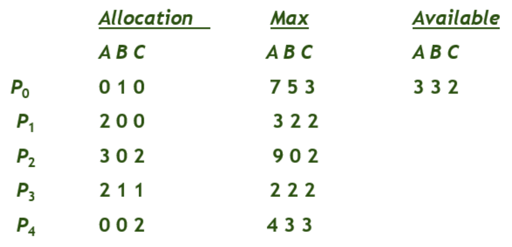
Scheme of Evaluation

Internal Assessment Test 1 – March 2025

Sub:	**Operating Systems**							**Sub Code:**	**MMC104**
Date:	20-03-25	Duration:	90 mins	Max Marks:	50	**Sem:**	I	**Branch:**	**MCA**

Q.NO	**Description**	**Marks Distribution**	**Max Marks**
1	• **Explain TLB with neat diagram** • Explanation of TLB • Draw a TLB Architecture diagram	7 3	10
2	**Mention the different page table structure. Explain it brief** • List out the different page table structure • Explanation of each page table structure with neat diagrams	2 8	10
3	**Describe about the Page replacement algorithm in detail notes** • Define Page Replacement Algorithm • List out the Page Replacement Algorithm and also explain it brief • Give example of each Page Replacement Algorithm	2 6 2	10
4	**Consider the following reference string:** **6 1 1 2 0 3 4 6 0 2 1 2 1 2 0 3 2 1 4 0** **How many page faults would occur in case of i)FIFO ii) LRU iii)Optimal algorithm?** • Find out the solutions of page fault using FIFO,LRU and Optimal page replacement algorithm	10	10
5	**What is a deadlock? What are the necessary conditions for a deadlock to occur?** • Definition of deadlock • Give explain about the necessary conditions of deadlock •	2 8	10
6	**. With neat diagrams explain Resource Allocation graph.** • Explain concepts of RAG • Draw the diagram of RAG	6 4	10
7	**Write and explain about the Banker's algorithm with an example** • Explanation of Banker's Algorithm	10	10

8	Consider the following  Answer the following question: i) Construct a need matrix. ii) Is the system in a safe state? If yes what is the safe sequence? iii) If process P1 makes a request (1 0 2) can the request be granted? • Find out the solution of need matrix • Find out the solution of safe sequence	**4** **6**	**10**
9	**What are the various methods used to access file?** • List out the methods used to access file • Explain about the three methods in detail	**2** **8**	**10**
10	**Explain the VFS with a Schematic diagram** • Explanation of the VFS • Draw the VFS Schematic diagram	**7** **3**	**10**

	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	

Internal Assessment Test 2 – March 2025

Operating Systems							Sub Code:	MMC104
20/03/2025	**Duration:**	**90 min's**	**Max Marks:**	**50**	**Sem:**	**I**	**Branch:**	**MCA**

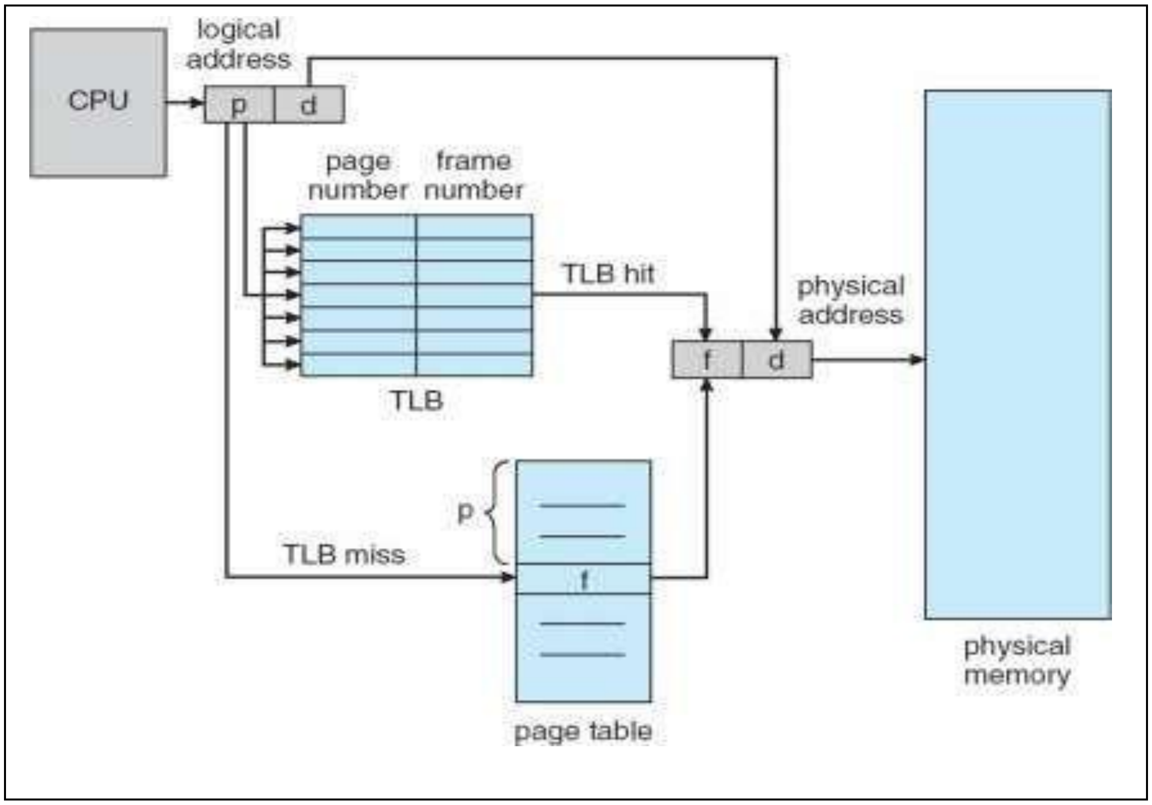
PART I

1. Explain TLB with neat diagram

The hardware implementation of the page table can be done by using dedicated registers. But the usage of the register for the page table is satisfactory only if the page table is small.

If the page table contains a large number of entries then we can use TLB (translation Look-aside buffer), a special, small, fast look-up hardware cache.

- The TLB is an associative, high-speed memory.
- Each entry in TLB consists of two parts: a tag and a value.
- When this memory is used, then an item is compared with all tags simultaneously. If the item is found, then the corresponding value is returned.



When a logical address is generated by the CPU, its page number is presented to the TLB. If the page number is found, its frame number is immediately available and is used to access memory.

If the page number is not in the TLB (known as a TLB miss), a memory reference to the page table must be made. When the frame number is obtained, we can use it to access memory. In addition, we add the page number and frame number to the TLB, so that they will be found quickly on the next reference. If the TLB is already full of entries, the operating system must select one for replacement. Replacement policies range from least recently used (LRU) to random. Furthermore, some TLBs allow entries to be wired down, meaning that they cannot be removed from the TLB. Typically, TLB entries for kernel code are often wired down.

The percentage of times that a particular page number is found in the TLB is called the **hit ratio**. An 80-percent hit ratio means that we find the desired page number in the TLB 80 percent of the time. If it takes 20 nanoseconds to search the TLB, and 100 nanoseconds to access memory, then a mapped memory access takes 120 nanoseconds when the page number is in the TLB. If we fail to find the page number in the TLB (20 nanoseconds), then we must first access memory for the page table and frame number (100 nanoseconds), and then access the desired byte in memory (100 nanoseconds), for a total of 220 nanoseconds.

To find the effective memory-access time, we must weigh each case by its probability: (Where P is Hit ratio)

EAT(effective access time) = P x hit memory time + (1-P) x miss memory time.

= 0.80 x 120 + 0.20 x 220

= 140 nanoseconds.

In this example, we suffer a 40-percent slowdown in memory access time (from 100 to 140 ns).

For a 98-percent hit ratio, we have

EAT(effective access time)= P x hit memory time + (1-P) x miss memory time.

= 0.98 x 120 + 0.02 x 220

= 122 nanoseconds.

This increased hit rate produces only a 22-percent slowdown in access time.

Example:

What will be the EAT if hit ratio is 70%, time for TLB is 30ns and access to main memory is 90ns?

