Operating System(315319)

Unit 2 : Process Management

Hours:10 **Marks: 14**

Prepared by:

Mrs. Kousar Ayub A.

Lecturer(Selection Grade)

Computer Engg. Dept

M. H. Saboo Siddik Polytechnic

Learning Outcomes

The learners will be able to:

- Processes: process state, process control block.
- Process Scheduling: scheduling queues, types of schedulers, context switch
- Inter Process Communication: Shared memory system, Message passing system
- Threads: Benefits, User and Kernel level threads, Multithreading Models: One to One, Many to One, Many to Many
- Execute process commands like: top, ps, kill, wait, sleep, exit, nice

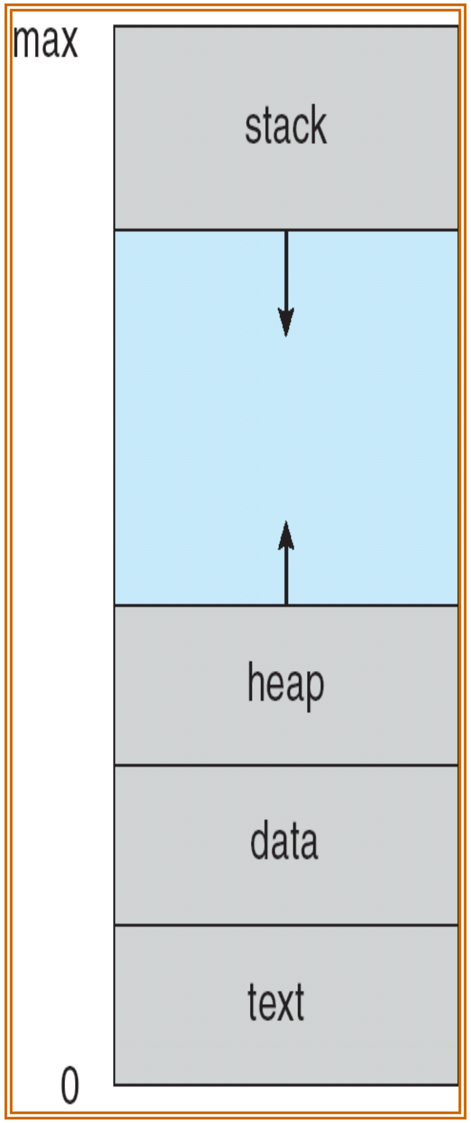
Process Management

- Process: process state, process control block (PCB).
- Process scheduling: Scheduling queues, Scheduler, context switch.
- Inter-process Communication(IPC): Introduction, shared memory and message passing system.
- Threads - Benefits, user and kernel threads, Multithreading Models - Many to one, one to one, many to many.
- Execute Process commands like ps, wait, sleep, exit, kill

Process Concept

- Process – a program in execution; process execution must progress in sequential fashion.
- A process is a program in execution. Process is also called as job, task and unit of work.
- A process is defined as, "an entity which represents the basic unit of work to be implemented in the system".
- A process includes:
  - program counter
  - stack
  - data section

Process in Memory



Each process has following sections:

1. A **Text section** that contains the program code.

2. A **Data section** that contains global and static variables.

3. The **heap** is used for dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.

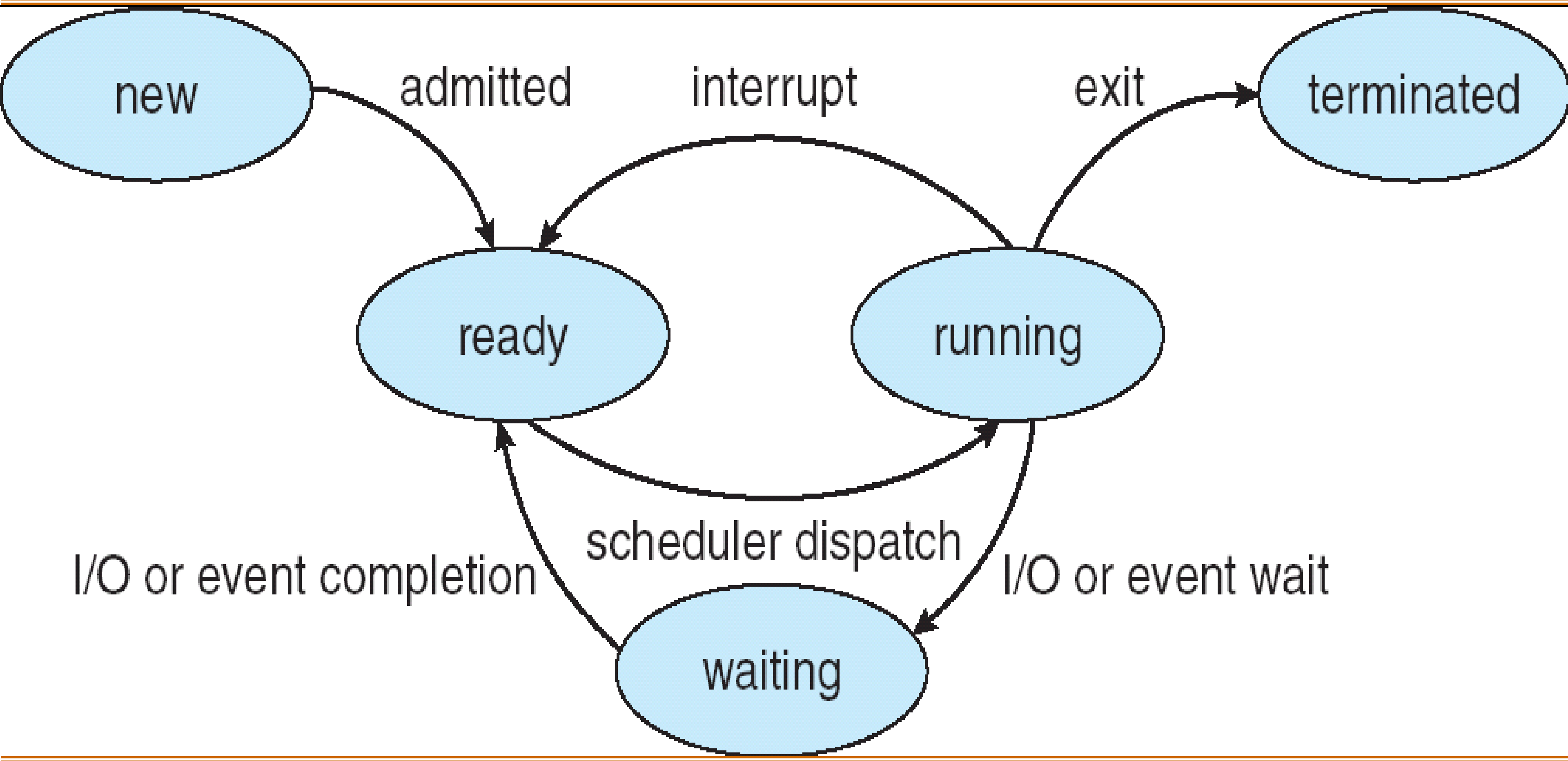
4. The **stack** is used for local variables. A process stack which contains the temporary data (such as subroutine parameters, return addresses, and temporary variables). Space on the stack is reserved for local variables when they are declared (at function entrance or elsewhere, depending on the language), and the space is freed up when the variables go out of scope.

5. A **program counter** that contains the contents of processor's registers

Process State

- As a process executes, it changes *state*
  - new: The process is being created.
  - ready: The process is waiting to be assigned to a process.
  - running: Instructions are being executed.
  - waiting: The process is waiting for some event to occur.
  - terminated: The process has finished execution.

Diagram of Process State



Tables

- Four different types of tables maintained by the operating system:

1. Memory Tables - are used to keep track of both main and secondary memory. Some of main memory is reserved for use by the operating system; the remainder is available for use by processes.

2. I/O Tables- are used by the operating systems to manage the I/O devices and channels of the computer system. At any given time, an I/O device may be available or assigned to a particular process.

