**GATE PRACTICE PROBLEMS BASED ON CPU SCHEDULING ALGORITHMS**

**Problem-01:**

Consider three process, 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 does the CPU remain idle?

1. 0%
2. 10.6%
3. 30.0%
4. 89.4%

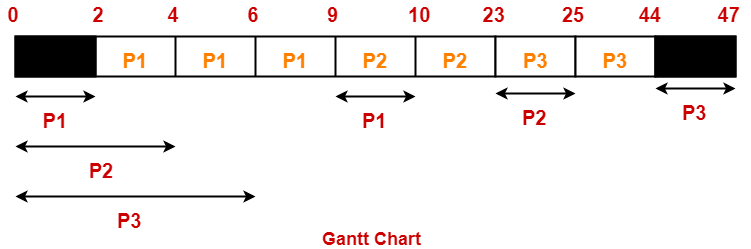
**Solution-**

According to question, we have-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Total Burst Time** | **I/O Burst** | **CPU Burst** | **I/O Burst** |
| **Process P1** | 10 | 2 | 7 | 1 |
| **Process P2** | 20 | 4 | 14 | 2 |
| **Process P3** | 30 | 6 | 21 | 3 |

The scheduling algorithm used is Shortest Remaining Time First.

**Gantt Chart-**



Percentage of time CPU remains idle

= (5 / 47) x 100

= 10.638%

Thus, Option (B) is correct.

**Problem-02:**

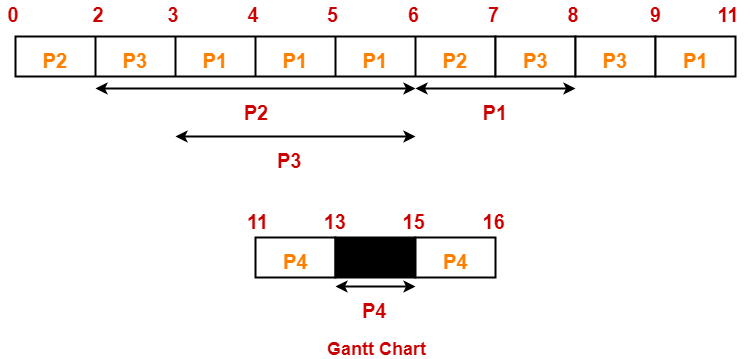
Consider the set of 4 processes whose arrival time and burst time are given below-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Process No.** | **Arrival Time** | **Burst Time** | | |
| **CPU Burst** | **I/O Burst** | **CPU Burst** |
| **P1** | 0 | 3 | 2 | 2 |
| **P2** | 0 | 2 | 4 | 1 |
| **P3** | 2 | 1 | 3 | 2 |
| **P4** | 5 | 2 | 2 | 1 |

If the CPU scheduling policy is Shortest Remaining Time First, calculate the average waiting time and average turn around time.

**Solution-**

**Gantt Chart-**



Now, we know-

* Turn Around time = Exit time – Arrival time
* Waiting time = Turn Around time – Burst time

**Also read-** [**Various Times Of Process**](https://www.gatevidyalay.com/turn-around-time-response-time-waiting-time/)

|  |  |  |  |
| --- | --- | --- | --- |
| **Process Id** | **Exit time** | **Turn Around time** | **Waiting time** |
| P1 | 11 | 11 – 0 = 11 | 11 – (3+2) = 6 |
| P2 | 7 | 7 – 0 = 7 | 7 – (2+1) = 4 |
| P3 | 9 | 9 – 2 = 7 | 7 – (1+2) = 4 |
| P4 | 16 | 16 – 5 = 11 | 11 – (2+1) = 8 |

Now,

* Average Turn Around time = (11 + 7 + 7 + 11) / 4 = 36 / 4 = 9 units
* Average waiting time = (6 + 4 + 4 + 8) / 4 = 22 / 5 = 4.4 units

**Problem-03:**

Consider the set of 4 processes whose arrival time and burst time are given below-

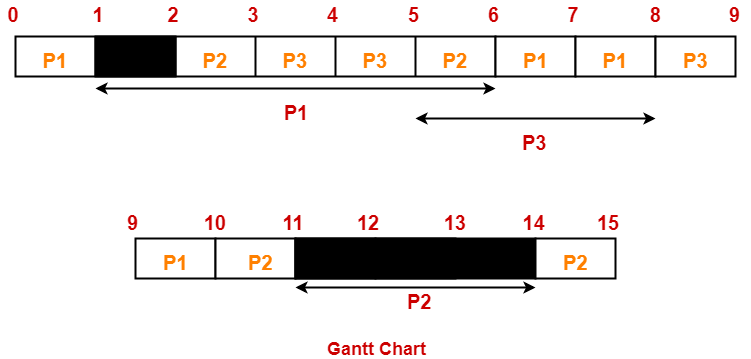
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Process No.** | **Arrival Time** | **Priority** | **Burst Time** | | |
| **CPU Burst** | **I/O Burst** | **CPU Burst** |
| **P1** | 0 | 2 | 1 | 5 | 3 |
| **P2** | 2 | 3 | 3 | 3 | 1 |
| **P3** | 3 | 1 | 2 | 3 | 1 |

If the CPU scheduling policy is Priority Scheduling, calculate the average waiting time and average turn around time. (Lower number means higher priority)

**Solution-**

The scheduling algorithm used is Priority Scheduling.

**Gantt Chart-**



Now, we know-

* Turn Around time = Exit time – Arrival time
* Waiting time = Turn Around time – Burst time

|  |  |  |  |
| --- | --- | --- | --- |
| **Process Id** | **Exit time** | **Turn Around time** | **Waiting time** |
| P1 | 10 | 10 – 0 = 10 | 10 – (1+3) = 6 |
| P2 | 15 | 15 – 2 = 13 | 13 – (3+1) = 9 |
| P3 | 9 | 9 – 3 = 6 | 6 – (2+1) = 3 |

Now,

* Average Turn Around time = (10 + 13 + 6) / 3 = 29 / 3 = 9.67 units
* Average waiting time = (6 + 9 + 3) / 3 = 18 / 3 = 6 units

**PRACTICE PROBLEMS BASED ON DEADLOCK IN OS-**

**Problem-01:**

A system is having 3 user processes each requiring 2 units of resource R. The minimum number of units of R such that no deadlock will occur-

1. 3
2. 5
3. 4
4. 6

**Solution-**

In worst case,

The number of units that each process holds = One less than its maximum demand

So,

* Process P1 holds 1 unit of resource R
* Process P2 holds 1 unit of resource R
* Process P3 holds 1 unit of resource R

Thus,

* Maximum number of units of resource R that ensures deadlock = 1 + 1 + 1 = 3
* Minimum number of units of resource R that ensures no deadlock = 3 + 1 = 4

**Problem-02:**

A system is having 10 user processes each requiring 3 units of resource R. The minimum number of units of R such that no deadlock will occur \_\_\_\_\_?

