UNIT-III

CPU Scheduling: Scheduling Concepts, Performance Criteria, Process States, Process Transition Diagram, Scheduler, Process Control Block (PCB), Process address Space, Process identification information, Threads and Their Management, Scheduling Algorithms, Multiprocessor Scheduling, Deadlock System Model, Deadlock Characterization, Prevention, Avoidance and detection, Recovery from deadlock.

I.CPU Scheduling

(A).Scheduling Concept

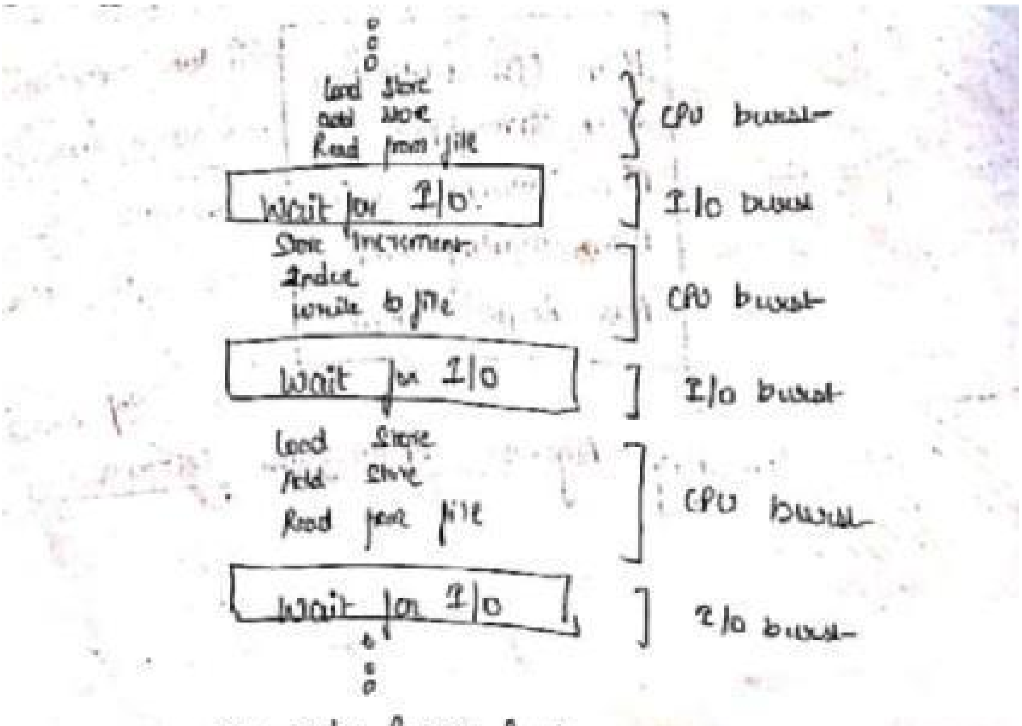
CPU scheduling is the basis of multiprogrammed operating system. By switching the CPU among process, the operating system can make the computer more productive.

Several jobs are ready to run at the same time, the system must choose among them, making this decision is CPU scheduling.

In a single processor system, only one processor can run at a time, other must wait until the CPU is free one can be rescheduled.

The success of CPU scheduling depends on an observed property of processes. Process execution consist of cycles of CPU execution and input output wait. Process alternate b/w these 2 stages.

Process execution begins with CPU burst i.e. followed by input/output burst, which is followed by answer CPU burst and then another input / output burst and so on. Eventually, the final CPU burst ends with a System request to terminate execution.



CPU - I/O Burst Cycle

(B).Scheduling Criteria

The criteria include the following-

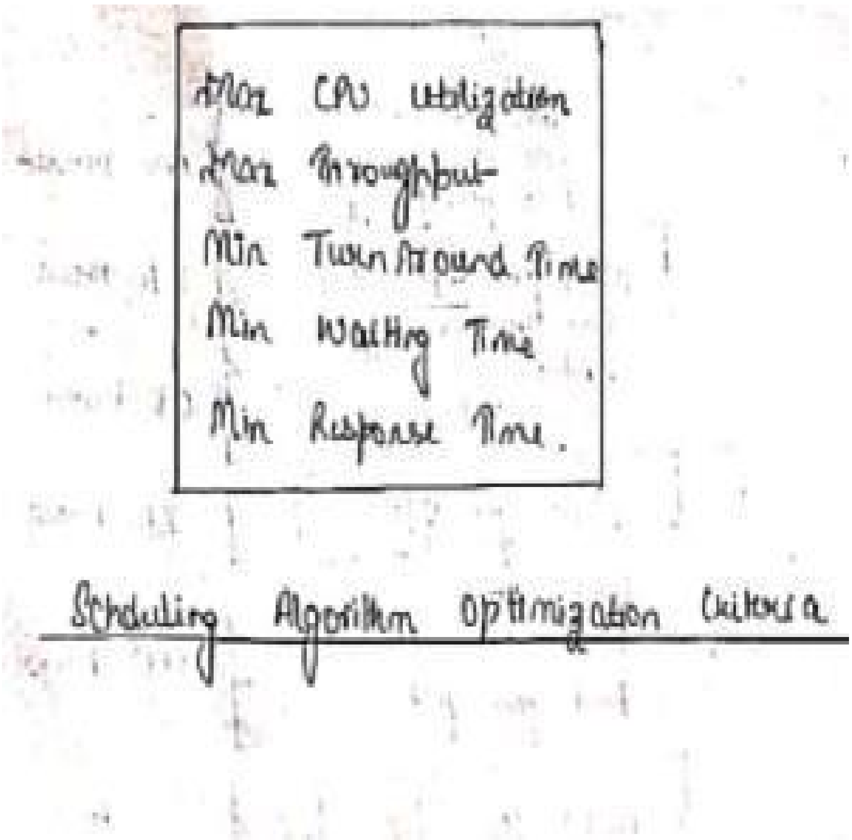
(B.1)CPU utilization-We want to keep the CPU as busy as possible.

(B.2)Throughout- If the CPU is busy executing processes, then work is being done one measure of work is the no of processes and are completed per unit time, Call throughout. For long processes, this rate maybe one, processes per hour and for short processes, it may be ten, process per second.

(B.3)Turnaround time– The interval from the time of submission of a process to the time of completion is called Turnaround time.

(B.4)Waiting time–It is the sum of periods spent waiting in the ready Queue.

B.5) Response time–Another measure is the time from the submission of the request until the first response is produced. This measure called response time is the time it takes to start responding, not the time it takes to output the response.



Max CPU utilization
Max Throughput
Min Turn Around Time
Min Waiting Time
Min Response Time.

Scheduling Algorithm optimization Criteria

(C).Process Transition Diagram:

Fig shows a process transition diagram

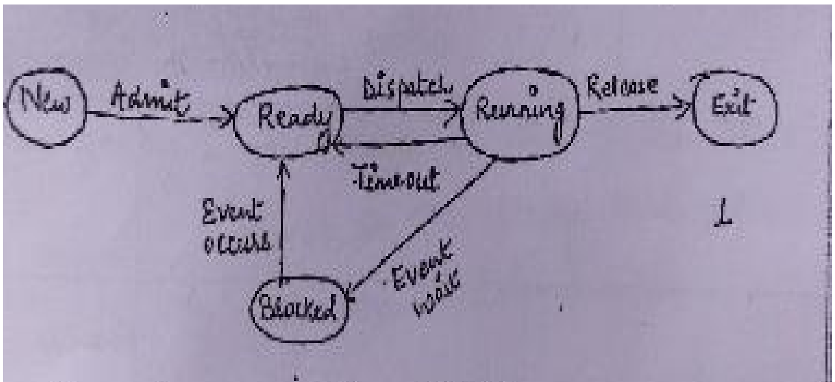


Fig 1: Process Transition Diagram.

The possible transition are as follows

1.A new process is created to execute a Program.

2.New- Ready: The operating system will move a process from the new state to ready state when it is prepared to take on an additional process.

3.Ready– Running: When it is time to select a process to run, the operating system chooses one of the process in the ready state.

4.Running- Exit: The currently running process is terminated by the operating system if the process indicates that it has completed.

5.Running- Ready: The most common reason for this transaction is that the running process as reached the maximum all allowable time for uninterrupted execution.

6.**Running- Blocked:** A process is put in the blocked state if it is requests something for which it must wait, For example- A process may be blocked when it waiting for another process to provide data or waiting for a message from another process.

7.Blocked - Ready: A process in the blocked state is moved to the ready state when the event for which it has been waiting occurs.

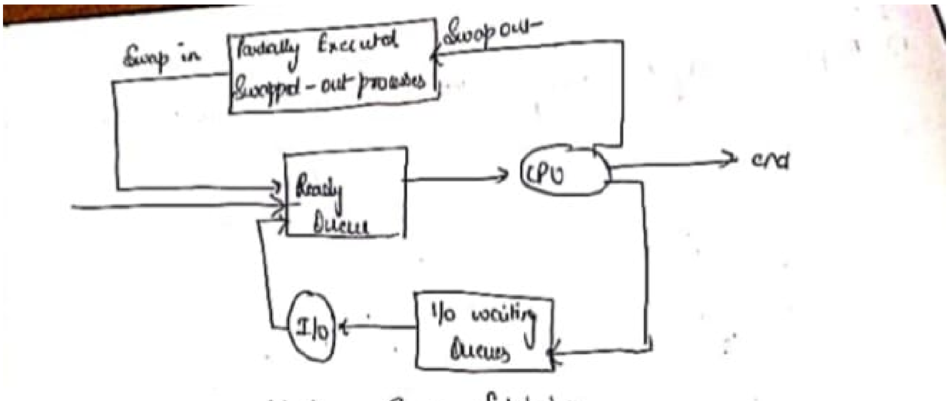
8.Ready- Exit: For clarity, this transition is not shown on the state diagram. In some systems, a parent may terminate a child process at any time. Also, if parents terminates, all child processes associated with that parent may be terminated.

(D).Scheduler

1. A process migrates among the various scheduling queues throughout its life time.
2. The operating system must select, for scheduling purposes, processes from these queues in some fashion. The selection process is carried out by the appropriate scheduler.
3. Often in a batch system, more processes are submitted than can be executed immediately. Those processes are spooled to a mass-storage device (typically a disk),where they are kept for later execution.
4. The long-term scheduler or job scheduler, selects processes from this pool and loads them into memory for execution.
5. The short-term scheduler or CPU scheduler, selects from among the processes that are ready to execute all allocates the CPU to one of them.
6. The long-term scheduler controls the degree of multiprogramming (the number of processes in memory). If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.
7. Thus the long-term scheduler may need to be invoked only when a process leaves the system. Because of longer interval between executions, the long-term scheduler can afford to take more time to decide which process should be selected for execution.
8. It is important that the long term scheduler make a careful selection. In general, most processes can be described as Either I/O Bound or CPU Bound.

