4) Scheduling

Page No. 47
Date

Scheduling objectives:-

One of the important function / objective of an operating system is CPU Job scheduling. Main objective of operating system is maximum utilization of CPU and I/O devices.

Multiple processes are ready in main memory and CPU switches from one process to another and maximum type it is kept busy. There should be proper schedule for a CPU, so that it can be utilized maximum and there will be no delays in process executions.

Scheduling concept:-

The idea is very simple. A process is executed until it must wait, typically for the completion of some I/O request.

When one process waits, the operating system gives the CPU to another process. This pattern continues. Every time a process waits, another process can take over the use of CPU.

CPU and I/O burst cycle:

Process execution consist of a cycle of CPU execution and I/O wait. Process alternate between these two states. Process execution begins with a CPU burst that is followed by an I/O burst, which is followed by another CPU burst, and so on.

Date

Types of scheduling :-

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

1. When a process switches from running state to waiting state
2. When process switches from running state to ready state
3. When process switches from waiting state to ready state
4. When a process terminates.

In situations 1 and 4, there is no choice of scheduling. A new process must be selected for execution. There is a choice for situations 2 and 3.

In 1 and 4, it is non-preemptive or cooperative

	Preemtive	Non-preemtive
1	In preemptive scheduling process can be taken out of CPU forcefully during its execution	In non-preemtive scheduling process can switch to waiting state on its own
2	Multiple processes can run one process can be preempted to run another process	When process is assigned to CPU, it keeps CPU till required waiting or terminate on its own
3	Process can switch from running to ready state	Process can switch from running to waiting state

④	Process can switch from waiting to ready state	Process may get terminate
⑤	It needs specific platform	It is platform and hardware independent
⑥	Modern operating system follows this approach	Older operating system follows this approach.

CPU scheduling criteria :-

Following are the CPU scheduling criteria :

① CPU utilization:
* We want to keep the CPU as busy as possible
* CPU utilization can range from 0 to 100 percent
* In a real system, it should range 40 - 90%

② Throughput :
* Number of processes that are completed per unit time is called as throughput.
* For long processes, this rate may be one process per hour ; for short transactions, it may be ten processes per second.

③ Turnaround time :
* It is how long it takes to execute the process completely.
* The interval from the time of submission of a process to the time of completion is known as turnaround time.
* Turnaround time is the sum of periods spent waiting to get into memory, waiting in ready queue, executing on the CPU doing the I/O.

④ Waiting time:
* Waiting time is the sum of periods spent waiting in the ready queue.
* Waiting time must be less.

⑤ Response time:
* It is a time from the submission of the request until the first response is produced.
* It is the time it takes to start responding, not the time it takes to complete.

Scheduling algorithms:-

* First Come First Serve (FCFS) scheduling algorithm
It is non-preemptive and simplest scheduling algorithm. In this, the process that requests the CPU first is allocated CPU first. Implementation of FCFS is managed with a FIFO queue.

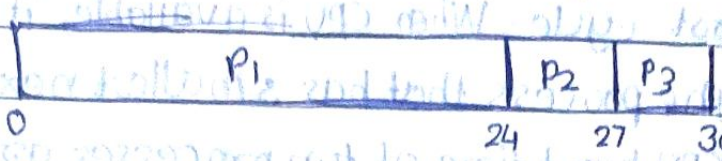
Advantage:- The code is simple to write and understand

Disadvantage:- The average waiting time is long.

Example:

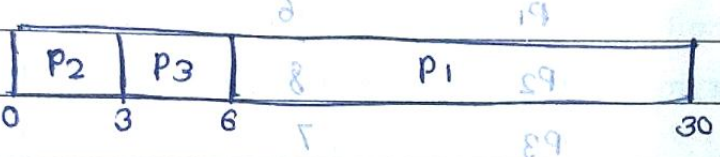
Process	Burst Time
P_1	24
P_2	3
P_3	3

If the processes in the order P_1, P_2, P_3, the result shown in the following Gantt chart:



Average waiting time = (0+24+27)/3
= 51/3 = 17 mS

If process arrive in order P_2, P_3, P_1 then



Average waiting time = (0+3+6)/3
= 9/3 = 3 mS

The average waiting time under an FCFS is generally not minimal and may vary greatly. There is a 'convoy effect' as all the processes wait for one big process. The effect results in lower CPU and device utilization than might be possible if the shorter process were allowed to go first. It is troublesome for time-sharing systems.

