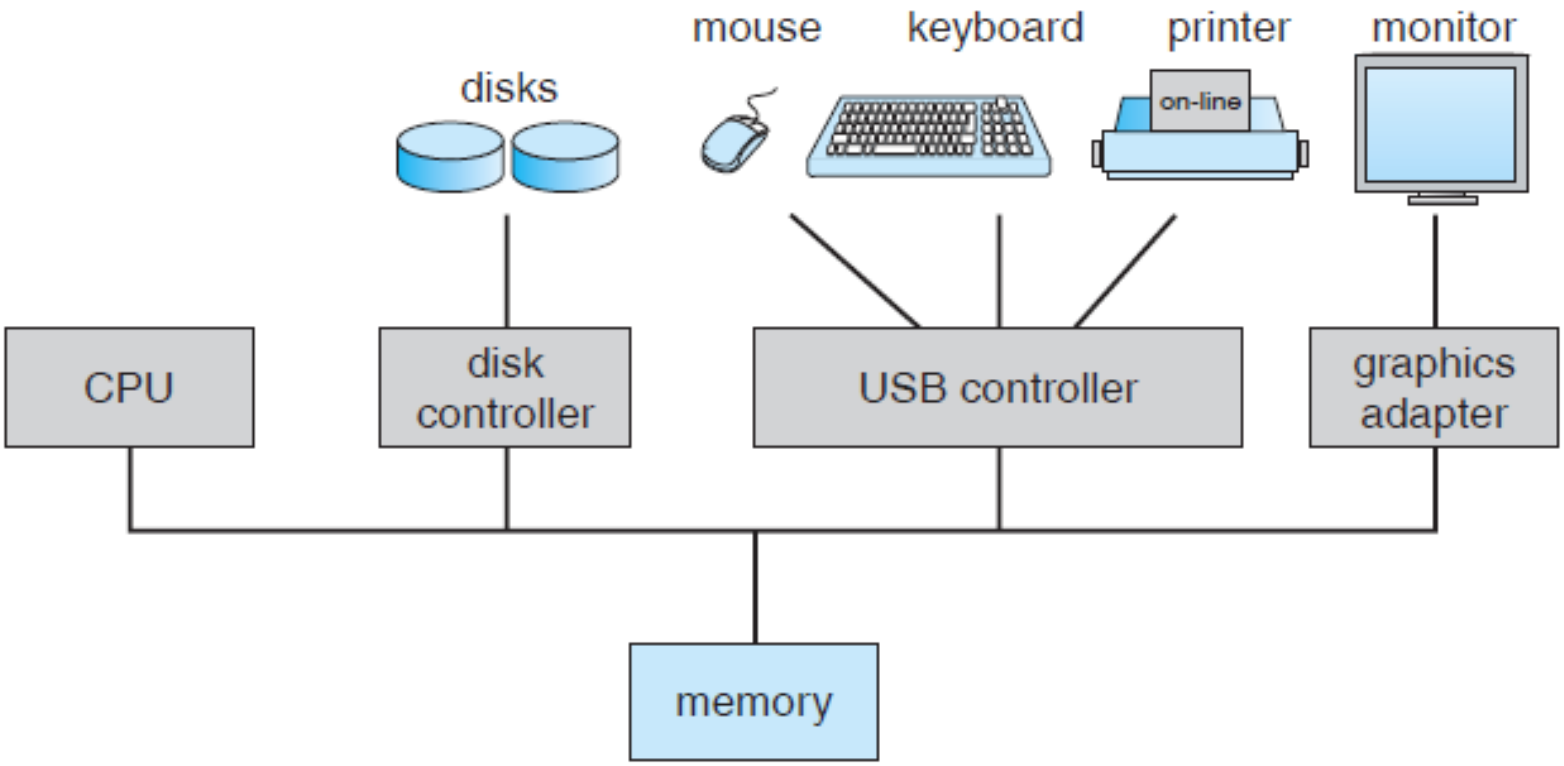
Computer System Components

- **Hardware —** represents the physical parts of the computer (hard disk drive, monitor, etc)
- **Operating system —** controls the hardware and coordinates its use among the various application programs for the various users.
- **Application programs —** define the ways in which the system resources are used to solve the computing problems of the users.
- **Users**

Computer System Components



Modern Computer System



Computer System Operation

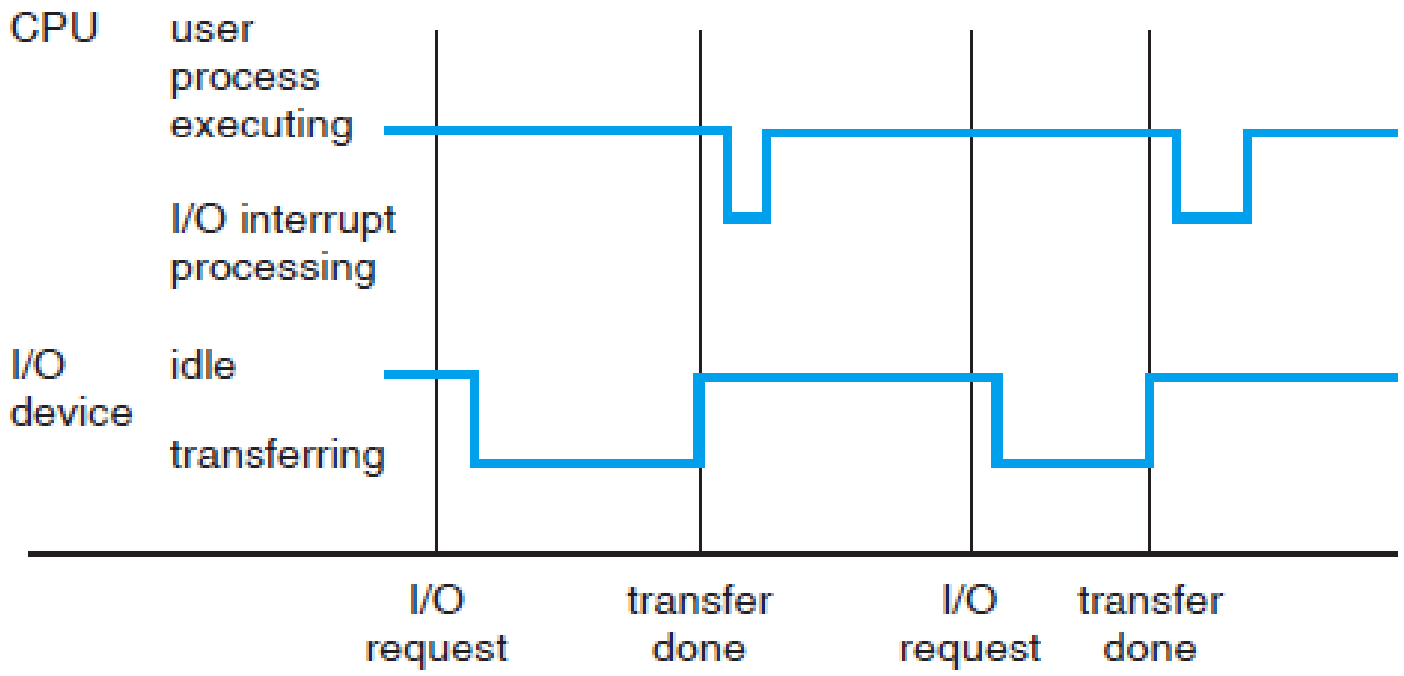
- I/O devices and the CPU can execute concurrently.
- Each device controller is in charge of a particular device type.
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an interrupt.

