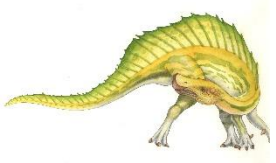
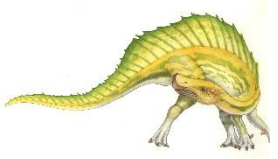
Ch1: Introduction to OS (12M)



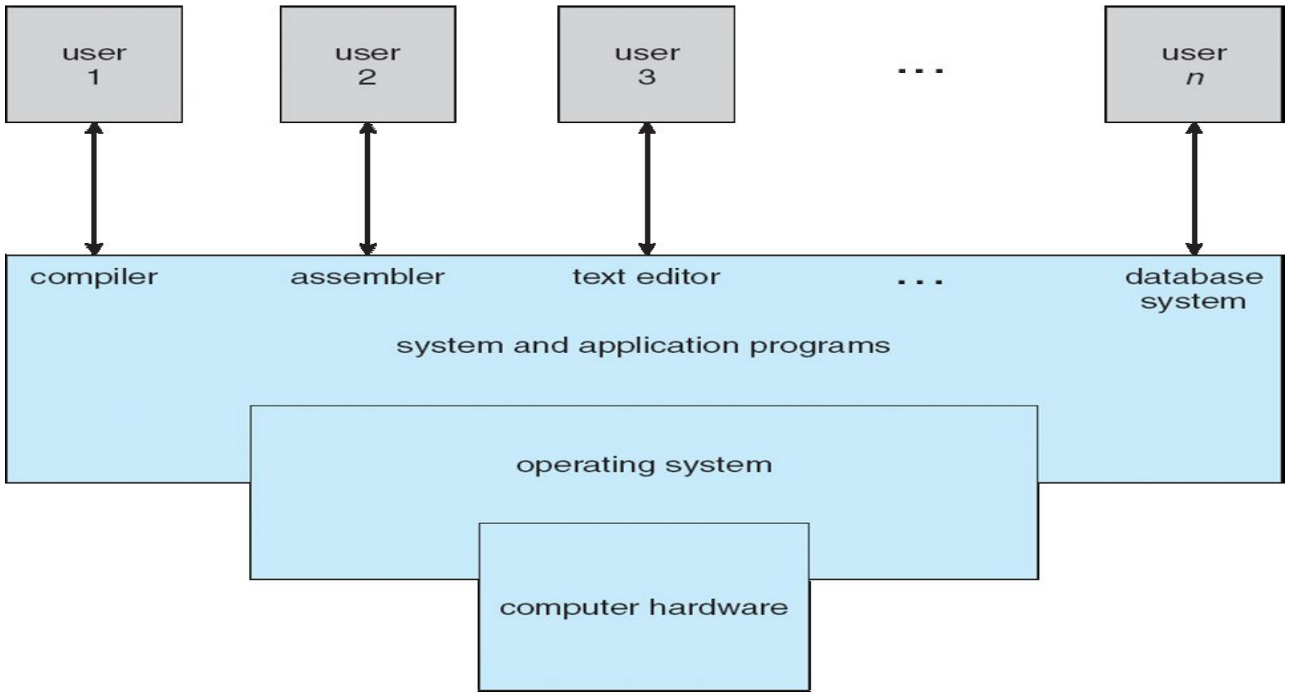
What is an Operating System?

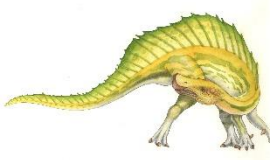
- It is a program/software that that's controls the execution of applications programs such as compilers, database systems, games etc
- It acts as an intermediary between a user of a computer and the computer hardware.
- In other words it is an interface between applications and hardware.
- OS manages the computer hardware resources
- The operating system provides the means for the proper use of these resources in the operation of the computer system.
- 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 manner



Operating systems as a User interface

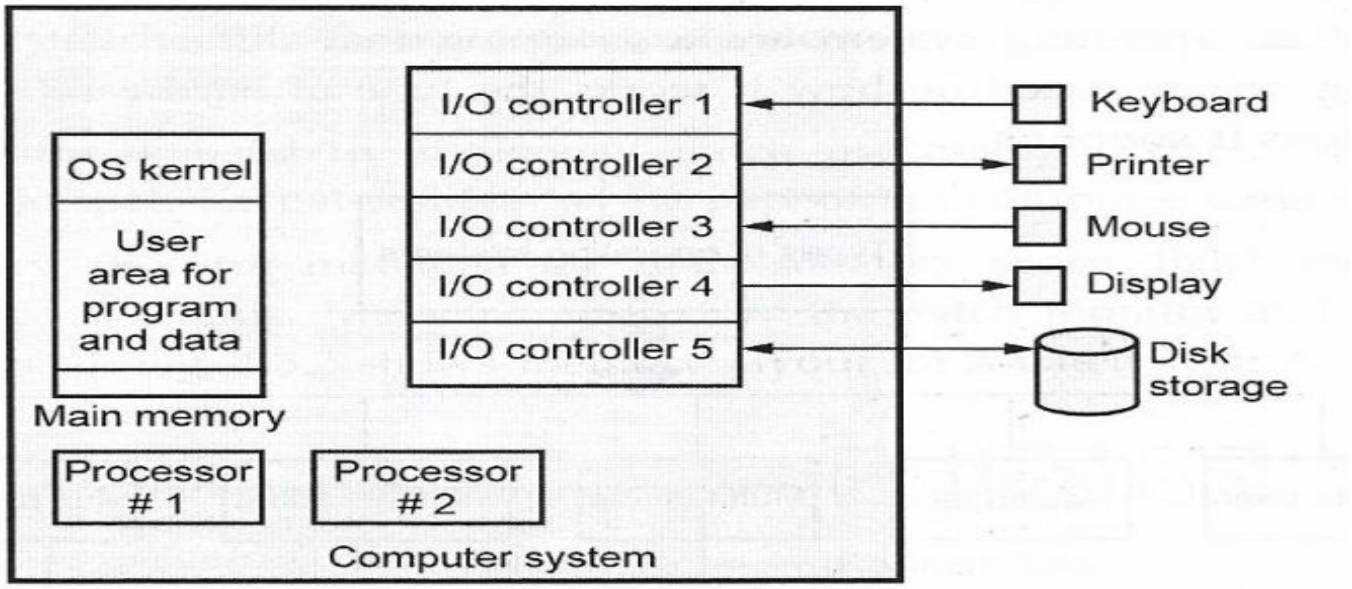
- Operating systems interact directly with the hardware to provide an interface to other system software.
- System software and hard ware exist to support the creation and effective use application software.

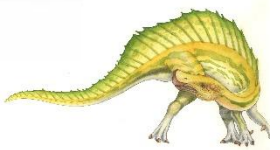




Operating System as Resource Manager

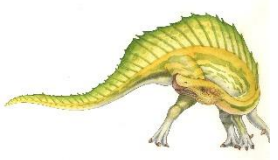
- A computer system has a set of resources. These resource provides various functions to the user like data movement, storing of data and program, operation on data which are control by an operating system.
- The OS is responsible for controlling the use of a computer's resources, such as I/O, main and secondary memory, and processor execution time.
- Below Fig. shows OS as a resource manager
-





The operating system is responsible for managing all the resources. A portion of the OS is in main memory. This portion of the OS is called kernel.

- User program and data are also stored in remaining parts of the memory.
- I/O device is controlled by OS and it decides when an I/O device can be used by program in execution. Processor is one type of resource and OS control the execution of user program on the processor.
- Allocation of main memory is controlled by operating system with the help of memory management hardware.
- Resource management includes sharing resources in different ways. Time and space are the two concept for resource sharing.



The services provided by OS

The OS typically provides services in the following areas:

- **Program development**: The OS provides a variety of facilities and services, such as editors and debuggers, to assist the programmer in creating programs.
- **Program execution**: A number of steps need to be performed to execute a program. Instructions and data must be loaded into main memory, I/O devices and files must be initialized, and other resources must be prepared.
- **Access to I/O devices:** Each I/O device requires its own peculiar set of instructions The OS provides a uniform interface that hides these details so programmers can access such devices.
- **Controlled access to files:** For file access, the OS must reproduce a detailed understanding of the nature of the I/O device (disk drive, tape drive) and the structure of the data contained in the files on the storage medium.
- **Error detection and response**, In each case, the OS must provide a response that clears the errors such as **:** internal and external hardware errors and various software errors condition with the least impact on running applications.

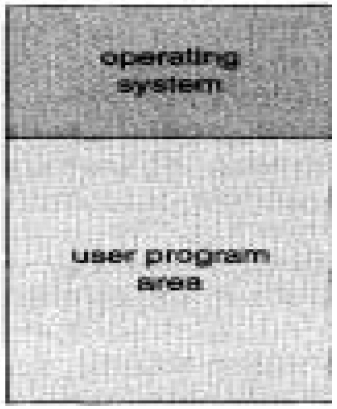
Mainframe systems

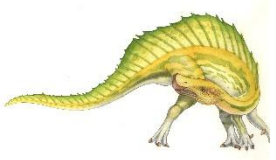
- Batch
- Multi Programmed
- Multitasking
- Time sharing
- Desktop

Batch system

- Batch system process a collection of jobs, called batch. Batch is a sequence of user jobs/process.
- Job is predefined sequence of commands, programs and data that are combined into a single unit.
- Each job in the batch is independent of other jobs in the batch.
- Jobs with the similar needs are batched together to speed up processing.
- The primary function of the batch system is to service the jobs in a batch one after another without the need of user interaction.
- Jobs are processed in the order of submission i.e. first come first serve fashion.

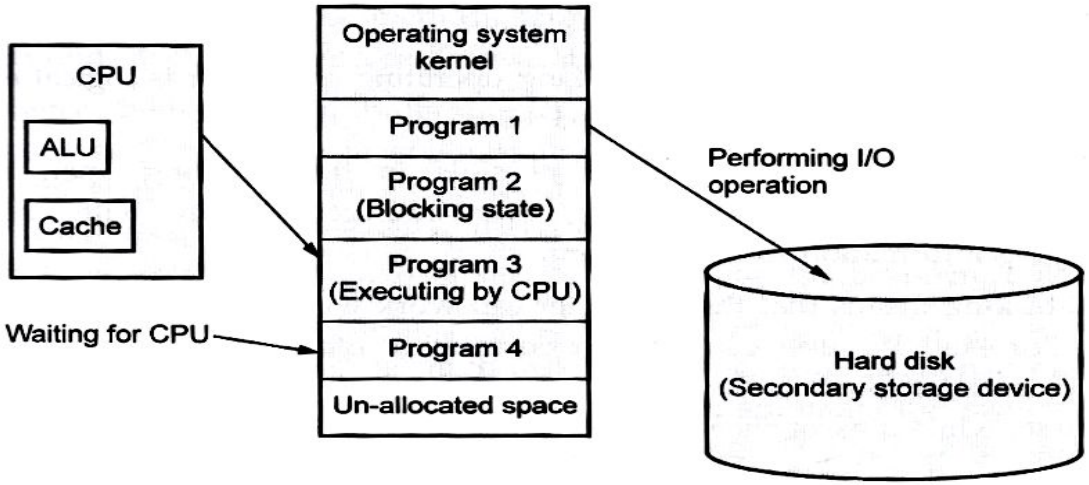
- For example card readers, pendrives are the input device in batch systems



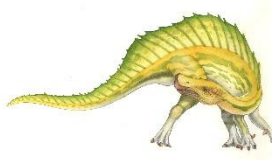


Multi Programmed

- CPU remains idle in the batch system. At any time either CPU or I/O device also remains idle . To keep CPU busy ,more than one program must be loaded into memory for execution.
- When two or more programs are ready to execute and are in the memory at the same time on a single processor then it is referred as multiprogramming operating system.
- Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.
- Resource management is the main aim of multiprogramming OS
- When the operating system keeps number of programs into the memory simultaneously, It selects one program from the memory at a time and executes it
- As long as at least one program needs to execute, the CPU is never idle
- .