- An input output Bound process is one that spends more of its time doing input output than it spends doing computations.
- A CPU Bound process, in contrast, generates input output request in frequently, using more of its time doing computations.

9. It is important that the long-term scheduler select a good process mix of input output Bound and CPU Bound processes.
10. On some system, the long-term scheduler may be absent or minimal. For example, time sharing system such as UNIX and Microsoft Windows system often have no long-term scheduler but simply put every new process in memory for the short-term scheduler.
11. The stability of these systems depends either on a physical limitation (such as the number of available terminals) or on the self-adjusting nature of human users.
12. Some operating system, such as time-sharing system, may introduce an additional, intermediate level of scheduling, this medium-term scheduler is diagrammed.



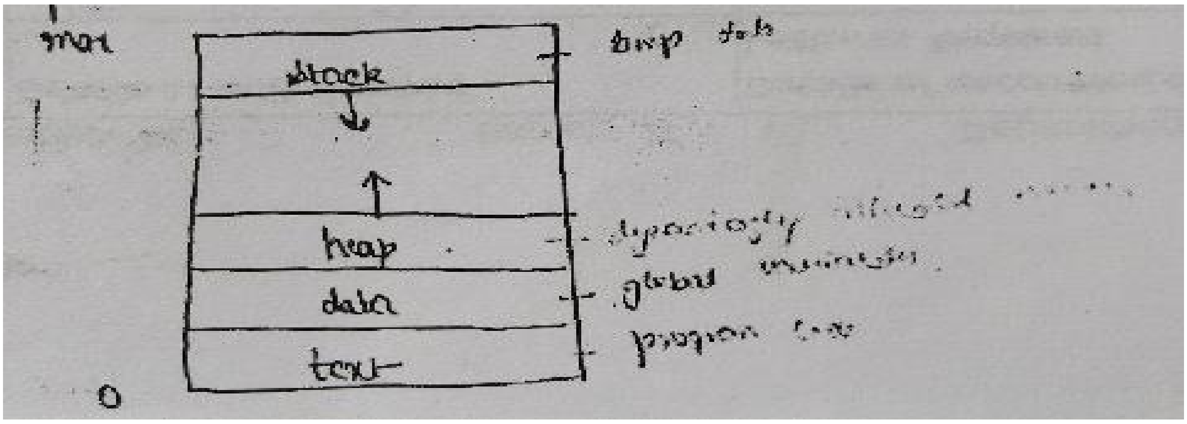
Medium Term Scheduler

Idea: Sometimes it can be advantages to remove processes from memory and thus reduce the degree of multiprogramming. Later, the process can be reintroduced into memory, and its execution can be continued where it left off. This scheme is called Swapping.

(E).Process Address Space

Initially, a process is a program in execution.

- A process is more than a program code, which is called a text section.
- It also includes the current activity, as represented by the value of program counter.
- A process also includes the process stack, which contains temporary data.
- A Data section which contains Global variables.
- A process may also include a heap, which is memory i.e. dynamically allocated the during process run time.



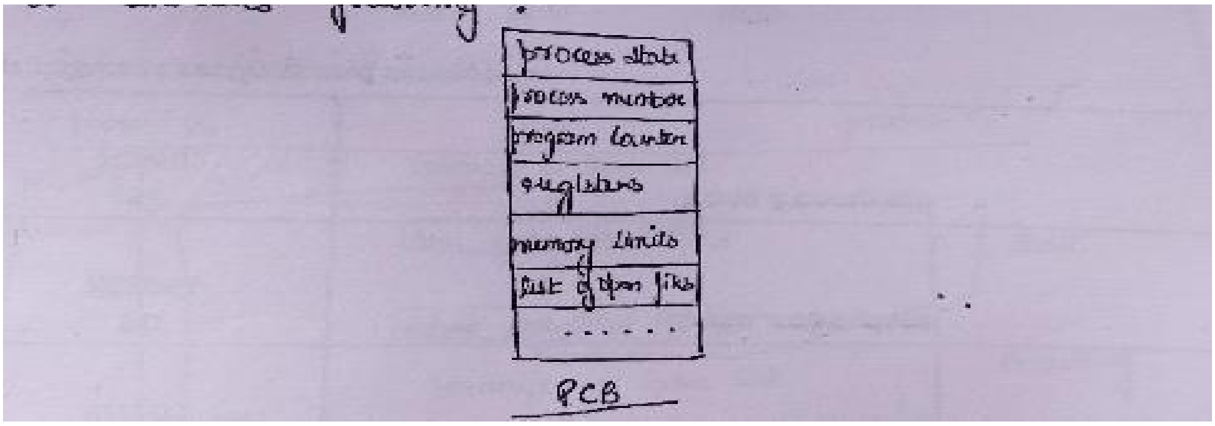
(F).Process Identification Information:-

In Virtually all operating system, each process is assigned a unique numeric identifier. The identifier may simply be an index into the primary process table, if there is no numeric identifier, there must be the mapping that allows the operating system to locate the appropriate tables on the basis of process identifier. This identifier is useful in variety of ways. Many of the other tables controlled by the operating system may use processes identifier to cross the reference process tables.

(G).Process Control Block

Each process is represented in the OS by a process control block (PCB) – also called a Task Control Block (TCB).

It contains following:-



PCB

- **<u>Process State</u>:** The state may be new, ready, running, waiting and exit.
- **<u>Program Counter:</u>** The counter indicates the address of the next is not to be executed for this process.
- **<u>CPU Registers:</u>** The registers vary in number and type depending on the computer architecture.
- **<u>CPU Scheduling Information:</u>** This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.
- **<u>Memory Management:</u>** This information may include such a information as the value of the base and limit registers, the page tables or segment tables depending on memory system used by operating system.
- **<u>Accounting Information:</u>** This information includes the amount of CPU and real time used, time limits, account numbers, and so on.
- **<u>I/O Status Information-</u>**<u>This info includes the list of I/O devices allocated to the process, a list of open files and so on.</u>

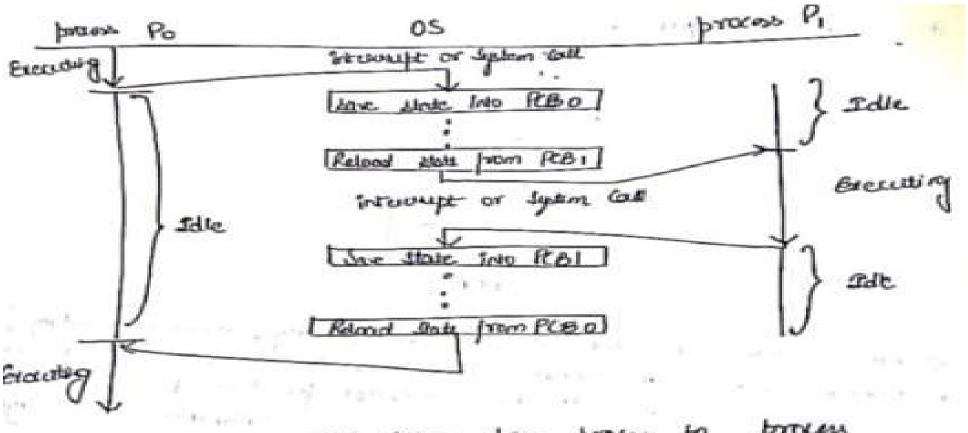


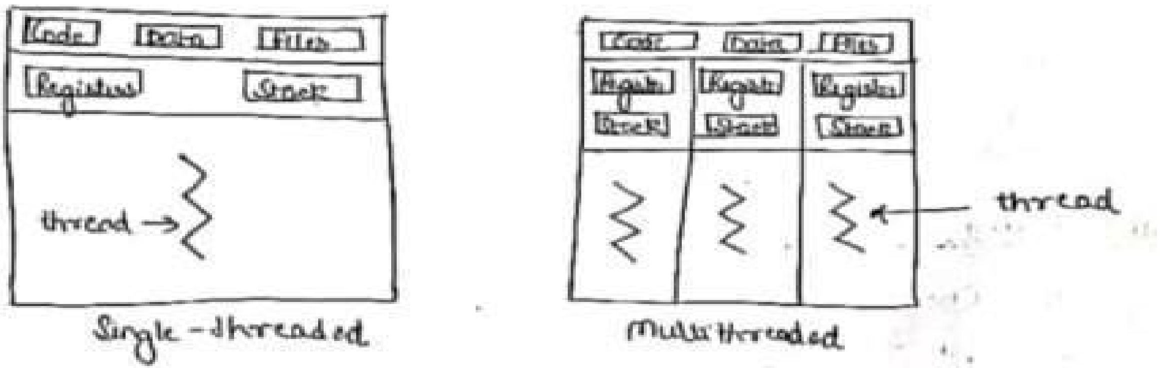
Diagram showing CPU switch from process to process

(H).Threads And Their Management

Introduction

A thread sometimes called a light weight process (LWP) is a basic unit of CPU Utilization, it comprises a thread ID, a program counter, a register set and a Stack.

A traditional (or heavy weight) process (HWP) has a single thread of control. If the process has multiple threads of control, it can do more than one task at a time.



Single and multithreaded processes

Benefits

The benefit of multithreaded programming can be broken down into 4 major categories-

- Responsiveness
- Resources sharing
- Economy
- Utilization of Multiprocessor Architecture.

User and Kernel Thread

User Thread:-User Threads are supported above the kernel and are implemented by a thread library at the user level. The library provides support for thread creation, scheduling and management with no support for the Kernel.

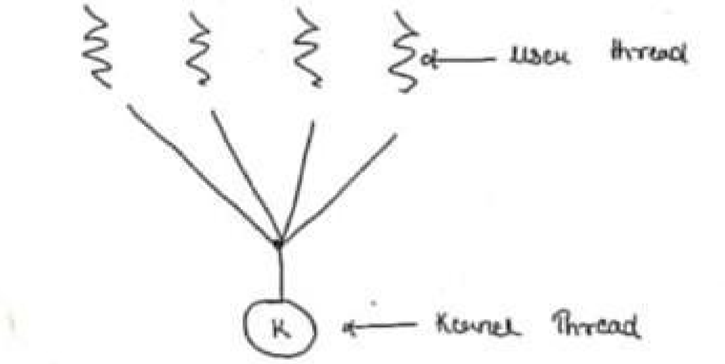
User-level Threads are generally fast of create and manage.

Kernel Thread: are supported directly by the operating system. The Kernel performs thread creation, Scheduling and management in Kernel space. Because thread management is done by the operating system, Kernel threads are generally slower to create and manage than user threads.

