OPERATING SYSTEMS

ABOUT OS

- It is basically a software abstracting the hardware.
- It acts as an interface between the User and Hardware
- It simplifies coding and application development (aka APIs)
- It is a controller and delegates the processes and resources between hardware devices.

OS SERVICES

- User Interface
- Program execution
- I/O operation
- File – system manipulation
- Comms
- Error detection
- Resource Allocation
- Accounting
- Protection and Security

GOALS OF OS

- Convenience (User friendly)
- Efficiency
- Portability
- Reliability
- Scalability
- Robustness

TYPES OF OS

Uniprogramming OS

This allows only one process in the main memory at a time. This does not efficiently utilize the CPU and I/O.

Multiprogramming OS

This allows more than one process in the main memory at a time. This has better resources utilization. The number of processes in the main memory is referred to as the **Degree of multiprogramming.**

NOTE – The Degree of Multiprogramming for Uniprogramming OS = 1.

There are 2 types of Multiprogramming OS –

- **Non-Preemptive –** Once a process starts, then the process leaves the main memory by its own choice and no external event can stop the program and force it to free up the CPU resources.
- **Preemptive –** OS can forcefully take out the running program from the CPU if the need arises.

Multitasking OS (Preemptive)

This is an extension of the Multiprogramming OS where the processes run in a round – robin fashion. Each process gets some execution time and then the OS switches to execute a different process. So basically, the processes are executing in a linear round-robin manner. However, the switching is so fast that a user thinks that the OS is executing the processes in parallel. Therefore, this OS is also called **Pseudo – parallel OS** or **Time – Sharing OS.**

Multiprocessing OS

This is an OS used to control the systems with multiple CPUs. These are of two types –

- **Tightly Coupled –** In this case there are multiple CPUs that share the same main memory. This is also called a **Share Memory Multiprocessing OS.**
- **Loosely Coupled –** In this case there are multiple CPUs that each have their own memory element. This is also called a **Distributed Multiprocessing OS.**

This CPU design has a major advantage that it doesn't have a single point of failure.

Multiuser OS

These OS enable multiple users to work on the system at the same time. All Linux, Unix OS are Multiuser OS.

Real – time OS

It enables systems which work on real time data and events. In these systems, the events and processes have deadlines to maintain and are expected to finish their execution within their deadlines. Based on deadlines, we have two types of Real – time OS –

- **Hard time OS –** In this case, the deadlines of the processes are not malleable and strict af.
- **Soft time OS –** In this case, the deadlines have some leniency.

Embedded OS

It is the OS used in Embedded systems. These systems are embedded into physical machines to make them “smart”. In this OS, the user interaction is very minimal and the functionality of these systems are very specific.

Handheld OS

These are OS developed for handheld devices like mobiles, tabs, watches etc.

PARTS OF OS

There are two main parts of OS –

- **Kernel –** The bundle of features and programs the OS provides.
- **Shell –** This the way for the user to give the command to the kernel. There are two types of shells –
    - Command Prompt
    - Graphic User Interface (GUI)

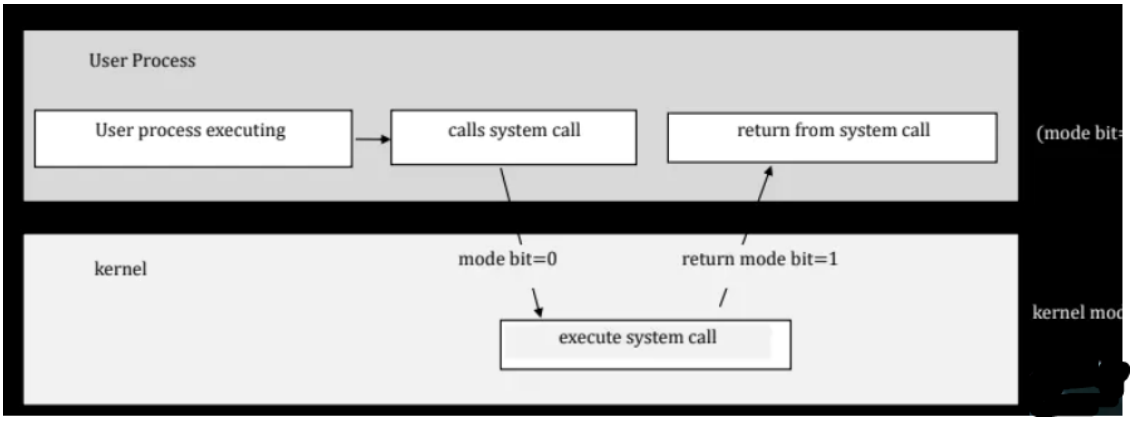
SYSTEM CALLS

A system call is a way for the programs to interact with the OS.

DUAL OPERATION MODES

The OS functions in two different operation modes –

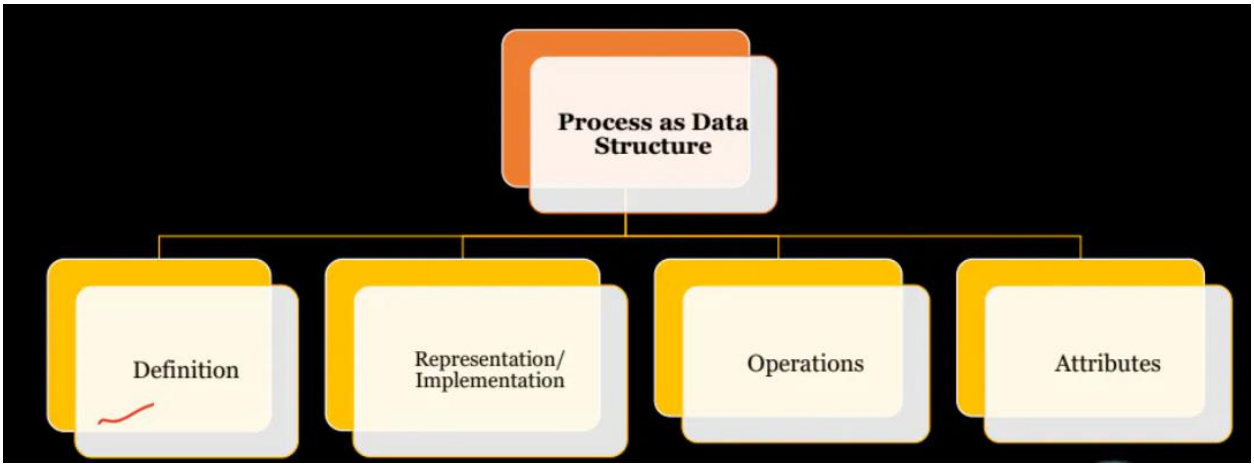
- **User Mode –** In this mode, the user can execute functions. However, the privileged operations can’t be executed here. Mode bit = 1.
- **Kernel/System/Supervisor/Privileged Mode –** In this mode, the kernel is allowed to execute privileged operations. Mode bit = 0.



PROCESS

When a program goes for execution, an additional **run – time entities** and values are added. This is when the program becomes a **process.**

$$Process = Program + Run - time\ entities$$



Definition of a Process

This explains the functions of the process, the inputs it takes and the outputs it provides.

Representation of a Process

Every process has the following four parts to it –

- **Code or text section –** This is where the program and instructions are stored.
- **Data Section –** This is where the global and static variables that are going to be used are stored
- **Heap Section –** This is used for dynamic memory allocation
- **Stack –** This is used to store dynamic or local variables and return addresses for function calls. This is also referred to as **Activation Record.**

Operations of a Process

- Create
- Schedule
- Run
- Wait/Block
- Suspend
- Resume
- Terminate

Attributes of a Process

- PID
- Program Counter
- Gen Purpose Registers
- List of devices
- Type
- Size
- Memory Limit
- Priority
- State
- List of files

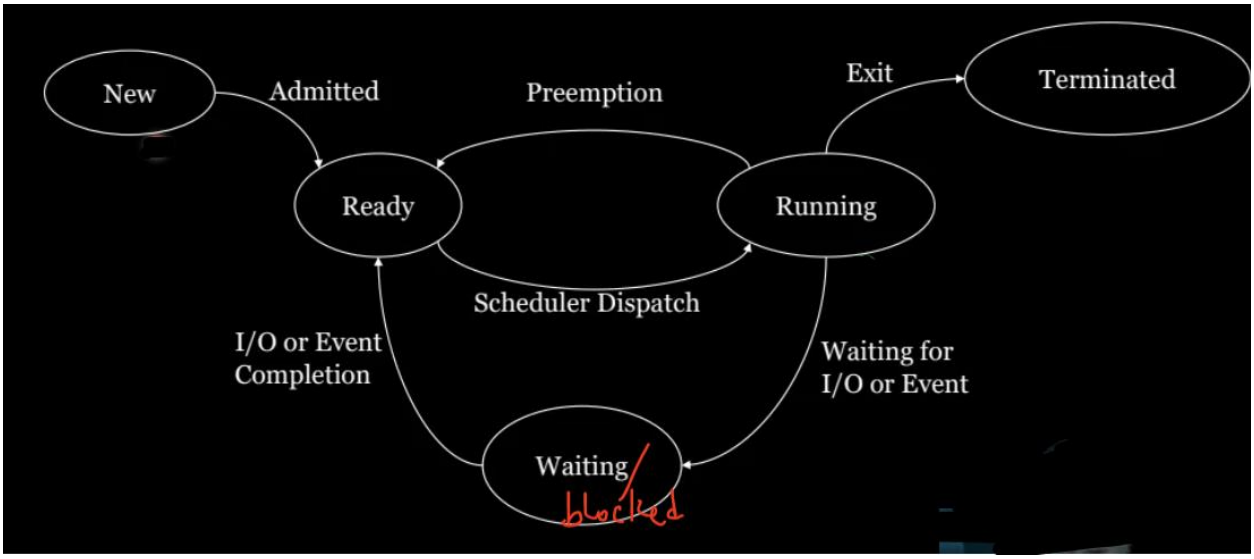
All these information is stored in a **Process Control block (PCB).** The values inside the PCB define the **context of the process.** The PCB is also very helpful when it comes to multiple processes running at the same time. Suppose we have two processes P1 and P2. P1 starts executing and the PCB values of P1 are sent to the CPU. In between, P2 wants to start running. In that case, the PCB values from the CPU for P1 are updated in the OS and P2 starts running. Once P2 finishes, the updated PCB values for P1 are sent to the CPU to continue the execution from where it was left off. This is called **context switch.**

$$Context\ Switch = Store\ PCB\ values\ of\ current\ process + Get\ PCB\ values\ of\ next\ process$$

This is done by a unit called **Dispatcher.** The time taken by the dispatcher to perform a context switch is called **Context Switch Time.**

$$Context\ Switch\ time \quad \propto \quad PCB\ Size$$

PROCESS STATES



All processes are in **New** state by default in the secondary memory. Once a process begins, it is first sent to the CPU and resources are allocated for it. This happened during **admission** and the process goes into

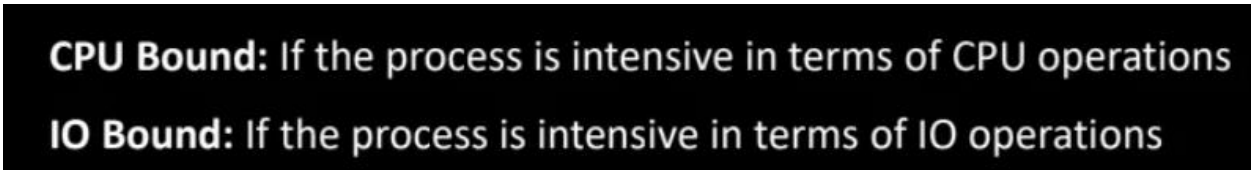
Ready state. Now, the Dispatcher will switch the context and change the state to **Running** when the process starts executing.

Once the process finishes execution, the state is changed to **Terminated** and during the **exit phase,** the process releases the resources and this is called **resourced de-allocation.**

In some cases, the process execution is often paused as the process needs input from I/O operations. When such an even occurs, the process is sent to **Waiting/Blocked** state and when the I/O input is provided, the process moves to the **Ready** state.

Suppose there is a case where the process is in Running state. In this state, there can be a case where for some reason the process is preempted out. In such a case, the process will return back to **Ready** state. Also, this goes without saying that this occurs only in Preemptive Multiprogramming systems.

When the process is in **Ready, Running or Waiting** states, the process is present in the main memory.



PROCESS SCHEDULING

Since there are a large number of processes in the system, there is need for scheduling to improve performance and optimization.

SCHEDULING QUEUE

There are three main queues when it comes to scheduling –

- **Job Queue –** This is where all the processes in the **New** state are stored.
- **Ready Queue –** This is where all the processes in the **Ready** state are stored
- **Device Queue –** This is a unique queue which is used by I/O devices. Each I/O device has a separate device queue.

TYPES OF SCHEDULERS

Job Scheduler

This is responsible for bringing the processes from New state to Ready state. This is also called **Long – term scheduler.** Since this scheduler controls how many processes enter the Ready state aka main memory, this scheduler can **Increase the degree of multiprogramming.**

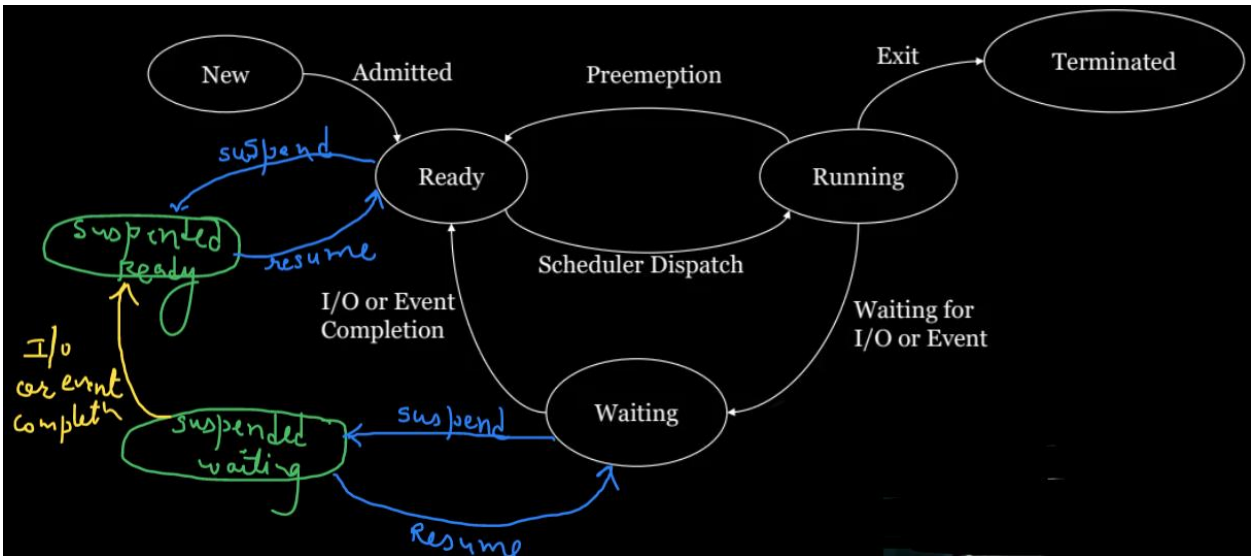
Short – Term Scheduler

This scheduler decides what process should run next. This is called Short – term scheduler since it is used more frequently when compared to Job Scheduler. This is also called **CPU Scheduler.**

Mid – Term Scheduler

There can be a case where the main memory gets filled up due to large number of processes being present at the same time. In these cases, the Mid – term scheduler will take the LRU process and send it back to the hard disk so as to create space in the main memory for the next process. This is called **swapping.**

If the processes are swapped out based on **priority,** then the process is called **Rolling.** Now, if we include the swapping and rolling processes, the flow chart becomes something like this –



SCHEDULER	DEGREE OF MULTI-PROGRAMMING
Short term	No effect
Long term	Can only increase
Mid term	Can either increase or decrease

NOTE – The process which is I/O bound should be scheduled first in the CPU for better CPU utilization. This is because the I/O bound process will spend a very little time in the CPU and then go into the blocked state to wait for the I/O device's response. In this time, the other processes can use the CPU resources and finish execution.

QUESTIONS

In operating system, each process has its own?

a) Address space and global variables

b) Open files

c) Resources to be used

d) All of the mentioned

A process can be terminated due to?

a) normal exit

b) fatal error

c) killed by another process

d) all of the mentioned

Consider a system with n processes and m CPUs. Maximum and Minimum number of processes in each of the following states possible?

		max	min
1.	Ready state	n	0
2.	Running State	m	0
3.	Blocked State	n	0

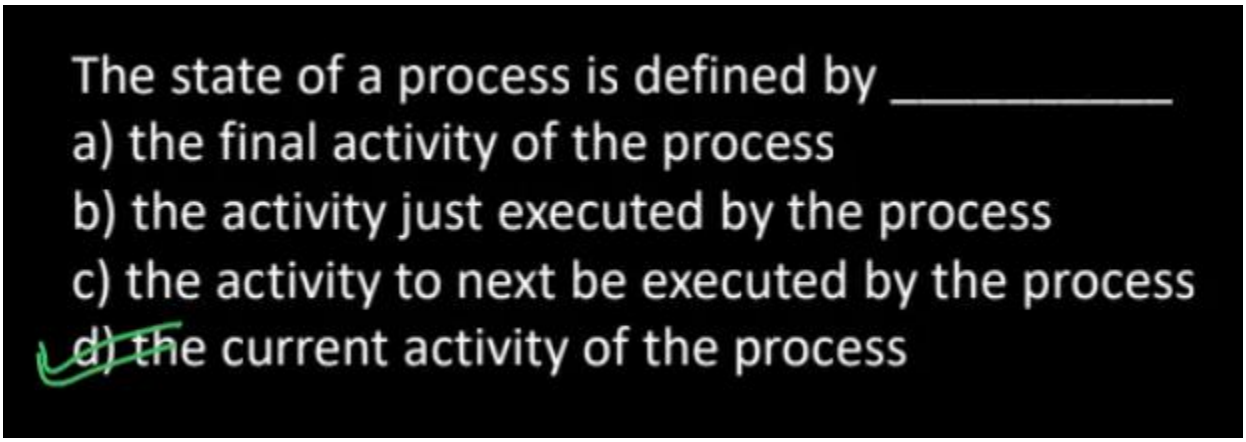
A Process Control Block(PCB) does not contain which of the following?

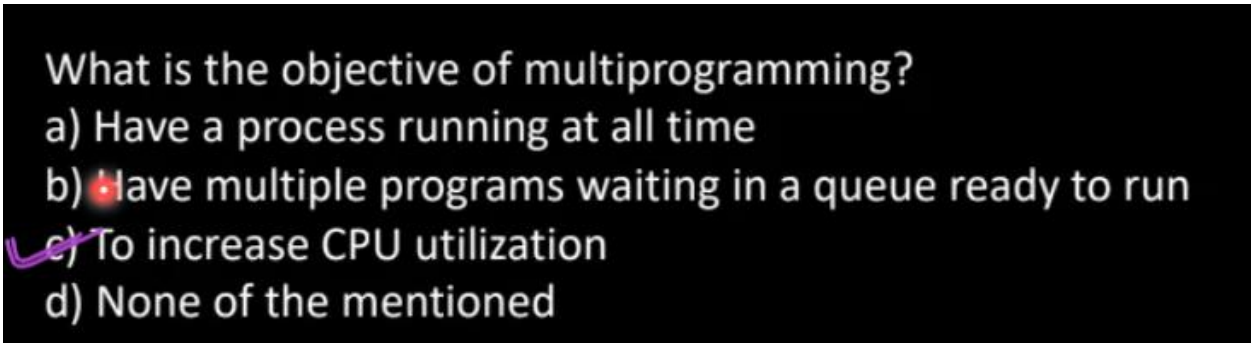
a) Code

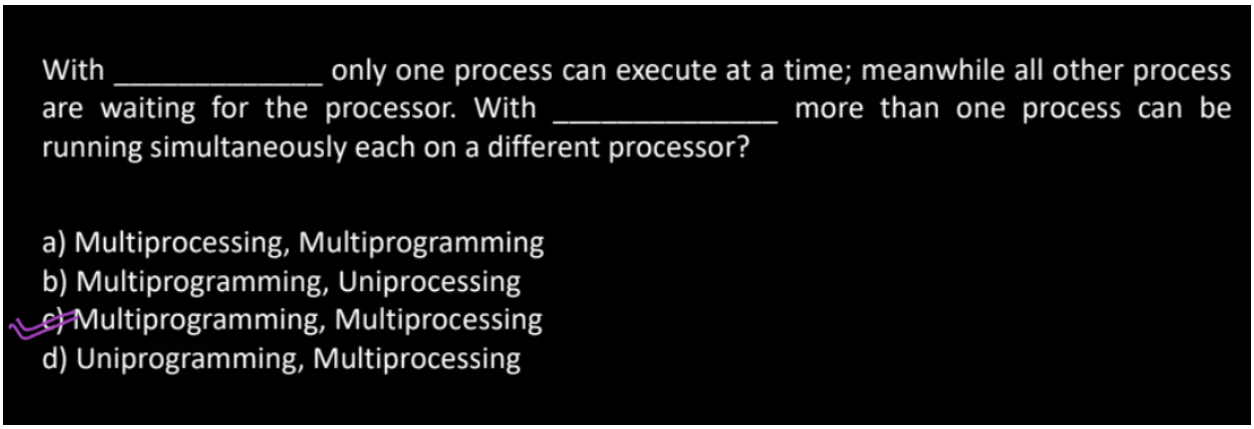
b) Stack

c) Bootstrap program

d) Data







CPU SCHEDULING

This is the scheduler which decides the process to run next so as to maximize the throughput and minimize the waiting time. There are two major types of scheduling algorithms –

- **Non-Preemptive –** If a process is currently running in the CPU, the next process can only be sent once the current process finishes execution.
- **Preemptive –** The current running process can be paused and replaced by another process irrespective of the fact that the current process has finished execution or not.

CPU Scheduling goals

- Minimize wait time
- Minimize Turn – around time (TAT)
- Maximize Throughput (CPU utilization)
- Improve fairness

Scheduling Times

- **Arrival Time –** Time instant at which the process was admitted to the CPU.
- **Burst Time –** The time taken by the process to execute.
- **Waiting Time –** The amount of time the process remains in the Ready state waiting for CPU resources.
- **Completion Time –** Time instant at which process is terminated.
- **Response Time –** Time taken by the process to respond and start executing. In other words, it is the time different between Arrival time and the Beginning of the Burst time.
- **Turn Around Time (TAT) –** Total time the process spends in the system.
- **Scheduling Length –** This is the maximum process TAT in the current list of processes.
- **Throughput –** The number of processes per unit time.

$$TAT = Completion\ Time - Arrival\ Time = Waiting\ Time + Burst\ Time$$

$$Scheduling\ length = \max(Completion\ time) - \min(Arrival\ Time)$$

CPU SCHEDULING PROCESSES

FIRST COME FIRST SERVE (FCFS)

The process with the minimum arrival time will be scheduled first as it came in the CPU first (hence the name). This is a **non-preemptive process.** Now, there can be a case where multiple processes have the same arrival time. In such cases, the process with the smallest ID will be scheduled first.

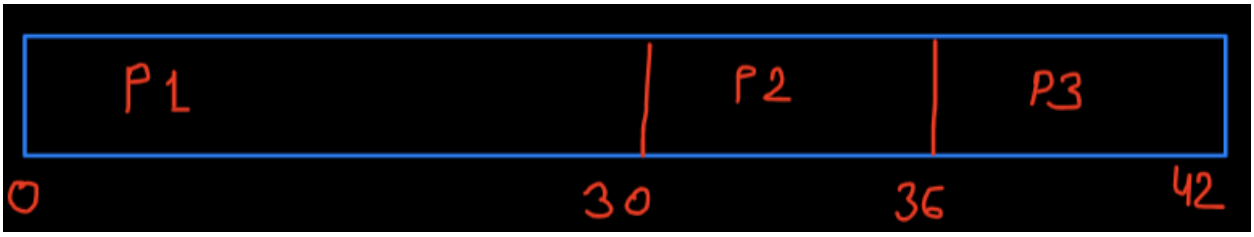
Question

How will the CPU with FCFS process will schedule the processes as shown follow?

Process	Arrival Time	Burst Time
P1	0	30
P2	0	6
P3	0	6

Answer

Since the arrival times of the processes are same, we schedule based on the Process IDs. Hence, P1 executes first, then P2 and finally P3. The Gantt Charge for the same is shown below –



Now, we have –

METRIC	PROCESS P1	PROCESS P2	PROCESS P3
Arrival Time	0	0	0
Burst Time	30	6	6
Completion Time	30	36	42
TAT	30	36	42
Waiting Time	0	30	36
Response Time	0	30	36

Avg Waiting time = Avg Response Time = 22 units

Avg Completion Time = Avg TAT time = 36 units

Scheduling Length = 42 – 0 = 42 units

Question

Process	Arrival Time	Burst Time
P1	0	5
P2	1	1
P3	2	2
P4	3	4
P5	4	5
P6	5	3

Answer

METRIC	P1	P2	P3	P4	P5	P6
Arrival Time	0	1	2	3	4	5
Burst Time	5	1	2	4	5	3
Completion Time	5	6	8	12	17	20
TAT	5	5	6	9	13	15

Waiting Time	0	4	4	5	8	12
Response Time	0	4	4	5	8	12

Avg Waiting time = Avg Response Time = 33/6

Avg TAT = 53/6

L = 20 – 0 = 20

Throughput = 6/20

Question

Process	Arrival Time	Burst Time
P1	5	4
P2	8	2
P3	6	3
P4	3	1
P5	2	2
P6	7	7

Answer

METRIC	P1	P2	P3	P4	P5	P6
Arrival Time	5	8	6	3	2	7
Burst Time	4	2	3	1	2	7
Completion Time	9	21	12	5	4	19
TAT	4	13	6	2	2	12
Waiting Time	0	11	3	1	0	5
Response Time	0	11	3	1	0	5

Avg Waiting time = Avg Response Time = 20/6

Avg TAT = 39/6

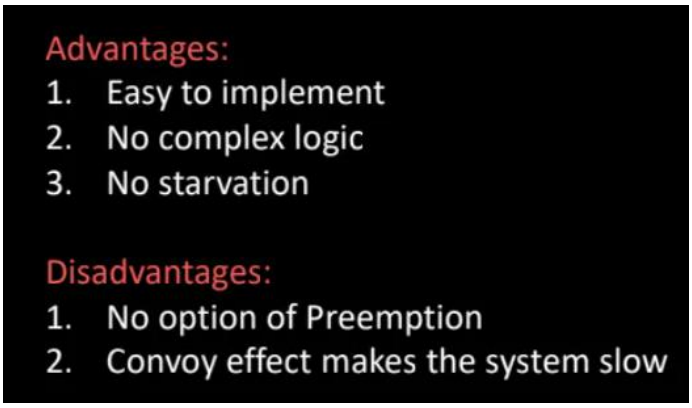
L = 21 – 2 = 19

Throughput = 6/19

Convoy Effect

In the above FCFS scheduling algorithm, the process with the smaller arrival time is scheduled first. There can be a case where a large process comes first and is therefore scheduled first. In such a case, the other

smaller processes will have to wait for a large amount of time to start execution and in turn the CPU performance is affected. This is the **Conway Effect** and it occurs **only in FCFS scheduling process.**



SHORTEST JOB FIRST (SJF)

In this case, **the job with the smallest Burst time is scheduled first**. This helps the system to overcome the Convoy Effect. Additionally, if there are multiple processes with the **same burst time, then FCFS is applied** among those processes.

Question

How will the CPU with SJF process will schedule the processes as shown follow?

Process	Arrival Time	Burst Time
P1	0	30
P2	0	6
P3	0	6

Answer

Here, the order of execution is – P2, P3 and P1.

Process	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	30	42	42	12
P2	0	6	6	6	0
P3	0	6	12	12	6

Avg Waiting Time = 6

Avg TAT = 20

Question

How will the CPU with SJF process will schedule the processes as shown follow?

Process	Arrival Time	Burst Time
P1	0	6
P2	1	3
P3	2	4
P4	4	2
P5	5	1
P6	6	5
P7	8	2

Answer

METRIC	P1	P2	P3	P4	P5	P6	P7
Arrival Time	0	1	2	4	5	6	8
Burst Time	6	3	4	2	1	5	2
Completion Time	6	14	18	9	7	23	11
TAT	6	13	16	5	2	17	3
Waiting Time	0	10	12	3	1	12	1
Response Time	0	10	12	3	1	12	1

Avg Waiting time = Avg Response Time = 39/7

Avg TAT = 62/7

L = 23 – 0 = 23

Throughput = 7/23

Starvation

In SJF, there can be a case where multiple smaller processes keep coming in and will keep getting scheduled first. This will prevent a larger process from getting the CPU resources and therefore it will keep waiting for a long time. This is called **Starvation.** SJF is vulnerable to starvation but **FCFS is not** as we are not favoring any process in FCFS**.**

Advantages:
1. Minimum average waiting time among non-preemptive scheduling
2. Better throughput in continuous execution

Disadvantages:
1. No practical implementation because Burst time is not known in advance
2. No option of Preemption
3. Longer Processes may suffer from starvation

SHORTEST REMAINING TIME FIRST (SRTF)

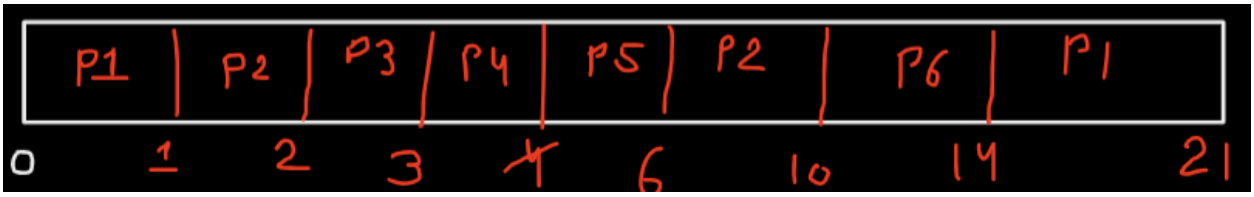
This is basically a preemptive version of SJF. So, the criteria is still Burst Time itself but this time we can preempt the processes as well.

Question

Process	Arrival Time	Burst Time
P1	0	8
P2	1	5
P3	2	1
P4	3	2
P5	4	1
P6	5	4

Answer

For this case, the Gantt Chart looks like this –



METRIC	P1	P2	P3	P4	P5	P6
Arrival Time	0	1	2	3	4	5
Burst Time	8	5	1	2	1	4
Completion Time	21	10	3	5	6	14
TAT	21	9	1	2	2	9
Waiting Time	13	4	0	0	1	5
Response Time	0	0	0	0	1	5

Avg TAT = 44/6

Avg Waiting Time = 23/6

Avg Response Time = 1

NOTE – As we know, Response Time is the time difference between the arrival time and the time instant it starts execution. In non – preemptive algorithms, if a process starts running, it will only relinquish the

CPU resources once it is done. Therefore, the **waiting and response time for non-preemptive processes are the same.**

On the other hand, if we include preemptive algorithms, then a single process can be preempted midway through its execution and therefore, the waiting time increases and need not be equal to response time.

NOTE – Among all the scheduling algorithms, SRTF gives the **minimum avg. waiting time.** If we are just considering the non – preemptive algorithms, SJF gives the **minimum avg. waiting time.**

Question

Process	Arrival Time	Burst Time
P1	4	7
P2	5	5
P3	3	1
P4	1	2
P5	2	1
P6	0	4

Answer

METRIC	P1	P2	P3	P4	P5	P6
Arrival Time	4	5	3	1	2	0
Burst Time	7	5	1	2	1	4
Completion Time	20	13	5	3	4	8
TAT	16	8	2	2	2	8
Waiting Time	9	3	1	0	1	4
Response Time	9	3	1	0	1	0

Avg TAT = 38/6

Avg Waiting Time = 18/6

Avg Response Time = 14/6

Advantages:
1. Minimum average waiting time among all scheduling algorithm
2. Better throughput in continue run

Disadvantages:
1. No practical implementation because Burst time is not known in advance
2. Longer Processes may suffer from starvation

HIGHEST RESPONSE RATIO NEXT (HRRN)

In the SRTF algorithm, there is still a possibility for starvation of the larger processes. To overcome that, HRRN was introduced. This algorithm schedules the processes based on **Response Ratio.** Here,

$$Response\ Ratio(R) = \frac{Waiting\ time(W) + Burst\ Time(S)}{Burst\ Time(S)}$$

We can also infer,

$$R \propto W \qquad and \qquad R \propto \frac{1}{S}$$

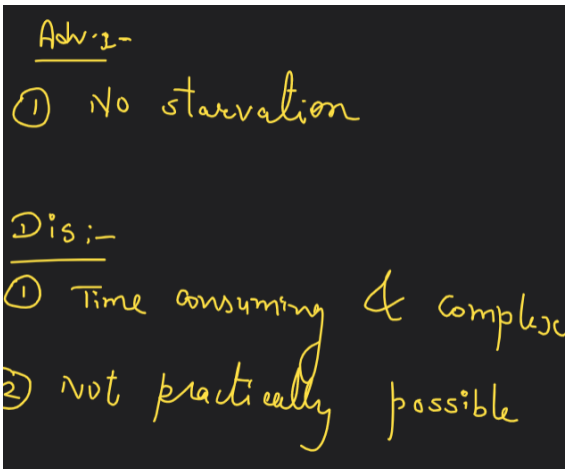
This algorithm is only for **Non-Preemptive Processes.** In case two processes have the same HRRN, then schedule based on SJF.

Question

Process	Arrival Time	Burst Time
P1	0	3
P2	2	6
P3	4	4
P4	6	5
P5	8	2

Answer

TIME INSTANT	PROCESS BEING EXECUTED
1	P1 (only process)
2	
3	
4	P2 (only process)
5	
6	
7	
8	
9	
10	P3 as $HRRN = \frac{5+4}{4} = \frac{9}{4}$ which is the highest
11	
12	
13	
14	P5 as $HRRN = \frac{5+2}{2} = 3.5$ which is the highest
15	
16	P4 (only remaining process)
17	
18	
19	
20	



PRIORITY BASED ALGORITHM

In this case, the question mentions the priority of the processes and the process with the highest priority is scheduled first. This can be either preemptive or non-preemptive. There can be two types of priorities –

- **Static –** Remains constant
- **Dynamic –** Subject to change

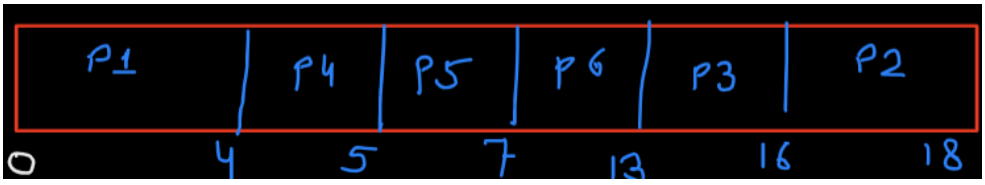
Question

Use non-preemptive and preemptive priority scheduling.

