Unit-iii MEMORY MANAGEMENT

UNIT-III:
Memory Management: Swapping, Contiguous Memory Allocation, Paging, structure of the Page Table, Segmentation
Virtual Memory Management:
Virtual Memory, Demand Paging, Page-Replacement Algorithms, Thrashing

TOPIC-1 Memory management Concepts

is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

This tutorial will teach you basic concepts related to Memory Management.

Memory Management

- Memory consists of a large array of words or bytes, each with its own address. The CPU fetches instructions from memory according to the value of the program counter. These instructions may cause additional loading from and storing to specific memory addresses.
- Memory unit sees only a stream of memory addresses. It does not know how they are generated.
- Program must be brought into memory and placed within a process for it to be run.
- Input queue – collection of processes on the disk that are waiting to be brought into memory for execution.
- User programs go through several steps before being run.

Memory Management is the process of controlling and coordinating computer memory, assigning portions known as blocks to various running programs to optimize the overall performance of the system.

It is the most important function of an operating system that manages primary memory. It helps processes to move back and forward between the main memory and execution disk. It helps OS to keep track of every memory location, irrespective of whether it is allocated to some process or it remains free.

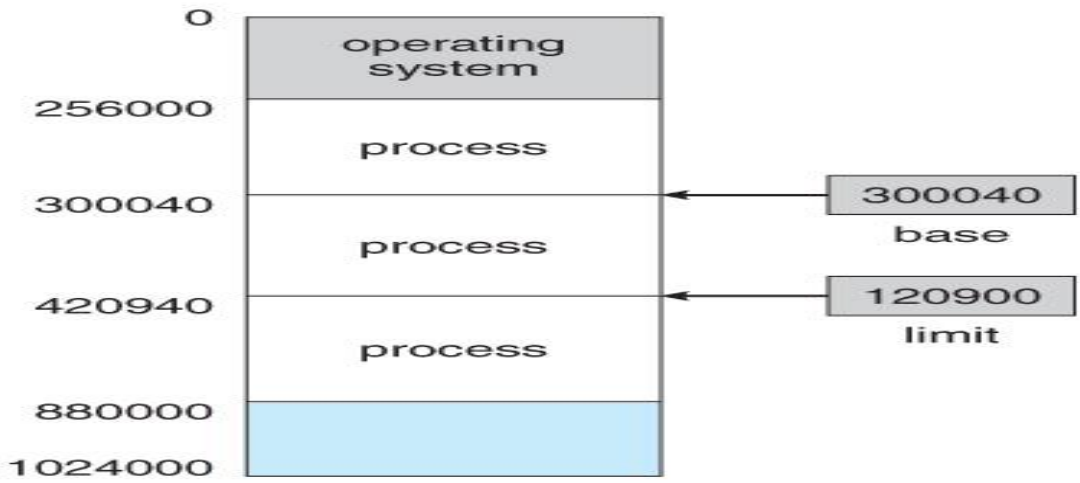
- It allows you to check how much memory needs to be allocated to processes that decide which processor should get memory at what time.
- Tracks whenever inventory gets freed or unallocated. According to it will update the status.
- It allocates the space to application routines.
- It also make sure that these applications do not interfere with each other.
- Helps protect different processes from each other
- It places the programs in memory so that memory is utilized to its full extent.
- Obviously memory accesses and memory management are a very important part of modern computer operation. Every instruction has to be fetched from memory before it can be executed, and most instructions involve retrieving data from memory or storing data in memory or both.

- The advent of multi-tasking OSes compounds the complexity of memory management, because because as processes are swapped in and out of the CPU, so must their code and data be swapped in and out of memory, all at high speeds and without interfering with any other processes.
- Shared memory, virtual memory, the classification of memory as read-only versus read-write, and concepts like copy-on-write forking all further complicate the issue.

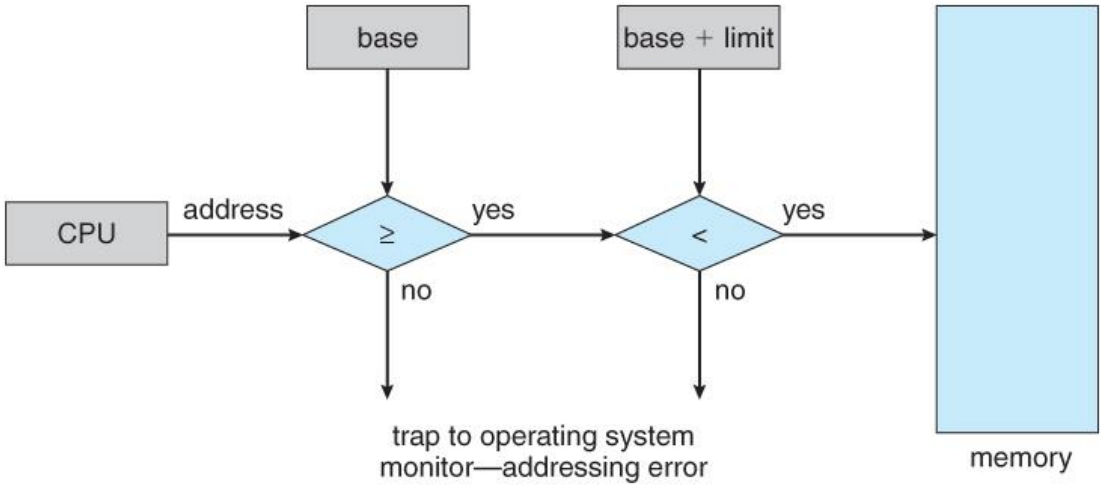
1 Basic Hardware

- It should be noted that from the memory chips point of view, all memory accesses are equivalent. The memory hardware doesn't know what a particular part of memory is being used for, nor does it care. This is almost true of the OS as well, although not entirely.
- The CPU can only access its registers and main memory. It cannot, for example, make direct access to the hard drive, so any data stored there must first be transferred into the main memory chips before the CPU can work with it. (Device drivers communicate with their hardware via interrupts and "memory" accesses, sending short instructions for example to transfer data from the hard drive to a specified location in main memory. The disk controller monitors the bus for such instructions, transfers the data, and then notifies the CPU that the data is there with another interrupt, but the CPU never gets direct access to the disk.)
- Memory accesses to registers are very fast, generally one clock tick, and a CPU may be able to execute more than one machine instruction per clock tick.
- Memory accesses to main memory are comparatively slow, and may take a number of clock ticks to complete. This would require intolerable waiting by the CPU if it were not for an intermediary fast memory **cache** built into most modern CPUs. The basic idea of the cache is to transfer chunks of memory at a time from the main memory to the cache, and then to access individual memory locations one at a time from the cache.
- User processes must be restricted so that they only access memory locations that "belong" to that particular process. This is usually implemented using a base register and a limit register for each process, as shown in Figures 8.1 and 8.2 below. **Every** memory access made by a user process is checked against these two registers, and if a memory access is attempted outside the valid range, then a fatal error is generated. The OS obviously has access to all existing memory locations, as this is necessary to swap users' code and data in

and out of memory. It should also be obvious that changing the contents of the base and limit registers is a privileged activity, allowed only to the OS kernel.



A base and a limit register define a logical addresss space



Hardware address protection with base and limit registers

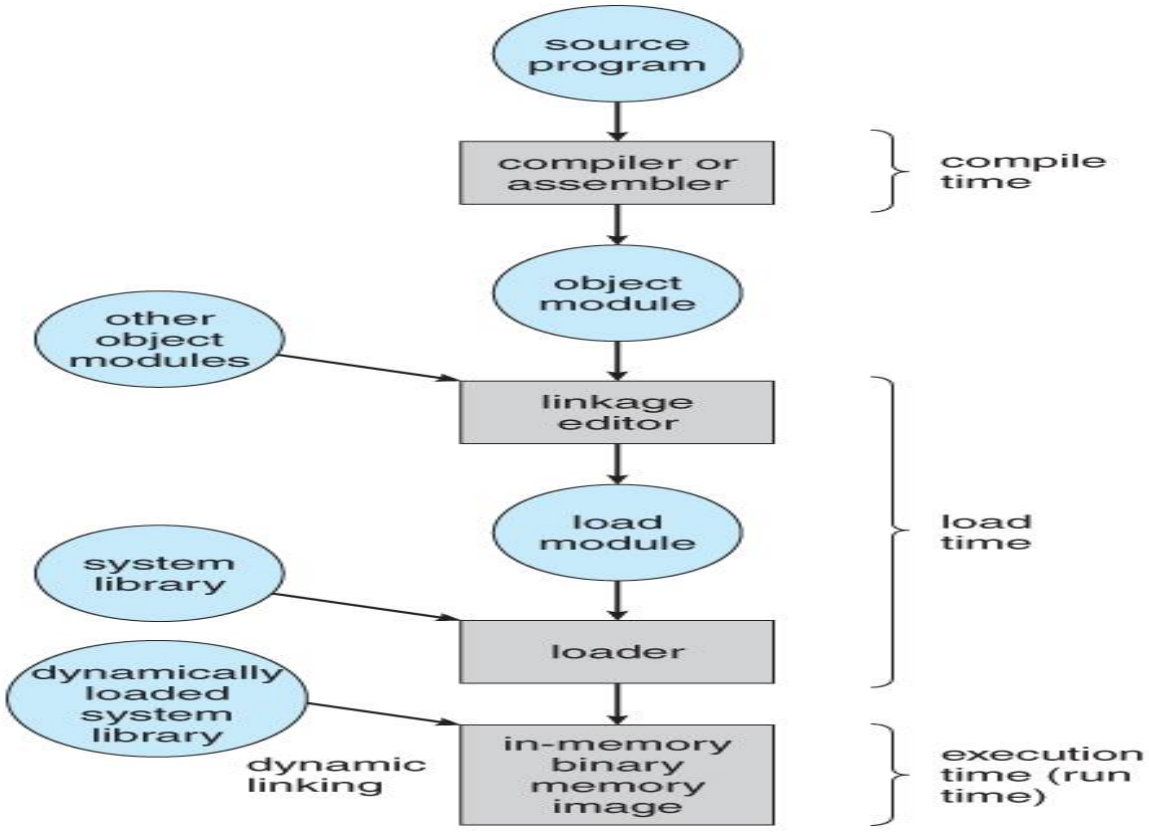
Address Binding

- User programs typically refer to memory addresses with symbolic names such as "i", "count", and "averageTemperature". These symbolic names must be mapped or **bound** to physical memory addresses, which typically occurs in several stages:
  - **Compile Time** - If it is known at compile time where a program will reside in physical memory, then **absolute code** can be generated by the compiler, containing actual physical addresses. However if the load address changes at

some later time, then the program will have to be recompiled. DOS .COM programs use compile time binding.

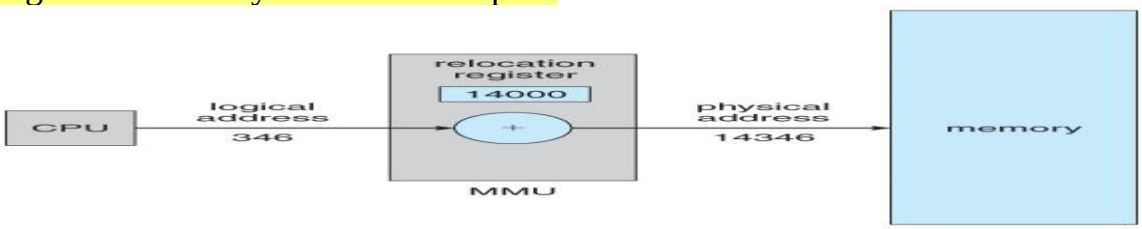
- **Load Time** - If the location at which a program will be loaded is not known at compile time, then the compiler must generate **relocatable code**, which references addresses relative to the start of the program. If that starting address changes, then the program must be reloaded but not recompiled.
- **Execution Time** - If a program can be moved around in memory during the course of its execution, then binding must be delayed until execution time. This requires special hardware, and is the method implemented by most modern OSes.

- Figure 8.3 shows the various stages of the binding processes and the units involved in each stage:



Multistep processing of a user program

Logical Versus Physical Address Space



Dynamic relocation using a relocation register

Logical Versus Physical Address Space

- The concept of a logical address space that is bound to a separate physicaladdress space is central to proper memory management.
  - Logical address – address generated by the CPU; also referred to as virtual address.
  - Physical address – address seen by the memory unit.
- The set of all logical addresses generated by a program is a logical address space; the set of all physical addresses corresponding to these logical addresses are a physical address space.

- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme.

Logical And Physical Addresses

An address generated by the CPU is commonly refereed as **Logical Address**, whereas the address seen by the memory unit that is one loaded into the memory address register of the memory is commonly refereed as the **Physical Address**. The compile time and load time address binding generates the identical **logical and physical addresses**. However, the execution time addresses binding scheme results in differing **logical and physical addresses**.

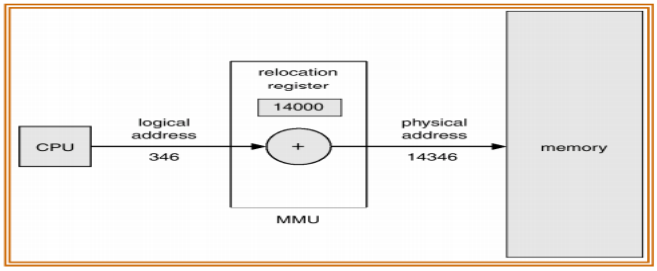
The set of all **logical addresses** generated by a program is known as **Logical Address Space**, where as the set of all **physical addresses** corresponding to these logical addresses is **Physical Address Space**. Now, the run time mapping from virtual address to **physical address** is done by a hardware device known as **Memory Management Unit**. Here in the case of mapping the base register is known as **relocation register**. The value in the relocation register is added to the address generated by a user process at the time it is sent to memory .Let's understand this situation with the help of example: If the base register contains the value 1000,then an attempt by the user to address location 0 is dynamically relocated to location 1000,an access to location 346 is mapped to location 1346.

Memory-Management Unit (MMU)

Hardware device that maps virtual to physical address

- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
- The user program deals with *logical* addresses; it never sees the *real* physical addresses

- The run-time mapping from virtual to physical addresses is done by a hardware device called the memory management unit (MMU).

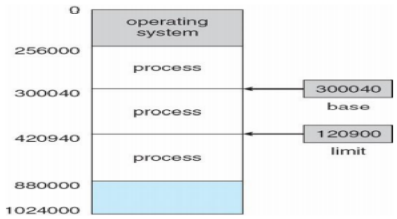


- This method requires hardware support slightly different from the hardware configuration. The base register is now called a relocation register. The value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
- The user program never sees the real physical addresses. The program can create a pointer to location 346, store it in memory, manipulate it and compare it to other addresses. The user program deals with logical addresses. The memory mapping hardware converts logical addresses into physical addresses. The final location of a referenced memory address is not determined until the reference is made.

The user program never sees the **real physical address** space, it always deals with the **Logical addresses**. As we have two different type of addresses **Logical address** in the range (0 to max) and **Physical addresses** in the range(R to R+max) where R is the value of relocation register. The user generates only **logical addresses** and thinks that the process runs in location to 0 to max. As it is clear from the above text that user program supplies only logical addresses, these **logical addresses** must be mapped to **physical address** before they are used.

Base and Limit Registers

A pair of **base** and **limit** registers define the logical address space



HARDWARE PROTECTION WITH BASE AND LIMIT

Dynamic Loading

- Rather than loading an entire program into memory at once, dynamic loading loads up each routine as it is called. The advantage is that unused routines need never be loaded, reducing total memory usage and generating faster program startup times. The downside is the added complexity and overhead of checking to see if a routine is loaded every time it is called and then then loading it up if it is not already loaded.

Dynamic Linking and Shared Libraries

- With **static linking** library modules get fully included in executable modules, wasting both disk space and main memory usage, because every program that included a certain routine from the library would

have to have their own copy of that routine linked into their executable code.

- With **dynamic linking**, however, only a stub is linked into the executable module, containing references to the actual library module linked in at run time.
    - This method saves disk space, because the library routines do not need to be fully included in the executable modules, only the stubs.
    - We will also learn that if the code section of the library routines is **reentrant**, (meaning it does not modify the code while it runs, making it safe to re-enter it), then main memory can be saved by loading only one copy of dynamically linked routines into memory and sharing the code amongst all processes that are concurrently using it. (Each process would have their own copy of the **data** section of the routines, but that may be small relative to the code segments.) Obviously the OS must manage shared routines in memory.
    - An added benefit of **dynamically linked libraries** (**DLLs**, also known as **shared libraries** or **shared objects** on UNIX systems) involves easy upgrades and updates. When a program uses a routine from a standard library and the routine changes, then the program must be re-built (re-linked) in order to incorporate the changes. However if DLLs are used, then as long as the stub doesn't change, the program can be updated merely by loading new versions of the DLLs onto the system. Version information is maintained in both the program and the DLLs, so that a program can specify a particular version of the DLL if necessary.
    - In practice, the first time a program calls a DLL routine, the stub will recognize the fact and will replace itself with the actual routine from the DLL library. Further calls to the same routine will access the routine directly and not incur the overhead of the stub access. (Following the UML **Proxy Pattern**.)
    - (Additional information regarding dynamic linking is available a

Dynamic Linking

- Linking is postponed until execution time.
- Small piece of code, stub, is used to locate the appropriate memory-resident library routine, or to load the library if the routine is not already present.
- When this stub is executed, it checks to see whether the needed routine is already in memory. If not, the program loads the routine into memory.
- Stub replaces itself with the address of the routine, and executes the routine.
- Thus the next time that code segment is reached, the library routine is executed directly, incurring no cost for dynamic linking.
- Operating system is needed to check if routine is in processes' memory address.
- Dynamic linking is particularly useful for libraries.

Dynamic Linking

- Linking postponed until execution time
- Small piece of code, *stub*, used to locate the appropriate memory-resident library
- routine Stub replaces itself with the address of the routine, and executes the routine
- Operating system needed to check if routine is in processes' memory address Dynamic
- linking is particularly useful for libraries
- System also known as **shared libraries**

TOPIC-2 Swapping

Swapping

- A process must be loaded into memory in order to execute.
- If there is not enough memory available to keep all running processes in memory at the same time, then some processes who are not currently using the CPU may have their memory swapped out to a fast local disk called the **backing store.**
- 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**.

- If compile-time or load-time address binding is used, then processes must be swapped back into the same memory location from which they were swapped out. If

execution time binding is used, then the processes can be swapped back into any available location.

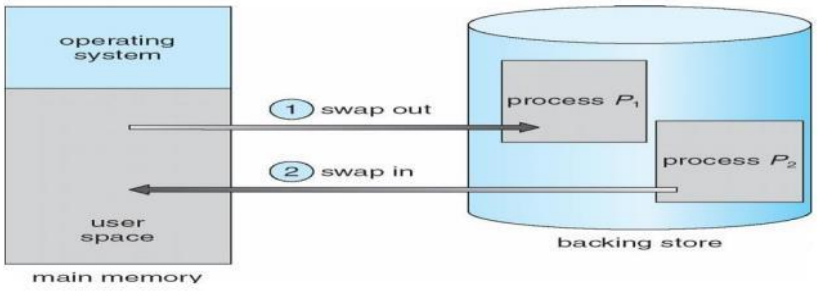
- Swapping is a very slow process compared to other operations. For example, if a user process occupied 10 MB and the transfer rate for the backing store were 40 MB per second, then it would take 1/4 second (250 milliseconds) just to do the data transfer. Adding in a latency lag of 8 milliseconds and ignoring head seek time for the moment, and further recognizing that swapping involves moving old data out as well as new data in, the overall transfer time required for this swap is 512 milliseconds, or over half a second. For efficient processor scheduling the CPU time slice should be significantly longer than this lost transfer time.
- To reduce swapping transfer overhead, it is desired to transfer as little information as possible, which requires that the system know how much memory a process **is** using, as opposed to how much it **might** use. Programmers can help with this by freeing up dynamic memory that they are no longer using.
- It is important to swap processes out of memory only when they are idle, or more to the point, only when there are no pending I/O operations. (Otherwise the pending I/O operation could write into the wrong process's memory space.) The solution is to either swap only totally idle processes, or do I/O operations only into and out of OS buffers, which are then transferred to or from process's main memory as a second step.
- Most modern OSes no longer use swapping, because it is too slow and there are faster alternatives available. (e.g. Paging.) However some UNIX systems will still invoke swapping if the system gets extremely full, and then discontinue swapping when the load reduces again. Windows 3.1 would use a modified version of swapping that was somewhat controlled by the user, swapping process's out if necessary and then only swapping them back in when the user focused on that particular window.

