www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

UNIT – IV MEMORY MANAGEMENT and VIRTUAL MEMORY

Main Memory

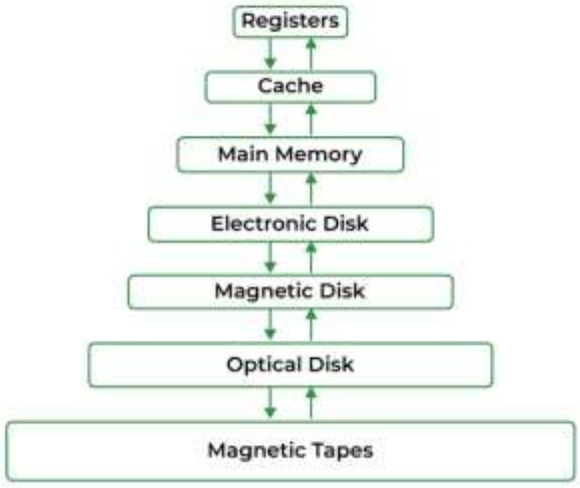
Main Memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions.

Main memory is a repository of rapidly available information shared by the CPU and I/O devices.

Main memory is the place where programs and information are kept when the processor is effectively utilizing them.

Main memory is associated with the processor, so moving instructions and information into and out of the processor is extremely fast.

Main memory is also known as RAM (Random Access Memory). This memory is volatile. RAM loses its data when a power interruption occurs.



Memory Management

In a multiprogramming computer, the Operating System resides in a part of memory, and the rest is used by multiple processes.

The task of subdividing the memory among different processes is called Memory Management.

Memory management is a method in the operating system to manage operations between main memory and disk during process execution.

The main aim of memory management is to achieve efficient utilization of memory.

Requirement of Memory Management

- Allocate and de-allocate memory before and after process execution.
- To keep track of used memory space by processes.
- To minimize fragmentation issues.
- To proper utilization of main memory.
- To maintain data integrity while executing of process.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

Logical Address Space

An address generated by the CPU is known as a "Logical Address". It is also known as a Virtual address. Logical address space can be defined as the size of the process. A logical address can be changed.

Physical Address Space

An address seen by the memory unit (i.e the one loaded into the memory address register of the memory) is commonly known as a "Physical Address". A Physical address is also known as a Real address.

The set of all physical addresses corresponding to these logical addresses is known as Physical address space.

A physical address is computed by MMU. The run-time mapping from virtual to physical addresses is done by a hardware device Memory Management Unit (MMU). The physical address always remains constant.

Static and Dynamic Loading

Loading a process into the main memory is done by a loader. There are two different types of loading :

(i) Static Loading

Static Loading is basically loading the entire program into a fixed address. It requires more memory space.

(ii) Dynamic Loading

The entire program and all data of a process must be in physical memory for the process to execute. So, the size of a process is limited to the size of physical memory. To gain proper memory utilization, dynamic loading is used. In dynamic loading, a routine is not loaded until it is called. All routines are residing on disk in a relocatable load format.

One of the advantages of dynamic loading is that the unused routine is never loaded. This loading is useful when a large amount of code is needed to handle it efficiently.

Static and Dynamic Linking

To perform a linking task a linker is used. A linker is a program that takes one or more object files generated by a compiler and combines them into a single executable file.

(i) Static Linking

In static linking, the linker combines all necessary program modules into a single executable program. So there is no runtime dependency. Some operating systems support only static linking, in which system language libraries are treated like any other object module.

(ii) Dynamic Linking

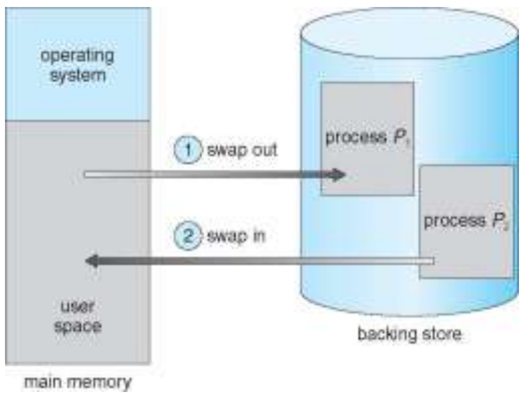
The basic concept of dynamic linking is similar to dynamic loading. In dynamic linking, "Stub" is included for each appropriate library routine reference. A stub is a small piece of code. When the stub is executed, it checks whether the needed routine is already in memory or not. If not available then the program loads the routine into memory.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

SWAPPING

When a process is executed it must have resided in memory. Swapping is a process of swapping a process temporarily into a secondary memory from the main memory, which is fast compared to secondary memory. A swapping allows more processes to be run and can be fit into memory at one time.



The main part of swapping is transferred time and the total time is directly proportional to the amount of memory swapped.

Swapping is also known as roll-out, or roll because if a higher priority process arrives and wants service, the memory manager can swap out the lower priority process and then load and execute the higher priority process. After finishing higher priority work, the lower priority process swapped back in memory and continued to the execution process.

Advantages

- If there is low main memory so some processes may has to wait for much long but by using swapping process do not have to wait long for execution on CPU.
- It utilize the main memory.
- Using only single main memory, multiple process can be run by CPU using swap partition.
- The concept of virtual memory start from here and it utilize it in better way.
- This concept can be useful in priority based scheduling to optimize the swapping process.

Disadvantages

- If there is low main memory resource and user is executing too many processes and suddenly the power of system goes off there might be a scenario where data get erase of the processes which are took parts in swapping.
- Chances of number of page faults occur
- Low processing performance

Example:

Suppose the user process's size is 2048KB and is a standard hard disk where swapping has a data transfer rate of 1Mbps. Calculate how long it will take to transfer from main memory to secondary memory.

User process size is 2048Kb

Data transfer rate is 1Mbps = 1024 kbps

Time = process size / transfer rate

= 2048 / 1024 = 2 seconds or 2000 milliseconds

Now taking swap-in and swap-out time, the process will take 4000 ms or 4 seconds.

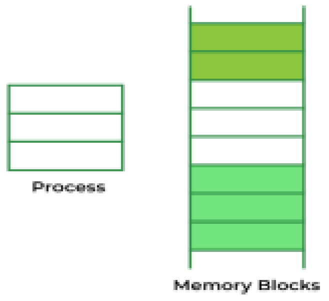
www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

Contiguous Memory Allocation

The main memory should accommodate both the operating system and the different client processes. Therefore, the allocation of memory becomes an important task in the operating system. The memory is usually divided into two partitions: one for the resident operating system and one for the user processes.

We normally need several user processes to reside in memory simultaneously. Therefore, we need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory. In adjacent memory allotment, each process is contained in a single contiguous segment of memory.



Memory Allocation

To gain proper memory utilization, memory allocation must be allocated efficient manner. One of the simplest methods for allocating memory is to divide memory into several fixed-sized partitions and each partition contains exactly one process. Thus, the degree of multiprogramming is obtained by the number of partitions.