Interrupts

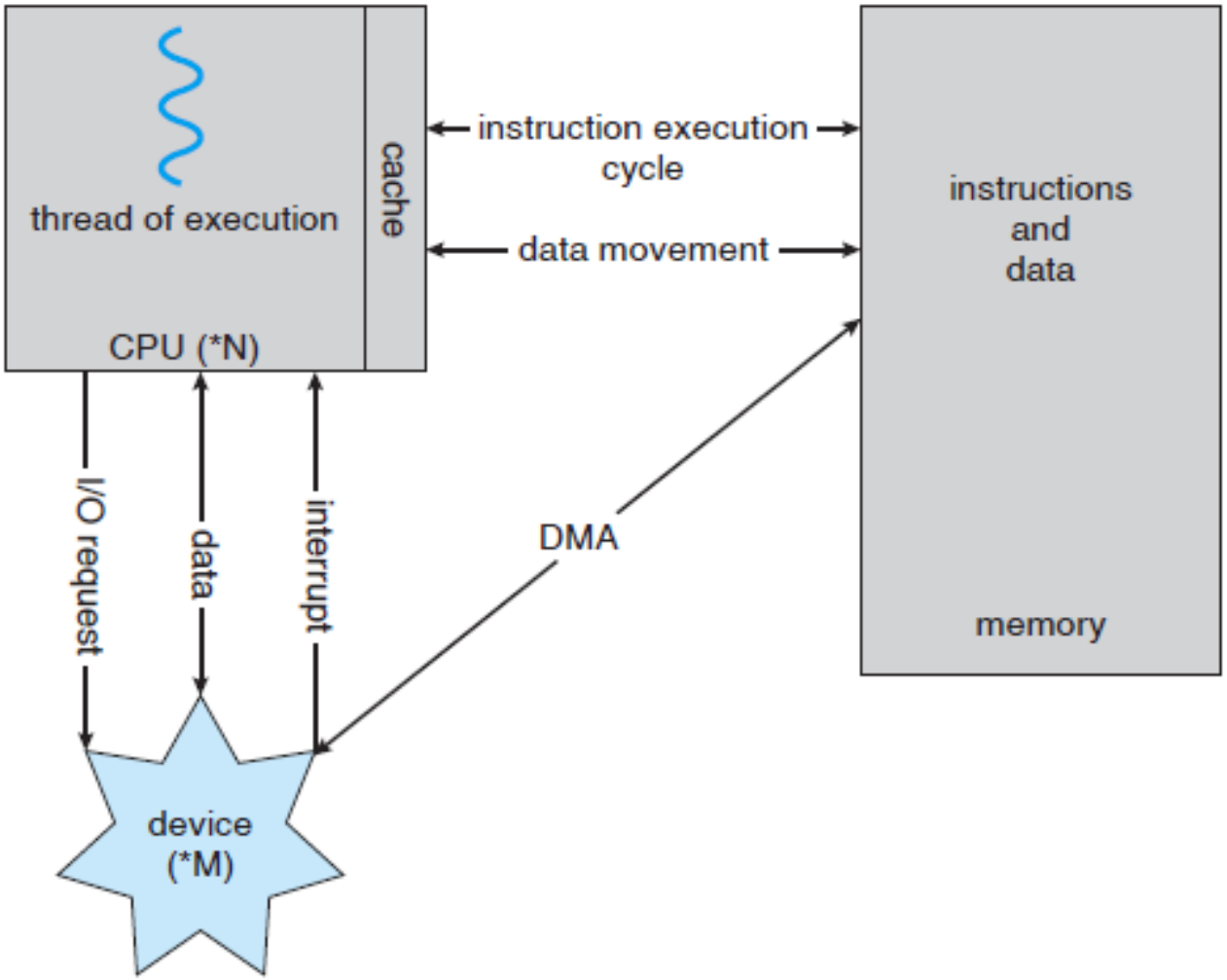
- Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction.
- Incoming interrupts are disabled while another interrupt is being processed to prevent a lost interrupt

Interrupt handling

- The operating system preserves the state of the CPU by storing registers and the program counter
- Separate segments of code determine what action should be taken for each type of interrupt


CPU
user process executing
I/O interrupt processing
I/O device
idle
transferring
I/O request
transfer done
I/O request
transfer done