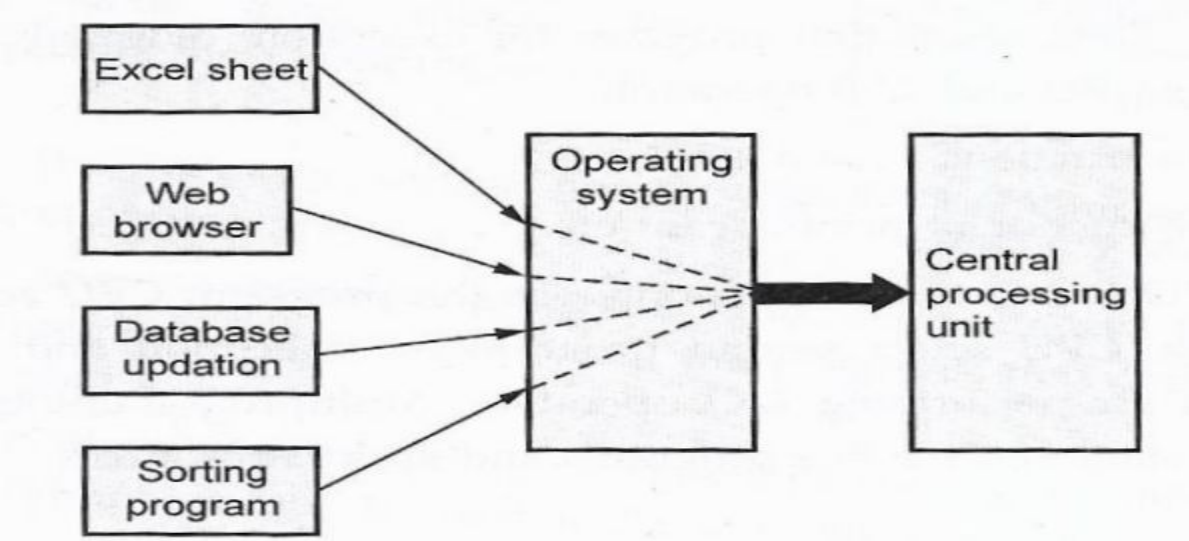
- In multiprogramming operating system, programs are competing for resources.
- Suppose there are four programs for execution. All four programs are loaded into the memory. CPU select first program for execution. Normally programs contain instruction for CPU and I/O operation
- When any I/O instruction is run into the program, CPU select next program for processing. CPU select second program for execution and I/O system select first program for performing I/O operation.
- In this way multiprogramming operating system monitors the state of all active programs and system resources.
- Multiprogramming OS do not provide user interaction with the computer system



Multitasking (time sharing)

- Multi-tasking means that the computer can work with more than one program at a time.
- Time sharing (or multitasking) is a logical extension of multiprogramming OS
  - The CPU executes multiple jobs by switching among them, but the switches occur frequently and the users can interact with each program while it is running.
  - User interaction with program is possible in time sharing OS. During execution of the program, user interacts directly with the program, supplying information to the program.
  - The response time should be short typically within 1 second or so.
  - A time-shared operating system allows many users to share the computer simultaneously
- Many users share the computer system simultaneously in time sharing operating system. Time sharing system uses multiprogramming and CPU scheduling. Each user has at least one separate program in memory.

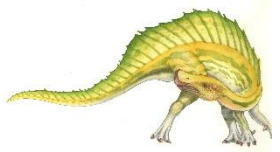
- In In time sharing system, each user is given a time slice for executing his/her job in round robin fashion, job continues until the time slice ends.
- Time sharing system is more complex than multiprogramming operating system.
- Major problem with time sharing system is protection and security of data.



- User is working with the information from one database on the screen analyzing data while the computer is sorting information from another database ,while excel sheet is performing calculations on a separate worksheet.

Multiprocessor System

- Multiprocessor system means more than one processor in close communication. All the processor share common bus, clock, memory and peripheral devices.
- By increasing the number of processors, we expect to get more work done in less time
- Multiprocessor system is also called parallel system or tightly coupled systems. They handle system calls, do memory management, provide a file system, and manage I/O devices.
- The most common multiprocessor systems use **symmetric multiprocessing** (**SMP**), in which each peer CPU processor performs all tasks, including operating-system functions and user processes.
- **Features of Multiprocessor Systems**
- 1. The processor should support efficient context switching operation
- 2. It supports large physical address space and larger virtual address space.
- 3. If one processor fails, then other processors should save the interrupted process state so that execution of the process can continue



- Multiprocessor systems are of two types: symmetric multiprocessing and asymmetric multiprocessing.

1.4.4 Difference between Symmetric and Asymmetric Multiprocessor

Symmetric Multiprocessor	Asymmetric Multiprocessor
Symmetric multiprocessing treats all processors are equals, an I/O can be processed on any CPU.	Asymmetric multiprocessing has one master CPU and the remainder CPUs are slaves.
Symmetric multiprocessing is easier to implement in operating systems	Asymmetric multiprocessing is difficult to implement.
Single OS manages all processor cores simultaneously.	Separate OS, or separate copy of same OS manage each core
Master-slave concept is not used.	It uses concept of master-slave concept.
The processors communicate with each other through shared memory.	Shared memory is not used for communication.

Process description & control

Process concept

- Process means a program in execution; process execution must progress in sequential order. It also called task
- Process is an active entity that requires a set of resources including a processor, program counter, registers to perform its function
- Each process has its own address space. Address space is divided into two regions :
- 1. Text region 2. Data region 3. Stack region
- **Text region**: It stores the code that that the processor executes.
- **Data region** : It stores variables and dynamically allocated memory that the process uses during execution.
- **Stack region** : It stores instructions and local variables for active procedure calls

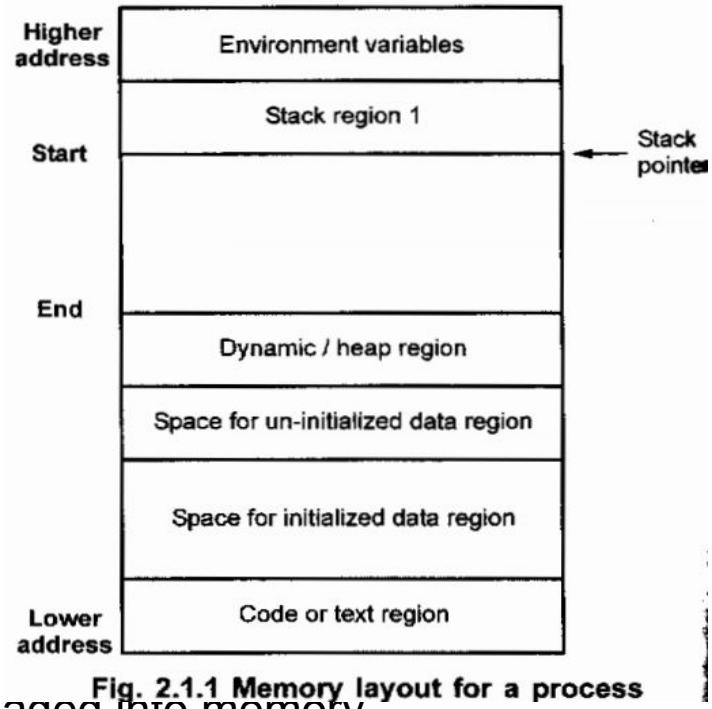


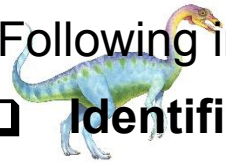
Fig. 2.1.1 Memory layout for a process

- Program becomes process when executable file loaded into memory.
- A process can run to completion only when all requested hardware and software resources have been allocated to the process.

- After booting operating system, it creates foreground processes and background processes.
- Foreground processes interact with user and background processes is used by the system. Background processes are related e-mail, web pages, news and printing.
- Heap is used for dynamic memory allocation and stack contains program counter.
- An initialized data region contains the static and global variables that are initialized by the user.
- Un-initialized data region contains all static and global variables that are initialized by kernel to zero.

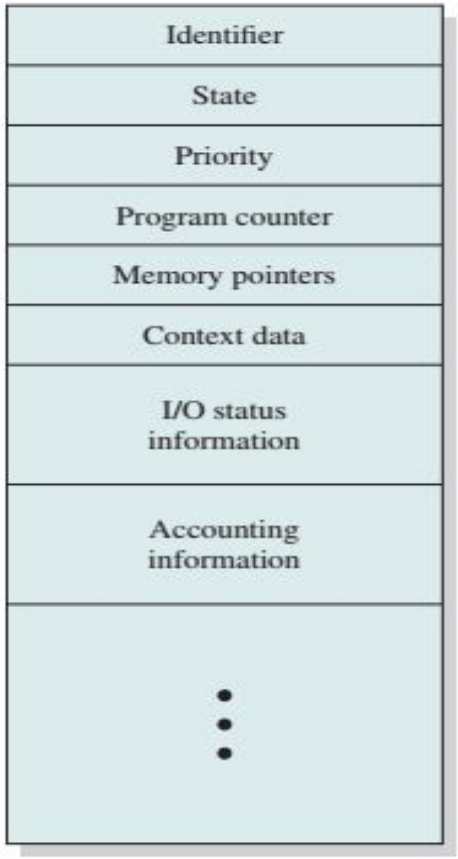
Process control Block

- The process control block is the key tool that enables the OS to support multiple processes and to provide for multiprocessing.
- When OS creates process, it creates this process descriptor. In some operating system, it calls Process Control Block (PCB).
- Process control block will change according to the operating system. PCB is also called **task control block**
- The significant point about the process control block is that it contains sufficient information so it is possible to interrupt a running process and later resume execution as if the interruption had not occurred.
- When a process is created by operating system, it allocates a PCB for it.
- OS initializes PCB and puts PCB on the correct queue.
- When the OS stops running a process, it saves the registers values in the PCB
- When process changes the state, the operating system must update information in the processes process control block.



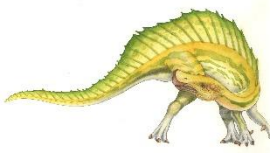
Following information is stored in process control block.

- **Identifier:** A unique identifier associated with this process, to distinguish it from all other processes.
- **State:** If the process is currently executing, it is in the running state.
- **Priority**: Priority level relative to other processes.
- **Program counter**: It indicates the address of the next instruction in the program to be executed.
- **Memory pointers**: Include pointers to the program code and data associated with this process, plus any memory blocks shared with other processes.
- **Context data**: These are data that are present in registers i the process is executing.
- **I/O status information**: It maintains information about the open files, list of
- devices allocated to the process etc•
- **Accounting information**: May include the amount of processor time and clock time used, time limits, account numbers, and so on



Process states

- Each process has an execution state which indicates what process is currently doing. The process descriptor is the basic data structure used to represent specific state for each process.
- As a process executes, it changes state
- The process state is represented by using state diagram.
- A state diagram is composed of a set of states and transitions between states
- State diagram is used by process manager to determine the type of service to provide to the process .
- When process is in new state, the program remains in the secondary storage



- New: The new process is being created by OS
- Ready: The process is waiting to be assigned to a processor
- Running: Instructions are being executed. Operating system allocates all the hardware and software resources to the process for execution.
- Blocked/waiting: The process is waiting for some event to occur such as the completion of an input-output operation.
- Exit/End/terminated: The process has finished execution and releases all resources

