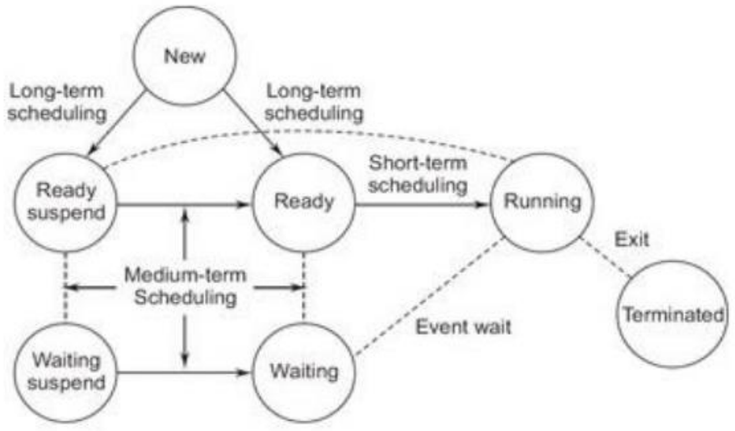
Unit-3rd – CPU Scheduling

Scheduling

The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization. The objective of time sharing is to switch the CPU among processes so frequently. In uniprocessor only one process is running. A process migrates between various scheduling queues throughout its lifetime. The process of selecting processes from among these queues is carried out by a **scheduler**. The aim of processor scheduling is to assign processes to be executed by the processor. Scheduling affects the performance of the system, because it determines which process will wait and which will progress.

Long-term-Scheduling: - Long term scheduling is performed when a new process is created. It is shown in the figure below. If the number of ready processes in the ready queue becomes very high, then there is an overhead on the operating system (i.e., processor) for maintaining long lists, context switching and dispatching increases. Therefore, allow only limited number of processes in to the ready queue. The "long-term scheduler" managers this. Long-term scheduler determines which programs are admitted into the system for processing. Once when admit a process or job, it becomes process and is added to the queue for the short-term scheduler. In some systems, a newly created process begins in a swapped-out condition; in which case it is added to a queue for the medium-term scheduler scheduling manage queues to minimize queuing delay and to optimize performance.



The long-term scheduler limits the number of processes to allow for processing by taking the decision to add one or more new jobs, based on FCFS (First-Come, first-serve) basis or priority or execution time or Input/output requirements. Long-term scheduler executes relatively infrequently.

Medium-term-Scheduling: - Medium-term scheduling is a part of the swapping function. When part of the main memory gets freed, the operating system looks at the list of suspend ready processes, decides which one is to be swapped in (depending on priority, memory and other resources required, etc.). This scheduler works in close conjunction with the long-term scheduler. It will perform the swapping-in function among the swapped-out processes. Medium-term scheduler executes somewhat more frequently.

Short-term-Scheduling: - Short-term scheduler is also called as dispatcher. Short-term scheduler is invoked whenever an event occurs, that may lead to the interruption of the current running process. For example clock interrupts, I/O interrupts, operating system calls, signals, etc. Short-term scheduler executes most frequently. It selects from among the processes that are ready to execute and allocates the CPU to one of them. It must select a new process for the CPU frequently. It must be very fast.

Scheduling Criteria

Scheduling criteria is also called as scheduling methodology. Key to multiprogramming is scheduling. Different CPU scheduling algorithm have different properties .The criteria used for comparing these algorithms include the following:

CPU-Utilization: - Keep the CPU as busy as possible. It ranges from 0 to 100%. In practice, it ranges from 40 to 90%.

Throughput: - Throughput is the rate at which processes are completed per unit of time.

Turnaround-time: - This is the how long a process takes to execute a process. It is calculated as the time gap between the submission of a process and its completion.

Waiting time: - Waiting time is the sum of the time periods spent in waiting in the ready queue.

Response-time: - Response time is the time it takes to start responding from submission time. It is calculated as the amount of time it takes from when a request was submitted until the first response is produced.

Fairness: - Each process should have a fair share of CPU.

Non-preemptive-Scheduling: - In non-preemptive mode, once if a process enters into running state, it continues to execute until it terminates or blocks itself to wait for Input/output or by requesting some operating system service.

Preemptive-Scheduling: - In preemptive mode, currently running process may be interrupted and moved to the ready State by the operating system. When a new process arrives or when an interrupt occurs, preemptive policies may incur greater overhead than non-preemptive version but preemptive version may provide better service.