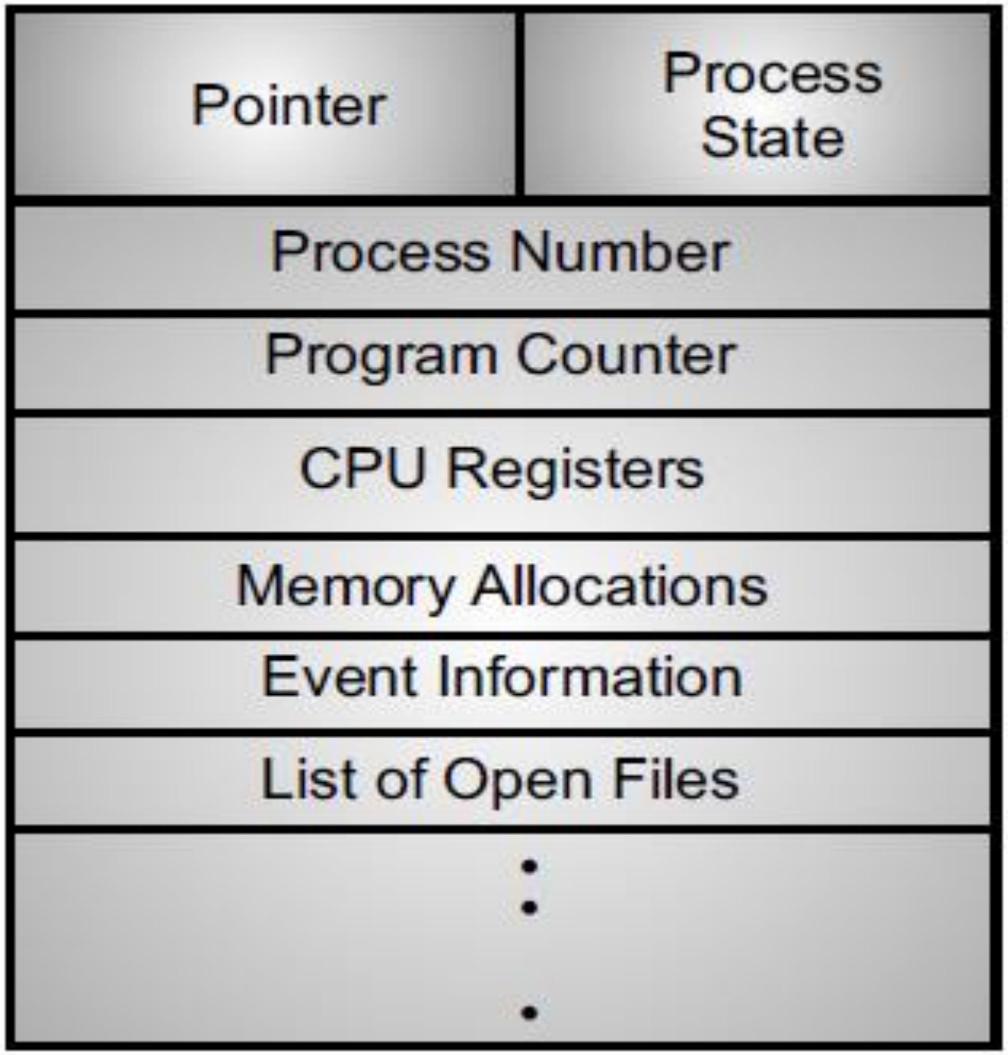
3. File Tables -provide information about the existence of files, their location on secondary memory, their current status another attributes.

4. Process Tables- are used to manage processes a process must include a program or set of programs to be executed.

Process Control Block (PCB)

- Each process is represented in the operating system by a Process Control Block (PCB) also called as Task Control Block (TCB).
- When a process is created, operating system creates a corresponding PCB and released whenever the process terminates.
- A PCB stores descriptive information pertaining to a process, such as its state, program counter, memory management information, information about its scheduling, allocated resources, accounting information, etc. that is required to control and manage a particular process.

Process Control Block (PCB)



Process Control Block (PCB)

1. **Process Number:** Each process is identified by its process number, called Process Identification Number (PID). Every process has a unique process-id through which it is identified. The process-id is provided by the OS. The process id of two processes could not be same because process-id is always unique.

2. **Priority:** Each process is assigned a certain level of priority that corresponds to the relative importance of the event that it services process priority is the preference of the one process over other process for execution. Priority may be given by the user/system manager or it may be given internally by OS. This field stores the priority of a particular process.

3. **Process State:** This information is about the current state of the process. The state may be new, ready, running, and waiting, halted, and so on.

4. **Program Counter:** The counter indicates the address of the next instruction to be executed for this process.

Process Control Block (PCB)

5. CPU Registers: The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward.

6. CPU Scheduling Information: This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

7. Memory Management Information: This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system.

Process Control Block (PCB)

- **8. Accounting Information:** This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

- **9. I/O Status Information:** This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

- **10. File Management:** It includes information about all open files, access rights etc.

- **11. Pointer:** Pointer points to another process control block. Pointer is used for maintaining the scheduling list.

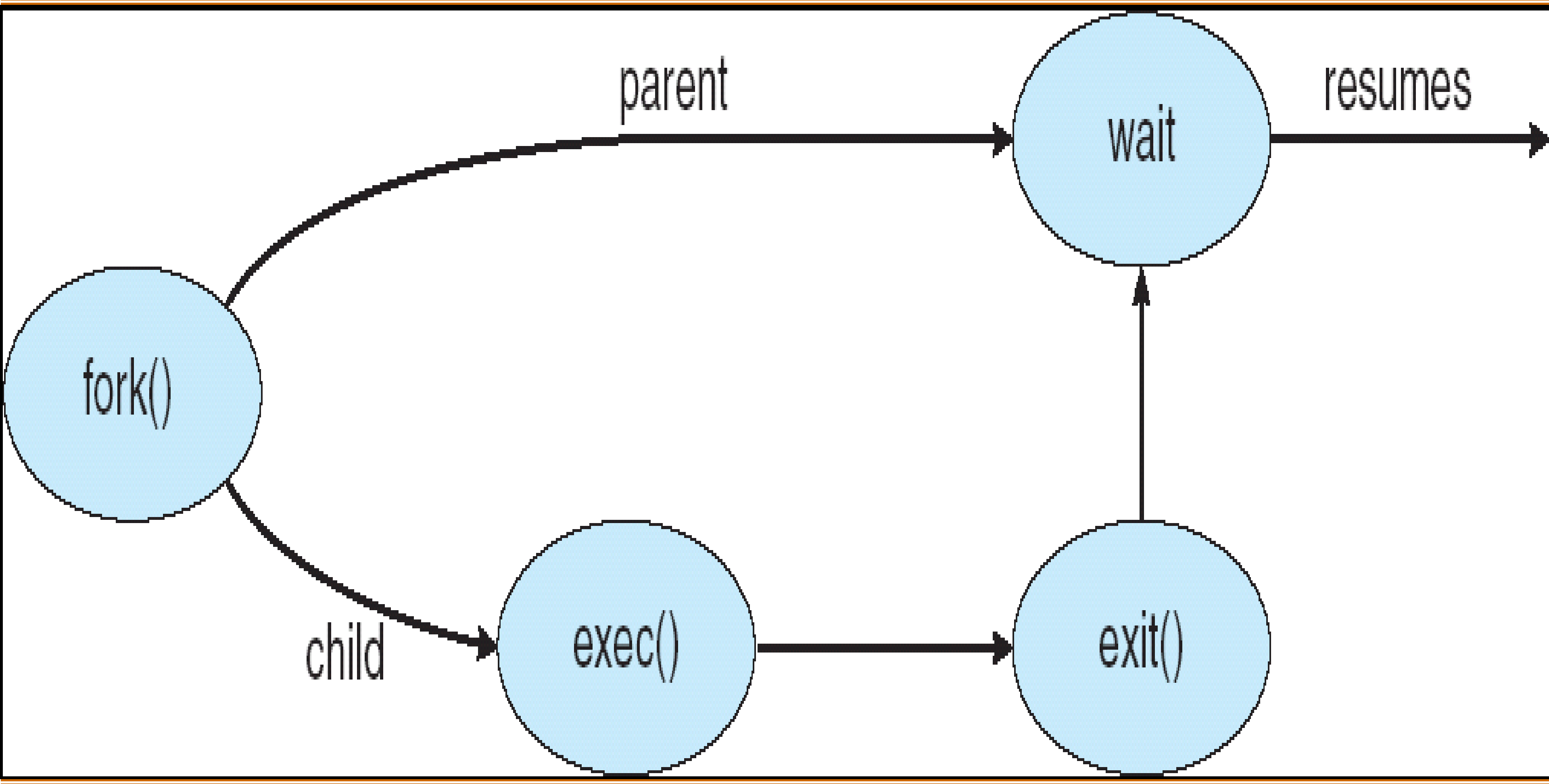
Process Creation

- When a new process is to be added to those currently being managed, the operating system builds the data structures that are used to manage the process and allocates address space in main memory to the process. This is the creation of a new process.
- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Resource sharing
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently
  - Parent waits until children terminate

