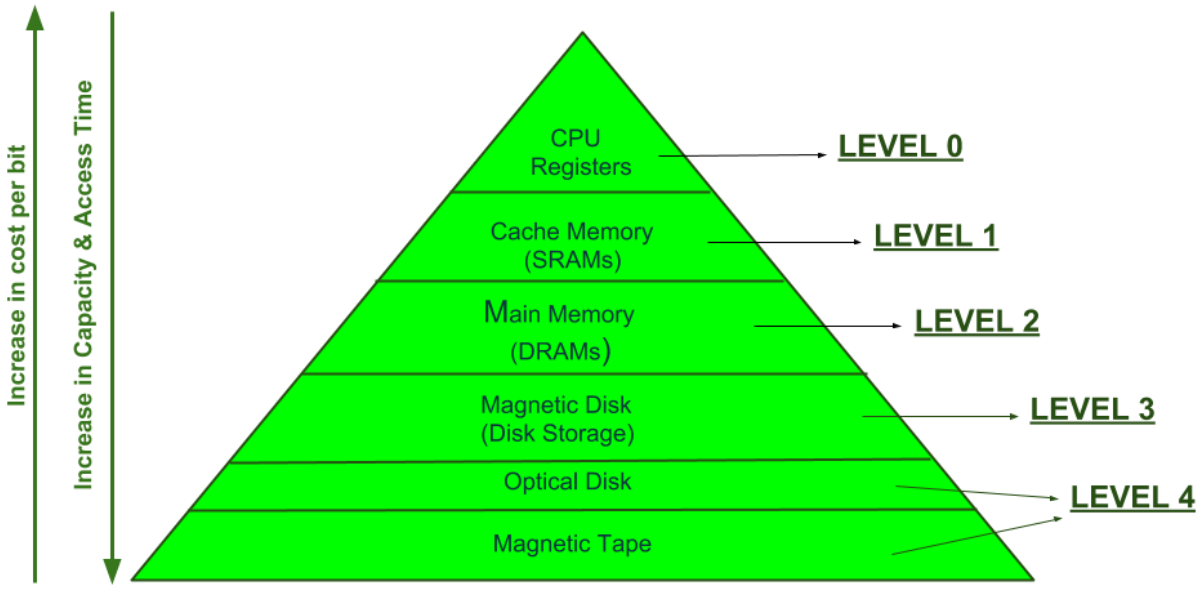
Unit III

Memory Hierarchy Design and its Characteristics

In the Computer System Design, Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time. The Memory Hierarchy was developed based on a program behavior known as locality of references.The figure below clearly demonstrates the different levels of memory hierarchy



MEMORY HIERARCHY DESIGN

This Memory Hierarchy Design is divided into 2 main types:

1. **External Memory or Secondary Memory –**
   Comprising of Magnetic Disk, Optical Disk, Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via I/O Module.

2. **Internal Memory or Primary Memory –**
   Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

We can infer the **following** characteristics of Memory Hierarchy Design from above figure:

1. **Capacity:**
   It is the global volume of information the memory can store. As we move from top to bottom in the Hierarchy, the capacity increases.

2. **Access Time:**
   It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.
3. **Performance:**
   Earlier when the computer system was designed without Memory Hierarchy

design, the speed gap increases between the CPU registers and Main Memory due to large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement was made in the form of Memory Hierarchy Design because of which the performance of the system increases. One of the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.

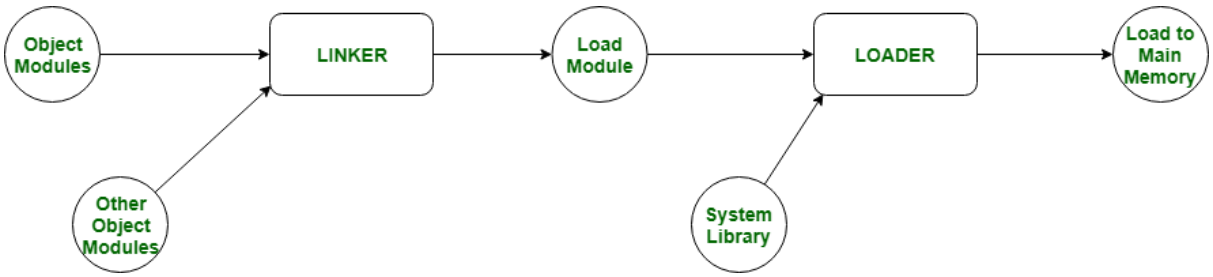
4. **Cost per bit:**
   As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

linking and loading the process

Linking and **Loading** are the utility programs that play a important role in the execution of a program. Linking intakes the object codes generated by the assembler and combines them to generate the executable module. On the other hand, the loading loads this executable module to the main memory for execution.

Loading:
Bringing the program from secondary memory to main memory is called Loading.

Linking:
Establishing the linking between all the modules or all the functions of the program in order to continue the program execution is called linking.



Differences between Linking and Loading:

1. The key difference between linking and loading is that the linking generates the executable file of a program whereas, the loading loads the executable file obtained from the linking into main memory for execution.
2. The linking intakes the object module of a program generated by the assembler. However, the loading intakes the executable module generated by the linking.
3. The linking combines all object modules of a program to generate executable modules it also links the library function in the object module to built-in libraries of the high-level programming language. On the other hand, loading allocates space to an executable module in main memory.

