# UNIT III MEMORY MANAGEMENT

***Main Memory - Swapping - Contiguous Memory Allocation - Paging - Structure of the Page Table - Segmentation, Segmentation with paging; Virtual Memory - Demand Paging - Copy on Write - Page Replacement - Allocation of Frames - Thrashing.***

## MAIN MEMORY

Memory can be defined as a collection of data in a specific format.
It is used to store instructions and process data.
The memory comprises a large array or group of words or bytes, each with its own location.
The primary motive of a computer system is to execute programs. These programs, along with the information they access, should be in the main memory during execution.
The CPU fetches instructions from memory according to the value of the program counter.
To achieve a degree of multiprogramming and proper utilization of memory, memory management is important.
Many memory management methods exist, reflecting various approaches, and the effectiveness of each algorithm depends on the situation.

### What is Main Memory?

The main memory is central to the operation of a modern computer.
Main Memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions. Main memory is a repository of rapidly available information shared by the CPU and I/O devices.
Main memory is the place where programs and information are kept when the processor is effectively utilizing them.
Main memory is associated with the processor, so moving instructions and information into and out of the processor is extremely fast.
Main memory is also known as RAM (Random Access Memory).
This memory is a volatile memory. RAM lost its data when a power interruption occurs.

### What is Memory Management?

In a multiprogramming computer, the operating system resides in a part of memory and the rest is used by multiple processes. The task of subdividing the memory among different processes is called memory management. Memory management is a method in the operating system to manage operations between main memory and disk during process execution. The main aim of memory management is to achieve efficient utilization of memory.

### Why Memory Management is required?

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

**Logical and Physical Address Space:**

**Logical Address space:**

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

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

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

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

**Static and Dynamic Loading:**

Loading a process into the main memory is done by a loader.

There are two different types of loading:

**Static loading:-** loading the entire program into a fixed address. It requires more memory space.

**Dynamic loading:-** The entire program and all data of a process must be in physical memory for the process to execute. So, the size of a process is limited to the size of physical memory.

To gain proper memory utilization, dynamic loading is used. In dynamic loading, a routine is not loaded until it is called. All routines are residing on disk in a relocatable load format.

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

**Static and Dynamic linking:**

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

**Static linking:**

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

**Dynamic linking:**

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

**SWAPPING**

A process needs to be in memory for execution.

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.

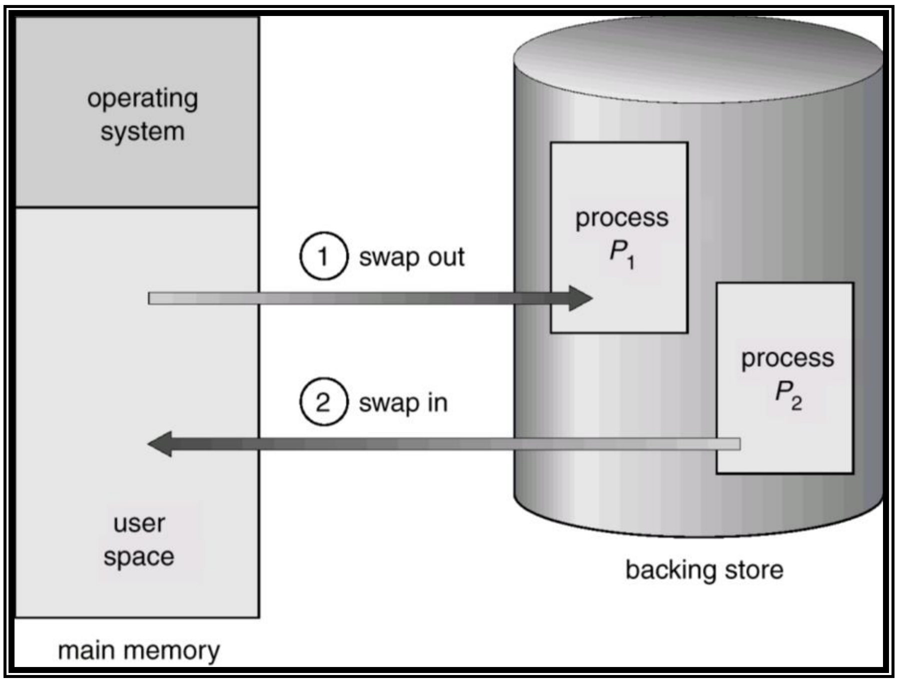
Swapping is the technique used by an operating system for efficient management of memory space of a computer system.

