Processes

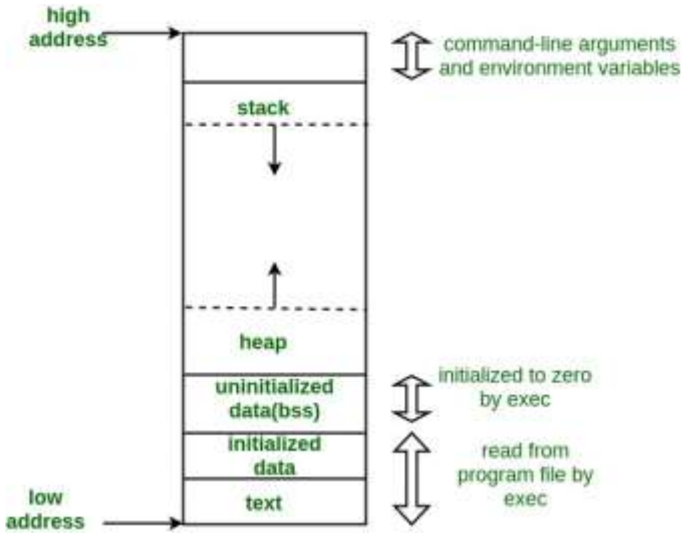
The term **“process”** was first used by the designers of the multics system in the 1960’s. A process is a program in execution and process execution must progress in sequential fashion.

Process exits in a limited span of time.

Two or more process could be executing the same program, each using their own data and resource.

The process memory is divided into four sections for efficient operation:

- The **text category** is composed of integrated program code, which is read from fixed storage when the program is launched.
- The **data class** is made up of global and static variables, distributed and executed before the main action.
- Heap is used for flexible, or dynamic memory allocation and is managed by calls to new, delete, malloc, free, etc.
- The stack is used for local variables. The space in the stack is reserved for local variables when it is announced.


high address
command-line arguments and environment variables
stack
heap
uninitialized data(bss)
initialized to zero by exec
initialized data
read from program file by exec
text
low address

Process State

When process executes, it changes state. Process state is defined as the current activity of the process. Process state contains five states. Each process is one of the following states.

new: The process is being created.

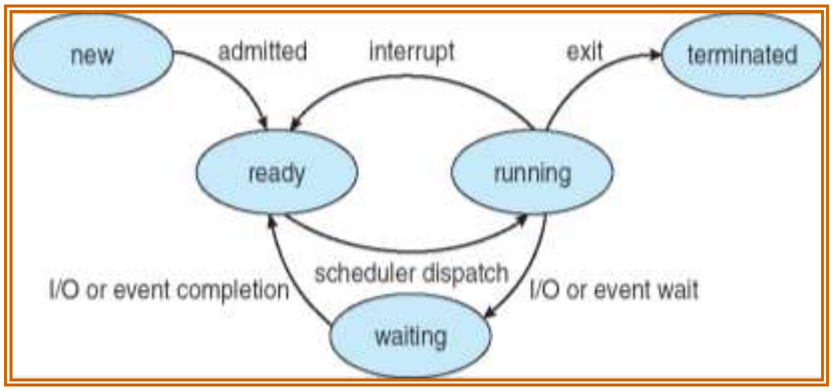
running: Instructions are being executed.

waiting: The process is waiting for some event to occur.

ready: The process is waiting to be assigned to a process.

terminated: The process has finished execution.

Diagram of Process State



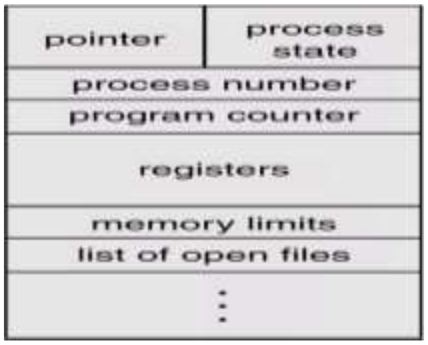
Dispatching

The assignment of the CPU to the first process on the ready list is called dispatching and is performed by a system entity called the Dispatcher.

Process Control Block (PCB)

The manifestation of a process in OS is a PCB or Process descriptor. Each process contains the PCB. PCB is a Data Structure containing certain important information about the process including,

- Unique identification of the process
- A pointer to the process's parent
- Process state
- Program counter
- CPU register
- Memory management information
- Account information



Pointer

Pointer points to the another PCB. Pointer is used for maintaining the scheduling list.

Process state

Process state may be new, ready, running, waiting and so on.

Program counter

It indicates the address of the next instruction to be executed.

CPU register

It includes general purpose register, stack pointer, and accumulators etc.

Memory management Informtion

locations including value of base and limit registers, page tables and other virtual memory information.

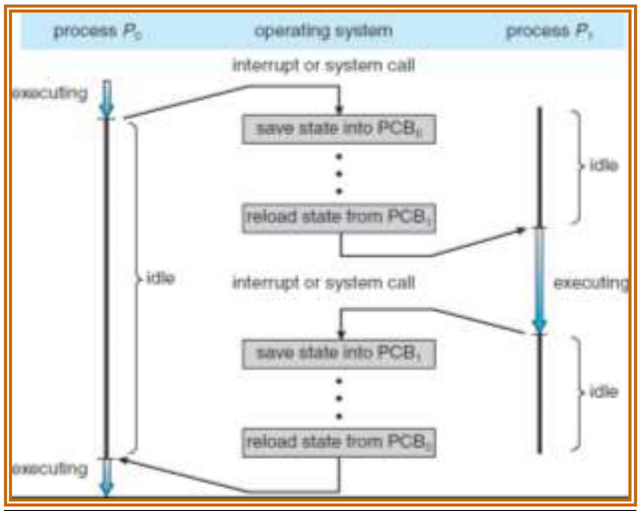
Accounting information

the amount of CPU and real time used, time limits, account numbers, job or process numbers etc.