Multithreading Models-

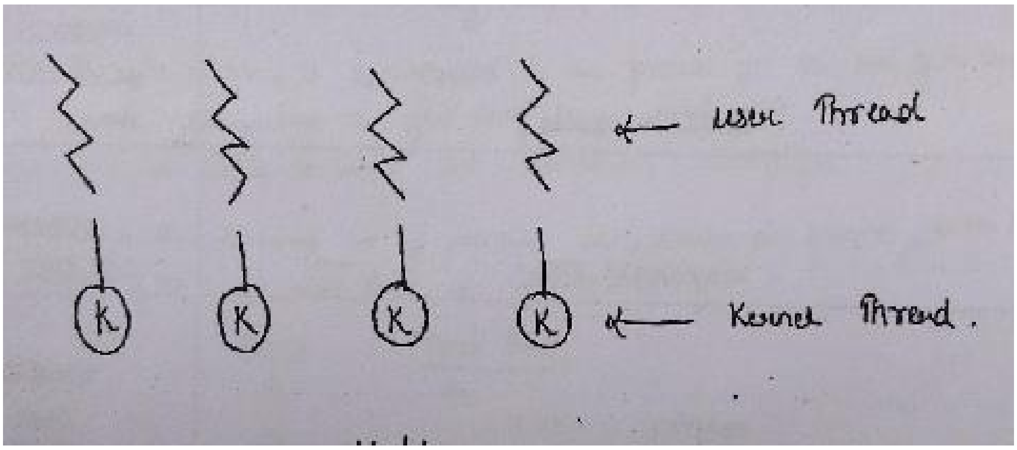
(i).Many- to- One- Model

The many-to-one-model maps many user-level threads o one Kernel thread. Thread management is done in user space, so it is efficient, but the entire process with black if a thread makes a blocking system call.



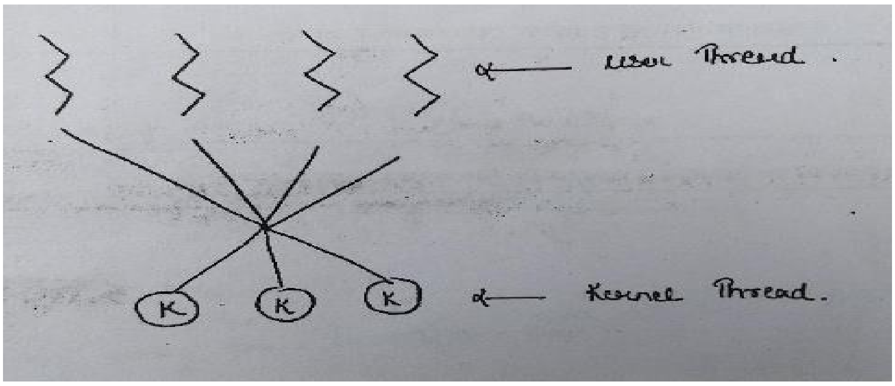
(ii)One-To-One Model:

The one-to-one model maps each user thread to a kernel thread. It provides more concurrency than the many-to –one model by allowing another thread to run. When a thread makes a blocking system call; it also allows multiple threats to run in paralled on multi processors.



(iii)Many-to-Many Model:

The many-to-many model multiplexes many user-level threads to a smaller or equal number of kernel threads.



H. Scheduling Algorithms

(A).First Come, First Serve Scheduling:- (FCFS)

The simplest CPU scheduling algorithm is FCFS. The process that requests the CPU first is allocated the CPU first.

The implementation can be managed with a FIFO queue.

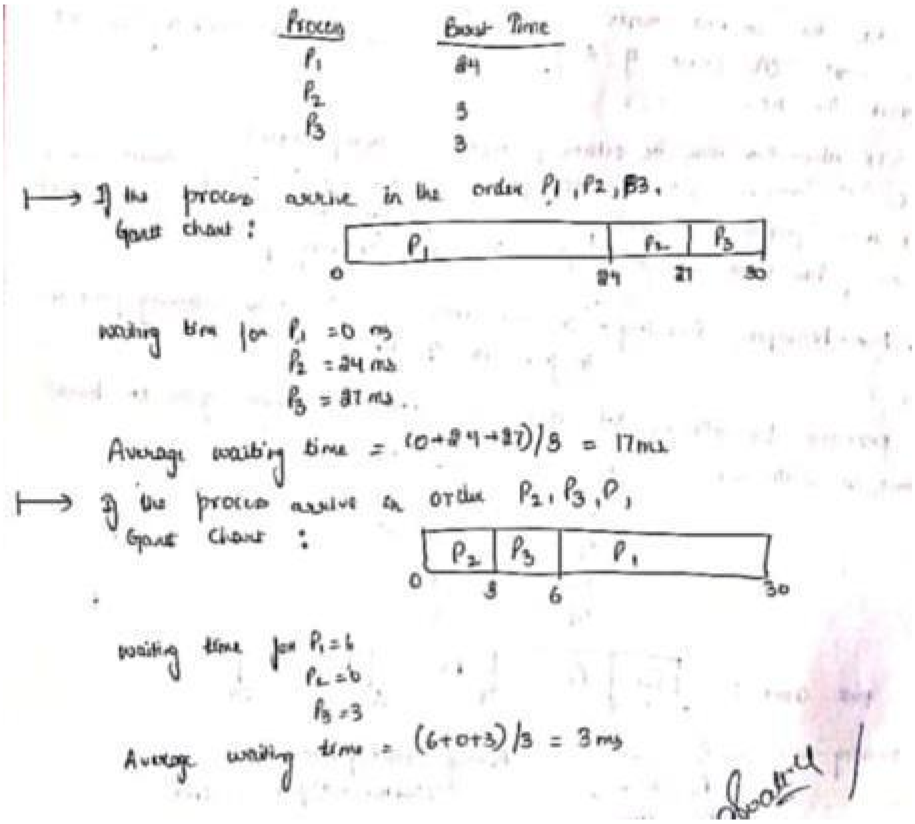
When the process enters the ready queue, its PCB is linked on to the tail of the queue.

When the CPU is free, it is allocated to the process at the head of queue.

The average waiting time is often quit long-**disadvantage**.

The code is simple to write and understand-**advantage**.

Ex- Consider the following set of processes that arrive at time 0, with the length of the CPU burst time in milliseconds-



Process	Burst Time
P_1	24
P_2	3
P_3	3

⟼ If the process arrive in the order P_1, P_2, P_3,
Gantt chart :

P_1	P_2	P_3

0 24 27 30

waiting time for $P_1 = 0$ ms
$P_2 = 24$ ms
$P_3 = 27$ ms.

Average waiting time $= (0+24+27)/3 = 17$ms

⟼ If the process arrive in order P_2, P_3, P_1
Gantt chart :

P_2	P_3	P_1

0 3 6 30

waiting time for $P_1 = 6$
$P_2 = 0$
$P_3 = 3$

Average waiting time $= (6+0+3)/3 = 3$ms

NOTE:- FCFS scheduling algorithm is non preemptive. (it means once the CPU allocated to a process that process, that process keeps the CPU until it releases CPU, either by terminating or by requesting input output).

FCFS algorithm is particularly troublesome for time sharing systems. There is CONVOY effect as all the other processes wait for one big process to get off the CPU. This effects result in lower CPU utilization and device utilization than might be possible is the shorter processes are allowed to go first.

(B).SHORTEST-JOB FIRST SCHEDULING (SJF)

This algorithm Associates with each process the length of processes next CPU burst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst.

If the next CPU burst of 2 processes are same, FCFS scheduling is used to break the tie. The SJF algorithm may be Either Preemptive or non preemptive.

The choice arises when a new process arrives at a ready queue While a previous process isexecuting. The new process may have a shorter next CPU burst than what is left of the currently executing process.

(B.1)Non-Preemptive Scheduling-It will allow the currently running process to finish its CPU burst.

Ex-Consider the following set of processes, with the length of the CPU burst time in milliseconds-

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

Gantt chart :

P_4	P_1	P_3	P_2

0 3 9 16 24

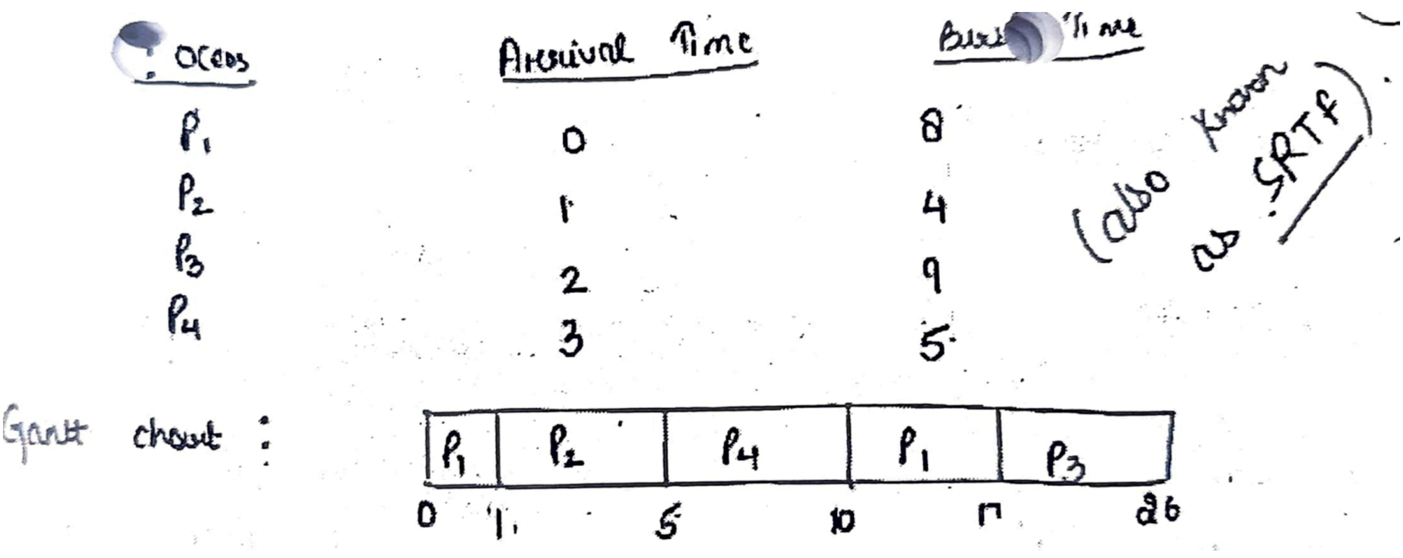
waiting time for P_1 = 3 ms

P_2 = 16 ms

P_3 = 9 ms

Average waiting time =

(3+16+9+0)/4 =



Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

(also known as SRTF).