P = 70% = 70/100 = 0.7

Hit memory time = 30ns + 90ns = 120ns

Miss memory time = 30ns + 90ns + 90ns = 210ns Therefore,

EAT = P x Hit + (1-P) x Miss

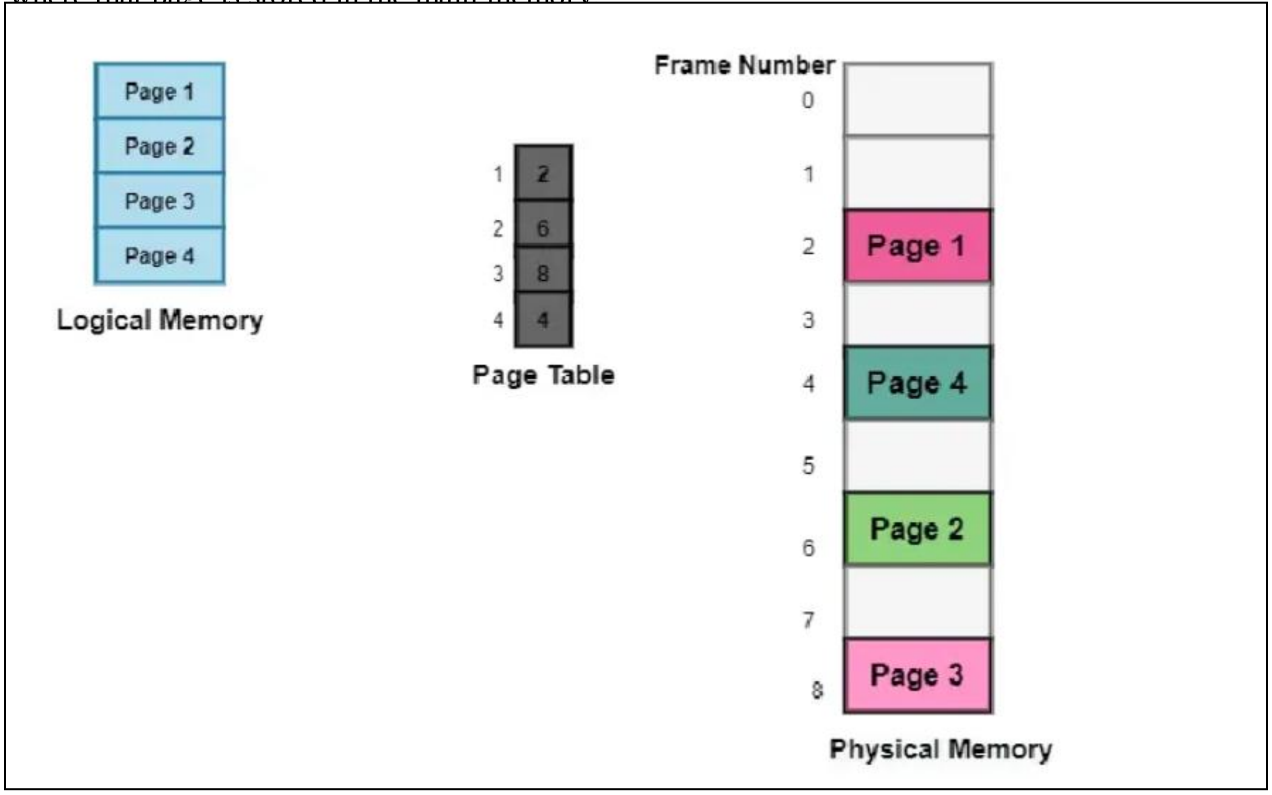
= 0.7 x 120 + 0.3 x 210

=840 + 63.0

=147 ns

2. Mention the different page table structure. Explain it brief

Structure of Page Table in Operating Systems
The data structure that is used by the virtual memory system in the operating system of a computer in order to store the mapping between physical and logical addresses is commonly known as Page Table. As we had already told you that the logical address that is generated by the CPU is translated into the physical address with the help of the page table.
Thus page table mainly provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.



- **PTBR** means page table base register and it is basically used to hold the base address for the page table of the current process.
- Each process has its own independent page table.

Techniques used for Structuring the Page Table
Some of the common techniques that are used for structuring the Page table are as follows:
1. Hierarchical Paging
2. Hashed Page Tables
3. Inverted Page Tables
Let us cover these techniques one by one;

Multilevel or Hierarchical Page Table

• Page tables can consume a significant amount of memory.

• For example: A 32-bit virtual address space using 4 kB pages.

Page size = 2^{12} bytes

Space for page numbers = 2^{32} - 2^{12}

= 2^{20} bytes

• So page table may consists of upto 1 million entries, one for each page, for a total address capacity of about four billion bytes.

• Hierarchical page table is also called as forward mapped page table. This approach is so effective that many modern operating systems employ it.

• In this method, each level containing a table that stores pointers to tables in the level below. Each table in the hierarchy is the size of one page. It enables the operating system to transfer page tables between main memory and secondary storage device easily.

For two levels of page table

Virtual address contains page number and displacement into that page.

Virtual address (v) = (p, t, d)

Where (p, t)=Page number

d = Displacement

• Here p is an index into the outer page table.

• Fig. 4.6.1 shows hierarchical page table.

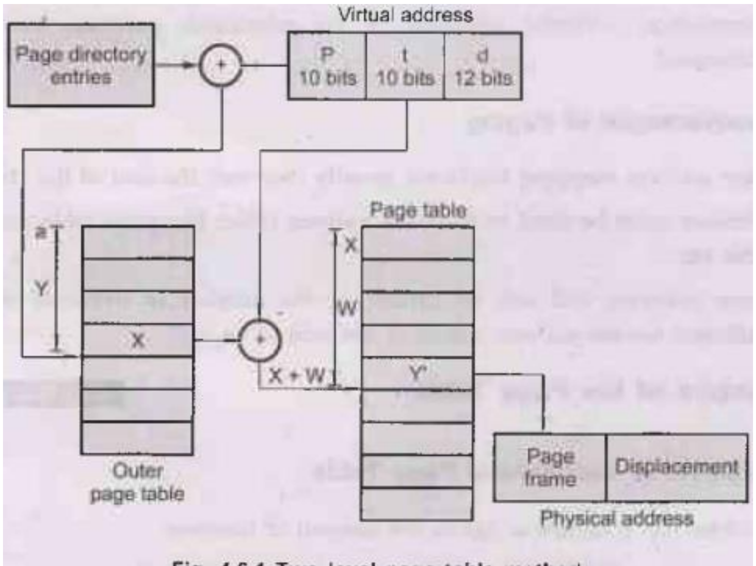


Fig. 4.6.1 Two level page table method

• Intel IA - 32 architecture support two levels of page

Advantages

1. It is compact and support sparse address space.

2. It easily manages the memory.

3. It reduces table fragmentation.

Disadvantages

1. Overhead due to one more page table.

2. On TLB miss, two loads from memory will be required to get the right address translation information from the page table.

Hashed Page Tables

• Hashed page table support address space of 32 bits and more. Hash value is used as virtual page number.

• Hash table contains a link list of elements. Each element consists of following fields:

1. Virtual page number

2. Mapped page frame value.

3. Pointer to the next element. Hash table improves search speed.

Fig. 4.6.2 shows block diagram of hashed

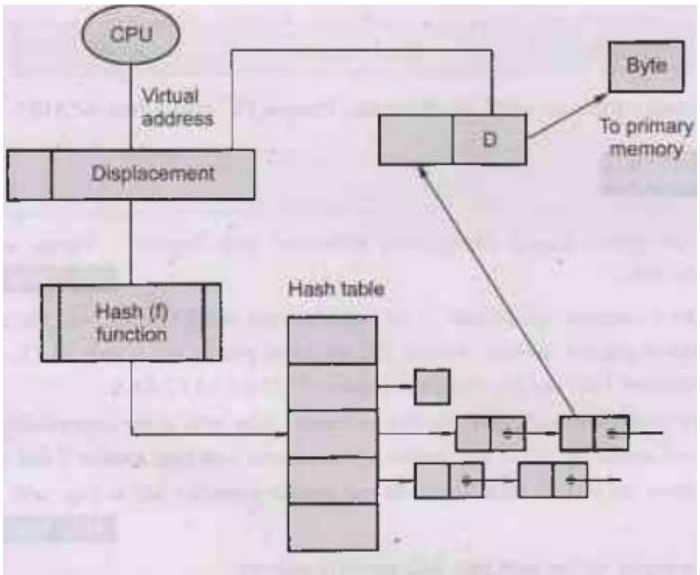


Fig. 4.6.2 Hashed page table

Working

1. Virtual page number is taken from virtual address.

2. Virtual page number is hashed into the hash table.

3. Virtual page number is compared with the first element of linked list.

4. Both the value is matched, that value is (i.e. page frame) used for calculating physical address.

5. If the both value is not matched, the entire linked list is searched for a matching.

• Clustered page table is same as hashed page table but only difference is that each entry in the hash table refers to several pages.