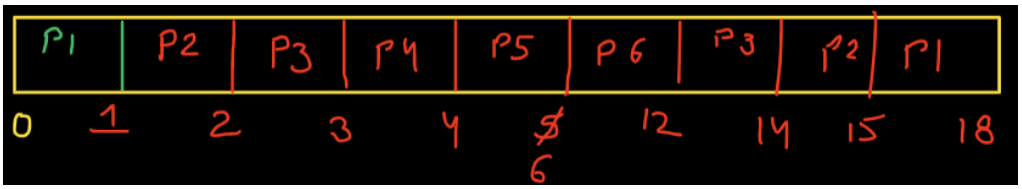
Process	Arrival Time	Burst Time	Priority
P1	0	4	4
P2	1	2	5
P3	2	3	6
P4	3	1	10(Highest)
P5	4	2	9
P6	5	6	7

Answer

For Non-preemptive scheduling –



For Preemptive Scheduling –



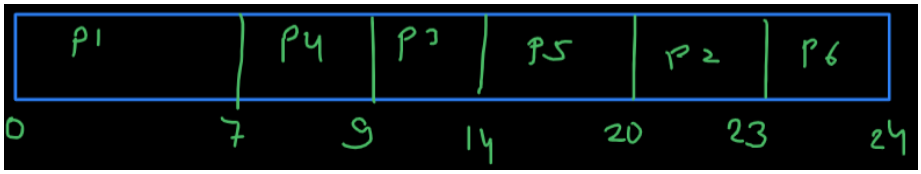
Question

How will the processes be scheduled for non-preemptive and preemptive priority scheduler?

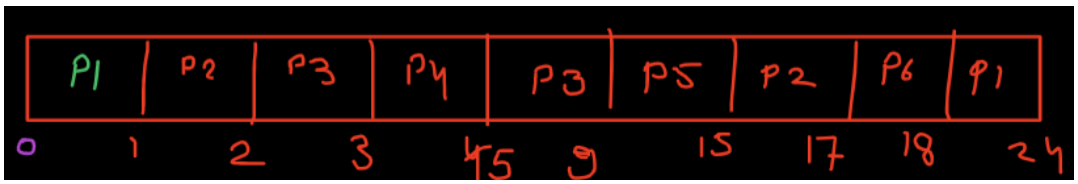
Process	Arrival Time	Burst Time	Priority
P1	0	7	9
P2	1	3	4
P3	2	5	2
P4	3	2	1 (Highest)
P5	4	6	3
P6	5	1	8

Answer

For Non-preemptive scheduler –



For Preemptive Scheduler –



Advantages:

1. Better response for real time situations

Disadvantages:

1. Low Priority Processes may suffer from starvation

NOTE – In a dynamic priority scheduling algorithm, there is a solution for starvation called **ageing.** This means that as the waiting time increases of the process, then the priority also increases which means that after a certain waiting time, the process has the highest priority and is scheduled next. Since priority doesn't change in static priority scheduling, we can't have ageing in static priority scheduling.

ROUND ROBIN (RR)

In this case, the scheduling is done based on arrival time and **Time Quantum (q).** Time quantum is basically the maximum slice of processor for which a process can run for at one time. This means that even if the process runs for q time units and is not completed, it will be preempted and the next process is scheduled.

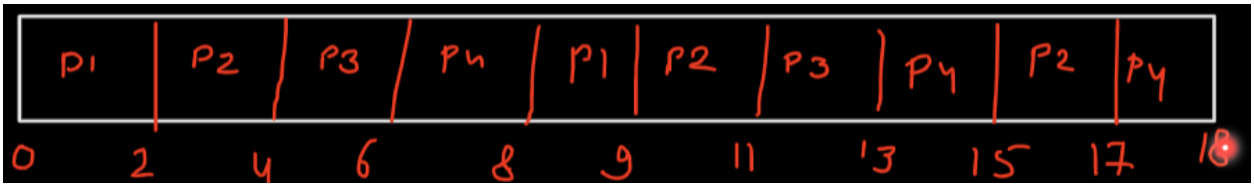
Since there is preemption, this is a **preemptive scheduling algorithm.**

Question

Schedule the following processes using RR method with q = 2.

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

Answer



METRIC	P1	P2	P3	P4
Arrival Time	0	0	0	0
Burst Time	3	6	4	5
Completion Time	9	17	13	18
TAT	9	17	13	18
Waiting Time	6	11	9	13
Response Time	0	2	4	6

Avg TAT = 57/4

Avg Waiting Time = 39/4

Avg Response Time = 3

L = 18 – 0 = 18

No of context switches = 9

NOTE – There is an easier way to find the number of context switches without drawing the Gantt Chart. First, we need to calculate the total number of quanta required to finish the processes –

$$Q_{total} = \left(\sum_{i=1}^{4} ceil\left(\frac{Burst\ time\ of\ Pi}{q} \right) \right) = ceil(1.5) + ceil(3) + ceil(2) + ceil(2.5) = 10$$

Now, since the CPU is clear in the beginning, there is no context switch for the first process. Hence, we get –

$$No\ of\ context\ switches = Q_{total} - 1 = 10 - 1 = \mathbf{9}$$

Question

Schedule the following processes using RR method with q = 2.

Process	Arrival Time	Burst Time
P1	0	6
P2	1	5
P3	2	4
P4	3	3
P5	4	2
P6	5	4

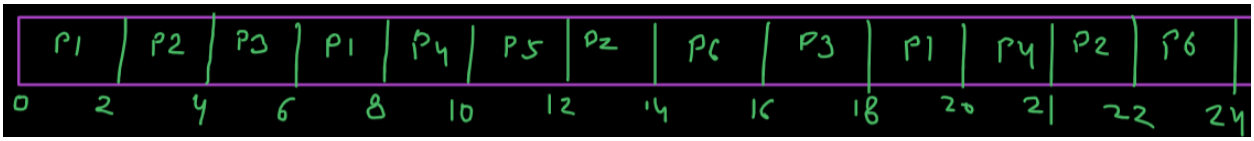
Answer

Here, the arrival times are different and so we need to be very careful how the processes are ordered. This can be simplified if we maintain a ready queue for each time instant –

TIME INSTANT	READY QUEUE	PROCESS EXECUTING	REMAINING BURST TIMES					
			P1	P2	P3	P4	P5	P6
0	P1	P1	5	5	4	3	2	4
1	P2	P1	4	5	4	3	2	4
2	P3, P1	P2	4	4	4	3	2	4
3	P3, P1, P4	P2	4	3	4	3	2	4
4	P1, P4, P5, P2	P3	4	3	3	3	2	4
5	P1, P4, P5, P2, P6	P3	4	3	2	3	2	4

6	P4, P5, P2, P6, P3	P1	3	3	2	3	2	4
7	P4, P5, P2, P6, P3	P1	2	3	2	3	2	4
8	P5, P2, P6, P3, P1	P4	2	3	2	2	2	4
9	P5, P2, P6, P3, P1	P4	2	3	2	1	2	4
10	P2, P6, P3, P1, P4	P5	2	3	2	1	1	4
11	P2, P6, P3, P1, P4	P5	2	3	2	1	0	4
12	P6, P3, P1, P4	P2	2	2	2	1	0	4
13	P6, P3, P1, P4	P2	2	1	2	1	0	4
14	P3, P1, P4, P2	P6	2	1	2	1	0	3
15	P3, P1, P4, P2	P6	2	1	2	1	0	2
16	P1, P4, P2, P6	P3	2	1	1	1	0	2
17	P1, P4, P2, P6	P3	2	1	0	1	0	2
18	P4, P2, P6	P1	1	1	0	1	0	2
19	P4, P2, P6	P1	0	1	0	1	0	2
20	P2, P6	P4	0	1	0	0	0	2
21	P6	P2	0	0	0	0	0	2
22		P6	0	0	0	0	0	1
23		P6	0	0	0	0	0	0

So now, we can create a Gantt chart from this table –



METRIC	P1	P2	P3	P4	P5	P6
Arrival Time	0	1	2	3	4	5
Burst Time	6	5	4	3	2	4
Completion Time	20	22	18	21	12	24
TAT	20	21	16	18	8	19
Waiting Time	14	16	12	15	6	15
Response Time	0	1	2	5	6	9

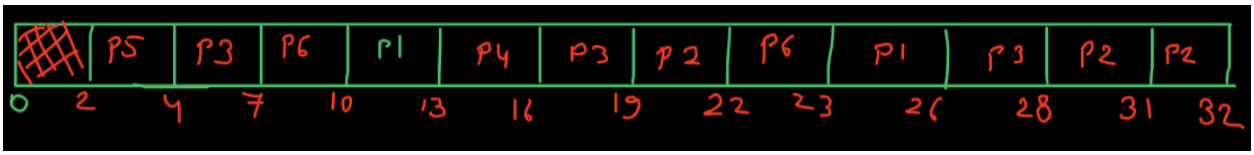
Question

Schedule the following processes using RR method with q = 3.

Process	Arrival Time	Burst Time
P1	5	6
P2	8	7
P3	3	8
P4	6	3
P5	2	2
P6	4	4

Answer

Applying the method above, we can write the following Gantt chart –



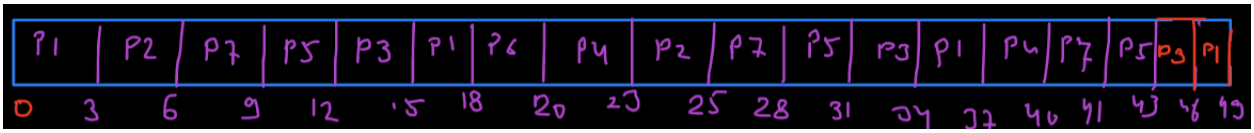
Question

Schedule the following processes using RR method with q = 3.

Process	Arrival Time	Burst Time
P1	0	12
P2	0	5
P3	3	9
P4	5	6
P5	2	8
P6	4	2
P7	1	7

Answer

Applying the method above, we can write the following Gantt chart –



Advantages:
1. All processes execute one by one, so no starvation
2. Better interactiveness
3. Burst time is not required to be known in advance

Disadvantages:
1. Average waiting time and turnaround time is more
2. Can degrade to FCFS

NOTE – In the RR scheduling, we have always assumed that the context switching time between processes (s) is negligible compared to the value of quanta (q). However, there can be case where q and s can be comparable. In such cases, the CPU efficiency is given as –

$$CPU\ efficiency = \frac{q}{q+s}$$

THROUGHPUT

It is defined as the average number of processes executed in unit time. When we have a set number of processes, then all the scheduling algorithms have the same throughput. However, if we have a continuous flow of processes coming in, then the scheduling algorithm which executes the smallest jobs first give the best throughput. Hence, for continuous process streams, **SJF** or **SRTF** will have better throughput.

NOTE

SRTF scheduling algorithm is a preemptive algorithm. However, suppose we have the following cases –

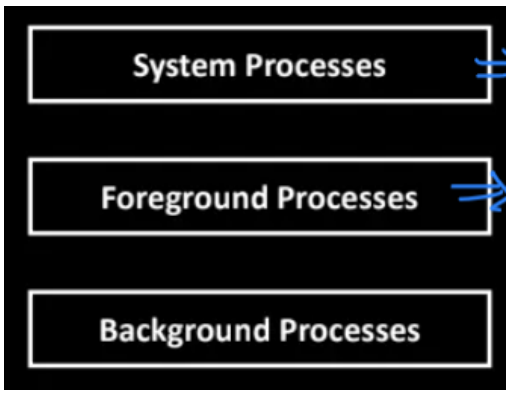
- If all the processes arrive at the same time.
- If the processes arriving later have a larger burst time.

In these 2 cases, there will be **no preemption** in SRTF. Hence, for these cases, SRTF works as a **non-preemptive** algorithm.

Similarly, in Round Robin scheduling, if the value of quantum is larger than the burst times of the processes, then the processes will finish execution before any preemption takes place. Hence, in this case the Round Robin algorithm degrades to a **First – come – First – Serve (FCFS).**

MULTILEVEL QUEUE SCHEDULING (MLQ)

This is the type of scheduling where we can use multiple scheduling algorithms at the same time. So, to enable this, the processes are divided into different groups aka **queues.** Then, for each queue we apply a different kind of scheduling algorithm as required. For example, we can divide the ready queue into three main queues –



In this scenario, we can see that each queue will implement a different scheduling algorithm and now we will have 3 processes to be executed – one from each queue. But out of these 3, which one to execute first? For this scenario, we have two choices –

1. **Fixed Priority Preemptive Scheduling –** In this case, the process that belongs to the higher queue gets executed first. So, in our example, the system processes will be executed before foreground processes which will be executed before the background processes. Also, this is a Preemptive algorithm.
2. **Time Slicing –** In the Fixed Priority algorithm, it is possible that there are so many system processes that the other processes don't get executed in turn causing **starvation.** To fix that, we use Time Slicing. It is similar to Round Robin as in that each of the 3 processes queues get some time slice to execute. This allows processes from each queue to get executed in batches and reduce starvation.

MULTI-LEVEL FEEDBACK QUEUE SCHEDULING (MLFQ)

In this case, the priorities of the queues and processes are dynamic i.e. if the process is waiting in the queue for long time, its priority is increased to ensure it gets scheduled sooner.

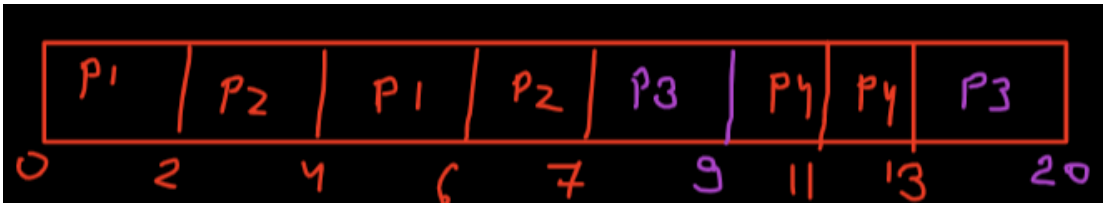
Question

Use MLQ Fixed Priority Preemptive scheduling

Queue 1: RR with Q=2
Queue 2: FCFS

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

Answer



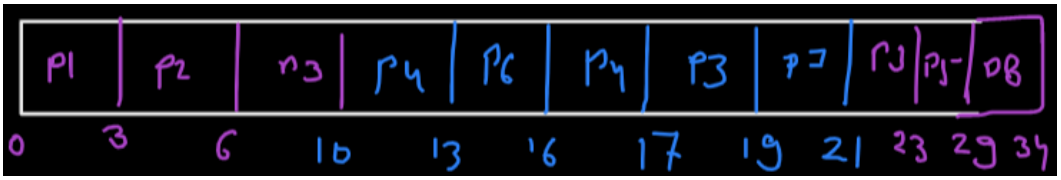
Question

Use MLQ Fixed Priority Preemptive scheduling

Queue 1: RR with Q=3
Queue 2: FCFS

Process	Arrival Time	Burst Time	Queue
P1	0	3	1
P2	0	3	1
P3	2	8	2
P4	10	4	1
P5	11	6	2
P6	11	3	1
P7	19	2	1
P8	13	5	2

Answer



Question

Consider a process scenario in which each process executes first in CPU then goes for IO operation, then once again process needs a CPU bursts and then terminates. Following is given a process scenario in which for CPU execution system uses non preemptive SJF algorithm. Consider system has enough number of resources to carry out IO operations for all processes in parallel. What is the average waiting time for the execution for the processes?

PROCESS	ARRIVAL TIME	CPU BURST 1	I/O BURST TIME	CPU BURST 2
P1	0	3	9	4
P2	0	6	4	7
P3	0	4	1	2
P4	0	5	3	1

Answer

PROCESS	AT	CT	TAT	WT
P1	0	18	18	2
P2	0	36	36	19
P3	0	14	14	7
P4	0	19	19	10

Here,

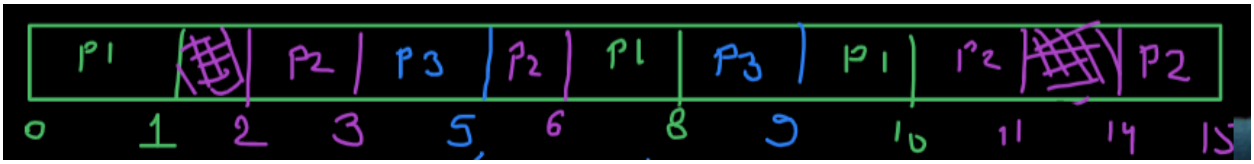
$$WT = TAT - (CPU\ Burst\ 1 + IO\ Burst + CPU\ Burst\ 2)$$

Question

The arrival time, priority, and duration of the CPU and I/O bursts for each of three processes P1,P2 and P3 are given in the table below. Each process has a CPU burst followed by an I/O burst followed by another CPU burst. Assume that each process has its own I/O resource. The multi-programmed operating system uses preemptive priority scheduling. What are the finish times of the processes P1, P2 and P3 ?

Process	Arrival Time	Priority	CPU, IO, CPU Bursts
P1	0	2	1, 5, 3
P2	2	3 (Lowest)	3, 3, 1
P3	3	1 (Highest)	2, 3, 1

Answer



PROCESS	AT	CT	TAT	WT
P1	0	10	10	1
P2	2	15	13	6
P3	3	9	6	0

Question

If the waiting for a process is p for IO and there are n processes in the memory, then the CPU utilization is?

Answer

If there are no IO operations, then the CPU utilization will be **1.** Suppose, we have 1 process which spends $\boldsymbol{p}$ fraction of time in IO. Then,

$$CPU\ utilization = 1 - p$$

Now, assume that there are 2 processes using IO – each spending p fraction of time in IO. Then,

$$CPU\ utilization = 1 - p^2$$

Therefore, if we extend the same principle for n processes, we get –

$$\boldsymbol{CPU\ utilization = 1 - p^n}$$

MULTITHREADING

A **thread** is a component of a process. Basically, each process is made up of multiple threads. Having multiple threads to make up a process enables resource sharing among the threads and helps improve performance.

Shared Among Threads	Unique For Each Thread
Code Section	Thread Id
Data Section	Register Set
OS Resources	Stack
Open Files & Signals	Program Counter

TYPES OF THREADS

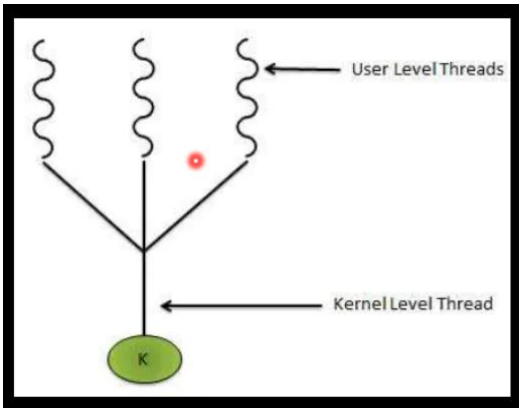
- **Kernel Level Threading –** When the multithreading has been done for OS processes

- **User Level Threading –** When the multithreading has been done for User processes

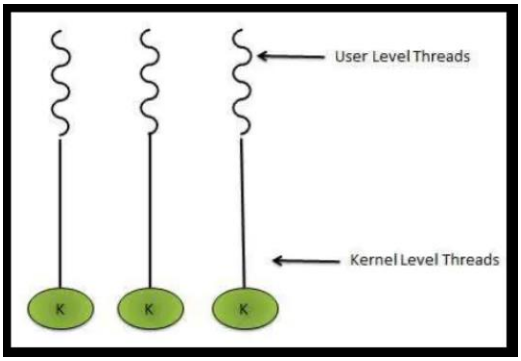
User Threads	Kernel Thread
Multithreading in user process	Multithreading in kernel process
Created without kernel intervention	Kernel itself is multithreaded
Context switch is very fast	Context switch is slow
If one thread is blocked, OS blocks entire process	Individual thread can be blocked
Generic and can run on any OS	Specific to OS
Faster to create and manage	Slower to create and manage

MULTITHREADING MODELS

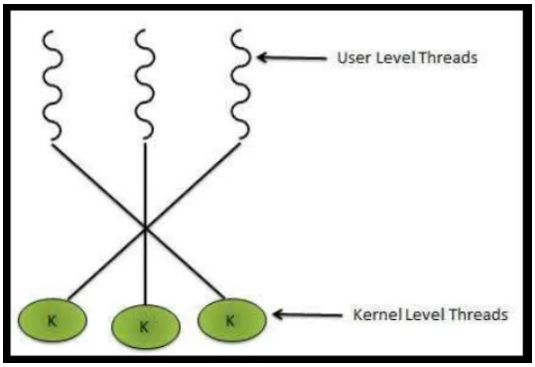
Many-To-One Model



One-To-One Model

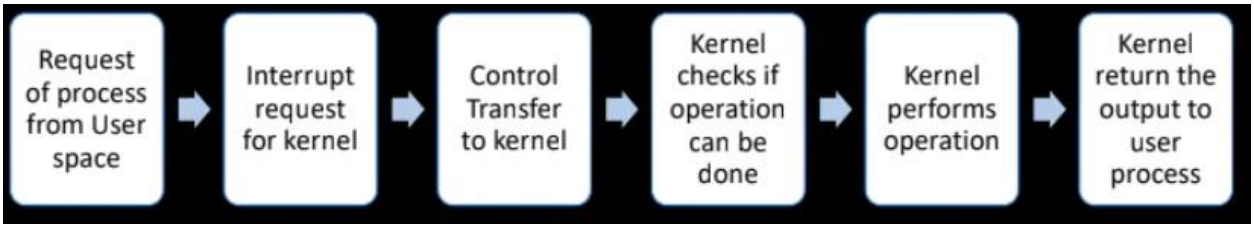


Many-To-Many Model



SYSTEM CALL

It is a programmatic way in which the computer program requests a service from the kernel.



FORK()

Fork() is a system call used to create a child process which runs concurrently with the parent process.

It takes no parameters and returns an integer value

- Negative Value: creation of a child process was unsuccessful
- Zero: Returned to the newly created child process
- Positive value: Returned to parent or caller. The value contains process ID of newly created child process

Question

A process executes the code

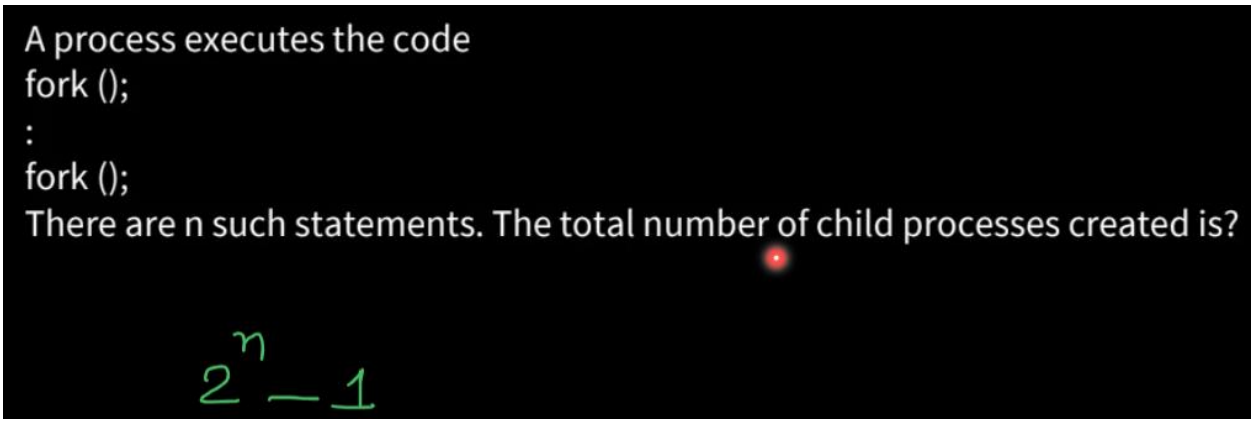
```
fork ();
fork ();
fork ();
```

The total number of child processes created is

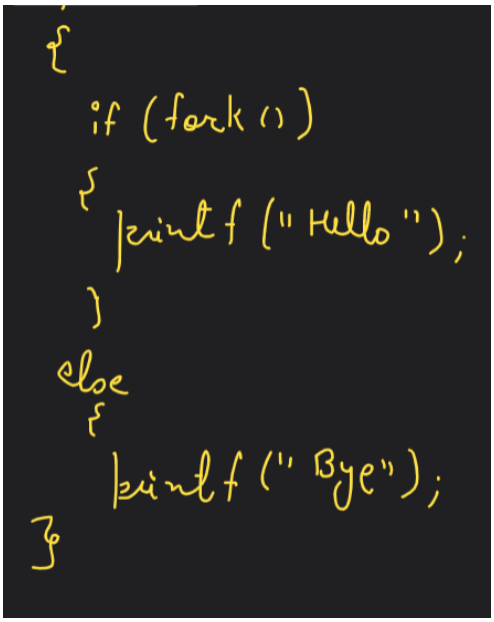
(A) 3 (C) 7

(B) 4 (D) 8

Question



Question



Answer

When we have fork(), the value returned is 1 for Parent process and 0 for child process. Therefore, we get the output as –

HelloBye

PROCESS TYPES BASED ON COMMUNICATION

- **Independent –** These are the processes that don't communicate with any other processes.

- **Cooperative/Coordinated/Communicating –** These are the processes that communicate with the other processes. Due to this communication, these processes can affect execution of other processes and can also be affected by other processes.

SYNCHRONIZATION

This is needed from Communicating Processes. It is the way for the communicating processes to still provide the required/expected result even though the process can be affected by any other process. If Synchronization is not performed, then we can have the following problems –

- Inconsistency
- Data loss
- Deadlock

In a process, normally only a few instructions require communication and hence need synchronization. The rest of the instructions will not be communicating and hence won't require any synchronization. The section of the process involved with communication is called a **Critical Section.**

WAYS FOR COMMUNICATION

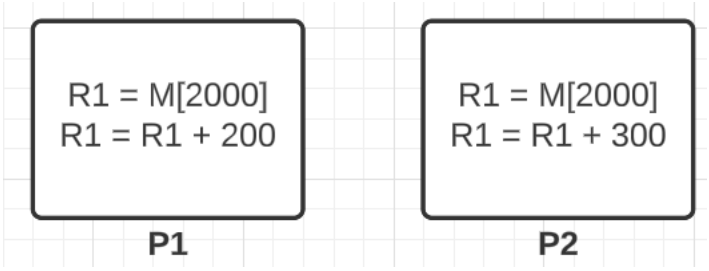
- **Message Passing –** Messages, data or variables are sent from one process to another.
- **Shared resource –** Hardware or software resources will be shared between 2 processes and this resource acts as a communicating medium.

RACE CONDITION

This is an undesired situation where the final result of the concurrent processes will depend on the sequence in which the processes complete their execution. Suppose, we have memory location 2000 populated with value 100.

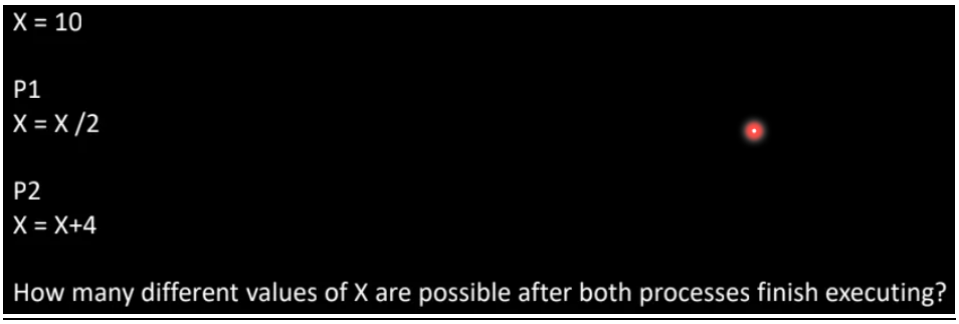
$$M[2000] = 100$$

Let there be 2 processes as follows –



If P1 finishes before P2, then the value in R1 will be **400.** However, if P1 finishes after P2, then the value in R1 will be **300.** This is a classic example of Race condition as the two processes are sharing a resource.

Question



Answer

There can be 4 cases –

CONDITION	RESULT
P1 finishes first, writes back X and then P2 finishes executing	X = 9
P2 finishes first, writes back X and then P1 finishes executing	X = 7
P1 starts first but P2 starts before P1 finishes	X = 14
P2 starts first but P1 starts before P2 finishes	X = 5

SOLUTIONS OF CRITICAL SECTION PROBLEM

For a **perfect solution,** the following 3 conditions need to be satisfied –

- **Mutual Exclusion –** If one process is executing in the critical section, then the other process should not be allowed to execute in the same critical section.
- **Progress –** If there is no process executing in the critical section and at least 1 process is waiting and ready to use the critical section, then the process should be allowed (progress) to use the critical section.
- **Bounded Waiting –** If a process is executing in critical section and there is a process waiting for the critical section, then the waiting time should be bounded i.e. the first process should not be allowed back in the critical section before the waiting process is allowed to execute in the critical section.

We have designed the solution to work when there are 2 processes trying to execute the critical section and hence the following solutions are called **2 – Process Solutions.** However, before proceeding further we need to understand that every solution is a **Software solution** and hence is basically a code snippet. There are three main components of the solution code –

- **Entry Section –** This is the part of the code that ensures Mutual Exclusion, Progress and Bounded Waiting is implemented.
- **Critical Section**
- **Exit Section –** This is the last part of the code that acts as an announcement that the process has finished the execution of the critical section.

SOLUTION 1

For this solution, the process code is given as follows –

```
bool lock = FALSE;
while (TRUE)
{
    //Entry section
    while (lock);
    lock = TRUE;

    //Critical section
    CS

    //Exit section
    lock = FALSE;

    //Remaining section
    RS
}
```

The bool variable $\boldsymbol{lock}$ is set to False by default. When the variable lock is set to False, it indicates that the **Critical section is free.** On the other hand, if lock is True, then it indicates that the **Critical section is NOT free.** Let us assume that initially lock is True. So, the code snippet control will reach the **Entry section** and the while loop will keep on iterating. As soon as the lock becomes False, the Entry section while loop breaks and the control moves forward. In this case, lock is set to True by the process so that **no other process can use the critical section.** Then, the code executes the critical section. After it is done, it enters the **Exit section** where lock is set back to False indicating to the other processes that the **critical section is available.** Once the critical section is executed, the remaining part of the process can be executed.

Given this solution, the next step would be to check if the Entry section satisfies the 3 pre-requisites. To do that, we need to label the statements –

```
//PROCESS 1
while (lock); //Statement S1
lock = TRUE; // Statement S2
```

```
//PROCESS 2
while (lock); //Statement S3
lock = TRUE; // Statement S4
```

In a normal execution, these statements can be executed in any order as long as S2 executes after S1 and S4 executes after S3. Let us take the following case –

$$S1 \rightarrow S3 \rightarrow S2 \rightarrow S4$$

In this case,

- First, Process 1 will execute S1. Since lock is False to begin with, the while loop breaks and the control moves forward.
- Then, Process 2 will execute S3 and similar to S1, the while loop will break and the control moves forward.
- Next, Process 1 will execute S2 and set the lock to True.

- Finally, S4 will be executed by process 2 and it will also set the lock to True.

TL;DR – Both the processes will pass the entry stage and set the lock to True. Therefore, both the processes are accessing the critical section at the same time and thus **mutual exclusion is not followed in this case.**

Now, we can start checking for **Progress.** To check progress, we take the initial case where the Critical section is free and then we check whether any of the processes can access the critical section. So initially, lock is False indicating that the critical section is free. Now, either Process 1 or Process 2 can enter the critical section and set the lock as True. Hence, **the solution satisfies the progress condition.**

Finally, we check the **Bounded waiting** time. To do so, we first start with a case where one process is using the critical section while the other is waiting for it. We will assume that Process P1 is using the critical section and Process P2 is waiting. Now, once the critical section is over, Process P1 sets the lock value to False. However, since it is in an infinite loop, Process 1 will again reach the entry section and therefore will again access the Critical section. Therefore, **Process 2 ends up waiting as Process 1 repeatedly accesses the critical section.** Therefore, there is **No Bounded waiting.**

Therefore, **Solution 1 is NOT a Perfect Solution.**

SOLUTION 2

For this solution, the process codes will be given as follows –

```
int turn = 0;
```

```
//PROCESS 1
while (TRUE)
{
    while (turn != 0); //Statement S1

    //Critical section
    CS

    //Exit section
    turn = 1

    //Remaining section
    RS
}
```

```
//PROCESS 2
while (TRUE)
{
    while (turn != 1); //Statement S2

    //Critical section
    CS

    //Exit section
    turn = 0

    //Remaining section
    RS
}
```

Check for Mutual Exclusion

First, we assume S1 executes first and then S2. In that case, S1 returns False as turn is initially set to 0. With that, Process 1 enters the critical section. However, Statement S2 returns True and hence, Process 2 doesn't enter the Critical section. Similarly, if we assume S2 executes first then Process 2 doesn't enter the critical section and then S1 executes and Process 1 enters the critical section.

Since the Entry section of the 2 processes are **mutually exclusive conditions,** it ensures that this solution satisfies **mutual exclusion condition.**

Check for Progress

For this, we assume there are no processes accessing the critical section. Now, if Process P1 starts executing, it will end up in the Critical section as turn = 0. On the other hand, turn = 0 will prevent Process P2 from entering. So, if the critical section is initially free, Process 1 can enter to access it but Process 2 can't. Hence, there is **NO Progress.**

Check for Bounded Waiting

We assume that Process 1 is currently accessing the Critical Section and Process 2 is waiting. Now, when the critical section finishes execution, Process 1 sets turn to 1. This prevents the Process 1 to re-enter the critical section but allows Process 2 to enter. Now, Process 2 is using the critical section and Process 1 is waiting. Once Process 2 finishes execution, then it sets turn to 0 and therefore, Process 1 enters the critical section and Process 2 can't re-enter the critical section. Since, no process can re-enter the critical section after execution, this solution **satisfies Bounded Waiting.**

Since this solution doesn't satisfy the Progress condition, this solution is **NOT a Perfect Solution.**

SOLUTION 3 – PETERSON'S SOLUTION

In this case, we define 2 variables –

```
int turn;
bool flag[2] = {FALSE, FALSE};
```

In the flag array, the first element is the value for Process 1 and second element is the value for Process 2. When the value is FALSE, then it means that the process is not accessing the critical section. The code for the 2 processes are shown below –

```
while (TRUE)
{
    //Entry section
    flag[0] = TRUE;
    turn = 1;
    while (flag[1] && turn == 1);

    //Critical section
    C.S.

    //Exit section
    flag[0] = FALSE;
}
```

Process 1

```
while (TRUE)
{
    //Entry section
    flag[1] = TRUE;
    turn = 0;
    while (flag[0] && turn == 0);

    //Critical section
    C.S.

    //Exit section
    flag[1] = FALSE;
}
```

Process 2

Check for Mutual Exclusion

Let us re – label the entry section statements as follows –

```
//PROCESS 1
while (TRUE)
{
    flag[0] = TRUE; //Statement S1
    turn = 1 //Statement S2
    while (flag[1] && turn == 1) //Statement S3
}
```

```
//PROCESS 2
while (TRUE)
{
    flag[1] = TRUE; //Statement S4
    turn = 0 //Statement S5
    while (flag[0] && turn == 0) //Statement S6
}
```

To check for mutual exclusion, the we take the various combinations of the statements S1 – S6. However, no matter the combination, the variable **turn** will be set to either 0 or 1 and hence will allow either S3 or S6 to return True/False. Basically, there is no scenario in which **Statement S3 and S6 will both return True or False.** Hence, this solution **satisfies mutual exclusion condition.**

Check for Progress

Let us assume that the no process is accessing the critical section. In this case, we can either have Process 1 or Process 2 access and execute in the critical section. Hence, any process can access the critical section of the critical section is free. Hence, this solution **has Progress.**

Check for Bounded Waiting

Let us assume that Process 1 is executing in the critical section and Process 2 is waiting to execute in the critical section. So, Process 1 sets the following values –

```
flag[0] = True

turn = 1
```

Now, Process 2 starts executing and it sets the following value –

```
flag[1] = True

turn = 0
```

Since flag[0] is True and turn is 0, Process 2 is stuck in the infinite while loop aka it is waiting for the critical section to become free. Now, let's assume Process 1 finishes execution and sets flag[0] to False. Now, Process 1 tries to re-enter the critical section and sets the following values –

```
flag[0] = True

turn = 1
```

When Process 1 enters the while loop, flag[1] is True (thanks to Process 2) and turn is 1. Hence, Process 1 is stuck in the infinite while loop and hence is waiting to enter the critical section. On the other hand, Process 2 has flag[0] as True but turn is 1 and hence the code breaks out of the while loop and enters the critical section. Therefore, now Process 1 is waiting for critical section while Process 2 is executing in the critical section.

