ZEBACAR

MCA I, SEM I

A.Y. 2023-24

Subject:- Operating Systems Concepts

By

Prof. Ashok Deokar

Topic No 2.
Process Management and Synchronization

2.1 Process Control Block (PCB)

What is Process?

A Program does nothing unless its instructions are executed by a CPU. A program in execution is called a **process.**

There may exist more than one process in the system which may require the same resource at the same time. Therefore, the operating system has to manage all the processes and the resources in a convenient and efficient way.

Some resources may need to be executed by one process at one time to maintain the consistency otherwise the system can become inconsistent and deadlock may occur.

2.2 Job and processor scheduling

States of Process

A process is in one of the following states:

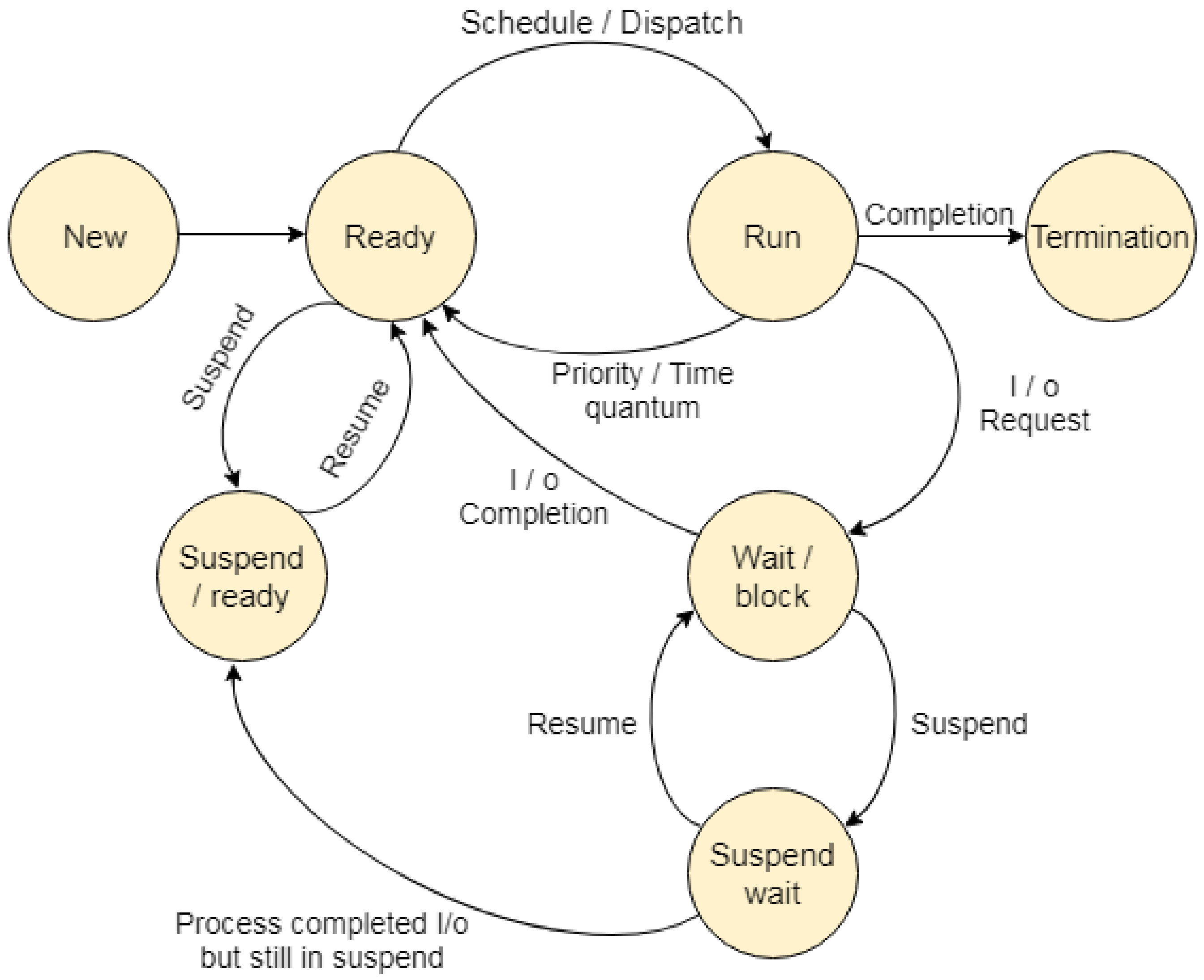
1. **New:** A program which is going to be picked up by the OS into the main memory is called a new process.
2. **Ready:** Whenever a process is created, it directly enters in the ready state, in which, it waits for the CPU to be assigned. The OS picks the new processes from the secondary memory and put all of them in the main memory.

The processes which are ready for the execution and reside in the main memory are called **ready state processes**. There can be many processes present in the ready state.

2.2 Job and processor scheduling

3. **Run:** One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm. if we have only one CPU in our system, the number of running processes for a particular time will always be one. If we have n processors in the system then we can have n processes running simultaneously.
4. **Wait (or Block):** When a process waits for a certain resource to be assigned or for the input from the user then the OS move this process to the block or wait state and assigns the CPU to the other processes.
5. **Complete (or Terminated):** When a process finishes its execution, it comes in the termination state.
6. **Suspended Ready:** When the ready queue becomes full, some processes are moved to a suspended ready state
7. **Suspended Block:** When the waiting queue becomes full.

2.2 Job and processor scheduling



2.1 Process Control Block (PCB)

What is PCB?

A **Process Control Block in OS (PCB)** is a data structure used by the operating system to manage information about a process. It contains information about the process state, memory allocation, CPU usage, I/O devices, and other resources used by the process. The process control block is also known as a **task control block**.

2.1 Process Control Block (PCB)

Structure of the Process Control Block

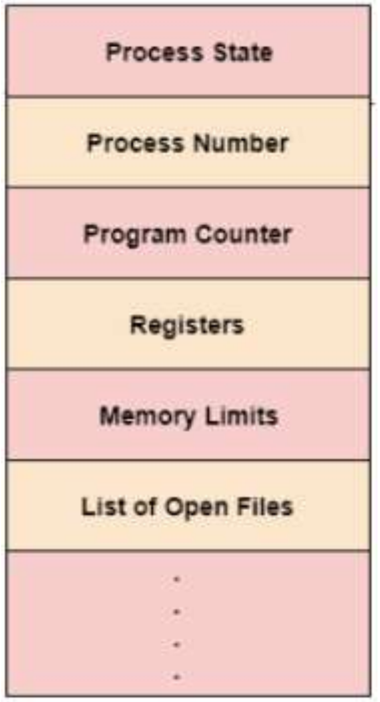
The process control stores many data items that are needed for efficient process management.

Process Number:- This shows the number of the particular process.

Process State:- This specifies the process state i.e. new, ready, running, waiting or terminated.

Program Counter:-The Program Counter keeps an account of the address of the subsequent instruction to be executed by the process.

Registers:-The CPU Registers store values that are critical in context switching.



Process Control Block (PCB)

2.1 Process Control Block (PCB)

List of Open Files:-These are the different files that are associated with the process

CPU Scheduling Information:-The process priority, pointers to scheduling queues etc. is the CPU scheduling information that is contained in the PCB.

I/O Status Information:-This information includes the list of I/O devices used by the process, the list of files etc.

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.

2.2 Job and processor scheduling

Job Scheduling:- Job scheduling is the process where different tasks get executed at pre-determined time or when the right event happens. A job scheduler is a system that can be integrated with other software systems for the purpose of executing or notifying other software components when a pre-determined, scheduled time arrives.

There are three types of Job Scheduler.

1. **Long term scheduler**
2. **Short term scheduler**
3. **Medium term scheduler**

2.2 Job and processor scheduling

1. **Long term scheduler/ job Scheduler**

It chooses the processes from the pool (secondary memory) and keeps them in the ready queue maintained in the primary memory.

Long Term scheduler mainly controls the degree of Multiprogramming. The purpose of long term scheduler is to choose a perfect mix of IO bound and CPU bound processes among the jobs present in the pool.

If the job scheduler chooses more IO bound processes then all of the jobs may reside in the blocked state all the time and the CPU will remain idle most of the time. This will reduce the degree of Multiprogramming. Therefore, the Job of long term scheduler is very critical and may affect the system for a very long time.

2.2 Job and processor scheduling

2. Short term scheduler / CPU scheduler

- It selects one of the Jobs from the ready queue and dispatch to the CPU for the execution.
- A scheduling algorithm is used to select which job is going to be dispatched for the execution. The Job of the short term scheduler can be very critical in the sense that if it selects job whose CPU burst time is very high then all the jobs after that, will have to wait in the ready queue for a very long time.
- This problem is called starvation which may arise if the short term scheduler makes some mistakes while selecting the job.

2.2 Job and processor scheduling

3. Medium term scheduler

- Medium term scheduler takes care of the swapped out processes. If the running state processes needs some IO time for the completion then there is a need to change its state from running to waiting.
- Medium term scheduler is used for this purpose. It removes the process from the running state to make room for the other processes. Such processes are the swapped out processes and this procedure is called swapping. The medium term scheduler is responsible for suspending and resuming the processes.
- It reduces the degree of multiprogramming. The swapping is necessary to have a perfect mix of processes in the ready queue.

2.2 Job and processor scheduling

Processor scheduling:- Processor scheduling is the allocation of a computer's processor power to specific tasks. The practice uses the term "scheduling" because it assigns a specific percentage of time the processor is running to individual tasks. Processor scheduling is used to prevent specific tasks from monopolizing all of a computer's processor resources.

2.2 Job and processor scheduling

CPU Scheduling in Operating Systems

Scheduling of processes/work is done to finish the work on time. **CPU Scheduling** is a process that allows one process to use the CPU while another process is delayed (in standby) due to unavailability of any resources such as I / O etc, thus making full use of the CPU. The purpose of CPU Scheduling is to make the system more efficient, faster, and fairer.

Whenever the CPU becomes idle, the operating system must select one of the processes in the line ready for launch. The selection process is done by a temporary (CPU) scheduler. The Scheduler selects between memory processes ready to launch and assigns the CPU to one of them.

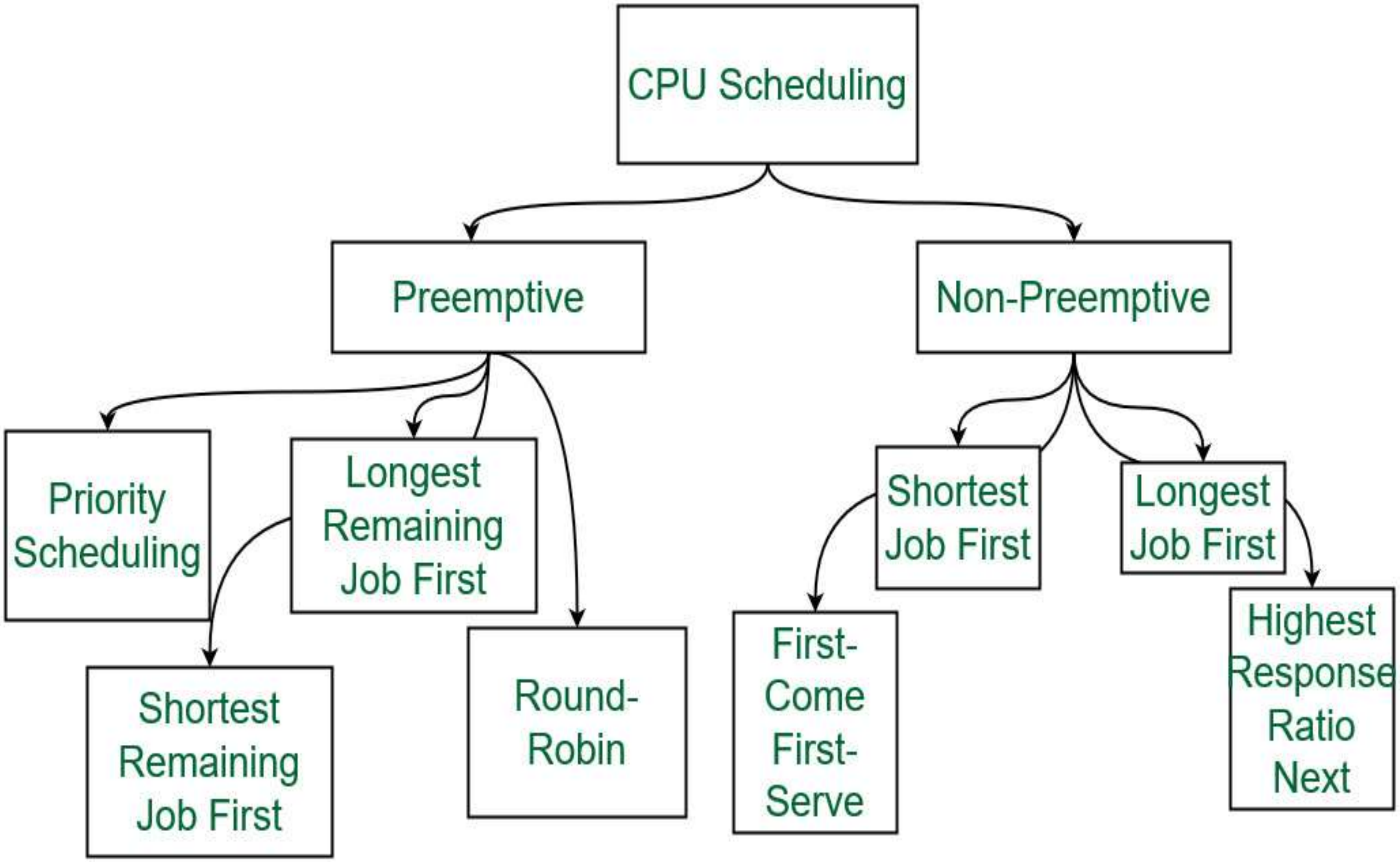
2.2 Job and processor scheduling

Different types of CPU Scheduling:-

Preemptive Scheduling: Preemptive scheduling is used when a process switches from running state to ready state or from the waiting state to the ready state.