**Solution-**

In worst case,

The number of units that each process holds = One less than its maximum demand

So,

* Process P1 holds 2 units of resource R
* Process P2 holds 2 units of resource R
* Process P3 holds 2 units of resource R and so on.
* Process P10 holds 2 units of resource R

Thus,

* Maximum number of units of resource R that ensures deadlock = 10 x 2 = 20
* Minimum number of units of resource R that ensures no deadlock = 20 + 1 = 21

**Problem-03:**

A system is having 3 user processes P1, P2 and P3 where P1 requires 2 units of resource R, P2 requires 3 units of resource R, P3 requires 4 units of resource R. The minimum number of units of R that ensures no deadlock is \_\_\_\_\_?

**Solution-**

In worst case,

The number of units that each process holds = One less than its maximum demand

So,

* Process P1 holds 1 unit of resource R
* Process P2 holds 2 units of resource R
* Process P3 holds 3 units of resource R

Thus,

* Maximum number of units of resource R that ensures deadlock = 1 + 2 + 3 = 6
* Minimum number of units of resource R that ensures no deadlock = 6 + 1 = 7

**Problem-04:**

A system is having 3 user processes P1, P2 and P3 where P1 requires 21 units of resource R, P2 requires 31 units of resource R, P3 requires 41 units of resource R. The minimum number of units of R that ensures no deadlock is \_\_\_\_\_?

**Solution-**

In worst case,

The number of units that each process holds = One less than its maximum demand

So,

* Process P1 holds 20 units of resource R
* Process P2 holds 30 units of resource R
* Process P3 holds 40 units of resource R

Thus,

* Maximum number of units of resource R that ensures deadlock = 20 + 30 + 40 = 90
* Minimum number of units of resource R that ensures no deadlock = 90 + 1 = 91

**Problem-05:**

If there are 6 units of resource R in the system and each process in the system requires 2 units of resource R, then how many processes can be present at maximum so that no deadlock will occur?

**Solution-**

In worst case,

The number of units that each process holds = One less than its maximum demand

So,

* Process P1 holds 1 unit of resource R
* Process P2 holds 1 unit of resource R
* Process P3 holds 1 unit of resource R
* Process P4 holds 1 unit of resource R
* Process P5 holds 1 unit of resource R
* Process P6 holds 1 unit of resource R

Thus,

* Minimum number of processes that ensures deadlock = 6
* Maximum number of processes that ensures no deadlock = 6 – 1 = 5

**Problem-06:**

If there are 6 units of resource R in the system and each process in the system requires 3 units of resource R, then how many processes can be present at maximum so that no deadlock will occur?

**Solution-**

In worst case,

The number of units that each process holds = One less than its maximum demand

So,

* Process P1 holds 2 units of resource R
* Process P2 holds 2 units of resource R
* Process P3 holds 2 units of resource R

Thus,

* Minimum number of processes that ensures deadlock = 3
* Maximum number of processes that ensures no deadlock = 3 – 1 = 2

**Problem-07:**

If there are 100 units of resource R in the system and each process in the system requires 2 units of resource R, then how many processes can be present at maximum so that no deadlock will occur?

**Solution-**

In worst case,

The number of units that each process holds = One less than its maximum demand

So,

* Process P1 holds 1 unit of resource R
* Process P2 holds 1 unit of resource R
* Process P3 holds 1 unit of resource R and so on.
* Process P100 holds 1 unit of resource R

Thus,

* Minimum number of processes that ensures deadlock = 100
* Maximum number of processes that ensures no deadlock = 100 – 1 = 99

**Problem-08:**

If there are 100 units of resource R in the system and each process in the system requires 4 units of resource R, then how many processes can be present at maximum so that no deadlock will occur?

**Solution-**

In worst case,

The number of units that each process holds = One less than its maximum demand

So,

* Process P1 holds 3 units of resource R
* Process P2 holds 3 units of resource R
* Process P3 holds 3 units of resource R and so on.
* Process P33 holds 3 units of resource R
* Process P34 holds 1 unit of resource R

Thus,

* Minimum number of processes that ensures deadlock = 34
* Maximum number of processes that ensures no deadlock = 34 – 1 = 33

**Problem-09:**

If there are 100 units of resource R in the system and each process in the system requires 5 units of resource R, then how many processes can be present at maximum so that no deadlock will occur?

**Solution-**

In worst case,

The number of units that each process holds = One less than its maximum demand

So,

* Process P1 holds 4 units of resource R
* Process P2 holds 4 units of resource R
* Process P3 holds 4 units of resource R and so on.
* Process P25 holds 4 units of resource R

Thus,

* Minimum number of processes that ensures deadlock = 25
* Maximum number of processes that ensures no deadlock = 25 – 1 = 24

**Problem-10:**

A computer system has 6 tape drives with n processes competing for them. Each process needs 3 tape drives. The maximum value of n for which the system is guaranteed to be deadlock free-

1. 2
2. 3
3. 4
4. 1

**Solution-**

In worst case,

The number of tape drives that each process holds = One less than its maximum demand

So,

* Process P1 holds 2 tape drives
* Process P2 holds 2 tape drives
* Process P3 holds 2 tape drives

Thus,

* Minimum number of processes that ensures deadlock = 3
* Maximum number of processes that ensures no deadlock = 3 – 1 = 2

**Problem-11:**

Consider a system having m resources of the same type. These resources are shared by 3 processes A, B and C which have peak demands of 3, 4 and 6 respectively. For what value of m, deadlock will not occur?

1. 7
2. 9
3. 10
4. 13

**Solution-**

In worst case,

The number of units that each process holds = One less than its maximum demand

So,

* Process A holds 2 units of resource R
* Process B holds 3 units of resource R
* Process C holds 5 units of resource R

Thus,

* Maximum number of units of resource R that ensures deadlock = 2 + 3 + 5 = 10
* Minimum number of units of resource R that ensures no deadlock = 10 + 1 = 11

So, any number of units greater than 11 will ensure no deadlock.

Thus, Option (D) is correct.

