**Unit – 3**

**Part –A**

**11) Distinguish between paging and segmentation?**

A)

| BASIS FOR COMPARISON | PAGING | SEGMENTATION |
|---|---|---|
| Basic | A page is of fixed block size. | A segment is of variable size. |
| Fragmentation | Paging may lead to internal fragmentation. | Segmentation may lead to external fragmentation. |
| Address | The user specified address is divided by CPU into a page number and offset. | The user specifies each address by two quantities a segment number and the offset (Segment limit). |
| Size | The hardware decides the page size. | The segment size is specified by the user. |

| | | |
|---|---|---|
| Table | Paging involves a page table that contains base address of each page. | Segmentation involves the segment table that contains segment number and offset (segment length). |

**12) State the purpose of TLB?**

A) A translation lookaside buffer (TLB) is a memory cache that stores recent translations of virtual memory to physical addresses for faster retrieval.

**13) Explain the basic approach of page replacement?**

A) If no frame is free is available, find one that is not currently being used and free it. A frame can be freed by writing its contents to swap space, and changing the page table to indicate that the page is no longer in memory. Now the freed frame can be used to hold the page for which the process faulted

**14) Distinguish between page table and inverted page table?**

A)

| Page table | Inverted page table |
|---|---|
| A **page table** is maintained by the operating system on a per process basis. | **Inverted page table** is a global page table maintained by the operating system for all the processes. |
| **Every process has its own page table**, and that is why we do not need to store any process identifier(s) in the page table | There is just one page table in the entire system, implying that additional information needs to be stored in the page table to identify page table entries corresponding to each process. |

**15) State the benefits of a virtual memory system?**

A) i) large programs can be written as virtual space available is huge compare to physical memory

ii) less i/o required, leads to faster and easy swapping of process

iii) more physical memory available as programs are stored on virtual memory, so they occupy very less space on actual physical memory

**16) Distinguish between demand paging and pure demand paging?**

A)

| Demand paging | Pure demand paging |
|---|---|
| In demand paging, a page is not loaded into main memory until it is needed | In pure demand paging, even a single page is not loaded into memory initially. Hence pure demand paging causes a page fault |
| In Demand paging follows that pages should only be brought into memory if the executing process demands them | pure demand paging swapping, where all memory for a process is swapped from secondary storage to main memory during the process startup. |

**17) Explain the calculation of effective access time of a demand-paged memory system?**

A) Effective Access Time (EAT)

EAT = (1 – p) x memory access + p (page fault overhead + swap page out + swap page in + restart overhead)

Example: Memory access time = 200 nanoseconds • Average page-fault service time = 8 milliseconds • EAT = (1 – p) x 200 + p (8 milliseconds) = (1 – p x 200 + p x 8,000,000 = 200 + p x 7,999,800

**18) Explain page fault and its effect on the performance of the demand paged memory system?**

A) Page Fault Rate – $0 \leq p \leq 1.0$ – if $p = 0$ no page faults – if $p = 1$, every reference is a fault

**19) Explain the need for page-replacement.?**

A) page replacement algorithm are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page.

**20) List various page replacement algorithms?**

A) 1) FIFO Page Replacement Algorithm
2) Second Chance Page Replacement Algorithm
3) Optimal Page Replacement Algorithm
4) Least Recently Used (LRU) Page Replacement Algorithm

5) LRU Approximation Page Replacement Algorithm
6) Counting Based Page Replacement Algorithm

**21) Distinguish between local and global page replacement strategies?**

A)

| Local | Global |
|---|---|
| When a process incurs a page fault, a local page replacement algorithm selects for replacement some page that belongs to that same process | A global replacement algorithm is free to select any page in memory. |

**22) Distinguish between equal and proportional frame allocation strategies?**

- A) Equal allocation
  - Each process gets the same number of frames. Divide the frames equally between the processes.

- Proportional allocation
  - Allocate frames according to size of process.

**23) Explain the concept of thrashing and why thrashing should be avoided in a system?**

A) We can say that when page fault ratio decreases below level, it is called thrashing.

Trashing should be avoided because it results in severe performance problems.

# Part –B

1) **Describe the following? a) Virtual Memory b) Cache Memory c) Auxiliary Memory**

### Virtual Memory

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.

The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

## Cache Memory

The data or contents of the main memory that are used again and again by CPU, are stored in the cache memory so that we can easily access that data in shorter time.

Whenever the CPU needs to access memory, it first checks the cache memory. If the data is not found in cache memory then the CPU moves onto the main memory. It also transfers block of recent data into the cache and keeps on deleting the old data in cache to accomodate the new one.

## Auxiliary Memory

Devices that provide backup storage are called auxiliary memory. **For example:** Magnetic disks and tapes are commonly used auxiliary devices. Other devices used as auxiliary memory are magnetic drums, magnetic bubble memory and optical disks.

It is not directly accessible to the CPU, and is accessed using the Input/Output channels.

2) **Explain in detail the requirements that memory management technique needs to satisfy?**

## Memory Management Requirements

**Relocation** - the user should not have to know where the program is going to be located. The hardware and OS must work together in this process of relocation.

– Programmer does not know where the program will be placed in memory when it is executed

– While the program is executing, it may be swapped to disk and returned to main memory at a different location (relocated)

– Memory references must be translated in the code to actual physical memory address

**Protection** - keep the process from accessing the address space of another process

– Processes should not be able to reference memory locations in another process without permission

– Impossible to check absolute addresses at compile time

– Must be checked at run time

– Memory protection requirement must be satisfied by the processor (hardware) rather than the operating system (software)

• Operating system cannot anticipate all of the memory references a program will make

**Sharing** - this cooperation is necessary in an OS: shared data areas, share code (DDL)

- Allow several processes to access the same portion of memory

– Better to allow each process access to the same copy of the program rather than have their own separate copy

**Logical organization** - processes are composed of modules or varing sizes, modules independently compiled, modules with different protection needs even to the degree of sharing.

– Programs are written in modules

– Modules can be written and compiled independently

– Different degrees of protection given to modules (read-only, execute-only)

– Share modules among processes

**Physical organization** - typically a two level organization: main memory and secondary memory. The user/programmer cannot know how a program or process will be split across the levels.

– Memory available for a program plus its data may be insufficient

• Overlaying allows various modules to be assigned the same region of memory

- Programmer does not know how much space will be available

### 3) Explain a) Paging b) Page table structure c) Translation look-aside buffer d) Segmentation

# Paging

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non – contiguous.