Non-Preemptive Scheduling: Non-Preemptive scheduling is used when a process terminates, or when a process switches from running state to waiting state.

2.2 Job and processor scheduling



2.2 Job and processor scheduling

Context Switching:

- The task of switching a CPU from one process to another process is called context switching. Context-switch times are highly dependent on hardware support (Number of CPU registers).
- Whenever an interrupt occurs (hardware or software interrupt), the state of the currently running process is saved into the PCB and the state of another process is restored from the PCB to the CPU.
- Context switch time is an overhead, as the system does not do useful work while switching.

When Does Context Switching Happen?

1. When a high-priority process comes to a ready state (i.e. with higher priority than the running process)
2. Preemptive CPU scheduling is used.

2.2 Job and processor scheduling

CPU Switch from Process to Process

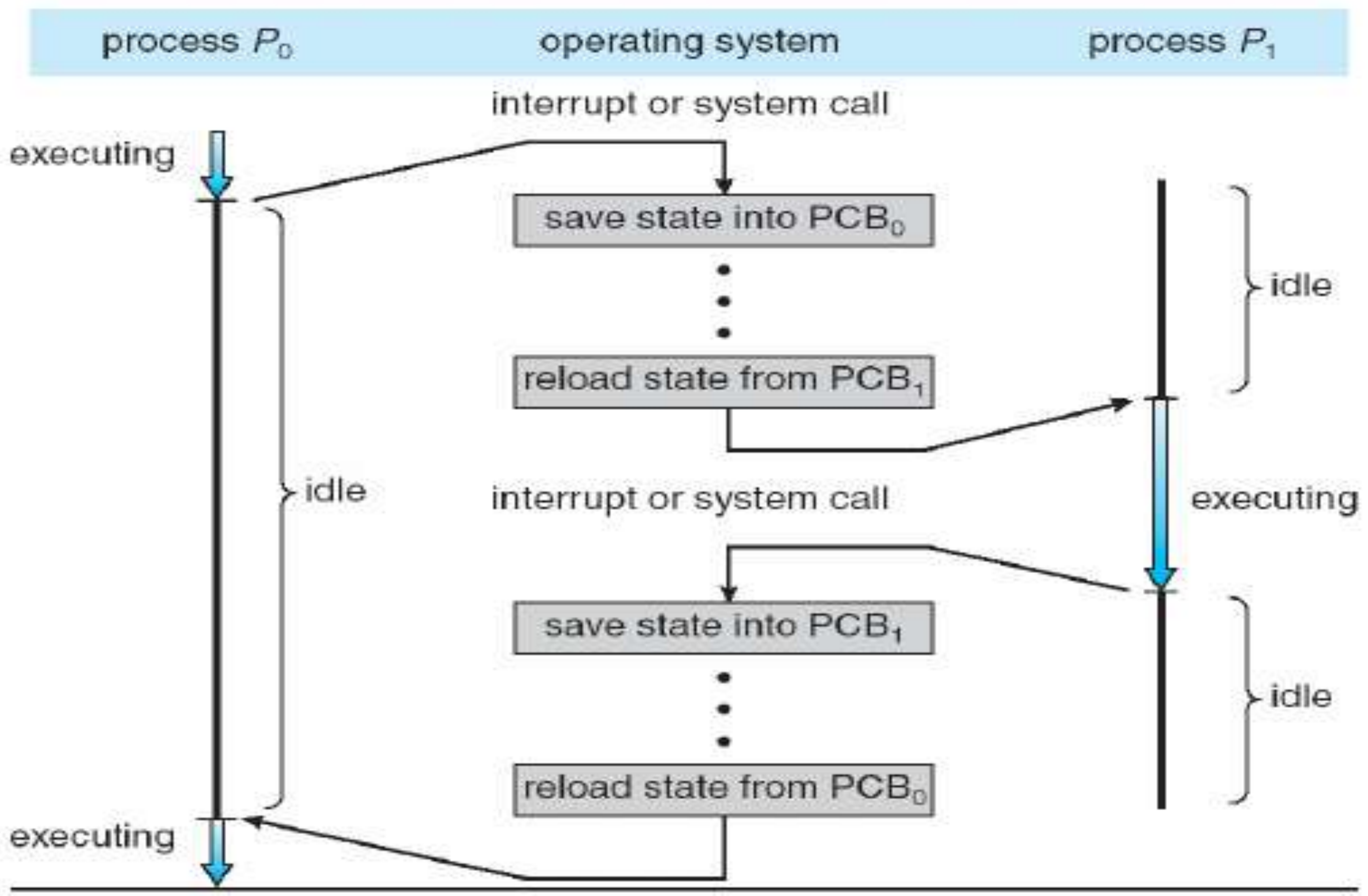


Figure: Diagram showing CPU switch from process to process.

2.2 Job and processor scheduling

What are the different terminologies to take care of in any CPU Scheduling algorithm?

Arrival Time: Time at which the process arrives in the ready queue.

Completion Time: Time at which process completes its execution.

Burst Time: Time required by a process for CPU execution.

Turn Around Time: Time Difference between completion time and arrival time.

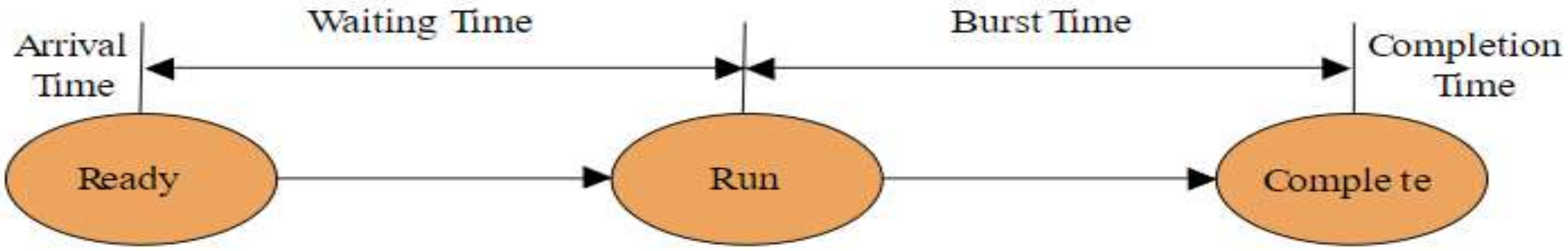
Turn Around Time = Completion Time – Arrival Time

Waiting Time(W.T): Time Difference between turn around time and burst time.

Waiting Time = Turn Around Time – Burst Time

2.2 Job and processor scheduling

Various Times related to the Process



$$CT - AT = WT + BT$$

$$TAT = CT - AT$$

$$\text{Waiting Time} = TAT - BT$$

TAT	⟶	Turn around time
BT	⟶	Burst time
AT	⟶	Arrival time

2.2 Job and processor scheduling

Objectives of Process Scheduling Algorithm:

1. Utilization of CPU at maximum level. **Keep CPU as busy as possible**.
2. **Allocation of CPU should be fair**.
3. **Throughput should be Maximum**. i.e. Number of processes that complete their execution per time unit should be maximized.
4. **Minimum turnaround time**, i.e. time taken by a process to finish execution should be the least.
5. There should be a **minimum waiting time** and the process should not starve in the ready queue.
6. **Minimum response time.** It means that the time when a process produces the first response should be as less as possible.

2.2 Job and processor scheduling

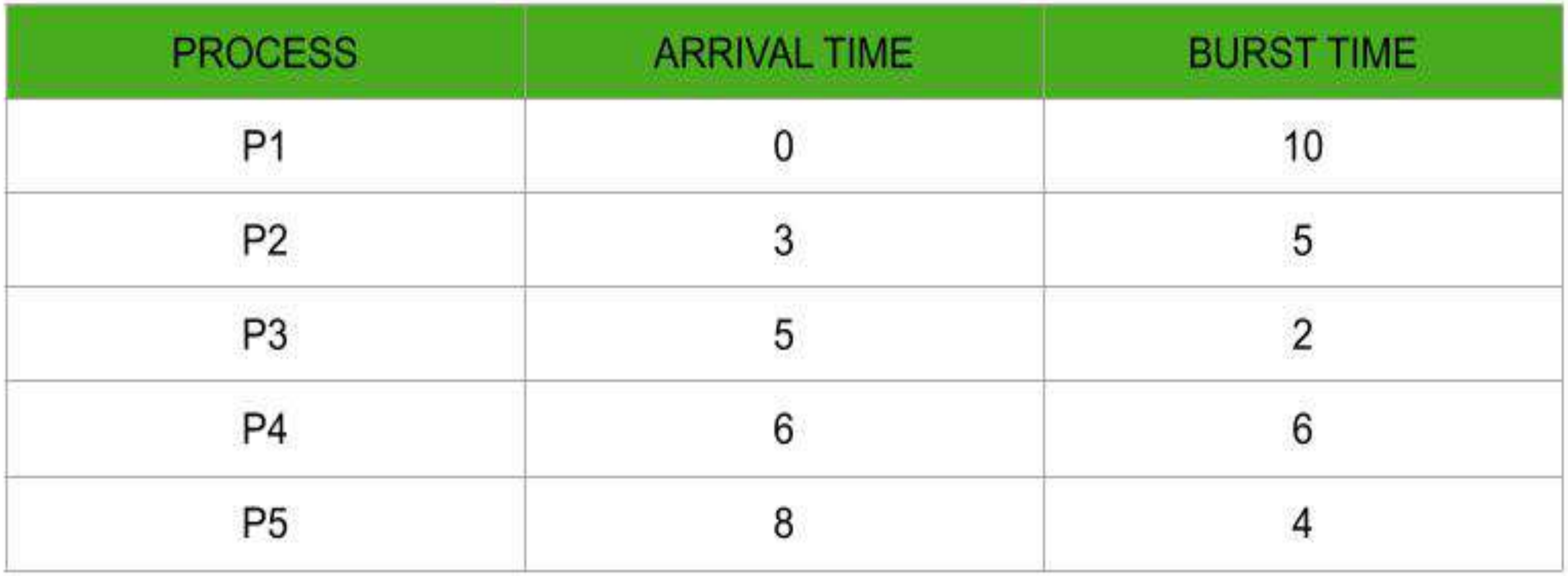
CPU Scheduling Algorithms:-

1. First Come First Serve:

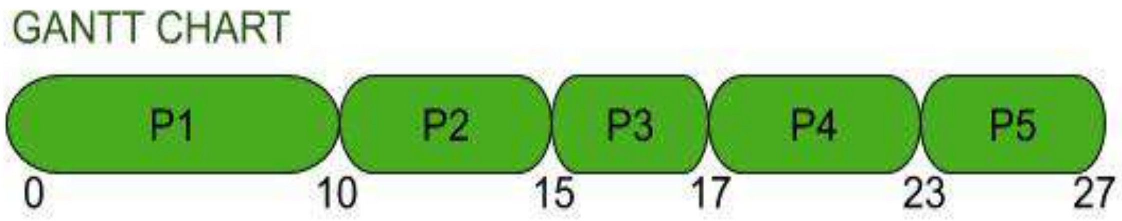
First Come First Serve CPU Scheduling Algorithm shortly known as FCFS is the first algorithm of CPU Process Scheduling Algorithm. In this whichever process enters the ready queue first is executed first. This shows that First Come First Serve Algorithm follows First In First Out (FIFO) principle.