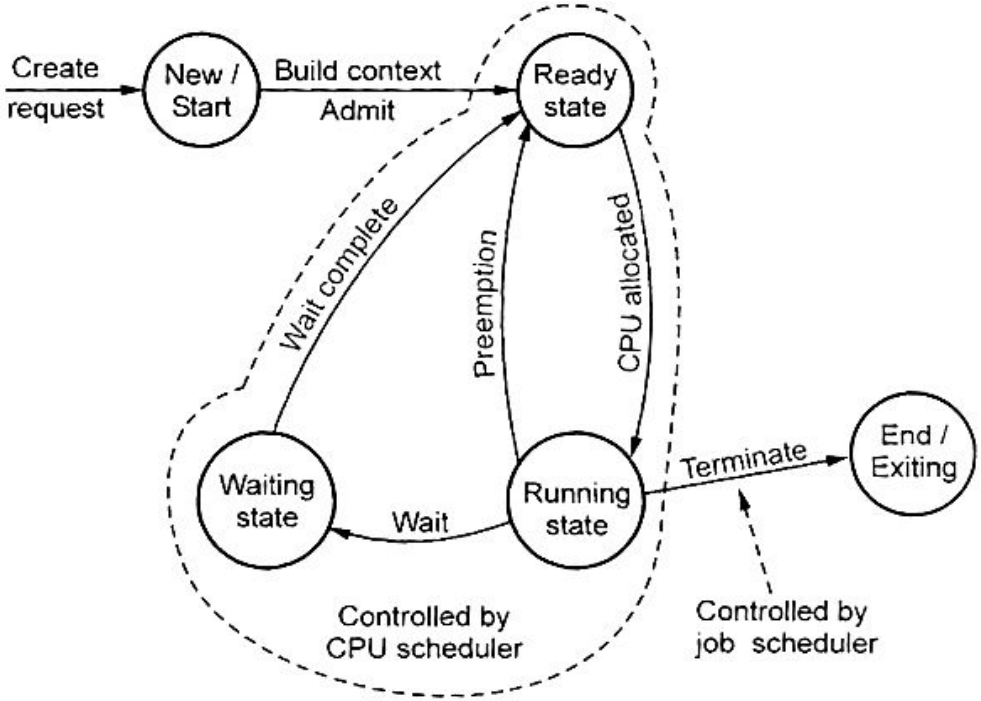


Fig. 2.1.4 Process state diagram

Creation & termination of processes

The following operations are performed on the process:

1. Process creation
2. Process termination

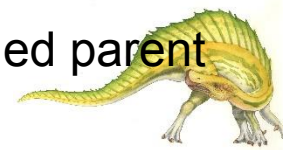
Process creation

- Process creation is a task of creating new processes.
- There are different situations in which a new process is created

1. Starting of new batch job, 2. User request for creating new process

3. To provide new services by OS 4. System call from currently running process.

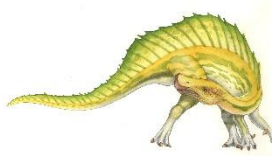
- There are different ways to create new process.
- In a batch environment, a process is created in response to the submission of a job.
- In an interactive environment, a process is created when a new user attempts to log on.
- A new process can be created at the time of initialization of operating system or when system calls such as fork () are initiated by other processes.
- The process, which creates a new process using system calls, is called parent process while the new process that is created is called child process.



- For example, a new process is created every time a user logs on to a computer system, an application program such a MS Word is initiated, or when a document printed.
- When OS creates process, it builds the data structure for managing process and allocates address space in main memory

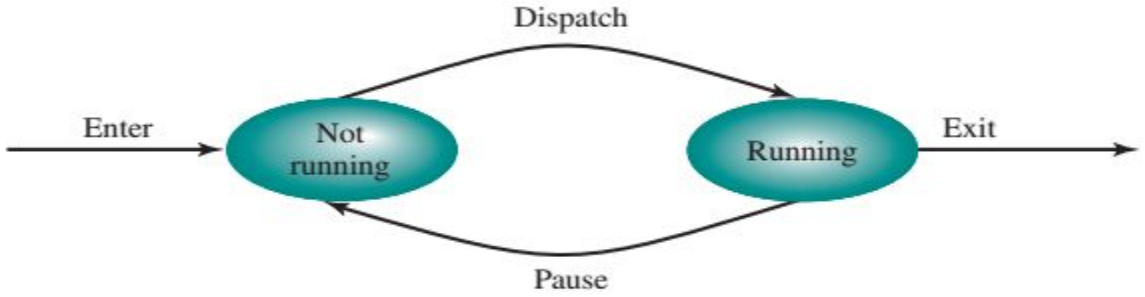
Process termination

- Process termination is an operation in which a process is terminated after the execution of its last instruction.
- This operation is used to terminate or end any process.
- When a process is terminated, the resources that were being utilized by the process are released by the operating system.
- When a child process terminates, it sends the status information back to the parent process before terminating.
- The child process can also be terminated by the parent process if the task performed by the child process is no longer needed.
- OS de-allocate all the resources hold by this process.



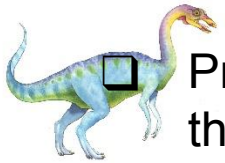
A Two-State Process Model

- In this model, a process may be in one of the two states: Running or Not Running, as shown in below figure

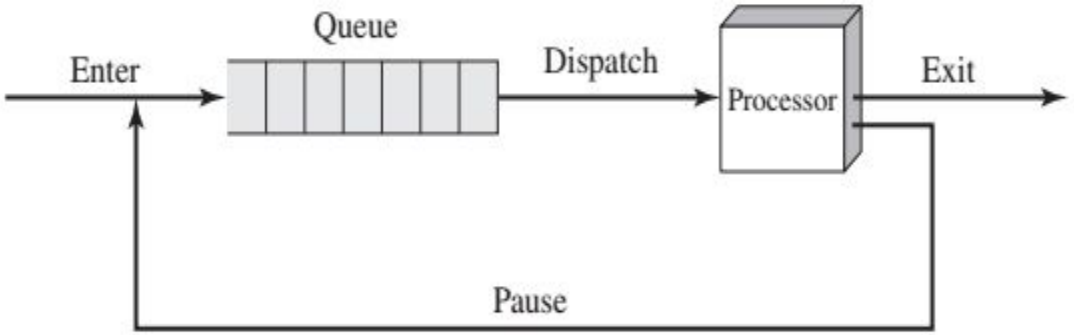


(a) State transition diagram

- When the OS creates a new process, it creates a process control block for the process and enters that process into the system in the Not Running state.
- This Process control block contains information relating to each process, including current state and location in memory.
- The process exists, is known to the OS, and is waiting for an opportunity to execute.
- From time to time, the currently running process will be interrupted, and the dispatcher portion of the OS will select some other process to run.
- The former process moves from the Running state to the Not Running state, and one of the other processes moves to the Running state.

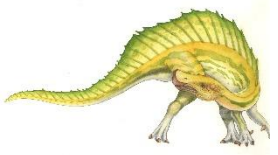


- Processes that are not running are to be kept in some sort of queue, waiting their turn to execute.



(b) Queueing diagram

- Above figure suggests a structure
- There is a single queue in which each entry is a pointer to the process control block of a particular process.
- Each data block in the queue represents one process.
- A process that is interrupted is transferred to the queue of waiting processes.
- Alternatively, if the process has completed or aborted, it is discarded.
- In either case, the dispatcher takes another process from the queue to execute.



Five-State Process Model

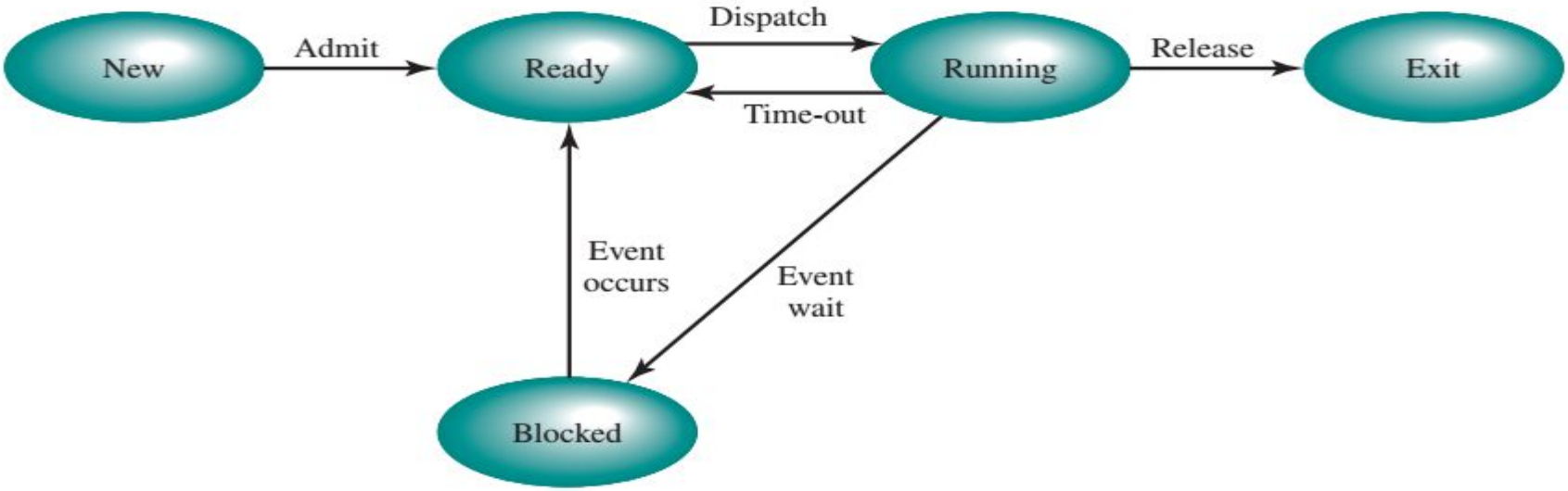


Figure 3.6 Five-State Process Model

New: A process that has just been created but has not yet been admitted to the pool of executable processes by the OS.

Running: The process that is currently being executed

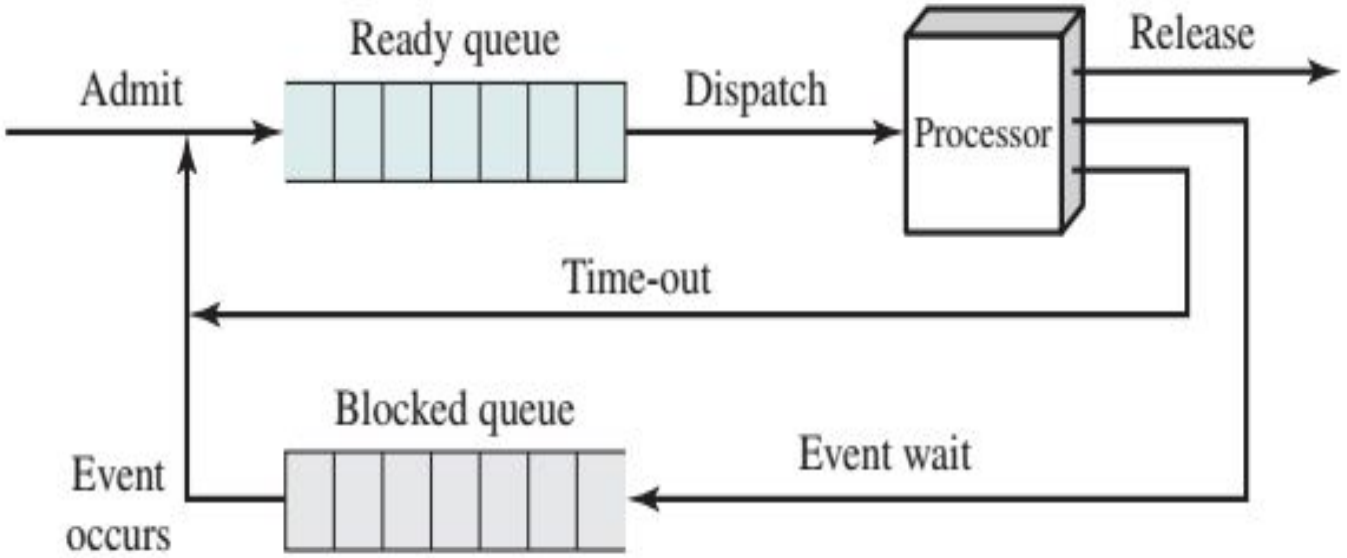
Ready: A process that is prepared to execute when given the opportunity.

Blocked/Waiting: A process that cannot execute until some event occurs, such as the completion of an I/O operation.

Exit: A process that has been released from the pool of executable processes by the OS, either because it halted or because it aborted for some reason.