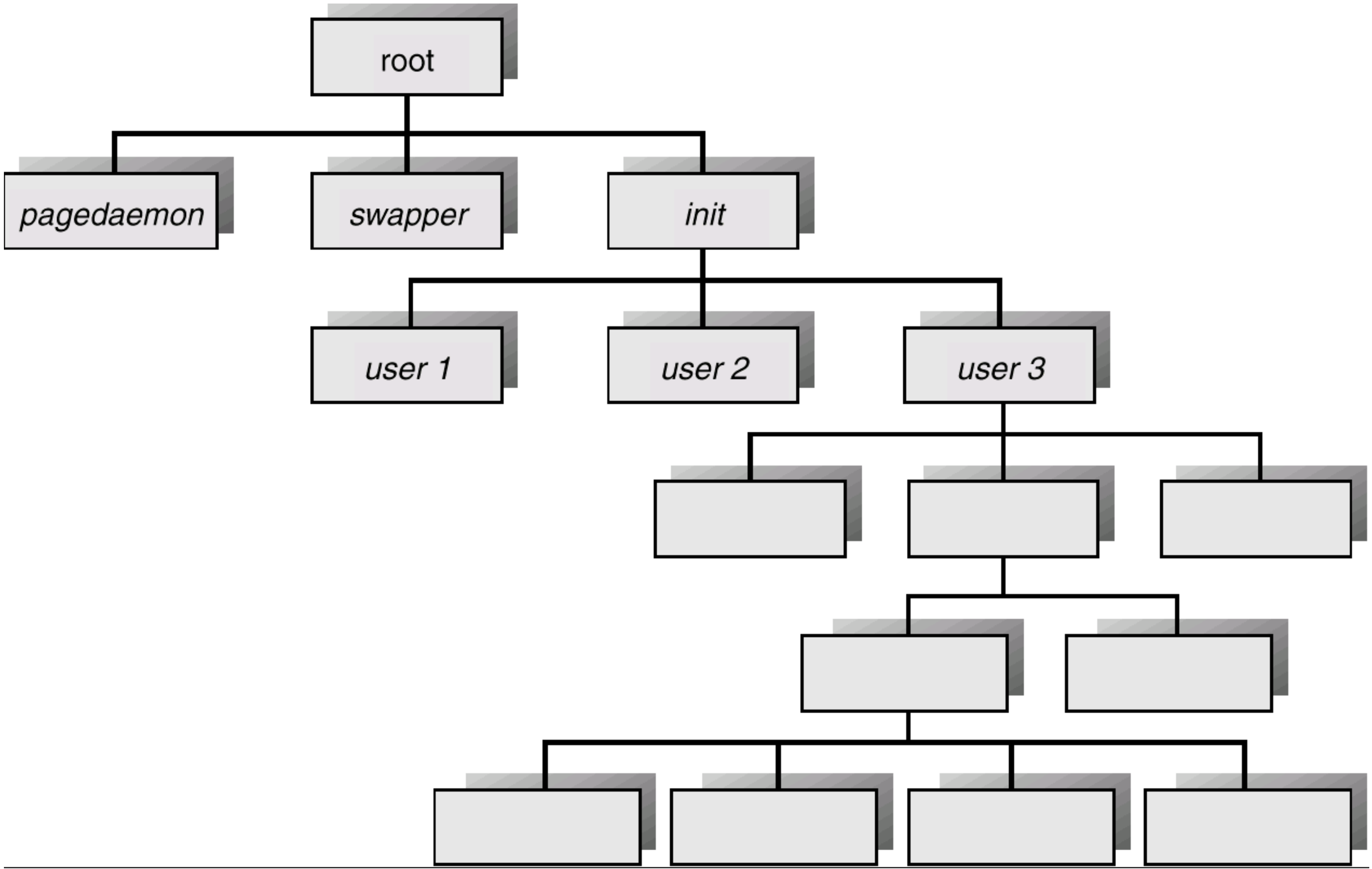
Process Creation (Cont.)

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **fork** system call creates new process
  - **exec** system call used after a **fork** to replace the process' memory space with a new program
- System call CreateProcess() in Windows and fork() in Unix which tells the operating system to create a new process.

Process Creation



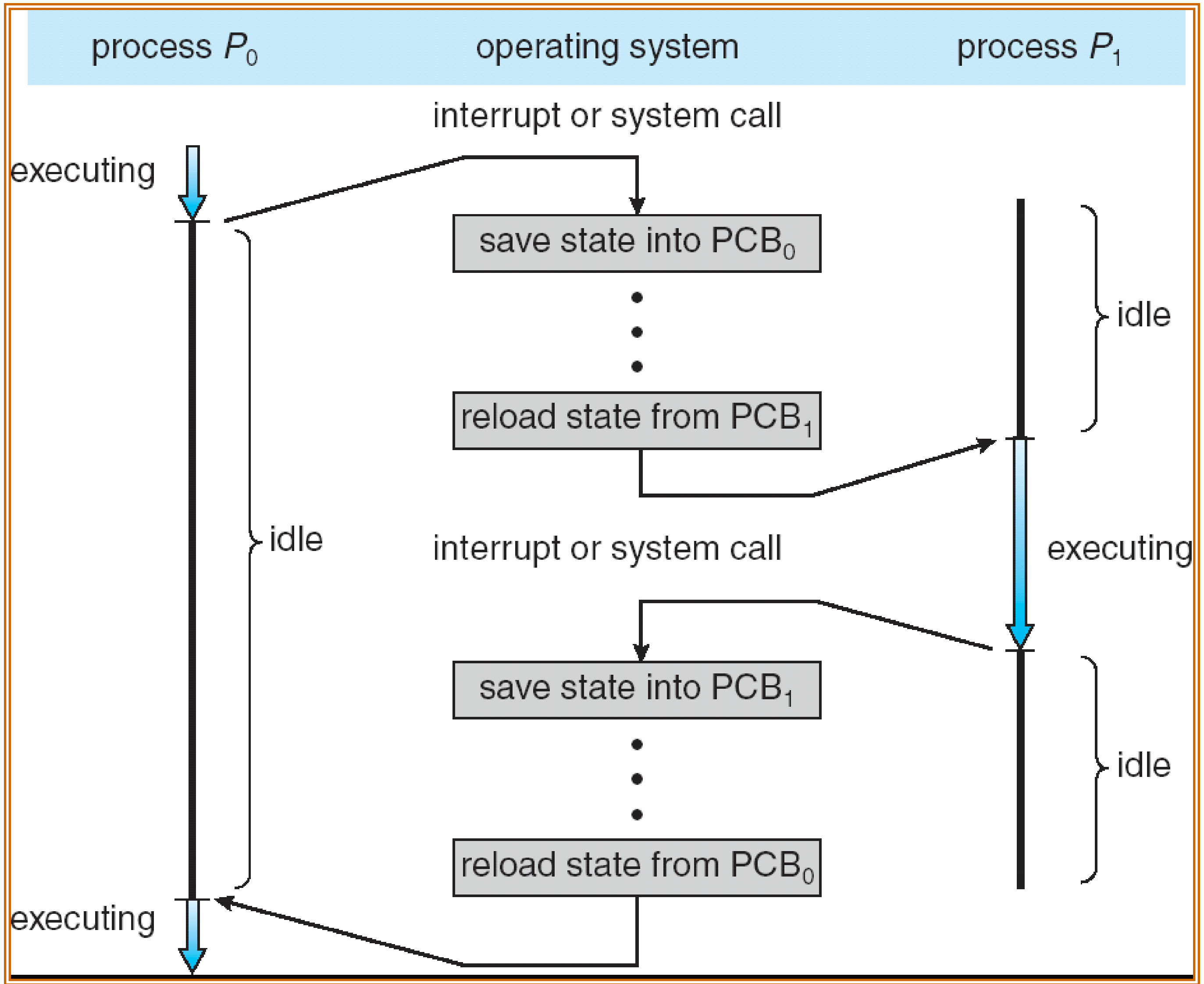
A Tree of Processes On A Typical UNIX System



Process Termination

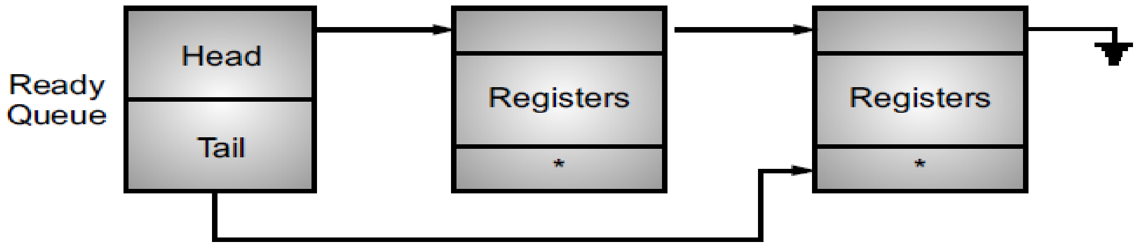
- Depending upon the condition, a process may be terminated either normally or forcibly by some another process.
- Normal termination occurs when the process completes its task (operation) and invokes an appropriate system call ExitProcess( ) in Windows and exit( ) in Unix to tell the operating system that it is finished.
- A process may cause abnormal termination of some another process. For this, the process invokes an appropriate system call TerminateProcess( ) in Windows and kill( ) in Unix that tells the operating system to kill some other process.

