G18

Deadlock, CPU and I/O scheduling. Memory management and virtual memory, File systems.

DEADLOCK

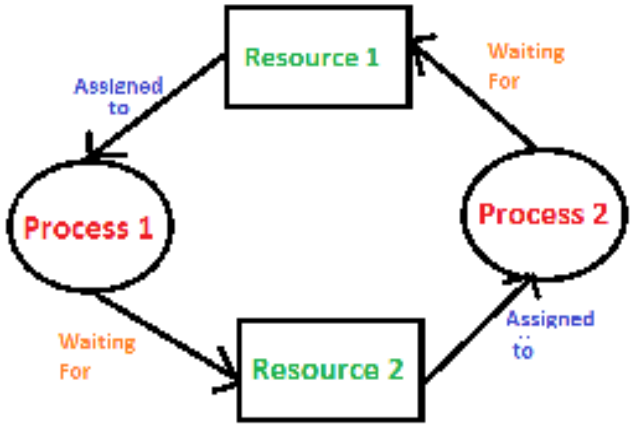
106119024

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

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

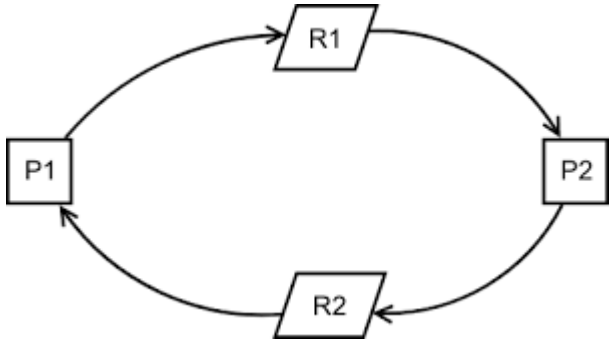
DEADLOCK

A deadlock is a situation in which two computer programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs ceasing to function



two processes compete for exclusive access to a shared resource,

This results in a circular waiting pattern that can never be broken, and both processes remain blocked indefinitely.



Necessary conditions for Deadlock

➔ The four necessary conditions for a deadlock situation are

❑ **mutual exclusion**

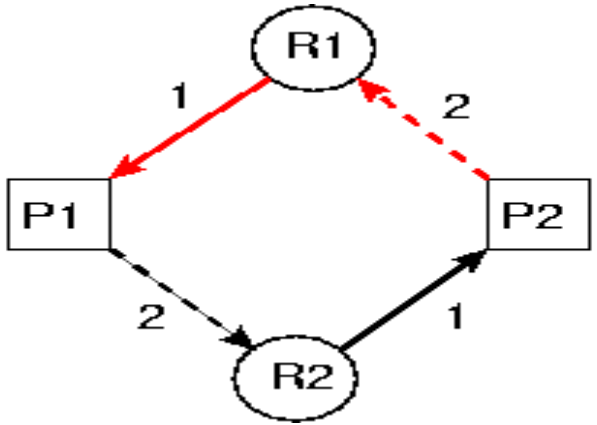
Resources cannot be shared

❑ **no preemption**

Resource can be released voluntarily by the process itself

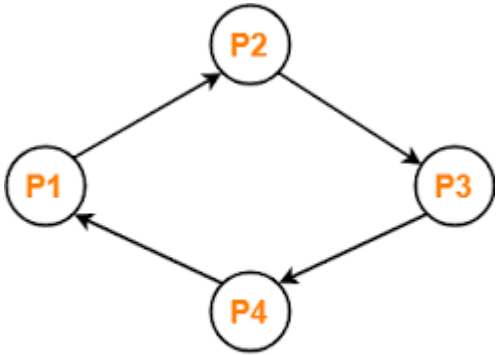
❑ hold and wait

A process is holding atleast 1 resource and waiting for other resource



❑ circular set.

each process must be waiting for a resource which is being held by another process,



Circular Wait

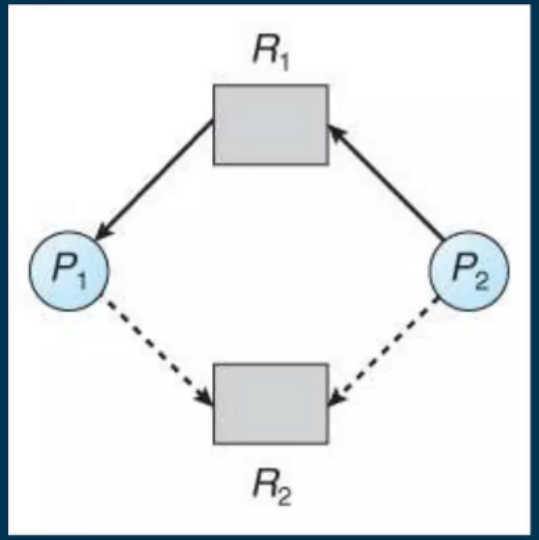
If you can make any one of the above condition as false then deadlock can be prevented

Approaches to Handling Deadlocks

- **Deadlock Prevention**
  - disallow 1 of the 3 necessary conditions of deadlock occurrence, or prevent the circular wait condition from happening
- **Deadlock Avoidance**
  - do not grant a resource request if this allocation might lead to deadlock
- **Deadlock Detection and Recovery**
  - grant resource requests when possible, but periodically check for the presence of deadlock and then take action to recover from it

Deadlock Prevention

At least one of 4 deadlock conditions is never satisfied.



Elimination of

→ Mutual Exclusion
→ Hold & Wait Condition
→ No Preemption Condition
→ Circular Wait

Deadlock Prevention

Mutual Exclusion

We can deny this situation by simple protocol i.e.,

“ Convert All non-sharable resources to sharable Resources “

Deadlock Prevention

Hold and Wait

We can deny this situation with the following Protocols:

→ **A Process can Request the Resources only when the Process has None.**

→ **Each Process to request and be allocated all its Resources before it begins Execution.**

Deadlock Prevention

No Preemption

To ensure that this condition does not hold, we use the following Protocol:

→ **We preempt the desired resources from the waiting process and allocate them to the requesting Process.**

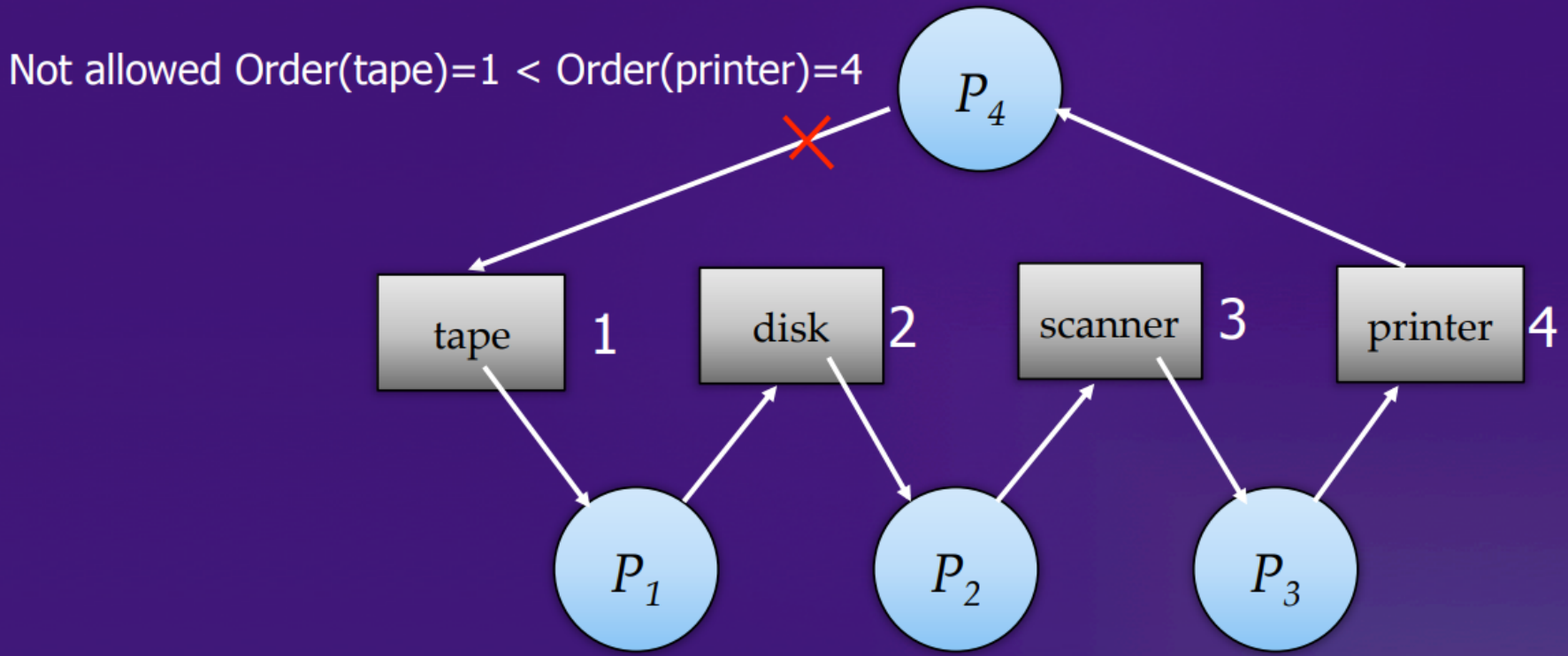
Deadlocks Prevention

Circular Wait

We ensure that circular wait must not happened if we apply a simple solution:

→ **Numbering all the Resources types and each Process Request resources in an Increasing order of enumeration**.