2.2 Job and processor scheduling

1. First Come First Serve:



PROCESS	ARRIVAL TIME	BURST TIME
P1	0	10
P2	3	5
P3	5	2
P4	6	6
P5	8	4

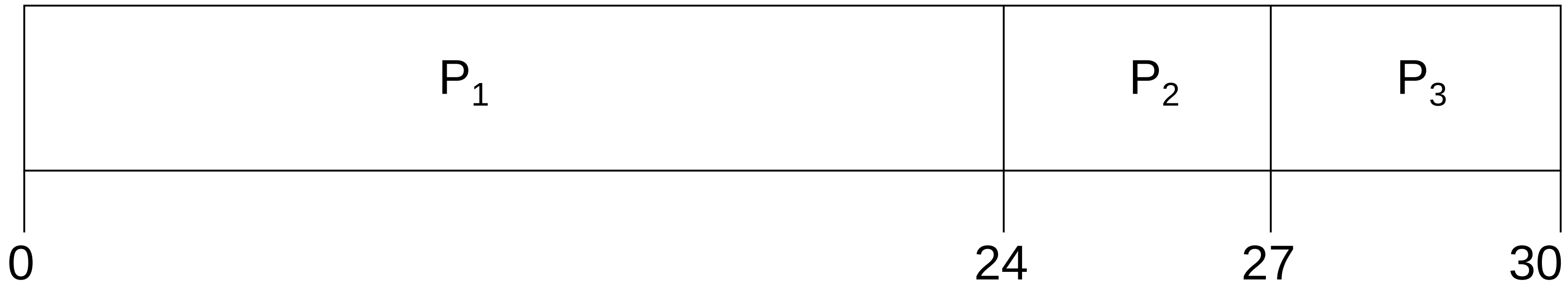


First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

Suppose that the processes arrive in the order: P_1 , P_2 , P_3

The Gantt Chart for the schedule is:



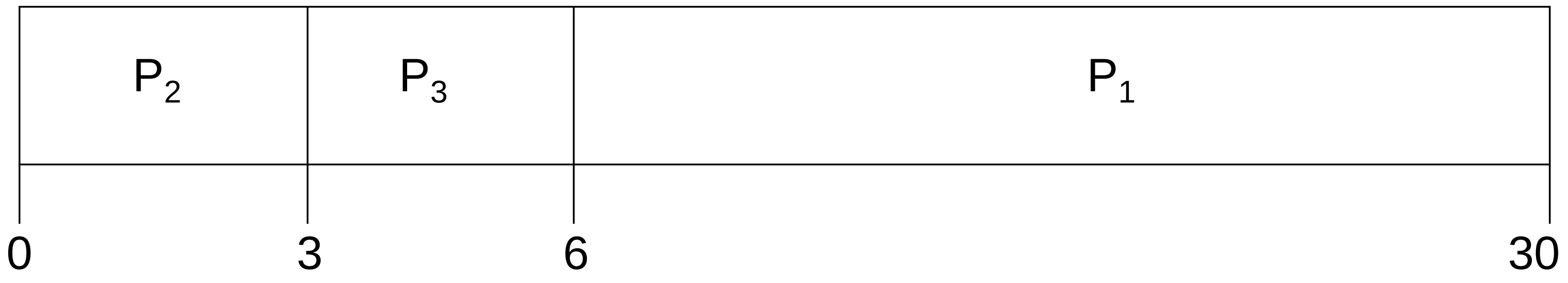
Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

P_2 , P_3 , P_1

The Gantt chart for the schedule is:



Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$
Average waiting time: (6 + 0 + 3)/3 = 3
Much better than previous case

2.2 Job and processor scheduling

Advantages of FCFS :-

- It is an easy algorithm to implement since it does not include any complex way.

- Every task should be executed simultaneously as it follows FIFO queue.

- FCFS does not give priority to any random important tasks first so it’s a fair scheduling.

2.2 Job and processor scheduling

Disadvantages Of FCFS Scheduling:-

- FCFS results in convoy effect which means if a process with higher burst time comes first in the ready queue then the processes with lower burst time may get blocked and that processes with lower burst time may not be able to get the CPU if the higher burst time task takes time forever.

- If a process with long burst time comes in the line first then the other short burst time process have to wait for a long time, so it is not much good as time-sharing systems.

- Since it is non-preemptive, it does not release the CPU before it completes its task execution completely.

2.2 Job and processor scheduling

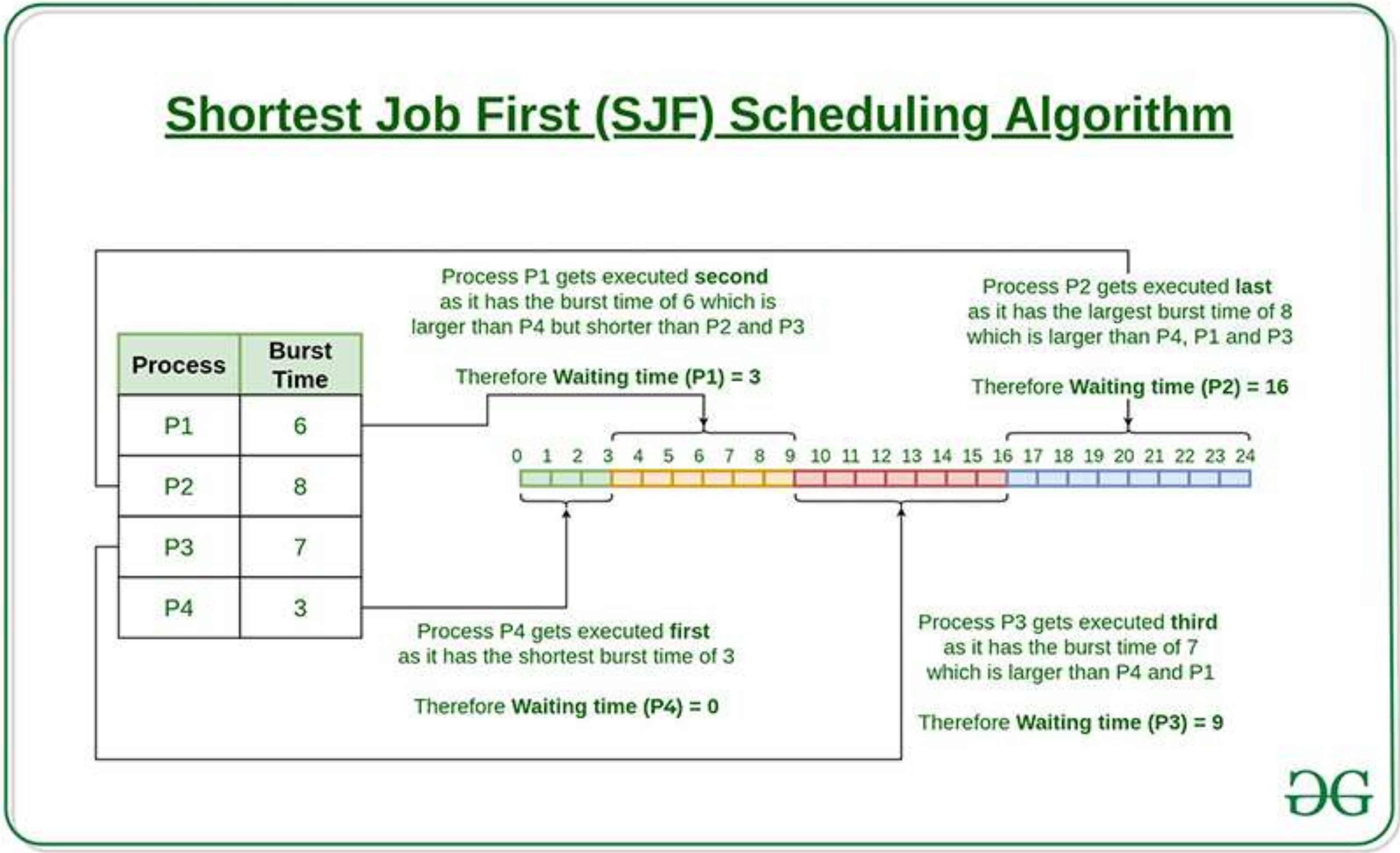
2. Shortest Job First(SJF):

Till now, we were scheduling the processes according to their arrival time (in FCFS scheduling). However, SJF scheduling algorithm, schedules the processes according to their burst time.

In SJF scheduling, the process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next.

However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.

2.2 Job and processor scheduling



Shortest Job First (SJF) Scheduling Algorithm

Process	Burst Time
P1	6
P2	8
P3	7
P4	3

Process P1 gets executed **second** as it has the burst time of 6 which is larger than P4 but shorter than P2 and P3

Therefore **Waiting time (P1) = 3**

Process P2 gets executed **last** as it has the largest burst time of 8 which is larger than P4, P1 and P3

Therefore **Waiting time (P2) = 16**

Process P4 gets executed **first** as it has the shortest burst time of 3

Therefore **Waiting time (P4) = 0**

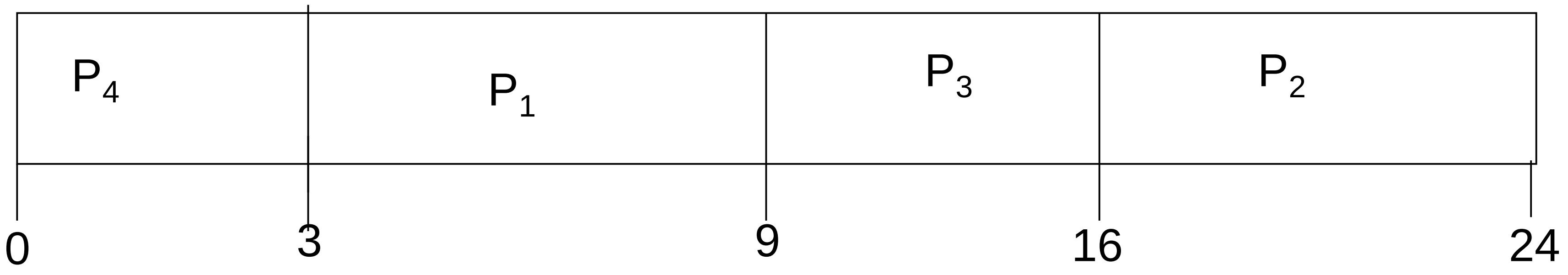
Process P3 gets executed **third** as it has the burst time of 7 which is larger than P4 and P1

Therefore **Waiting time (P3) = 9**

Shortest-Job-First (SJF) Scheduling

Process Arrival	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

SJF scheduling chart



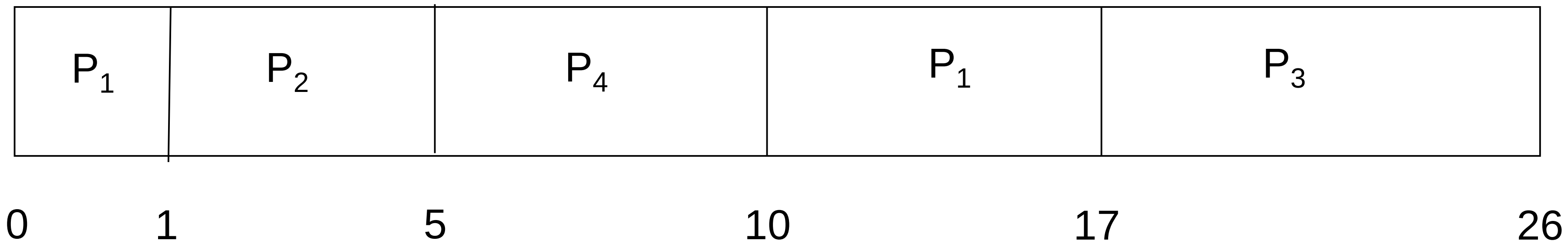
Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

SJF Preemptive

Now we add the concepts of varying arrival times and preemption to the analysis

<u>Process</u>	<u>*Arrival* Time</u>	<u>Burst Time</u>
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Preemptive SJF Gantt Chart



Average waiting time = [(10-1)+(1-1)+(17-2)+5-3)]/4 = 26/4 = 6.5 msec

2.2 Job and processor scheduling

2. Shortest Job First(SJF):

Advantages:

- Shortest jobs are favored.
- It is probably optimal, in that it gives the minimum average waiting time for a given set of processes.

Disadvantages:

- SJF may cause starvation if shorter processes keep coming. This problem is solved by aging(Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time).
- It cannot be implemented at the level of short-term CPU scheduling.

