UNIT II

What is Process Scheduling?

The act of determining which process is in the **ready** state, and should be moved to the **running** state is known as **Process Scheduling**.

The prime aim of the process scheduling system is to keep the CPU busy all the time and to deliver minimum response time for all programs. For achieving this, the scheduler must apply appropriate rules for swapping processes IN and OUT of CPU.

Scheduling fell into one of the two general categories:

- **Non Pre-emptive Scheduling:** When the currently executing process gives up the CPU voluntarily.

Or

Non-preemptive scheduling is a CPU scheduling method in which **once a process is assigned the CPU, it continues execution until it finishes or enters the waiting (I/O) state**.

- **Pre-emptive Scheduling:** When the operating system decides to favour another process, pre-empting the currently executing process.

  Or
  Preemptive scheduling is a CPU scheduling method in which the **operating system can interrupt (preempt) a running process** and assign the CPU to another process.

What are Scheduling Queues?

In an Operating System, **scheduling queues** are data structures used to **organize processes according to their current state** so the OS can manage CPU and I/O efficiently.

1. 1. Job Queue

The Job Queue contains **all processes in the system**, usually stored on **secondary storage (disk)**, waiting to be admitted into main memory.

1. Managed by

- **Long-Term Scheduler**

2. Purpose

- Controls the **degree of multiprogramming**
- Decides which jobs enter the system

2. 2. Ready Queue

1. Definition

The Ready Queue contains **processes that are in main memory and ready to use the CPU**, but are waiting for CPU allocation.

2. Managed by

- **Short-Term Scheduler (CPU Scheduler)**

3. Purpose

- Source of processes selected for execution on the CPU

3. 3. Device Queue (I/O Queue)

1. Definition

A Device Queue contains **processes waiting for a specific I/O device**, such as:

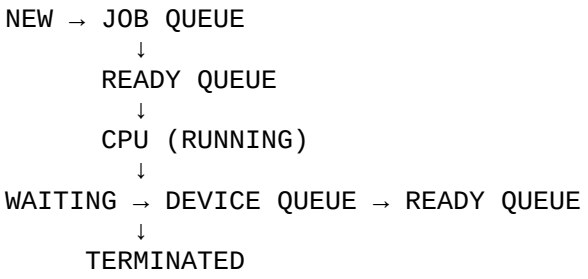
- Printer
- Disk
- Keyboard
- Network

Each I/O device has its **own queue**.