Swapping involves performing two tasks called swapping in and swapping out.

The task of placing the pages or blocks of data from the hard disk to the main memory is called swapping in.

The task of removing pages or blocks of data from main memory to the hard disk is called swapping out.

The swapping technique is useful when larger program is to be executed or some operations have to perform on a large file.



**Swapping of two processes using a disk as a backing store**

**Memory Management Techniques:**

The memory management techniques can be classified into following main categories:

- Contiguous memory allocation
- Non-Contiguous memory allocation

**Contiguous memory allocation:**

In contiguous memory allocation each process is contained in a single contiguous block of memory. Memory is divided into several fixed size partitions. Each partition contains exactly one process. When a partition is free, a process is selected from the input queue and loaded into it. The free blocks of memory are known as holes. The set of holes is searched to determine which hole is best to allocate.

**Memory Protection**

Memory protection is a phenomenon by which we control memory access rights on a computer. The main aim of it is to prevent a process from accessing memory that has not been allocated to it. Hence prevents a bug within a process from affecting other processes, or the operating system itself, and instead results in a segmentation fault or storage violation exception being sent to the disturbing process, generally killing of process.

**Memory Allocation**

Memory allocation is a process by which computer programs are assigned memory or space. It is of three types:

**First Fit:**

The first hole that is big enough is allocated to program. i.e. In the first fit, the first available free hole fulfills the requirement of the process allocated.

**Best Fit:**

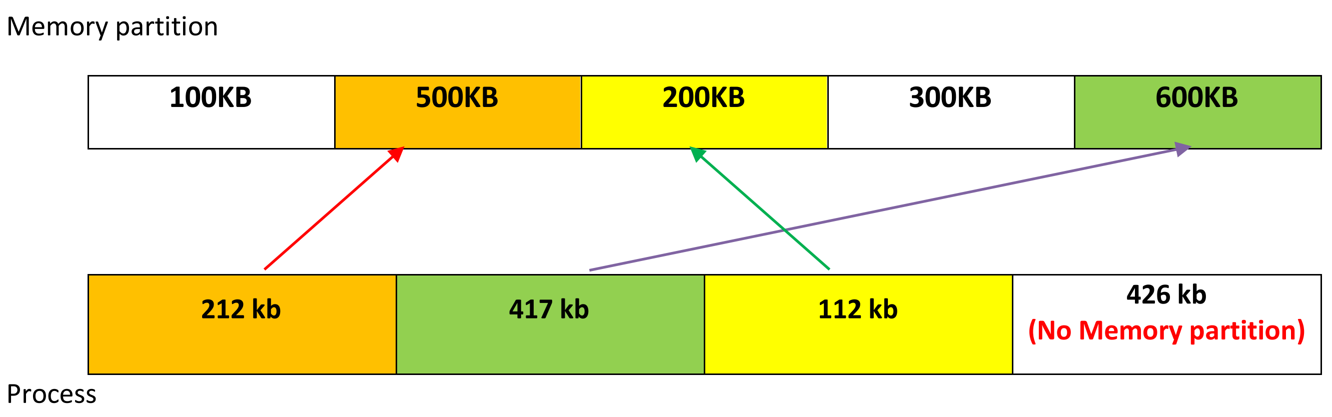
The smallest hole that is big enough is allocated to program. i.e. In the best fit, allocate the smallest hole that is big enough to process requirements. For this, we search the entire list, unless the list is ordered by size.