Swapping

A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped and Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

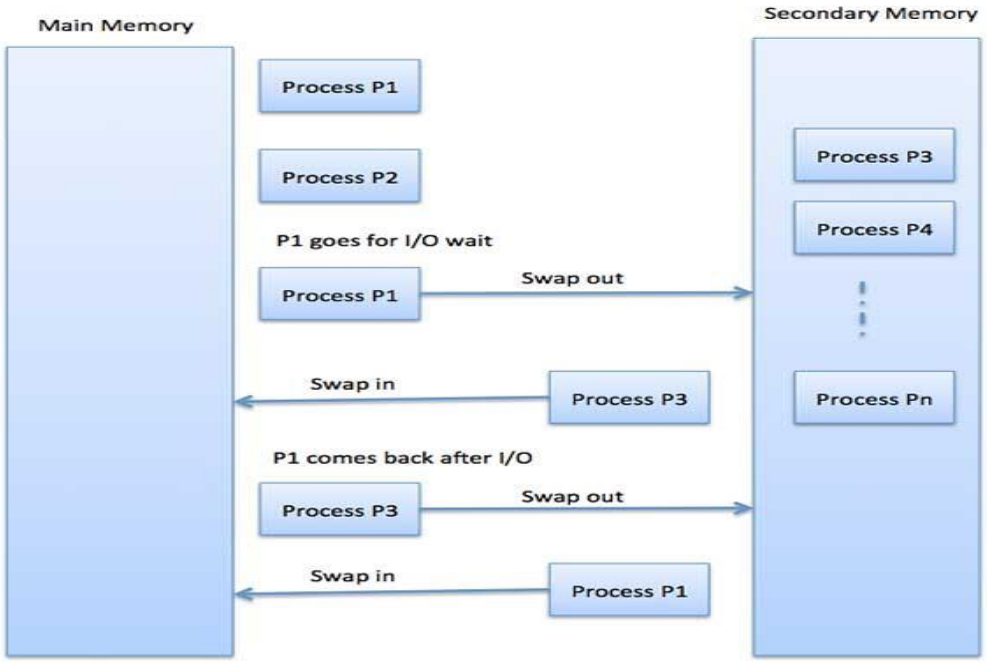
System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Schematic View of Swapping



Swapping of two processes using a disk as a backing store

Or for better understanding



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.

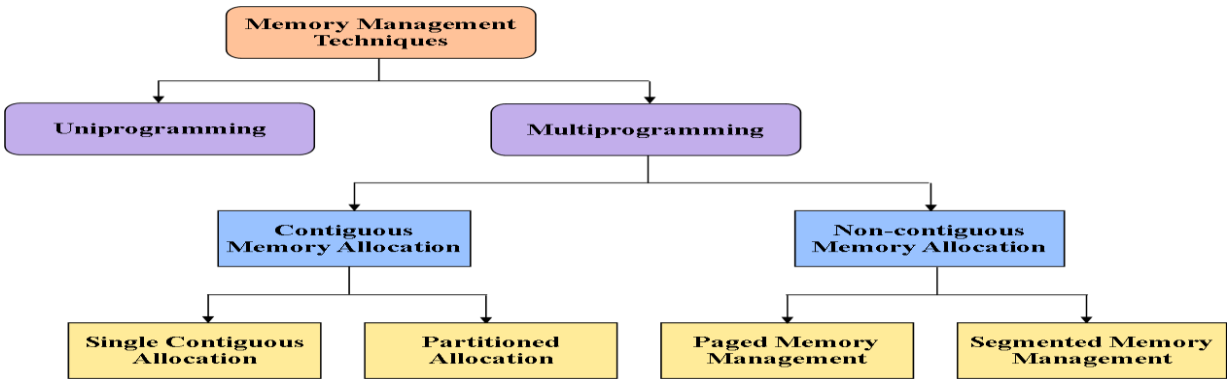
Benefits of Swapping

Here, are major benefits/pros of swapping:

- It offers a higher degree of multiprogramming.
- Allows dynamic relocation. For example, if address binding at execution time is being used, then processes can be swap in different locations. Else in case of compile and load time bindings, processes should be moved to the same location.
- It helps to get better utilization of memory.
- Minimum wastage of CPU time on completion so it can easily be applied to a priority-based scheduling method to improve its performance.

Topic-3MemoryManagement Techniques In Operating System

Memory is the focal piece of the considerable number of operations of a computer system. So in this section, we will learn out about the various kinds of memory management techniques and furthermore the advantages and disadvantages of various memory management techniques. Figure 1 represents to the absolute most pivotal memory management techniques:

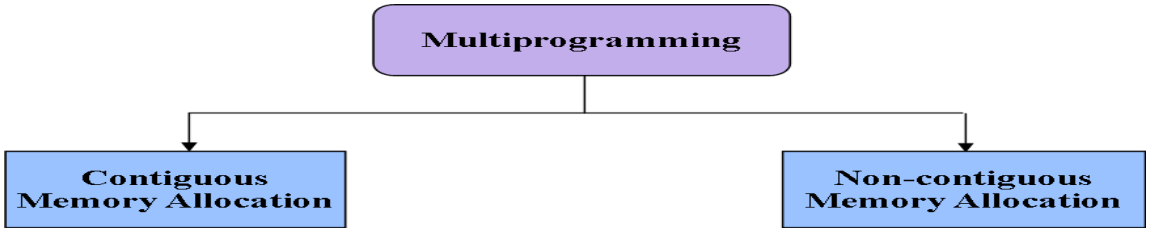


Uni-programming memory management:

In uni-programming technique, the RAM is isolated into two categories where's one category is for leaving the operating system and the other category is for the client process. Here the fence register is utilized which contains the last address of the parts of operating system. The operating system will contrast the client information addresses and the fence register and in the event that it is distinctive that implies the client isn't entering the area of operating system. The fence register is additionally called a limit register and is utilized to keep a client from entering the operating system region. Here the CPU use is poor and thus multiprogramming is utilized.

Multi-programming memory management

In the multi-programming, the various clients can share the memory at the same time. By multiprogramming we mean there will be more than one procedure in the main memory and if the running procedure needs to hang tight for an occasion like input/output then as opposed to sitting on ideal condition CPU will do a context switch and will pick another procedure.

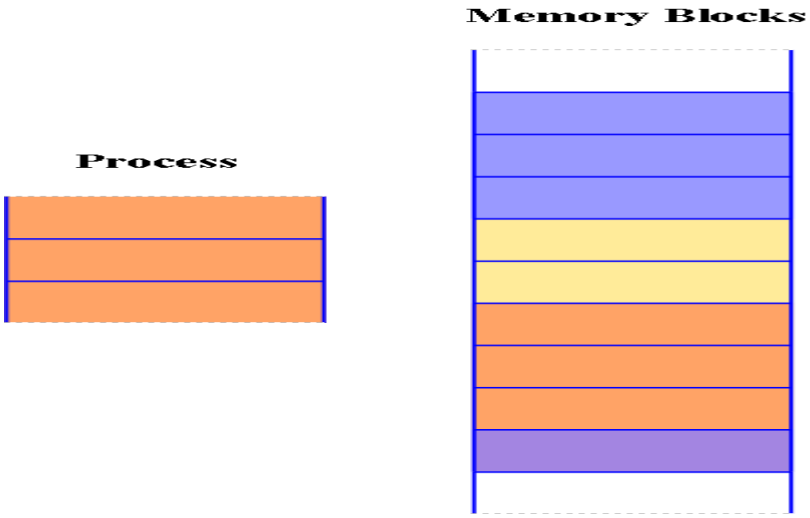


Types of multiprogramming memory management techniques

Contiguous memory allocation

In the procedure of contiguous memory allocation, all the accessible memory space stays together in one spot. It implies openly accessible memory partitions are not dissipated to a whole extent over the entire memory space.

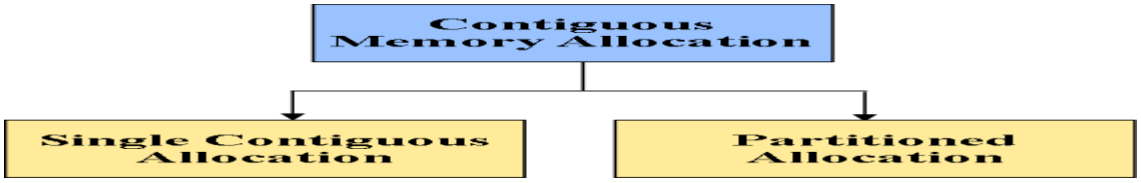
In the contiguous memory allocation, both operating system and the client must live in the primary memory. The primary memory is isolated into two segments where's one segment is for the operations and other is for the client program.



Contiguous Memory Allocation

In the contiguous memory allocation when any client procedure demands the memory a solitary segment of the contiguous memory block is given to that procedure as indicated by its need. We can accomplish the contiguous memory allocation by separating memory into the fixed-sized partition.

A solitary procedure is distributed in that fixed-sized single partition. Yet, this will build the level of multiprogramming that implies more than one procedure in the principle memory that limits the quantity of fixed partition done in memory. Internal fragmentation expands in light of the contiguous memory allocation.



Types of Contiguous Memory Management Technique

Single Contiguous memory Allocation or fixed sized partition:

It is the most effortless memory management technique. In this strategy, a wide range of computer memory aside from a little part which is held for the working framework is accessible for one application. In other words it is also known as fixed sized partition of the system that separates memory into fixed-size segments (might possibly be of a similar size). In this whole partition is permitted to a procedure and if there is some wastage inside the segment is apportioned to a procedure and if there is some wastage inside the segment, at that point it is called an internal fragmentation.

For instance, the MS-DOS operating system designates memory along these lines. An embedded system likewise runs on a solitary application.

- Advantage: Management or accounting is simple.
- Disadvantage: Occurrence of Internal fragmentation.

Also check: Memory Fragmentation in Operating System

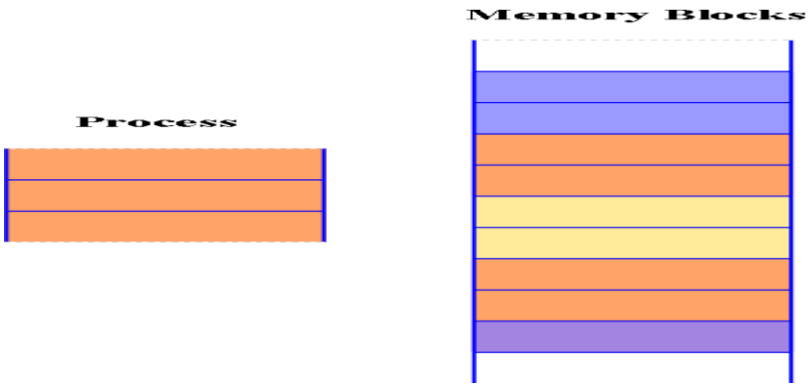
Partitioned Allocation

It is also called variable size segments/partitions. In which system isolates essential memory into different memory segments, which are generally adjacent classes of memory. Each segment stores all the data for a particular task/job. This technique comprises allocating a segment to occupation when it begins and unallocated when it closes. In the variable size partition, the memory is treated as one unit, and space allotted to a procedure is actually equivalent to require and the extra space can be reused once more.

- Advantage: This technique is free from internal fragmentation.
- Disadvantage: Management is troublesome as memory is getting absolutely divided after some time.

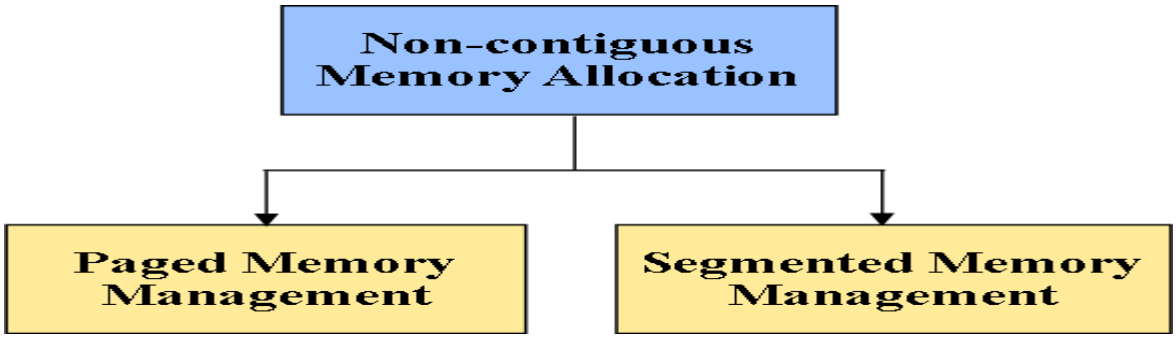
Non-contiguous memory allocation

In the allocation of non-contiguous memory the accessible free memory space is dispersed to a great extent and all the free memory space isn't in one spot. This technique is time-consuming. In the non-contiguous memory allocation, a procedure will procure the memory space however it isn't at one spot it is at the various areas as indicated by the procedure prerequisite. This strategy of non-touching memory allocation reduces the wastage of memory which prompts internal and external fragmentation. This uses all the free memory space which is made by alternate processes.



Non-Contiguous Memory Allocation

Types of non-contiguous memory allocation



Types of non-contiguous memory allocation

Paged Memory Management

This strategy category the computers primary memory into the fixed-size units that is known as page frames. This hardware memory management unit maps pages into frames which ought to be apportioned on a page premise.

- Advantages of paged memory management: It is free of external fragmentation.
- Disadvantages of paged memory management:
    - It makes the interpretation extremely delayed as primary memory get to multiple times.
    - A page table is a burden over the framework which consumes impressive space.

Segmented Memory Management

Management of Segmented memory is the only memory management technique that doesn't provide the client's program with a direct and adjoining address space.

Segments need equipment support as a segment table. It contains the physical address of the area in memory, size, and other information like access assurance bits and status.

Also check: Demand Paging in Operating System

Advantages of segmented memory management technique

- Allow the memory ability to be 1 MB despite the fact that the addresses related with the individual directions are 16 bits wide.
- Allow the utilization of independent memory regions for the program code and information and stack part of the program.
- It permits a program and additionally its information to be put into various areas of memory at whatever point the program is end.
- Multitasking turns out to be simple

Disadvantages of segmented memory management technique:

- Availability of external fragmentation
- Algorithms of memory managements are costly.
- Segmentation discovers free memory areas sufficiently large.
- Paging keeps rundown of free pages.
- Segments of inconsistent size not fit also for trading.

PART-1 CONTNIOUS MEMORY MEMORY ALLOCATION

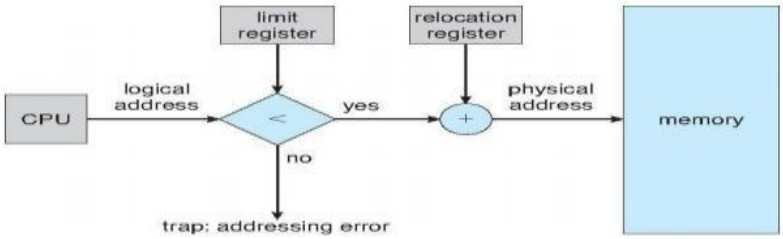
Contiguous Allocation

- Main memory usually into two partitions:
- Resident operating system, usually held in low memory with interrupt vector
- User processes then held in high memorynRelocation registers used to protect user processes from each other, and from changing operating-system code and data

Base register contains value of smallest physical address

- Limit register contains range of logical addresses – each logical address must be less than the limit register
- MMU maps logical address *dynamically*

Hardware Support for Relocation and Limit Registers



- Multiple-partition allocation
- Hole – block of available memory; holes of various size are scattered throughout memory
- When a process arrives, it is allocated memory from a hole large enough to accommodate it

Contiguous memory allocation is one of the efficient ways of allocating main memory to the processes. The memory is divided into two partitions. One for the Operating System and another for the user processes. Operating System is placed in low or high memory depending on the interrupt vector placed. In contiguous memory allocation each process is contained in a single contiguous section of memory.

Memory protection

Memory protection is required to protect Operating System from the user processes and user processes from one another. A relocation register contains the value of the smallest physical address for example say 100040. The limit register contains the range of logical address for example say 74600. Each logical address must be less than limit register. If a logical address is greater than the limit register, then there is an addressing error and it is trapped. The limit register hence offers memory protection.

The MMU, that is, Memory Management Unit maps the logical address dynamically, that is at run time, by adding the logical address to the value in relocation register. This added value is the physical memory address which is sent to the memory.

The CPU scheduler selects a process for execution and a dispatcher loads the limit and relocation registers with correct values. The advantage of relocation register is that it provides an efficient way to allow the Operating System size to change dynamically.

Memory Allocation

Main memory usually has two partitions −

Low Memory − Operating system resides in this memory.

High Memory − User processes are held in high memory.

Operating system uses the following memory allocation mechanism.

S.N.	Memory Allocation & Description
1	**Single-partition allocation** In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.
2	**Multiple-partition allocation** In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

Partition Allocation Methods in Memory Management

In the operating system, the following are four common memory management techniques.

Single contiguous allocation: Simplest allocation method used by MS-DOS. All memory (except some reserved for OS) is available to a process.

Partitioned allocation: Memory is divided into different blocks or partitions. Each process is allocated according to the requirement.

Paged memory management: Memory is divided into fixed-sized units called page frames, used in a virtual memory environment.

Segmented memory management: Memory is divided into different segments (a segment is a logical grouping of the process' data or code).In this management, allocated memory doesn't have to be contiguous.

Most of the operating systems (for example Windows and Linux) use Segmentation with Paging. A process is divided into segments and individual segments have pages.

In **Partition Allocation**, when there is more than one partition freely available to accommodate a process's request, a partition must be selected. To choose a particular partition, a partition allocation method is needed. A partition allocation method is considered better if it avoids internal fragmentation.

When it is time to load a process into the main memory and if there is more than one free block of memory of sufficient size then the OS decides which free block to allocate.

There are different Placement Algorithm:

A. First Fit

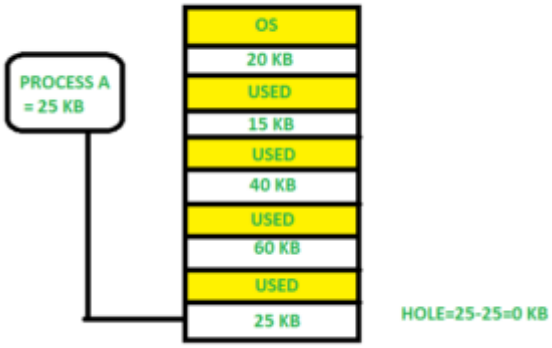
B. Best Fit

C. Worst Fit

D. Next Fit

1. First Fit: In the first fit, the partition is allocated which is the first sufficient block from the top of Main Memory. It scans memory from the beginning and chooses the first available block that is large enough. Thus it allocates the first hole that is large enough.

2. Best Fit Allocate the process to the partition which is the first smallest sufficient partition among the free available partition. It searches the entire list of holes to find the smallest hole whose size is greater than or equal to the size of the process.



3. Worst Fit Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory. It is opposite to the best-fit algorithm. It searches the entire list of holes to find the largest hole and allocate it to process.