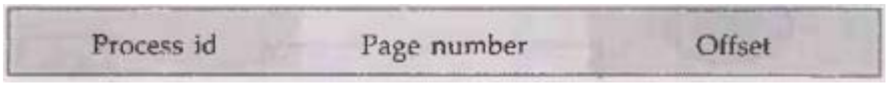
Inverted Page Table

• It uses a page table that contains an entry for each physical frame. This ensures that the page table occupies a fixed fraction of memory. The size is proportional to the physical memory.

• Page table overhead increases with address space size. Page table get too big to fit in memory.

• Inverted page table has one entry for each real page of memory. Lookup time is increased because it requires a search on the inverted table.

• Logical address is as follows



Process id	Page number	Offset

• Inverted page table is used by Itanium, Power PC and Ultra SPARC.

PART II

3. Describe about the Page replacement algorithm in detail notes

Page Replacement technique uses the following approach. If there is no free frame, then we will find the one that is not currently being used and then free it. A-frame can be freed by writing its content to swap space and then change the page table in order to indicate that the page is no longer in the memory.

1. First of all, find the location of the desired page on the disk.

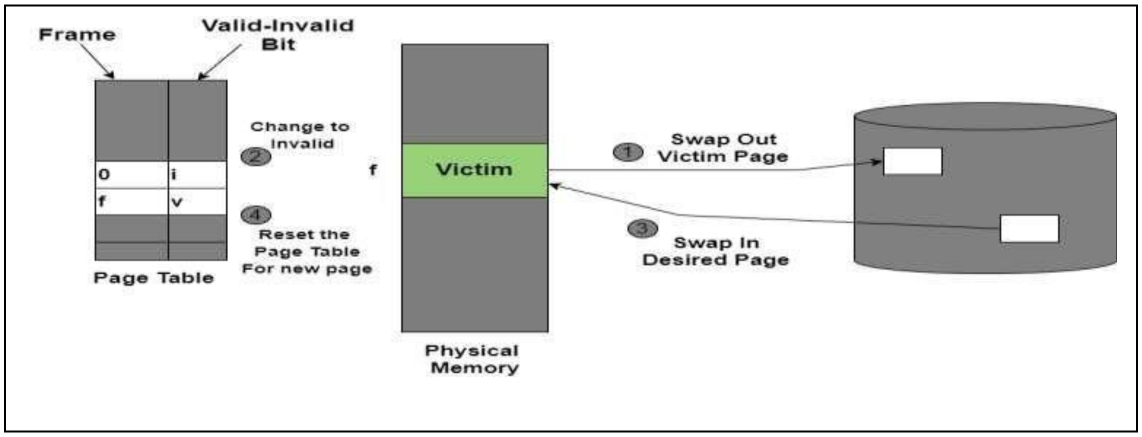
2. Find a free Frame:

a) If there is a free frame, then use it.

b) If there is no free frame then make use of the page-replacement algorithm in order to select the victim frame.

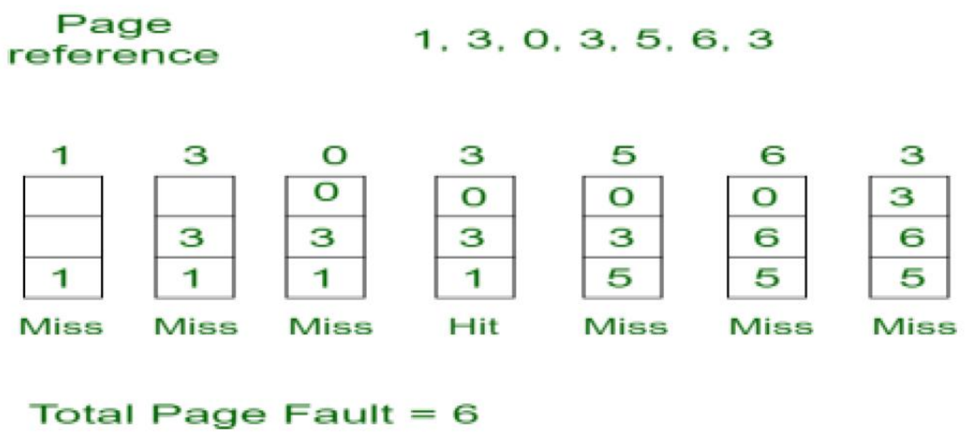
c) Then after that write the victim frame to the disk and then make the changes in the page table and frame table accordingly.
3. After that read the desired page into the newly freed frame and then change the page and frame tables.

4. Restart the process.

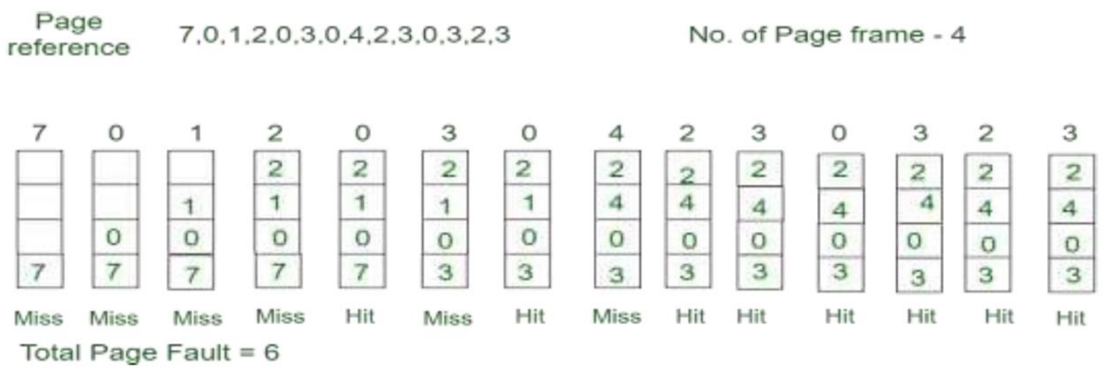


PAGE REPLACEMENT ALGORITHM
In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.
A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page faults happen.
In case of a page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace.
The target for all algorithms is to reduce the number of page faults.
(i) First-In-First-Out (FIFO) Page Replacement Algorithm:
This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.
Example 1:
Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3 page frames. Find the number of page faults.

Initially, all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.** When 3 comes, it is already in memory so **No Page Faults.** Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. When 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 and its cause **Page Fault.** Finally, when 3 come it is not available so it replaces 0 ie **page fault.**



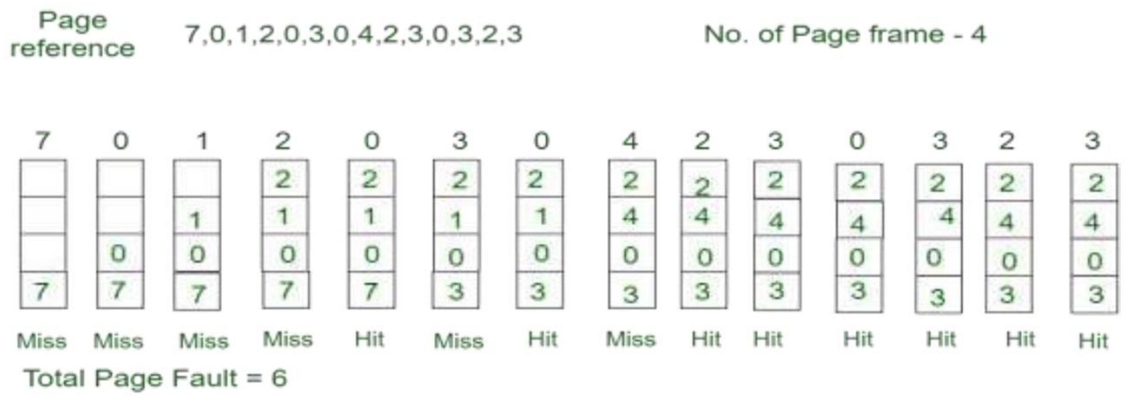
(ii) Optimal Page replacement:
In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.
Example:
Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frame.



Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots ie **4 Page faults.** 0 is already there, so **No Page fault,** when 0 came**.** When 3 came it will take the place of 7 because it is not used for the longest duration of time in the future and its cause **Page fault.** 0 is already there, so **No Page fault.** 4 will takes place of 1 and its cause

Page Fault. Now for the further page reference string, **No Page Fault** because they are already available in the memory.
Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

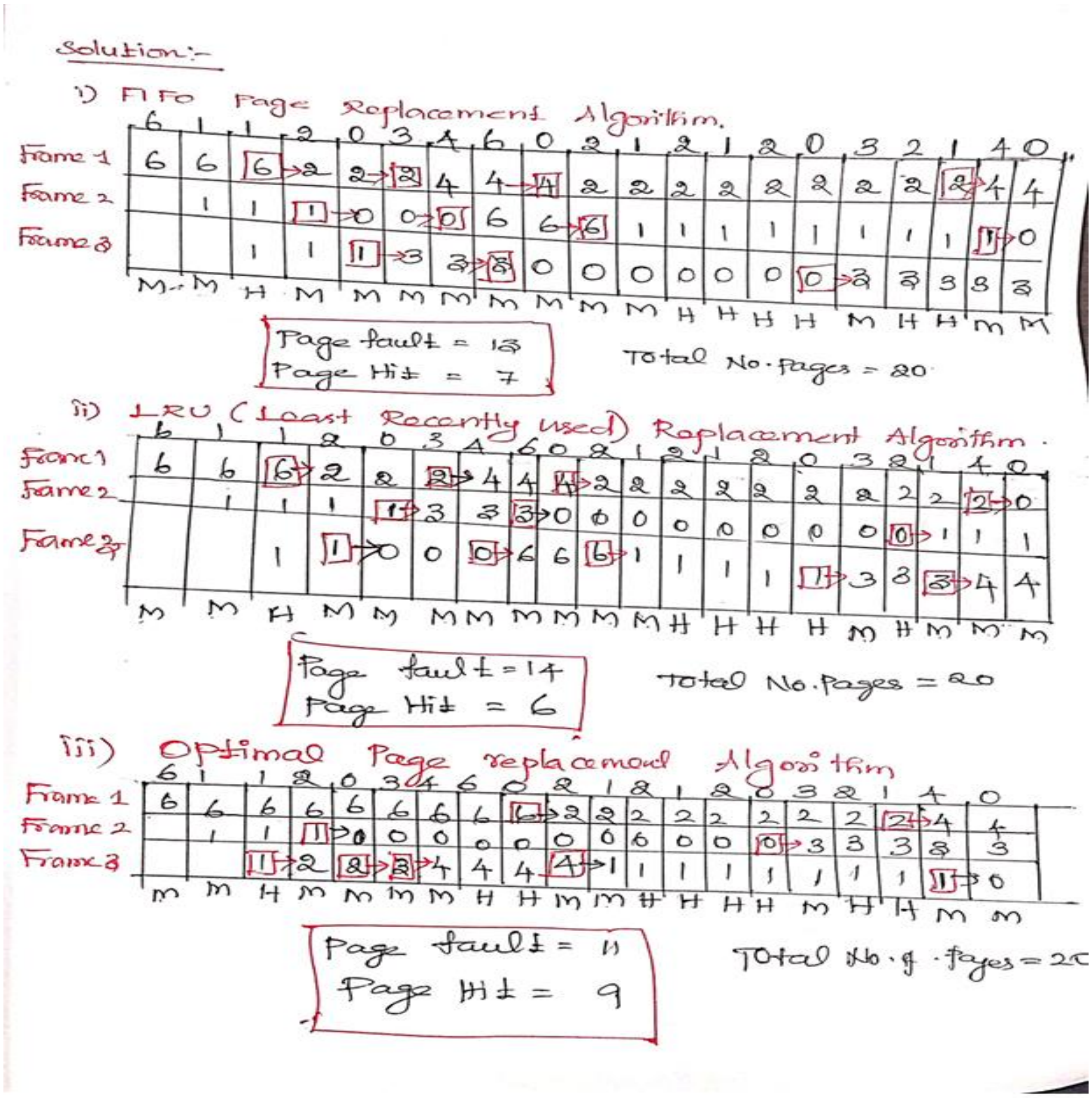
(iii) Least Recently Used:
In this algorithm, page will be replaced which is least recently used.
Example:
Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frames.



Here LRU has same number of page fault as optimal but it may differ according to question.

Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots, **4 Page faults.** 0 is already there, so **No Page fault.** When 3 came it will take the place of 7 because it is least recently used and its cause **Page fault**. 0 is already in memory, so **No Page fault**. 4 will takes place of 1 and its cause **Page Fault.** Now for the further page reference string **No Page fault** because they are already available in the memory.

4. Consider the following reference string:
6 1 1 2 0 3 4 6 0 2 1 2 1 2 0 3 2 1 4 0
How many page faults would occur in case of i) FIFO ii) LRU iii) Optimal algorithm?



Solution:-

i) FIFO Page Replacement Algorithm.

	6	1	1	2	0	3	4	6	0	2	1	2	1	2	0	3	2	1	4	0
Frame 1	6	6	$\boxed{6}\to$	2	2 $\to$	$\boxed{2}$	4	4 $\to$	$\boxed{4}$	2	2	2	2	2	2	2	2	$\boxed{2}\to$	4	4
Frame 2		1	1	$\boxed{1}\to$	0	0 $\to$	$\boxed{0}$	6	6 $\to$	$\boxed{6}$	1	1	1	1	1	1	1	1	$\boxed{1}\to$	0
Frame 3			1	1	$\boxed{1}\to$	3	3 $\to$	$\boxed{3}$	0	0	0	0	0	0	$\boxed{0}\to$	3	3	3	3	3
	M	M	H	M	M	M	M	M	M	M	M	H	H	H	H	M	H	H	M	M

Page fault = 13
Page Hit = 7

Total No. pages = 20

ii) LRU (Least Recently used) Replacement Algorithm.

	6	1	1	2	0	3	4	6	0	2	1	2	1	2	0	3	2	1	4	0
Frame 1	6	6	$\boxed{6}\to$	2	2	$\boxed{2}\to$	4	4	$\boxed{4}\to$	2	2	2	2	2	2	2	2	2	$\boxed{2}\to$	0
Frame 2		1	1	1	$\boxed{1}\to$	3	3	$\boxed{3}\to$	0	0	0	0	0	0	0	0	$\boxed{0}\to$	1	1	1
Frame 3			1	$\boxed{1}\to$	0	0	$\boxed{0}\to$	6	6	$\boxed{6}\to$	1	1	1	1	$\boxed{1}\to$	3	3	$\boxed{3}\to$	4	4
	M	M	H	M	M	M	M	M	M	M	M	H	H	H	H	M	H	M	M	M

Page fault = 14
Page Hit = 6

Total No. Pages = 20

iii) Optimal Page replacement Algorithm

	6	1	1	2	0	3	4	6	0	2	1	2	1	2	0	3	2	1	4	0
Frame 1	6	6	6	6	6	6	6	6	$\boxed{6}\to$	2	2	2	2	2	2	2	2	$\boxed{2}\to$	4	4
Frame 2		1	1	$\boxed{1}\to$	0	0	0	0	0	0	0	0	0	0	$\boxed{0}\to$	3	3	3	3	3
Frame 3			$\boxed{1}\to$	2	$\boxed{2}\to$	$\boxed{2}\to$	4	4	4	$\boxed{4}\to$	1	1	1	1	1	1	1	1	$\boxed{1}\to$	0
	M	M	H	M	M	M	M	H	H	M	M	H	H	H	H	M	H	H	M	M