**Worst Fit:**

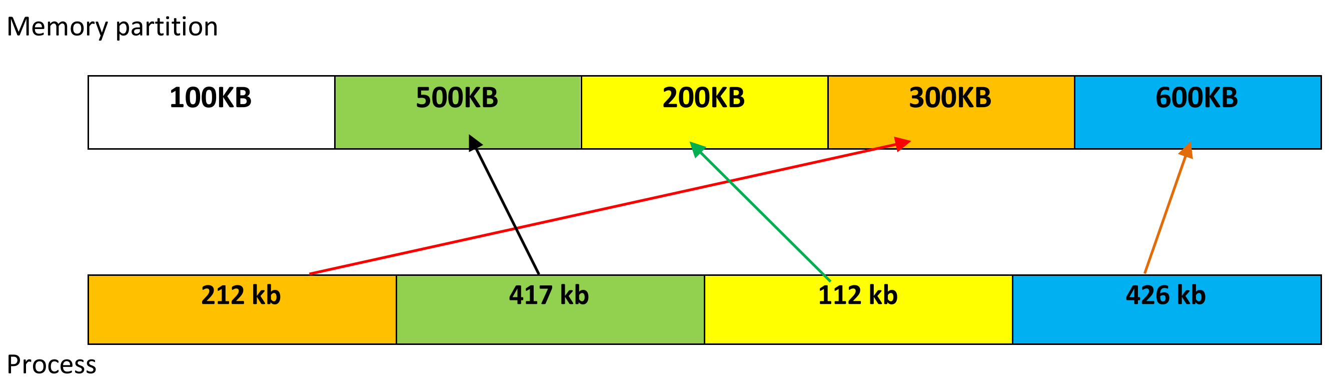
The largest hole that is big enough is allocated to program. i.e. In the worst fit, allocate the largest available hole to process. This method produces the largest leftover hole.

Given memory partitions of **100KB, 500KB, 200KB, 300KB and 600KB (in order)**, how would each of the first-fit, best-fit and worst-fit algorithms place processes of **212KB, 417KB, 112KB and 426KB (in order)**? Which algorithm makes the most efficient use of memory?

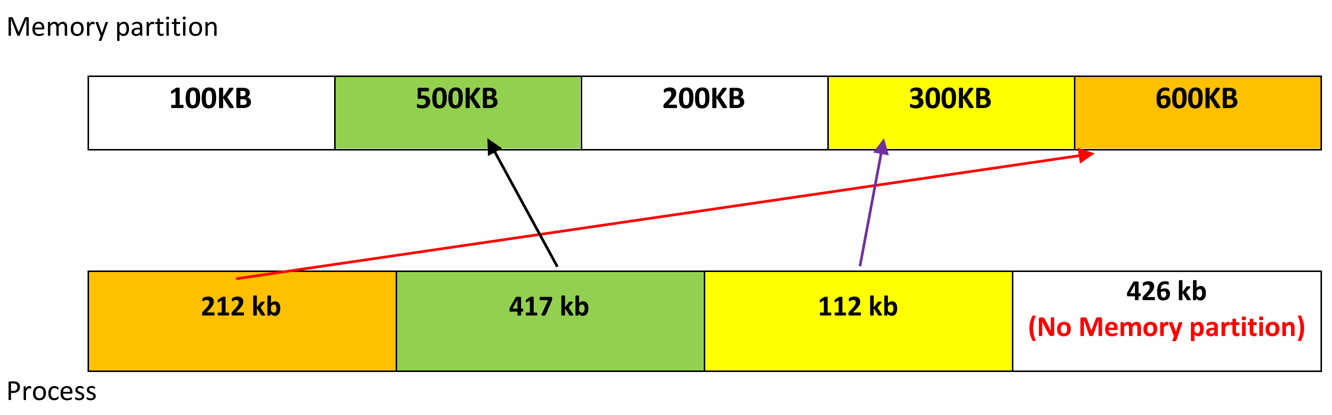
**First fit:**



**Best fit:**



**Worst fit:**



**Best fit** makes the most efficient use of memory.

**Fragmentation:**

Fragmentation is defined as when the process is loaded and removed after execution from memory, it creates a small free hole. These holes can not be assigned to new processes because holes are not combined or do not fulfill the memory requirement of the process. To achieve a degree of multiprogramming, we must reduce the waste of memory or fragmentation problems. In the operating systems two types of fragmentation:

**Internal fragmentation:**

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

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

**External fragmentation:**

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

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

Both the first fit and best-fit systems for memory allocation affected by external fragmentation. To overcome the external fragmentation problem Compaction is used. In the compaction technique, all free memory space combines and makes one large block. So, this space can be used by other processes effectively.