4. Next Fit: Next fit is similar to the first fit but it will search for the first sufficient partition from the last allocation point.

Is Best-Fit really best?
Although best fit minimizes the wastage space, it consumes a lot of processor time for searching the block which is close to the required size. Also, Best-fit may perform poorer than other algorithms in some cases. For example, see the below exercise.

Exercise: Consider the requests from processes in given order 300K, 25K, 125K, and 50K. Let there be two blocks of memory available of size 150K followed by a block size 350K.

Which of the following partition allocation schemes can satisfy the above requests?
A) Best fit but not first fit.
B) First fit but not best fit.
C) Both First fit & Best fit.
D) neither first fit nor best fit.

Solution: Let us try all options.
Best Fit:
300K is allocated from a block of size 350K. 50 is left in the block.
25K is allocated from the remaining 50K block. 25K is left in the block.
125K is allocated from 150 K block. 25K is left in this block also.
50K can't be allocated even if there is 25K + 25K space available.

First Fit:
300K request is allocated from 350K block, 50K is left out.
25K is be allocated from the 150K block, 125K is left out.
Then 125K and 50K are allocated to the remaining left out partitions.
So, the first fit can handle requests.

So option B is the correct choice.

Fragmentation

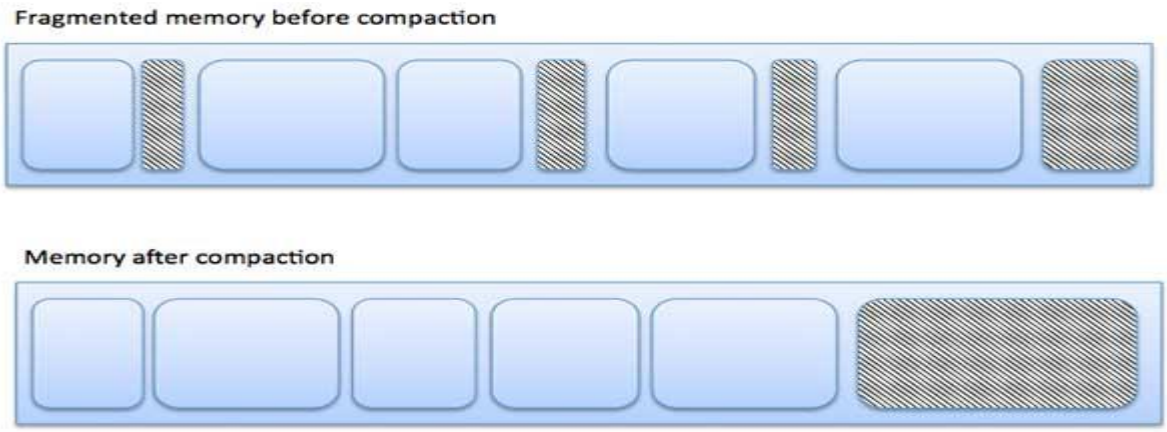
As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types −

S.N.	Fragmentation & Description
1	**External fragmentation** Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it

	cannot be used.
2	**Internal fragmentation** Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory −



External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the

process.

that is Fragmentation and what are its types?

In **contiguous memory allocation** whenever the processes come into RAM, space is allocated to them. These spaces in RAM are divided either on the basis of **fixed partitioning**(the size of partitions are fixed before the process gets loaded into RAM) or **dynamic partitioning** (the size of the partition is decided at the run time according to the size of the process). As the process gets loaded and removed from the memory these spaces get broken into small pieces of memory that it can't be allocated to the coming processes. This problem is called **fragmentation**. In this

blog, we will study how these free space and fragmentations occur in memory. So, let's get started.

Fragmentation

Fragmentation is an unwanted problem where the memory blocks cannot be allocated to the processes due to their small size and the blocks remain unused. It can also be understood as when the processes are loaded and removed from the memory they create free space or hole in the memory and these small blocks cannot be allocated to new upcoming processes and results in inefficient use of memory. Basically, there are two types of fragmentation:

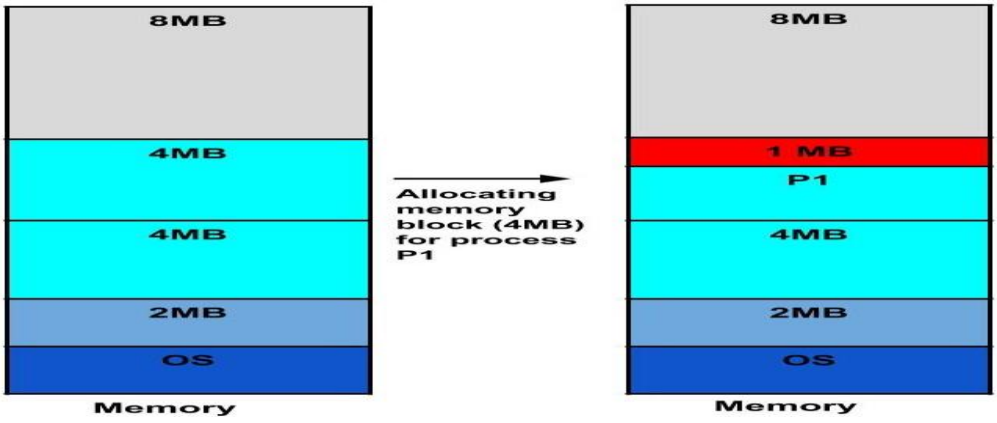
- Internal Fragmentation
- External Fragmentation

Internal Fragmentation

In this fragmentation, the process is allocated a memory block of size more than the size of that process. Due to this some part of the memory is left unused and this cause internal fragmentation.

Example: Suppose there is fixed partitioning (i.e. the memory blocks are of fixed sizes) is used for memory allocation in RAM. These sizes are 2MB, 4MB, 4MB, 8MB. Some part of this RAM is occupied by the Operating System (OS).

Now, suppose a process P1 of size 3MB comes and it gets memory block of size 4MB. So, the 1MB that is free in this block is wasted and this space can't be utilized for allocating memory to some other process. This is called **internal fragmentation**.



How to remove internal fragmentation?

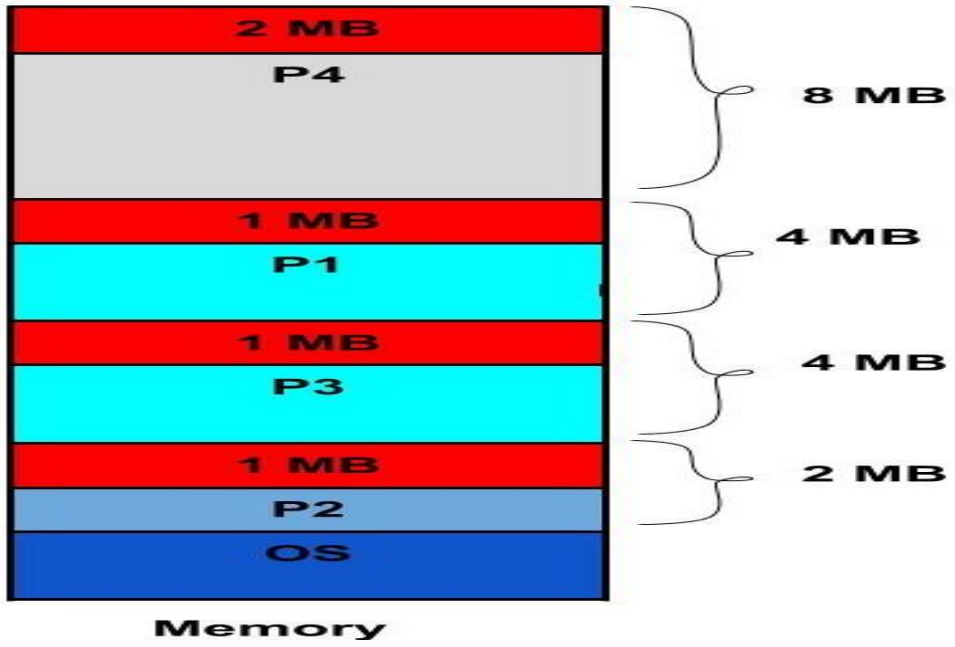
This problem is occurring because we have fixed the sizes of the memory blocks. This problem can be removed if we use dynamic partitioning for allocating space to the process. In dynamic partitioning, the process is allocated only that much amount of space which is required by the process. So, there is no internal fragmentation.

External Fragmentation

In this fragmentation, although we have total space available that is needed by a process still we are not able to put that process in the memory because that space is not contiguous. This is called external fragmentation.

Example: Suppose in the above example, if three new processes P2, P3, and P4 come of sizes 2MB, 3MB, and 6MB respectively. Now, these processes get memory blocks of size 2MB, 4MB and 8MB respectively allocated.

So, now if we closely analyze this situation then process P3 (unused 1MB)and P4(unused 2MB) are again causing internal fragmentation. So, a total of 4MB (1MB (due to process P1) + 1MB (due to process P3) + 2MB (due to process P4)) is unused due to internal fragmentation.



Now, suppose a new process of 4 MB comes. Though **we have a total space of 4MB** still **we can't allocate this memory** to the process. This is called **external fragmentation**.

How to remove external fragmentation?

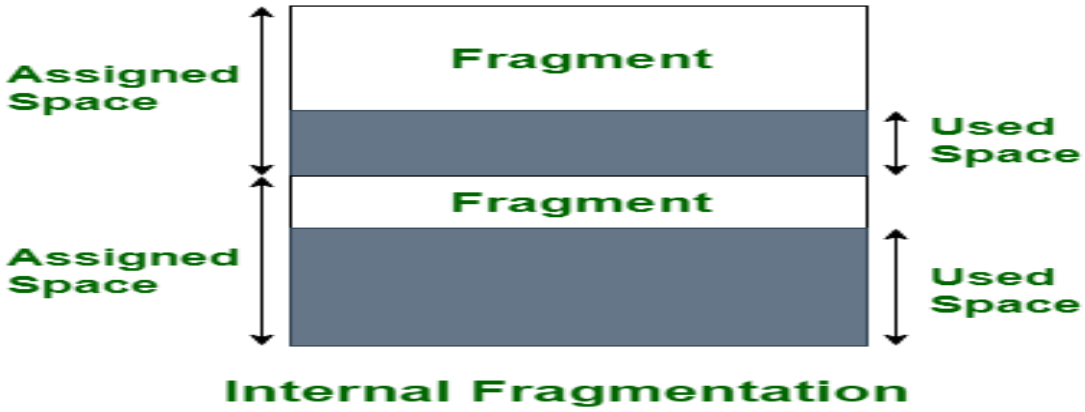
This problem is occurring because **we are allocating memory continuously** to the processes. So, if we remove this condition external fragmentation can be reduced. This is what done in **paging & segmentation**(non-contiguous memory allocation techniques) where memory is allocated non-contiguously to the processes. We will learn about paging and segmentation in the next blog.

Another way to remove external fragmentation is **compaction**. When dynamic partitioning is used for memory allocation then external fragmentation can be reduced by **merging all the free memory** together in one large block. This technique is also called **defragmentation.** This larger block of memory is then used for allocating space according to the needs of the new processes.

Difference between Internal and External fragmentation

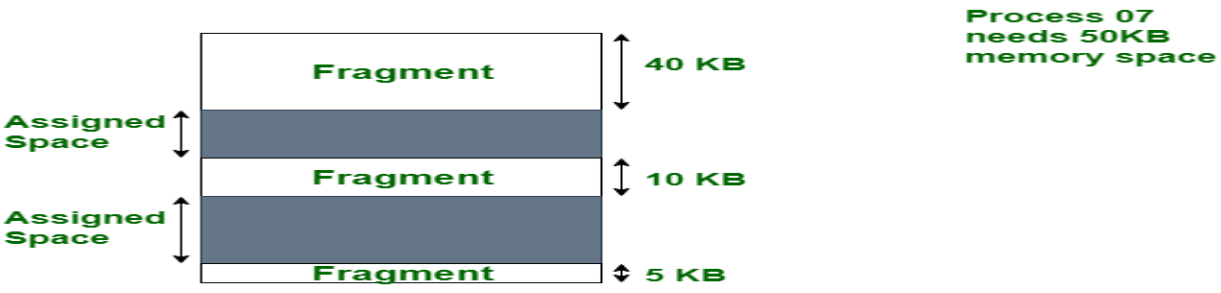
There are two types of fragmentation in OS which are given as: Internal fragmentation, and External fragmentation.

Internal Fragmentation:
Internal fragmentation happens when the memory is split into mounted sized blocks. Whenever a method request for the memory, the mounted sized block is allotted to the method. just in case the memory allotted to the method is somewhat larger than the memory requested, then the distinction between allotted and requested memory is that the Internal fragmentation.



The above diagram clearly shows the internal fragmentation because the difference between memory allocated and required space or memory is called Internal fragmentation.

External Fragmentation:
External fragmentation happens when there's a sufficient quantity of area within the memory to satisfy the memory request of a method. however the process's memory request cannot be fulfilled because the memory offered is during a non-contiguous manner. Either you apply first-fit or best-fit memory allocation strategy it'll cause external fragmentation.



In above diagram, we can see that, there is enough space (55 KB) to run a process-07 (required 50 KB) but the memory (fragment) is not contiguous. Here, we use compaction, paging or segmentation to use the free space to run a process.

Difference between Internal fragmentation and External fragmentation:-

S.NO	Internal fragmentation	External fragmentation
1.	In internal fragmentation fixed-sized memory, blocks square measure appointed to process.	In external fragmentation, variable-sized memory blocks square measure appointed to method.
2.	Internal fragmentation happens when the method or process is larger than the memory.	External fragmentation happens when the method or process is removed.
3.	The solution of internal fragmentation is best-fit block.	Solution of external fragmentation is compaction, paging and segmentation.
4.	Internal fragmentation occurs when memory is divided into fixed sized partitions.	External fragmentation occurs when memory is divided into variable size partitions based on the size of processes.
5.	The difference between memory allocated and required space or memory is called Internal fragmentation.	The unused spaces formed between non-contiguous memory fragments are too small to serve a new process, is called External fragmentation .

Buddy System – Memory allocation technique

Prerequisite – Partition Allocation Methods

Static partition schemes suffer from the **limitation** of having the fixed number of active processes and the usage of space may also not be optimal. The **buddy system** is a memory allocation and management algorithm that manages memory in **power of two increments**. Assume the memory size is 2U, suppose a size of S is required.

- **If 2U-1<S<=2U:** Allocate the whole block
- **Else:** Recursively divide the block equally and test the condition at each time, when it satisfies, allocate the block and get out the loop.

System also keep the record of all the unallocated blocks each and can merge these different size blocks to make one big chunk.

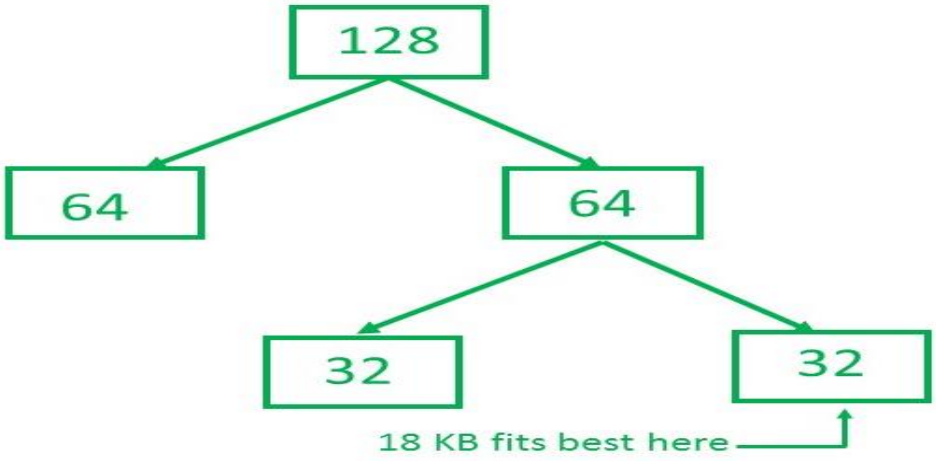
Advantage –

- Easy to implement a buddy system
- Allocates block of correct size
- It is easy to merge adjacent holes
- Fast to allocate memory and de-allocating memory

Disadvantage –

- It requires all allocation unit to be powers of two
- It leads to internal fragmentation

Example –
Consider a system having buddy system with physical address space 128 KB.Calculate the size of partition for 18 KB process.
Solution –



So, size of partition for 18 KB process = 32 KB. It divides by 2, till possible to get minimum block to fit 18 KB.

Advantage –

- Easy to implement a buddy system
- Allocates block of correct size
- It is easy to merge adjacent holes
- Fast to allocate memory and de-allocating memory

Disadvantage –

- It requires all allocation unit to be powers of two
- It leads to internal fragmentation.

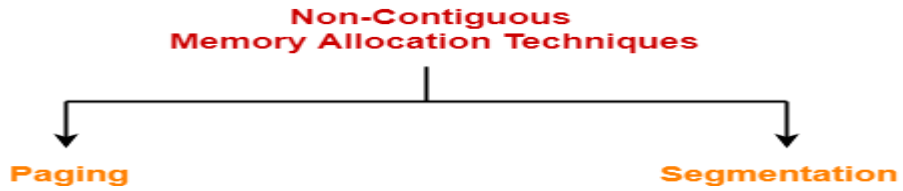
PART-II NON CONTINUOUS MEMORY ALLOCATION

Non-Contiguous Memory Allocation-

- **Non-contiguous memory allocation is a memory allocation technique.**
- **It allows to store parts of a single process in a non-contiguous fashion.**
- **Thus, different parts of the same process can be stored at different places in the main memory.**

Techniques-

- There are two popular techniques used for non-contiguous memory allocation-



Paging-

- **Paging is a fixed size partitioning scheme.**
- In paging, secondary memory and main memory are divided into equal fixed size partitions.
- The partitions of secondary memory are called as **pages**.
- The partitions of main memory are called as **frames**.
- 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

Paging

- Paging is a memory management scheme that permits the physical address space of a process tc be non contiguous.
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, for example 512 bytes).
- Divide logical memory into blocks of 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 backing stor is divided into fixed sized blocks that are of the same size as the memory frames.
- The hardware support for paging is illustrated in below figure.
- Every address generated by the CPU 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. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.

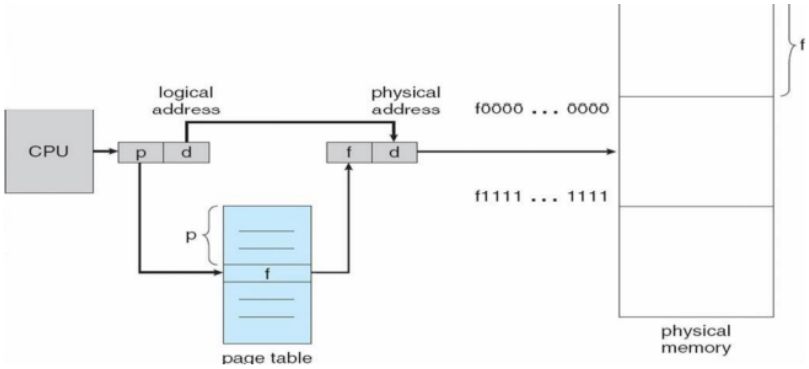
Paging

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 that's set up to emulate the computer's RAM. 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.

- Paging is a memory management scheme that permits the physical address space of a process to be non contiguous.
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, for example 512 bytes).
- Divide logical memory into blocks of 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 backing store is divided into fixed sized blocks that are of the same size as the memory frames.
- The hardware support for paging is illustrated in below figure.
- Every address generated by the CPU 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. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.