**Problem-12:**

Consider a system having m resources of the same type being shared by n processes. Resources can be requested and released by processes only one at a time. The system is deadlock free if and only if-

1. The sum of all max needs is < m+n
2. The sum of all max needs is > m+n
3. Both of above
4. None of these

**Solution-**

We have derived above-

Maximum number of units of resource R that ensures deadlock = (∑xi – n)

Thus, For no deadlock occurrence,

Number of units of resource R must be > (∑xi – n)

i.e. m > (∑xi – n)

or ∑xi < m + n

Thus, Correct Option is (A).

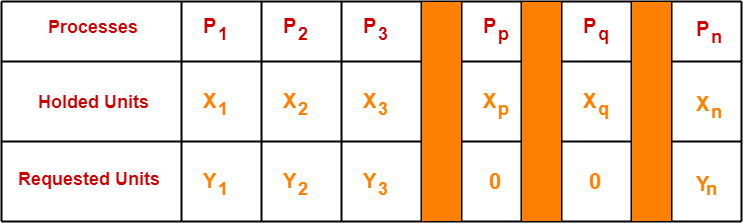
**Problem-13:**

Consider the following snapshot of a system running n processes. Process i is holding xi instances of a resource R for 1<=i<=n. Currently, all instances of R are occupied. Further, for all i, process i has placed a request for an additional yi instances while holding the xi instances it already has. There are exactly two processes p and q such that yp = yq = 0. Which of the following can serve as a necessary condition to guarantee that the system is not approaching a deadlock?

1. min(xp, xq) < maxk≠p,qyk
2. xp + xq >= mink≠p,qyk
3. min(xp, xq) < 1
4. min(xp, xq) >1

**Solution-**

According to question, we have-



* Clearly, processes Pp and Pq do not require any additional resource.
* So they continue their execution.
* After getting executed completely, they release the units allocated to them.
* Thus, the total units that get free up = xp + xq

Now,

* To ensure that other processes are executed without any deadlock, the total amount of units freely available currently ( xp + xq ) must be able to meet the requirements of some other process.
* If available ( xp + xq) units could not meet the requirement of any other process, then certainly there would be deadlock.

Thus, for no deadlock, the necessary condition is-

xp + xq >= min yk where k ≠ p, q

Thus, Correct Option is (B).

**NOTE-**

* It is very important to note that the above condition is just a necessary condition and not at all a sufficient condition to avoid the deadlock.
* The above condition just ensures that the system is able to proceed from the current state.
* It does not guarantee that there won’t be a deadlock before all the other processes are finished.
* The sufficient condition to avoid the deadlock would be either xp + xq >= ∑ yi or xp + xq >= max ykwhere k ≠ p, q.

**PRACTICE PROBLEMS BASED ON BANKER’S ALGORITHM-**

**Problem-01:**

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?

1. P0
2. P1
3. P2
4. None of the above since the system is in a deadlock

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Alloc** | | | **Request** | | |
|  | **X** | **Y** | **Z** | **X** | **Y** | **Z** |
| **P0** | 1 | 2 | 1 | 1 | 0 | 3 |
| **P1** | 2 | 0 | 1 | 0 | 1 | 2 |
| **P2** | 2 | 2 | 1 | 1 | 2 | 0 |

**Solution-**

According to question-

* Total = [ X Y Z ] = [ 5 5 5 ]
* Total \_Alloc = [ X Y Z ] = [5 4 3]

Now,

Available

= Total – Total\_Alloc

= [ 5 5 5 ] – [5 4 3]

= [ 0 1 2 ]

**Step-01:**

* With the instances available currently, only the requirement of the process P1 can be satisfied.
* So, process P1 is allocated the requested resources.
* It completes its execution and then free up the instances of resources held by it.

Then,

Available

= [ 0 1 2 ] + [ 2 0 1]

= [ 2 1 3 ]

**Step-02:**

* With the instances available currently, only the requirement of the process P0 can be satisfied.
* So, process P0 is allocated the requested resources.
* It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [ 2 1 3 ] + [ 1 2 1 ]

= [ 3 3 4 ]

**Step-03:**

* With the instances available currently, the requirement of the process P2 can be satisfied.
* So, process P2 is allocated the requested resources.
* It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [ 3 3 4 ] + [ 2 2 1 ]

= [ 5 5 5 ]

Thus,

* There exists a safe sequence P1, P0, P2 in which all the processes can be executed.
* So, the system is in a safe state.
* Process P2 will be executed at last.

Thus, Option (C) is correct.

**Problem-02:**

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 P0, P1 and P2. 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.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Allocation** | | | **Max** | | |
|  | **X** | **Y** | **Z** | **X** | **Y** | **Z** |
| **P0** | 0 | 0 | 1 | 8 | 4 | 3 |
| **P1** | 3 | 2 | 0 | 6 | 2 | 0 |
| **P2** | 2 | 1 | 1 | 3 | 3 | 3 |

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 safe state. Consider the following independent requests for additional resources in the current state-

REQ1: P0 requests 0 units of X, 0 units of Y and 2 units of Z

REQ2: P1 requests 2 units of X, 0 units of Y and 0 units of Z

Which of the following is TRUE?

1. Only REQ1 can be permitted
2. Only REQ2 can be permitted
3. Both REQ1 and REQ2 can be permitted
4. Neither REQ1 nor REQ2 can be permitted

**Solution-**

According to question,

Available = [ X Y Z ] = [ 3 2 2 ]

Now,

Need = Max – Allocation

So, we have-

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Allocation** | | | **Max** | | | **Need** | | |
|  | **X** | **Y** | **Z** | **X** | **Y** | **Z** | **X** | **Y** | **Z** |
| **P0** | 0 | 0 | 1 | 8 | 4 | 3 | 8 | 4 | 2 |
| **P1** | 3 | 2 | 0 | 6 | 2 | 0 | 3 | 0 | 0 |
| **P2** | 2 | 1 | 1 | 3 | 3 | 3 | 1 | 2 | 2 |

Currently, the system is in safe state.

(It is given in question. If we want, we can check)

**Checking Whether REQ1 Can Be Entertained-**

* Need of P0 = [ 0 0 2 ]
* Available = [ 3 2 2 ]

Clearly,

* With the instances available currently, the requirement of REQ1 can be satisfied.
* So, banker’s algorithm assumes that the request REQ1 is entertained.
* It then modifies its data structures as-

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Allocation** | | | **Max** | | | **Need** | | |
|  | **X** | **Y** | **Z** | **X** | **Y** | **Z** | **X** | **Y** | **Z** |
| **P0** | 0 | 0 | **3** | 8 | 4 | 3 | 8 | 4 | **0** |
| **P1** | 3 | 2 | 0 | 6 | 2 | 0 | 3 | 0 | 0 |
| **P2** | 2 | 1 | 1 | 3 | 3 | 3 | 1 | 2 | 2 |

Available

= [ 3 2 2 ] – [ 0 0 2 ]

= [ 3 2 0 ]

* Now, it follows the safety algorithm to check whether this resulting state is a safe state or not.
* If it is a safe state, then REQ1 can be permitted otherwise not.

