CPU Scheduling

In the uniprogrammming systems like MS DOS, when a process waits for any I/O operation to be done, the CPU remains idol. This is an overhead since it wastes the time and causes the problem of starvation. However, In Multiprogramming systems, the CPU doesn't remain idle during the waiting time of the Process and it starts executing other processes. Operating System has to define which process the CPU will be given.

In Multiprogramming systems, the Operating system schedules the processes on the CPU to have the maximum utilization of it and this procedure is called **CPU scheduling**. The Operating System uses various scheduling algorithm to schedule the processes.

This is a task of the short term scheduler to schedule the CPU for the number of processes present in the Job Pool. Whenever the running process requests some IO operation then the short term scheduler saves the current context

procedure is called **CPU scheduling**. The Operating System uses various scheduling algorithm to schedule the processes.

This is a task of the short term scheduler to schedule the CPU for the number of processes present in the Job Pool. Whenever the running process requests some IO operation then the short term scheduler saves the current context of the process (also called PCB) and changes its state from running to waiting. During the time, process is in waiting state; the Short term scheduler picks another process from the ready queue and assigns the CPU to this process. This procedure is called **context switching**.

What is saved in the Process Control Block?

The Operating system maintains a process control block during the lifetime of the process. The Process control block is deleted when the process is terminated or killed. There is the following information which is saved in the process control block and is changing with the

Before working on the difference between all three schedulers, let us recall the concept of process scheduler.

Process scheduler is a part of the Operating system which schedules the process. If the process is in ready, waiting and running state it schedules it correctly and is also responsible for the allocation of the CPU processor to a specific task within a time interval.

Also, when the task or process is completed it doesn't allow the CPU to sit idle. It will allocate the other task depending on its current state. If the process was in ready state it will allocate it to the CPU in this way it always keeps the CPU busy all of the time.

Characteristics of good process scheduler

The characteristics of a good process scheduler are as follows −

- It increases the utilisation of the CPU as it keeps it busy all the time by assigning the ready state processes

Types of Schedulers

There are three types of schedulers available which are as follows −

- Long Term Scheduler
- Short Term Scheduler
- Medium Term Scheduler

Differences

The major differences between long term, medium term and short term scheduler are as follows −

Long term scheduler	Medium term scheduler	Short term scheduler
Long term scheduler is a job scheduler.	Medium term is a process of swapping schedulers.	Short term scheduler is called a CPU scheduler.
The speed of long term is lesser than the short term.	The speed of medium term is in between short and long term scheduler.	The speed of short term is fastest among the other two.
Long term controls the degree of multiprogramming.	Medium term reduces the degree of multiprogramming.	The short term provides lesser control over the degree of multiprogramming.
The long term is almost nil or minimal in the time sharing system.	The medium term is a part of the time sharing system.	Short term is also a minimal time sharing system.

term, medium term and short term scheduler are as follows −

Long term scheduler	Medium term scheduler	Short term scheduler
Long term scheduler is a job scheduler.	Medium term is a process of swapping schedulers.	Short term scheduler is called a CPU scheduler.
The speed of long term is lesser than the short term.	The speed of medium term is in between short and long term scheduler.	The speed of short term is fastest among the other two.
Long term controls the degree of multiprogramming.	Medium term reduces the degree of multiprogramming.	The short term provides lesser control over the degree of multiprogramming.
The long term is almost nil or minimal in the time sharing system.	The medium term is a part of the time sharing system.	Short term is also a minimal time sharing system.
The long term selects the processes from the pool and loads them into memory for execution.	Medium term can reintroduce the process into memory and execution can be continued.	Short term selects those processes that are ready to execute.

Bhanu Priya
Updated on 30-Nov-2021 12:18:57

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

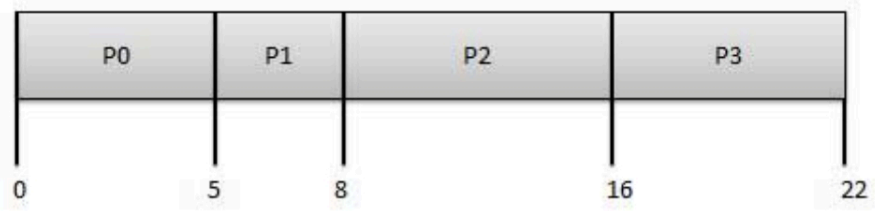
These algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low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

First Come First Serve (FCFS)

First Come First Serve (FCFS)

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

Average Wait Time: (0+4+6+13) / 4 = 5.7

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

Shortest Job Next (SJN)

- This is also known as **shortest job first**, 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.