It is desirable to maximize CPU utilization and throughput, and to minimize turnaround time, waiting time and response time.

Process Control Block (PCB)

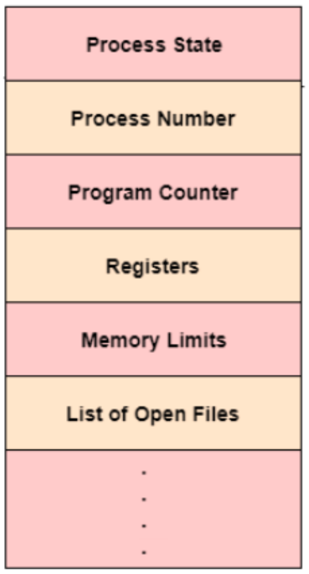
Process Control Block is a data structure that contains information of the process related to it. The process control block is also known as a task control block, entry of the process table, etc.

It is very important for process management as the data structuring for processes is done in terms of the PCB. It also defines the current state of the operating system.

Structure of the Process Control Block

The process control stores many data items that are needed for efficient process management. Some of these data items are explained with the help of the given diagram −

The following are the data items −



Process Control Block (PCB)

1. **Process State: -** This specifies the process state i.e. new, ready, running, waiting or terminated.
2. **Process Number: -** This shows the number of the particular process.
3. **Program Counter: -** This contains the address of the next instruction that needs to be executed in the process.
4. **Registers: -** This specifies the registers that are used by the process. They may include accumulators, index registers, stack pointers, general purpose registers etc.
5. **List of Open Files: -** These are the different files that are associated with the process
6. **CPU Scheduling Information: -** The process priority, pointers to scheduling queues etc. is the CPU scheduling information that is contained in the PCB. This may also include any other scheduling parameters.
7. **Memory Management Information: -** The memory management information includes the page tables or the segment tables depending on the memory system used. It also contains the value of the base registers, limit registers etc.
8. **I/O Status Information: -** This information includes the list of I/O devices used by the process, the list of files etc.

9. **Accounting information: -** The time limits, account numbers, amount of CPU used, process numbers etc. are all a part of the PCB accounting information.

10. **Location of the Process Control Block: -** The process control block is kept in a memory area that is protected from the normal user access. This is done because it contains important process information. Some of the operating systems place the PCB at the beginning of the kernel stack for the process as it is a safe location.

Process Address Space

Address space is a space in computer memory. And process Address Space means a space that is allocated in memory for a process. Every process has an address space

Address Space can be of two types

1. Physical Address Space
2. Virtual Address Space

1. Physical address space: - The physical address is the actual location in the memory that exists physically. System accesses the data in the main memory, with the help of physical address. Everything in the computer has a unique physical address. We need to map it to make the address accessible. MMU is responsible for mapping.

2. Virtual Address Space: - Virtual Address Space is an address space that is created outside the main memory inside the virtual memory, and it is created in the hard disk.When our main memory is less and we want to get more benefit from this less memory, then we create virtual memory.

Logical Address is the address is always generated by CPU while running a program. The logical address is also called virtual address. Virtual address does not exist physically, so we can say that physical address is the logical address.

How to access the physical memory location by CPU?

Logical Address is used as a reference to access the physical memory location by CPU. The Memory-Management Unit uses the address-binding methods for mapping the logical address to its corresponding physical address.

PARAMENTER	LOGICAL ADDRESS	PHYSICAL ADDRESS
Basic	ADDRESS generated by CPU is called LOGICAL ADDRESS.	PHYSICAL ADDRESS is the actual location in a memory unit.
Generation	generated by the CPU	Computed by MMU
Access	Logical address can be used to access the physical address by the user.	The user can't directly access physical address but it is possible indirectly.
Visibility	Logical address of a program is visible to the user.	Physical address of program is not visible to the user.
Address Space	Set of all logical addresses generated by CPU are called logical addresses.	Addresses mapped to the corresponding logical addresses.