Thus, no process can re – enter the critical section consecutively and so, this solution **satisfies the Bounded Waiting condition.**

Therefore, **this is a PERFECT SOLUTION.**

HARDWARE SOLUTIONS

As seen, the above 3 solutions were all software solutions. However, we have 2 **CPU functions** that are part of the Hardware solution to help with synchronization –

- TestAndSet()
- Swap()

TestAndSet()

Let us take the case of the first software solution code –

```
bool lock = FALSE;
while (TRUE)
{
    //Entry section
    while (lock);
    lock = TRUE;

    //Critical section
    CS

    //Exit section
    lock = FALSE;

    //Remaining section
    RS
}
```

Using TestAndSet, we replace the entire entry section as follows –

```
bool lock = FALSE;

bool TestAndSet(bool *temp)
{
    bool rv = *temp;
    *temp = TRUE;
    return rv;
}
```

```
//PROCESS
while (TRUE)
{
    //Entry section
    while (TestAndSet(&lock));

    //Critical section
    CS;

    //Exit section
    lock = FALSE;
}
```

One thing to note is that **TestAndSet** is **NOT a function** but it is a **single CPU instruction.** This means that it can't be preempted.

The TestAndSet first reads the lock variable and stores it in the local variable **rv.** Then, the function sets the value of lock to True. Initially, the lock is set to False. When the TestAndSet instruction is called, it takes the lock value as False and sends that to variable rv. After that, the function returns rv which is False but also sets the lock value as True thus indicating to the other processes that the current process is executing the critical section. Once the critical section is done, the exit section changes the lock back to False.

The advantage of using this is that the **Entry section of the process is ATOMIC** and hence, between two processes there is **mutual exclusion.** We can also see that if the critical section is free (lock = FALSE), then any process can access it and set the lock to TRUE to prevent any other process from entering the critical section. Therefore, **progress condition is also met.** Finally, we can see that if a process P1 is executing critical section, then the other process will be waiting. After the execution, the lock is set to False. At this time, the other process can also enter the critical section and even the process P1 can re-enter the critical section. Therefore, **there is NO Bounded Waiting.**

Swap()

```
bool lock = FALSE;

void Swap(bool *a, bool *b)
{
    bool temp = *a;
    *a = *b;
    *b = temp;
}
```

```
//PROCESS
bool key = FALSE;
while (TRUE)
{
    //Entry section
    key = TRUE;
    while (key == TRUE)
        Swap(&lock, &key);

    //Critical section
    CS;

    //Exit section
    lock = FALSE;
}
```

Swap() is also a CPU instruction and hence **can't be preempted.** Here, lock is a global variable while key is a local variable. Now, we check for the three conditions –

Mutual Exclusion

First, Process P1 executes. It sets the local variable key to True. After this, the while loop satisfies and hence, the Swap instruction is executed. Once the Swap is done, the lock will now be TRUE while the key will become FALSE. Thus, the process breaks out of the while loop and enters the critical section.

Now, let us consider the entry section of another process P2. The locak variable key for P2 is set to True and the while condition passes. Then, the process performs the swap. However, Process P1 had already performed the swap and lock now has True value. Therefore, when Process P2 performs Swap, it is swapping key which is True with lock which is True. Hence, Process P2 will keep waiting and not enter the critical section.

In short, the process that performs Swap first will enter the critical section and the other processes (even if they perform the swap) will not be able to enter the critical section. Thus, **mutual exclusion is satisfied.**

Progress

This is very intuitive as we can see of the critical section is free i.e. lock is False, then any of the processes can enter the critical section after performing Swap. Hence, the **progress condition is satisfied.**

Bounded Waiting

Let us say that process P1 has finished executing the critical section. At this point, the lock is set back to False. If the Process P1 is re-run, it again sets the key value to True, performs the swap and enters the critical section again. Since this re-entry is possible, the **Bounded waiting condition is NOT satisfied.**

BUSY WAITING

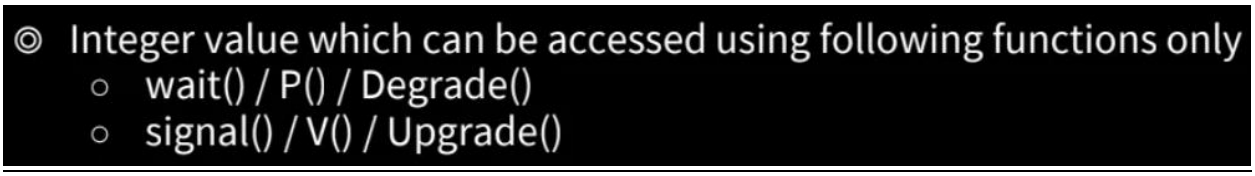
In both the software and hardware solutions, if a Process P1 is executing the critical section, another process P2 is waiting. However, waiting means that process P2 is stuck in an indefinite while loop. This means that even though P2 is waiting, it is still taking up CPU time for while loop execution. This situation is called **Busy Waiting.**

SYNCHRONIZATION TOOLS

After looking at the Software and hardware solutions, we now look at Synchronization tools –

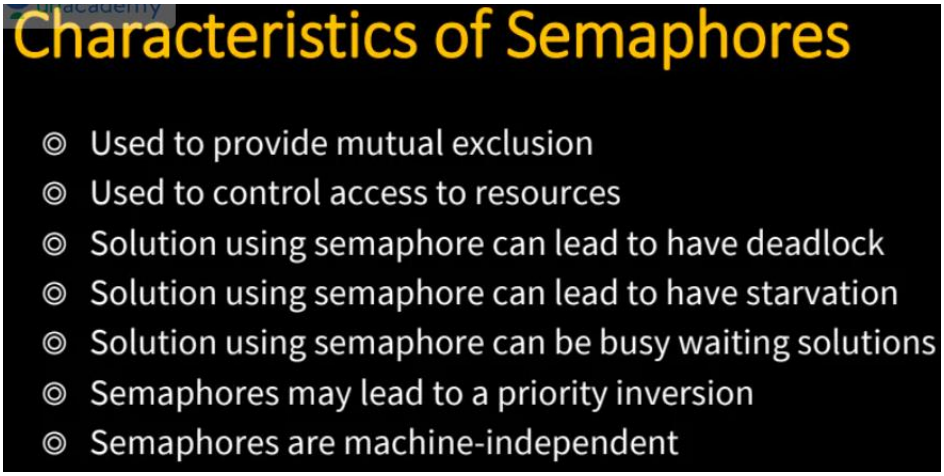
- Semaphores
- Monitor

SEMAPHORES

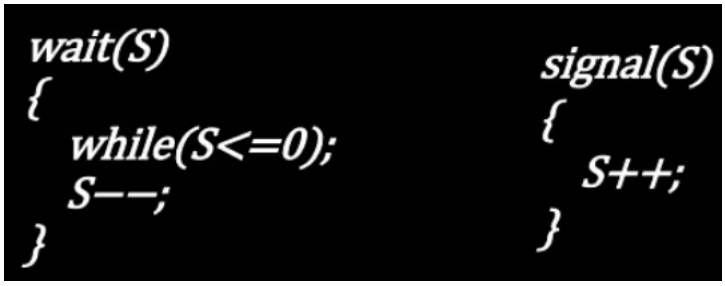


The ***wait()*** function decrements the semaphore value while the ***signal()*** function increments the semaphore value. If the semaphore value reaches zero, then wait() function will not run as it can't be further decremented. Semaphores can be of two types –

- **Binary –** These can take the value of either 0 or 1. These are used to solve the critical section problems like mutual exclusion, bounded waiting etc.
- **Counting –** These can take any integer value and are used to control the access of a resource that has multiple instances.



The logic for the wait() and signal() functions can be given as –



As we can see, in the wait() function, there is an indefinite while loop hence making semaphore solutions as **busy waiting solutions.**

CRITICAL SECTION SOLUTION USING SEMAPHORES

The code for a process will be as follows –

```
int S = 1;
```

```
//PROCESS 1
while (TRUE)
{
    //Entry section
    wait(S);
    //Critical section
    CS;
    //Exit section
    signal(S);
}
```

```
//PROCESS 2
while (TRUE)
{
    //Entry section
    wait(S);
    //Critical section
    CS;
    //Exit section
    signal(S);
}
```

Let us perform the three checks here –

Mutual Exclusion

First process 1 starts execution and the wait() function decrements the value of S to 0. After that, it starts the critical section. Once that happens, Process 2 will also try to execute. However, since S = 0, the wait() function can't execute and therefore Process 2 doesn't enter the critical section. Thus, **mutual exclusion is achieved.**

Progress

Let us assume that the critical section is free. In that case, the value of S = 1. Therefore, any process can start executing, execute the wait() function and start executing in the critical section. Therefore, **progress condition is satisfied.**

Bounded Waiting

Let us assume that process 1 is executing in the critical section. In that case, the value of S is 0. Once the process is done executing, the value of S is reverted back to 1 by the signal() function. Once done, it is quite evident that process 1 can re – enter the execution and critical section by re – running the wait() function. Thus, the **bounded waiting condition is NOT satisfied.**

NOTE

The above solution is for a **binary semaphore.** In case it was a **counting semaphore** with S = 10, then **10 processes** can access the critical section at a time.

Question

Consider a counting semaphore S, initialized with value 10. What should be the value of S after executing 6 times P() and 8 time V() function on S?

Answer

As we know, ***P()*** is the wait() function and it decrements the semaphore while ***V()*** is the signal() function which increments the semaphore. Hence,

$$S_{final} = 10 - 6 + 8 = \mathbf{12}$$

Question

Consider a semaphore S, initialized with value 37. Which of the following options gives the final value of S=12?

a) Execution of 22 P() and 15 V()

b) Execution of 25 P()

c) Execution of 33 P() and 8 V()

d) Execution of 31 P() and 6 V()

Answer

Option b), c) and d) are correct answers.

Question

Some processes want to run **52 P()** and then **23 V().** Out of this, it is also needed that **25 P()** operations get stuck. In this case, what should be the initial value of the semaphore?

Answer

Since we want to run 52 P() operations but at the same time want 25 P() operations to fail, then the total number of P() operations to be run = 52 – 25 = **27 P()** operations. Hence,

$$S - 27 + 23 = 0$$

$$\boldsymbol{S = 4}$$

Question

Consider a semaphore S, initialized with value 1. Consider 10 processes P1, P2 …. P10. All processes have same code as given below but, one process P10 has signal(S) in place of wait(S). If all processes to be executed only once, then maximum number of processes which can be in critical section together ?

```
process
{
  wait(S)
      C.S.
  signal(s)
}
```

Answer

As per the question, the process codes are given as follows –

P1, P2, ….., P9

```
process
{
  wait(S)
     C.S.
  signal(s)
}
```

P10

```
process
{
  signal(S)
     C.S.
  signal(s)
}
```

Hence, the following sequence is possible –

- P1 starts operation and sets S = 0
- P10 starts operation and sets S = 1
- P2 starts operation and sets S = 0

After this, no other process can be executed in the critical section as the wait() function will not return anything and it is given in the question that the operations execute only once. Hence, at one time a maximum of **3 processes can access the critical section.**

Question

Solve the same question above if the process code is as follows –

```
while(True)
{
 wait(S)
     C.S.
  signal(s)
}
```

Answer

In this case, there is an infinite loop so the processes can run infinite number of times. Thus, we can have **all 10 processes access the critical section** since we can have P10 run alternatively to increment the semaphore.

Question

Assume there are 2 processes – P1 and P2. The requirement is to have P1 and P2 fire alternatively starting with P2. What is the minimum number of semaphores needed?

Answer

Since we need to have the 2 processes fire alternatively, P1's exit section should enable P2's entry section and vice – versa. Therefore, we can write the programs as –

```
int S1 = 0;
int S2 = 1;
```

```
//PROCESS 1
while (TRUE)
{
    wait(S1);
    CS;
    signal(S2);
}
```

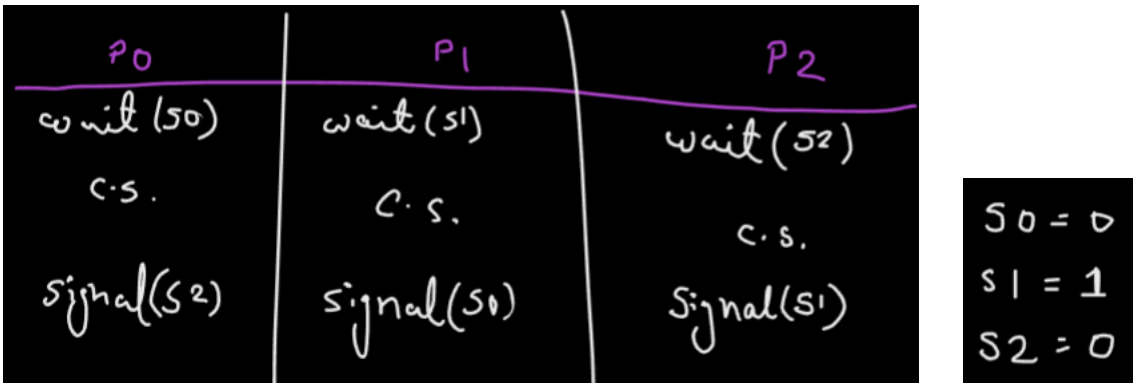
```
//PROCESS 2
while (TRUE)
{
    wait(S2);
    CS;
    signal(S1);
}
```

As per the above code, Process 2 will fire first and then it will be followed by Process 1. Then, the processes will alternate as expected. Hence, we need a minimum of **2 semaphores.**

Question

Consider a scenario where 3 processes are available: P0, P1 and P2. The processes must be executed in order P1, P0 and P2 only. Write a piece of code to control the sequence of execution with minimum number of semaphores?

Answer



Question

Consider the following threads, T_1, T_2, and T_3 executing on a single processor, synchronized using three binary semaphore variables, S_1, S_2, and S_3, operated upon using standard **wait**() and **signal**(). The threads can be context switched in any order and at any time.

T_1	T_2	T_3
while(true){	while(true){	while(true){
wait(S_3);	wait(S_1);	wait(S_2);
print("C");	print("B");	print("A");
signal(S_2); }	signal(S_3); }	signal(S_1); }

Which initialization of the semaphores would print the sequence BCABCABCA . . .?

A. $S_1 = 1; S_2 = 1; S_3 = 1$
B. $S_1 = 1; S_2 = 1; S_3 = 0$
C. $S_1 = 1; S_2 = 0; S_3 = 0$
D. $S_1 = 0; S_2 = 1; S_3 = 1$

Answer

Basically, as per the question we need the threads to fire in the order –

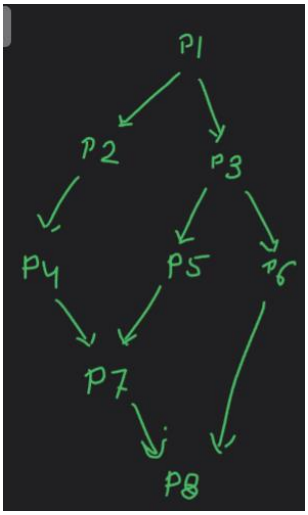
$$T_2 \rightarrow T_1 \rightarrow T_3$$

Hence, **Option C** is the correct initial condition for this.

Question

Find the minimum number of semaphores and the code of the processes needed to execute the following processes in that particular order.

Use only binary semaphores.



Answer

```
int S1 = 1;
int S2 = S3 = S4 = S5 = S6 = S71 = S72 = S81 = S82 = 0;
```

```
//PROCESS P1
wait(S1);
CS;
signal(S2);
signal(S3);

//PROCESS P2
wait(S2);
CS;
signal(S4);

//PROCESS P3
wait(S3);
CS;
signal(S5);
signal(S6);
```

```
//PROCESS P4
wait(S4);
CS;
signal(S71);

//PROCESS P5
wait(S5);
CS;
signal(S72);

//PROCESS P6
wait(S6);
CS;
signal(S81);
```

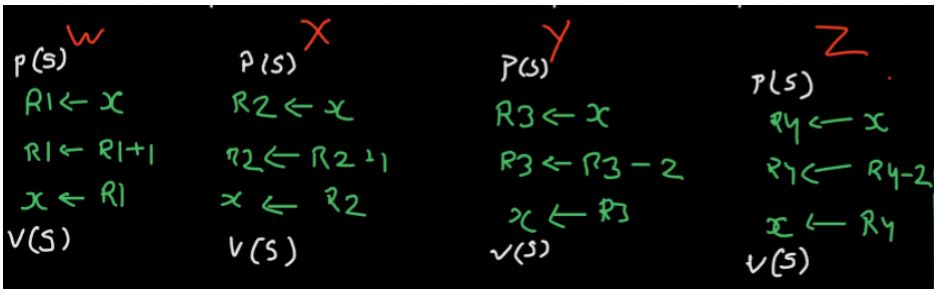
```
//PROCESS P7
wait(S71);
wait(S72);
CS;
signal(S82);

//PROCESS P8
wait(S81);
wait(S82);
CS;
signal(S1);
```

Question

A shared variable *x*, initialized to zero, is operated on by four concurrent processes W, X, Y, Z as follows. Each of the process W and X reads *x* from memory , increments by one, stores it to memory and then terminates. Each of the processes Y and Z reads *x* from memory , decrements by two, stores it to memory and then terminates. Each processes before reading *x* invokes the P operation (i.e., wait) on a counting semaphore S and invokes *the* V operation (i.e., signal) on the semaphore S after storing *x* to memory. Semaphore S is initialized to two. What is the maximum possible value of *x* after all processes complete execution?

Answer

As per the question, we have –



Since, the initial value of the semaphore is 2, we can have 2 concurrent processes making use of the shared resource. Hence, we can assume a case as follows –

- W and Y execute and access the critical section together making $S = 0$.
- First, Y finishes execution and sets value of $x = -2$ and $S = 1$
- Next, W finishes execution and sets the value of $x = 1$ and $S = 2$

Similarly, even processes X and Z are firing together and they will also return $x = 1$. Hence,

$$x_{max} = 1 + 1 = \mathbf{2}$$

Question

Given below is a program which when executed spawns two concurrent processes:

```
Semaphore X:=0;
/* Process now forks into concurrent processes P1 & P2 */
     P1 : repeat forever        P2:repeat forever
     V(X);                           P(X);
     Compute;                   Compute;
     P(X);                           V(X);
```

Consider the following statements about processes P1 and P2:

I: It is possible for process P1 to starve.

II. It is possible for process P2 to starve.

Answer

Since the semaphore is initialized with 0, Process P2 will never execute as P(x) will not return anything. On the other hand, Process P1 will fire as V(x) will set x = 1 and then P(x) will make x = 0. Thus, it is possible for process P2 to starve.

On the other hand, lets say P1 runs once and then P2 starts running. However, it gets preempted before V(x) statement. In that case, process P1 will be the one which can starve. Hence, **both statements are correct.**

Question

Suppose there are 2 processes – P1 and P2. It is given that –

$$Statements\ of\ P1 = \{S11, S12, S13, S14, S15\}$$

$$Statements\ of\ P2 = \{S21, S22, S23, S24\}$$

Let's assume that the order of the statements need to be as follows –

$$S11 \rightarrow S21 \rightarrow S22 \rightarrow S12 \rightarrow S13 \rightarrow S14 \rightarrow S23 \rightarrow S15 \rightarrow S24$$

Answer

```
int S1 = 1;
int S2 0;
```

```
//PROCESS P1
wait(S1);
S11;
signal(S2);
wait(S1);
S12;
S13;
S14;
signal(S2);
wait(S1);
S15
signal(S2);
```

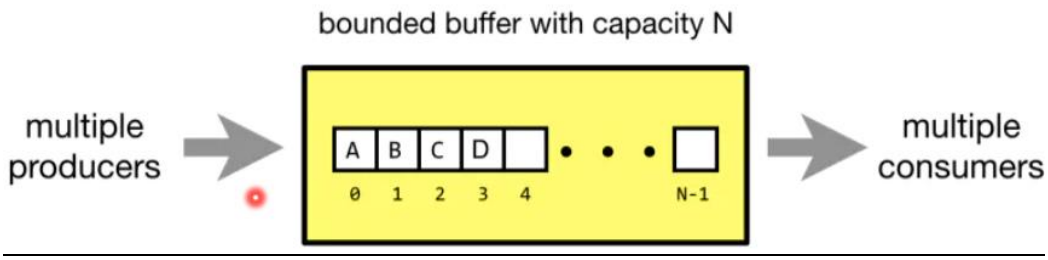
```
//PROCESS P2
wait(S2);
S21;
S22;
signal(S1);
wait(S2);
S23;
signal(S1);
wait(S2);
S24;
```

CLASSIC SYNCHRONIZATION PROBLEMS

- Producer – Consumer problem (Bounded – buffer problem)
- Reader – writer problem
- Dining philosopher problem

Bounded – Buffer problems

Let us assume that there are multiple **producers** which produce items and push them onto a buffer while there are multiple **consumers** that pull the items from the buffer to use it. Here, the **buffer** which is in the main memory will act as a shared resource as shown below –



The problem here is that the buffer is **bounded** aka it has a limited capacity. So here, there are two main problems which occur -

- If the buffer is full, then the producers can't add any more items in the buffer.
- If the buffer is empty, then the consumers can't pull any items from the buffer.

To solve this problem, we need to have **three semaphores** as follows –

- **Mutex –** It is a binary semaphore that helps in **mutual exclusion,** hence the name. It is initialized to 1
- **Full –** It is a counting semaphore that counts the number of occupied slots in the buffer. It is initialized to 0
- **Empty –** It is a counting semaphore that counts the number of empty slots in the buffer. It is initialized to N.

The codes for the producer and consumer are given below –

```
//PRODUCER
wait(Empty); //Decrement the number of empty slots
wait(Mutex); //For mutual exclusion

//PRODUCER CODE

signal(mutex);
signal(full); //Increment the number of occupied slots
```

```
//CONSUMER
wait(Full); //Decrement the number of full slots
wait(Mutex); //For mutual exclusion

//CONSUMER CODE

signal(mutex);
signal(Empty); //Increment the number of empty slots
```

Now, let us assume a case where the producer entry section is swapped –

```
//PRODUCER
wait(Mutex);
wait(Empty);
```

```
//CONSUMER
wait(Full);
wait(Mutex);
```

In this case, let's assume the buffer is **Full**. First, the producer will set the Mutex = 0. Then, the wait(Empty) will not return anything as the buffer is empty. Then, when we try to run the consumer code, the Full is decremented by 1 but wait(Mutex) will not return anything. Therefore, neither Producer not Consumer will be able to execute. Hence, **there is a deadlock.**

The same thing happens if we have an **Empty** buffer and we swap the statements of the consumer.

Reader – Writer Problem

In this case, the Reader is an entity which will be reading a shared resource while the Writer is the entity which will be writing to the shared resource. In this case, there can be three cases –

- When two readers are reading a shared resource. This case is fine and has no errors.
- When a reader is reading and a writer is writing on the shared resource, then there will be a data inconsistency.
- When two writers are writing on the shared resource, then there will be data inconsistency.

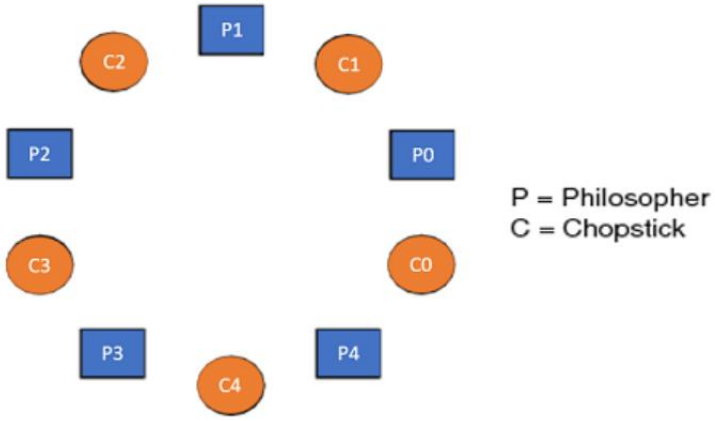
To solve this problem, we need to have three semaphores –

- **Mutex –** A binary semaphore for mutual exclusion initialized to 1.
- **Wrt –** Binary semaphore to restrict readers and writers when writing is going on initialized to 0.
- **Readcount –** Integer variable to denote the number of active readers initialized to 0.

```
//WRITING PROCESS
wait(wrt);
//writing process
signal(wrt);


//READING PROCESS
wait(mutex);
readcount++;
if (readcount == 1)
    wait(wrt);
signal(mutex);
//perform reading
wait(mutex);
readcount--;
if (readcount == 0)
    signal(wrt);
signal(mutex);
```

Dining Philosopher Problem

The dining philosopher's problem is the classical problem of synchronization which says that Five philosophers are sitting around a circular table and their job is to think and eat alternatively. A bowl of noodles is placed at the center of the table along with five chopsticks for each of the philosophers. To eat a philosopher needs both their right and a left chopstick. A philosopher can only eat if both immediate left and right chopsticks of the philosopher is available. In case if both immediate left and right chopsticks of the philosopher are not available then the philosopher puts down their (either left or right) chopstick and starts thinking again.



The solution is very simple here. We first assign a binary semaphore for each chopstick. If a philosopher picks the two chopsticks, then their semaphores will be decremented so that they can't be used by the other philosophers. Once done, the semaphores would then be incremented.

```
CS[k] = {C1, C2, C3, ... ,Ck}; //Array of chopstick semaphores

//Philosopher 'i'
wait(CS[i]);
wait(CS[(i+1) % k]);
//Start Eating
signal(CS[i]);
signal(CS[(i+1) % k]);
//Start Thinking
```

In this solution, suppose we have the following processes –

- P0 takes C0
- P1 takes C1
- P2 takes C2
- P3 takes C3
- P4 takes C4

In this case, each philosopher has 1 chopstick each but at the same time, no additional chopstick is remaining on the table. Hence, the philosophers can't eat and proceed thus making this a **Deadlock.**

To solve the deadlock problem, we can have the following pointers –

- Have $k - 1$ philosophers and k chopsticks.
- Ensure that two chopsticks are available before allowing the philosopher to eat.

SOLUTION WITHOUT BUSY WAITING

When we run wait(), then there is a case of busy waiting where the processes will keep running the wait() function endlessly. To solve this issue, we can send the processes who are stuck in the wait() function to the blocked queue and wake them back up once they are ready to execute.

```
wait(Semaphore s){
   s=s-1;
   if (s<0) {
      // add process to queue
      block();
   }
}
```

```
signal(Semaphore s){
   s=s+1;
   if (s<=0) {
      // remove process p from queue
      wakeup(p);
   }
}
```

PRIORITY INVERSION

This is the case where the higher priority process is stalled and is waiting for a low priority process to complete its execution**.** Suppose we have a low priority process executing in critical section. Now, a process with higher priority is encountered. However, it can't preempt the lower priority process from executing in the critical section. Therefore, the higher priority process is waiting for the low priority process to complete its execution and is a classic example of **priority inversion.**

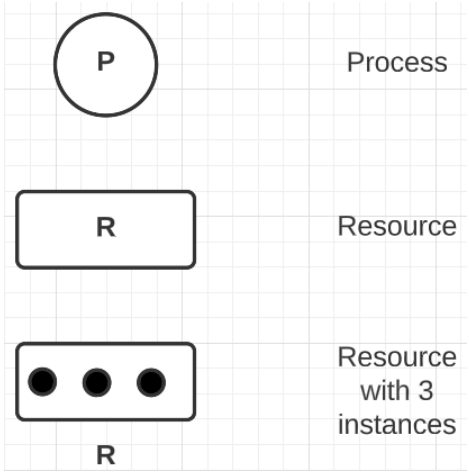
DEADLOCK

This is a case where none of the processes can proceed with their execution and the system is indefinitely stalled. For a deadlock, the following conditions need to be satisfied –

- **Mutual Exclusion –** One resource can be used by one process only. If one resource is shared with multiple processes, then with parallel execution deadlock can be resolved.
- **Hold and Wait –** Deadlock can only occur when every resource is holding some resources and are waiting for other resources.
- **No preemption –** There should be no forceful preemption of resources for a deadlock.
- **Circular Wait –** The waiting of resources should be done in circular manner for a deadlock to occur.

RESOURCE ALLOCATION GRAPH

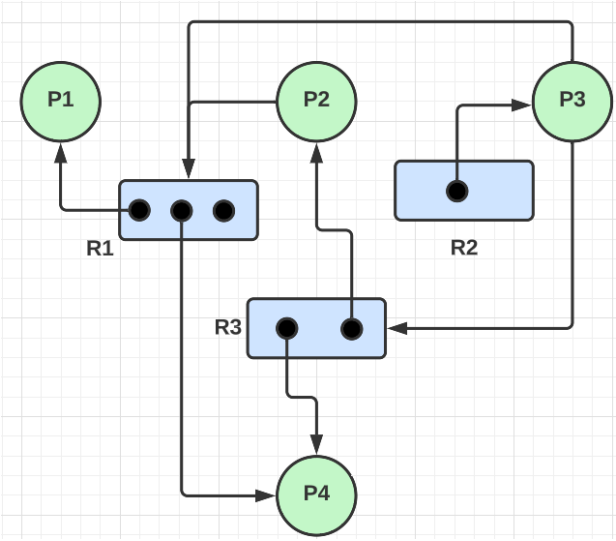
It is a graphical representation of the various processes and resources and how they are allocated. There can be two types of vertices – **processes** and **resources.**



On the other hand, there can be two types of edges –

- **Allocation Edge –** This represents resource allocation and goes **from resource to process.**
- **Request Edge –** This represents the request of process for resource and goes **from process to resource.**

For example, let us assume the graph below –



- Resources R1, R2 and R3 have 3, 2 and 1 instances respectively.
- P1 has 1 instance of R1 allocated.
- P2 has 1 instance of R3 allocated while it also requests R1.
- P3 has 1 instance of R2 allocated while it requests R1 and R3.
- P4 has 1 instance of R1 and 1 instance of R3 allocated.

The above information can be written in the form of a table as well –

NOTE – Allocated edges go from resource instance to the process since an **instance is allocated to a process.** On the other hand, a request edge goes from process to entire resource as **process requests an entire resource and not an instance.** This is because if a resource has 3 instances, it means that there are three resources of the same type. For example, if we say there are 3 instances of a printer, that means there are 3 printers of the exact type.

RECOVERY FROM DEADLOCK

There are three ways to deal with deadlock –

- Make sure we prevent deadlock in such a way that deadlocks doesn't occur.
- Allow deadlock to occur and when it does, we try to resolve it.
- Ignore and pretend that a deadlock never occurred. This is the most common solution

Deadlock Prevention

In this case, the system tries to prevent any of the 4 necessary conditions for a deadlock from occurring. Let us look at each way in brief –

PREVENT MUTUAL EXCLUSION

In this case, we prevent any sort of mutual exclusion. So, no process is waiting for another process to complete execution. This can be done in 2 ways –

- Increase the number of resources so that there is **no shared resource.**
- Make sure all the processes are independent processes.

These two asks are quite costly and un – reasonable. Hence, preventing mutual exclusion is **not practically possible.**

PREVENT HOLD AND WAIT

In this case, we prevent the hold and wait condition. For a deadlock, all processes need to hold some resources while wait for other resources. To prevent this, we employ 2 conditions –

- First, a process will only hold if it has all the resources. So if a process needs 3 resources and if and only if all 3 resources are free and available, the process will hold all 3 and execute without interruption. Hence, it will **hold without waiting** for any other resource.
- Second, even if there is a single resource that is not available, the process will release all resources and wait till all the required resources are free. Hence, it will **wait without holding** any resource.

This solution seems to be practically possible but there are 2 major drawbacks –

- Suppose a process needs resources R1, R2 and R3 for 100ns, 150ns and 3ns. In this case, the process will hold all resources till it finishes execution. Thus, resource R3 is held for 253ns even though it has to be used only for 3ns. Thus, there is **poor utilization of resources.**
- Additionally, there are resources like hard disk which are frequently used. These resources will then be on hold most of the time. Thus, if a process needs the hard disk, it will be waiting for a long time for the hard disk to be free. Therefore, there is a **possibility of starvation**.

ENABLE PREEMPTION

As we know, deadlock only occurs when resources can't be preempted from the processes. So, to prevent deadlock we can enable preemption of resources. However, if we preempt a resource from a process in the middle of execution, then we can't be sure what will be the further state of the process and it is highly likely that the process will reach an unstable state. Hence, this is **also not a practical solution.**

PREVENT CIRCULAR WAITING

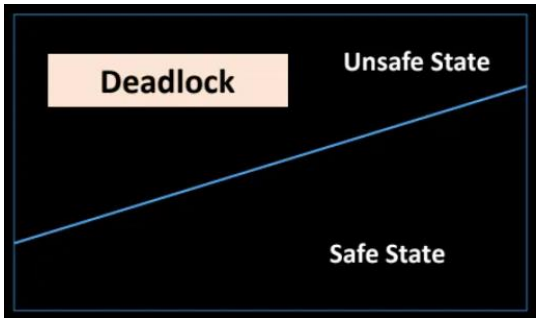
Final condition for deadlock is circular waiting. To prevent deadlock, we can have an additional condition as follows –

"If a process is holding a resource R_i, then it can only request another resource R_j such that $j > i$. If it has to request a resource R_k such that $k < i$, then it has to relinquish R_i".

In short, let's assume we have 5 processes from R1 to R5 and a process is currently holding resource R3. In this case, if the process wants R4 or R5, then it can directly request. However, if it wants resources R1 or R2, it has to relinquish R3 first. This way, **there is no chance of circular waiting.** However, the **execution time of the processes increases** by a large margin.

<u>Deadlock Avoidance</u>

In this method, the OS ensures that the system is in a ***"Safe State"*** which is basically a condition where the allocation of resources are done such that all process can execute without any errors and discrepancies.