Address Translation

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

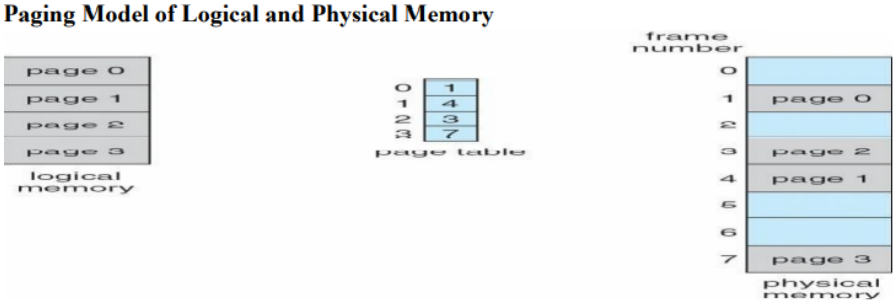
```
Logical Address = Page number + page offset
```

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

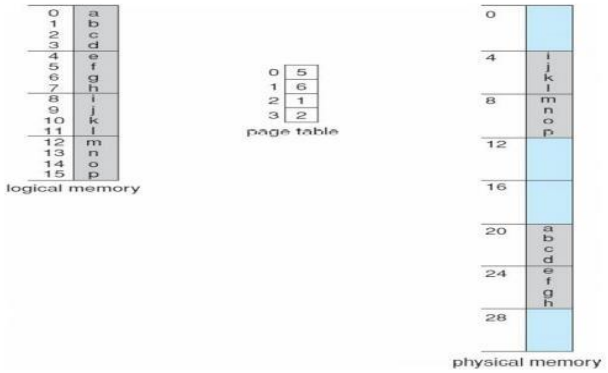
```
Physical Address = Frame number + page offset
```

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.

Paging Model of Logical and Physical Memory

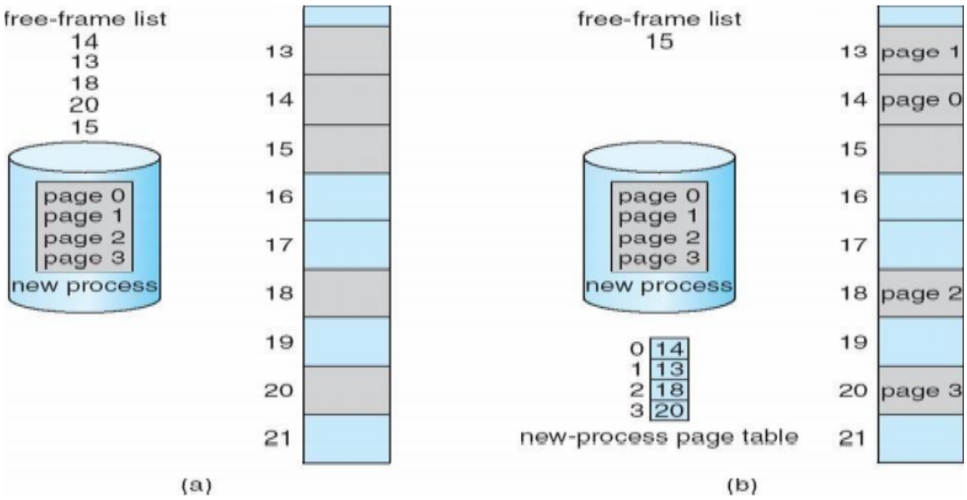


Paging Example



32-byte memory and 4-byte pages

Free Frames



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.

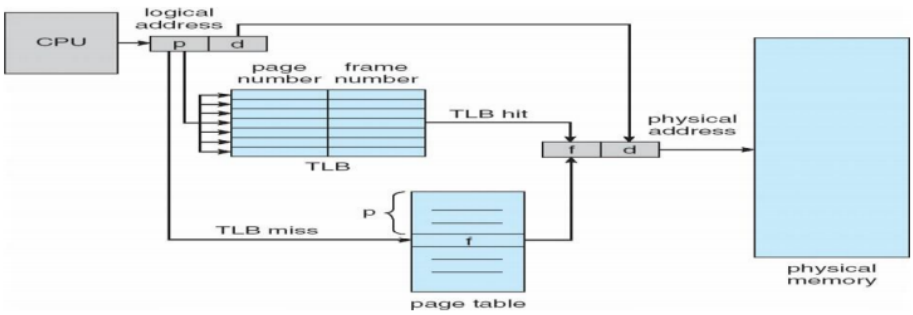
This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Implementation of Page Table

- Page table is kept in main memory
- **Page-table base register (PTBR)** points to the page table
- **Page-table length register (PRLR)** indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.

The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs)**

Paging Hardware With TLB



Example:

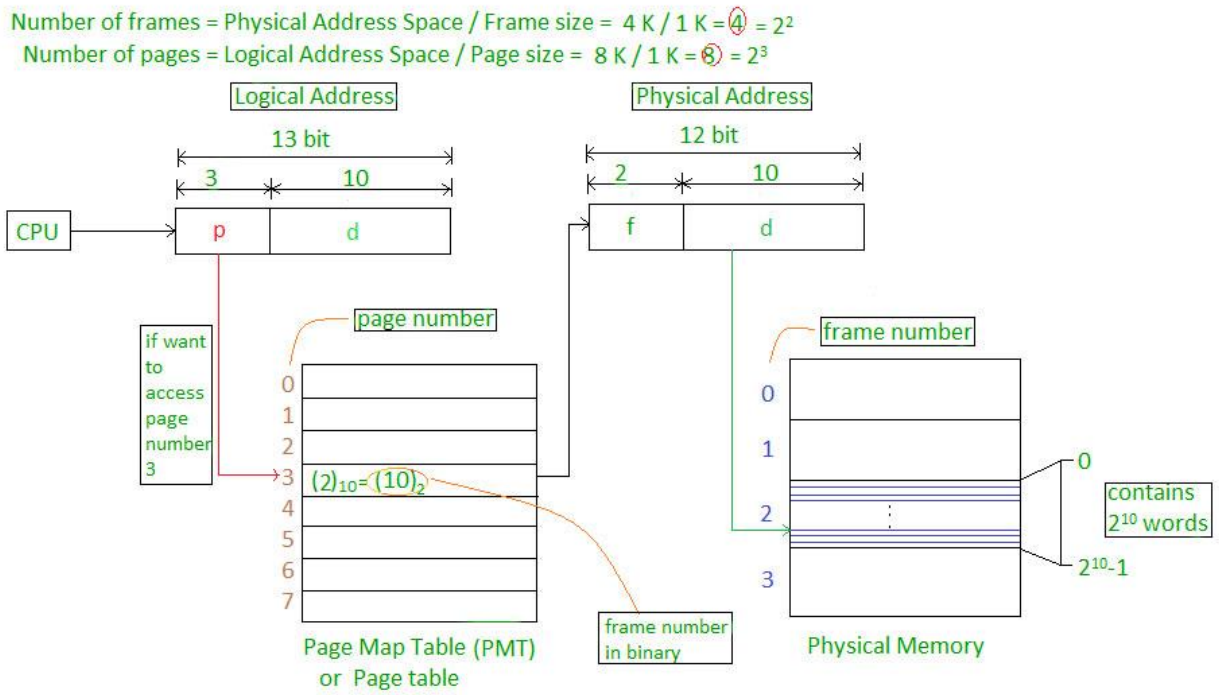
- If Logical Address = 31 bit, then Logical Address Space = 231 words = 2 G words (1 G = 230)
- If Logical Address Space = 128 M words = 27 * 220 words, then Logical Address = log2 227 = 27 bits
- If Physical Address = 22 bit, then Physical Address Space = 222 words = 4 M words (1 M = 220)
- If Physical Address Space = 16 M words = 24 * 220 words, then Physical Address = log2 224 = 24 bits

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 paging technique.

- The Physical Address Space is conceptually divided into a number of fixed-size blocks, called **frames**.
- The Logical address Space is also splitted into fixed-size blocks, called **pages**.
- Page Size = Frame Size

Let us consider an example:

- Physical Address = 12 bits, then Physical Address Space = 4 K words
- Logical Address = 13 bits, then Logical Address Space = 8 K words
- Page size = frame size = 1 K words (assumption)



Address generated by CPU is divided into

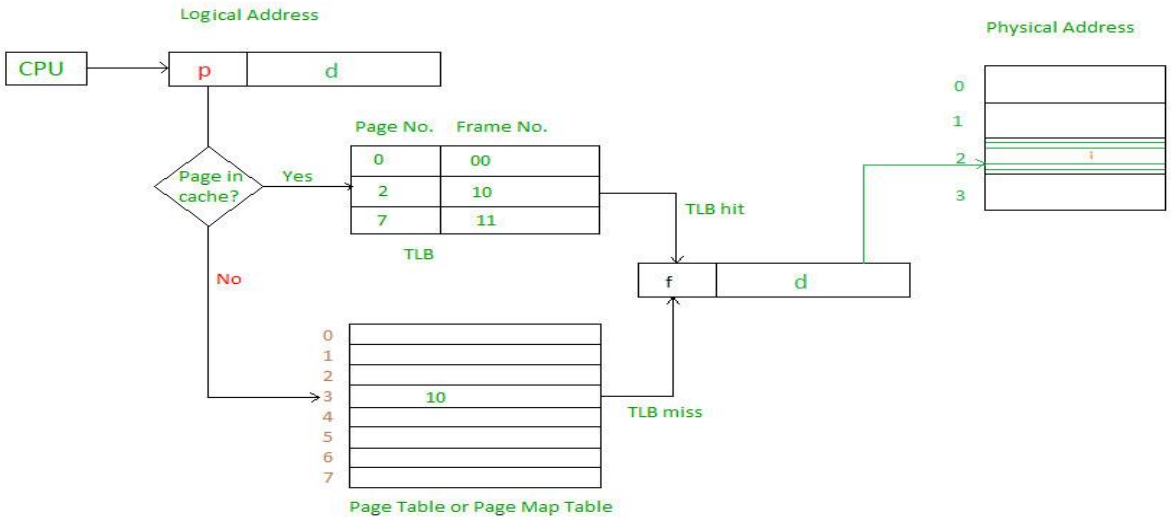
- **Page number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
- **Page offset(d):** Number of bits required to represent particular word in a page or page size of Logical Address Space or word number of a page or page offset.

Physical Address is divided into

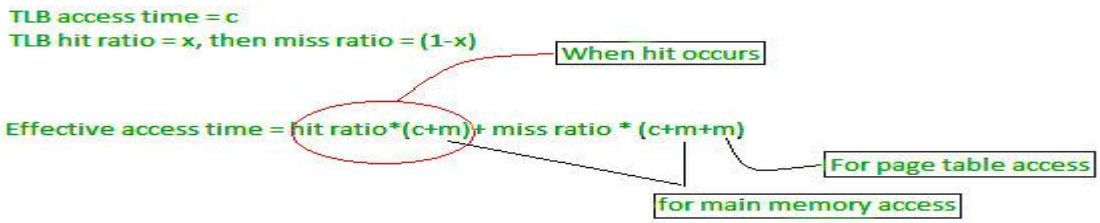
- **Frame number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number.
- **Frame offset(d):** Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

The hardware implementation of page table can be done by using dedicated registers. But the usage of register for the page table is satisfactory only if page table is small. If page table contain large number of entries then we can use TLB(translation Look-aside buffer), a special, small, fast look up hardware cache.

- The TLB is 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 corresponding value is returned.



Main memory access time = m
If page table are kept in main memory,
Effective access time = m(for page table) + m(for particular page in page table)

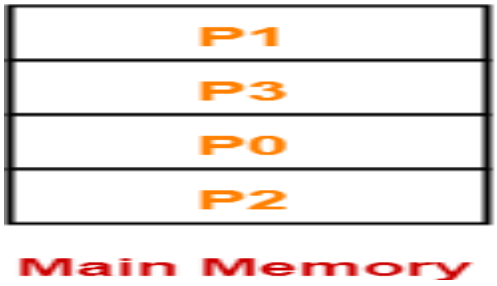
- 

ANOTHER WAY PAGING CONCEPT

- Each process is divided into parts where size of each part is same as page size.
- The size of the last part may be less than the page size.
- The pages of process are stored in the frames of main memory depending upon their availability.

Example-

- Consider a process is divided into 4 pages P_0, P_1, P_2 and P_3.
- Depending upon the availability, these pages may be stored in the main memory frames in a non-contiguous fashion as shown-



Translating Logical Address into Physical Address-

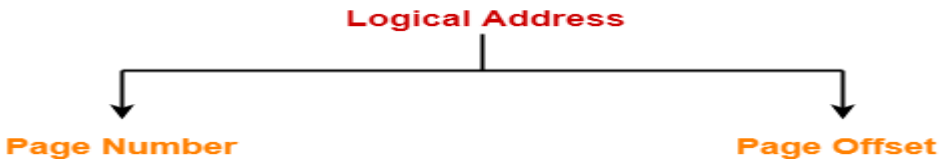
- CPU always generates a logical address.
- A physical address is needed to access the main memory.

Following steps are followed to translate logical address into physical address-

Step-01:

CPU generates a logical address consisting of two parts-

1. Page Number
2. Page Offset



- Page Number specifies the specific page of the process from which CPU wants to read the data.
- Page Offset specifies the specific word on the page that CPU wants to read.

Step-02:

- For the page number generated by the CPU,
- Page Table provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.

Step-03:

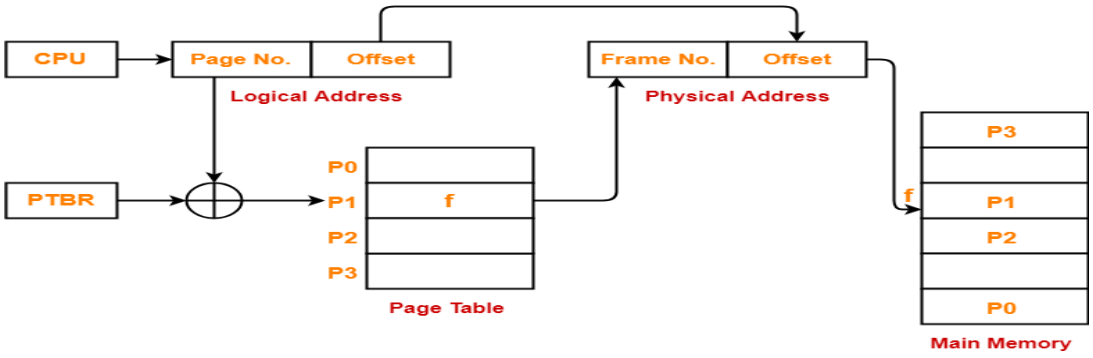
- The frame number combined with the page offset forms the required physical address.

- Frame number specifies the specific frame where the required page is stored.

- Page Offset specifies the specific word that has to be read from that page

iagram-

The following diagram illustrates the above steps of translating logical address into physical address-



Translating Logical Address into Physical Address

Advantages-

The advantages of paging are-

- It allows to store parts of a single process in a non-contiguous fashion.
- It solves the problem of external fragmentation.

Disadvantages-

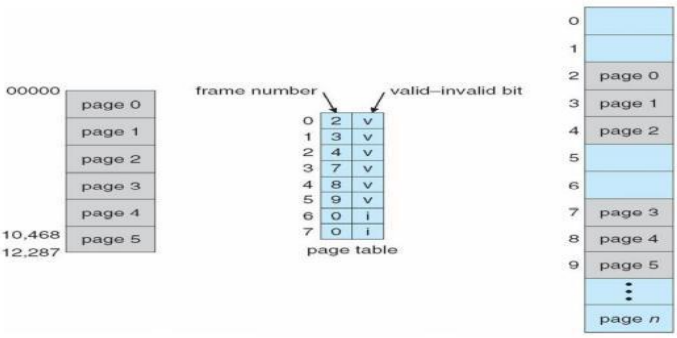
The disadvantages of paging are-

- It suffers from internal fragmentation.
- There is an overhead of maintaining a page table for each process.
- The time taken to fetch the instruction increases since now two memory accesses are required.

MEMORY PROTECTIN IN PAGING

Memory Protection

- Memory protection implemented by associating protection bit with each frame
- **Valid-invalid** bit attached to each entry in the page table:
- "valid" indicates that the associated page is in the process' logical address space, and is thus a legal
- page "invalid" indicates that the page is not in the process' logical address space
- Valid (v) or Invalid (i) Bit In A Page Table



Shared Pages

Shared code

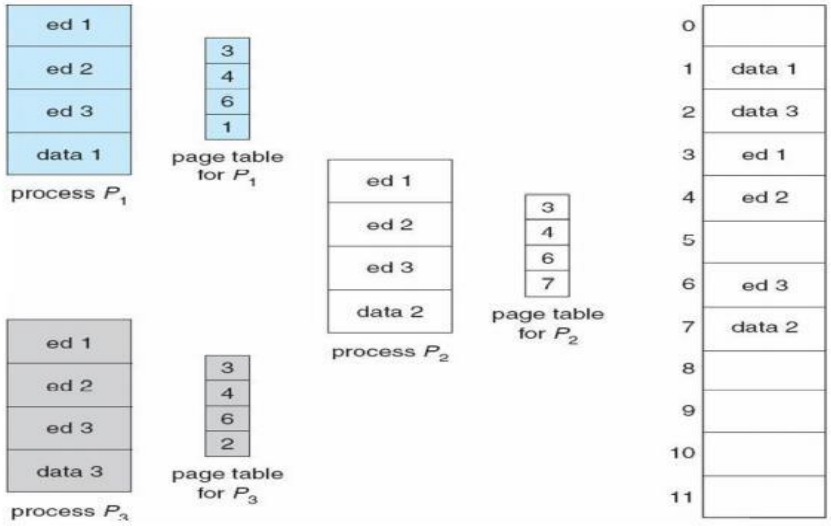
- One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
- Shared code must appear in same location in the logical address space of all processes

Private code and data

Each process keeps a separate copy of the code and data

-
- The pages for the private code and data can appear anywhere in the logical address space

Shared Pages Example



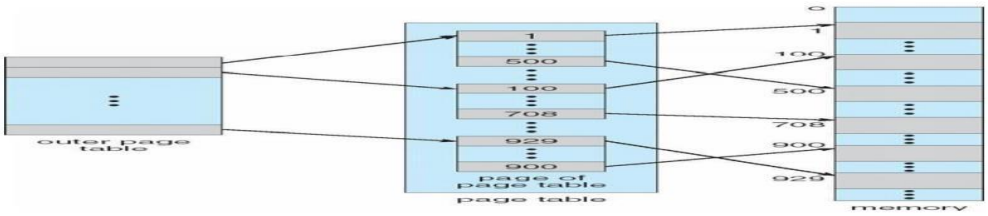
Structure of the Page Table

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

Break up the logical address space into multiple page tables A simple technique is a two-level page table

Two-Level Page-Table Scheme



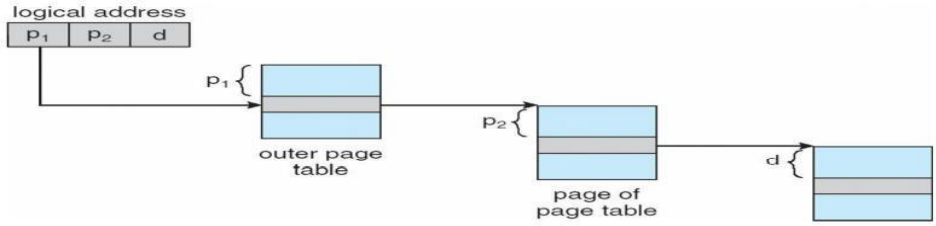
Two-Level Paging Example

- A logical address (on 32-bit machine with 1K page size) is divided
- into: a page number consisting of 22 bits
- a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
- a 12-bit page number a 10-bit page offset
- Thus, a logical address is as follows:
-

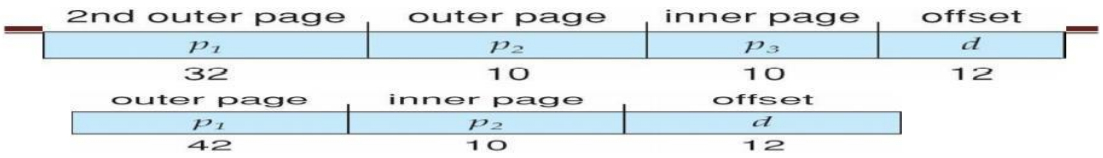
where pi is an index into the outer page table, and $p2$ is the displacement within the page of the outer page table

Page number		page offset
p_{12}^{i}	p_{10}^{2}	d_{10}

Address-Translation Scheme



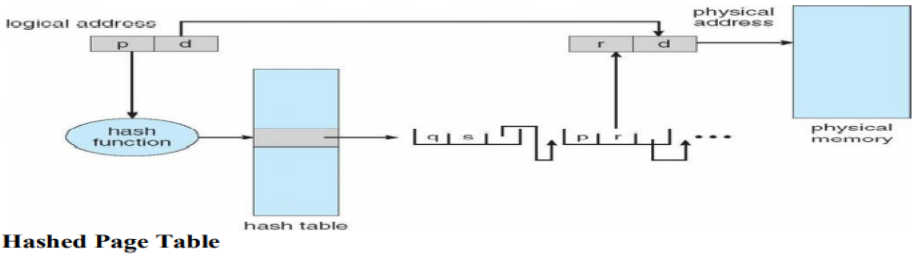
Three-level Paging Scheme



Hashed Page Tables

- Common in address spaces > 32 bits
- The virtual page number is hashed into a page table
- This page table contains a chain of elements hashing to the same
- location Virtual page numbers are compared in this chain searching for
- a match

If a match is found, the corresponding physical frame is extracted



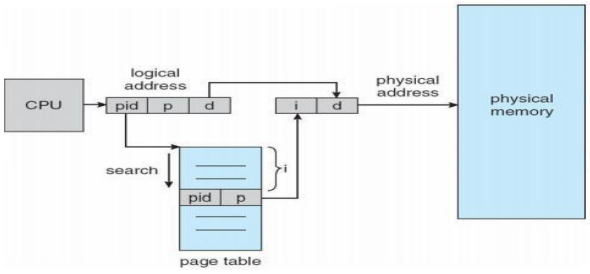
Hashed Page Table

Inverted Page Table

One entry for each real page of memory

- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries

Inverted Page Table Architecture



Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging –

- Paging reduces external fragmentation, but still suffers 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.

