2 Process Management

2.1 PROCESSES

Basics of process:

- A program under execution is known as a process.
- A process is different from a program/text section as a process also includes the current activity as depicted by the program counter and contents of the CPU.
- A process also contains a process stack, which holds temporary data like function parameters, return address and local variables, and heap memory allocated to the process dynamically during run time.
- It contains a data section/global section which has global variables.

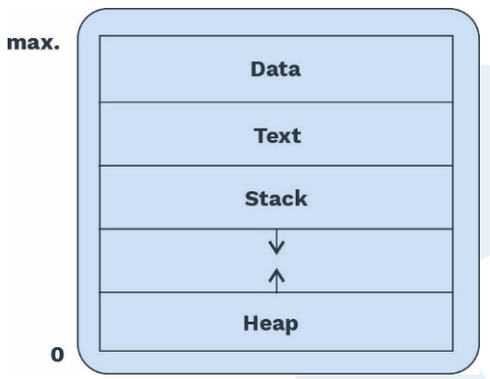


Fig. 2.1 Process Image in Memory

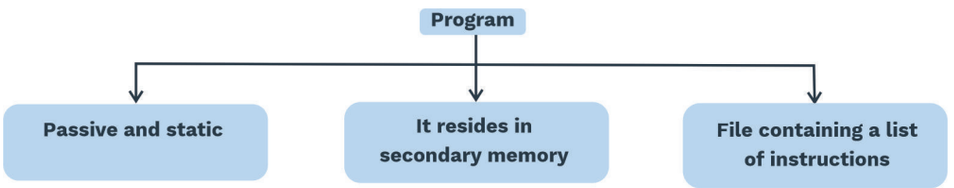


Fig. 2.2 Program

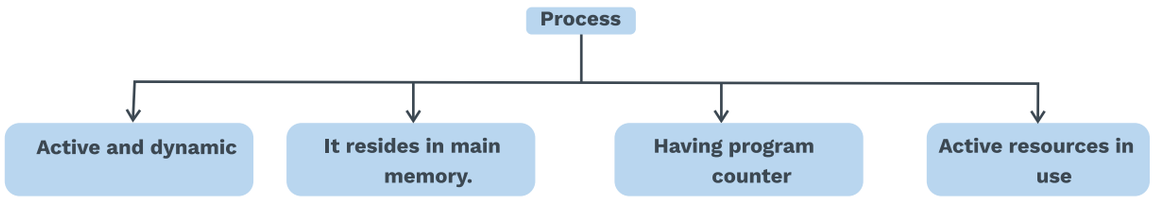


Fig. 2.3 Process

Relationship between process and program:

- **One-one:**

A single execution of a sequential program.

- **Many to one:**

Simultaneous execution of processes of a program.
The process is visualised as Abstract Data Structure (ADT).

Abstract data structure has:

- Definition
- Structure/Implementation
- Operations
- Attributes

Operations on process:

Process execution is a complex function which involves the following operations:

1) Creation of process:

A new process is constructed by the parent process or system for execution.

- When we start the computer system, the user can demand for the creation of a new process.
- This new process can be created by a process itself while executing.

2) Scheduling:

In this event, the state of a process is changed from ready to running or from new to ready or from ready to blocked, etc.

3) Blocking:

Whenever an input/output system call gets invoked by a process, the operating system blocks the process. In block mode, the process completes I/O operations or waits for an event.

4) Preemption:

A process executing in its allotted time is preempted by other processes waiting for execution.

5) Termination:

The event of completing the process is termed process termination.
In other words, it is the deallocation of computer resources allocated to it.
Some scenarios where process termination happens:

- When the execution of the process is completed, indicating that it has finished.
- When there are service errors present in the process, the operating system will terminate the process.
- Any problem in hardware will also lead to the termination of the process.
- Sometimes a process may get terminated by some other process.

Process attributes:
Identification related: Process ID, group ID, parent process ID, etc.
CPU related: Program counter, general-purpose registers, priority, states, etc.
Memory related: Size, memory limit, etc.
File related: List of open files, etc.
Device related: List of connected devices.
Protection: Access rights.

Note:

- All attributes of the process are kept in PCB (Process Control Block).
- Every process has its own PCB.
- PCB is used to represent a process in the Operating System.
- PCB is a kernel data structure.

Process control block (PCB):

- It is also called as task control block.
- It is a representation of the process in the Operating System.

Structure of PCB:

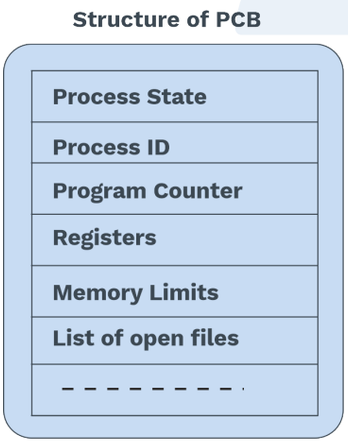


Fig. 2.4 Process Control Block

Process state:
The state of a process can be new, ready, running, waiting, halted, etc.

Program counter:
It holds the address of the next instruction to be executed.

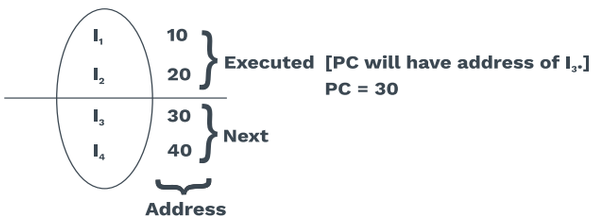


Fig. 2.5 Instruction Sequence of a Process

CPU registers:
They depend on computer architecture, i.e. accumulator, index register, stack pointers, etc.

Process ID:
It is a unique ID which is assigned by the Operating System at the time of process creation.

Memory management information:
It includes values of base and limit registers, page tables, etc.
PCB of the processes will be stored in the main memory. PCBs are implemented in the memory using a **doubly linked list.**

Rack Your Brain

Can the ready queue be implemented with a data structure other than a linked list?
If yes, why do we use a linked list popularly?

Context switch:

Definition:

Switching the CPU to another process requires performing a state save of the current process and restoring the other process

- Interrupts cause the OS to change the CPU from one task to run a kernel routine.
- Such operations frequently happen in the general-purpose system, and when that happens, the system needs to save the current process state so that it can restore later when continued.
- The context here is the PCB of the process. It contains CPU register values and other information.

- Context switching is purely overhead because the system does not perform any useful work while switching.
- Its speed varies from system to system, memory speed and the number of registers being copied.

Process state models:
State of the process changes during its execution. Process state tells the current activity of that process, i.e. whether the process is running on CPU or using I/O device, etc. A process can be in one of the following states:
New: Creation of a new process.
Ready: Waiting to get scheduled by the processor.
Running: Executed by the processor.
Waiting: Waiting for some event to occur (such as an I/O completion or reception of a signal).
Terminated: Execution completed.
Suspend ready: A process in the ready state, which is moved to secondary memory from the main memory due to lack of the resources (RAM).
Suspend wait: Instead of removing the process from the ready queue, it is better to remove the blocked process, which is waiting for some resources in the main memory. Since it is already waiting for some resources to get free, it is better to wait in the disk and give space to the high priority process.

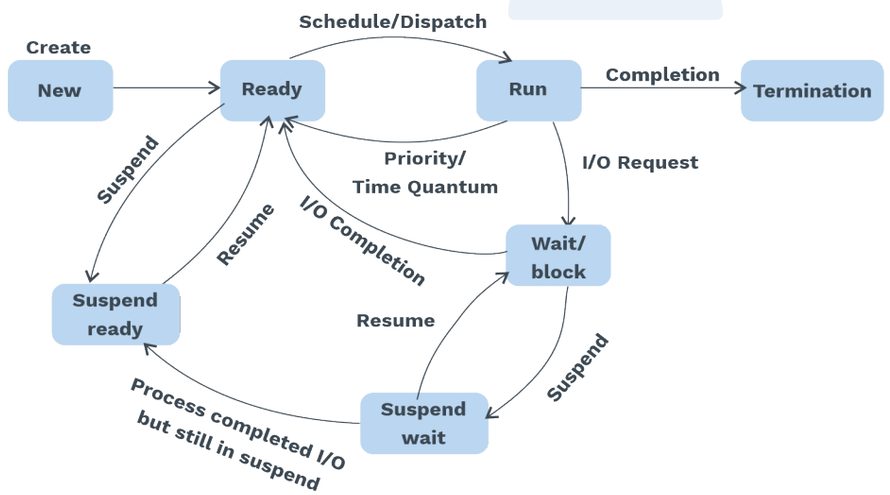


Fig. 2.6 Process State Model

SOLVED EXAMPLES

Q1 **A running process gets into blocked state if it _____**
Which of the following is/are TRUE?
a) needs data stored on a hard disk.
b) needs a resource which is held by other process.
c) is waiting for an event to occur such as scrolling the mouse.
d) completes its execution.

Sol: **Options: a), b), c)**

A running process gets into blocked state:
- if it needs to read/write data from secondary storage.
- if it needs a resource which is currently held by other process.
- if it is waiting for an event to happen either by user or other processes.
- completion of a process leads to its termination.

Important commands:
Fork () system call:
A system call that creates a new process is known as a child process. It is identical to the calling one, i.e. parent process.