Figure 3.6 indicates the types of events that lead to each state transition for a process; the possible transitions are as follows:

- **Null to New:** A new process is created to execute a program.
- **New to Ready:** The OS will move a process from the New state to the **Ready state** when it is prepared to take on an additional process.
- **Ready to Running:** When it is time to select a process to run, the OS chooses one of the processes in the Ready state which is the job of the scheduler or dispatcher.
- **Running to Exit:** The currently running process is terminated by the OS if the process indicates that it has completed or if it aborts.
- **Running to Ready**: The most common reason for this transition is that the running process has reached the maximum allowable time for uninterrupted execution

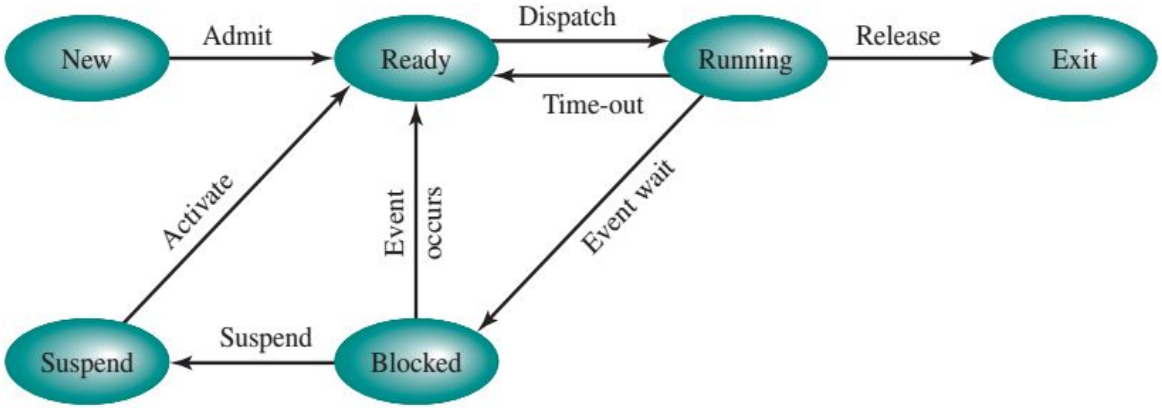


- Above figure shows queueing discipline of five state model implemented with two queues: a Ready queue and a Blocked queue.
- As each process is admitted to the system, it is placed in the Ready queue and OS to choose another process to run.
- When the OS to choose another process to run, it selects one from the Ready queue. In first-in-first-out queue mode.
- When a running process is removed from execution, it is either terminated or placed in the Ready or Blocked queue.
- Finally, when an event occurs, any process in the Blocked queue that has been waiting on that event only is moved to the Ready queue.

Suspended Processes

- We know that each process to be executed must be loaded into main memory.
- In uniprocessor system, processor remains idle because of execution speed mismatch of processor and I/O devices. An I/O device is slower than processor
- Also memory holds multiple processes and the processor select another process when one process is blocked. But the processor is so much faster than I/O that it will be common for all of the processes in memory to be waiting for I/O. Thus, even with multiprogramming, a processor could be idle most of the time.
- Thus Main memory could be expanded to accommodate more processes and when none of the processes in main memory is in the Ready state, the OS swaps one of the blocked processes out on to disk into a suspend queue
- The OS then brings in another process from the suspend queue or it honors a new-process request. Execution then continues with the newly arrived process

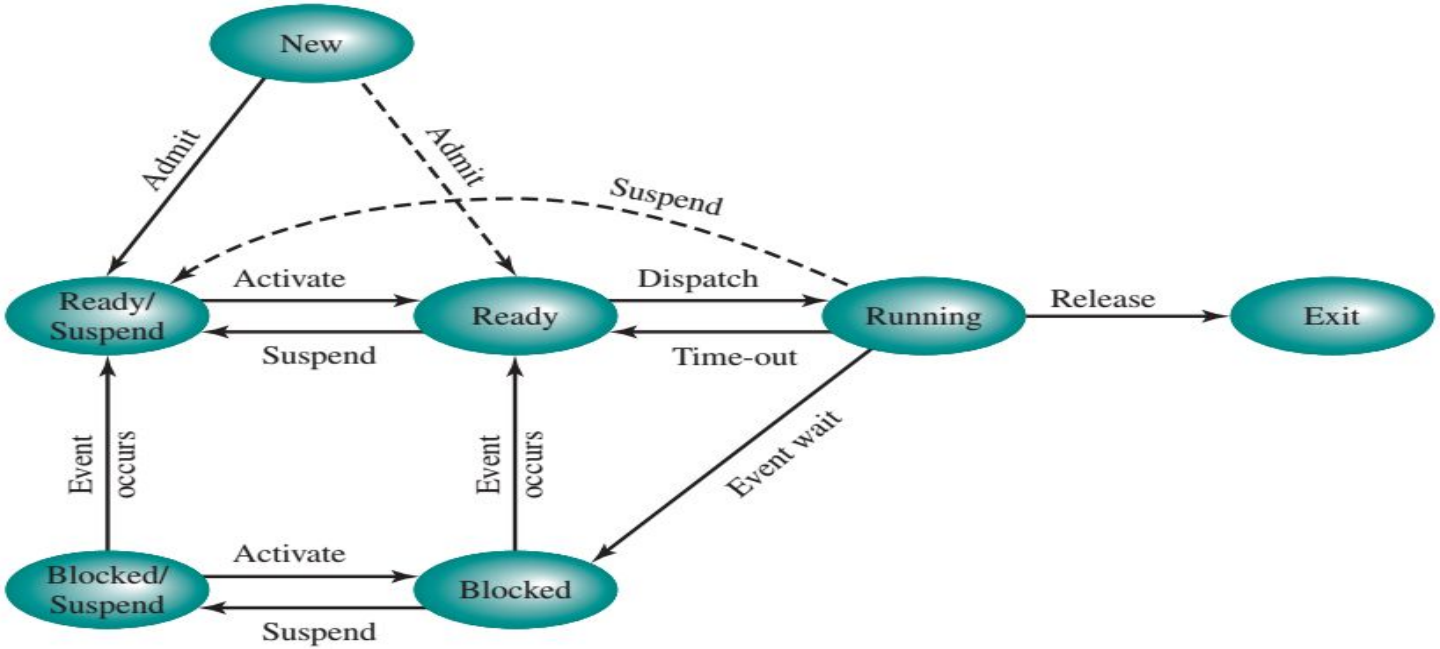
- When all of the processes in main memory are in the blocked state , the OS suspend one process by putting it in the suspend state and transferring it to disk.
- When the OS has performed a swapping-out operation, it has two choices for selecting a process to bring into main memory: It can admit a newly created process, or it can bring in a previously suspended process.



(a) With one Suspend state

- Above f **uspend States**
- There are two independent concepts here: whether a process is waiting on an event (blocked or not), and whether a process has been swapped out of main memory (suspended or not).

- To accommodate this 2 * 2 combination, we need four states:
- **1. Ready:** The process is in main memory and available for execution.
  2. Blocked: The process is in main memory and awaiting an event.
- **3. Blocked/Suspend:** The process is in secondary memory and awaiting an event.
  4. Ready/Suspend: The process is in secondary memory but is available for execution as soon as it is loaded into main memory .



(b) With two Suspend states

Blocked to Blocked/Suspend: The process is in secondary memory and awaiting an event to occur. If there are no ready processes at least one blocked process is swapped out to make room for another process that is not blocked.

Blocked/Suspend to Ready/Suspend: A process in the Blocked/Suspend state is moved to the Ready/Suspend state when the event for which it has been waiting occurs.

Ready/Suspend to Ready: When there are no ready processes in main memory, the OS will need to bring one in to continue execution. Also a process in the Ready/Suspend state has higher priority than any of the processes in the Ready state.

New to Ready/Suspend and New u **Ready:** When a new process is created, it can either be added to the Ready queue or the Ready/Suspend queue. In either case, the OS must create a process control block and allocate an address space to the process. I

Any State u **Exit:** Typically, a process terminates while it is running, either because it has completed or because of some fatal fault condition

PROCESS DESCRIPTION

- OS acts as entity that manages the use of system resources by processes

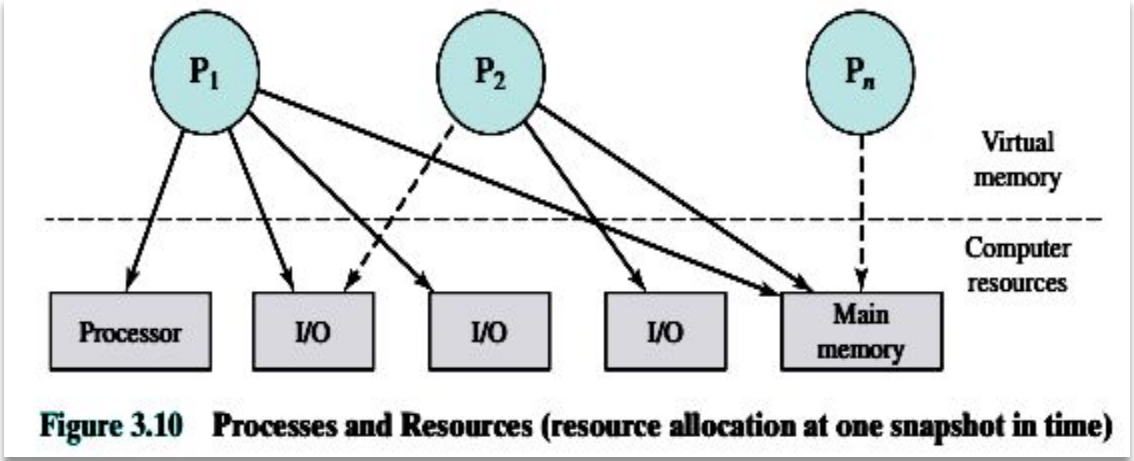
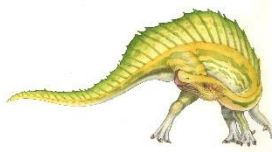


Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)

- Consider a multiprogramming environment, let (P1, ……Pn) be the number of process that have been created and exist in virtual memory.
- Each process, during the course of its execution, needs access to certain system resources, including the processor, I/O devices, and main memory
- In the figure, process P1 is running; at least part of the process is in main memory, and it has control of two I/O devices.
- Process P2 is also in main memory, but is blocked waiting for an I/O device allocated to P1.
- Process P*n* has been swapped out and is therefore suspended.