- **Multiple partition allocation**

  A process is selected from the input queue and loaded into the free partition. When the process terminates, the partition becomes available for other processes.

- **Fixed partition allocation**

  The operating system maintains a table that indicates which parts of memory are available and which are occupied by processes.

  Initially, all memory is available for user processes and is considered one large block of available memory. This available memory is known as a "Hole".

  When the process arrives and needs memory, we search for a hole that is large enough to store this process. If the requirement is fulfilled then we allocate memory to process, otherwise keeping the rest available to satisfy future requests.

While allocating a memory sometimes dynamic storage allocation problems occur, which concerns how to satisfy a request of size **n** from a list of free holes. There are some solutions to this problem:

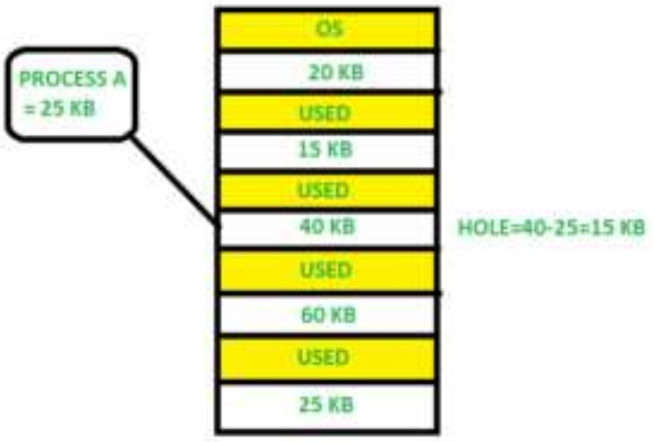
www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

First Fit

In the First Fit, the first available free hole fulfil the requirement of the process allocated.

Here, in this diagram, a 40 KB memory block is the first available free hole that can store process A (size of 25 KB), because the first two blocks did not have sufficient memory space.

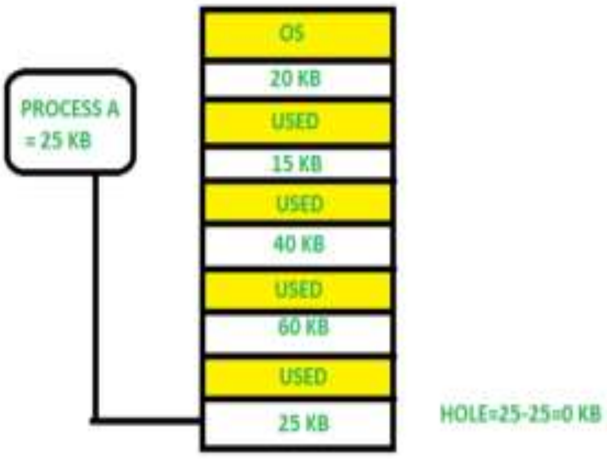


Best Fit

In the Best Fit, allocate the smallest hole that is big enough to process requirements. For this, we search the entire list, unless the list is ordered by size.

Here in this example, first, we traverse the complete list and find the last hole 25KB is the best suitable hole for Process A(size 25KB).

In this method, memory utilization is maximum as compared to other memory allocation techniques.

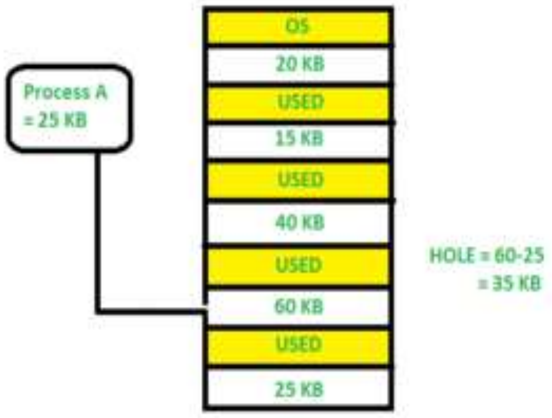


Worst Fit

In the Worst Fit, allocate the largest available hole to process. This method produces the largest leftover hole.

Here in this example, Process A (Size 25 KB) is allocated to the largest available memory block which is 60KB.

Inefficient memory utilization is a major issue in the worst fit.



www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

FRAGMENTATION

Fragmentation is defined as when the process is loaded and removed after execution from memory, it creates a small free hole. These holes can not be assigned to new processes because holes are not combined or do not fulfill the memory requirement of the process.

To achieve a degree of multiprogramming, we must reduce the waste of memory or fragmentation problems. In the operating systems two types of fragmentation:

1. **Internal fragmentation**

   Internal fragmentation occurs when memory blocks are allocated to the process more than their requested size. Due to this some unused space is left over and creating an internal fragmentation problem.

   Example: Suppose there is a fixed partitioning used for memory allocation and the different sizes of blocks 3MB, 6MB, and 7MB space in memory. Now a new process p4 of size 2MB comes and demands a block of memory. It gets a memory block of 3MB but 1MB block of memory is a waste, and it can not be allocated to other processes too. This is called internal fragmentation.

2. **External fragmentation**

   In External Fragmentation, we have a free memory block, but we can not assign it to a process because blocks are not contiguous.

   Example: Suppose (consider the above example) three processes p1, p2, and p3 come with sizes 2MB, 4MB, and 7MB respectively. Now they get memory blocks of size 3MB, 6MB, and 7MB allocated respectively. After allocating the process p1 process and the p2 process left 1MB and 2MB. Suppose a new process p4 comes and demands a 3MB block of memory, which is available, but we can’t assign it because free memory space is not contiguous. This is called external fragmentation.

Both the first-fit and best-fit systems for memory allocation are affected by external fragmentation.

To overcome the external fragmentation problem **Compaction** is used. In the compaction technique, all free memory space combines and makes one large block. So, this space can be used by other processes effectively.

Another possible solution to the external fragmentation is to allow the logical address space of the processes to be non-contiguous, thus permitting a process to be allocated physical memory wherever the latter is available.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

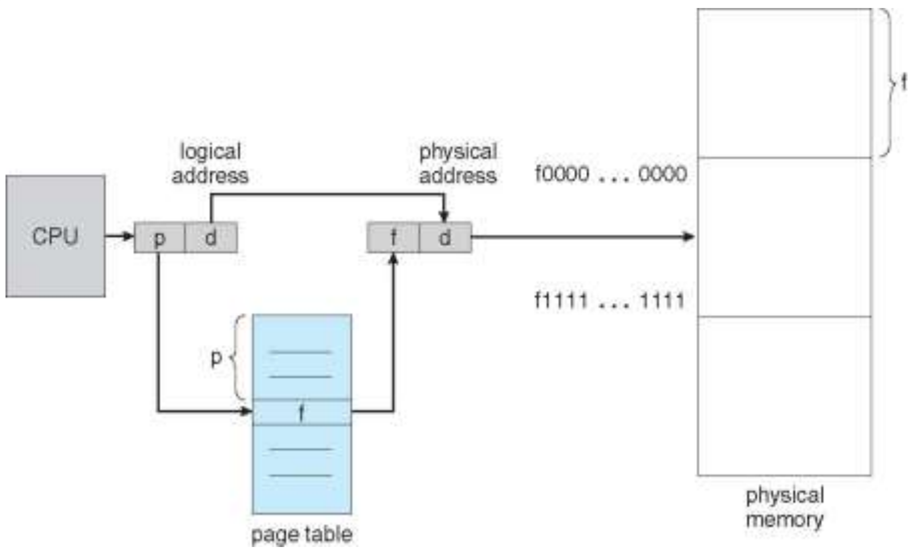
www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

PAGING

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

The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device and this mapping is known as the paging technique.