- Makes a copy of text, data, stack and heap.
- Starts executing on that new copy.
- When the child process is created using fork(), a new memory location with a copy of all the parent's data will be allocated to the child process.
- Both parent and child processes will have the same virtual address but different physical address.
- The next instruction after the fork() will be executed by both child and parent processes.
- The program counter, CPU registers and open files, which are present in the parent process, are also used by the child process.
- It does not take any parameter and returns an integer value containing the newly created child process ID.
- If fork() returns:

Negative value: Child process creation unsuccessful.
Zero: Returned to the newly created child process.
Positive value: Returned to parent or caller.

Whether the parent process will execute first or the child process will be decided by the operating system.

Fork example:

CODE:

```
# include <stdio.h>
  int main ()
  {
    fork ();
    printf("Hello
world\n");
    return 0;
  }
```

O/P

```
Helloworld
Helloworld
```

Inefficient to physically copy memory from parent to child.

- Code (text section) remains identical after fork ().
- Even portions of the data section, heap and stack may remain identical after fork ().

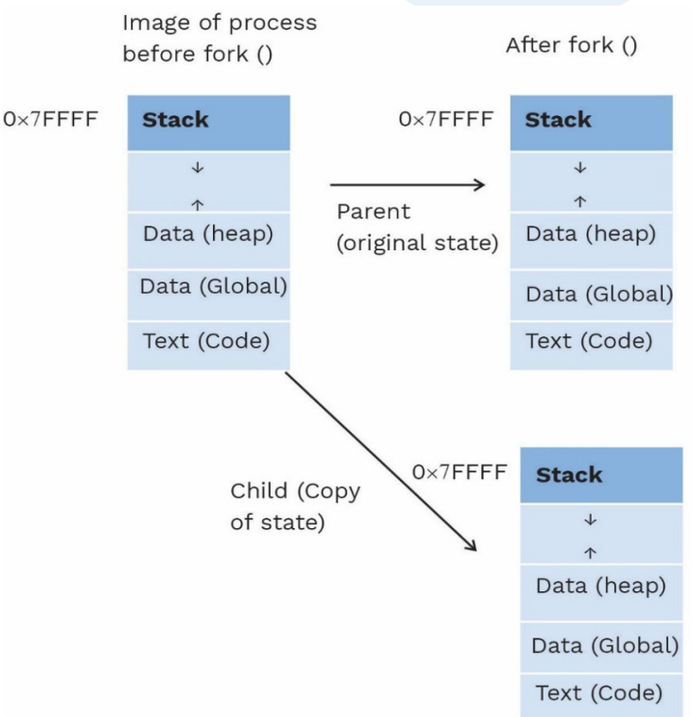


Fig. 2.7 Before and After Fork() Process Image in Physical Memory

Copy-on-write:

- OS memory management policy lazily copy pages only when they are modified.
- Initially map same physical page to child virtual memory space (but in read mode).
- Write to child virtual page triggers page protection violation (exception).
- OS handles exception by making physical copy of page and remapping child virtual page to that page.

SOLVED EXAMPLES

Q2 **Calculate number of times hello is printed?**

CODE:

```
int main ()
{
fork();
fork();
fork();
printf("Hello");
return 0;
 }
```

Sol: Range: 8 to 8

The total number of times 'hello' is printed is equal to the number of processes created and the total number of the processes when forks are in series = 2^n, where 'n' is the number of fork system calls, so 2^3=8. Execution of parent and child process happens in which order is not decided by application, OS decides which process will get control first, parent or child.

Note:

- Parent and child processes run in parallel by running on different processors on a multi-processor system, and if it is a uniprocessor system, then by context switching.
- When n forks are in series, number of child processes created is 2^n-1, and the total number of processes is 2^n.

Previous Years' Question

Consider the following code fragment.

```
if (fork()==0)
{
    a=a+5;
    printf("%d,%d\n",a,&a);
}
else
{
    a=a-5;
    printf ("%d,%d\n",a,&a);
}
```

Let u,v be the values printed by the parent process and x,y be the values printed by the child process. Which one of the following is true?

a) u=x+10 and v=y **b)** u=x+10 and v!=y
c) u+10= x and v=y **d)** u+10= x and v!=y **(GATE CS - 2005)**

2.2 SCHEDULING

Scheduling queues and state queuing diagram:

OS maintains two types of queue in memory

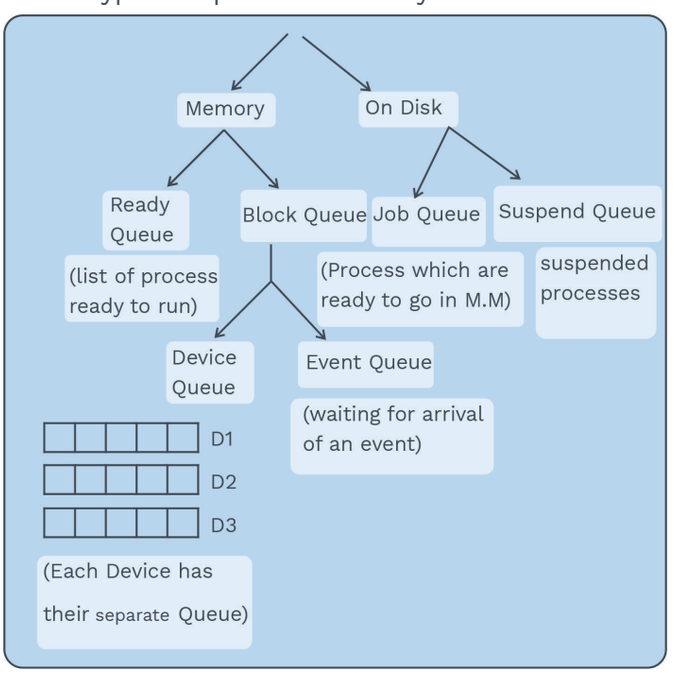


Fig. 2.8 Types of Queues

- **Job queue:** When a process is created, it is placed into job queue, which is maintained in secondary memory.
- **Ready queue:** When the process is brought in the main memory, it is kept inside the ready queue.
- Ready queue uses linked list as the data structure, and the header of the ready queue contains the pointers to the starting and ending PCBs of the processes in the list.
- **I/O queue:** In order to complete its execution, a process may require to do any other event, such as I/O operation. Till the I/O operation gets completed, the state of the process changes from running to block state. In the block state, there are multiple processes present, and the context of the PCBs of all the processes are stored in a I/O Queue.

Queuing diagram of process scheduling:

Queuing diagram is the common representation used for process scheduling discussion.

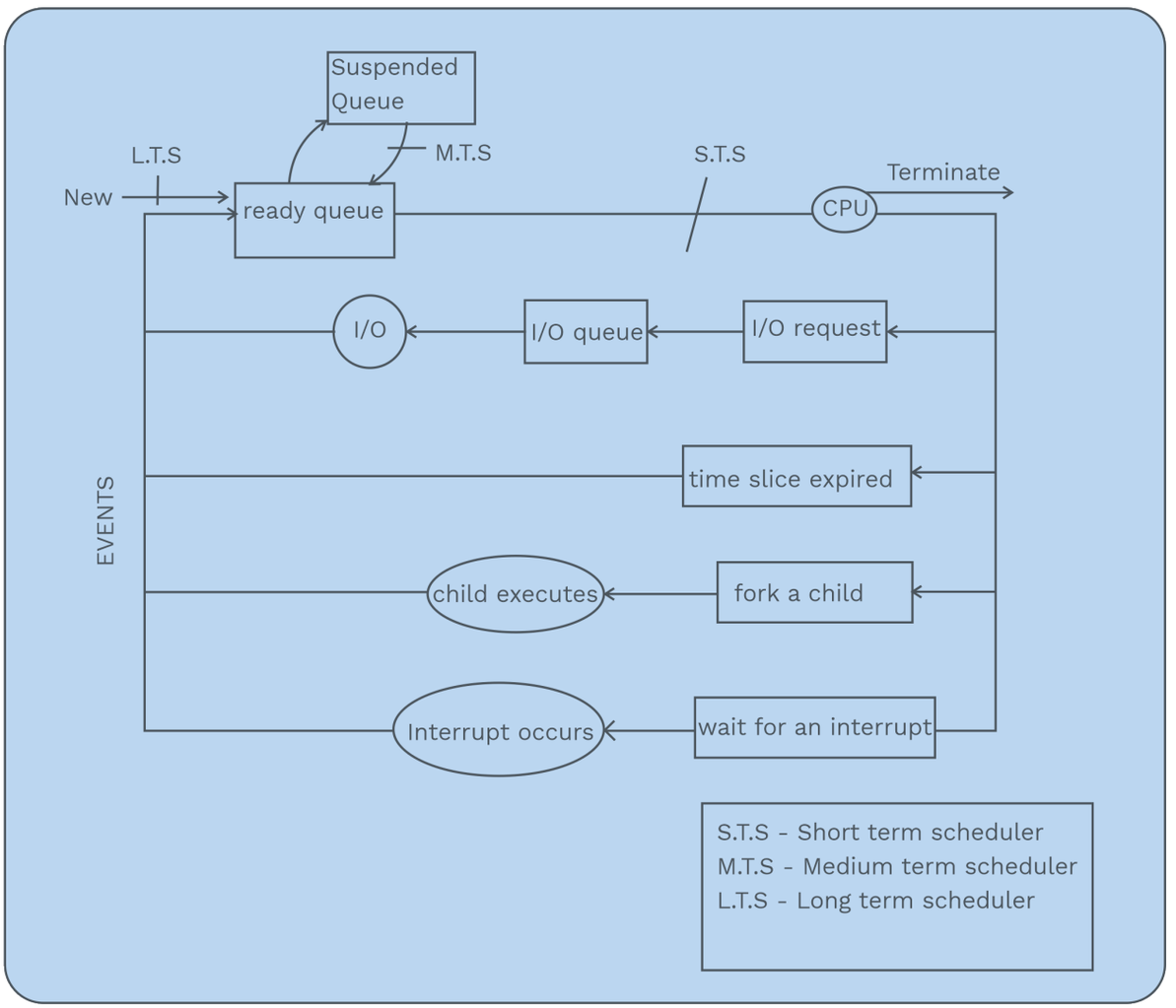
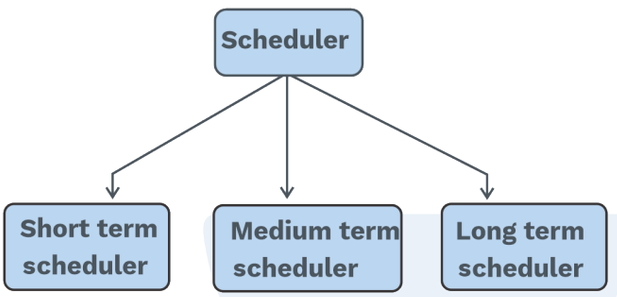