- Logical Address or Virtual Address (represented in bits): An address generated by the CPU
- Logical Address Space or Virtual Address Space( represented in words or bytes): The set of all logical addresses generated by a program
- Physical Address (represented in bits): An address actually available on memory unit
- Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses

## Example:

- If Logical Address = 31 bit, then Logical Address Space = $2^{31}$ words = 2 G words (1 G = $2^{30}$)

**Page Table:**

A **page table** is the data structure used by a virtual memory system in a computer operating system to store the mapping between virtual addresses and physical addresses. Virtual addresses are used by the program executed by the accessing process, while physical addresses are used by the hardware, or more specifically, by the RAM subsystem.

**Translation look-aside buffer:**

A translation lookaside buffer (TLB) is a memory cache that stores recent translations of virtual memory to physical addresses for faster retrieval.

When a virtual memory address is referenced by a program, the search starts in the CPU. First, instruction caches are checked. If the required memory is not in these very fast caches, the system has to look up the memory's physical address. At this point, TLB is checked for a quick reference to the location in physical memory.

## Segmentation

A Memory Management technique in which memory is divided into variable sized chunks which can be allocated to processes. Each chunk is called a Segment. A table stores the information about all such segments and is called Segment Table.

4) **Explain why the "principle of locality" is crucial to the use of virtual memory? What is accomplished by page buffering?**

- The principle of locality states that program and data references within a process tend to cluster.
- This validates the assumption that only a few pieces of a process are needed over a short period of time.
- This also means that it should be possible to make intelligent guesses about which pieces of a process will be needed in the near future, which avoids thrashing.
- These two things mean that virtual memory is an applicable concept and that it is worth implementing.

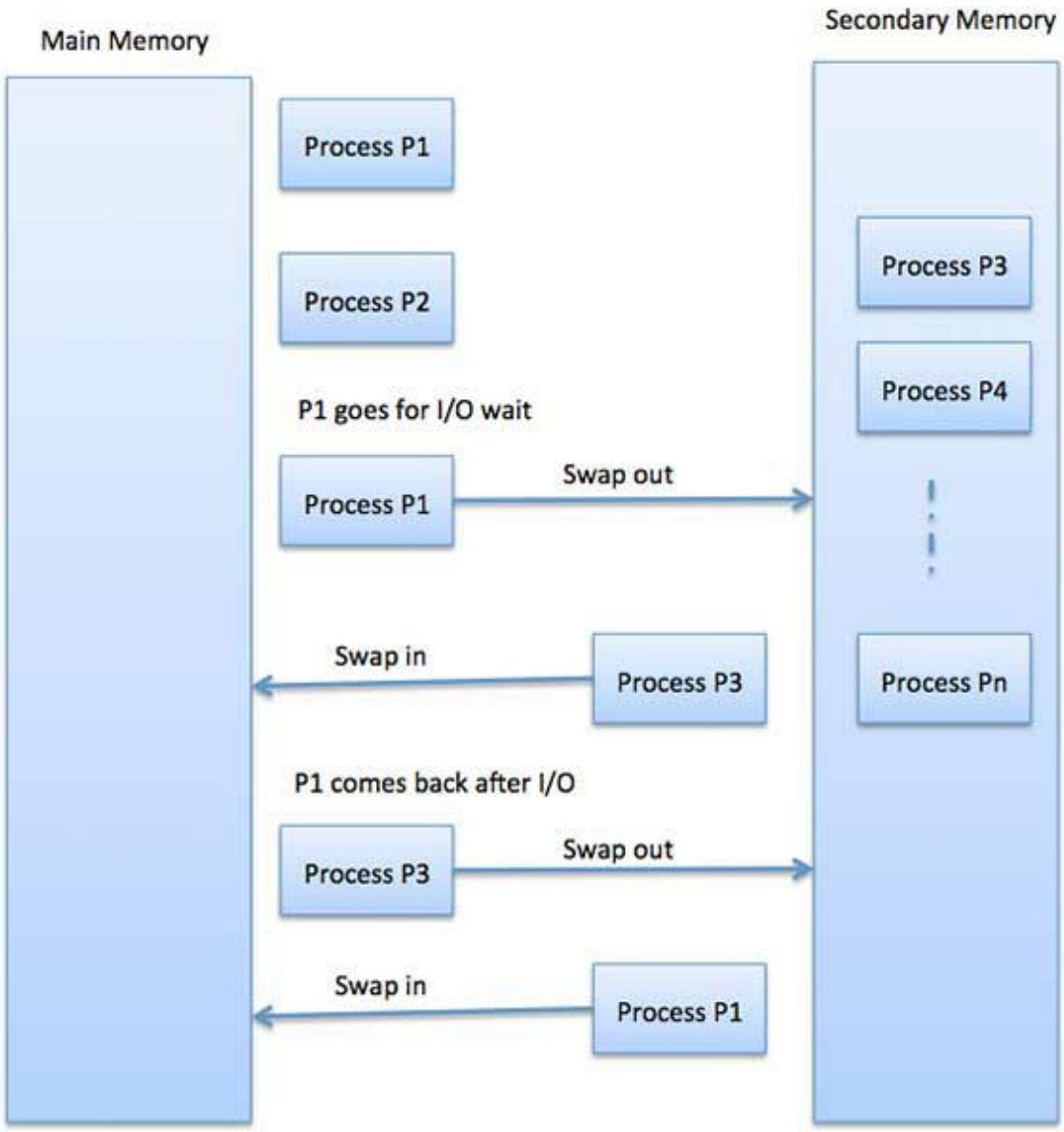
Page buffering essentially creates a cache of pages by assigning a replacement page to one of two lists: the free page list or the modified page list. The page to be replaced remains in memory.

5) **Discuss briefly the swapping concept with necessary examples?**

# Swapping

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason **Swapping is also known as a technique for memory compaction**.




The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Let us assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to or from memory will take

```
2048KB / 1024KB per second
= 2 seconds
= 2000 milliseconds
```

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.

6) **Describe contiguous memory allocation concept with advantages and disadvantages?**

- Contiguous memory allocation is a classical memory allocation model that assigns a process consecutive memory blocks (that is, memory blocks having consecutive addresses).
- Contiguous memory allocation is one of the oldest memory allocation schemes. When a process needs to execute, memory is requested by the process. The size of the process is compared with the amount of contiguous main memory available to execute the process. If sufficient contiguous memory is found, the process is allocated memory to start its execution. Otherwise, it is added to a queue of waiting processes until sufficient free contiguous memory is available.
- The contiguous memory allocation scheme can be implemented in operating systems with the help of two registers, known as the base and limit registers.
- When a process is executing in main memory, its base register contains the starting address of the memory location where the process is executing, while the amount of bytes consumed by the process is stored in the limit register.
- The CPU generates the logical or virtual address, which is converted into an actual address with the help of the memory management unit (MMU). The base address register is used for address translation by the MMU. Thus, a physical address is calculated as follows:

- Physical Address = Base register address + Logical address/Virtual address

Advantages:

1) This Memory allocation provides the direct and easy access.