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

**Paging**

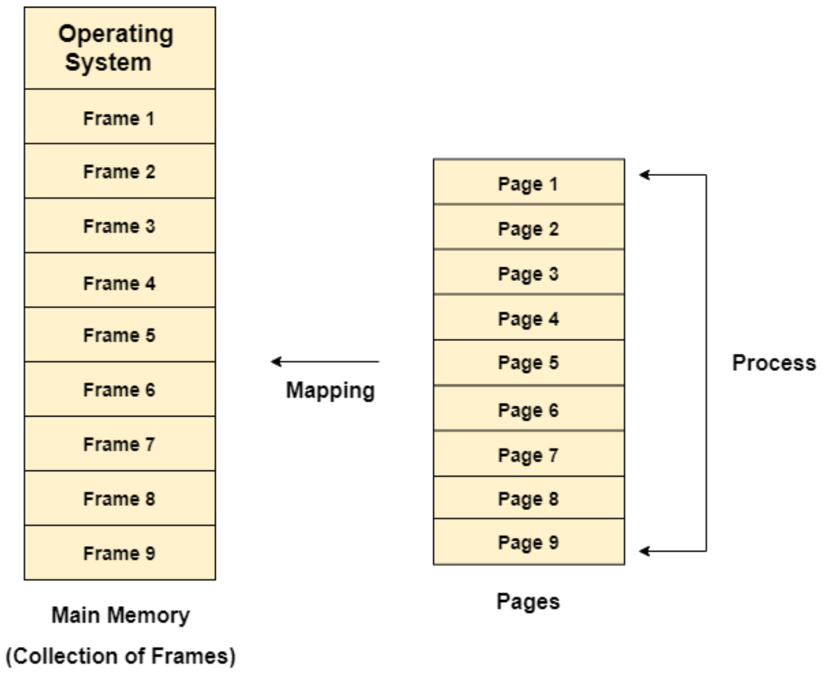
In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.

The main idea behind the paging is to divide each process in the form of pages. The main memory will also be divided in the form of frames.

One page of the process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes.

Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.

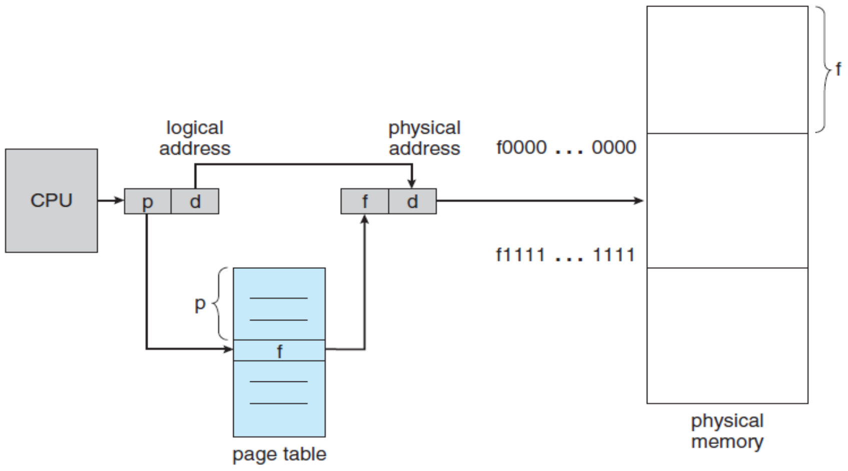
Different operating system defines different frame sizes. The sizes of each frame must be equal. Considering the fact that the pages are mapped to the frames in Paging, page size needs to be as same as frame size.



