

# **Module 2**

## **PROCESS MANAGEMENT**

## **Process**

A program is a set of statements or instructions.

A process is a program in execution.

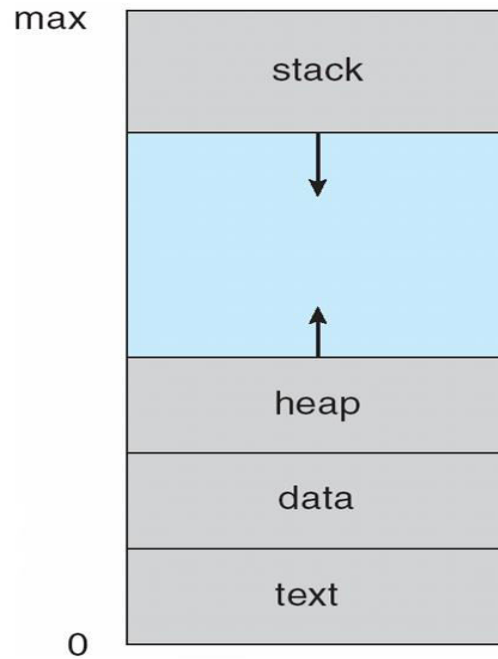
Program is a passive entity.

Process is an active entity.

Program resides in hard disk.

Process resides in RAM.

The memory allocated by the operating system for a process consists of four parts as shown in following figure.



The text part contains instructions or statements of the process.

Data part contains global variables.

Stack contains temporary data like function parameters, return values and so on.

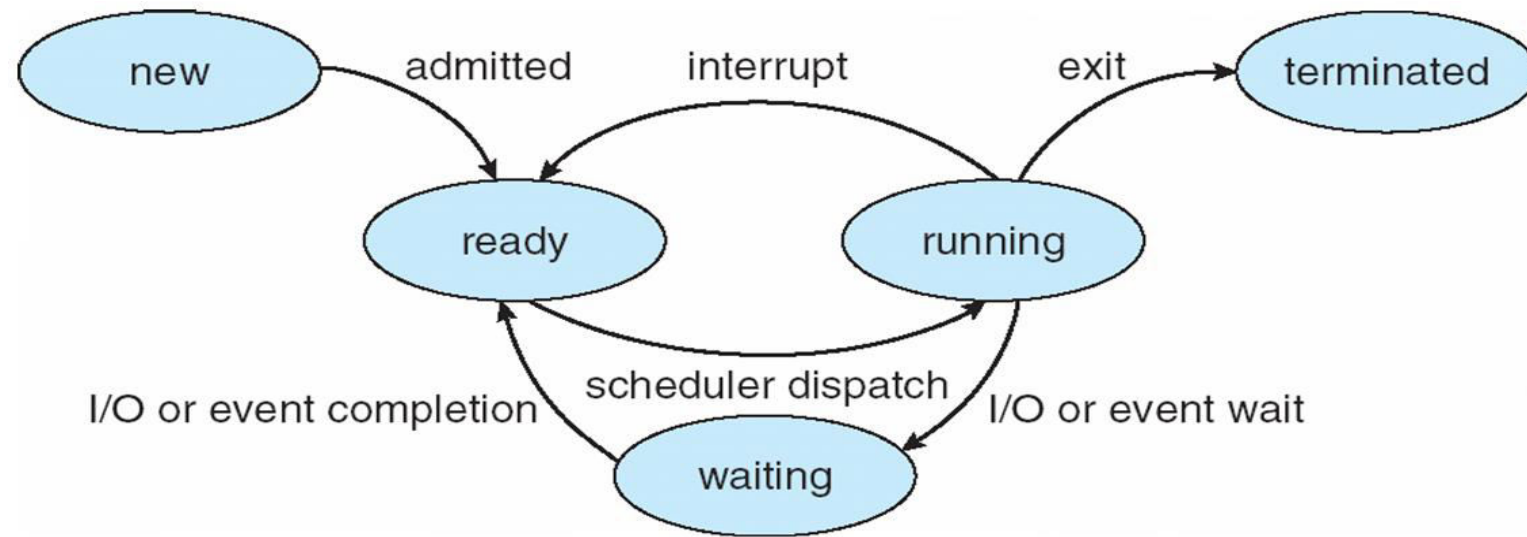
Heap part contains the objects that are dynamically created during execution of process.

## **Process States**

A process at any particular time will be in any one of the following states

- 1) New - the process is being created.
- 2) Ready - the process is ready for execution.
- 3) Running – the process is executing.
- 4) Waiting – the process is waiting for an event to occur.
- 5) Exit – the process completed its execution.

The change in state of a process is indicated through the following diagram



When the process is ready for execution on allocation of CPU then the process moves from new state to ready state.

When CPU is allocated to the process then the process moves from ready state to running state.

During execution of the process:

- 1) If the process is completed then the process moves from running state to exit state.
- 2) If any i/o operation is requested then the process moves from running state to waiting state.
- 3) If the allocated time is over then the process moves from running state to ready state.

When the requested i/o operation is completed then the process moves from waiting state to ready state.

## Process Control Block (PCB)

Operating system creates a process control block for each process in the system.

Process control block contains information about the process.

The information stored in the process control block of process is:

Process state
CPU registers
CPU scheduling information
Memory management information
Accounting information
I/O information
.
.
.

**Process state:** indicates current state of the process.

**CPU registers:** indicates the values stored in the registers.

**CPU-scheduling information:** indicates scheduling information like priority of process.

**Memory management information:** indicates the starting and ending positions or addresses of process in the RAM.

**Accounting information:** indicates process id, amount of CPU time required and so on.

**I/O status information:** indicates the list of i/o devices allocated to the process, the list of open files and so on.

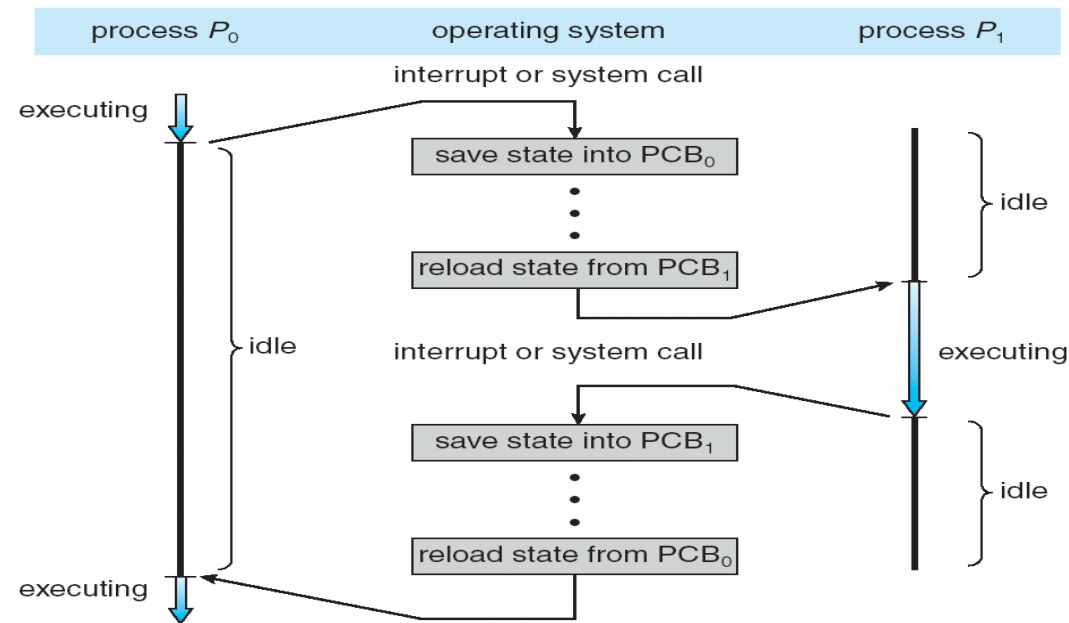


## Context switch

Switching the CPU from one process to other process is called context switch.