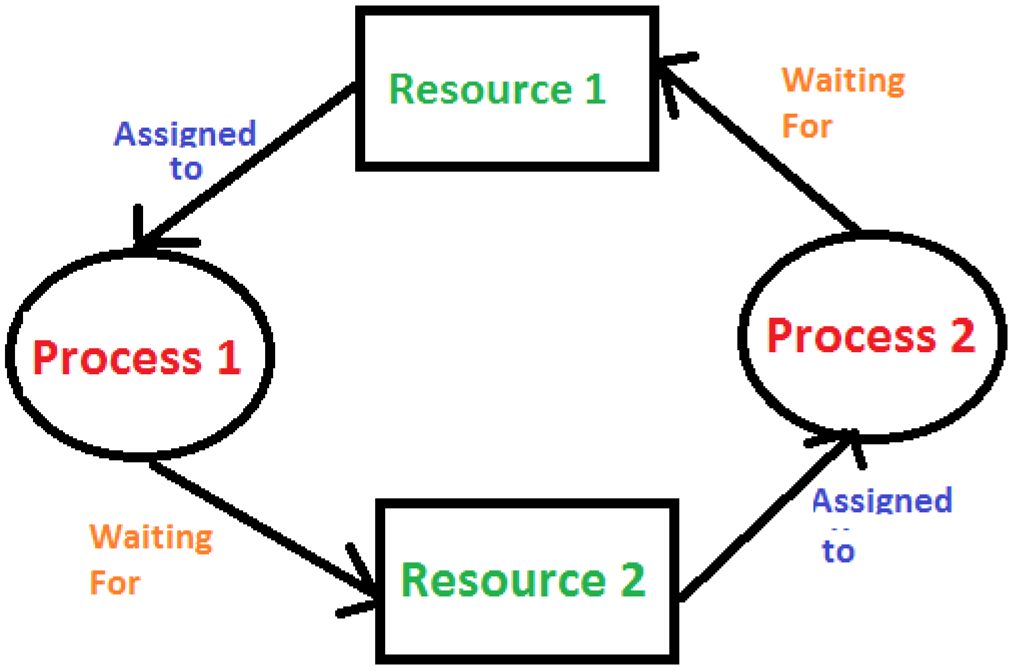
Page fault = 11
Page Hit = 9

Total No. of pages = 2

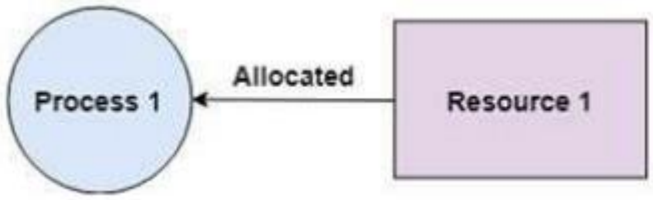
PART III

5. What is a deadlock? What are the necessary conditions for a deadlock to occur?

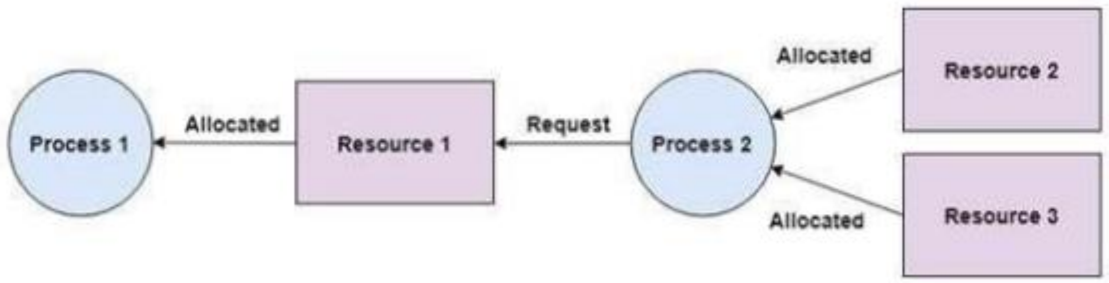
Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process. Consider an example when two trains are coming toward each other on same track and there is only one track, none of the trains can move once they are in front of each other. Similar situation occurs
in operating systems when there are two or more processes hold some resources and wait for resources held by other(s).For example, in the below diagram, Process 1 is holding Resource1 and waiting for resource2which isacquiredbyprocess2, and process2 is waiting for resource 1.



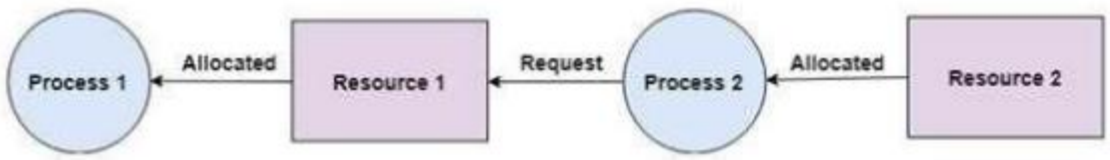
Deadlock Characterization
A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.
A deadlock occurs if the four common conditions should true. But these conditions are not mutually exclusive. They are given as follows:
Mutual Exclusion
There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



Hold and Wait
A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.

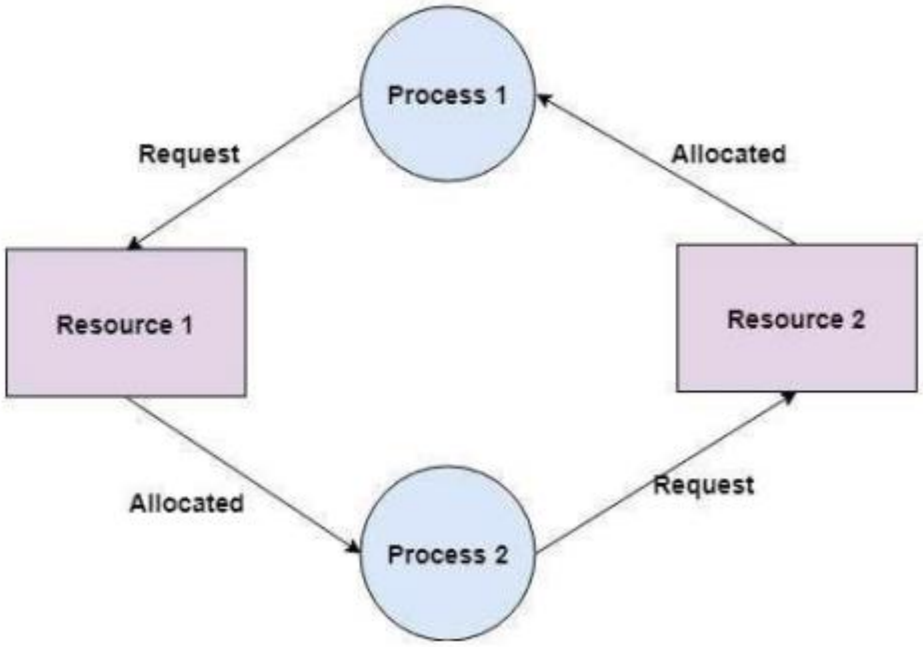


No Preemption
A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource1 from Process
1. It will only be released when Process1 relinquishes it voluntarily after its execution is complete.



Circular Wait

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.