In first gantt chart on this page:
Waiting time for P1 = 0
Waiting time for P2 = 24
Waiting time for P3 = 27
Divided by no. of processes

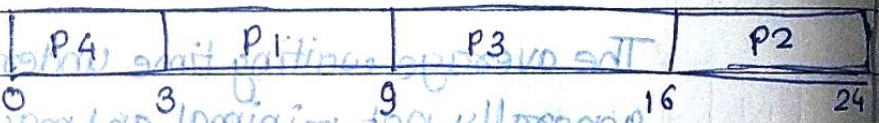
Shortest Job First Scheduling algorithm: (SJF):

It is associated with the length of next CPU burst cycle. When CPU is available, it is assigned to the process that has smallest next CPU burst. If CPU burst time of two processes are same, then FCFS is used to break the time.

Example:-

Process	Burst time
P_1	6
P_2	8
P_3	7
P_4	3

By using Gantt chart:



Avg. waiting time = (0+3+9+16)/4

= (28)/4 = 7 mS

waiting time of P4 = 0

waiting time of P1 = 3

waiting time of P3 = 9

waiting time of P2 = 16

Divided by no. of processes.

Advantages: ① Most optimal and gives minimum average waiting time

② SJF scheduling is frequently used in long-term scheduling.

Disadvantages :- ① Difficulty to know the next CPU request.

② It cannot be implemented at the level of short term scheduling.

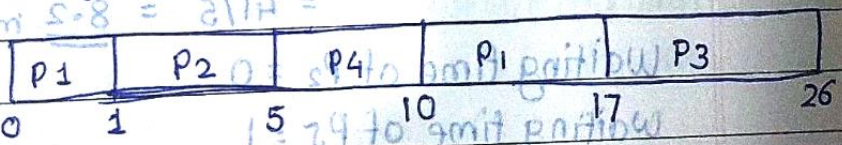
Shortest Remaining Time First / Preemptive SJF scheduling Algorithm (SRTN) :-

A preemptive SJF algorithm will preempt the currently executing process if the new processes CPU burst time is less than what is left of the currently executing process.

Example :-

Process	Arrival Time	Burst time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

By using Gantt chart :



$$\text{Avg. waiting time} = \frac{[(10-1)+(1-1)+(17-2)+(5-3)]}{4}$$

$$= [9+0+15+2]/4$$

$$= 26/4 = 6.5 \text{ ms}$$

In case non-preemptive, the waiting time will result in 7.75 ms.

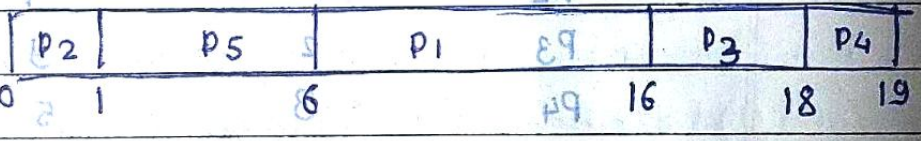
Priority Schedulling :-

SJF algorithm is a special case of priority schedulling algorithm. A process with highest priority is allocated to the CPU first. Equal priority processes are schedulled by FCFS order. Low the number is, high it is in priority.

Example :-

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

By using Gantt chart :



Avg. waiting time = (0 + 1 + 6 + 16 + 18) / 5

= 41/5 = 8.2 mS

Waiting time of P_2 = 0

waiting time of P_5 = 1

Waiting time of P_1 = 6

Waiting time of P_3 = 16

Waiting time of P_4 = 18

Divided by no. of processes

Disadvantage :- A major problem with priority scheduling algorithm is indefinite blocking or starvation. A process which is ready to run but waiting for CPU can be considered blocked. A priority scheduling algorithm can leave some low-priority processes waiting indefinitely.

The solution to the problem of indefinite blocking is aging. Aging involves gradually increasing the priority of processes that wait in the system for a long time.

Round-Robin Scheduling (RR) :-

It is designed especially for time sharing systems. It is similar to FCFS scheduling but preemption is added to switch between processes. A small unit of time, called a time quantum or time slice, is defined. A time quantum is generally from 10 to 100 milliseconds in length.

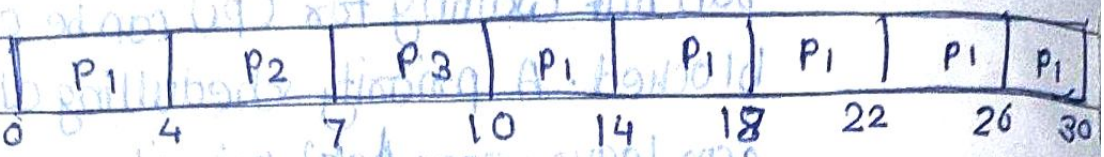
Disadvantage :- Average waiting time is long.