The PCBs of the processes are updated when the CPU is switched from one process to other process.

The following diagram shows state change in process control block when context switch occurs.







Initially CPU is allocated to process  $P_0$ .

During execution of process  $P_0$ , if process  $P_0$  invokes a system call or requests any i/o operation then

- 1) the execution of process  $P_0$  is stopped
- 2) the state of process  $P_0$  is saved into  $PCB_0$
- 3) the state of process  $P_1$  is reloaded from  $PCB_1$
- 4) the CPU is allocated to process  $P_1$ .

While executing process  $P_1$ , if process  $P_1$  invokes a system call or requests any i/o operation then

- 1) the execution of process  $P_1$  is stopped
- 2) the state of process  $P_1$  is saved into  $PCB_1$
- 3) the state of process  $P_0$  is reloaded from  $PCB_0$
- 4) the CPU is allocated to process  $P_0$ .

## **Process Scheduling**

To increase the utilization of CPU, the operating system loads number of programs at a time into RAM.

If number of programs is ready for execution then the operating system has to decide an order for executing the programs i.e. the operating system has to schedule the execution of programs.

### **Scheduling queues**

Operating system maintains two types of queues

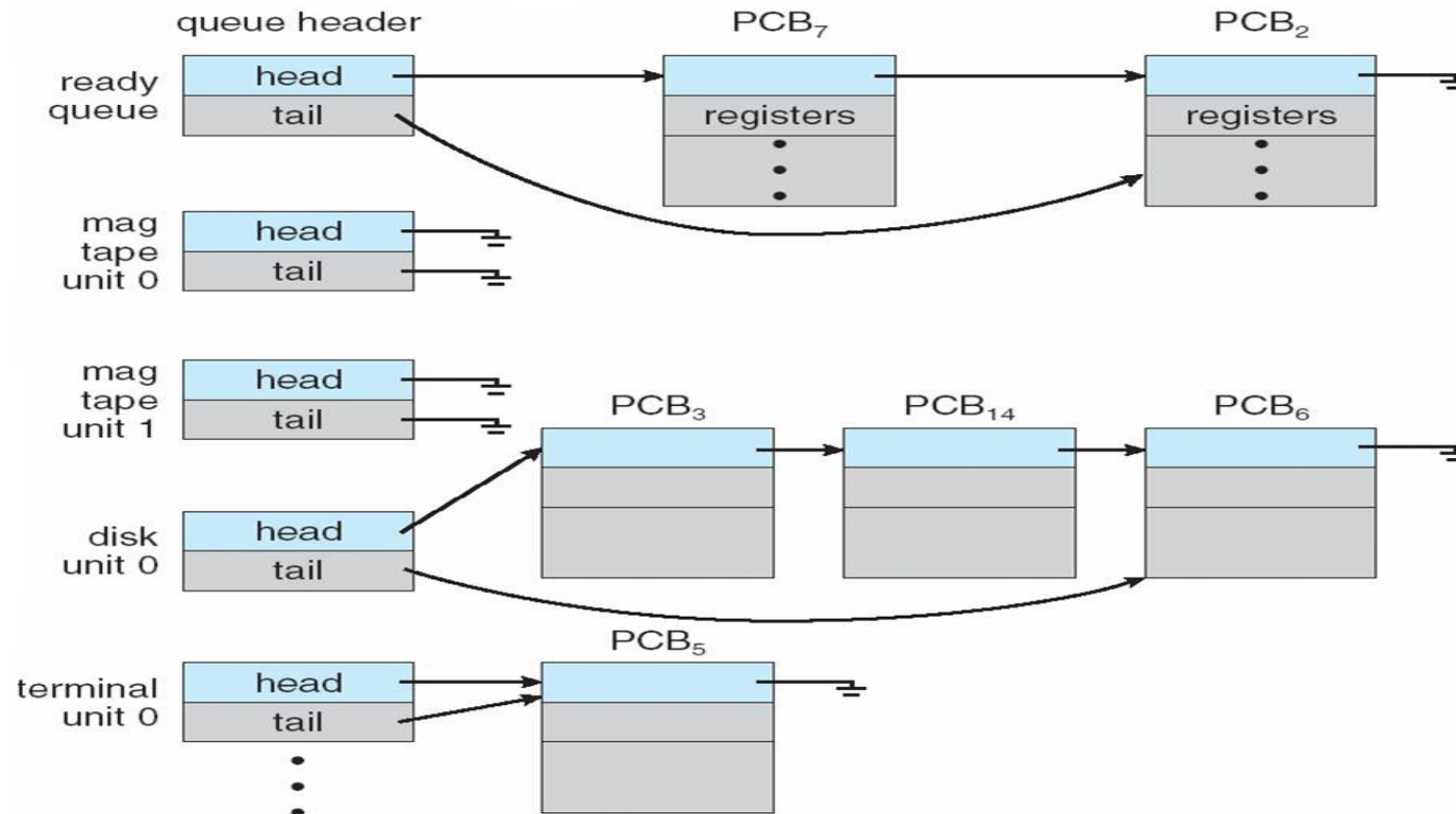
- 1) Ready queue
- 2) Device queues

#### **Ready queue**

The ready queue contains the PCBs of processes that are ready for execution.

## Device queue

A device queue is maintained for each device. The device queue of a device contains the PCBs of the processes that are waiting for the device.



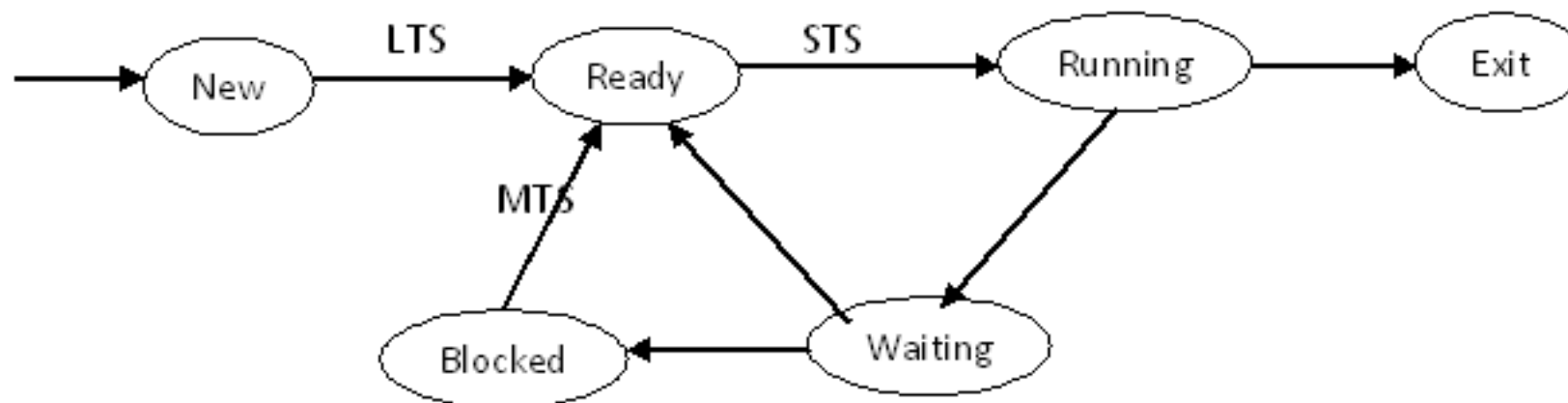
## Schedulers

A process moves around different queues during its life time.

