Chapter 3

Deadlock and CPU Scheduling

LEARNING OBJECTIVES

- *Deadlock*
- *System model*
- *Bridge crossing*
- *Resources*
- *Resource allocation graph*
- *Methods of handling deadlocks*
- *Deadlock prevention*
- *Resource allocation denial (OR) banker's algorithm*
- *Deadlock detection*
- *Dining philosophers problem*
- *Scheduling algorithms*
- *Scheduling policies*
- *Round Robin scheduling*
- *Shortest remaining time*
- *Highest response ratio next*
- *Multilevel feedback queue scheduling*

Deadlock

It is a situation where a process or set of processes is blocked, waiting for some resource that is held by other waiting processes.

System Model

Let the resource types be R_1, R_2 ... R_m (like CPU cycles, memory space, input/output (I/O) devices, etc.). Each resource type R_i has W_i instances; each process utilizes a resource as follows:

Request A process, needing a resource, will request the operating system (OS) for assignment of the needed resource. Then the process waits, till operating system assigns it an instance of the requested resource.

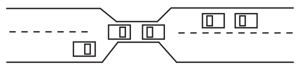
Assignment The OS will assign to the requesting process an instance of the requested resource, whenever, it is available. Then, the process comes out of its waiting state.

Use The process will use the assigned resource. In case, the resource is non-sharable, the process will have exclusive access to it.

Release After the process finished with the use assigned resource, it will return the resource to the system pool. The released resource can now be assigned to another waiting process.

Example:

Bridge crossing



Traffic is allowed only in one direction. Each section of a bridge can be viewed as a resource.

If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback). Several cars may have to be backed up if a deadlock occurs.

Problem of starvation (infinite wait) is possible.

Resources

Types of resources:

1. Reusable resources
2. Consumable resources

Reusable resources These resources can be safely used by only one process at a time, and are not depleted by that use.

Examples: Processors, I/O channels, main and secondary memory, devices and files, etc.

Consider two processes P and Q that compete for exclusive access to a disk file D and tape drive T. Let their implementation is as shown below:

Table 1 *Process P*

Step	Action
P_0	Request (D)
P_1	Lock (D)
P_2	Request (T)
P_3	Lock (T)
P_4	Perform function
P_5	Unlock (D)
P_6	Unlock (T)

Table 2 *Process Q*

Step	Action
Q_0	Request (T)
Q_1	Lock (T)
Q_2	Request (D)
Q_3	Lock (D)
Q_4	Perform Function
Q_5	Unlock (T)
Q_6	Unlock (D)

P and *Q* are executing on a single processor in interleaved fashion. Then deadlock occurs if each process holds one resource and requests the other.

For example, deadlock occurs if the multiprogramming system interleaves the execution of the two processes as follows:

$$P_0, P_1, Q_0, Q_1, P_2, Q_2$$

One strategy to deal with this type of deadlocks is to impose system design constraints concerning the order in which resources can be requested.

Consumable resources A consumable resource is one that can be created and destroyed. There is no limit on the number of consumable resources of a particular type.

Examples: Interrupts, signals, messages, etc.
Consider the following pair of processes, in which each process attempts to receive a message from the other process and then send a message to the other process:

P_1	P_2
. . . .	. . . .
receive (P_2);	receive (P_1);
. . . .	. . . .
send (P_2, M_1);	send (P_1, M_2);

Deadlock occurs in above case, if the receive is blocking. There is no single effective strategy that can deal with all types of deadlocks.

Deadlock Characteristics

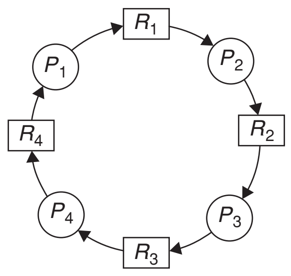
Deadlock is an undesirable state of the system. The following are the four conditions that must hold simultaneously for a deadlock to occur:

Mutual exclusion A resource can be used by only one process at a time. If another process requests for that resource then the requesting process must be delayed until the resource has been released.

Hold-and-wait Some processes must be holding some resources in a non-sharable mode and at the same time must be waiting to acquire some more resources, which are currently held by other processes in a non-sharable mode.

No pre-emption Resources granted to a process can be released back to the system only as a result of the voluntary action of that process, after the process has completed its task.

Circular wait Deadlocked processes are involved in a circular chain such that each process holds one or more resources being requested by the next process in the chain.



Resource Allocation Graph

A deadlock is described in terms of a directed graph called a system Resource Allocation Graph (RAG). It consists of two sets:

1. The set of vertices, *V*
2. The set of edges, *E*

The set of vertices is again divided into two categories.

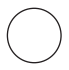
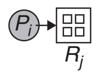
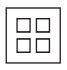
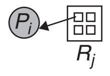
The set of all active processes in the system is $P = \{P_1, P_2, \dots P_n\}$ and the set of all different type of resources i.e., $R = \{R_1, R_2 \dots R_m\}$

There are two types of edges in the RAG:

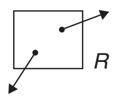
1. A directed edge from the process P_i to resources type R_j and is denoted by $P_i \rightarrow R_j$. It signifies that the *i*th process is requesting one unit of the resource type *j*. This edge is request edge.
2. A directed edge from the resource R_i to process P_j denoted by $R_i \rightarrow P_j$. It signifies that one unit of *i*th resource is held by the process *j*. This edge is also called as an *allocation edge/assignment edge*.

Notations Used in RAGs

We denote a process by a circle and each resource by a rectangle.

Process		P_i requests instance of R_j	
Resource Type with four instances		P_i is holding an instance of R_j	

However, if we have more number of instances of a resource type, then it is denoted by more dots.



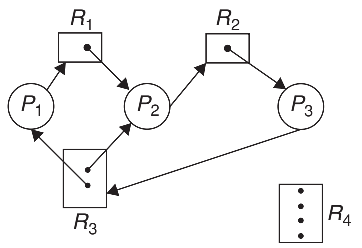


Figure 1 RAG with deadlock.

Notes:

1. If a cycle exists in the RAG, there may or may not be a deadlock.
2. Acyclic RAG implies no deadlock.
3. No deadlock implies acyclic RAG. This means that cycles can be there even if there is no deadlock.

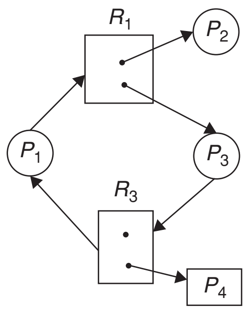


Figure 2 RAG with no deadlock but contains cycle.

Notes:

1. If graph contains no cycles, then no deadlock.
2. If graph contains a cycle

If there is only one instance per resource type, then deadlock occurs.

If there are several instances per resource type, deadlock may occur.

Methods of Handling Deadlocks

There are three approaches to deal with deadlocks. They are

1. Deadlock prevention
2. Deadlock avoidance
3. Deadlock detection

Deadlock Prevention

1. The strategy of deadlock prevention is to design a system in such a way that the possibility of deadlock is excluded.
2. Two classes of deadlock prevention are
   - Indirect method
   - Direct method

Indirect method Prevent the occurrence of one of three necessary conditions of deadlock i.e., mutual exclusion, No pre-emption and hold and wait.

Direct method Prevent the occurrence of circular wait.

Prevention Techniques

Mutual exclusion This is supported by the OS.

Hold and wait

1. This condition can be prevented by requiring that a process request all of its required resources at one time and blocking the process until all requests can be granted simultaneously.
2. But this prevention does not yield good results because
   - Long waiting time required
   - Not efficient use of allocated resources
   - A process may not know all the required resources in advance.