I/O status information

List of I/O devices allocated to this process, a list of open files, and so on.

CPU Switch From Process to Process



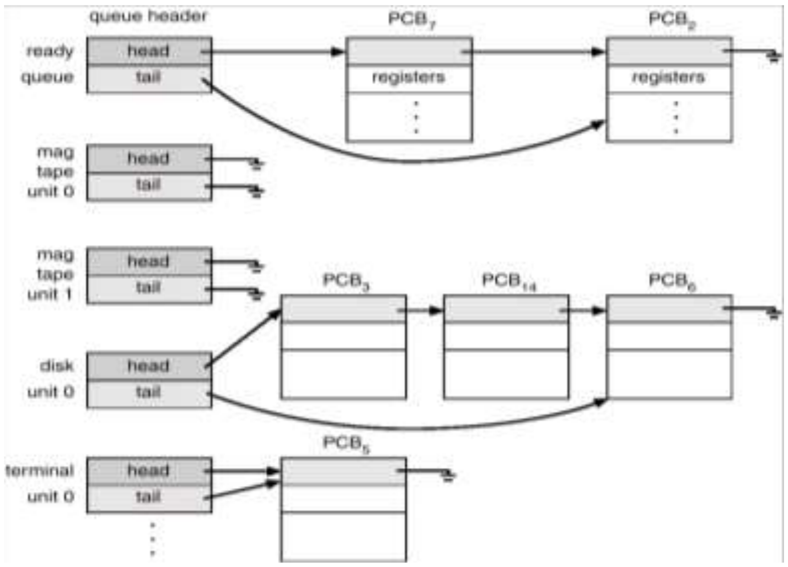
Process Scheduling Queues

Scheduling is to decide which process to execute and when. The objective of multi-program is, to have some process running at all times, so as to maximize CPU utilization. In Timesharing, switch the CPU frequently that users can interact with the program while it is running.

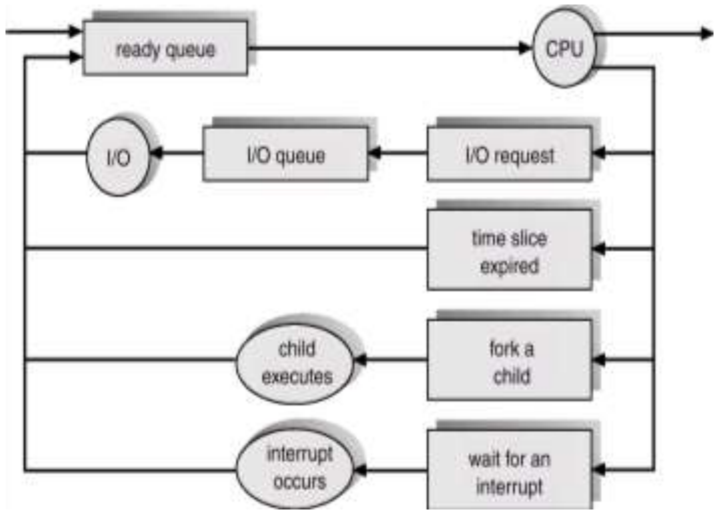
Scheduling Queues

1. **job queue** -The processes enter the system, they are put into a job queue. This Queue consists of all processes in the system.
2. **Ready queue** – set of all processes residing in main memory, and are ready and waiting to execute are kept on a list is called Ready Queue. This queue is generally stored as linked list. A ready queue header contains two pointers.(Head, Tail).
3. **Device queues** – set of processes waiting for an I/O device. Each device has its own queue.

Ready Queue and Various I/O Device Queues



Representation of Process Scheduling



A new process is initially put in the ready queue. It waits in the ready queue until it is selected for execution. Once a process is allocated CPU, the following events may occur.

- A process could issue an I/O request, and then be placed in an I/O queue.
- A process could create a new process
- The process could be removed forcibly from CPU, as a result of an interrupt and put back in the ready queue.
- When process terminates, it is removed from all queues.
- PCB and its other resources are de-allocated.

Schedulers

A process migrates between the various scheduling queues throughout its lifetime. The OS must select, for scheduling process. The selection process is carried out by a scheduler.

Long-term scheduler (or job scheduler)

- Selects which processes from this pool and loads them into memory for execution.
- It may take long time.
- The long-term scheduler executes less frequently.
- The long-term scheduler controls degree of multiprogramming.

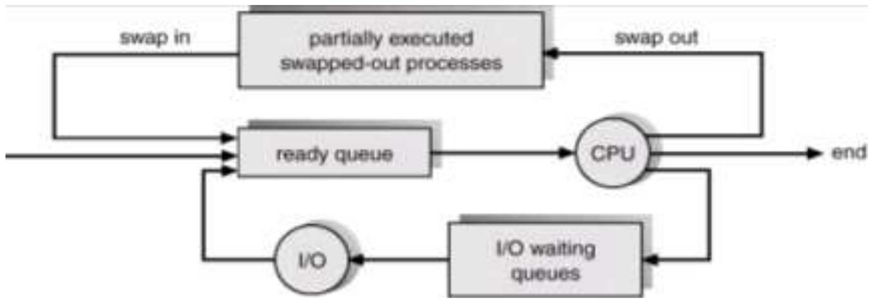
Short-term scheduler (or CPU scheduler)

- Selects which process should be ready to execute and allocates the CPU.
- The STS must select a new process for the CPU frequently.
- STS is executed at least once every 100 milliseconds.
- The STS must be fast.
- If it takes 10 milliseconds to decide to execute a process for 100 ms, then 9 % of CPU is used (or wasted) simply for scheduling work.

Medium-term scheduler

Some OS introduced a medium-term scheduler using swapping. It can be advantageous, to remove the processes from the memory and reduce the multiprogramming. At some later time, the process can be reintroduced into main memory and its execution can be continued when it left off. This scheme is called "**Swapping**".