The operating system uses different types of schedulers to select processes from these queues. Different schedulers used by the operating system are

- 1) Long Term Scheduler (LTS)
- 2) Medium Term Scheduler (MTS)
- 3) Short Term Scheduler (STS) or CPU scheduler

The following process state diagram shows when the three schedulers are activated by the operating system.



When a process requests any i/o operation during its execution then the process moves to the waiting state.

A waiting process is in the RAM.

When number of processes in the RAM are in waiting state then the operating system moves some of the waiting processes from the RAM to the hard disk in order to load new processes into the RAM.

When a waiting process is moved from RAM to hard disk then the process goes to the blocked state.

After completion of the i/o operation, the process goes to the ready state from either waiting or blocked state.

### **Long term Scheduler**

Long term scheduler selects one process from the list of processes in the hard disk and loads that process into RAM for execution.

### **Medium term scheduler**

Medium term scheduler selects one process from the list of waiting processes in the hard disk and loads that process into RAM.



### **Short term scheduler or CPU scheduler**

Short term scheduler selects one process from the ready queue and allocates the CPU to that process.

Short term scheduler uses scheduling algorithms for selecting the processes.

The scheduling algorithms are divided into two categories

- 1) Preemptive
- 2) Non-preemptive

In preemptive scheduling, the CPU can be switched from current process to other process forcibly.

In non-preemptive scheduling, the CPU cannot be switched from current process to other process until the current process releases the CPU.

## Scheduling Criteria

Parameters used to evaluate the performance of CPU scheduling algorithms.

The different scheduling criteria are

**CPU utilization:** the percentage of time for which the CPU is busy.

**Throughput:** the number of processes completed per unit time.

**Turnaround time:** difference between the time at which the process has arrived into system and the time at which the process has completed its execution.

$$\text{turnaround time} = \text{completion time} - \text{arrival time}$$

**Waiting time:** the sum of periods for which the process is waiting in the ready queue.

$$\text{waiting time} = \text{start time} - \text{arrival time} / \text{turnaround time} - \text{burst time}$$

**Response time:** difference between the time at which the process has arrived into system and the time at which the first response has come out from the process.

A good scheduling algorithm should maximize CPU utilization, throughput and minimize turnaround time, waiting time and response time.

### **CPU scheduling algorithms**

- 1) First Come First Serve (FCFS)
- 2) Shortest Job First (SJF)
- 3) Priority
- 4) Round Robin (RR)

## **First Come First Serve (FCFS)**

FCFS is non-preemptive scheduling algorithm.

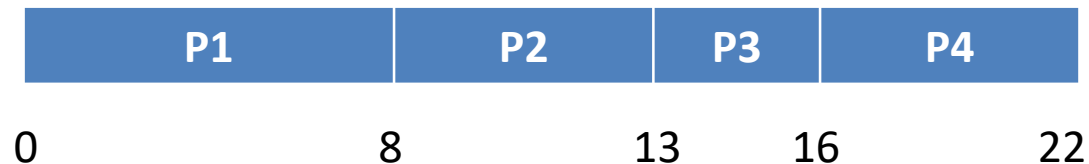
With FCFS scheduling, the processes are executed in the order in which they enter into the system.

Consider the following set of processes

Process	Burst time	Arrival time
P1	8	0
P2	5	0
P3	3	0
P4	6	0

The execution of these processes with FCFS is indicated with following Gantt chart

Gantt chart:



Process	Burst time	Arrival time	Turnaround time	Waiting time
P1	8	0	$8-0=8$	$8-8=0$
P2	5	0	$13-0=13$	$13-5=8$
P3	3	0	$16-0=16$	$16-3=13$
P4	6	0	$22-0=22$	$22-6=16$

The average waiting time is  $(0+8+13+16)/4=9.25$

Ex:

Process	Burst time	Arrival time
P1	8	0
P2	5	1
P3	3	2
P4	6	4

The execution of these processes with FCFS is indicated with following Gantt chart

Gantt chart:



Process	Burst time	Arrival time	Turnaround time	Waiting time
P1	8	0	$8-0=8$	$8-8=0$
P2	5	1	$13-1=12$	$12-5=7$
P3	3	2	$16-2=14$	$14-3=11$
P4	6	4	$22-4=18$	$18-6=12$

The average waiting time is  $(0+7+11+12)/4=7.5$

**Advantages:**

- 1) Simple and easy to implement.

**Disadvantages:**

- 1) Not suitable for time sharing operating systems.
- 2) Average waiting time is high.

## **Shortest Job First**

SJF has two versions: preemptive and non-preemptive.

With SJF algorithm, the process with least burst time is executed first.

If the burst time of two or more processes is same then the processes are executed in FCFS order.

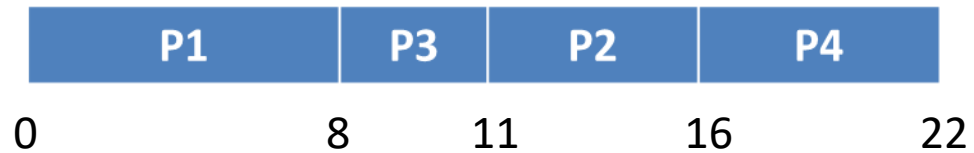


Ex:

Process	Burst time	Arrival time
P1	8	0
P2	5	1
P3	3	2
P4	6	4

The execution of these processes with non-preemptive SJF is indicated with following Gantt chart

Gantt chart:



Process	Burst time	Arrival time	Turnaround time	Waiting time
P1	8	0	$8-0=8$	$8-8=0$
P2	5	1	$16-1=15$	$15-5=10$
P3	3	2	$11-2=9$	$9-3=6$
P4	6	4	$22-4=18$	$18-6=12$

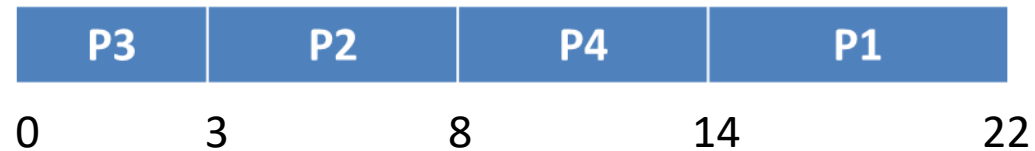
The average waiting time is  $(0+10+6+12)/4=7$

### Non-preemptive SJF

Process	Burst time	Arrival time
P1	8	0
P2	5	0
P3	3	0
P4	6	0

The execution of these processes with non-preemptive SJF is indicated with following Gantt chart

Gantt chart:



Process	Burst time	Arrival time	Turnaround time	Waiting time
P1	8	0	$22-0=22$	$22-8=14$
P2	5	0	$8-0=8$	$8-5=3$
P3	3	0	$3-0=3$	$3-3=0$
P4	6	0	$14-0=14$	$14-6=8$

The average waiting time is  $(14+3+0+8)/4=6.25$

## **Preemptive SJF**

With preemptive SJF, current process is allowed to execute till the arrival of next process.