Scheduling Algorithms

Scheduling algorithms or scheduling policies are mainly used for short-term scheduling. The main objective of short-term scheduling is to allocate processor time in such a way as to optimize one or more aspects of system behavior. For these scheduling algorithms assume only a single processor is present. Scheduling algorithms decide which of the processes in the ready queue is to be allocated to the CPU is basis on the type of scheduling policy and whether that policy is either preemptive or non-preemptive. For scheduling arrival time and service time are also will play a role.
List of scheduling algorithms are as follows:

1. **First-come, first-served scheduling (FCFS) algorithm**
2. **Shortest Job First Scheduling (SJF) algorithm**
3. **Non-preemptive priority Scheduling algorithm**
4. **Preemptive priority Scheduling algorithm**
5. **Round-Robin Scheduling algorithm**

1) First-come First-served Scheduling (FCFS): - First-come First-served Scheduling follow first in first out method. As each process becomes ready, it joins the ready queue. When the current running process ceases to execute, the oldest process in the Ready queue is selected for running. That is first entered process among the available processes in the ready queue. The average waiting time for FCFS is often quite long. It is non-preemptive.

TURNAROUND TIME = WAITING TIME + SERVICE TIME

Advantages: -

a. It is simple and easy to understand.

b. It can be easily implemented using queue data structure.

c. It does not lead to starvation.

Disadvantages: -

a. It does not consider the priority or burst time of the processes.

b. It suffers from **convoy effect**.

2) Shortest Job First Scheduling (SJFS): - This algorithm associates each process the length of the next CPU burst. Shortest job first scheduling is also called as shortest process next (SPN). The process with the shortest expected processing time is selected for execution, among the available processes in the ready queue. Thus, a short process will jump to the head of the queue over long jobs. If the next CPU bursts of two processes are the same then FCFS scheduling is used to break the tie. SJF scheduling algorithm is probably optimal. It gives the minimum average time for a given set of processes.
It cannot be implemented at the level of short term CPU scheduling. There is no way of knowing the shortest CPU burst. SJF can be preemptive or non-preemptive.
A preemptive SJF algorithm will preempt the currently executing process if the next CPU burst of newly arrived process may be shorter than what is left to the currently executing process.
A Non-preemptive SJF algorithm will allow the currently running process to finish. Preemptive SJF Scheduling is sometimes called Shortest Remaining Time First algorithm.

Advantages: -

a. It gives superior turnaround time performance to shortest process next because a short job is given immediate preference to a running longer job.
b. Throughput is high.

Disadvantages: -

a. Elapsed time (i.e., execution-completed-time) must be recorded, it results an additional overhead on the processor.
b. Starvation may be possible for the longer processes.

Priority Scheduling: - The SJF is a special case of general priority scheduling algorithm. A Priority (an integer) is associated with each process. The CPU is allocated to the process with the highest priority. Generally smallest integer is considered as the highest priority. Equal priority processes are scheduled in First Come First serve order. It can be preemptive or Non-preemptive.

3) Non-preemptive-Priority-Scheduling: - In this type of scheduling the CPU is allocated to the process with the highest priority after completing the present running process.

Advantage: - Good response for the highest priority processes.

Disadvantage: - Starvation may be possible for the lowest priority processes.

4) Preemptive-Priority-Scheduling: - In this type of scheduling the CPU is allocated to the process with the highest priority immediately upon the arrival of the highest priority process. If the equal priority process is in running state, after the completion of the present running process CPU is allocated to this even though one more equal priority process is to arrive.

Advantage: - Very good response for the highest priority process over non-preemptive version of it.

Disadvantage: - Starvation may be possible for the lowest priority processes.

5) Round-Robin Scheduling: - This type of scheduling algorithm is basically designed for time sharing system. It is similar to FCFS with preemption added. Round-Robin Scheduling is also called as time-slicing scheduling and it is a preemptive version based on a clock. That is a clock interrupt is generated at periodic intervals usually 10-100ms. When the interrupt occurs, the currently running process is placed in the ready queue and the next ready job is selected on a First-come, First-serve basis. This process is known as time-slicing, because each process is given a slice of time before being preempted.

One of the following happens:

a. The process may have a CPU burst of less than the time quantum or
b. CPU burst of currently executing process be longer than the time quantum. In this case the a context switch occurs the process is put at the tail of the ready queue. In round-robin scheduling, the principal design issue is the length of the time quantum or time-slice to be used. If the quantum is very short, then short processes will move quickly.