CPU Switch From Process to Process

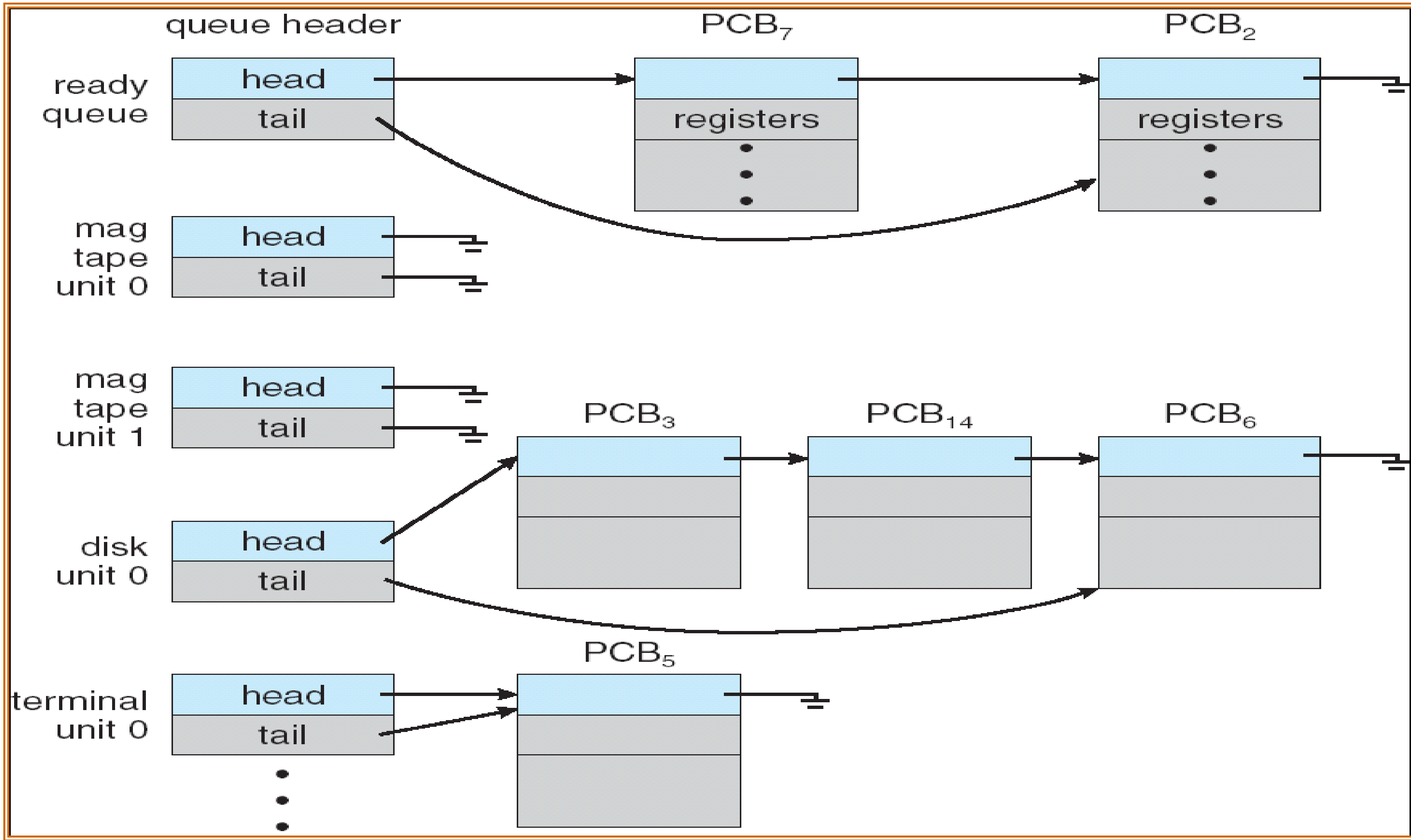


Scheduling Queue

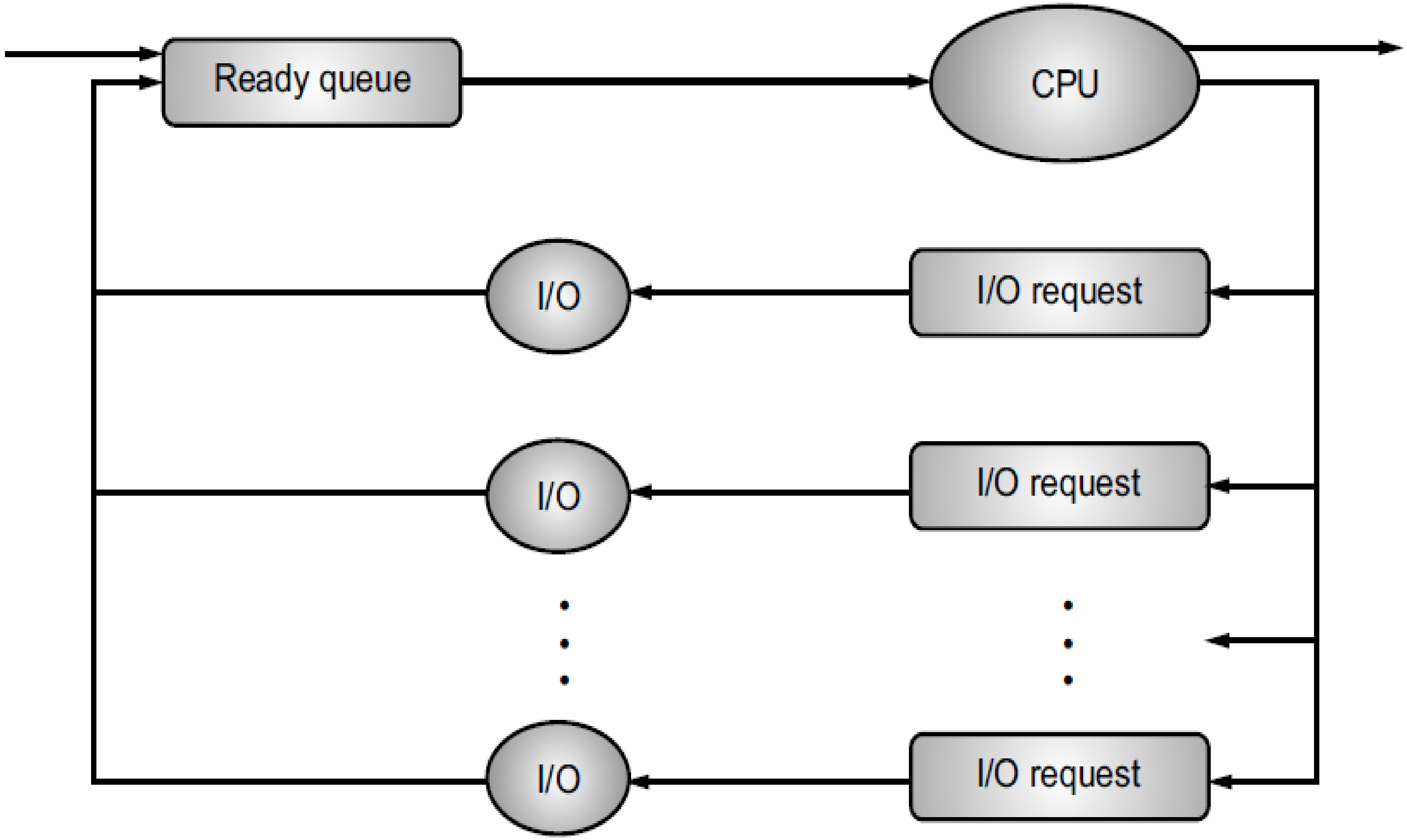
- For a uniprocessor system, there will never be more than one running process. If there are more than one processes, the rest will have to wait until the CPU is free and can be rescheduled.
- The processes, which are ready and waiting to execute, are kept on a list called the **ready queue.**
- The list is generally a linked list. A ready queue header will contain pointers to the first and last PCB's in the list. Each PCB has a pointer field which points to the next process in the ready queue.
- There are also other queues in the system.
- **Job queue** – set of all processes in 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 Device Queues