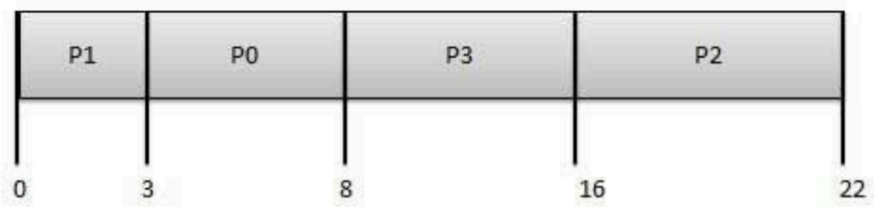
Given: Table of processes, and their Arrival time, Execution time

Process	Arrival Time	Execution Time	Service Time
P0	0	5	0
P1	1	3	5

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

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

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

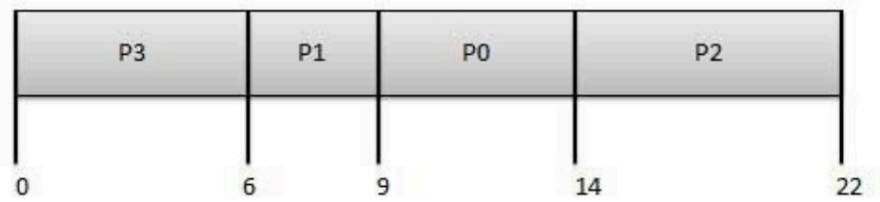
Priority Based Scheduling

- 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 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

Shortest Remaining Time

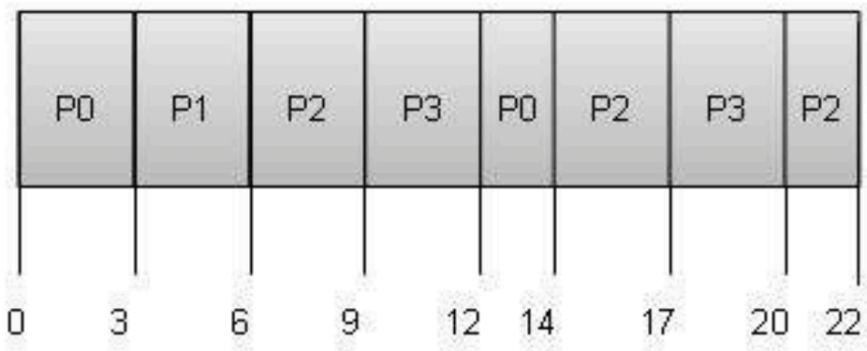
- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.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

- 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.

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.

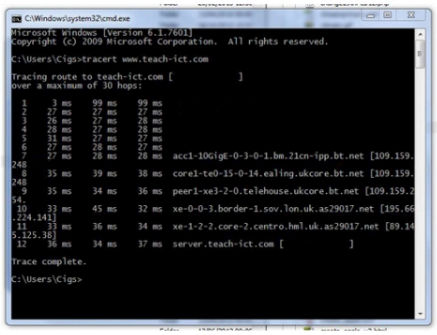
- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

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.

Previous Page Next Page

4. Interactive operating system

An interactive operating system is one that allows the user to directly interact with the operating system whilst one or more programs are running.



There will be an user interface in place to allow this to happen. It could be a command line style of interface or it could be a graphical interface.

Most operating systems are of this type

4. Interactive - Teach-ICT A2 Level ICT OCR exam board - Operating Systems

About featured snippets Feedback

People also ask

What are examples of interactive system?

However, the term 'interactive system' can be applied to a much broader range of devices, for example:

- Mobile telephones.
- Cash dispensing machines.
- The World Wide Web.
- Car navigation systems.
- Video recorders.
- Machines driven call centres (e.g. for telephone banking).
- Workflow system to co-ordinate a teams work-efforts.

https://www.cs.uct.ac.za › htmls

Interactive Systems - University of Cape Town

More results

- Lack of protection.