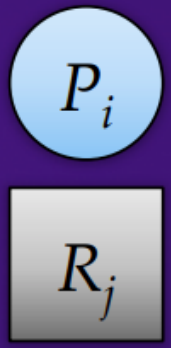
Each process can request resources only in an increasing order of enumeration.



Resource-Allocation Graph

A set of vertices V and a set of edges E

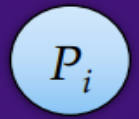
- V is partitioned into two types:
  - ✓ $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system
  - ✓ $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system



- **request edge** – directed edge $P_i \rightarrow R_j$
- **assignment edge** – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph

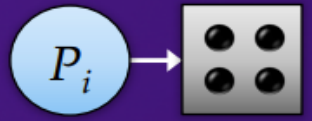
- Process



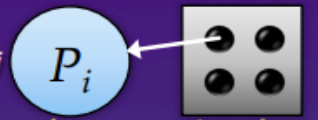
- Resource Type with 4 instances

Sequence of process resource utilization

- P_i requests instance of R_j



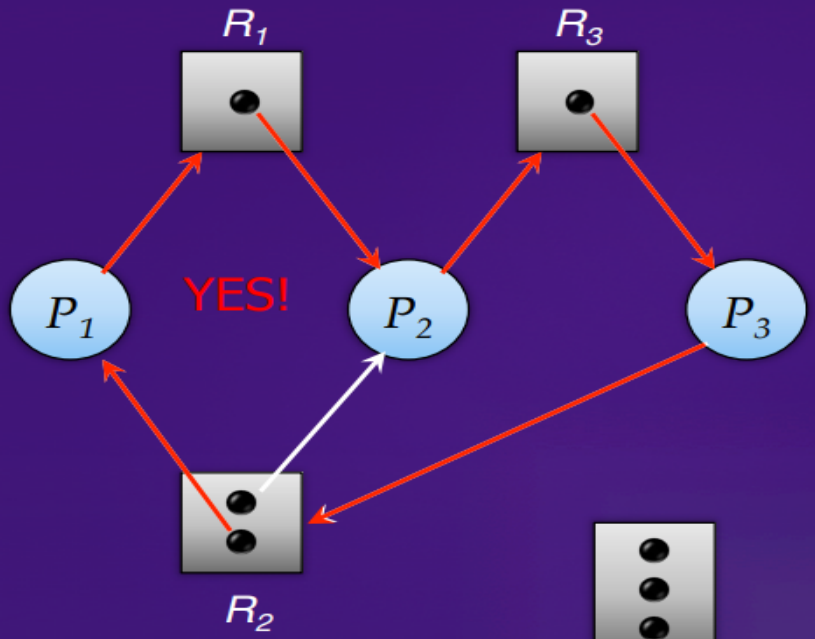
- P_i is holding an instance of R_j



- P_i releases an instance of R_j

Determining Deadlock from RAG

Resource-Allocation Graph

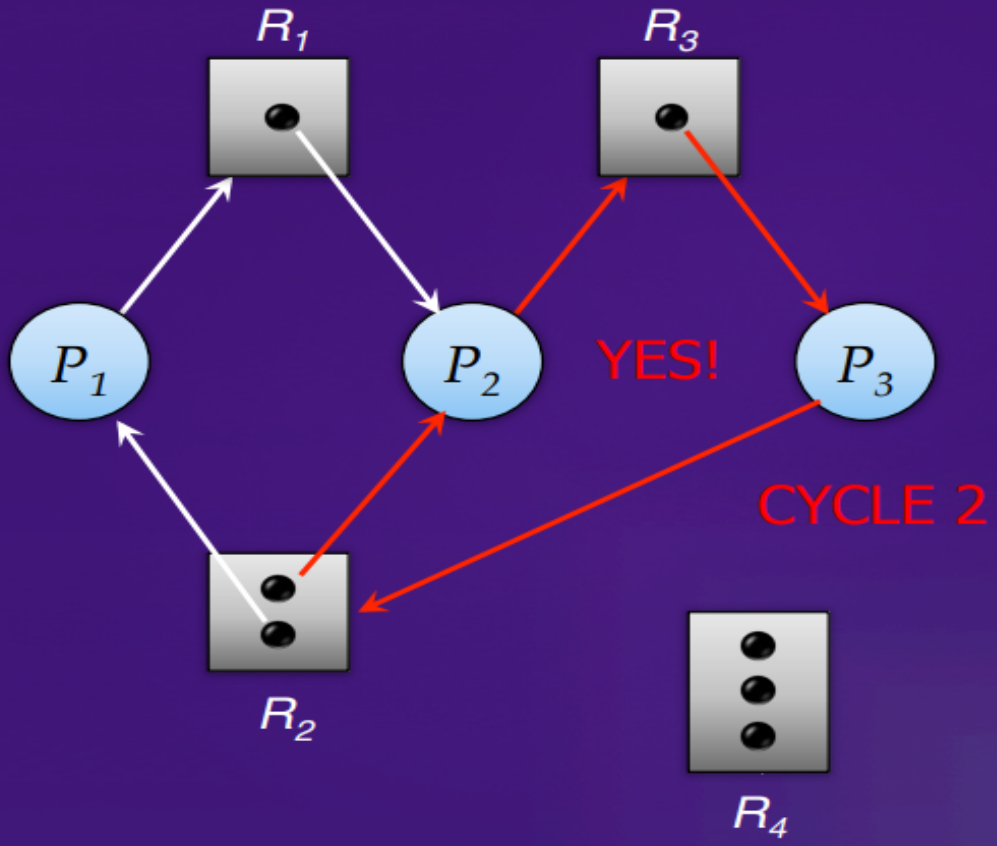


CYCLE 1

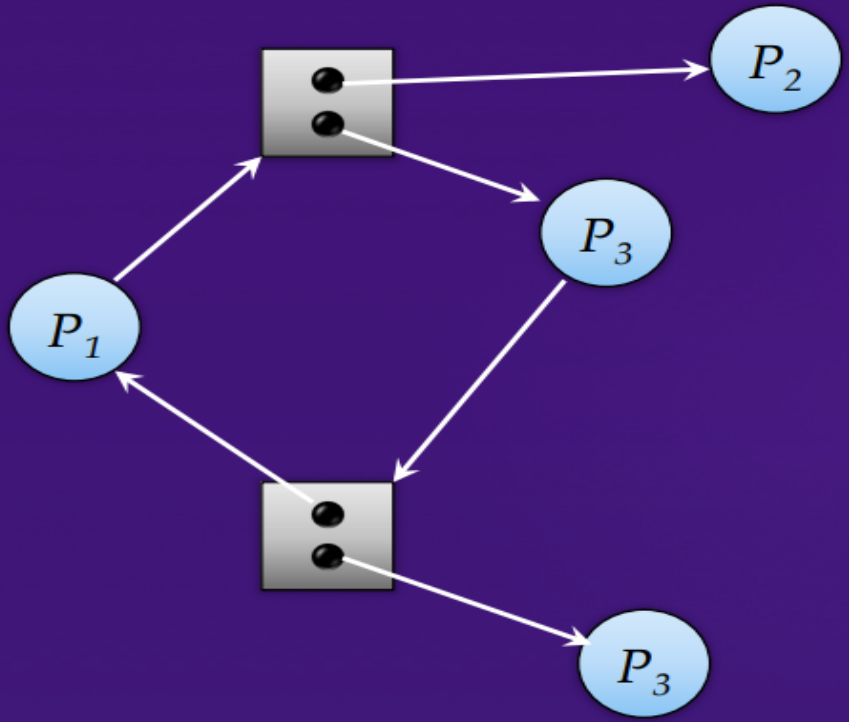
YES!