Gantt chart :

P1	P2	P4	P1	P3

0 1 5 10 17 26

Process P1 started at time 0, since it is the only process in Queue. Process P2 arrives at time 1. The remaining tine for P1 (7ms) is largerthan the time required by P2 (4ms), to P1 is preempted and P2 is scheduled.

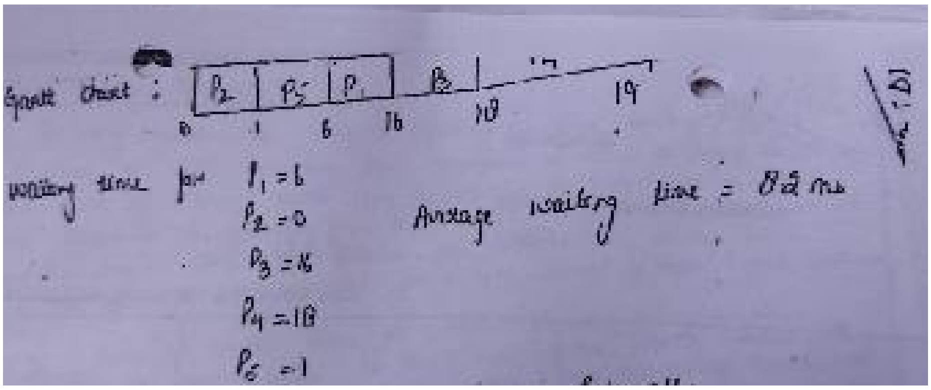
Average waiting time= [(10-1)+(17-2)+(5-3)]/4 = 26/4 = 6.5ms

<u>(C).Priority Scheduling:</u>

The SJF algorithm is a special case of general priority schedule in algorithm. A Priority (larger) is associated with each process, and the CPU is allocated to the process with the highest priority. Equal priority processes are schedule in FCFS order.

On SJF algorithm is simply a Priority algorithm where the priority is the inverse of next CPU burst. The larger the CPU burst, the lower the priority.

Ex:- Consider the following set of processes, assumed to have arrive at time 0 in the order P1, P2,, P5, with the length of CPU burst.



Gantt chart :

P2	P5	P1	P3	P4

0 1 6 16 18 19

waiting time for P1 = 6

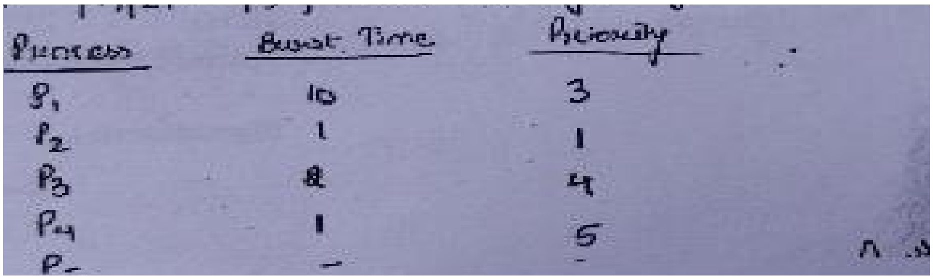
P2 = 0

P3 = 16

P4 = 18

P5 = 1

Average waiting time = 8.2 ms



Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	[illegible]	[illegible]

Priorities can be defined either internally or externally. Internally define priorities use some measurable quantity concept the priority of process such as time limits, memory requirements the no of open files etc.

Problem with priority scheduling **indefinite blocking or starvation.**

A process that is ready to run but lacking the CPU can be consider Loco-priority process waiting indefinitely for the process. A Priority scheduling algorithm can leave some low- priority process waiting indefinitely for the CPU.

A solution to the problem of indefinite blockage of low priority process is aging. Aging is a technique of gradually increasing the priority of process that wait in the system for a long time.

For example, 4 priorities range from 0 (low) to 121 (high), we could increment the priority of a waiting process by 1 every 15minutes. Eventually, even a process with an initial priority of 0 word have the highest priority in the system and would be executed.

In fact, it would take no more than 32 hours for a Priority 0 process to age to a Priority 127 process.

(D)Round Robin Scheduling (RR):-

Round Robin Scheduling algorithm Is design especially for time-sharing systems. It is similar to FCFS scheduling but preemption is added to switch b/w processes. A small unit of time, called time Quantum, or time slice is defined.

A time Quantum is generally from 10 to 100 microseconds. The ready queue is treated as circular queue. The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to one time Quantum.

Implement:-

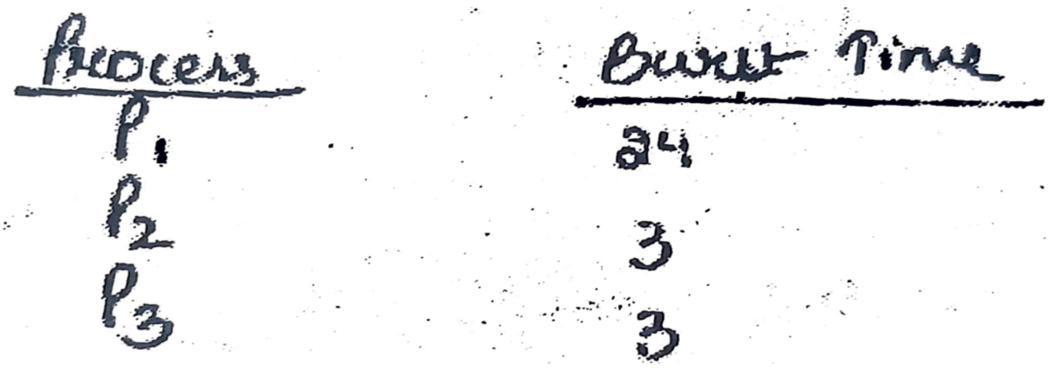
Keep the ready queue as a FIFO queue of process. New processes are added to the tail of the ready queue.

TheCPUschedulerpicksthefirstprocessfromthereadyqueue,setsatto interrupt after 1 time Quantum and dispatches the process.

One of the two things will happen—

- The process may have CPU burst of less than 1 time Quantum. In this case, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue.
- Otherwise, if the CPU burst of the currently running process is longer than 1 time Quantum, the time will go off and will cause an interrupted to the operating system. A context switch will be executed and the process will be put at the tail of the ready queue. The CPU schedule will then and selected the next process in the ready to queue.

  Ex-Consider the following set of processes that arrive at, time 0,with the length of CPU burst in ms:



Process	Burst Time
P1	24
P2	3
P3	3

Time Quant am=4ms

Gant Chart:

P1	P2	P3	P1	P1	P1	P1	P1

0 4 7 10 14 18 22

26 30

Waiting Time for P1=(10-4)

P2=4

P3=7

Average Waiting Time=5.66 ms

Note:-In the RR scheduling, no process is allocated the CPU for more than Quantum in a now. If a process CPU burst exceeds 1 time Quantum process is preempted and is put backin the ready queue. The RR school is thus preemptive-

- Typically higher average turn around the SJF, but better response.
- Quantum should be large compared to context switch time.
- Quantum usually10ms to100ms,context switch 210ms.

Performance:-

- Q large–FIFO
- Q small-Q must be large context switch, otherwise overhead is too high.

(E).Multilevel Queue Scheduling Algorithm:-

In thus the processes are easily classified into groups. For example- a common division is made between foreground (interactive) process and background (batch) process.

These two types of processes have different response time requirements and so many have different scheduling needs.

- It priorities the ready queue in several separate queue.
- The processes are permanently assigned to the one queue, generally based on some property of the process, such as memory size, process priority or process type.

Each queue has its own scheduling algorithm:

- Foreground - RR
- Background-FCFS

Scheduling must be done b/w and queues:

- Fixed priority scheduling(i.e. serve all from fore ground than from background). Possibility of standard.
- Time slice-Each Queue gets a certain amount of CPU time which it can schedule it amongst its processes i.e. 80% to foreground in RR.
- 20% to background in FCFS which is commonly implemented as fixed preemptive scheduling.

F.Multilevel Feedback Queue Scheduling Algorithm:-

When the multilevel queue scheduling algorithm is used, processes are permanently assigned to a queue when they enter the system.

If there are separate queues for foreground and background process, for example, processes don't move from one queue to another since process don't change their foreground or background nature. This setup has the advantage of low scheduling overhead but it is inflexible.

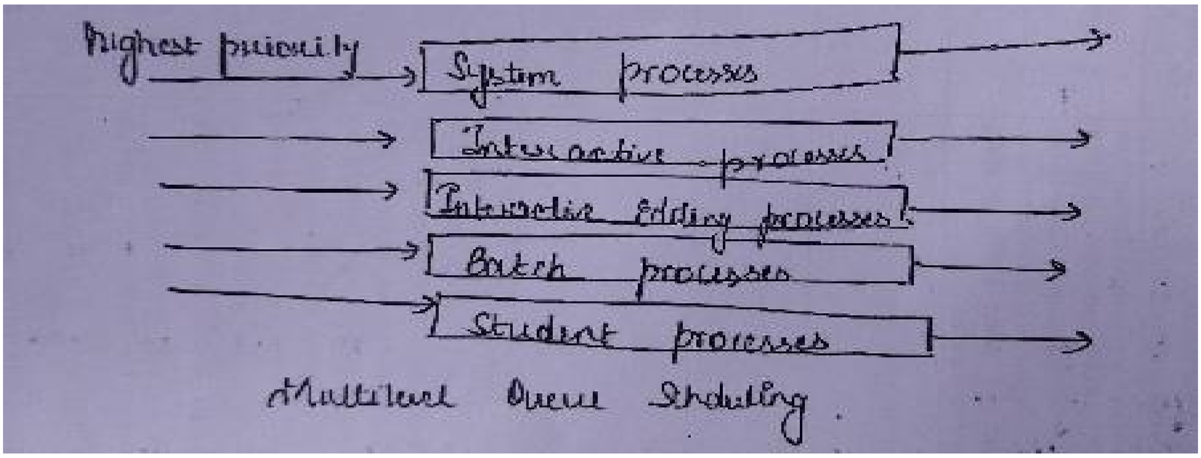
- It allow a process to move b/w queues.
- The idea is to separate processes according to the characteristics of their CPU burst. If

process uses too much CPU time, it will be move to lower priority queue.

- If the process uses too much CPU time, it will be moved to lower priority queue.
- In addition, a process that waits too long in a lower priority queue may be moved to a higher priority queue. This form of aging prevents starvation.