2. Interactive Operating System

In an Interactive operating system, there is a direct interaction between the user and the computer. Mostly, all personal computers use Interactive operating systems. In this kind of operating system, the user enters some command in the system and the system works according to it.

3. Real-Time Operating System

An RTOS is a data processing system whose response time to the input is very short. RTOS is also known as the brain of the real-time system because of its

Concurrent Processes in Operating System

Concurrent processing is a computing model in which multiple processors execute instructions simultaneously for better performance. Concurrent means, which occurs when something else happens. The tasks are broken into subtypes, which are then assigned to different processors to perform simultaneously, sequentially instead, as they would have to be performed by one processor. Concurrent processing is sometimes synonymous with parallel processing. The term real and virtual concurrency in concurrent processing:

1. **Multiprogramming Environment:** In a multiprogramming environment, there are multiple tasks shared by one processor. While a virtual concept can be achieved by the operating system, if the processor is allocated for each individual task, the virtual concept is visible if each task has a dedicated

concurrency in concurrent processing:

1. **Multiprogramming Environment:** In a multiprogramming environment, there are multiple tasks shared by one processor. While a virtual concept can be achieved by the operating system, if the processor is allocated for each individual task, the virtual concept is visible if each task has a dedicated processor. The multilayer environment is shown in figure.

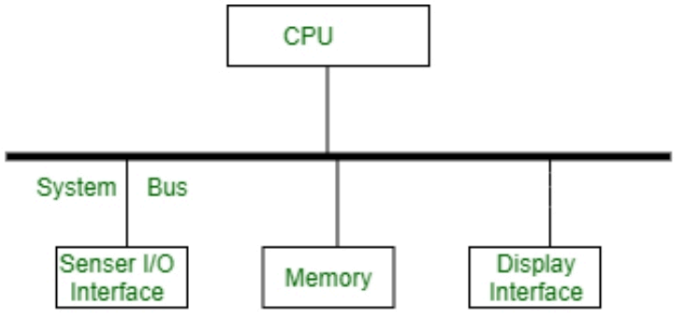


Figure - Multiprogramming (Single CPU) Environment

2. **Multiprocessing Environment :** In multiprocessing environment two or more processors are used with shared memory. Only one virtual address space is used, which is common for all processors. All tasks reside in shared

2. **Multiprocessing Environment :** In multiprocessing environment two or more processors are used with shared memory. Only one virtual address space is used, which is common for all processors. All tasks reside in shared memory. In this environment, concurrency is supported in the form of concurrently executing processors. The tasks executed on different processors are performed with each other through shared memory. The multiprocessing environment is shown in figure.

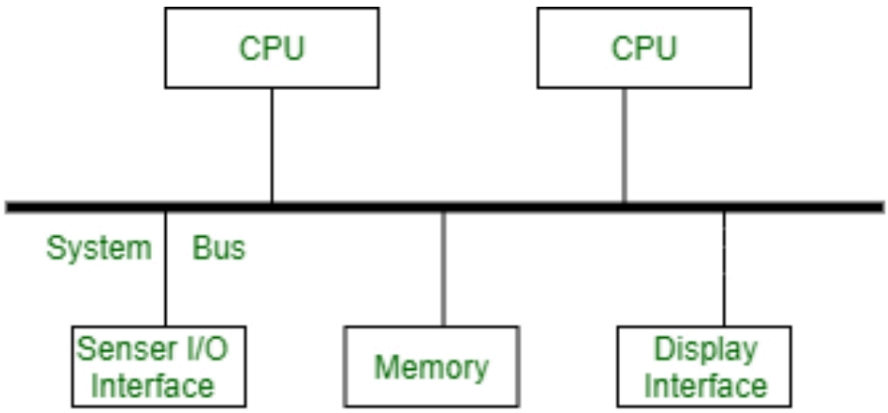


Figure - Multiprocessing Environment

3. **Distributed Processing**

Figure - Multiprocessing Environment

3. **Distributed Processing Environment :** In a distributed processing environment, two or more computers are connected to each other by a communication network or high speed bus. There is no shared memory between the processors and each computer has its own local memory. Hence a distributed application consisting of concurrent tasks, which are distributed over network communication via messages. The distributed processing environment is shown in figure.