Advantages

1. Works well for processes that perform a single burst of activity.
2. No pre-emption necessary.

No pre-emption Prevention strategies for 'no pre-emption' are

1. 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 released and if necessary request them again together with the additional resources.
2. If a process requests a resource that is currently held by another process, the OS may pre-empt the second process and require it to release its resources. This technique works only when two processes do not have same priority.

Advantages

Convenient when applied to resources whose state can be saved and restored easily.

Disadvantage

Pre-empts more often than necessary.

Circular wait One way to ensure that this condition never holds is to impose a total ordering of all resource types and to require that each process requests resource in an increasing order of enumeration, i.e., if a process has been allocated resources of type *R*, then it may subsequently request only those resources of types following *R* in the ordering.

Advantages

1. Feasible to enforce via compile time checks.
2. No run-time computation required.

Disadvantages

1. Disallows incremental resources requests.

Note: The deadlock prevention strategies are conservative and undercommits resources.

Deadlock Avoidance

1. This approach allows the three necessary conditions of deadlock but makes judicious choices to assure that deadlock point is never reached.
2. Deadlock avoidance allows more concurrency than prevention.
3. A decision is made dynamically whether the current resource allocation request will, if granted, potentially lead to a deadlock.

4. It requires the knowledge of future process requests.
5. Two techniques to avoid deadlock:
   - Process Initiation Denial
   - Resource Allocation Denial

Process initiation denial In this technique, do not start a process if its demands might lead to deadlock.

Consider a system of 'n' processes and 'm' different types of resources. Let us define the following vectors and matrices:

Resources $R = (R_1, R_2, \ldots R_m)$
R_1: amount of type 1 resources
R_2: amount of type 2 resources
Available $= V = (V_1, V_2, \ldots V_m)$

'V' specifies total amount of each resource not allocated to any process.

$$\text{Claim } C = \begin{array}{c} P_1 \\ P_2 \\ P_n \end{array}\begin{bmatrix} C_{11} & C_{12}\cdots & C_{1m} \\ C_{21} & C_{22}\cdots & C_{2m} \\ C_{n1} & C_{n2}\cdots & C_{nm} \end{bmatrix}$$

C_{ij} = requirement of process i for resources j

$$\text{Allocation } A = \begin{array}{c} P_1 \\ P_2 \\ P_n \end{array}\begin{bmatrix} A_{11} & A_{12}\cdots & A_{1m} \\ A_{21} & A_{22}\cdots & A_{2m} \\ A_{n1} & A_{n2}\cdots & A_{nm} \end{bmatrix}$$

A_{ij} = Current allocation to process i of resource j.
The following relationships must hold:

1. All resources are either available or allocated, that is,

$$R_j = V_j + \sum_{i=1}^{n} A_{ij},\ \forall j$$

2. No process can claim more than the total amount of resources in the system, that is,

$$C_{ij} \text{″} R_j,\ \forall i, j$$

3. No process is allocated more resources of any type than the process originally claimed to need, that is,

$$A_{ij} \text{″} C_{ij},\ \forall i, j$$

With these properties satisfied, we can define a deadlock avoidance policy that refuses to start a new process if its resource requirements might lead to deadlock. Start a new process P_{n+1} only if

$$R_j \geq C_{(n+1)j} + \sum_{i=1}^{n} C_{ij},\ \forall j$$

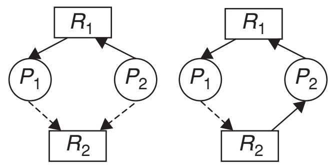
that is, a process is only started if the maximum claim of all current processes plus those of the new process can be met.

Resource allocation denial (OR) banker's algorithm Consider a system with a fixed number of processes and a fixed number of resources. At any time, a process may have zero or more resources allocated to it.

State: The state of a system reflects the current allocation of resources to processes.

Safe state

1. When a process requests an available resource, the system must decide if immediate allocation leaves the system in a safe state.
2. A state is safe if the system can allocate resources to each process in some order and still avoid deadlock.
3. More formally, a system is in safe state if there exists a safe sequence of all processes.
4. A sequence of processes $<P_1, P_2, \ldots, P_n>$ is a safe sequence for the current allocation state, if for each P_i, the resource requests that P_i can still make can be satisfied by the currently available resources plus the resources held by all the P_j, with $j < i$.
5. When P_j is finished, P_i can obtain the needed resources, completed its designated task, return its allocated resources and terminates.
6. When P_i terminates, P_{i+1} can obtain its needed resources and so on.
7. If a system is in safe state, no deadlock occurs.
8. If a system is in unsafe state deadlock may occur.
9. Avoidance ensures that a system will never enter an unsafe state.



10. The dotted line in the above graph represents a claim edge, i.e., a process may request that resource sometime in the future.
11. A request can only be granted if it does not result in the formation of a cycle in the graph.
12. If P_2 request R_2, we cannot allocate it, since this would create a cycle.
13. A cycle indicates the system is in *unsafe state*.

Example 1: Consider the following state of a system consisting of three processes and two resources:

$$R = (R_1, R_2) = (5\ 3)$$
$$V = (V_1, V_2) = (2\ 1)$$

$C =$

	R_1	R_2
P_1	4	3
P_2	2	1
P_3	3	3

$A =$

	R_1	R_2
P_1	2	1
P_2	1	1
P_3	0	0

Is this is a safe state?

Solution:
To check whether the state is safe or not, identify whether any one of the three process can run to completion with the resources available, that is, $C_{ij} - A_{ij}$ ″ V_j, ∀j

$$C - A = \begin{array}{c|cc} & R_1 & R_2 \\ \hline P_1 & 2 & 2 \\ P_2 & 1 & 0 \\ P_3 & 3 & 3 \end{array}$$

We can identify that
[1 0] < [2 1].
$\therefore P_2$ can execute first.
After P_2 execution it will release all its resources then
$V = (2\ 1) + (1\ 1) = (3\ 2)$.
Now P_1 can execute as (2 2) < (3 2).
After that P_1 can release its resources then
$V = (3\ 2) + (2\ 1) = (5\ 3)$
Now P_3 can execute and release the resources after completion.
Hence, the safe sequence is $< P_2, P_1, P_3 >$.

Example 2: Now suppose for the above system the allocation matrix

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 1 \\ 2 & 1 \end{pmatrix} \text{ and } V = (0\,0)$$

Then no process can run to completion as no $C_{ij} - A_{ij}$ ″ V_j, ∀j.
Hence, the system is in unsafe state.

Detection Algorithm for Several Instances of a Resource Type

Safety algorithm

To find out whether or not a system is in a safe state.

Step I: Let ‘work’ and ‘finish’ be the two vectors of length m and n.
Initialize: Work = Available and Finish [i] = false;

Step II: Find i such that both:
(a) Finish [i] = false;
(b) Need ″ Work (Need = claim − Allocation)
If no such i exists, go to step 4.

Step III: Work = Work + Allocation
Finish [i] = true
Go to step 2

Step IV: If Finish [i] = true for all i, then the system is in safe state else it is in unsafe state. This algorithm takes $O(m \rightarrow n^2)$ operations to decide whether a state is safe.

Resource–Request algorithm

Let Request$_i$ be the request vector for process, P_i. If Request$_i[j] = k$, the process P_i wants k instances of resource type R_j. When a request for resources is made by process, P_i, the following actions are taken.

Step I: If Request$_i$ ″ Need$_i$, go to step 2. Else raise an exception (error) as the process has exceeded its maximum claim.

Step II: If Request$_i$ ″ Available, go to step 3. Else P_1 must wait, since the resources are not available.

Step IIII: Have the system pretend to have allocated the requested resources to process, P_i, by modifying the state as follows.

Available = Available − Request$_i$
Allocation = Allocation + Request$_i$
Need$_i$ = Need$_i$ − Request$_i$

If the resulting resource–allocation is safe, then the transaction is completed and process, P_i is allocated its resources. However, if the new state is unsafe then P_i must wait for Requesti and the old resource allocation state is restored.