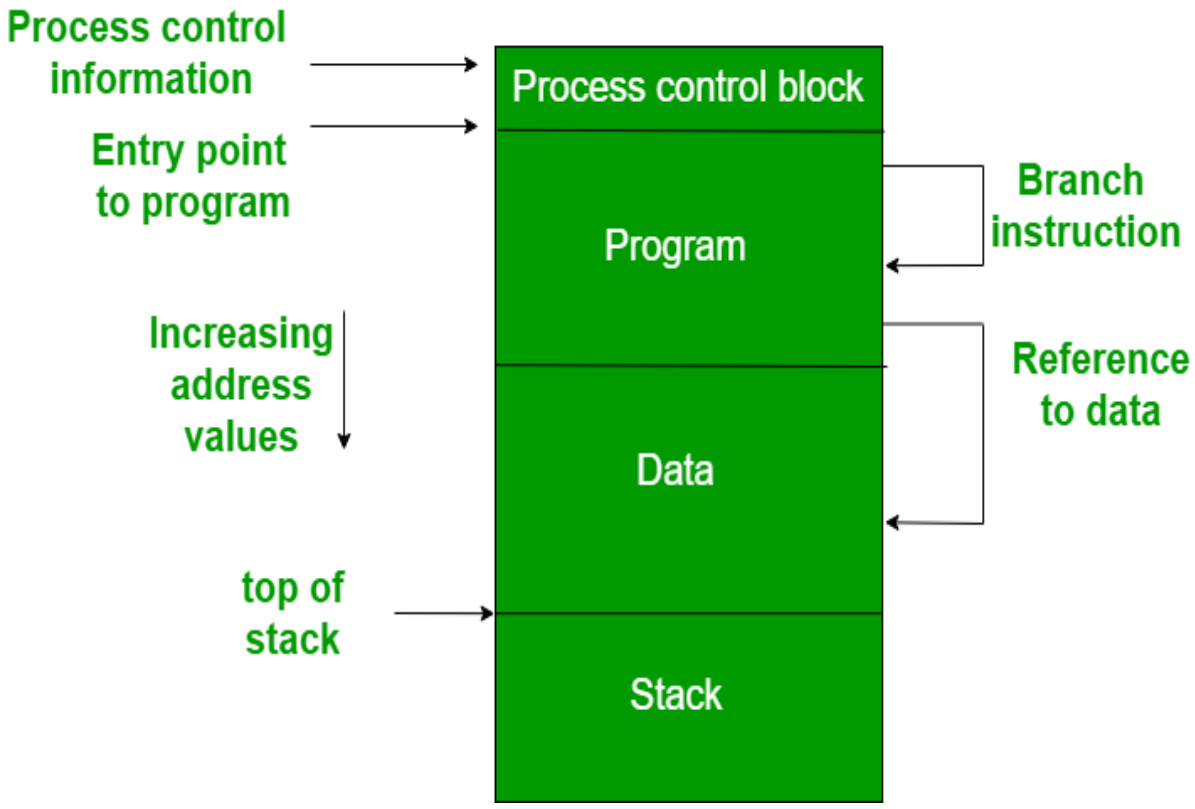
Memory Management requirement

Memory management keeps track of the status of each memory location, whether it is allocated or free. It allocates the memory dynamically to the programs at their request and frees it for reuse when it is no longer needed. Memory management meant to satisfy some requirements that we should keep in mind.

These Requirements of memory management are:

1. **Relocation –** The available memory is generally shared among a number of processes in a multiprogramming system, so it is not possible to know in advance which other programs will be resident in main memory at the time of execution of his program. Swapping the active processes in and out of the main memory enables the operating system to have a larger pool of ready-to-execute process.

   When a program gets swapped out to a disk memory, then it is not always possible that when it is swapped back into main memory then it occupies the previous memory location, since the location may still be occupied by another process. We may need to **relocate** the process to a different area of memory. Thus there is a possibility that program may be moved in main memory due to swapping.



The figure depicts a process image. The process image is occupying a continuous region of main memory. The operating system will need to know many things including the location of process control information, the execution stack, and the code entry. Within a program, there are memory references in various instructions and these are called logical addresses.

After loading of the program into main memory, the processor and the operating system must be able to translate logical addresses into physical addresses. Branch instructions contain the address of the next instruction to be executed. Data reference instructions contain the address of byte or word of data referenced.

1. **Protection –** There is always a danger when we have multiple programs at the same time as one program may write to the address space of another program. So every process must be protected against unwanted interference when other process tries to write in a process whether accidental or incidental. Between relocation and protection requirement a trade-off occurs as the satisfaction of relocation requirement increases the difficulty of satisfying the protection requirement.

   Prediction of the location of a program in main memory is not possible, that's why it is impossible to check the absolute address at compile time to assure protection. Most of the programming language allows the dynamic calculation of address at run time. The memory protection requirement must be satisfied by the processor rather than the operating system because the operating system can hardly control a process when it occupies the processor. Thus it is possible to check the validity of memory references.

2. **Sharing –** A protection mechanism must have to allow several processes to access the same portion of main memory. Allowing each processes access to the same copy of the program rather than have their own separate copy has an advantage.

   For example, multiple processes may use the same system file and it is natural to load one copy of the file in main memory and let it shared by those processes. It is the task of Memory management to allow controlled access to the shared areas of memory without compromising the protection. Mechanisms are used to support relocation supported sharing capabilities.

3. **Logical organization –** Main memory is organized as linear or it can be a one-dimensional address space which consists of a sequence of bytes or words. Most of the programs can be organized into modules, some of those are unmodifiable (read-only, execute only) and some of those contain data that can be modified. To effectively deal with a user program, the operating system and computer hardware must support a basic module to provide the required protection and sharing. It has the following advantages:

   - Modules are written and compiled independently and all the references from one module to another module are resolved by `the system at run time.
   - Different modules are provided with different degrees of protection.

- There are mechanisms by which modules can be shared among processes. Sharing can be provided on a module level that lets the user specify the sharing that is desired.

4. **Physical organization –** The structure of computer memory has two levels referred to as main memory and secondary memory. Main memory is relatively very fast and costly as compared to the secondary memory. Main memory is volatile. Thus secondary memory is provided for storage of data on a long-term basis while the main memory holds currently used programs. The major system concern between main memory and secondary memory is the flow of information and it is impractical for programmers to understand this for two reasons:

    - The programmer may engage in a practice known as overlaying when the main memory available for a program and its data may be insufficient. It allows different modules to be assigned to the same region of memory. One disadvantage is that it is time-consuming for the programmer.
    - In a multiprogramming environment, the programmer does not know how much space will be available at the time of coding and where that space will be located inside the memory.

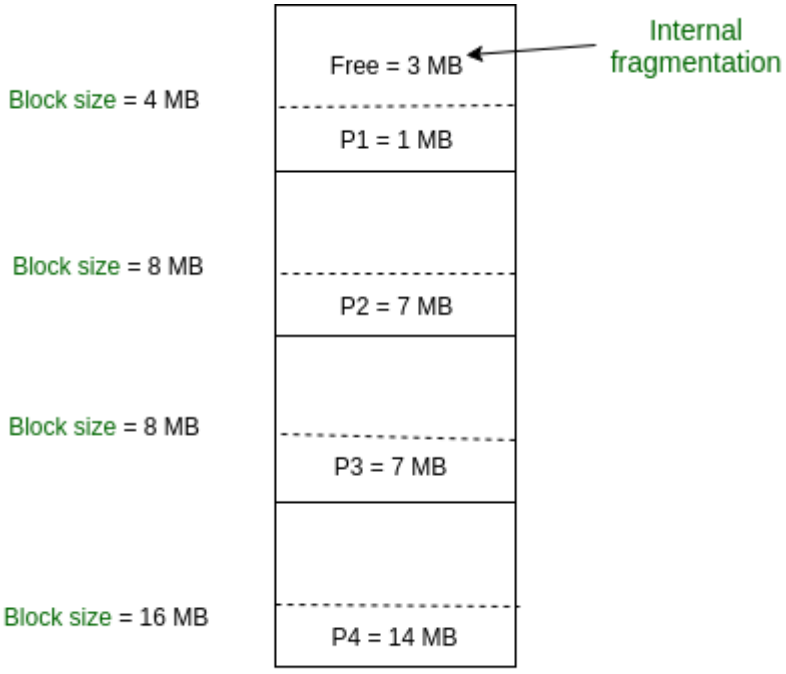
Fixed (or static) Partitioning

In operating systems, Memory Management is the function responsible for allocating and managing computer's main memory. Memory Management function keeps track of the status of each memory location, either allocated or free to ensure effective and efficient use of Primary Memory.