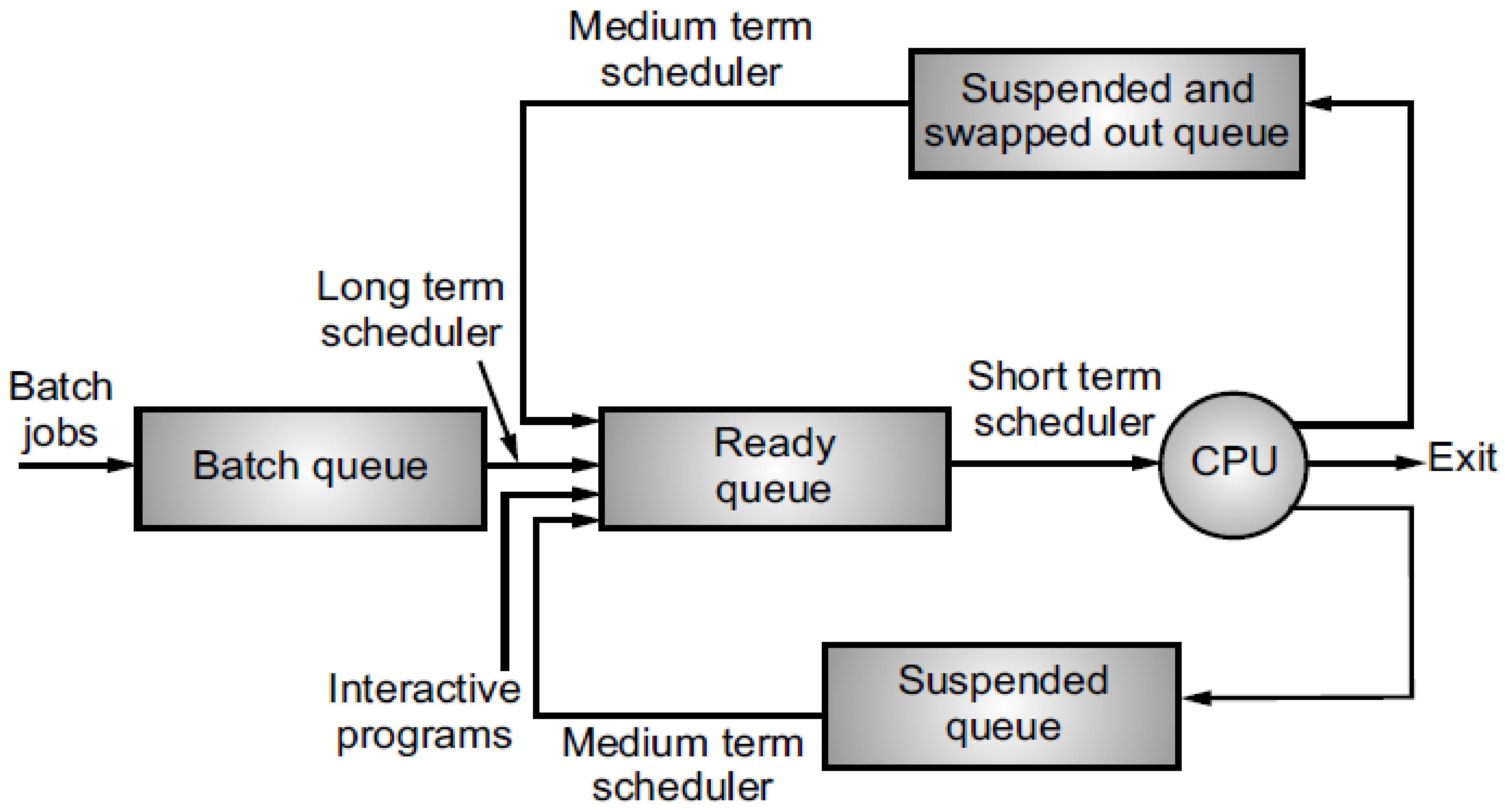
Queuing diagram representation of CPU Scheduling



Schedulers

- Schedulers are special system software's which handles process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run.
- Schedulers are of three types namely, Long Term Scheduler, Short Term Scheduler and Medium Term Scheduler.
- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue. Long term scheduler determines which programs are admitted to the system for processing. Job scheduler selects processes from the queue and loads them into memory for execution.
- **Medium Term Scheduler:** On some systems, the long term scheduler may be absent or minimal.
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU. The short term scheduler selects from memories, which are ready to execute and allocates the CPU to one of them. It is also called as CPU scheduler

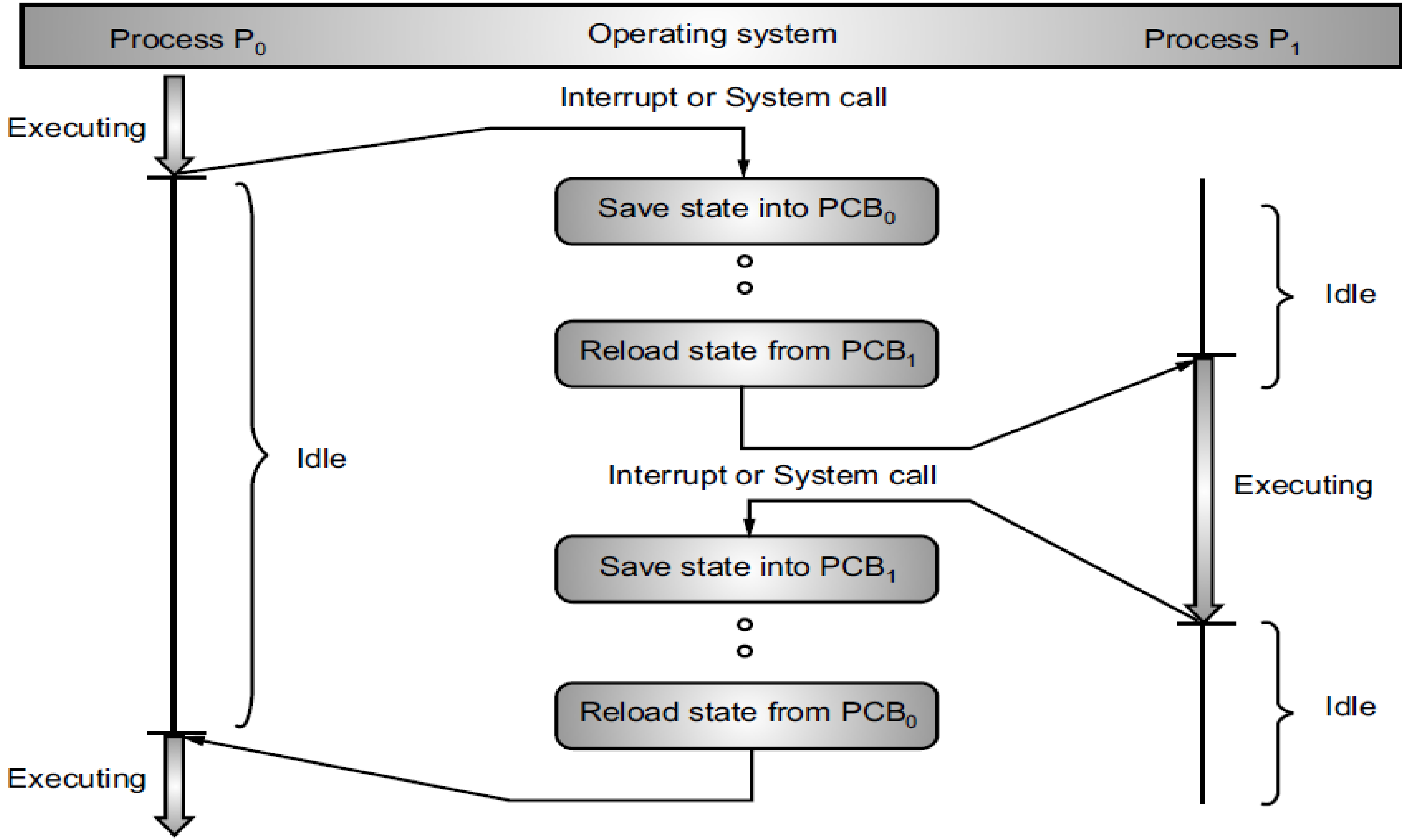
Working of Schedulers



Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process. This task is known as a context switch.
- CPU switching from one process to another process is called a context switch.
- Context switch times are highly dependent on hardware support. Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied and the existence of special instructions.
- Context switch times are highly dependent on hardware support.

Context Switch



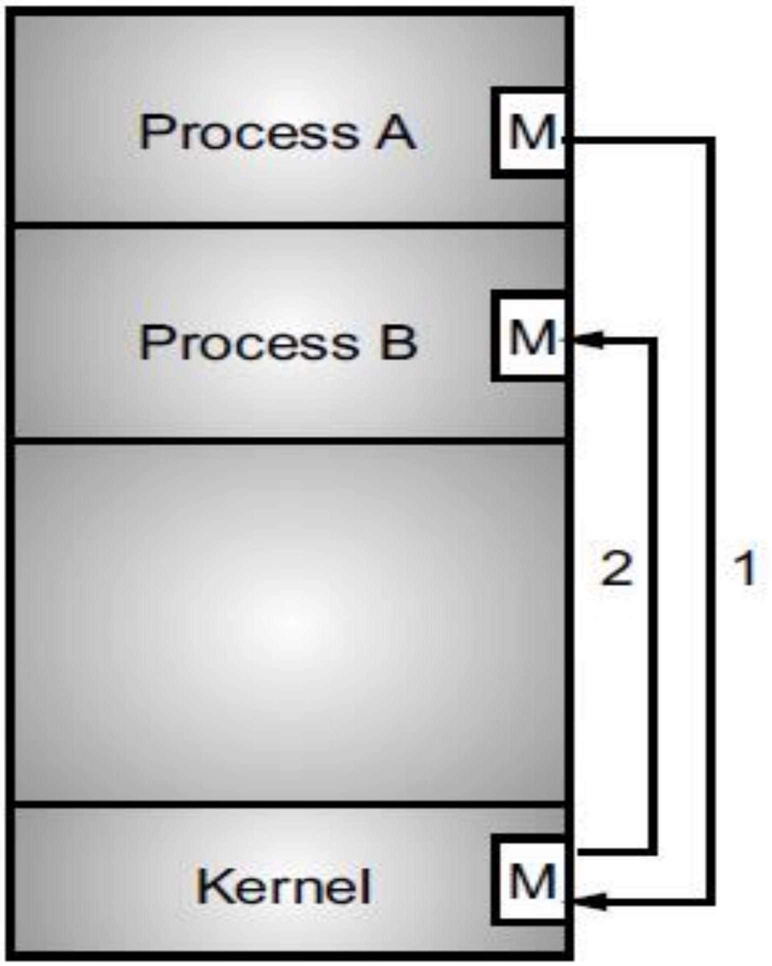
Interprocess Communication (IPC)