Advantages of deadlock avoidance technique

1. Not necessary to pre-empt and rollback processes.
2. Less restrictive than deadlock prevention.

Disadvantages

1. Future resource requirement must be known in advance.
2. Processes can be blocked for long periods.
3. Exists fixed number of resources for allocation.

Deadlock Detection

1. This technique does not limit resource access or restrict process action.
2. Requested resources are granted to processes whenever possible.

Deadlock detection is used by employing an algorithm that tracks the circular waiting and killing one or more processes so that the deadlock is removed.

The system state is examined periodically to determine if a set of processes is deadlocked.

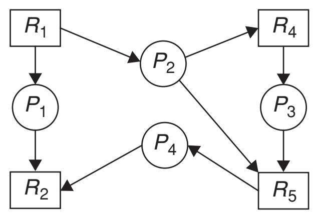
A deadlock is resolved by aborting and restarting a process, relinquishing all the resources that the process held.

For single instance of each resource type If in the RAG, every resource has only one instance (or single instance) then we define a deadlock detection algorithm that uses a variant of the RAG and is called a wait-for-graph.

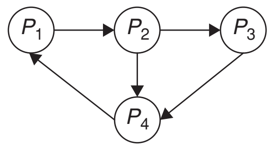
How can we get this graph from RAG? : We can get this by removing the nodes of type resource and collapsing the appropriate edges. Wait-for-graph has a cycle, then there is deadlock in the system.

To detect deadlocks, the system needs to maintain the wait-for-graph and to periodically invoke an algorithm. The complexity of this algorithm is $O(n^2)$ where n is the number of vertices in the graph.

Consider the RAG:



We draw the wait-for-graph by removing all nodes that represents resources and collapsing their edges.



Wait-for-graph

The system is in deadlock state.
Cycle –P_1, P_2, P_4, P_1
Cycle –P_1, P_2, P_3, P_4, P_1

Deadlock Detection Algorithm for Several Instances of Resource Type

Consider the Allocation matrix, A, Request matrix Q (Q_{ij} represents the amount of resource of type j requested by process i), Resource vector R and available vector V.

The algorithm proceeds by marking processes that are not deadlocked. Initially all processes are unmarked.

1. Mark each process that has a row in the Allocation matrix of all zeros.
2. Initialize a temporary vector W to equal the Available vector.
3. Find an index i such that process i is currently unmarked and the ith row of Q is less than or equal to W, that is, Q_{ik} ″ W_k, for 1 ″ k ″ M. If no such row is found, terminate the algorithm.
4. If such a row is found, mark process i and add the corresponding row of the allocation matrix to W. That is, set $W_k = W_k + A_{ik}$ for 1 ″ k ″ M. Return to step 3.

A deadlock exists if and only if there are unmarked processes at the end of the algorithm. Each unmarked process is deadlocked.

Example 3: Let the

Request matrix Q =

	R_1	R_2	R_3
P_1	1	1	1
P_2	1	0	0
P_3	1	1	1
P_4	1	1	1

Allocation matrix A =

	R_1	R_2	R_3
P_1	1	1	0
P_2	1	0	0
P_3	0	1	1
P_4	0	1	1

Resource vector $R = (3\ 3\ 3)$
Available Vector $V = (1\ 0\ 1)$
Is deadlock existing in this system?

Solution:
$W = (1\ 0\ 1)$
The request of P_2 is less than W. So $W = W + (1\ 0\ 0) = (2\ 0\ 1)$
So mark P_2. No other unmarked process has a row Q that is less than or equal to W.
Terminate the algorithm.
$\therefore P_1, P_3, P_4$ are in deadlock.

Advantages

1. Never delays process initiation
2. Facilitates online handling

Disadvantages

1. Inherent pre-emption losses

Deadlock Recovery

The possible deadlock recovery strategies are as follows:

1. Abort all deadlocked processes.
2. Back up each deadlocked process to some previously defined checkpoint and restart all processes.
3. Successively abort deadlocked processes until deadlock no longer exists.
4. Successively pre-empt resources until deadlock no longer exists.

Dining Philosophers Problem

Consider the following solution for dining philosophers problem using semaphores:

```
Semaphore fork[5] = {1};
int i;
void philosopher(int i)
{
while (true)
{
think( );
wait(fork[i]);
wait(fork[(i + 1) mod 5]);
eat( );
signal(fork[(i + 1) mod 5]);
signal(fork[i]);
}
}
void main( )
{
Begin(Philosopher(0),Philosopher(1
),     Philosopher(2),     Philosopher(3),
Philosopher(4));
}
```

Here, each philosopher picks up first the fork on the left and then the fork on the right. After the philosopher is finished eating, the two forks are placed on the table. But this solution leads to deadlock, if all of the philosophers are hungry at the same time, they all sit down, they all pick up the fork on their left and they all reach out for the other fork, which is not there.

A refined solution to dining philosophers problem which is deadlock free is shown below:

```
Semaphore fork[5] = {1};
Semaphore room = {4};

int i;
void philosopher(int i)
{
while(true)
{
think(  );
wait(room);
wait(fork[i]);
wait(fork[(i + 1) mod 5]);
eat( );
signal(fork[(i + 1) mod 5]);
signal(fork[i]);
signal(room);
}
}
void main ( )
{
Begin
{
(Philosopher (0), Philosopher (1), Philosopher
(2), Philosopher (3), Philosopher (4));
}
}
```

This solution is free from deadlock and starvation.

CPU Scheduling

1. The objective of multi programmed OS is to maximize CPU utilization by having some process running at all times.
2. The objective of time shared OS is to switch the CPU among processes so frequently that the users can interact with each program while it is executing.
3. When there are more than one process ready to execute with the processor, a selection decision needs to be made to pick a process for execution from among the ready processes. This activity is called *process scheduling*.

Scheduling queue: It maintains information of all ready processes for CPU devices. It is maintained as a linked list.

Types of Scheduling Queue

1. ***Job queue:*** It consists of all processes in the system.
2. ***Ready queue:*** It consists of all processes that are residing in the main memory and are ready but waiting to execute on CPU.
3. ***Device queue:*** It consists of processes waiting for a particular I/O device. Each device has its own queue.

Process CPU–I/O Burst Cycle

The execution of process consists of CPU burst and I/O burst. The execution of process starts with CPU burst and I/O burst, which are executed alternatively.

The alternating sequence of CPU and I/O burst are shown below:

$$\left.\begin{array}{l}\text{Read } a\\ \text{Inc } a\\ \text{Read } x\end{array}\right\}\text{ CPU Burst}$$

I/O waiting } I/O Burst

$$\left.\begin{array}{l}\text{Dec } x\\ \text{Store } x\end{array}\right\}\text{ CPU Burst}$$

I/O waiting } I/O Burst

.
.
.
.

There should be proper balance between CPU bound process and I/O bound process in a schedule.

Scheduler

A process migrates between various scheduling queues throughout its lifetime. The process of selecting processes from the queues is carried out by scheduler.