**Step-01:**

* With the instances available currently, only the requirement of the process P1 can be satisfied.
* So, process P1 is allocated the requested resources.
* It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [ 3 2 0 ] + [ 3 2 0 ]

= [ 6 4 0 ]

Now,

* It is not possible to entertain any process.
* The system has entered the deadlock state which is an unsafe state.
* Thus, REQ1 will not be permitted.

**Checking Whether REQ2 Can Be Entertained-**

* Need of P1 = [ 2 0 0 ]
* Available = [ 3 2 2 ]

Clearly,

* With the instances available currently, the requirement of REQ1 can be satisfied.
* So, banker’s algorithm assumes the request REQ2 is entertained.
* It then modifies its data structures as-

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Allocation** | | | **Max** | | | **Need** | | |
|  | **X** | **Y** | **Z** | **X** | **Y** | **Z** | **X** | **Y** | **Z** |
| **P0** | 0 | 0 | 1 | 8 | 4 | 3 | 8 | 4 | 2 |
| **P1** | **5** | 2 | 0 | 6 | 2 | 0 | **1** | 0 | 0 |
| **P2** | 2 | 1 | 1 | 3 | 3 | 3 | 1 | 2 | 2 |

Available

= [ 3 2 2 ] – [ 2 0 0 ]

= [ 1 2 2 ]

* Now, it follows the safety algorithm to check whether this resulting state is a safe state or not.
* If it is a safe state, then REQ2 can be permitted otherwise not.

**Step-01:**

* With the instances available currently, only the requirement of the process P1 can be satisfied.
* So, process P1 is allocated the requested resources.
* It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [ 1 2 2 ] + [ 5 2 0 ]

= [ 6 4 2 ]

**Step-02:**

* With the instances available currently, only the requirement of the process P2 can be satisfied.
* So, process P2 is allocated the requested resources.
* It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [ 6 4 2 ] + [ 2 1 1 ]

= [ 8 5 3 ]

**Step-03:**

* With the instances available currently, the requirement of the process P0 can be satisfied.
* So, process P0 is allocated the requested resources.
* It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [ 8 5 3 ] + [ 0 0 1 ]

= [ 8 5 4 ]

Thus,

* There exists a safe sequence P1, P2, P0 in which all the processes can be executed.
* So, the system is in a safe state.
* Thus, REQ2 can be permitted.

Thus, Correct Option is (B).

**Problem-03:**

A system has 4 processes and 5 allocatable resource. The current allocation and maximum needs are as follows-

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Allocated** | | | | | **Maximum** | | | | |
| **A** | 1 | 0 | 2 | 1 | 1 | 1 | 1 | 2 | 1 | 3 |
| **B** | 2 | 0 | 1 | 1 | 0 | 2 | 2 | 2 | 1 | 0 |
| **C** | 1 | 1 | 0 | 1 | 1 | 2 | 1 | 3 | 1 | 1 |
| **D** | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 2 | 2 | 0 |

If Available = [ 0 0 X 1 1 ], what is the smallest value of x for which this is a safe state?

**Solution-**

Let us calculate the additional instances of each resource type needed by each process.

We know,

Need = Maximum – Allocation

So, we have-

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Need** | | | | |
| **A** | 0 | 1 | 0 | 0 | 2 |
| **B** | 0 | 2 | 1 | 0 | 0 |
| **C** | 1 | 0 | 3 | 0 | 0 |
| **D** | 0 | 0 | 1 | 1 | 0 |

**Case-01: For X = 0**

If X = 0, then-

Available

= [ 0 0 0 1 1 ]

* With the instances available currently, the requirement of any process can not be satisfied.
* So, for X = 0, system remains in a deadlock which is an unsafe state.

**Case-02: For X = 1**

If X = 1, then-

Available

= [ 0 0 1 1 1 ]

**Step-01:**

* With the instances available currently, only the requirement of the process D can be satisfied.
* So, process D is allocated the requested resources.
* It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [ 0 0 1 1 1 ] + [ 1 1 1 1 0 ]

= [ 1 1 2 2 1 ]

* With the instances available currently, the requirement of any process can not be satisfied.
* So, for X = 1, system remains in a deadlock which is an unsafe state.

**Case-02: For X = 2**

If X = 2, then-

Available

= [ 0 0 2 1 1 ]

**Step-01:**

* With the instances available currently, only the requirement of the process D can be satisfied.
* So, process D is allocated the requested resources.
* It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [ 0 0 2 1 1 ] + [ 1 1 1 1 0 ]

= [ 1 1 3 2 1 ]

**Step-02:**

* With the instances available currently, only the requirement of the process C can be satisfied.
* So, process C is allocated the requested resources.
* It completes its execution and then free up the instances of resources held by it.

Then-

Available

= [ 1 1 3 2 1 ] + [ 1 1 0 1 1 ]

= [ 2 2 3 3 2 ]

**Step-03:**

* With the instances available currently, the requirement of both the processes A and B can be satisfied.
* So, processes A and B are allocated the requested resources one by one.
* They complete their execution and then free up the instances of resources held by it.

Then-

Available

= [ 2 2 3 3 2 ] + [ 1 0 2 1 1 ] + [ 2 0 1 1 0 ]

= [ 5 2 6 5 3 ]

Thus,

* There exists a safe sequence in which all the processes can be executed.
* So, the system is in a safe state.
* Thus, minimum value of X that ensures system is in safe state = 2.

**PRACTICE PROBLEMS BASED ON CONTIGUOUS MEMORY ALLOCATION-**

**Problem-01:**

Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB and 250 KB. These partitions need to be allocated to four processes of sizes 357 KB, 210 KB, 468 KB and 491 KB in that order.