Is deadlock possible?

Resource-Allocation Graph



Resource Allocation Graph With A Cycle But No Deadlock



Conclusions from RAG

- 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.

Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.
- System is in **safe state** if there exists a sequence

$$<P_1, P_2, \ldots, P_n>$$

of ALL the processes in the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j < i$

- That is:
  - ✓ If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.
  - ✓ When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
  - ✓ When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

A system is in a safe state only if there exists a only if there exists a safe sequence.

Deadlock Avoidance

Deadlock can be avoided if certain information about processes are available to the operating system before allocation of resources.

For every resource request, the system sees whether granting the request will cause the system to enter an unsafe state that means this state could result in deadlock.

The system then only grants the requests that will lead to safe states.

AVOIDANCE ALGORITHMS

- Single instance of a resource type
  - Use a resource-allocation graph
- Multiple instances of a resource type
  - Use the banker's algorithm

Deadlock Avoidance

Avoidance – Safe/Unsafe states

- A state is safe is the system can allocate resources to each process in some order and still avoid a deadlock. More formally, a system is in a safe state only if there exists a safe sequence.
- If no safe sequence exists, then the state is said to be unsafe.

Deadlock Avoidance

Banker's Algorithm - Definitions

- Multiple resource instances
- Each process must a priori claim maximum use
- When a process requests a resource it may have to wait
- When a process gets all its resources it must return them in a finite amount of time

Deadlock Avoidance

Banker's Algorithm - Definitions

Let n = number of processes, and m = number of resources types.

- **Available** – Vector of length m indicates the number of available resources of each type. If $Available[j] = k$, then k instances of resource type R_j are available.
- **Max** : $n \times m$ matrix. Defines the maximum demand of each process. If $Max[i,j] = k$, then process P_i may request at most k instances of resource type R_j.
- **Allocation** : $n \times m$ matrix. Defines the number of resources of each type currently allocated to each process. If $Allocation[i,j] = k$, then process P_i is currently allocated k instances of resource type R_j.
- **Need** : $n \times m$ matrix. Indicates the remaining resource need of each process. If $Need[i,j] = k$, then process P_i may need k more instances of resource type R_j to complete its task. Note that

$$Need[i,j] - Max[i,j] - Allocation[i,j]$$

Resource-Request Algorithm for Process P_i

Request = request vector for process P_i. If $Request_i[j] = k$ then process P_i wants k instances of resource type R_j

1. If $Request_i \leq Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim

2. If $Request_i \leq Available$, go to step 3. Otherwise P_i must wait, since resources are not available

3. Pretend to allocate requested resources to Pi by modifying the state as follows:

$$Available = Available - Request;$$
$$Allocation_i = Allocation_i + Request_i;$$
$$Need_i = Need_i - Request_i$$

- If safe ⇒ the resources are allocated to P_i
- If unsafe ⇒ P_i must wait, and the old resource-allocation state is restored

Deadlock Avoidance

Banker's Algorithm

- 5 processes P0 through P4
- Each process must claim **Max** use in advance.
- Resource Types: A (10 instances), B (5instances), and C (7 instances)

Process	Allocation	Max	Need	Initial	Avail
	A B C	A B C	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 3	10 5 7	3 3 2
P1	2 0 0	3 2 2	1 2 2		
P2	3 0 2	9 0 2	6 0 0		
P3	2 1 1	2 2 2	0 1 1		
P4	0 0 2	4 3 3	4 3 1		

Snapshot at time T_0:

Deadlock Avoidance

Banker's Algorithm – P_0 Request (4 3 0)

- Check that Request ≤ Available (that is, (4 3 0) ≤ (3 3 2) ⇒ false

Process	Allocation	Max	Need	Initial	Avail
	A B C	A B C	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 3	10 5 7	3 3 2
P1	2 0 0	3 2 2	1 2 2		
P2	3 0 2	9 0 2	6 0 0		
P3	2 1 1	2 2 2	0 1 1		
P4	0 0 2	4 3 3	4 3 1		

Snapshot at time T_1:

Deadlock Avoidance

Banker's Algorithm – P_1 Request (1 0 2)

- Check that Request ≤ Available (that is, (1 0 2) ≤ (3 3 2) ⇒ true
- Execute safety algorithm shows that sequence < P1, P3, P4, P0, P2> satisfies safety requirement

Process	Allocation	Max	Need	Initial	Avail
	A B C	A B C	A B C	A B C	A B C
P0	0 1 0	7 5 3	7 4 3	10 5 7	2 3 0
P1	3 0 2	3 2 2	1 2 2		
P2	3 0 2	9 0 2	6 0 0		
P3	2 1 1	2 2 2	0 1 1		
P4	0 0 2	4 3 3	4 3 1		

Snapshot at time T_1:

Deadlock Detection

Deadlock detection involves

periodically checking the system state to determine if a deadlock has occurred or is likely to occur, and taking appropriate actions to resolve or recover from the deadlock.

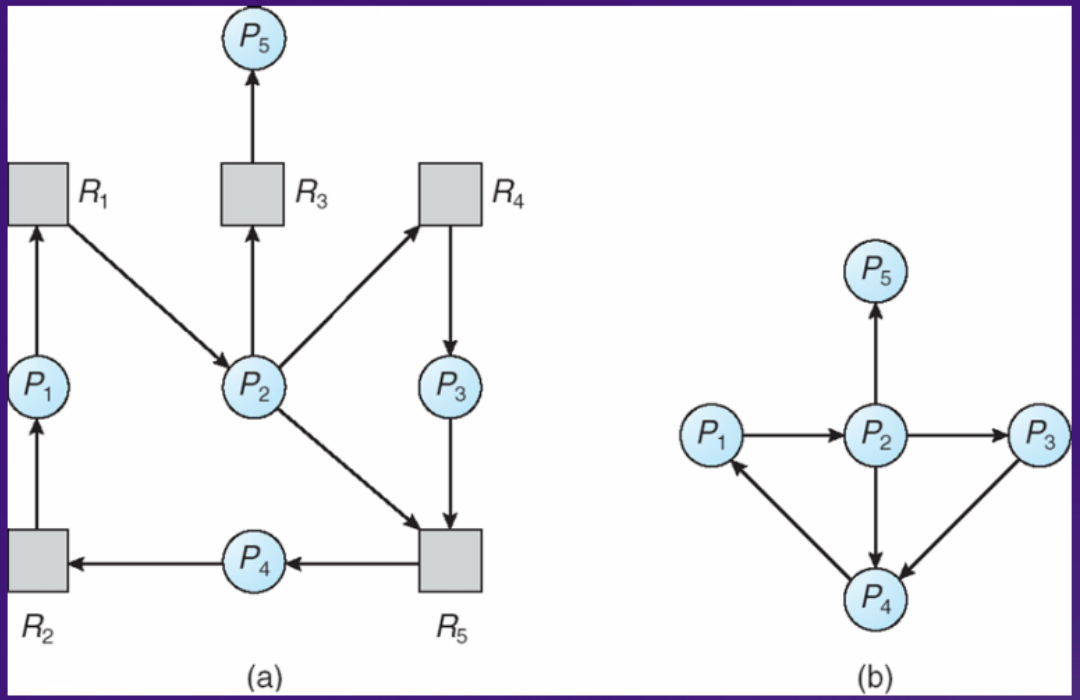
Deadlock Detection

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme

Single Instance of Each Resource Type

- Maintain wait-for graph
  - Nodes are processes
  - $P_i \rightarrow P_j$ if P_i is waiting for P_j
- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock
- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph

Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph

Corresponding wait-for graph

Several Instances of a Resource Type

- **Available**: A vector of length *m* indicates the number of available resources of each type.
- **Allocation**: An *n* x *m* matrix defines the number of resources of each type currently allocated to each process.
- **Request**: An *n* x *m* matrix indicates the current request of each process. If *Request* $[i_j] = k$, then process P_i is requesting *k* more instances of resource type. R_j.

Detection Algorithm

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively Initialize:
   (a) *Work* = *Available*
   (b) For $i = 1,2, \ldots, n$, if $Allocation_i \neq 0$, then *Finish*[i] = false;otherwise, *Finish*[i] = *true*

2. Find an index *i* such that both:
   (a) *Finish*[*i*] == *false*
   (b) $Request_i \leq Work$

   If no such *i* exists, go to step 4

3. $Work = Work + Allocation_i$
   Finish[*i*] = *true*
   go to step 2

4. If *Finish*[*i*] == false, for some *i*, $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if *Finish*[*i*] == *false*, then P_i is deadlocked

This algorithm requires an order of O(m x n^2) operations to detect whether the system is in deadlocked state.