- The Physical Address Space is conceptually divided into several fixed-size blocks, called **frames**.
- The Logical Address Space is also split into fixed-size blocks, called **pages**.
- Page Size = Frame Size



The address generated by the CPU is divided into:

- **Page Number(p)**

  Number of bits required to represent the pages in Logical Address Space or Page number
- **Page Offset(d)**

  Number of bits required to represent a particular word in a page or page size of Logical Address Space or word number of a page or page offset.

Physical Address is divided into:

- **Frame Number(f)**

  Number of bits required to represent the frame of Physical Address Space or Frame number frame
- **Frame Offset(d)**

  Number of bits required to represent a particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

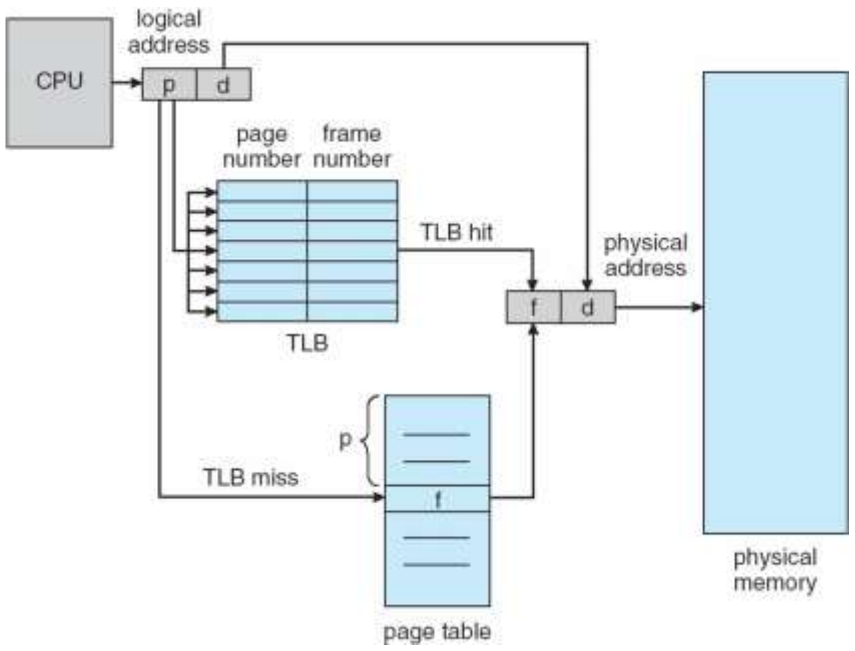
www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

PAGING HARDWARE WITH TLB

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

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

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

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

SEGMENTATION

A process is divided into Segments. The chunks that a program is divided into which are not necessarily all of the exact sizes are called segments. Segmentation gives the user's view of the process which paging does not provide.

Here the user's view is mapped to physical memory. There is no simple relationship between logical addresses and physical addresses in segmentation.

A table stores the information about all such segments and is called **Segment Table**. It maps a two-dimensional Logical address into a one-dimensional Physical address. It's each table entry has:

Base Address:

It contains the starting physical address where the segments reside in memory.

Segment Limit:

Also known as segment offset. It specifies the length of the segment.

The address generated by the CPU is divided into:

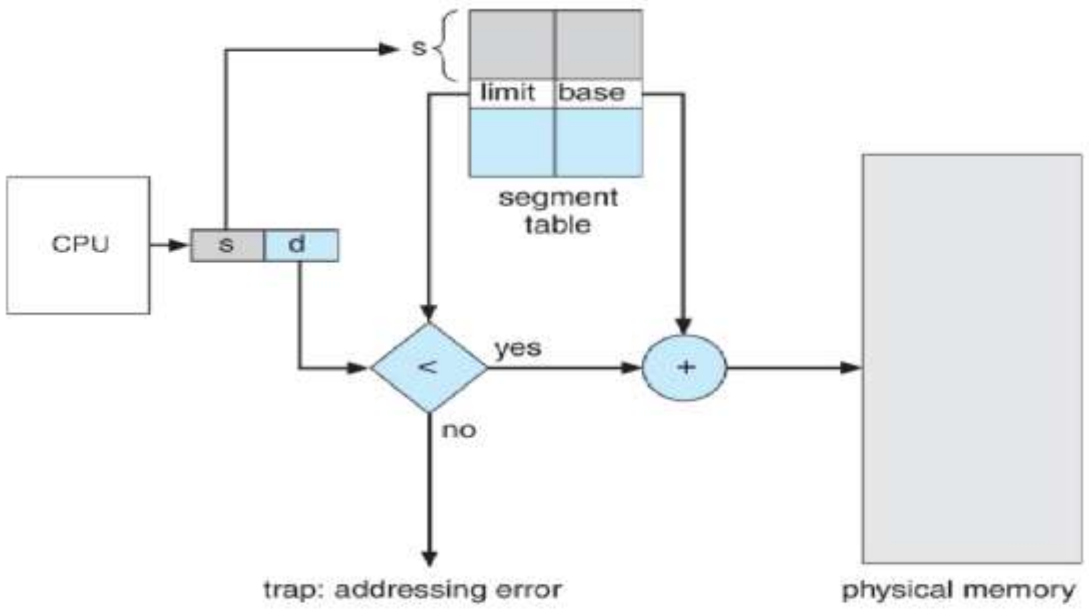
Segment number (s):

Number of bits required to represent the segment.

Segment offset (d):

Number of bits required to represent the size of the segment.

