OPERATING SYSTEM

ULEMU MPONELA
DMI ST JOHN THE BAPTIST UNIVERSITY : UNIT-II

Table of Contents

Unit II CPU Scheduling:

CPU Schedulers - Scheduling Criteria - Scheduling Algorithms. Process Synchronization: Critical - Section Problem – Semaphores. Deadlocks: Characterization - Methods for Handling Deadlocks - Deadlock Prevention - Avoidance - Detection -Recovery.

CPU SCHEDULER

CPU scheduling is a process that allows one process to use the CPU while the execution of another process is on hold (in waiting state) due to unavailability of any resource like I/O, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast, and fair.

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the short-term scheduler (or CPU scheduler). The scheduler selects from among the processes in memory that are ready to execute and allocates the CPU to one of them.

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 utilisation**

The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible. Theoretically, CPU utilisation 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.

SCHEDULING ALGORITHMS

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 we 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-pre-emptive or pre-emptive. Non-pre-emptive algorithms are designed so that once a process enters the running state, it cannot be pre-empted until it completes its allotted time, whereas the pre-emptive scheduling is based on priority where a scheduler may pre-empt a low priority running process anytime when a high priority process enters into a ready state.

i. First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-pre-emptive, pre-emptive 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

ii. Shortest Job Next (SJN)

- 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

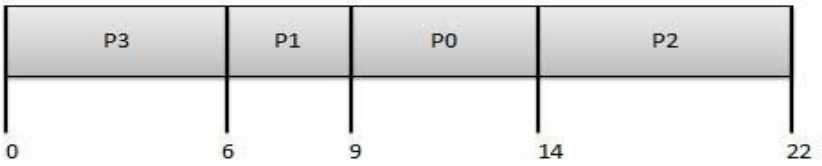
iii. Priority Based Scheduling

- Priority scheduling is a non-pre-emptive 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 first and so on.
- Processes with same priority are executed on first come first 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

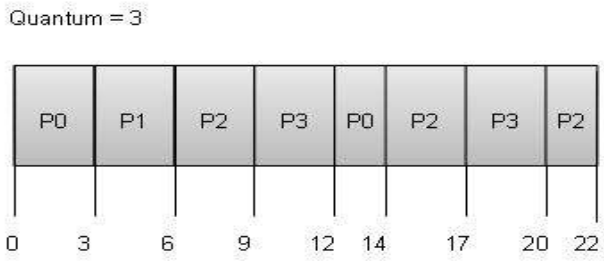
Average Wait Time: (0 + 10 + 12 + 2)/4 = 24 / 4 = 6

iv. Shortest Remaining Time

- Shortest remaining time (SRT) is the pre-emptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be pre-empted 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.

v. Round Robin Scheduling

- Round Robin is the pre-emptive 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 pre-empted and other process executes for a given time period.
- Context switching is used to save states of pre-empted processes.



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

vi. 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.

PROCESS SYNCHRONIZATION

Process Synchronization means coordinating the execution of processes such that no two processes access the same shared resources and data. It is required in a multi-process system where multiple processes run together, and more than one process tries to gain access to the same shared resource or data at the same time.

Changes made in one process aren't reflected when another process accesses the same shared data. It is necessary that processes are synchronized with each other as it helps avoid the inconsistency of shared data.

For example: A process P1 tries changing data in a particular memory location. At the same time another process P2 tries reading data from the same memory location. Thus, there is a high probability that the data being read by the second process is incorrect.

CRITICAL SECTION PROBLEMS

The critical section is a code segment where the shared variables can be accessed. An atomic action is required in a critical section i.e. only one process can execute in its critical section at a time. All the other processes have to wait to execute in their critical sections.

Critical Section is the part of a program which tries to access shared resources. That resource may be any resource in a computer like a memory location, Data structure, CPU or any IO device.

The critical section cannot be executed by more than one process at the same time; operating system faces the difficulties in allowing and disallowing the processes from entering the critical section.

The critical section problem is used to design a set of protocols which can ensure that the Race condition among the processes will never arise.

In order to synchronize the cooperative processes, our main task is to solve the critical section problem.

Solution to the Critical Section Problem

The critical section problem needs a solution to synchronize the different processes. The solution to the critical section problem must satisfy the following conditions –

- Mutual Exclusion

  Mutual exclusion implies that only one process can be inside the critical section at any time. If any other processes require the critical section, they must wait until it is free.

- Progress

  Progress means that if a process is not using the critical section, then it should not stop any other process from accessing it. In other words, any process can enter a critical section if it is free.

- Bounded Waiting

  Bounded waiting means that each process must have a limited waiting time. It should not wait endlessly to access the critical section.

- **Semaphores**

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

The definitions of wait and signal are as follows –

- **Wait**

  The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

- **Signal**

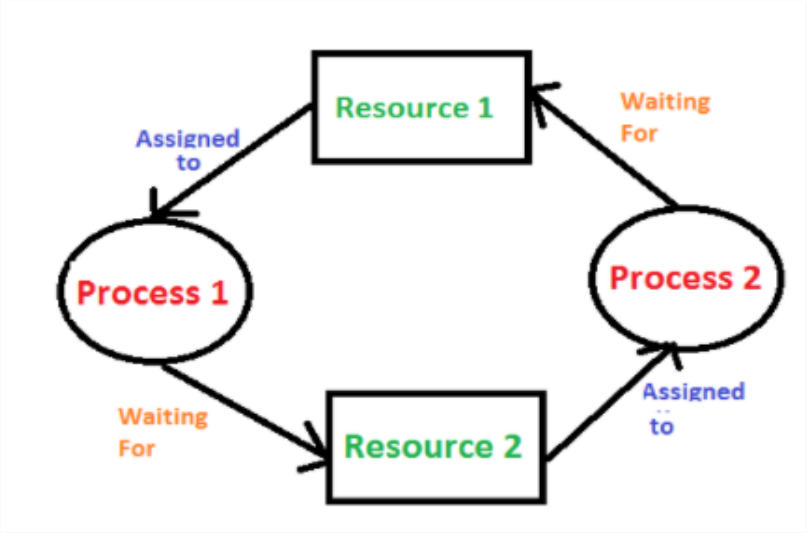
The signal operation increments the value of its argument S

DEADLOCKS

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

Consider an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other. A similar situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s).

For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



METHODS FOR HANDLING DEADLOCKS

There are a number of methods for handling deadlocks. These are:

i. Deadlock prevention

It is very important to prevent a deadlock before it can occur. So, the system checks each transaction before it is executed to make sure it does not lead to deadlock. If there is even a slight chance that a transaction may lead to deadlock in the future, it is never allowed to execute.

ii. Deadlock Avoidance

It is better to avoid a deadlock rather than take measures after the deadlock has occurred. The wait for graph can be used for deadlock avoidance. This is however only useful for smaller databases as it can get quite complex in larger databases.

iii. Deadlock Detection

A deadlock can be detected by a resource scheduler as it keeps track of all the resources that are allocated to different processes. After a deadlock is detected, it can be resolved using the following methods

- All the processes that are involved in the deadlock are terminated. This is not a good approach as all the progress made by the processes is destroyed.
- Resources can be pre-empted from some processes and given to others till the deadlock is resolved.

iv. Deadlock Recovery

When a Deadlock Detection Algorithm determines that a deadlock has occurred in the system, the system must recover from that deadlock. There are two main ways of recovering:

- Either by process termination. Whereby we kill one or more processes
- Or by resource pre-emption, whereby we pre-empt some resources from processes and give those resources to other processes.