Detection Algorithm Example

- Five processes P_0 through P_4;
- Three resources types A (7 instances), *B* (2 instances), and *C* (6 instances)
- Sequence <P0, P2, P3, P1, P4> will result in *Finish*[*i*] = true for all *i*

Process	Allocation	Request	Available
	A B C	A B C	A B C
P0	0 1 0	0 0 0	0 0 0
P1	2 0 0	2 0 2	
P2	3 0 3	0 0 0	
P3	2 1 1	1 0 0	
P4	0 0 2	0 0 2	

Snapshot at time T_0

Example (Cont.)

- P_2 requests an additional instance of type *C*

Process	Request
	A B C
P0	0 0 0
P1	2 0 1
P2	0 0 1
P3	1 0 0
P4	0 0 2

- State of system?
  - ✓ Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes; requests
  - ✓ Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4

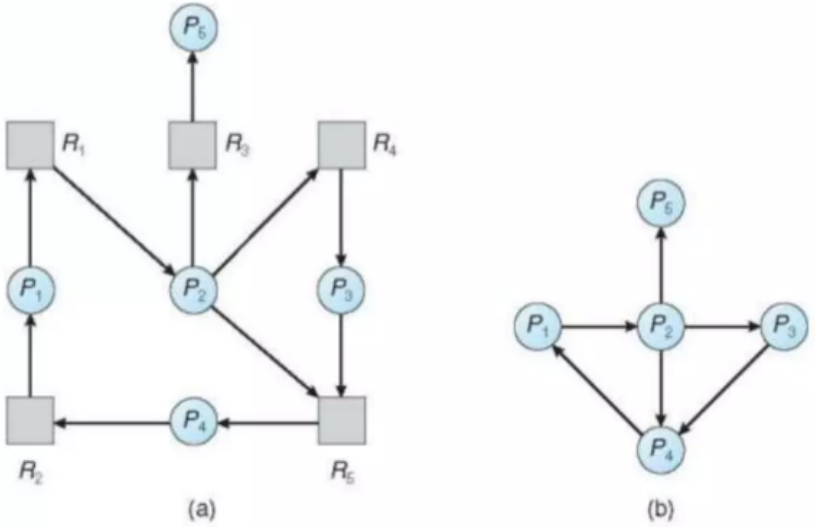
Deadlock Detection

DETECTION-ALGORITHM USAGE

- When, and how often, to invoke depends on:
  - How often a deadlock is likely to occur?
  - How many processes will need to be rolled back?
    - one for each disjoint cycle
- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes "caused" the deadlock.

Deadlock Detection

RESOURCE-ALLOCATION GRAPH AND WAIT-FOR GRAPH



Resource-Allocation Graph Corresponding wait-for graph

Once Deadlock has been detected, Some Strategy is needed for recovery. The various approaches of recovering from deadlocks are:

- **Process Termination**
- **Resource Preemption**

Deadlock Recovery

- Abort all deadlocked processes
- Abort one process at a time until the deadlock cycle is eliminated
- In which order should we choose to abort?
  1. Priority of the process
  2. How long process has computed, and how much longer to completion
  3. Resources the process has used
  4. Resources process needs to complete
  5. How many processes will need to be terminated
  6. Is process interactive or batch?

Process termination occurs when the **process** is terminated The exit() system call is used by most operating systems for **process termination**. This **process** leaves the processor and releases all its resources.

Deadlock Recovery

RECOVERY FROM DEADLOCK: RESOURCE PREEMPTION

- **Selecting a victim** – minimize cost
- **Rollback** – return to some safe state, restart process for that state
- **Starvation** – same process may always be picked as victim, include number of rollback in cost factor

Exercise:

Suppose n processes, P1, …. Pn share m identical resource units, which can be reserved and released one at a time. The maximum resource requirement of process Pi is Si, where Si > 0. Which one of the following is a sufficient condition for ensuring that deadlock does not occur? (GATE CS 2005)

(a) $\forall i, s_i < m$

(b) $\forall i, s_i < n$

(c) $\sum_{i=1}^{n} s_i < (m+n)$

(d) $\sum_{i=1}^{n} s_i < (m * n)$

Answer (c)

In the extreme condition, all processes acquire Si-1 resources and need 1 more resource. So $\sum_{i=1}^{n}(Si-1)$ <m

=> $\sum_{i=1}^{n} Si$ <m+n

following condition must be true to make sure that deadlock never occurs.

CPU AND I/O Scheduling in Operating Systems

106119092

Process Scheduling

- Process Scheduling is the process of the process manager handling the removal of an active process from the CPU and selecting another process based on a specific strategy.
- Process Scheduling is an integral part of Multi-programming applications. Such operating systems allow more than one process to be loaded into usable memory at a time and the loaded shared CPU process uses repetition time.

Types Of Process Scheduling

- Long term or Job Scheduler
- Short term or CPU Scheduler
- Medium-term Scheduler

Need to Schedule Processes

Scheduling is important in many different computer environments. One of the most important areas is scheduling which programs will work on the CPU. This task is handled by the Operating System (OS) of the computer and there are many different ways in which we can choose to configure programs.

Process Scheduling allows the OS to allocate CPU time for each process. Another important reason to use a process scheduling system is that it keeps the CPU busy at all times. This allows you to get less response time for programs.

Context Switching

- Considering that there may be hundreds of programs that need to work, the OS must launch the program, stop it, switch to another program, etc. The way the OS configures the system to run another in the CPU is called Context Switching. If the OS keeps context-switching programs in and out of the provided CPUs, it can give the user a tricky idea that he or she can run any programs he or she wants to run, all at once.

Need for CPU Scheduling Algorithm

CPU scheduling is the process of deciding which process will own the CPU to use while another process is suspended. The main function of the CPU scheduling is to ensure that whenever the CPU remains idle, the OS has at least selected one of the processes available in the ready-to-use line.

In Multiprogramming, if the long-term scheduler selects multiple I / O binding processes then most of the time, the CPU remains an idle. The function of an effective program is to improve resource utilization.

Scheduling **Criteria**

- **CPU utilization:** The main purpose of any CPU algorithm is to keep the CPU as busy as possible.
- **Throughput**: The average CPU performance is the number of processes performed and completed during each unit
- **Response Time**: In a collaborative system, turnaround time is not the best option. The process may produce something early and continue to compute the new results while the previous results are released to the user
- **Turn Around Time**: Time Difference between completion time and arrival time.

  Turn Around Time = Completion Time – Arrival Time

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

  Waiting Time = Turn Around Time – Burst Time

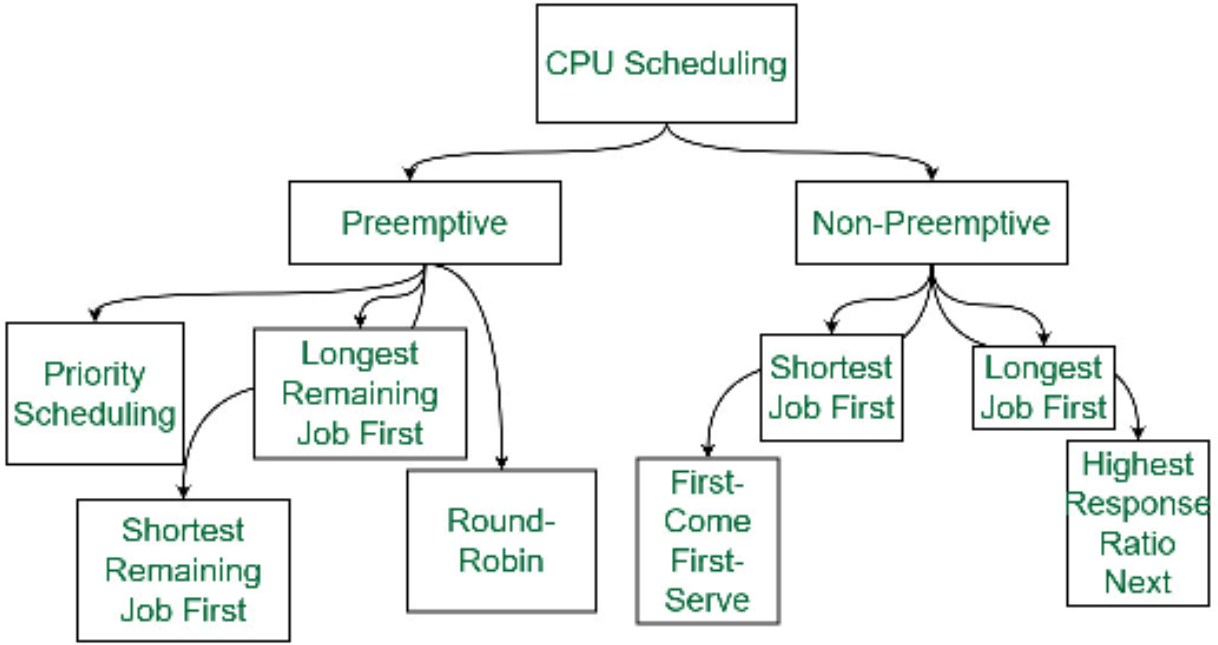
Objectives of Process Scheduling Algorithm