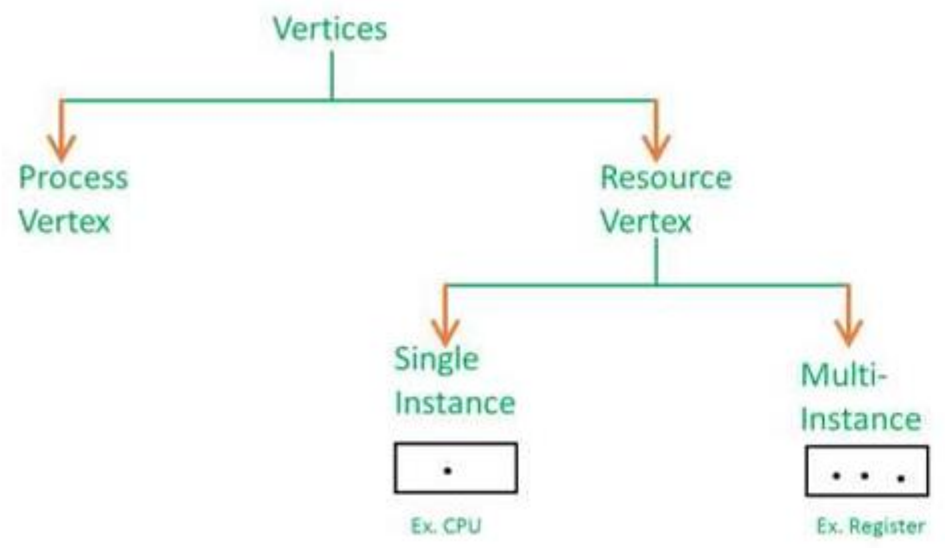


6. With neat diagrams explain Resource Allocation graph.

As Banker's algorithm using some kind of table like allocation, request, available all that thing to understand what is the state of the system. Similarly, if you want to understand the state of the system instead of using those table, actually tables are very easy to represent and understand it, but then still you could even represent the same information in the graph. That graph is called Resource
Allocation Graph (RAG).

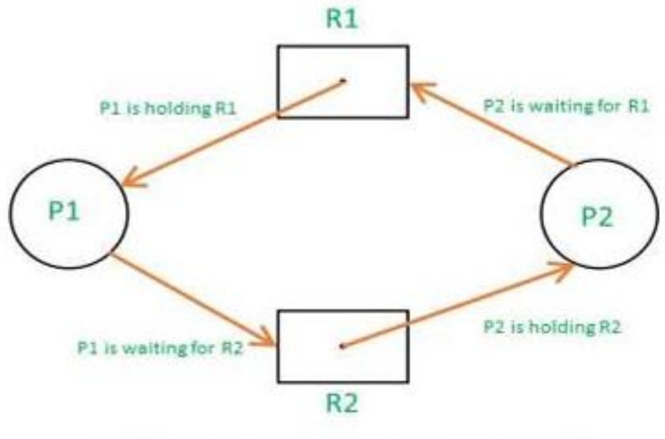
So, resource allocation graph is explained to us what is the state of the system in terms of processes and resources. Like how many resources are available, how many are allocated and

what is the request of each process. Everything can be represented in terms of the diagram. One of the advantages of having a diagram is, sometimes it is possible to see a deadlock directly by using RAG, but then you might not be able to know that by looking at the table. But the tables are better if the system contains lots of process and resource and Graph is better if the system contains less number of process and resource.
We know that any graph contains vertices and edges. So RAG also contains vertices and edges. In RAG vertices are two type:
Process vertex – Every process will be represented as a process vertex. Generally, the process will be represented with a circle.
Resource vertex - Every resource will be represented as a resource vertex.
It is also two types:
Single instance type resource: It represents as a box, inside the box, there will be one dot. So the number of dots indicate how many instances are present of each resource type.
Multi-resource instance type resource: It also represents as a box, inside the box, there will be many dots present.



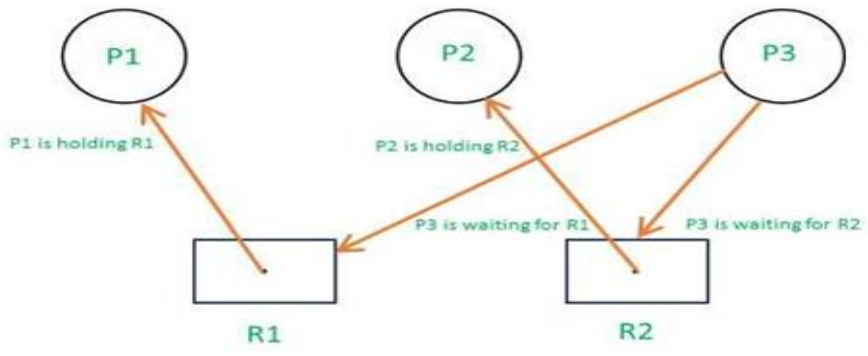
Now coming to the edges of RAG There are two types of edges in RAG:
1. **Assign Edge** – If you already assign a resource to a process then it is called Assign edge.
2. **Request Edge** – It means in future the process might want some resource to complete the execution that is called request edge.
So, if a process is using a resource, an arrow is drawn from the resource node to the process node. If a process is requesting a resource, an arrow is drawn from the process node to the resource node.

Example 1 (Single instances RAG) –



SINGLE INSTANCE RESOURCE TYPE WITH DEADLOCK

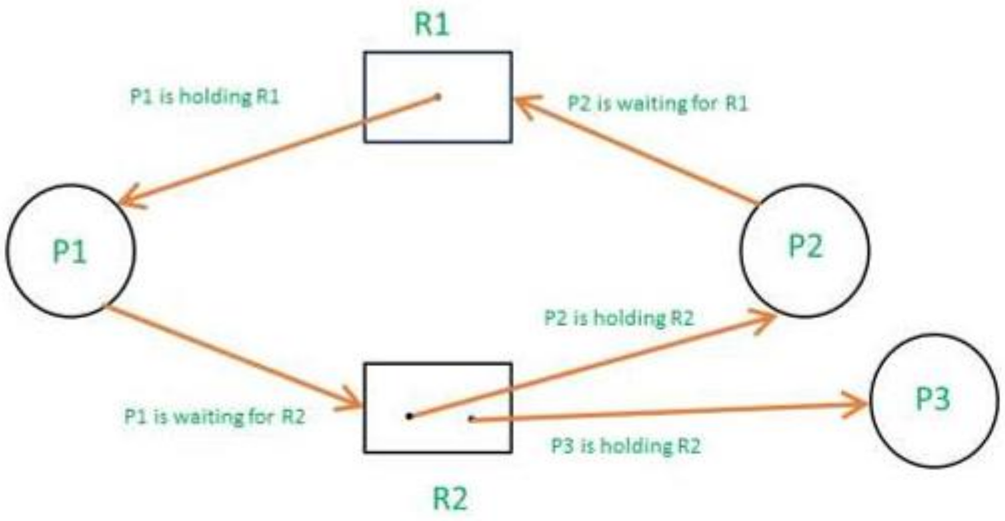
If there is a cycle in the Resource Allocation Graph and each resource in the cycle provides only one instance, then the processes will be in deadlock. For example, if process P1 holds resource R1, process P2 holds resource R2 and process P1 is waiting for R2 and process P2 is waiting for R1, then process P1 and process P2 will be in deadlock.