Types of Processor Scheduling

There are three types of processor scheduling:

1. Long-term scheduling
2. Medium-term scheduling
3. Short-term scheduling

The following figure relates the scheduling functions to the process state transition diagram:

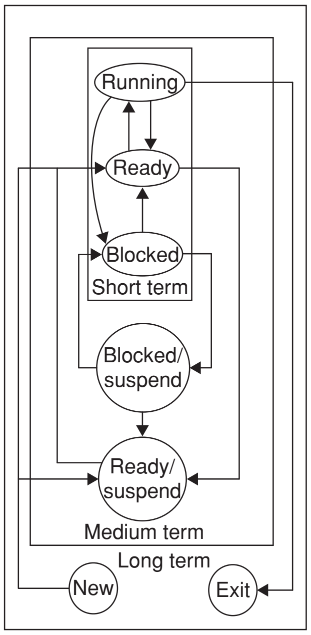


Figure 3 Levels of Scheduling

Long-term Scheduling

1. This is performed when a new process is created. This is also called *job scheduling*.
2. This is a decision whether to add a new process to the set of process that is currently active.
3. It controls the degree of multiprogramming.
4. The long-term scheduler creates processes from the queue when it can.
5. This involves two decisions:
    - The Scheduler must decide when the OS can take on one or more additional processes.
    - The scheduler must decide which job(s) to accept and turn into processes.

Medium-term Scheduling

1. It is a part of swapping function.
2. This is a decision whether to add a process to those that are at least partially in main memory and therefore avail for execution.

Short-term Scheduling

1. It is the decision regarding which ready process to be executed next.
2. This is also known as CPU scheduler.
3. This is invoked whenever an event occurs that may lead to the blocking of the current process or that may provide an opportunity to pre-empt a currently running process in favour of another.
4. Another term involved in short-term scheduling is *dispatcher* which is a module that gives control of the CPU to the process selected by short-term scheduler.

Examples: Clock interrupts, I/O interrupts, OS calls, etc.

Scheduling Algorithms

Scheduling Criteria

The commonly used scheduling criteria can be categorized along two dimensions:

1. User oriented versus system oriented.
2. Performance related versus others.

User-oriented, performance-related criteria

1. ***Turnaround time:*** It is the time taken to execute a process. It is calculated as the interval from the time of submission of a process to the time of completion.
   Turnaround time = waiting time + execution time + time spent in I/O + time spent to get into memory
2. ***Response time:*** Amount of time it takes from when a request was submitted until the first response is produced. It should be minimum.
3. ***Deadlines:*** When process completion deadlines can be specified, the scheduling discipline should subordinate other goals to that of maximizing the percentage of deadlines met.

Waiting time: It is the amount of time that a process spends in ready queue and doing I/O, and it should be minimum.

User-oriented, other criteria

Predictability: A given job should run in about the same amount of time and at about the same cost regardless of the load on the system.

System-oriented, performance-related criteria

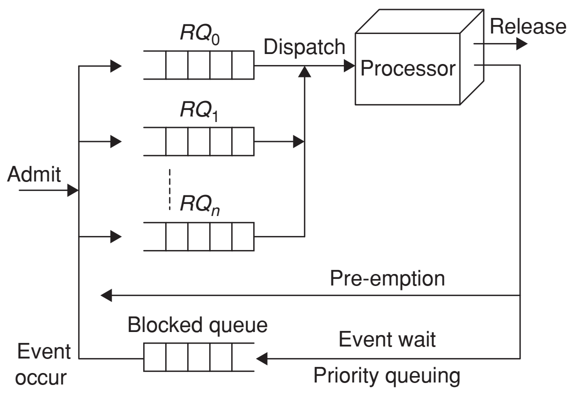
1. ***Throughput:*** The scheduling policy should attempt to maximize the number of processes completed per unit of time. This is a measure of how much work is being performed.
2. ***Processor (CPU) utilization:*** This is the percentage of time that the processor is busy. In real system, it should range from 40–90%.

System-oriented, other criteria

1. ***Fairness:*** No process should suffer from starvation.
2. ***Enforcing priorities:*** Should favour higher priority processes and use Aging technique in order to increase the priority of processes is that wait in the system for long time.
3. ***Balancing resources:*** Should keep the resources of the system busy.

Use of priorities

1. Each process is assigned a priority, and the scheduler will always choose a process of higher priority over one of lower priority.



Here RQ_0, RQ_1,⊃RQ_n are ready queues with priority (RQ_i) > priority (RQ_j) for $i < j$. The scheduler starts with RQ_0 processes, if it is empty choose a process from RQ_1 and so on.

Scheduling Policies

We will discuss the following scheduling algorithms:

1. FCFS
2. Round Robin
3. SPN
4. SRN
5. HRRN
6. Feedback

Before discussing these algorithms, let us discuss some basic concepts:

Selection function

1. Determines which process among ready process, is selected next for execution.
2. This may depend on priority, resource requirement, execution characteristics of process.
3. Execution characteristics of a process include
   w = waiting time
   e = execution time,
   $S = w + e$

Decision mode It is of the following two types:

1. Non-pre-emptive
2. Pre-emptive
   - In non-pre-emptive scheduling, if once the CPU has been allocated to a process, the process can keep the CPU until it releases it, either by terminating or switching to waiting state.
   - In pre-emptive scheduling, CPU can be taken away from a process during execution.

Comparison of non-pre-emptive and pre-emptive scheduling

	Non-pre-emptive Scheduling	Pre-emptive Scheduling
1.	In non-pre-emptive scheduling, if once a process has been allocated CPU then the CPU cannot be taken away from that process.	In pre-emptive scheduling, the CPU can be taken away before the completion of the process.
2.	No preference is given when a higher priority job comes.	It is useful when a higher priority job comes as here the CPU can be snatched from a lower priority process.
3.	The treatment of all processes is fairer.	The treatment of all processes is not fairer as CPU snatching is done either due to constraints or due to higher priority, process request for its execution.
4.	It is a cheaper scheduling method. First come first served is an example.	It is a costlier scheduling method. Round Robin is an example.

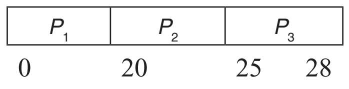
First come, first served scheduling (FCFS) The process that requests the CPU first is allocated the CPU first. It is non-pre-emptive scheduling and average waiting time is quite long.

Example 4: Consider the following processes:

Process	Cpu Burst Time (Millisecond)
P_1	20
P_2	5
P_3	3

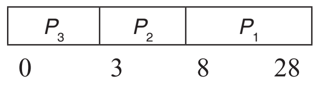
Find the average waiting time.

Solution:
Suppose they are in the order P_1, P_2, P_3 at time 0. So, Gantt chart is



$$\text{Average waiting time } = \frac{(0+20+25)}{3}$$
$$= 15 \text{ ms}$$

If they arrived in order P_3, P_2, P_1 then, Gantt chart is



$$\text{Average waiting time } = \frac{0+3+8}{3} = \frac{11}{3}$$
$$= 3.67 \text{ ms}$$

Notes:

1. Throughput is not that much emphasized.
2. Response time may be high especially if there is a large variance in process execution times.
3. Minimum overhead required.
4. It penalizes short processes, also penalizes I/O bound processes.
5. There is no possibility of starvation.

Advantages

1. Simple and brutally fair.
2. It is suitable for batch systems.

Disadvantages

1. The average waiting time is not minimal.
2. Not suitable for time sharing systems like Unix.
3. ***Convoy effect:*** Short process behind long process results in lower CPU utilization.

Round Robin scheduling

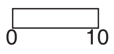
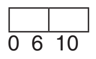
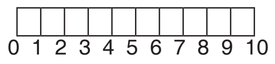
1. It is designed for time sharing system.
2. Similar to FCFS with pre-emption added.
3. Each process gets a small central CPU time (a time slice) usually 10 – 100 ms.
4. After time slice has elapsed and added to the end of the ready queue.
5. The scheduler picks the first process from the ready queue, sets a timer to interrupt after one time quantum and then dispatches the process. One of the following happens.
6. The process may have a CPU burst of less than 1 time quantum. (or)
7. CPU burst time of the currently executing process is longer than one time quantum. In this case, the timer will go off, cause an interrupt, a context switch is then

executed and the process is put at the tail of the ready queue.

8. Average waiting time is quite long.

Performance of Round Robin scheduling

Let us assume that we have onlys one process of 10 time units.