At the arrival of next process, CPU is allocated to the process with shortest remaining burst time.

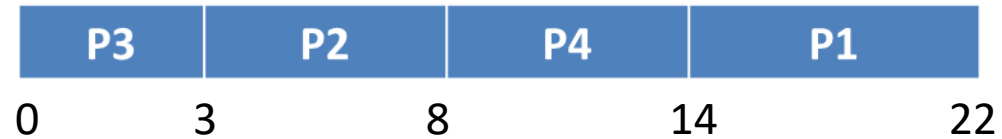
Another name of preemptive SJF is “Shortest Remaining Time First” (SRTF).

For example, consider the following set of processes

Process	Burst time	Arrival time
P1	8	0
P2	5	0
P3	3	0
P4	6	0

The execution of these processes with preemptive SJF is indicated with following Gantt chart

Gantt chart:



Process	Burst time	Arrival time	Turnaround time	Waiting time
P1	8	0	22-0=22	22-8=14
P2	5	0	8-0=8	8-5=3
P3	3	0	3-0=3	3-3=0
P4	6	0	14-0=14	14-6=8

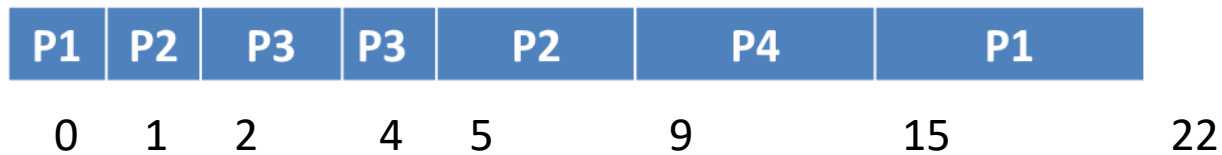
The average waiting time is  $(14+3+0+8)/4=6.25$

Ex:

Process	Burst time	Arrival time
P1	8	0
P2	5	1
P3	3	2
P4	6	4

The execution of these processes with preemptive SJF is indicated with following Gantt chart

Gantt chart:



Process	Burst time	Arrival time	Turnaround time	Waiting time
P1	8	0	22-0=22	22-8=14
P2	5	1	9-1=8	8-5=3
P3	3	2	5-2=3	3-3=0
P4	6	4	15-4=11	11-6=5

The average waiting time is  $(14+3+0+5)/4=5.5$

**Advantages:**

Compared to FCFS, average waiting time is less.

**Disadvantages:**

It can be used only when the burst times of processes are known in advance. Generally, the burst times of processes are not available.

## **Priority**

Priority scheduling also has two versions: preemptive and non-preemptive.

With priority algorithm, the process with highest priority is executed first.

If the priority of two or more processes is same then the processes are executed in FCFS order.

The priorities of processes are generally indicated as 1, 2, 3 and so on.

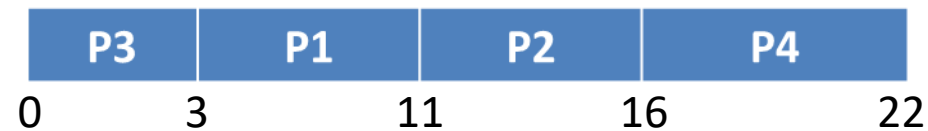
Lower number indicates highest priority.

### Non-preemptive Priority

Process	Burst time	Arrival time	Priority
P1	8	0	2
P2	5	0	3
P3	3	0	1
P4	6	0	4

The execution of these processes with non-preemptive Priority is indicated with following Gantt chart

Gantt chart:



Process	Burst time	Arrival time	Turnaround time	Waiting time
P1	8	0	$11-0=11$	$11-8=3$
P2	5	0	$16-0=16$	$16-5=11$
P3	3	0	$3-0=3$	$3-3=0$
P4	6	0	$22-0=22$	$22-6=16$

The average waiting time is  $(3+11+0+16)/4=7.5$

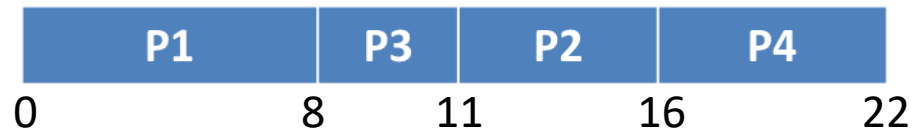


Ex:

Process	Burst time	Arrival time	Priority
P1	8	0	2
P2	5	1	3
P3	3	2	1
P4	6	4	4

The execution of these processes with non-preemptive Priority is indicated with following Gantt chart

Gantt chart:



Process	Burst time	Arrival time	Turnaround time	Waiting time
P1	8	0	$8-0=8$	$8-8=0$
P2	5	1	$16-1=15$	$15-5=10$
P3	3	2	$11-2=9$	$9-3=6$
P4	6	4	$22-4=18$	$18-6=12$

The average waiting time is  $(0+10+6+12)/4=7$

### **Preemptive Priority**

In preemptive Priority, current process is allowed to execute until the arrival of next process.

At the arrival of next process, CPU is allocated to the process with highest priority.

Process	Burst time	Arrival time	Priority
P1	8	0	2
P2	5	0	3
P3	3	0	1
P4	6	0	4

The execution of these processes with preemptive Priority is indicated with following Gantt chart

Gantt chart:



Process	Burst time	Arrival time	Turnaround time	Waiting time
P1	8	0	11-0=11	11-8=3
P2	5	0	16-0=16	16-5=11
P3	3	0	3-0=3	3-3=0
P4	6	0	22-0=22	22-6=16

The average waiting time is  $(3+11+0+16)/4=7.5$

Ex:

Process	Burst time	Arrival time	Priority
P1	8	0	2
P2	5	1	3
P3	3	2	1
P4	6	4	4

The execution of these processes with preemptive Priority is indicated with following Gantt chart

Gantt chart:



Process	Burst time	Arrival time	Turnaround time	Waiting time
P1	8	0	11-0=11	11-8=3
P2	5	1	16-1=15	15-5=10
P3	3	2	5-2=3	3-3=0
P4	6	4	22-4=18	18-6=12

The average waiting time is  $(3+10+0+12)/4=6.25$

## **Disadvantages:**

A process may continuously wait for the CPU. This situation is called starvation.

If there is a continuous flow of higher priority processes into the system then the lower priority processes will never get the CPU.

So, the lower priority processes will wait continuously.

A solution to the problem of starvation is aging.

**Aging** is increasing the priority of processes which have been waiting for long time.

## **Round Robin (RR)**

Round robin algorithm is designed for time sharing operating systems.

RR is similar to FCFS.

RR is preemptive scheduling algorithm.

A time quantum or time slice is used.

The time quantum is generally from 10 to 100 milliseconds.

With RR, the processes are executed in FCFS order and each process is allowed to execute for the time which is specified by quantum time.

If the process is completed within the time specified by quantum time then it will be taken out from the ready queue.