In short, the resource allocation will happen in such a way that the resulting state of the system doesn't cause a deadlock in the system. To ensure the safe state, the OS uses something called the **Banker's Algorithm.**

Suppose, we have the processes as follows –

Process	Allocation	Max	Available
P1	1	3	1
P2	5	8	
P3	3	4	
P4	2	7	

The pre-requisite for the Banker's algorithm is that every process must announce what is the current amount of resources it has allocated and what is the worst case maximum number of resources it needs. After this, we first calculate the **Current Need** which is defined as follows –

$$Current\ Need = Max - Allocation$$

Therefore,

PROCESS	CURRENT NEED
P1	2
P2	3
P3	1
P4	5

As we can see, Process P3 only needs 1 more resource and we have 1 resource available. Hence, we can provide that resource to P3 to finish its execution. After that, P3 releases all the resources and with that, we have **4 resources available.**

Now, we can either provide the available resources to either P1 or P2. We continue like this and we can get the following operation order –

$$P3 \rightarrow P1 \rightarrow P2 \rightarrow P4$$

The above sequence is also called **safe sequence** as operating in this sequence will allow the system to stay in the safe space. At the same time, we can have the following safe sequences as well –

$$P3 \rightarrow P2 \rightarrow P1 \rightarrow P4$$

$$P3 \rightarrow P1 \rightarrow P4 \rightarrow P3$$

$$P3 \rightarrow P2 \rightarrow P1 \rightarrow P4$$

Therefore, **there can be multiple safe sequences.**

Question

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

Answer

PROCESS	CURRENT NEED			
	A	B	C	D
P1	0	0	0	0
P2	0	7	5	0
P3	1	0	0	2
P4	0	0	2	2
P5	0	6	4	2

Using the above table, we can find a safe sequence as follows –

$$P1 \rightarrow P3 \rightarrow P2 \rightarrow P4 \rightarrow P5$$

Question

Process	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
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			

What will happen if process P_0 requests one additional instance of resource type A and two instances of resource type C?

Answer

Since P_0 requests one additional instance of resource type A and two instances of resource type C, the table can be re-written as –

PROCESS	ALLOCATION			MAX			AVAILABLE		
	A	B	C	A	B	C	A	B	C
P0	1	1	2	7	5	3	2	3	0
P1	2	0	0	3	2	2			
P2	3	0	2	9	0	2			
P3	2	1	1	2	2	2			
P4	0	0	2	4	3	3			

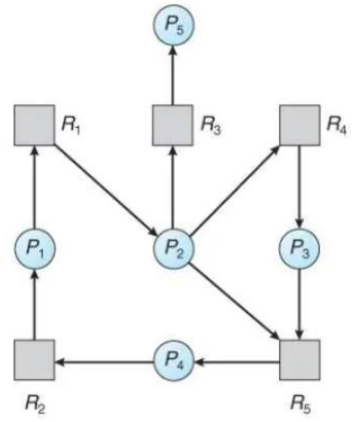
Thus, we get –

PROCESS	CURRENT NEED		
	A	B	C
P0	6	4	1
P1	1	2	2
P2	6	0	0
P3	0	1	1
P4	4	3	1

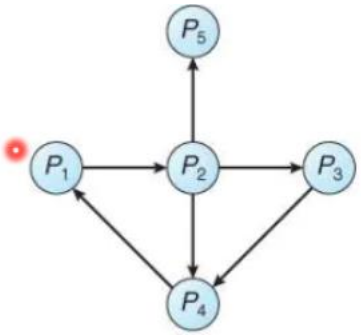
In the above case, we have **zero instances of C** available. Hence, the only process we can run is P2. However, P2 requires 6 instance of A and we have only **2 instances of A** available. Therefore, this system will be in an **unsafe state.**

WAIT FOR GRAPH

This is a graph derived from the resource – allocation graph and it consists of only processes. It indicates **which processes are waiting for which all processes.** Let us take the resource – allocation graph as shown below –



In this graph, Process P1 is waiting for Resource R1. Also, Resource R1 is being allocated and used by process P2. Therefore, we can conclude and say that **Process P1 is waiting for Process P2.** Similarly, we write the same for the rest of the processes as well –



DEADLOCK DETECTION

Since deadlock avoidance is a complex and time-consuming process, sometimes we let the OS go into deadlock and then detect if there has been deadlock. For detecting a deadlock, we can have 2 cases –

- If all the resources have single instance
- If the resources have varying number of instances

For the case where we have a single instance for each resource, we can detect deadlock if **there is a cycle in the wait for graph.**

When we are dealing with multiple instances, we will get a table similar to the one we saw during deadlock avoidance except they will give us the need/request of the resources directly. Therefore, if we are able to execute all the processes, then **there is no dead lock.** The deadlock detection can be done either after each resource allocation or whenever the CPU performance is going down (which may indicate a deadlock).

To resolve the deadlock, we can use three methods –

- Inform the system operator to manually handle the deadlock
- Terminate one or more processes in the deadlock
- Preempt resources to resolve the deadlock.

Many factors that can go into deciding which processes to terminate next:

1. Process priorities.
2. How long the process has been running, and how close it is to finishing.
3. How many and what type of resources is the process holding
4. How many more resources does the process need to complete
5. How many processes will need to be terminated
6. Whether the process is interactive or batch

Question

Consider a system with 3 processes A, B and C. All 3 processes require 4 resources each to execute. The minimum number of resources the system should have such that deadlock can never occur?

Answer

It is given that each process will need at least 4 resources to finish. So the maximum resources that we can allot such that none of the processes complete will be $3 + 3 + 3 = 9$. So if we have 9 resources that are divided equally, there will be a deadlock as none of the processes will be able to execute.

However, if we add just one more resource to any of the processes, they get 4 resources and one of them can execute. After that, the process which has finished execution will free up the resources and then the rest of the 2 processes can finish execution. Hence, the minimum number of resources needed is **10.**

NOTE

In general, suppose we have n processes and each of them can request a maximum of k resources, then the minimum number of resources needed to avoid deadlock is –

$$n * (k - 1) + 1$$

Question

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 ____.

Answer

Using the above expression, we can say that –

$$3 * (k - 1) + 1 \leq 4$$

Therefore,

$$\boldsymbol{k_{max} = 2}$$

TYPES OF LOCKS

Lock is a variable that ensure mutual exclusion for critical access as we have seen before. It can of various types –

- **Spinlock –** This is the case where a process is constantly trying to get the lock but is unable to. This is also called **busy waiting.**
- **Livelock –** When the processes are in the running state but are stuck and are unable to proceed further.
- **Deadlock –** When the processes are in the blocked state but are stuck and are unable to proceed further.
- **Semaphores –** These are lock variables that can be incremented/decremented and are used for synchronization
- **Reentrant lock –** A thread can reenter the lock on a resource multiple times without any deadlock.

MEMORY MANAGEMENT

This is another module of the OS which determines how memory is stored and managed in the system. It has the following main functions –

- Memory allocation
- Memory deallocation
- Memory protection

To store processes, the OS can have two approaches –

- **Contiguous –** The processes will be stored in a continuous/sequential manner. The entire process is a single entity. In this, the process memory is decided and it is easy to implement memory security. It can be of two types –
    - Fixed partitions
    - Variable partitions
- **Non-contagious –** The process is broken down into parts and are stored in a non – continuous manner. This can also be of 2 types –
    - Paging
    - Segmentation

Fixed Partition Contiguous Memory

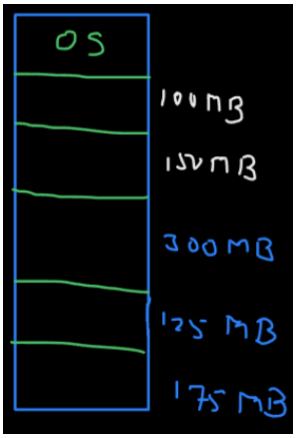
OS basically divides the memory into a fixed **number** of partitions (size of the partitions may vary) in such a way that each partition will store one process each. In this case, the maximum number of processes in the main memory (degree of multiprogramming) is the same of the number of partitions.

Suppose we have a partition of size 100B allocated in the main memory. However, there is a process of 60B which is stored in the partition. Then, there is a wastage of 40B. This phenomenon where extra space (more than required) is allocated to a process is called **Internal Fragmentation.**

Given the number of partitions and the list of incoming processes, it is necessary to have some algorithms or procedure to allocate the processes to the partition. There are a few such algorithms as listed below –

- **First Fit –** It is similar to FCFS where the first partition to have enough space to store the process will be assigned the process. There is moderate internal fragmentation.
- **Best Fit –** In this case, the algorithm find the partition with the minimum size with every iteration and then tries to fit the process in there. This way, the algorithm allows for minimal internal fragmentation but ends up consuming a lot of time.
- **Worst Fit –** This is the same as the Best fit except this algorithm searches for the largest partition with every iteration. Hence, the internal fragmentation and the time consumed are both high.
- **Next Fit –** It is the same as First Fit except the partition checking doesn't start from the beginning but from the last filled partition.

For example, let us assume there is a memory with partitions as shown below –



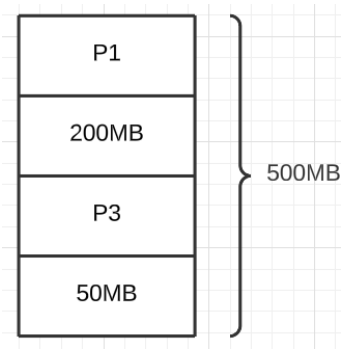
Let's assume that a process of size 110MB decided to come. Then,

- First fit will assign it to the 150MB partition
- Best fit will assign it to the 125MB partition
- Worst fit will assign it to the 300MB partition
- Suppose we already have the 150MB partition filled. Then, the Next fit will start from the 150MB partition and then move forward. So it will assign the process to the 300MB partition.

Variable Partition Contiguous Memory

In this case, there are no pre-defined partitions created. Instead, the entire memory is kept as an empty space. Once a process comes, then a partition of size same as the process size is **dynamically created** and the process is stored there. Since the partitions are created with the same size as the incoming process, there is **no internal fragmentation.**

Let us take an example here with main memory of size 500MB. First, a process P1 of size 100MB arrives and then a process P2 of size 200MB arrives. After that, let us assume a process P3 of size 150MB arrives and P2 finishes execution, hence de-allocating the memory space.

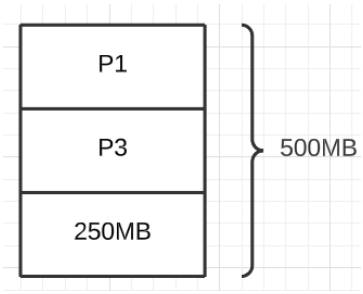


Now, let's assume a process P4 arrives of size 250MB. In this case, **we have the total memory to accommodate the incoming process but there is no contiguous memory large enough to accommodate the incoming process.** This phenomenon is referred to as **External Fragmentation.**

One thing to note is that suppose P4 was of size 300MB, then there was no way to store it even if we had contiguous memory. Therefore, this is **NOT External Fragmentation.**

Compaction – This is a solution for external fragmentation. In this case, all the allocated memory is re-located to one part of the main memory such that the empty space becomes contiguous in the other end. Hence, this will remove external fragmentation. However, since we are re-locating multiple processes, this is a time consuming solution.

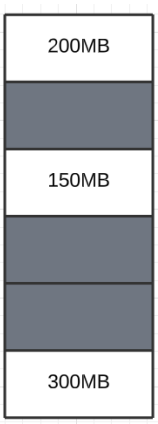
So, if we apply compaction to the example above, then the memory becomes –



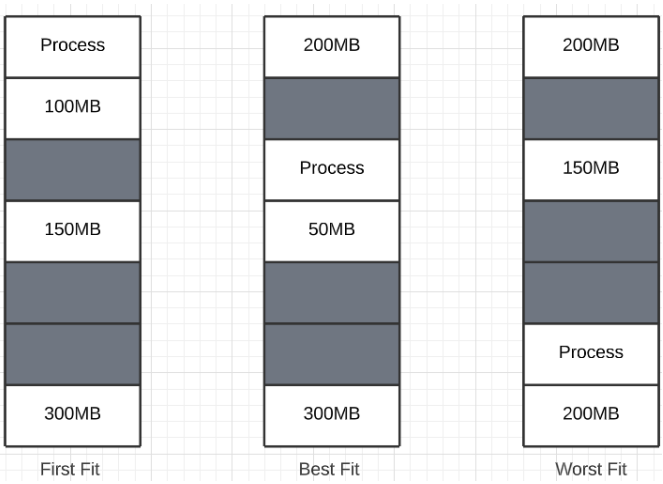
Thus, now process P4 can be accommodated without any external fragmentation.

Partition Allocation Process

Just like Fixed partition, we can use First Fit, Best Fit and Worst Fit algorithms for partition allocation here. Let us assume that the main memory is as follows –



Let us assume that a process of size 100MB comes in. Then, we can see the following allocation –



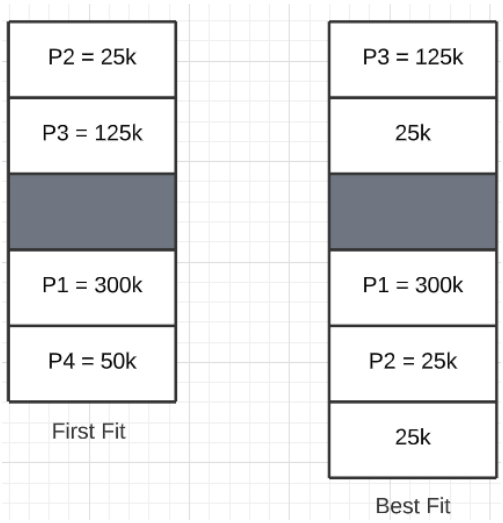
In this case, the Worst Fit is actually an advantageous algorithm as it results in a partition with the maximum contiguous memory.

Question

Consider the requests from processes in given order 300K, 25K, 125K, and 50K. Let there be two blocks of memory available of size 150K followed by a block size 350K. Which of the following partition allocation schemes can satisfy the above requests? (variable partition MMT)
A) Best fit but not first fit
B) First fit but not best fit
C) Both First fit & Best fit
D) neither first fit nor best fit

Answer

We can see that –



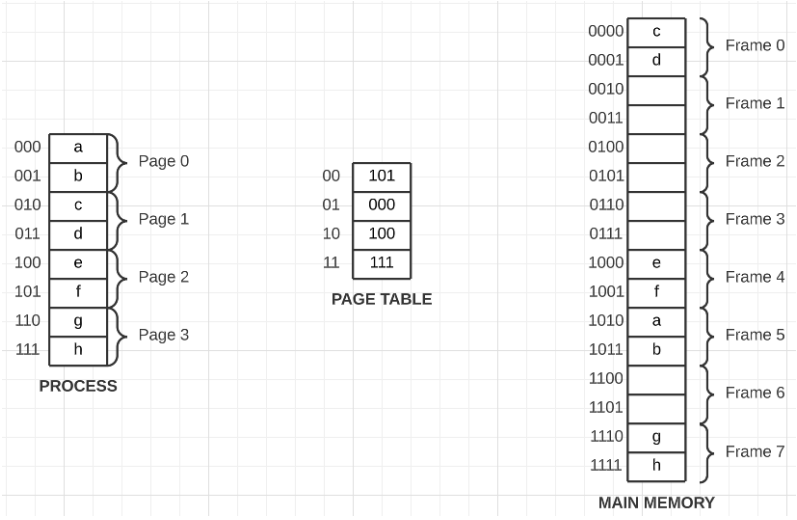
Therefore, we can see that the processes can be accommodated using First fit but not using Best fit. Hence, **Option B is the correct answer.**

PAGING

It is a type of non – contiguous memory management. In this case, each **process** is divided into **equal sized** partitions called **Pages.** At the same time, the **main memory** is also divided into **equal sized** partitions called **Frames.** The process pages are scattered throughout the frames in main memory and they need not be contiguous.

For each process, we store the Page – Frame mapping in a **Page Table.** Thus, the number of entries in the page table is the same as the number of pages in the process. Also, since page – frame mapping is a one-to-one mapping, then it can be said that **Frame size is the same as the Page size.** Additionally, the **Page Table is also stored in main memory (Frames).** The address of the process bytes are called **Logical Address** and the address of the Main Memory is called **Physical Address.**

Let us take an example below –



In the example above,

$$Main\ memory\ size = No\ of\ Frames * Frame\ size = 8 * 2 = \mathbf{16B}$$

$$Process\ size = No\ of\ Pages * Page\ size = 4 * 2 = \mathbf{8B}$$

Also,

$$Logical\ Address\ of\ \boldsymbol{a} = \mathbf{000}$$

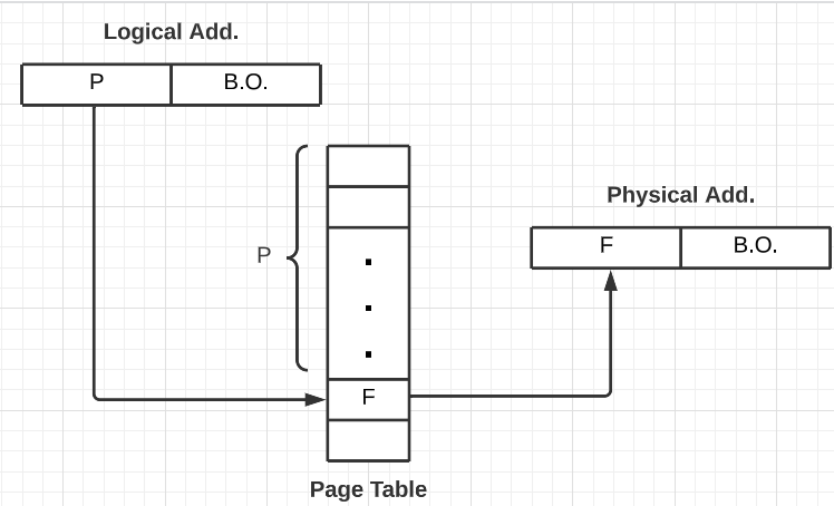
$$Physical\ Address\ of\ \boldsymbol{a} = \mathbf{1010}$$

Additionally,

$$Logical\ Address = Page\ Number + Byte\ Offset$$

$$Physical\ Address = Frame\ Number + Byte\ Offset$$

So, logical address of a is 000. The first 2 bits give us the page number and the last bit gives us the byte offset. So, a is in **Page number 0 and is at Byte 0.** Similarly, for the physical address, the first 3 bits will give us the Frame number and the last bit will be the Byte offset. Therefore, a is in **Frame number 5 (101) and is at Byte 0.** Thus, we can translate the logical address to physical address as follows –



Now, we encounter the final piece of the puzzle. Each process has a Page Table and these page tables are stored in the frames in the main memory itself. So, how is the OS going to know which frame has the page table of the given process? That is where a **Special Purpose register** is used whose name is **Page Table Base Register (PTBR).** The PTBR will store the base address for the page table of the current process.

NOTE – The collection of all the physical addresses is called the **Physical Address Space.** At the same time, the collection of all the logical addresses is called the **Logical Address Space.**

Question

Consider a paged memory system where the logical address of 28 bits and physical address of 34 bits. If each page table entry is of 4 bytes and page size is 2KB then:

1. Number of pages in process?
2. Number of frames in main memory?
3. Number of bits for page number?
4. Number of for frames?
5. Number of entries in page table?
6. Page table size?

Answer

$$Byte\ offset = \log_2(Page\ Size) = 11\ bits$$

Hence,

$$Page\ address = Logical\ Address - Byte\ offset = 28 - 11 = \mathbf{17\ bits}$$

Hence, there are **128k pages.** Also,

$$Frame\ address = Physical\ address - Byte\ offset = 34 - 11 = \mathbf{23\ bits}$$

Hence, there are **8M frames.** Now, it is given that each entry in the Page table is of 4 bytes. Therefore,

$$Page\ table\ size = 2^{17} * 4 = 2^{19} = \mathbf{512kB}$$

NOTE

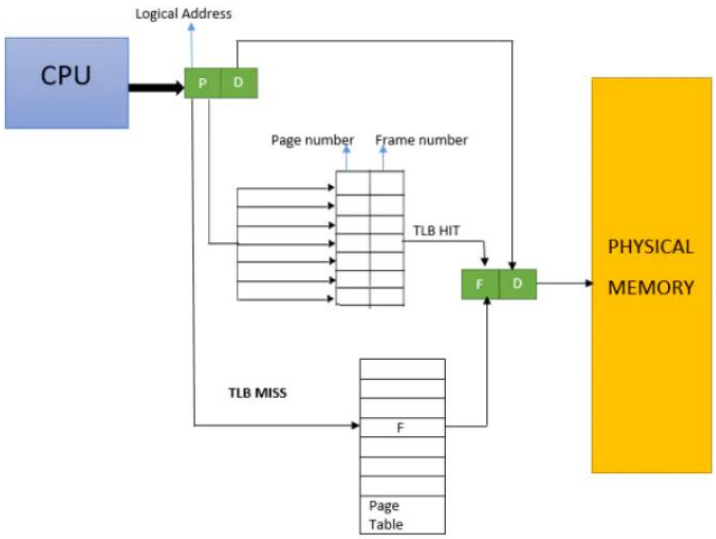
In the above question, the size of page table entry is 4B or 32 bits. However, we know that we store the frame number in the page table which is just 23 bits. So, in this case the page table entry has an addition **9 bits of extra information.** In cases where they haven't mentioned the page table entry size, assume it to be the same as the frame number size.

TRANSLATION LOOKASIDE BUFFER (TLB)

To access data from main memory with paging, we need 2 memory accesses – first one to get the Page table and the next one to get the data from the main memory post translation. So, the paging performance is not great as it takes a lot of time to get just one data point from the main memory.

There is a solution where we can store the page table in the CPU registers itself. This can work and improve the performance as the main memory will be accessed just once. However, it can only work if the number of page tables are very small (less than 10). Therefore, this is not a feasible solution.

The solution to improve the performance comes in the form of **Translational Lookaside Buffer (TLB).** It is basically a small memory which stores only the most frequently accessed page tables. This allows for faster access of the page tables for the frequently occurring processes.



First, we take the page address from the logical address and then access the TLB. If there is a hit in the TLB, then we can directly get the frame number and we can get the physical address. However, if there is a TLB miss, then we go the normal route and then we look for the page table and then get the physical address.

Hence, when there is a hit, there is only 1 single TLB access (negligible) and 1 main memory access. However, when there is a miss, there is 1 TLB access and 2 main memory accesses. Thus, if we assume the TLB has a hit ratio of H, then

$$Avg\ time = [H * (t_{TLB} + t_{MM})] + [(1 - H) * (t_{TLB} + 2 * t_{MM})]$$

Question

Suppose we have a TLB with 80% hit ratio. The TLB access time is 30ns and Main memory access time is 200ns. What is the average memory access time with and without TLB?

Answer

Without TLB,

$$Avg\ time = 2 * t_{MM} = \mathbf{400}\boldsymbol{ns}$$

With TLB,

$$Avg\ time = [0.8 * (30 + 200)] + [0.2 * (30 + 400)] = \mathbf{270}\boldsymbol{ns}$$

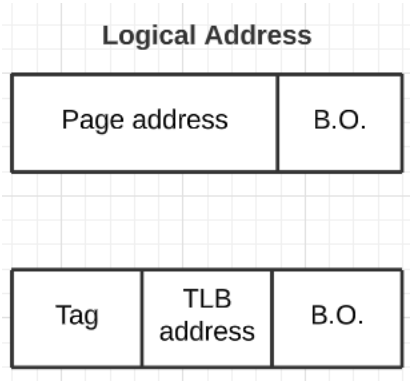
Thus,

$$Avg\ speed\ up = \frac{400}{270} \approx \mathbf{1.5}$$

TLB MAPPING

We can treat the TLB as a sort of cache. Each entry has an address and a tag bits and TLB bits. That is why we can have 3 type of mappings as well –

- **Direct mapping –** We perform mod operation to get the address mapping.
- **Set associative mapping –** We perform mod operation with set number
- **Fully associative mapping –** Values can be mapped anywhere since its fully associative.



Question

A computer system implements a 31- bit virtual address, page size of 8 kilobytes, and a 256-entry translation look-aside buffer (TLB) organized as direct mapped. The minimum length of the TLB tag in bits is _____?

Answer

$$Byte\ offset = \log_2(8K) = 13\ bits$$

$$TLB\ bits = \log_2(256) = 8\ bits$$

Therefore,

$$Tag\ bits = Virtual\ address - TLB\ bits - Byte\ offset = 31 - 8 - 13 = \mathbf{10\ bits}$$

Question

A computer system implements a 44- bit virtual address, page size of 1 kilobytes, and a 16KB look-aside buffer (TLB) organized as direct mapped. Each page table entry is of 4bytes. The minimum length of the TLB tag in bits is _____?

Answer

$$Byte\ offset = \log_2(1K) = 10\ bits$$

$$TLB\ bits = \log_2\left(\frac{16K}{4}\right) = 12\ bits$$

Therefore,

$$Tag\ bits = 44 - 12 - 10 = \mathbf{22\ bits}$$

Question

A computer system implements a 42- bit virtual address, 2GB physical address space, page size of 2KB, and an 8KB look-aside buffer (TLB) organized as direct mapped. Each page table entry contains a valid bit, a dirty bit and 2 protection bits along with the translation. The minimum length of the TLB tag in bits is _____?

Answer

$$Byte\ offset = \log_2(2K) = 11\ bits$$

$$Physical\ address\ size = \log_2(2G) = 31\ bits$$

Therefore,

$$Frame\ size = 31 - 11 = 20\ bits$$

Thus,

$$Page\ entry\ size = 20 + 1 + 1 + 2 = 24\ bits = 3B$$

Hence,

$$No\ of\ page\ entries = \frac{8k}{3B} = 2.6 * 2^{10} = 2^{12}$$

Therefore,

$$Number\ of\ Tag\ bits = 42 - 12 - 11 = \mathbf{19\ bits}$$

NOTE

The entire process of memory allocation, paging and segmentation is done in the time the CPU takes to move a process from the New state to the Ready state.

INTERNAL FRAGMENTATION IN PAGING

Let us assume there is a case like –

$$Page\ size = 4KB$$

Also, let's assume there is a process with size 9KB. In that case, it will require 3 pages and thus require 3 frames in the main memory. However, in the 3rd frame, it will require only 1KB and the other 3KBs will go to waste. This is the internal fragmentation problem in paging.

One solution to this is to reduce the page table size. So let's assume that the page size has now been reduced to 2KB. Then, the process will now require 5 frames and will have a wasted space of just 1KB. However, when we reduce the page size, the same memory will require more pages and more frames which will in turn increase the page table size.

SEGMENTATION

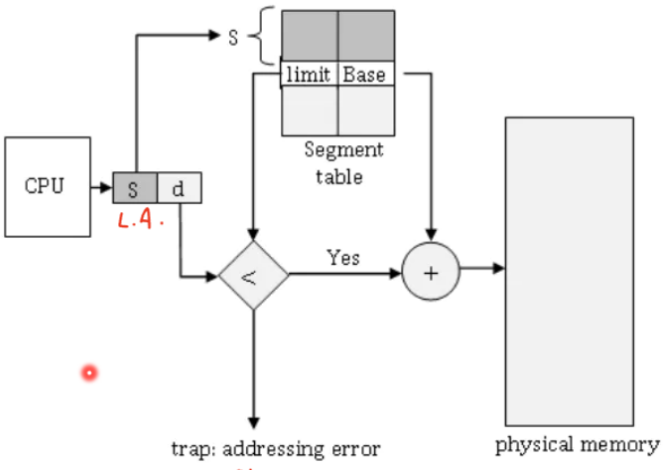
This is another non – contiguous memory management type where each process is divided in **logically related partitions** and these partitions are **need not be equal.** These partitions are called **segments.**

Since the process will now be divided into unequal partitions, it doesn't make sense to divide the main memory into equal partitions (frames) like we did in paging. Thus, for segmentation the **main memory is not divided.** The segments are stored in the main memory wherever they can find space. The **segment table** gives the following information –

- Segment number
- The segment starting address (Base)
- The size of the segment (Limit)

The Logical Address to Physical Address translation occurs as follows –

- The Logical Address consists of 2 parts – the segment number and the byte offset
- First get the segment number and match it to the segment table base value.
- From the segment table, get the Limit value from the row
- Compare Limit to the Byte offset
    - If Byte offset is greater, then it results in **Segmentation Fault**
    - If Byte offset is lesser, then the result will be valid
- If the Logical address is valid, the Base and the Limit is added together to get the range of physical addresses. The range varies from $Base$ to $Base + Limit$



Question

Maximum segment size = 16KB
Number of segments in process = 2^{10}
Logical address = _____ bits ??

Answer

$$Segment\ bits = \log_2(2^{10}) = 10\ bits$$

$$Byte\ offset\ or\ limit = \log_2(16K) = 14\ bits$$

Therefore,

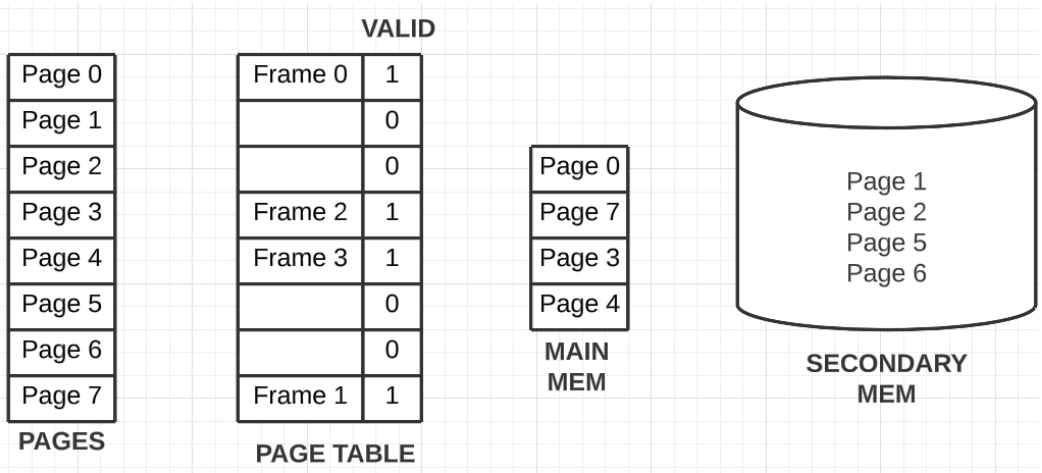
$$Logical\ address = 10 = 14 = \mathbf{24\ bits}$$

Since the segments are of different sizes, there can be a case where the gap between two segments is free but too small to accommodate any other segment. Hence, segmentation suffers from **External Fragmentation.**

VIRTUAL MEMORY

This is another feature of the OS which enables a larger process to run with a smaller memory. The concept here is that we only bring the necessary content for a process in main memory and the rest of the content is kept in the secondary memory. This is implemented using paging.

Let us take an example where we have 8 pages in total and out of those, only 4 pages are present in the frames in the main memory and the rest of the pages are in the secondary memory. However, the page table will have the information for all the pages. The pages who are not in the main memory will have a garbage value in the page table. Thus, there is a need for valid bits in the page table.



In the above case, if the process needs to access Page 5, it first goes to the page table. There, Page 5 has the valid bit set to 0 indicating that this page is not present in the main memory and it would have to be extracted from the secondary memory. This is referred to as a **Page Fault.** Page Fault is a **software interrupt** and once encountered, OS takes over and brings the faulted page from the secondary memory to the main memory and once done, the instruction that caused a Page fault is **Restarted (not resumed).** This entire process that OS does after a page fault is called a **Page Fault Service routine**.

During page fault service routine, a page is pulled from the secondary memory to the main memory and valid bit is changed. If there is a frame in the main memory available, then it is well and good. Else, **page replacement must be done in the main memory.** This process of bringing the pages into the main memory when and only when CPU demands it is called **Demand Paging.** In case there are **zero pages in main memory initially,** then there will always be a page fault for the first time and this is called **Pure Demand paging.**

To save time, there is a **write back method** is used in the virtual memory. That means, the page table has an additional **modified bit** as well. Whenever, the page table content is changed, the bit is set to 1. When that page is being replaced, the modified value is stored back in the secondary memory when the modified bit is set to 1.

Question

Assume,

- Main memory access time = 200ns
- Page fault service time = 5ms
- Page fault rate = 0.01

Answer

$$Effective\ memory\ access\ time = (0.01 * 5ms) + (0.99 * 2 * 200ns) = \mathbf{50.396\mu s}$$

Question

Assume that the memory access without page fault is 400ns. Additionally, the time taken to move a page from secondary memory to main memory is 10000ns. If page fault rate is 2%, then what is the effective memory access time? Assume the page table has no dirty bit.

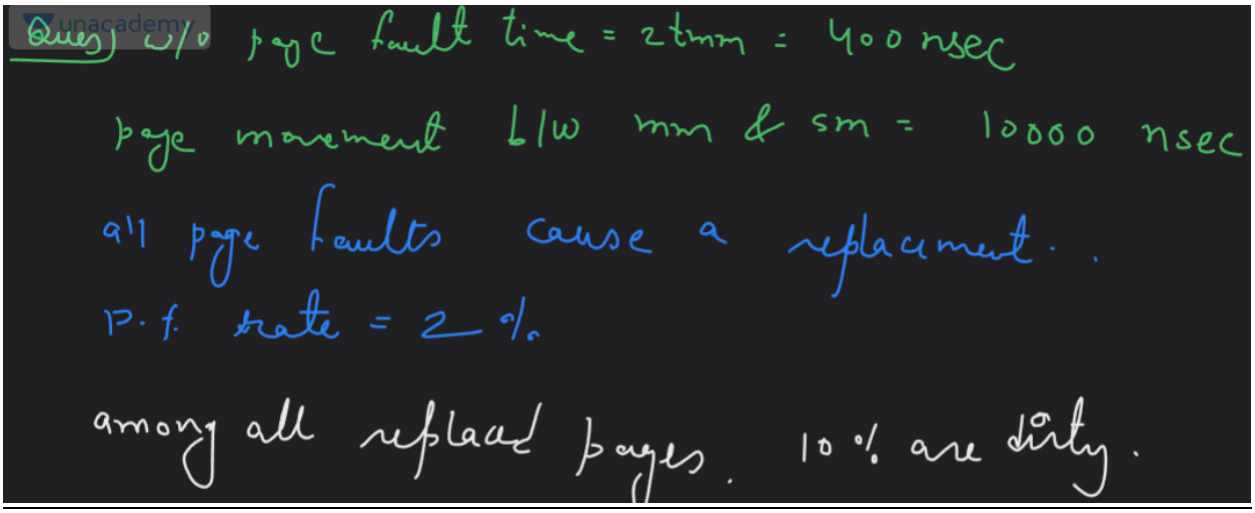
Answer

One thing to note here is that if there is a page fault, then there is a need to take one frame from the secondary memory to the main memory and there is a need to take a frame from main memory back to the secondary memory (write back as there are no dirty bits).