Segmentation in Operating System

A process is divided into Segments. The chunks that a program is divided into which are not necessarily all of the same sizes are called segments. Segmentation gives user's view of the process which paging does not give. Here the user's view is mapped to physical memory.

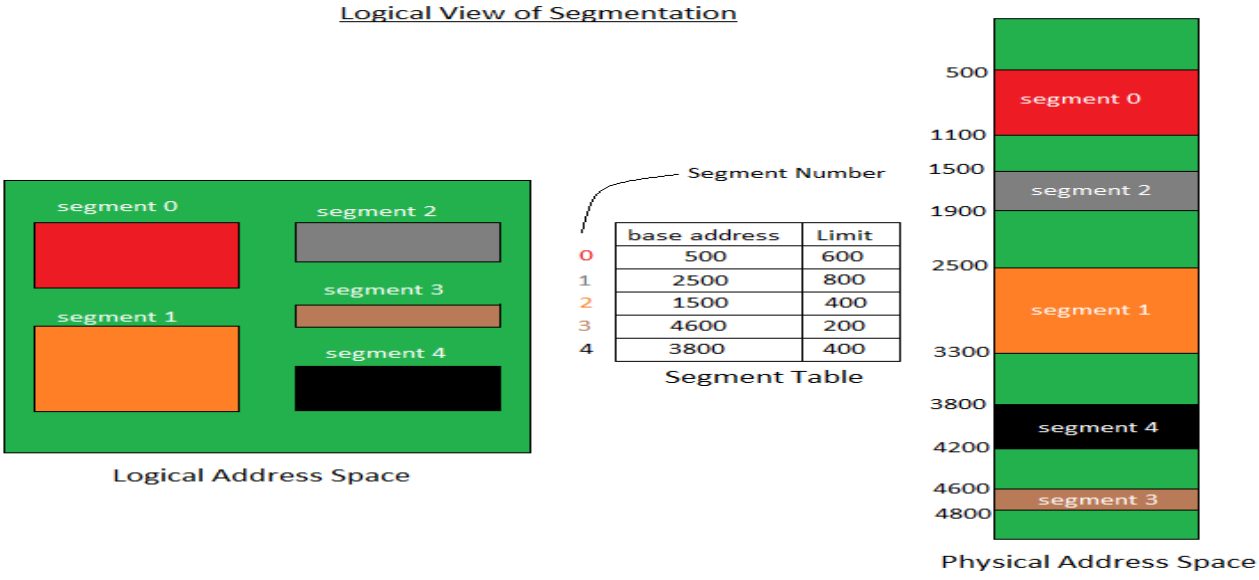
There are types of segmentation:

1. **Virtual memory segmentation –**
   Each process is divided into a number of segments, not all of which are resident at any one point in time.
2. **Simple segmentation –**
   Each process is divided into a number of segments, all of which are loaded into memory at run time, though not necessarily contiguously.

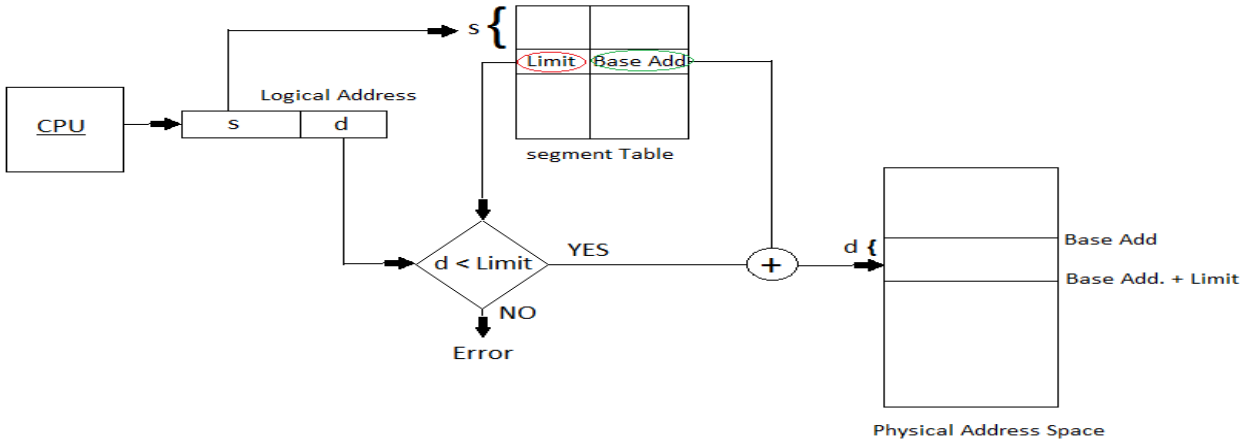
There is no simple relationship between logical addresses and physical addresses in segmentation. A table stores the information about all such segments and is called Segment Table.

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.



Translation of Two dimensional Logical Address to one dimensional Physical Address.



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.

OR

ANOTHER DATA OF SEGMENTATION

Segmentation-

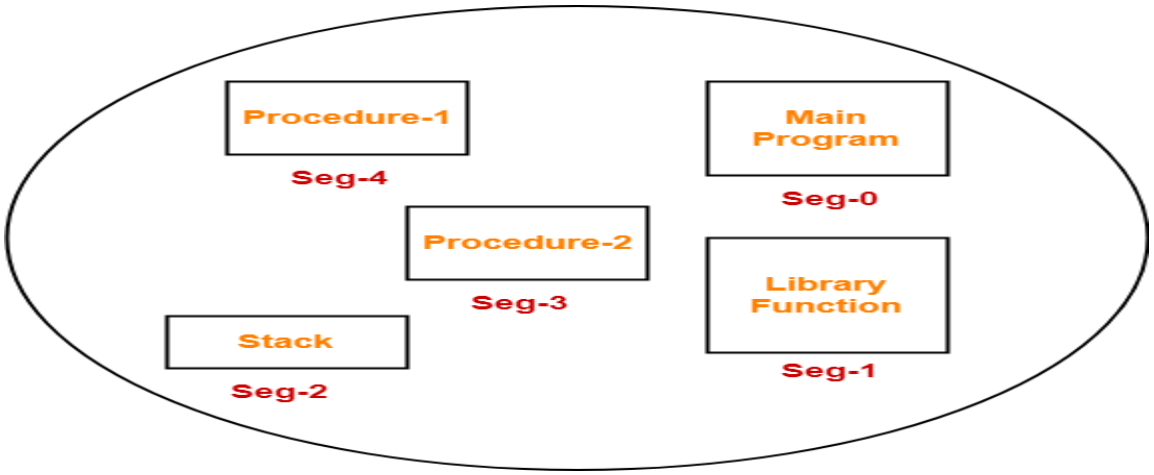
- Like Paging, Segmentation is another non-contiguous memory allocation technique.
- In segmentation, process is not divided blindly into fixed size pages.
- Rather, the process is divided into modules for better visualization.

Characteristics-

- Segmentation is a variable size partitioning scheme.
- In segmentation, secondary memory and main memory are divided into partitions of unequal size.

- The size of partitions depend on the length of modules.
- The partitions of secondary memory are called as **segments**.

Example-



Segment Table-

- Segment table is a table that stores the information about each segment of the process.
- It has two columns.
- First column stores the size or length of the segment.
- Second column stores the base address or starting address of the segment in the main memory.
- Segment table is stored as a separate segment in the main memory.
- Segment table base register (STBR) stores the base address of the segment table.

For the above illustration, consider the segment table is-

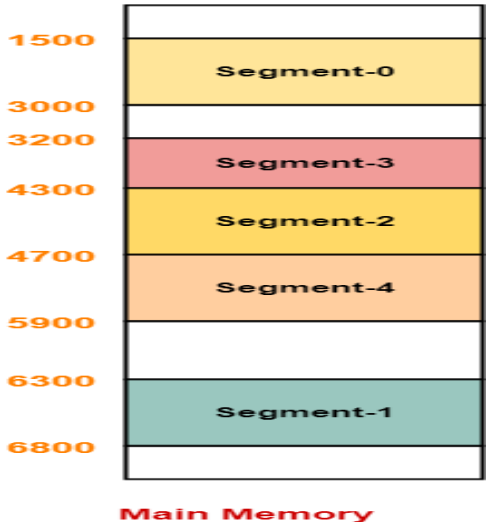
	Limit	Base
Seg-0	1500	1500
Seg-1	500	6300
Seg-2	400	4300
Seg-3	1100	3200
Seg-4	1200	4700

Segment Table

Here,

- Limit indicates the length or size of the segment.
- Base indicates the base address or starting address of the segment in the main memory.

In accordance to the above segment table, the segments are stored in the main memory as-



Translating Logical Address into Physical Address-

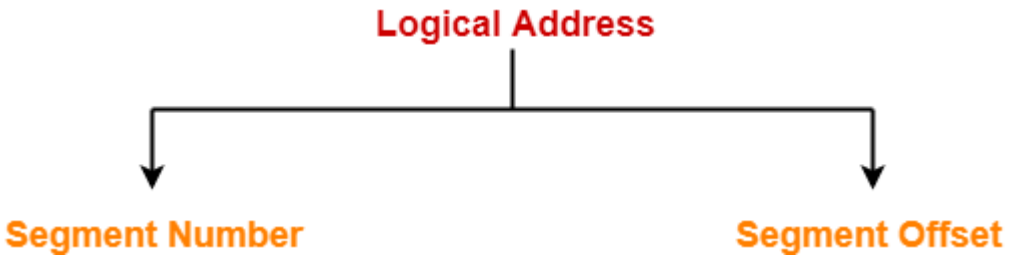
- CPU always generates a logical address.
- A physical address is needed to access the main memory.

Following steps are followed to translate logical address into physical address-

Step-01:

CPU generates a logical address consisting of two parts-

1. Segment Number
2. Segment Offset



- Segment Number specifies the specific segment of the process from which CPU wants to read the data.
- Segment Offset specifies the specific word in the segment that CPU wants to read.

Step-02:

- For the generated segment number, corresponding entry is located in the segment table.
- Then, segment offset is compared with the limit (size) of the segment.

Now, two cases are possible-

Case-01: Segment Offset >= Limit

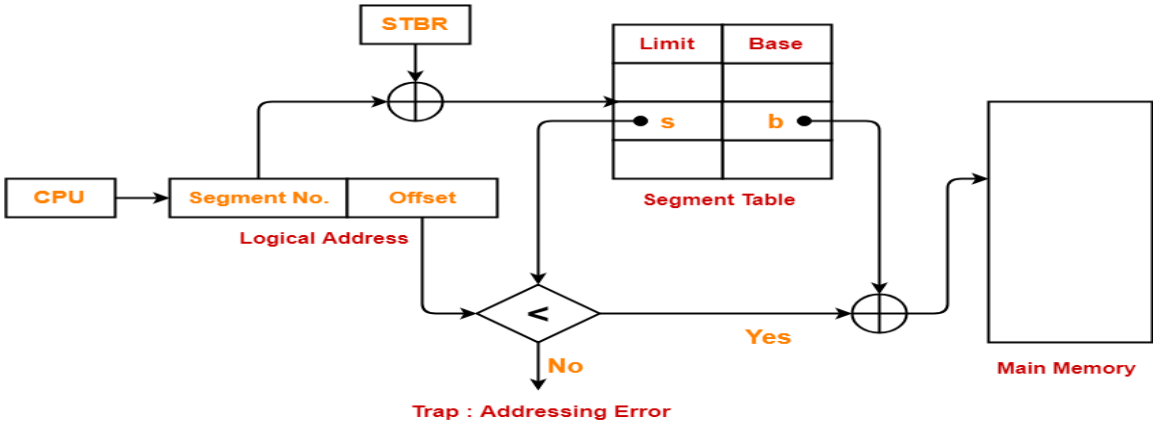
- If segment offset is found to be greater than or equal to the limit, a trap is generated.

Case-02: Segment Offset < Limit

- If segment offset is found to be smaller than the limit, then request is treated as a valid request.
- The segment offset must always lie in the range [0, limit-1],
- Then, segment offset is added with the base address of the segment.
- The result obtained after addition is the address of the memory location storing the required word.

Diagram-

The following diagram illustrates the above steps of translating logical address into physical address-



Translating Logical Address into Physical Address

Advantages-

The advantages of segmentation are-

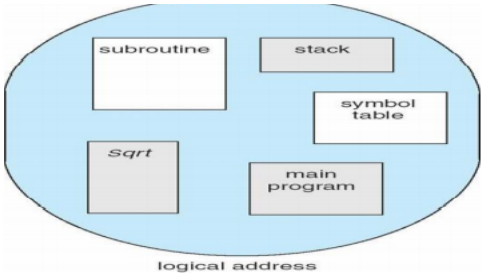
- It allows to divide the program into modules which provides better visualization.
- Segment table consumes less space as compared to Page Table in paging.
- It solves the problem of internal fragmentation.

Disadvantages-

The disadvantages of segmentation are-

- There is an overhead of maintaining a segment table for each process.
- The time taken to fetch the instruction increases since now two memory accesses are required.
- Segments of unequal size are not suited for swapping.
- It suffers from external fragmentation as the free space gets broken down into smaller pieces with the processes being loaded and removed from the main memory

ANOTHER CONCEPT



User's View of a Program

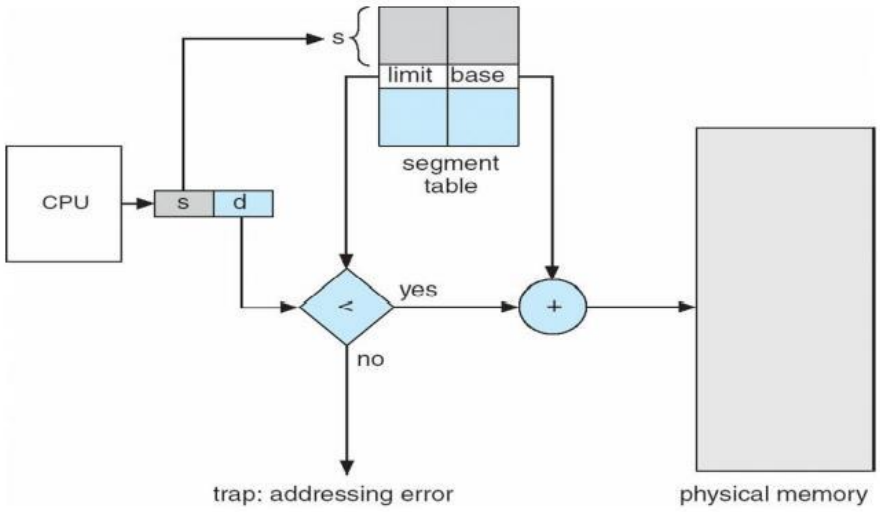
Segmentation Architecture

- Logical address consists of a two tuple:

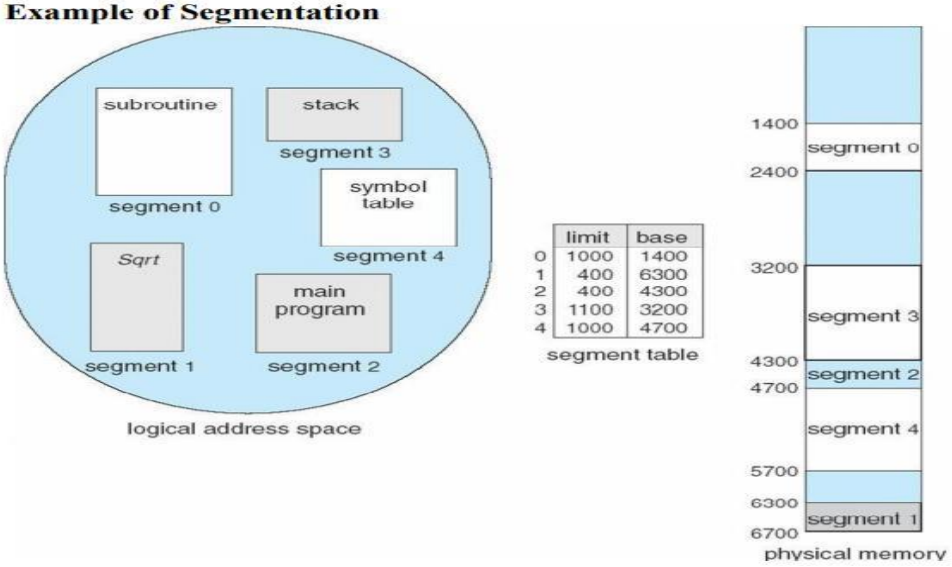
o <segment-number, offset>,

- **Segment table** – maps two-dimensional physical addresses; each table entry has:space
- **base** – contains the starting physical address where the segments reside in memory
- **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program; segment number ***s*** is legal if ***s*** < **STLR**
- Protection
- With each entry in segment table associate:
- validation bit = 0 Þ illegal segment
- read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation
- problem A segmentation example is shown in the following diagram

Segmentation Hardware



Example of Segmentation



Segmentation with paging

Instead of an actual memory location the segment information includes the address of a page table for the segment. When a program references a memory location the offset is translated to a memory address using the page table. A segment can be extended simply by allocating another memory page and adding it to the segment's page table.

An implementation of virtual memory on a system using segmentation with paging usually only moves individual pages back and forth between main memory and secondary storage, similar to a paged non-segmented system. Pages of the segment can be located anywhere in main memory and need not be contiguous. This usually results in a reduced amount of input/output between primary and secondary storage and reduced memory fragmentation.

SEGMENTATION WITH PAGING

- Paging and Segmentation are the non-contiguous memory allocation techniques.
- Paging divides the process into equal size partitions called as pages.