2. Managed by

- **I/O Scheduler**

4. 4. Movement of Processes Between Queues

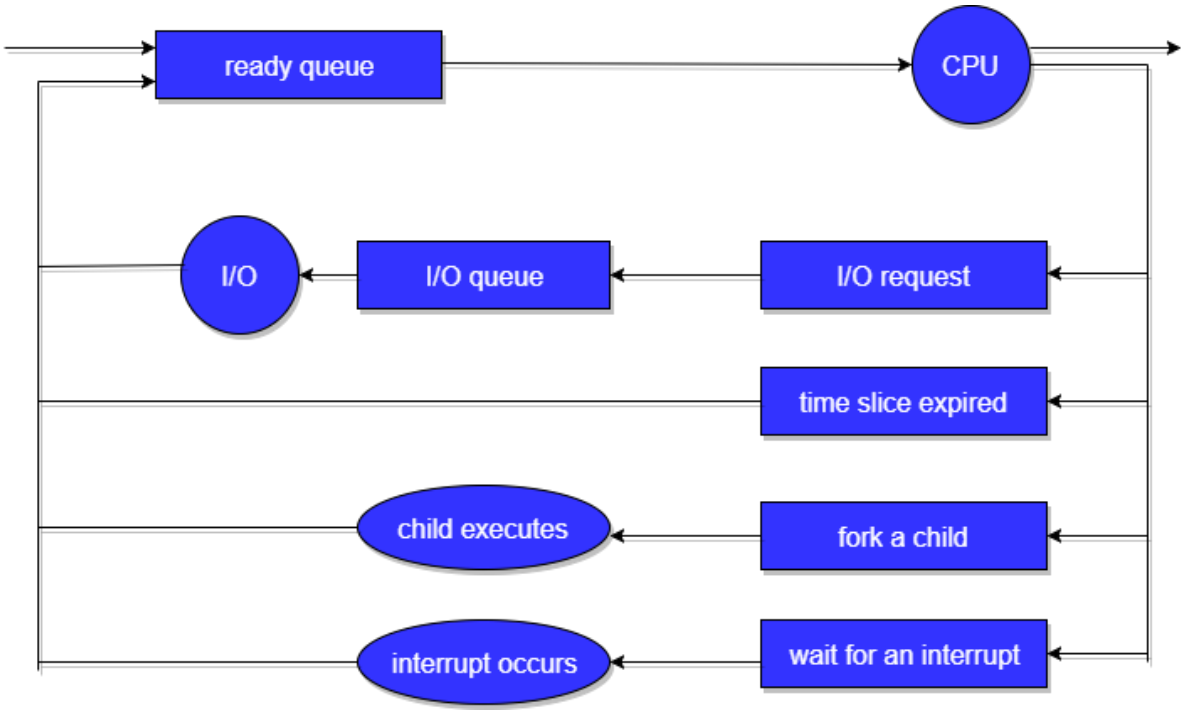


NOTES:-

- All processes, upon entering into the system, are stored in the **Job Queue**.
- Processes in the Ready state are placed in the **Ready Queue**.
- Processes waiting for a device to become available are placed in **Device Queues**. There are unique device queues available for each I/O device.

A new process is initially put in the **Ready queue**. It waits in the ready queue until it is selected for execution (or dispatched). Once the process is assigned to the CPU and is executing, one of the following several events can occur:

- The process could issue an I/O request, and then be placed in the **I/O queue**.
- The process could create a new sub process and wait for its termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.



In the first two cases, the process eventually switches from the
waiting state to the ready state, and is then put back in the ready queue. A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources
deallocated
.

CPU SCHUDULER

In **Operating Systems**, a **scheduler** is a component that decides **which process gets the CPU and when**.

1.

Types of Schedulers

There are three types of schedulers available:

1. Long Term Scheduler
2. Short Term Scheduler
3. Medium Term Scheduler

Let's discuss about all the diffferent types of Schedulers in detail: (od24IT-01).

Long Term Scheduler

Long term scheduler runs less frequently. Long Term Schedulers decide which program must get into the job queue. From the job queue, the Job Processor, selects processes and loads them into the memory for execution. Primary aim of the Job Scheduler is to maintain a good degree of Multiprogramming. An optimal degree of Multiprogramming means the average rate of process creation is equal to the average departure rate of processes from the execution memory.