SINGLE INSTANCE RESOURCE TYPE WITHOUT DEADLOCK

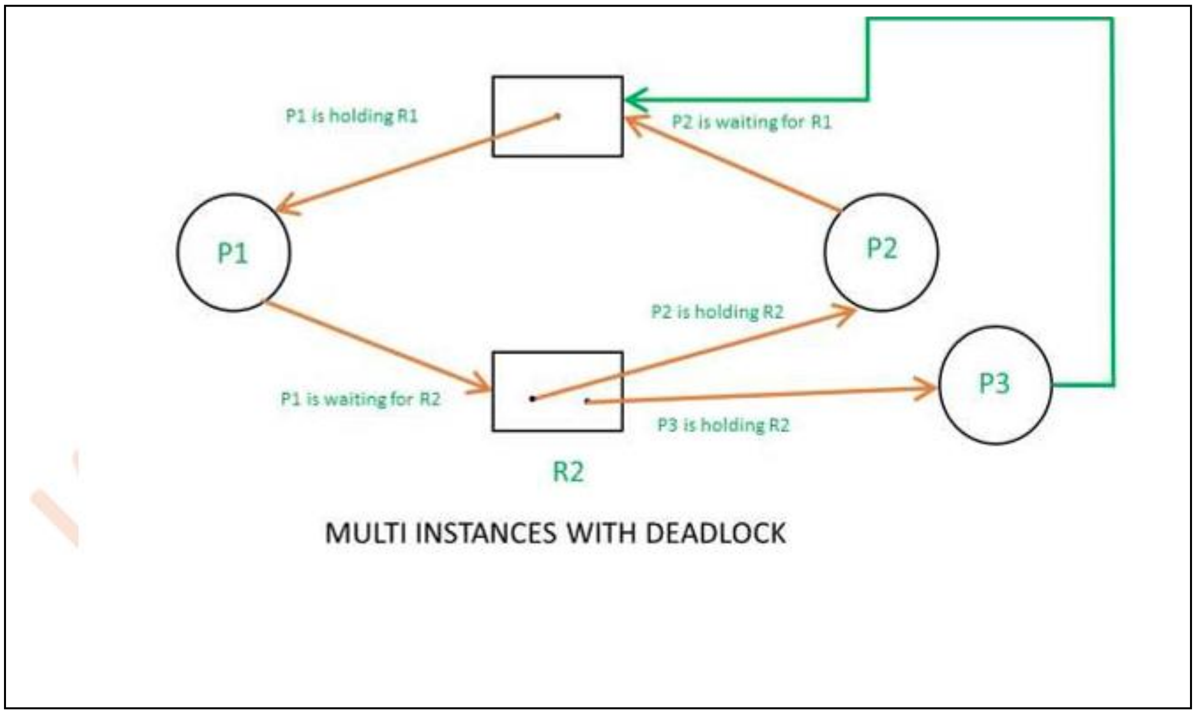
Here’s another example, that shows Processes P1 and P2 acquiring resources R1 and R2 while process P3 is waiting to acquire both resources. In this example, there is no deadlock because there is no circular dependency. So cycle in single-instance resource type is the sufficient condition for deadlock.

Example 2 (Multi-instances RAG) –



MULTI INSTANCES WITHOUT DEADLOCK

From the above example, it is not possible to say the RAG is in a safe state or in an unsafe state.



7. Write and explain about the Banker’s algorithm with an example

Bankers’ Algorithm is resource allocation and deadlock avoidance algorithm which test the entire request made by processes for resources, it checks for the safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it doesn’t allow the request made by the process.

Following Data structures are used to implement the Banker’s Algorithm:

Let ‘n’ be the number of processes in the system and ‘m’ be the number of resources types.

Available:

 It is a 1-d array of size ‘m ’indicating the number of available resources of each type.

 Available[j] = k means there are ‘k’ instances of resource type Rj

Max:

 It is a 2-d array of size‘n*m’ that defines the maximum demand of each processing system.

 Max[i,j]=k means process Pi may request atmost ‘k’ instances of resource type Rj.

Allocation:

 It is a 2-darrayofsize ‘n*m ’that defines the number of resources of each type currently
allocated to each process.

 Allocation[i,j]=k means process Pi is currently allocated ‘k’ instances of resource type Rj

Need:

 Itisa2-darrayofsize ‘n*m’ that indicates the remaining resource need of each process.

 Need[i,j] =k means process Pi currently need ‘k’ instances of resource type Rj

 For its execution.

 Need[i,j] =Max[i,j]–Allocation[i,j]

Allocationi specifies the resources currently allocated to process Pi and Need specifies the additional resources that process Pimay still request to complete its task.

Banker’salgorithm consists of Safety algorithm and Resource request algorithm Safety
Algorithm
The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1) Let Work and Finish be vectors of length ‘m’ and ‘n ’respectively.

Initialize: Work = Available
Finish[i]= false; for i=1,2,3,4....n
2) Find an i such that both
a) Finish[i]= false
b) Needi<=Work
If no such I exists go to step(4)
3) Work= Work +Allocation[i]
Finish[i]=true go
to step (2)
4) If Finish[i]=true for all i
Then the system is in a safe state

Resource-Request Algorithm

Let Requesti be the request array for process Pi.Requesti[j]=k means process Piwants k
instances of resource type Rj.When a request for resources is made by process Pi,the following
actions are taken:
1) If Requesti<=Needi
Go to step(2);otherwise,raiseanerrorcondition,sincetheprocesshasexceededitsmaximum claim.
2) If Requesti<=Available
Go to step(3);otherwise, Pimust wait, since the resources are not available.
3) HavethesystempretendtohaveallocatedtherequestedresourcestoprocessPiby modifying the state as follows:
Available = Available – Request i
Allocationi=Allocationi+Requesti
Needi= Needi– Requesti

8. Consider the following

	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	

Answer the following question: i) Construct a need matrix. ii) Is the system in a safe state? If yes what is the safe sequence? iii) If process P1 makes a request (1 0 2) can the request be granted?

Solution:

- **Find out the need matrix**
- **Check if the system in a safe state with steps**
- **Find out the steps if process P1 makes a request (1 0 2) can the request be granted?**

Step 1: Construct the Need Matrix

The Need Matrix is calculated as:

Need[i][j]=Max[i][j]−Allocation[i][j]Need[i][j] = Max[i][j] -
Allocation[i][j]Need[i][j]=Max[i][j]−Allocation[i][j]
Process Allocation (A B C) Max (A B C) Need (A B C)
P0 0 1 0 7 5 3 7 4 3
P1 2 0 0 3 2 2 1 2 2
P2 3 0 0 9 0 2 6 0 2
P3 2 1 1 2 2 2 0 1 1
P4 0 0 2 4 3 3 4 3 1
Step 2: Check if the System is in a Safe State
To check for a safe state, we follow these steps:
1. Initialize:
o Available resources: (A B C) = (3 3 2)
o Finish[] = {false, false, false, false, false} (Initially, no process has finished)
o Safe sequence = {} (Initially empty)
2. Find a process whose Need ≤ Available
P3: Need(0 1 1) ≤ Available(3 3 2) ✓
 Allocate resources to P3, update Available:
New Available = (3+2, 3+1, 2+1) = (5 4 3)
 Finish[P3] = true
 Safe sequence = { P3 }
P1: Need(1 2 2) ≤ Available(5 4 3) ✓
 Allocate resources to P1, update Available:
New Available = (5+2, 4+0, 3+0) = (7 4 3)
 Finish[P1] = true
 Safe sequence = { P3, P1 }
P4: Need(4 3 1) ≤ Available(7 4 3) ✓
 Allocate resources to P4, update Available:

New Available = (7+0, 4+0, 3+2) = (7 4 5)
 Finish[P4] = true
 Safe sequence = { P3, P1, P4 }

P0: Need(7 4 3) ≤ Available(7 4 5) ✅
 Allocate resources to P0, update Available:
New Available = (7+0, 4+1, 5+0) = (7 5 5)
 Finish[P0] = true
 Safe sequence = { P3, P1, P4, P0 }

P2: Need(6 0 2) ≤ Available(7 5 5) ✅
 Allocate resources to P2, update Available:
New Available = (7+3, 5+0, 5+0) = (10 5 5)
 Finish[P2] = true
 Safe sequence = { P3, P1, P4, P0, P2 }

3. All processes can finish, so the system is in a SAFE STATE.
Safe Sequence: P3→P1→P4→P0→P2P3 → P1 → P4 → P0 → P2P3→P1→P4→P0→P2

Step 3: Check if P1's Request (1 0 2) Can Be Granted
P1 is requesting (A B C) = (1 0 2).
We check if the request can be granted by verifying:
1. Request ≤ Available:
(1 0 2) ≤ (3 3 2) ?
Yes, so the request can be granted.
2. New Allocation &amp; Available Resources after Granting P1's Request:
New Allocation for P1 = (2+1, 0+0, 0+2) = (3 0 2)
New Available = (3-1, 3-0, 2-2) = (2 3 0)
New Need for P1 = (1-1, 2-0, 2-2) = (0 2 0)

3. Check for a Safe Sequence Again with Updated Values:
Available resources = (2 3 0)
Following a similar process as Step 2, if a safe sequence exists, the request is granted.