Fig. 2.9 Queuing Diagram of Process Scheduling

- When a process is created, it is brought into the main memory by Long-Term Scheduler (LTS) and placed in the ready queue, where it waits to get scheduled.
- The ready queue contains many processes; Short-Term Scheduler (STS) selects one process and schedule it on the CPU.
- The process can be removed from the CPU in the following ways:
- Process may require I/O operations, so it is placed in I/O queue.
- Process can create a new subprocess by calling fork() system call and then waits for the subprocess completion.
- If an interrupt occurs, the process can be preempted forcefully from the CPU.

- In preemptive scheduling, the time slice of the process may expire.
- In all the above situations, the process will switch from the waiting state to the ready state, where it is again placed onto the ready queue.
- This cycle is continued until the process gets terminated.

Types of scheduler:



1) STS (short-term scheduler)

- It selects one process from among the processes in the ready queue for allocation of CPU.
- Its frequency of execution is more compared to the other schedulers.
- It is invoked each time the CPU requires a new process for execution.

2) MTS (medium-term scheduler)

- It is used to decrease the number of processes in the main memory, so as to complete the remaining processes faster by reducing the CPU contention, giving better performance times.
- It is used to decrease the load on CPU.
- It is used in swapping out the processes from the main memory to the secondary memory.

3) LTS (long-term scheduler)

- It is responsible for choosing processes from the job queue to put in the ready queue.
- It controls degree of multiprogramming.
- It should choose a good mix of CPU-bound and IO-bound processes to keep its system's throughput high.

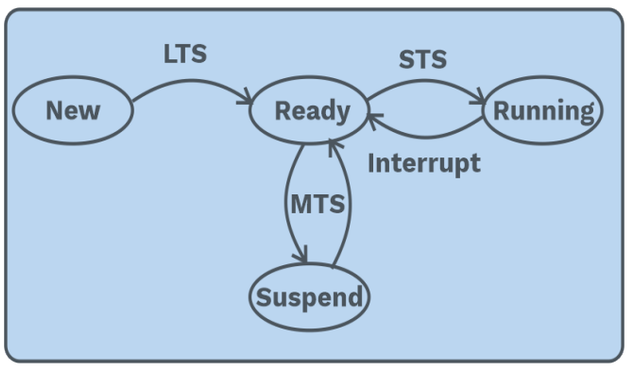


Fig. 2.10 Types of Scheduler

Dispatcher:

- It is a module/component of the CPU scheduling function.
- Short-term scheduler is responsible for selecting a job from the ready queue. The selected job is loaded on the CPU by the dispatcher.
- It switches the context of the already running process with the selected process, which is to be loaded on the CPU.

Dispatch latency:

Definition:

It is the time taken by the dispatcher to preempt the already running process and load the selected process on the CPU.

2.3 CPU SCHEDULING

Introduction:

- Single processor system can execute one process at a time; other processes in the ready queue must wait until the processor gets freed from the current running process.
- In a uni-programming system, the CPU has to remain idle till the current process completes its I/O.
- In a multiprogramming system, environment CPU cannot be left idle because many processes are in a ready queue waiting to get the CPU when the current executing Process waits for I/O.
- Scheduling is done before hand to orchestrate all the above tasks 'on' all computer resources. CPU is one of the primary computer resources; hence CPU scheduling is required.

CPU-I/O burst cycle:
Execution of process includes CPU cycle and I/O cycle, and these cycles repeat until the process gets completed.

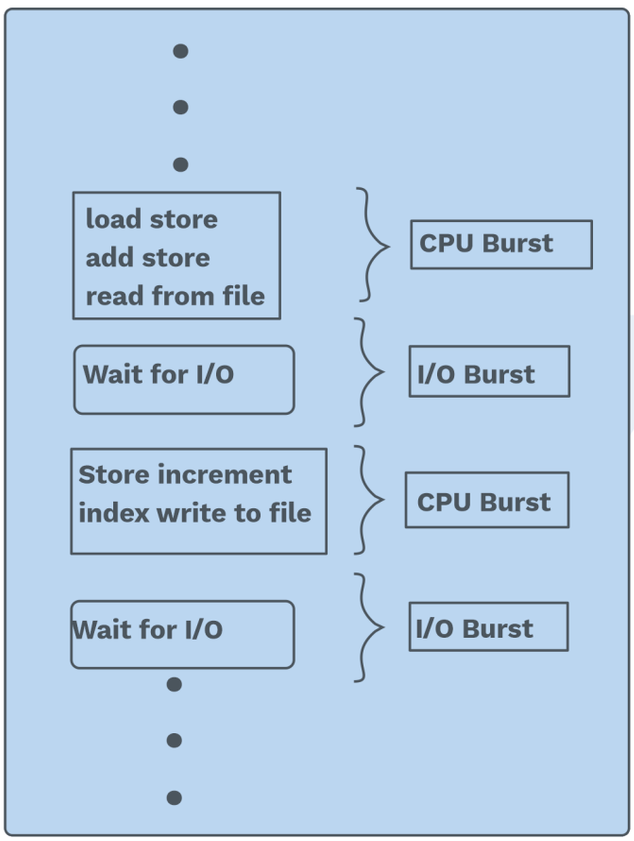


Fig. 2.11 Alternating Sequence of CPU and I/O

CPU-bound processes have:
- Long CPU bursts
- Short I/O bursts

I/O-bound processes have:
- Short CPU bursts
- Long I/O bursts

Goals of CPU scheduling:
CPU scheduling is required to choose one among the many processes present inside the ready queue for execution.
- **Fairness:** CPU must be allocated to every process fairly.

- **Efficiency:** Ideally, it should be 100% for CPU as we want to keep CPU busy.
- **Response time:** Response time for the interactive users must be reduced.
- **Throughput:** Maximum number of jobs must be executed per unit of time.
- **Waiting time:** Waiting time of the processes in the ready queue must be reduced.
- **Turnaround time:** Time between the termination of the process and the submission of the process must be reduced.

Preemptive and non-preemptive algorithms:

Preemptive scheduling: In preemptive scheduling, a process scheduled onto the CPU can be replaced by some other process based on priority, time, etc.

E.g.: Shortest Remaining Time First (SRTF), Round Robin Scheduling Algorithm (RR), etc.

Non-preemptive scheduling: In non-preemtive scheduling, once a process is scheduled onto the CPU, it cannot be replaced by any other process unless it completes its execution or is waiting for some event to occur.

E.g.: (First Come First Serve (FCFS)), Shortest Job First (SJF), etc.

Preemptive Scheduling	Non-preemptive Scheduling
1) A processor can be preempted to execute the different processes in the middle of any current process execution.	**1)** Once the processor starts the execution of a process, it must finish it before executing the other. It cannot be paused in the middle.
2) CPU utilisation is more efficient compared to non-preemptive.	**2)** CPU utilisation is less effie i e nt compared to preemptive.
3) Waiting and response time of preemptive scheduling is less.	**3)** Waiting and response time of the nonpreemptive is more.
4) 1t is flexible	**4)** It is rigid.
5) **E.g.:** Shortest remaining time first, Round Robin, etc.	**5)** **E.g.:** FCFS, SJF, highest response ratio next, etc.

Concept of starvation:

In a preemptive environment, low-priority processes have to wait for high-priority processes to complete their execution.

Sometimes these low-priority processes wait for an indefinite amount of time which leads to the problem of starvation.
One solution for starvation is aging, i.e. temporarily increasing the priority of long-waiting processes so that they can execute after a short wait time and do not go in an indefinite wait.

Scheduling criteria:

Goal of CPU Scheduling

1) To maximise the throughput and CPU utilisation

2) To minimise the average turnaround time (TAT), average waiting time and average response time of processes

Where to apply: Ready state

Who will apply: Short-term scheduler

When to apply:

Running → Termination

Running → Wait

Running → Ready

Wait → Ready

- Choosing a particular CPU scheduling algorithm among different scheduling algorithms may favour a particular group of processes over others.
- Many CPU scheduling algorithms have been considered, and characteristics used for comparison can make a considerable difference in which algorithms are found to be the best.