Ex-Consider an example of multilevel queue scheduling algorithm queues, listed below in order of priority:-

- System process
- Interactive process
- Interactive editing process
- Batch process
- Student process



- Each queue has absolute priority over lower priority queues. No process in the batch queue. For example, could run unless the queue for system process, interactive process, and interactive editing process were all empty.

- If an interactive editing process entered the ready queue while a batch process was running, the batch process would be preempted.

(I).Multi Processor Scheduling:-When a computer system contains more than a single processes, several new issues are introduced into the design of scheduling function.

Multiprocessor System can be categorized as follows:

- **Loosely coupled or distributed multi processor:** consist of a collection of relatively autonomous systems, each processor having its own memory and input output channels.
- **Functionally Specialized Processor:** an example is input output processor. In this case, there is a master, general propose processor; specialized processors are controlled by the master processor and provides service to it.
- **Tightly coupled Multiprocessing:** consist of set of processor that share a common main

memory and are under the integrated control of any operating system.

Scheduling on a Multiprocessor involves 3 interrelated issues.

(i) The assignment of process to processor.

(ii) The use of multiprogramming on individual processors.

(iii) The actual dispatching of a process.

Assignment of process to processor: If we assume that architecture of the multiprocessor is uniform, in the sense that no particular physical advantage with respect to access to main memory or to input output devices, then the simplest scheduling approach is to treat processor as a pooled resource and assign processes to processor on demand.

They are two approaches, that can be used for assigning processes to processor.

1.Masterlslave Architecture: In this type of architecture, key Kernel functions of operating system always run on a particular processor.The other processor may only executes user programs.The master of responsible for scheduling jobs. Once a process is active, if the slave needs service, it must send request to the master and wait for the service to be performed.

There are two **disadvantage** of this approach.

1.A failure of the master brings down the whole system.

2.The master can become a performance bottleneck.

2.Peer Architecture : In a peer architecture, the kernel can execute on any processor, and each processor does self scheduling from the pool of available processes.

The operating system must ensure that two processor do not choose the same process and that the processes are not somehow lost from the queue.

Techniques must be employed to resolve and synchronize competing claims to resources.

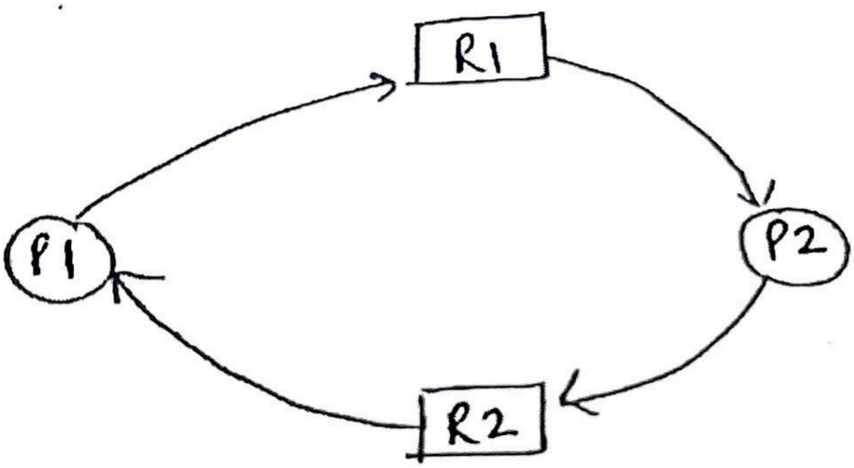
DEADLOCK

(A).INTRO:

In a multiprogramming Environment, several process may complete for a finite number of resources. A process request resources, if there sources are not available at that time, the process enter a wait state.

It may happen that waiting processes will never again change state, because the resources they have requested are held by other waiting processes. This situation is called a deadlock.

(B).SYSTEM MODEL:



A System consists of finite number of resources to be distributed among a number of competing processes. There sources are partitioned into several types, each consisting of some number of identical instances.

A Process must request are source before using it and must release the resource after using it. A process may request as many resources as it requires to carry out its designed task. Obviously, the number of resources requested may not exceed the total number of resources available in the system.

Under the normal mode of operation, a process may utilize are source in only the following sequence.

Request: The process request the resource. If the request cannot be granted immediately, then the requesting process must wait until it can acquire the resource.

Use: The process can operate on the resource (for example if the resource is a printer, the process can print on the printer).

Release: The process releases there source.

Definition: A set of processes is in a deadlocked state when every process in the set is waiting, for an event that can be caused only by another process in the set.

To illustrate a deadlocked state, consider a system with three CD RW drives. Suppose each of three processes holds one of these CD RW drives. If each process now requests another drive, the three processes will be in a deadlocked state. Each is waiting for the event “CD RW is released”, Which can be caused only by one of the other waiting processes. This example, illustrates a deadlock involving the same resource type.

Deadlock may also involve different resource types. For example, consider a system with one printer and one DVD drive. Suppose that process Pi is holding DVD and process Pj is holding printer. If Pi requests the printer and Pj requests the DVD drive, a deadlock occurs.

(C).Deadlock Characterization:

(C.1)Necessary Conditions

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

- Mutual Exclusion
- Hold and Wait
- No Preemption
- Circular Wait

- **Mutual Exclusion:** At least one resource must be held in a non sharable-mode, that is only one process at a time can use their source. If another process requests that resource, the requesting process must be delayed until there source has been released.

- **Hold and Wait:** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

- **No Preemption:** Resources cannot be preempted, that is a resource can be released only voluntarily by the process holding it, after that process has completed the task.

- **Circular Wait:** A Set {p1,p2,…..Pn} of waiting processes must exist such that p1 is waiting for a resource held by p2. p2 is waiting for a resource held by p3 and so on.

All four conditions must hold for deadlock to occur.

The circular wait condition implies the hold and wait condition, so the four conditions are not completely independent.

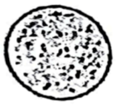
(C.2)Resource Allocation Graph:

Deadlock can be described more preciously in terms of a directed graph called a system resource allocation graph. This graph consists of a set of vertices V and a set of Edges E. These to vertices V portioned into two different types of nodes:

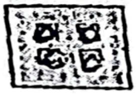
P= {P1,P2,,Pn},the set consisting of all active processes in the system and R={R1,R2,Rm},the set consisting of all resource types in the system.

A directed edge form process Pi to resource type Rj is denoted by Pi->Rj,it signifies that process Pi has requested an instance of resource type Rj and is currently waiting for that resource. A directed edge from Resource type Rj to process Poi is denoted by Rj->Pi; it signifies that an instance of resource type Rj is allocated to process Pi. A directed edge from Pi->Rj is called request edge. A directed edge from Rj->Pi is called assignment edge. Pictorially, were present each process Pi as a circle and each resource type Rj as a rectangle. Since resource type Rj may have more than one instance, we represent each such instance as a dot within the rectangle. Mute that request edge points to only the rectangle Rj, where as an assignment edge must also design at one of the dots in the rectangle

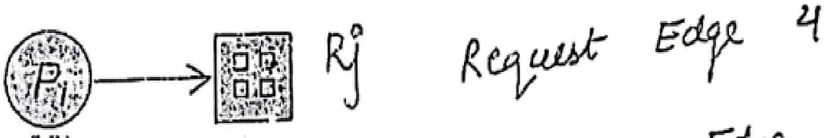
1. Process



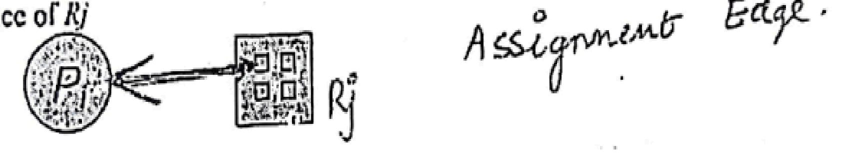
2. Resource Type with 4 instances



3. *Pi* requests instance of *Rj*



4. *Pi* is holding an instance of *Rj*



When process Pi request an instance of resource type Rj, are request edge is inserted in the resource allocation graph. When this request can be fulfilled, the request edge is instantaneously transformed to an assignment edge. When the process no longer needs access to their source, it release the resource, as a result, the assignment edge is deleted.

There source allocation graph shown in Fig.1 depicts the following situation.

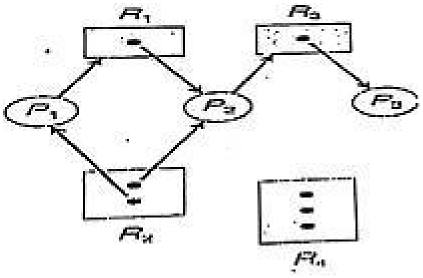


Figure I: Resource Allocation Graph

The Set P, R and E.

- P{P1,P2,P3}
- R{R1,R2,R3,R4}
- E{P1>R1,P2>R3,R1>P2,R2>P2,R2>P1,R3>P3}

Resources Instances

- One instances of Resource type R1.
- Two instances of Resource type R2.
- One instances of Resource type R3.
- Three instances of Resource type.

(C.3)Process States:

Process P1 is holding on instance of resource type R2 and is waiting for an instance of resource type R1.

Process P2 is holding on instance of resource type R2 and is waiting for an instance of resource type R3. Process P3 is holding one instance of resource type R3.

Given the definition of resource allocation graph, it can be shown that graph contains no cycling then no process in the system is deadlocked. If the graph does contain a cycle, then deadlock occur.

If each resource type has exactly one instance, then a cycle implies that a deadlock has occurred. If the cycle involves only a set of resource types, each of which has only a single instance, then a deadlock has occurred. Each process involved in the cycle is deadlocked. In this case, a cycle in the pump his both a necessary and a sufficient condition for the existence of deadlock.

If each resource type has several instances, then a cycle does not necessary imply that a deadlock has occurred. In this case, a cycle in the graph is a necessary but not a sufficient condition for the existence of deadlock.

To illustrate this concept, were turn to the resource allocation graph depicted in fig.2. Suppose that process P3 requests an instance of resource type R2. Since no resource instance is currently available, a request edge P3->R2 is added to the graph (fig.1). At this point, two minimal cycles exist in the system.

P1->R1->P2->R3->P3->R2->P1 P2->R3->P3->R2->P2

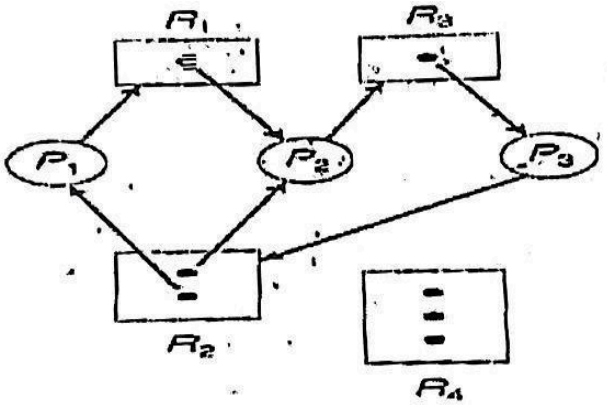


Figure 2: Resource allocation graph with a deadlock

Process P1, P2 and P3 are deadlocked. Process P2 is waiting for the resource R3, which is held by process P3. Process P3 is waiting for either process P1 or process P2 to release resource R2. In Now consider the resource allocation graph in Figure 3.