- Segmentation divides the process into unequal size partitions called as segments.

Segmented Paging-

Segmented paging is a scheme that implements the combination of segmentation and paging.

Working-

In segmented paging,

- Process is first divided into segments and then each segment is divided into pages.
- These pages are then stored in the frames of main memory.
- A page table exists for each segment that keeps track of the frames storing the pages of that segment.
- Each page table occupies one frame in the main memory.
- Number of entries in the page table of a segment = Number of pages that segment is divided.
- A segment table exists that keeps track of the frames storing the page tables of segments.
- Number of entries in the segment table of a process = Number of segments that process is divided.
- The base address of the segment table is stored in the segment table base register.

Translating Logical Address into Physical Address-

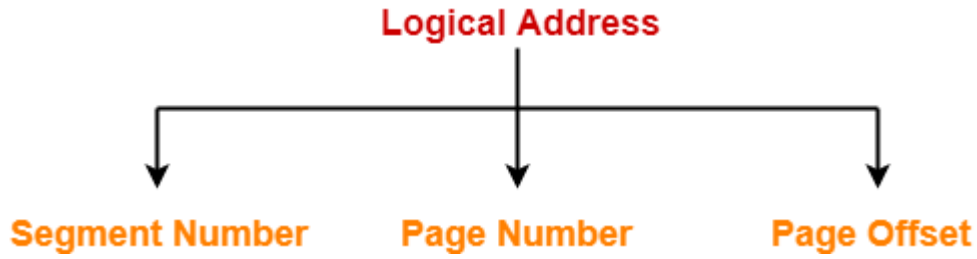
- CPU always generates a logical address.
- A physical address is needed to access the main memory.

Following steps are followed to translate logical address into physical address-

Step-01:

CPU generates a logical address consisting of three parts-

1. Segment Number
2. Page Number
3. Page Offset



- Segment Number specifies the specific segment from which CPU wants to reads the data.
- Page Number specifies the specific page of that segment from which CPU wants to read the data.
- Page Offset specifies the specific word on that page that CPU wants to read.

Step-02:

- For the generated segment number, corresponding entry is located in the segment table.
- Segment table provides the frame number of the frame storing the page table of the referred segment.
- The frame containing the page table is located.

Step-03:

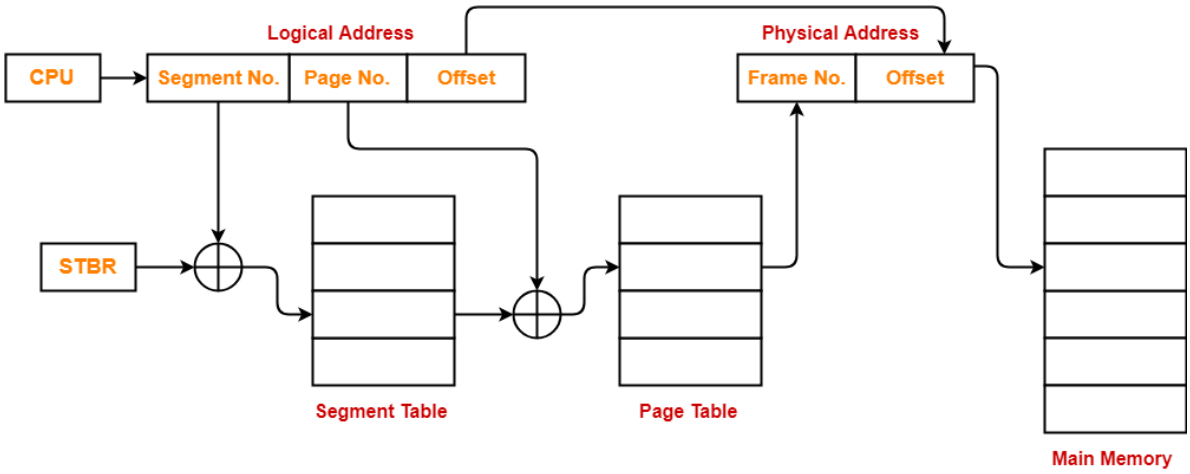
- For the generated page number, corresponding entry is located in the page table.
- Page table provides the frame number of the frame storing the required page of the referred segment.
- The frame containing the required page is located.

Step-04:

- The frame number combined with the page offset forms the required physical address.
- For the generated page offset, corresponding word is located in the page and read.

Diagram-

The following diagram illustrates the above steps of translating logical address into physical address-



Translating Logical Address into Physical Address

Advantages-

The advantages of segmented paging are-

- Segment table contains only one entry corresponding to each segment.
- It reduces memory usage.
- The size of Page Table is limited by the segment size.
- It solves the problem of external fragmentation.

Disadvantages-

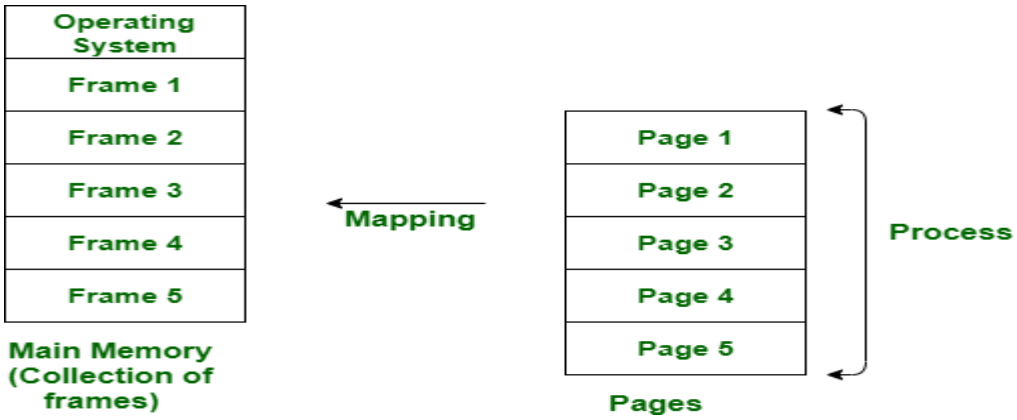
The disadvantages of segmented paging are-

- Segmented paging suffers from internal fragmentation.
- The complexity level is much higher as compared to paging.

Difference Between Paging and Segmentation

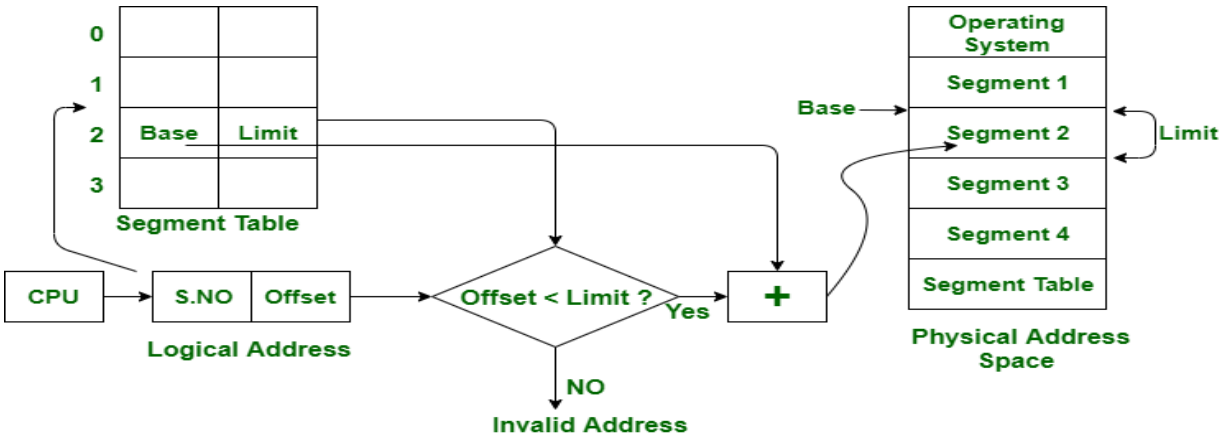
Paging:
Paging is a method or techniques which is used for non-contiguous memory allocation. It is a fixed size partitioning theme (scheme). In paging, both main memory and secondary memory are divided into equal fixed size partitions. The partitions of secondary memory area unit and main memory area unit known as as pages and frames respectively.

Paging is a memory management method accustomed fetch processes from the secondary memory into the main memory in the form of pages. in paging, each process is split into parts wherever size of every part is same as the page size. The size of the last half could also be but the page size. The pages of process area unit hold on within the frames of main memory relying upon their accessibility.



Segmentation:
Segmentation is another non-contiguous memory allocation scheme like paging. like paging, in segmentation, process isn't divided indiscriminately into mounted(fixed) size pages. It is variable size partitioning theme. like paging, in segmentation, secondary and main memory are not divided into partitions of equal size. The partitions of secondary memory area unit known as as segments. The details concerning every segment are hold in a table known as as segmentation table. Segment table contains two main data concerning segment, one is Base, which is the bottom address of the segment and another is Limit, which is the length of the segment.

In segmentation, CPU generates logical address that contains Segment number and segment offset. If the segment offset is a smaller amount than the limit then the address called valid address otherwise it throws miscalculation because the address is invalid.



The above figure shows the translation of logical address to physical address.

Difference between Paging and Segmentation:

S.NO	Paging	Segmentation
1.	In paging, program is divided into fixed or mounted size pages.	In segmentation, program is divided into variable size sections.
2.	For paging operating system is accountable.	For segmentation compiler is accountable.
3.	Page size is determined by hardware.	Here, the section size is given by the user.
4.	It is faster in the comparison of segmentation.	Segmentation is slow.
5.	Paging could result in internal fragmentation.	Segmentation could result in external fragmentation.
6.	In paging, logical address is split into page number and page offset.	Here, logical address is split into section number and section offset.
7.	Paging comprises a page table which encloses the base address of every page.	While segmentation also comprises the segment table which encloses segment number and segment offset.
8.	Page table is employed to keep up the page data.	Section Table maintains the section data.
9.	In paging, operating system must maintain a free frame list.	In segmentation, operating system maintain a list of holes in main memory.
10.	Paging is invisible to the user.	Segmentation is visible to the user.
11.	In paging, processor needs page number, offset to calculate absolute address.	In segmentation, processor uses segment number, offset to calculate full address.

PART-11VIRTUAL MEMORY MANAGEMENT

Virtual Memory

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.

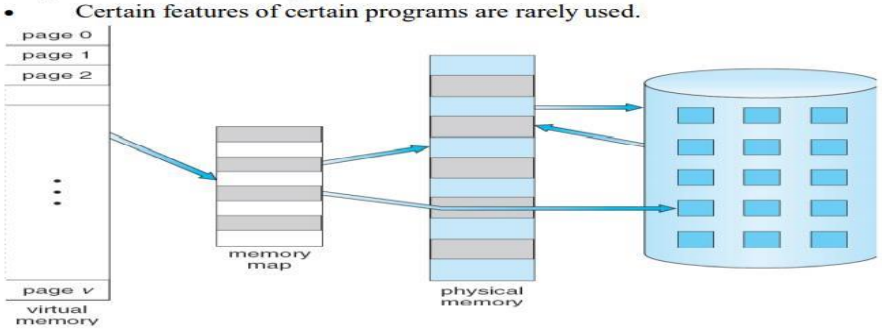


Fig. Diagram showing virtual memory that is larger than physical memory.

Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Virtual Memory

- It is a technique which allows execution of process that may not be compiled within the primary memory.
- It separates the user logical memory from the physical memory. This separation allows an extremely large memory to be provided for program when only a small physical memory is available.
- Virtual memory makes the task of programming much easier because the programmer no longer needs to working about the amount of the physical memory is available or not.
- The virtual memory allows files and memory to be shared by different processes by page sharing.
- It is most commonly implemented by demand paging.

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.

OR

Virtual Memory in Operating System

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of 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 amount of secondary memory is available not by the actual number of the 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.

1. 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 main memory such that it occupies different places in main memory at different times during the course of execution.
2. A process may be broken into 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 use of 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.

OR

What is Virtual Memory?

Virtual Memory is a storage mechanism which offers user an illusion of having a very big main memory. It is done by treating a part of secondary memory as the main memory. In Virtual memory, the user can store processes with a bigger size than the available main memory.

Therefore, instead of loading one long process in the main memory, the OS loads the various parts of more than one process in the main memory. Virtual memory is mostly implemented with demand paging and demand segmentation.

Why Need Virtual Memory?

Here, are reasons for using virtual memory:

- Whenever your computer doesn't have space in the physical memory it writes what it needs to remember to the hard disk in a swap file as virtual memory.
- If a computer running Windows needs more memory/RAM, then installed in the system, it uses a small portion of the hard drive for this purpose.

How Virtual Memory Works?

In the modern world, virtual memory has become quite common these days. It is used whenever some pages require to be loaded in the main memory for the execution, and the memory is not available for those many pages.

So, in that case, instead of preventing pages from entering in the main memory, the OS searches for the RAM space that are minimum used in the recent times or that are not referenced into the secondary memory to make the space for the new pages in the main memory.

Let's understand virtual memory management with the help of one example.

For example:

Let's assume that an OS requires 300 MB of memory to store all the running programs. However, there's currently only 50 MB of available physical memory stored on the RAM.

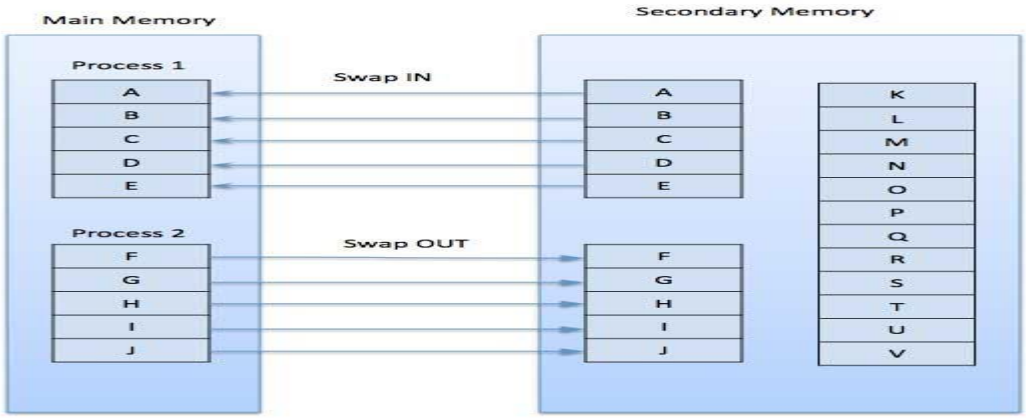
he OS will then set up 250 MB of virtual memory and use a program called the Virtual Memory Manager(VMM) to manage that 250 MB.

- So, in this case, the VMM will create a file on the hard disk that is 250 MB in size to store extra memory that is required.
- The OS will now proceed to address memory as it considers 300 MB of real memory stored in the RAM, even if only 50 MB space is available.
- It is the job of the VMM to manage 300 MB memory even if just 50 MB of real memory space is available.

DEMAND PAGING CONCEPT

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program’s pages out to the disk or any of the new program’s pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program’s pages as they are referenced.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Advantages

Following are the advantages of Demand Paging −

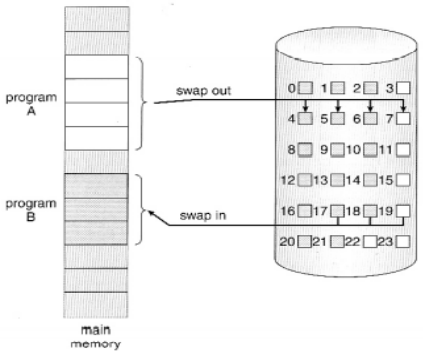
- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

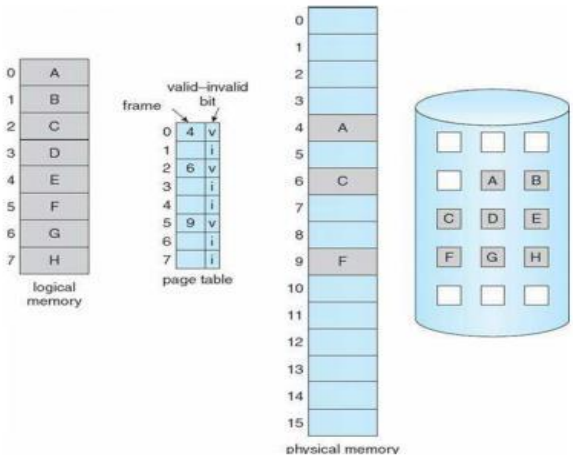
- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques. OR

Demand Paging

A demand paging system is similar to the paging system with swapping feature. When we want to execute a process we swap it into the memory. A swapper manipulates entire process where as a pager is concerned with the individual pages of a process. The demand paging concept is using pager rather than swapper. When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again. Instead of swapping in a whole process, the pager brings only those necessary pages into memory. The transfer of a paged memory to contiguous disk space is shown in below figure.



Thus it avoids reading into memory pages that will not used any way decreasing the swap time and the amount of physical memory needed. In this technique we need some hardware support to distinct between the pages that are in memory and those that are on the disk. A valid and invalid bit is used for this purpose. When this bit is set to valid it indicates that the associate page is in memory. If the bit is set to invalid it indicates that the page is either not valid or is valid but currently not in the disk.



Marking a page invalid will have no effect if the process never attempts to access that page. So while a process executes and access pages that are memory resident, execution proceeds normally. Access to a page marked invalid causes a page fault trap. It is the result of the OS's failure to bring the desired page into memory.

Procedure to handle page fault

If a process refers to a page that is not in physical memory then

- We check an internal table (page table) for this process to determine whether the reference was valid or invalid.
- If the reference was invalid, we terminate the process, if it was valid but not yet brought in, we have to bring that from main memory.
- Now we find a free frame in memory.
- Then we read the desired page into the newly allocated frame.
- When the disk read is complete, we modify the internal table to indicate that the page is now in

Demand Paging

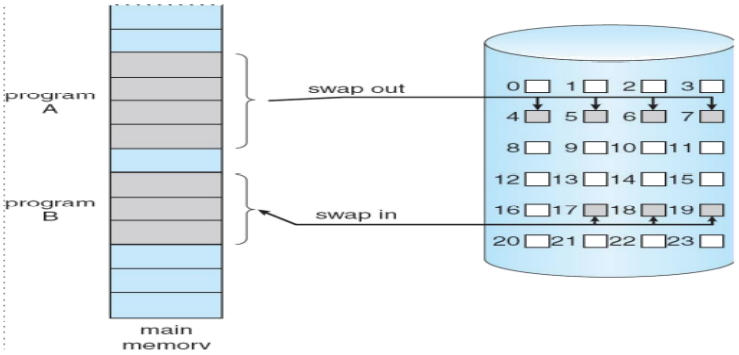
A demand paging is similar to a paging system with swapping(Fig 5.2). When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory.

When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed.

Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the valid-invalid bit scheme. Where valid and invalid pages can be checked checking the bit and marking a page will have no effect if the process never attempts to access the pages. While the process executes and accesses pages that are memory resident, execution proceeds normally.

Fig. Transfer of a paged memory to continuous disk space

Fig. Transfer of a paged memory to continuous disk space



Access to a page marked invalid causes a page-fault trap. This trap is the result of the operating system's failure to bring the desired page into memory.