Operating System Control Structures

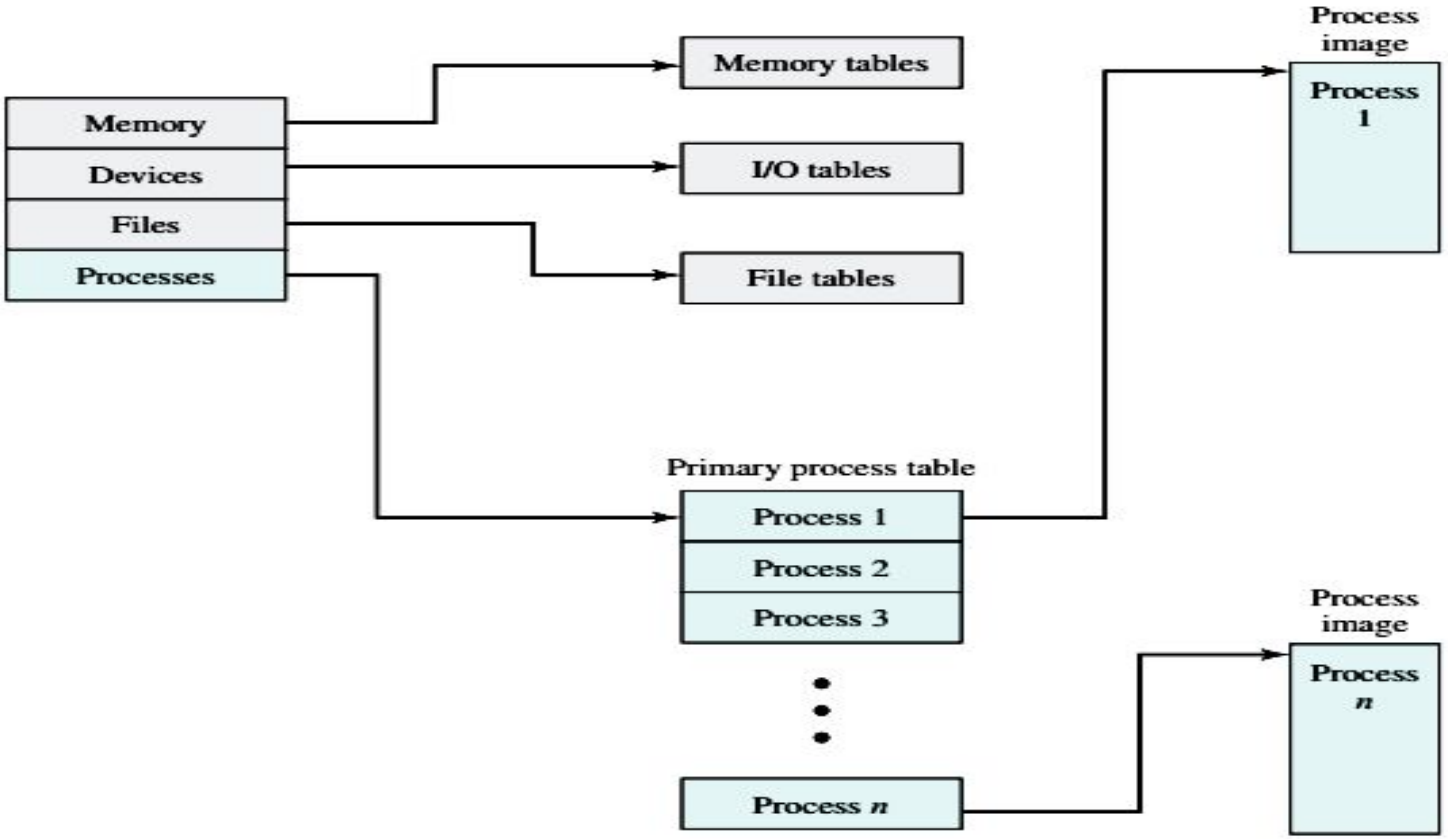


Figure 3.11 General Structure of Operating System Control Tables

- If the OS is to manage processes and resources, it must have information about the current status of each process and resource i.e. the OS constructs and maintains tables of information about each entity that it is managing.
- Figure 3.11, which shows four different types of tables maintained by the OS: memory, I/O, file, and process.

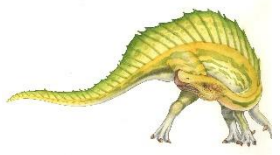
- **Memory Tables** :Memory tables are used to keep track of both main and secondary memory.
- Here the processes are maintained on secondary memory
- The memory tables must include the following information:
- 1. Allocation of main memory to processes
- 2. Allocation of secondary memory to processes
- 3. Information needed to manage virtual memory
- **I/O Tables** :Used by the OS to manage the I/O devices and channels of the computer
- Here 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 the existence of files, their location on secondary memory, their current status, and other attributes

Process Tables: The OS must maintain process tables to manage processes.

Process Control Structures

- To manage and control a process the OS needs to know two things first, it must know where the process is located; second, it must know the attributes of the process that are necessary for its management (e.g., process ID and process state).
- ***Process Location***
- The collection of program, data, stack, and attributes can be referred as the process image.
- The location of a process image will depend on the memory management.
- The process image is maintained as a continuous, block of memory. This block is maintained in secondary memory, usually disk so that the OS can manage the process.
- To execute the process, the entire process image must be loaded into main memory, or at least virtual memory.
- Thus, the OS needs to know the location of each process on disk and, for each such process that is in main memory, the location of that process in main memory
- Typical Elements of a Process Image are **User Data** , **User Program, Stack** and **Process Control Block**



***Process attributes*:**

- The collection of attributes is referred to as a *process control block* .
- We can group the process control block information into three general categories:
  1. Process identification
  2. Processor state information
  3. Process control information

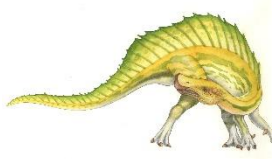
Process identification

- Each process is assigned a unique numeric identifier, which may simply be an index into the primary process table.
- Many of the other tables controlled by the OS may use process identifiers to cross reference process tables

Processor State Information

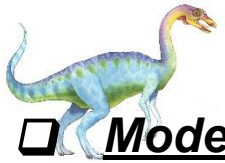
- **It** consists of the contents of processor registers.
- While a process is running, the information is in the registers

- When a process is interrupted, all of this register information must be saved so it can be restored when the process resumes execution.
- The nature and number of registers involved depend on the design of the processor.
- Typically, the register set will include uservisible registers, control and status registers, and stack pointers.
- **Registers** are the *smaller* and the *fastest* accessible **memory units in the CPU**
- **User-Visible Registers** :These registers are visible to the assembly or machine language programmers
- **Control and Status Registers** :These registers holds the **address or data** that is important to **control the processor's operation**. **They are not visible** to the users.
- **Stack Pointers** :A stack is used to store parameters and calling addresses for procedure and system calls. The stack pointer points to the top of the stack.



Process control information

- This is the additional information needed by the OS to control and coordinate the various active processes
- It contains scheduling and state Information , Data structuring, Interprocess Communication , Process Privileges, Memory Management
- **Scheduling and state Information** :This is information that is needed by the operating system to perform its scheduling function.
- **Data structuring: A** process may be linked to other process in a queue, ring, or some other structure
- **Interprocess Communication:** Various flags, signals, and messages are associated with communication.
- **Process Privileges :**Processes are granted privileges in terms of the memory that may be accessed and the types of instructions that may be executed.
- **Memory Management:** It is the function responsible for managing the computer's primary memory.



Process control

- ***Modes of Execution***:

1) **The less-privileged mode**: Certain instructions can only be executed in the more-privileged mode. These would include reading or altering a control register, primitive I/O instructions, and instructions that relate to memory management. In addition, certain regions of memory can only be accessed in the more-privileged mode. The less-privileged mode is often referred to as the **user mode**, because user programs typically would execute in this mode.

2) **The more-privileged mode :** The more-privileged mode is referred to as the **system mode**, **control mode**, or **kernel mode.**

- The reason for using two modes is that It is necessary to protect the OS and key operating system tables, such as process control blocks, from interference by user programs.

- ***Process Creation*: The** steps involved in actually creating the process are

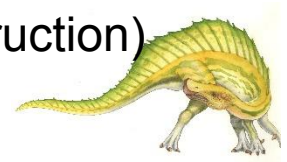
1. **Assign a unique process identifier to the new process.** At this time, a new entry is added to the primary process table, which contains one entry per process
2. **Allocate space for the process.** This includes all elements of the process image. Thus, the OS must know how much space is needed for the private user address space (programs and data) and the user stack .

.

3. **Initialize the process control block.** The process control information portion is initialized based on standard default values plus attributes. The processor state information portion will typically be initialized with most entries zero, except for the program counter and system stack pointers.
4. **Set the appropriate linkages.** For example, if the OS maintains each scheduling queue as a linked list, then the new process must be put in the Ready or Ready/ Suspend list.
5. **Create or expand other data structures.** For example, the OS may maintain an accounting file on each process to be used subsequently for billing.

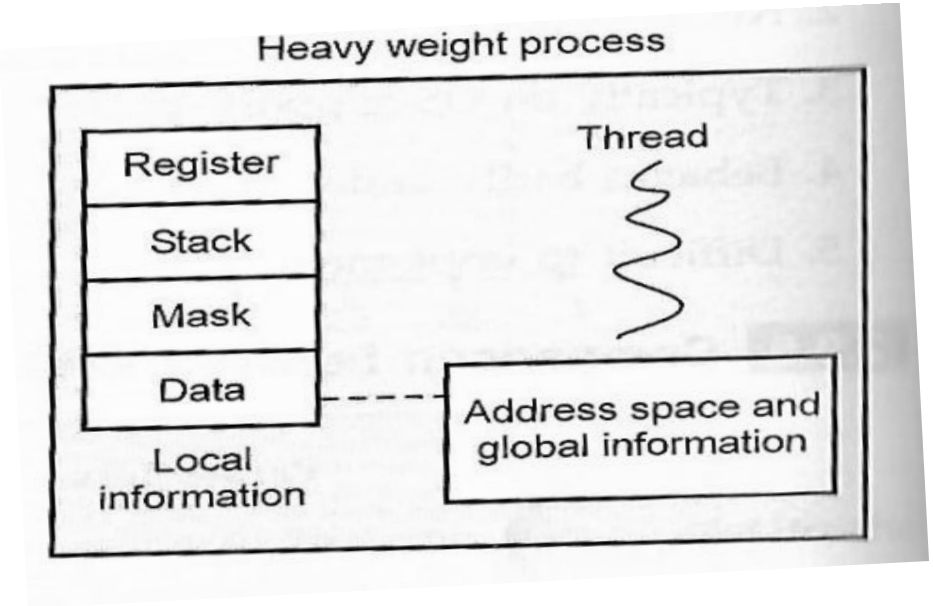
Process Switching

- A process switch may occur any time that the OS has gained control from the currently running process.
- That is during 1) supervisor Call(explicit request by the program)
- 2) Trap (an error resulted from the last instruction)
- 3) Interrupt (the cause is external to the execution of the current instruction)

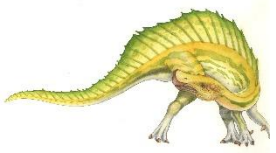


Threads overview

- Thread is a basic processing unit to which OS allocates processor time.
- Thread is a light weight process (LWP), is a basic unit of CPU utilization
  - It comprises thread ID ,program counter, register set and stack
  - It shares code section and data section belonging to same program
- Threads are scheduled on a processor. Each thread can execute a set of instructions independent of other threads and processes.



- Every program has at least one thread.
- When user double click on an icon by using mouse, the operating system creates a process and that process has on thread that runs the icons code.
- Each thread belongs to exactly one process and no thread can exist outside process. Each thread represents a separate flow of control.
- Normally , an OS will have a separate thread for each different activity and for each process and thread will perform OS activities on behalf of the process. In this condition ,each user is backed by a kernel thread.
- When process issues a system call to read a file, the process thread will take over ,find out which disk accesses to generate and issue low levels instructions. It then suspends until Disk finishes reading in the data.
- System call is an explicit request to the kernel mode via a software interrupt. When user mode process invokes a system call, the CPU switches to kernel mode and starts the execution of the kernel function.
- An example of an application that could make use of threads is a file server



- The key benefits of threads derive from the performance implications:

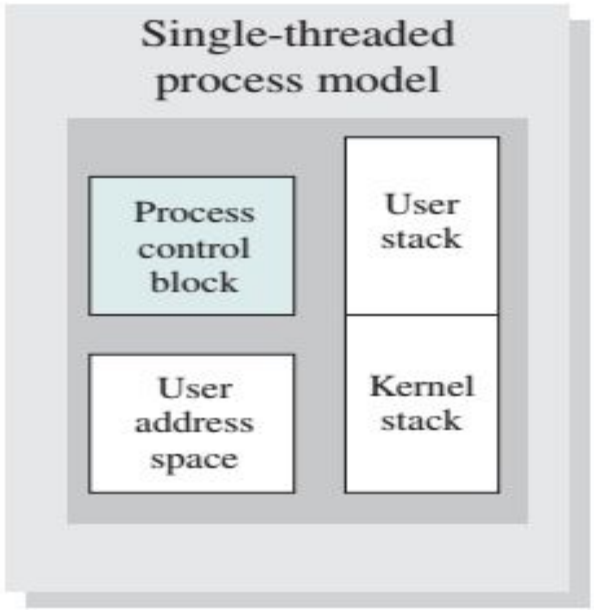
1. It takes far less time to create a new thread in an existing process, than to create a brand-new process.
2. It takes less time to terminate a thread than a process.
3. It takes less time to switch between two threads within the same process than to switch between processes.