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

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

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



The address generated by the CPU is divided into

**Page number(p):** Number of bits required to represent the pages in Logical Address Space or Page number

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

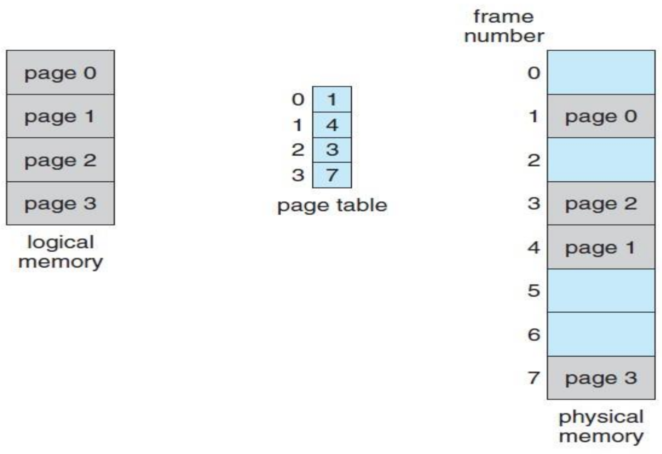
Physical Address is divided into

**Frame number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number frame

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

**Page Table**

Page Table is a data structure used by the virtual memory system to store the mapping between logical addresses and physical addresses.



**Paging model of logical and physical memory**

## Hardware Support

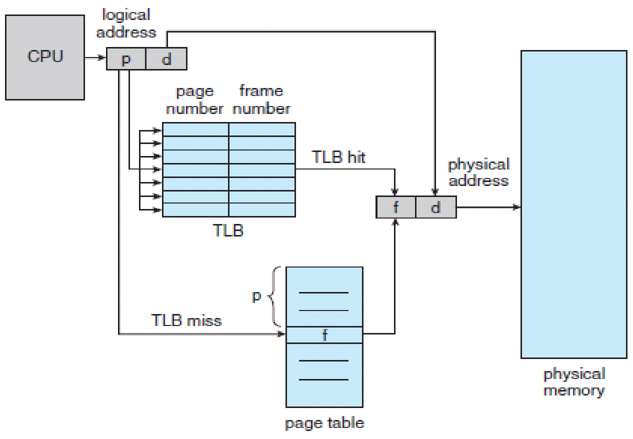
### Implementation of Page Table

- 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)
- Some TLBs store address-space identifiers (ASIDs) in each TLB entry – uniquely identifies each process to provide address-space protection for that process

### TLB

- The TLB is associative, high-speed memory.
- Each entry in the TLB consists of two parts:

  **a key (or tag) and a value.**
- When the associative memory is presented with an item, the item is compared with all keys simultaneously.
- If the item is found, the corresponding value field is returned.
- The TLB contains only a few of the page-table entries.
- When a logical address is generated by the CPU, its page number is presented to the TLB.
- If the page number is not in the TLB (known as a TLB miss), a memory reference to the page table must be made.
- Depending on the CPU, this may be done automatically in hardware or via an interrupt to the operating system.
- If the page number is found, its frame number is immediately available and is used to access memory.



**Paging Hardware with TLB**

**Hit Ratio -** The percentage of times that the page number of interest is found in the TLB is called the hit ratio.

**Effective Memory Access Time**

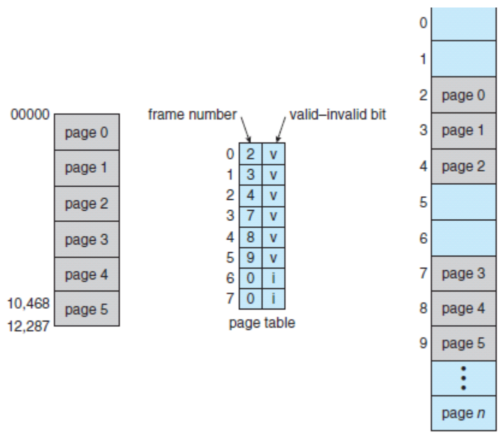
- An 80-percent hit ratio, for example, means that we find the desired page number in the TLB 80 percent of the time. If it takes 100 nanoseconds to access memory, then a mapped-memory access takes 100 nanoseconds when the page number is in the TLB.
- If we fail to find the page number in the TLB then we must first access memory for the page table and frame number (100 nanoseconds) and then access the desired byte in memory (100 nanoseconds), for a total of 200 nanoseconds.

effective access time = 0.80 × 100 + 0.20 × 200
= 120 nanoseconds

For a 99-percent hit ratio, which is much more realistic, we have effective access time = 0.99 × 100 + 0.01 × 200 = 101 nanoseconds