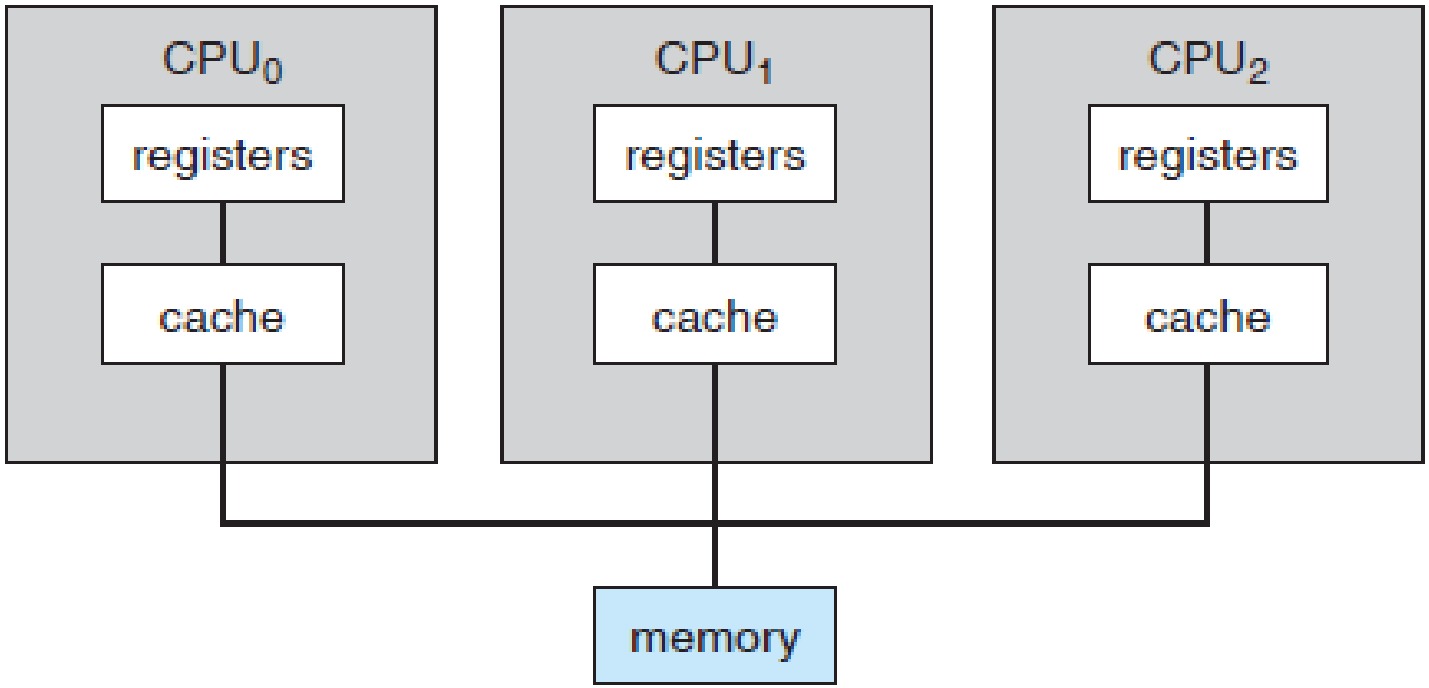
How a modern computer system works



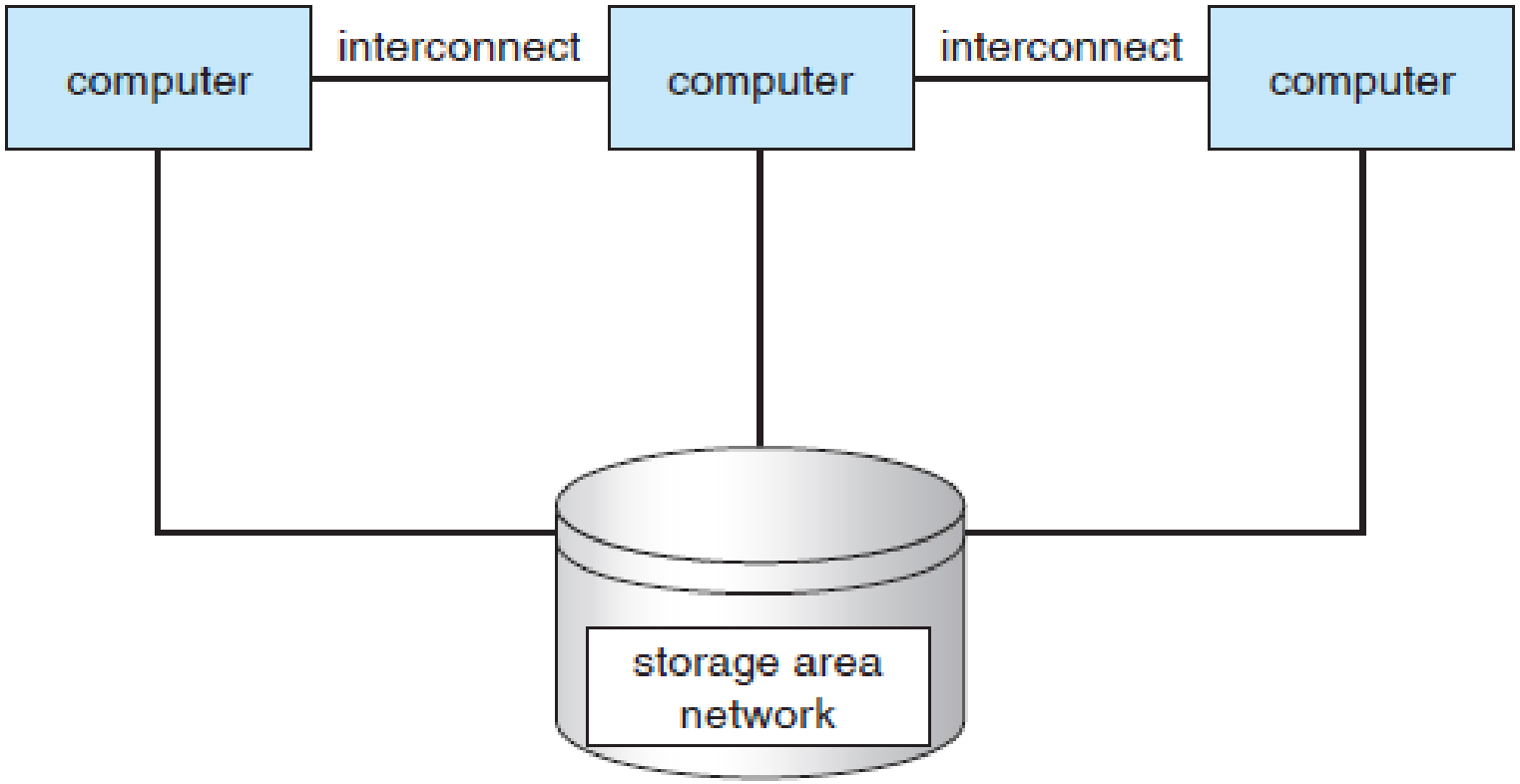
Computer-System Architecture

- Single processor systems
- Multiprocessor systems
- Clustered systems

Multiprocessing Architecture



Clustered System

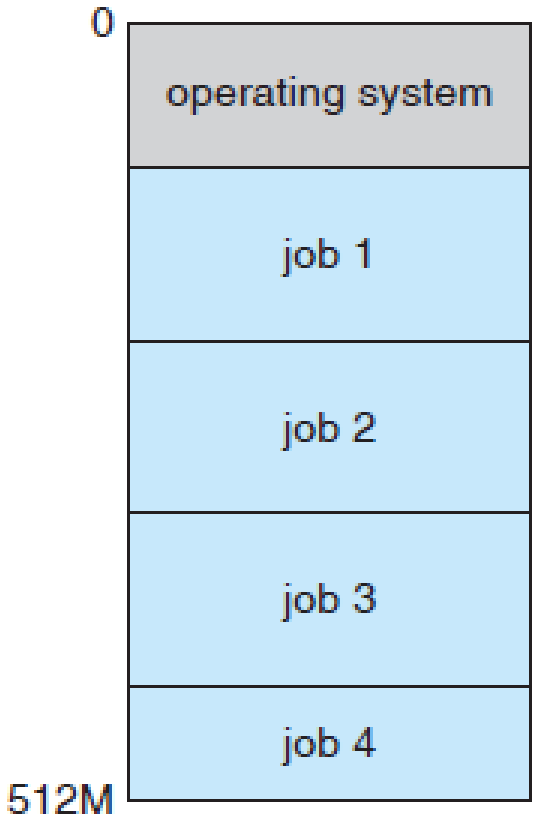


Multiprogramming

One of the most important aspects of operating systems is the ability of multi-programming.

- A single program cannot, in general, keep either the CPU or the I/O devices busy at all times.
- Single users frequently have multiple programs running.
- Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute

Memory layout for a multi-programming system



Data regulation

Due to multiple users and concurrent execution of multiple processes, access to data must be regulated, e.g.:

- Memory-addressing hardware ensures that a process can execute only within its own address space
- Device-control registers are not accessible to users, so the integrity of the various peripheral devices is protected.

Protection

- Protection is any mechanism for controlling the access of processes or users to the resources defined by a computer system
- Protection can improve reliability by detecting latent errors at the interfaces between component subsystems

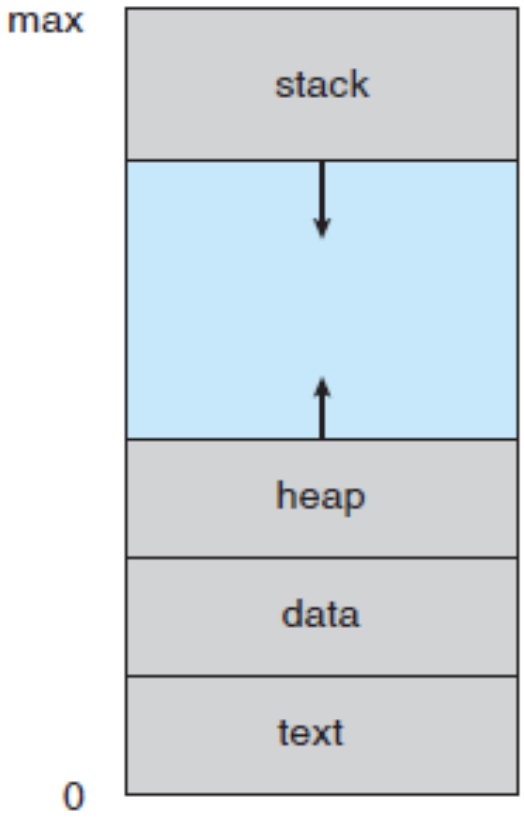
Security

- It is the job of security to defend a system from external and internal attacks.
  - Viruses and worms
  - Denial of service attacks
  - Identity theft
- Prevention of some of these attacks is considered an operating-system function on some systems, while other systems leave the prevention to additional software.