Perform the allocation of processes using-

1. First Fit Algorithm
2. Best Fit Algorithm
3. Worst Fit Algorithm

**Solution-**

According to question,

The main memory has been divided into fixed size partitions as-



Let us say the given processes are-

* Process P1 = 357 KB
* Process P2 = 210 KB
* Process P3 = 468 KB
* Process P4 = 491 KB

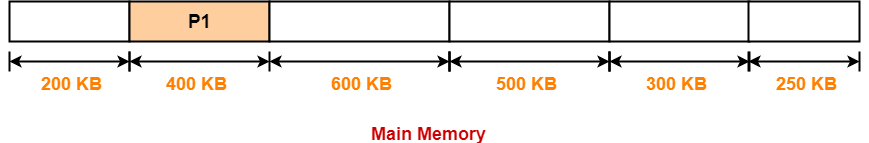
**Allocation Using First Fit Algorithm-**

In First Fit Algorithm,

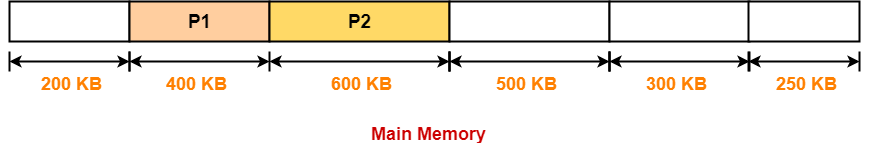
* Algorithm starts scanning the partitions serially.
* When a partition big enough to store the process is found, it allocates that partition to the process.

The allocation of partitions to the given processes is shown below-

**Step-01:**



**Step-02:**



**Step-03:**



**Step-04:**

* Process P4 can not be allocated the memory.
* This is because no partition of size greater than or equal to the size of process P4 is available.

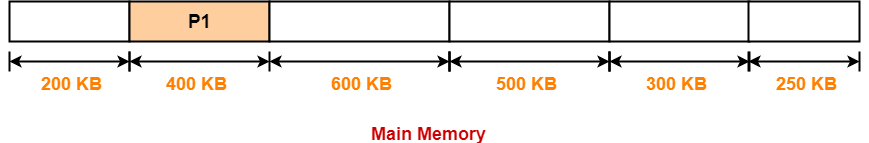
**Allocation Using Best Fit Algorithm-**

In Best Fit Algorithm,

* Algorithm first scans all the partitions.
* It then allocates the partition of smallest size that can store the process.

The allocation of partitions to the given processes is shown below-

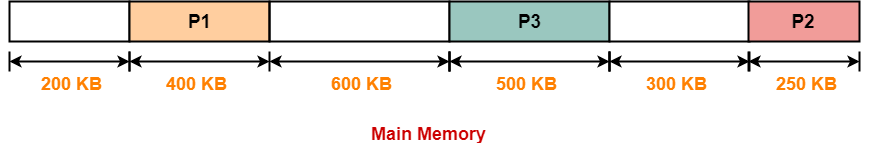
**Step-01:**



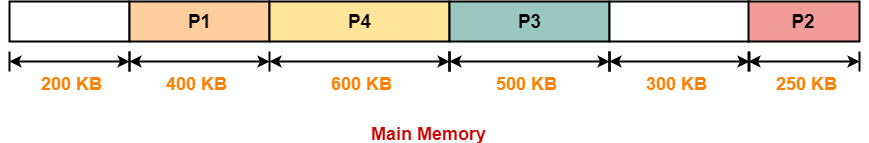
**Step-02:**



**Step-03:**



**Step-04:**



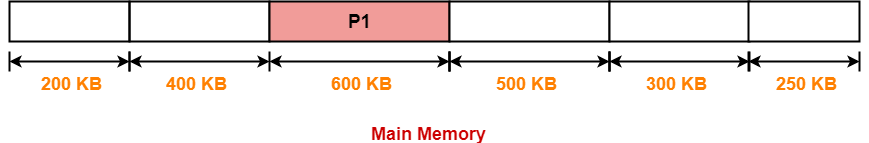
**Allocation Using Worst Fit Algorithm-**

In Worst Fit Algorithm,

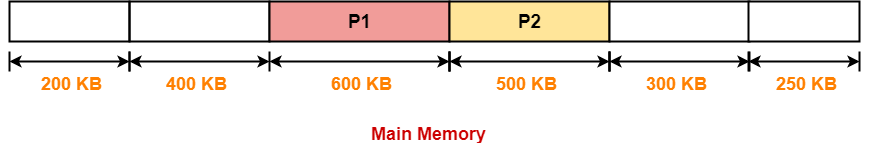
* Algorithm first scans all the partitions.
* It then allocates the partition of largest size to the process.

The allocation of partitions to the given processes is shown below-

**Step-01:**



**Step-02:**



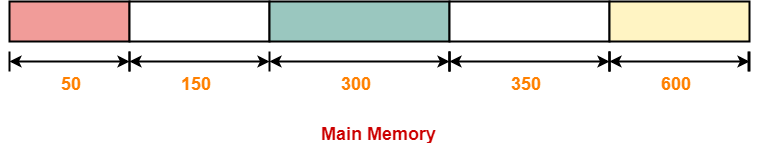
**Step-03:**

* Process P3 and Process P4 can not be allocated the memory.
* This is because no partition of size greater than or equal to the size of process P3 and process P4 is available.

To watch video solution, click [**here**](https://youtu.be/YcX-awpW9yc?list=PLmXKhU9FNesSFvj6gASuWmQd23Ul5omtD).

**Problem-02:**

Consider the following heap (figure) in which blank regions are not in use and hatched regions are in use-



The sequence of requests for blocks of size 300, 25, 125, 50 can be satisfied if we use-

1. Either first fit or best fit policy (any one)
2. First fit but not best fit policy
3. Best fit but not first fit policy
4. None of the above

**Solution-**

The allocation follows variable size partitioning scheme.

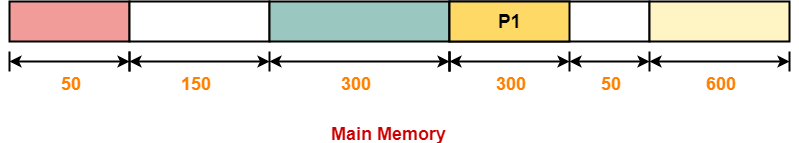
Let us say the given processes are-

* Process P1 = 300 units
* Process P2 = 25 units
* Process P3 = 125 units
* Process P4 = 50 units

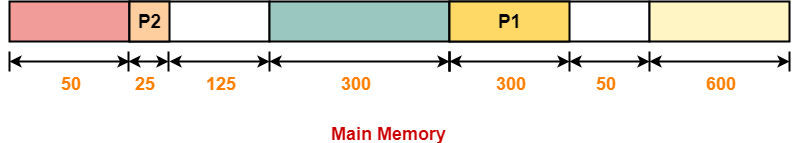
**Allocation Using First Fit Algorithm-**