2.2 Job and processor scheduling

3. Priority Scheduling:

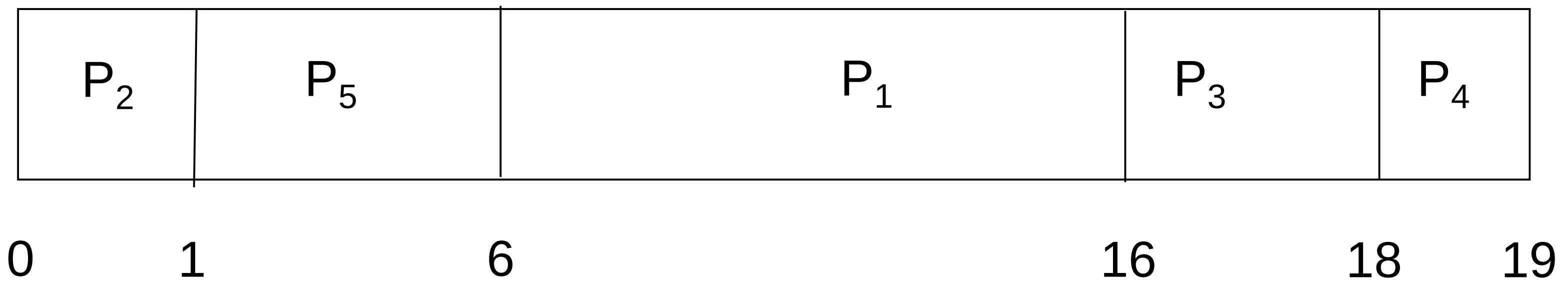
Priority Scheduling is a method of scheduling processes that is based on priority. In this algorithm, the scheduler selects the tasks to work as per the priority.

The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis. Priority depends upon memory requirements, time requirements, etc.

Priority Scheduling

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

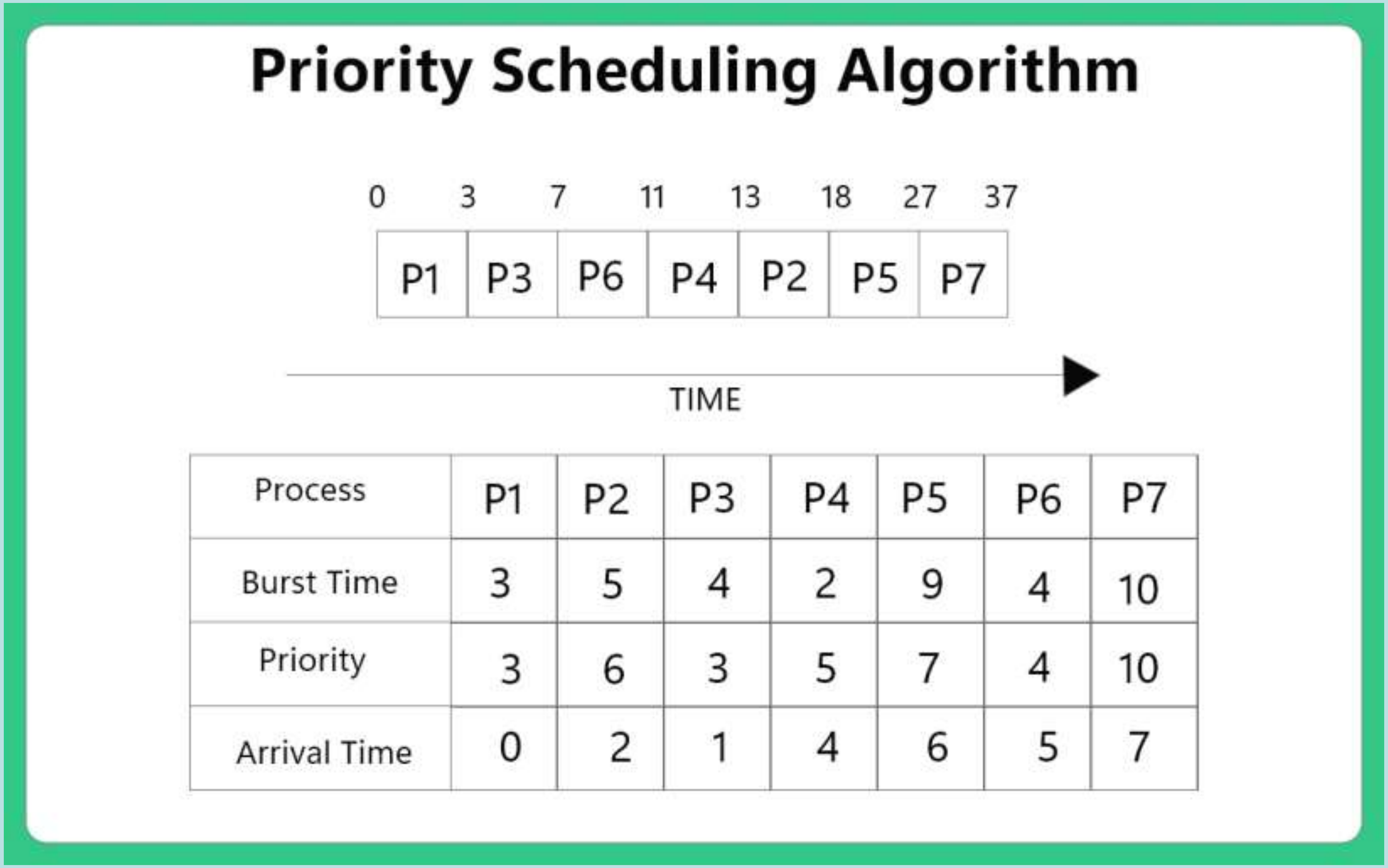
Priority scheduling Gantt Chart



Average waiting time = 8.2 msec

2.2 Job and processor scheduling

3. Priority Scheduling:



Process	P1	P2	P3	P4	P5	P6	P7
Burst Time	3	5	4	2	9	4	10
Priority	3	6	3	5	7	4	10
Arrival Time	0	2	1	4	6	5	7

2.2 Job and processor scheduling

3. Priority Scheduling:

Advantages:

- This provides a good mechanism where the relative importance of each process may be precisely defined.
- PB scheduling allows for the assignment of different priorities to processes based on their importance, urgency, or other criteria.

Disadvantages:

- If high-priority processes use up a lot of CPU time, lower-priority processes may starve and be postponed indefinitely. The situation where a process never gets scheduled to run is called starvation.
- Another problem is deciding which process gets which priority level assigned to it.

2.2 Job and processor scheduling

4. Round robin:

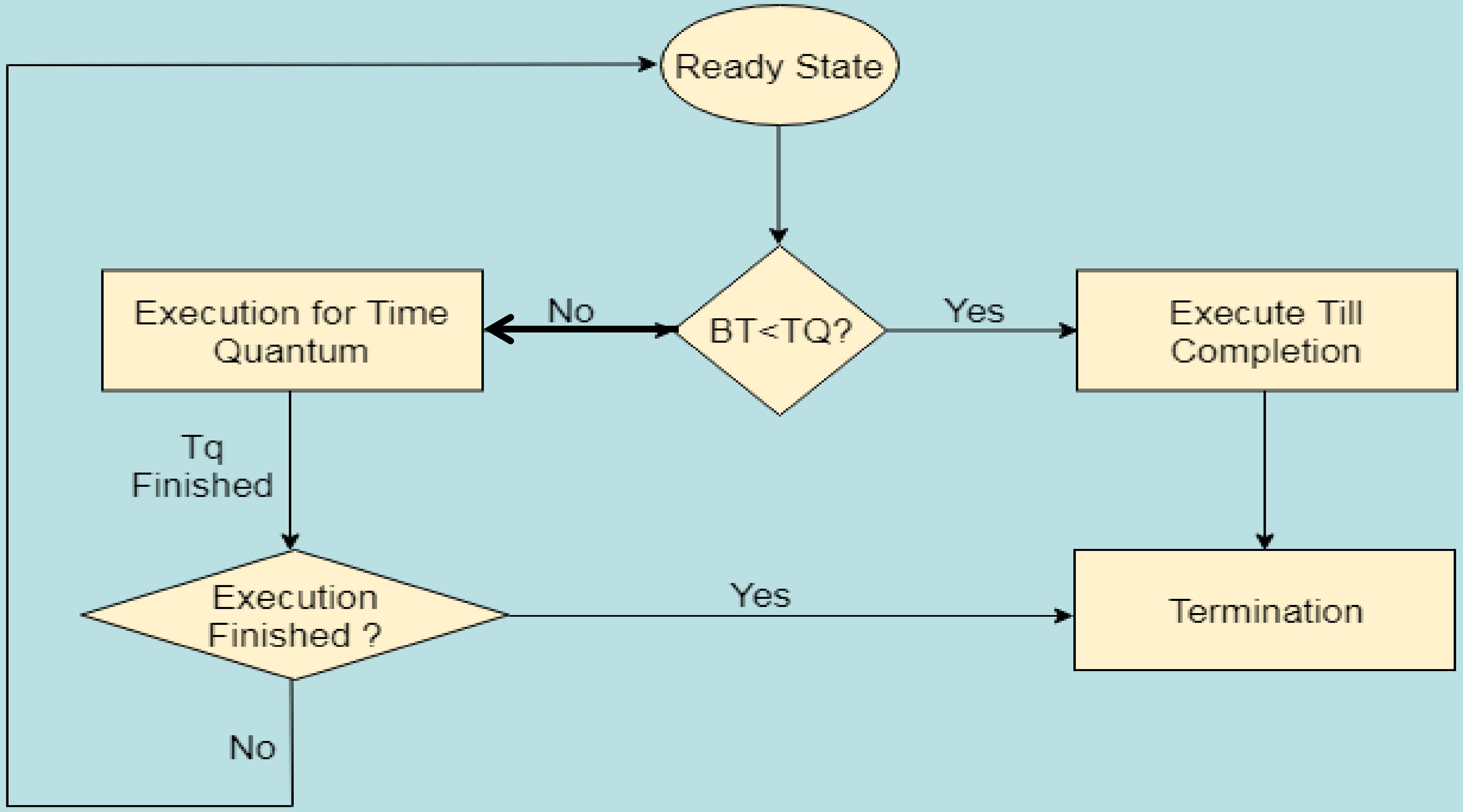
Round Robin CPU Scheduling is the most important CPU Scheduling Algorithm which is ever used in the history of CPU Scheduling Algorithms. Round Robin CPU Scheduling uses **Time Quantum (TQ).** The Time Quantum is something which is removed from the Burst Time and lets the chunk of process to be completed.

Time Sharing is the main emphasis of the algorithm. Each step of this algorithm is carried out cyclically. The system defines a specific time slice, known as a time quantum.

There is a lot of popularity for this Round Robin CPU Scheduling is because Round Robin works only in Pre-emptive state. This makes it very reliable.

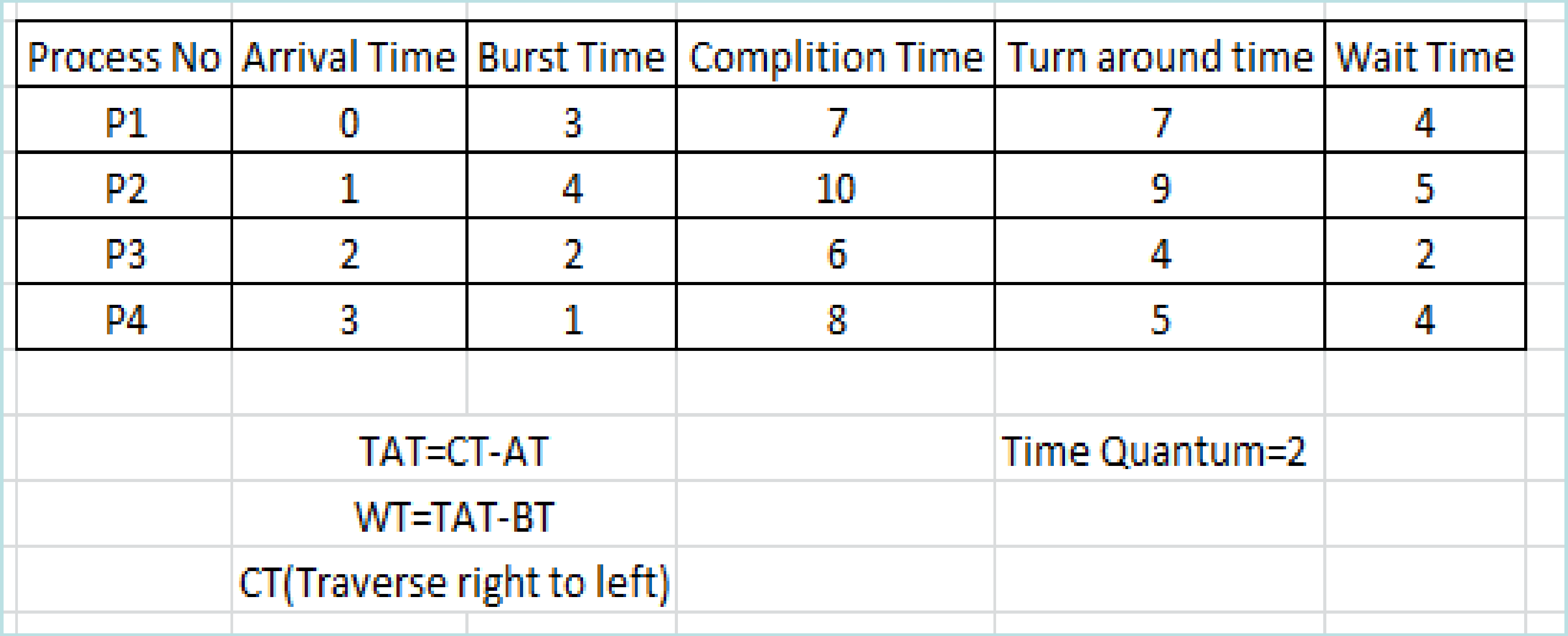
2.2 Job and processor scheduling

4. Round robin:



2.2 Job and processor scheduling

4. Round robin:



Process No	Arrival Time	Burst Time	Complition Time	Turn around time	Wait Time
P1	0	3	7	7	4
P2	1	4	10	9	5
P3	2	2	6	4	2
P4	3	1	8	5	4