Advantages: -

a. Round-robin is effective in a general-purpose, times-sharing system or transaction-processing system.
b. Fair treatment for all the processes.
c. Overhead on processor is low.
d. Overhead on processor is low.
e. Good response time for short processes.

Disadvantages: -

a. Care must be taken in choosing quantum value.

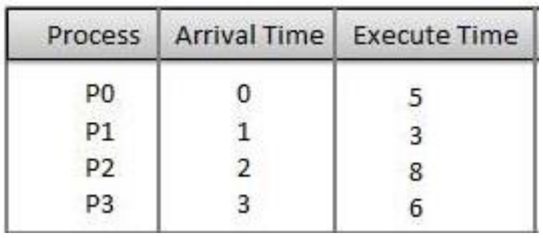
b. Processing overhead is there in handling clock interrupt.
c. Throughput is low if time quantum is too small.

Performance of RR Scheduling

a. If there are n processes in the ready queue and time quantum is q, then each process gets 1/n of the CPU time in chunks of at most q time units at once.
b. No process waits for more than (n-1)*q time units until the next time quantum.
c. The performance of RR depends on time slice. If it is large then it is the same as FCFS. If q is small then overhead is too high.

Examples- FCFS

PROCESS	BURST TIME	WAITING TIME	TURN AROUND TIME
P1	8	0	8
P2	3	8	11
P3	5	11	16
P4	3	16	19



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

SJFS

PROCESS	ARRIVAL TIME	BURST TIME	WAITING TIME	TURN AROUND TIME
P1	1	7	0	7
P2	3	3	7	10
P3	6	2	2	4
P4	7	10	14	24
P5	9	8	4	12

Priority Scheduling

PROCESS	PRIORITY	BURST TIME	ARRIVAL TIME	WAITING TIME	TURN AROUND TIME
P1	2	1	0	0	1
P2	6	7	1	14	21
P3	3	3	2	0	3
P4	5	6	3	7	13
P5	4	5	4	1	6
P6	10	15	5	25	40
P7	9	8	6	16	24

Round Robin Scheduling

PROCESS	BURST TIME	WAITING TIME	TURN AROUND TIME
P1	8	10	18
P2	3	4	7
P3	5	14	19
P4	3	11	14

Multiprocessors Scheduling

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.

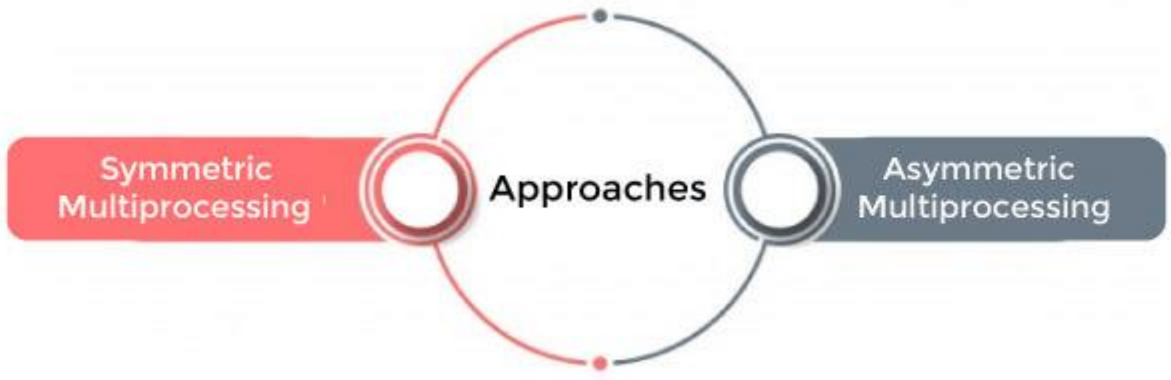
There are cases when the processors are identical, i.e., homogenous, in terms of their functionality in multiple-processor scheduling. We can use any processor available to run any process in the queue.

Multiprocessor systems may be ***heterogeneous*** (different 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.