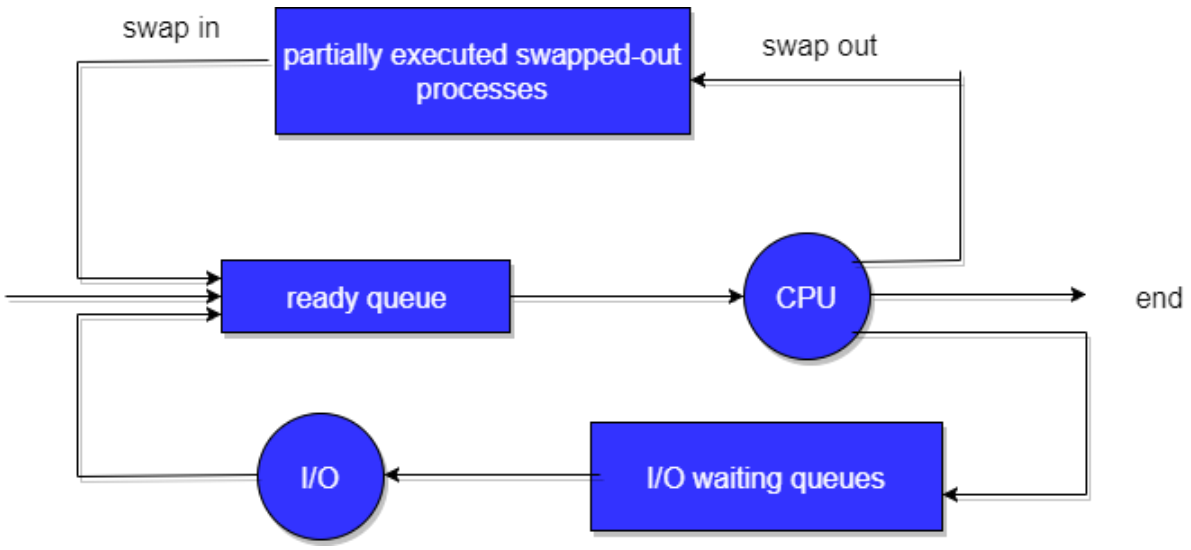
Short Term Scheduler

This is also known as CPU Scheduler and runs very frequently. The primary aim of this scheduler is to enhance CPU performance and increase process execution rate.

Medium Term Scheduler

This scheduler removes the processes from memory (and from active contention for the CPU), and thus reduces the degree of multiprogramming. At some later time, the process can be reintroduced into memory and its execution van be continued where it left of. This scheme is called **swapping**. The process is swapped out, and is later swapped in, by the medium term scheduler.

Swapping may be necessary to improve the process mix, or because a change in memory requirements has overcommitted available memory, requiring memory to be freed up. This complete process is descripted in the below diagram:



Addition of Medium-term scheduling to the queueing diagram.

CPU Scheduling Criteria

Different CPU scheduling algorithms have different properties and the choice of a particular algorithm depends on the various factors. Many criteria have been suggested for comparing CPU scheduling algorithms.
The criteria include the following:

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

2. **Throughput –**
   A measure of the work done by CPU is the number of processes being executed and completed per unit time. This is called throughput. The throughput may vary depending upon the length or duration of processes.

3. **Turnaround time –**
   For a particular process, an important criteria is how long it takes to execute that process. The time elapsed from the time of submission of a process to the time of completion is known as the turnaround time. Turn-around time is the sum of times spent waiting to get into memory, waiting in ready queue, executing in CPU, and waiting for I/O.

4. **Waiting time –**
   A scheduling algorithm does not affect the time required to complete the process once it starts execution. It only affects the waiting time of a process i.e. time spent by a process waiting in the ready queue.

5. **Response time –**
   In an interactive system, turn-around time is not the best criteria. A process may produce some output fairly early and continue computing new results while previous results are being output to the user. Thus another criteria is the time taken from submission of the process of request until the first response is produced. This measure is called response time.

There are various CPU Scheduling algorithms such as- (OD24COE).

- First Come First Served (FCFS)
- Shortest Job First (SJF)
- Longest Job First (LJF)
- Priority Scheduling
- Round Robin (RR)
- Shortest Remaining Time First (SRTF)
- Longest Remaining Time First (LRTF)

Operating System Scheduling algorithms (CSE DMI)

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which zxcwe are going to discuss in this chapter –

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling

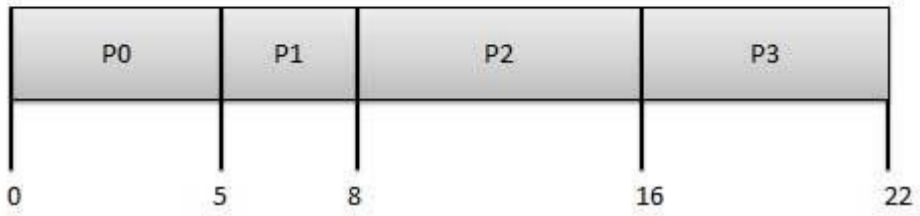
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

These algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

First Come First Serve (FCFS)

- Jobs are executed on fiirst come, fiirst serve basis.
- It is a non-preemptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	0 - 0 = 0
P1	5 - 1 = 4
P2	8 - 2 = 6
P3	16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.75

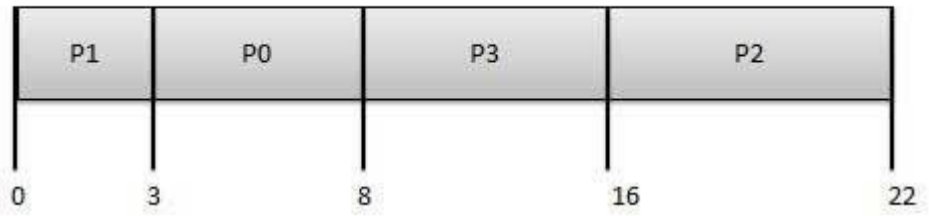
Shortest Job Next (SJN)