Swapping improves the process mix (I/O and CPU), when main memory is unavailable.



- ❖ The long-term scheduler should make a careful selection. Because of the longer interval b/w executions, the LTS can afford to take more time to select a process for execution.
- ❖ The processes are either I/O bound or CPU bound.

- An I/O bound process spends more time doing I/O than it spends doing computation.
- A CPU bound process spends most of the time doing computation.
- The LT scheduler should select a good process mix of I/O-bound and CPU-bound processes.
- If all the processes are I/O bound, the ready queue will be empty.
- If all the processes are CPU bound, the I/O queue will be empty, the devices will go unused and the system will be unbalanced.
- Best performance by best combination of CPU-bound and I/O-bound process.

Context Switch

- Context switch is a task of switching the CPU to another process by saving the state of old process and loading the saved state for the new process.
- When a context switch occurs, the kernal saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.
- Context-switch time is overhead; the system does no useful work while switching.
- Context-switch time is highly dependent on hardware support.
- Typical range from 1 to 1000 microseconds.

Operations on Processes

The processes in the system can execute concurrently. The OS must provide a mechanism for creation and termination.

(i)Process creation

- A process may create several new processes.
- Processes are created and deleted dynamically.
- Process which creates another process is called a ***parent*** process; the created process is called a ***child*** process.
- Child process may create another sub process.
- Syntax for creating new process is: CREATE (process ID, attributes).
- A process needs certain resources (CPU time, memory, files, I/O devices) to accomplish its task.
- When a process creates a sub processes, that sub process may be to obtain its resource directly from the OS, or it may be constrained to a subset of resources of the parent process.
- When a process creates a new process, two possibilities in terms of **execution and resource sharing.**

  <u>Resource sharing possibilities</u>
  - Parent and children share all resources.
  - Children share subset of parent's resources.
  - Parent and child share no resources.

  <u>Execution possibilities</u>
  - Parent and children execute concurrently.
  - Parent waits until children terminate.
- There are also two possibilities in terms of the **address space** of the new process:
  - The child process is a duplicate of the parent process.
  - Child process has a program loaded into it.

Example

In UNIX:

- Each process is identified by its process identifier.
- **fork** system call creates new process.
- **exec** system call used after a **fork** to replace the process' memory space with a new program.
- The new process is a copy of the original process.
- The exec system call is used after a fork by one of the two processes to replace the process memory space with a new program.

DEC VMS:

- Creates a new process, loads a specified program into that process, and starts it running.

WINDOWS NT supports both models:

- Parent address space can be duplicated or
- parent can specify the name of a program for the OS to load into the address space of the new process.

(ii)Process Termination

- ❖ Process executes last statement and asks the operating system to decide it (**exit**).
- ❖ Output data from child to parent (via **wait**).
- ❖ Process' resources are de-allocated by operating system.
- ❖ Parent may terminate the execution of children processes (**abort**).
- ❖ Child has exceeded allocated resources.
- ❖ Task assigned to child is no longer required.
- ❖ Parent is exiting, Operating system does not allow child to continue if its parent terminates.
- ❖ Cascading termination. (All children terminated).

COOPERATING PROCESSES

- The concurrent process executing in the OS may be either independent process or cooperating process.
- Independent process **cannot** affect or be affected by the execution of another process.
- Cooperating process **can** affect or be affected by the execution of another process.

Advantages of process cooperation

1. **Information sharing:** several users may be interest in the same piece of information.
2. **Computation speed-up:** If we want a particular task to run faster, we must break it into subtasks and run in parallel.
3. **Modularity:** Constructing the system in modular fashion, dividing the system functions into separate process.
4. **Convenience:** User will have many tasks to work in parallel (Editing, compiling, printing).

Processes can communicate with each other through both:

- Shared Memory
- Message passing

The following figure shows a basic structure of communication between processes via the shared memory method and via the message passing method.

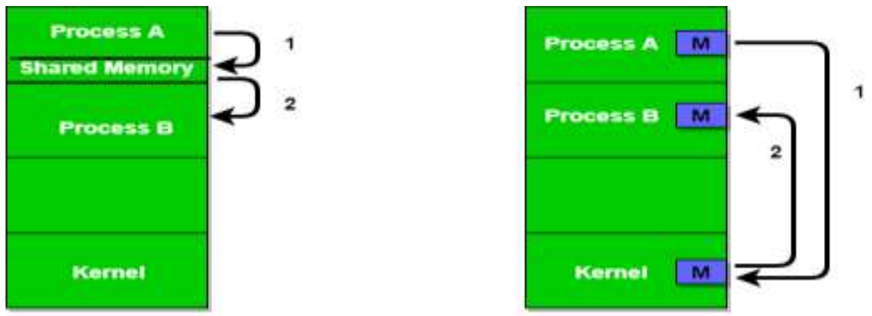


Figure 1 - Shared Memory and Message Passing

(i) Shared Memory

Communication between processes using shared memory requires processes to share some variable, and it completely depends on how the programmer will implement it.

One way of communication using shared memory can be imagined like this: Suppose process1 and process2 are executing simultaneously, and they share some resources or use some information from another process. Process1 generates information about certain computations or resources being used and keeps it as a record in shared memory. When process2 needs to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly.

Processes can use shared memory for extracting information as a record from another process as well as for delivering any specific information to other processes.

Ex: Producer-Consumer problem

A producer process produces information that is consumed by a consumer process. For example, a print program produces characters that are consumed by the printer driver.

A producer can produce one item while the consumer is consuming another item. The Producer and Consumer must be synchronized. The consumer does not try to consume an item, the consumer must wait until an item is produced.