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

Different types of CPU Scheduling Algorithms

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

Types of Scheduling Algorithms



First Come First Serve:

- **FCFS** considered to be the simplest of all operating system scheduling algorithms. First come first serve scheduling algorithm states that the process that requests the CPU first is allocated the CPU first and is implemented by using a FIFO Queue

Characteristics of FCFS:

- Tasks are always executed on a First-come, First-serve concept.
- FCFS is easy to implement and use.
- This algorithm is not much efficient in performance, and the wait time is quite high.

Advantages of FCFS:

- Easy to implement
- First come, first serve method

Disadvantages of FCFS:

- FCFS suffers from the **Convoy effect**.
- The average waiting time is much higher than the other algorithms.
- FCFS is very simple and easy to implement and hence not much efficient.

FCFS Problem

Consider the set of 5 processes whose arrival time and burst time are given below:

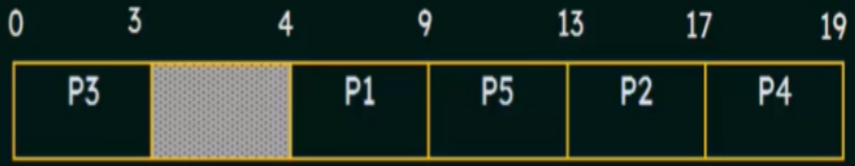
Process ID	Arrival Time	Burst Time
P1	4	5
P2	6	4
P3	0	3
P4	6	2
P5	5	4

Calculate the average waiting time and average turnaround time,
if FCFS Scheduling Algorithm is followed.

Solution:

Process ID	Arrival Time	Burst Time
P1	4	5
P2	6	4
P3	0	3
P4	6	2
P5	5	4

Gantt Chart:



The shaded box represents the idle time of CPU

Turn Around time = Completion time - Arrival time

Waiting time = Turn Around time - Burst time

Process ID	Completion Time	Turnaround Time	Waiting Time
P1	9	9 - 4 = 5	5 - 5 = 0
P2	17	17 - 6 = 11	11 - 4 = 7
P3	3	3 - 0 = 3	3 - 3 = 0
P4	19	19 - 6 = 13	13 - 2 = 11
P5	13	13 - 5 = 8	8 - 4 = 4

Now,

Average Turn Around time = (5 + 11 + 3 + 13 + 8) / 5
= 40 / 5
= 8 units

Average waiting time = (0 + 7 + 0 + 11 + 4) / 5
= 22 / 5
= 4.4 units

Shortest Job First(SJF):

- Shortest job first (SJF) is a scheduling process that selects the waiting process with the smallest execution time to execute next. Significantly reduces the average waiting time for other processes waiting to be executed.

Characteristics of SJF:

- Shortest Job First has the advantage of having a minimum average waiting time among all Scheduling Algorithms.
- It is associated with each task as a unit of time to complete.
- It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of aging.

Advantages of Shortest Job First:

- As SJF reduces the average waiting time thus, it is better than the first come first serve scheduling algorithm.
- SJF is generally used for long-term scheduling

Disadvantages of SJF:

- One of the demerits SJF has is starvation.
- Many times it becomes complicated to predict the length of the upcoming CPU request

Longest Job First(LJF):

- **Longest Job First(LJF)** scheduling process is just the opposite of the shortest job first (SJF), as the name suggests this algorithm is based upon the fact that the process with the largest burst time is processed first. Longest Job First is non-preemptive in nature.

Characteristics of LJF:

- Among all the processes waiting in a waiting queue, the CPU is always assigned to the process having the largest burst time.
- If two processes have the same burst time then the tie is broken using FCFS i.e. the process that arrived first is processed first.

Advantages of LJF:

- No other task can schedule until the longest job or process executes completely.
- All the jobs or processes finish at the same time approximately.

Disadvantages of LJF:

- Generally, the LJF algorithm gives a very high average waiting time and average Turnaround time for a given set of processes.
- This may lead to a convoy effect.

Priority Scheduling:

- **Preemptive Priority CPU Scheduling Algorithm** is a preemptive method CPU Scheduling Algorithm that works **based on the priority** of a process. In this algorithm, the editor sets the functions to be as important, meaning that the most important process must be done first. In the case of any conflict, that is, where there is more than one processor with equal value, then the most important CPU planning algorithm works on the basis of the FCFS (First Come First Serve) algorithm.

Characteristics of Priority Scheduling:

- Schedules tasks based on priority.
- When the higher priority work arrives while a task with less priority is executed, the higher priority work takes the place of the less priority one and
- The latter is suspended until the execution is complete.
- Lower the number assigned, the higher the priority level of a process.

Advantages of Priority Scheduling:

- The average waiting time is less than FCFS
- Less complex

Disadvantages of Priority Scheduling:

- One of the most common demerits of the Preemptive priority CPU scheduling algorithm is the Starvation Problem.

Round robin:

- Round Robin is a CPU scheduling algorithm where each process is cyclically assigned a fixed time slot. It is the preemptive version of the First come First Serve CPU Scheduling algorithm. Round Robin CPU Algorithm generally focuses on Time Sharing technique.

Characteristics of Round Robin:

- It's simple, easy to use, and starvation-free as all processes get the balanced CPU allocation.
- One of the most widely used methods in CPU scheduling as a core.
- It is considered preemptive as the processes are given to the CPU for a very limited time.

Advantages of Round robin:

- Round robin seems to be fair as every process gets an equal share of CPU.
- The newly created process is added to the end of the ready queue.

Round Robin Problem

Consider the set of 5 processes whose arrival time and burst time are given below:

Process ID	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

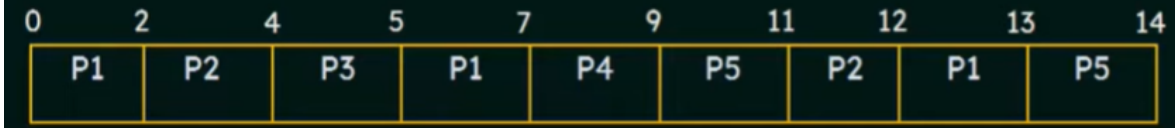
If the CPU scheduling policy is Round Robin with time quantum = 2 units, calculate the average waiting time and average turn around time.

Process ID	Arrival Time	Burst Time
P1	0	~~5~~ - ~~3~~ - ~~1~~
P2	1	~~3~~ - ~~1~~
P3	2	~~1~~ -
P4	3	~~2~~
P5	4	~~3~~ - ~~1~~

Time Quantum
2 Units

Clock

Gantt Chart:



Turn Around time = Completion time – Arrival time

Waiting time = Turn Around time – Burst time

Process ID	Completion Time	Turnaround Time	Waiting Time
P1	13	13 – 0 = 13	13 – 5 = 8
P2	12	12 – 1 = 11	11 – 3 = 8
P3	5	5 – 2 = 3	3 – 1 = 2
P4	9	9 – 3 = 6	6 – 2 = 4
P5	14	14 – 4 = 10	10 – 3 = 7

Process ID	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	1
P4	3	2
P5	4	3

Average Turn Around time

= (13 + 11 + 3 + 6 + 10) / 5

= 43 / 5 = **8.6 units**

Average waiting time

= (8 + 8 + 2 + 4 + 7) / 5

= 29 / 5 = **5.8 units**

Shortest Remaining Time First:

- **Shortest remaining time first** is the preemptive version of the Shortest job first which we have discussed earlier where the processor is allocated to the job closest to completion. In SRTF the process with the smallest amount of time remaining until completion is selected to execute.