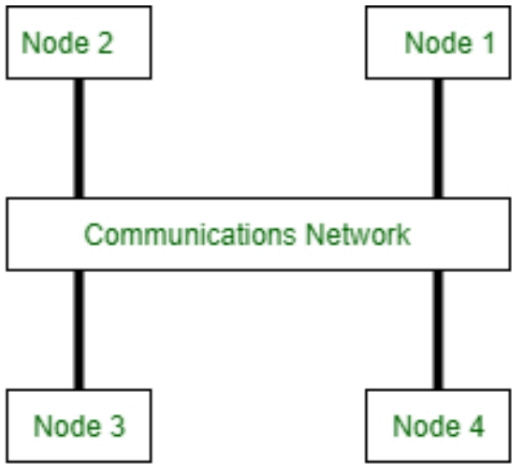


Figure - Distributed processing Environment

Article Tags : Operating Systems | Write From Home

Precedence Graph in Operating System

Prerequisite – Process Synchronization **Precedence Graph** is a directed acyclic graph which is used to show the execution level of several processes in operating system. It consists of nodes and edges. Nodes represent the processes and the edges represent the flow of execution.

Properties of Precedence Graph :

Following are the properties of Precedence Graph:

- It is a directed graph.
- It is an acyclic graph.
- Nodes of graph correspond to individual statements of program code.
- Edge between two nodes represents the execution order.
- A directed edge from node A to node B shows that statement A executes first and then Statement B executes.

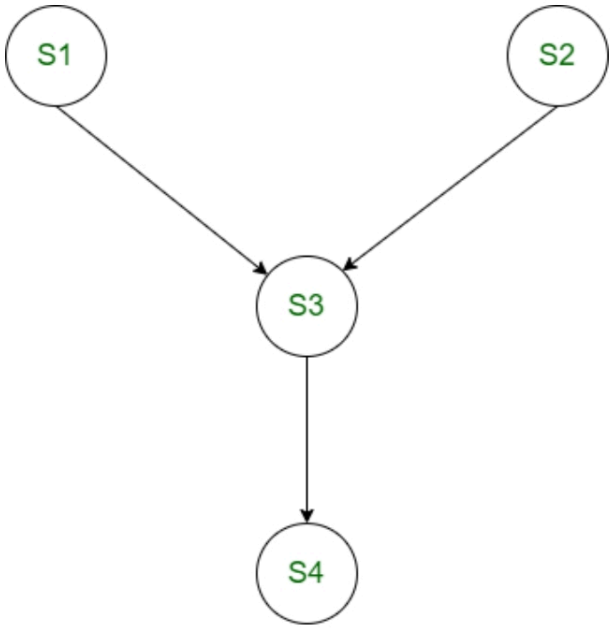
Consider the following code:

Consider the following code:

```
S1 : a = x + y;
S2 : b = z + 1;
S3 : c = a - b;
S4 : w = c + 1;
```

If above code is executed concurrently, the following precedence relations exist:

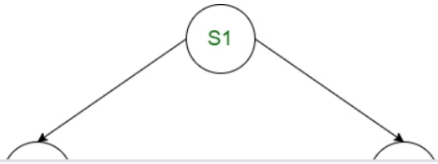
- c = a – b cannot be executed before both a and b have been assigned values.
- w = c + 1 cannot be executed before the new values of c has been computed.
- The statements a = x + y and b = z + 1 could be executed concurrently.



Example: Consider the following precedence relations of a program:

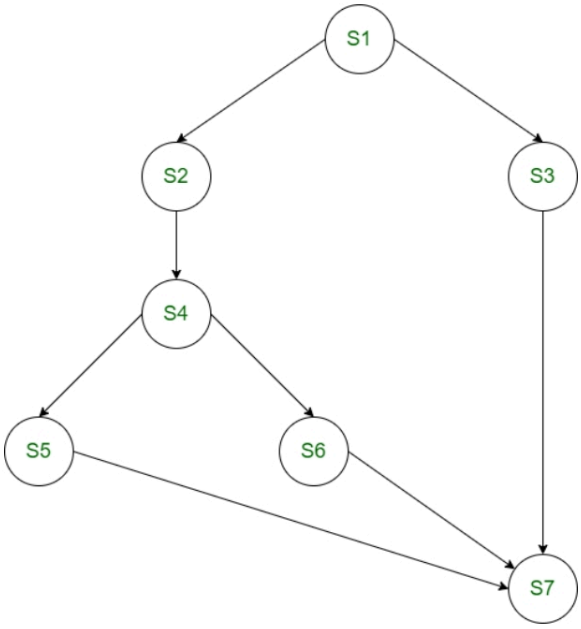
1. S2 and S3 can be executed after S1 completes.
2. S4 can be executed after S2 completes.
3. S5 and S6 can be executed after S4 completes.
4. S7 can be executed after S5, S6 and S3 complete.