Processes

- Process is a program in execution; process execution must progress in a sequential fashion.
- A process includes:
  - Current activity (program counter, contents of processor's registers)
  - Stack
  - Data section
  - Text (code)
  - Heap

Process in Memory

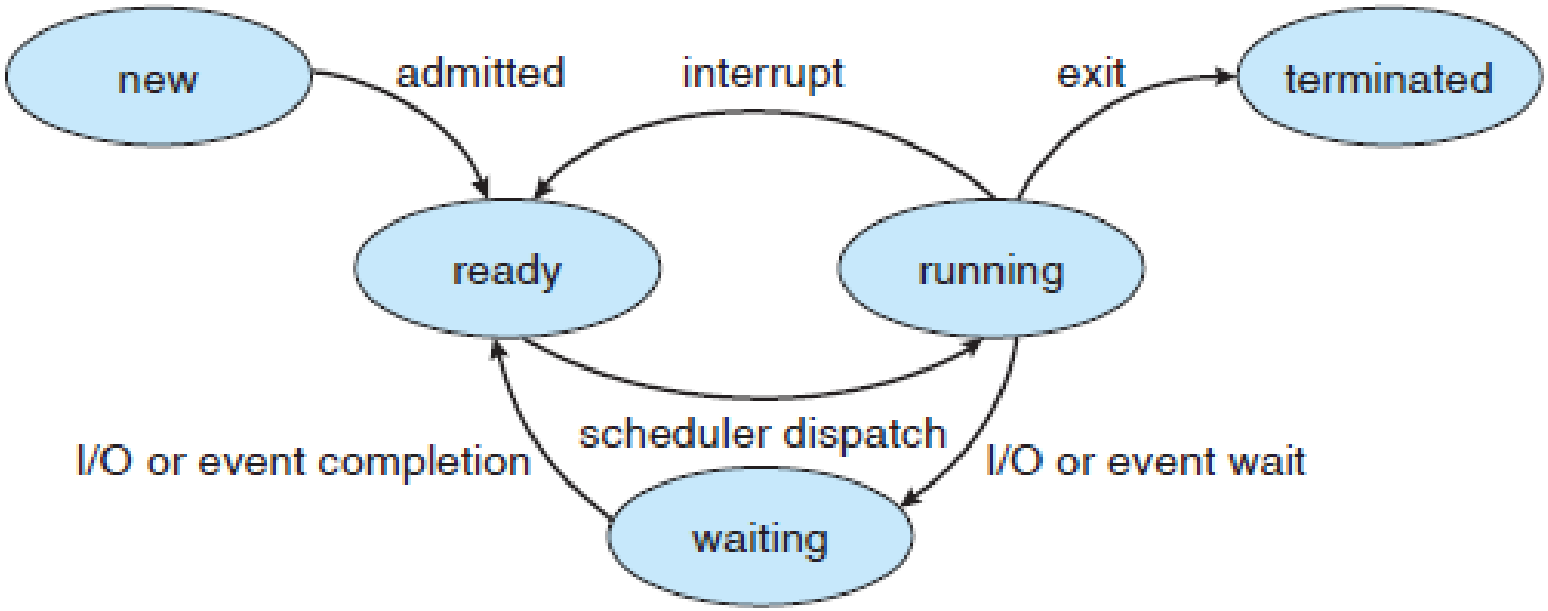


Process state

During the execution of a process, the process changes among the following states:

- **New:** the process is being created
- **Running:** Instructions are being executed
- **Waiting:** The process is waiting for some event to occur
- **Ready:** The process is waiting to be assigned to a processor
- **Terminated:** The process has finished execution

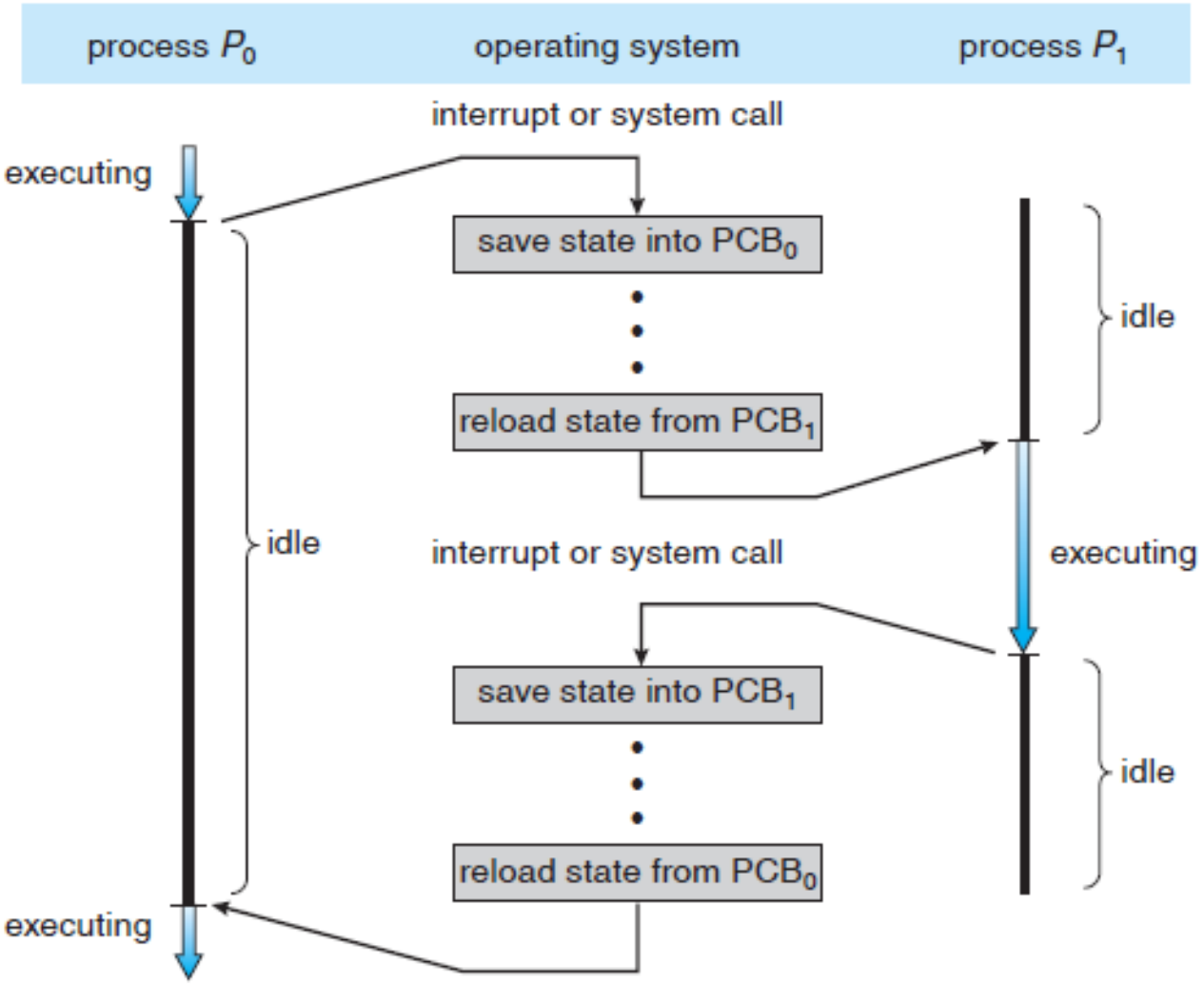
Diagram of Process State



Program Control Block

- Process state
- Program counter
- CPU registers
- Process privileges
- CPU scheduling information
- Memory management information
- Accounting information
- I/O status information