The **Segment number** is mapped to the segment table. The limit of the respective segment is compared with the offset. If the offset is less than the limit then the address is valid otherwise it throws an error as the address is invalid. In the case of valid addresses, the base address of the segment is added to the offset to get the physical address of the actual word in the main memory.



www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

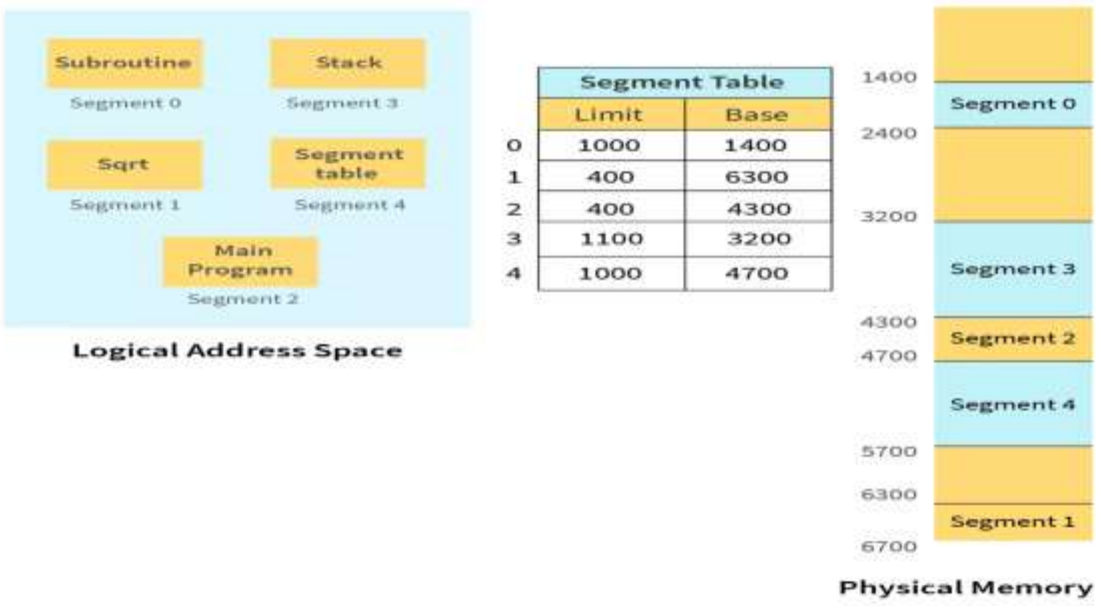
www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

Example of Segmentation

Let us assume we have five segments namely: Segment-0, Segment-1, Segment-2, Segment-3, and Segment-4. Initially, before the execution of the process, all the segments of the process are stored in the physical memory space. We have a segment table as well. The segment table contains the beginning entry address of each segment (denoted by **base**). The segment table also contains the length of each of the segments (denoted by **limit**).

As shown in the image below, the base address of Segment-0 is 1400 and its length is 1000, the base address of Segment-1 is 6300 and its length is 400, the base address of Segment-2 is 4300 and its length is 400, and so on.

The pictorial representation of the above segmentation with its segment table is shown below.



Segment Table

	Limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

SEGMENTATION WITH PAGING

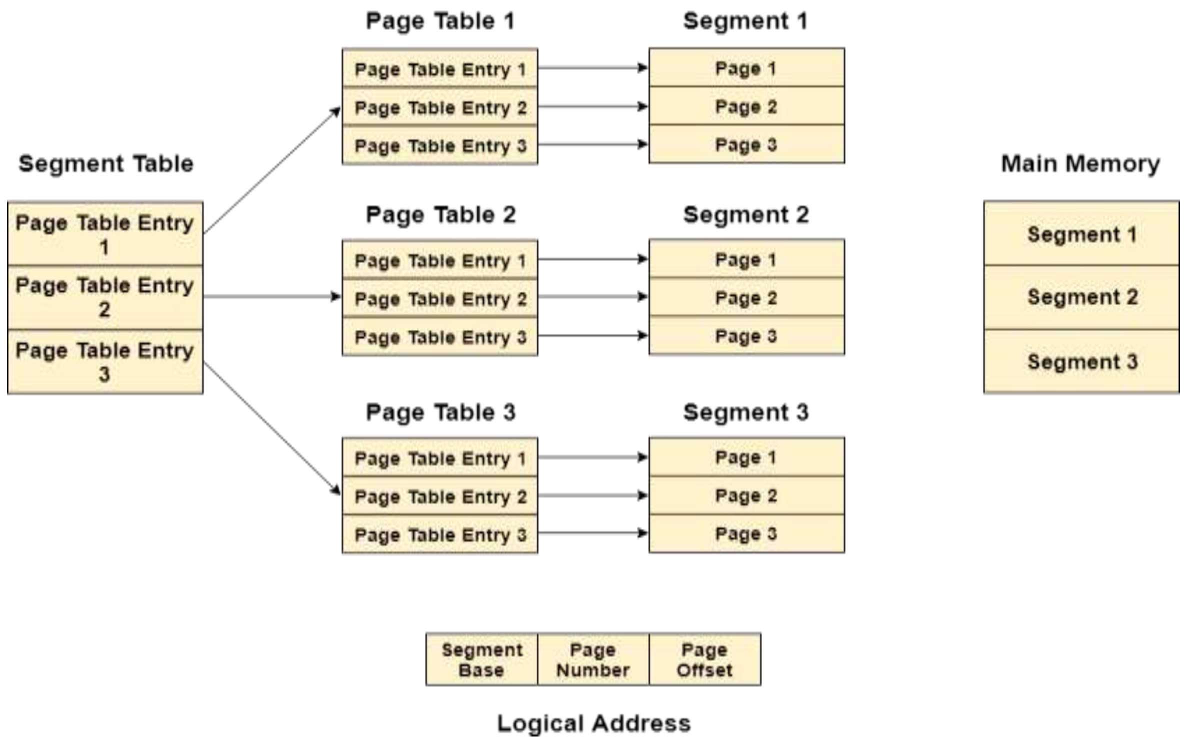
Pure segmentation is not very popular and not being used in many of the operating systems. However, Segmentation can be combined with Paging to get the best features out of both the techniques.

In Segmented Paging, the main memory is divided into variable size segments which are further divided into fixed size pages.

- Pages are smaller than segments.
- Each Segment has a page table which means every program has multiple page tables.
- The logical address is represented as Segment Number (base address), Page number and page offset.

Segment Number → It points to the appropriate Segment Number.
Page Number → It Points to the exact page within the segment
Page Offset → Used as an offset within the page frame

Each Page Table contains, the various information about every page of the segment. The Segment Table contains the information about every segment. Each segment table entry points to a page table entry and every page table entry is mapped to one of the page within a segment.