- Interprocess communication (IPC) is a set of programming interfaces that allow a programmer to coordinate activities among different program processes that can run concurrently in an operating system.
- The Interprocess Communication (IPC) is a set of techniques for the exchange of data among multiple processes.
- IPC is required in all multiprocessing systems, but it is not generally supported by single-process operating systems such as DOS, OS/2 and MS-Windows etc.
- IPC is particularly useful in a distributed environment where the communicating processes may reside on different computers connected with a network. For example, chat program used on the World Wide Web. IPC is best provided by a message passing system.

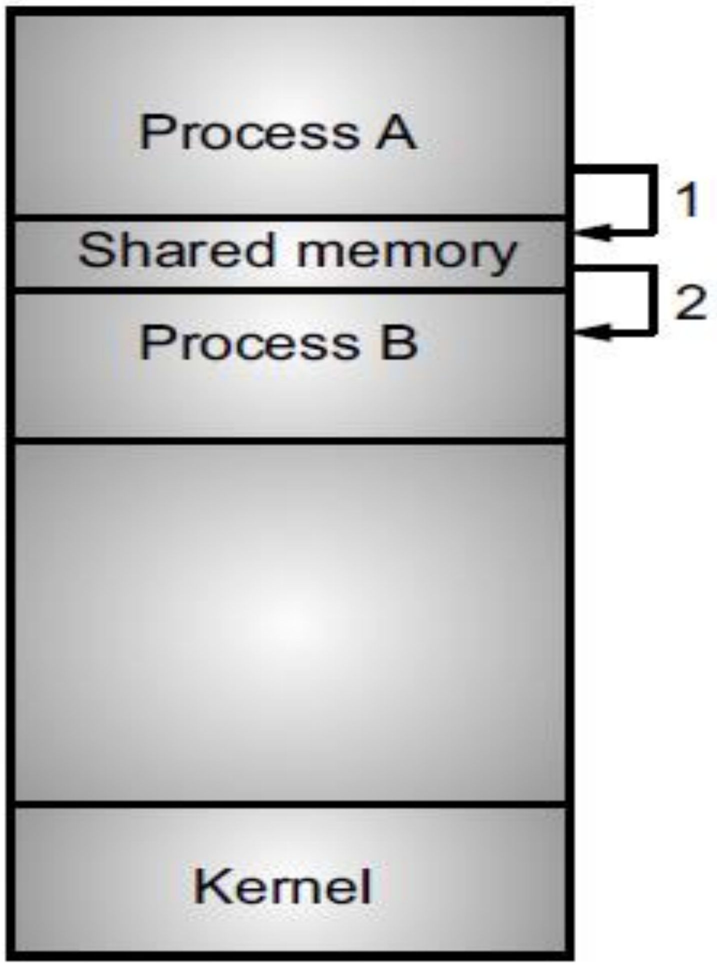
Models of inter Inter-process Communication (IPC)

- Two fundamental models allows inter-process communication as explained below:
- **1. Shared Memory Model:** Two processes exchange data or information through sharing region. They can read and write data from and to this region.
- **2. Message Passing Model:** In message passing model the data or information is exchanged in the form of messages.

Communications Models

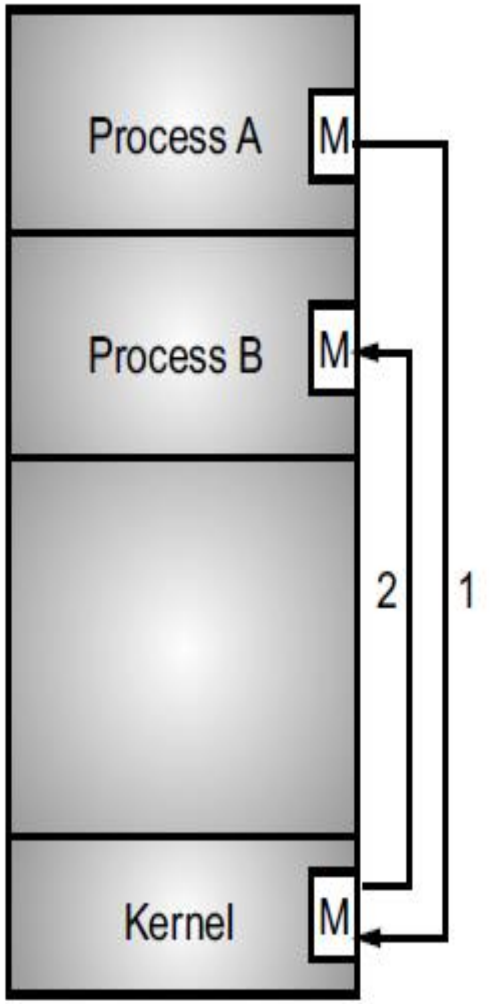


(a) Message Passing



(b) Shared memory

Message Passing

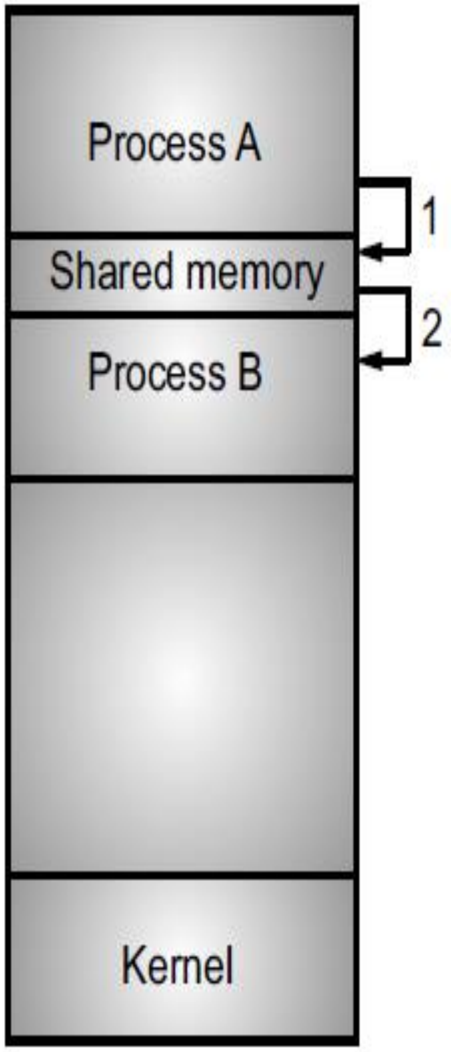


(a) Message Passing

Message Passing:

In this model, communication takes place by exchanging messages between cooperating processes. It allows processes to communicate and synchronize their action without sharing the same address space. It is particularly useful in a distributed environment when communication process may reside on a different computer connected by a network. Communication requires sending and receiving messages through the kernel. The processes that want to communicate with each other must have a communication link between them. Between each pair of processes exactly one communication link.