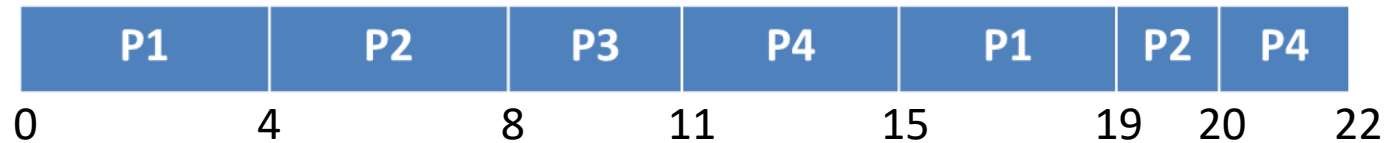
Otherwise, it will be added at the end of ready queue.

Process	Burst time	Arrival time
P1	8	0
P2	5	0
P3	3	0
P4	6	0

quantum time=4

The execution of these processes with RR is indicated with following Gantt chart

Gantt chart:



Process	Burst time	Arrival time	Turnaround time	Waiting time
P1	8	0	19-0=19	19-8=11
P2	5	0	20-0=20	20-5=15
P3	3	0	11-0=11	11-3=8
P4	6	0	22-0=22	22-6=16

The average waiting time is  $(11+15+8+16)/4=12.5$

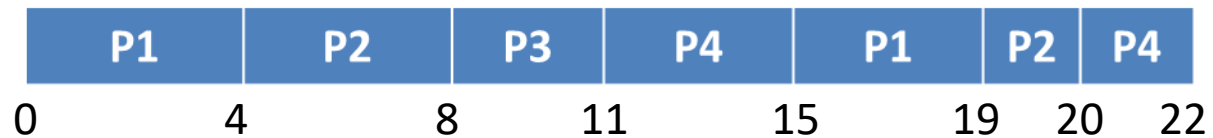
Ex:

Process	Burst time	Arrival time
P1	8	0
P2	5	1
P3	3	2
P4	6	4

Quantum time=4

The execution of these processes with RR is indicated with following Gantt chart

Gantt chart:



Process	Burst time	Arrival time	Turnaround time	Waiting time
P1	8	0	19-0=19	19-8=11
P2	5	1	20-1=19	19-5=14
P3	3	2	11-2=9	9-3=6
P4	6	4	22-4=18	18-6=12

The average waiting time is  $(11+14+6+12)/4=10.75$



**Advantages:**

- 1) All processes are given equal priority.

**Disadvantages:**

- 1) The average waiting time is high.
- 2) If quantum time is less then more number of context switches will occur. More number of context switches leads to wastage of time.
- 3) If quantum time is high then the processes have to wait for more time.

## **Operations on Processes**

There are two basic operations on processes:

- 1) Process creation
- 2) Process termination.

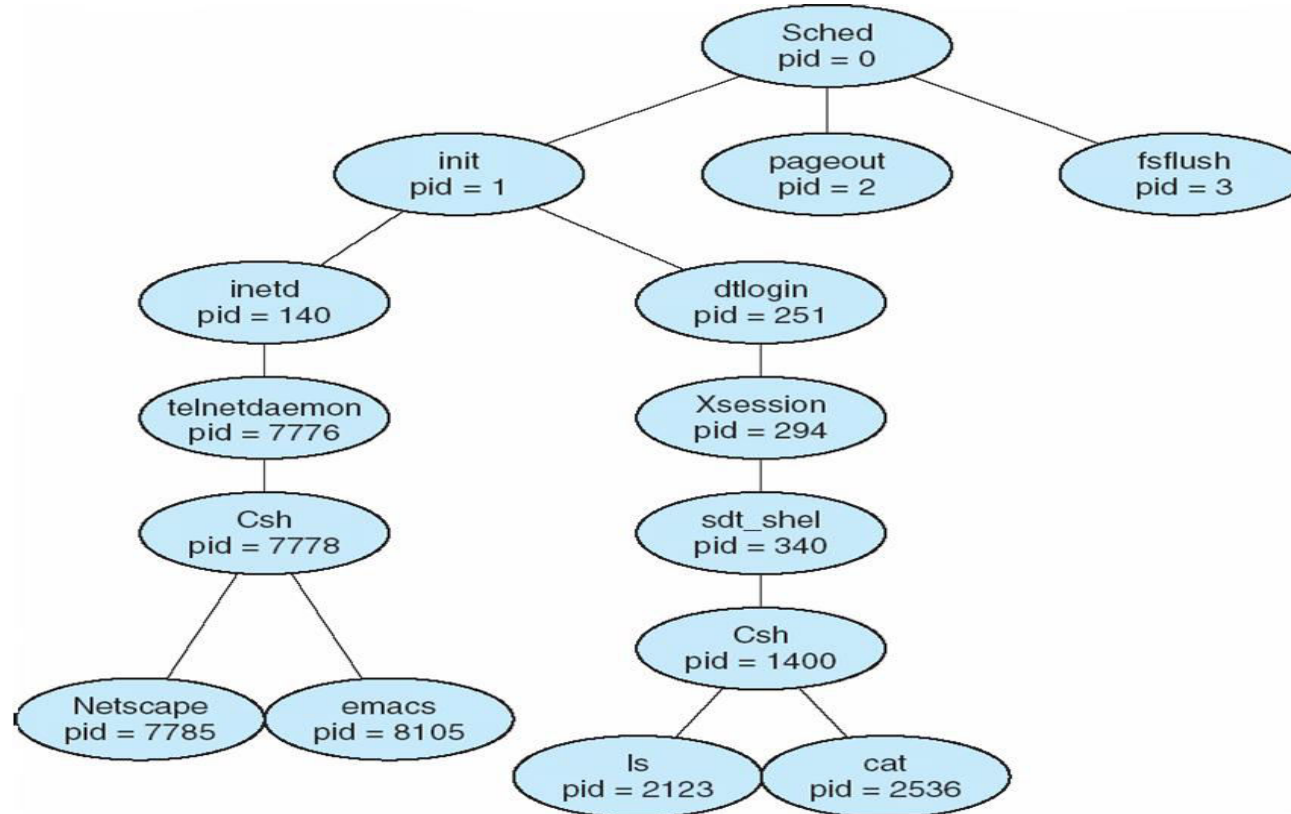
### **Process Creation**

A process may create several new processes during its execution.

The process which creates new processes is called the parent process and the created processes are called the child processes.

Each child process may create other processes.

This creation leads to formation of a tree of processes.



Each process is associated with a unique integer number called process identifier.

This process identifier is used to identify the processes.

Each process requires some resources.

A child process obtains its required resources in any one of the following ways:

- 1) Directly from the operating system
- 2) From its parent process

The parent process has to distribute some resources and share some resources among its child processes.

In addition to resources, the parent process may pass some data to its child processes.

For example, if a child process is created for displaying an image file on the monitor then the parent process has to pass the address of file to the child process.

The child process may be a duplicate of the parent process (child has the same program and data as parent) or the child has a new program.

When a parent process creates a child process then the parent and child processes can execute concurrently or the parent process waits for the completion of some or all of its child processes.

In UNIX operating system, the `fork()` system call is used to create a new process.

When a process creates a child process using `fork()` system call then a copy of the parent process is created.

The parent and child may execute concurrently.

If the parent has nothing to do then it waits for the completion of the child processes.

In Win32 operating system, the `CreateProcess()` system call is used to create a new process.