2) Number of disk required in type of memory allocation is reduced to minimum

3) In case of contiguous memory allocation the good performance remains a positive factor.

Disadvantages:

1) For new files it is very difficult to find the spaces here.

2) Further more you can't extend the file .

3)The one big disadvantage is the difficulty about fragmentation.

## 7) Differentiate the main memory organization schemes of contiguous- memory allocation, segmentation, and paging with respect to the following:

### a. external fragmentation

### b. internal fragmentation

### c. ability to share code across processes

- Contiguous memory allocation scheme suffers from external fragmentation as address spaces are allocated contiguously and holes develop as old processes dies and new processes are initiated.
- It also does not allow processes to share code, since a process's virtual memory segment is not broken into non-contiguous fine grained segments. Pure segmentation also suffers from external fragmentation as a segment of a process is laid out contiguously in physical memory and fragmentation would occur as segments of dead processes are replaced by segments of new processes.
- Segmentation, however, enables processes to share code;
- for instance, two different processes could share a code segment but have distinct data segments. Pure paging does not suffer from external fragmentation, but instead suffers from internal fragmentation. Processes are allocated in page granularity and if a page is not completely utilized, it results in internal fragmentation and a corresponding wastage of space.

Paging also enables processes to share code at the granularity of pages.

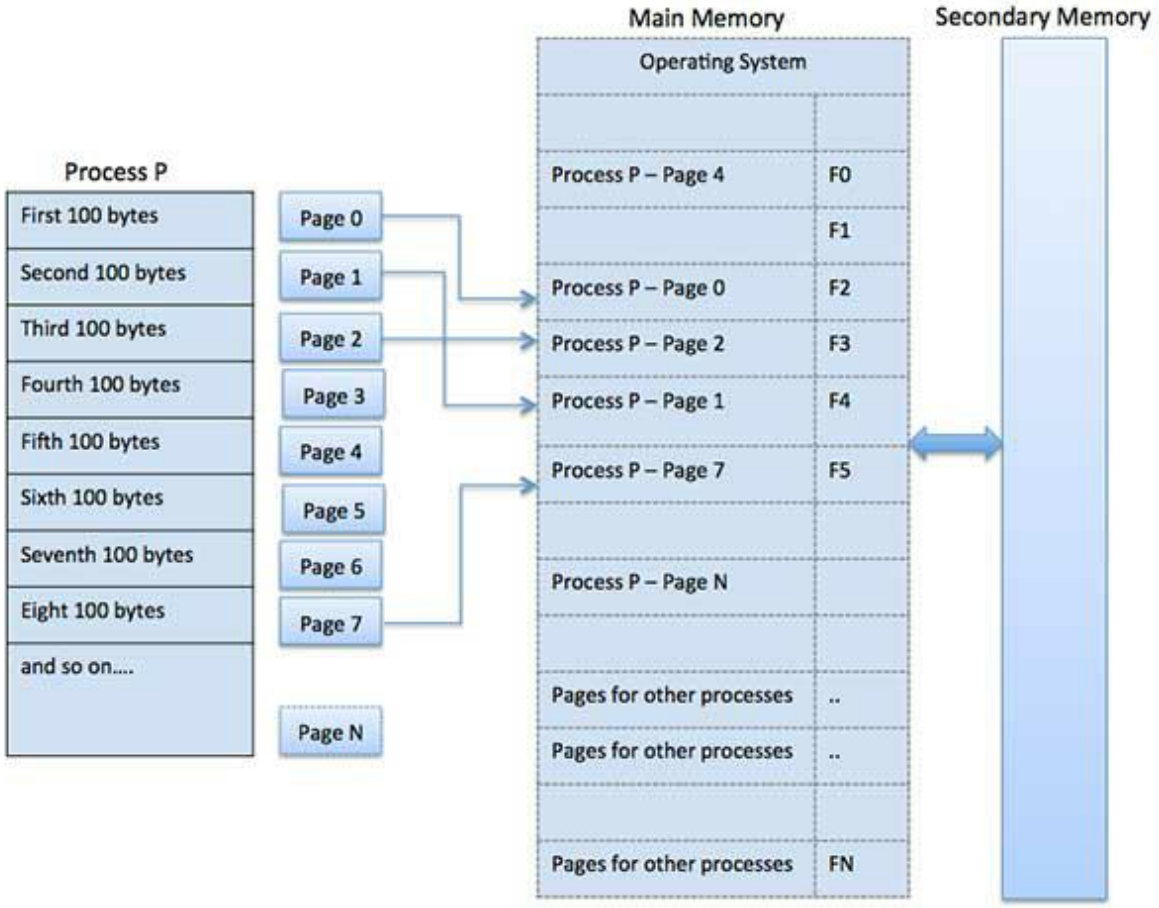
## 8) Differentiate between internal and external fragmentation and Which one occurs in paging scheme?

| BASIS FOR COMPARISON | INTERNAL FRAGMENTATION | EXTERNAL FRAGMENTATION |
|---|---|---|
| Basic | It occurs when fixed sized memory blocks are allocated to the processes. | It occurs when variable size memory space are allocated to the processes dynamically. |
| Occurrence | When the memory assigned to the process is slightly larger than the memory requested by the process this creates free space in the allocated block causing internal fragmentation. | When the process is removed from the memory, it creates the free space in the memory causing external fragmentation. |
| Solution | The memory must be partitioned into variable sized blocks and assign the best fit block to the process. | Compaction, paging and segmentation. |

### 9) Explain briefly about paging with neat diagram?

# Paging

- Paging technique plays an important role in implementing virtual memory.
- Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.
- Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.