Characteristics of Shortest remaining time first:

- SRTF algorithm makes the processing of the jobs faster than the SJF algorithm, given its overhead charges are not counted.
- The context switch is done a lot more times in SRTF than in SJF and consumes the CPU's valuable time for processing. This adds up to its processing time and diminishes its advantage of fast processing.

Advantages of SRTF:

- In SRTF the short processes are handled very fast.
- The system also requires very little overhead since it only makes a decision when a process completes or a new process is added.

Disadvantages of SRTF:

- Like the shortest job first, it also has the potential for process starvation.
- Long processes may be held off indefinitely if short processes are continually added.

Longest Remaining Time First:

- **Longest remaining time first** is a preemptive version of the longest job first scheduling algorithm. This scheduling algorithm is used by the operating system to program incoming processes for use in a systematic way. This algorithm schedules those processes first which have the longest processing time remaining for completion.

Characteristics of longest remaining time first:

- Among all the processes waiting in a waiting queue, the CPU is always assigned to the process having the largest burst time.
- If two processes have the same burst time then the tie is broken using FCFS i.e. the process that arrived first is processed first.

Advantages of LRTF:

- No other process can execute until the longest task executes completely.
- All the jobs or processes finish at the same time approximately.

Disadvantages of LRTF:

- This algorithm gives a very high average waiting time and average Turnaround time for a given set of processes.
- This may lead to a convoy effect.

Highest Response Ratio Next:

- **Highest Response Ratio Next** is a non-preemptive CPU Scheduling algorithm and it is considered as one of the most optimal scheduling algorithms. The name itself states that we need to find the response ratio of all available processes and select the one with the highest Response Ratio. A process once selected will run till completion.

Characteristics of Highest Response Ratio Next:

- The **criteria** for HRRN is **Response Ratio,** and the **mode** is **Non-Preemptive** HRRN is considered as the modification of SJF to reduce the problem of starvation.
- In comparison with SJF, the CPU is allotted to the next process which has the **highest response ratio** , and not to the process having less burst time.
- ***Response Ratio = (W + S)/S*** *(**W** is the waiting time of the process so far and **S** is the Burst time of the process)*

Advantages of HRRN:

- HRRN Scheduling algorithm generally gives better performance than the SJF.
- There is a reduction in waiting time for longer jobs and also it encourages shorter jobs.

Disadvantages of HRRN:

- The implementation of HRRN scheduling is not possible as it is not possible to know the burst time of every job in advance.
- In this scheduling, there may occur an overload on the CPU.

I/O scheduling in Operating Systems

I/O Operations

- Operating System has a certain portion of code that is dedicated to managing Input/Output in order to improve the reliability and the performance of the system.
- A computer system contains CPUs and more than one device controller connected to a common bus channel, generally referred to as the device driver.
- These device drivers provide an interface to I/O devices for communicating with the system hardware promoting ease of communication and providing access to shared memory.

I/O Requests in operating systems

- I/O Requests are managed by Device Drivers in collaboration with some system programs inside the I/O device. The requests are served by OS using three simple segments :
- **I/O Traffic Controller:** Keeps track of the status of all devices, control units, and communication channels.
- **I/O scheduler:** Executes the policies used by OS to allocate and access the device, control units, and communication channels.
- **I/O device handler:** Serves the device interrupts and heads the transfer of data.

I/O Scheduling in operating systems

- Scheduling is used for efficient usage of computer resources avoiding deadlock and serving all processes waiting in the queue. To know more about CPU Scheduling refer to CPU Scheduling in Operating Systems.

I/O Traffic Controller has 3 main tasks :

- The primary task is to check if there's at least one path available.
- If there exists more than one path, it must decide which one to select.
- If all paths are occupied, its task is to analyze which path will be available at the earliest.

I/O Scheduler functions similarly to a Process scheduler it allocates the devices, control units, and communication channels. However, under a heavy load of I/O requests, Scheduler must decide what request should be served first and for that we have multiple queues to be managed by OS.

- The major difference between a Process scheduler and an I/O scheduler is that I/O requests are not preempted: Once the channel program has started, it's allowed to continue to completion.
- Although it is feasible because programs are relatively short (50 to 100 ms). Some modern OS allows I/O Scheduler to serve higher priority requests.
- In simpler words, If an I/O request has higher priority then they are served before other I/O requests with lower priority.
- I/O scheduler works in coordination with the I/O traffic controller to keep track of which path is being served for the current I/O request.

- **I/O Device Handler** manages the I/O interrupts (if any) and scheduling algorithms. A few I/O handling algorithms are :

1. FCFS [First come first serve].
2. SSTF [Shortest seek time first].
3. SCAN
4. Look
    1. N-Step Scan
    2. C-SCAN
    3. C-LOOK

Every scheduling algorithm aims to minimize arm movement, mean response time, and variance in response time. An overview of all I/O scheduling algorithms is described below :

- **First Come First Serve [FCFS] :** It is one of the simplest device-scheduling algorithms since it is easy to program and essentially fair to users (I/O devices). The only barrier could be the high seek time, so any other algorithm that can surpass the minimum seek time is suitable for scheduling.
- **Shortest Seek Time First [SSTF] :** It uses the same ideology as the Shortest Job First in process scheduling, where the shortest processes are served first and longer processes have to wait for their turn. Comparing the SJF concept in I/O scheduling, the request with the track closest to the one being served (The one with the shortest distance to travel on disk) is next to be satisfied. The main advantage over FCFS is that it minimizes overall seek time. It favors easy-to-reach requests and postpones traveling to those that are out of the way.
- **SCAN Algorithm:** SCAn uses a status flag that tells the direction of the arm, it tells whether the arm is moving towards the center of the disk or to the other side. This algorithm moves the arm from the end of the disk to the center track servicing every request in its way. When it reaches the innermost track, it reverses the direction and moves towards outer tracks on the disk, again servicing every request in its path.
- **LOOK [Elevator Algorithm] :** It's a variation of the SCAN algorithm, here arm doesn't necessarily go all the way to either side on disk unless there are requests pending. It looks ahead to a request before servicing it. A big question that arises is "Why should we use LOOK over SCAN?". The major advantage of LOOK over SCAN is that it discards the indefinite delay of I/O requests.

Gate Questions

GATE-CS-2017

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.

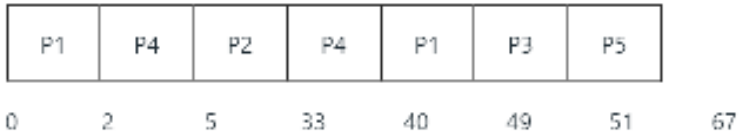
Process	Arrival Time	Burst Time	Priority
P_1	0	11	2
P_2	5	28	0
P_3	12	2	3
P_4	2	10	1
P_5	9	16	4

The average waiting time (in milliseconds) of all the processes using a preemptive priority scheduling algorithm is ____.

(A) 29
(B) 30
(C) 31
(D) 32

Solution

Gantt chart:



Process Table:

Process	Arrival Time	Burst Time	Priority	Completion time (CT)	Turnaround time (TAT) TAT= CT – AT	Waiting time (WT) WT = TAT-BT
P_1	0	11	2	49	49	38
P_2	5	28	0	33	28	0
P_3	12	2	3	51	39	37
P_4	2	10	1	40	38	28
P_5	9	16	4	67	58	42

$$\text{Average waiting time} = \frac{\sum waiting\ time\ of\ all\ the\ processes}{number\ of\ processes}$$

$$= \frac{38+0+37+28+42}{5} = 29 \text{ milliseconds}$$

Gate Questions

GATE-CS-2017

Consider the following set of processes, assumed to have arrived at time 0. Consider the CPU scheduling algorithms Shortest Job First (SJF) and Round Robin (RR). For RR, assume that the processes are scheduled in the order P1, P2, P3, and P4.

If the time quantum for RR is 4 ms, then the absolute value of the difference between the average turnaround times (in ms) of SJF and RR (rounded off to 2 decimal places) is _____.