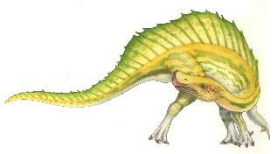
Examples of the uses of threads in a single-user multiprocessing system:

1. **Foreground and background work:** For example, in a spreadsheet program, one thread could display menus and read user input, while another thread executes user commands and updates the spreadsheet.
2. **Asynchronous processing**: For example, as a protection against power failure, one can design a word processor to write its random access memory (RAM)

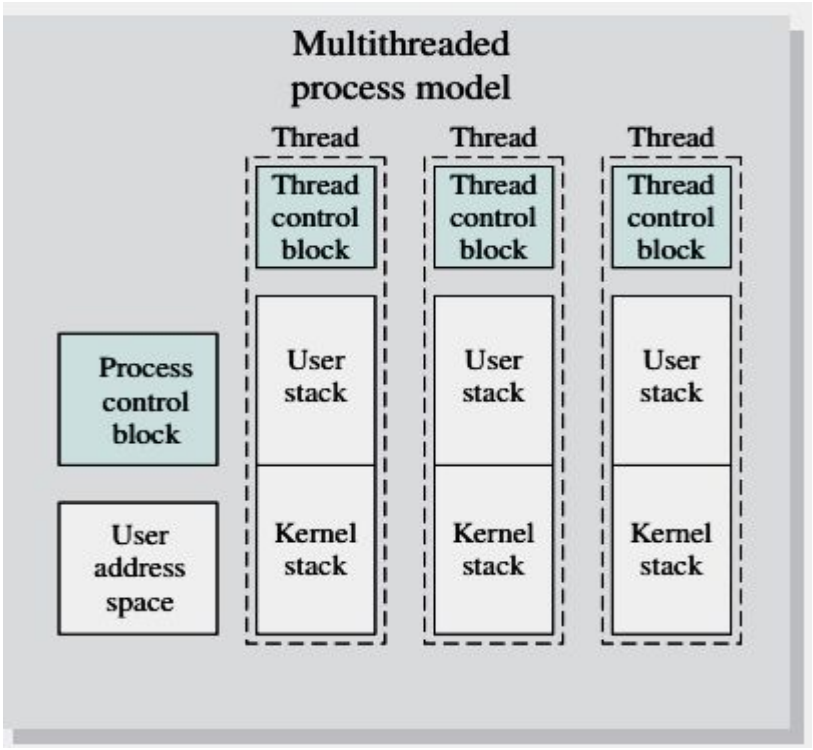
Distinction between Single and multi threaded process model

- In a single-threaded process model , the representation of a process includes its process control block and user address space, as well as user and kernel stacks to manage the call/return behavior of the execution of the process.
- While the process is running, it controls the processor registers.
- The contents of these registers are saved when the process is not running





- In a multithreaded environment, there is still a single process control block and user address space associated with the process.
- However but now there are separate stacks for each thread, as well as a separate control block for each thread containing register values, priority, and other thread-related state information.
- Thus, all of the threads of a process share the state and resources of that process. They reside in the same address space and have access to the same data.



Difference between thread and process

Thread	Process
Thread is also called lightweight process.	Process is also called heavyweight process.
Operating system is not required for thread switching.	Operating system interface is required for process switching.
One thread can read, write or even completely clean another threads stack.	Each process operates independently of the other process.
All threads can share same set of open files and child processes.	In multiple processing, each process executes the same code but has its own memory and file resources.
If one thread is blocked and waiting then second thread in the same task can run.	If one server process is blocked then other server process cannot execute until the first process is unblocked.
Uses fewer resources.	Uses more resources.

Multithreading

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

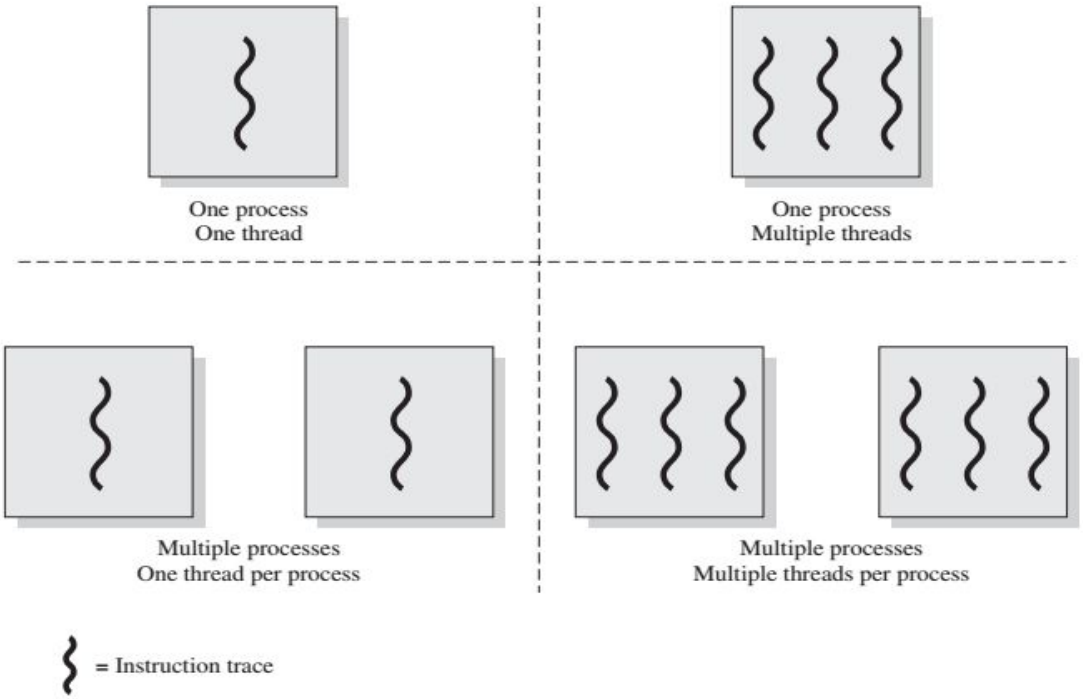
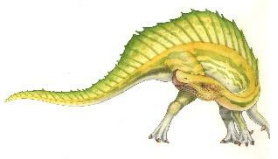


Figure 4.1 Threads and Processes

- In a multithreaded environment, a process is defined as the unit of resource allocation and a unit of protection.

- The following are associated with processes:

1. A virtual address space that holds the process image
2. Protected access to processors, other processes files, and I/O resources
3. A thread execution state (Running, Ready, etc.)
4. A saved thread context when not running
5. An execution stack.



Multithreading Modules

1. **One to one**

- In this model, one user level thread maps with one kernel level thread. If there are three user threads then system creates three separate kernel thread.
- In this method, the operating system allocates data structure that represents kernel threads. Here multiple threads are run in parallel on multiprocessor system

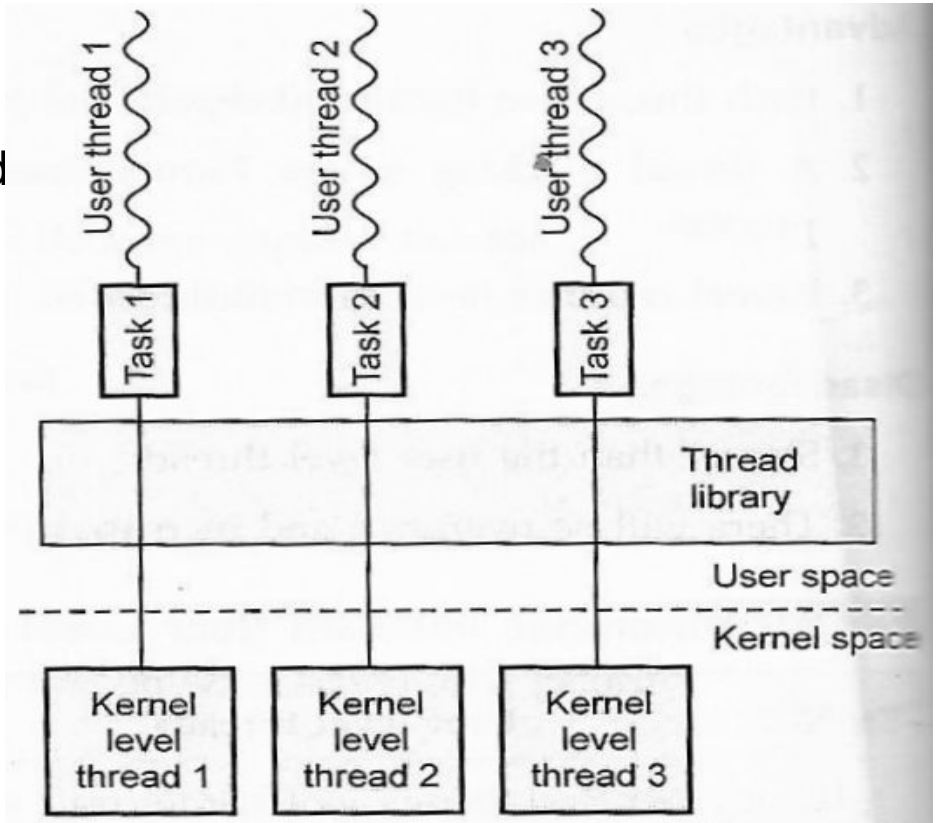


Fig. 2.12.1 One to one multithreading model

- Windows 95/XP and Linux operating system uses this one to one thread mapping

2. **Many to one**

- Many to one mapping means many user thread maps with one kernel thread.
- Thread library handle thread management in user space.
- The many-to-one model does not allow individual processes to be split across multiple CPUs.
- This type of relationship provides an effective context-switching environment
- Example Solaris Unix OS uses green thread for implementation

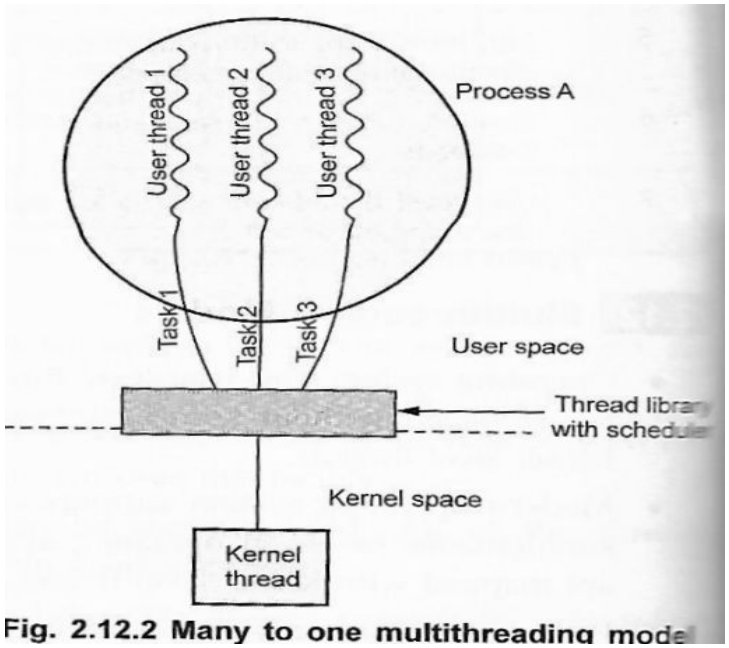
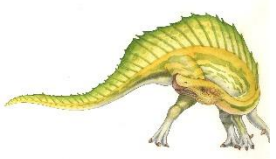
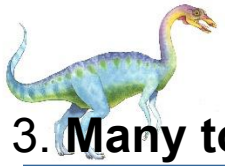


Fig. 2.12.2 Many to one multithreading model





3. Many to many

- In the many to many mapping, many user-level threads maps with equal number of kernel threads.
- Many to many mapping is also called M-to-N thread mapping. Thread pooling is used to implement this method.
- Users can create required number of thread and there is no limitation for user. Again here blocking Kernel system calls do not block the entire process.
- This model support for splitting processes across multiple processors.