These characteristics are:

1) CPU utilisation:
- CPU has to be kept busy for a maximum amount of time.
- In real, the utilisation of CPU may lie between 40% to 90%.

2) Throughput:
- Number of processes that finished their execution in per unit time.
- Completion rate depends on the size of the processes. If the process size is long, then the completion rate can be a few processes per unit of time; else, for short processes, more processes can be completed per unit of time.

3) Turnaround time:

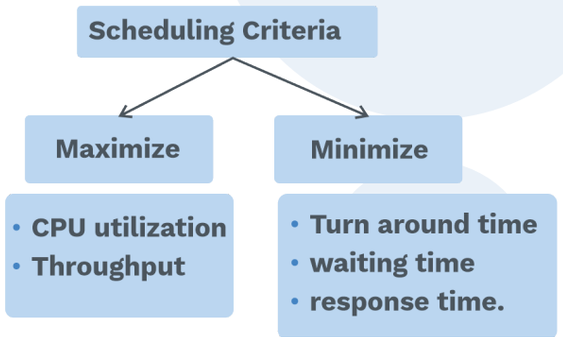
- It is the total time for which the process remains in the system, from the time it is submitted to the time it is completed.
- Turnaround time = Completion time of the process - Arrival time of the process

4) Waiting time:

- CPU scheduling does not affect the process burst time but affects how much time a process spends waiting in the ready queue.
- It is the sum of time for which the process was ready for execution, waiting for the CPU.

5) Response time:

- Time between the first response from the CPU and the first submission to the ready queue.



Let a process p_i from 'n' processes(P_1,P_2,P_3,.....,P_n)

- T.A.T = (C.T)–(A.T)
- WT = TAT– (BT + IO)
- WT = (CT – AT) – (BT + IO)

Abbreviation
W.T – Waiting time
B.T – Burst Time
RQ – Ready queue
A.T – Arrival time
C.T. – Completion time
TAT – Turn around time
IO – Input/outputtime

Let the following notation for process be Pi

1) $A.T(P_i)=A_i$

2) $B.T(P_i)=X_i$

3) $C.T(P_i)=C_i$

4) $TAT\ (P_i)=C_i-A_i$

5) $W.T(P_i)=TAT_i-X_i$

6) Average TAT $(P_i)=\frac{1}{n}\sum_{i=1}^{n}(C_i-A_i)$

7) Average WT $(P_i)=\frac{1}{n}\sum_{i=1}^{n}((C_i-A_i)-X_i)$

8) Total time for all process (schedule length) = $\max(C_i)-\min(A_i)$

9) $\text{Throughput}=\frac{n(\text{total processes})}{(\text{total time for all process})}$

10) Deadline $(P_i)=D_i$

Scheduling algorithms:

1) First come-first serve scheduling (FCFS)

Concept: Process that requested the CPU First will be allocated the CPU first
Criteria: Based on arrival time
Mode: Non-preemptive

SOLVED EXAMPLES

Q3 **Find the average turnaround time and average waiting time using FCFS algorithm?**

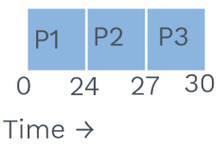
Process	Arrival time	Burst time
P1	0	24
P2	2	3
P3	3	3

a) Avg. TAT = 25.3 and Avg. WT = 15.3
b) Avg. TAT = 24.3 and Avg. WT = 15.3
c) Avg. TAT = 30 and Avg. WT = 14.3
d) Avg. TAT = 27.3 and Avg. WT = 15.3

Sol: **Option: a)**

Process	Arrival time	Burst time	TAT	WT
P1	0	24	24	0
P2	2	3	25	22
P3	3	3	27	24
		Avg.	25.3	15.3

Gantt chart:



$$\text{Avg TAT} = \frac{24+25+27}{3} = 25.3 \text{ units}$$

$$\text{Avg W.T} = \frac{0+22+24}{3} = 15.3 \text{ units}$$

Q4

Process ID	A.T.	B.T.	1/0	B.T
P1	0	3	2	2
P2	0	2	4	1
P3	2	1	3	2
P4	5	2	2	1

Consider the FCFS algorithm with non-zero context switch overhead of 111 unit. I/O operations can overlap any number of processes. Find the percentage of CPU

a) CPU context overhead = 33%, percentage of CPU idleness = 3%, CPU efficiency = 64%

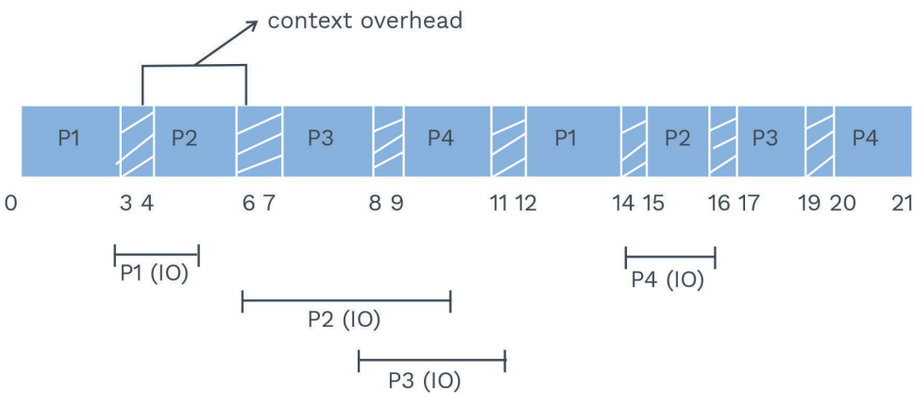
b) CPU context overhead = 30% , percentage of CPU idleness = 10%, CPU efficiency = 60%

c) CPU context overhead = 33.3%, percentage of CPU idleness = 5%, CPU efficiency = 61.67%

d) CPU context overhead = 33.3%, percentage of CPU idleness = 0%, CPU efficiency = 66.67%

Sol: **Option: c)**

Let's make Gantt chart using the given information.



So let total context switch overhead be x = 7 units
Total CPU idleness be y = 0 units
So %CPU idleness (y) = 0%

$$\%\text{CPU overhead due to context switch (x)} = \frac{7}{21} \times 100 = 33.3\%$$

So %CPU utilisation = 100 – (x + y)
= 100 – (33.3 + 0)
= 66.67%

Observations:

1) FCFS is non-preemptive.
2) Average waiting time in FCFS is generally not minimal.
3) In a dynamic situation, FCFS may experience a convoy effect, which causes low CPU utilisation.

Convoy effect:

It is a phenomenon associated with the FCFS algorithm in which the whole OS slows down due to a few slow processes.

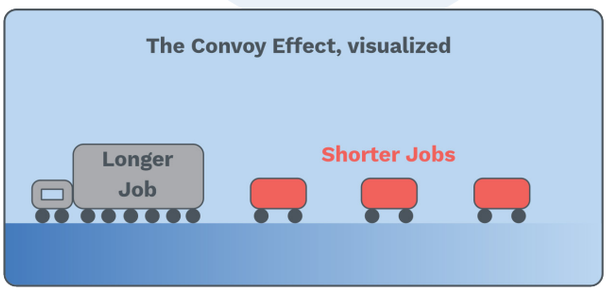


Fig. 2.13 Convoy Effect

- FCFS is non-preemptive in nature when a process is allocated CPU; other processes can get CPU only after the current process leaves CPU voluntarily.
- One long CPU extensive process takes a long time to execute while other short I/O intensive waits more for their turn, so whole system utilisation decreases.
- Preemptive scheduling algorithm like Shortest Remaining Time First can be used to avoid the convoy effect because processes having small size will not have to wait for more CPU time.

Note:

In any CPU scheduling algorithm, unless given in the question, if arrival times of the process are the same, then schedule the process which has the lowest process id.

2) Shortest job first scheduling (SJF)

Criteria: Burst time

Mode: Non-preemptive scheduling

Concept: Allocate the CPU to the process with smallest next CPU burst. If two or more processes are having the same burst time, the FCFS is used to break the tie.