There are two Memory Management Techniques: **Contiguous**, and **Non-Contiguous**. In Contiguous Technique, executing process must be loaded entirely in main-memory. Contiguous Technique can be divided into:

1. Fixed (or static) partitioning
2. Variable (or dynamic) partitioning

Fixed Partitioning:
This is the oldest and simplest technique used to put more than one processes in the main memory. In this partitioning, number of partitions (non-overlapping) in RAM are **fixed but size** of each partition may or **may not be same**. As it is **contiguous** allocation, hence no spanning is allowed. Here partition are made before execution or during system configure.



Fixed size partition

As illustrated in above figure, first process is only consuming 1MB out of 4MB in the main memory.
Hence, Internal Fragmentation in first block is (4-1) = 3MB.
Sum of Internal Fragmentation in every block = (4-1)+(8-7)+(8-7)+(16-14)= 3+1+1+2 = 7MB.

Suppose process P5 of size 7MB comes. But this process cannot be accommodated inspite of available free space because of contiguous allocation (as spanning is not allowed). Hence, 7MB becomes part of External Fragmentation.

There are some advantages and disadvantages of fixed partitioning.

Advantages of Fixed Partitioning –

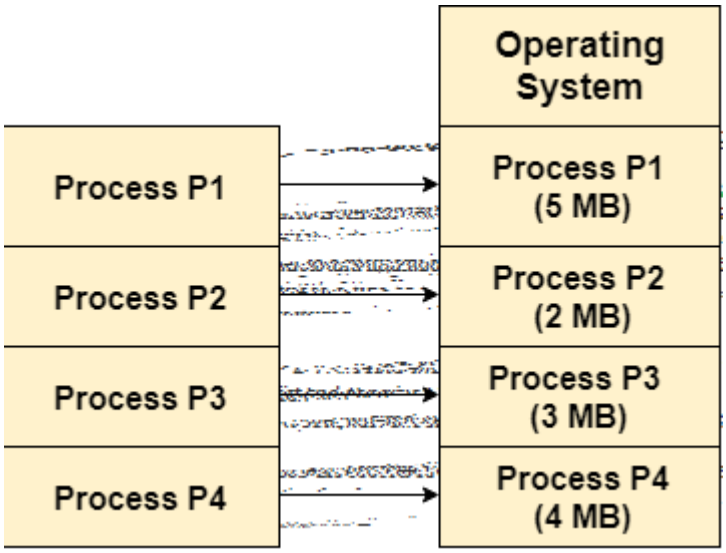
1. **Easy to implement:**
   Algorithms needed to implement Fixed Partitioning are easy to implement. It simply requires putting a process into certain partition without focussing on the emergence of Internal and External Fragmentation.
2. **Little OS overhead:**
   Processing of Fixed Partitioning require lesser excess and indirect computational power.

Disadvantages of Fixed Partitioning –

1. **Internal Fragmentation:**
   Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This can cause internal fragmentation.
2. **External Fragmentation:**
   The total unused space (as stated above) of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form (as spanning is not allowed).
3. **Limit process size:**
   Process of size greater than size of partition in Main Memory cannot be accommodated. Partition size cannot be varied according to the size of incoming process's size. Hence, process size of 32MB in above stated example is invalid.
4. **Limitation on Degree of Multiprogramming:**
   Partition in Main Memory are made before execution or during system configure. Main Memory is divided into fixed number of partition. Suppose if there are partitions in RAM and are the number of processes, then condition must be fulfilled. Number of processes greater than number of partitions in RAM is invalid in Fixed Partitioning.

Dynamic Partitioning

Dynamic partitioning tries to overcome the problems caused by fixed partitioning. In this technique, the partition size is not declared initially. It is declared at the time of process loading. The first partition is reserved for the operating system. The remaining space is divided into parts. The size of each partition will be equal to the size of the process. The partition size varies according to the need of the process so that the internal fragmentation can be avoided.

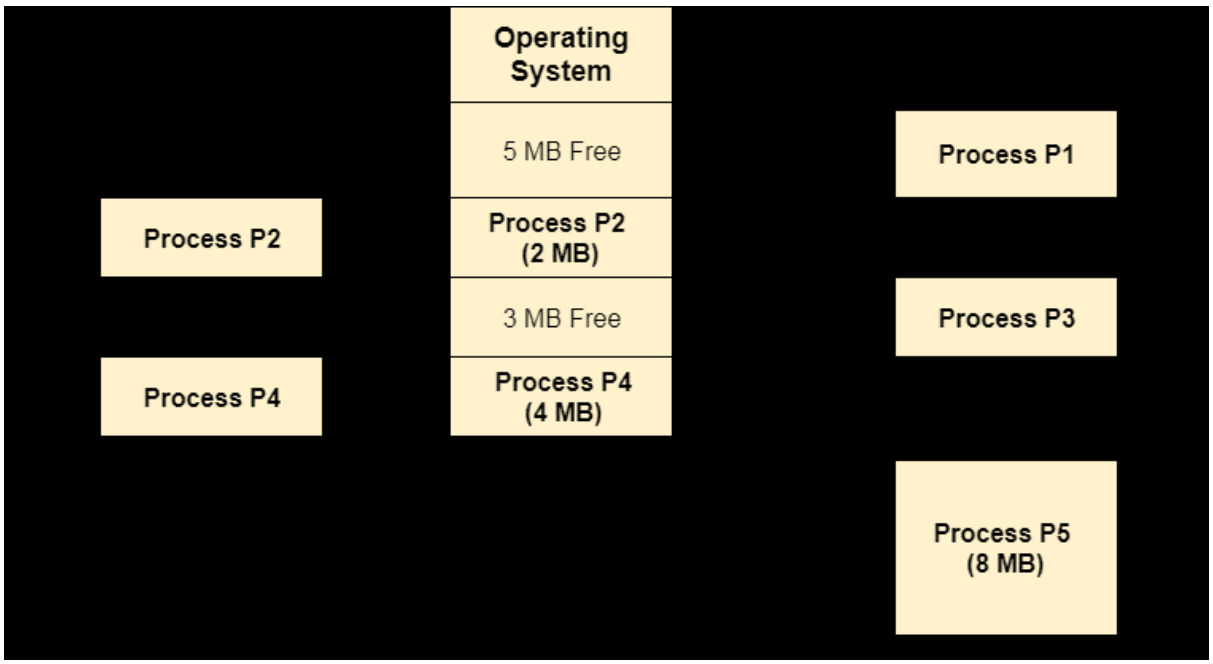


Advantages of Dynamic Partitioning over fixed partitioning

1. No Internal Fragmentation Given the fact that the partitions in dynamic partitioning are created according to the need of the process, It is clear that there will not be any internal fragmentation because there will not be any unused remaining space in the partition. 2. No Limitation on the size of the process In Fixed partitioning, the process with the size greater than the size of the largest partition could not be executed due to the lack of sufficient contiguous memory. Here, In Dynamic partitioning, the process size can't be restricted since the partition size is decided according to the process size. 3. Degree of multiprogramming is dynamic Due to the absence of internal fragmentation, there will not be any unused space in the partition hence more processes can be loaded in the memory at the same time. Disadvantages of dynamic partitioning

External Fragmentation Absence of internal fragmentation doesn't mean that there will not be external fragmentation. Let's consider three processes P1 (1 MB) and P2 (3 MB) and P3 (1 MB) are being loaded in the respective partitions of the main memory. After some time P1 and P3 got completed and their assigned space is freed. Now there are two unused partitions (1 MB and 1 MB) available in the main memory but they cannot be used to load a 2 MB process in the memory since they are not contiguously located. The rule says that the process must be contiguously present in

the main memory to get executed. We need to change this rule to avoid 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

SIMPLE AND MULTILEVEL PAGING

PAGING:

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

Logical Address or Virtual Address (represented in bits):

An address generated by the CPU

Logical Address Space or Virtual Address Space(represented in words or bytes): The set of all logical addresses generated by a program

Physical Address (represented in bits):

An address actually available on memory unit.

Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses

Example:

If Logical Address = 31 bit, then Logical Address Space = 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

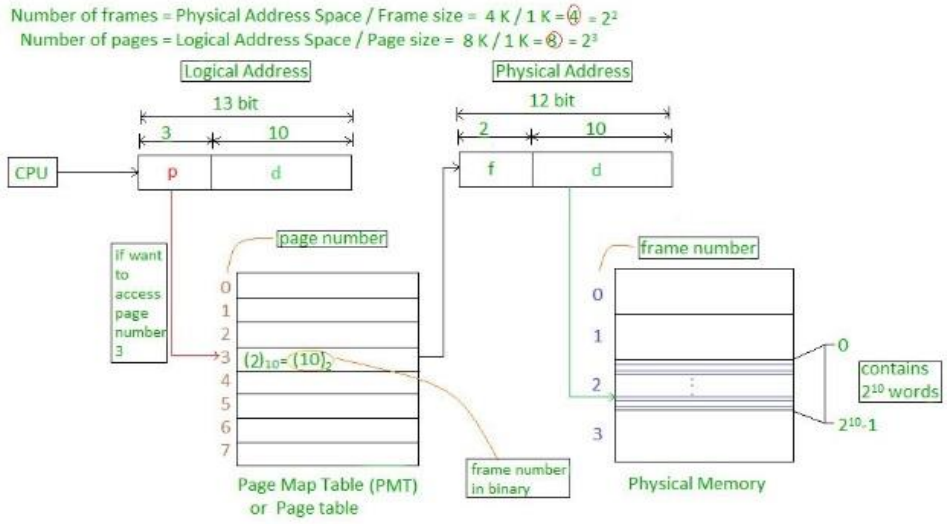
If Physical Address = 22 bit, then Physical Address Space = 222 words = 4 M words (1 M = 220)

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.

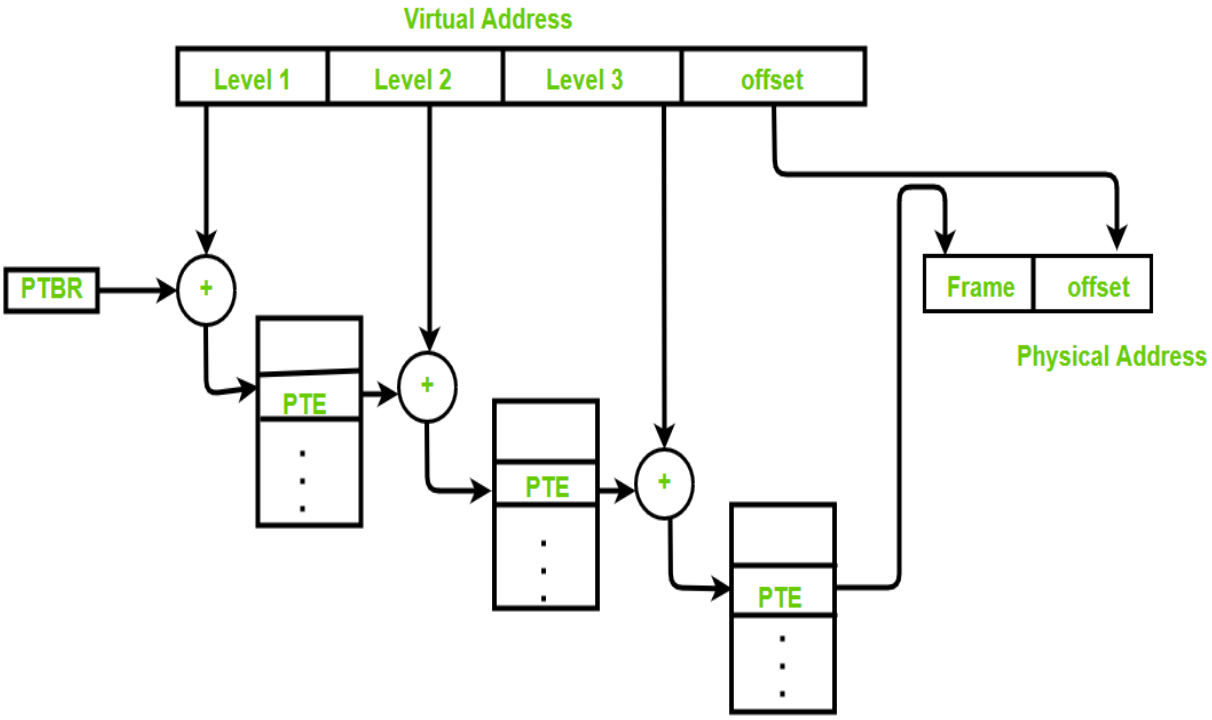


MULTILEVEL PAGING:

Multilevel Paging is a paging scheme which consist of two or more levels of page tables in a hierarchical manner. It is also known as hierarchical paging. The entries of the level 1 page table are pointers to a level 2 page table and entries of the level 2 page tables are pointers to a level 3 page table and so on. The entries of the last level page table are stores actual frame information. Level 1 contain single page table and address of that table is stored in PTBR (Page Table Base Register).

Virtual address:

In multilevel paging whatever may be levels of paging all the page tables will be stored in main memory.So it requires more than one memory access to get the physical address of page frame. One access for each level needed. Each page table entry **except** the last level page table entry contains base address of the next level page table.



3 Level paging system

Reference to actual page frame:

- Reference to PTE in level 1 page table = PTBR value + Level 1 offset present in virtual address.
- Reference to PTE in level 2 page table = Base address (present in Level 1 PTE) + Level 2 offset (present in VA).
- Reference to PTE in level 3 page table= Base address (present in Level 2 PTE) + Level 3 offset (present in VA).
- Actual page frame address = PTE (present in level 3).Generally the page table size will be equal to the size of page.