When a process creates a child process using `CreateProcess()` system call then the child process executes a new program.

The parent process waits for completion of the child process.

## Process Termination

A process terminates when the last statement of the process is executed.

When a process terminates, it informs to its parent about its completion.

The operating system releases all resources from the completed process.

A parent process may terminate its child process when the child process uses its allocated resources for long time (or) the task executed by the child process is no longer required (or) the parent is exiting.

Some operating systems do not allow a child process to exist when the parent process is terminated.

In this case, when the parent process terminates then the operating system terminates all children of the process.

This type of termination is called **cascading termination**.

## Inter-process Communication

In a computer system, number of processes may be executing concurrently.

A process is called **independent** if it is not communicating with any other process.

A process is called **cooperating** if it is communicating with any other process.

Processes must communicate for

**Sharing resources:** for example, if two or more processes require data from same file (resource).

**Speeding up the computation:**

when a program has to be completed in less time then the program is divided into number of parts and the parts are executed concurrently on the processors (processing units) of the system.

The parts must communicate with each other as they belong to the same program.

Two methods are used for implementing inter-process communication

- 1) Shared memory
- 2) Message passing

### **Shared Memory**

With shared memory method, a region of memory shared by processes is created.

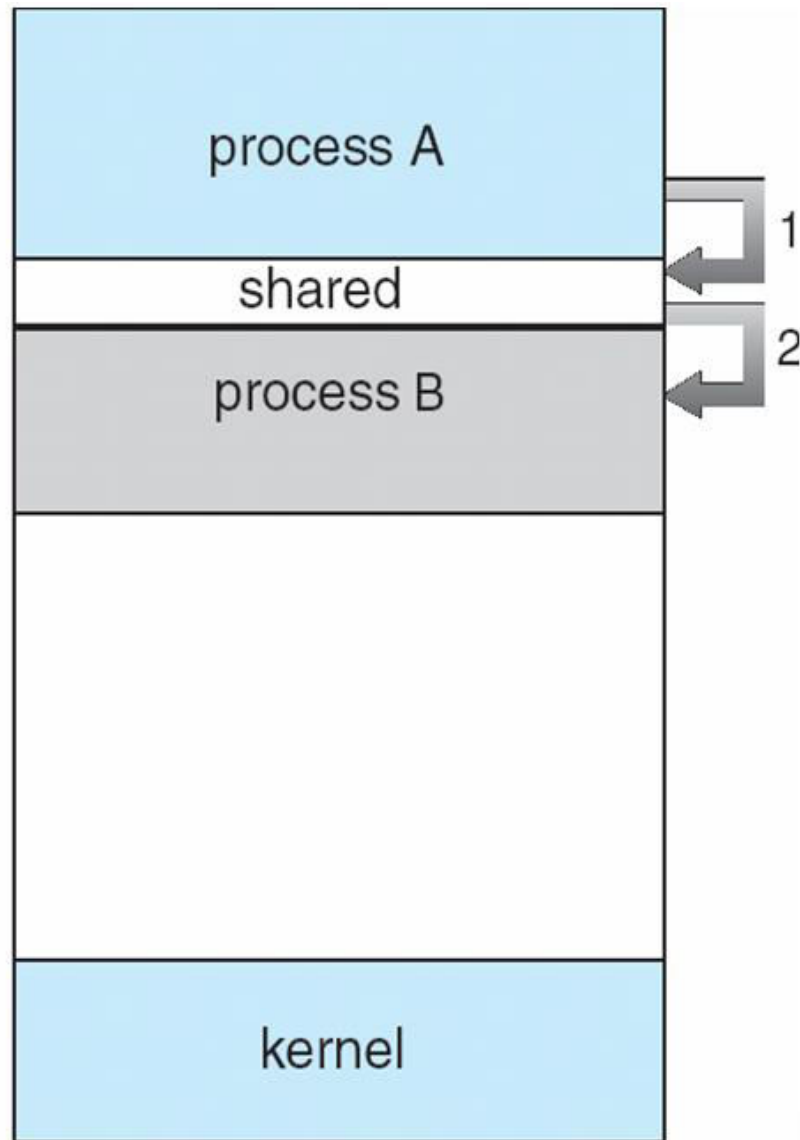
The processes can exchange information by reading and writing data to the shared memory.

A process creates the shared memory segment.

Any process which wants to communicate with the process creating the shared memory must attach the shared memory to its address space.

Generally, the operating system prevents one process from accessing other process memory. This restriction is omitted in the shared memory method.





Producer – consumer problem.

In this problem, there are two processes: **producer** and **consumer**.

There is a **buffer** for storing items.

The producer process stores items into the buffer and the consumer process takes items from the buffer.

Two pointers (**in** and **out**) are maintained for the buffer.

The in pointer points to the next free slot in the buffer and out pointer points to the slot from which item can be taken out from the buffer.

A variable called '**count**' indicates the number of items in the buffer.

The buffer, **sizeofbuffer** and count must be stored in shared memory as these are accessed by both processes.

The producer process has to wait until a free slot is available in the buffer.

The consumer process has to wait until an item is available in the buffer.

The code of the producer process is

```
while (true)
{
    while(count==sizeofbuffer)
    ;
    buffer[in]=item;
    in=(in+1)%sizeofbuffer;
    count=count+1;
}
```

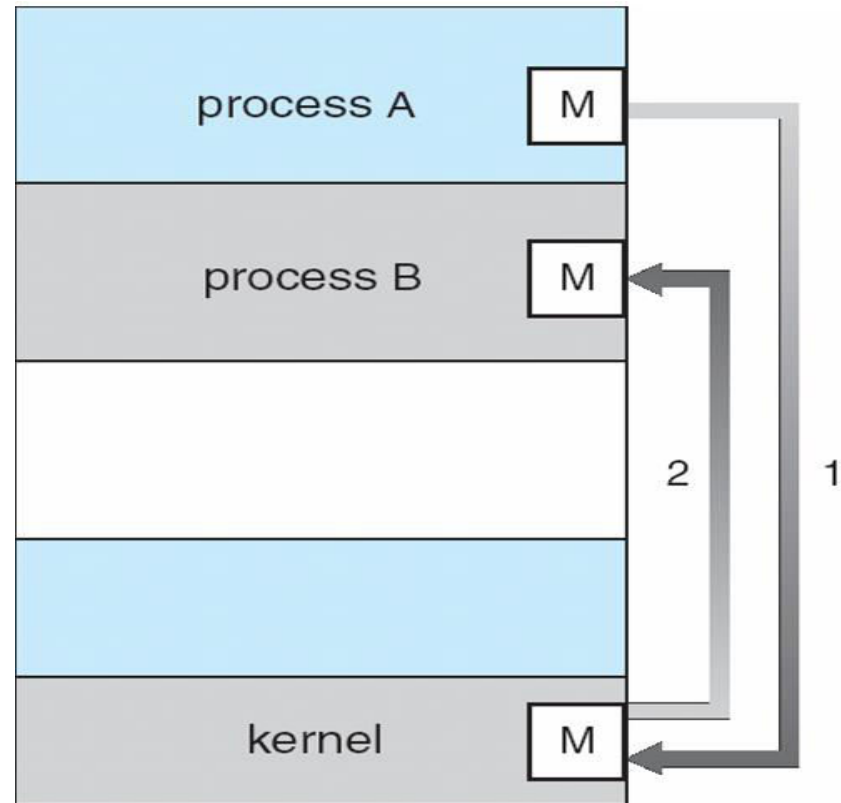
The code for consumer process is

```
while (true)
{
    while(count==0)
    ;
    item=buffer[out];
    out=(out+1)%sizeofbuffer;
    count=count-1;
}
```

Using shared memory, large amount of data can be exchanged between processes in less time.