SOLVED EXAMPLES

Q5 **Find the average turnaround time and average waiting time using SJF algorithm ?**

Pid	A.T	B.T
P1	0	4
P2	1	3
P3	2	5
P4	3	2
P5	4	1
P6	5	6

a) Avg. TAT = 7.5 and Avg. WT = 4.1

b) Avg. TAT = 7 and Avg. WT = 4.3

c) Avg. TAT = 7.8 and Avg. WT = 5.3

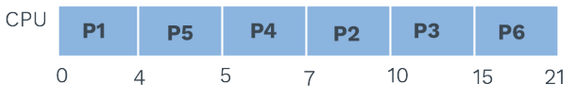
d) Avg. TAT = 7.8 and Avg. WT = 4.3

Sol: **Option: d)**

Pid	A.T	B.T	T.A.T	W.T
P1	0	4	4	0
P2	1	3	9	6
P3	2	5	13	8
P4	3	2	4	2
P5	4	1	1	0
P6	5	6	16	10

Ready Queue: P1 P2 P3 P4 P5 P6

Gantt chart



$$\text{Avg. TAT}\left(\text{completion time-arrival time}\right) = \frac{\sum \text{TAT}(P_i)}{n}$$
$$= \frac{4+9+13+4+1+16}{6}$$
$$= 7.8 \text{ units}$$

$$\text{Avg. waiting time}\left(\text{TAT-B.T}\right) = \frac{\sum \text{Wt}(P_i)}{n}$$
$$= \frac{0+6+8+2+0+10}{3}$$
$$= 4.3 \text{ units}$$

Q6 **Consider the following table :**

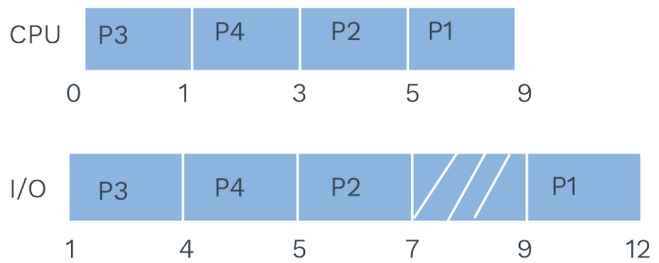
Process	CPU Burst time	I/O service time
P_1	4	3
P_2	2	2
P_3	1	3
P_4	2	1

Assume that all processes are arrived at time 0. The first process starts executed from '0' unit time. Every process completes its CPU request then gets I/O service. If any two process need same amount of CPU service time then prefer the process which has less I/O service request. Find the time at which process P_4 completes both CPU and I/O requests. Processes are scheduled using SJF for CPU services and I/O scheduling is done using FCFS scheduling.

Sol: **Range: 5 - 5**

Let us make Gantt chart

Gantt chart:



So P_4 completed its CPU cycle at '3' unit time and waited '1' time units for I/0. As P_3 is doing I/0 and then executes I/0 for 1 unit time and completes at '5' unit time.

3) Shortest remaining time first (SRTF):

Concept: It is the same as the shortest job first. Scheduling but with preemptive mode.

Criteria: Based on burst time.

Mode: Preemption.

Q7 **Find the average turnaround time and average waiting time using SRTF algorithm?**

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

Which of the following is correct?

a) Avg. TAT = 6 and Avg. WT = 2.5

b) Avg. TAT = 6.25 and Avg. WT = 2.75

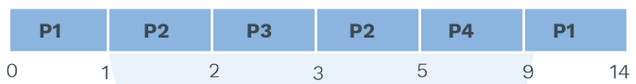
c) Avg. TAT = 7 and Avg. WT = 2.5

d) Avg. TAT = 6.3 and Avg. WT = 2.8

Sol: **Option: b)**

Process	Arrival time	Burst time	TAT	WT
P1	0	6	14	8
P2	1	3	4	1
P3	2	1	1	0
P4	3	4	6	2

Gantt chart:



Average waiting time = $WT_{AVG} = \frac{8+1+0+2}{4} = \frac{11}{4} = 2.75$

Avg. turnaround time = $TAT_{(AVG)} = \frac{14+4+1+6}{4} = 6.25$

Note:

- P SRTF gives overall less waiting time compared to other scheduling algorithms.
- It can be implemented with predicted burst time.

Prediction technique for burst time of processes:

For CPU scheduling algorithms like SJF or SRTF, which is based on burst times of all processes, it is not possible to get burst time data of processes at the start of the execution, but we can predict burst time based on previous data.

Few prediction techniques are:

1) Size of process:

Let P_i be a process, t_i is the actual burst time and τ_i is the predicted burst time of process.

Previous Data	Current Prediction
P_{old} = 198 KB	P_{new} = 200 KB
t_{old} = 20 ms	T_{new} = 20 ms

So burst time is predicted based on the size of the process.

2) Process types:

Burst times are predicted based on the type of processes going to be scheduled.

Various types of burst time:

a) OS processes – $[4–6]_{ms}$

b) User interactive process – $[10–12]_{ms}$

c) Foreground process – $[20–25]_{ms}$

d) Background process – $[50–60]_{ms}$

3) Averaging previous CPU bursts:

Next CPU burst of the process is the average of its previous CPU burst. It is dynamic in nature. If t_i is the actual burst time of process P_i and τ_i is the predicted burst time of process P_i

$$\tau_{n+1} = \frac{1}{n}\sum_{i=1}^{n} t_i$$

4) Exponential averaging:

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

SOLVED EXAMPLES

Q8 Calculate the exponential averaging with τ_1 = 10, a = 0.5, and the algorithm is SRTF with previous runs as 8, 7, 4, 16.

Sol: Range: 7.5 to 7.5

τ_1=10, α=0.5. putting in formula

[∴ As SRTF is applied, so process will be served 4, 7, 8, 16]

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

$\Rightarrow \tau_2$=0.5*4+0.5*10 $[t_1=4 \ \& \ \tau_1=10]$

$\Rightarrow \tau_2$=7

$\Rightarrow \tau_3$=0.5 * 7+0.5 * 7 $[t_2=7 \ \& \ \tau_2.7]$ $\Rightarrow \tau_4$=0.5*8+0.5*7 $[t_3=8, \tau_3=7]$

$\Rightarrow \tau_3 = 7$ $\Rightarrow \tau_4 = 7.5$

Previous Years' Question

Consider three processes, all arriving at time 0 with total execution time of 10, 20 and 30 units, respectively. Each process spends the first 20% of execution time doing 1/0, the next 70% of time doing computation and the last 10% of time doing 1/0 again. The operating system uses the shortest remaining time first scheduling algorithm and schedules a new process, either when the running process gets blocked on 1/0 or when the running process finishes its compute burst, assuming that all i/O operations can be overlapped as much as possible. For what time of percentage does the CPU remain IDLE?

a) 0% **b)** 10.6%

c) 30.0% **d)** 89.4%

Sol: b) **(GATE CS-2006)**

4) Longest remaining time first (LRTF):

Definition:

LRTF, which stands for Longest Remaining Time First, is a scheduling algorithm used by the operating system to schedule the incoming processes so that they can be executed in a systematic way. This algorithm schedules those processes first which have the longest processing time remaining for completion. This algorithm can also be called as the preemptive version of the LJF scheduling algorithm.

Criteria: Burst time (largest first)
Mode: Preemptive

SOLVED EXAMPLES

Q9 **Find the average turnaround time and average waiting time using LRTF algorithm?**

PNo	A.T	B.T
P1	0	2
P2	0	4
P3	0	8

Which of the following is correct?

a) Avg. TAT = 13.25 and Avg. WT = 8.3
b) Avg. TAT = 12.7 and Avg. WT = 8.2
c) Avg. TAT = 13 and Avg. WT = 8.3
d) Avg. TAT = 13 and Avg. WT = 8

Sol: Option: **c)**

PNo	A.T	B.T	TAT	WT
P1	0	2	12	10
P2	0	4	13	9
P3	0	8	14	6

Gantt chart:

CPU	P3	P2	P3	P2	P3	P1	P2	P3	P1	P2	P3
0	4	5	6	7	8	9	10	11	12	13	14

$$\text{Avg TAT} = \frac{13+12+14}{3} = \frac{39}{3} = 13$$

$$\text{Avg W.T} = \frac{10+9+6}{3} = \frac{25}{3} = 8.3$$

5) Round-robin scheduling (RR):

Definition:

Round-robin (RR) is one of the algorithms employed by operating systems and network schedulers in computing. As the term is generally used, time slices (also known as time quanta) are assigned to each process in equal portions and in circular order, handling all processes without priority (also known as cyclic executive). Round-robin scheduling is simple, easy to implement and starvation-free.

Criteria: Order of arrival
Mode: Preemptive
Concept:

1) It is FCFS (preemptive) based on the time out interval.
2) Ready queue is maintained for scheduling.
3) Small time quantum leads to more context switches (overhead).
4) Large T.Q makes RR act as FCFS.

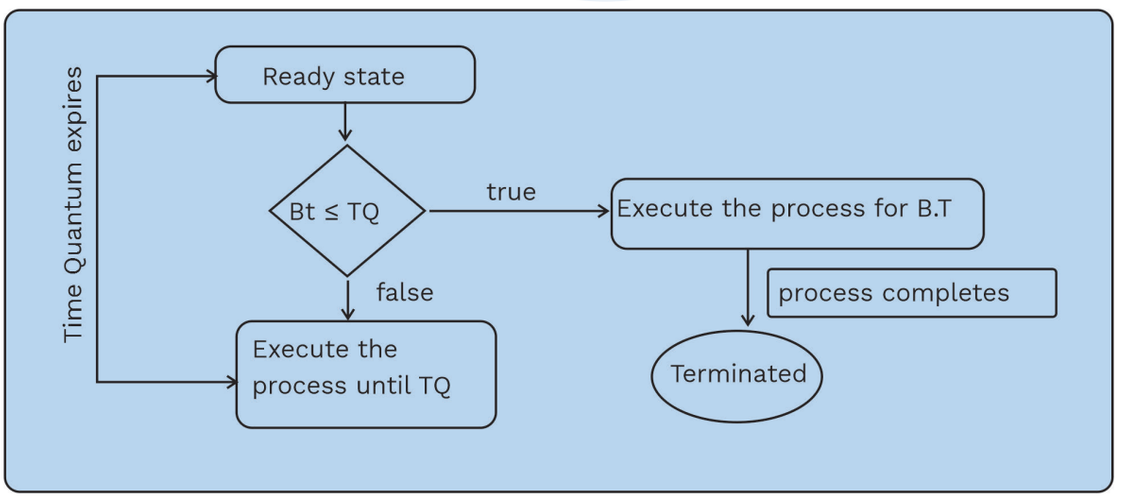


Fig. 2.14 RR Scheduling Flow Chart

SOLVED EXAMPLES

Q10 **What is the average turnaround time and average waiting time using round robin scheduling algorithm, when time quantum is 5 units?**