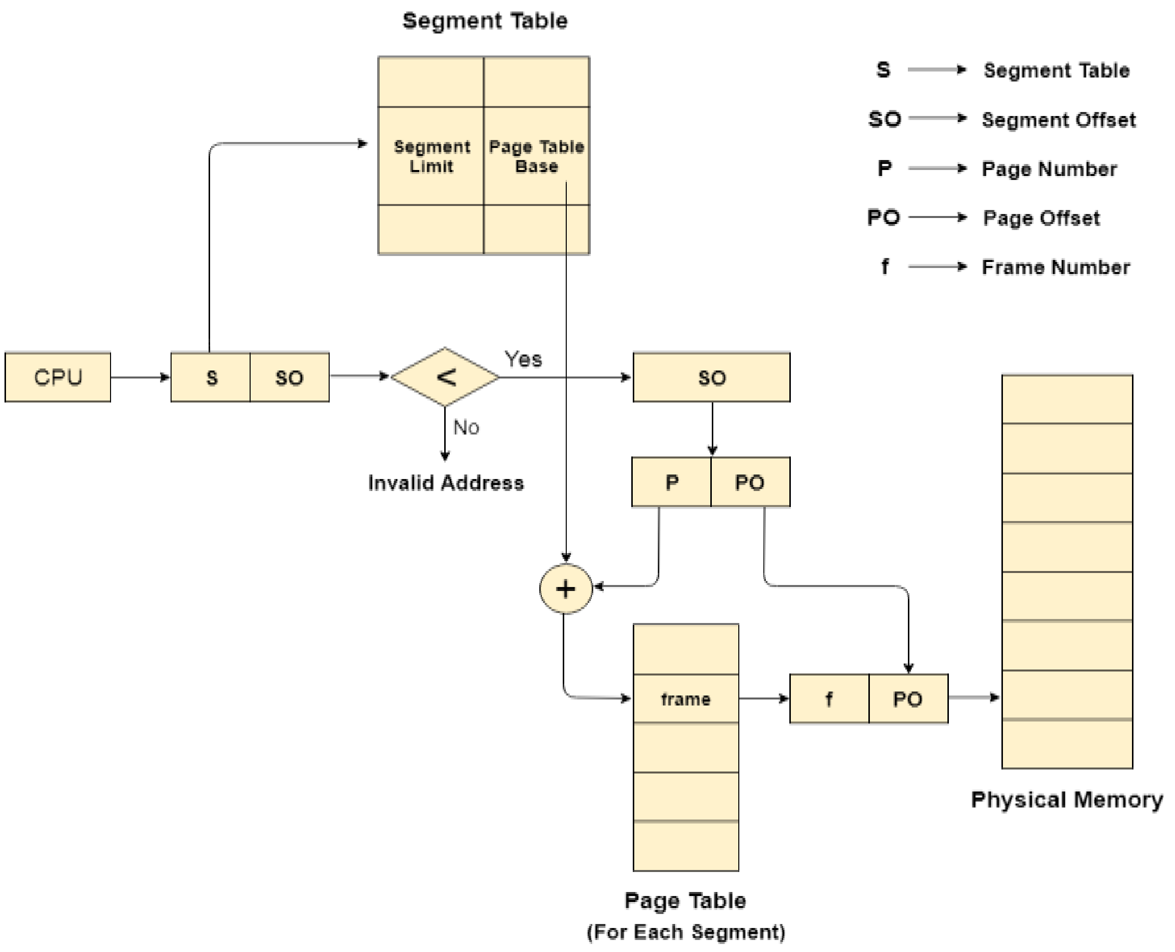
www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

Translation of logical address to physical address

The CPU generates a logical address which is divided into two parts: Segment Number and Segment Offset. The Segment Offset must be less than the segment limit. Offset is further divided into Page number and Page Offset. To map the exact page number in the page table, the page number is added into the page table base.

The actual frame number with the page offset is mapped to the main memory to get the desired word in the page of the certain segment of the process.



Advantages of Segmented Paging

- It reduces memory usage.
- Page table size is limited by the segment size.
- Segment table has only one entry corresponding to one actual segment.
- External Fragmentation is not there.
- It simplifies memory allocation.

Disadvantages of Segmented Paging

- Internal Fragmentation will be there.
- The complexity level will be much higher as compare to paging.
- Page Tables need to be contiguously stored in the memory.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

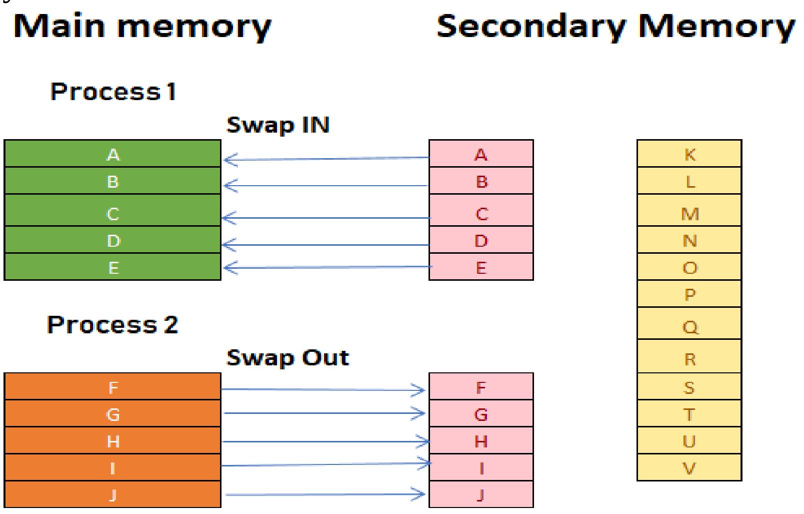
www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

DEMAND PAGING

Demand paging can be described as a memory management technique that is used in operating systems to improve memory usage and system performance. **Demand paging** is a technique used in virtual memory systems where pages enter main memory only when requested or needed by the CPU.

In demand paging, the operating system loads only the necessary pages of a program into memory at runtime, instead of loading the entire program into memory at the start.

A page fault occurred when the program needed to access a page that is not currently in memory. The operating system then loads the required pages from the disk into memory and updates the page tables accordingly. This process is transparent to the running program and it continues to run as if the page had always been in memory.



A demand paging mechanism is very much similar to a paging system with swapping where processes stored in the secondary memory and pages are loaded only on demand, not in advance.

So, when a context switch occurs, the OS never copy any of the old program's pages from the disk or any of the new program's pages into the main memory. Instead, it will start executing the new program after loading the first page and fetches the program's pages, which are referenced.

During the program execution, if the program references a page that may not be available in the main memory because it was swapped, then the processor considers it as an invalid memory reference. That's because the page fault and transfers send control back from the program to the OS, which demands to store page back into the memory.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

VIRTUAL MEMORY

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of the main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites and program-generated addresses are translated automatically to the corresponding machine addresses.

The size of virtual storage is limited by the addressing scheme of the computer system and the amount of secondary memory available not by the actual number of main storage locations.

It is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.

All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of the main memory such that it occupies different places in the main memory at different times during the course of execution.

A process may be broken into a number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time address translation and the use of a page or segment table permits this.

If these characteristics are present then, it is not necessary that all the pages or segments are present in the main memory during execution. This means that the required pages need to be loaded into memory whenever required. Virtual memory is implemented using Demand Paging or Demand Segmentation.

PURE DEMAND PAGING

Pure demand paging is a specific implementation of demand paging. The operating system only loads pages into memory when the program needs them. In on-demand paging only, no pages are initially loaded into memory when the program starts, and all pages are initially marked as being on disk.