Simple Segmentation

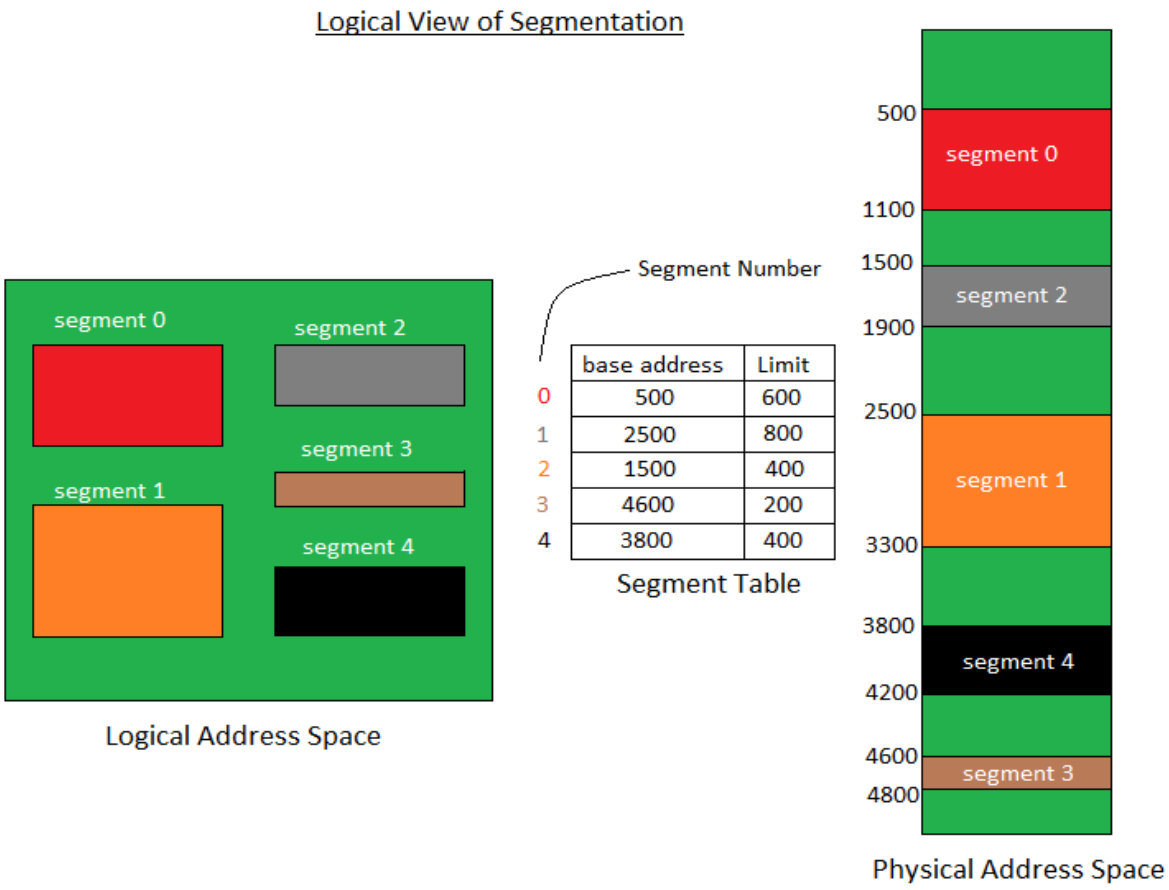
A process is divided into Segments. The chunks that a program is divided into which are not necessarily all of the same sizes are called segments. Segmentation gives user's view of the process which paging does not give. Here the user's view is mapped to physical memory. There are types of segmentation:

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.

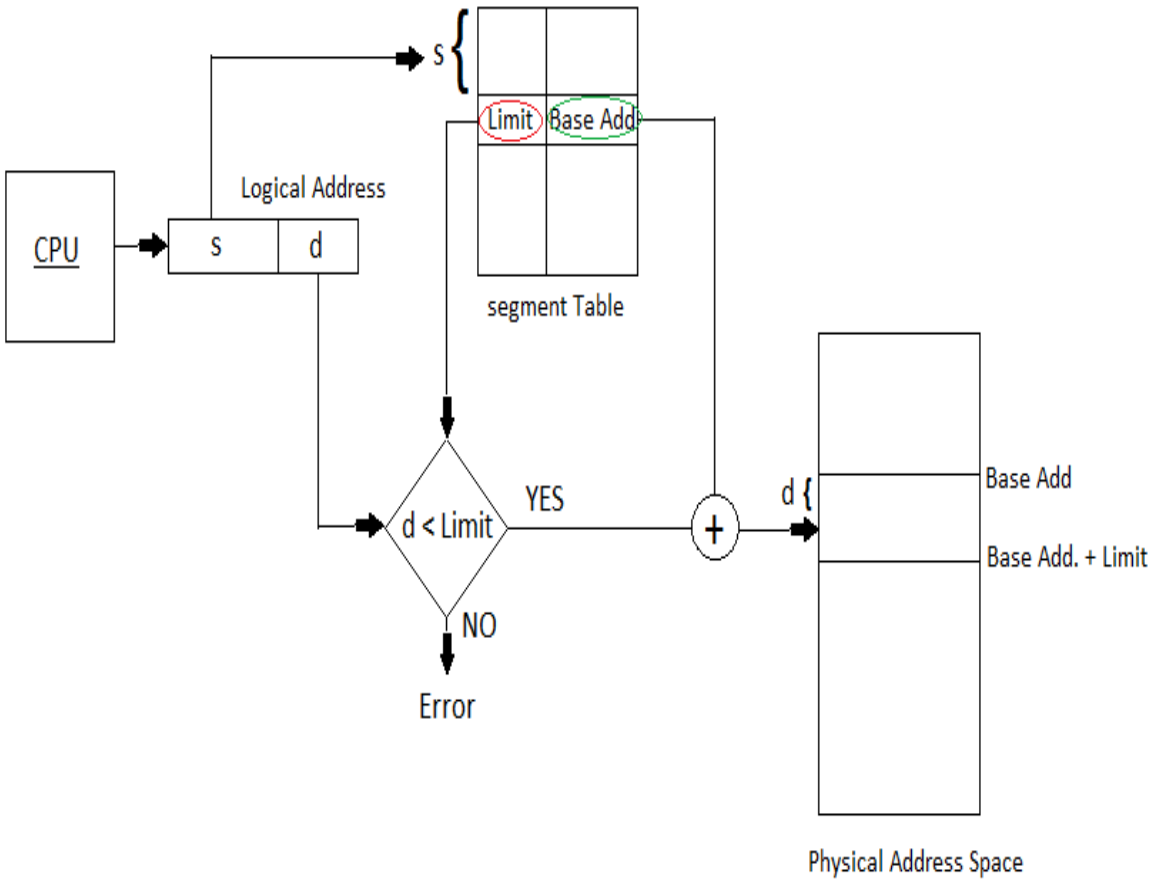
There is no simple relationship between logical addresses and physical addresses in segmentation. A table stores the information about all such segments and is called Segment Table.

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.