Process	AT	B.T
P1	0	5
P2	1	7
P3	3	3
P4	4	6

Which of the following is correct?

a) Avg. TAT = 12.75 and Avg. WT = 7.5
b) Avg. TAT = 12 and Avg. WT = 7
c) Avg. TAT = 12.75 and Avg. WT = 7
d) Avg. TAT = 12 and Avg. WT = 7.5

Sol: **Option: a)**

Process	AT	B.T	TAT	W.T
P1	0	5	5	0
P2	1	7	19	12
P3	3	3	10	7
P4	4	6	17	11

Time Quantum (TQ) = 5 unit

Gantt chart:

CPU	P1	P2	P3	P4	P2	P4
0	5	10	13	18	20	21

$$\text{Avg TAT} = \frac{5+19+10+17}{4} = \frac{51}{4} = 12.75$$

$$\text{Avg WT} = \frac{0+12+7+11}{4} = \frac{30}{4} = 7.5$$

Q11 **A system implementing Round Robin Process Scheduler, with a context switch delay of 3 time units. Which of the following is/are the values of the time quantum, for which the context switch overhead stays below 50%? (Calculations should be rounded off upto 2 decimal points)**

a) 7 time units
b) 3 time units
c) 4 time units
d) 6 time units

Sol: **Options: a), c), d)**

Let context switch delay be expressed as TD = 3 time units and time quantum be expressed as TQ.

Context switching overhead is given as, TO = TD / (TQ + TD)

- A) TQ = 7 time units. TO = 3 / (7 + 3) = 0.3 < 0.5
- B) TQ = 3 time units. TO = 3 / (3 + 3) = 0.5 == 0.5
- C) TQ = 4 time units. TO = 3 / (4 + 3) = 0.43 < 0.5
- D) TQ = 6 time units. TO = 3 / (6 + 3) = 0.33 < 0.5

Previous Years' Question

Consider a process sharing the CPU in a round-robin fashion. Assuming that each process switch takes's' seconds, what must be the quantum size 'q' such that he overhead resulting from process switching is minimised, but at the same time each process is guaranteed to get its turn at the CPU at least every 't' seconds?

a) $q \leq \frac{t\text{-}ns}{n\text{-}1}$ **b)** $q \geq \frac{t\text{-}ns}{n\text{-}1}$

c) $q \leq \frac{t\text{-}ns}{n+1}$ **d)** $q \geq \frac{t\text{-}ns}{n+1}$

Sol: a) **(GATE CS-2066)**

Advantages	Disadvantages
1) This works best for shorter jobs **2)** Starvation is not possible	**1)** It works poor if all jobs have the same length **2)** Very large quantum will make it behave s FCFS

Rack Your Brain

How do you say whether a time quantum in Round Robin Scheduling Algorithm is fairfor all processes or not?

6) Highest response ratio next (HRRN):

Criteria: Response ratio.

Mode: Non-preemptive.

$$\text{Response Ratio } (RR) = \frac{\text{Turn around time}}{\text{Burst time}}$$

$$= \frac{\text{Waiting time} + \text{Burst time}}{\text{Burst time}}$$

Concept:

1) HRRN selects a process which has the highest response ratio to schedule as next process into CPU.

2) It minimises the average turnaround time.

3) It favours both longer and shorter processes.

4) Starvation is not possible.

5) Non-preemptive in nature.

SOLVED EXAMPLES

Q12 **HRRN scheduling is used; compute the average turnaround time and average waiting time?**

Process	A.T.	B.T
P1	0	4
P2	2	1
P3	5	2
P4	4	2

Which of the following is correct?

a) Avg. TAT = 3.5 and Avg. WT = 1.5
b) Avg. TAT = 3.5 and Avg. WT = 1.25
c) Avg. TAT = 3 and Avg. WT = 1.25
d) Avg. TAT = 3.25 and Avg. WT = 1.5

Sol: **Option: b)**

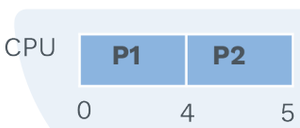
Process	A.T	B.T	W.T	TAT
P1	0	4	0	0
P2	2	1	2	3
P3	5	2	2	4
P4	4	2	1	3

Ans: At t = 0, only P1 is in the ready queue

At t = 4, P2 and P4 are in the ready queue.
So to assign CPU, we calculate their response ratio.

$$RR(\text{process } 2) = \frac{WT_2 + BT_2}{BT_2}$$
$$= \frac{2+1}{1} = 3$$
$$RR(\text{process } 4) = \frac{WT_4 + BT_4}{BT_4}$$
$$= \frac{0+2}{2} = 1$$

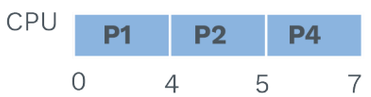
P2 has a large response ratio.



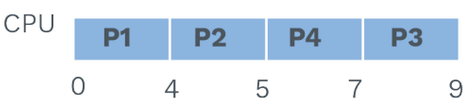
Then again at t=5, $RR(\text{process } 4) = \frac{1+2}{2} = 1.5$

$RR(\text{process } 3) = \frac{0+5}{5} = 1$

P4 has the highest response ratio.



Now only P3 is left, so we directly push it into the CPU after P4.



Avg. waiting time $= \frac{0+2+2+1}{4} = \frac{5}{4} = 1.25$

Avg. Turnaround time $= \frac{4+3+4+3}{4} = 3.5$

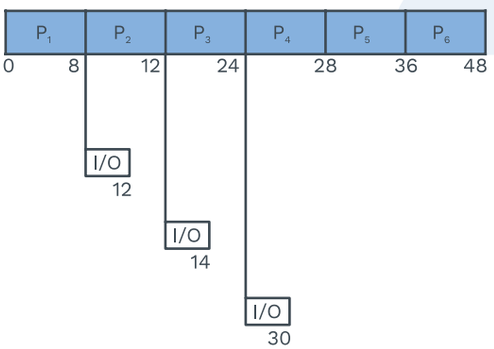
Q13 **Three processes with their execution time are given below:**

Process	Service Time		
	CPU	I/O	CPU
P_1	8	4	8
P_2	4	2	4
P_3	12	6	12

All times in milliseconds, and all processes arrive at time 0. Processes start executing on CPU first, then go for 1/0 and at last finish execution with CPU. 1/0 computations can be done parallelly and no overhead for context switch. What is the average waiting time (in milliseconds) of these processes when scheduled using highest response ratio next (HRRN) algorithm?

Sol: **Range: 10.66 - 10.67**

At time 0 response ratio for all the processes are 1. So process with the lowest number gets scheduled.



$$\text{Response Relation}\left(P_i\right) = \frac{WT\left(P_i\right) + ST\left(P_i\right)}{ST\left(P_i\right)}$$

Response Ratio at time 8:	Response Ratio at time 12	Response Ratio at time 24
$P_2 = \frac{8+10}{10} = 1.8$	$P_1 = \frac{0+12}{12} = 1$	$P_1 = \frac{12+8}{12} = 1.67$
$P_3 = \frac{8+30}{30} = 1.26$	$P_3 = \frac{12+30}{30} = 1.4$	$P_2 = \frac{10+6}{6} = 2.6$
P_2 is scheduled at time 8	P_3 is scheduled at time 12	P_2 is scheduled at 24

WT(P1) = 16, WT(P2) = 10, WT(P3) = 6

$$\text{Average waiting time} = \frac{16 + 10 + 6}{3} = \frac{32}{3} = 10.66$$

7. Priority-based scheduling algorithm:

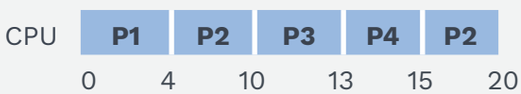
Concept: 1) Based on the priority of the processes, they are kept in the ready queue.
2) CPU allocated to highest priority first, and ties are broken by arrival time.

Criteria: Priority assigned

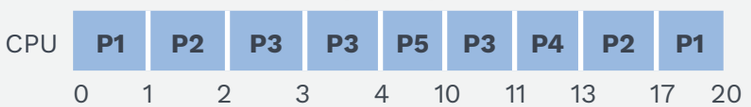
Mode: Non-preemptive/preemptive

Grey Matter Alert!

	Priority	Pid	A.T	B.T
(low)	4	P1	0	4
	6	P2	1	5
	8	P3	2	3
	7	P4	3	2
(high)	9	P5	4	6



(Non-preemptive)



(Preemptive)

Disadvantages:

1) Once priority is assigned to the process, it will not change during the execution of the process.

2) Low-priority process may get infinite blocking/starvation.

3) To prevent starvation problem, "Aging" mechanism is used.

4) Aging is the technique in which the priority of the process increases with the waiting time, i.e. if a process is waiting for the CPU for a longer duration of time, then its priority increases gradually.