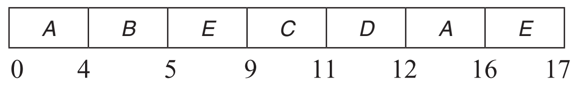
Process time = 10	Quantum	Switch
	12	0
	6	1
	1	9

Example 5: Consider the following processes, arrival times and CPU processing requirements with Round Robin scheduling algorithm.

Process	CPU time	Arrival time
A	8	0
B	1	1
C	2	3
D	1	4
E	5	2

What will be the mean turnaround time if time quantum is 4 msec?

Solution:
Plotting the Gantt chart



Turnaround time = Finish time – Arrival time
TAT of *A* = 16 – 0 =16
$B = 5 - 1 = 4$
$C = 11 - 3 = 8$
$D = 12 - 4 = 8$
$E = 17 - 2 = 15$

$$\text{Mean Turnaround time } = \frac{16+4+8+8+15}{5}$$
$$= 10.2 \text{ msec}$$

1. If there are n processes in the ready queue and time quantum q, then each process gets $\frac{1}{n}$ of the CPU time in chunks of at most q time units at once.
2. No process waits more than $(n - 1)q$ time units until the next time quantum.
3. The performance of Round Robin depends on time slice. If it is larger it is same as FCFS. If q is very small overhead is too high as the number of context switches increases.

Notes:

1. Throughput is low if quantum is too small.
2. Provides good response time for short processes.
3. Minimum overhead.
4. All processes treated fairly.
5. No starvation.

Shortest process next (SPN)

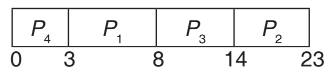
1. This is a non-pre-emptive policy in which the process with the shortest expected processing time is selected next.
2. A short process will jump to the head of the queue past longer jobs.

Example 6: Consider the following process, such that all have arrived at time = 0

Process	Burst time
P_1	5
P_2	9
P_3	6
P_4	3

Find the average waiting time using SPN.

Solution: Gantt chart is



$$\text{Average waiting time } = \frac{0+3+8+14}{4} = 6.25 \text{ ms}$$

Notes:

1. Difficulty with this policy is that we need to know or at least estimate the required processing time of each process.
2. High throughput is possible.
3. Provides good response time for short processes.
4. High overhead.
5. It penalizes long processes.
6. There is a possibility of starvation.

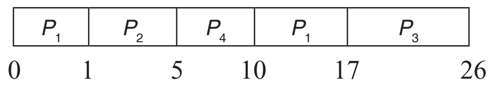
Shortest remaining time (SRT)

1. It is a pre-emptive version of SPN.
2. Here the scheduler always chooses the process that has the shortest expected remaining processing time.
3. When a new process joins the ready queue, it may in fact have a shorter remaining time than the currently running process.
4. Accordingly, the scheduler may pre-empt the current process when a new process becomes ready.

Example 7: Consider the following process. Find the average waiting time for SRT.

Process	Arrival time	Burst time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Solution: Gantt chart



So average waiting time

$$\frac{(10-1)+(1-1)+(17-2)+(5-3)}{4}$$

$$\frac{9+0+15+2}{4}=\frac{26}{4}$$

$$=6.5 \text{ ms}$$

Here P_3 starts at time 17 but the arrival time was at 2. So waiting time of P_3 will be (17 – 2).

Notes:

1. High throughput.
2. Provides good response time.
3. High overhead.
4. Penalizes long processes.
5. Starvation is possible.

Highest Response Ratio Next (HRRN)

This algorithm works on the principle the executes the job first which has the highest response ratio. We define response ratio as the ratio between turnaround time and response time.

$$\text{Response Ratio } = \frac{(W+S)}{S}, \text{ where}$$

W – Time spend waiting for the processor
S – Service time
This response ratio is also named as normalized turnaround time.

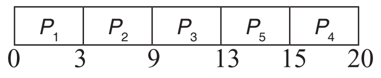
Example 8: Consider 5 processes with their arrival and service times:

Process	Arrival time	Service time
$P1$	0	3
$P2$	2	6
$P3$	4	4
$P4$	6	5
$P5$	8	2

What is the average turnaround time using HRRN technique?

Solution:

Gantt chart



Average turnaround time

$$=\frac{3+(6+1)+(4+5)+(5+9)+(2+5)}{5}=\frac{40}{5}=8 \text{ ms}$$

Notes:

1. It is a non-pre-emptive scheduling algorithm.
2. High throughput.
3. Provides good response time.
4. High overhead.
5. Good balance of any type processes.
6. No starvation.

Multilevel feedback queue scheduling

1. In multilevel queue scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes cannot move between queues.
2. Processes can move between queues. If a process uses too much CPU time, it will be moved to a lower priority queue.
3. I/O bound and interactive processes are put into higher priority queue.
4. 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.

In general, a multilevel feedback queue scheduler is defined by following parameters:

1. The number of queues.
2. The scheduling algorithm for each queue.
3. The method used to determine when to upgrade a process to a higher priority queue.
4. The method used to determine when to denote a process to a lower priority queue.
5. The method used to determine in which queue a process will enter when process needs service.

Notes:

1. Pre-emptive at time quantum.
2. Throughput is not that much emphasized.
3. Response time is not that much emphasized.
4. High overhead.
5. Favours I/O bound processes.
6. Starvation is possible.

Exercises

Practice Problem I

Directions for questions 1 to 19: Select the correct alternative from the given choices.

1. Consider the following processes; find the average waiting time using non-pre-emptive priority scheduling?

Process	Arrival time (ms)	Burst time (ms)	Priority
P_0	0	10	5
P_1	1	6	4
P_2	3	2	2
P_3	5	4	0

(A) 2.36 ms (B) 0.31 ms
(C) 7.75 ms (D) 13.25 ms

2. Consider a set of five processes whose arrival time, CPU times needed are given below.

Process	CPU time (in m sec)	Arrival time (in msec)
P_1	10	5
P_2	5	2
P_3	3	0
P_4	20	4
P_5	2	3

If the CPU scheduling policy is SJF, find the average waiting time (with pre-emption).
(A) 4.8 ms (B) 5.6 ms
(C) 2.16 ms (D) 2.8 ms

3. Consider the following snapshot of a system:

	Allocation				Max				Available			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*	*A*	*B*	*C*	*D*
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

Which of the following is true?
(i) The system is in a safe state.
(ii) If a request from process *P*, arrives for (0, 4, 2, 0). Then the request can be granted.
(A) Only (i) (B) Only (ii)
(C) Both (i) and (ii) (D) Neither (i) nor (ii)

4. In a Round Robin scheduling context switch time is 4 units, the average process running time before blocking is 6 units then CPU efficiency is
(A) 0.2 (B) 0.4
(C) 0.6 (D) 0.1

5. A short-term scheduler executes at least once every 20 msec. If it takes 2 msec to decide to execute a process for 2 msec, what is the percentage of CPU time wasted?
(A) 8% (B) 9%
(C) 10% (D) 11%

6. Consider a system which has n resources of the same type. The n resources are shared among three processes *A*, *B*, *C*, which have high demands of 3, 5, 6, respectively. For what value of n will deadlock not occur?
(A) 11 (B) 10
(C) 9 (D) 15

7. A comparative study of scheduling algorithm was performed, the average arrival time in the queue is 5 *m* sec and waiting time of the processes is 10 msec. What is the average queue length of the waiting processes?
(A) 50 (B) 60
(C) 70 (D) 80

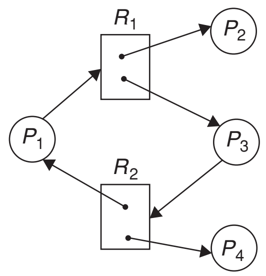
8. A CPU scheduling algorithm determines an order for the execution of its scheduled processes. Given five processes to be scheduled on one processor, how many possible different schedules are there?
(A) 50 (B) 100
(C) 120 (D) 150

9. Consider the following set of jobs (processes) along with their Arrival Time (AT), start time (ST) and Finish Time (FT). Find weighted turnaround time.