CPU switch from process to process

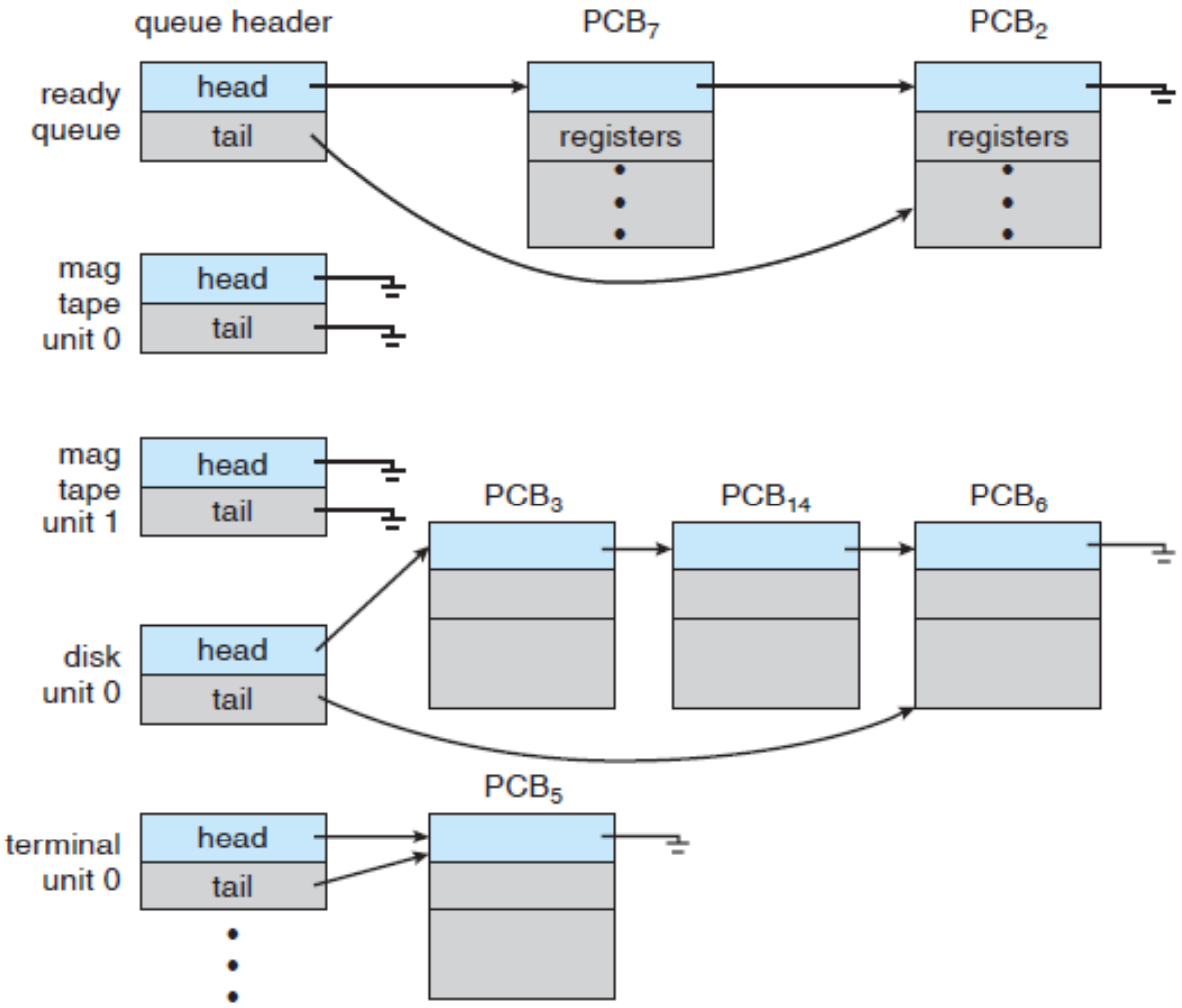


Process Scheduling Queues

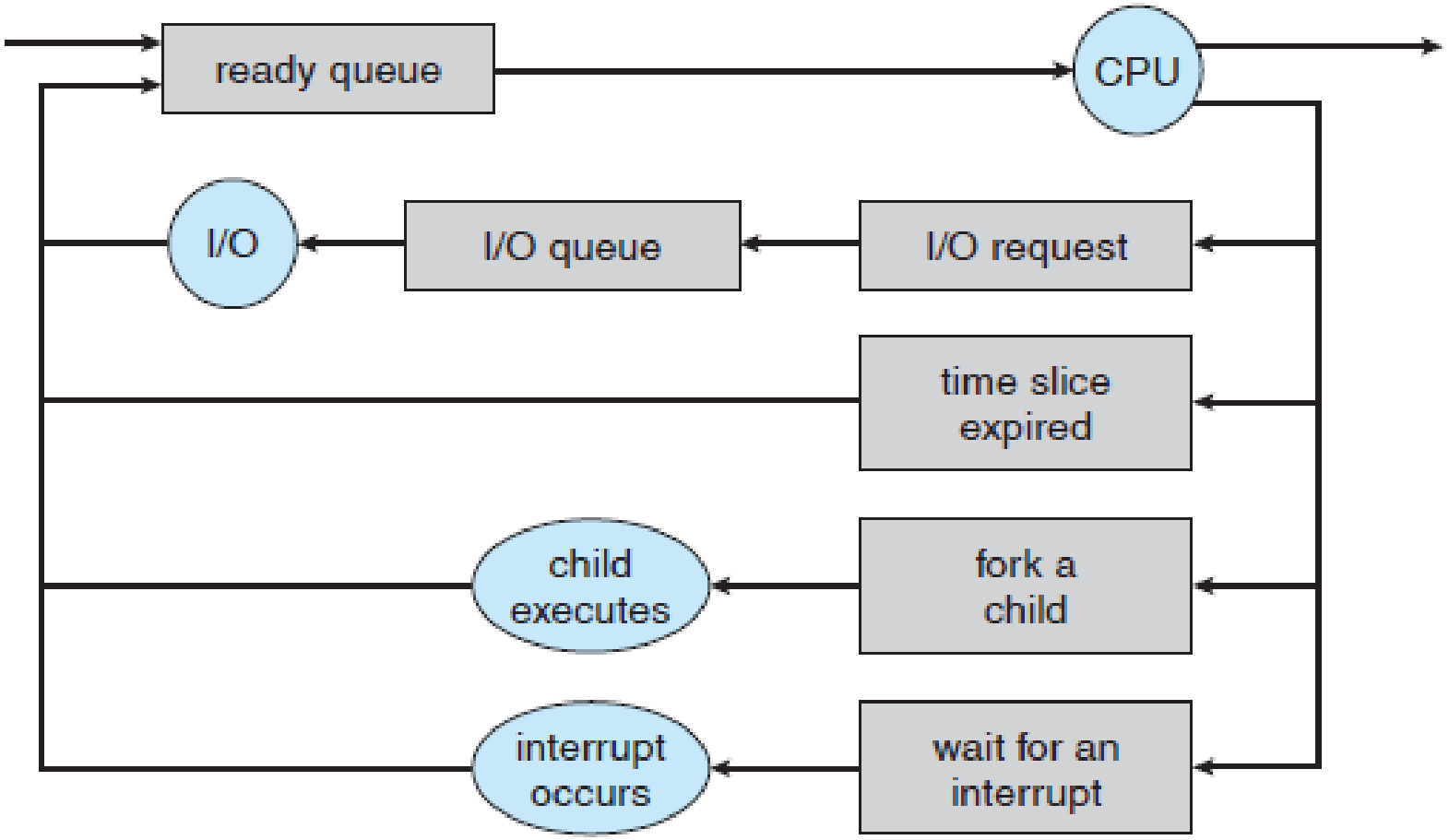
Process migration among the following queues:

- **Job queue:** set of all processes within the system
- **Ready queue:** set of all processes residing in main memory, ready and waiting to execute
- **Device queues:** set of processes waiting for an I/O device

Ready Queue and Various I/O Queues



Queueing Diagram Representation of Process Scheduling



Interprocess Communication

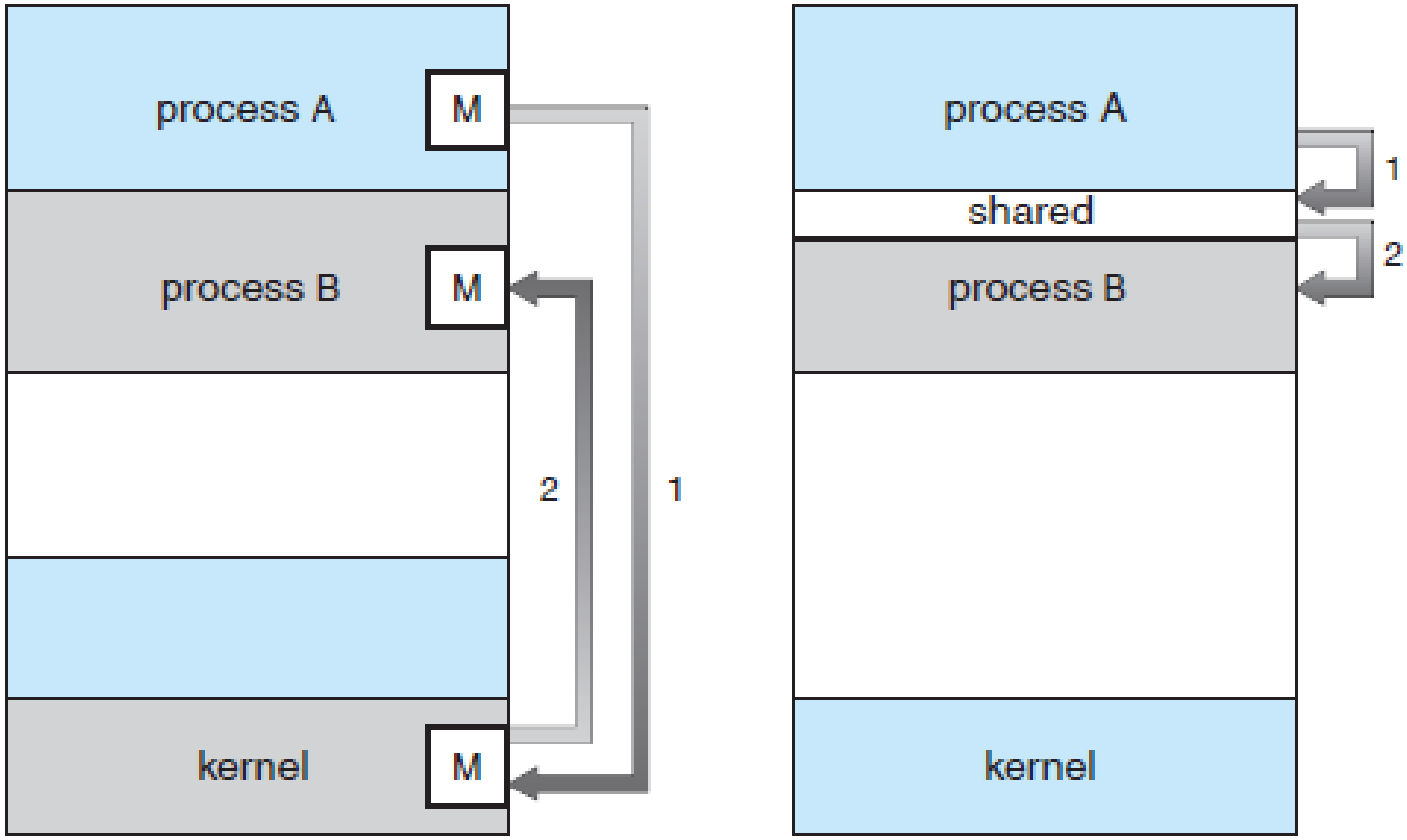
There are several reasons for providing an environment that allows process cooperation:

- **Information sharing:** Several users may be interested in the same piece of information
- **Computation speedup:** If we want a particular task to run faster, we must break it into subtasks. This **usually** works with multiple processing elements, is it possible to work with only one processing element?
- **Convenience:** Even an individual user may work on many tasks at the same time.

Interprocess Communication Mechanisms

- **Shared memory:** A region of memory that is shared by cooperating processes is established.
- **Message passing:** Communication takes place by means of messages exchanged between the cooperating processes.

Communication models



(a) Message passing

(b) Shared memory

Shared Memory

- The piece of shared memory is considered critical if more than one processes write on it. Therefore, the protection of that piece of memory is necessary.

Message Passing

- Two commands (send, receive)
- Direct or indirect communication
- Synchronous or asynchronous communication