Unbounded-Buffer

- no practical limit on the size of the buffer.
- Producer can produce any number of items.
- Consumer may have to wait

Bounded-Buffer

- assumes that there is a fixed buffer size.

Bounded-Buffer – Shared-Memory Solution:

Shared data

```
#define BUFFER_SIZE 10
Typedef struct
{
. . .
} item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

Bounded-Buffer – Producer Process:

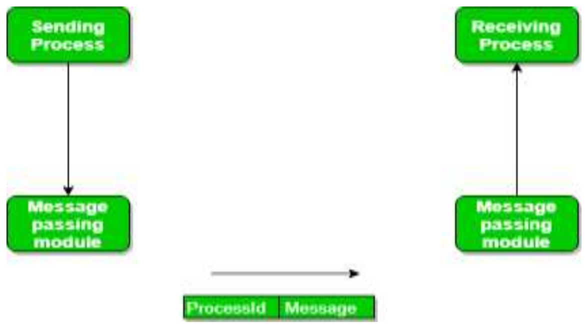
```
item next Produced;
while (1)
{
 while (((in + 1) % BUFFER_SIZE) == out);       /* do nothing */
 buffer[in] = nextProduced;
 in = (in + 1) % BUFFER_SIZE;
}
```

Bounded-Buffer – Consumer Process:

```
item next Consumed;
while (1)
{
 while (in == out);                             /* do nothing */
 next Consumed = buffer[out];
 out = (out + 1) % BUFFER_SIZE;
}
```

(ii) Messaging Passing Method

In this method, processes communicate with each other without using any kind of shared memory. If two processes want to communicate with each other, they proceed as follows



:

- Establish a communication link (if a link already exists, no need to establish it again.)
- Start exchanging messages using basic primitives.
- The message size can be of fixed size or of variable size. If it is of fixed size, it is easy for an OS designer but complicated for a programmer and if it is of variable size then it is easy for a programmer but complicated for the OS designer.

- Cooperating process to communicate with each other via an inter process communication (IPC).
- IPC provides a Mechanism to allow processes to communicate and to synchronize their actions.
- If P and Q want to communicate, a communication link exists between them and exchange messages via send/receive. OS provides this facility.
- IPC facility provides two operations:

  Send (*message*) – message size fixed or variable.
  Receive (*message*)
- Implementation of communication link by following.
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

Methods for logical implementation of a link

i. Direct communication.
ii. Indirect communication.

Direct Communication

- Each processes must name each other explicitly:
  - Send (*P, message*) – send a message to process P.
  - Receive (*Q, message*) – receive a message from process Q.
- Links are established automatically.
- A link is associated with exactly one pair of communicating processes.
- Between each pair there exists exactly one link.
- The link may be unidirectional, but is usually bi-directional.
- This exhibits both symmetry and asymmetry in addressing

  Symmetry:
  Both the sender and the receiver processes must name the other to communicate.

  Asymmetry:
  Only sender names the recipient, the recipient is not required to name the sender. The send and receive primitives are as follows.
  - Send (P, message)– send a message to process P.
  - Receive (id, message)– receive a message from any process.

Disadvantage of direct communication

Changing a name of the process creates problems.

Indirect Communication

- The messages are sent and received from mailboxes (also referred to as ports).
- A mailbox is an object
- Process can place messages.
- Process can remove messages.
- Two processes can communicate only if they have a shared mailbox.

- Primitives are defined as:
  send (*A*, *message*) – send a message to mailbox A
  receive (*A*, *message*) – receive a message from mailbox A.
- A mailbox may be owned either by a process or by the OS.
- If the mailbox is owned by a process, then we distinguish b/w the owner (who can only receive msg through this mailbox) and the user (who can only send msg to the mailbox).
- A mailbox may be owned by the OS is independent and provide a mechanism,
  - create a mailbox
  - receive messages through mailbox
  - destroy a mail box.

Mailbox sharing problem

The processes P1, *P2*, and *P3* all share mailbox A. *Processes P1*, sends; *P2* and *P3* receive the message from A. Who gets a message?

Solutions

- Allow a link to be associated with at most two processes.
- Allow only one process at a time to execute a receive operation.
- Allow the system to select arbitrarily the receiver. The system may identify the receiver to the sender.

Synchronization

- Message passing may be either blocking or non-blocking.
- **Blocking** is considered **synchronous. Non-blocking** is considered **asynchronous.**
- **send** and **receive** primitives may be either blocking or non-blocking.

Blocking send

The sending process is blocked until the message is received by the receiving process or by the mailbox.

Non-blocking send

The sending process sends the message and resumes operation.

Blocking receive

The receiver blocks until a message is available.

Non-blocking receive

The receiver receives either a valid message or a null.

Buffering

- A link has some capacity that determines the number of messages that can reside in it temporarily.
- Queue of messages is attached to the link; implemented in one of three ways.

Zero capacity

- The link cannot have any messages in it.
- Sender must wait for receiver.

Bounded capacity

- finite length of *n* messages
- Sender must wait if link full.

Unbounded capacity

- infinite length
- Sender never waits.

CPU scheduling

CPU scheduling is the process of deciding which process will own the CPU to use while another process is suspended. The main function of the CPU scheduling is to ensure that whenever the CPU remains idle, the OS has at least selected one of the processes available in the ready-to-use line.

In Multiprogramming, if the long-term scheduler selects multiple I / O binding processes then most of the time, the CPU remains an idle. The function of an effective program is to improve resource utilization.

If most operating systems change their status from performance to waiting then there may always be a chance of failure in the system. So in order to minimize this excess, the OS needs to schedule tasks in order to make full use of the CPU and avoid the possibility of deadlock.

Objectives of Process Scheduling Algorithm

- Utilization of CPU at maximum level. Keep CPU as busy as possible.
- Allocation of CPU should be fair.
- Throughput should be Maximum. i.e. Number of processes that complete their execution per time unit should be maximized.
- Minimum turnaround time, i.e. time taken by a process to finish execution should be the least.
- There should be a minimum waiting time and the process should not starve in the ready queue.