Previous Years' Question

The arrival time, priority and duration of the CPU and 1/0 bursts for each of the three processes P_1, P_2, and P, are given in the table below. Each process has a CPU burst followed by an 1/0 burst followed by another CPU burst. Assume that each process has its own 1/0 resource.

Priority	Arrival time	Priority	Burst Duration (CPU)	Burst Duration (I/0)	Burst Duration (CPU)
P1	0	2	1	5	3
P2	2	3 (lowest)	3	3	1
P3	3	2 (highest)	2	3	1

The multiprogrammed OS uses preemptive priority scheduling. What are the finish times of the process P_1, P_2 and P_3?

a) 11, 15, 9 **b)** 10, 15, 9

c) 11, 16, 10 **d)** 12, 17, 11

Sol: d) **(GATE CS-2006)**

8) Multilevel queue scheduling:

Concept:

a) Ready queue is divided into several separate queues for different kinds of processes.

b) Processes are allocated permanently to one queue based on some property.

c) Different scheduling algorithms are applied for different queues.

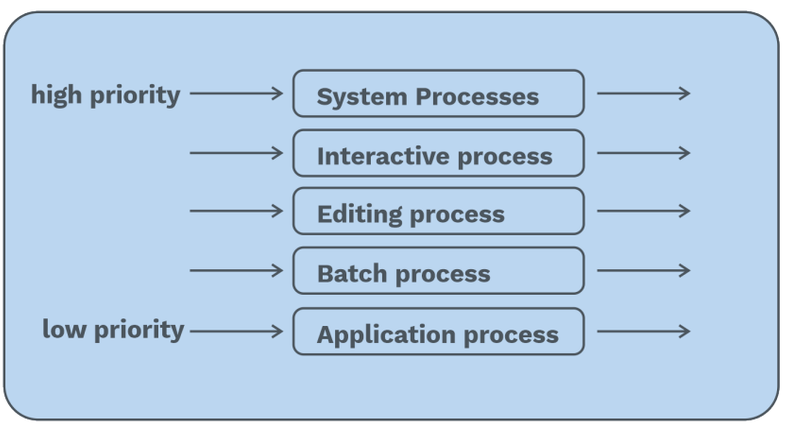


Fig. 2.15 Multilevel Queues

SOLVED EXAMPLES

Q14 **Consider a uniprocessor system which uses multilevel queue scheduling for processes. For example, given a set of four processes in the following table.**

Process	Queue No.	Arrival_Time	Burst_Time
P_1	1	0	5
P_2	1	2	3
P_3	2	4	6
P_4	1	9	4

All times are in milliseconds.

Queue No. 1 has the higher priority than Queue No. 2. Processes in Queue No. 1 et scheduled using Round Robin Scheduling with Time Quantum 2 ms and processes in Queue No. 2 are scheduled using First Come First Serve Scheduling. What is the average turnaround time (in milliseconds) of these processes?

Sol: **Range: 7.75 - 7.75**

As Queue No. 1 has the higher priority, processes in it also has a higher priority than processes in Queue No. 2.

Gantt chart:

Turnaround time (TAT) calculations:

TAT (P1) = 8–0 = 8

TAT (P2) = 7–2 = 5

TAT (P3) = 18–4 = 14

TAT (P4) = 13–9 = 4

$$\text{Average TAT} = \frac{8+5+14+4}{4} = \frac{31}{4} = 7.75\,\text{ms}$$

Note:

- Too much CPU time (low priority).
- 1/0 bound and interactive process (higher priority).
- Aging can be used to present starvation.

9) Multi-level feedback queue scheduling:

a) In this, every new process is scheduled to Queue 1 and based on the scheduling algorithm, if the process completes its execution, then it leaves the system, else it gets preempted and goes to the next queue.

b) Level i^{th} process preempted goes level $(i+1)^{th}$ ready queue.

c) MLQ and MLFQ both favour short burst time jobs.

d) Aging can be used to prevent starvation.

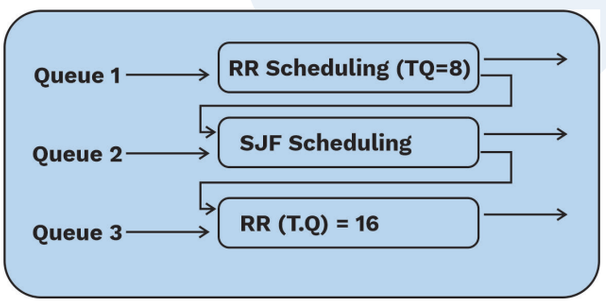


Fig. 2.16 Multi-level Feedback Queues

A MLFQ scheduler is defined by the following parameters:

i) Number of queues.

ii) Scheduling techniques required for each queue.

iii) Rules required to determine when to move a process to a higher priority queue or vice versa.

Q15 Consider a uniprocessor system that implements Multilevel Feedback Queue Scheduling algorithm for processes. There are 6 levels (1 to 6) of queue present In the system. The level 1 queue has a time quantum of 2 milliseconds and in each of the further level queues, time quantum Increases by 6 milliseconds. Currently, the system has only one CPU bound process requiring a CPU burst of 48 milliseconds.
Let 'X' be the number of times the process gets interrupted before completion and 'Y' be the queue level at which the process lastly moves before termination. What is the absolute difference between X and Y?

Sol: **Range: 1 - 1**

		Time Quantum	Process Execution	(48 ms)
L_1	Queue	2	2	
L_2	Queue	8(2+6)	8	
L_3	Queue	14(8+6)	14	
L_4	Queue	20(14+6)	20	
L_5	Queue	26(20+6)	4	Process executes 48 ms at level 5 Queue
L_6	Queue	32(26+6)	–	

→ Process gets interrupted four times at levels (1,2,3 and 4). So X = 4.
→ Process terminates at level 5 queue, so Y = 5.
→ Absolute difference between X and Y is 1.

2.4 THREADS

1) Thread is considered a lightweight process. As process, it is a unit of CPU utilisation.
2) A thread has its own program counter, a register set and a stack.
3) Code section, data section and other operating system resources are shared among all the threads of a process.
4) A heavyweight process is simply a single thread process.

Fig. 2.17 Single Thread Process

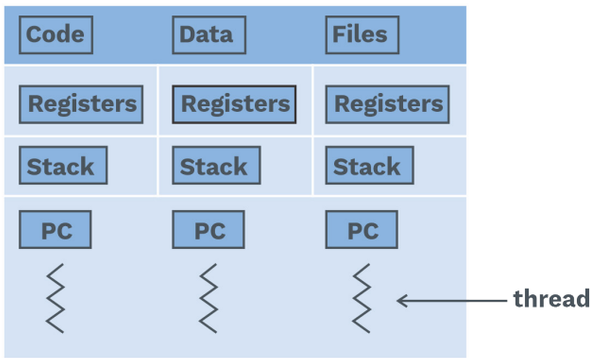


Fig. 2.18 Multithreaded Process

Modern PCs consist of many multithreaded software applications.
For example:

1) Web browser:

A web browser having multiple threads may contain one thread that shows the text or images and other threads may retrieve data from the web.

2) Word processor:

A word processor might contain one thread to show graphics, one thread can be used to respond to the keystrokes, while other threads can be used to check spellings and grammatical errors. Would they have been a single-threaded process, different services have to wait for their turn to get executed, and the waiting time would be enormous.

Benefits of multithreaded process:

1) **Responsiveness:**
   Multithreading in an interactive application that allows the process to continue its execution even if a part of it gets blocked or takes a longer time. Multithreading decreases the responsiveness of the system.

2) **Resource sharing:**
   Code section, data section and files are shared among all the threads of a process.
   A single application can have many threads present in the same address space using resource sharing.

3) **Economy:**
   It is more economical to use threads as they share the process resources.

4) **Utilisation of multiprocessor systems:**
If a process has several threads, then each thread can be scheduled on a separate processor, thus multiple processors can be effectively utilised.

Types of threads:

User level threads (ULT)

- These threads are created and implemented at the user level without the knowledge of the operating system, i.e. the operating system does not know the existence of these threads.
- Implementation of ULT is easy.
- Context switching time in ULT (user level threads) is less compared to KLT (kernel level thread).
- If any one of the user level threads performs a blocking operation, then the entire process gets blocked as OS does not see ULT but considers it as a single process.
- Context switching requires no hardware support.
- ULT favours non-blocking processes.

Kernel level threads (KLT):

- Kernel level threads are implemented by operating system and they are executed in kernel space.
- Kernel level threads are recognised by the operating system.
- Implementation of kernel thread is complicated.
- Context switch time in KLT is more.
- If one kernel thread performs blocking operation, then another thread can continue execution.
- Hardware support is needed.
- KLT favours blocking processes.

SOLVED EXAMPLES

Q16 **Find the TRUE statement(s) from the following.**

a) Threads of a user process communicate without invoking the kernel.

b) With improved software design, multithreaded applications execute faster.

c) A thread can be terminated when it completes its task or when another thread terminates it.

d) Process creation typically requires more CPU cycles than thread creation.

Sol: **Options: a), b), c), d)**

Threads communicate via their shared space.

An application can have multiple independent segments when each segment can execute simultaneously on a multiprocessor system.

Threads can be terminated on the completion of task or when another thread kills it.

Process needs more information than a thread.

Multithreading models in operating system:

Many–to–one model:

- In this model, many user level threads are mapped to a single kernel thread.
- If a single user level thread performs blocking operation, then the whole process will get blocked.
- Only one kernel thread can operate on a single processor at a time.
- Since, in this model, kernel level thread can be accessed by only one user level thread, so this model does not allow one process to be distributed among many CPUs.