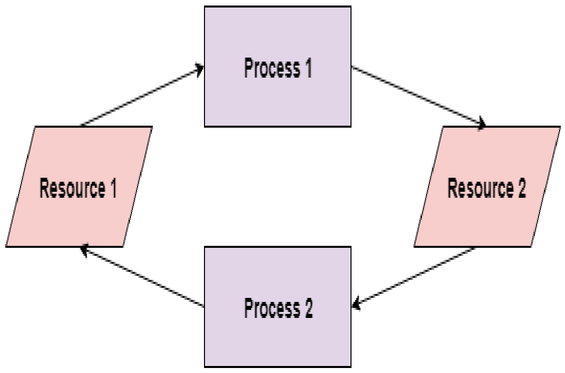


1. Symmetric Multiprocessing: It is used where each processor is ***self-scheduling***. All processes may be in a common ready queue, or each processor may have its private queue for ready processes. The scheduling proceeds further by having the scheduler for each processor examine the ready queue and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

Deadlock

A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.



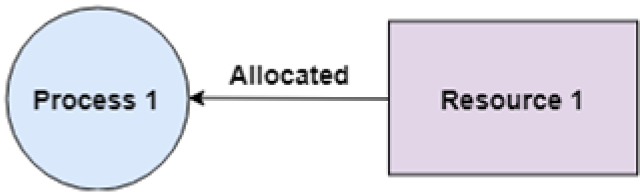
In the above diagram, the process 1 has resource 1 and needs to acquire resource 2. Similarly process 2 has resource 2 and needs to acquire resource 1. Process 1 and process 2 are in deadlock as each of them needs the other's resource to complete their execution but neither of them is willing to relinquish their resources.

Coffman Conditions

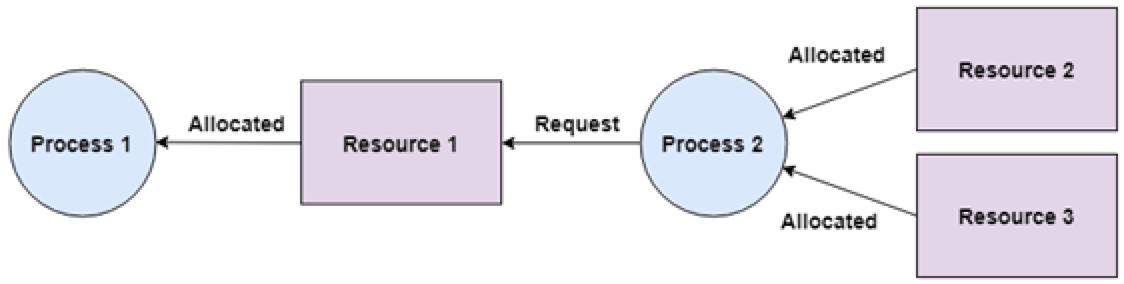
A deadlock occurs if the four Coffman conditions hold true. But these conditions are not mutually exclusive.

The Coffman conditions are given as follows −

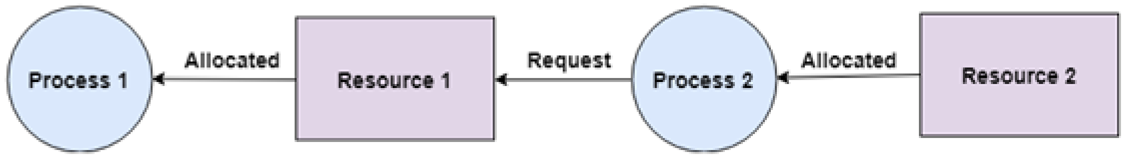
- **Mutual Exclusion: -** There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



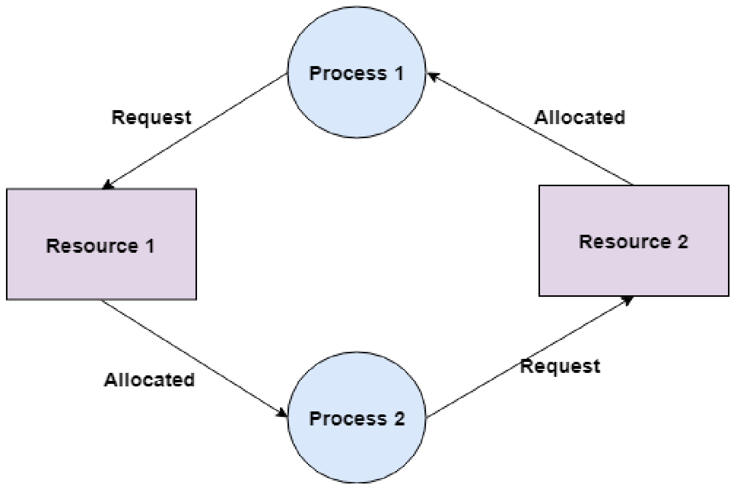
- **Hold and Wait: -** A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



- **No Preemption: -** A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



- **Circular Wait: -** A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



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 preempted from some processes and given to others till the deadlock is resolved.