Benefits of the Demand Paging

So in the Demand Paging technique, there are some benefits that provide efficiency of the operating system.

- **Efficient use of physical memory**: Query paging allows for more efficient use because only the necessary pages are loaded into memory at any given time.
- **Support for larger programs:** Programs can be larger than the physical memory available on the system because only the necessary pages will be loaded into memory.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

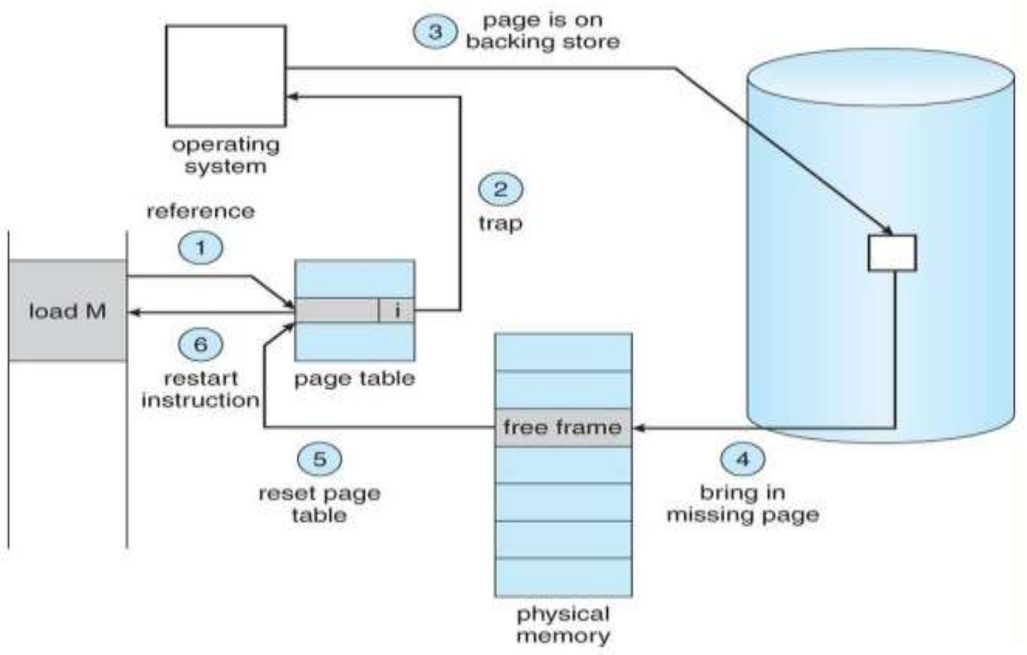
- **Faster program start:** Because only part of a program is initially loaded into memory, programs can start faster than if the entire program were loaded at once.
- **Reduce memory usage:** Query paging can help reduce the amount of memory a program needs, which can improve system performance by reducing the amount of disk I/O required.

Drawbacks of the Demand Paging

- **Page Fault Overload:** The process of swapping pages between memory and disk can cause a performance overhead, especially if the program frequently accesses pages that are not currently in memory.
- **Degraded performance:** If a program frequently accesses pages that are not currently in memory, the system spends a lot of time swapping out pages, which degrades performance.
- **Fragmentation:** Query paging can cause physical memory fragmentation, degrading system performance over time.
- **Complexity:** Implementing query paging in an operating system can be complex, requiring complex algorithms and data structures to manage page tables and swap space.

Working Process of Demand Paging

Suppose we want to run a process P which has four pages P0, P1, P2, and P3. Currently, in the page table, we have pages P1 and P3. So there are some steps that are followed in the working process of the demand paging in the operating system.



A Page Fault happens when you access a page that has been marked as invalid. The paging hardware would notice that the invalid bit is set while translating the

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

address across the page table, which will cause an operating system trap. The trap is caused primarily by the OS's failure to load the needed page into memory.

Procedure of page fault handling

1. Firstly, an internal table for this process to assess whether the reference was valid or invalid memory access.
2. If the reference becomes invalid, the system process would be terminated. Otherwise, the page will be paged in.
3. After that, the free-frame list finds the free frame in the system.
4. Now, the disk operation would be scheduled to get the required page from the disk.
5. When the I/O operation is completed, the process's page table will be updated with a new frame number, and the invalid bit will be changed. Now, it is a valid page reference.
6. If any page fault is found, restart these steps from starting.

Page Hit

When the CPU attempts to obtain a needed page from main memory and the page exists in main memory (RAM), it is referred to as a **"Page Hit"**.

Page Miss

If the needed page has not existed in the main memory (RAM), it is known as **"Page Miss"** or **“Page Fault”.**

Page Fault Time

The time it takes to get a page from secondary memory and recover it from the main memory after loading the required page is known as **"Page Fault Time"**.

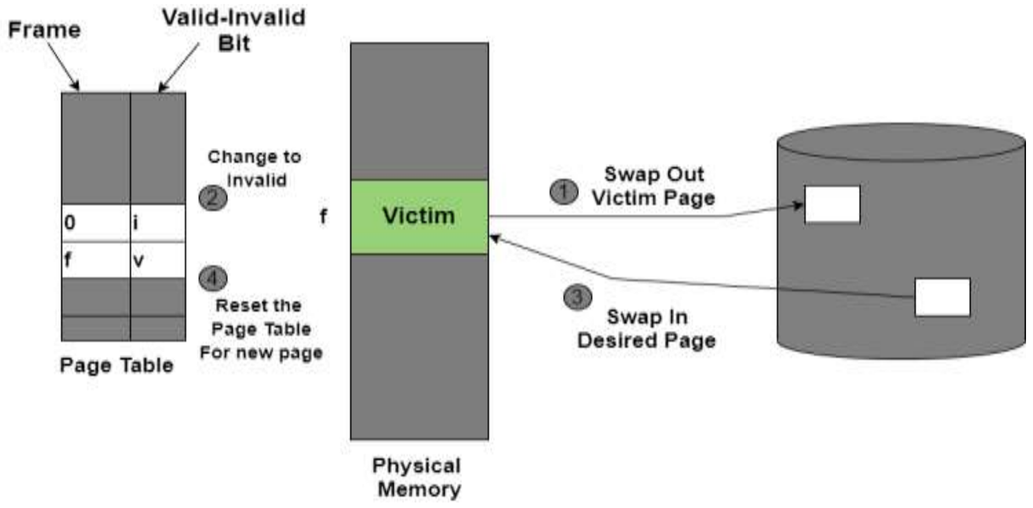
www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

BASIC PAGE REPLACEMENT ALGORITHM

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.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

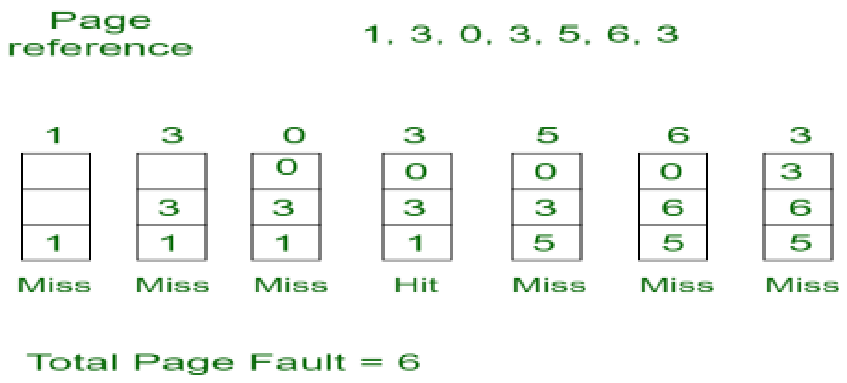
www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

(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.