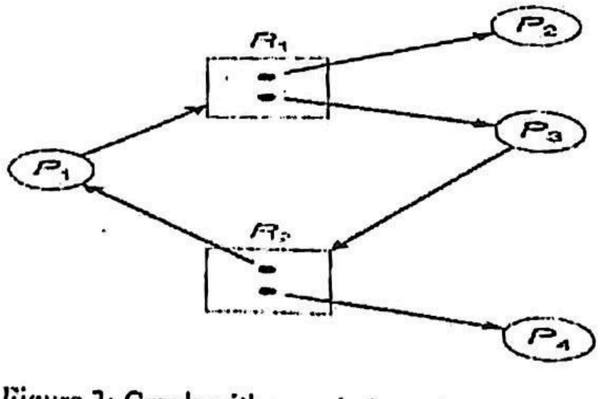


Figure 3: Graph with a cycle but with no deadlock

In this example, we also have a cycle

P1->R1->P3->R2->P1

However, there is no deadlock. Observe that process P4 may release its instance of resource type R2. That resource can then be allocated to P3, breaking the cycle.

In summary, if a resource allocation graph does not have a cycle, then the system is not in a deadlocked state. If there is a cycle, then the system may or may not be in a deadlocked state.

If graph contains no cycles=>no deadlock

If graph contains a cycle=>

If only one instance per resource type, then deadlock

If several instances per resource type, possibility of deadlock.

(D).Methods for Handling Deadlock:

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

Deadlock Prevention provides a set of methods for ensuring that at least one of the necessary conditions cannot hold. These methods prevent deadlocks by constraint show requests for resources can be made.

Deadlock Avoidance requires that the operating system be given in advance additional information concerning which resources a process will request and use during its life time. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process.

Deadlock Detection: If a system does not employ either deadlock prevention or deadlock avoidance algorithm, than a deadlock situation may arise. In this environment the system can provide an algorithm that examines the state of the system to determine whether a deadlock has occurred an algorithm to recover from the deadlock.

(E).Deadlock Prevention:

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

(E.1)Mutual Exclusion

Mutual exclusion is not a problem for sharable resources an example is a “read-only” file which is a resource that can be accessed simultaneously.

Problem: Some resources are inherently not sharable so, denying the mutual exclusion condition cannot be enforced in general.

(E.2)Hold and Wait

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

One protocol that can be used requires each process to request and be allocated all its resources before it begins execution.

An alternative protocol allows a process to request resources only, when the process has none. A process, that copies data from a tape drive to a disk; file sorts the disk; file and then prints the result to a printer. If all resources must be requested at the beginning of the process, then the process must initially request the tape drive, disk file and the printer. It will hold the printer for its entire execution, even though it needs the printer only at the end.

There are two main disadvantage to these protocols.

- Resource Utilization may below
- Starvation is Possible

(E.3)No Preemption

To ensure that this condition does not hold, we can use the following protocol. If a process that is holding some resources requests another resource that cannot be immediately allocated to it. Then all resources currently being held are preempted.

The preempted resource are added to the list of resources for which the process is waiting. The process will be restarted only when it can regain it sold resources as well as the new ones that it is requesting.

(E.4)Circular Wait

Example:

- Processes need three resources memory, disk and printer.

Consider two cases:

- **Case1:**Processes pick own order in which to ask for resources.
- **Case2:** Each process asks first for memory then disk, then the printer. Which of these cases can result in deadlock?
- **Approach:** impose and order on resource acquisition
  All the N types of resources in the system are linearly ordered
- Each is given a number called rank, In the range1,2………,N
- The resources of the same type all have the same rank
- Different types of resources get distinct ranks
- Processes are required to sequence their resource requests in strictly increasing order of rank i.e. they asks for all the “smaller” rank resources first.

(F).Deadlock Avoidance

The simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need. Given this a prior info, it is possible to construct an algorithm that ensures that the system will never enter a deadlocked state. Such an algorithm defines the deadlock avoidance state to ensure that a circular wait condition can never exist. The resource allocation state is defined by the number of available and allocated resources and the maximum elements of the processes. In the following sections, we explore two deadlock avoidance algorithms.

(F.1) Safe State

A state is safe if the system can allocate resources to each process in some order and still avoid a deadlock, More formally, a system is in a sage state only if there exists a safe sequence. A sequence of processes <P,P2…..,Pn> is safe sequence for the current allocation state if, for each Pi the resource requests that Pi can still make can be satisfied by the currently available resources plus the resources held by all Pj, with j<i.

In this situation, if the resources that Pi needs are not immediately available, then Pi can wait until all Pj have finished. Ehen they have finished, Pi can obtain all of its needed resources, complete its designed task, return its allocated resources, and terminate.

When Pi terminates, Pj can obtain its needed resources, and so on. If no such sequence exists, then the system state is said to be unsafe.

A safe state is not a deadlocked state, Conversely a deadlocked state is an unsafe state. Not all unsafe states are deadlocked. However,(Figure 4). An unsafe state may lead to a deadlock's long as the state is safe, the operating system can avoid unsafe states. In an unsafe state, the operating system cannot prevent processes form requesting resources in such a way that a deadlock occurs.

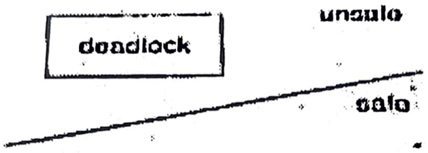


Figure 4: Safe, Unsafe, Deadlock State

(F) Resource Allocation Graph Algorithm

In addition to request and assignment edges, we introduce a new type of edge, called a claim edge. A claim edge Pi->Rj indicates that process Pi may request resource Rj sometime in the future. The edge resembles a request edge in direction but is represented in the graph by a dashed line. When process Pi requests resource Rj the claim edge Pi->Rj is converted to a request edge. Similarly, when a resource Rj is released by Pi, the assignment edge Rj->Pi is reconverted to a claim edge.

Now suppose that process Pi requests resource Rj. The request can be granted only if converting the request edge Pi->Rj to an assignment Rj->Pi edge does not result in the formation of a cycle in the resource allocation graph. We check for safety by using a cycled detection algorithm. An algorithm for detecting a cycle in this graph requires an order on n2 operation, where n is the number of processes in the system,
If no cycle exists, then the allocation of the resources will leave the system in safe state. If a cycle is found, then the allocation will put the system in an unsafe state. In that case, process Pi will have to wait for its requests to be satisfied.

Banker’s Algorithm:

The resource allocation graph (RAG) algorithm is not applicable to a resource allocation system with multiple instances of each resource type.
This algorithm is named as Banker’s algorithm, the name was chosen because the algorithm could be used in a banking system to ensure that the bank never allocated its available cash in such a way that it could no longer satisfy the needs of its customers.
When a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This number may not exceed the total number of resources

in the system. When a user requests a set of resources, the system must determine whether the allocation of these resources will leave. The system in a safe state. If it will, the resources are allocated, otherwise the process must wait until some other process releases enough resources.

Data Structure Used-

It uses the following data structures-

1. Resources R=(R1,R2…..Rm)

Total amount of each resource in the system.

2. Available V=(V1,V2…..Vm)

Total amount of each resource not allocated to any process

3.

$$\mapsto \text{Claim} \quad C = \begin{bmatrix} C_{11} & - - - & C_{1m} \\ C_{21} & & C_{2m} \\ \vdots & & \vdots \\ C_{n1} & & C_{nm} \end{bmatrix}$$

Cij= requirement of process i for Resource j

4. **Aij= current allocation to process i of Resource j**

The matrix claim gives the maximum requirement of each process, with one row dedicated to each process. This information must be declared in advance by a process for deadlock avoidance to work.

The matrix allocation gives the current allocation to each process.

Safety Algorithm-

This algorithm is used to check whether or not a system is in a safe state.

This algorithm can be described as follows.

1. Let work and finish be vectors of length m and n respectively. Initialize
   Work = available and
   Finish[i]=false for i=0,1,2…..n-1
2. Find an index j such that both
   a) Finish [i] false
   b) Need i<= Work

   If no such I exists go to step 4.

3. Work= Work + Allocation

   Finish[i]=true

   Go to step2.
4. If Finish[i] =true for all i, then the system is in safe state.

Resource Request Algorithm-

Next, we describe the algorithm for determining whether requests can be safely granted.

Let Request I be the request vector for process Pi. If Request i[j]=k then process Pi wants k instances of resource type Rj. When a request for resources is made by process Pi, the following actions are taken.

1. If request$_i$ <=Need$_i$ go to step 2 **o**therwise, raise an error condition since the process has executed its maximum claim.
2. If Request$_i$ <= Available$_i$ go to step 3 otherwise Pi must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows.