TAT=CT-AT

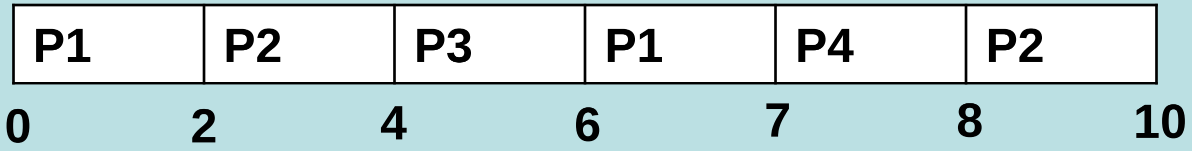
Time Quantum=2

WT=TAT-BT

CT(Traverse right to left)

Ready queue P1 P2 P3 P1 P4 P2

Process execution sequence

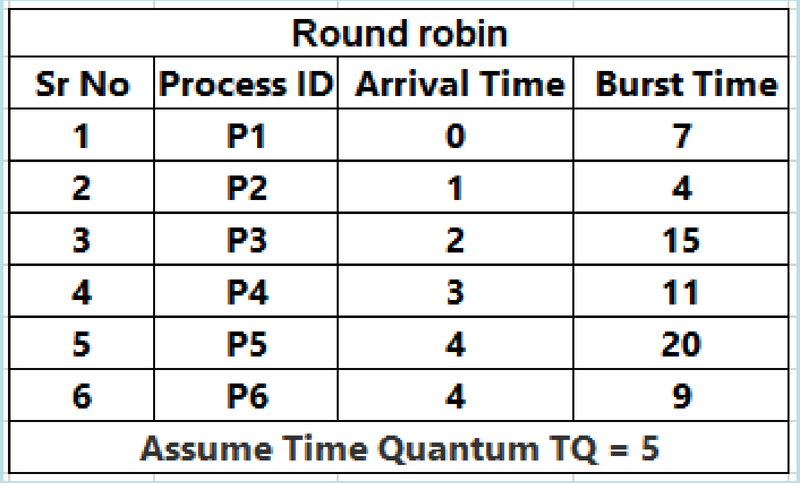


2.2 Job and processor scheduling

4. Round robin:

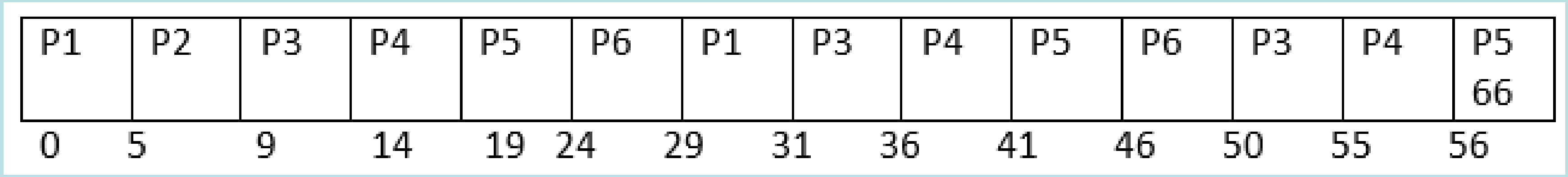
Example:-

Ready Queue:- P1, P2, P3, P4, P5, P6, P1, P3, P4, P5, P6, P3, P4, P5



Round robin			
Sr No	Process ID	Arrival Time	Burst Time
1	P1	0	7
2	P2	1	4
3	P3	2	15
4	P4	3	11
5	P5	4	20
6	P6	4	9
Assume Time Quantum TQ = 5			

Gantt chart:

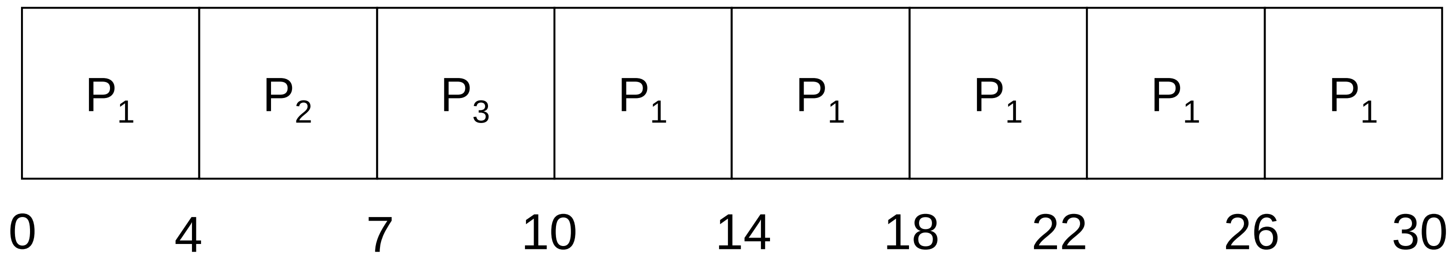


P1	P2	P3	P4	P5	P6	P1	P3	P4	P5	P6	P3	P4	P5 66
0	5	9	14	19	24	29	31	36	41	46	50	55	56

RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

The Gantt chart is:



2.2 Job and processor scheduling

4. Round robin:

Advantages

1. A fair amount of CPU is allocated to each job.
2. Because it doesn't depend on the burst time, it can truly be implemented in the system.
3. It is not affected by the convoy effect or the starvation problem as occurred in First Come First Serve CPU Scheduling Algorithm.

2.2 Job and processor scheduling

4. Round robin:

Disadvantages

1. Low Operating System slicing times will result in decreased CPU output.
2. Round Robin CPU Scheduling approach takes longer to swap contexts.
3. Time quantum has a significant impact on its performance.
4. The procedures cannot have priorities established.

2.4 Process hierarchies

Process hierarchies:-

In an operating system, the process hierarchy refers to the organization and structure of processes within a system. It establishes the relationship between parent and child processes, creating a hierarchical structure.

Parent-Child Relationship: In a process hierarchy, each process except for the root process has a parent process from which it is created. The parent process is responsible for creating, managing, and controlling its child processes. This creates a hierarchical relationship where processes form a tree-like structure.

2.4 Process hierarchies

- **Root Process:** The root process is the top-level process in the hierarchy and has no parent process. It is usually the first process that is created when the operating system starts. All other processes are descendants of the root process.
- **Child Processes:** Each process, except for the root process, can create one or more child processes. When a process creates a child process, the child process inherits certain attributes and resources from its parent, such as the memory space, file descriptors, and environment variables.
- **Process Group:** Processes within the same process hierarchy can be organized into process groups. A process group is a collection of related processes that can be managed and controlled collectively. Process groups enable operations such as signaling and process control across multiple processes.

2.4 Process hierarchies

- **Process Tree:** The process hierarchy forms a tree-like structure, often referred to as the process tree. The root process is at the top of the tree, and child processes branch out from their parent processes. The process tree represents the relationships and dependencies between processes within the system.
- **Process ID (PID):** Each process in the system is assigned a unique process identifier (PID) that distinguishes it from other processes. PIDs are used for process management and identification purposes, allowing the operating system to track and manipulate processes.
- **Process Termination:** When a process terminates, either voluntarily or due to an error or completion, its resources are released, and it is removed from the process hierarchy

2.5 Problems of concurrent processes

Problems in Concurrency :

- **Sharing global resources –** If two processes both make use of a global variable and both perform read and write on that variable, then the order in which various read and write are executed is critical.
- **Optimal allocation of resources –** It is difficult for the operating system to manage the allocation of resources optimally.
- **Locating programming errors –** It is very difficult to locate a programming error because reports are usually not reproducible.
- **Locking the channel –** It may be inefficient for the operating system to simply lock the channel and prevents its use by other processes.

2.5 Problems of concurrent processes

Issues of Concurrency :

- **Non-atomic –** Operations that are non-atomic but interruptible by multiple processes can cause problems.
- **Race conditions –** A race condition occurs of the outcome depends on which of several processes gets to a point first.
- **Blocking –** Processes can block waiting for resources. A process could be blocked for long period of time waiting for input from a terminal. If the process is required to periodically update some data, this would be very undesirable.
- **Starvation –** It occurs when a process does not obtain service to progress.
- **Deadlock –** It occurs when two processes are blocked and hence neither can proceed to execute.

2.8 Process Synchronization

- **Process Synchronization** is the coordination of execution of multiple processes in a multi-process system to ensure that they access shared resources in a controlled and predictable manner. It aims to resolve the problem of race conditions and other synchronization issues in a concurrent system.
- The main objective of process synchronization is to ensure that multiple processes access shared resources without interfering with each other and to prevent the possibility of inconsistent data due to concurrent access. To achieve this, various synchronization techniques such as semaphores, monitors, and critical sections are used.

2.8 Process Synchronization

In a multi-process system, synchronization is necessary to ensure data consistency and integrity, and to avoid the risk of deadlocks and other synchronization problems

On the basis of synchronization, processes are categorized as one of the following two types:

1. **Independent Process**: The execution of one process does not affect the execution of other processes.
2. **Cooperative Process**: A process that can affect or be affected by other processes executing in the system.

Process synchronization problem arises in the case of Cooperative processes also because resources are shared in Cooperative processes.

2.8 Process Synchronization

Race Condition:- When more than one process is executing the same code or accessing the same memory or any shared variable in that condition there is a possibility that the output or the value of the shared variable is wrong so for that all the processes doing the race to say that my output is correct this condition known as a race condition.

Several processes access and process the manipulations over the same data concurrently, then the outcome depends on the particular order in which the access takes place.

2.8 Process Synchronization

Race Condition Example:-

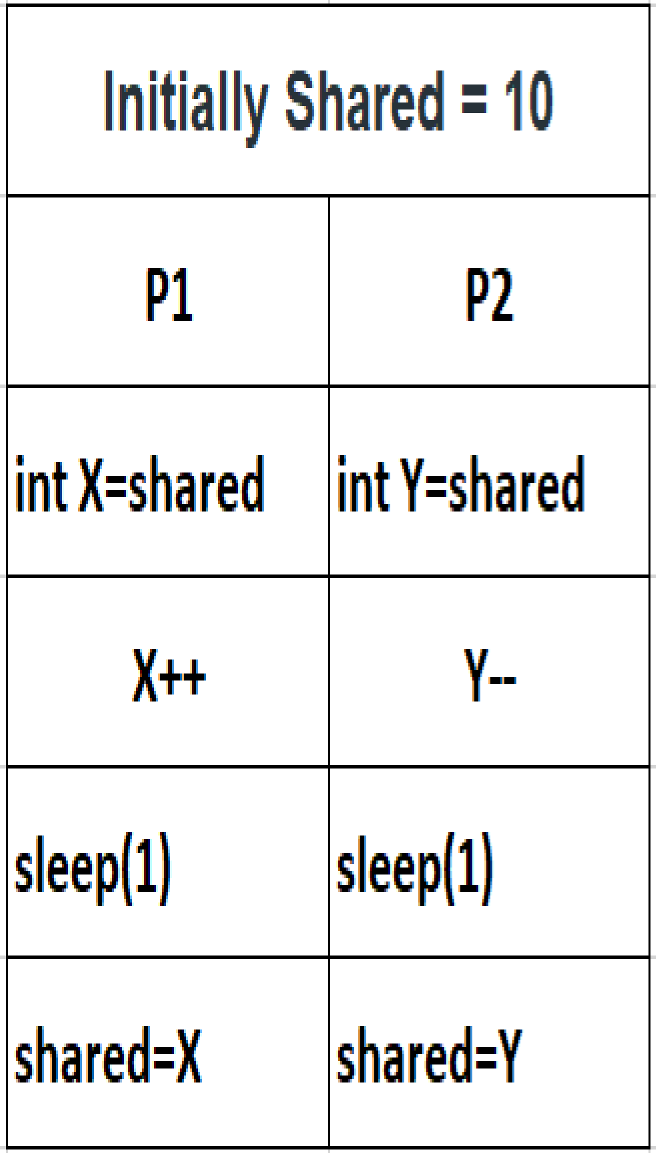
There are two processes P1 and P2 which share common variable (shared=10) , both processes are present in ready – queue and waiting for its turn to be execute. Suppose, Process P1 first come under execution, and CPU store common variable between them (shared=10) in local variable (X=10) and increment it by 1(X=11) , after then when CPU read line sleep(1) ,it switches from current process P1 to process P2 present in ready-queue. The process P1 goes in waiting state for 1 second .