- This is also known as **shortest job fiirst**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

Given: Table of processes, and their Arrival time, Execution time

Process	Arrival Time	Execution Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	14
P3	3	6	8

Process	Arrival Time	Execute Time	Service Time
P0	0	5	3
P1	1	3	0
P2	2	8	16
P3	3	6	8



Waiting time of each process is as follows −

Process	Waiting Time

P0	0 - 0 = 0
P1	5 - 1 = 4
P2	14 - 2 = 12
P3	8 - 3 = 5

Average Wait Time: (0 + 4 + 12 + 5)/4 = 21 / 4 = 5.25

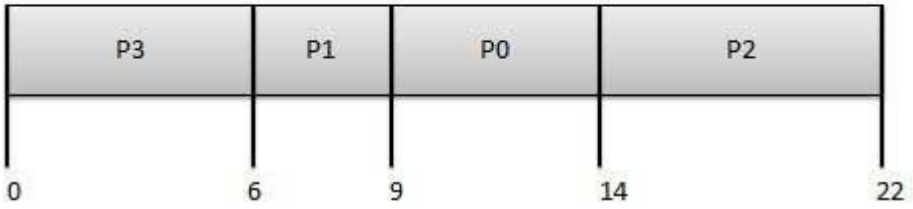
Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed fiirst and so on.
- Processes with same priority are executed on first come frst served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Given: Table of processes, and their Arrival time, Execution time, and priority.
Here we are considering 1 is the lowest priority.

Process	Arrival Time	Execution Time	Priority	Service Time
P0	0	5	1	0
P1	1	3	2	11
P2	2	8	1	14
P3	3	6	3	5

Process	Arrival Time	Execute Time	Priority	Service Time
P0	0	5	1	9
P1	1	3	2	6
P2	2	8	1	14
P3	3	6	3	0



Waiting time of each process is as follows −

Process	Waiting Time
P0	0 - 0 = 0
P1	11 - 1 = 10
P2	14 - 2 = 12
P3	5 - 3 = 2

Average Wait Time: (0 + 10 + 12 + 2)/4 = 24 / 4 = 6

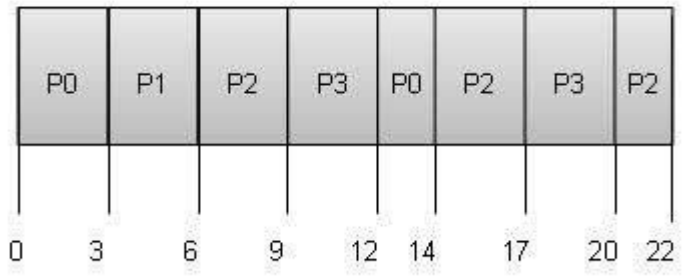
Shortest Remaining Time

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.

Quantum = 3



Wait time of each process is as follows −

Process	Wait Time : Service Time - Arrival Time
P0	(0 - 0) + (12 - 3) = 9
P1	(3 - 1) = 2
P2	(6 - 2) + (14 - 9) + (20 - 17) = 12
P3	(9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4
= 8.5

Multiple-Level Queues Scheduling

Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

Multiple Processors Scheduling in Operating System

Multiple processor scheduling or multiprocessor scheduling focuses on designing the system's scheduling function, which consists of more than one processor. Multiple CPUs share the load (load sharing) in multiprocessor

scheduling so that various processes run simultaneously. In general, multiprocessor scheduling is complex as

compared to single processor scheduling. In the multiprocessor scheduling, there are many processors, and they are identical, and we can run any process at any time.

The multiple CPUs in the system are in close communication, which shares a common bus, memory, and other peripheral devices. So we can say that the system is tightly coupled. These systems are used when we want to process a bulk amount of data, and these systems are mainly used in satellite, weather forecasting, etc.

There are cases when the processors are identical, i.e., homogenous, in terms of their functionality in multiple-processor scheduling. We can use any processor available to run any process in the queue.