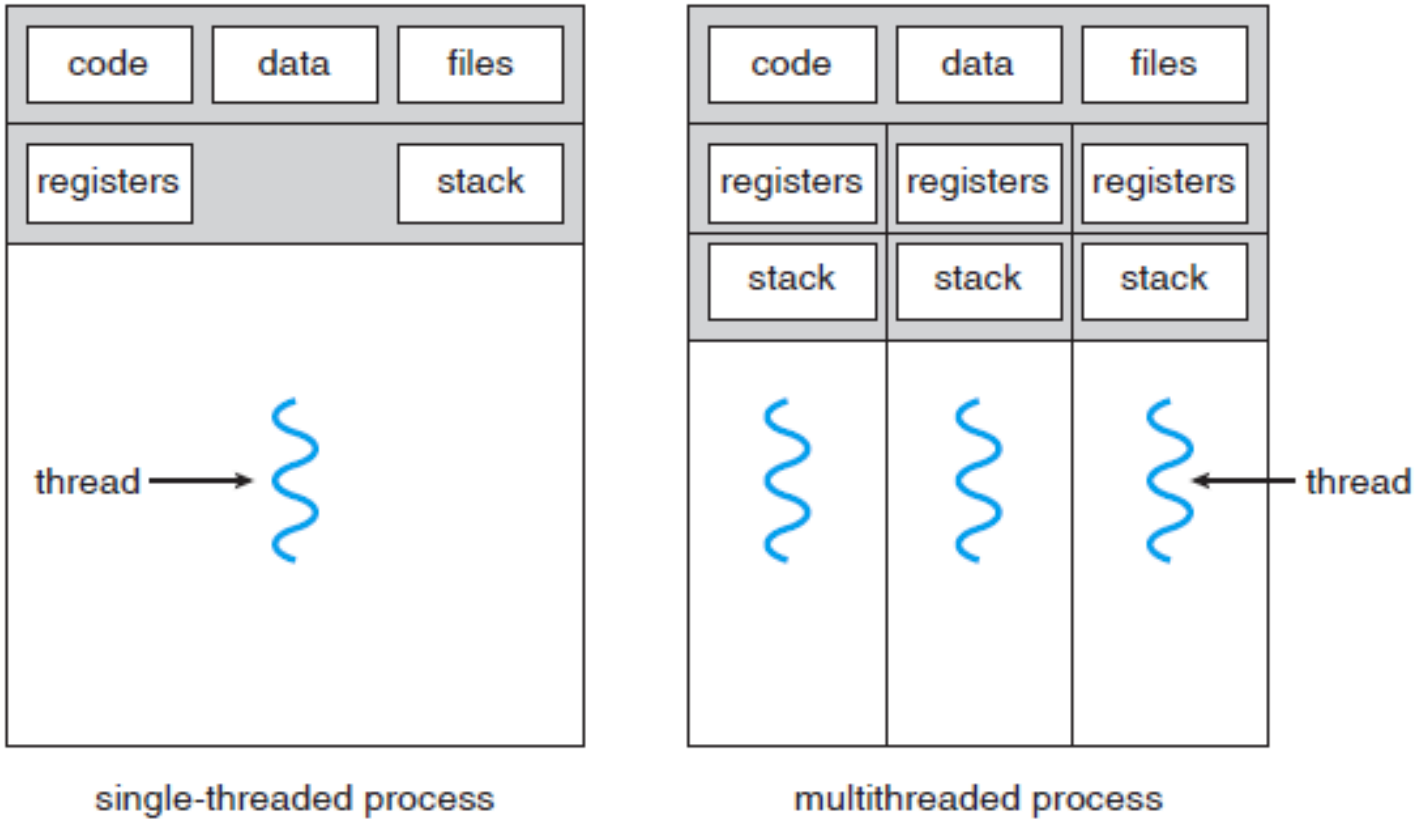
Threads

- A thread is a piece of code that must be executed in a concurrent/parallel way with other threads
- A thread comprises a thread ID, a program counter, a register set, and a stack
- It shares with other threads belonging to the same process its code section, data section, and other operating-system resources.

Threads (2)

- A traditional (or heavyweight) process has a single thread of control. Such a process is called **single-threaded**.
- If a process has multiple threads of control can perform more than one task at a time. Such a process is called **multi-threaded**.

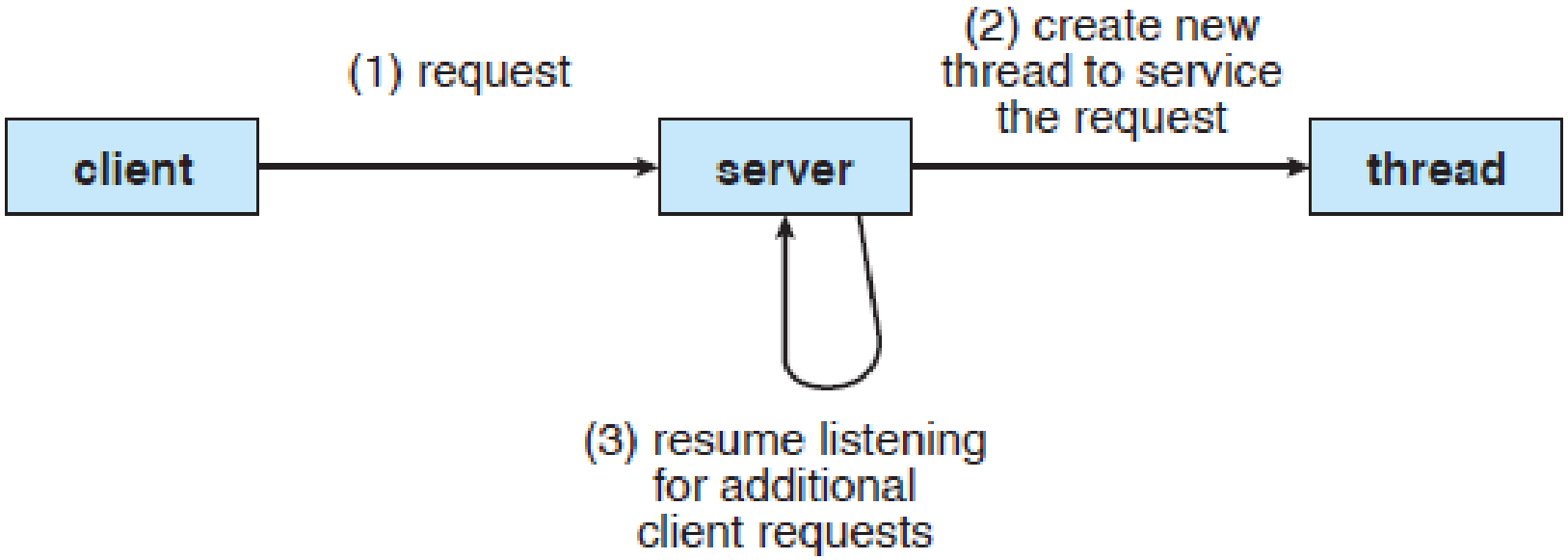
Single-threaded and multi-threaded processes



single-threaded process

multithreaded process

Example for the usefulness of threads



Major Benefits of Threads

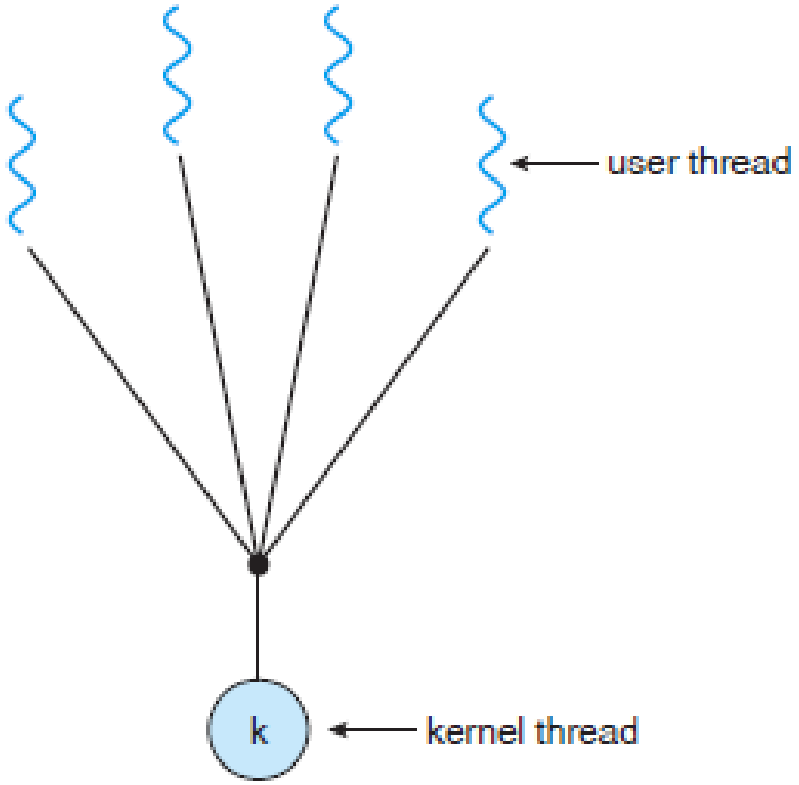
- Responsiveness (blocking part of a process)
- Resource sharing (memory)
- Economy (process creation is costly)
- Scalability (multi-processor architectures)