Final Answer
1. Need Matrix:
P0: (7 4 3)
P1: (1 2 2)
P2: (6 0 2)
P3: (0 1 1)
P4: (4 3 1)
2. Safe State:✅Yes
Safe Sequence:{ P3, P1, P4, P0, P2 }
3. Can P1&#39;s Request (1 0 2) Be Granted?✅Yes

PART V

9. What are the various methods used to access file?

File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files −

- • Sequential access
- • Direct/Random access
- • Indexed sequential access

1. Sequential Access

- • A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other. This access method is the most primitive one.
- • The idea of Sequential access is based on the tape model which is a sequential access device.
- • The Sequential access method is best because most of the records in a file are to be processed. For example, transaction files.
- • **Example:** Compilers usually access files in this fashion.

In Brief:

- • Data is accessed one record right after another is an order.
- • Read command cause a pointer to be moved ahead by one.
- • Write command allocate space for the record and move the pointer to the new End of File.
- • Such a method is reasonable for tape.

Advantages of sequential access

- • It is simple to program and easy to design.
- • Sequential file is best use if storage space.

Disadvantages of sequential access

- • Sequential file is time consuming process.
- • It has high data redundancy.
- • Random searching is not possible.

2. Direct Access

- • Sometimes it is not necessary to process every record in a file.
- • It is not necessary to process all the records in the order in which they are present in the memory. In all such cases, direct access is used.
- • The disk is a direct access device which gives us the reliability to random access of any file block.
- • In the file, there is a collection of physical blocks and the records of that blocks.

• • **Example:** Databases are often of this type since they allow query processing that involves immediate access to large amounts of information. All reservation systems fall into this category.

In brief:

• • This method is useful for disks.

• • The file is viewed as a numbered sequence of blocks or records.

• There are no restrictions on which blocks are read/written, it can be dobe in any order.

• • User now says "read n" rather than "read next".

• • "n" is a number relative to the beginning of file, not relative to an absolute physical disk location.

Advantages:

• • Direct access file helps in online transaction processing system (OLTP) like online railway reservation system.

• • In direct access file, sorting of the records are not required.

• • It accesses the desired records immediately.

• • It updates several files quickly.

• • It has better control over record allocation.

Disadvantages:

• • Direct access file does not provide backup facility.

• • It is expensive.

• • It has less storage space as compared to sequential file.

3. Indexed Sequential Access

• • The index sequential access method is a modification of the direct access method.

• • Basically, it is kind of combination of both the sequential access as well as direct access.

• • The main idea of this method is to first access the file directly and then it accesses sequentially.

• • In this access method, it is necessary for maintaining an index.

• • The index is nothing but a pointer to a block.

• • The direct access of the index is made to access a record in a file.

• • The information which is obtained from this access is used to access the file. Sometimes the indexes are very big.

• • So to maintain all these hierarchies of indexes are built in which one direct access of an index leads to information of another index access.

• • It is built on top of Sequential access.

• • It uses an Index to control the pointer while accessing files.

Advantages:

• • In indexed sequential access file, sequential file and random file access is possible.

- • It accesses the records very fast if the index table is properly organized.
- • The records can be inserted in the middle of the file.
- • It provides quick access for sequential and direct processing.
- • It reduces the degree of the sequential search.

Disadvantages:

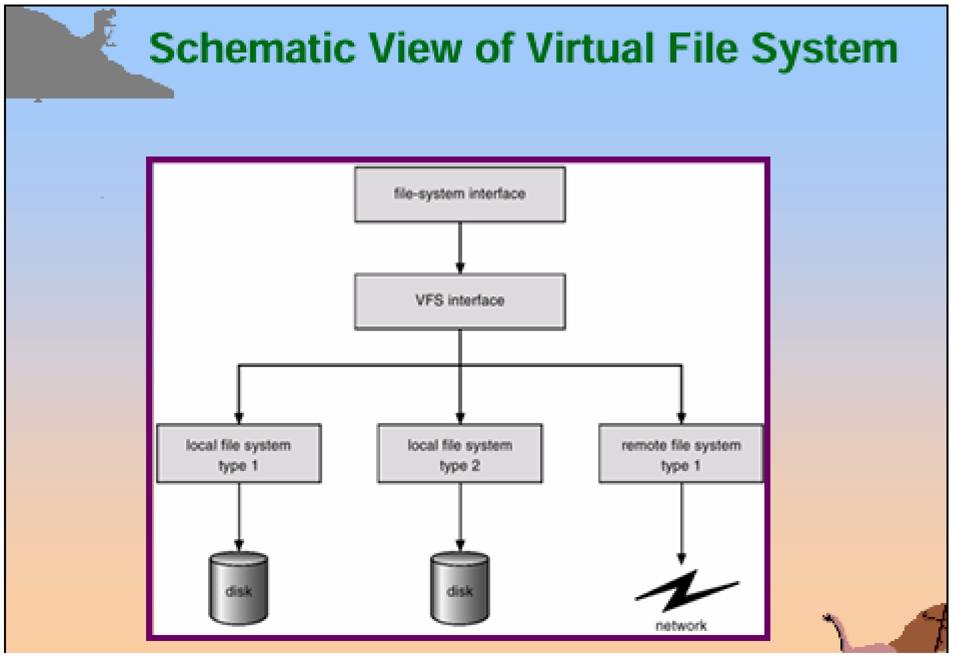
- • Indexed sequential access file requires unique keys and periodic reorganization.
- • Indexed sequential access file takes longer time to search the index for the data access or retrieval.
- It is less efficient in the use of storage space as compared to other file organizations.

10. Explain the VFS with a Schematic diagram

A **Virtual File System (VFS)** is an abstraction layer in an operating system that provides a unified interface for interacting with different file systems. It enables applications to access files stored in different formats (e.g., **ext4, NTFS, FAT32**) without requiring modifications.

- To provide a common interface for multiple file systems.
- To allow applications to access files uniformly, irrespective of their underlying file system.
- To enhance portability by decoupling the application layer from storage-specific details.
- To support **network file systems (NFS, SMB, etc.)** seamlessly.

Architecture of VFS



The **VFS architecture** consists of three major layers:

1. **System Call Interface Layer:**
    - Provides standard system calls like open(), read(), write(), close(), etc.
    - Acts as an intermediary between applications and the OS.
2. **VFS Layer:**
    - Provides a unified interface for different file systems.
    - Uses **inodes, directory entries, and file descriptors** to manage files.
    - Maps file operations to the correct file system.
3. **File System Interface Layer:**
    - Contains implementations of various file systems (e.g., ext4, NTFS, FAT32).
    - Handles file-specific operations such as reading, writing, and metadata management.
4. **Storage Layer:**
    - Represents the physical storage devices (HDDs, SSDs, USBs, etc.).
    - The lowest layer in the VFS architecture.

Working of VFS

1. A user application **requests a file operation** (e.g., open("file.txt")).
2. The **system call interface** processes the request and forwards it to the **VFS layer**.
3. The **VFS** determines which file system (ext4, NTFS, FAT32, etc.) is responsible for the file.
4. The request is **routed to the correct file system driver**.
5. The respective file system **fetches the data from the physical storage**.
6. The **data is returned to the user application**.

Key Features of VFS

- **File System Independence:** Supports multiple file systems seamlessly.
- **Uniform API:** Provides a standard interface for file operations.
- **Efficiency:** Caches frequently accessed files for performance improvement.
- **Network File System Support:** Allows remote file access (e.g., via NFS, SMB).
- **Security & Permissions:** Enforces access control and security policies.

Advantages of VFS

- **Portability:** Applications work across different file systems without modification.
- **Interoperability:** Allows multiple file systems to coexist in a single OS.
- **Performance Optimization:** Uses caching and buffering to improve efficiency.
- **Flexibility:** Supports adding new file systems dynamically.

Examples of File Systems Managed by VFS

File System	Operating System
ext4	Linux
NTFS	Windows
FAT32	Windows, Linux
HFS+	macOS

XFS	Linux

The **Virtual File System (VFS)** is a crucial component of modern operating systems. It provides a uniform interface for accessing different file systems, enhancing **portability, performance, and interoperability**. By abstracting the underlying file system details, VFS simplifies file management and allows seamless interaction with storage devices.