   Available= Available-Request$_i$

   Allocation$_i$= Allocation$_i$ + Request$_i$

   Need$_i$= Need$_i$-Request$_i$

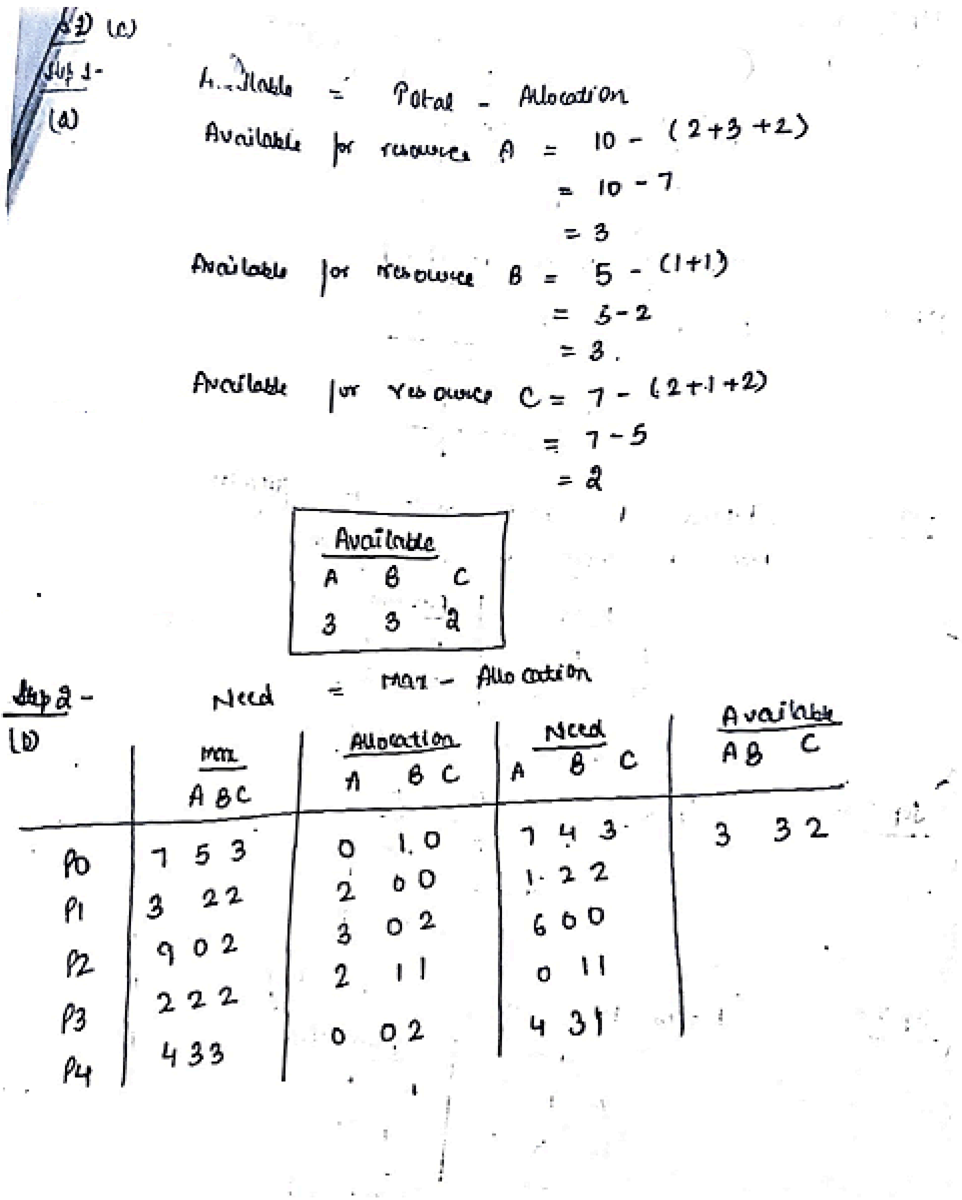
   If the resulting resource allocation state is safe, the transaction is completed, and process Pi is allocated its resources However, if the new state is unsafe then Pi must wait for Request$_i$, and the old resources allocation state is restored.

Ques) Consider the following snapshot of the system, that contains 5 processes and 3 resource type A, B and C having 10, 5 and 7 instances respectively.

a) Is the system in safe state?

b) What will happen if P1 request resource (1,0,0). Can it be granted immediately?

c) Draw RAG.



Q2) (c)

Step 1-

(a)

Available = Total - Allocation

Available for resource A = 10 - (2+3+2)
= 10 - 7
= 3

Available for resource B = 5 - (1+1)
= 5-2
= 3.

Available for resource C = 7 - (2+1+2)
= 7-5
= 2

Available		
A	B	C
3	3	2

Step 2-

(b)

Need = Max - Allocation

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

Step 3: Process P0 need cannot be fulfilled

Process P1 need can be fulfilled.

can be fulfilled.

Need			Available		
A	B	C	A	B	C
1	2	2	3	3	2

So, P1 can be Executed

Step 4-

$$\text{Available} = \begin{array}{rrr} A & B & C \\ \hline 3 & 3 & 2 \\ +\,2 & 0 & 0 \\ \hline 5 & 3 & 2 \end{array}$$

Step 5-

Process P2 need cannot be fulfilled

Process P3 need can be fulfilled.

Need			Available		
A	B	C	A	B	C
0	1	1	5	3	2

So, P3 can be executed

Step 6 -

$$\text{Available} = \begin{array}{rrr} A & B & C \\ \hline 5 & 3 & 2 \\ +\,2 & 1 & 1 \\ \hline 7 & 4 & 3 \end{array}$$

Step 7 -

Process P4 need can be fulfilled.

Need			Available		
A	B	C	A	B	C
4	3	1	7	4	1

Step 8-

So, P4 can be executed.

$$\text{Available} = \begin{array}{rrr} A & B & C \\ \hline 7 & 4 & 1 \\ +\,0 & 0 & 2 \\ \hline \end{array}$$

Step 9 - [illegible]

Need			Available		
A	B	C	A	B	C
7	4	3	7	4	3

So, P0 can be executed.

Step 10 - Available =

$$\begin{array}{rccc} & A & B & C \\ & 7 & 4 & 3 \\ + & 0 & 1 & 0 \\ \hline & 7 & 5 & 3 \end{array}$$

Step 11 - Process P2 need can be fulfilled.

Need			Available		
A	B	C	A	B	C
6	0	0	7	5	3

So, P2 can be executed.

Step 12 - Thus, the safe sequence is

< P1, P3, P4, P0, P2 >

(c) Hence the system is safe or is not in a deadlock.

(d)

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

The new System State is Safe.
Safe Sequence – < P1, P3, P4, P0, P2 >
So, the request can be granted immediately.

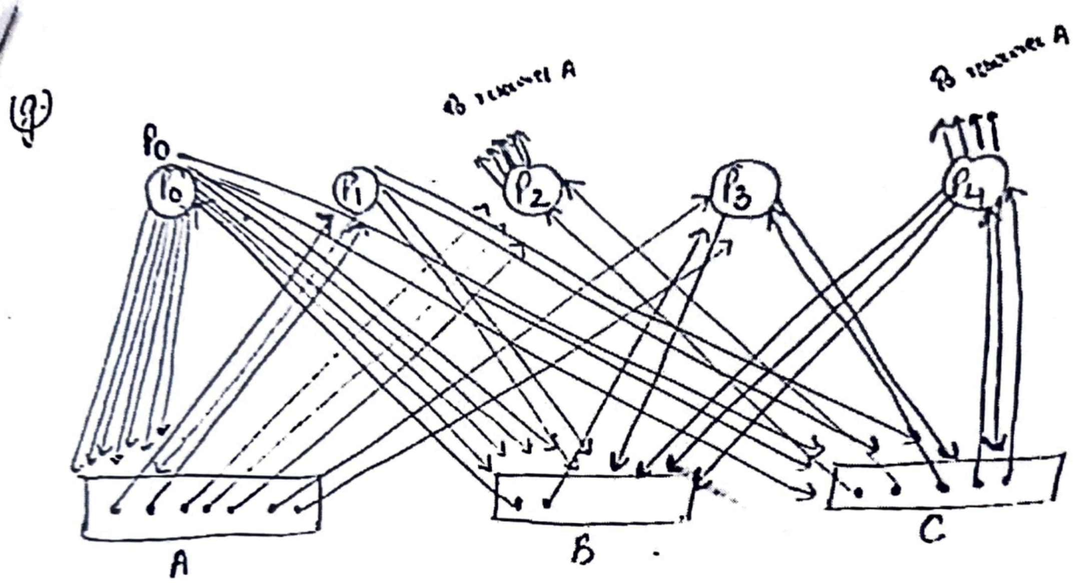
[Note –
show th
in all
points
in pard

(e)

	Allocation A B C	Need A B C	Available A B C
P0	0 1 0	7 4 3	0 0 2 ~~+2 1 1~~ ~~2 1 3~~ ~~3 3 2~~ ~~5 4 5~~ ~~2 0 0~~
P1	2 0 0	1 2 2	
P2	3 0 2	6 0 0	
P3	2 1 1	0 1 1	
P4	3 3 2	1 0 1	

< ~~P1 P3, P4, P1~~

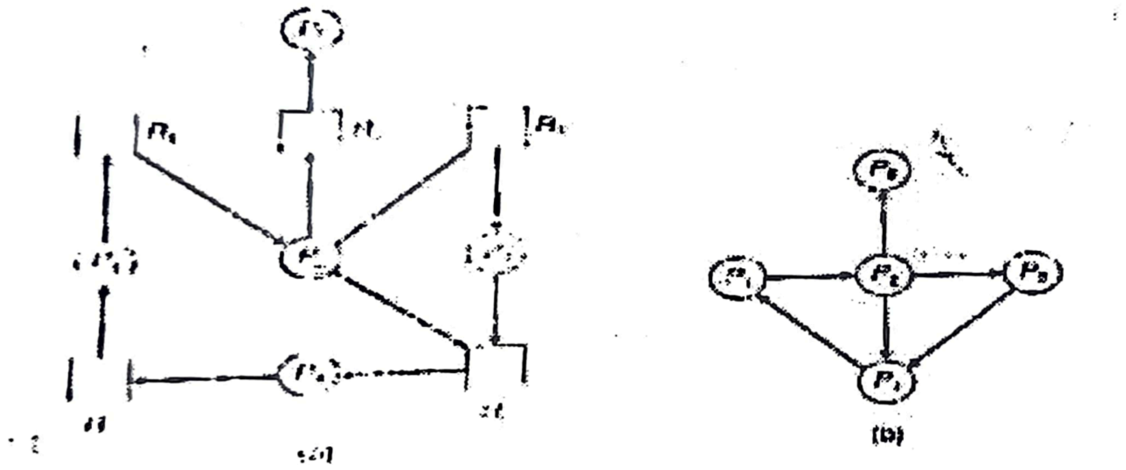
State is Unsafe.
No Safe Sequence
Request cannot be granted Immediately.



(G) Deadlock Detection-

If a system does not employ either deadlock prevention or avoidance algorithm, then a deadlock situation may occur. In this environment the system may provide.
An algorithm that examines the state of the system to determine whether a deadlock has occurred.
An algorithm to recover from the deadlock.

If all resources have only a single instance, then we can define a deadlock detection algorithm that uses a variant of the resource allocation graph, called wait for graph. We obtain this graph from the resource allocation graph by removing the resource nodes and collapsing the appropriate edges.
An edge from Pi to Pj in a wait for graph implies that process Pi is waiting for Process Pj to release a resource that Pi needs.
An edge Pi->Pj exists in a wait for graph if and only if the corresponding resource allocation graph containing two edges Pi->Rq and Rq->Pj for some resource Rq. For example in fig 7 we present n resource allocation graph and its corresponding wait for graph.



Resource Allocation Graph

Wait For Graph

Figure 7

A deadlock exists in the system if and only if the wait for graph contains a cycle. To detect Deadlock, The system needs to maintain the wait for graph and periodically invoke an algorithm that searches for a cycle in the graph. An algorithm to detect a cycle in a graph requires an order of n_2 operation, where it is the number of vertices in the graph.