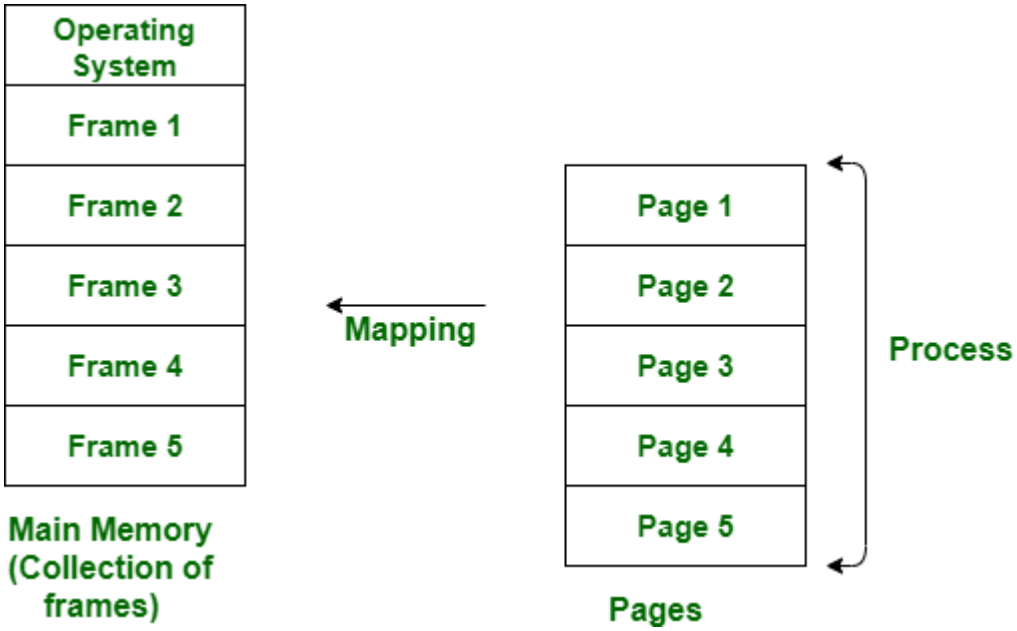
Disadvantage of Segmentation –

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

segmentation and paging

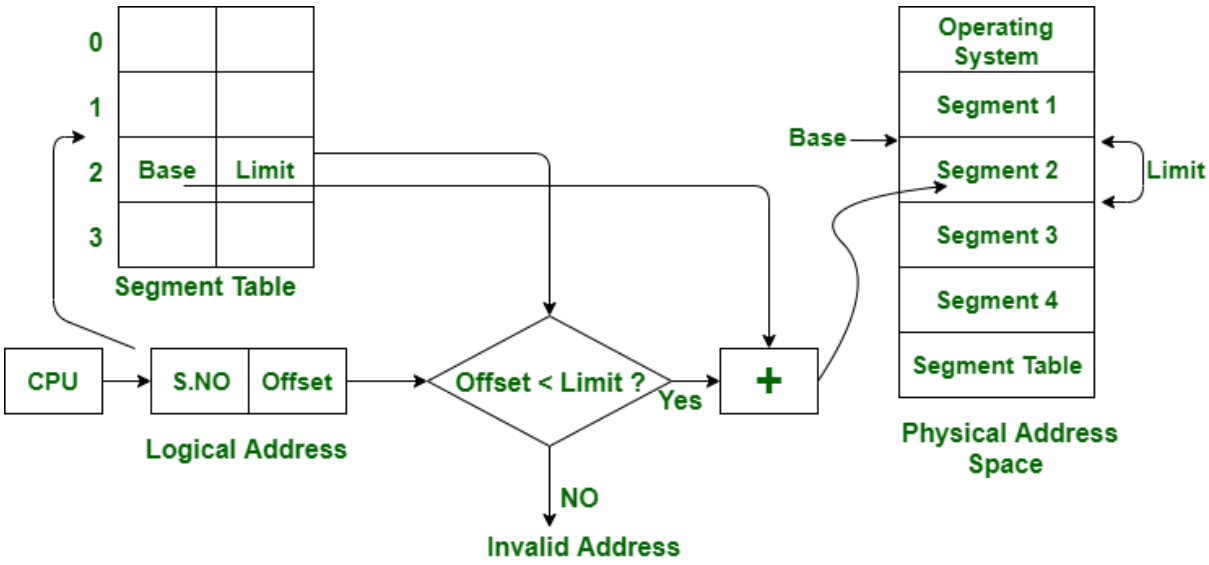
Paging**:**
Paging is a method or techniques which is used for non-contiguous memory allocation. It is a fixed size partitioning theme (scheme). In paging, both main memory and secondary memory are divided into equal fixed size partitions. The partitions of secondary memory area unit and main memory 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	Segmentation could result in external

S.NO	Paging	Segmentation
	fragmentation.	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.

Virtual Memory

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.

Demand Paging :

The process of loading the page into memory on demand (whenever page fault occurs) is known as demand paging.

The process includes the following steps :

1. If CPU try to refer a page that is currently not available in the main memory, it generates an interrupt indicating memory access fault.
2. The OS puts the interrupted process in a blocking state. For the execution to proceed the OS must bring the required page into the memory.
3. The OS will search for the required page in the logical address space.
4. The required page will be brought from logical address space to physical address space. The page replacement algorithms are used for the decision making of replacing the page in physical address space.
5. The page table will updated accordingly.
6. The signal will be sent to the CPU to continue the program execution and it will place the process back into ready state.
7. Hence whenever a page fault occurs these steps are followed by the operating system and the required page is brought into memory.

Advantages :

● More processes may be maintained in the main memory: Because we are going to load only some of the pages of any particular process, there is room for more processes. This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in the ready state at any particular time.

● A process may be larger than all of main memory: One of the most fundamental restrictions in programming is lifted. A process larger than the main memory can be executed because of demand paging. The OS itself loads pages of a process in main memory as required.

● It allows greater multiprogramming levels by using less of the available (primary) memory for each process.

Page Fault Service Time :

The time taken to service the page fault is called as page fault service time. The page fault service time includes the time taken to perform all the above six steps.

Let Main memory access time is: m

Page fault service time is: s

Page fault rate is : p

Then, Effective memory access time = (p*s) + (1-p)*m

Sample reference,

Let Main memory access time is: m

Page fault service time is: s

Page fault rate is : p

Then, Effective memory access time = (p*s) + (1-p)*m.

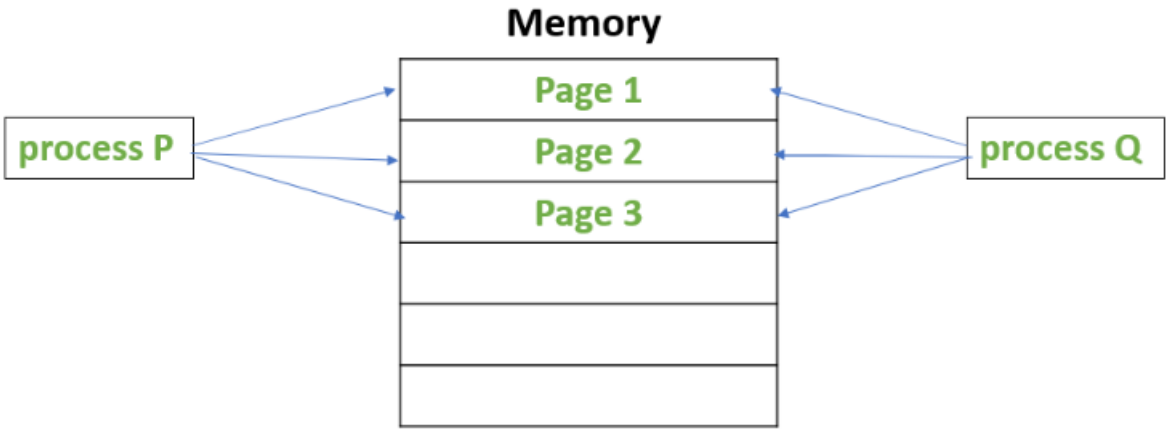
Copy on write

Copy on Write or simply COW is a resource management technique. One of its main use is in the implementation of the fork system call in which it shares the virtual memory(pages) of the OS.