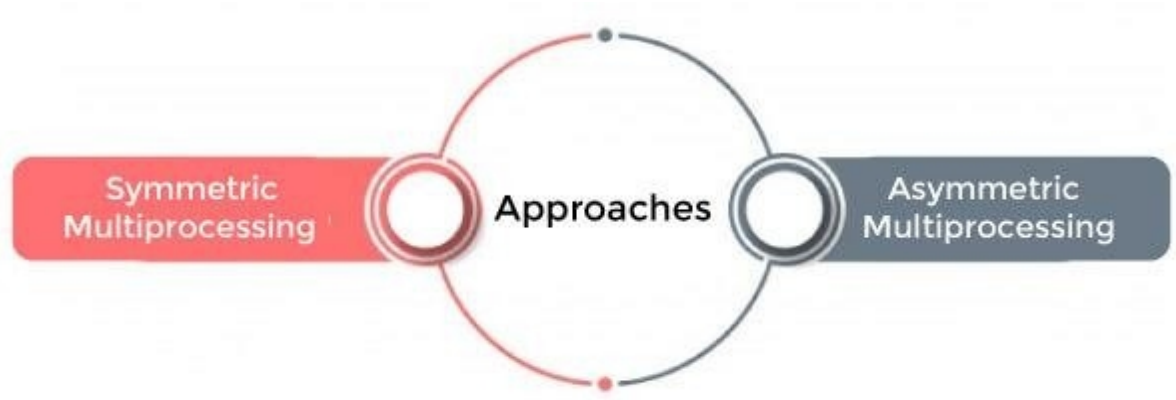
Multiprocessor systems may be ***heterogeneous*** (diffferent kinds of CPUs) or ***homogenous*** (the same CPU). There may be special scheduling constraints, such as devices connected via a private bus to only one

CPU.

There is no policy or rule which can be declared as the best scheduling solution to a system with a single processor. Similarly, there is no best scheduling solution for a system with multiple processors as well.

Approaches to Multiple Processor Scheduling

There are two approaches to multiple processor scheduling in the operating system: Symmetric Multiprocessing and Asymmetric Multiprocessing.



1. **Symmetric Multiprocessing:** It is used where each processor is ***self-scheduling***. All processes may be in a common ready queue, or each processor may have its private queue for ready processes. The scheduling proceeds further by having the scheduler for each processor examine the ready queue and select a process to execute.

2. **Asymmetric Multiprocessing:** It is used when all the scheduling decisions and I/O processing are handled by a single processor called the ***Master Server***. The other processors execute only the ***user code***. This is simple and reduces the need for data sharing, and this entire scenario is called Asymmetric Multiprocessing.

Thread scheduling

The scheduling of thread involves two boundary scheduling:

1. Scheduling of Kernel-Level Threads by the system scheduler.
2. Scheduling of User-Level Threads or ULT to Kernel-Level Threads or KLT by using Lightweight process or LWP.

Lightweight process (LWP):
The Lightweight process is threads that act as an interface for the User-Level Threads to access the physical CPU resources.
The number of the lightweight processes depends on the type of application, for an I\O bound application the number of LWP depends on the user level threads, and for CPU bound application each thread is connected to a separate kernel-level thread.

In real-time, the first boundary of thread scheduling is beyond specifying the scheduling policy and the priority, therefore, it requires two controls to be specified for the User level threads:

1. **Contention scope** – Control scope defines the extent to which contention takes place. Contention refers to the competition among the ULTs to access the KLTs.

Contention scope can be further classified into **Process Contention Scope (PCS)** and **System Contention Scope (SCS).**

- **Process Contention Scope:** Process Contention Scope is when the contention takes place in the same process.
- System contention scope (SCS): System Contention Scope refers to the contention that takes place among all the threads in the system.

1. **Allocation domain** – The allocation domain is a set of multiple (or single) resources for which a thread is competing.

Advantages of PCS over SCS:

The advantages of PCS over SCS are as follows:

1. It is cheaper.
2. It helps reduce system calls and achieve better performance.
3. If the SCS thread is a part of more than one allocation domain, the system will have to handle multiple interfaces.
4. PCS thread can share one or multiple available LWPs, while every SCS thread needs a separate LWP. Therefore, for every system call, a separate KLT will be created.