## Address Translation

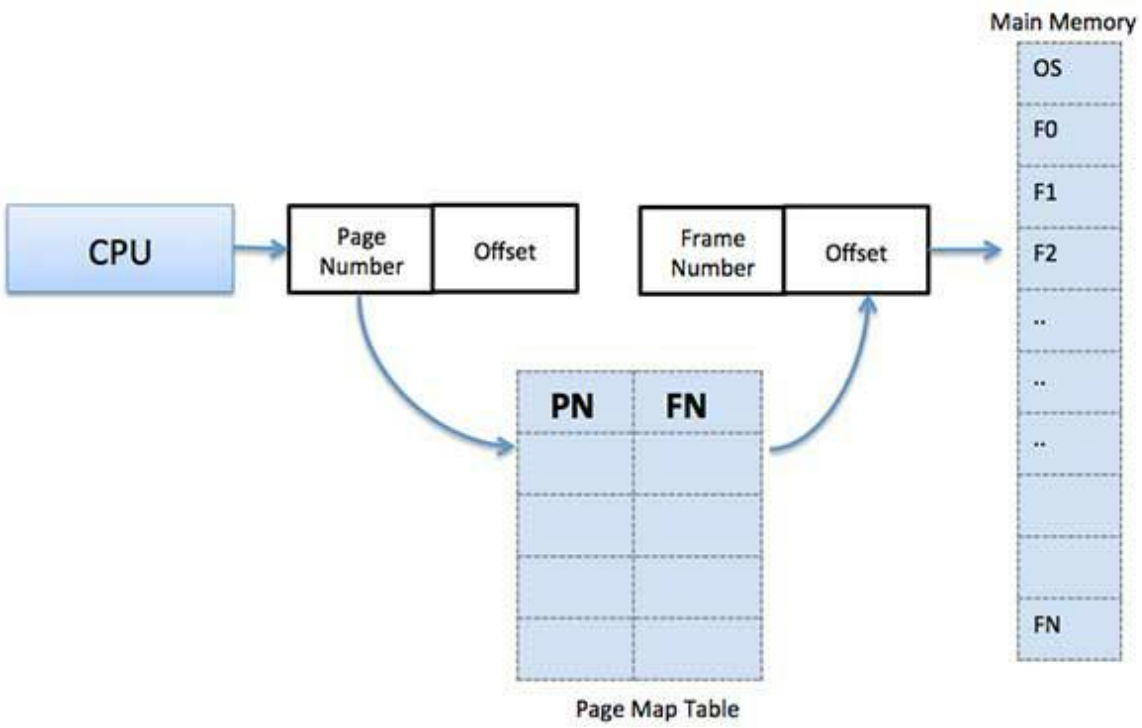
Page address is called **logical address** and represented by **page number**and the **offset**.

```
Logical Address = Page number + page offset
```

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

```
Physical Address = Frame number + page offset
```

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.




When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

## Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging −

- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.

- Page table requires extra memory space, so may not be good for a system having small RAM.

## 10 Discuss the following a) Hierarchical paging b) Inverted page Tables

Hierarchical –

- A multi-tiered table which breaks up the virtual address into multiple parts
- Hierarchical reduces that memory a lot by only adding subtables that are actually in use.
- Still, every process has a root page table. And if the memory footprint of the processes is scattered, there may still be a lot of unnecessary entries in secondary tables.
- This is a far better solution regarding memory than Basic and introduces only a marginal computation increase.

## Inverted page Tables

- An alternate approach is to use the **Inverted Page Table** structure that consists of one page table entry for every frame of the main memory.
- So the number of page table entries in the Inverted Page Table reduces to the number of frames in physical memory and a single page table is used to represent the paging information of all the processes.
- Through inverted page table, the overhead of storing an individual pagetable for every process gets eliminated and only a fixed portion of memory is required to store the paging information of all the processes together.
- This technique is called as inverted paging as the indexing is done with respect to the frame number instead of the logical page number.
- Each entry in the page table contains the following fields.

- **Page number –** It specifies the page number range of the logical address.
- **Process id –** An inverted page table contains the address space information of all the processes in execution. Since two different processes can have similar set of virtual addresses, it becomes necessary in Inverted Page Table to store a process Id of each process to identify it's address space uniquely. This is done by using the combination

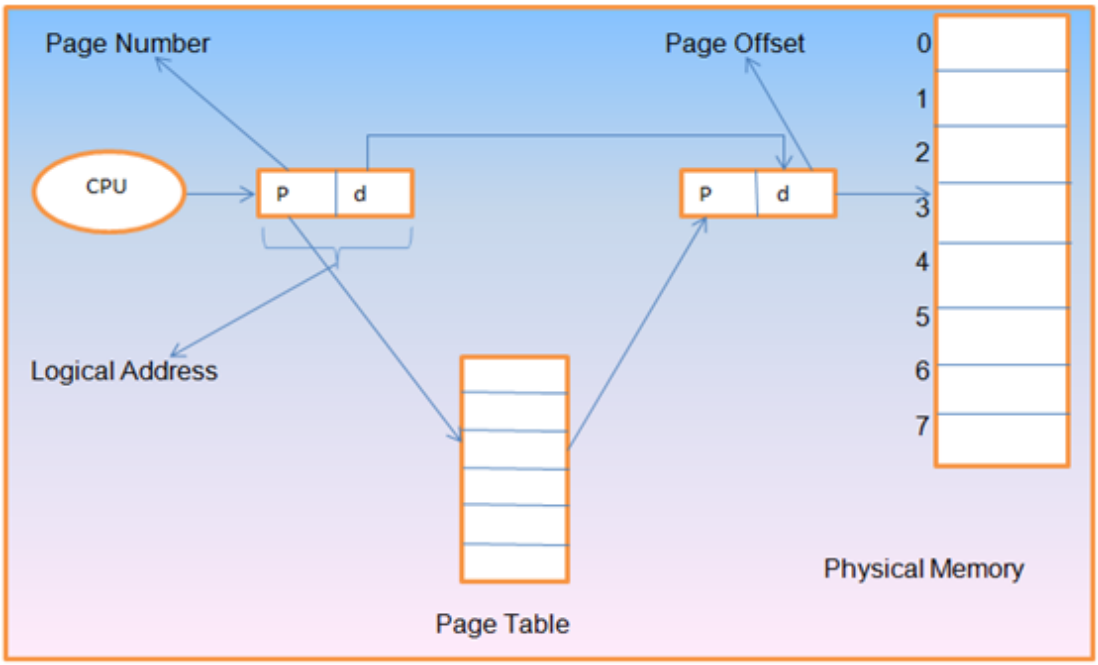
of PId and Page Number. So this Process Id acts as an address space identifier and ensures that a virtual page for a particular process is mapped correctly to the corresponding physical frame.

- **Control bits –** These bits are used to store extra paging-related information. These include the valid bit, dirty bit, reference bits, protection and locking information bits.
- **Chained pointer –** It may be possible sometime that two or more processes share a part of main memory. In this case, two or more logical pages map to same Page Table Entry then a chaining pointer is used to map the details of these logical pages to the root page table.