- Minimum response time. It means that the time when a process produces the first response should be as less as possible.

Terminologies

- **Arrival Time:** Time at which the process arrives in the ready queue.
- **Completion Time:** Time at which process completes its execution.
- **Burst Time:** Time required by a process for CPU execution.
- **Turn Around Time:** Time Difference between completion time and arrival time.

  Turn Around Time = Completion Time – Arrival Time
- **Waiting Time(W.T):** Time Difference between turn around time and burst time.

  Waiting Time = Turn Around Time – Burst Time

THE SCHEDULING CRITERIA

CPU utilization:

The main purpose of any CPU algorithm is to keep the CPU as busy as possible. Theoretically, CPU usage can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the system load.

Throughput:

The average CPU performance is the number of processes performed and completed during each unit. This is called throughput. The output may vary depending on the length or duration of the processes.

Turn round Time:

For a particular process, the important conditions are how long it takes to perform that process. The time elapsed from the time of process delivery to the time of completion is known as the conversion time. Conversion time is the amount of time spent waiting for memory access, waiting in line, using CPU, and waiting for I / O.

Waiting Time:

The Scheduling algorithm does not affect the time required to complete the process once it has started performing. It only affects the waiting time of the process i.e. the time spent in the waiting process in the ready queue.

Response Time:

In a collaborative system, turn around time is not the best option. The process may produce something early and continue to computing the new results while the previous results are released to the user. Therefore another method is the time taken in the submission of the application process until the first response is issued. This measure is called response time.

Types of CPU Scheduling Algorithms

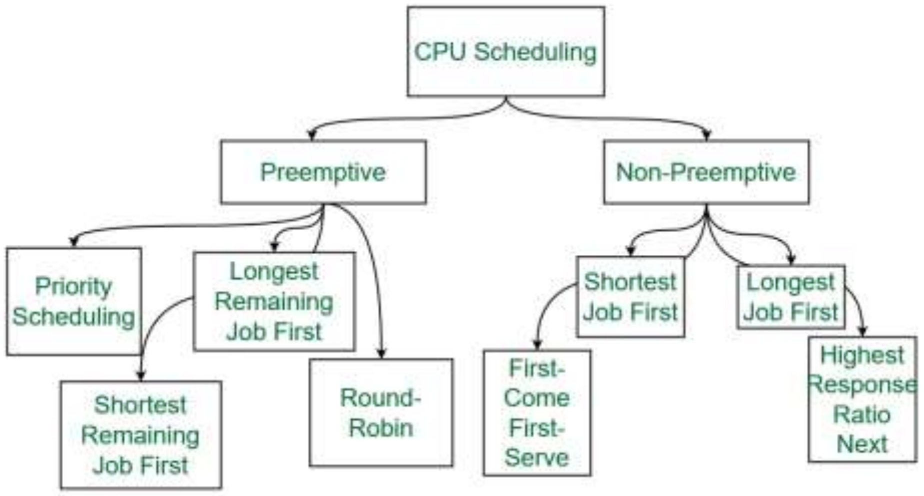
There are mainly two types of scheduling methods:

<u>Preemptive Scheduling</u>:

Preemptive scheduling is used when a process switches from running state to ready state or from the waiting state to the ready state.

<u>Non-Preemptive Scheduling</u>:

Non-Preemptive scheduling is used when a process terminates , or when a process switches from running state to waiting state.



1. First Come First Serve Scheduling:

FCFS considered to be the simplest of all operating system scheduling algorithms. First come first serve scheduling algorithm states that the process that requests the CPU first is allocated the CPU first and is implemented by using FIFO queue.

Characteristics:

- ➢ FCFS supports non-preemptive and preemptive CPU scheduling algorithms.
- ➢ Tasks are always executed on a First-come, First-serve concept.
- ➢ FCFS is easy to implement and use.
- ➢ This algorithm is not much efficient in performance, and the wait time is quite high.

Advantages:

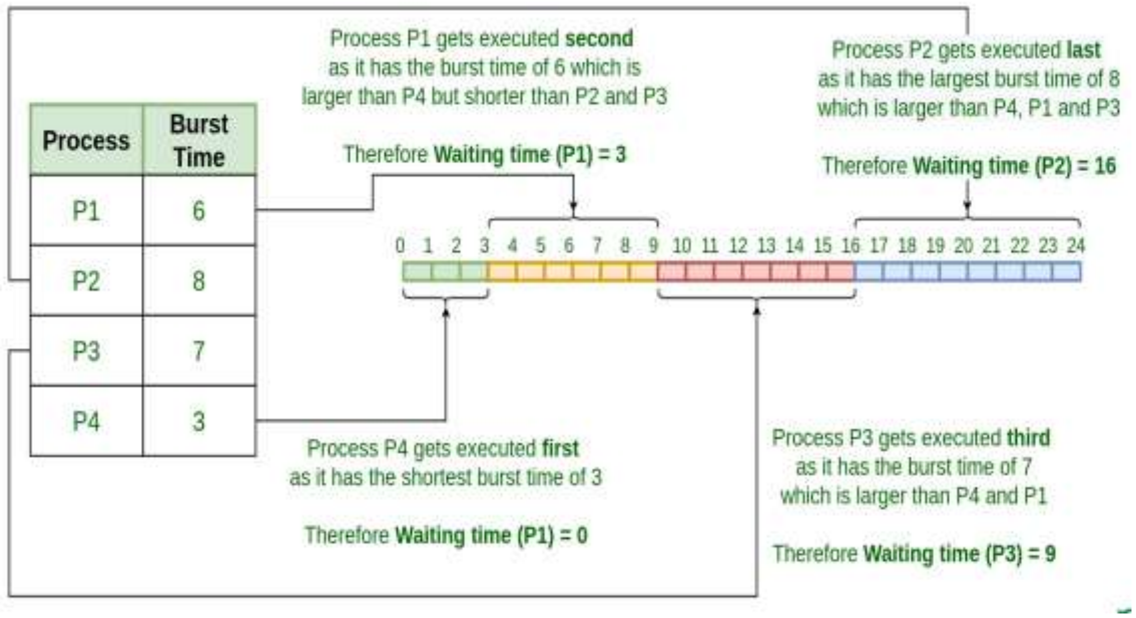
- ➢ Easy to implement
- ➢ First come, first serve method

Disadvantages:

- ➢ FCFS suffers from **Convoy effect**.
- ➢ The average waiting time is much higher than the other algorithms.
- ➢ FCFS is very simple and easy to implement and hence not much efficient.

2. Shortest Job First(SJF) Scheduling:

Shortest job first (SJF) is a scheduling process that selects the waiting process with the smallest execution time to execute next. This scheduling method may or may not be preemptive. Significantly reduces the average waiting time for other processes waiting to be executed. The full form of SJF is Shortest Job First.


Process P1 gets executed **second** as it has the burst time of 6 which is larger than P4 but shorter than P2 and P3
Therefore **Waiting time (P1) = 3**
Process P2 gets executed **last** as it has the largest burst time of 8 which is larger than P4, P1 and P3
Therefore **Waiting time (P2) = 16**
Process | Burst Time
P1 6
P2 8
P3 7
P4 3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Process P4 gets executed **first** as it has the shortest burst time of 3
Therefore **Waiting time (P1) = 0**
Process P3 gets executed **third** as it has the burst time of 7 which is larger than P4 and P1
Therefore **Waiting time (P3) = 9**

Characteristics:

- ➢ Shortest Job first has the advantage of having a minimum average waiting time among all operating system scheduling algorithms.
- ➢ It is associated with each task as a unit of time to complete.
- ➢ It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing.

Advantages:

- ➢ As SJF reduces the average waiting time thus, it is better than the first come first serve scheduling algorithm.
- ➢ SJF is generally used for long term scheduling

Disadvantages:

- ➢ One of the demerit SJF has is starvation.
- ➢ Many times it becomes complicated to predict the length of the upcoming CPU request

3. Longest Job First(LJF) Scheduling:

This is just opposite of shortest job first (SJF), as the name suggests this algorithm is based upon the fact that the process with the largest burst time is processed first. Longest Job First is non-preemptive in nature.

Characteristics:

- Among all the processes waiting in a waiting queue, CPU is always assigned to the process having largest burst time.
- If two processes have the same burst time then the tie is broken using FCFS i.e. the process that arrived first is processed first.
- LJF CPU Scheduling can be of both preemptive and non-preemptive types.

Advantages:

- No other task can schedule until the longest job or process executes completely.
- All the jobs or processes finish at the same time approximately.

Disadvantages:

- Generally, the LJF algorithm gives a very high average waiting time and average turn-around time for a given set of processes.