## Message passing

With message passing, processes can communicate by exchanging short messages.



Two operations called **send(message)**, **receive(message)** and a **communication link** are used for exchanging the messages.

The following three issues need to be considered.

- 1) Naming
- 2) Synchronization
- 3) Buffering

### **1) Naming**

A direct communication link or an indirect communication link is used between the communicating processes.

#### **Direct Communication:**

With direct communication, the sender process has to mention the name of receiver process.

Similarly, the receiver process has to mention the name of sender process.

The syntax of send/ receive operations is

send(P, message) - sends a message to process P.

receive(Q, message) - receives a message from process Q.

In Direct Communication:

- 1) A communication link is established between every pair of processes that want to communicate.
- 2) A link is associated with exactly two processes.
- 3) There exists exactly one link between every pair of processes.

**Indirect communication:**

With this mechanism, the messages are sent to and received from mailboxes or ports.

The processes can communicate by sending and receiving messages from mailboxes.

The syntax of send/receive operations is

send(A, message) – sends a message to mailbox A.

receive(A, message) – receives a message from mailbox A.

In Indirect Communication:

- 1) A link is established between a pair of processes if both processes have a shared mailbox.
- 2) There exists more number of links between a pair of processes.

## 2) Synchronization

Communication between processes takes place through calls to send() and receive() operations.

Message passing is either **blocking** or **non blocking** – also known as synchronous and asynchronous.

**Blocking send:** the sending process is blocked until the message is received by the receiver or mailbox.

**Non blocking send:** the sending process continues its operations after sending the message.

**Blocking receive:** the receiver blocks until it receive the message.

**Non blocking receive:** the receiver does not block for the message to receive.

The sender and receiver can use any combination of send() and receive().



### 3) Buffering

In both direct and indirect communication, the messages are stored in a buffer.

Three types of buffers can be used

**Buffer with zero capacity:** in this case, the sender must block until the receiver receives the message.

**Buffer with bounded capacity:** the buffer has a finite length.

In this case, the sender can continue its execution after sending a message if the buffer is not full.

**Buffer with unbounded capacity:** the buffer has infinite capacity.

In this case, the sender never blocks.

If the processes running in different systems want to communicate then message passing is easier to implement than shared memory.

Message passing allows exchanging of small amounts of data.

It takes more time to exchange messages as messages are exchanged through kernel.

## **Multithreaded Programming**

A thread is a basic unit of CPU utilization.

A thread is part of a process.

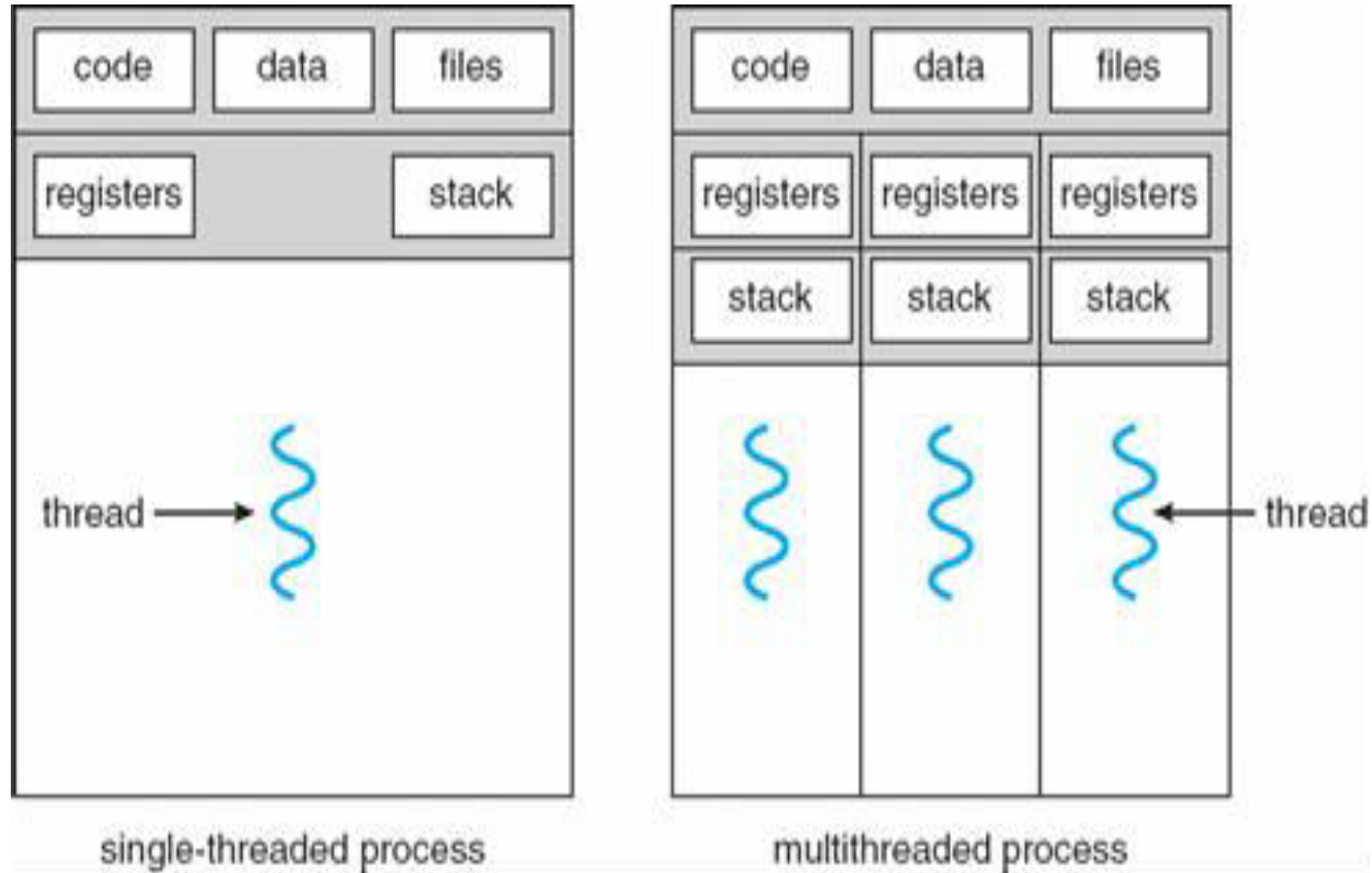
A single threaded process can perform only one task at a time.

A multithreaded process can perform number of tasks at a time.

A thread is associated with a program counter, a set of registers and a stack.

The threads belonging to a process shares the code, data and resources of the process.

The following figure shows the difference between a single threaded process and a multithreaded process.



Most of the software running on modern computer systems are multithreaded.

The threads of a process may be performing different tasks or same task.

For example, in a word processor, a thread displays graphics, a thread responds to key strokes and a thread checks spelling and grammar.

In a web server, there are number of threads one for each client request.

When a client makes a request, a thread is created by the server for processing that request.

Most operating system kernels are multithreaded.

Number of threads is running in the kernel.