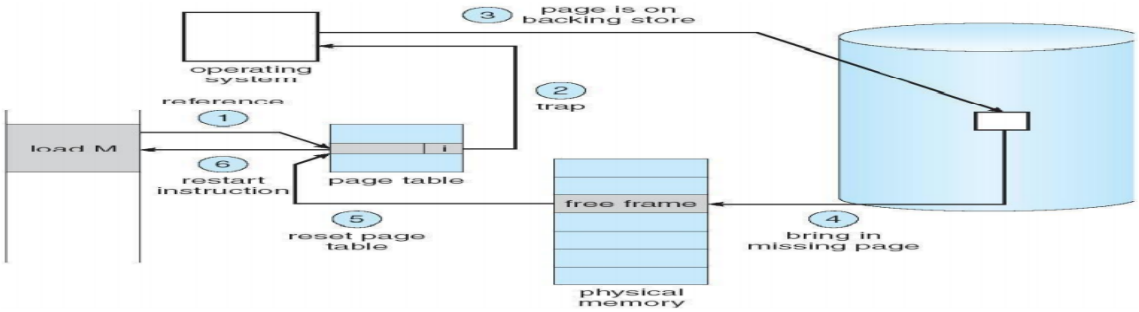
Initially only those pages are loaded which will be required the process immediately.
The pages that are not moved into the memory are marked as invalid in the page table. For

an invalid entry the rest of the table is empty. In case of pages that are loaded in the memory, they are marked as valid along with the information about where to find the swapped out page.
When the process requires any of the page that is not loaded into the memory, a page fault trap is triggered and following steps are followed,

1. The memory address which is requested by the process is first checked, to verify the request made by the process.
2. If its found to be invalid, the process is terminated.
3. In case the request by the process is valid, a free frame is located, possibly from a free-frame list, where the required page will be moved.
4. A new operation is scheduled to move the necessary page from disk to the specified memory location. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)
5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to valid.

Fig. Steps in handling a page fault

Fig. Steps in handling a page fault



6. The instruction that caused the page fault must now be restarted from the beginning.

There are cases when no pages are loaded into the memory initially, pages are only loaded when demanded by the process by generating page faults. This is called **Pure Demand Paging**.
The only major issue with Demand Paging is, after a new page is loaded, the process starts execution from the beginning. It is not a big issue for small programs, but for larger programs it affects performance drastically.

Difference between Demand Paging and Segmentation

Demand Paging:
Demand paging is identical to the paging system with swapping. In demand paging, a page is delivered into the memory on demand i.e., only when a reference is made to a location on that page. Demand paging combines the feature of simple paging and implement virtual memory as it has a large virtual memory. Lazy swapper concept is implemented in demand paging in which a page is not swapped into the memory unless it is required.

Segmentation:
Segmentation is the arrangement of memory management. According to the segmentation the logical address space is a collection of segments. Each segment has a name and length. Each logical address have two quantities segment name and the segment offset, for simplicity we use the segment number in place of segment name.

The difference between Demand Paging and Segmentation are as follows:

S.No.	Demand Paging	Segmentation
1.	In demand paging, the pages are of equal size.	While in segmentation, segments can be of different size.
2.	Page size is fixed in the demand paging.	Segment size may vary in segmentation as it grants dynamic increase of segments.
3.	It does not allows sharing of the pages.	While segments can be shared in segmentation.
4.	In demand paging, on demand pages are loaded in the memory.	In segmentation, during compilation segments are allocated to the program.
5.	Page map table in demand paging manages record of pages in memory.	Segment map table in segmentation demonstrates every segment address in the memory.
6.	It provides large virtual memory and have more efficient use of memory.	It provides virtual memory and maximum size of segment is defined by the size of memory.

Page Replacement Algorithms in Operating Systems

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 new page comes in.

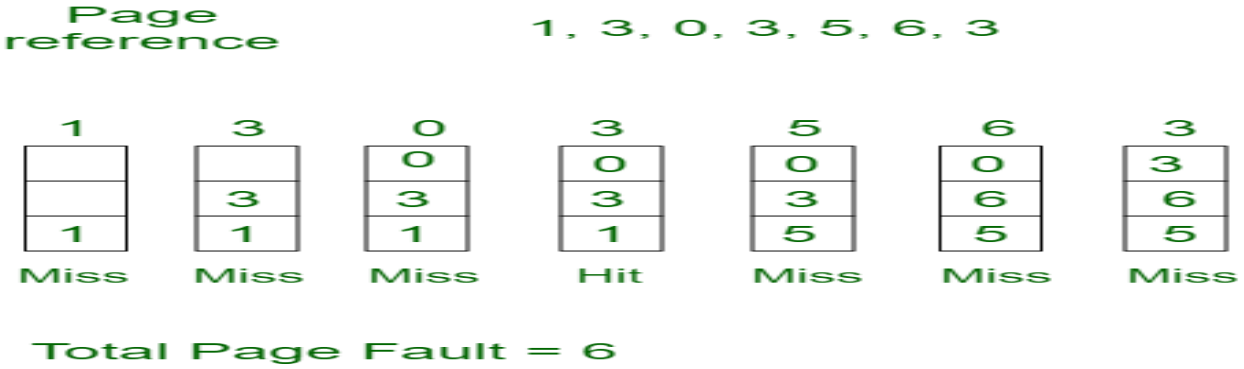
Page Fault – A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Since actual physical memory is much smaller than virtual memory, page faults happen. In case of 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.

Page Replacement Algorithms :

- **First In First Out (FIFO) –**
  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 with 3 page frames.Find 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 —> **0 Page Faults.**
Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.**
6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 Page Fault.**
Finally when 3 come it is not avilable so it replaces 0 **1 page fault**

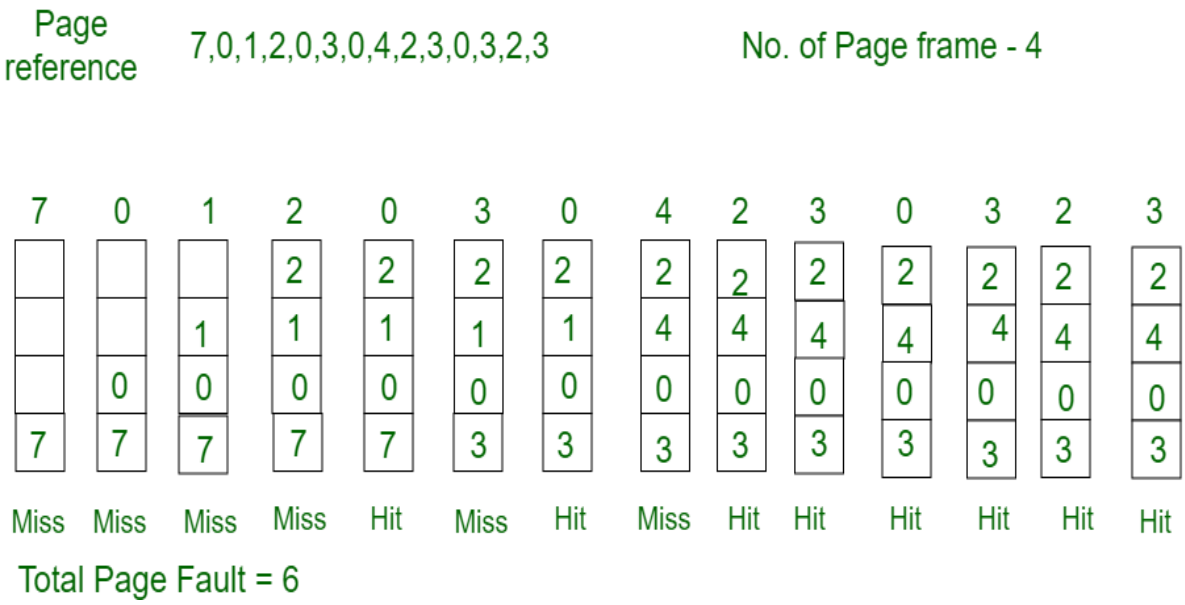
-

Belady's anomaly – Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

-

- **Optimal Page replacement –**
  In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Example-2: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.



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 —> **0 Page fault.**
when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.**
0 is already there so —> **0 Page fault.**.
4 will takes place of 1 —> **1 Page Fault.**

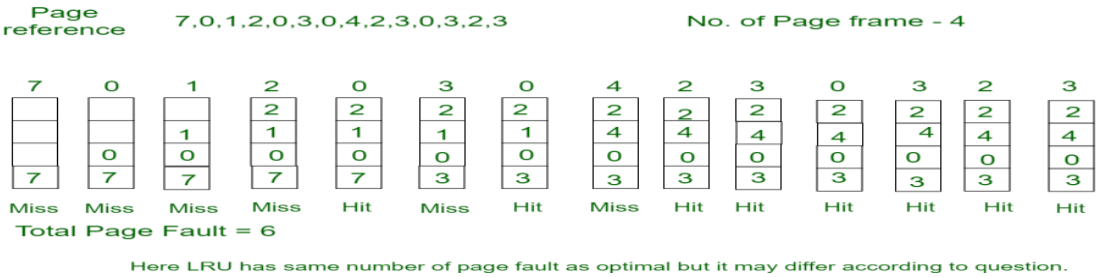
Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

-

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

Least Recently Used –
In this algorithm page will be replaced which is least recently used.**Example-3**Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page

frames.Find number of page faults.



Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already their so —> **0 Page fault.**
when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**
0 is already in memory so —> **0 Page fault**.
4 will takes place of 1 —> **1 Page Fault**
Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

OR

Page Replacement

As studied in Demand Paging, only certain pages of a process are loaded initially into the memory. This allows us to get more number of processes into the memory at the same time. but what happens when a process requests for more pages and no free memory is available to bring them in. Following steps can be taken to deal with this problem :

1. Put the process in the wait queue, until any other process finishes its execution thereby freeing frames.
2. Or, remove some other process completely from the memory to free frames.
3. Or, find some pages that are not being used right now, move them to the disk to get free frames. This technique is called **Page replacement** and is most commonly used. We have some great algorithms to carry on page replacement efficiently.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults,

Reference String

The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference.

The latter choice produces a large number of data, where we note two things.

- For a given page size, we need to consider only the page number, not the entire address.
- If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.

Page Replacement

- Each process is allocated frames (memory) which hold the process's pages (data)
- Frames are filled with pages as needed – this is called demand paging

- Over-allocation of memory is prevented by modifying the page-fault service routine to replace pages
- The job of the page replacement algorithm is to decide which page gets victimized to make room for a new page
- Page replacement completes separation of logical and physical memory

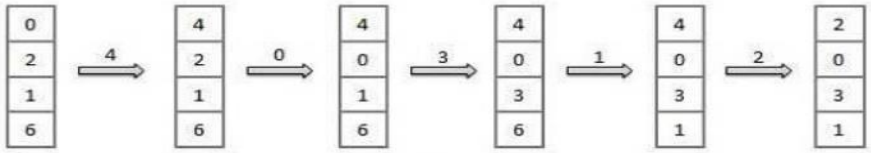
- For example, consider the following sequence of addresses – 123,215,600,1234,76,96
- If page size is 100, then the reference string is

1,2,6,12,0,0 First In First Out (FIFO) algorithm

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

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

Misses : x x x x x x x x x



Fault Rate = 9 / 12 = 0.75

FIFO algorithm

- Replaces pages based on their order of arrival: oldest page is replaced
- Example using 4 frames:

Reference #	1	2	3	4	5	6	7	8	9	10	11	12
Page referenced	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Frames	1	1	1	1	1	1	5	5	5	5	4	4
_ = faulting page		2	2	2	2	2	2	1	1	1	1	5
			3	3	3	3	3	3	2	2	2	2
				4	4	4	4	4	4	3	3	3

- Analysis: 12 page references, 10 page faults, 6 page replacements. Page faults per number of frames = 10/4 = 2.5

Optimal Page algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.

- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

```
Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1
Misses           : x  x  x  x  x        x
```

Optimal algorithm

- Ideally we want to select an algorithm with the lowest page-fault rate
- Such an algorithm exists, and is called (unsurprisingly) the optimal algorithm:
- Procedure: replace the page that will not be used for the longest time (or at all) – i.e. replace the page with the greatest forward distance in the reference string
- Example using 4 frames:

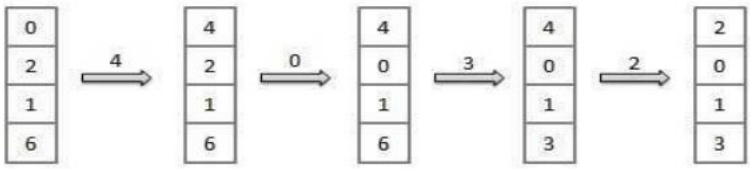
Reference #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Page referenced	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Frames	1	1	1	1	1	1	1	1	1	1	4	4
_ = faulting page		2	2	2	2	2	2	2	2	2	2	2
			3	3	3	3	3	3	3	3	3	3
				4	4	4	5	5	5	5	5	5

- Analysis: 12 page references, 6 page faults, 2 page replacements. Page faults per number of frames = 6/4 = 1.5
- Unfortunately, the optimal algorithm requires special hardware (crystal ball, magic mirror, etc.) not typically found on today's computers
- Optimal algorithm is still used as a metric for judging other page replacement algorithms

Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

```
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

LFU algorithm (page-based)

- procedure: replace the page which has been referenced least often
- For each page in the reference string, we need to keep a reference count. All reference counts start at 0 and are incremented every time a page is referenced.
- example using 4 frames:

Reference #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Page referenced	**1**	**2**	**3**	**4**	**1**	**2**	**5**	**1**	**2**	**3**	**4**	**5**
Frames	$^{1}\underline{1}$	$^{1}1$	$^{1}1$	$^{1}1$	$^{2}1$	$^{2}1$	$^{2}1$	$^{3}1$	$^{3}1$	$^{3}1$	$^{3}1$	$^{3}1$
_ = faulting page		$^{1}\underline{2}$	$^{1}2$	$^{1}2$	$^{1}2$	$^{2}2$	$^{2}2$	$^{2}2$	$^{3}2$	$^{3}2$	$^{3}2$	$^{3}2$
^{n} = reference count			$^{1}\underline{3}$	$^{1}3$	$^{1}3$	$^{1}3$	$^{1}\underline{5}$	$^{1}5$	$^{1}5$	$^{2}\underline{3}$	$^{2}3$	$^{2}\underline{5}$
				$^{1}\underline{4}$	$^{1}4$	$^{1}4$	$^{1}4$	$^{1}4$	$^{1}4$	$^{1}4$	$^{2}4$	$^{2}4$

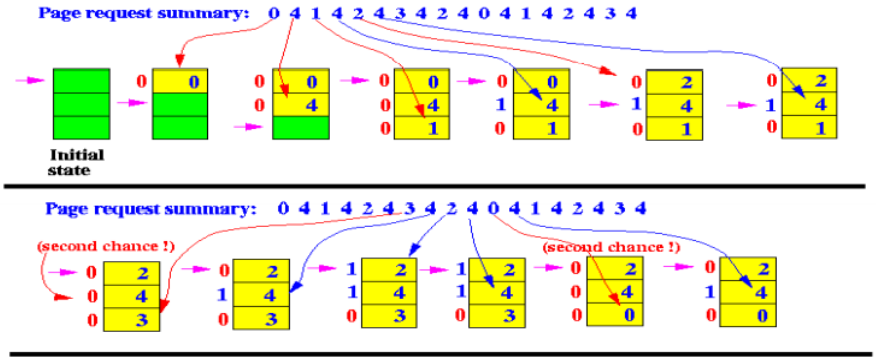
- At the 7th page in the reference string, we need to select a page to be victimized. Either 3 or 4 will do since they have the same reference count (1). Let's pick 3.
- Likewise at the 10th page reference; pages 4 and 5 have been referenced once each. Let's pick page 4 to victimize. Page 3 is brought in, and its reference count (which was 1 before we paged it out a while ago) is updated to 2.
- Analysis: 12 page references, 7 page faults, 3 page replacements. Page faults per number of frames = 7/4 = 1.75

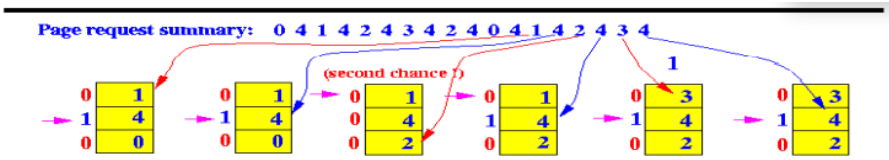
Second chance page replacement algorithm

- **Second Chance** replacement policy is called the **Clock** replacement policy...
- In the Second Chance page replacement policy, the candidate pages for removal are **consider** in a round robin matter, and a page that has been **accessed between consecutive considerations** will not be replaced.

The page replaced is the one that - considered in a round robin matter - has not been accessed since its last consideration.

- Implementation:
  - Add a "second chance" bit to each memory frame.
  - Each time a memory frame is referenced, set the "second chance" bit to ONE (1) - this will give the frame a second chance...
  - A new page read into a memory frame has the second chance bit set to ZERO (0)
  - When you need to find a page for removal, look in a round robin manner in the memory frames:
    - If the second chance bit is ONE, reset its second chance bit (to ZERO) and continue.
    - If the second chance bit is ZERO, replace the page in that memory frame.
- The following figure shows the behavior of the program in paging using the Second Chance page replacement policy:





- We can see notably that the **bad** replacement decision made by FIFO is **not present** in Second chance!!!
- There are a total of **9 page read operations** to satisfy the total of 18 page requests - just as good as the more computationally expensive LRU method !!!

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.

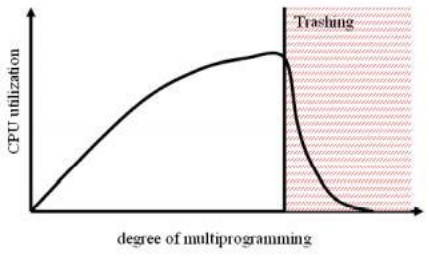
- **Class 1: (0,0)** – neither recently used nor modified - the best page to replace.
- **Class 2: (0,1)** – not recently used but modified - the page will need to be written out before replacement.
- **Class 3: (1,0)** – recently used but clean - probably will be used again soon.
- **Class 4: (1,1)**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

Trashing concept

Thrashing

- If a process does not have "enough" pages, the page-fault rate is very high
  - low CPU utilization
  - OS thinks it needs increased multiprogramming
  - adds another process to system
- Thrashing is when a process is busy swapping pages in and out
- Thrashing results in severe performance problems. Consider the following scenario, which is based on the actual behaviour of early paging systems. The operating system monitors CPU utilization. If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new process to the system. A global page replacement algorithm is used; it replaces pages with no regard to the process to which they belong. Now suppose that a process enters a new phase in its execution and needs more frames.



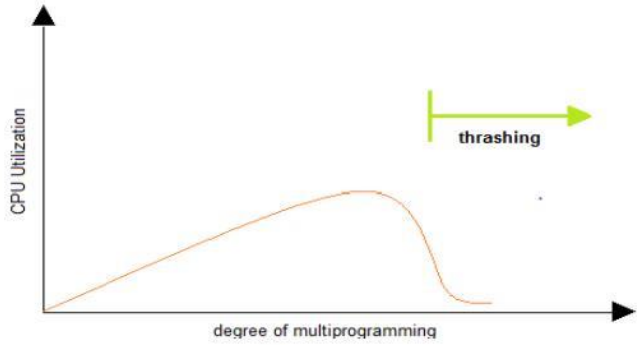
Or

Thrashing

If the number of frames allocated to a low-priority process falls below the minimum number required by the computer architecture, we must suspend that process' execution. We should then page out its remaining pages, freeing all its allocated frames. This provision introduces a swap-in, swap-out level of intermediate CPU scheduling.

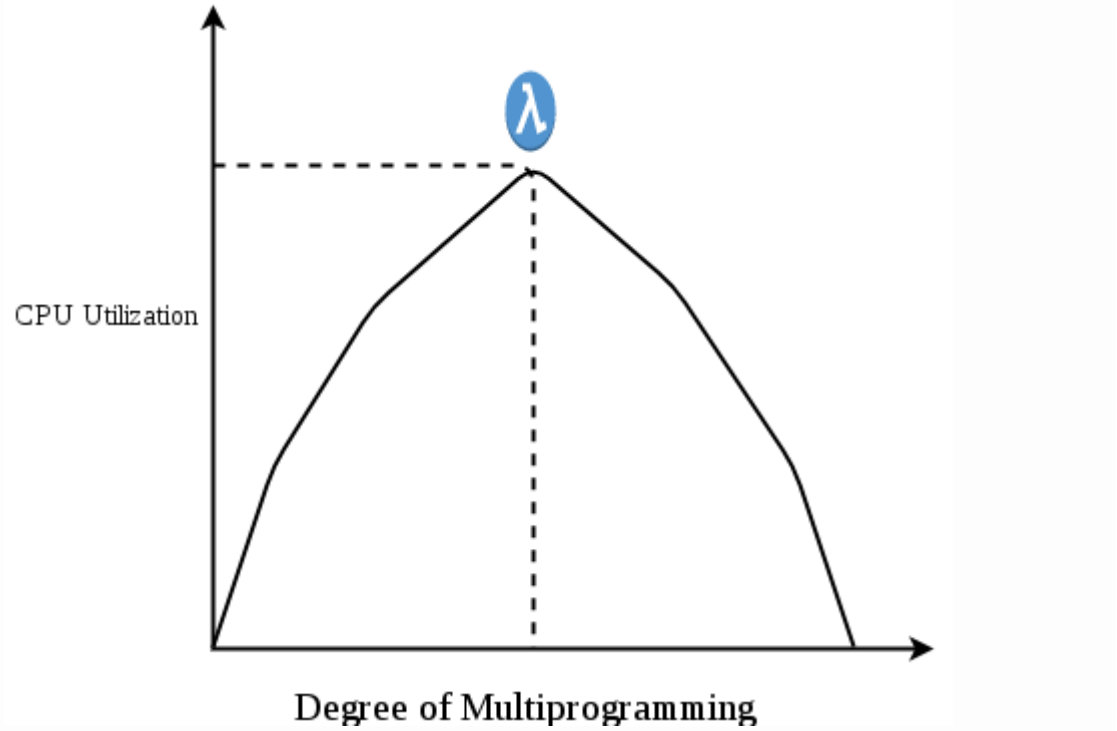
In fact, look at any process that does not have "enough" frames. Although it is technically possible to reduce the number of allocated frames to the minimum, there is some (larger) number of pages in active use. If the process does not have this number of frames, it will quickly page fault. At this point, it must replace some page. However, since all its pages are in active use, it must replace a page that will be needed again right away. Consequently, it quickly faults again, and again, and again. The process continues to fault, replacing pages for which it then faults and brings back in right away.

This high paging activity is called **thrashing**. A process is thrashing if it is spending more time paging than executing.



or

Thrashing :



At any given time, only few pages of any process are in main memory and therefore more processes can be maintained in memory. Furthermore time is saved because

unused pages are not swapped in and out of memory. However, the OS must be clever about how it manages this scheme. In the steady state practically, all of main memory will be occupied with process's pages, so that the processor and OS has direct access to as many processes as possible. Thus when the OS brings one page in, it must throw another out. If it throws out a page just before it is used, then it will just have to get that page again almost immediately. Too much of this leads to a condition called Thrashing. The system spends most of its time swapping pages rather than executing instructions. So a good page replacement algorithm is required.

In the given diagram, initial degree of multi programming upto some extent of point(lamda), the CPU utilization is very high and the system resources are utilized 100%. But if we further increase the degree of multi programming the CPU utilization will drastically fall down and the system will spent more time only in the page replacement and the time taken to complete the execution of the process will increase. This situation in the system is called as thrashing.

Causes of Thrashing :

1. **High degree of multiprogramming** : If the number of processes keeps on increasing in the memory than number of frames allocated to each process will be decreased. So, less number of frames will be available to each process. Due to this, page fault will occur more frequently and more CPU time will be wasted in just swapping in and out of pages and the utilization will keep on decreasing.
   For example:
   Let free frames = 400
   Case 1: Number of process = 100
   Then, each process will get 4 frames.
2. **Case 2**: Number of process = 400
   Each process will get 1 frame.
   Case 2 is a condition of thrashing, as the number of processes are increased,frames per process are decreased. Hence CPU time will be consumed in just swapping pages.
3.
4. **Lacks of Frames**:If a process has less number of frames then less pages of that process will be able to reside in memory and hence more frequent swapping in and out will be required. This may lead to thrashing. Hence sufficient amount of frames must be allocated to each process in order to prevent thrashing.

Recovery of Thrashing :

- Do not allow the system to go into thrashing by instructing the long term scheduler not to bring the processes into memory after the threshold.
- If the system is already in thrashing then instruct the mid term schedular to suspend some of the processes so that we can recover the system from thrashing.

Structure of Page Table

30th January 2021 by Neha T Leave a Comment

Structure of page table simply defines, in how many ways a page table can be structured. Well, the paging is a memory management technique where a large process is divided into pages and is placed in physical memory which is also divided

into frames. Frame and page size is equivalent. The operating system uses a page table to map the logical address of the page generated by CPU to its physical address in the main memory.

In this section, we will discuss three common methods that we use to structure a page table.

Structure of Page Table

1. Hierarchical Page Table
2. Hashed Page Table
3. Inverted Page Table

Hierarchical Page Table

As we knew when the CPU access a page of any process it has to be in the main memory. Along with the page, the page table of the same process must also be stored in the main memory. Now, what if the size of the page table is larger than the frame size of the main memory.

In that case, we have to breakdown the page table at multiple levels in order to fit in the frame of the main memory. Let us understand this with the help of an example.

Consider that the size of main memory (physical memory) is 512 MB = 229 (i.e. physical memory can be represented with 29 bits)

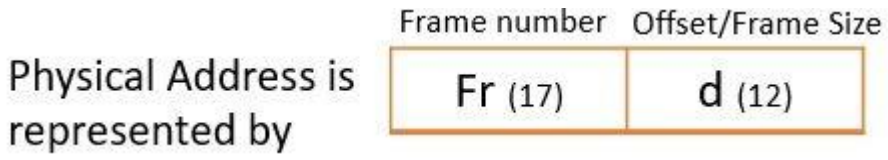
This physical memory is divided into a number of frames where each frame size = 4 KB = 212 (i.e. frame size can be represented with 12 bits)

Physical memory in bits = 29

Frame size in bits = 12

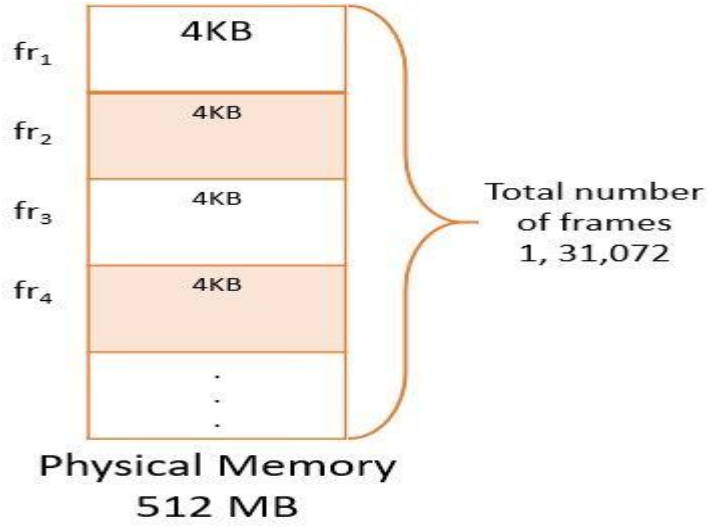
Number of bits used to represent frame number= 29 – 12 = 17.

We represent physical memory as:



Total number of frames would be 217 = 1, 31,072

Now we have a physical memory of 512 MB which is divided into 131072 frames where each frame size is 4 KB.

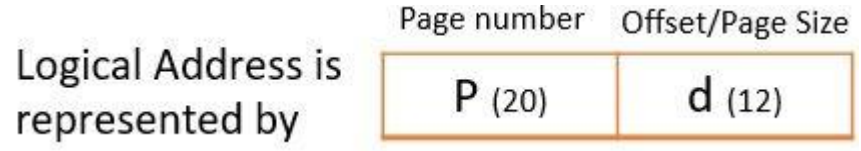


Today modern system supports logical address space up to 232 to 264 bytes.

Consider we have a process whose size is 4 GB = 232

The process is now divided into a number of pages and we know that page size is equal to frame size = 4 KB = 212

Now the logical address is represented as:



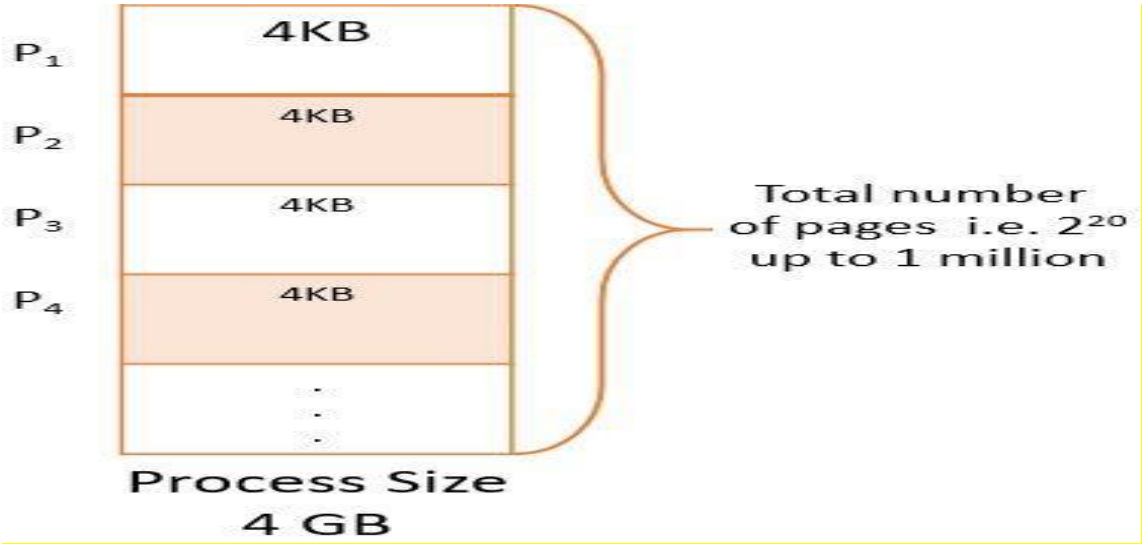
Logical address in bits = 32

Page size in bits = 12

Number of bits used to represent page number= 32 – 12 = 20.

So the total number of the pages of the process would be = 220 i.e. up to 1 million.

Now we have a process of size 4 GB which is divided into 1 million pages each of size 4KB.



Next, we have to implement a page table to store the information of pages of process i.e. which page is stored in which frame of the memory. As we have 1 million pages of the process there will be 1 million entries in the page table.

Now in page table the page number provided by CPU in logical address act as an index which leads you to the frame number where this page is stored in main memory. This means each entry of the page table has the frame number.

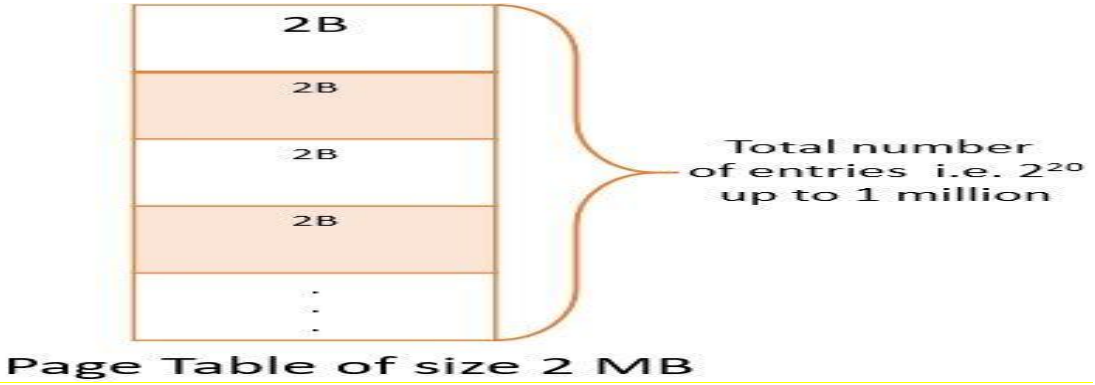
As we have seen above the frame number is represented by 17 bits so the size of each entry will be of 17 bits i.e. approx. 2 bytes.

Size of page table = number of entries * size of each entry

= 220 * 2

= 221

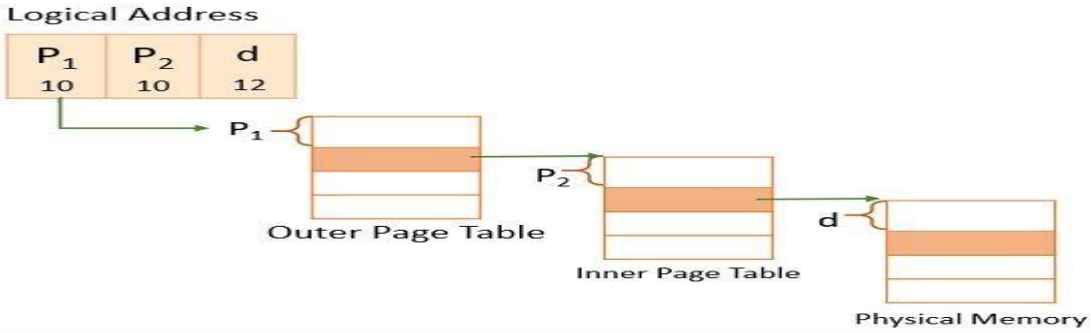
=2 MB



So can you store the page table of size 2 MB in a frame of the main memory where frame size is 4 KB? It is impossible.

So we need to divide the page table this division of page table can be accomplished in several ways. You can perform two-level division, three-level division on page table and so on.

Let us discuss two-level paging in which a page table itself is paged. So we will have two-page tables' inner page table and outer page table. We have a logical address with page number 20 and page offset 12. As we are paging a page table the page number will further get split to 10-bit page number and 10 bit offset as you can see in the image below.



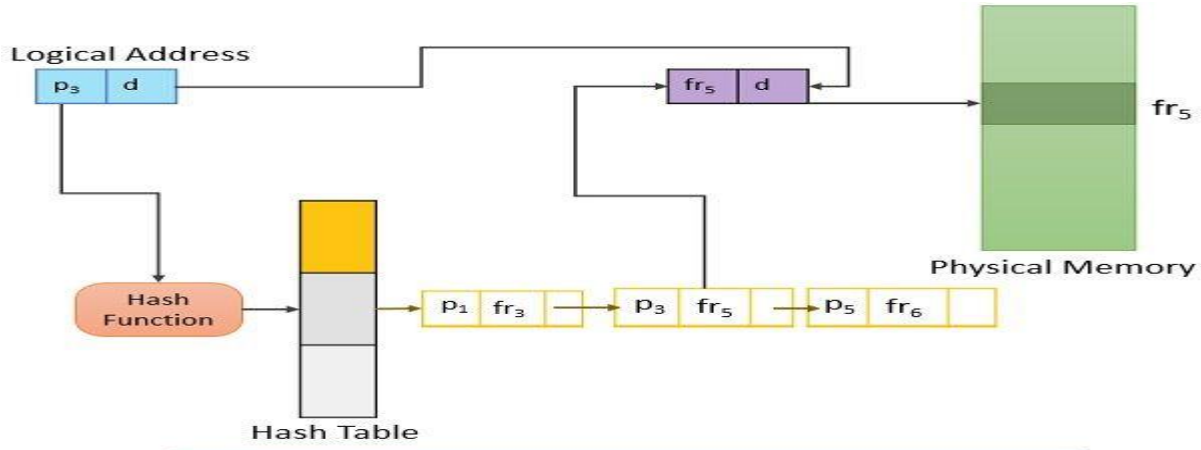
Structure of Two –Level Hierarchical Page Table

Here P1 would act as an index and P2 would act as an offset for the outer page table. Further, the P2 would act as an index and d would act as an offset to inner page table to map the logical address of the page to the physical memory.

Hashed Page Table

When the logical address space is beyond 32 bits in such case the page table is structured using hashed page table. Though we can structure the large page table using the multilevel page table it would consist of a number of levels increases the complexity of the page table.

The hashed page table is a convenient way to structure the page table where logical address space is beyond 32 bits. The hash table has several entries where each entry has a link list. Each link list has a set of linked elements where each element hash to the same location. Each element has three entries page number, frame number and pointer to the next element. We would understand the working of this page table better with the help of an example.



Structure of Hash Page Table

The CPU generates a logical address for the page it needs. Now, this logical address needs to be mapped to the physical address. This logical address has two entries i.e. a page number (P3 in our case) and an offset.

The page number from the logical address is directed to the hash function. The hash function produces a hash value corresponding to the page number. This hash value directs to an entry in the hash table. As we have studied earlier, each entry in the hash table has a link list. Here the page number is compared with the first element's first entry if a match is found then the second entry is checked.

In our example, the logical address includes page number P3 which does not match the first element of the link list as it includes page number P1. So we will move ahead and check the next element; now this element has a page number entry i.e. P3 so further we will check the frame entry of the element which is fr5. To this frame number, we will append the offset provided in the logical address to reach the page's physical address.

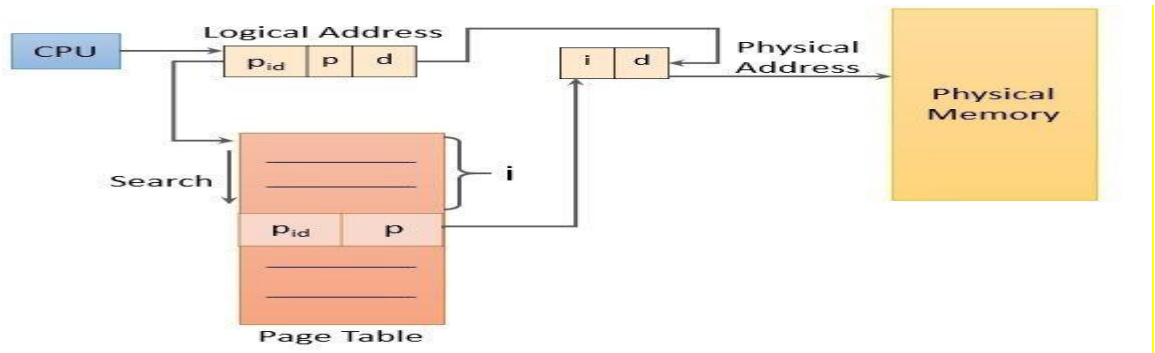
So, this is how the hashed page table works to map the logical address to the physical address.

Inverted Page Table

The concept of normal paging says that every process maintains its own page table which includes the entries of all the pages belonging to the process. The large process may have a page table with millions of entries. Such a page table consumes a large amount of memory.

Consider we have six processes in execution. So, six processes will have some or the other of their page in the main memory which would compel their page tables also to be in the main memory consuming a lot of space. This is the drawback of the paging concept.

The inverted page table is the solution to this wastage of memory. The concept of an inverted page table involves the existence of single-page table which has entries of all the pages (may they belong to different processes) in main memory along with the information of the process to which they are associated. To get a better understanding consider the figure below of inverted page table.



Structure of Inverted Page Table

The CPU generates the logical address for the page it needs to access. This time the logical address consists of three entries process id, page number and the offset. The process id identifies the process, of which the page has been demanded, page number indicates which page of the process has been asked for and the offset value indicates the displacement required.

The match of process id along with associated page number is searched in the page table and say if the search is found at the ith entry of page table then i and offset together generates the physical address for the requested page. This is how the logical address is mapped to a physical address using the inverted page table.

Though the inverted page table reduces the wastage of memory but it increases the search time. This is because the entries in an inverted page table are sorted on the basis of physical address whereas the lookup is performed using logical address. It happens sometimes that the entire table is searched to find the match.

So these are the three techniques that can be used to structure a page table that helps the operating system in mapping the logical address of the page required by CPU to its physical address.