## 11) Draw and explain the working procedure of paging hardware in detail

- Paging is a memory-management technique that provides the non contiguous address space in main memory.
- Paging avoids external fragmentation. In this technique physical memory is broken into fixed-sized blocks called frames and logical memory is divided into blocks of the same size called pages.
- When a process is to be executed, its pages are loaded into any available memory frames from the backing store.
- The address generated by CPU is called logical address and it is divided into two parts a page number (p) and a page offset (d).
- The page number is used as an index into a page table.
- The page table contains the base address of each page in physical memory.

- The combination of base address and page offset is used to map the page in physical memory address. Hardware decides the page size.



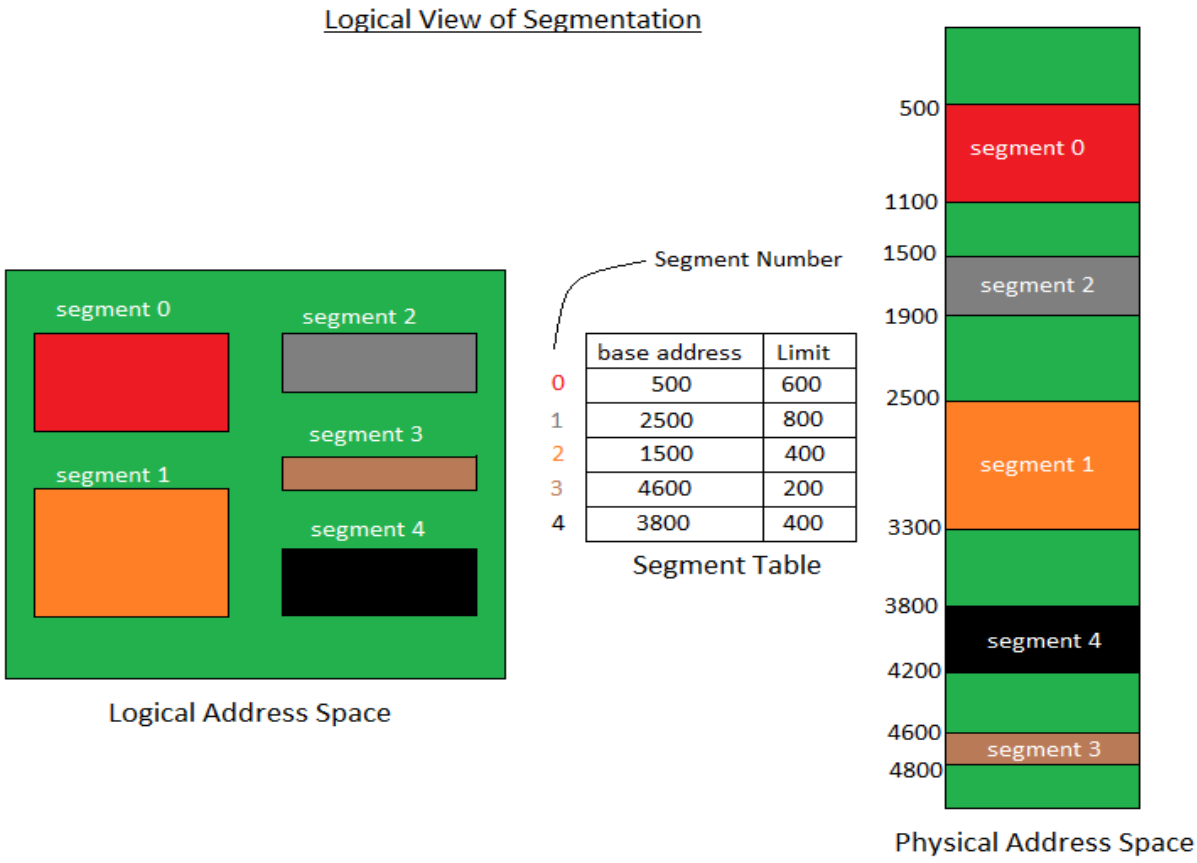

**12) Explain the basic concepts of segmentation with neat diagrams?**

A) A Memory Management technique in which memory is divided into variable sized chunks which can be allocated to processes. Each chunk is called a Segment.
A table stores the information about all such segments and is called Segment Table.

Segment Table: It maps two dimensional Logical address into one dimensional Physical address.
It's each table entry has

- Base Address: It contains the starting physical address where the segments reside in memory.

- Limit: It specifies the length of the segment.




Address generated by the CPU is divided into:

- Segment number (s): Number of bits required to represent the segment.
- Segment offset (d): Number of bits required to represent the size of the segment.

Advantages of Segmentation:

- No Internal fragmentation.
- Segment Table consumes less space in comparison to Page table in paging.

Disadvantage of Segmentation:

- As processes are loaded and removed from the memory, the free memory space is broken into little pieces, causing External fragmentation.

**13) Define page fault? When does a page fault occur? Describe the action taken by OS when page fault occurs?**

A) page fault: When the page (data) requested by a program is not available in the memory, it is called as a page fault.

A page fault occurs when an access to a page that has not been brought into main memory takes place. The operating system verifies the memory access, aborting the program if it is invalid. If it is valid, a free frame is located and I/O is requested to read the needed page into the free frame. Upon completion of I/O, the process table and page table are updated and the instruction is restarted.

**14) State and explain about virtual memory concept with neat diagram?**

A) Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there.

In real scenarios, most processes never need all their pages at once, for following reasons :

- Error handling code is not needed unless that specific error occurs, some of which are quite rare.
- Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
- Certain features of certain programs are rarely used.

Benefits of having Virtual Memory

1. Large programs can be written, as virtual space available is huge compared to physical memory.
2. Less I/O required, leads to faster and easy swapping of processes.
3. More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.
4.