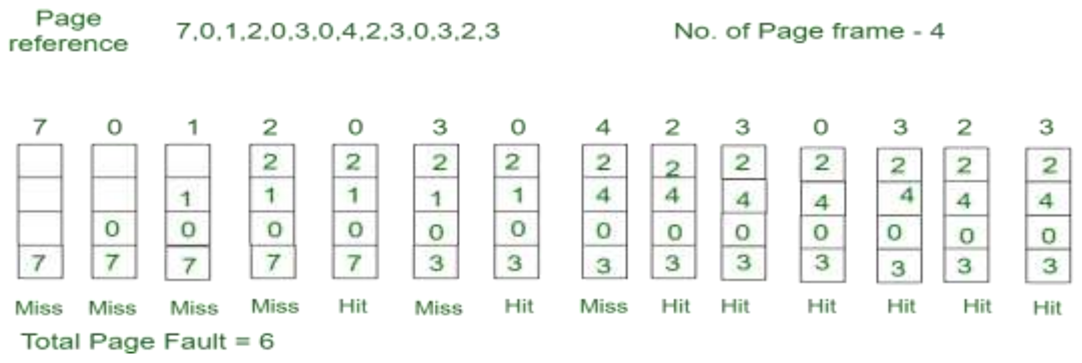
Initially, all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.** When 3 comes, it is already in memory so **No Page Faults.** Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. When 6 comes, it is also not available in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



www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

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.

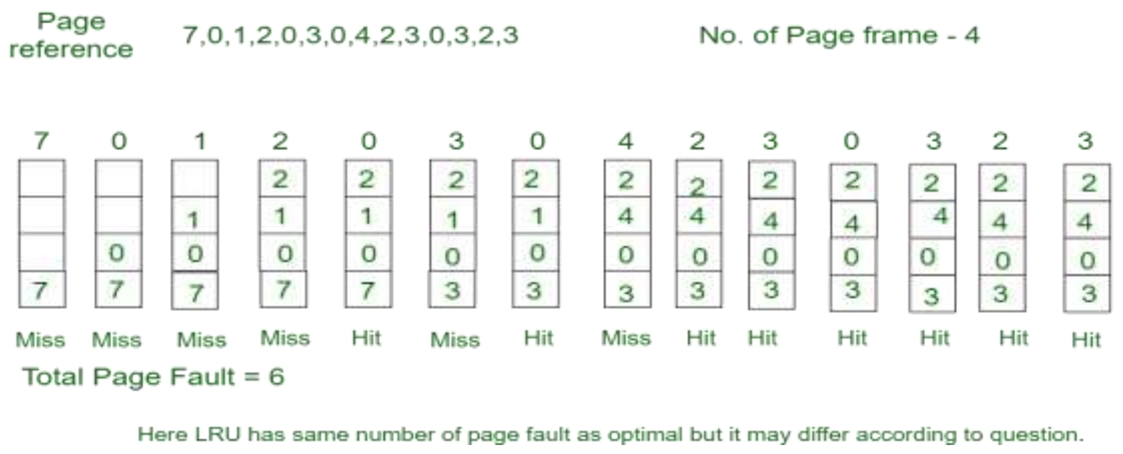
Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

(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.



Page reference
7,0,1,2,0,3,0,4,2,3,0,3,2,3
No. of Page frame - 4
7 0 1 2 0 3 0 4 2 3 0 3 2 3
Miss Miss Miss Miss Hit Miss Hit Miss Hit Hit Hit Hit Hit Hit
Total Page Fault = 6
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.

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

Belady's Anomaly

Generally, on increasing the number of frames to a process virtual memory, its execution becomes faster as fewer page faults occur. Sometimes the reverse happens, i.e. more page faults occur when more frames are allocated to a process. This most unexpected result is termed **Belady's Anomaly**.

Belady's Anomaly is the name given to the phenomenon where increasing the number of page frames results in an increase in the number of page faults for a given memory access pattern.

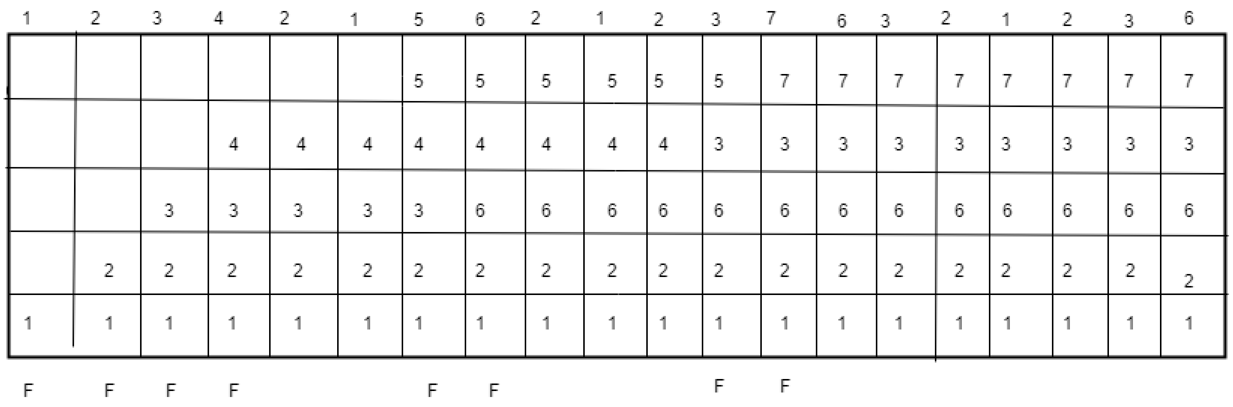
Question 1:

Consider the following page reference string:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

Calculate the number of page faults related to LRU, FIFO and optimal page replacement algorithms. Assume 5 page frames and all frames are initially empty.