2.8 Process Synchronization

Now CPU execute the Process P2 line by line and store common variable (Shared=10) in its local variable (Y=10) and decrement Y by 1(Y=9),after then when CPU read sleep(1) , the current process P2 goes in waiting state and CPU remains idle for sometime as there is no process in ready-queue, after completion of 1 second of process P1 when it comes in ready-queue, CPU takes the process P1 under execution and execute the remaining line of code (store the local variable (X=11) in common variable (shared=11)),CPU remain idle for sometime waiting for any process in ready-queue ,after completion of 1 second of Process P2, when process P2 comes in ready-queue, CPU start executing the further remaining line of Process P2(store the local variable (Y=9) in common variable (shared=9)).

2.8 Process Synchronization

Note: We are assuming the final value of common variable(shared) after execution of Process P1 and Process P2 is 10 (as Process P1 increment variable (shared=10) by 1 and Process P2 decrement variable (shared=11) by 1 and finally it becomes shared=10). But we are getting undesired value due to lack of proper synchronization.



Initially Shared = 10	
P1	P2
int X=shared	int Y=shared
X++	Y--
sleep(1)	sleep(1)
shared=X	shared=Y

2.8 Process Synchronization

Actual meaning of race-condition

- If the order of execution of process(first P1 -> then P2) then we will get the value of common variable (shared) = 9.

- If the order of execution of process(first P2 -> then P1) then we will get the final value of common variable (shared) =11.

- Basically, Here the (value1 = 9) and (value2=11) are racing , If we execute these two process in our computer system then sometime we will get 9 and sometime we will get 11 as final value of common variable(shared). This phenomenon is called Race-Condition.

2.8 Process Synchronization

Advantages of Process Synchronization

- Ensures data consistency and integrity
- Avoids race conditions
- Prevents inconsistent data due to concurrent access
- Supports efficient and effective use of shared resources

Disadvantages of Process Synchronization

- Adds overhead to the system
- Can lead to performance degradation
- Increases the complexity of the system
- Can cause deadlocks if not implemented properly.

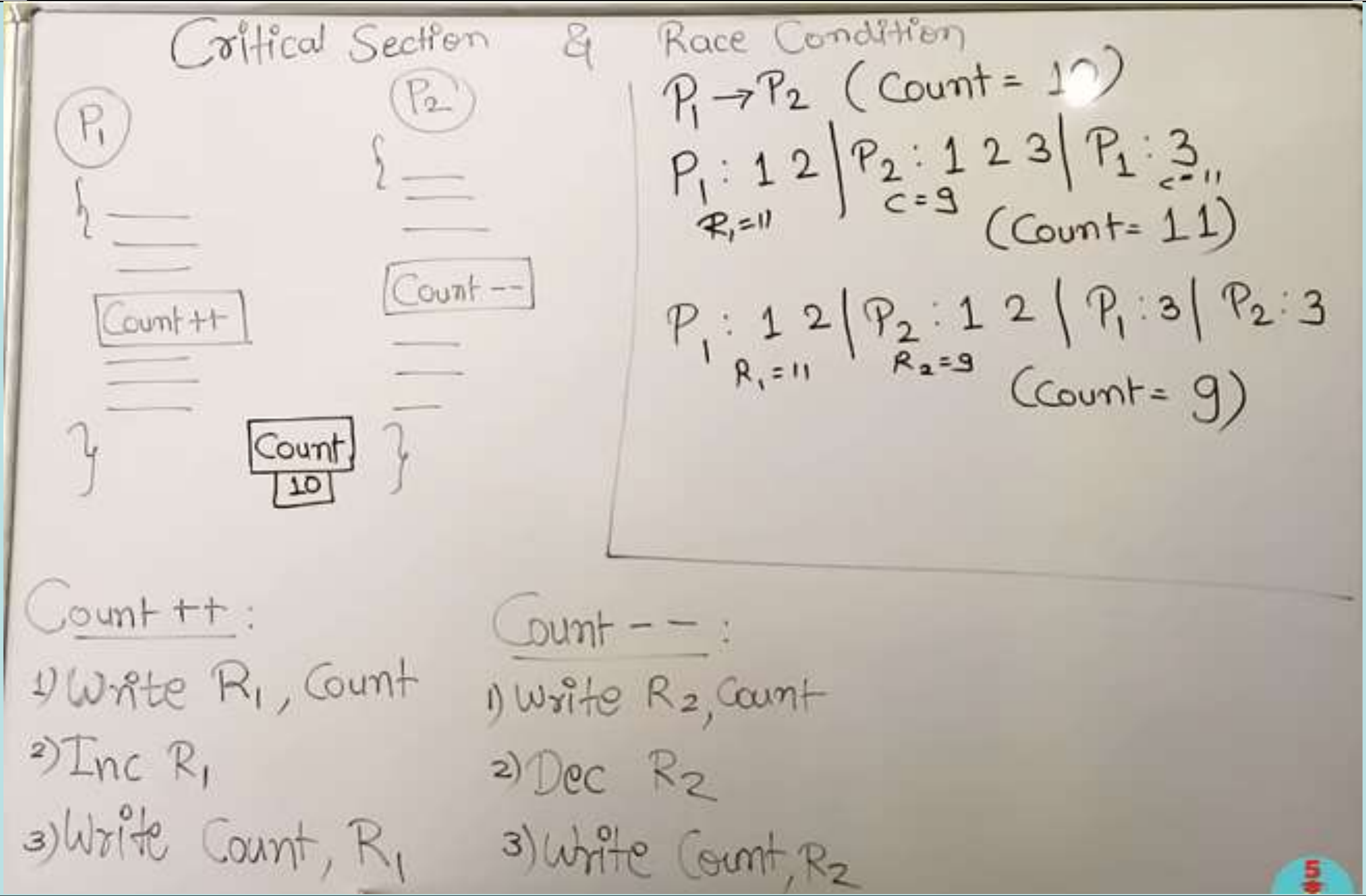
2.6 Critical sections

Critical Section refers to the segment of code or the program that tries to access or modify the value of the variables in a shared re source.

A critical section is a piece of the program that can be accessed by a single process at a given point in time.

Simultaneous access to shared resources can lead to unsound behavior, therefore parts of the program where the shared resource is accessed need to be protected in ways that avoid simultaneous access. This protected section is the critical section or critical region.

2.6 Critical sections



Critical Section & Race Condition

P1 { Count++ }

P2 { Count-- }

Count 10

$P_1 \rightarrow P_2$ (Count = 10)

$P_1 : 1\ 2 \mid P_2 : 1\ 2\ 3 \mid P_1 : 3$

$R_1 = 11$ $C = 9$ $C = 11$

(Count = 11)

$P_1 : 1\ 2 \mid P_2 : 1\ 2 \mid P_1 : 3 \mid P_2 : 3$

$R_1 = 11$ $R_2 = 9$

(Count = 9)

Count++ :

1) Write R_1, Count
2) Inc R_1
3) Write Count, R_1

Count-- :

1) Write R_2, Count
2) Dec R_2
3) Write Count, R_2

2.6 Critical sections

Rules for Critical Section:

Mutual Exclusion:- It makes sure that no more than one process executes in its critical section at any point in time.

Progress:- When there are no processes in the critical section, the processes that are not in the reminder section should continue within a certain time period.

Bound Waiting:-There exists a limit on the number of times other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

2.7 Mutual exclusion

Mutual Exclusion is a property of process synchronization that states that “no two processes can exist in the critical section at any given point of time”.

Mutual exclusion methods are used in concurrent programming to avoid the simultaneous use of a common resource, such as a global variable, by pieces of computer code called critical sections.

The requirement of mutual exclusion is that when process P1 is accessing a shared resource R1, another process should not be able to access resource R1 until process P1 has finished its operation with resource R1.

Deadlock

A set of n processes is called in a Deadlock state when every process of that set is waiting for some resource , which is occupied by some other process & that too is waiting for the resource occupied by the next waiting process.

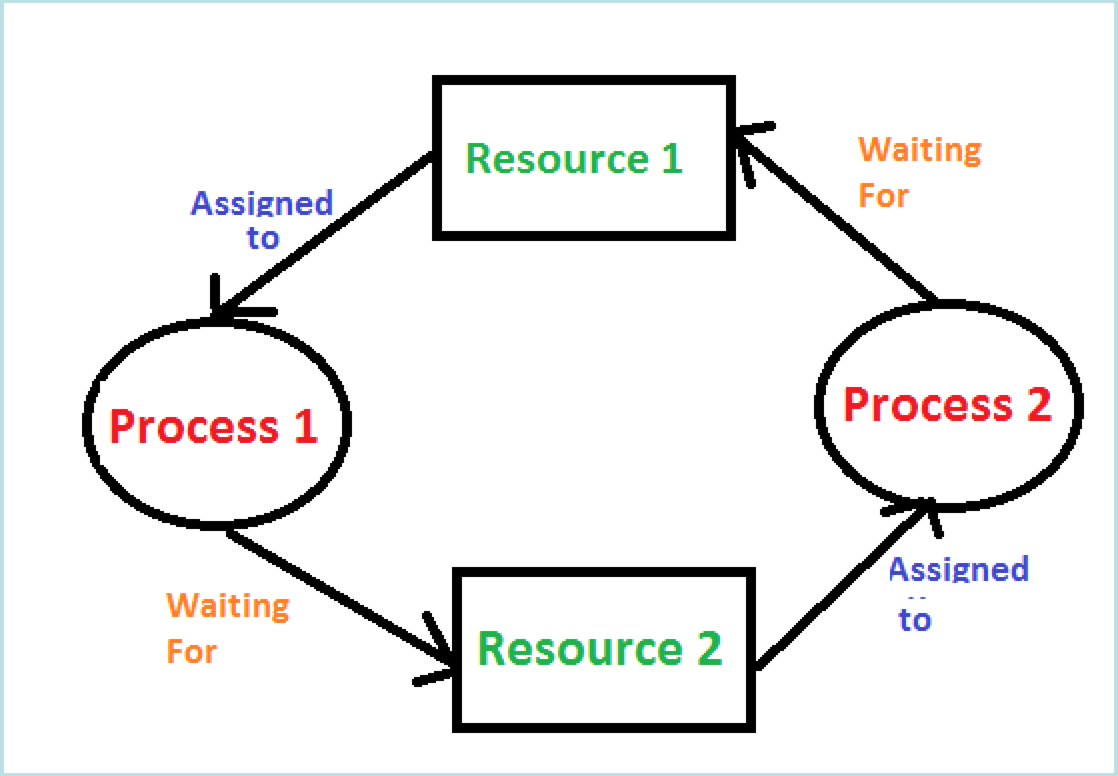
In deadlock all the processes will be in waiting & no one can complete the execution because of the unavailability of required no. of resources.

2.9 Deadlock

A ***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.

A process in operating system uses resources in the following way.

1. Requests a resource
2. Use the resource
3. Releases the resource



Necessary conditions for occurrence of Deadlock

1. Mutual Exclusion : At a time only one process can use the resource. If another process requests that resource then the requesting process must be delayed until the resource has been released.

2. Hold & Wait : A Process must be holding at least one resource & waiting to acquire additional resources that are currently hold by other process.

3. No Preemption : Resources can not be preempted, i.e. resources can be released by the process only after its completion.

4. Circular Wait : A set of waiting processes like { P0, P1,…,Pn} must exists such that P0 is waiting for a resource that is hold by P1, P1 is waiting for a resource that is occupied with P2 and with Pn is waiting for a resource that is held by P0.

Methods for handling Deadlock

1. Deadlock Prevention
2. Deadlock Avoidance

$\left.\begin{matrix}1\\2\end{matrix}\right\}$ Ensures that deadlock never occurs.

3. Deadlock detection & Recovery
4. Deadlock ignorance

Methods for handling Deadlock

1. Deadlock Prevention :

Deadlock prevention ensures that one of the four necessary conditions for deadlock cannot hold.