Recovery from Deadlock

1. For Process: Process Termination

To eliminate the deadlock, we can simply kill one or more processes. For this, we use two methods:

a. **Abort all the Deadlocked Processes**: Aborting all the processes will certainly break the deadlock but at a great expense. The deadlocked processes may have been computed for a long time, and the result of those partial computations must be discarded and there is a probability of recalculating them later.

b. **Abort one process at a time until the deadlock is eliminated**: Abort one deadlocked process at a time, until the deadlock cycle is eliminated from the system. Due to this method, there may be considerable overhead, because, after aborting each process, we have to run a deadlock detection algorithm to check whether any processes are still deadlocked.

2. Resource Preemption

To eliminate deadlocks using resource preemption, we preempt some resources from processes and give those resources to other processes. This method will raise three issues –

a. **Selecting a victim**: We must determine which resources and which processes are to be preempted and also in order to minimize the cost.
b. **Rollback**: We must determine what should be done with the process from which resources are preempted. One simple idea is total rollback. That means aborting the process and restarting it.
c. **Starvation**: In a system, it may happen that the same process is always picked as a victim. As a result, that process will never complete its designated task. This situation is called **Starvation** and must be avoided.

AKTU QUESTIONS
UNIT-3

<table>
<tr><td>1</td><td>Explain threads</td><td>AKTU 2016-17</td><td>2MM</td></tr>
<tr><td>2</td><td>What do you understand by Process? Explain various states of process with suitable diagram. Explain process control block.</td><td>AKTU 2016-17</td><td>5MM</td></tr>
<tr><td>3</td><td>What is a deadlock? Discuss the necessary conditions for deadlock with examples</td><td>AKTU 2016-17</td><td>15MM</td></tr>
<tr><td>4</td><td>Describe Banker’s algorithm for safe allocation.</td><td>AKTU 2016-17</td><td>15MM</td></tr>
<tr><td>5</td><td>What are the various scheduling criteria for CPU scheduling</td><td>AKTU 2017-18</td><td>2MM</td></tr>
<tr><td>6</td><td>What is the use of inter process communication and context switching</td><td>AKTU 2017-18</td><td>2MM</td></tr>
<tr><td>7</td><td>Discuss the usage of wait-for graph method</td><td>AKTU 2017-18</td><td>2MM</td></tr>
<tr><td>8</td><td>Consider the following snapshot of a system:
<table>
<tr><td></td><td colspan="3">Allocated</td><td colspan="3">Maximum</td><td colspan="3">Available</td></tr>
<tr><td>Process</td><td>R1</td><td>R2</td><td>R3</td><td>R1</td><td>R2</td><td>R3</td><td>R1</td><td>R2</td><td>R3</td></tr>
<tr><td>P1</td><td>2</td><td>2</td><td>3</td><td>3</td><td>6</td><td>8</td><td>7</td><td>7</td><td>10</td></tr>
<tr><td>P2</td><td>2</td><td>0</td><td>3</td><td>4</td><td>3</td><td>3</td><td></td><td></td><td></td></tr>
<tr><td>P3</td><td>1</td><td>2</td><td>4</td><td>3</td><td>4</td><td>4</td><td></td><td></td><td></td></tr>
</table>
</td><td>AKTU 2017-18</td><td>7MM</td></tr>
</table>

<table>
<tr><td></td><td>Answer the following questions using the banker's algorithm:
1) What is the content of the matrix need?
2) Is the system in a safe state?</td><td></td><td></td></tr>
<tr><td>9</td><td>Is it possible to have a deadlock involving only a single process? Explain</td><td>AKTU 2017-18</td><td>7MM</td></tr>
<tr><td>10</td><td>Describe the typical elements of the process control block.</td><td>AKTU 2018-19</td><td>2MM</td></tr>
<tr><td>11</td><td>What are the various scheduling criteria for CPU scheduling?</td><td>AKTU 2018-19</td><td>2MM</td></tr>
<tr><td>12</td><td>What is the safe state and an unsafe state ?</td><td>AKTU 2018-19</td><td>2MM</td></tr>
<tr><td>13</td><td>Define Process. Explain various steps involved in change of a process state with neat transition diagram.</td><td>AKTU 2018-19</td><td>7MM</td></tr>
<tr><td>14</td><td>Consider the following process:
<table>
<tr><th>Process</th><th>Arrival Time</th><th>Burst Time</th></tr>
<tr><td>P1</td><td>0</td><td>8</td></tr>
<tr><td>P2</td><td>1</td><td>4</td></tr>
<tr><td>P3</td><td>2</td><td>9</td></tr>
<tr><td>P4</td><td>3</td><td>5</td></tr>
</table>
What is the average waiting and turn around time for these process with:
FCFS Scheduling
.Preemptive SJF Scheduling</td><td></td><td></td></tr>
<tr><td>15</td><td>Consider the following process:
<table>
<tr><th>Process</th><th>Arrival Time</th><th>Burst Time</th></tr>
</table></td><td>AKTU 2017-18</td><td>7MM</td></tr>
</table>

P1	0	8
P2	1	4
P3	2	9
P4	3	5

Draw Gantt chart and find the average waiting time and average turnaround time:

iii. FCFS Scheduling

.SRTF Scheduling

Consider the following process:

Process	Arrival Time	Burst Time	Priority
P1	0	6	3
P2	1	4	1
P3	2	5	2
P4	3	8	4

Draw Gantt chart and find the average waiting time and average turnaround time:

(i) SRTF Scheduling

(ii) Round robin (time quantum:3)

16	What is the need for Process Control Block (PCB)?	AKTU 2015-16	2MM
17	Draw process state transition diagram	AKTU 2015-16	2MM
18	Define the multilevel feedback queues scheduling.	AKTU 2015-16	2MM
19	Discuss the performance criteria for CPU Scheduling.	AKTU 2015-16	2MM

20	Draw and Explain the Process State Transition Diagram?	UPTU 2011-12	
21	List various performance criteria for Scheduling algorithms: 5 process A,B,C,D,E require CPU burst 3,5,2, 5 and 5 units. Their arrival times in the system are 0, 1, 3, 9 and 12. Draw Gantt Chart and compute the average turn around time and average waiting time of these processes for the shortest Job first and Shortest Remaining Time First Scheduling Algorithms.	UPTU 2011-12	
22	Explain the need for process control block.	UPTU 2009-10	
23	Discuss the performance criteria for CPU Scheduling.	UPTU 2009-10	
24	Describe the necessary conditions for deadlock to occur.	UPTU 2009-10	
25	Explain the following Scheduling algorithms- (i) Multilevel feedback Queues Scheduling (ii) FIFO Scheduling	UPTU 2009-10	
26	What do you understand be CPU Scheduling?	UPTU 2008-09	
27	Explain the different conditions of deadlock.	UPTU 2008-09	
28	Draw the state diagram of a process and label various transitions. Explain the need of process suspension.	UPTU 2006-07	
29	Define the following- (i) Dispatch (ii) Dispatch lateral (iii) Scheduling (iv) Swapping (v) Context Switching	UPTU 2006-07	
30	Difference between user thread and kernel thread. What is thread cancellation? Explain its type.	UPTU 2006-07	
31	Explain various thread model with their relative advantages and disadvantages.	UPTU 2006-07	
32	Explain w.r.t. IPC- (i) Synchronous and Asynchronous communication (ii) Need of Buffering and Implementation.	UPTU 2006-07	

<table>
<tr>
<td>33</td>
<td>Consider the set of processes and following Scheduling algo-
(i) FCFS (ii) RR (quantum =2) (iii) RR (Q=1)
If there is a tie within the processes, the tie is broken in favor of the oldest process.
(i) If the Scheduler takes 0.2 unit of CPU time in content Switching for the completed job and 0.1 unit of additional CPU time for incomplete jobs for saving their context. Calculate the percentage of CPU time wasted in each case.
<table>
<tr><th>Process</th><th>Arrival Time</th><th>Execution Time</th></tr>
<tr><td>A</td><td>0</td><td>4</td></tr>
<tr><td>B</td><td>2</td><td>7</td></tr>
<tr><td>C</td><td>3</td><td>3</td></tr>
<tr><td>D</td><td>3.5</td><td>3</td></tr>
</table>
</td>
<td>UPTU 2006-07</td>
<td></td>
</tr>
</table>

EXTRA QUESTIONS

Ques.1 For the process 1, draw a chart illustrating their execution using-
(a). FCFS
(b). SJF
(c). Shortest Remaining Time
(d). Round Robin (Quantum =2)
(e). Round Robin (Quantum =1)

	AT	
Process	0.00	**ET**
A	0	3
B	1.00	6
C	1	4
D	4.00	2
	1	
	6.00	
	1	

Ques.2 For the process 1, what is the average turn around time (rounding to the nearest hundred) using all the above parts.

Ques.3 What is the wait time for all the above process?
Ques.4 For the process 1, draw a chart illustrating their execution using priority scheduling. A large priority number has higher priority.
a. Preemptive
a. Non- preemptive
Process

Process	**Arrival Time**	**Burst**	**Priority**
A	0.0000	4	3
B	1.0001	3	4
C	2.0001	3	6
D	3.0001	5	5

Ques.5 For the above process 1, What is the turn around time for each process?
a. Preemptive
a. Non-Preemptive

Ques.6 For the above Process 1, What is the average throughput?
a. Preemptive
a. Non-Preemptive

Ques.7 For the process 2, draw a chart illustrating their executing using:
a. FCFS
a. SJF
b. Shortest Remaining Time
c. Round Robbin (Quantum =2)
d. Round Robbin (Quantum =1)

Process 2

Process	Arrival Time	Processing Time
A	0.000	4
B	2.001	7
C	3.001	2
D	3.002	2

Ques.8 For the above process, what is the turn around time and average waiting time of each process.
a. FCFS
a. SJF
b. Shortest Remaining Time
c. Round Robbin (Quantum =2)
d. Round Robbin (Quantum =1)

Ques.9 For the process 3, draw a chart illustrating their execution using priority Scheduling A larger priority number has higher priority. What is the average waiting time?
a. Preemptive
a. Non-Preemptive

Process 3

Proces s	Arrival Time	Burs t	Priorit y
A	0.0000	5	4
B	2.0001	4	2
C	2.0001	2	6
D	4.0001	4	3

Ques.10 (a) Describe the Resource allocation Graph.
(b) Draw the reduced resource allocation Graph.
(c) Is the system deadlocked?

Resource Usage

Process	Current Allocation			Outstanding Requests			Resources Allocation		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	2	0	0	1	1	0	0	0	0
P2	3	1	0	0	0	0			
P3	1	3	0	0	0	1			
P4	0	1	1	0	1	0			