Shared Memory



(b) Shared memory

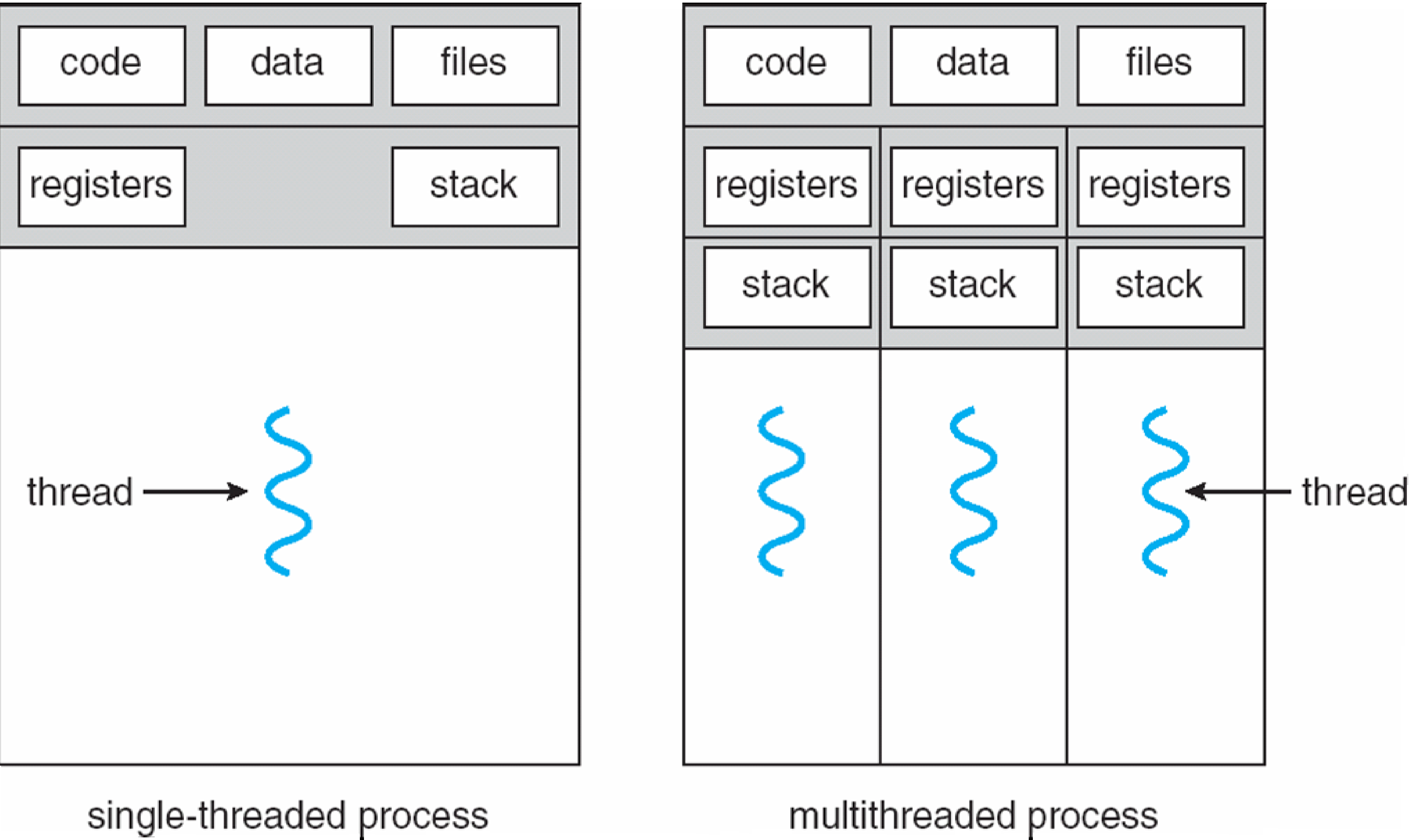
Shared memory:

In this a region of the memory residing in an address space of a process creating a shared memory segment can be accessed by all processes who want to communicate with other processes. All the processes using the shared memory segment should attach to the address space of the shared memory. All the processes can exchange information by reading and/or writing data in shared memory segment. The form of data and location are determined by these processes who want to communicate with each other. These processes are not under the control of the operating system. The processes are also responsible for ensuring that they are not writing to the same location simultaneously. After establishing shared memory segment, all accesses to the shared memory segment are treated as routine memory access and without assistance of kernel.

Thread

- Thread Is a light weight process (LWP), is a basic unit of CPU utilization , are fundamental components of modern operating systems that significantly enhance application performance and responsiveness by enabling concurrent execution within a single program
- It comprises thread ID ,program counter,register set
- It shares code section and data section belonging to same program

Single and Multithreaded Processes



Benefits

- Responsiveness
- Resource Sharing
- Economy
- Utilization of MP architecture /Scalability

Benefits

- Responsiveness. Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user. This quality is especially useful in designing user interfaces. For instance, consider what happens when a user clicks a button that results in the performance of a time-consuming operation. A single-threaded application would be unresponsive to the user until the operation had completed. In contrast, if the time-consuming operation is performed in a separate thread, the application remains responsive to the user.
- 2. Resourcesharing.

Processescanonlyshareresourcesthroughtechniques such as shared memory and message passing. Such techniques must be explicitly arranged by the programmer. However, threads share the memoryandtheresourcesoftheprocesstowhichtheybelongbydefault. The benefit of sharing code and data is that it allows an application to have several different threads of activity within the same address space.

- 3. Economy.Allocatingmemoryandresourcesforprocesscreationiscostly. Because threads share the resources of the process to which they belong, it is more economical to create and context-switch threads. Empirically gauging the difference in overhead can be difficult, but in general it is significantly more time consuming to create and manage processes than threads. In Solaris, for example, creating a process is about thirty times slower than is creating a thread, and context switching is about fivetimes slower.
- 4. Scalability. The benefits of multithreading can be even greater in a multiprocessor architecture, where threads may be running in parallel on different processing cores. A single-threaded process can run on only one processor, regardless how many are available.

User Threads

- Thread management done by user-level threads library
- Three primary thread libraries:
- POSIX Pthreads
- Win32 threads
- Java threads

Kernel Threads

- Supported by the Kernel

Examples

- Windows XP/2000
- Solaris
- Linux
- Tru64 UNIX
- Mac OS X

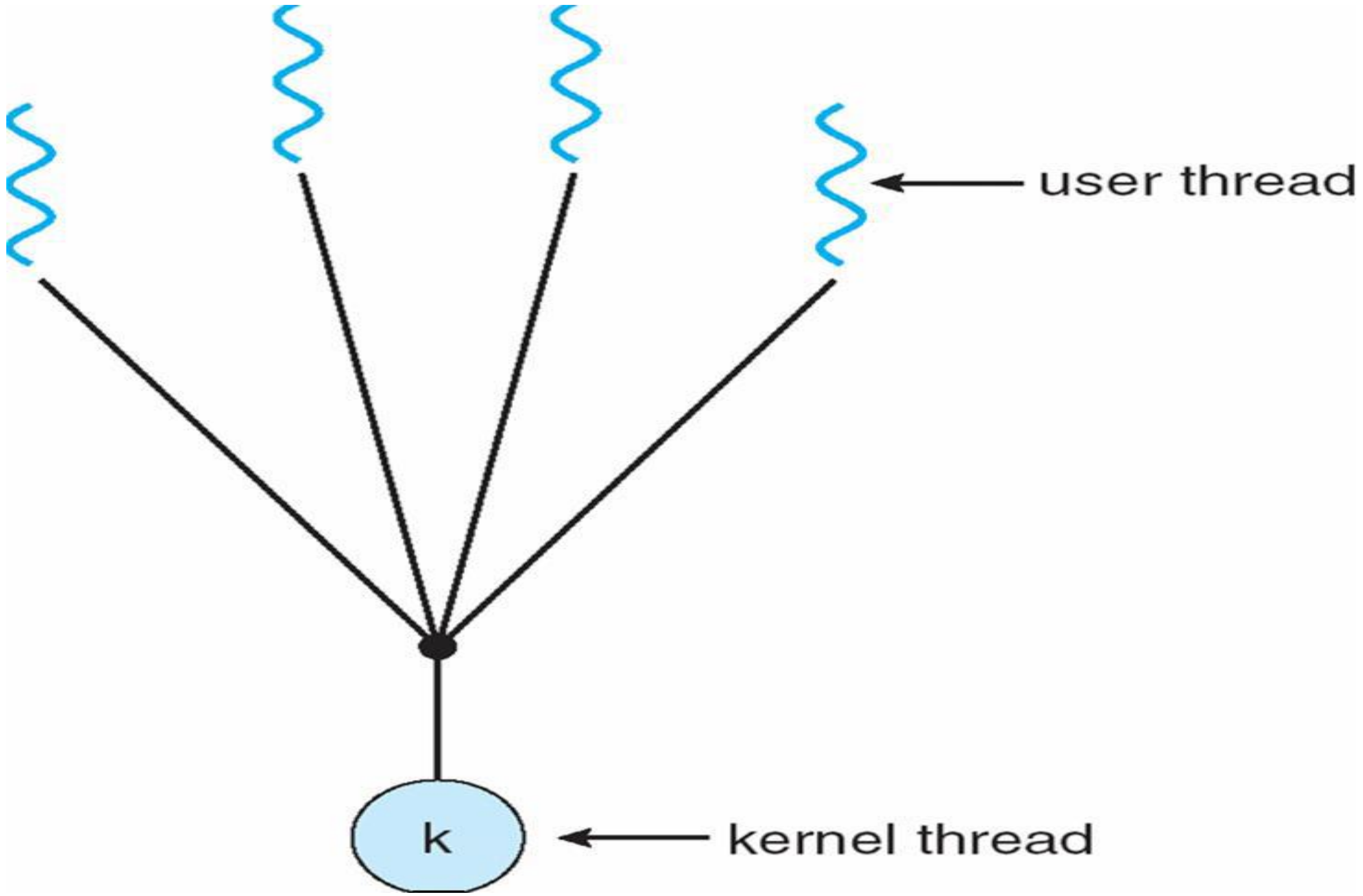
Multithreading Models

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

Many-to-One

- Many user-level threads mapped to single kernel thread
- Examples:
- Solaris Green Threads
- GNU Portable Threads