Hence,

$$Avg\ access\ time = (0.98 * 400) + [0.02 * (10000 + 10000)] = \mathbf{792ns}$$

Question



Answer

For page fault, we know that we need to pull a page from secondary memory to main memory which will take 10000 ns. However, we can see that only 10% of pages are dirty and therefore only 10% are written back. Thus,

$$Mem\ access\ time = (0.98 * 400) + [0.02 * (10000 + 0.1 * 10000)] = \mathbf{612}\boldsymbol{ns}$$

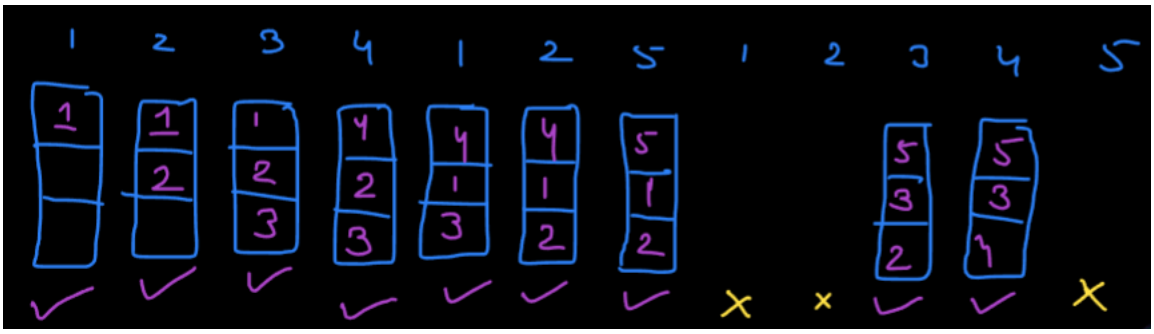
NOTE

When a page is replaced from main memory and if it is available in the TLB, then the TLB entry is also made invalid.

PAGE REPLACEMENT POLICIES

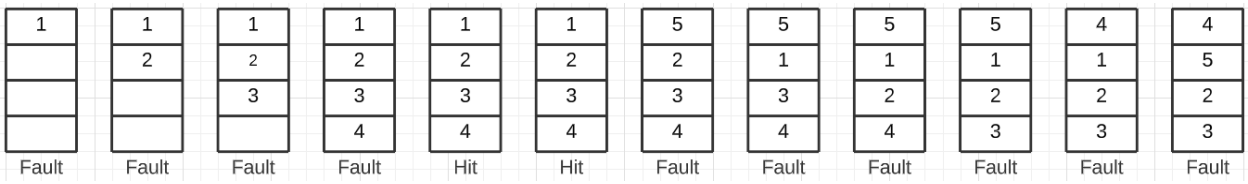
First In First Out (FIFO)

As the name suggests, the oldest page will get replaced. For example, lets assume there are 3 frames in the main memory and the page sequence is as follows – 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

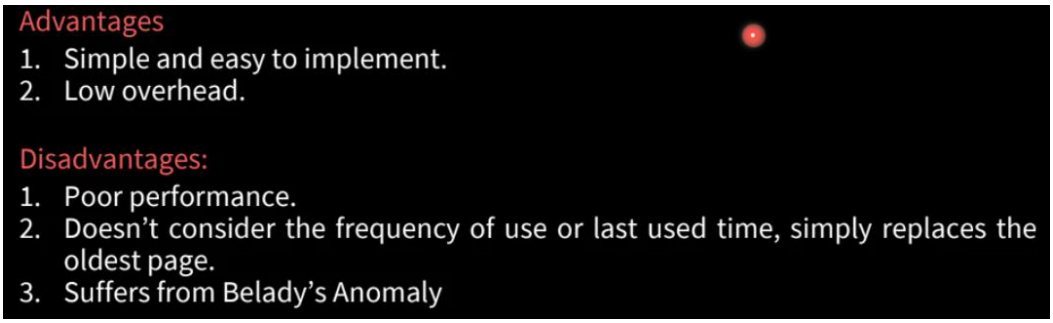


Thus, we have a total of **9 faults and replacements.**

Now, suppose we increase the number of frames to 4. Then, we get –

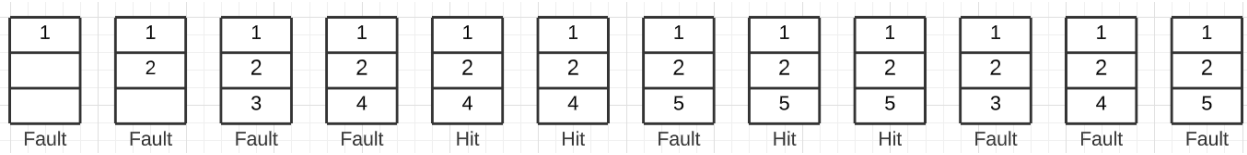


In this case, there are **10 page faults.** One thing to notice here is that there can be scenarios where **for specific sequences in FIFO, increasing the number of frames can result in more page faults.** This is referred to as **Belady's Anamoly.**



Last In First Out (LIFO)

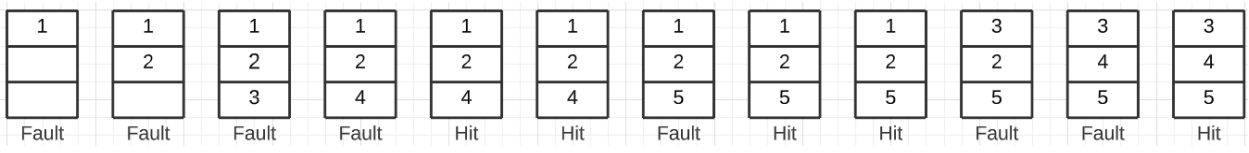
It is the opposite of FIFO. Let us take the same sequence as mentioned above with 3 frames. Then,



Thus, we have a total of **8 page faults.**

Optimal Policy

In this case, we go check all the future references and replace the page that will not be referred to in the near future. Suppose we take the same example as before with 3 frames. Then, we get –



Hence, the number of faults has been decreased to **7 faults.** However, this solution is not practically possible since there is no way to find the future references. However, this policy gives the minimum number of faults and therefore acts as a standard against which all other policies are compared.

Advantages

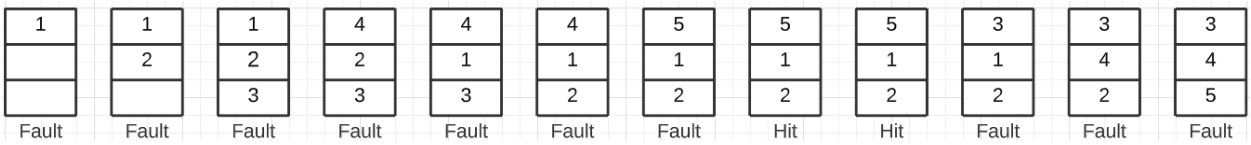
1. Easy to Implement
2. Simple data structures are used
3. Highly efficient

Disadvantages:

1. Requires future knowledge of the program
2. Time-consuming

LEAST RECENTLY USED

The page which was least recently referenced is replaced. Taking the above example, we get –



Hence, there are **10 page faults.**

Advantages

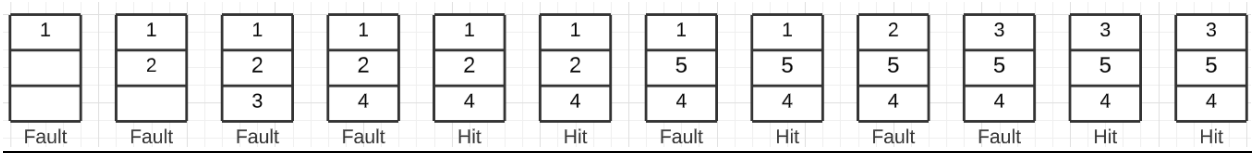
1. Efficient.
2. Doesn't suffer from Belady's Anomaly

Disadvantages:

1. Complex Implementation
2. Expensive
3. Requires hardware support

LEAST RECENTLY USED

The page which was most recently referenced is replaced. Taking the above example, we get –



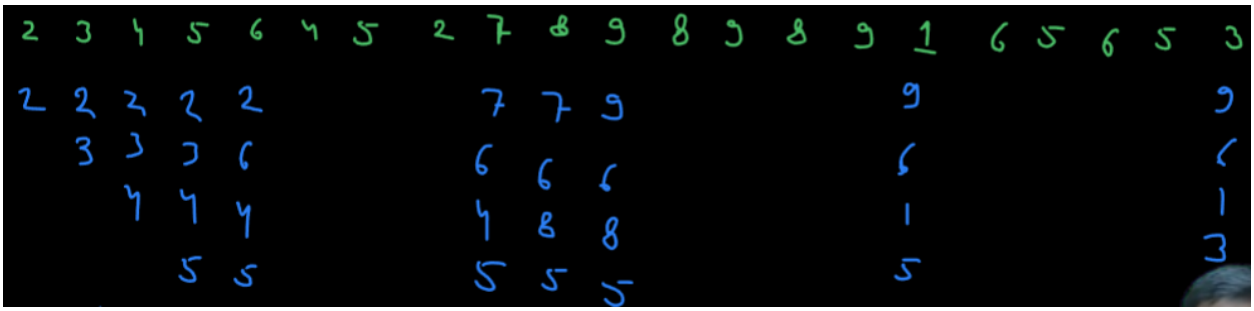
Hence, there are **7 page faults.**

Question

Consider the following page references:
2, 3, 4, 5, 6, 4, 5, 2, 7, 8, 9, 8, 9, 8, 9, 1, 6, 5, 6, 5, 3
Using optimal policy and 4 frames. Memory access time is 2ms and page fault service time is 40ms. The effective memory access time is?

Answer

For optimal policy, we get –



Hence, there are a total of **10 page faults.** Thus,

$$Fault\ ratio = \frac{10}{21}$$

When there is no fault,

$$Time = 2 * t_{MM} = 2 * 2 = 4ms$$

It is given that for page fault, it takes $40ms$. Hence, we get –

$$Avg\ access\ time = \left(\frac{10}{21} * 40\right) + \left(\frac{11}{21} + 4\right) = \mathbf{21.14ms}$$

Question

Take the previous question and assume that –

- Memory access time without page fault = 4ms
- Memory access time with page fault and without replacement = 30ms
- Memory access time with page fault and with replacement = 50ms

Answer

The initial 4 faults are due to the frames being empty and hence don't involve any replacement. The rest of the 6 faults replace the frame value. Hence,

$$Avg\ access\ time = \left(\frac{4}{21} * 30\right) + \left(\frac{6}{21} * 50\right) + \left(\frac{11}{21} * 4\right) = \mathbf{22.095ms}$$

COUNTING ALGORITHMS

These are again page replacement algorithms but they maintain a count of the number of times a page has been referred to decide which page to replace. They are of 2 types –

- Least Frequently Used (LFU)

- Most Frequently Used (MFU)

Suppose we have multiple pages with the same frequency, then we resort to FIFO to settle the tie.

Least Frequently Used (LFU)

Let us take the case where we have 3 frames and the sequence is as follows –

1 2 0 3 0 4 2 3 0 3 2

REQUEST	PAGE FREQUENCIES					FRAME NUMBER			RESULT
	0	1	2	3	4	0	1	2	
-	0	0	0	0	0	-	-	-	Initially empty
1	0	1	0	0	0	1	-	-	Fault
2	0	1	1	0	0	1	2	-	Fault
0	1	1	1	0	0	1	2	0	Fault
3	1	1	1	1	0	3	2	0	Fault
0	2	1	1	1	0	3	2	0	Hit
4	2	1	1	1	1	3	4	0	Fault
2	2	1	2	1	1	2	4	0	Fault
3	2	1	2	2	1	2	3	0	Fault
0	3	1	2	2	1	2	3	0	Hit
3	3	1	2	3	1	2	3	0	Hit
2	3	1	3	3	1	2	3	0	Hit

As we can see, the fault ratio is 7/11.

Most Frequently Used (MFU)

Let us take the same example as above –

REQUEST	PAGE FREQUENCIES					FRAME NUMBER			RESULT
	0	1	2	3	4	0	1	2	
-	0	0	0	0	0	-	-	-	Initially empty
1	0	1	0	0	0	1	-	-	Fault
2	0	1	1	0	0	1	2	-	Fault
0	1	1	1	0	0	1	2	0	Fault
3	1	1	1	1	0	3	2	0	Fault
0	2	1	1	1	0	3	2	0	Hit
4	2	1	1	1	1	3	2	4	Fault
2	2	1	2	1	1	3	2	4	Hit
3	2	1	2	2	1	3	2	4	Hit
0	3	1	2	2	1	3	0	4	Fault
3	3	1	2	3	1	3	0	4	Hit
2	3	1	3	3	1	2	0	4	Fault

Here, the fault ratio is also 7/11

NOTE

Suppose there are x frames and a total of n pages such that $n > x$. Let's assume that the sequence $1, 2, 3, \ldots, n$ is repeated m times. Then,

$$Page\ faults\ in\ LIFO - Page\ faults\ in\ Optimal = m - 1$$

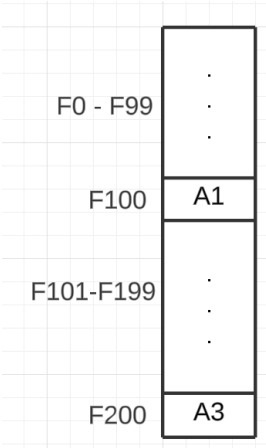
FRAME ALLOCATION

As we know, the main memory has limited number of frames that must be shared among the processes that need to be executed. The allocation of frames to the various processes is termed as **frame allocation.**

Frame allocation can be done in two ways –

- **Equal allocation –** All processes get equal number of pages. So, if we have p processes and f frames, then each process gets f/p number of frames.
- **Proportional allocation –** In this case, the frame allocation is done based on process size. So, we basically use $f \propto sizeof(p)$ condition. In short, larger processes get more frames when compared to smaller processes.

Suppose there is a process $P1$ which is divided into **3 pages –** A1, A2 and A3. Let's assume process P1 gets allocated **2 frames** in the main memory and pages A1 and A3 are present in the frames F100 and F200 respectively.

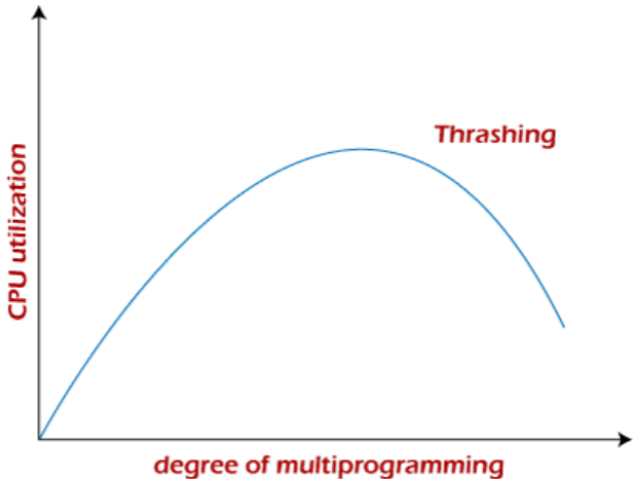


Now, suppose the process requests for page A2. This would cause a **Page Fault** and there would be a need for page replacement. This page replacement in the main memory can happen in 2 ways –

- **Local –** This means that one of the pages of Process P1 itself will get replaced. So, page A2 will either replace page A1 or Page A3. In short, the page being replaced be in a frame belonging to the process. Hence, the **number of frames for the process** will remain the **same.**
- **Global** – This means that page A2 can replace any of the frames from F0 to F200. This is useful when we have high priority processes as the high priority process can increase the number of frames being allocated to them. Finally, this also ensures a more efficient use of frames and improves throughput.

THRASHING

We know that as we increase the number of processes aka Degree of multiprogramming, the CPU utilization increases. This is true till a certain number of processes as increasing the number of processes also reduces the number of frames being allocated for a process. Thus, after a certain number of processes, the number of frames per process is so less that the CPU spends more time dealing with page faults rather than performing useful work. Hence, beyond a certain threshold, **increasing the degree of programming reduces CPU utilization.** This is referred to as **Thrashing.**



Working Set Model

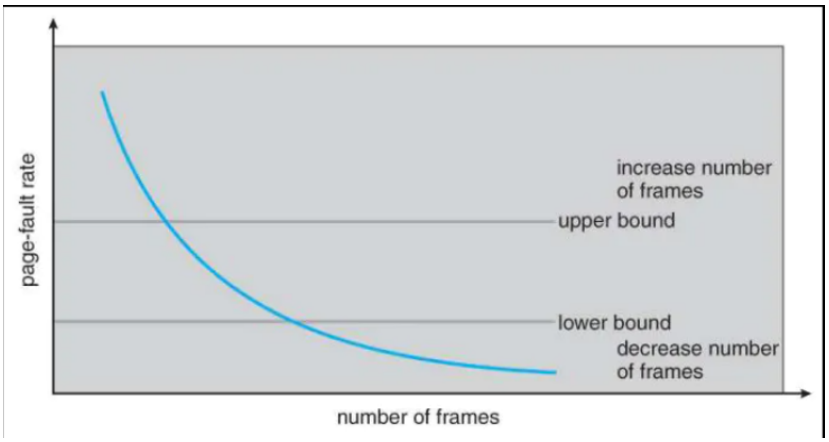
This is a simple solution to thrashing. Basically, every process has a certain number of pages that are requested and used more frequently than others. So, the OS will ensure that these pages are present in the main memory frames so that during the frequent access, there are much lesser page faults and hence the amount of thrashing reduces.

Page Fault Frequency

In this case, the OS constantly checks the amount of page faults that are occurring for the various processes. Also, the OS defines a upper and a lower bound for the page fault percentage. If the page fault goes above the upper bound, the OS **dynamically allocates more frames** for that process to **reduce the page fault percentage.**

On the other hand, if the page fault goes below the lower bound, the OS **dynamically de-allocates frames** for that process to **increase the page fault percentage** so that it may be above the lower bound. This will enable the OS to allocate additional frames for other processes that need it.

In short, OS dynamically changes amount of frames allocated to processes such that each process has a controlled page fault rate and lowered thrashing.



MULTI – LEVEL PAGING

As we know, the page table is also stored in the main memory on frames. Suppose the frame size is 1KB while the page table size is 32KB. In that case, we would need **32 frames** to store the page table. So now, we have scattered **the page table over 32 frames in a non – contiguous manner.**

Hence, we need **another page table** to store the location of the original page table which in turn stores the location of the process pages. This is called **2 – level paging** and the general concept is referred to as **multi – level paging.** Usually, the 2nd level page table is also referred to as the **Outer Page table.**

Let us understand this using an example. Let's assume –

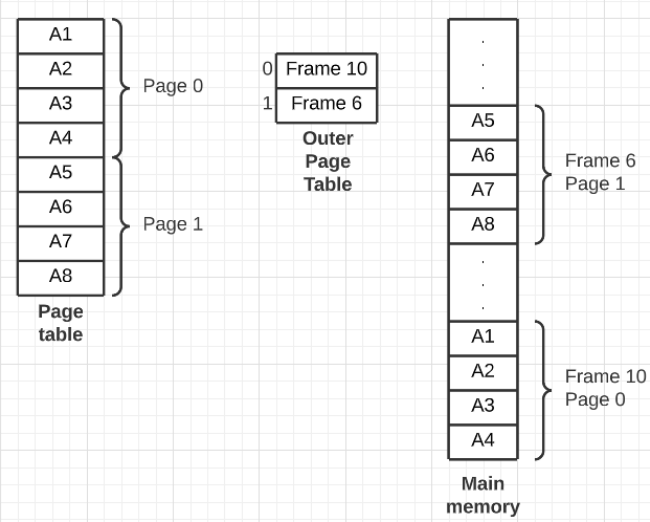
- Process size = 8KB
- Page size = 4KB
- Page table entry size = 1KB

In this case, we can see that the 8KB process will be divided into 2 pages of 4KB size each. Each page will have 4 entries of 1KB each. Now, we also know that –

$$Page\ size = Frame\ size = 4KB$$

So, the page table needs to be stored in **2 frames.** Till now, we were not facing the issue of multi – level paging since we were assuming the page table is stored in a **contiguous manner.** So all we had to do was store the base address of the page table in the **PTBR register** and then read in a contiguous manner. However, now we assume that the page table is stored in a **non – contiguous manner** and so, PTBR register can't store the base address anymore.

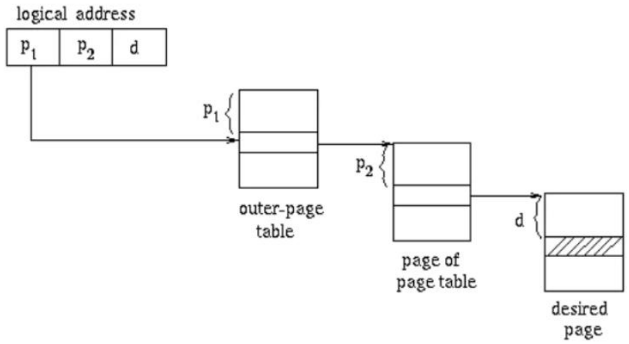
To solve this, we create **an outer page table** which will have 2 entries and whose base address will be in the PTBR register.



Now, the logical address will be divided into **three parts –**

Outer page table Page no.	Inner page table entry	Byte offset

In this case, **PTBR will store the base address of the Outer Page table.**



$$Number\ of\ pages\ required\ to\ store\ outer\ table = 1$$

$$Number\ of\ pages\ required\ to\ store\ inner\ table = 2^{p_1}$$

<u>Question</u>

Assume,

- Process size = 2KB
- Page size = 32B
- Page table entry size = 2B

Find the total memory required to store the pages.

Answer

$$No\ of\ pages = \frac{2K}{32B} = 64$$

Hence,

$$p_1 + p_2 = 6\ bits$$

Now,

$$No\ of\ entries\ in\ 1\ page = \frac{32B}{2B} = 16$$

Therefore, we need **4 bits** to search for individual entries in a page. Hence,

$$p_2 = 4$$

$$p_1 = 2$$

Therefore,

$$Total\ number\ of\ pages = 2^2 + 1 = 5$$

$$Size\ of\ memory\ required = 5 * 32B = \mathbf{160B}$$

Question

Assume,

- Page size = 1KB
- Virtual address = 30 – bits
- Page table entry size = 4B

Find the number of levels required for Page table.

Answer

$$Byte\ offset = \log_2(1K) = 10\ bits$$

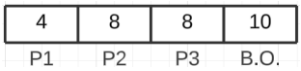
Therefore,

$$p = 30 - 10 = 20\ bits$$

We also know,

$$No.of\ entries\ in\ 1\ page = \frac{1KB}{4B} = 256$$

Hence, for every level we need 8 bits. So, the address would look like –



Hence, it needs **3 level paging.**

INVERTED PAGE TABLE

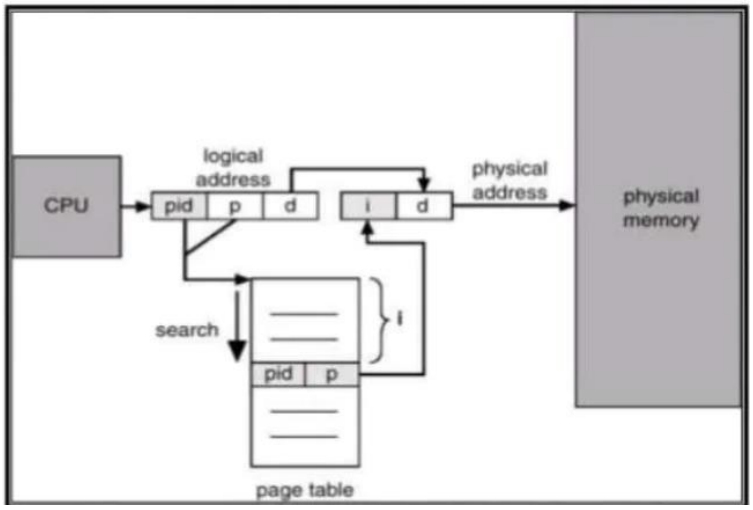
Normally, in a page table, each entry is the frame number on which the page is stored and each page gets its own entry. For example, let's say that there are 256 pages. Then the page table will have 256 entries. However, suppose the main memory has only 32 frames. Then, out of the 256 entries in the page table, only 32 will be valid at any given time and the rest 224 entries will be invalid. If that be the case, then why store them?

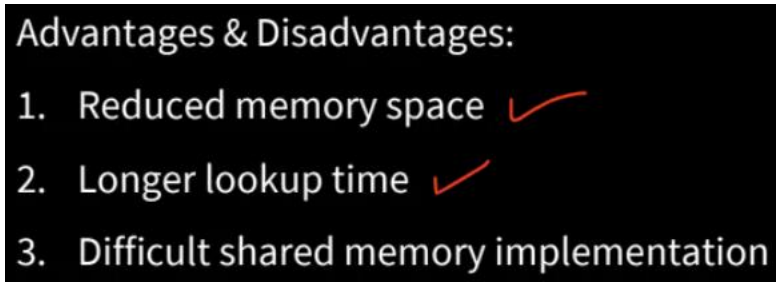
This is where the inverted page table comes in. Instead of storing the frame number for each page entry, we store the page number associated with each frame. So, for the above example the inverted page table will have only 32 entries out of which each entry will correspond to a frame number and the value will be the page being mapped to it.

This makes the storage much more efficient and smaller. However, there is a catch. The access becomes more complicated as the inverted page table has the pages from **all processes.** So, apart from the logical address, the CPU needs to provide the process ID as well. Hence, each entry will have the following information –

- Page number
- Process ID
- Control bits
- Chained pointer

Once the CPU sends the process ID and the page number, we traverse the inverted page table linearly to check for hit and miss/fault.





HASHED PAGE TABLE

A hash page table is basically treating the page table as a hash. To accommodate all the pages in the table without any collisions, the hash table implements **chaining.**

MULTI – LEVEL PAGING AND TLB

For a n – level paging,

$$Main\ memory\ access\ time\ (wihout\ TLB) = (n+1) * t_{MM}$$

We can confirm this by checking with regular paging. Regular paging has $n = 1$ (single level). Thus, the memory access time is taken as $2 * t_{MM}$ which we have seen before as well. Now,

$$Memory\ access\ time\ (with\ TLB) = H(t_{TLB} + t_{MM}) + (1-H)(t_{TLB} + (n+1)t_{MM})$$

Now, if we also include the fact that page fault ratio is p, we get –

$$Avg\ mem\ time = H(t_{TLB} + t_{MM}) + (1-H)(t_{TLB} + n * t_{MM} + [p * Service\ time + (1-p) * t_{MM}])$$

Question

Assume,

- TLB access time = 10ns
- Main memory access time = 200ns
- TLB hit rate = 80%
- Page Fault = 1%
- Page fault service time = 1000μs
- Single level paging

Find average memory access time.

Answer

$$Avg\ mem\ time = [0.8 * (10 + 200)] + [0.2 * (10 + 200 + \{0.01 * 1000000 + 0.99 * 200\})]$$
$$= \mathbf{2249.6ns}$$

Question

Assume,

- TLB access time = 20ns
- Main memory access time = 200ns
- TLB hit rate = 90%
- Page Fault = 0.1%
- Page fault service time w/o dirty page = 1000μs

- Page fault service time w/o dirty page = 2000μs
- Dirty page probability = 5%
- Single level paging

Find average memory access time.

Answer

$$EMAT = [0.9 * (20 + 200)] + \{0.1 * [20 + 200 + (0.999 * 200 + 0.001 * \{0.95 * 10^6 + 0.05 * 2 * 10^6\})]\}$$

$$\boldsymbol{EMAT = 344.98\ ns}$$

Question

Assume,

- TLB access time = 10ns
- Main memory access time = 300ns
- TLB hit rate = 95%
- Page Fault = 0.01%
- Page transfer time between memory and HDD = 5000μs
- Single level paging

Find average memory access time.

Answer

$$EMAT = [0.95 * (10 + 300)] + \{0.05 * [10 + 300 + (0.9999 * 300 + 0.0001 * 5 * 10^6)]\}$$
$$= \boldsymbol{349.9985\ ns}$$

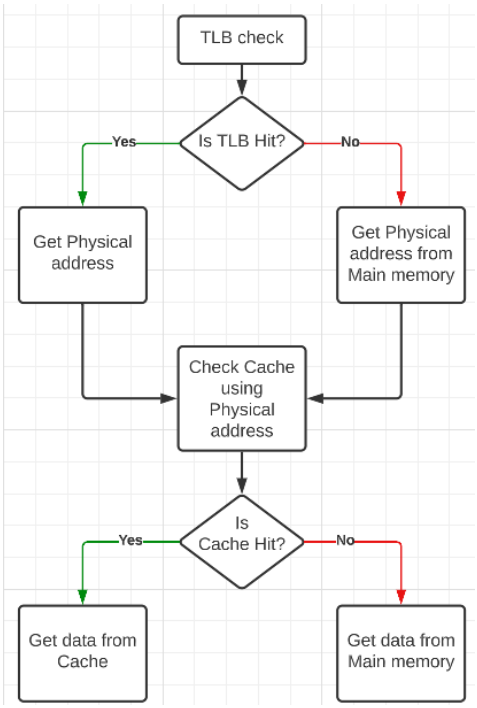
NOTE

In the case of regular page table, it is possible that we can have two separate page tables with the same page in the same frame. This is called **Page sharing.** However, in the case of inverted page table, we are not only storing the page number but also the process id which **is unique.** Hence, it is **difficult to implement page sharing in inverted page table.**

TLB WITH CACHE

1. First we check the TLB to get the physical address
2. Then, we use that physical address to check if the entry is there in cache or not

For our discussion, we are restricting to only the case where the cache is physically addresses and not where the cache is virtually addresses.



FILED SYSTEM

A **File** is a **named collection** of **related info** that is recorded on **secondary storage.** Every file must have the following attributes –

1. Name
2. Extension
3. Size
4. Date
5. Author
6. Created, Modified, Accessed
7. Attributes: Read-only, hidden
8. Default Program
9. Security Details

A **File System** is a **OS module** which is responsible for managing, controlling and organizing files and related structures. To understand the File system concept, we first need to understand **Disk Partitioning.** The Disk partition can be done in 2 ways –

- **Physical –** This is done by the manufacturer and is when the disk is divided into sectors, tracks and surfaces. Also called a **low – level** partition.
- **Logical –** This is done by the user where the disk will be logically partitioned. These logical partitions are also referred to as **Drives.** Also called a **high – level** partition. The drives/partitions can be of 2 types –

- **Primary –** This can store both OS and user files. Also referred to as **C drive** in windows.
- **Extended –** This can store only the user files. These are the **D, E, F** and other such drives.

NOTE

When we perform logical partitioning, the sectors are grouped either in groups of 1 or 2 and these groups are called **Disk Blocks.**

$$No\ of\ disk\ blocks = \frac{Disk\ size}{Disk\ block\ size}$$

FILE DIRECTORY STRUCTURE

There are three possible file directory structure –

- **Single level –** In this case, there is a single directory for all files. Hence, no matter what the file type is, **each file must have a unique name.**
- **Two level –** In this case, the outer directory is a directory of users. Each user in turn has a separate single level directory
- **Tree –** The directory is a directory of drives and in each drive, we can have nested folders and files. This is what is being implemented in modern systems.

FREE SPACE MANAGEMENT

The free space management in the OS is used to keep a track of the number of free blocks in the disk. This can be tracked using 2 methods –

- **Free List –** In this case, a linked list is created of all the free blocks. As and when data is pushed or released, the linked list is updated.
- **Bitmap –** For each block, an additional bit is stored which indicates whether the block is occupied (bit = 0) or free (bit = 1).

In general, free list is faster as it has only the free blocks. So just pick the 1st one and be done with. On the other hand, in the bitmap each block can either be free or occupied. So, in bitmap we need to traverse and search the blocks to encounter the first free block. However, the size of bitmap doesn't change as the number of blocks remains constant. On the other hand, the number of free blocks keeps changing and hence the size of Free List will be variable.

$$Size\ of\ bitmap = No\ of\ disk\ blocks = \frac{Disk\ size}{Disk\ block\ size}$$

Question

A particular disk unit uses a bit string to record the occupancy or vacancy of its disk blocks with '0' denoting vacant block and '1' denoting occupied block. A 32-bit part of this string has Hexadecimal value of D4F2A001. The percentage of occupied blocks on the disk for this part is?

Answer

It is given in the question that a 32 – bit Bitmap array is implemented. The current value of the bitmap –

$$(D4F2A001)_{16} = (11010100111100101010000000000001)_2$$

As seen, there are **12 occupied blocks** out of a total 32 blocks. Hence,

$$Occupied\ blocks\ perc = \frac{12}{32} * 100 = \mathbf{37.5}\%$$

Question

A system directory is kept in 4 disk blocks each of size 2Kbytes. It is a single level-directory and each directory entry is of size 32-bits. The maximum number of files possible in this system is?

Answer

It is given that the directory is a single – level directory. Hence, each file has its own directory entry. Now, it is also given that the directory entries are stored in 4 blocks of 2KB each. Hence,

$$Size\ of\ entire\ directory = 4 * 2K = 8KB$$

It is also given,

$$Size\ of\ each\ directory = 32b = 4B$$

Therefore,

$$No\ of\ files = No\ of\ directory\ entries = \frac{8K}{4} = \mathbf{2^{11}}\ \boldsymbol{files}$$

Question

There is a file of size 2562 bytes while the disk block size is 100 bytes. For each block on which the file is stored, the file system stores 2 bytes file system entry. What is the total number of blocks needed to store the file?

Answer

$$No\ of\ blocks\ for\ just\ file\ data = \frac{2562}{100} = 25.62 \approx 26$$

We also know that for each block, we store 2B of system data. Hence,

$$System\ data = 26 * 2 = 52B$$

Thus, to store additional 52B, we need one more block. Therefore,

$$\boldsymbol{Total\ no\ of\ blocks = 27}$$

Question

Assume a directory with the entry size per block as 4 bytes and each block is of 100 bytes. Suppose there are 3 files –

- File 1 = 5629 bytes
- File 2 = 10212 bytes
- File 3 = 9236 bytes

Find the total size needed to store the files.

Answer

$$Blocks\ for\ File\ 1 = \frac{5629}{100} = 56.29 \approx 57$$

$$Blocks\ for\ File\ 1\ directory = \frac{57 * 4}{100} \approx 3$$

$$Blocks\ for\ File\ 2 = \frac{10212}{100} = 102.12 \approx 103$$

$$Blocks\ for\ File\ 2\ directory = \frac{103 * 4}{100} \approx 5$$

$$Blocks\ for\ File\ 3 = \frac{9236}{100} = 92.36 \approx 93$$

$$Blocks\ for\ File\ 3\ directory = \frac{93 * 4}{100} \approx 4$$

$$Total\ number\ of\ blocks = 57 + 3 + 103 + 5 + 93 + 4 = \boldsymbol{265\ blocks}$$

Hence, memory required $\boldsymbol{26.5\ KB}$

FILE ALLOCATION METHODS

These are the methods to store the files in the file system. There can be of three types –

Contiguous Allocation

In this case, the file is stored in continuous or contiguous blocks. For example, if there is a file that requires 3 blocks and has a starting from Block number 11, then the file will be stored in blocks 11, 12 and 13.