1. Mutual Exclusion Condition
2. Hold & Wait
3. No Preemption of Resources
4. Circular wait

Methods for handling Deadlock

1. Deadlock Prevention :

1. Mutual Exclusion Condition

- The mutual-exclusion condition exists with non sharable resources. Because non-sharable resources cannot be accessed by several processes at the same time. This condition can be avoided by the use of sharable resources. The sharable process can be accessed by more than one process at a time & so they need not require to wait.

Methods for handling Deadlock

1. Deadlock Prevention : 2. Hold & Wait Condition

- To ensure that Hold & Wait condition never occurs in the system, system must guarantee that whenever a process requests a resource, then it holds the other required resources. This algorithm says that a process should be allocated with all resources before starting the execution.

- Another approach says that a process should request the resource when it has none. i.e. before putting up the request, the process should release all its resources.

- But the disadvantage of these to approaches are :

  i. Resource utilization may be very low

  ii. Starvation is possible.

Methods for handling Deadlock

1. Deadlock Prevention : 3. No Preemption

- To prevent the deadlock this condition should not hold.
- To ensure that this condition does not hold, the following protocol can be used :
  - If a process is holding some resources & requests for another resource/s that can not be allocated to it immediately & so process may have to wait. In such case the system will ask the requested process to release its all resources voluntarily. So all such resources are preempted & released implicitly. With this the preempted resources are added to the list of available resources. Such process can restart only when it can reassign its old as well as new resources.

Methods for handling Deadlock

1. Deadlock Prevention : 4. Circular Wait

- By imposing the total ordering of all resources in an increasing order, we can ensure that circular wait condition can never hold.

Methods for handling Deadlock

1. Deadlock Prevention : 4. Circular Wait

- By imposing the total ordering of all resources in an increasing order, we can ensure that circular wait condition can never hold.

Possible Drawbacks of deadlock prevention are :

1. Low device Utilization
2. Reduced System Throughput

Methods for handling Deadlock

✓ Deadlock Prevention

1. Deadlock Avoidance
2. Deadlock detection & Recovery
3. Deadlock ignorance

Methods for handling Deadlock

2. Deadlock Avoidance

- It overcome the drawback of deadlock prevention.
- Avoidance can be there if system knows the requirement of each user in advance i.e. before starting the allotment.
- For this each process should declare the maximum number of resources of each type that it may require.
- A deadlock avoidance algorithm dynamically examines the resource allocation state before starting the allocation to ensure that a circular wait condition can never exist
- Here the resource allocation state is defined by the number of available resources & allocated resources & the maximum demand of the process.

Methods for handling Deadlock

2. Deadlock Avoidance

- This algorithm checks a system to find if, it is in safe state or in unsafe state.
- A system is in safe state if the system can allocate resources to each process up to its maximum demand.
- A system is said to be in safe state if and only if there exists a safe sequence.
- Safe sequence is the sequence which satisfies need of all processes.
- If no such sequence exists then the system is said to be in unsafe state.

An unsafe state may lead to deadlock, but not always. But a safe state is always not a deadlock state.

Methods for handling Deadlock

What is safe sequence ????

A sequence of processes P1,P2, ….,Pn is a safe sequence for the current allocation state if,

For each Pi process, the request can be satisfied by the currently available resources of Pi, plus the resources held by all the Pj, with j < i.

In this case Pi process can wait till all Pj have finished. When they have finished, Pi can obtain all of its required resources, then it will complete the task & return all its allocated resources to the list of available resources.

When Pi terminates, P_{i+1} can obtain its all resources & so can complete the task & so on.

Methods for handling Deadlock

Deadlock Avoidance Algorithms

1. Resource allocation Algorithm
2. Banker’s Algorithm
    a. Resource Request Algorithm
    b. Safety Algorithm

Deadlock Handling : Deadlock Avoidance Algo.

1. Resource allocation Algorithm

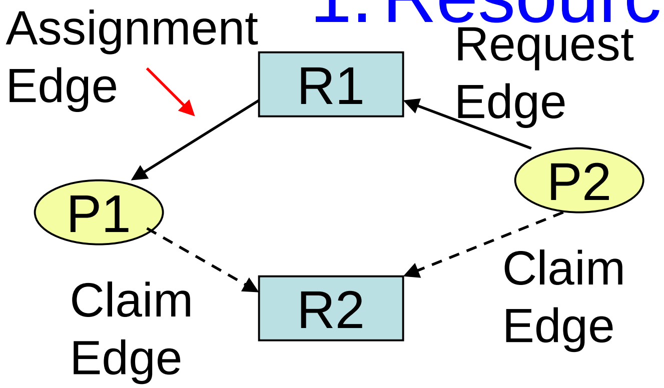
- Resource allocation graph consists of three edges :

  1. Request Edge (Pi, Rj) : Pi is requesting for Rj.
  2. Assignment Edge (Rj, Pi) : Rj is allocated to Pi
  3. Claim Edge (Pi, Rj) : Pi claims for Rj. Represented with dashed line. When in near future a request comes from Pi for Rj , then the claim edge is converted to request edge.

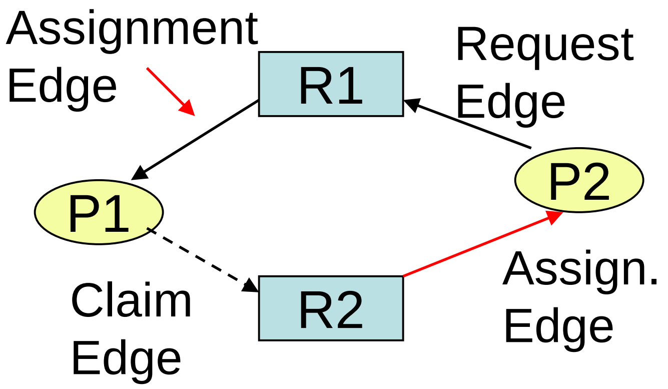
- Deadlock occurrence can be avoided by not allowing the formation of cycle in the resource allocation graph.
- With no cycle the system is in safe state.
- If cycle is found in the graph then that allocation will put the system in unsafe state.

Deadlock Handling : Deadlock Avoidance Algo.

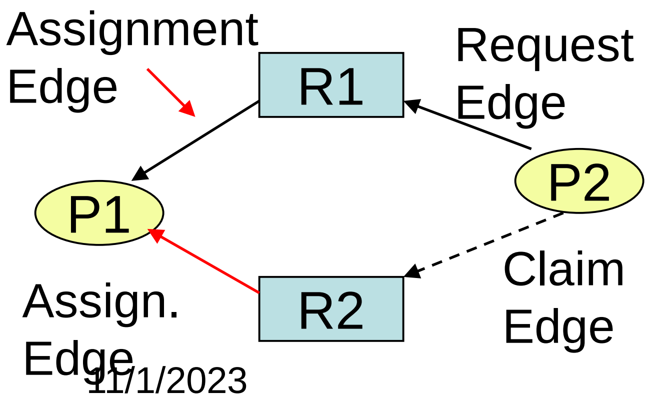
1. Resource allocation Algorithm



Assume that P1 & P2 are two processes & requires R1 & R2 resources for execution. At some point of time R1 is assigned to P1. P2 makes a request for R1. Further R2 is claimed by P1 as well as P2 for execution.



Currently R2 is free. Now if system allocate R2 resource to P2 then in resource allocation graph cycle will be formed. This will bring the system to an unsafe state.



But if system assigns R2 to process P1, then no cycle will be formed in the graph. This brings the system in safe state. With this P1 can finish off its execution with resources R1 & R2. After this R1 & R2 will be free & can be assigned to P2 for execution.

Deadlock Handling : Deadlock Avoidance Algo.

1. Resource allocation Algorithm

This algorithm is suitable for single instance of each resources but not for multiple instances of resources.

Deadlock Handling : Deadlock Avoidance Algo.

2. Banker’s Algorithm

a) Resource Request Algorithm

When a request comes for resources, then allocate them (Not actually) & observe the resource allocation state with Safety algorithm.

b) Safety Algorithm

Check for safe sequence in resource allocation state (Observed snapshot).

Deadlock Handling : Deadlock Avoidance Algo.

2. Bankers Algorithm

Used to avoid deadlock for multiple instances of resources.

Consider following set of data structure :

n = Total no. of processes in the system, m = the no. of resource types

Available = A vector of length m. It indicates the no. of available resources of each type. Available [j] = K means there are K instances of resource type j

Max = an n * m matrix. It defines the maximum demand of each process. Max [i, j] = K means process Pi may request maximum k instances of Rj resources.

Allocation = an n * m matrix. It defines the no. of resources currently allocated to each process. Allocation [i, j] = K means Pi is currently allocated with k instances of resource Rj.

Need = An n * m matrix. It indicates the remaining resource requirement

Need [I, j] = k means process Pi requires K more instances of resource type Rj.

$$\boxed{\text{Need } [i, j] = \text{Max } [i, j] - \text{Allocation } [i, j]}$$

Deadlock Handling : Deadlock Avoidance Algo.

Bankers Algorithm : a) Resource Request Algorithm

Let $request_i$ = request for process Pi , $Need_i$ = need for process Pi

If $request_i$ [j] = K // process Pi wants K instances of resource type Rj

When such a request is made by Pi then the following actions are taken :

Calculate the need of each job for each resource type (Ni=Mi-Ai)

1. Check to see if $request_i <= Need_i$. If yes go to step 2 else error, as process has exceeded its maximum claim.
2. if $request_i <=$ Available, if yes then go to step 3 else Resources are not available. Process Pi should wait.
3. The system will allocate the resources by modifying the states :

   Available = Available – $request_i$

   $Allocation_i$ = $Allocation_i$ + $request_i$

   $Need_i$ = $Need_i$ - $request_i$
4. Call the safety algorithm to check whether the system state is safe or unsafe after the allocation of resources.

At this stage if the resource allocation state is in safe state then the transaction is completed. Process Pi is allocated with its required resources. But if this allocation state is unsafe state then Pi must wait for $request_i$ & the old state is restored.

Deadlock Handling : Deadlock Avoidance Algo.

Bankers Algorithm : b) Safety Algorithm

Calculate the need of each job for each resource type (Ni=Mi-Ai)

1. Assume that : work = A vector of length m.
   Finish = A vector of length n.

2. Initialize : work = Available
   Finish[i] = false ; for i = 0 to n

3. Find an i (starting with first process), such that a) Finish[i] = false && b) $Need_i$ <= work, if found an i then got to step 4 else got to step 3 to check next i.

4. work = work + $Allocation_i$
   Finish [i] = true : Select the ith process, keep it aside & go back to step 3 for checking of the next process in sequence.

5. If Finish [i] = true for all values of i , then such system state is called as safe state.

Deadlock Handling : Deadlock Avoidance Algo.

Bankers Algorithm : Example to check the safe state

Consider a system with processing of five processes. A,B, & C are three types of resources. A has 10 instances, B has 5 & C has 7. At some time T0 the system reading is as follows :

P	Allocation			Max.			available			
	A	B	C	A	B	C	A	B	C	
P0	0	1	0	7	5	3	3	3	2	
P1	2	0	0	3	2	2				
P2	3	0	2	9	0	2				
P3	2	1	1	2	2	2				
P4	0	0	2	4	3	3				

Currently allocated resources

$\text{Need}_i = \text{Max}_i - \text{Allocation}$

Deadlock Handling : Deadlock Avoidance Algo.

Bankers Algorithm : Example

Consider a system with processing of five processes. A,B, & C are three types of resources. A has 10 instances, B has 5 & C has 7. At some time T0 the system reading is as follows :

P	Allocation			Max.			available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				0	1	1
P4	0	0	2	4	3	3				4	3	1

Currently allocated resources

$Need_i = Max_i - Allocation_i$

Bankers Algorithm : Safety Algorithm Example

Step 1. : work = Available = 3,3,2 , Initially Finish [i] = false

start with P0 Need0 = 7,4,3 Finish[0] = false step 2. : for i = 0 to 4 : compare Need0 <= Work ?? 7, 4, 3 <= 3, 3, 2 ….. No check for P1 : Need1 <= work ?? 1, 2, 2 <= 3, 3, 2 … Yes Step 3 : Fulfill the need of P1 by assigning the desired resources. Work = work + allocation_i work = 3, 3, 2 + 2, 0, 0 = 5, 3, 2 finish [1] = true select P1 process for safe sequence & go to step 2 for the similar checking of remaining processes i.e. for P2, P3, & P4

Finally when finish [i] for all P's will become true then it will form a safe sequence.