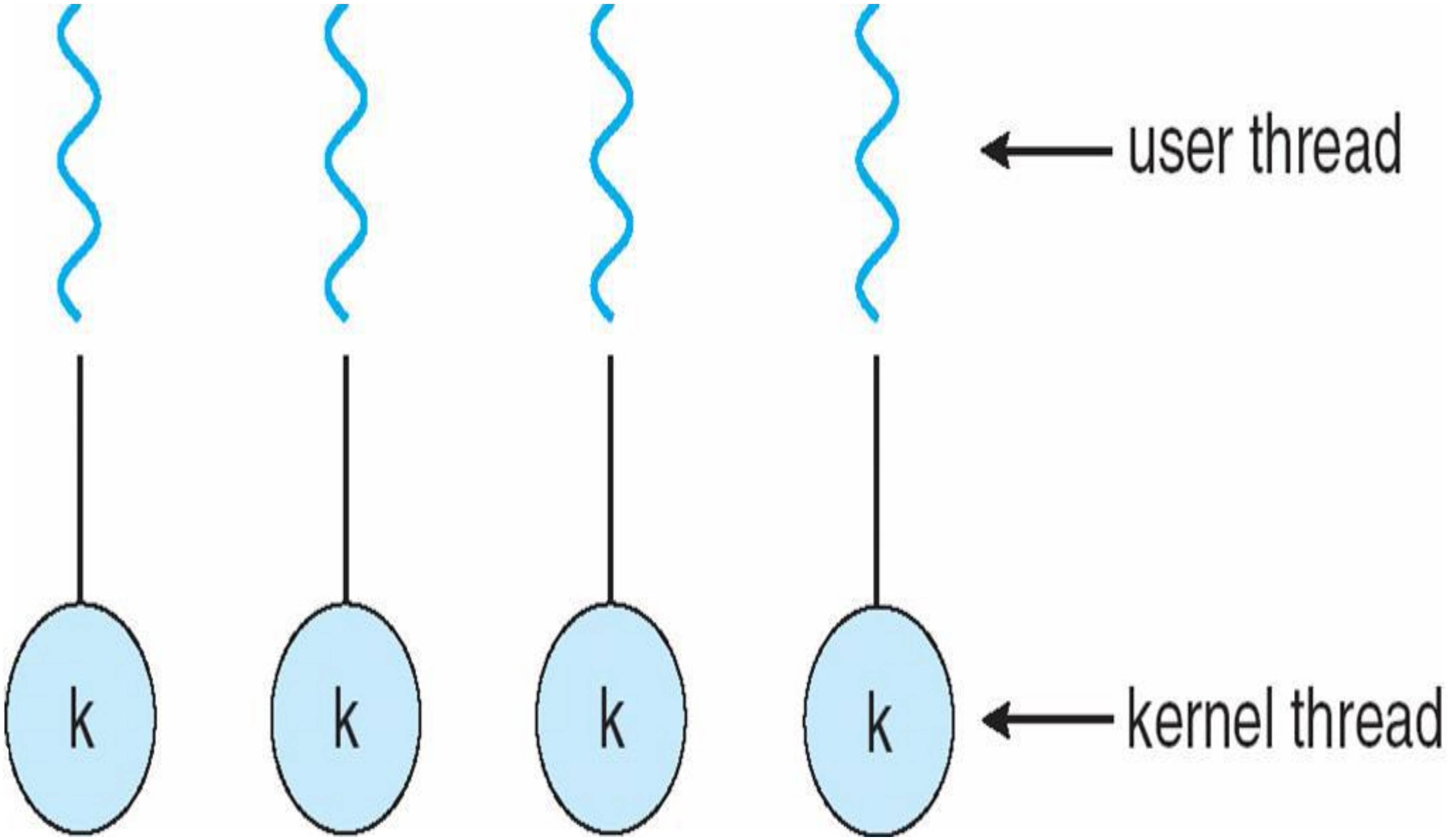
Many-to-One Model



One-to-One

- Each user-level thread maps to kernel thread
- Examples
- Windows NT/XP/2000
- Linux
- Solaris 9 and later

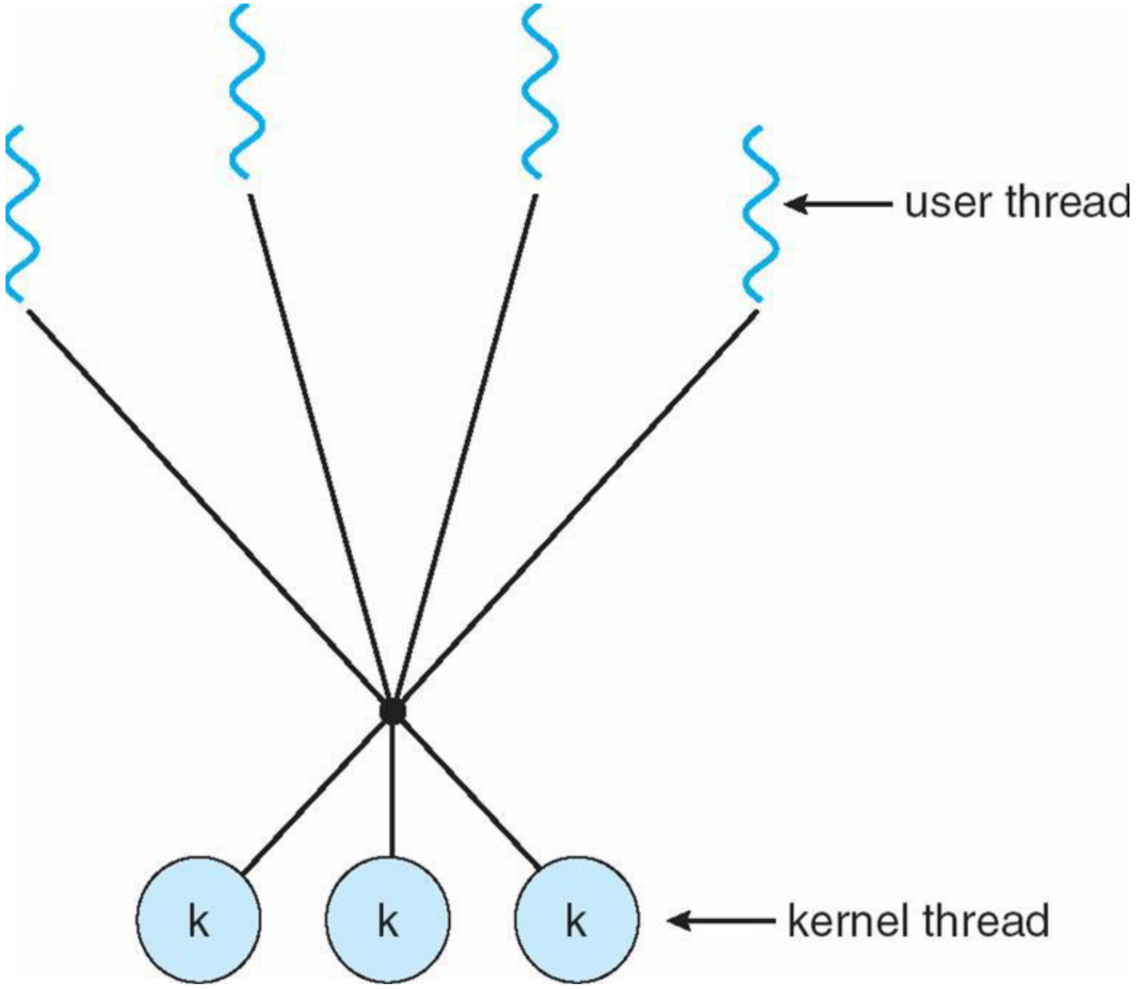
One-to-one Model



Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Solaris 2,HP-UX,and Tru 64 UNIX

Many-to-Many Model



common Linux process management commands:

Top

- The top command provides a real-time, dynamic view of running processes. It displays system summary information (CPU, memory, swap usage) and a list of processes sorted by CPU usage by default.
- Ps
- The ps (process status) command displays a snapshot of the current processes

Process command

- Kill

The kill command sends a signal to a process, typically to terminate it. It requires the Process ID (PID) of the target process. The default signal is SIGTERM (15), which requests a graceful shutdown. SIGKILL (9) forces termination.

- Wait
- The wait command in shell scripting suspends execution until a specified background process or all background processes have completed.

Process command

- sleep
- The sleep command pauses execution for a specified duration in seconds, minutes, hours, or days.
- exit
- The exit command terminates the current shell script or interactive shell session. It can optionally return an exit status code.
- nice
- The nice command runs a command with a modified scheduling priority (niceness value). A lower niceness value (e.g., -20) means higher priority, while a higher value (e.g., 19) means lower priority.

Thank you…..

You can mail your Queries to :

a2kousarj@gmail.com