In contiguous allocation, suppose we have a process that requires 520 bytes of memory and each block in the disk is of 100B. Then, we will need 6 blocks to store the file. However, in the 6th block, we will be using only 20B of data and the rest 80B is useless. Thus, this is a classic case of **internal fragmentation.**

Additionally, we know that the above process needs 6 blocks to be stored. Suppose there is a case where 6 blocks are free in the disk but there are not consecutive. In that case, even if there are 6 blocks free in the disk, the above process can't be stored. This is a classic case of **External fragmentation.**

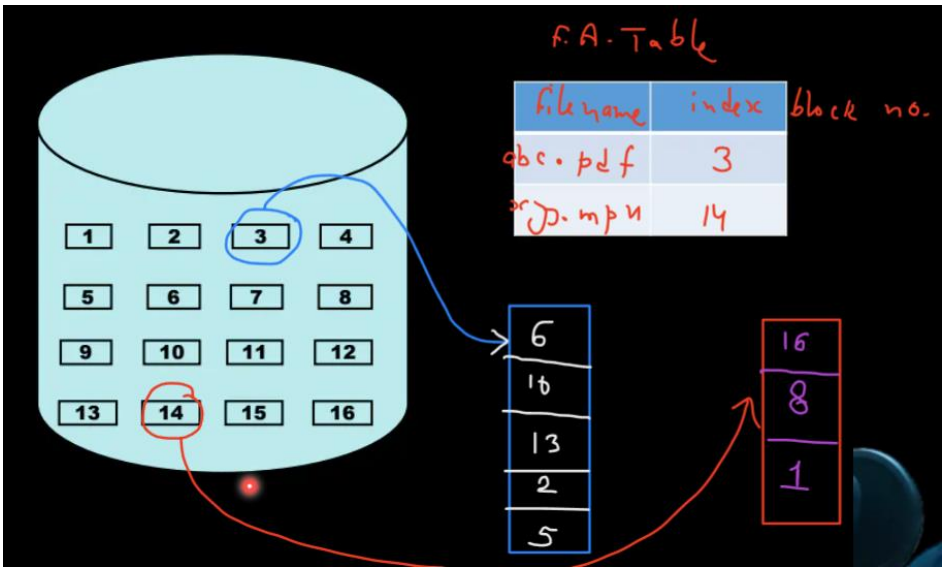
If let's say the above process is stored in 6 contiguous blocks. Now, the process size increases by 120B to be a total of 640B. In that case, we need to have 7 contiguous blocks to store the process. However, we may not have 7 contiguous blocks in the disk. Hence, **file size increase is very inflexible**.

Finally, since we know that the file is stored in contiguous manner, we know exactly which part of the process will be stored in which block. So, we can perform **sequential or random access**.

Linked Allocation

This is basically a case where all the blocks are in a linked list. So every block will store process data and at the same time will also store the address of the next block. In this case, there is no **external fragmentation** because as long as there is free block, we can use it. However, **internal fragmentation** will still be present as the last block may not be completely full. Also, since we can use any free block, the **increase in file size also becomes flexible.** Finally, we can **only have sequential access** as the next block address is present in the previous block.

Indexed Allocation

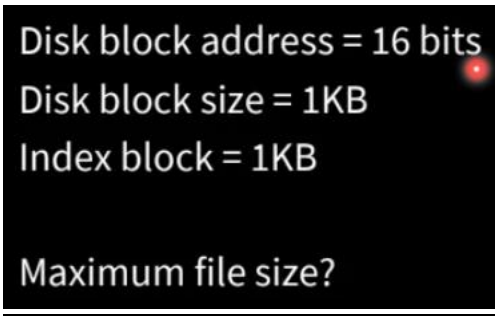


In this case, there is an index table where for every file, a specific block is indicated. This is the index block. The data in this block is the block numbers where the actual data is stored. So, in the above diagram, for file $abc.pdf$, the index block is Block 3. When we navigate to block 3, we see that the blocks stored in it are Blocks 6, 10, 13, 2 and 5. The data in the file is stored in those blocks in that sequence.

This method **resolves external fragmentation** and at the same time, **improves file size increase flexibility.** Even though it still **suffers from internal fragmentation,** the data can be **accessed sequentially or randomly.**

An additional problem with this method is that it uses up one entire block as an index block so the blocks are not efficiently used to store the files.

Question



Answer

$$Block\ address\ sizez = 16\ bits = 2B$$

Hence,

$$Index\ block\ can\ store = \frac{1KB}{2} = 512\ block\ address$$

Therefore, the maximum number of blocks that can be used for a file is 512. Hence,

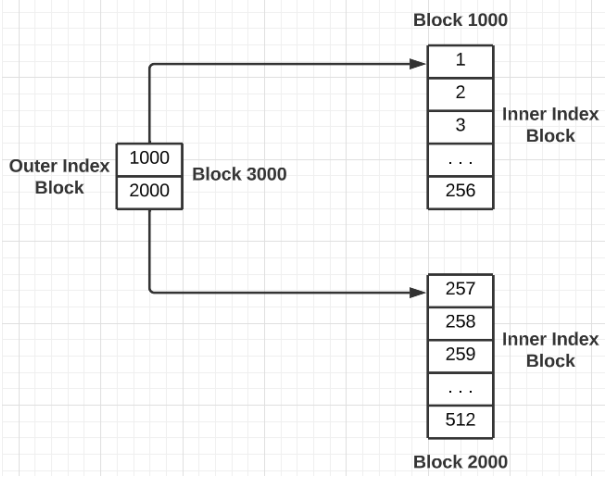
$$Maximum\ file\ size = 512 * 1KB = \mathbf{512KB}$$

MULTI – LEVEL INDEXING

Let's take the case where we have each disk of size 256 bytes and the block address size is 8 bits. In that case, each index block can store 256 blocks. Thus,

$$Max\ file\ size = 256 * 256B = 64KB$$

So, using 1 index block, we can store a max file of 64KB. However, what if the file size is 128KB? In this case, we need 2 index blocks and this is called **2 – level indexing.**



So now, with this arrangement we get –

- Maximum number of blocks in outer index block = 256
- Maximum number of blocks in inner index block = 256

Hence,

$$Max\ number\ of\ blocks = 256 * 256 = 64K$$

Therefore,

$$Max\ file\ size = 64K * 256B = \mathbf{16MB}$$

Question

- Disk block number = 32 bits
- Disk block size = 2KB

Find the max file size in 2 – level indexing.

Answer

$$Disk\ address = 32\ bits = 4B$$

Hence,

$$Index\ block\ contents = \frac{2KB}{4B} = 512\ blocks$$

Hence,

$$Max\ file\ size = 512 * 512 * 2K = \mathbf{512\ MB}$$

UNIX I – NODE

Unix Index Node (INODE) is a data structure in a Unix File System that is used to describe a file object.

Every file has its own INODE structure. There are four main types of pointers in INODE –

- **Direct –** These are usually 12 pointers which will directly store the blocks where the file data is present.
- **Single Indirect –** In case the 12 pointers are not enough, then the single indirect pointer will store the block for 1 – level index mapping.
- **Double Indirect –** 2 – level mapping
- **Triple Indirect –** 3 – level mapping

Question

The index node (inode) of a Unix-like file system has 12 direct, one single-indirect and one double-indirect pointer The disk block size is 4 kB, and the disk block addresses 32-bits long. The maximum possible file size is (rounded off to 1 decimal place) _____ GB?

Answer

$$Disk\ block\ address = 32\ b = 4B$$

$$No\ of\ blocks\ in\ index\ block = \frac{4K}{4} = 1024$$

Using Direct Pointers

$$Max\ file\ size = 12 * 4K = 48KB$$

Using Single – Indirect Pointer

$$Max\ file\ size = 1K * 4K = 4MB$$

Using Double – Indirect Pointer

$$Max\ file\ size = 1K * 1K * 4K = 4GB$$

Hence,

$$Max\ file\ size\ (total) = 4GB + 4MB + 48KB \approx \mathbf{4.3\ GB}$$

DISK SCHEDULING

Since Disk is an I/O device, we need to schedule the disk requests. We consider the disk to have cylinders, surfaces, tracks and sectors as seen in COA.

First – come – First – serve (FCFS)

As the name suggests, the requests are served in the order in which they arrive. Suppose, we have –

$$Service\ request = 72 \rightarrow 160 \rightarrow 33 \rightarrow 130 \rightarrow 14 \rightarrow 6 \rightarrow 180$$

These are all cylinder numbers. Now, let's assume that the read/write header is initially at cylinder number 50. Then,

$$No\ of\ cylinder\ movements = (72-50) + (160-72) + (160-33) + (130-33) + (130-14) + (14-6) + (180-6) = \mathbf{632}$$

Now, suppose seek time is $0.5ms$, then

$$Total\ time\ taken = 632 * 0.5 = 316ms$$

Advantages:

- Every request gets a fair chance
- No indefinite postponement

Disadvantages:

- Does not try to optimize seek time
- May not provide the best possible service

Shortest Seek Time First (SSTF)

The movement that requires minimum seek time/cylinder movement is serviced first. Using the above example, we get the sequence as follows –

$$50 \rightarrow 33 \rightarrow 14 \rightarrow 6 \rightarrow 72 \rightarrow 130 \rightarrow 160 \rightarrow 180$$

From this, we get –

$$Total\ number\ of\ cylinder\ moves = \mathbf{218}$$

$$Total\ seek\ time = \mathbf{109\ ms}$$

This is much faster than FCFS.

Advantages:

- Average Response Time decreases
- Throughput increases

Disadvantages:

- Overhead to calculate seek time in advance
- Can cause Starvation for a request if it has higher seek time as compared to incoming requests
- High variance of response time as SSTF favors only some requests

Scan (Elevator)

In this case, we first proceed to complete the requests that are larger than the starting point and we reach till the max cylinder possible (199 in our example) and then, we switch direction to complete lower requests till we reach the min request.

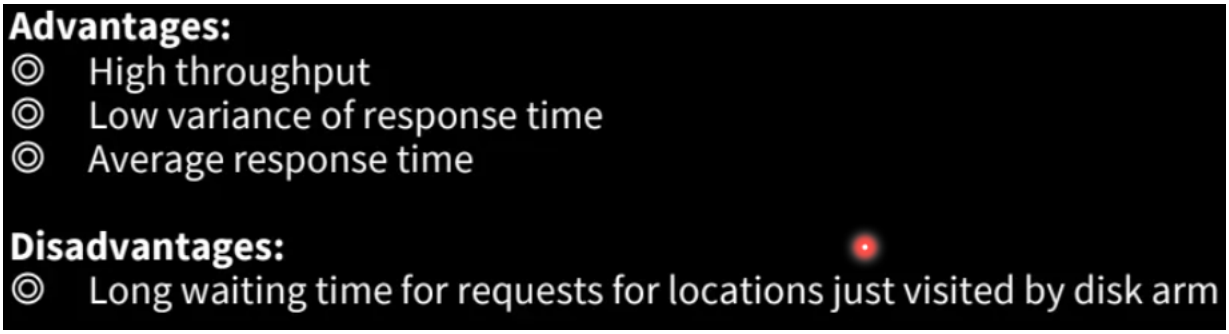
So, the sequence will become –

$$50 \rightarrow 72 \rightarrow 130 \rightarrow 160 \rightarrow 180 \rightarrow 199 \rightarrow 33 \rightarrow 14 \rightarrow 6$$

Here,

$$Total\ number\ of\ cylinder\ moves = (199 - 50) + (199 - 6) = \mathbf{342}$$

$$Total\ time\ taken = \mathbf{171}\ \boldsymbol{ms}$$



Circular Scan (C-Scan)

This is the same as the scan above, however the requests can only be fulfilled when we move from smaller to larger values. So once we hit 199, we need to move back to 0 and then start fulfilling the requests again.

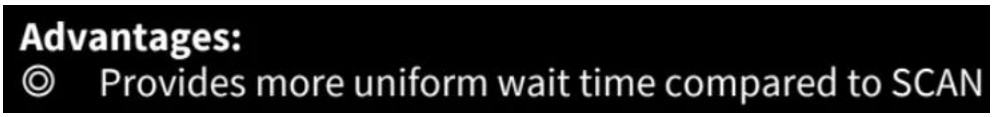
Hence,

$$50 \rightarrow 72 \rightarrow 130 \rightarrow 160 \rightarrow 180 \rightarrow 199 \rightarrow 0 \rightarrow 6 \rightarrow 14 \rightarrow 33$$

Thus,

$$Total\ number\ of\ cylinder\ moves = \mathbf{381}$$

$$Total\ seek\ time = \mathbf{190.5}\ \boldsymbol{ms}$$



Look

This is exactly like Scan but here, we don't need to go all the way to the end to turn the direction of access. So, the sequence will become –

$$50 \rightarrow 72 \rightarrow 130 \rightarrow 160 \rightarrow 180 \rightarrow 33 \rightarrow 14 \rightarrow 6$$

Here,

$$Total\ number\ of\ cylinder\ moves = (180 - 50) + (180 - 6) = \mathbf{304}$$

$$Total\ time\ taken = \mathbf{152}\ \boldsymbol{ms}$$

Circular Look (C – look)

This is exactly like C – Scan but here, we don't need to go all the way to the last cylinder. So, the sequence will become –

$$50 \rightarrow 72 \rightarrow 130 \rightarrow 160 \rightarrow 180 \rightarrow 6 \rightarrow 14 \rightarrow 33$$

Thus,

$$Total\ number\ of\ cylinder\ moves = \mathbf{331}$$

$$Total\ seek\ time = \mathbf{165.5}\ \boldsymbol{ms}$$

Question

- Total number of cylinders = 1 – 1000
- Current cylinder position = 61
- Requests = $\{236, 529, 331, 2, 5, 72, 50, 96, 381\}$

Find the number of head movements when SSTF scheduling is used.

Answer

Using SSTF, we get the sequence as –

$$61 \rightarrow 50 \rightarrow 72 \rightarrow 96 \rightarrow 5 \rightarrow 2 \rightarrow 236 \rightarrow 331 \rightarrow 381 \rightarrow 529$$

Thus,

$$No\ of\ head\ movements = \mathbf{678}$$

Question

In the above question, if the seek time is 2ms and it takes an additional 1ms when there is a change in direction. What is the total time taken?

Answer

$$No\ of\ direction\ changes = 3$$

$$Total\ time = (678 * 2) + (3 * 1) = \mathbf{1.359}\ \boldsymbol{s}$$

Question

Consider a process scenario in which each process executes first in CPU then goes for IO operation, then once again process needs a CPU bursts and then terminates. Following is given a process scenario in which for CPU execution system uses non preemptive SJF algorithm. Consider system has enough number of resources to carry out IO operations for only 2 processes in parallel at a time. What is the average waiting time for the execution for the processes?

PROCESS	ARRIVAL TIME	CPU BURST 1	I/O BURST TIME	CPU BURST 2
P1	0	2	8	3
P2	0	4	5	6
P3	0	3	7	3
P4	0	6	4	2

Answer

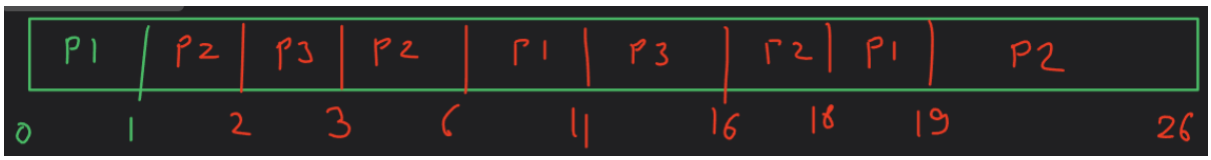
$$Avg\ waiting\ time = \frac{38}{4} = 9.5$$

Question

Consider a process scenario in which each process executes first in CPU then goes for IO operation, then once again process needs a CPU bursts and then terminates. Following is given a process scenario in which for CPU execution system uses preemptive SRTF algorithm. Consider system has enough number of resources to carry out IO operations for all processes in parallel at a time. What is the average waiting time for the execution for the processes?

PROCESS	ARRIVAL TIME	CPU BURST 1	I/O BURST TIME	CPU BURST 2
P1	0	6	7	1
P2	1	4	2	9
P3	2	1	6	5

Answer



$$Avg\ waiting\ time = 5.67$$

Question

Consider a following process scenario in which each process first executes on CPU for given time duration then goes for IO operations and then again executes on CPU before termination. CPU uses non-preemptive SJF algorithm and consider each process has its separate set of IO devices to work in parallel with other process.

PROCESS	ARRIVAL TIME	CPU BURST 1	I/O BURST TIME	CPU BURST 2
P1	0	6	4	7
P2	1	4	3	5
P3	2	5	5	2
P4	4	2	2	4

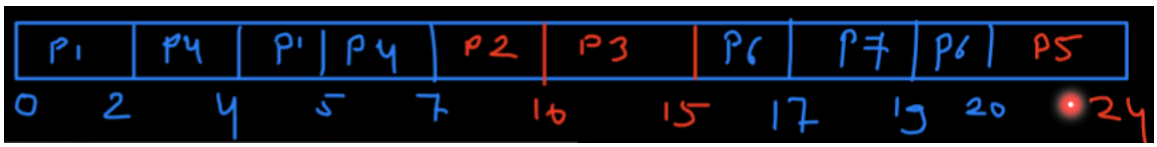
Answer

Question

Multilevel Queue Scheduling, with fixed priority preemptive algorithm
Queue 1: RR with Q=2
Queue 2: SJF

Process	Arrival Time	Burst Time	Queue
P1	0	3	1
P2	1	3	2
P3	2	5	2
P4	1	4	1
P5	11	4	2
P6	15	3	1
P7	16	2	1

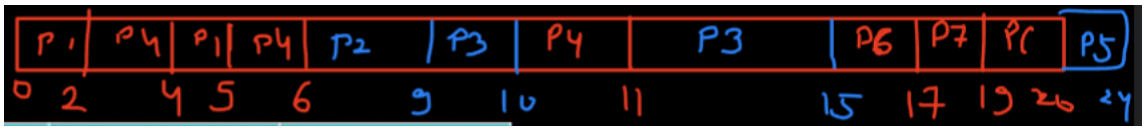
Answer



Question

Suppose in the above question, the queue Q1 is given a 60% time slice and Q2 is given a 40% time slice in every 10 – unit time.

Answer



Question

A computer system has 2GB of RAM and OS occupies 256MB of RAM. All the processes are of 128MB and have same characteristics. If the goal is 99% CPU utilization, then the maximum I/O wait that can be tolerated?

Answer

We know,

$$Total\ RAM = OS + (n * Size\ of\ a\ process)$$

Thus,

$$2^{31} = 2^{28} + (n * 2^{27})$$

$$2^{28} * (2^3 - 1) = n * 2^{27}$$

$$\boldsymbol{n = 14}$$

We also know that,

$$CPU\ utilization = 1 - p^n$$

Thus,

$$0.99 = 1 - p^{14}$$

$$\boldsymbol{p \approx 72\%}$$

Question

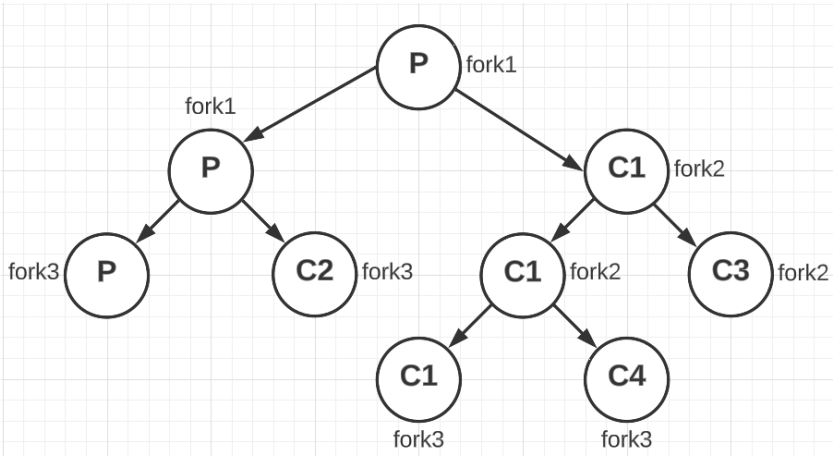
```
#include <stdio.h>
#include <unistd.h>
int main()
{
   if (fork() || fork())
       fork();
   printf("1 ");
   return 0;
}
```

Answer

The important concept here is that when a $fork()$ function is used, the parent function returns 1 and the child function returns 0. Let us also re-label the various fork functions in the code above –

```
if (fork1() || fork2())
    fork3();
```

Here is how the code will execute –



Since there are a final 5 processes, the output will be **1 1 1 1 1.**

Question

The following two functions $P1$ and $P2$ that share a variable B with an initial value of 2 execute concurrently.

```
P1() {
    C = B - 1;
    B = 2 * C;
}
```

```
P2(){
    D = 2 * B;
    B = D - 1;
}
```

The number of distinct values that B can possibly take after the execution is_____.

Answer

$$B = 2,3,4$$

Question

Consider three concurrent processes P1, P2 and P3 as shown below, which access a shared variable D that has been initialized to 100.

P1	P2	P3
: : D = D + 20 : :	: : D = D – 50 : :	: : D = D + 10 : :

The processes are executed on a uniprocessor system running a time-shared operating system. If the minimum and maximum possible values of D after the three processes have completed execution are X and Y respectively, then the value of Y – X is _____.

Answer

$$X = 50 \ ; \ Y = 130$$

Hence,

$$Y - X = \mathbf{80}$$

Question

Assume that the OS is using a Strict Priority Scheduler. Thread A (priority 30) is currently holding Lock 1, and Thread B (priority 60) is waiting to acquire Lock 1. If a third thread (Thread C) is introduced to the system, which of the following priorities for Thread C will lead to priority inversion? (Assume that higher number represent higher priority)

(a) 20 (b) 40 (c) 50 (d) 70

Answer

Since Thread A is holding the lock, it can't be preempted. Therefore, if any thread comes after Thread A, it will have to wait irrespective of the priority. Therefore, for priority inversion, the incoming thread should be of a higher priority when compared to Thread A. Thus, **b), c)** and **d) are the correct options.**

Question

Consider a machine with 64 MB physical memory and a 32-bit virtual address space. If the page size s 4 KB, what is the approximate size of the page table?

1. 16 MB
2. 8 MB
3. 2 MB
4. 24 MB

Answer

$$Byte\ offset = \log_2(page\ size) = \log_2(4K) = 12\ bits$$

Hence,

$$Frame\ address = Physical\ address - Byte\ offset = \log_2(64M) - 12 = 14\ bits$$

Thus,

$$Page\ table\ size = No\ of\ page\ table\ entries * Size\ of\ Frame\ address$$

$$Size\ of\ page\ table\ address = Virtual\ address - Byte\ offset = 32 - 12 = 20$$

Therefore,

$$Paage\ table\ size = 2^{20} * 14 = 14Mb \approx \mathbf{2MB}$$

Question

A computer system implements 8 kilobyte pages and a 32−bit physical address space. Each page table entry contains a valid bit, a dirty bit, three permission bits, and the translation. If the maximum size of the page table of a process is 24 megabytes, the length of the logical address supported by the system is _____ bits?

Answer

$$Byte\ offset = \log_2(8k) = 13\ bits$$

Therefore,

$$Frame\ address = 32 - 13 = 19\ bits$$

From the question, we get –

$$Each\ page\ table\ entry = 19 + 1 + 1 + 3 = 24\ bits$$

We know that,

$$Page\ table\ size = Page\ entry\ size * 2^{Page\ address}$$

$$24 * 2^{23}\ bits = 24 * 2^{Page\ address}$$

$$Page\ address = 23\ bits$$

Hence,

$$Logical\ address = 23 + 13 = \mathbf{36\ bits}$$

Question

Which of the following statements is/are true when the kernel switches execution of the current process to another process? (MSQ)

(a) The kernel must save the register values (including IP (Instruction Pointer), SPO pointer) etc.) of the current process to memory.

(b) The kernel must save the content of the current process's memory on disk.

(e) The kernel must close all the files opened by the current process.

(d) The kernel must make the page table of the next process active.

Answer

When there is a context switch, then the kernel basically needs to pause the current program execution, switch to a different process and then return back to the current process to resume the execution. Hence, there is a need to store the register value. Thus, **Option a) is correct.**

When a kernel changes contexts, it saves the IP, SP and other register values but it doesn't need to save the memory to the disk. Hence, **Option b) is incorrect.**

Let us say that we are using MS Word and then we switch to Google Chrome. That doesn't mean that the OS closes MS word. So yeah, **Option c) is incorrect.**

Finally, to execute the next process after a context switch, the kernel needs to form the PCB and for that, it needs to have the process information. Hence, there is a need to make the page table active and get the value from main memory. This, **Option d) is correct.**

Question

Consider a fixed partition MMT where there are 5 partitions of size 100MB, 250MB, 200MB, 500MB and 300MB. All Partitions are initially empty. The following process requests are made in the given order:

Process	Size
P1	150MB
P2	400MB
P3	270MB
P4	180MB
P5	80MB

Find –

1. The degree of programming for Best, First and Worst fit.
2. The internal fragmentation for the First, Best and Worst fit.

Answer

METHOD	DEGREE OF MULTI-PROGRAMMING	INTERNAL FRAGMENTATION
First Fit	5	270
Best Fit	5	270
Worst Fit	4	570

Question

Consider variable partition MMT where there are 4 partitions of size 250MB, 200MB, 500MB and 400MB. The following process requests are made in the given order:

Process	Size
P1	150MB
P2	400MB
P3	270MB
P4	180MB
P5	80MB
P6	50MB

Provide the processes are stored for First fit, Best fit and Worst Fit policies?

Answer

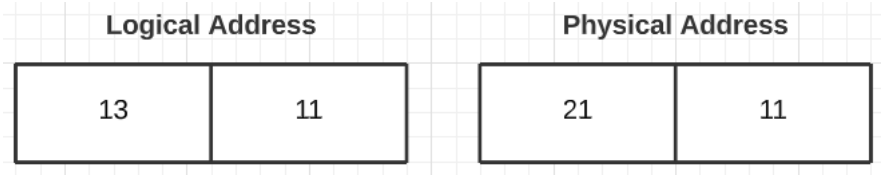
All three processes will be able to provide to all the processes.

Question

Consider a paged memory system where the process size is 16MB and main memory size is 4GB. The page size is 2KB.

1. Number of pages in process?
2. Number of frames in main memory?
3. Number of bits for page number?
4. Number of for frames?
5. Number of entries in page table?
6. Page table size?

Answer



1. 2^{13}
2. 2^{21}
3. 13
4. 21
5. $2^{13} = 8K$
6. $2^{13} * 21$

Question

Consider a process with 10 pages and a main memory with 3 frames. Assume that the sequence of pages is as follows –

$$0, 1, 2, \dots, 9, 0, 1, \dots, 9$$

What is the difference in the number of page faults for FIFO policy and optimal policy?

Answer

For FIFO, we have –

INCOMING PAGE	FRAMES			RESULT
	0	1	2	
0	0	-	-	Fault
1	0	1	-	Fault
2	0	1	2	Fault

3	3	1	2	Fault
4	3	4	2	Fault
5	3	4	5	Fault
6	6	4	5	Fault
7	6	7	5	Fault
8	6	7	8	Fault
9	9	7	8	Fault
0	9	0	8	Fault
1	9	0	1	Fault
2	2	0	1	Fault
3	2	3	1	Fault
4	2	3	4	Fault
5	5	3	4	Fault
6	5	6	4	Fault
7	5	6	7	Fault
8	8	6	7	Fault
9	8	9	7	Fault

Hence, when we use FIFO we get **20 page faults.** For Optimal, we have –

INCOMING PAGE	**FRAMES**			**RESULT**
	0	**1**	**2**	
0	0	-	-	Fault
1	0	1	-	Fault
2	0	1	2	Fault
3	0	1	3	Fault
4	0	1	4	Fault
5	0	1	5	Fault
6	0	1	6	Fault
7	0	1	7	Fault
8	0	1	8	Fault
9	0	1	9	Fault
0	0	1	9	Hit
1	0	1	9	Hit
2	2	1	9	Fault
3	3	1	9	Fault
4	4	1	9	Fault
5	5	1	9	Fault
6	6	1	9	Fault
7	7	1	9	Fault
8	8	1	9	Fault
9	8	1	9	Hit

Hence, with Optimal policy, we get **17 page faults.** Thus,

$$Difference = 20 - 17 = \mathbf{3}$$

Question

Assume there are 8 pages and there are 4 frames in the main memory. The sequence is as follows –

$$1, 2, \ldots, 8, 1, 2, \ldots, 8, 1, 2, \ldots, 8$$

Find the number of page faults using

- FIFO
- Optimal
- LIFO
- MRU

Answer

- FIFO = 24
- Optimal = 16
- LIFO = 18
- MRU = 17

Question

Consider a computer system with ten physical page frames. The system is provided with an access sequence $a_1, a_2, \ldots, a_{20}, a_1, a_2, \ldots, a_{20}$ where each a_i number. The difference in the number of page faults between the last-in-first-out page replacement policy and the optimal page replacement policy is _____

Answer

Number of page faults for Optimal = **30**

Number of page faults for LIFO = **31**

Hence, difference = **1**

Question

A computer has twenty physical page frames which contain pages numbered 101 through 120. Now a program accesses the pages numbered 1, 2, …, 100 in that order, and repeats the access sequence THRICE. Which one of the following page replacement policies experiences the same number of page faults as the optimal page replacement policy for this program?
(A) Least-recently-used
(B) First-in-first-out
(C) Last-in-first-out
(D) Most-recently-used

Answer

The answer is None of the above.

Question

The following page addresses, in the given sequence, were generated by a program:

1 2 3 4 1 3 5 2 1 5 4 3 2 3

This program is run on a demand paged virtual memory system, with main memory size equal to 4 pages. Indicate the page references for which page faults occur for the following page replacement algorithms.

A. LRU
B. FIFO

Assume that the main memory is initially empty

Answer

LRU = 9 page faults

FIFO = 8 page faults

Question

A memory page containing a heavily used variable that was initialized very early and is in constant use is removed then

A. LRU page replacement algorithm is used

B. FIFO page replacement algorithm is used

C. LFU page replacement algorithm is used

D. None of the above

Answer

Since a variable is frequently used, then LRU or LFU will not replace it. In FIFO on the other hand, if there is a variable it will be replaced sooner or later. Hence, **FIFO** is the logical answer.

Question

Which of the following page replacement algorithms suffers from Belady's anamoly?

A. Optimal replacement

B. LRU

C. FIFO

D. Both (A) and (C)

Answer

Option C is the correction option.

Question

The address sequence generated by tracing a particular program executing in a pure demand based paging system with 100 records per page with 1 free main memory frame is recorded as follows. What is the number of page faults?

0100, 0200, 0430, 0499, 0510, 0530, 0560, 0120, 0220, 0240, 0260, 0320, 0370

Answer

Number of page faults = **7**

Question

Dirty bit for a page in a page table

A. helps avoid unnecessary writes on a paging device
B. helps maintain LRU information
C. allows only read on a page
D. None of the above

Answer

Option A is the correct answer

Question

When a program tries to access a page that is mapped in address space but not loaded in physical memory, then _____?
a) segmentation fault occurs
b) fatal error occurs
c) page fault occurs
d) no error occurs

Answer

The question says that a page is present in the page table but not in the frames of the main memory. So when accessed, there will be a **page fault** and the page will have to be accessed from the secondary memory and brought to the main memory. Thus, **Option C** is the correct answer.

Question

Effective access time is directly proportional to _____
a) page-fault rate
b) hit ratio
c) memory access time
d) none of the mentioned

Answer

If the **page fault rate increases**, then more time is taken to get a page from the secondary memory to the main memory and hence, **effective access time increases.**

If the **hit rate increases,** then more pages can be directly accessed from the page table itself and there is no need to access the secondary memory. Hence, **effective access time decreases.**

If the **memory access time increases,** then of course the **effective access time increases.**

Hence, **Option A and C are correct.**

Question

It is advantageous for the page size to be large because:
(A) Less unreferenced data will be loaded into memory
(B) Virtual address will be smaller
(C) Page table will be smaller
(D) Large program can be run

Answer

When there is a page fault, then we would need to pull a page from the secondary memory to the main memory. Hence, if the page size is large, **more unreferenced data will be loaded into memory.** Also, if we increase page size, it doesn't change the size of the page table or the virtual memory. If the page size increases, then simply the number of pages will reduce. Thus, the size remains the same and also **virtual address will be the same.** No matter if the page size is small or large, **any size program can be run.** Finally, the one correct answer is that **page table will be smaller** as there are very few pages since each page is larger.

Question

A demand paged memory environment has physical memory access time of 50 microseconds and page fault service time of 10 milliseconds. If the page fault rate is 5% then the effective memory access time is _____ microseconds?

Answer

$$Avg\ access\ time = (0.95 * 2 * 50) + (0.05 * 10000) = \mathbf{595}\boldsymbol{\mu s}$$

Question

A demand paged memory environment has physical memory access time of 50 microseconds and page fault service time of 5000 microseconds if the replaced page is not dirty. The page fault service time of 100 milliseconds if a dirty page is replaced. Assume that among all pages which are getting replaced, only 2% are dirty, and 95% page hit ratio then the effective memory access time is _____ microseconds?

Answer

$$Avg\ access\ time = (0.95 * 2 * 50) + [0.05 * (0.98 * 5000 + 0.02 * 100 * 1000)] = \mathbf{440}\boldsymbol{\mu s}$$

Question

A main memory can hold 3 page frames and initially all of them are vacant. Consider the following stream of page requests :

2, 3, 2, 4, 6, 2, 5, 6, 1, 4, 6

If the stream uses FIFO replacement policy, the hit ratio h will be?

(a) 11/3
(b) 1/11
(c) 3/11
(d) 2/11

Answer

$$Hit\ ratio = \frac{2}{11}$$

Question

A virtual memory system has only 2-page frames which are empty initially. Using demand paging the following sequence of page reference is passed through this system.

9, 8, 7, 8, 7, 9, 7, 9, 8, 9

Minimum possible number of page faults?

Answer

$$Min\ page\ fault = \mathbf{5}$$

Question

Ques :- 12. Consider a binary search algorithm to search an element in an array of 'n' numbers. Assume that this array spans over multiple pages with each page holding 'p' elements (n>p). Every memory access will generate a page fault until the search range is less than 'p'. The minimum value of 'p' that reduces the page fault is

Answer

The minimum page fault we can have is **1.** To have 1 page fault, we need all the elements on the same page. Hence, the value of p to reduce the page fault should be $\boldsymbol{n}$**.**

Question

Locality of reference implies that the page reference being made by a process

A. will always be to the page used in the previous page reference
B. is likely to be to one of the pages used in the last few page references
C. will always be to one of the pages existing in memory
D. will always lead to a page fault

Answer

Option B is the correct answer

Question

Thrashing

A. reduces page I/O
B. decreases the degree of multiprogramming
C. implies excessive page I/O
D. improve the system performance

Answer

Page I/O is another way to refer to Page fault. Hence, thrashing **implies excessive page fault/page IO.**

Question

Consider a virtual memory system with FIFO page replacement policy. For an arbitrary page access pattern, increasing the number of page frames in main memory will

A. always decrease the number of page faults
B. always increase the number of page faults
C. sometimes increase the number of page faults
D. never affect the number of page faults

Answer

Option C is the correct answer as it refers to **Belady's anamoly.**

The optimal page replacement algorithm will select the page that

A. Has not been used for the longest time in the past
B. Will not be used for the longest time in the future
C. Has been used least number of times
D. Has been used most number of times

The minimum number of page frames that must be allocated to a running process in a virtual memory environment is determined by

A. the instruction set architecture
B. page size
C. number of processes in memory
D. physical memory size

Increasing the RAM of a computer typically improves performance because:

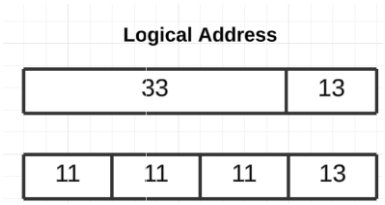
A. Virtual Memory increases
B. Larger RAMs are faster
C. Fewer page faults occur
D. Fewer segmentation faults occur

Question

Consider a virtual memory system with physical memory of 8GB, a page size of 8KB and 46-bit virtual address. Assume every page table exactly fits into a single page. If page table entry size is 4B then how many levels of page tables would be required.