Bankers Algorithm : Safety Algorithm Example : solution

Step 1 find the need for each process : $Need_i = Max_i - Allocation$

P	Allocation			Max.			available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	3	2	7	4	3
P1	2	0	0	3	2	2				1	2	2
P2	3	0	2	9	0	2				6	0	0
P3	2	1	1	2	2	2				0	1	1
P4	0	0	2	4	3	3				4	3	1

Safe sequence is {P1,P3,P4,P0,P2}

Step 2 find the safe sequence with the help of safety algorithm:

Work = available = 3, 3, 2, finish [i] = false

P_i	$Need_i$ <= work	Y/N	Work = work + $Allocation_i$	Finish[i]
P0	7, 4, 3 <= 3, 3, 2	N		false
P1	1, 2, 2 <= 3, 3, 2	Y	Work = 3,3,2 + 2,0,0 = 5,3,2	True
P2	6, 0, 0 <= 5, 3, 2	N		False
P3	0, 1, 1 <= 5, 2, 2	Y	Work = 5,3,2 + 2,1,1= 7,4,3	True
P4	4, 3, 1 <= 7, 4, 3	Y	Work = 7,4,3 + 0,0,2= 7,4,5	True
P0	7, 4, 3 <= 7, 4, 5	Y	Work = 7,4,5 + 0,1,0= 7,5,5	True
P2	6, 0, 0 <= 7, 5, 5	Y	Work = 7,5,5 + 3,0,2= 10,5,7	True

Example of Bankers Algorithm

Consider the snapshot:

P	Allocation A B C D	Max. A B C D	Available A B C D	Need A B C D
P0	0 0 1 2	0 0 1 2	1 5 2 0	0 0 0 0
P1	1 0 0 0	1 7 5 0		0 7 5 0
P2	1 3 5 4	2 3 5 6		1 0 0 2
P3	0 6 3 2	0 6 5 2		0 0 2 0
P4	0 0 1 4	0 6 5 6		0 6 4 2

a) Is the system in safe state ?

b) If a request of resources (0, 4,2,0) arrives from P1 Process, can the request be granted immediately?

a) Step 1 : Find the need. (Max – Allocation) Apply the safety algorithm.

{P0, P2, P3, P4, P1}

b) If request comes from P1 for resources like 0 4 2 0 then to check whether it can be granted immediately or not, apply Resource Request Algorithm.

Available = Available – Requesti Allocation = Allocationi + Requesti Needi = Needi - Requesti

P	Requesti	Available	Requi<=Needi	Y/N	Requi<=Avail.	Y/N	Avail.	Alloca.	need
P1	0 4 2 0	1 5 2 0	0 4 2 0<=0 7 5 0	Y	0 4 2 0<=1 5 20	Y	1520 - 0420 =1100	1000 +0420 =1420	0750 -0420 =0330

Now apply the safety algorithm to see that system is in safe state after this allocation or not.

Example of Bankers Algorithm

a) Is the system in safe state ?

b) If a request of resources (0, 4,2,0) arrives from P1 Process, can the request be granted immediately?

Now the snapshot will be:

	Allocation	Max.	Available	Need
P	A B C D	A B C D	A B C D	A B C D
P0	0 0 1 2	0 0 1 2	1 5 2 0	0 0 0 0
P1	1 4 2 0	1 7 5 0	– 0 4 2 0	0 3 3 0
P2	1 3 5 4	2 3 5 6	= 1 1 0 0	1 0 0 2
P3	0 6 3 2	0 6 5 2		0 0 2 0
P4	0 0 1 4	0 6 5 6		0 6 4 2

Available = available - Request$_i$

Find the safe sequence with the help of safety algorithm :

Work = available = 1,1,0,0 finish [i] = false

P$_i$	Need$_i$ <= work	Y/N	Work = work + Allocation$_i$	Finish[i]
P0	**0,0,0,0 <= 1,1,0,0**	**Y**	**Work=1,1,0,0+0,0,1,2=1,1,1,2**	**True**
P1	**0,3,3,0 <= 1,1,1,2**	**N**		**False**
P2	**1,0,0,2 <= 1,1,1,2**	**Y**	**Work=1,1,1,2+1,3,5,4=2,4,6,6**	**True**
P3	**0,0,2,0 <= 2,4,6,6**	**Y**	**Work=2,4,6,6+0,6,3,2=2,10,9,8**	**True**
P4	**0,6,4,2 <= 2,10,9,8**	**Y**	**Work = 2,10,9,8+0,0,1,4 = 2,10,10,12**	**True**
P1	**0,3,3,0<=2,10,10,12**	**Y**	**Work=2,10,10,12+1,4,2,0 = 3,14,12,15**	**True**

System is in safe state with sequence {P0,P2,P3,P4,P1} & So request of Process P1 for resources (0,4,2,0) can be granted immediately.

Methods for handling Deadlock

- ✓ Deadlock Prevention
- ✓ Deadlock Avoidance

1. Deadlock detection & Recovery
2. Deadlock ignorance

Methods for handling Deadlock : Deadlock Detection

To implement deadlock detection algorithm :

a. System must maintain information about the current allocation of the resources to different processes along with any outstanding resource allocation requests.

b. System must provide & have an algorithm to determine whether the system has entered in to deadlock state.

   Deadlock detection algo. For single instance resources.

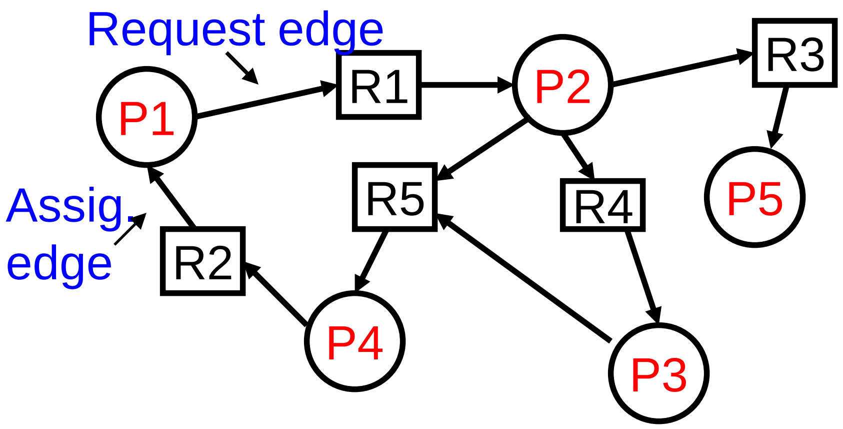
   Deadlock detection algo. for multiple instance resources

c. System must provide a recovery algorithm.

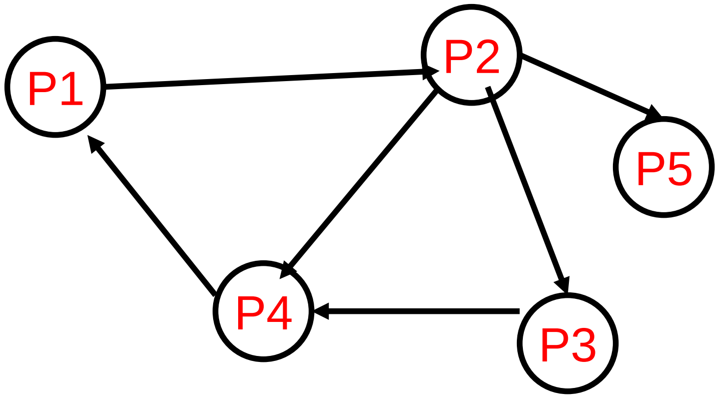
Methods for handling Deadlock : Deadlock Detection

Deadlock detection algorithm for single instance resources:

- This algorithm is also called as wait-for graph.
- This graph is derived from the resource allocation graph by removing the nodes of type resource and then by collapsing the appropriate edges.
- A deadlock exists in the system if & only if the wait-for graph contains a cycle.
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph.



Resource Allocation Graph



Wait-for Graph

Methods for handling Deadlock : Deadlock Detection

Deadlock detection algorithm for multiple instances of a resource type:

Assume the data structure set for this are :

Available = number of resources available of each type. Allocation = An n*m matrix, that determines the no. of allocated resources to each process. Request = An n*m matrix indicates the current request of each process. Request[i ,j] = K means process Pi is requesting K instances of resource type Rj.

Step1 : Let work & Finish are two variables of length m & n respectively. Initialize work= Available & Finish[i] = false for all i = 1 to n. Step2 : find an i such that Finish[i] = false && $Request_i$ <= work If no such i exists then go to step4 otherwise goto step3 to change the available (work) value. Step3 : work= work + Allocation, Finish[i] = true goto step2 for further checking. Step4 : If finish[i] = false for some i = 1 to n then the system is in deadlock. In this case the process Pi is deadlocked.

This operation requires m * n^2 operations to detect whether the system is in deadlock state or not?

Deadlock Detection algorithm : Multiple Instances

P	Allocation			Request			available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	0			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

Consider the snap sot of a system with 5 processes with 3 types of resources. Detect whether the deadlock is there or not.

Solution :

Set initially all Finish[i] = false work= Available

find an i so that Finish[i] = false && Request$_i$ <= Available

P$_i$	Request$_i$ <= work	Y/N	Work = work + Allocation$_i$	Finish[i]
P0	0,0,0 <= 0,0,0	Y	Work = 0,0,0 + 0,1,0 = 0,1,0	True
P1	2,0,2 <= 0,1,0	N		False
P2	0,0,0 <= 0,1,0	Y	Work = 0,1,0 + 3,0,3 = 3,1,3	True
P3	1,0,0 <= 3,1,3	Y	Work = 3,1,3 + 2,1,1 = 5,2,4	True
P4	0,0,2 <= 5,2,4	Y	Work = 5,2,4 + 0,0,2 = 5,2,6	True
P1	2,0,2 <= 5,2,6	Y	Work = 5,2,6 + 2,0 0 = 5,2,8	True

Deadlock is not detected with sequence {P0,P2,P3,P4,P1}

Deadlock Detection algorithm : Multiple Instances

P	Allocation			Request			available		
	A	B	C	A	B	C	A	B	C
P0	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	2			
P2	3	0	3	0	0	1			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			

If P2 demands a resource type C – 1 instance then detect the deadlock???

Solution :

Set initially all Finish[i] = false   work= Available

find an i so that Finish[i] = false && $Request_i$ <= Available

P_i	$Request_i$ <= work	Y/N	Work = work + $Allocation_i$	Finish[i]
P0	0,0,0 <= 0,0,0	Y	Work = 0,0,0 + 0,1,0 = 0,1,0	True
P1	2,0,2 <= 0,1,0	N		False
P2	0,0,1 <= 0,1,0	N		False
P3	1,0,0 <= 0,1,0	N		False
P4	0,0,2 <= 0,1,0	N		False
P1	2,0,2 <= 0,1,0	N		False

Deadlock is detected with processes P1, P2,P3,P4 are deadlocked

Methods for handling Deadlock

- ✓ Deadlock Prevention
- ✓ Deadlock Avoidance
- ✓ Deadlock detection &
- Recovery
- Deadlock ignorance

Deadlock Recovery

Deadlock Recovery

Two solutions are there for deadlock recovery :

1. Abort one or more processes to break the circular wait.

2. Resource Preemption.
    - Selecting a victim
    - Roll-Back
    - Starvation

Deadlock Recovery

1. Abort one or more processes to break the circular wait.

- Abort all deadlocked processes. This method will break the cycle, but is very expensive.
- Abort one process at a time until the deadlock cycle is eliminated. But this method includes many overheads, because after each process is aborted, a deadlock detection algorithm must be invoked to detect whether any process still causes deadlock or not.
- By aborting a process, system can reclaim all the resources allocated to the terminated process & include them in the list of available resources to fulfill the need of other processes.

Deadlock Recovery

1. Abort one or more processes to break the circular wait. Selection of process to be terminated depends upon :

1. The priority of process.
2. How much time it will take to complete the assigned task & how long it has computed.
3. How many & what type of resources the process has used.
4. What is the resource need i.e how many resources the process required to complete the assigned task.
5. What is the type of process – interactive or batch
6. How many processes needs to be terminated.

Deadlock Recovery

2. Resource Preemption

- This method says that preempt some resources from one or more of the deadlocked processes and then allocate these preempted resources to other processes.
- This is repeated with multiple processes until the deadlock cycle is broken.

Preemption of resources requires to consider three different issues for deadlock recovery

- Selecting a victim
- Roll-Back
- Starvation

Deadlock Recovery

- Selecting a victim
  - Victim is that process which is selected for preemption.
  - The selection is based on following parameters :
    - The cost
    - The time resources were associated with a process.
    - How much time is left for the process for its completion.
    - The number of resources a process is currently holding and how much it requires more.

Deadlock Recovery

- Roll-Back
  - When a process is preempted from its resources then that process can not continue with its normal execution.
  - At this stage such process must be roll-back to some safe state and then should restart it from that state.
  - But since it is difficult to decide the safe state for any such process, & so for simplicity, system does the total roll-back.
  - Total roll-back means abort such process & then restart it.

- Starvation :
  - Preemption of resources from a process may lead to starvation.
  - System must take care that the resources must not be preempted always from the same process.