Job no.	AT	ST	FT
1	10.0	10.0	10.3
2	10.2	10.3	10.8
3	10.4	10.8	10.9
4	10.5	10.9	11.3
5	10.8	11.3	11.4

(A) 3.04 (B) 2.04
(C) 4.04 (D) 0.56

10. Is the following resource allocation graph in a deadlock state?



(A) Yes (B) No
(C) Not predictable (D) Insufficient data

11. Starvation of longer jobs happens in one of the following scheduling algorithm?
(A) Shortest run remaining time first
(B) Round Robin
(C) Highest response ratio next
(D) First-come first-served

12. Suppose n processes, $P_1 \ldots P_n$ share n identical resource units, which can be reserved and released one at a time. The maximum resource requirement of process P_i is S_i, where $S_i > 0$. Which one of the following is a sufficient condition for ensuring that deadlock does not occur?

(A) $+ i, S_i < m$ (B) $+ i, S_i < n$

(C) $\sum_{i=1}^{n} S_i < (m+n)$ (D) $\sum_{i=1}^{n} S_i < (m*n)$

13. A system with following processes and resources exists. Check the system for safe state and find the safe sequence of processes

	Allocation			Max			Available		
	X	*Y*	*Z*	*X*	*Y*	*Z*	*X*	*Y*	*Z*
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

(A) $< P_1, P_3, P_4, P_2, P_0 >$
(B) $< P_3, P_4, P_2, P_0, P_1 >$
(C) $< P_2, P_4, P_0, P_1, P_3 >$
(D) The system is in unsafe state.

14. Does the below statements be executed concurrently?

$S_1 : a = x + y$
$S_2 : b = z + 1$

(A) Yes (B) No
(C) Not predictable (D) None of the above

15. Let *A*, *B*, *C* be three jobs. Their arrival time and execution time are shown below. By applying monoprogramming and multiprogramming (use Round Robin with time slice 1 unit) approaches, calculate the amount of reduction in turnaround time?

Job	Arrival time	Execution time
A	1	2
B	2	6
C	3	1

(A) 3.33 (B) 4.33
(C) 5.33 (D) 2.33

16. Consider a system with three processes *A*, *B*, *C* with 15 tape drivers. Process A has 4 tape drives but requires 14 tape drives.
Process *B* has 5 tape drives but requires 9 tape drives.
Process *C* has 3 tape drives but requires 7 tape drives.

Among the following processes which will enter the deadlock state?

(A) *A*, *B* only (B) *A*, *B*, *C*
(C) *A*, *C* only (D) *B*, *C* only

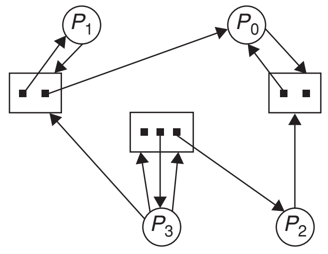
17. Assume that the following jobs are to be executed on a uniprocessor system:

Job id	CPU burst time
P	4
Q	1
R	8
S	1
T	2

The jobs are assumed to have arrived at 0, and in the order *P*, *Q*, *R*, *S* and *T*. Calculate the departure with time slice (completion time) for job *P* if scheduling is Round Robin with time slices of 1 unit (slice).

(A) 4 (B) 10
(C) 11 (D) 12

Common data for questions 18 and 19: Consider the following Resource Allocation Graph:



18. The system is in a deadlock state. This remark is:

(A) True (B) False
(C) Impossible to determine
(D) Unpredictable

19. Which one is a safe sequence?

(A) P_0, P_1, P_2, P_3 (B) P_1, P_0, P_2, P_3
(C) P_2, P_0, P_1, P_3 (D) Both (A) and (C)

Practice Problem 2

Directions for questions 1 to 15: Select the correct alternative from the given choices.

1. Let there be five processes (P_1 to P_5) and three resource types *A*, *B*, *C*.
Resource type *A* has 10 instances,
Resource type *B* has 5 instances,
Resource type *C* has 7 instances.

Suppose that at time T_0, the following snapshot of the system has been taken.

	Allocation			Max			Available		
P_1	0	1	0	7	5	3	3	3	2
P_2	2	0	0	3	2	2			
P_3	3	0	2	9	0	2			
P_4	2	1	1	2	8	2			
P_5	0	0	2	4	3	3			

Which of the following statement is true?

(A) The system is in safe state.
(B) The system has process initiation denial problem.
(C) No process causes initiation denial
(D) Both (A) and (C)

2. Consider three CPU intensive processes, which require 20, 30 and 40 time units and arrive at times 0, 2 and 4, respectively. How many context switches are needed if the operating system implements a shortest remaining time first scheduling algorithm? Do not count the context switches at time zero and at the end.
(A) 0 (B) 1
(C) 2 (D) 3

3. Consider the set of processes P_1 to P_5 with the following CPU burst times. Find the average turnaround time using shortest remaining time first.

Process	CPU burst time	Arrival time
P_1	3	0
P_2	6	2
P_3	4	4
P_4	5	6
P_5	2	8

(A) 1.3 ms (B) 3.5 ms
(C) 5.8 ms (D) 7.2 ms

4. All processes are arriving at time 0, find the average waiting time.

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

(A) 8.2 ms (B) 4.1 ms
(C) 2.0 ms (D) 1.3 ms

5. Consider a set of three processes P_1, P_2 and P_3 with their priorities and arrival times as given below.

Process	Burst time	Priority	Arrival time
P_1	10	3	0
P_2	5	2	1
P_3	2	1(highest)	2

Find the average waiting time.
(A) 1 ms (B) 2 ms
(C) 3 ms (D) 4 ms

6. The portion of the process scheduler in an OS that dispatches processes is concerned with:
(A) assigning ready processes to the CPU
(B) activating suspended I/O bound processes
(C) temporarily suspending processes when the CPU load is too great.
(D) All the above

7. In a time-sharing OS, when the time slot given to a process is completed, the process goes from the running state to the
(A) blocked state (B) ready state
(C) suspended state (D) terminated state

8. On a system with n CPUs, what is the maximum number of processes that can be in the ready state?
(A) *n* processes
(B) No process can be in ready state
(C) There is no limit to the number of processes in the ready state
(D) None of the above

9. Consider a set of *n* tasks with known runtimes $r_1, r_2, \ldots, r_n$ to be run on a uniprocessor machine. Which of the following processor scheduling algorithms will result in the maximum throughput?
(A) Round Robin
(B) SJF
(C) Highest response ratio next
(D) First-come first-served

10. Match the following:

A	Critical region	I	Hoare's monitor
B	Wait/signal	II	Mutual exclusion
C	Working set	III	Principle of locality
D	Deadlock	IV	Circular wait

(A) A – II, B – I, C – III, D – IV
(B) A – I, B – II, C – III, D – IV
(C) A – II, B – I, C – IV, D – III
(D) A – I, B – II, C – IV, D – III

11. Consider three processes *A*, *B*, *C* to be scheduled as per SRT algorithm. A is known to be scheduled first and when A has been running for 7 units of time, *C* has arrived. *C* has run for 1 unit of time when *B* has arrived and completed running in 2 units of time, what could be the minimum time of executions for *A* and *C*?
(A) 11 and 4 (B) 11 and 3
(C) 12 and 3 (D) 12 and 4

12. Select the correct statements from below:
(i) SRT and SPN can cause starvation for larger processes.
(ii) FCFS can potentially block small processes in favour of much larger processes.
(iii) Round Robin algorithm gives fair treatment to all the processes.
(iv) FCFS is a pre-emptive algorithm.
(v) The throughput for Round Robin is high even for small time slices.
(A) (i), (ii), (iii) (B) (i), (ii), (iv)
(C) (iii), (iv), (v) (D) (i), (ii), (iii), (v)

13. Three processes share four resource units that can be reserved and released only one at a time. Each process needs a maximum of two units. Then
(A) there is a possibility of deadlock
(B) no deadlock will occur

(C) there will be a circular wait
(D) nothing can be predicted about dead lock.