The allocation of partitions to the given processes is shown below-

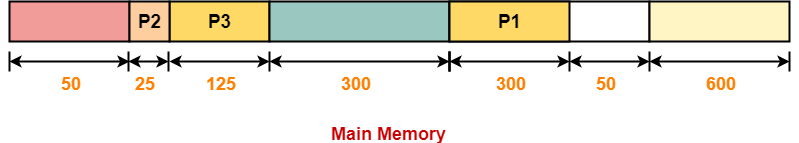
**Step-01:**



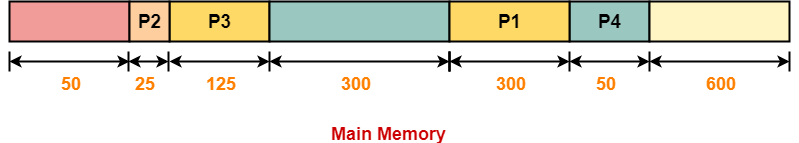
**Step-02:**



**Step-03:**



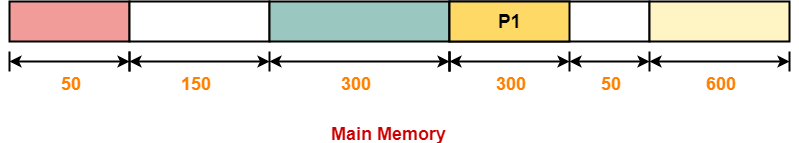
**Step-04:**



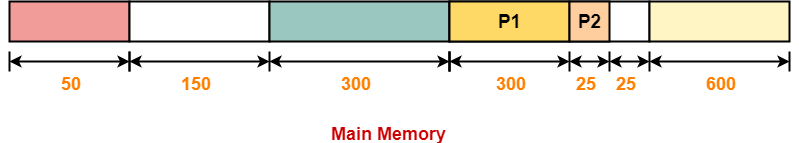
**Allocation Using Best Fit Algorithm-**

The allocation of partitions to the given processes is shown below-

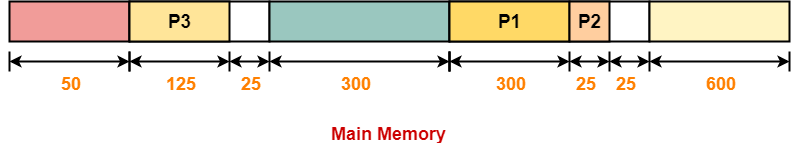
**Step-01:**



**Step-02:**



**Step-03:**



**Step-04:**

* Process P4 can not be allocated the memory.
* This is because no partition of size greater than or equal to the size of process P4 is available.

Thus,

* Only first fit allocation policy succeeds in allocating memory to all the processes.
* Option (B) is correct.

## ****Paging in OS****

* Paging is a non-contiguous memory allocation technique.
* [**Page Table**](https://www.gatevidyalay.com/page-table-paging-in-operating-system/) is a data structure that performs the mapping of page number to the frame number.

**Important Formulas-**

The following list of formulas is very useful for solving the numerical problems based on paging.

**For Main Memory-**

* Physical Address Space = Size of main memory
* Size of main memory = Total number of frames x Page size
* Frame size = Page size
* If number of frames in main memory = 2X, then number of bits in frame number = X bits
* If Page size = 2X Bytes, then number of bits in page offset = X bits
* If size of main memory = 2X Bytes, then number of bits in physical address = X bits

**For Process-**

* Virtual Address Space = Size of process
* Number of pages the process is divided = Process size / Page size
* If process size = 2X bytes, then number of bits in virtual address space = X bits

**For Page Table-**

* Size of page table = Number of entries in page table x Page table entry size
* Number of entries in pages table = Number of pages the process is divided
* Page table entry size = Number of bits in frame number + Number of bits used for optional fields if any

**NOTE-**

* In general, if the given address consists of ‘n’ bits, then using ‘n’ bits, 2n locations are possible.
* Then, size of memory = 2n x Size of one location.
* If the memory is byte-addressable, then size of one location = 1 byte.
* Thus, size of memory = 2n bytes.
* If the memory is word-addressable where 1 word = m bytes, then size of one location = m bytes.
* Thus, size of memory = 2n x m bytes.

**PRACTICE PROBLEMS BASED ON PAGING AND PAGE TABLE-**

**Problem-01:**

Calculate the size of memory if its address consists of 22 bits and the memory is 2-byte addressable.

**Solution-**

We have-

* Number of locations possible with 22 bits = 222 locations
* It is given that the size of one location = 2 bytes

Thus, Size of memory

= 222 x 2 bytes

= 223 bytes

= 8 MB

**Problem-02:**

Calculate the number of bits required in the address for memory having size of 16 GB. Assume the memory is 4-byte addressable.

**Solution-**

Let ‘n’ number of bits are required. Then, Size of memory = 2n x 4 bytes.

Since, the given memory has size of 16 GB, so we have-

2n x 4 bytes = 16 GB

2n x 4 = 16 G

2n x 22 = 234

2n = 232

∴ n = 32 bits

**Problem-03:**

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 \_\_\_\_\_.

1. 2
2. 4
3. 8
4. 16

**Solution-**

Given-

* Number of bits in logical address = 32 bits
* Page size = 4KB
* Page table entry size = 4 bytes

**Process Size-**

Number of bits in logical address = 32 bits

Thus,

Process size

= 232 B

= 4 GB

**Number of Entries in Page Table-**

Number of pages the process is divided

= Process size / Page size

= 4 GB / 4 KB

= 220 pages

Thus,

Number of entries in page table = 220 entries

**Page Table Size-**

Page table size

= Number of entries in page table x Page table entry size

= 220 x 4 bytes

= 4 MB

Thus, Option (B) is correct.

**Problem-04:**

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

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

**Solution-**

Given-

* Size of main memory = 64 MB
* Number of bits in virtual address space = 32 bits
* Page size = 4 KB

We will consider that the memory is byte addressable.