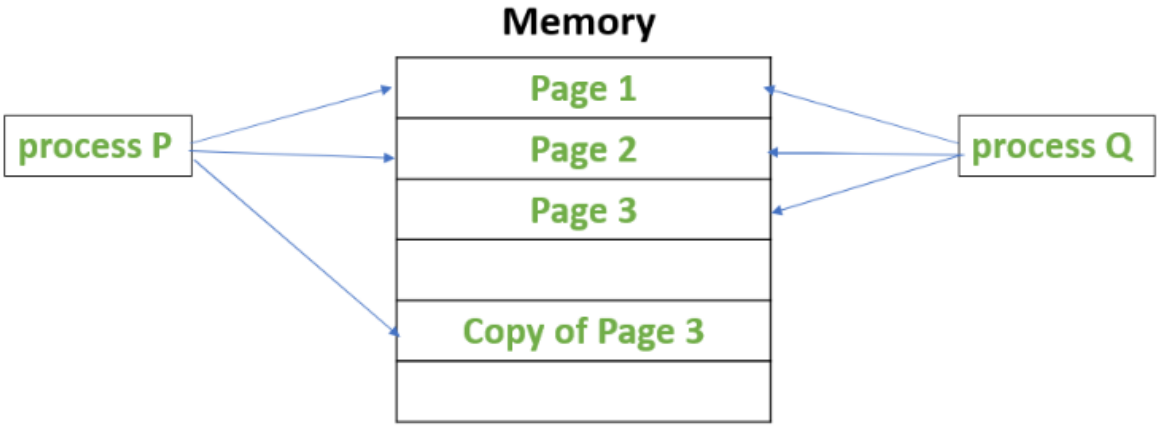
In UNIX like OS, fork() system call creates a duplicate process of the parent process which is called as the child process.

The idea behind a copy-on-write is that when a parent process creates a child process then both of these processes initially will share the same pages in memory and these shared pages will be marked as copy-on-write which means that if any of these processes will try to modify the shared pages then only a copy of these pages will be created and the modifications will be done on the copy of pages by that process and thus not affecting the other process.

Suppose, there is a process P that creates a new process Q and then process P modifies page 3.
The below figures shows what happens before and after process P modifies page 3.

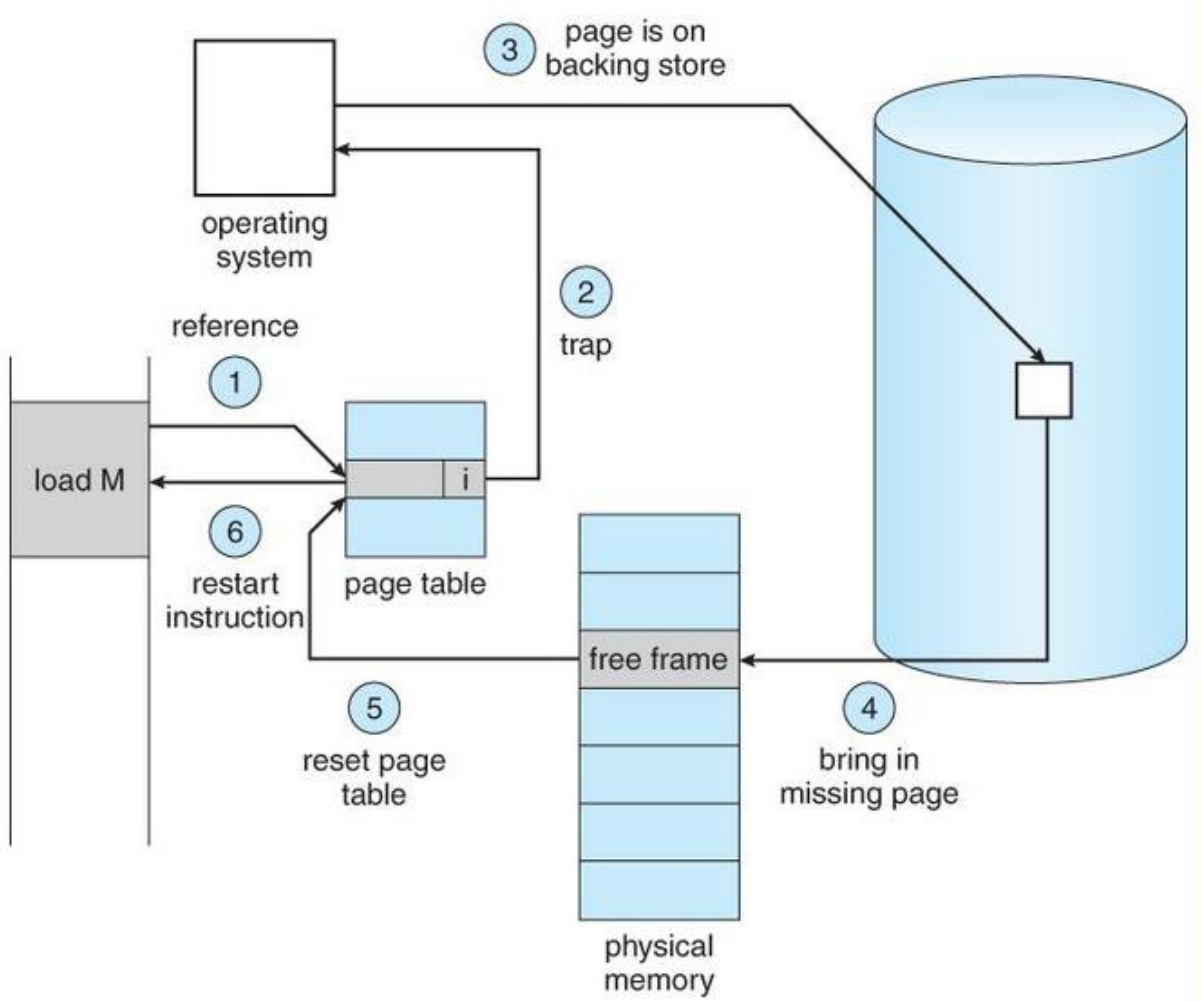


Before process P modifies Page 3



After process P modifies Page 3

A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM. So when page fault occurs then following sequence of events happens :



- The computer hardware traps to the kernel and program counter (PC) is saved on the stack. Current instruction state information is saved in CPU registers.
- An assembly program is started to save the general registers and other volatile information to keep the OS from destroying it.
- Operating system finds that a page fault has occurred and tries to find out which virtual page is needed. Some times hardware register contains this required information. If not, the operating system must retrieve PC, fetch instruction and find out what it was doing when the fault occurred.
- Once virtual address caused page fault is known, system checks to see if address is valid and checks if there is no protection access problem.
- If the virtual address is valid, the system checks to see if a page frame is free. If no frames are free, the page replacement algorithm is run to remove a page.