Processes	P_1	P_2	P_3	P_4
Burst time (in ms)	8	7	2	4

(A) 5.0

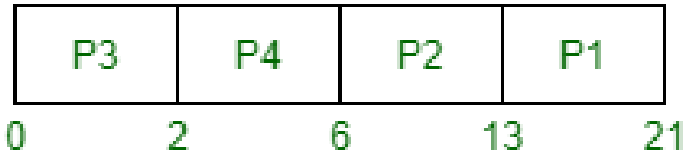
(B) 5.25

(C) 5.50

(D) 5.75

Solution

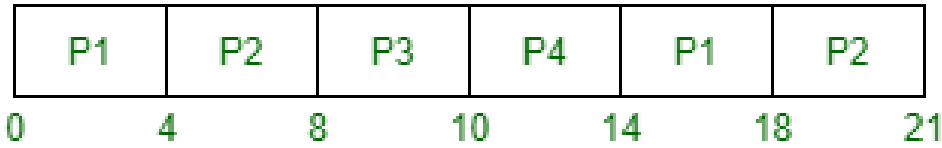
According to Shortest Job First (or SJF) CPU Scheduling, gantt chart is,



Therefore, Average Turn Around Time (TAT) is,

= `{(21 - 0) + (13 - 0) + (2 - 0) + (6 - 0)} / 4 = 10.5`

According to Round Robin (RR) CPU Scheduling with time quantum 4, gantt chart is,



Therefore, Average Turn Around Time (TAT) is,

= `{(18 - 0) + (21 - 0) + (10 - 0) + (14 - 0)} / 4 = 15.75`

Therefore, the Absolute Difference

= `15.75 - 10.5 = 5.25`

MEMORY MANAGEMENT

106119062

What Is Memory Management In OS?

- Memory Management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution.
- It is the most important function of an operating system that manages primary memory.
- It helps processes to move back and forward between the main memory and execution disk. It helps OS to keep track of every memory location, irrespective of whether it is allocated tosome process or it remains free.
- Moreover, to increase performance, several processes are executed simultaneously. For this, we must keep several processes in the main memory, so it is even more important to manage them effectively.

Why Use Memory Management?

- Here, are reasons for using Memory Management:
- It allows you to check how much memory needs to be allocated to processes that decide which processor should get memory at what time.
- Tracks whenever inventory gets freed or unallocated. According to it will update the status.
- It allocates the space to application routines.
- It also make sure that these applications do not interfere with each other. Helps protect different processes from each other.
- It places the programs in memory so that memory is utilized to its full extent.

Memory Management Techniques

1) Single Contiguous Allocation:

It is the easiest memory management technique. In this method, all types of computer's memory except a small portion which is reserved for the OS is available for one application.

2) Partitioned Allocation:

It divides primary memory into various memory partitions, which is mostly contiguous areas of memory. Every partition stores all the information for a specific task or job. This method consists of allotting a partition to a job when it starts & unallocated when it ends.

Memory Management Techniques

3) Paged Memory Management:

This method divides the computer's main memory into fixed-size units known as page frames. This hardware memory management unit maps pages into frames which should be allocated on a page basis.

4) Segmented Memory Management:

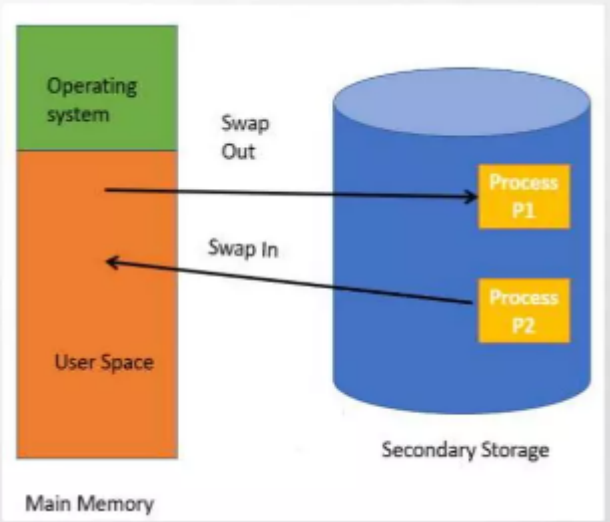
Segmented memory is the only memory management method that does not provide the user's program with a linear and contiguous address space. Segments need hardware support in the form of a segment table.

Partition Allocation

- Memory is divided into different blocks or partitions. Each process is allocated according to the requirement. Partition allocation is an ideal method to avoid internal fragmentation. Below are the various partition allocation schemes:
- First fit
- Best fit
- Worst fit
- Next fit

Types of partition

- First Fit: In this type fit, the partition is allocated, which is the first sufficient block from the beginning of the main memory.
- Best Fit: It allocates the process to the partition that is the first smallest partition among the free partitions.
- Worst Fit: It allocates the process to the partition, which is the largest sufficient freely available partition in the main memory.
- Next Fit: It is mostly similar to the first Fit, but this Fit, searches for the first sufficient partition from the last allocation point. It allows you to check how much memory needs to be allocated to processes that decide which processor should get memory at what time.



What Is Swapping?

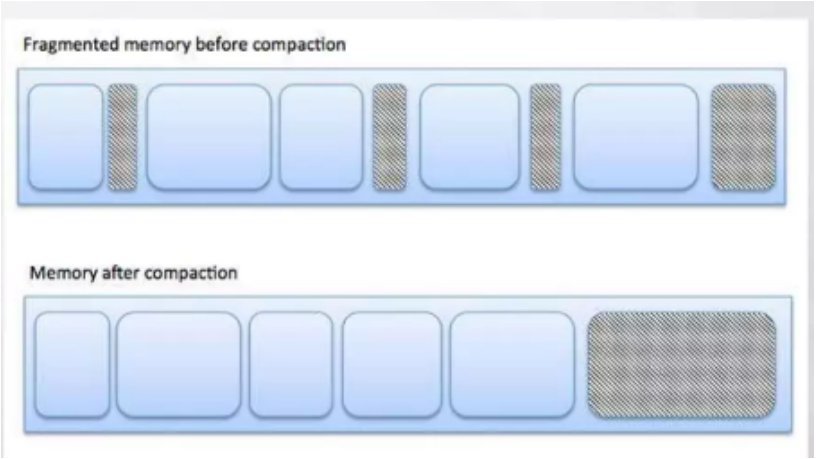
- Swapping is a method in which the process should be swapped temporarily from the main memory to the backing store. It will be later brought back into the memory for continue execution.
- Backing store is a hard disk or some other secondary storage device that should be big enough in order to accommodate copies of all memory images for all users. Here, are major benefits/pros of swapping.

Benefits of Swapping

- It offers a higher degree of multiprogramming.
- Allows dynamic relocation. For example, if address binding at execution time is being used, then processes can be swap in different locations. Else in case of compile and load time bindings, processes should be moved to the same location.
- It helps to get better utilization of memory.
- Minimum wastage of CPU time on completion so it can easily be applied to a priority-based scheduling method to improve its performance.

What Is Fragmentation?

- As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.



Types of Fragmentation

- Internal fragmentation:

  Internal fragmentation occurs when memory is allocated in fixed-size blocks, but the allocated block is larger than the amount of memory actually required by the process. The unused memory within the allocated block is called internal fragmentation. This can result in wasted memory and can be particularly problematic in embedded systems with limited memory resources.

  The internal fragmentation can be reduced by assigning the smallest partition, which is still good enough to carry the entire process.

Types of Fragmentation

External fragmentation:

External fragmentation, on the other hand, occurs when memory is allocated in variable-size blocks, but the available memory is not contiguous. This means that there may be free memory scattered throughout the system, but none of it is large enough to accommodate a request for a contiguous block of memory. This can lead to an inability to allocate memory for new processes, despite the fact that there is sufficient total memory available.

External fragmentation can be reduced by rearranging memory contents to place all free memory together in a single block.

What is Paging?

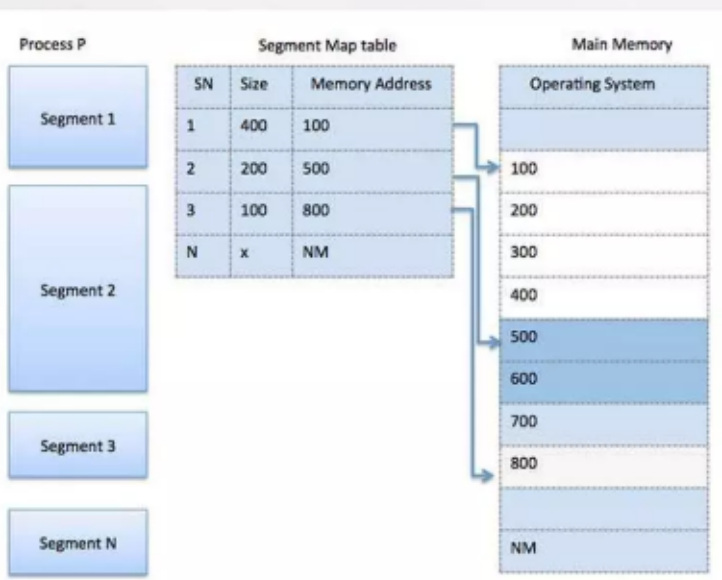
- Paging is a technique that eliminates the requirements of contiguous allocation of main memory. In this, the main memory is divided intofixed-size blocks of physical memory called frames. The size of a frame should be kept the same as that of a page to maximize the main memory and avoid external fragmentation.