Answer

Most of the above information is **ABSOLUTELY USELESS!!!** We can simply see from the VA address, we get –



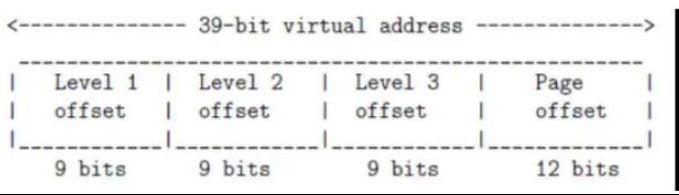
We get,

$$No\ of\ page\ entries = \frac{Page\ table\ size}{Page\ entry\ size} = \frac{8K}{4B} = 2^{11}$$

Thus, we can see that it requires **3 levels.**

Question

Consider a three-level page table to translate a 39-bit virtual address to a physical address as shown below:



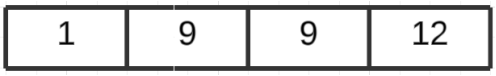
The page size is 4 KB = (1KB = 2^{10} bytes) and page table entry size at every level is 8 bytes. A process P is currently using 2 GB (1 GB = 2^{30} bytes) virtual memory which OS mapped to 2 GB of physical memory. The minimum amount of memory required for the page table of P across all levels is _____ KB

Answer

Considering,

$$p1 = p2 = p3 = 9\ bits$$

It is given that the process P has a 2GB virtual memory which relates to 31 bits. Hence, we can write the virtual memory of Process P as –



Hence,

$$Total\ number\ of\ pages = 1 + 2^1 + (2^1 * 2^9) = 1027$$

Hence

$$Total\ memory\ for\ page\ table = 1027 * 4K = \mathbf{4.108MB}$$

Question

Match the pairs in the following question.

List - I		List - II	
(A)	Virtual Memory	(p)	Temporal locality
(B)	Shared memory	(q)	Spatial Locality
(C)	Look-ahead buffer	(r)	Address Translation
(D)	Look-aside buffer	(s)	Mutual exclusion

Answer

$$A \rightarrow r$$

$$B \rightarrow s$$

$$C \rightarrow q$$

$$D \rightarrow p$$

Question

In a two-level virtual memory, the memory access time for main memory, $t_M = 10^{-8}$ sec, and the memory access time for the secondary memory, $t_D = 10^{-3}$ sec. What must be the hit ratio, H such that the access efficiency is within 80 percent of its maximum value?

Answer

For the best-case scenario –

- There should be a hit in the outer table (1 MM access)
- There should be a hit in the inner table (1 MM access)
- The page should be pulled from the respective frame from the main memory (1 MM access)

Hence,

$$Best\ case = 3 * t_M = 30\ ns$$

In the worst case (maximum value)

- We check the outer and inner tables (2 MM access) and there will be misses.
- Hence, we pull the page from secondary memory (1 SM access)

Hence, we get –

$$30 = 0.8 * \{(H * 30) + (1 - H) * (20 + 1000000)\}$$

Hence,

$$\boldsymbol{H = 99.99\%}$$

Question

Indicate all the false statements from the statements given below:

A. The amount of virtual memory available is limited by the availability of the secondary memory
B. Any implementation of a critical section requires the use of an indivisible machine- instruction ,such as test-and-set.
C. The use of monitors ensure that no dead-locks will be caused .
D. The LRU page-replacement policy may cause thrashing for some type of programs.
E. The best fit techniques for memory allocation ensures that memory will never be fragmented.

Answer

A. Virtual memory basically uses the help of secondary memory for implementation. This means that the implementation is restricted by secondary memory availability.
B. Critical section can be implemented however the programmer wants to design it. So, there is no need for just atomic instructions
C. Option C is Out of syllabus
D. Thrashing can occur in any page replacement policy
E. Best fit allocation faces little internal fragmentation when compared to other techniques but still has fragmentation.

Hence, the false statements are **B and E.**

Question

In a virtual memory system the address space specified by the address lines of the CPU must be _____ than the physical memory size and _____ than the secondary storage size.

A. smaller, smaller
B. smaller, larger
C. larger, smaller
D. larger, larger

Answer

Option **C** is the correct answer

Question

If an instruction takes i microseconds and a page fault takes an additional j microseconds, the effective instruction time if on the average a page fault occurs every k instruction is:

A. $i + \frac{j}{k}$

B. $i + (j \times k)$

C. $\frac{i+j}{k}$

D. $(i+j) \times k$

Answer

Let's assume that k instructions have been executed. That means, we have had $k-1$ regular instructions and 1 instruction with page fault. Hence,

$$Time\ for\ k\ instruction = (k-1) * i + 1 * (i+j) = ki + j$$

Thus,

$$Avg\ time = \frac{ki+j}{k} = \boldsymbol{i + \frac{j}{k}}$$

Hence, **Option A is correct.**

Which of the following is/are advantage(s) of virtual memory?

A. Faster access to memory on an average.
B. Processes can be given protected address spaces.
C. Linker can assign addresses independent of where the program will be loaded in physical memory.
D. Program larger than the physical memory size can be run.

A multi-user, multi-processing operating system cannot be implemented on hardware that does not support

A. Address translation
B. DMA for disk transfer
C. At least two modes of CPU execution (privileged and non-privileged)
D. Demand paging

Which of the following statements is false?

A. Virtual memory implements the translation of a program's address space into physical memory address space
B. Virtual memory allows each program to exceed the size of the primary memory
C. Virtual memory increases the degree of multiprogramming
D. Virtual memory reduces the context switching overhead

Question

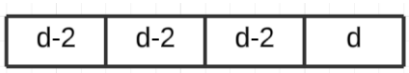
Assume the virtual memory address is 46 bits long and one page table entry is of size 4B. If the system is implementing a 3 – level mapping, then what is the page size?

Answer

Let the page size be d bits. Then,

$$Number\ of\ pages = \frac{2^d}{4B} = 2^{d-2}$$

Hence, the virtual address can be written as –



Hence,

$$d - 2 + d - 2 + d - 2 + d = 46$$

Therefore,

$$\boldsymbol{d = 10\ bits}$$

This makes the page size to be **1KB**

Question

In a particular Unix OS, each data block is of size 1024 bytes, each node has 10 direct data block addresses and three additional addresses: one for single indirect block, one for double indirect block and one for triple indirect block. Also, each block can contain addresses for 128 blocks. Which one of the following is approximately the maximum size of a file in the file system?

A. 512 MB
B. 2 GB
C. 8 GB
D. 16 GB

Answer

For Direct Data pointer,

$$Max\ file\ size = 10 * 1024 = 10\ KB$$

For Single Indirect pointer,

$$Max\ file\ size = 128 * 1024 = 128\ KB$$

For Double Indirect pointer,

$$Max\ file\ size = 128 * 128 * 1024 = 16\ MB$$

For Triple Indirect pointer,

$$Max\ file\ size = 128 * 128 * 128 * 1024 = 2\ GB$$

Hence,

$$Max\ file\ size\ (total) \approx \boldsymbol{2GB}$$

Therefore, **Option B is correct.**

Question

In a computer system, four files of size 11050 bytes, 4990 bytes, 5170 bytes and 12640 bytes need to be stored. For storing these files on disk, we can use either 100 byte disk blocks or 200 byte disk blocks (but can't mix block sizes). For each block used to store a file, 4 bytes of bookkeeping information also needs to be stored on the disk. Thus, the total space used to store a file is the sum of the space taken to store the file and the space taken to store the book keeping information for the blocks allocated for storing the file. A disk block can store either bookkeeping information for a file or data from a file, but not both.
What is the total space required for storing the files using 100 byte disk blocks and 200 byte disk blocks respectively?

(A) 35400 and 35800 bytes
(B) 35800 and 35400 bytes
(C) 35600 and 35400 bytes
(D) 35400 and 35600 bytes

Answer

For 100 byte blocks

$$Blocks\ for\ File\ 1 = \frac{11050}{100} = 110.5 \approx 111$$

$$Bookkeeping\ info\ for\ File\ 1 = \frac{4 * 111}{100} = 4.44 \approx 5$$

$$Blocks\ for\ File\ 2 = \frac{4990}{100} = 49.9 \approx 50$$

$$Bookkeeping\ info\ for\ File\ 2 = \frac{4 * 50}{100} = 2$$

$$Blocks\ for\ File\ 3 = \frac{5170}{100} = 51.7 \approx 52$$

$$Bookkeeping\ info\ for\ File\ 3 = \frac{4 * 52}{100} = 2.08 \approx 3$$

$$Blocks\ for\ File\ 4 = \frac{12640}{100} = 126.4 \approx 127$$

$$Bookkeeping\ info\ for\ File\ 4 = \frac{4 * 127}{100} = 5.08 \approx 6$$

For 200 byte blocks

$$Blocks\ for\ File\ 1 = \frac{11050}{200} = 55.25 \approx 56$$

$$Bookkeeping\ info\ for\ File\ 1 = \frac{4 * 56}{200} = 1.12 \approx 2$$

$$Blocks\ for\ File\ 2 = \frac{4990}{200} = 24.95 \approx 25$$

$$Bookkeeping\ info\ for\ File\ 2 = \frac{4 * 25}{200} = 0.5 \approx 1$$

$$Blocks\ for\ File\ 3 = \frac{5170}{200} = 25.85 \approx 26$$

$$Bookkeeping\ info\ for\ File\ 3 = \frac{4 * 26}{200} = 0.52 \approx 1$$

$$Blocks\ for\ File\ 4 = \frac{12640}{200} = 63.2 \approx 64$$

$$Bookkeeping\ info\ for\ File\ 2 = \frac{4 * 64}{200} = 1.28 \approx 2$$

Therefore,

$$Total\ blocks\ for\ 100B\ blocks = 356$$

$$Total\ blocks\ for\ 200B\ blocks = 177$$

$$Total\ memory\ for\ 100B\ blocks = 35600\ B$$

$$Total\ memory\ for\ 200B\ blocks = 35400\ B$$

Hence, **Option C is the correct answer**

Question

In a file allocation system, which of the following allocation scheme(s) can be used if no external fragmentation is allowed ?

1. Contiguous
2. Linked
3. Indexed

Answer

Both 2 and 3 can be used to prevent external fragmentation.

The data blocks of a very large file in the Unix file system are allocated using

A. continuous allocation
B. linked allocation
C. indexed allocation
D. an extension of indexed allocation

Question

A FAT (file allocation table) based file system is being used and the total overhead of each entry in the FAT is 4 bytes in size. Given a 100×10^6 bytes disk on which the file system is stored and data block size is 10^3 bytes, the maximum size of a file that can be stored on this disk in units of 10^6 bytes is _____.

Answer

$$1\ block\ size = 10^3\ B$$

$$Number\ of\ blocks = \frac{100 * 10^6}{10^3} = 10^5$$

Thus,

$$Size\ of\ FAT = 4 * 10^5 = 0.4 * 10^6 B$$

Hence,

$$Max\ file\ size = (100 * 10^6) - (0.4 * 10^6) = \mathbf{99.6 * 10^6\ B}$$

Question

A file system with a one-level directory structure is implemented on a disk with disk block size of $4K$ bytes. The disk is used as follows:

Disk-block 0	File Allocation Table, consisting of one 8-bit entry per data block, representing the data block address of the next data block in the file
Disk-block 1	Directory, with one 32 bit entry per file:
Disk-block 2	Data-block 1;
Disk-block 3	Data-block 2; etc.

a. What is the maximum possible number of files?

Answer

In Disk Block 0, the FAT consists of 8 – bit entry per block. Hence,

$$No\ of\ blocks = 2^8 = 256$$

In Disk Block 1, we have 4B entry for each file. Since the block size is 4KB,

$$Max\ number\ of\ files = \frac{4K}{4} = \mathbf{1K}$$

Now, if the question had asked for maximum file size, we would have –

$$Max\ file\ size = No.of\ data\ blocks * 1\ block\ size = (256 - 2) * 4K = \mathbf{1016KB} \approx \mathbf{0.99MB}$$

Question

Which of the following Operating System has least waiting time when executing instructions??

Select your answer

- ✗ Batch Operating System
- B Time Sharing Operating System
- C Multiprogramming Operating System
- D None of the above

Answer

Time Sharing Operating System or Multitasking Operating System is a logical extension to multiprogramming Operating System. Time Sharing Operating System allocates certain time to each process to get CPU and when time expires, process gets preempted and next process is allocated.

Question

Which takes more time, switching from Kernel to user mode or context switch?

Answer

Context switch requires more time as we need to store the current system state values in the stack before switching the context so that we may resume the execution.

Question

Consider the following statement:
Statement I : System which is using P:1 or P:Q mapping , user threads are scheduled by the kernel and kernel threads are scheduled by the user library.
Statement II : Kernel threads always associated with a process whereas user thread need not be associated with the process.
Which of the above statements are true ?

Answer

None of the statements are correct.

Question

System booting process.

Answer

1. POST
2. BIOS initiation
3. Check settings
4. Load OS in RAM

NOTE

In a fork() command –

- Parent and child processes share the same virtual address space

- Parent and child processes share the parent address space until one of them does a write operation
- Parent and child processes have the same program counter

Question

Which of the following is true regarding kernel level threads?

Select your answer

A It is easier to implement the kernel level thread as compare to user level thread.	✕ Context for kernel level thread is smaller as compared to user level thread
C Blocking of one kernel level thread block all the related thread of that program	D Scheduling of the kernel level thread requires hardware support.

Answer

a- it is false, since implementation of kernel level thread require hardware support and it is difficult as compared to user level thread.
b- It is false since kernel level threads are independent from each other. Hence, context will be larger.
c- Since each and every kernel level thread is independent of each other so there is no need of blocking of all the other threads.
d- It is true regarding kernel level threads.

Question

Suppose there are two browsers, Browser A and Browser B. Browser A uses a thread per tab, but Browser B uses a process per tab. Which of the following option/s is/are correct? _____? [Give Integer Type answer]

(I) Context switch is much faster in Browser A over Browser B.
(II) Browser A will have shared address space for its tabs. For Browser B, the address space is different.
(III) Browser A will require a specific interprocess communication interface where Browser B does not require it.
(IV) If 1 tab in Browser A does some malicious activity, all tabs (threads) are affected.

Answer

Statements I, II and IV are correct.

QUESTION BANK

Question 1

Q In the following process state transition diagram for a uniprocessor system, assume that there are always some processes in the ready state:

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 P2 in blocked state can make transition E while another process P1 is in running state.

III) The OS uses preemptive scheduling.

IV) The OS uses non-preemptive scheduling.

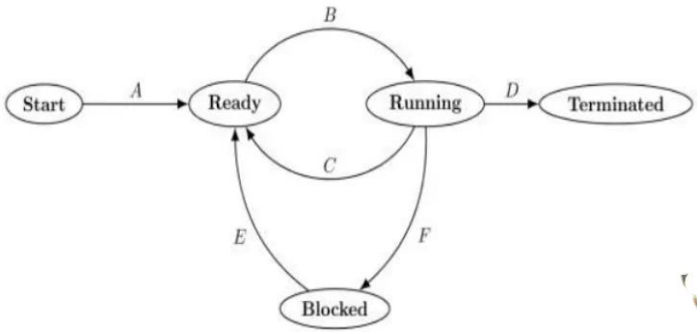
Which of the above statements are TRUE? **(GATE-2009) (2 Marks)**

a) I and II

b) I and III

c) II and III

d) II and IV



Question 2

Q The process state transition diagram of an operating system is as given below.

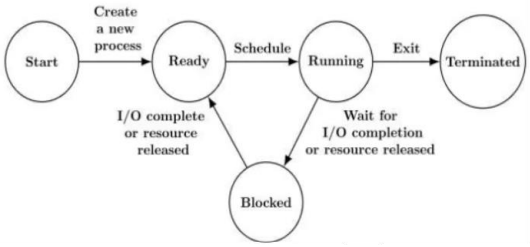
Which of the following must be FALSE about the above operating system? **(GATE-2006) (1 Marks)**

a) It is a multiprogram operating system

b) It uses preemptive scheduling

c) It uses non-preemptive scheduling

d) It is a multi-user operating system



Question 3

Q A computer handles several interrupt sources of which the following are relevant for this question.

- Interrupt from CPU temperature sensor (raises interrupt if CPU temperature is too high)
- Interrupt from Mouse (raises interrupt if the mouse is moved or a button is pressed)
- Interrupt from Keyboard (raises interrupt when a key is pressed or released)
- Interrupt from Hard Disk (raises interrupt when a disk read is completed)

Which one of these will be handled at the HIGHEST priority? **(GATE - 2011) (1 Marks)**

(A) Interrupt from Hard Disk

(B) Interrupt from Mouse

(C) Interrupt from Keyboard

(D) Interrupt from CPU temperature sensor

Question 4

Q Three processes arrive at time zero with CPU bursts of 16, 20 and 10 milliseconds. If the scheduler has prior knowledge about the length of the CPU bursts, the minimum achievable average waiting time for these three processes in a non-preemptive scheduler (rounded to nearest integer) is _____ milliseconds. **(GATE 2021) (1 MARKS)**

Question 5

Q For the processes listed in the following table, which of the following scheduling schemes will give the lowest average turnaround time? **(GATE-2015) (2 Marks)**

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

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

Question 6

Q 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? **(GATE - 2016) (1 Marks)**

(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

Question 7

Q consider the following set of processes and the length of CPU burst time given in milliseconds:

Process	CPU Burst Time	CT	TAT	WT
P_1	5			
P_2	7			
P_3	6			
P_4	4			

Assume that processes being scheduled with round robin scheduling algorithm with time quantum 4ms. Then the waiting for p_4 is _____. **(NET-DEC-2018)**

a) 0 **b) 4** **c) 6** **d) 12**

Question 8

Q 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? **(GATE-2013) (1 Marks)**

(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

Question 9

Q Consider the following four processes with arrival times (in milliseconds) and their length of CPU bursts (in milliseconds) as shown below: **(GATE - 2019) (2 Marks)**

Process	Arrival Time	CPU Time
P_1	0	3
P_2	1	1
P_3	3	3
P_4	4	Z

These processes are run on a single processor using pre-emptive Shortest Remaining Time First scheduling algorithm. If the average waiting time of the processes is 1 millisecond, then the value of Z is _____.

Question 10

Q Consider the following set of processes, with the arrival times and the CPU-burst times given in milliseconds **(GATE - 2017) (2 Marks)**

Process	Arrival Time	CPU Time
P_1	0	5
P_2	1	3
P_3	2	3
P_4	4	1

What is the average turnaround time for these processes with the pre-emptive shortest remaining processing time first (SRPT) algorithm?

(A) 5.50 **(B)** 5.75 **(C)** 6.00 **(D)** 6.25

Question 11

Q Consider the following CPU processes with arrival times (in milliseconds) and length of CPU bursts (in milliseconds) as given below: **(GATE - 2017) (1 Marks)**

Process	Arrival Time	CPU Time	CT	TAT	WT
P_1	0	7			
P_2	3	3			
P_3	5	5			
P_4	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.

Question 12

Q Consider the following table of arrival time and burst time for three processes P_0, P_1 and P_2.

Process	Arrival Time	Burst Time	CT	TAT	WT
P_0	0	9			
P_1	1	4			
P_2	2	9			

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?
(GATE-2011) (2 Marks)

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

Question 13

Q An operating system uses Shortest Remaining Time first (SRT) process scheduling algorithm. Consider the arrival times and execution times for the following processes:

Process	Arrival Time	CPU Time	CT	TAT	WT
P_1	0	20			
P_2	15	25			
P_3	30	10			
P_4	45	15			

What is the total waiting time for process P_2? **(GATE - 2007) (2 Marks)**

(A) 5 **(B)** 15 **(C)** 40 **(D)** 55

Question 14

Q 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	CT	TAT	WT
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 ____. **(GATE-2017) (2 Marks)**

Question 15

Q 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 scheduled 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 milliseconds and task preemptions are allowed, the first instance of T_3 completes its execution at the end of _____ milliseconds. **(GATE-2015) (2 Marks)**

Question 16

Q We wish to schedule three processes P_1, P_2 and P_3 on a uniprocessor system. The priorities, CPU time requirements and arrival times of the processes are as shown below.

Process	Priority	CPU time	Arrival time
P_1	10(highest)	20 sec	00:00:05
P_2	9	10 sec	00:00:03
P_3	8 (lowest)	15 sec	00:00:00

We have a choice of pre-emptive or non-pre-emptive scheduling. In pre-emptive scheduling, a late-arriving higher priority process can pre-empt a currently running process with lower priority. In non-pre-emptive scheduling, a late-arriving higher priority process must wait for the currently executing process to complete before it can be scheduled on the processor.

What are the turnaround times (time from arrival till completion) of P_2 using pre-emptive and non-pre-emptive scheduling respectively. **(GATE-2005) (2 Marks)**

(A) 30 sec, 30 sec **(B)** 30 sec, 10 sec

(C) 42 sec, 42 sec **(D)** 30 sec, 42 sec

Question 17

Q Consider the 3 processes, P_1, P_2 and P_3 shown in the table. **(GATE-2012) (2 Marks)**

The completion order of the 3 processes under the policies FCFS and RR2 (round robin scheduling with CPU quantum of 2-time units) are

Process	Arrival Time	Time Unit Required
P_1	0	5
P_2	1	7
P_3	3	4

(A) FCFS: P_1, P_2, P_3 RR: P_1, P_2, P_3

(B) FCFS: P_1, P_3, P_2 RR: P_1, P_3, P_2

(C) FCFS: P_1, P_2, P_3 RR: P_1, P_3, P_2

(D) FCFS: P_1, P_3, P_2 RR: P_1, P_2, P_3

Question 18

Q Consider four processes P, Q, R, and S scheduled on a CPU as per round robin algorithm with a time quantum of 4 units. The processes arrive in the order P, Q, R, S, all at time t = 0. There is exactly one context switch from S to Q, exactly one context switch from R to Q, and exactly two context switches from Q to R. There is no context switch from S to P. Switching to a ready process after the termination of another process is also considered a context switch. Which one of the following is NOT possible as CPU burst time (in time units) of these processes? **(GATE 2022) (2 MARKS)**

(A) P = 4, Q = 10, R = 6, S = 2

(B) P = 2, Q = 9, R = 5, S = 1

(C) P = 4, Q = 12, R = 5, S = 4

(D) P = 3, Q = 7, R = 7, S = 3

Question 19

Q Which of the following statement(s) is/are correct in the context of CPU scheduling?**(GATE 2021) (1 MARKS)**

(A) Turnaround time includes waiting time

(B) The goal is to only maximize CPU utilization and minimize throughput

(C) Round-robin policy can be used even when the CPU time required by each of the processes is not known apriori

(D) Implementing preemptive scheduling needs hardware support

---

Question 20

Q Consider n processes sharing the CPU in a round-robin fashion. Assuming that each process switch takes s seconds, what must be the quantum size q such that the overhead resulting from process switching is minimized but at the same time each process is guaranteed to get its turn at the CPU at least every t seconds? **(GATE-1998) (1 Marks) (NET-Dec-2012)**

a) $q \leq (t-ns)/(n-1)$ **c)** $q \leq (t-ns)/(n+1)$

b) $q \geq (t-ns)/(n-1)$ **d)** $q \geq (t-ns)/(n+1)$

Question 21

Q If the time-slice used in the round-robin scheduling policy is more than the maximum time required to execute any process, then the policy will **(GATE-2008) (1 Marks)**

(A) degenerate to shortest job first **(C)** degenerate to first come first serve

(B) degenerate to priority scheduling **(D)** none of the above

Question 22

Q Four jobs to be executed on a single processor system arrive at time 0^+ in the order A, B, C, D. their burst CPU time requirements are 4, 1, 8, 1 time units respectively. The completion time of A under round robin scheduling with time slice of one-time unit is. **(GATE-1996) (2 Marks) (ISRO-2008)**

(a) 10 **(c) 8**

(b) 4 **(d) 9**

Question 23

Q Which scheduling policy is most suitable for a time-shared operating system? **(GATE-1995) (1 Marks)**

(a) Shortest Job First **(c)** First Come First Serve

(b) Round Robin **(d)** Elevator

Question 24

Q 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 $t_{i/o}$ 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	$T_{i/o}$
A	100	500
B	350	500
C	200	500

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 _____. **(GATE-2014) (2 Mark)**

Question 25

Q Which of the following statements are true? **(GATE-2010) (1 Marks)**

1) Shortest remaining time first scheduling may cause starvation

2) Pre-emptive scheduling may cause starvation

3) Round robin is better than FCFS in terms of response time

(A) 1 only **(C)** 2 and 3 only

(B) 1 and 3 only **(D)** 1, 2 and 3

Question 26

Q Consider three processes, all arriving at time 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? **(GATE-2006) (2 Marks)**

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

Question 27

Q The arrival time, priority, and duration of the CPU and I/O bursts for each of three processes P_1, P_2 and P_3 are given in the table below. Each process has a CPU burst followed by an I/O burst followed by another CPU burst. Assume that each process has its own I/O resource. **(GATE-2006) (2 Marks)**

Process	Arrival Time	Priority	Burst duration, CPU, I/O CPU
P_1	0	2	1,5,3
P_2	2	3(L)	3,3,1
P_3	3	1(H)	2,3,1

The programmed operating system uses preemptive priority scheduling. What are the finish times of the processes P_1, P_2 and P_3 ?

(A) 11, 15, 9 **(B)** 10, 15, 9 **(C)** 11, 16, 10 **(D)** 12, 17, 11

Question 28

Q Consider three processes (process id 0, 1, 2 respectively) with compute time 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? **(GATE-2006) (2 Marks)**

(A) 13 units **(C)** 15 units

(B) 14 units **(D)** 16 units

Question 29

Q Consider a set of n tasks with known runtimes r_1, r_2, ... r_n to be run on a uniprocessor machine. Which of the following processor scheduling algorithms will result in the maximum throughput? **(GATE-2001) (1 Marks)**

(A) Round-Robin **(C)** Highest-Response-Ratio-Next

(B) Shortest-Job-First **(D)** First-Come-First-Served

Question 30

Q The following two functions P_1 and P_2 that share a variable B with an initial value of 2 execute concurrently.

$P_1()$	$P_2()$
{	{
C = B – 1;	D = 2 * B;
B = 2 * C;	B = D - 1;
}	}

The number of distinct values that B can possibly take after the execution is **(GATE-2015) (1 Mark)**

Question 31

Q When the result of a computation depends on the order of the processes execution, there is said to be **(GATE-1998) (1 Marks)**

a) cycle stealing

c) a time lock

b) race condition

d) a deadlock

Question 32

Q Consider three concurrent processes P_1 and P_2 and P_3 as shown below, which access a shared variable D that has been initialized to 100. **(GATE-2019) (2 Marks)**

P_1	P_2	P_3
.	.	.
.	.	.
D = D + 20	D = D - 50	D = D + 10
.	.	.
.	.	.

The process are executed on a uniprocessor system running a time-shared operating system. If the minimum and maximum possible values of D after the three processes have completed execution are X and Y respectively, then the value of Y–X is _____.

Question 33

Q Consider the methods used by processes P_1 and P_2 for accessing their critical sections whenever needed, as given below. The initial values of shared Boolean variables S_1 and S_2 are randomly assigned. **(GATE-2010) (1 Marks)** (NET-JUNE-2012)

$P_1()$	$P_2()$
While(S_1==S_2);	While(S_1!=S_2);
Critical section	Critical section
S_1=S_2;	S_1=not(S_2);

Which one of the following statements describes the properties achieved?

(A) Mutual exclusion but not progress

(B) Progress but not mutual exclusion

(C) Neither mutual exclusion nor progress

(D) Both mutual exclusion and progress

Question 34

Q Consider the following two-process synchronization solution. **(GATE-2016) (2 Marks)**

The shared variable turn is initialized to zero. Which one of the following is TRUE?

Process 0	Process 1
Entry: loop while (turn == 1);	Entry: loop while (turn == 0);
(critical section)	(critical section)
Exit: turn = 1;	Exit: turn = 0;

(a) This is a correct two-process synchronization solution.

(b) This solution violates mutual exclusion requirement

(c) This solution violates progress requirement.

(d) This solution violates bounded wait requirement

Question 35

Q Consider Peterson's algorithm for mutual exclusion between two concurrent processes i and j. The program executed by process is shown below.**(GATE-2001) (2 Mark)**

Repeat
flag[i] = T;
turn = j;
while(P) do no-op;
Enter critical section, perform actions, then critical section
flag[i] = f;
Perform other non-critical section actions
Until false;

For the program to guarantee mutual exclusion, the predicate P in the while loop should be.

(A) flag[j] = true and turn = i

(B) flag[j] = true and turn = j

(C) flag[i] = true and turn = j

(D) flag[i] = true and turn = i

Question 36

Q Two processes, P_1 and P_2, need to access a critical section of code. Consider the following synchronization construct used by the processes: Here, $wants_1$ and $wants_2$ are shared variables, which are initialized to false. Which one of the following statements is TRUE about the above construct? **(GATE-2007) (2 Mark)**

$P_1()$	$P_2()$
While(t)	While(t)
{	{
$wants_1$ = T	$wants_2$ = T
While ($wants_2$== T);	While ($wants_1$== T);
Critical section	Critical section
$wants_1$ = F	$wants_2$ = F
Remainder section	Remainder section
}	}

(A) It does not ensure mutual exclusion.

(B) It does not ensure bounded waiting.

(C) It requires that processes enter the critical section in strict alternation.

(D) It does not prevent deadlocks, but ensures mutual exclusion.

Question 37

Q Two processes X and Y need to access a critical section. Consider the following synchronization construct used by both the processes.
Here, varP and varQ are shared variables and both are initialized to false. Which one of the following statements is true? **(GATE-2015) (1 Mark)**

(A) The proposed solution prevents deadlock but fails to guarantee mutual exclusion

(B) The proposed solution guarantees mutual exclusion but fails to prevent deadlock

(C) The proposed solution guarantees mutual exclusion and prevents deadlock

(D) The proposed solution fails to prevent deadlock and fails to guarantee mutual exclusion

Process X	Process Y
While(t)	While(t)
{	{
varP = T;	varQ = T;
While (varQ == T)	While (varP == T)
{	{
Critical section	Critical section
varP = F;	varQ = F;
}	}
}	}

Question 38

Q Semaphores S and T. The code for the processes P and Q is shown below. Synchronization statements can be inserted only at points W, X, Y and Z.

Which of the following will always lead to an output starting with '001100110011' ? **(GATE-2004) (2 Marks)**

(A) P(S) at W, V(S) at X, P(T) at Y, V(T) at Z, S and T initially 1

(B) P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S initially 1, and T initially 0

(C) P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S and T initially 1

(D) P(S) at W, V(S) at X, P(T) at Y, V(T) at Z, S initially 1, and T initially 0

Process P	Process Q
While(t)	While(t)
{	{
W:	Y:
print '0';	print '1';
print '0';	print '1';
X:	Z:
}	}

Question 39

Q Semaphores S and T. The code for the processes P and Q is shown below. Synchronization statements can be inserted only at points W, X, Y and Z.

Which of the following will ensure that the output string never contains a substring of the form 01^n0 or 10^n1 where n is odd? **(GATE-2004) (2 Marks)**

(A) P(S) at W, V(S) at X, P(T) at Y, V(T) at Z, S and T initially 1

(B) P(S) at W, V(T) at X, P(T) at Y, V(S) at Z, S and T initially 1

(C) P(S) at W, V(S) at X, P(S) at Y, V(S) at Z, S initially 1

(D) V(S) at W, V(T) at X, P(S) at Y, P(T) at Z, S and T initially 1

Process P	Process Q
While(t)	While(t)
{	{
W:	Y:
print '0';	print '1';
print '0';	print '1';
X:	Z:
}	}

Question 40

Q Two concurrent processes P_1 and P_2 use four shared resources R_1, R_2, R_3 and R_4, as shown below.

Both processes are started at the same time, and each resource can be accessed by only one process at a time The following scheduling constraints exist between the access of resources by the processes:

i) P_2 must complete use of R_1 before P_1 gets access to R_1

ii) P_1 must complete use of R_2 before P_2 gets access to R_2.

iii) P_2 must complete use of R_3 before P_1 gets access to R_3.

iv) P_1 must complete use of R_4 before P_2 gets access to R_4.