- If frame selected is dirty, page is scheduled for transfer to disk, context switch takes place, fault process is suspended and another process is made to run until disk transfer is completed.
- As soon as page frame is clean, operating system looks up disk address where needed page is, schedules disk operation to bring it in.
- When disk interrupt indicates page has arrived, page tables are updated to reflect its position, and frame marked as being in normal state.
- Faulting instruction is backed up to state it had when it began and PC is reset. Faulting is scheduled, operating system returns to routine that called it.
- Assembly Routine reloads register and other state information, returns to user space to continue execution.

Demand Segmentation - Combined Demand

Definition:

➢ Demand segmentation is defined as the practice of analyzing demand data often divided into smaller sections (segments) to help measure performance or improve service levels. Demand segmentation analysis can be performed on pre-defined company segments, including products or locations.

➢ In Operating Systems, Segmentation is a memory management technique in which, the memory is divided into the variable size parts. Each part is known as segment which can be allocated to a process.

Similarly, why do we need segmentation in OS?

➢ Segmentation in Operating System. A process is divided into Segments. Segmentation gives user's view of the process which paging does not give. Here the user's view is mapped to physical memory.

Secondly, what is segmentation in operating system with examples?

➢ Segmentation in Operating System. Segmentation is more like, user's end of memory management scheme. Just like paging, it divides or segments the memory. But, while paging divides memory into fixed size, segmentation divides the memory in variable segments, which are then loaded into logical memory space.

➢ Operating system also uses demand segmentation, which is similar to demand paging. Operating system to uses demand segmentation where there is insufficient hardware available to implement ‘Demand Paging’.

Demand Segmentation - Combined Demand

Implementing Segmentation:

➢ Compiler needs to generate virtual address whose upper order bits are a segment number.

➢ Segmentation can be combined with a dynamic or static relocation system,

o Each segment is allocated a contiguous piece of physical memory.

o External fragmentation can be a problem again.

➢ Similar memory mapping algorithm as paging. We need something like the TLB if programs can have lots of segments.

➢ Let's combine the ease of sharing we get from segments with efficient memory utilization we get from pages.

Combining Segment and Paging:

➢ Treat virtual address space as a collection of segments (logical units) of arbitrary sizes.

➢ Treat physical memory as a sequence of fixed size page frames.

➢ Segments are typically larger than page frames.

❖ Map a logical segment onto multiple page frames by paging the segments.

Demand Segmentation - Combined Demand

Advantages of Segmentation:

➢ No internal fragmentation.

➢ Segment tables consume less memory than page tables (only one entry per actual segment as opposed to one entry per page in paging method).

➢ Because of the small segment table, memory reference is easy.

➢ Lends itself to sharing data among processes.

➢ Lends itself to protection.

➢ As the individual lines of a page do not form one logical unit, it is not possible to set a particular access right to a page.

➢ Note that each segment could be set up an access right.

Disadvantages of Segmentation:

➢ It can have external fragmentation.

➢ it is difficult to allocate contiguous memory to variable sized partition.

➢ Costly memory management algorithms.

SEGMENTATION

In Operating Systems, Segmentation is a memory management technique in which, the memory is divided into the variable size parts. Each part is known as segment which can be allocated to a process.

The details about each segment are stored in a table called as segment table. Segment table is stored in one (or many) of the segments.

Segment table contains mainly two information about segment:

1. Base: It is the base address of the segment
2. Limit: It is the length of the segment.

WHY SEGMENTATION IS REQUIRED?

Till now, we were using Paging as our main memory management technique. Paging is more close to Operating system rather than the User. It divides all the process into the form of pages regardless of the fact that a process can have some relative parts of functions which needs to be loaded in the same page.

Operating system doesn't care about the User's view of the process. It may divide the same function into different pages and those pages may or may not be loaded at the same time into the memory. It decreases the efficiency of the system.

It is better to have segmentation which divides the process into the segments. Each segment contain same type of functions such as main function can be included in one segment and the library functions can be included in the other segment

PAGING

Paging is a storage mechanism that allows OS to retrieve

processes from the secondary storage into the main memory in the form of pages. In the Paging method, the main memory is divided into small fixed-size blocks of physical memory, which is called frames. The size of a frame should be kept the same as that of a page to have maximum utilization of the main memory and to avoid external fragmentation. Paging is used for faster access to data, and it is a logical concept.

n computer science, thrashing occurs when a computer's virtual memory resources are overused, leading to a constant state of paging and page faults, inhibiting most application-level processing.[1] This causes the performance of the computer to degrade or collapse. The situation can continue indefinitely until either the user closes some running applications or the active processes free up additional virtual memory resources.

After completing initialization, most programs operate on a small number of code and data pages compared to the total memory the program requires. The pages most frequently accessed are called the working set. When the working set is a small percentage of the system's total number of pages, virtual memory systems work most efficiently and an insignificant amount of computing is spent resolving page faults. As the working set grows, resolving page faults remains manageable until the growth reaches a critical point. Then faults go up dramatically and the time spent resolving them overwhelms time spent on the computing the program was written to do. This condition is referred to as thrashing. Thrashing occurs on a program that works with huge data structures, as its large working set causes continual page faults that drastically slow down the system. Satisfying page faults may require freeing pages that will soon have to be re-read from disk.

The term is also used for various similar phenomena, particularly

movement between other levels of the memory hierarchy, where a process progresses slowly because significant time is being spent acquiring resources.

"Thrashing" is also used in contexts other than virtual memory systems; for example, to describe cache issues in computing or silly window syndrome in networking

WORKING SET MODEL

The working set model states that a process can be in RAM if and only if all of the pages that it is currently using (often approximated by the most recently used pages) can be in RAM. The model is an all or nothing model, meaning if the pages it needs to use increases, and there is no room in RAM, the process is swapped out of memory to free the memory for other processes to use.

Often a heavily loaded computer has so many processes queued up that, if all the processes were allowed to run for one scheduling time slice, they would refer to more pages than there is RAM, causing the computer to "thrash".

By swapping some processes from memory, the result is that processes—even processes that were temporarily removed from memory—finish much sooner than they would if the computer attempted to run them all at once. The processes also finish much sooner than they would if the computer only ran one process at a time to completion since it allows other processes to run and make progress during times that one process is waiting on the hard drive or some other global resource.

In other words, the working set strategy prevents thrashing while keeping the degree of multiprogramming as high as possible. Thus, it optimizes CPU utilization and throughput.