**Number of Bits in Physical Address-**

Size of main memory

= 64 MB

= 226 B

Thus, Number of bits in physical address = 26 bits

**Number of Frames in Main Memory-**

Number of frames in main memory

= Size of main memory / Frame size

= 64 MB / 4 KB

= 226 B / 212 B

= 214

Thus, Number of bits in frame number = 14 bits

**Number of Bits in Page Offset-**

We have,

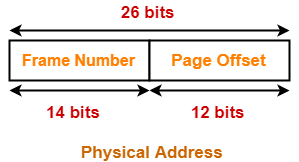
Page size

= 4 KB

= 212 B

Thus, Number of bits in page offset = 12 bits

So, Physical address is-



**Process Size-**

Number of bits in virtual address space = 32 bits

Thus,

Process size

= 232 B

= 4 GB

**Number of Entries in Page Table-**

Number of pages the process is divided

= Process size / Page size

= 4 GB / 4 KB

= 220 pages

Thus, Number of entries in page table = 220 entries

**Page Table Size-**

Page table size

= Number of entries in page table x Page table entry size

= Number of entries in page table x Number of bits in frame number

= 220 x 14 bits

= 220 x 16 bits      (Approximating 14 bits ≈ 16 bits)

= 220 x 2 bytes

= 2 MB

Thus, Option (C) is correct.

**Problem-05:**

In a virtual memory system, size of virtual address is 32-bit, size of physical address is 30-bit, page size is 4 Kbyte and size of each page table entry is 32-bit. The main memory is byte addressable. Which one of the following is the maximum number of bits that can be used for storing protection and other information in each page table entry?

1. 2
2. 10
3. 12
4. 14

**Solution-**

Given-

* Number of bits in virtual address = 32 bits
* Number of bits in physical address = 30 bits
* Page size = 4 KB
* Page table entry size = 32 bits

**Size of Main Memory-**

Number of bits in physical address = 30 bits

Thus,

Size of main memory

= 230 B

= 1 GB

**Number of Frames in Main Memory-**

Number of frames in main memory

= Size of main memory / Frame size

= 1 GB / 4 KB

= 230 B / 212 B

= 218

Thus, Number of bits in frame number = 18 bits

**Number of Bits used for Storing other Information-**

Maximum number of bits that can be used for storing protection and other information

= Page table entry size – Number of bits in frame number

= 32 bits – 18 bits

= 14 bits

Thus, Option (D) is correct.

**Overhead in Paging-**

In paging scheme, there are mainly two overheads-

**1. Overhead of Page Tables-**

* Paging requires each process to maintain a page table.
* So, there is an overhead of maintaining a page table for each process.

**2. Overhead of Wasting Pages-**

* There is an overhead of wasting last page of each process if it is not completely filled.
* On an average, half page is wasted for each process.

Thus,

Total overhead for one process

= Size of its page table + (Page size / 2)

**Optimal Page Size-**

Optimal page size is the page size that minimizes the total overhead.

It is given as-

https://www.gatevidyalay.com/wp-content/uploads/2018/11/Optimal-Page-Size.png

**Proof-**

Total overhead due to one process

= Size of its page table + (Page size / 2)

= Number of entries x Page table entry size + (Page size / 2)

= Number of pages the process is divided x Page table entry size + (Page size / 2)

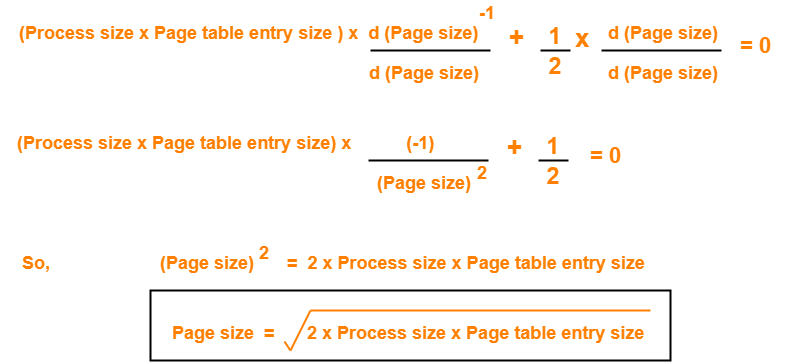
= (Process size / Page size) x Page table entry size + (Page size / 2)

Now,

For minimum overhead,

https://www.gatevidyalay.com/wp-content/uploads/2018/11/Optimal-Page-Size-Proof-Diagram-01.png

Keeping process size and page table entry size as constant, differentiating overhead with respect to page size, we get-



This page size minimizes the total overhead.

**PRACTICE PROBLEMS BASED ON OPTIMAL PAGE SIZE-**

**Problem-01:**

In a paging scheme, virtual address space is 4 KB and page table entry size is 8 bytes. What should be the optimal page size?

**Solution-**

Given-

* Virtual address space = Process size = 4 KB
* Page table entry size = 8 bytes

We know-

Optimal page size

= (2 x Process size x Page table entry size)1/2

= (2 x 4 KB x 8 bytes)1/2

= (216 bytes x bytes)1/2

= 28 bytes

= 256 bytes

Thus, Optimal page size = 256 bytes.

**Problem-02:**

In a paging scheme, virtual address space is 16 MB and page table entry size is 2 bytes. What should be the optimal page size?

**Solution-**

Given-

* Virtual address space = Process size = 16 MB
* Page table entry size = 2 bytes

We know-

Optimal page size

= (2 x Process size x Page table entry size)1/2

= (2 x 16 MB x 2 bytes)1/2

= (226 bytes x bytes)1/2

= 213 bytes

= 8 KB

Thus, Optimal page size = 8 KB.

**Problem-03:**

In a paging scheme, virtual address space is 256 GB and page table entry size is 32 bytes. What should be the optimal page size?

**Solution-**

Given-

* Virtual address space = Process size = 256 GB
* Page table entry size = 32 bytes

We know-

Optimal page size

= (2 x Process size x Page table entry size)1/2

= (2 x 256 GB x 32 bytes)1/2

= (244 bytes x bytes)1/2

= 222 bytes

= 4 MB

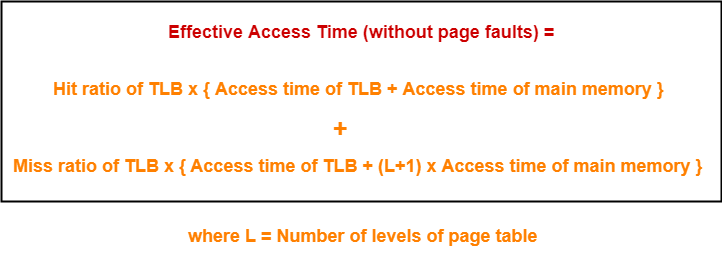
Thus, Optimal page size = 4 MB.