14. *N* processes share *M* resource units that can be reserved and released only one at a time. The maximum need of each process does not exceed *M* and the sum of all maximum needs is less than $M + N$, then
(A) there is a possibility of deadlock
(B) there will be no deadlock
(C) circular wait exists
(D) nothing can be predicted about dead lock.

15. Which of the following scheduling algorithms could result in starvation?
(A) First-come, first-served
(B) Shortest job first
(C) Round Robin
(D) Highest response ratio next

Previous Years' Questions

1. Consider three processes (process id 0, 1, 2, respectively) with compute bursts 2, 4 and 8 time units. All processes arrive at time zero. Consider the longest remaining time first (LRTF) scheduling algorithm. In LRTF ties are broken by giving priority to the process with the lowest process id. The average turnaround time is: **[2006]**

(A) 13 units (B) 14 units
(C) 15 units (D) 16 units

2. Consider three processes, all arriving at zero, with total execution time of 10, 20 and 30 units, respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of time does the CPU remain idle? **[2006]**

(A) 0% (B) 10.6%
(C) 30.0% (D) 89.4%

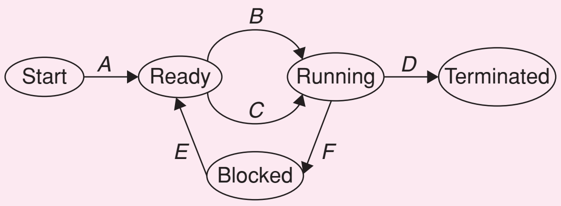
3. A single processor system has three resource types *X*, *Y* and *Z*, which are shared by three processes. There are five units of each resource type. Consider the following scenario, where the column *alloc* denotes the number of units of each resource type allocated to each process, and the column *request* denotes the number of units of each resource type requested by a process in order to complete execution. Which of these processes will finish *last*?

	alloc			request		
	X	*Y*	*Z*	*X*	*Y*	*Z*
P_0	1	2	1	1	0	3
P_1	2	0	1	0	1	2
P_2	2	2	1	1	2	0

(A) P_0 **[2007]**
(B) P_1
(C) P_2
(D) None of the above, since the system is in a deadlock.

4. Which of the following is *not* true of deadlock prevention and deadlock avoidance schemes? **[2008]**
(A) In deadlock prevention, the request for resources is always granted if the resulting state is safe
(B) In deadlock avoidance, the request for resources is always granted if the result state is safe
(C) Deadlock avoidance is less restrictive than deadlock prevention
(D) Deadlock avoidance requires knowledge of resource requirements *a priori*

5. In the following process state transition diagram for a uniprocessor system, assume that there are always some processes in the ready state: **[2009]**



Now consider the following statements:
I. If a process makes a transition *D*, it would result in another process making transition A immediately.
II. A process P_2 in blocked state can make transition *E*, while another process P_1 is in running state.
III. The OS uses pre-emptive scheduling.
IV. The OS uses non-pre-emptive scheduling.
Which of the above statements are true?
(A) I and II (B) I and III
(C) II and III (D) II and IV

6. Which of the following statements are true?
I. Shortest remaining time first scheduling may cause starvation
II. Pre-emptive scheduling may cause starvation
III. Round Robin is better than FCFS in terms of response time **[2010]**
(A) I only (B) I and III only
(C) II and III only (D) I, II and III

7. A system has n resources $R_0, \ldots, R_{n-1}$, and k processes $P_0, \ldots P_{k-1}$. The implementation of the resource request logic of each process P_i, is as follows:

```
if (i% 2 = = 0) {
if (i<n) request R_i;
if (i+2<n) request R_{i+2};
}
else {
if (i<n) request R_{n-i};
if (i+2<n) request R_{n-i-2};
}
```

In which one of the following situations is a deadlock possible? **[2010]**

(A) $n = 40, k = 26$ (B) $n = 21, k = 12$
(C) $n = 20, k = 10$ (D) $n = 41, k = 19$

8. Consider the following table of arrival time and burst time for three processes P_0, P_1 and P_2. **[2011]**

Process	Arrival time	Burst time
P_0	0 ms	9 ms
P_1	1 ms	4 ms
P_2	2 ms	9 ms

The pre-emptive shortest job first scheduling algorithm is used. Scheduling is carried out only at arrival or completion of processes. What is the average waiting time for the three processes?

(A) 5.0 ms (B) 4.33 ms
(C) 6.33 ms (D) 7.33 ms

9. Consider the three processes, P_1, P_2 and P_3 as shown in the table. **[2012]**

Process	Arrival time	Time units required
P_1	0	5
P_2	1	7
P_3	3	4

The completion order of the three processes under the policies FCFS and RR$_2$ (Round Robin scheduling with CPU quantum of 2 time units) are

(A) **FCFS:** P_1, P_2, P_3 **RR2:** P_1, P_2, P_3
(B) **FCFS:** P_1, P_3, P_2 **RR2:** P_1, P_3, P_2
(C) **FCFS:** P_1, P_2, P_3 **RR2:** P_1, P_3, P_2
(D) **FCFS:** P_1, P_3, P_2 **RR2:** P_1, P_2, P_3

10. A scheduling algorithm assigns priority proportional to the waiting time of a process. Every process starts with priority zero (the lowest priority). The scheduler re-evaluates the process priorities every T time units and decides the next process to schedule. Which one of the following is TRUE if the processes have no I/O operations and all arrive at time zero? **[2013]**

(A) This algorithm is equivalent to the first-come-first-serve algorithm.
(B) This algorithm is equivalent to the Round Robin algorithm.
(C) This algorithm is equivalent to the shortest-job-first algorithm.
(D) This algorithm is equivalent to the shortest-remaining-time-first algorithm.

11. An operating system uses the *Banker's algorithm* for deadlock avoidance when managing the allocation of three resource types X, Y and Z to three processes P_0, P_1, and P_2. The table given below presents the current system state. Here, the *allocation* matrix shows the current number of resources of each type allocated to each process and the *Max* matrix shows the maximum number of resources of each type required by each process during its execution. **[2014]**

	Allocation			Max		
	X	Y	Z	X	Y	Z
P_0	0	0	1	8	4	3
P_1	3	2	0	6	2	0
P_2	2	1	1	3	3	3

There are three units of type X, two units of type Y and two units of type Z still available. The system is currently in a *safe* state. Consider the following independent requests for additional resources in the current state:

REQ1: P_0 requests 0 units of X, 0 units of Y and two units of Z

REQ 2: P_1 requests two units of X, 0 units of Y and 0 units of Z

Which one of the following is true?

(A) Only REQ1 can be permitted
(B) Only REQ2 can be permitted
(C) Both REQ1 and REQ2 can be permitted
(D) Neither REQ1 nor REQ2 can be permitted

12. Consider the following set of processes that need to be scheduled on a single CPU. All the times are given in milliseconds.