Multi-threading models

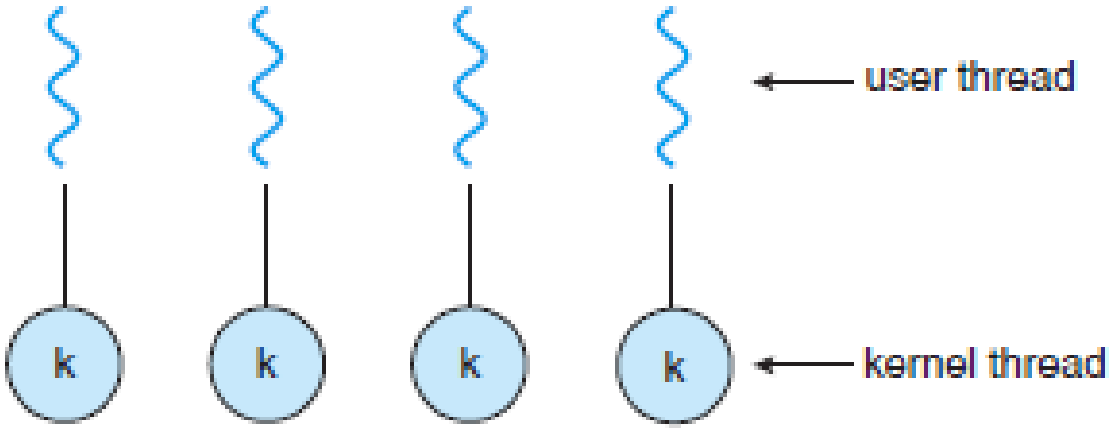
Threads split into two categories: (a) user threads, (b) kernel threads.

- Many-to-One model
- One-to-One model
- Many-to-Many model

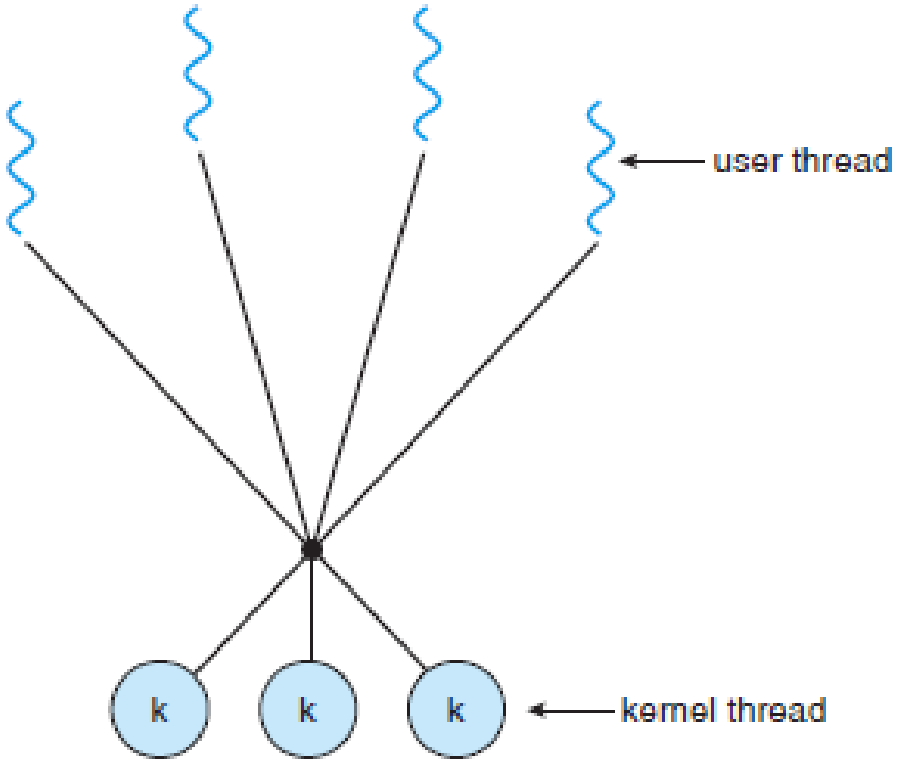
Many-to-One Model



One-to-One Model



Many-to-Many Model



Thread pools

Thread pools offer the following benefits:

- Servicing a request with an existing thread is usually faster than waiting to create a thread
- A thread pool limits the number of threads that may exist at any point in time. This is important on systems that cannot support a large number of concurrent threads.

CPU Scheduler

- Whenever CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed
- The selection process is carried out by the CPU scheduler.
- The ready queue is not necessarily a first-in, first-out (FIFO) queue.

CPU Scheduling Decisions

- When a process switches from the running state to the waiting state (I/O request)
- When a process switches from the running state to the ready state
- When a process a process switches from the waiting state to the ready state
- When a process terminates

Scheduling Criteria

- CPU utilization
- Throughput
- Turnaround time
- Waiting time
- Response time

Scheduling Algorithms

- First-come, first-served (FCFS)
- Shortest-Job-First (SJF)
- Priority scheduling algorithm
- Round-Robin Scheduling

Process Synchronization

- Concurrent access to shared data may result in data inconsistency
- There is a need of mechanisms ensuring the orderly execution of cooperating processes that share a logical address space, so that data inconsistency is maintained

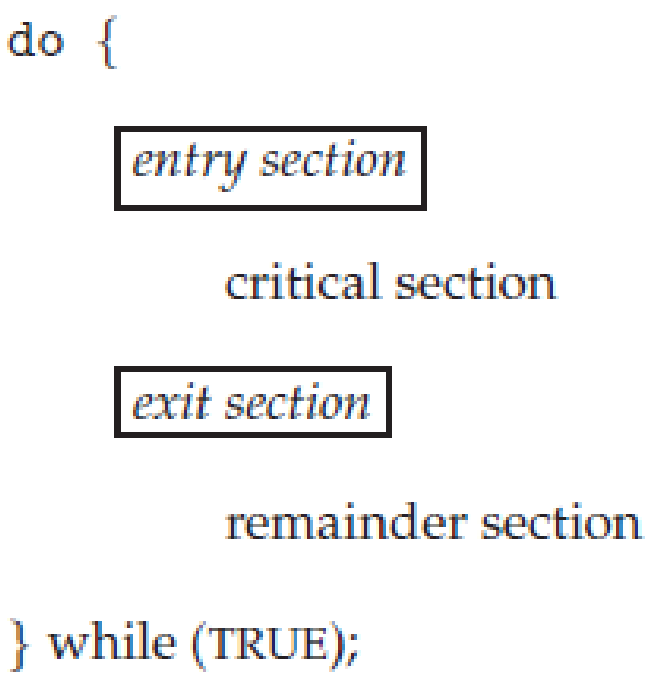
Critical Section

- Each process has a segment of code, called a critical section, in which the process may be:
  - Changing common variables
  - Updating a table
  - Writing a file
- The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section.

Critical Section Problem

- The critical section problem is to design a protocol that the processes can use to gain access to their critical sections.
- Each process must request permission to enter its critical section
- The section of code implementing this request is the **entry section**. The critical section may be followed by an **exit section**. The remaining code is the **remainder section**.

General structure of a typical process



Critical Section Problem (2)

A solution to the critical section problem must satisfy the following requirements:

- Mutual exclusion
- Progress
- Bounded waiting

Mutual exclusion

If process *Pi* is executing in its critical section, then no other processes can be executing in their critical sections.

Progress

If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next, and this selection cannot be postponed indefinitely.

Bounded Waiting

There exists a bound, or limit, on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.