Benefits of Paging

- Pages reduce external fragmentation.
- Simple to implement.
- Memory efficient.
- Due to the equal size of frames, swapping becomes very easy.
- It is used for faster access of data.

What Is Segmentation?

- Segmentation is a technique that eliminates the requirements of contiguous allocation of main memory.
- In this, the main memory is divided into variable-size blocks of physical memory called segments. It isbased on the way the programmer follows to structure their programs.



Segmentation

- With segmented memory allocation, each job is divided into several segments of different sizes, one for each module. The OS maintains a segment map table for all the processes. It also includes a list of free memory blocks along with its size, segment numbers, and its memory locations in the main memory or virtual memory.
- Segmentation memory management works very similar to paging but here segments are of variable-length whereas in paging pages are of fixed size. Functions, subroutines, stack, array, etc., are examples of such modules.

Virtual memory

- Virtual memory is a imaginary memory which we are assuming. If we have a material that exceed your memory at that time we need to use the concept of virtual memory.
- Virtual memory is temporary memory which is used along with the ram of the system.

IMPORTANCE OF VIRTUAL MEMORY

- When your computer runs out of physical memory it writes what it needs to remember to the hard disc in a swap file as virtual memory.
- If a computer running Windows requires more memory/RAM then there is installed in the system to run a program, etc, it uses a small section of the hard drive for this purpose

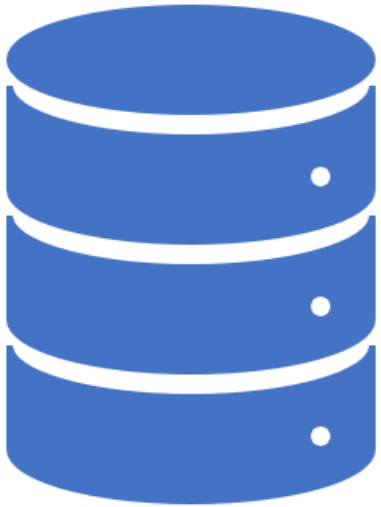
ADVANTAGES OF VIRTUAL MEMORY

- Allows processes whose aggregate memory requirements greater than the amount of physical memory, as infrequently used pages can reside on the disk.
- Virtual memory allows speed gain when only a particular segment of the program is required for the execution of the program.
- This concept is very helpful in implementing multiprogramming environment.

DISADVANTAGES OF VIRTUAL MEMORY

- Applications run slower if the system is using virtualmemory.
- It Takes more time to switch between applications.
- Less hard drive space for your use.
- It reduces system stability.

File Systems



What are files?

- In an operating system, a file is a collection of data that is stored on a computer's storage device, such as a hard disk or solid-state drive. A file can contain text, images, video, or any other type of digital content.

- Files are are typically organized into directories or folders. The operating system uses a file system to keep track of where each file is located on the storage device and to provide access to the files.

- Files can be created, copied, moved, renamed, and deleted by users or applications running on the operating system. Many operating systems also provide mechanisms for protecting files from unauthorized access or modification, such as file permissions or access control lists.

File structure

- File structure is a structure, which is according to a required format that operating system can understand.
- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.
- An object file is a sequence of bytes organized into blocks that are understandable by the machine.

File attributes

- Name - only information kept in human-readable form
- Identifier - unique tag (number) identifies file within file system
- Type - needed for systems that support different types
- Location - pointer to file location on device
- Size - current file size
- Protection - controls who can do reading, writing, executing
- Time, date, and user identification - data for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on the disk

File operations

- Create
- Write
- Read
- Delete
- Truncate

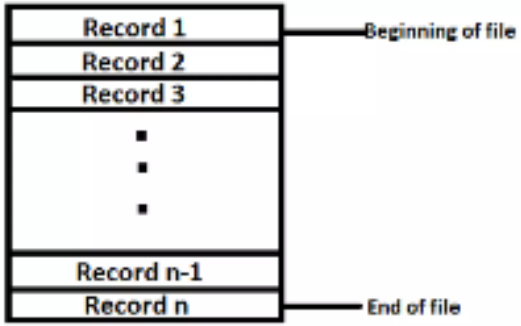
File Types

file type	usual extension	function
executable	exe, com, bin or none	read to run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm	binary file containing audio or A/V information

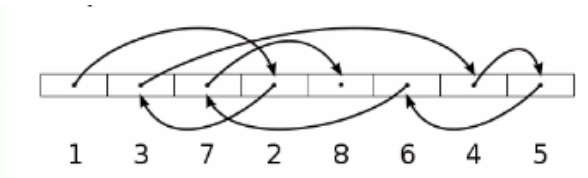
File access methods

- File access refers to the manner in which the records of a file may be accessed.
- There are several ways to access files
- Sequential access
- Direct/Random access
- Indexed sequential access

Sequential Access



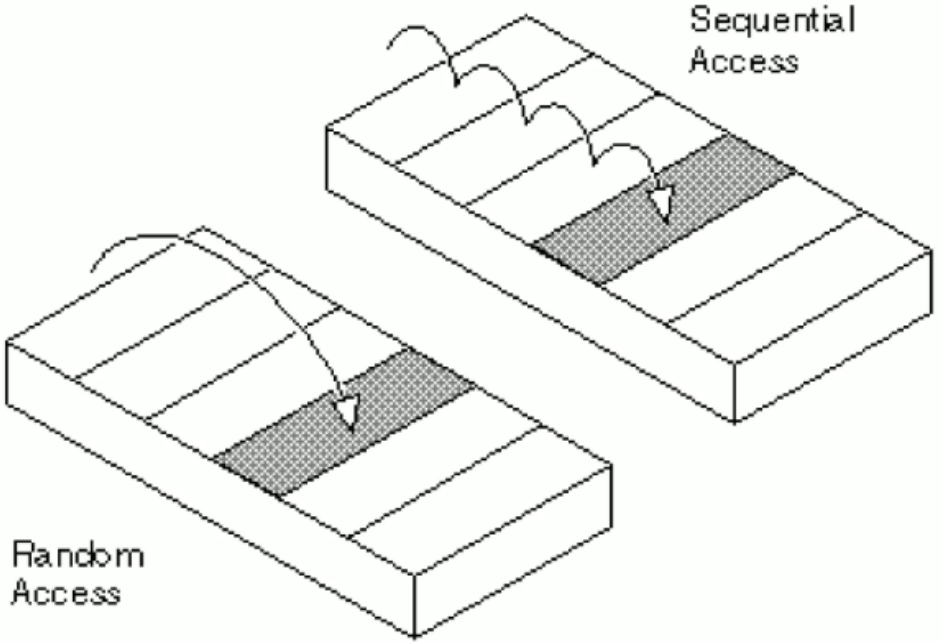
- A sequential access is that in which the records are accessed in some sequence i.e the information in the file is processed in order, one record after the other.



Direct/Random Access

- Random access file organization provides, accessing the records directly.
- Each record has its own address on the file with by the help ofwhich it can be directly accessed for reading or writing.
- The records need not be in any sequence within the file and theyneed not be in adjacent locations on the storage medium.

Sequential v/s Random Access



Indexed sequential access

- This approach combines the advantages of both sequential and direct file access.
- An index is created for each file which contains pointers to various blocks.
- Index is searched sequentially and its pointer is used to access the file directly.

Example

Example to understand Indexed Sequential Access

	KEY	KEY	KEY	KEY
Track 1	12	14	18	20
Track 2	26	34	41	60
Track 3	71	73	74	75
Track 4	77	78	79	82
Track 5	89	92	93	98

(Prime Area)

Track No.	Highest key
1	20
2	60
3	75
4	82
5	98

TRACK INDEX

Protection in file systems

- We want to keep our information safe from physical damage(reliability) and improper access(protection).
- We can protect our information by controlled access.
- Types of access:
- Read
- Write
- Execute
- Delete

Access control

- This is the most common approach.
- ALCs(access control lists) are maintained.
- The main problem of ALCs is their length.
- The technique has two undesirable consequences
  - Construction of list is tedious.
  - Size of the directory once defined cannot be changed

Protection through Password

- To associate password with each file.
- Use a separate password for each file. But it would be a tedious job for the user to remember all the passwords
- Solution to this is to choose same password but if the password is once known to someone else than all the data will be revealed.
- Hence, the security in this method is on all or nothing basis.