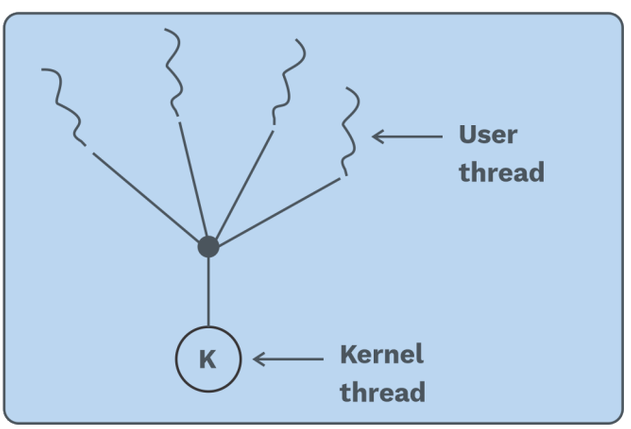


Fig. 2.19 Many to One Model

One–to–one model:

- In this model, each user thread is handled by a separate kernel thread.
- This model overcomes the problems involving blocking system calls and dividing the process across multiple CPUs.
- There is a significant overhead in managing one-to-one model.
- Most implementations of this model place a limit on how many threads can be created.

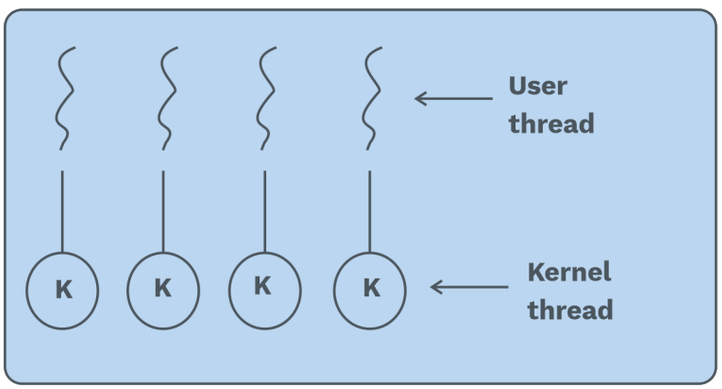


Fig. 2.20 One-to-One Model

Many–to–many model:

- In this model, any number of user level threads can be mapped to an equal or less number of kernel level threads.
- This model implements the best features of both one-to-one model and many-to-one model.
- There is no restriction on the number of threads created by the user.
- If any kernel level thread is blocked, then the entire process is not blocked.
- A process can be split among many processors.

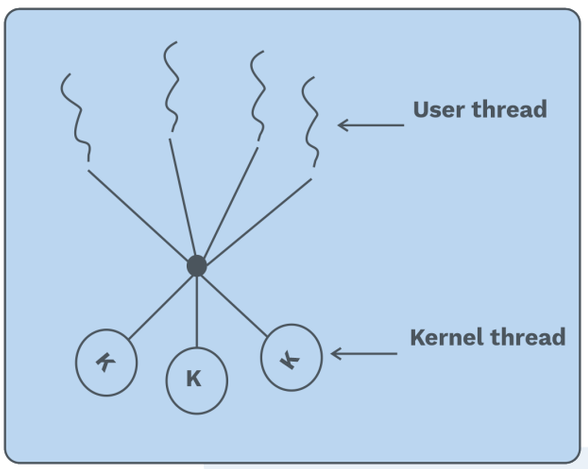


Fig. 2.21 Many-to-Many Model

SOLVED EXAMPLES

Q17 **Consider a multiprocessor system with two processors. Currently, there are two processes, P and Q, executing on separate processors. Let P have five user level hreads and Q have four kernel level threads. If one of the threads of each process gets blocked, then which one of the following is TRUE?**

a) All threads in process Q get blocked, but there is only one thread in process P that gets blocked.

b) All threads in process P get blocked, but there is only one thread in process Q that gets blocked

c) All threads in process P and process Q get blocked.

d) Only one thread in process P gets blocked and only one thread in process Q gets blocked.

Sol: **Option: b)**

All user level threads of a process get blocked, even single thread blocks; this is also a major drawback with user level threads. But in the case of kernel level threads of a process, if one thread gets blocked, other threads continue to execute.

Difference between multitasking and multithreading:

Multitasking

In the case of multitasking, these multiple jobs are executed on a single CPU in time-sharing mode. CPU is switched among different jobs.

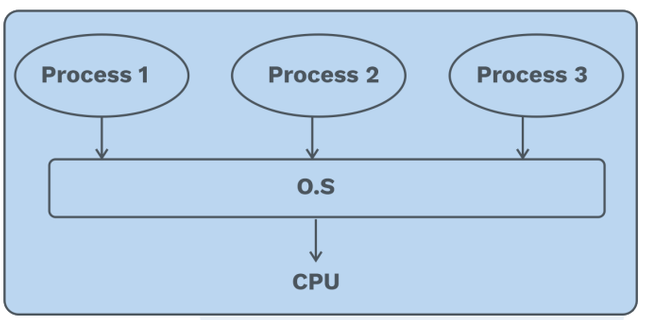


Fig. 2.22 Multitasking

Multithreading

In the case of multithreading, a process is divided into multiple threads, which can execute concurrently, thus increasing the power of the system.

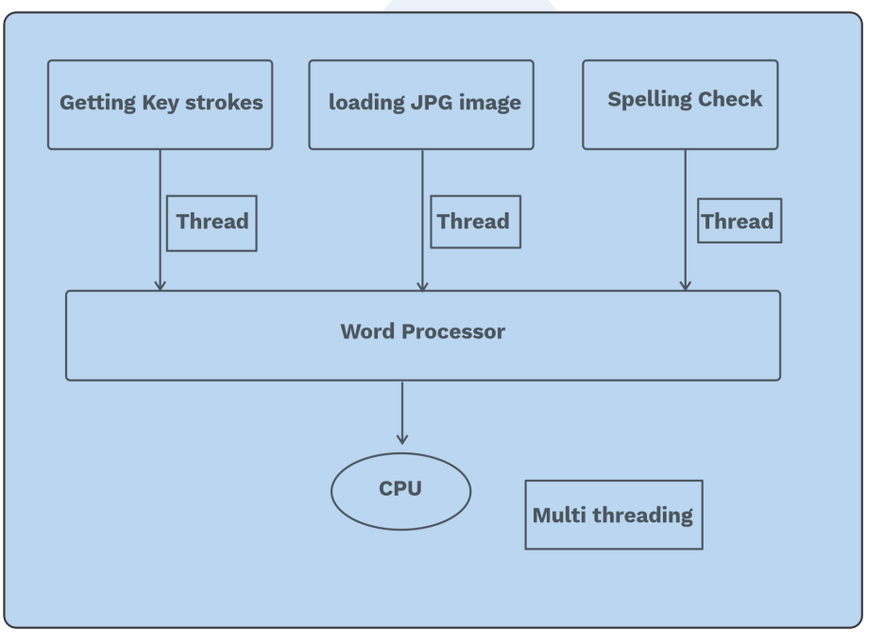


Fig. 2.23 Multithreading

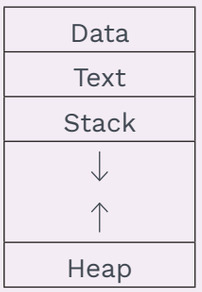
Multitasking	Multithreading
1) Multiple jobs are executed on a single CPU in time-sharing mode	**1)** Many threads are created from a process through which system's throughput is increased
2) It involves CPU switching between jobs	**2)** It involves switching between threads
3) Separate memory space is allocated to each job	**3)** Threads of a process share the same address space
4) There is no resource sharing among different jobs, each job will have its own resources	**4)** Threads share resources
5) It is slow	**5)** It is faster than multitasking
6) Termination of the process takes more time	**6)** Termination of thread takes less time

Rack Your Brain

Will multithreading give any benefits to a unicore processor?

Chapter Summary

- Process: **1)** Process is a program in execution
  2)



Process image

- Process operations:
  1) Create
  2) Schedule
  3) Run
  4) Block /Suspend
  5) Resume
  6) Terminate

- Process control block: Resumption of the process in OS.
- Types of process states:
  1) New
  2) Running
  3) Waiting
  4) Ready
  5) Terminated
  6) Suspend

- Dispatcher: It is a module/component for context switch operation in OS.
- Types of scheduler:
  1) Short-term scheduler
  2) Middle-term scheduler
  3) Long-term scheduler

- Context switch: Operation of storing the current process and loading another

- Goals of CPU sheduling:
  1) Maximise throughput and CPU utilisation
  2) Minimise turnaround time and waiting time.

- Scheduling algorithms:
  1) FCFS
  2) SJF/SRTF
  LJF/LRTF
  4) RR
  5) Priority
  6) HRRN
  7) MLQ
  8) MLFQ
- Threads: It is a basic unit of CPU utilisation
- Multithreading benefits:
  1) Responsiveness
  2) Resource sharing
  3) Economy
  4) Utilisation of multiprocessor architectures
- Types of threads:
  1) User level threads (ULT)
  2) Kernel level threads (KLT)
- Multithreading models:
  1) Many-to-one
  2) Many-to-many
  3) One-to-one