There are no other scheduling constraints between the processes. If only binary semaphores are used to enforce the above scheduling constraints, what is the minimum number of binary semaphores needed? **(GATE-2005) (2 Marks)**

P_1	P_2
Compute:	Compute;
Use R_1;	Use R_1;
Use R_2;	Use R_2;
Use R_3;	Use R_3;.
Use R_4;	Use R_4;

Question 41

Q Three concurrent processes X, Y, and Z execute three different code segments that access and update certain shared variables. Process X executes the P operation (i.e., wait) on semaphores a, b and c; process Y executes the P operation on semaphores b, c and d; process Z executes the P operation on semaphores c, d, and a before entering the respective code segments. After completing the execution of its code segment, each process invokes the V operation (i.e., signal) on its three semaphores. All semaphores are binary semaphores initialized to one. Which one of the following represents a deadlock-free order of invoking the P operations by the processes? **(GATE-2013) (1 Marks)**

(A) X: P(a)P(b)P(c) Y: P(b)P(c)P(d) Z: P(c)P(d)P(a)
(B) X: P(b)P(a)P(c) Y: P(b)P(c)P(d) Z: P(a)P(c)P(d)
(C) X: P(b)P(a)P(c) Y: P(c)P(b)P(d) Z: P(a)P(c)P(d)
(D) X: P(a)P(b)P(c) Y: P(c)P(b)P(d) Z: P(c)P(d)P(a)

X	Y	Z
CS	CS	CS

Question 42

Q Consider two processes P_1 and P_2 accessing the shared variables X and Y protected by two binary semaphores S_X and S_Y respectively, both initialized to 1. P and V denote the usual semaphone operators, where P decrements the semaphore value, and V increments the semaphore value. The pseudo-code of P_1 and P_2 is as follows: **(GATE-2004) (2 Marks)**
In order to avoid deadlock, the correct operators at L_1, L_2, L_3 and L_4 are respectively

(A) $P(S_Y)$, $P(S_X)$; $P(S_X)$, $P(S_Y)$

(B) $P(S_X)$, $P(S_Y)$; $P(S_Y)$, $P(S_X)$

(C) $P(S_X)$, $P(S_X)$; $P(S_Y)$, $P(S_Y)$

(D) $P(S_X)$, $P(S_Y)$; $P(S_X)$, $P(S_Y)$

P_1	P_2
While true do {	While true do {
L_1 :	L_3 : .....
L_2 :	L_4 : .....
X = X + 1;	Y = Y + 1;
Y = Y – 1;	X = Y – 1;
$V(S_X)$;	$V(S_X)$;
$V(S_Y)$;	$V(S_Y)$;
}	}

Question 43

Q A shared variable x, initialized to zero, is operated on by four concurrent processes W, X, Y, Z as follows. Each of the processes W and X reads x from memory, increments by one, stores it to memory, and then terminates. Each of the processes Y and Z reads x from memory, decrements by two, stores it to memory, and then terminates. Each process before reading x invokes the P operation (i.e., wait) on a counting semaphore S and invokes the V operation (i.e., signal) on the semaphore S after storing x to memory. Semaphore S is initialized to two. What is the maximum possible value of x after all processes complete execution? **(GATE-2013) (2 Marks)**

(A) -2 **(B)** -1 **(C)** 1 **(D)** 2

W	X	Y	Z
Wait(S)	Wait(S)	Wait(S)	Wait(S)
R(x)	R(x)	R(x)	R(x)
X = x + 1	X = x + 1	X = x - 2	X = x - 2
W(x)	W(x)	W(x)	W(x)
Signal(S)	Signal(S)	Signal(S)	Signal(S)

Question 44

Q Process P_1 repeatedly adds one item at a time to a buffer of size n, and process P_2 repeatedly removes one item at a time from the same buffer using the programs given below. In the programs, K, L, M and N are unspecified statements. **(GATE-2004) (2 Marks)**

$P_1()$	$P_2()$
while(T) {	while(T) {
K;	M;
P(mutex);	P(mutex);
Add an item to the buffer;	Remove an item to the buffer;
V(mutex);	V(mutex);
L;	N;
}	}

The statements K, L, M and N are respectively

(A) P(full), V(empty), P(full), V(empty)

(B) P(full), V(empty), P(empty), V(full)

(C) P(empty), V(full), P(empty), V(full)

(D) P(empty), V(full), P(full), V(empty)

Question 45

Q Consider the procedure below for the Producer-Consumer problem which uses semaphores:
Semaphore n = 0;
Semaphore s = 1;

Void Producer ()	Void Consumer ()
{	{
While(true)	While(true)
{	{
Produce ();	semWait(s);
SemWait(s);	semWait(n);
addToBuffer();	RemovefromBuffer();
semSignal(s);	semSignal(s);
SemSignal(n);	consume();
}	}
}	}

Which one of the following is TRUE? **(GATE-2014) (2 Marks)**

(A) The producer will be able to add an item to the buffer, but the consumer can never consume it.

(B) The consumer will remove no more than one item from the buffer.

(C) Deadlock occurs if the consumer succeeds in acquiring semaphore s when the buffer is empty.

(D) The starting value for the semaphore n must be 1 and not 0 for deadlock-free operation.

Question 46

Q Consider the following solution to the producer-consumer synchronization problem. The shared buffer size is N. Three semaphores *empty*, *full* and *mutex* are defined with respective initial values of 0, N and 1. Semaphore *empty* denotes the number of available slots in the buffer, for the consumer to read from. Semaphore *full* denotes the number of available slots in the buffer, for the producer to write to. The placeholder variables, denoted by P, Q, R and S, in the code below can be assigned either *empty* or *full*. The valid semaphore operations are: *wait()* and *signal()*. **(GATE-2018) (2 Marks)**

Producer:	Consumer:
Do{	Do{
Wait(P);	Wait(R);
Wait(mutex);	Wait(mutex);
//Add item to buffer	//Consume item from buffer
Signal(mutex);	Signal(mutex);
Signal(Q);	Signal(S);
} while(1);	} while(1);

Which one of the following assignments to P, Q, R and S will yield the correct solution?

(A) P: *full*, Q: *full*, R: *empty*, S: *empty*
(B) P: *empty*, Q: *empty*, R: *full*, S: *full*
(C) P: *full*, Q: *empty*, R: *empty*, S: *full*
(D) P: *empty*, Q: *full*, R: *full*, S: *empty*

Question 47

Barrier is a synchronization construct where a set of processes synchronizes globally i.e. each process in the set arrives at the barrier and waits for all others to arrive and then all processes leave the barrier. Let the number of processes in the set be three and S be a binary semaphore with the usual P and V functions. Consider the following C implementation of a barrier with line numbers shown on left.

The variables process_arrived and process_left are shared among all processes and are initialized to zero. In a concurrent program all the three processes call the barrier function when they need to synchronize globally.

```
void barrier (void) {
1:  P(S);
2:  process_arrived++;
3.  V(S);
4:  while (process_arrived !=3);
5:  P(S);
6:  process_left++;
7:  if (process_left==3) {
8:    process_arrived = 0;
9:    process_left = 0;
10: }
11: V(S);
}
```

Q The above implementation of barrier is incorrect. Which one of the following is true? **(GATE-2006) (2 Marks)**

a) The barrier implementation is wrong due to the use of binary semaphore S.

b) The barrier implementation may lead to a deadlock if two barrier in invocations are used in immediate succession.

c) Lines 6 to 10 need not be inside a critical section.

d) The barrier implementation is correct if there are only two processes instead of three.

Question 48

Q Synchronization in the classical readers and writers problem can be achieved through use of semaphores. In the following incomplete code for readers-writers problem, two binary semaphores mutex and wrt are used to obtain synchronization **(GATE-2007) (2 Marks)**

The values of S_1, S_2, S_3, S_4, (in that order) are

(A) signal (mutex), wait (wrt), signal (wrt), wait (mutex)

(B) signal (wrt), signal (mutex), wait (mutex), wait (wrt)

(C) wait (wrt), signal (mutex), wait (mutex), signal (wrt)

(D) signal (mutex), wait (mutex), signal (mutex), wait (mutex)

Writer()	Reader()
	Wait(mutex)
	Readcount++
	If(readcount ==1)
	S_1
Wait(wrt)	S_2
Writing is performed	CS //Read
Signal(wrt)	S_3
	Readcount--
	If(readcount ==0)
	S_4
	signal(mutex)

Question 49

Q Let m[0]...m[4] be mutexes (binary semaphores) and P[0] .... P[4] be processes. Suppose each process P[i] executes the following:

wait (m[i]); wait(m[(i+1) mode 4]);

---

release (m[i]); release (m[(i+1)mod 4]);

This could cause: **(GATE-2000) (1 Marks)**

(A) Thrashing

(B) Deadlock

(C) Starvation, but not deadlock

(D) None of the above

Question 50

Q A solution to the Dining Philosophers Problem which avoids deadlock is: **(GATE-1996) (2 Marks)**

(A) ensure that all philosophers pick up the left fork before the right fork

(B) ensure that all philosophers pick up the right fork before the left fork

(C) ensure that one particular philosopher picks up the left fork before the right fork, and that all other philosophers pick up the right fork before the left fork

(D) None of the above

Question 51

Q Consider a non-negative counting semaphore S. The operation P(S) decrements S, and V(S) increments S. During an execution, 20 P(S) operations and 12 V(S) operations are issued in some order. The largest initial value of S for which at least one P(S) operation will remain blocked is _____. **(GATE-2016) (1 Marks)**

Question 52

Q A counting semaphore was initialized to 10. Then 6P (wait) operations and 4V (signal) operations were completed on this semaphore. The resulting value of the semaphore is **(GATE-1998) (1 Marks)**

Question 53

Q At a particular time of computation the value of a counting semaphore is 7. Then 20 P operations and 15V operations were completed on this semaphore. If the new value of semaphore is will be **(GATE-1992) (1 Marks)**

Question 54

Q Fetch_And_Add(X,i) is an atomic Read-Modify-Write instruction that reads the value of memory location X, increments it by the value i, and returns the old value of X. It is used in the pseudocode shown below to implement a busy-wait lock. L is an unsigned integer shared variable initialized to 0. The value of 0 corresponds to lock being available, while any non-zero value corresponds to the lock being not available.

AcquireLock(L)	ReleaseLock (L)

{
while (Fetch_And_Add(L,1))
L = 1;
}
	{
	L = 0;
	}

This implementation **(GATE-2012) (2 Marks)**

(A) fails as L can overflow

(B) fails as L can take on a non-zero value when the lock is actually available

(C) works correctly but may starve some processes

(D) works correctly without starvation

Question 55

Q The enter_CS() and leave_CS() functions to implement critical section of a process are realized using test-and-set instruction as follows:

```
void enter_CS(X)
{
   while test-and-set(X) ;
}
```

```
void leave_CS(X)
{
   X = 0;
}
```

In the above solution, X is a memory location associated with the CS and is initialized to 0. Now consider the following statements: **(GATE-2009) (2 Marks)**

I. The above solution to CS problem is deadlock-free

II. The solution is starvation free.

III. The processes enter CS in FIFO order.

IV. More than one process can enter CS at the same time.

Which of the above statements is TRUE?

(A) I only **(B)** I and II

(C) II and III **(D)** IV only

Question 56

Q Consider the following policies for preventing deadlock in a system with mutually exclusive resources.

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

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

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

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

Which of the above policies can be used for preventing deadlock? **(GATE-2015) (2 Marks)**

(A) Any one of I and III but not II or IV

(B) Any one of I, III and IV but not II

(C) Any one of II and III but not I or IV

(D) Any one of I, II, III and IV

Question 57

Q An operating system implements a policy that requires a process to release all resources before making a request for another resource. Select the TRUE statement from the following: **(GATE-2008) (2 Marks)**

(A) Both starvation and deadlock can occur

(B) Starvation can occur but deadlock cannot occur

(C) Starvation cannot occur but deadlock can occur

(D) Neither starvation nor deadlock can occur

Question 58

Q Which of the following is NOT a valid deadlock prevention scheme? **(GATE-2000) (2 Marks)**

(A) Release all resources before requesting a new resource

(B) Number the resources uniquely and never request a lower numbered resource than the last one requested.

(C) Never request a resource after releasing any resource

(D) Request all required resources be allocated before execution.

Question 59

Q 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 larges value of K that will always avoid deadlock is _____. **(GATE-2018) (1 Marks)**

Question 60

Q A system has 6 identical resources and N processes competing for them. Each process can request at most 2 resources. Which one of the following values of N could lead to a deadlock? **(GATE-2015) (1 Marks)**

a) 1

b) 2

c) 3

d) 6

Question 61

Q 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 **(GATE-2014) (2 Marks)**

(A) 6

(B) 7

(C) 8

(D) 9

Question 62

Q Suppose n processes, P_1, …. P_n share m 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? **(GATE-2005) (2 Marks)**

a) $\forall_i$, $S_i < m$

b) $\forall_i$, $S_i < n$

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

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

Question 63

Q A single processor system has three resource types X, Y and Z, which are shared by three processes. There are 5 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? **(GATE-2007) (2 Marks)**

(A) P_0

(B) P_1

(C) P_2

	Allocation			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

(D) None of the above, since the system is in a deadlock

Question 64

Q Consider the following snapshot of a system running n concurrent processes. Process i is holding X_i instances of a resource R, $1 \le i \le n$. Assume that all instances of R are currently in use. Further, for all i, process i can place a request for at most Y_i additional instances of R while holding the X_i instances it already has. Of the n processes, there are exactly two processes p and q such that $Y_p = Y_q = 0$. Which one of the following conditions guarantees that no other process apart from p and q can complete execution? **(GATE-2019) (2 Marks)**

a) $X_p + X_q < \text{Min}\{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$

b) $X_p + X_q < \text{Max}\{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$

c) $\text{Min}(X_p, X_q) \ge \text{Min}\{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$

d) $\text{Min}(X_p, X_q) \le \text{Max}\{Y_k \mid 1 \le k \le n, k \ne p, k \ne q\}$

Question 65

Q 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.

	Allocation			Max		
	E	F	G	E	F	G
P_0	1	0	1	4	3	1
P_1	1	1	2	2	1	4
P_2	1	0	3	1	3	3
P_3	2	0	0	5	4	1

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.
From the perspective of deadlock avoidance, which one of the following is true? **(GATE-2018) (2 Marks)**

(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

---

Question 66

Q A system shares 9 tape drives. The current allocation and maximum requirement of tape drives for 3 processes are shown below: Which of the following best describes the current state of the system? **(GATE-2014) (2 Marks)**

Process	Current Allocation	Maximum Requirement	Current Requirement	Current Available
P_1	3	7		
P_2	1	6		
P_3	3	5		

(A) Safe, Deadlocked

(C) Not Safe, Deadlocked

(B) Safe, Not Deadlocked

(D) Not Safe, Not Deadlocked

Question 67

Q 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. There are 3 units of type X, 2 units of type Y and 2 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:

	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			

REQ_1: P_0 requests 0 units of X, 0 units of Y and 2 units of Z
REQ_2: P_1 requests 2 units of X, 0 units of Y and 0 units of Z
Which one of the following is TRUE? **(GATE-2014) (2 Marks)**

(A) Only REQ_1 can be permitted.

(B) Only REQ_2 can be permitted.

(C) Both REQ_1 and REQ_2 can be permitted.

(D) Neither REQ_1 nor REQ_2 can be permitted

Question 68

Q Which of the following is NOT true of deadlock prevention and deadlock avoidance schemes? **(GATE-2008) (2 Marks)**

(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

---

Question 69

Q Consider the following snapshot of a system running n processes. Process P_i is holding X_i instances of a resource R, $1 <= i <= n$. currently, all instances of R are occupied. Further, for all i, process i has placed a request for an additional Y_i instances while holding the X_i instances it already has. There are exactly two processes p and q such that $Y_p = Y_q = 0$. Which one of the following can serve as a necessary condition to guarantee that the system is not approaching a deadlock? **(GATE-2006) (2 Mark)**

(A) min (X_p, X_q) < max (Y_k) where k != p and k != q

(B) $X_p + X_q$ >= min (Y_k) where k != p and k != q

(C) max (X_p, X_q) > 1

(D) min (X_p, X_q) > 1

Question 70

Q Which of the following statements is/are TRUE with respect to deadlocks?**(GATE 2022) (1 MARKS)**

(A) Circular wait is a necessary condition for the formation of deadlock.

(B) In a system where each resource has more than one instance, a cycle in its wait-for graph indicates the presence of a deadlock.

(C) If the current allocation of resources to processes leads the system to unsafe state, then deadlock will necessarily occur.

(D) In the resource-allocation graph of a system, if every edge is an assignment edge, then the system is not in deadlock state.

Question 71

Q Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB, and 250 KB, where KB refers to kilobyte. These partitions need to be allotted to four processes of sizes 357 KB, 210 KB, 468 KB and 491 KB in that order. If the best fit algorithm is used, which partitions are NOT allotted to any process? **(GATE-2015) (2 Marks)**

(A) 200 KB and 300 KB **(B)** 200 KB and 250 KB **(C)** 250 KB and 300 KB **(D)** 300 KB and 400 KB

Question 72

Q Consider the following heap (figure) in which blank regions are not in use and hatched region are in use. **(GATE-1994) (2 Marks)**

The sequence of requests for blocks of size 300, 25, 125, 50 can be satisfied if we use

(a) either first fit or best fit policy (any one)

(b) first fit but not best fit policy

(c) best fit but first fit policy

(d) None of the above

Question 73

Q A 1000 Kbyte memory is managed using variable partitions but No compaction. It currently has two partitions of sizes 200 Kbytes and 260 Kbytes respectively. The smallest allocation request in Kbytes that could be denied is for? **(GATE-1996) (1 Marks)**

(a) 151

(c) 231

(b) 181

(d) 541

Question 74

Q Consider a system with byte-addressable memory, 32-bit logical addresses, 4 kilobyte page size and page table entries of 4 bytes each. The size of the page table in the system in megabytes is _____ **(GATE-2015) (2 Marks)**

Question 75

Q A Computer system implements 8 kilobyte pages and a 32-bit physical address space. Each page table entry contains a valid bit, a dirty bit three permission bits, and the translation. If the maximum size of the page table of a process is 24 megabytes, the length of the virtual address supported by the system is _____ bits? **(GATE-2015) (2 Marks)**

Question 76

Q Consider a machine with 64 MB physical memory and a 32-bit virtual address space. If the page size is 4KB, what is the approximate size of the page table? **(GATE-2001) (2 Mark)**

(a) 16 MB

(c) 2 MB

(b) 8 MB

(d) 24 MB

Question 77

Q Consider a logical address space of 8 pages of 1024 words mapped with memory of 32 frames. How many bits are there in the physical address? **(NET-DEC-2011)**

(A) 9 bits

(C) 13 bits

(B) 11 bits

(D) 15 bits

Question 78

Q Consider a paging hardware with a TLB. Assume that the entire page table and all the pages are in the physical memory. It takes 10 milliseconds to search the TLB and 80 milliseconds to access the physical memory. If the TLB hit ratio is 0.6, the effective memory access time (in milliseconds) is _____. **(CS-2014) (2 Marks)**

Question 79

Q The hit ratio of a Translation Look Aside Buffer (TLAB) is 80%. It takes 20 nanoseconds (ns) to search TLAB and 100 ns to access main memory. The effective memory access time is _____. **(NET-SEP-2013)**

(A) 36 ns

(C) 122 ns

(B) 140 ns

(D) 40 ns

Question 80

Q A computer system implements a 40-bit virtual address, page size of 8 kilobytes, and a 128-entry translation look-aside buffer (TLB) organized into 32 sets each having four ways. Assume that the TLB tag does not store any process id. The minimum length of the TLB tag in bits is _____ **(GATE-2015) (2 Marks)**

(A) 20

(C) 11

(B) 10

(D) 22

Question 81

Q A CPU generates 32-bit virtual addresses. The page size is 4 KB. The processor has a translation look-aside buffer (TLB) which can hold a total of 128-page table entries and is 4-way set associative. The minimum size of the TLB tag is? **(GATE-2006) (2 Marks)**

(A) 11 bits **(B)** 13 bits **(C)** 15 bits **(D)** 20 bits

Question 82

Q A paging system tlb takes 10ns main memory takes 50ns what is effective memory access time if tlb hit ratio is 90%? **(GATE-2008) (1 Marks)**

a) 54

c) 65

b) 60

d) 75

Question 83

Q Which one of the following statements is FALSE? **(GATE 2022) (2 MARKS)**

(A) The TLB performs an associative search in parallel on all its valid entries using page number of incoming virtual address.

(B) If the virtual address of a word given by CPU has a TLB hit, but the subsequent search for the word results in a cache miss, then the word will always be present in the main memory.

(C) The memory access time using a given inverted page table is always same for all incoming virtual addresses.

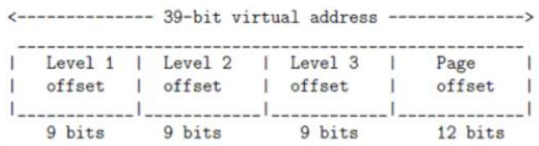
(D) In a system that uses hashed page tables, if two distinct virtual addresses V1 and V2 map to the same value while hashing, then the memory access time of these addresses will not be the same.

Question 84

Q Consider a computer system with 40-bit virtual addressing and page size of sixteen kilobytes. If the computer system has a one-level page table per process and each page table entry requires 48 bits, then the size of the per-process page table is _____ megabytes. **(GATE-2016) (2 Marks)**

Question 85

Q Consider a three-level page table to translate a 39-bit virtual address to a physical address as shown below:



The page size is 4 KB = (1KB = 210 bytes) and page table entry size at every level is 8 bytes. A process P is currently using 2 GB (1 GB = 230 bytes) virtual memory which OS mapped to 2 GB of physical memory. The minimum amount of memory required for the page table of P across all levels is _____ KB**(GATE 2021) (2 MARKS)**

(A)4108 (A)3081

(A)1027 (A)4698

Question 86

Q A computer uses 46–bit virtual address, 32–bit physical address, and a three–level paged page table organization. The page table base register stores the base address of the first–level table (T1), which occupies exactly one page. Each entry of T1 stores the base address of a page of the second–level table (T2). Each entry of T2 stores the base address of a page of the third–level table (T3). Each entry of T3 stores a page table entry (PTE). The PTE is 32 bits in size. The processor used in the computer has a 1 MB 16 way set associative virtually indexed physically tagged cache. The cache block size is 64 bytes.
What is the size of a page in KB in this computer? **(GATE-2013) (2 Marks)**

(A) 2 **(B)** 4 **(C)** 8 **(D)** 16

Question 87

Q In a paged memory, the page hit ratio is 0.40. The time required to access a page in secondary memory is equal to 120 ns. The time required to access a page in primary memory is 15 ns. The average time required to access a page is _____. **(NET-JULY-2018)**

(a) 105 **(b)** 68 **(c)** 75 **(d)** 78

Question 88

Q Consider a process executing on an operating system that uses demand paging. The average time for a memory access in the system is M units if the corresponding memory page is available in memory, and D units if the memory access causes a page fault. It has been experimental measured that the average time taken for a memory access in the process is X units.
Which one of the following is the correct expression for the page fault rate experienced by the process? **(GATE-2018) (2 Marks)**

(A) (D – M) / (X – M)

(B) (X – M) / (D – M)

(C) (D – X) / (D – M)

(D) (X – M) / (D – X)

Question 89

Q Let the page fault service time be 10ms in a computer with average memory access time being 20ns. If one-page fault is generated for every 10^6 memory accesses, what is the effective access time for the memory? **(GATE-2011) (1 Marks)**

(A) 21ns **(B)** 30ns **(C)** 23ns **(D)** 35ns

Question 90

Q Suppose the time to service a page fault is on the average 10 milliseconds, while a memory access takes 1 microsecond. Then a 99.99% hit ratio results in average memory access time of? **(GATE-2000) (1 Marks).**

(A)1.9999 milliseconds **(B)**1 millisecond

(C)9.999 microseconds **(D)**1.9999 microseconds

Question 91

Q Consider the reference string 0 1 2 3 0 1 4 0 1 2 3 4 If FIFO page replacement algorithm is used, then the number of page faults with three-page frames and four-page frames are _____ and _____ respectively. **(NET-JUNE-2016)**

a) 10, 9 **b)** 9, 9 **c)** 10, 10 **d)** 9,10

Question 92

Q Assume that there are 3 page frames which are initially empty. If the page reference string is 1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6, the number of page faults using the optimal replacement policy is_____. **(GATE-2014) (2 Marks)**

Question 93

Q Consider a demand paging system with four page frames (initially empty) and LRU page replacement policy. For the following page reference string, the page fault rate, defined as the ratio of number of page faults to the number of memory accesses (rounded off to one decimal place) is_____. **(GATE 2022) (2 MARKS)**

7, 2, 7, 3, 2, 5, 3, 4, 6, 7, 7, 1, 5, 6,1

Question 94

Q. In which one of the following page replacement algorithms it is possible for the page fault rate to increase even when the number of allocated frames increases? **(GATE-2016) (1 Marks)**

(a) LRU (Least Recently Used **(b)** OPT (Optimal Page Replacement)

(c) MRU (Most Recently Used) **(d)** FIFO (First In First Out)

Question 95

Q A system uses FIFO policy for page replacement. It has 4-page frames with no pages loaded to begin with. The system first accesses 100 distinct pages in some order and then accesses the same 100 pages but now in the reverse order. How many page faults will occur? **(GATE-2010) (1 Marks)**

(A) 196 **(B)** 192 **(C)** 197 **(D)** 195

Question 96

Q A virtual memory system uses First In First Out (FIFO) page replacement policy and allocates a fixed number of frames to a process. Consider the following statements: **(GATE-2007) (2 Marks)**

P: Increasing the number of page frames allocated to a process sometimes increases the page fault rate.

Q: Some programs do not exhibit locality of reference.

Which one of the following is TRUE?
(A) Both P and Q are true, and Q is the reason for P
(B) Both P and Q are true, but Q is not the reason for P.
(C) P is false, but Q is true
(D) Both P and Q are false.

Question 97

Q A process has been allocated 3-page frames. Assume that none of the pages of the process are available in the memory initially. The process makes the following sequence of page references (reference string): 1, 2, 1, 3, 7, 4, 5, 6, 3, 1. If optimal page replacement policy is used, how many page faults occur for the above reference string? **(GATE-2007) (2 Marks)**
(A) 7 **(B)** 8 **(C)** 9 **(D)** 10

Question 98

Q A system uses 3-page frames for storing process pages in main memory. It uses the Least Recently Used (LRU) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below? 4, 7, 6, 1, 7, 6, 1, 2, 7, 2 **(GATE-2014) (2 Marks)**

Question 99

Q Consider a computer system with ten physical page frames. The system is provided with an access sequence $(a_1,a_2,\ldots,a_{20},a_1,a_2,\ldots,a_{20})$, where each a_i is a distinct virtual page number. The difference in the number of page faults between the last-in-first-out page replacement policy and the optimal page replacement policy is _____. **(GATE-2016) (1 Marks)**

Question 100

Q A computer has twenty physical page frames which contain pages numbered 101 through 120. Now a program accesses the pages numbered 1, 2, ..., 100 in that order, and repeats the access sequence THRICE. Which one of the following page replacement policies experiences the same number of page faults as the optimal page replacement policy for this program? **(GATE-2014) (2 Marks)**
(A) Least-recently-used **(B)** First-in-first-out
(C) Last-in-first-out **(D)** Most-recently-used

Question 101

Q Consider a disk system with 100 cylinders. The requests to access the cylinders occur in following sequence 4, 34, 10, 7, 19, 73, 2, 15, 6, 20. Assuming that the head is currently at cylinder 50, what is the time taken to satisfy all requests if it takes 1ms to move from one cylinder to adjacent one and shortest seek time first policy is used? **(GATE-2009) (1 Marks)**

(A) 95 ms **(B)** 119 ms **(C)** 233 ms **(D)** 276 ms

Question 102

Q Suppose a disk has 201 cylinders, numbered from 0 to 200. At some time, the disk arm is at cylinder 100, and there is a queue of disk access requests for cylinders 30, 85, 90, 100, 105, 110, 135 and 145. If Shortest-Seek Time First (SSTF) is being used for scheduling the disk access, the request for cylinder 90 is serviced after servicing _____ number of requests. **(GATE-2014) (1 Marks)**

Question 103

Q Suppose the following disk request sequence (track numbers) for a disk with 100 tracks is given: 45, 20, 90, 10, 50, 60, 80, 25, 70. Assume that the initial position of the R/W head is on track 50. The additional distance that will be traversed by the R/W head when the Shortest Seek Time First (SSTF) algorithm is used compared to the SCAN (Elevator) algorithm (assuming that SCAN algorithm moves towards 100 when it starts execution) is _____ tracks **(GATE-2015) (2 Marks)**

Question 104

Q Consider the situation in which the disk read/write head is currently located at track 45 (of tracks 0-255) and moving in the positive direction. Assume that the following track requests have been made in this order: 40, 67, 11, 240, 87. What is the order in which optimized C-SCAN would service these requests and what is the total seek distance?**(GATE-2015) (2 Marks)**

(A) 600 **(B)** 810 **(C)** 505 **(D)** 550

Question 105

Q Consider a disk queue with requests for I/O to blocks on cylinders 47, 38, 121, 191, 87, 11, 92, 10. The C-LOOK scheduling algorithm is used. The head is initially at cylinder number 63, moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____. **(GATE-2018) (2 Marks)**

Question 106

Q Consider an operating system capable of loading and executing a single sequential user process at a time. The disk head scheduling algorithm used is First Come First Served (FCFS). If FCFS is replaced by Shortest Seek Time First (SSTF), claimed by the vendor to give 50% better benchmark results, what is the expected improvement in the I/O performance of user programs? **(GATE-2004) (1 Marks)**

(A) 50% **(B)** 40% **(C)** 25% **(D)** 0%

Question 107

Q Consider a storage disk with 4 platters (numbered as 0, 1, 2 and 3), 200 cylinders (numbered as 0, 1, ..., 199), and 256 sectors per track (numbered as 0, 1, ... 255). The following 6 disk requests of the form [sector number, cylinder number, platter number] are received by the disk controller at the same time:

[120, 72, 2], [180, 134, 1], [60, 20, 0], [212, 86, 3], [56, 116, 2], [118, 16, 1]

Currently head is positioned at sector number 100 of cylinder 80, and is moving towards higher cylinder numbers. The average power dissipation in moving the head over 100 cylinders is 20 milliwatts and for reversing the direction of the head movement once is 15 milliwatts. Power dissipation associated with rotational latency and switching of head between different platters is negligible.

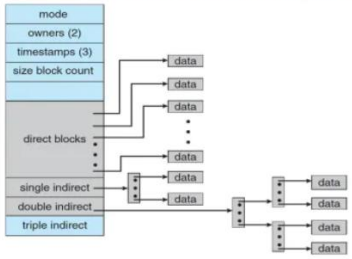
The total power consumption in milliwatts to satisfy all of the above disk requests using the Shortest Seek Time First disk scheduling algorithm is _____. **(GATE-2018) (2 Marks)**

Question 108

Q Consider a disk pack with a seek time of 4 milliseconds and rotational speed of 10000 rotations per minute (RPM). It has 600 sectors per track and each sector can store 512 bytes of data. Consider a file stored in the disk. The file contains 2000 sectors. Assume that every sector access necessitates a seek, and the average rotational latency for accessing each sector is half of the time for one complete rotation. The total time (in milliseconds) needed to read the entire file is _____. **(GATE-2015) (2 Marks)**

Question 109

Q A file system with 300 Gbyte disk uses a file descriptor with 8 direct block addresses, 1 indirect block address and 1 doubly indirect block address. The size of each disk block is 128 Bytes and the size of each disk block address is 8 Bytes. The maximum possible file size in this file system is **(GATE-2012) (2 Marks)**



(A) 3 Kbytes

(B) 35 Kbytes

(C) 280 Bytes

(D) Dependent on the size of the disk

Question 110

Q The index node (inode) of a Unix-like file system has 12 direct, one single-indirect and one double-indirect pointers. The disk block size is 4 kB, and the disk block address is 32-bits long. The maximum possible file size is (rounded off to 1 decimal place) _____ GB. **(GATE-2014) (2 Marks)**

Question 111

Q A Unix-style i-node has 10 direct pointers and one single, one double and one triple indirect pointer. Disk block size is 1 Kbyte, disk block address is 32 bits, and 48-bit integers are used. What is the maximum possible file size? **(GATE-2004) (2 Marks)**

(A) 2^{24} bytes

(C) 2^{34} bytes

(B) 2^{32} bytes

(D) 2^{48} bytes

Question 112

Q A process executes the code

```
fork();
fork();
fork();
```

the total number of child processes created is **(GATE - 2012) (1 Marks)**

a) 3

c) 7

b) 4

d) 8

Question 113

Q A process executes the following code **(GATE - 2008) (2 Marks)**

```
for (i=0; i<n; i++)
fork();
```

a) n

c) 2^n

b) $(2^n) - 1$

d) $(2^{n+1}) - 1$

Question 114

Q The following C program is executed on a Unix/Linux system:

```
#include <unistd.h>
int main ()
{
    int i ;
    for (i=0; i<10; i++)
        if (i%2 == 0)
                fork ( ) ;
    return 0 ;
}
```

The total number of child processes created is _____**(GATE-2019) (1 Marks)**

Question 115

Q Let u, v be the values printed by the parent process, and x, y be the values printed by the child process. Which one of the following is TRUE? **(GATE-2005)(2 Marks)**

a) u = x + 10 and v = y

b) u = x + 10 and v != y

c) u + 10 = x and v = y

d) u + 10 = x and v != y

```
if (fork() == 0)
{
        a = a + 5;
        printf ("%d, %d /n", a, &a);
}
else
{
        a = a –5;
        printf("%d, %d /n", a, &a);
}
```

Question 116

Q Which one of the following is FALSE? **(GATE - 2014) (1 Marks)**

(A) User level threads are not scheduled by the kernel.

(B) When a user level thread is blocked, all other threads of its process are blocked.

(C) Context switching between user level threads is faster than context switching between kernel level threads.

(D) Kernel level threads cannot share the code segment

Question 117

Q Let the time taken to switch between user and kernel modes of execution be t_1 while the time taken to switch between two processes be t_2. Which of the following is TRUE? **(GATE-2011) (1 Marks)**

(A) $t_1 > t_2$

(B) $t_1 = t_2$

(C) $t_1 < t_2$

(D) nothing can be said about the relation between t_1 and t_2

Question 118

Q Consider the following statements about user level threads and kernel level threads. Which one of the following statements is FALSE? **(GATE - 2007) (1 Marks)**

A) Context switch time is longer for kernel level threads than for user level threads.

B) User level threads do not need any hardware support.

C) Related kernel level threads can be scheduled on different processors in a multi-processor system

D) Blocking one kernel level thread blocks all related threads

---

Question 119

Q Consider the following statements with respect to user-level threads and kernel supported threads

i. context switch is faster with kernel-supported threads

ii. for user-level threads, a system call can block the entire process

iii. Kernel supported threads can be scheduled independently

iv. User level threads are transparent to the kernel

Which of the above statements are true? **(GATE-2004) (1 Marks)**

(A) (ii), (iii) and (iv) only

(B) (ii) and (iii) only

(C) (i) and (iii) only

(D) (i) and (ii) only

Question 120

Q Which of the following is/are shared by all the threads in a process? **(GATE - 2017) (1 Marks)**

I. Program Counter **II.** Stack **III.** Address space **IV.** Registers

(A) I and II only

(C) IV only

(B) III only

(D) III and IV only

ANSWER KEY

1	C	**31**	B	**61**	B	**91**	D		
2	B	**32**	80	**62**	C	**92**	7		
3	D	**33**	A	**63**	C	**93**	0.6		
4	12	**34**	C	**64**	A	**94**	D		
5	C	**35**	B	**65**	A	**95**	A		
6	A	**36**	D	**66**	B	**96**	B		
7	D	**37**	A	**67**	B	**97**	A		
8	B	**38**	B	**68**	A	**98**	6		
9	2	**39**	B	**69**	A	**99**	1		
10	A	**40**	2	**70**	A,D	**100**	D		

11	3	**41**	B	**71**	A	**101**	B		
12	A	**42**	D	**72**	B	**102**	3		
13	B	**43**	D	**73**	B	**103**	10		
14	29	**44**	D	**74**	4	**104**	C		
15	12	**45**	C	**75**	36	**105**	346		
16	D	**46**	C	**76**	C	**106**	D		
17	C	**47**	B	**77**	D	**107**	85		
18	D	**48**	C	**78**	122	**108**	14020		
19	A,C,D	**49**	B	**79**	B	**109**	B		
20	A	**50**	C	**80**	D	**110**	4.004		
21	C	**51**	7	**81**	C	**111**	C		
22	D	**52**	8	**82**	C	**112**	C		
23	B	**53**	2	**83**	C	**113**	B		
24	1000	**54**	A,B	**84**	384	**114**	31		
25	D	**55**	A	**85**	4108	**116**	D		
26	B	**56**	D	**86**	C	**117**	C		
27	B	**57**	B	**87**	D	**118**	D		
28	A	**58**	C	**88**	B	**119**	A		
29	C	**59**	2	**89**	B	**120**	B		
30	3	**60**	D	**90**	D				