- This may lead to convoy effect.

4. Priority Scheduling:

Preemptive Priority CPU Scheduling Algorithm is a pre-emptive method of CPU scheduling algorithm that works **based on the priority** of a process. In this algorithm, the editor sets the functions to be as important, meaning that the most important process must be done first. In the case of any conflict, that is, where there are more than one processor with equal value, then the most important CPU planning algorithm works on the basis of the FCFS

Characteristics:

- Schedules tasks based on priority.
- When the higher priority work arrives while a task with less priority is executed, the higher priority work takes the place of the less priority one and
- The latter is suspended until the execution is complete.
- Lower is the number assigned, higher is the priority level of a process.

Advantages:

- The average waiting time is less than FCFS
- Less complex

Disadvantages:

- One of the most common demerits of the Preemptive priority CPU scheduling algorithm is the Starvation Problem. This is the problem in which a process has to wait for a longer amount of time to get scheduled into the CPU. This condition is called the starvation problem.

5. Round Robin Scheduling:

Round Robin is a CPU scheduling algorithm where each process is cyclically assigned a fixed time slot. It is the preemptive version of First come First Serve CPU Scheduling algorithm. Round Robin CPU Algorithm generally focuses on Time Sharing technique.

Characteristics:

- It’s simple, easy to use, and starvation-free as all processes get the balanced CPU allocation.
- One of the most widely used methods in CPU scheduling as a core.
- It is considered preemptive as the processes are given to the CPU for a very limited time.

Advantages:

- Round robin seems to be fair as every process gets an equal share of CPU.
- The newly created process is added to the end of the ready queue.

6. Shortest Remaining Time First Scheduling (SRTF):

SRTF is the preemptive version of the Shortest job first which we have discussed earlier where the processor is allocated to the job closest to completion. In SRTF the process with the smallest amount of time remaining until completion is selected to execute.

Characteristics:

- SRTF algorithm makes the processing of the jobs faster than SJF algorithm, given it’s overhead charges are not counted.
- The context switch is done a lot more times in SRTF than in SJF and consumes the CPU’s valuable time for processing. This adds up to its processing time and diminishes its advantage of fast processing.

Advantages:

- In SRTF the short processes are handled very fast.
- The system also requires very little overhead since it only makes a decision when a process completes or a new process is added.

Disadvantages:

- Like the shortest job first, it also has the potential for process starvation.
- Long processes may be held off indefinitely if short processes are continually added.

7. Longest Remaining Time First:

The longest remaining time first is a preemptive version of the longest job first scheduling algorithm. This scheduling algorithm is used by the operating system to program incoming processes for use in a systematic way. This algorithm schedules those processes first which have the longest processing time remaining for completion.

Characteristics:

- Among all the processes waiting in a waiting queue, the CPU is always assigned to the process having the largest burst time.
- If two processes have the same burst time then the tie is broken using FCFS i.e. the process that arrived first is processed first.
- LJF CPU Scheduling can be of both preemptive and non-preemptive types.

Advantages:

- No other process can execute until the longest task executes completely.
- All the jobs or processes finish at the same time approximately.

Disadvantages:

- This algorithm gives a very high average waiting time and average turn-around time for a given set of processes.
- This may lead to a convoy effect.

8. Highest Response Ratio Next:

Highest Response Ratio Next is a non-preemptive CPU Scheduling algorithm and it is considered as one of the most optimal scheduling algorithms. The name itself states that we need to find the response ratio of all available processes and select the one with the highest Response Ratio. A process once selected will run till completion.

Characteristics:

- The **criteria** for HRRN is **Response Ratio** and the **mode** is **Non Preemptive.**
- HRRN is considered as the modification of Shortest Job First to reduce the problem of starvation.

- In comparison with SJF, during the HRRN scheduling algorithm, the CPU is allotted to the next process which has the **highest response ratio** and not to the process having less burst time.

  Response Ratio = (W + S)/S

  Here, **W** - Waiting time of the process

  S - Burst time of the process.

Advantages:

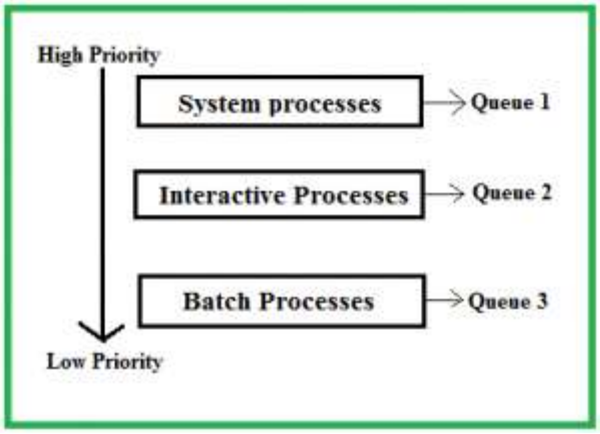
- HRRN Scheduling algorithm generally gives better performance than the shortest job first Scheduling.
- There is a reduction in waiting time for longer jobs and also it encourages shorter jobs.

Disadvantages:

- The implementation of HRRN scheduling is not possible as it is not possible to know the burst time of every job in advance.
- In this scheduling, there may occur an overload on the CPU.

9. Multiple Queue Scheduling:

Processes in the ready queue can be divided into different classes where each class has its own scheduling needs. For example, a common division is a **foreground (interactive)** process and a **background (batch)** process. These two classes have different scheduling needs. For this kind of situation **Multilevel Queue Scheduling** is used.



The description of the processes in the above diagram is as follows:

- **System Processes:** The CPU itself has its process to run, generally termed as System Process.
- **Interactive Processes:** An Interactive Process is a type of process in which there should be the same type of interaction.

- **Batch Processes:** Batch processing is generally a technique in the Operating system that collects the programs and data together in the form of a **batch** before the **processing** starts.

Advantages:

- The main merit of the multilevel queue is that it has a low scheduling overhead.

Disadvantages:

- Starvation problem
- It is inflexible in nature

10. Multilevel Feedback Queue Scheduling:

Multilevel Feedback Queue Scheduling (MLFQ) CPU Scheduling is like **Multilevel Queue Scheduling** but in this process can move between the queues. And thus, much more efficient than multilevel queue scheduling.

Characteristics:

- In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system, and processes are not allowed to move between queues.
- As the processes are permanently assigned to the queue, this setup has the advantage of low scheduling overhead,
- But on the other hand disadvantage of being inflexible.

Advantages:

- It is more flexible
- It allows different processes to move between different queues

Disadvantages:

- It also produces CPU overheads
- It is the most complex algorithm.

Comparison between various CPU Scheduling algorithms

Here is a brief comparison between different CPU scheduling algorithms:

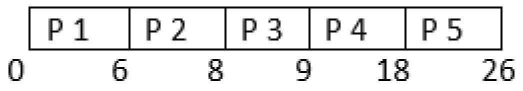
Algorithm	**Allocation is**	**Complexity**	**Average waiting time (AWT)**	**Preemption**	**Starvation**	**Performance**
FCFS	According to the arrival time of the processes, the CPU is allocated.	Simple and easy to implement	Large.	No	No	Slow
SJF	Based on the lowest CPU burst time (BT).	More complex than FCFS	Smaller than FCFS	No	Yes	Good
SRTF	Same as SJF the allocation of the CPU is based on the lowest CPU burst time (BT). But it is preemptive.	More complex than FCFS	Depending on arrival time, process size	Yes	Yes	Good
RR	According to the order of the process arrives with fixed time quantum (TQ)	The complexity depends on TQ	Large than SJF and Priority scheduling.	Yes	No	Fair
Priority Pre-emptive	According to the priority. The bigger priority task executes first	Less complex	Smaller than FCFS	Yes	Yes	Well
Priority non-preemptive	According to the priority with monitoring the new incoming higher priority jobs	Less complex than Priority preemptive	Smaller than FCFS	No	Yes	Most beneficial with batch systems

Algorithm	**Allocation is**	**Complexity**	**Average waiting time (AWT)**	**Pre emp tion**	**Star vatio n**	**Performa nce**
MLQ	According to the process that resides in the bigger queue priority	More complex than the priority	Smaller than FCFS	No	Yes	Good
MLFQ	According to the process of a bigger priority queue.	It is the most Complex	Smaller than all scheduling	No	No	Good

Example 1 (FCFS)

1. Process ID	Process Name	Burst Time (ms)
P 1	A	6
P 2	B	2
P 3	C	1
P 4	D	9
P 5	E	8

Gantt Chart



Process ID	**Arrival Time (ms)**	**Burst Time (ms)**	**Completion Time (ms)**	**Turn Around Time (ms)**	**Waiting Time (ns)**
P 1	0	6	6	6	0
P 2	2	2	8	8	6
P 3	3	1	9	9	8
P4	4	9	18	18	9
P 5	5	8	26	26	18

Average Turn Around Time = (6 + 8 + 9 +18 +26) / 5 = 67 / 5 = 13.4 ms
Average Waiting Time = (0 + 6 + 8 + 9 + 18) / 5 = 41 / 5 = 8.2 ms

Example 2 (FCFS)

ProcessID	Process Name	Burst Time
P 1	A	79
P 2	B	2
P 3	C	3
P 4	D	1
P 5	E	25
P 6	F	3

Process Id	**Burst Time (BT)**	**Completion Time (CT)**	**Turn Around Time (TAT)**	**Waiting Time (WT)**
P 1	79	79	79	0
P 2	2	81	81	79
P 3	3	84	84	81
P 4	1	85	85	84
P 5	25	110	110	85
P 6	3	113	113	110

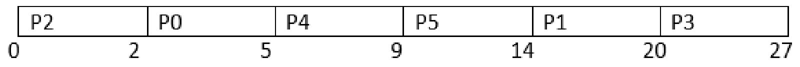
Avg Waiting Time = (0 + 79 + 81 + 84 + 85 + 110) /6 = 73.17 ms
Avg Turn Around Time **=** (79 + 81 + 84 + 85 + 110 +113) / 6 = 92 ms

Example 3 (SJF)

Process ID	Arrival Time	Burst Time
P0	1	3
P1	2	6
P2	1	2
P3	3	7
P4	2	4
P5	5	5

Non Pre-Emptive Shortest Job First CPU Scheduling

Gantt Chart:

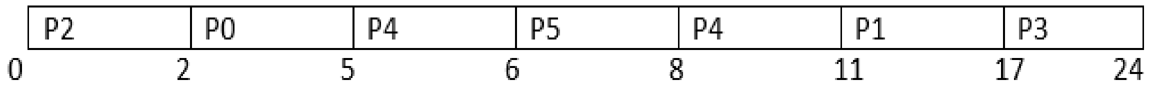


Process ID	**Arrival Time**	**Burst Time**	**Completion Time**	**Turn Around Time TAT=CT–AT**	**Waiting Time WT=CT–BT**
P0	1	3	5	4	1
P1	2	6	20	18	12
P2	0	2	2	2	0
P3	3	7	27	24	17
P4	2	4	9	7	4
P5	5	5	14	10	5

Average Waiting Time = (1 + 12 + 17 + 0 + 5 + 4) / 6 = 39 / 6 = 6.5 ms
Average Turn Around Time = (4 +18 + 2 +24 + 7 + 10) / 6 = 65 / 6 = 10.83 ms

Pre Emptive Shortest Job First CPU Scheduling

Gantt chart:



Process ID	**Arrival Time**	**Burst Time**	**Completion Time**	**Turn Around Time TAT=CT-AT**	**Waiting Time WT=CT–BT**
P0	1	3	5	4	1
P1	2	6	17	15	9
P2	0	2	2	2	0
P3	3	7	24	21	14
P4	2	4	11	9	5
P5	6	2	8	2	0

Average Turn Around Time = (4 +15 + 2 + 21 + 9 + 2) / 6 = 53 / 6 = 8.83 ms
Average Waiting Time = (1 + 9 + 0 + 14 + 5 + 0) /6 = 29 / 6 = 4.83 ms

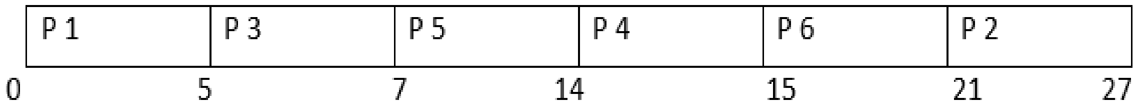
Example 4 (PRIORITY)

S. No	Process ID	Arrival Time	Burst Time	Priority
1	P 1	0	5	5
2	P 2	1	6	4
3	P 3	2	2	0
4	P 4	3	1	2
5	P 5	4	7	1
6	P 6	4	6	3

(5 has the least priority and 0 has the highest priority)

Solution:

Gantt Chart:



Process Id	**Arrival Time**	**Burst Time**	**Priority**	**Completion Time**	**Turn Around Time TAT=CT-AT**	**Waiting Time WT=TAT-BT**
P 1	0	5	5	5	5	0
P 2	1	6	4	27	26	20
P 3	2	2	0	7	5	3
P 4	3	1	2	15	12	11
P 5	4	7	1	14	10	3
P 6	4	6	3	21	17	11

Avg Waiting Time = (0 + 20 + 3 + 11 + 3 + 11) / 6 = 48 / 6 = 8 ms
Avg Turn Around Time = (5 + 26 + 5 + 11 + 10 + 17) / 6 = 74 / 6 = 12.33 ms

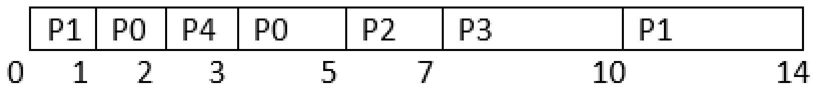
Example 5 (Round Robin)

Time Quantum = 1 ms

Process ID	Arrival Time	Burst Time
P0	1	3
P1	0	5
P2	3	2
P3	4	3
P4	2	1

Solution:

Gantt Chart:



Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time
P0	1	3	5	4	1
P1	0	5	14	14	9
P2	3	2	7	4	2
P3	4	3	10	6	3
P4	2	1	3	1	0

Avg Turn Around Time = (4+14+4+6+1) / 5 = 5.8 ms
Avg Waiting Time = (1+9+2+3+0) / 5 = 3 ms