We have discussed-

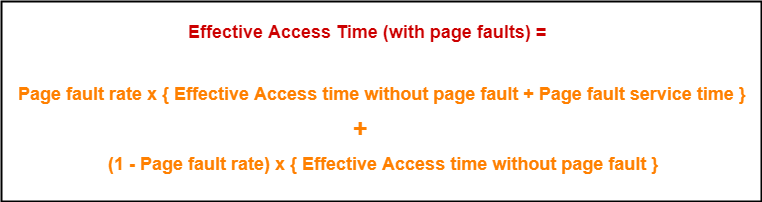
* A page fault occurs when the referenced page is not found in the main memory.
* Page fault handling routine is executed on the occurrence of page fault.
* The time taken to service the page fault is called as **page fault service time**.

**Effective Access time-**

In a multilevel paging scheme using TLB without any possibility of page fault, effective access time is given by-



In a multilevel paging scheme using TLB with a possibility of page fault, effective access time is given by-



**Also Read-** [**Page Replacement Algorithms**](https://www.gatevidyalay.com/page-replacement-algorithms-page-fault/)

**PRACTICE PROBLEMS BASED ON PAGE FAULTS IN OS-**

**Problem-01:**

Let the page fault service time be 10 ms in a computer with average memory access time being 20 ns. If one page fault is generated for every 106 memory accesses, what is the effective access time for the memory?

1. 21 ns
2. 30 ns
3. 23 ns
4. 35 ns

**Solution-**

Given-

* Page fault service time = 10 ms
* Average memory access time = 20 ns
* One page fault occurs for every 106 memory accesses

**Page Fault Rate-**

It is given that one page fault occurs for every 106 memory accesses.

Thus,

Page fault rate

= 1 / 106

= 10-6

**Effective Access Time With Page Fault-**

It is given that effective memory access time without page fault = 20 ns.

Now, substituting values in the above formula, we get-

Effective access time with page fault

= 10-6 x { 20 ns + 10 ms } + ( 1 – 10-6 ) x { 20 ns }

= 10-6 x 10 ms + 20 ns

= 10-5 ms + 20 ns

= 10 ns + 20 ns

= 30 ns

Thus, Option (B) is correct.

**Problem-02:**

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-

1. 1.9999 milliseconds
2. 1 millisecond
3. 9.999 microseconds
4. 1.9999 microseconds
5. None of these

**Solution-**

Given-

* Page fault service time = 10 msec
* Average memory access time = 1 μsec
* Hit ratio = 99.99% = 0.9999

**Page Fault Rate-**

Page fault rate

= 1 – Hit ratio

= 1 – 0.9999

= 0.0001

**Effective Access Time With Page Fault-**

It is given that effective memory access time without page fault = 1 μsec.

Substituting values in the above formula, we get-

Effective access time with page fault

= 0.0001 x { 1 μsec + 10 msec } + 0.99999 x 1 μsec

= 0.0001 μsec + 0.001 msec + 0.9999 μsec

= 1 μsec + 0.001 msec

= 1 μsec + 1 μsec

= 2 μsec or 0.002 msec

Thus, Option (E) is correct.

**Problem-03:**

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-

1. i + j / k
2. i + j x k
3. (i + j) / k
4. (i + j) x k

**Solution-**

Given-

* Page fault service time = j μsec
* Average memory access time = i μsec
* One page fault occurs every k instruction

**Page Fault Rate-**

It is given that one page fault occurs every k instruction.

Thus,

Page fault rate

= 1 / k

**Effective Access Time With Page Fault-**

It is given that effective memory access time without page fault = i μsec

Now, substituting values in the above formula, we get-

Effective access time with page fault

= (1 / k) x { i μsec + j μsec } + ( 1 – 1 / k) x { i μsec }

= j / k μsec + i μsec

= i + j / k μsec

Thus, Option (A) is correct.

**Problem-04:**

Consider a system with a two-level paging scheme in which a regular memory access takes 150 nanoseconds and servicing a page fault takes 8 milliseconds. An average instruction takes 100 nanoseconds of CPU time and two memory accesses. The TLB hit ratio is 90% and the page fault rate is one in every 10,000 instructions. What is the effective average instruction execution time?

1. 645 nanoseconds
2. 1050 nanoseconds
3. 1215 nanoseconds
4. 1230 nanoseconds
5. None of these

**Solution-**

Given-

* Number of levels of page table = 2
* Main memory access time = 150 ns
* Page fault service time = 8 msec
* Average instruction takes 100 ns of CPU time and 2 memory accesses
* TLB Hit ratio = 90% = 0.9
* Page fault rate = 1 / 104 = 10-4

Assume TLB access time = 0 since it is not given in the question.

Also, TLB access time is much less as compared to the memory access time.

**Effective Access Time Without Page Fault-**

Substituting values in the above formula, we get-

Effective memory access time without page fault

= 0.9 x { 0 + 150 ns } + 0.1 x { 0 + (2+1) x 150 ns }

= { 0.9 x 150 ns } + { 0.1 x 450 ns }

= 135 ns + 45 ns

= 180 ns

**Effective Access Time With Page Fault-**

Substituting values in the above formula, we get-

Effective access time with page fault

= 10-4 x { 180 ns + 8 msec } + (1 – 10-4) x 180 ns

= 8 x 10-4 msec + 180 ns

= 8 x 10-7 sec + 180 ns

= 800 ns + 180 ns

= 980 ns

**Effective Average Instruction Execution Time-**

Effective Average Instruction Execution Time

= 100 ns + 2 x Effective memory access time with page fault

= 100 ns + 2 x 980 ns

= 100 ns + 1960 ns

= 2060 ns

Thus, Option (E) is correct.

**Problem-05:**

A demand paging system takes 100 time units to service a page fault and 300 time units to replace a dirty page. Memory access time is 1 time unit. The probability of a page fault is p. In case of a page fault, the probability of page being dirty is also p. It is observed that the average access time is 3 time units. Then the value of p is-

1. 0.194
2. 0.233
3. 0.514
4. 0.981
5. None of these

**Solution-**

Given-

* Page fault service time = 100 time units
* Time taken to replace dirty page = 300 time units
* Average memory access time = 1 time unit
* Page fault rate = p
* Probability of page being dirty = p
* Effective access time = 3 time units

Now, According to question-

3 time units = p x { 1 time unit + p x { 300 time units } + (1 – p) x { 100 time units } } + (1 – p) x { 1 time unit }

3 = p x { 1 + 300p + 100 – 100p } + (1 – p)

3 = p x { 101 + 200p } + (1 – p)

3 = 101p + 200p2 + 1 – p

3 = 100p + 200p2 + 1

200p2 + 100p – 2 = 0

On solving this quadratic equation, we get p = 0.019258

Thus, Option (E) is correct.