Solution:



1. S2 and S3 can be executed after S1 completes.
2. S4 can be executed after S2 completes.
3. S5 and S6 can be executed after S4 completes.
4. S7 can be executed after S5, S6 and S3 complete.

Solution:



Article Tags : GATE CS Operating Systems Process Synchronization

The Critical Section Problem

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. We need to provide a solution in such a way that the following conditions can be satisfied.

Requirements of Synchronization mechanisms

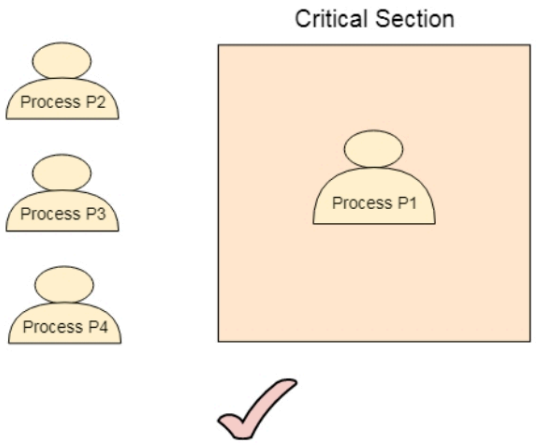
section problem. We need to provide a solution in such a way that the following conditions can be satisfied.

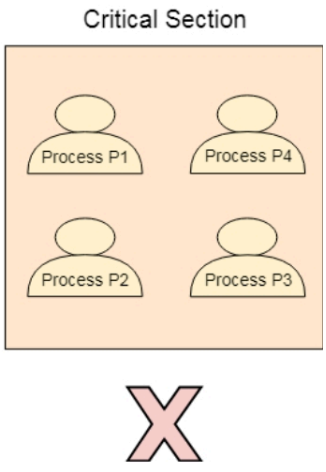
Requirements of Synchronization mechanisms

Primary

1. **Mutual Exclusion**

   Our solution must provide mutual exclusion. By Mutual Exclusion, we mean that if one process is executing inside critical section then the other process must not enter in the critical section.





2. **Progress**

   Progress means that if one process ... to execute into critical ... it should not stop other

2. **Progress**

   Progress means that if one process doesn't need to execute into critical section then it should not stop other processes to get into the critical section.

Secondary

1. **Bounded Waiting**

   We should be able to predict the waiting time for every process to get into the critical section. The process must not be endlessly waiting for getting into the critical section.

2. **Architectural Neutrality**

   Our mechanism must be architectural natural. It means that if our solution is working fine on one architecture then it should also run on the other ones as well.

Semaphores in Process Synchronization

Semaphore was proposed by Dijkstra in 1965 which is a very significant technique to manage concurrent processes by using a simple integer value, which is known as a semaphore. A semaphore is simply an integer variable that is shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment. Semaphores are of two types:

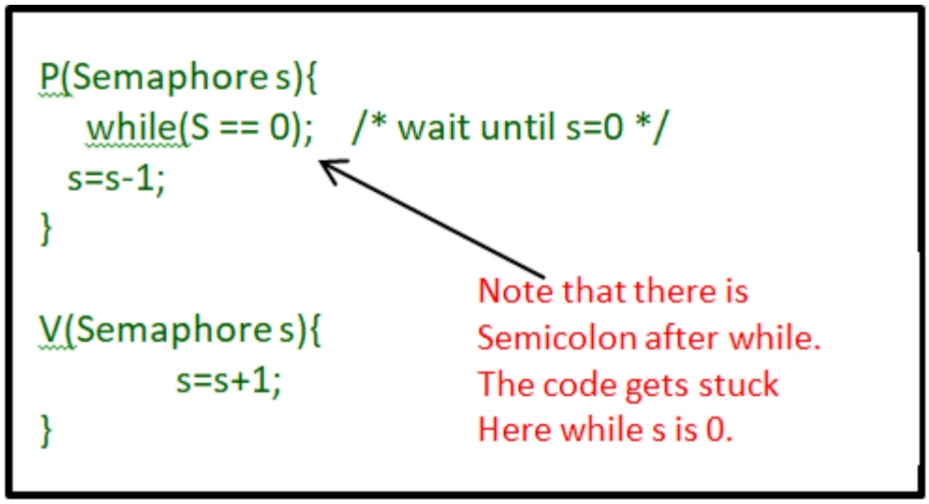
1. **Binary Semaphore –**
   This is also known as mutex lock. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problems with multiple processes.
2. **Counting Semaphore –**
   Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances

section problems with multiple processes.

2. **Counting Semaphore –**
   Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

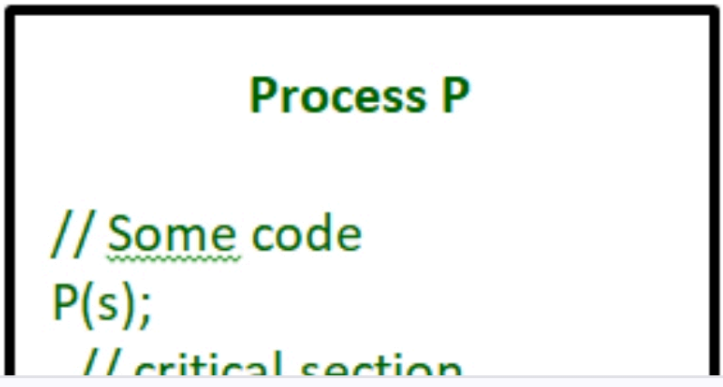
Now let us see how it does so.

First, look at two operations that can be used to access and change the value of the semaphore variable.



Some point regarding P and V operation:

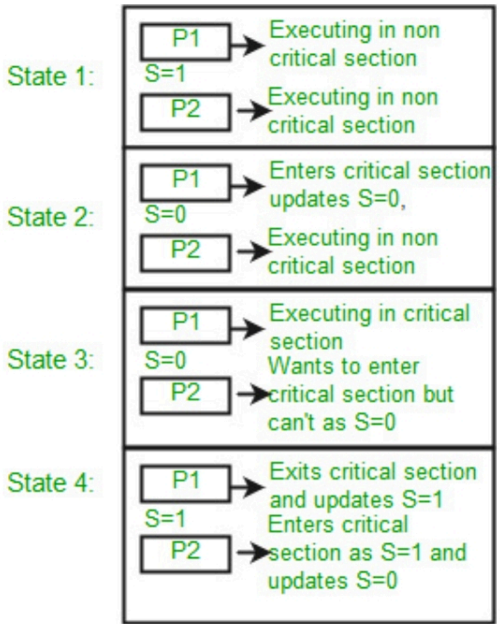
1. P operation is also called wait, sleep, or down operation, and V operation is also called signal, wake-up, or up operation.
2. Both operations are atomic and semaphore(s) is always initialized to one. Here atomic means that variable on which read, modify and update happens at the same time/moment with no pre-emption i.e. in-between read, modify and update no other operation is performed that may change the variable.
3. A critical section is surrounded by both operations to implement process synchronization. See the below image. The critical section of Process P is in between P and V operation.



```
V(s);
// remainder section
```

Now, let us see how it implements mutual exclusion. Let there be two processes P1 and P2 and a semaphore s is initialized as 1. Now if suppose P1 enters in its critical section then the value of semaphore s becomes 0. Now if P2 wants to enter its critical section then it will wait until s > 0, this can only happen when P1 finishes its critical section and calls V operation on semaphore s.

This way mutual exclusion is achieved. Look at the below image for details which is Binary semaphore.



Limitations :

1. One of the biggest limitations of semaphore is priority inversion.
2. Deadlock, suppose a process is trying to wake up another process which is not in a sleep state. Therefore, a deadlock may block indefinitely.
3. The operating system has to keep track of all calls to wait and to signal the semaphore.

Problem in this implementation of semaphore :

The main problem with semaphores is that they require busy waiting, If a process is in the critical section, then other processes trying to enter the critical section will be waiting until the critical section is not occupied by any process. Whenever any process waits then it continuously checks for semaphore value (look at this line while(s==0); in P operation) and