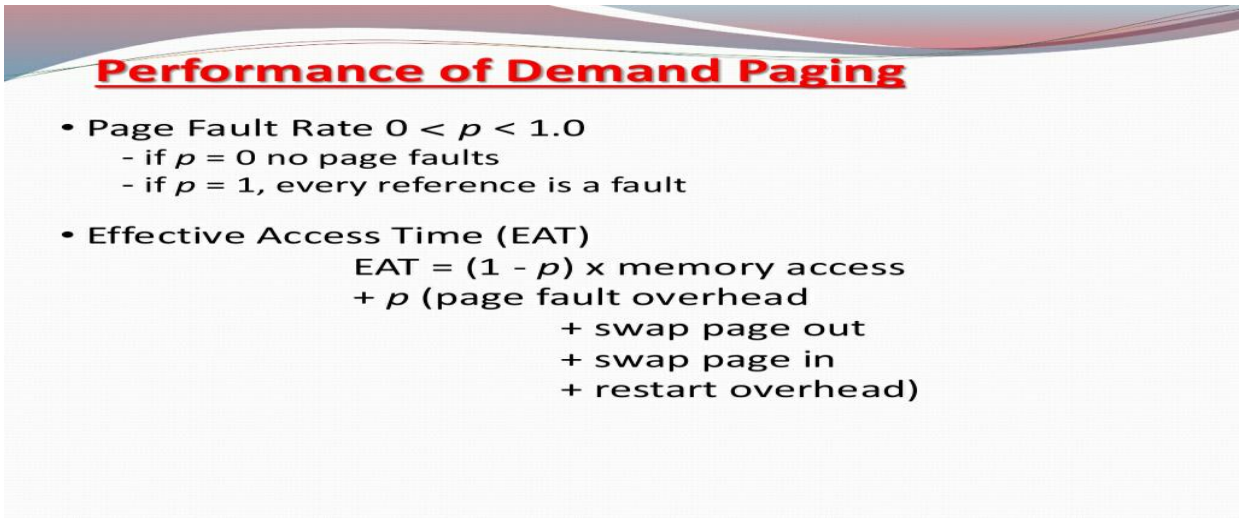
**15) Differentiate between paging and segmentation?**

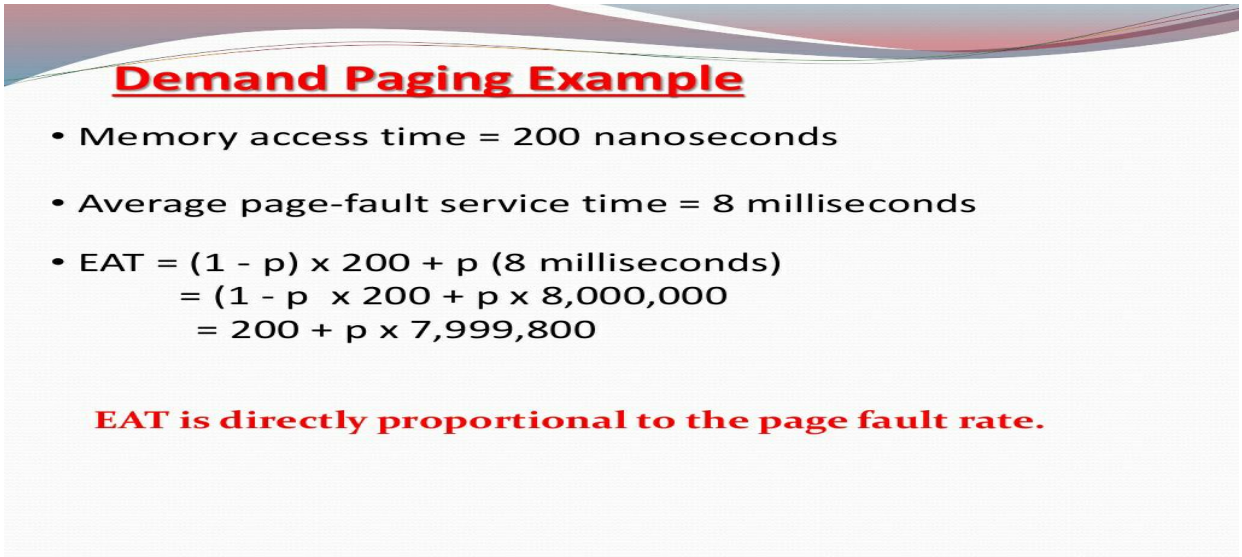
A)

| BASIS FOR COMPARISON | PAGING | SEGMENTATION |
|---|---|---|
| Basic | A page is of fixed block size. | A segment is of variable size. |
| Fragmentation | Paging may lead to internal fragmentation. | Segmentation may lead to external fragmentation. |
| Address | The user specified address is divided by CPU into a page number and offset. | The user specifies each address by two quantities a segment number and the offset (Segment limit). |
| Size | The hardware decides the page size. | The segment size is specified by the user. |
| Table | Paging involves a page table that contains base address of each page. | Segmentation involves the segment table that contains segment number and offset (segment length). |

**16) Explain briefly the performance of demand paging with necessary examples?**

**A)**




**17) Explain the basic Scheme of page replacement and about the various page replacement strategies with examples?**

A) In a operating systems that use paging for memory management, page replacement algorithm are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.

Page Fault – A page fault is a type of interrupt, raised by the hardware when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Various page replacement Strategies / Algorithms

1. The Optimal Page Replacement Algorithm − This algorithm replaces the page that will not be used for the longest period of time. The moment the page fault occurs, some set of pages are in memory. One of these page will be referenced on the very next instruction. Other pages may not be referenced until 10,100 or perhaps 1000 instructions. This information can be stored with each page in the PMT(Page Map Table).

| P# | Base | Offset | MISC |
|---|---|---|---|
| 1 | | | 10 |
| 2 | | | NIL |
| 3 | | | 1000 |
| ... | | | |
| 10 | | | 100 |

2. The optimal page algorithm simply removes the page with the highest number of such instructions implying that it will be needed in the most distant future. this algorithm was introduced long back and is difficult to implement because it requires future knowledge of the program behavior. However it is possible to implement optimal page replacement on the second run by using the page reference information collected on the first run.
3. NRU(Not Recently Used) Page Replacement Algorithm - This algorithm requires that each page have two additional status bits 'R' and 'M' called reference bit and change bit respectively. The reference bit(R) is automatically set to 1 whenever the page is referenced. The change bit (M) is set to 1 whenever the page is modified. These bits are stored in the PMT and are updated on every memory reference.

When a page fault occurs, the memory manager inspects all the pages and divides them into 4 classes based on R and M bits.