Example :-

Process	Burst Time
P_1	24
P_2	3
P_3	3

If we use the quantum time of 4 milliseconds then,

By using Gantt chart:



Avg. waiting time = [(10-4) + 4 + 7]/3

= (6+4+7)/3

= 17/3 = 5.66 mS

Waiting time of P1 = (10-4) = 6

Waiting time of P2 = 4

Waiting time of P3 = 7

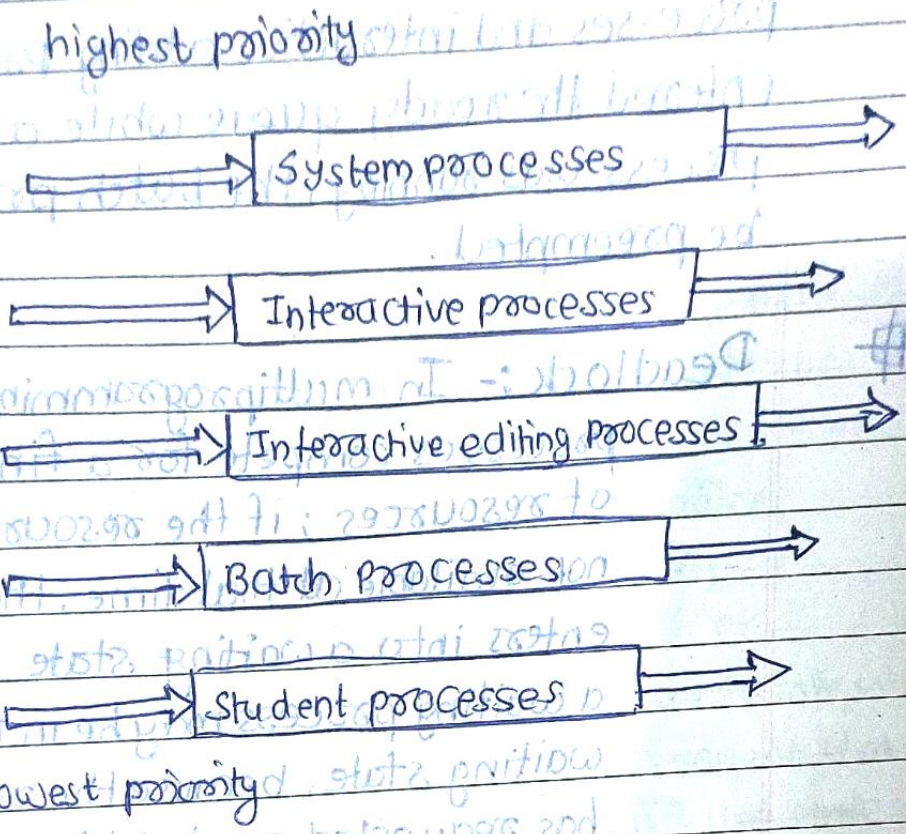
In RR scheduling algorithm, no process is allocated to the CPU for more than 1 time quantum in the row (unless it is the only runnable process). If the process's CPU burst exceeds 1 time quantum, that process is preempted and put back in the ready queue. Therefore RR scheduling algorithm is preemptive

Disadvantages: ① Performance depends heavily on the size of time quantum.

② Turnaround time also depends on size of time quantum

Multilevel Queue scheduling :-

A multilevel queue scheduling algorithm partitions the ready queue into several seperate queues as shown in figure.



Each queue has its own scheduling algorithm. For example, seperate queues might be used for foreground and background processes. The foreground queue might be scheduled by an RR algorithm while the background queue is scheduled by FCFS algorithm. In addition, there must be scheduling among queues, which is commonly implemented as fixed-priority preemptive scheduling. For example, the foreground queue may have absolute priority over the background queue.

As shown in diagram in previous page, each queue has absolute priority over lower-priority queues. For example, no process in the batch queue could run unless the queues for system processes, interactive processes and interactive editing process entered the ready queue while a batch process was running, the batch process will be preempted.

Deadlock :- In multiprogramming several processes compete for a finite number of resources; if the resources are not available at that time, the process enters into a waiting state. Sometimes a waiting process may be in forever waiting state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock.

System model :- The system consist of a finite number of resources to be distributed among a number of competing processes. The resources may be partitioned into several types, each consisting of some number of identical instances. If a system has 2 CPU's, then resource type CPU has 2 instances.

There are various synchronization tools, such as mutex locks and semaphores. These tools are also considered system resources, and they are the common sources of deadlock.

A process must request a resource before using it and must release the resource ~~before~~ after using it. The number of resources requested may not exceed the total number of resources.

Under normal mode of operation:

* Request :- The process requests a resource
* Use :- The process can operate the resource
* Release :- The process releases the resource.

Ex :- Request() and release() device,
Open() and close() file &
free() memory system cells.

A set of processes is in a deadlocked state when every process is in the waiting for an event that can be caused only by another process in the ~~state~~ set. The mainly concerned events are resource acquisition and release. The resources may be either physical resources or logical resources.