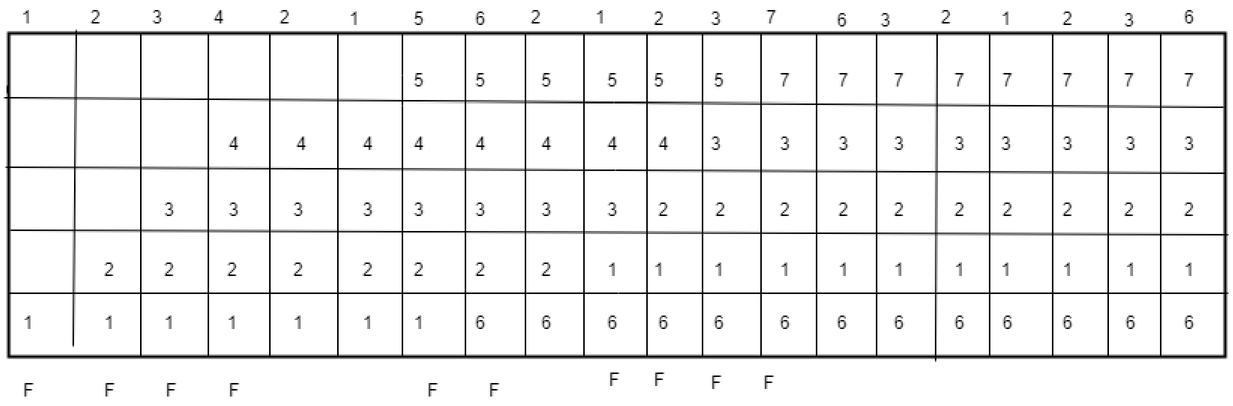
In LRU:



1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
						5	5	5	5	5	5	7	7	7	7	7	7	7	7
			4	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3
		3	3	3	3	3	6	6	6	6	6	6	6	6	6	6	6	6	6
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
F	F	F	F			F	F				F	F							

No. of Page fault = 8

In FIFO:



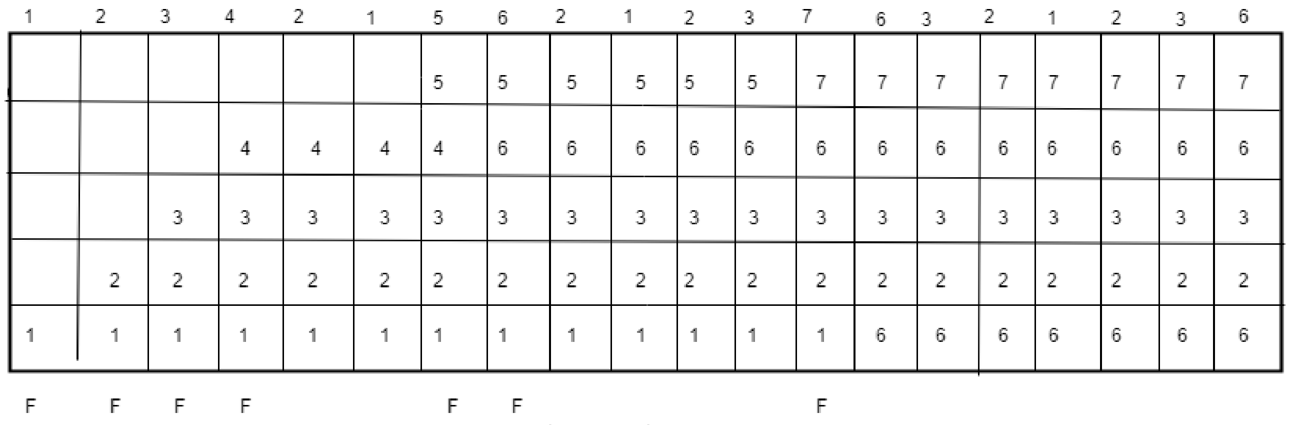
1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
						5	5	5	5	5	5	7	7	7	7	7	7	7	7
			4	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3
		3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2
	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	6	6	6	6	6	6	6	6	6	6	6	6	6
F	F	F	F			F	F		F	F	F	F							

No. of Page fault = 10

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

In Optimal:



1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
						5	5	5	5	5	5	7	7	7	7	7	7	7	7
			4	4	4	4	6	6	6	6	6	6	6	6	6	6	6	6	6
		3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
1	1	1	1	1	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6
F	F	F	F			F	F					F							

No. of Page fault = 7

Page Replacement Algorithm	No. of Page Fault
FIFO	10
LRU	8
Optimal	7

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh

www.android.universityupdates.in | www.universityupdates.in | https://telegram.me/jntuh

Question 2:

Consider the following page reference string:

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

How many page faults would occur for the following replacement algorithms, assuming one, two, three and four frames?

(i) LRU replacement
(ii) FIFO replacement
(iii) Optimal replacement

Remember that all frames are initially empty, so your first unique pages will all cost one fault each.

Solution

Frame : 1 No. of page fault is **20** for LRU, FIFO and Optimal page replacement.

Frame : 2

Ref. String	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6	**No.of Page Fault**
FIFO	1	1	3	3	2	2	5	5	2	2		3	3	6	6	2	2		3	3	**18**
		2	2	4	4	1	1	6	6	1		1	7	7	3	3	1		1	6	
LRU	1	1	3	3	2	2	5	5	2	2		2	7	7	3	3	1		3	3	**18**
		2	2	4	4	1	1	6	6	1		3	3	6	6	2	2		2	6	
Optimal	1	1	3	4		1	5	6		1		3	3	3		3	1		1	6	**15**
		2	2	2		2	2	2		2		2	7	6		2	2		3	3	

Frame : 3

Ref. String	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6	
FIFO	1	1	1	4		4	4	6	6	6		3	3	3		2	2		2		**15**
		2	2	2		1	1	1	2	2		2	7	7		7	1		1		
			3	3		3	5	5	5	1		1	1	6		6	6		6		
LRU	1	1	1	4		4	5	5	5	1		1	7	7		2	2			2	**15**
		2	2	2		2	2	6	6	6		3	3	3		3	3			3	
			3	3		1	1	1	2	2		2	2	6		6	1			6	
Optimal	1	1	1	1			1	1				3	3			3	3	3			**11**
		2	2	2			2	2				2	7			2	1	2			
			3	4			5	6				6	6			6	6	6			

Frame : 4

Ref. String	1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6	
FIFO	1	1	1	1			5	5	5	5		3	3	3		3	1		1		**14**
		2	2	2			2	6	6	6		6	7	7		7	7		3		
			3	3			3	3	2	2		2	2	6		6	6		6		
				4			4	4	4	1		1	1	1		2	2		2		
LRU	1	1	1	1			1	1				1	1	6			6				**10**
		2	2	2			2	2				2	2	2			2				
			3	3			5	5				3	3	3			3				
				4			4	6				6	7	7			1				
Optimal	1	1	1	1			1	1					7				1				**8**
		2	2	2			2	2					2				2				
			3	3			3	3					3				3				
				4			5	6					6				6				

www.android.previousquestionpapers.com | www.previousquestionpapers.com | https://telegram.me/jntuh