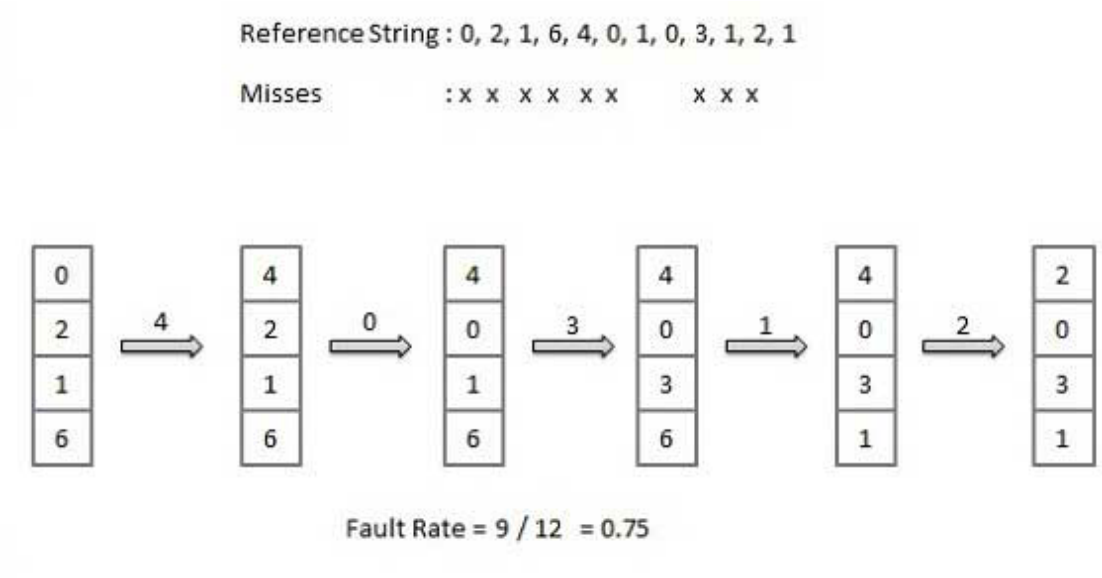
- Class 1: (0,0) − neither recently used nor modified - the best page to replace.
- Class 2: (0,1) − not recently used but modified - the page will need to be written out before replacement.
- Class 3: (1,0) − recently used but clean - probably will be used again soon.
- Class 4: (1,1) − recently used and modified - probably will be used again, and write out will be needed before replacing it.

This algorithm removes a page at random from the lowest numbered non-empty class.

Advantages:

- It is easy to understand.
- It is efficient to implement.

4. FIFO (First in First out) Page Replacement Algorithm − It is one of the simplest page replacement algorithm. The oldest page, which has spent the longest time in memory is chosen and replaced. This algorithm is implemented with the help of FIFO queue to hold the pages in memory. A page is inserted at the rear end of the queue and is replaced at the front of the queue.




In the fig., the reference string is 5, 4, 3, 2, 5, 4, 6, 5, 4, 3, 2, 6 and there are 3 frames empty. The first 3 reference (5, 4, 3) cause page faults and are brought into empty frames. The next reference (2) replaces page 5 because page 5 was loaded first and so on. After 7 page faults, the next reference is 5 and 5 is already in

memory so no page fault for this reference. Similarly for next reference 4. The + marks shows incoming of a page while circle shows the page chosen for removal.

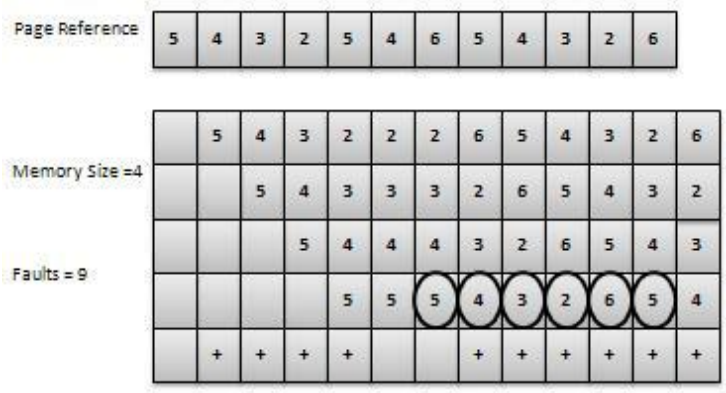
Advantages

- FIFO is easy to understand.
- It is very easy to implement.

Disadvantage

- Not always good at performance. It may replace an active page to bring a new one and thus may cause a page fault of that page immediately.
- Another unexpected side effect is the FIFO anomaly or Belady's anomaly. This anomaly says that the page fault rate may increase as the number of allocated page frames increases.

e.g. The following figure presents the same page trace but with a larger memory. Here number of page frame is 4.

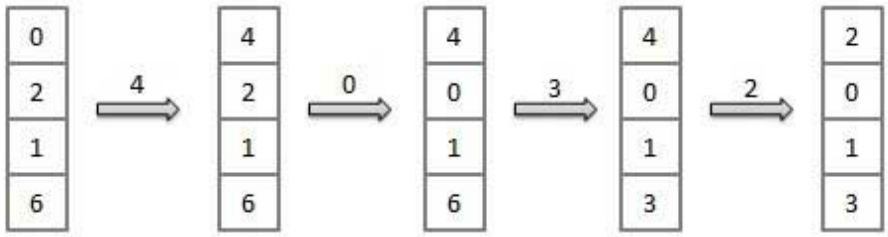



Here page faults are 10 instead of 9.

5. LRU(Least Recently Used) Algorithm − The Least Recently used (LRU) algorithm replaces the page that has not been used for the longest period of time. It is based on the observation that pages that have not been used for long time will probably remain unused for the longest time and are to be replaced.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x




Fault Rate = 8 / 12 = 0.67

Initially, 3 page frames are empty. The first 3 references (7, 0, 1) cause page faults and are brought into these empty frames. The next reference (2) replaces page 7. Since next page reference (0) is already in memory, there is no page fault. Now, for the next reference (3), LRU replacement sees that, of the three frames in memory, page 1 was used least recently, and thus is replaced. And thus the process continues.

Advantages

- LRU page replacement algorithm is quiet efficient.
- It does not suffer from Belady's Anomaly.

Disadvantages

- Its implementation is not very easy.
- Its implementation may require substantial hardware assistance.