Necessary conditions for deadlock :-

A deadlock situation can arise if the following four conditions hold simultaneously in a system.

- Mutual exclusion
- Hold & wait
- No preemption
- Circular wait.

— (Discussed further) —

Deadlock handling methods :-

We can deal with the deadlocks by using one of the three ways

- We can use a protocol to prevent or avoid deadlocks, ensure that system will never enter into deadlocked state.
- We can allow system to enter into a deadlock state, detect it and recover.
- We can ignore the problem altogether and pretend that deadlocks never occur in the system.

The third solution is the one used by most operating systems, including Linux and Windows. Then it is upto the application developer to write the programs that handle deadlocks.

To ensure that deadlocks never occur, the system can use either a deadlock prevention or a deadlock avoidance scheme.

Deadlock prevention :- It provides a set of methods to ensure that at least one of the necessary conditions cannot hold. These methods prevents deadlocks by constraining how requests for resources can be made.

Date

Deadlock avoidance requires that the operating system be given additional ~~operation~~ information in advance concerning which resources a process will request and use during its lifetime. With this additional knowledge, the operating system can decide for each request whether or not the process should wait.

Deadlock prevention :-

For a deadlock to occur, each of the four necessary conditions must hold. By ensuring that at least one of these conditions ~~occurs~~ cannot hold, we can prevent occurance of deadlock.

Examining 4 necessary conditions :-

① Mutual exclusion :-

The mutual exclusion condition must hold. That is, at least one resource should be non-shareable. Shareable resources do not require mutually exclusive access and thus cannot be involved in a deadlock.

Read-only files are good example of a shareable resource. If several processes attempt to open a read-only file at a same time, they can be granted simultaneous access to the simultaneous access. A process never needs to wait for shareable resource. In general, however, we cannot prevent deadlocks by denying the mutual-exclusion condition, because some resources can be non-shareable.

Date

② Hold and wait :

To ensure that the hold and wait condition never occurs in the system, we must guarantee that, whenever process requests resource, it does not hold any other resources.

One protocol allows process to request resources only when it has none. A process may request some resources and use them. Before it can request any additional resources, it must release all the resources that it is currently allocated.

An alternative protocol, that we can use, requires each process to request and be allocated all its resources, before it begins its execution.

Both these two protocols have two main disadvantages :

① Resource utilization may be low

② Starvation is possible

③ No Preemption :-

The third necessary condition for deadlocks is that there be no preemption of resources that have already been allocated.

To ensure that this condition does not hold, we can use following protocol:

If a process is holding some resources, and requests another resource that cannot be immidiately allocated to it, then all resources the process is currently holding is preempted.

This protocol is often applied to resources whose state can be easily saved and restored later. It cannot generally be applied to such resources as mutex locks or semaphores.

④ Circular wait :-

The fourth and final condition for deadlocks is the circular-wait condition. One way to ensure that this condition never holds is to impose a total ordering of all resource types and the require that each process requests resources in an increasing order of enumeration.

Alternatively, we can require that a process requesting an instance of resource type Rj must have released any resources Ri such that $F(Ri) \geq F(Rj)$. Note also that if several instances of the same resource type are needed, a single request for all of them must be issued.

If these two protocols are used, the circular wait condition cannot hold.

Deadlock avoidance :-

A deadlock avoidance algorithm dynamically examines the resource allocation state to ensure that a circular-wait condition can never exist.

The resource allocation state is defined by the number of available and allocated resources and the maximum demands of the process.

Safe state :- A state is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock. More formally, a system is in a safe state only if there exists a safe sequence.

Banker's Algorithm :-

Banker's Algorithm (deadlock avoidance algorithm) is applicable to a system with multiple resources of each type. It is less efficient than the resource allocation graph scheme.

The name was given to it because it can be used in banking system to ensure that the bank never allocated its available cash in such a way that it could no longer satisfy the needs of all its customers.

When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This numbe may not exceed the total number of resources in the system.

Data structures used in banker's algorithm:
Here n is the number of processes in the system and m is the number of resource types.

- Available :- A vector of length m indicates the number of available resources of each type. If Available [j] equals k, then k instances of resource type Rj are available.
- Max :- An n x m matrix defines the maximum demand of each process. If Max [i][j] equals k, then process Pi may request at most k instances of resource type Rj.
- Allocation :- An nxm matrix defines the number of resources of each type currently allocated to each process. If Allocation [i][j] equals k, then process Pi is allocated k instances of resource type Rj.
- Need :- An nxm matrix indicates the remaining resource need of each process. If Need [i][j] equals k, then process Pi may need k more instances of resource type Rj to complete its task. Note that Need [i][j] equals Max[i][j] - Allocation[i][j]

These data structures vary over time in both size and value.