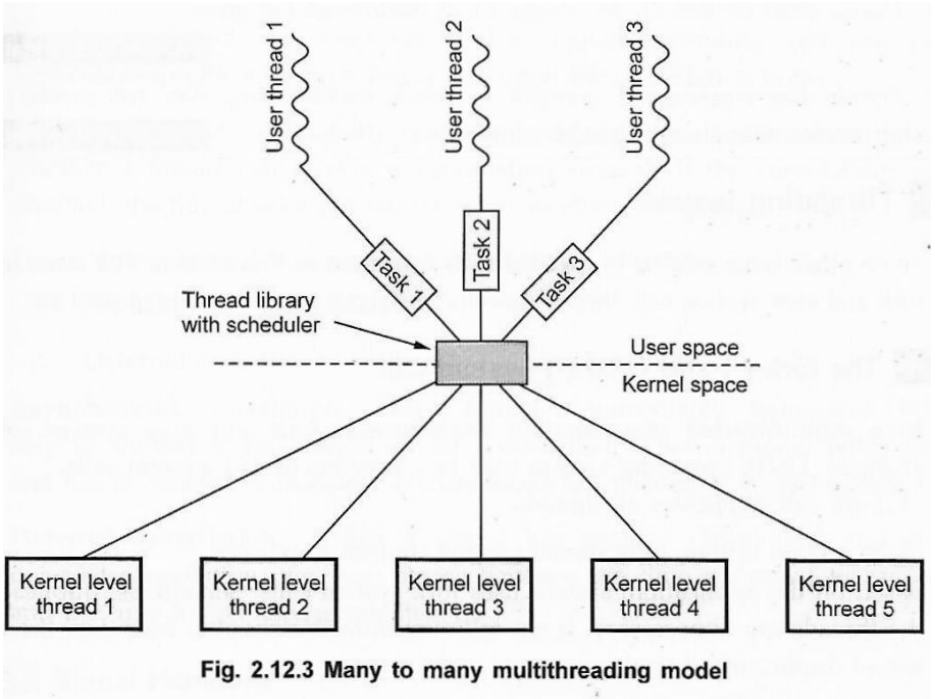
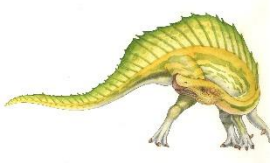


Fig. 2.12.3 Many to many multithreading model



Microkernel

Kernel

- Kernel is a software or code that resides in the central core of a operating system. It has complete control over the system.
- When os boots, kernel is the first part of the OS to load into memory. Kernel remains in memory for the entire duration of the computer session.
- Executing processes and handling interrupts are performed by kernel in kernel space.
- microkernels provide minimal process like defining memory address and process management, in addition to a communication facility.
- The main function of the microkernel is to provide a communication facility between the client program and the various services that are also running in user space.
- Communication is provided by *message passing* For example, if the client program wishes to access a file, it must interact with the file server. The client program and service never interact directly. Rather, they communicate indirectly by exchanging messages with the microkernel.
- It runs in kernel mode and rest run in normal user processes
- Microkernels and their user environments are usually implemented in the C++ or C programming languages with a little bit of assembly

Microkernel architecture

- Microkernel architecture is an architecture with kernel having the basic interaction with hardware and the basic Inter-Process Communication mechanisms.

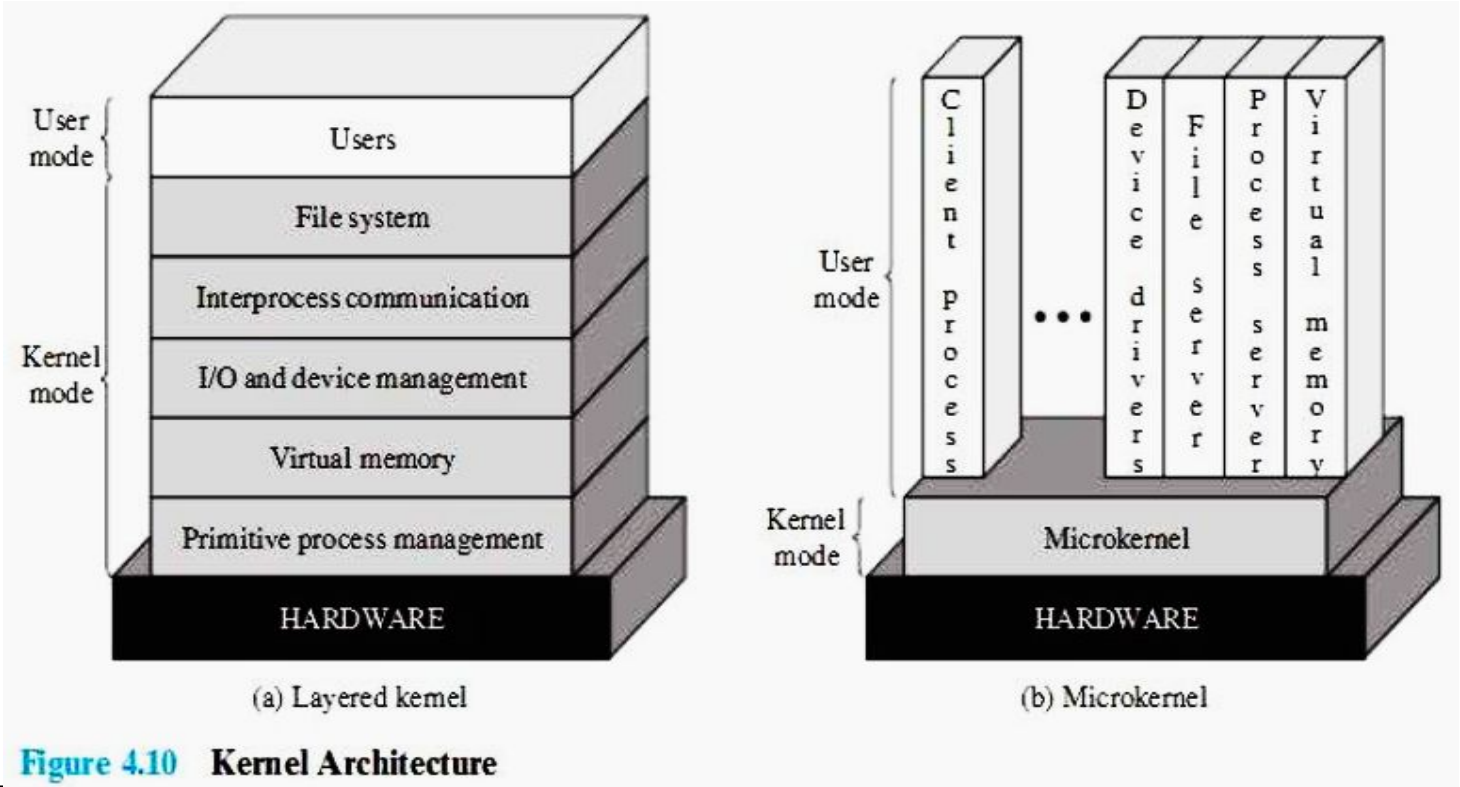
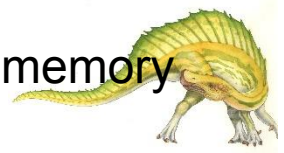
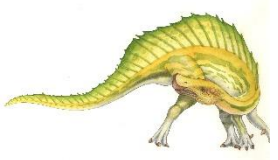


Figure 4.10 Kernel Architecture

- **A virtual memory** module outside the microkernel decides when to bring a page into memory and which page already in memory is to be replaced; the microkernel maps these page references into a physical address in main memory
- Page is the smallest unit of data for memory management in a virtual memory



- **INTERPROCESS COMMUNICATION:** The communication between processes or threads .
- These communication in microkernel operating system are messages.
- A message includes a header that identifies the sending and receiving process and a body that contains direct data, a pointer to a block of data, or some control information about the process.
- With microkernel architecture it is possible to handle interrupt as messages and to include I/O ports in address spaces

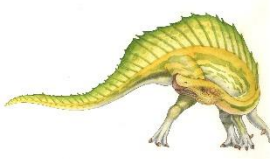


Benefits

1. **Responsiveness**. Multithreading 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
2. **Resource sharing:** The benefit of sharing resources is that it allows an application to have several different threads of activity within the same address space.
3. **Economy:** Allocating memory and resources for process creation is costly. Because threads share the resources of the process to which they belong, it is more economical to create and context-switch thread.
4. **Scalability.** Threads are running in parallel on different processors. Multithreading on a multi-CPU machine increases parallelism

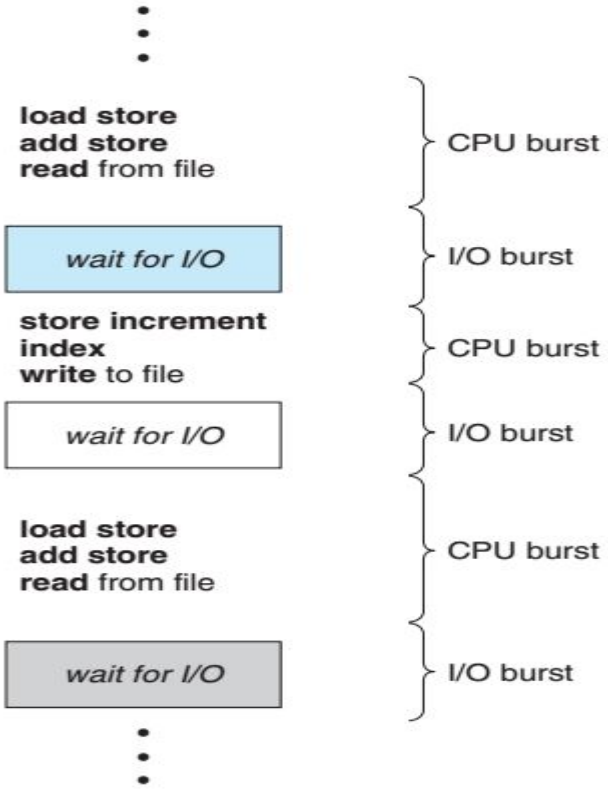
CPU Scheduling

- CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold(in waiting state) thereby making full use of CPU.
- The aim of CPU scheduling is to make the system efficient, fast and fair.
- When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process. This pattern continues. Every time one process has to wait, another process can take over use of the CPU.
- In other words it switches from one process to another process.
- The success of CPU scheduling depends on process execution which consists of a cycle of CPU execution and I/O wait.
- Process execution begins with a **CPU burst t**hat is followed by an **I/O burst**, which is followed by another CPU burst, then another I/O burst, and so on. Eventually, the final CPU burst ends with a system request to terminate execution as shown in figure



CPU–I/O Burst Cycle

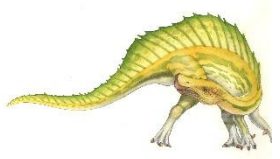
- Almost all processes alternate between two states in a continuing cycle, as shown in Figure 6.1 below :
  1. A CPU burst of performing calculations, and
  2. An I/O burst, waiting for data transfer in or out of the system.



gure 5.1 Alternating sequence of CPU and I/O bursts.

CPU Scheduler

- Whenever the CPU becomes idle, it is the job of the CPU Scheduler to select another process from the ready queue to run next process.
- The scheduler selects a process from the processes in memory that are ready to execute and allocates the CPU to that process.
- This selection process is carried out by the short-term scheduler (or CPU scheduler).
- It makes the decision as to which available process in the ready queue will be executed by the processor next.
- Short-term scheduling is the actual decision of which ready process to execute next
- CPU scheduler is responsible for multiplexing processes on the CPU.



Preemptive and non- preemptive Scheduling

- CPU-scheduling decisions may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state . This due to I/O request or an request of wait for the termination of one of the child processes
2. When a process switches from the running state to the ready state. This is due to when an interrupt occurs.
3. When a process switches from the waiting state to the ready state . This is due to when completion of I/O operation.
4. When a process terminates.

- When scheduling takes place only under circumstances 1 and 4, we say that the scheduling scheme is **non-preemptive** or cooperative.
- When scheduling takes place only under circumstances 2 and 3, we say it is **preemptive**.
- For conditions 1 and 4 there is no choice - A new process must be selected.
- For conditions 2 and 3 there is a choice - To either continue running the current process, or select a different one
- This **non-preemptive** scheduling method was used by Microsoft Windows 3.x and by the Apple Macintosh operating systems.

Dispatcher

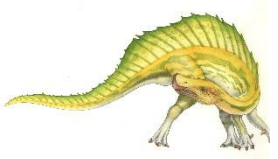
- Another component involved in the CPU-scheduling function is the dispatcher.
- The dispatcher is the unit that gives control of the CPU to the process which is selected by the short-term scheduler.
- In other words, when the short-term scheduler completes its job of selecting a process, it is the dispatcher which takes that process to the desired state/queue.
- This function involves the following:
  • **Switching context**: This is the task where switching the CPU from one process to another process requires saving the state of the old process and loading the saved state for the new process.
  • Switching to user mode
  • Jumping to the proper location in the user program to restart the progrm

Scheduling criteria

1. **CPU Utilization:** It is a average function of time during which the processor is busy. In other words it is the percentage of time that the processor is busy. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system).
2. **Throughput :** **If** the CPU is busy executing processes, then work is being done or rather say total amount of work done in a unit of time. In other words It is the total number of processes completed per unit time This may range from 10/second to 1/hour depending on the specific processes.
3. **Turnaround Time**: It is the amount of time taken to execute a particular process. In other words it is the interval from time of submission of the process to the time of completion of the process. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O. **Burst time + waiting time/ completion time- Arrival time**
4. **Waiting time:** It is amount of time a process has been waiting in the ready queue to acquire get control on the CPU. In other words it is the sum of the periods spent waiting in the ready queue. **Starting process time –Arrival time/ Turnaround time-burst time**
5. **Response time**: It is the time from the submission of a request until the first response is produced. In other words it is the time it takes to start responding, not the time it takes to output the response.