Each thread implements a functionality of the operating system.

## **Benefits**

### **1) Responsiveness:**

In a single threaded process, if the thread blocks then the user does not get response.

In a multithreaded process, if one or more threads are blocked then the user gets response from other threads of the process.

Multithreading increases responsiveness to the user.

### **2) Resource sharing:**

In multithreading, all threads of a process share resources allocated to that process automatically.

### **3) Economy:**

Creating a process is costly.

The operating system has to allocate number of resources to a process when it creates a process.

When a thread is created no need to allocate resources for the thread as the thread shares resources of the process in which it is created.

Process switching is costly compared to thread switching.

### **4) Scalability:**

In a multiprocessor system, a single threaded process can use only one processor.

A multithreaded process can use all processors for running the threads in parallel.

## Multithreading models

Two types of threads can be found in the computer system: User threads and Kernel threads

**User threads** (threads created in an application program) are created and managed by user.

User level threads are fast.

Creation of user level threads and switching between user level threads is easy.

**Kernel threads** (threads created in kernel) are managed by the operating system.

Kernel level threads are slower compared to user level threads due to management overhead.

Creation of kernel level threads and switching between kernel level threads is costly and time consuming.

A kernel level thread provide services to one or more user level threads.

Multithreading models indicate the relationship between user and kernel threads.

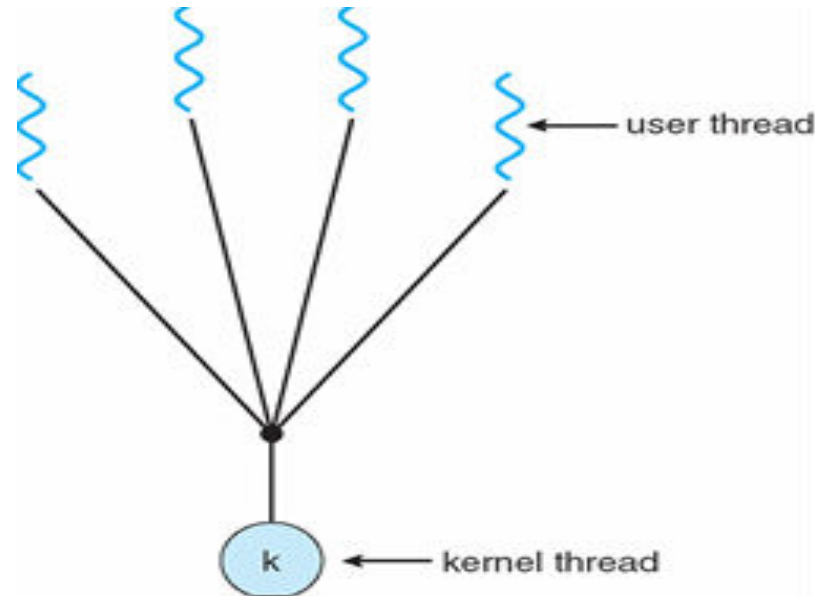
There are three types of models

- 1) Many-to-One model
- 2) One-to-One model
- 3) Many-to-Many model



## 1) Many-to-One model:

Number of user threads is mapped to one kernel thread as shown in following figure.



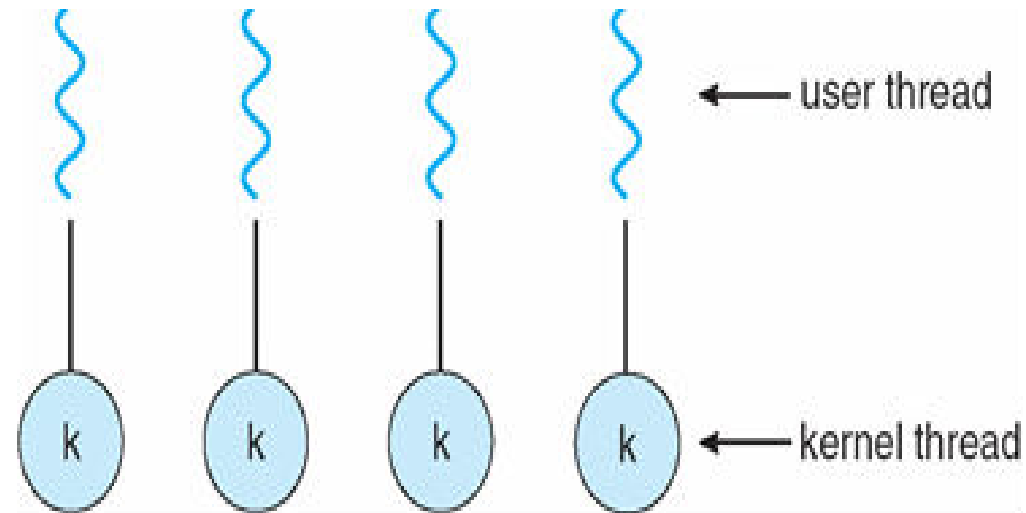
The kernel thread provide services to only one user thread at a time.

If the kernel thread is blocked then the entire user process is blocked.

It is not possible to run user threads in parallel.

## 2) One-to-One model:

Each user thread is mapped to a kernel thread as shown in following figure.



The user process is not blocked when one thread makes a blocking system call.

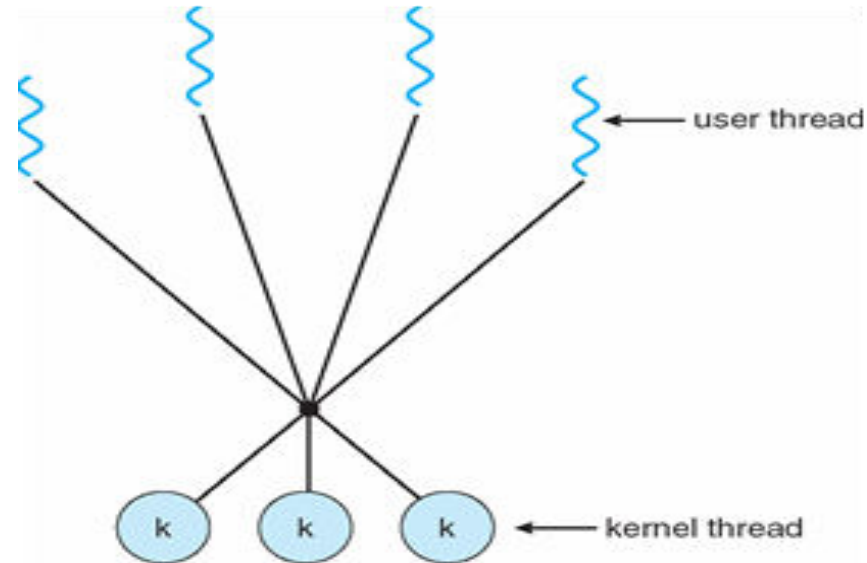
All user threads can run in parallel on multiple processors.

**A drawback with this method is:**

Creation of a user thread results in creation of a kernel thread.

### 3) Many-to-Many model:

The user threads are mapped to less or equal number of kernel threads as shown in following figure



The user threads can run in parallel on multiple processors.

The user can create any number of user threads.

When a thread makes a blocking system call, the kernel can schedule another thread for execution.

One variation of many-to-many model is **two-level model**.

In two-level model, user threads are mapped to less or equal number of kernel threads and a user thread can be bound to a kernel thread.