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.

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.

Deadlock Recovery: - A traditional operating system such as Windows doesn't deal with deadlock recovery as it is a time and space-consuming process. Real-time operating systems use Deadlock recovery.

1. Killing the process: - Killing all the processes involved in the deadlock. Killing process one by one. After killing each process check for deadlock again keep repeating the process till the system recovers from deadlock. Killing all the processes one by one helps a system to break circular wait condition.

2. Resource Preemption: - Resources are preempted from the processes involved in the deadlock, preempted resources are allocated to other processes so that there is a possibility of recovering the system from deadlock. In this case, the system goes into starvation.

Banker's Algorithm

Banker’s Algorithm is a resource allocation and deadlock avoidance algorithm. This algorithm test for safety simulating the allocation for predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue.

In simple terms, it checks if allocation of any resource will lead to deadlock or not, OR is it safe to allocate a resource to a process and if not then resource is not allocated to that process. Determining a safe sequence (even if there is only 1) will assure that system will not go into deadlock.

Banker’s algorithm is generally used to find if a safe sequence exists or not. But here we will determine the total number of safe sequences and print all safe sequences.

The data structures used are:

- Available vector
- Max Matrix
- Allocation Matrix
- Need Matrix

Advantages

Following are the essential characteristics of the Banker's algorithm:

1. It contains various resources that meet the requirements of each process.

2. Each process should provide information to the operating system for upcoming resource requests, the number of resources, and how long the resources will be held.

3. It helps the operating system manage and control process requests for each type of resource in the computer system.

4. The algorithm has a Max resource attribute that represents indicates each process can hold the maximum number of resources in a system.

Disadvantages

1. It requires a fixed number of processes, and no additional processes can be started in the system while executing the process.

2. The algorithm does no longer allows the processes to exchange its maximum needs while processing its tasks.

3. Each process has to know and state their maximum resource requirement in advance for the system.

4. The number of resource requests can be granted in a finite time, but the time limit for allocating the resources is one year.

Process	**Allocation**			**Max**			**Available**			**Need**		
	A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
P1	0	1	0	7	5	3	3	3	2	7	4	3
P2	2	0	0	3	2	2				1	2	2
P3	3	0	2	9	0	2				6	0	0
P4	2	1	1	2	2	2				0	1	1
P5	0	0	2	4	3	3				4	3	1

Step 1: For Process P1:

Need <= Available, 7, 4, 3 <= 3, 3, 2 condition is **false**.

So, we examine another process, P2.

Step 2: For Process P2:

Need <= Available, 1, 2, 2 <= 3, 3, 2 condition **true**

New available = available + Allocation

(3, 3, 2) + (2, 0, 0) => 5, 3, 2

Similarly, we examine another process P3.

Step 3: For Process P3:

P3 Need <= Available, 6, 0, 0 < = 5, 3, 2 condition is **false**.

Similarly, we examine another process, P4.

Step 4: For Process P4:

P4 Need <= Available, 0, 1, 1 <= 5, 3, 2 condition is **true**

New Available resource = Available + Allocation

5, 3, 2 + 2, 1, 1 => 7, 4, 3

Similarly, we examine another process P5.

Step 5: For Process P5:

P5 Need <= Available, 4, 3, 1 <= 7, 4, 3 condition is **true**

New available resource = Available + Allocation

7, 4, 3 + 0, 0, 2 => 7, 4, 5

Now, we again examine each type of resource request for processes P1 and P3.

Step 6: For Process P1:

P1 Need <= Available, 7, 4, 3 <= 7, 4, 5 condition is **true**

New Available Resource = Available + Allocation

7, 4, 5 + 0, 1, 0 => 7, 5, 5

So, we examine another process P2.

Step 7: For Process P3:

P3 Need <= Available, 6, 0, 0 <= 7, 5, 5 condition is true

New Available Resource = Available + Allocation

7, 5, 5 + 3, 0, 2 => 10, 5, 7

Hence, we execute the banker's algorithm to find the safe state and the safe sequence like P2, P4, P5, P1 and P3.