First Come First Serve Scheduling

- In the "**First come first serve**" scheduling algorithm, the process which arrives first, gets executed first, or we can say that the process which requests the CPU first, gets the CPU allocated first.
- First Come First Serve, is just like FIFO(First in First out) Queue data structure, where the data element which is added to the queue first, is the one who leaves the queue first.
- New process enters through the **tail** of the queue, and the scheduler selects process from the **head** of the queue.
- When new process enters in to the system, its process control block gets linked to the end of the ready queue and it is removed from the front of the queue.
- When a process waits or blocks, it is removed from the queue and it queues up again in FCFS queue when it gets ready.
- A perfect real life example of FCFS scheduling is buying tickets at ticket counter.

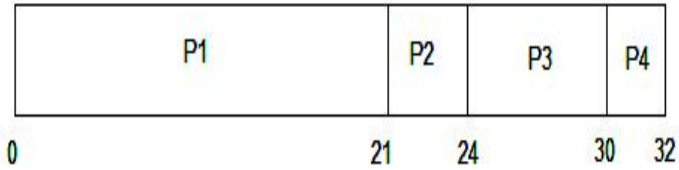


Example

- Consider the processes P1, P2, P3, P4 given in the below table, arrives for execution in the same order, with Arrival Time 0, and given Burst Time, let's find the average waiting time using the FCFS scheduling algorithm.
- First **P1** will be provided with the CPU resources,
- Hence, waiting time for P1 will be 0.
- P1 requires 21 ms for completion, hence waiting time for P2 will be 21 ms.
- Similarly, waiting time for process P3 will be execution time of P1 + execution time for P2, which will be (21 + 3) ms = 24 ms.
- For process P4 it will be the sum of execution times of P1, P2 and P3.

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The average waiting time will be = (0 + 21 + 24 + 30)/4 = 18.75 ms



This is the GANTT chart for the above processes

Shortest-Job-First Scheduling

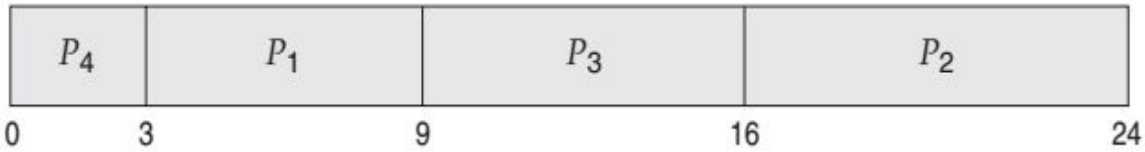
- This algorithm associates with each process the length of the process's next CPU burst.
- Use these lengths to schedule the process with the shortest time
- In other words when the CPU becomes available, it gets assigned to the process that has the smallest next CPU burst..
- The next CPU burst is generally predicted as an **exponential average** of the measured lengths of previous CPU bursts.
- We can define the exponential average with the following formula. Let *tn* be the length of the *n*th CPU burst, and let t*n*+1 be our predicted value for the next CPU burst. Then, for 0<a< 1 , define

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n.$$

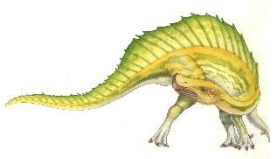
- This is the best approach to minimize waiting time.
- This scheduling algorithm is ideal if all the jobs/processes are available at the same time. (either Arrival time is 0 for all, or Arrival time is same for all)
- consider the following set of processes at arrival time 0, with the length of the CPU burst given in milliseconds:

Example

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3



- The waiting time is 3 milliseconds for process P1, 16 milliseconds for process P2, 9 milliseconds for process P3, and 0 milliseconds for process P4.
- Thus, the average waiting time is (3 + 16 + 9 + 0)/4 = 7 milliseconds.



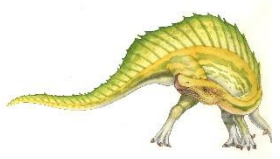
Priority Scheduling

- A priority is associated with each process, and the CPU select higher priority process first
- Equal-priority processes are scheduled in FCFS order
- Each process is assigned priority number for the purpose of CPU scheduling
- Priority scheduling can be either preemptive or non-preemptive.
- When a process arrives at the ready queue, its priority is compared with the priority of the currently running process.
- A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process which means the processing of the current process is stopped and the incoming new process with higher priority gets the CPU for its execution.
- A non-preemptive priority scheduling algorithm will simply put the new process at the head of the ready queue which means after the execution of the current process it will be processed.
- The Shortest-Job-First algorithm is a special case of the general **priority scheduling algorithm.**
- The larger the CPU burst, the lower the priority, and vice versa.
- A major problem with priority scheduling algorithms is **indefinite blocking**, or **starvation**.
- A solution to the problem of indefinite blockage of low-priority processes is aging. Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time

Example

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

The average waiting time is 8.2 milliseconds



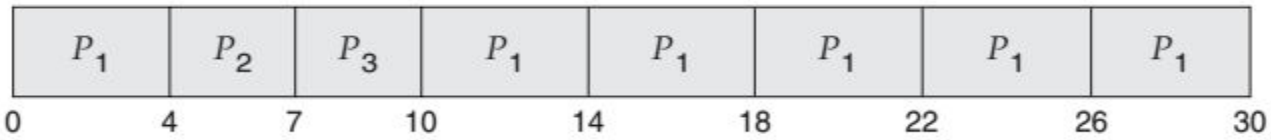
Round-Robin Scheduling

- The round-robin (RR) scheduling algorithm is designed especially for timesharing systems.
- A small unit of time, called a time quantum or time slice, is defined.
- Round robin scheduling is the preemptive scheduling in which every process get executed in a cyclic way, i.e. in this a particular time slice is allotted to each process which is known as time quantum.
- Every process, which is present in the queue for processing, CPU is assigned to that process for that time quantum.
- Now, if the execution of the process gets completed in that time quantum, then the process will get terminate otherwise the process will again go to the ready queue, and the previous process will wait for the turn to complete its execution.

Consider the following set of processes that arrive at time 0(arrival time), with the length of the CPU burst given in milliseconds:

Process	Burst Time
P_1	24
P_2	3
P_3	3

- If we use a time quantum of 4 milliseconds, then process *P*1 gets the first 4 milliseconds.
- Since it requires another 20 milliseconds, it is preempted after the first time quantum, and the CPU is given to the next process in the queue, process *P*2.
- Process *P*2 does not need 4 milliseconds it requires only 3 ms, so it quits before its time quantum expires at
- The CPU is then given to the next process, process *P*3 at (4+3+3) 10 milliseconds
- Once each process has received 1 time quantum, the CPU is returned to process *P*1 for an additional time quantum. The resulting RR schedule is as follows:



Multilevel Queue Scheduling

- Another class of scheduling algorithms has been created for situations in which processes are easily classified into different groups.
- For example, a common distribution is made between **foreground** (interactive) processes and **background** (batch) processes having different response-time.
- A multilevel queue scheduling algorithm partitions the ready queue into several separate queues where the processes are permanently assigned to one queue .
- Each queue has its own scheduling algorithm.
- The foreground queue might be scheduled by an RR algorithm, while the background queue is scheduled by an FCFS algorithm.
- Each queue has absolute priority over lower-priority queues.

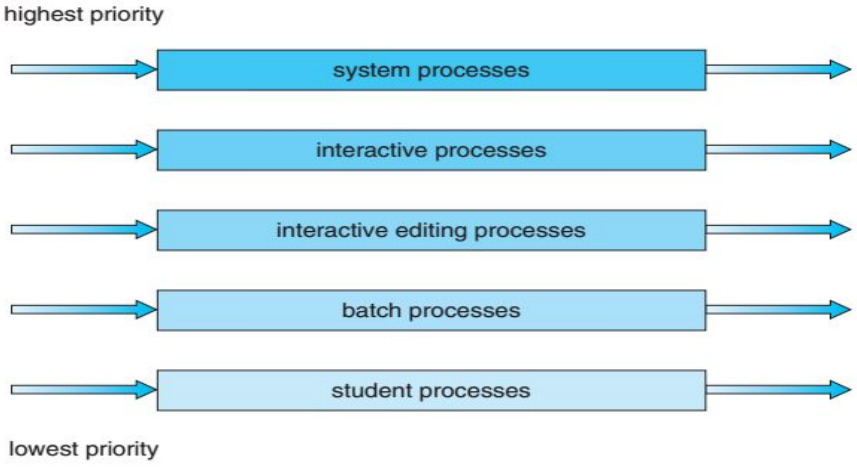
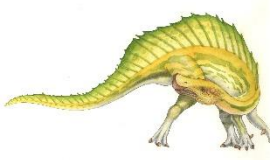
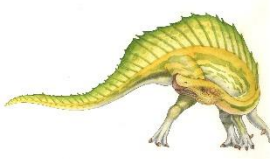


Figure 5.6 Multilevel queue scheduling.

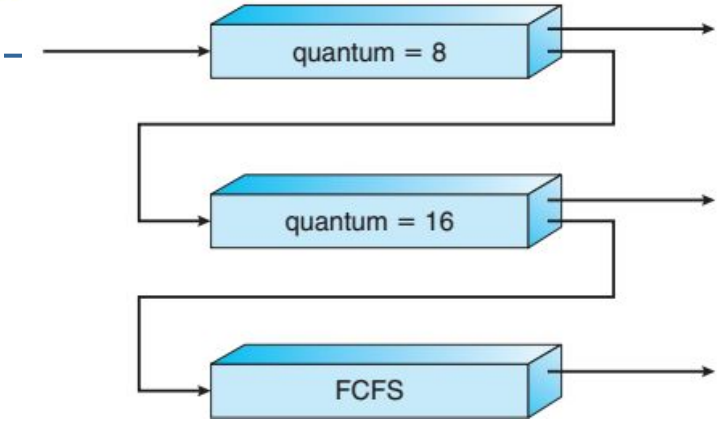


Multilevel Feedback Queue Scheduling

- This scheduling algorithm allows a process to move between queues and to separate processes according to the characteristics of their CPU bursts.
- If a process uses too much CPU time, it will be moved to a lower-priority queue. In addition, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.
- In general, a multilevel feedback queue scheduler is defined by the following parameters:
  - The number of queues
  - The scheduling algorithm for each queue
  - The method used to determine when to upgrade a process to a higher priority queue
- The method used to determine when to demote a process to a lower priority queue
- The method used to determine which queue a process will enter when that process needs service



Example

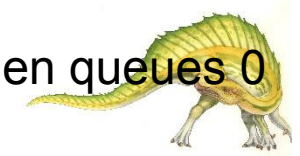


Queue 0 – RR with time quantum 8 milliseconds

Queue 1 – RR time quantum 16 milliseconds

Q2 – FCFS

- The scheduler first executes all processes in queue 0. Only when queue 0 is empty will it execute processes in queue 1. Similarly, processes in queue 2 will only be executed if queues 0 and 1 are empty.
- A process that arrives for queue 1 will preempt a process in queue 2. A process in queue 1 will in turn be preempted by a process arriving for queue 0.
- A process entering the ready queue is put in queue 0.
- A process in queue 0 is given a time quantum of 8 milliseconds. If it does not finish within this time,
  it is moved to the tail of queue 1. If queue 0 is empty, the process at the head of queue 1 is given a quantum of 16 milliseconds. If it does not complete, it is preempted and is put into queue 2.
- Processes in queue 2 are run on an FCFS basis but are run only when queues 0 and 1 are empty



Algorithm Evaluation