**18) Explain the Readers and Writers problem and its solution using the concept of semaphores?**

A) refer part-B 21A

**19) Explain the uses of the following: a. Mutex object b. Semaphore object c. Waitable timer object**

**A)**

**a)** A *mutex object* is a synchronization object whose state is set to signaled when it is not owned by any thread, and nonsignaled when it is owned. Only one thread at a time can own a mutex object, whose name comes from the fact that it is useful in coordinating mutually exclusive access to a shared resource. For example, to prevent two threads from writing to shared memory at the same time, each thread waits for ownership of a mutex object before executing the code that accesses the memory. After writing to the shared memory, the thread releases the mutex object.

b) The semaphore object is useful in controlling a shared resource that can support a limited number of users. It acts as a gate that limits the number of threads sharing the resource to a specified maximum number. For example, an application might place a limit on the number of windows that it creates. It uses a semaphore with a maximum count equal to the window limit, decrementing the count whenever a window is created and incrementing it whenever a window is closed. The application specifies the semaphore object in call to one of the wait functions before each window is created. When the count is zero—indicating that the window limit has been reached—the wait function blocks execution of the window-creation code.

c) A thread uses the CreateWaitableTimer or CreateWaitableTimerEx function to create a timer object. The creating thread specifies whether the timer is a manual-reset timer or a synchronization timer. The creating thread can specify a name for the timer object. Threads in other processes can open a handle to an existing timer by specifying its name in a call to the OpenWaitableTimer function. Any thread with a handle to a timer object can use one of the wait functions to wait for the timer state to be set to signaled

**20) Write short notes about the following: a. Binary Semaphores b. Bounded Waiting**

**A)**

**a)** Quite obvious, binary semaphore can have a value either 0 or 1. It means binary semaphore protect the access to a SINGLE shared resource, so the internal counter of the semaphore can only take the values 1 or 0.

So whenever you have a requirement for protecting the access to a SINGLE resource accessed by multiple threads, you can use Binary Semaphore.

**b)** After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request

is granted. So after the limit is reached, system must grant the process permission to get into its critical section.

**21) Explain the Readers and Writers problem and its solution using the concept of semaphores?**

A) What is Readers Writer Problem?

Readers writer problem is another example of a classic synchronization problem. There are many variants of this problem, one of which is examined below.

The Problem Statement

There is a shared resource which should be accessed by multiple processes. There are two types of processes in this context. They are reader and writer. Any number of readers can read from the shared resource simultaneously, but only one writer can write to the shared resource. When a writeris writing data to the resource, no other process can access the resource. A writer cannot write to the resource if there are non zero number of readers accessing the resource at that time.

The Solution

From the above problem statement, it is evident that readers have higher priority than writer. If a writer wants to write to the resource, it must wait until there are no readers currently accessing that resource.

Here, we use one mutex m and a semaphore w. An integer variable read_count is used to maintain the number of readers currently accessing the resource. The variable read_count is initialized to 0. A value of 1 is given initially to m and w.

Instead of having the process to acquire lock on the shared resource, we use the mutex m to make the process to acquire and release lock whenever it is updating the read_count variable.

The code for the writer process looks like this:

```
while(TRUE)

{

   wait(w);
```

```
   /* perform the write operation */

   signal(w);
}
```

And, the code for the reader process looks like this:

```
while(TRUE)
{
    //acquire lock
    wait(m);
    read_count++;
    if(read_count == 1)
        wait(w);

    //release lock
    signal(m);

    /* perform the reading operation */

    // acquire lock
    wait(m);
    read_count--;
    if(read_count == 0)
```

```
        signal(w);


    // release lock
    signal(m);
}
```

**Part –c**

**6) Analyze that we have a paging system with page table stored in memory A. If a memory reference takes 200 nanoseconds how long does a paged B. If we add associative registers and 75% of all page table references are memory reference take found in the associative registers, what is the effective memory reference time? Assume that finding a page table entry in the associative registers takes zero time, if the entry is there.**

A)

A) 2 memory accesses: page lookup followed by actual access => 2*200ns = 400ns

B) 75% * TLB hit-time + 25% * TLB miss-time = 75% * 200ns + 25% * 400ns = 250ns

**7) In two level nested loops, the outer index (i) runs from 1 to 5 and the inner index (j) runs from 1 to 10.The page faults seem to occur for every 7th innermost iterations. If it takes 0.02 micro second to load a new page what is the extra time required because of occurrence of page faults?**

A)

**8) Given memory partitions of 100K, 500K, 200K, 300K, and 600K (in order), how would each of the First-fit, Best-fit, and Worst-fit algorithms place processes of 212K, 417K, 112K, and 426K (in order)? Explain Which algorithm makes the most efficient use of memory?**

A) First-fit:

212K is put in 500K partition

417K is put in 600K partition

112K is put in 288K partition (new partition 288K = 500K - 212K)

426K must wait

Best-fit:

212K is put in 300K partition

417K is put in 500K partition
112K is put in 200K partition
426K is put in 600K partition

Worst-fit:
212K is put in 600K partition
417K is put in 500K partition
112K is put in 388K partition
426K must wait

In this example, best-fit turns out to be the best

**9) Suppose we have a demand paged memory. The page table is held in registers. It takes 8 milliseconds to service a page fault if an empty frame is available or the replaced page is not modified and 20 milliseconds if the replaced page is modified. Memory access time is 100 nanoseconds. Consider that the page to be replaced is modified 70 percent of the time. What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds?**

A) *Find the effective access time (EAT) for a given page-fault rate (p) - Can solve the following equation:*

EAT = (1-p)*(100) + (p) * (100+(1-0.7) * (8msec) + (0.7)*(20msec))
= 100 -100p + 100p + (2.4e6) *p + (14e6)*p (1ms=10^6ns)

200 = 100 + (16.4e6)*p
p = 100/16.4e6 = 6.1*10^-6 (or) 6.097598e-6

**(or)**

# Demand Paged Memory

PFST (not Modified) = 8 ms

PFST (modified) = 20 ms

MM access time = 100 ns

70 % Page to be replaced is modified

Eff. Access Time = 200 ns

So, Eff. Access Time = Page fault rate * (Modified % PFST (modified) + (1-Modified)% PFST (Not Modified) + (1-Page fault rate) * MMAT

Let's Say Page fault rate = p

$200\,ns = p * (0.7 * 20ms + 0.3 * 8ms) + (1-p) * 100ns$

$200\,ns = p\,(14 \times 10^6 ns + 2.4 \times 10^6 ns) + (1-p)\,100\,ns$

$1\,ms = 10^6\,ns$

$2 = 16.4 \times 10^4 p + (1-p)$

$1 = (16.4 \times 10^4 - 1)\,p$

$1 = (163999)\,p$

$p = \frac{1}{163999} = 6.1 \times 10^{-6}$

option (A)

**10) Consider a logical address space of eight pages of 1024 words each mapped onto a physical memory of 32 frames a) How many bits are in the logical address? b) How many bits are in the physical address?**

A) Addressing within a 1024-word page requires 10 bits because $1024 = 2^{10}$. Since the logical address space consists of $8 = 2^3$ pages, the logical addresses must be 10+3 = 13 bits. Similarly, since there are $32 = 2^5$ physical pages, phyiscal addresses are 5 + 10 = 15 bits long.

Physical Address (P = page number bits)

P P P P P - - - - - - - - - -

Logical Address (P = page number bits)

P P P - - - - - - - - - -