Process name	Arrival time	Execution time
A	0	6
B	3	2
C	5	4
D	7	6
E	10	3

Using the *shortest remaining time first* scheduling algorithm, the average process turnaround time (in msec) is ——. **[2014]**

13. Three processes A, B and C each execute a loop of 100 iterations. In each iteration of the loop, a process performs a single computation that requires t_c CPU milliseconds and then initiates a single I/O operation that lasts for t_{io} milliseconds. It is assumed that the computer where the processes execute has sufficient number of I/O devices and the OS of the computer assigns different I/O devices to each process. Also, the scheduling overhead of the OS is negligible. The processes have the following characteristics:

Process id	t_c	t_{io}
A	100 ms	500 ms
B	350 ms	500 ms
C	200 ms	500 ms

The processes *A*, *B* and *C* are started at times 0, 5 and 10 milliseconds, respectively, in a pure time sharing system (Round Robin scheduling) that uses a time slice of 50 milliseconds. The time in milliseconds at which process *C* would *complete* its first I/O operation is ——— **[2014]**

14. A system contains three programs and each requires three tape units for its operation. The minimum number of tape units which the system must have such that deadlocks never arise is ———. **[2014]**

15. An operating system uses *shortest remaining time first* scheduling algorithm for pre-emptive scheduling of processes. Consider the following set of processes with their arrival times and CPU burst times (in milliseconds):

Process	Arrival time	Burst time
P_1	0	12
P_2	2	4
P_3	3	6
P_4	8	5

The average waiting time (in milliseconds) of the processes is _____. **[2014]**

16. Consider a uniprocessor system executing three tasks T_1, T_2 and T_3, each of which is composed of an infinite sequence of jobs (or instances) which arrive periodically at intervals of 3, 7 and 20 milliseconds, respectively. The priority of each task is the inverse of its period, and the available tasks are scheduled in order of priority, with the highest priority task schedule first. Each instance of T_1, T_2 and T_3 requires an execution time of 1, 2 and 4 milliseconds, respectively. Given that all tasks initially arrive at the beginning of the 1st millisecond and task preemptions are allowed, the first instance of T_3 completes its execution at the end of ________ milliseconds. **[2015]**

17. A system has 6 identical resources and *N* processes competing for them. Each process can request atmost 2 resources. Which one of the following values of *N* could lead to a deadlock? **[2015]**

(A) 1 (B) 2
(C) 3 (D) 4

18. The maximum number of processes that can be in Ready state for a computer system with n CPUs is **[2015]**

(A) n (B) n^2
(C) 2^n (D) Independent of n

19. Consider the following policies for preventing deadlock in a system with mutually exclusive resources. **[2015]**

(1) Processes should acquire all their resources at the beginning of execution. If any resource is not available, all resources acquired so far are released.

(2) The resources are numbered uniquely, and processes are allowed to request for resources only in increasing resource numbers.

(3) The resources are numbered uniquely, and processes are allowed to request for resources only in decreasing resource numbers.

(4) The resources are numbered uniquely. A process is allowed to request only for a resource with resource number larger than its currently held resources.

Which of the above policies can be used for preventing deadlock?

(A) Any one of 1 and 3 but not 2 or 4
(B) Any one of 1, 3 and 4 but not 2
(C) Any one of 2 and 3 but not 1 or 4
(D) Any one of 1, 2, 3 and 4

20. For the processes listed in the following table, which of the following scheduling schemes will give the lowest average turnaround time? **[2015]**

Process	Arrival Time	Processing Time
A	0	3
B	1	6
C	4	4
D	6	2

(A) First Come First Serve
(B) Non-preemptive Shortest Job First
(C) Shortest Remaining Time
(D) Round Robin with Quantum value two

21. Consider an arbitrary set of CPU - bound processes with unequal CPU burst lengths submitted at the same time to a computer system. Which one of the following process scheduling algorithms would minimize the average waiting time in the ready queue? **[2016]**

(A) Shortest remaining time first
(B) Round-robin with time quantum less than the shortest CPU burst
(C) Uniform random
(D) Highest priority first with priority proportional to CPU burst length

22. Consider the following processes, with the arrival time and the length of the CPU burst given in milliseconds. The scheduling algorithm used is preemptive shortest remaining - time first.

Process	Arrival Time	Burst Time
P_1	0	10
P_2	3	6

P_3	7	1
P_4	8	3

The average turn around time of these processes is _____ milliseconds. **[2016]**

23. Consider the following CPU processes with arrival times (in milliseconds) and length of CPU bursts (in milliseconds) as given below :

Process	Arrival time	Burst time
P1	0	7
P2	3	3
P3	5	5
P4	6	2

If the pre-emptive shortest remaining time first scheduling algorithm is used to schedule the processes, then the average waiting time across all processes is milliseconds. **[2017]**

24. A system shares 9 tape drives. The current allocation and maximum requirement of tape drives for three processes are shown below:

Process	Current Allocation	Maximum Requirement
P1	3	7
P2	1	6
P3	3	5

Which of the following best describes current state of the system? **[2017]**

(A) Safe. Deadlocked
(B) Safe. Not Deadlocked
(C) Not Safe. Deadlocked
(D) Not Safe, Not Deadlocked

25. Consider the set of processes with arrival time (in milliseconds). CPU burst time (in milliseconds). and priority (0 is the highest priority) shown below. None of the processes have I/O burst time.

Process	Arrival Time	Burst Time	Priority
P_1	0	11	2
P_2	5	28	0
P_3	12	2	3
P_4	2	10	1
P_5	9	16	4

The average waiting time (in milliseconds) of all the processes using preemptive priority scheduling algorithm is_________. **[2017]**

26. Consider a system with 3 processes that share 4 instances of the same resource type. Each process can request a maximum of *K* instances. Resource instances can be requested and released only one at a time. The largest value of *K* that will always avoid deadlock is _______. **[2018]**

27. In a system, there are three types of resources: *E*, *F* and *G*. Four processes P_0, P_1, P_2 and P_3 execute concurrently. At the outset, the processes have declared their maximum resource requirements using a matrix named Max as given below. For example, Max[P_2, *F*] is the maximum number of instances of *F* that P_2 would require. The number of instances of the resources allocated to the various processes at any given state is given by a matrix named Allocation.

Consider a state of the system with the Allocation matrix as shown below, and in which 3 instances of *E* and 3 instances of *F* are the only resources available.

Allocation			
	E	F	G
P_0	1	0	1
P_1	1	1	2
P_2	1	0	3
P_3	2	0	0

Max			
	E	F	G
P_0	4	3	1
P_1	2	1	4
P_2	1	3	3
P_3	5	4	1

From the perspective of deadlock avoidance, which one of the following is true? **[2018]**

(A) The system is in safe state.
(B) The system is not in safe state, but would be safe if one more instance of *E* were available.
(C) The system is not in safe state, but would be safe if one more instance of *F* were available.
(D) The system is not in safe state, but would be safe if one more instance of *G* were available.

Answer Keys

Exercises

Practice Problem 1

1. C	**2.** A	**3.** C	**4.** C	**5.** B	**6.** D	**7.** A	**8.** C	**9.** A	**10.** B
11. A	**12.** C	**13.** A	**14.** A	**15.** B	**16.** B	**17.** C	**18.** B	**19.** D	

Practice Problem 2

1. B	**2.** C	**3.** D	**4.** A	**5.** C	**6.** A	**7.** B	**8.** C	**9.** B	**10.** A
11. D	**12.** A	**13.** B	**14.** B	**15.** B					

Previous Years' Questions

1. A	**2.** B	**3.** C	**4.** A	**5.** C	**6.** D	**7.** B	**8.** A	**9.** C	**10.** B
11. B	**12.** 7.2	**13.** 1000	**14.** 7	**15.** 5.5	**16.** 12	**17.** none	**18.** D	**19.** D	**20.** C
21. A	**22.** 8.2 to 8.3		**23.** 3	**24.** B	**25.** 29	**26.** 2	**27.** A		