**Protection**

- Memory protection in a paged environment is accomplished by protection bits associated with each frame. Normally, these bits are kept in the page table. One bit can define a page to be read- write or read-only.
- Every reference to memory goes through the page table to find the correct frame number. At the same time that the physical address is being computed, the protection bits can be checked to verify that no writes are being made to a read-only page.
- An attempt to write to a read-only page causes a hardware trap to the operating system (or memory-protection violation).
- One additional bit is generally attached to each entry in the page table: a valid-invalid bit.
  - ✓ When this bit is set to ``valid", the associated page is in the process's logical address space and is thus a legal (or valid) page.
  - ✓ When the bit is set to ``invalid", the page is not in the process's logical address space.
- Illegal addresses are trapped by use of the valid-invalid bit. The OS sets this bit for each page to allow or disallow access to the page.



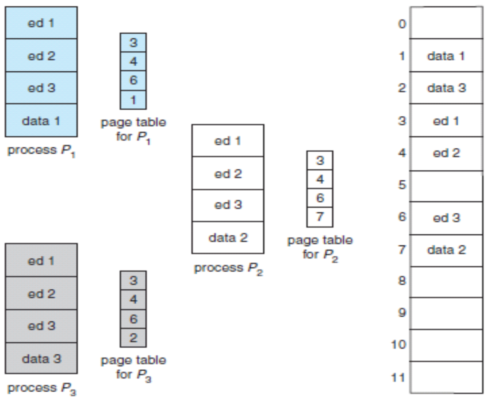
**Valid (v) or invalid (i) bit in a page table.**

- Item Addresses in pages 0, 1, 2, 3, 4, and 5 are mapped normally through the page table.
- Any attempt to generate an address in pages 6 or 7, however, will find that the valid- invalid bit is set to invalid, and the computer will trap to the OS.

**Shared Pages**

An advantage of paging is the possibility of sharing common code. This consideration is particularly important in a time-sharing environment

***Re-entrant code*** is non-self-modifying code; it never changes during execution. Thus, two or more processes can execute the same code at the same time.



**Sharing of code in a paging environment**

- Each process has its own copy of registers and data storage to hold the data for the process's execution. The data for two different processes will, of course, be different.
- Only one copy of the editor need be kept in physical memory. Each user's page table maps onto the same physical copy of the editor, but data pages are mapped onto different frames.

## STRUCTURE OF THE PAGE TABLE

The most common techniques for structuring the page table are

1. Hierarchical paging
2. Hashed page tables
3. Inverted page tables

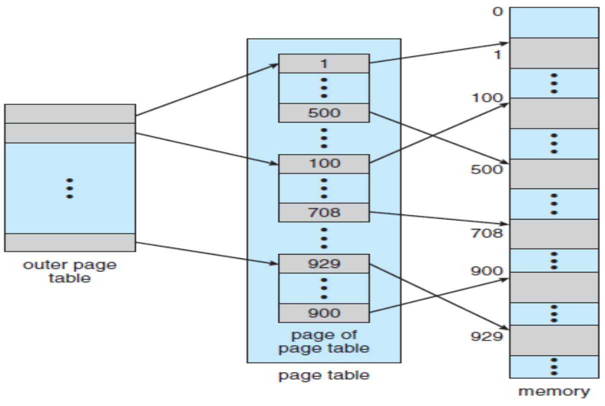
### Hierarchical Paging

The page table itself becomes large for computers with large logical address space (232 to 264).

Example:

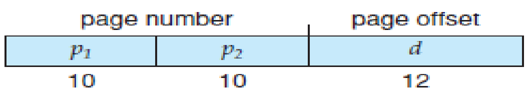
- Consider a system with a 32-bit logical address space. If the page size in such a system is 4 KB (212), then a page table may consist of up to 1 million entries (232/212).
- Assuming that each entry consists of 4 bytes,each process may need up to 4 MB of physical address space for the page table alone.
- The page table should be allocated contiguously in main memory.

- The solution to this problem is to divide the page table into smaller pieces.



One way of dividing the page table is to use a two-level paging algorithm, in which the page table itself is also paged as in the figure:

For example, consider again the system with a 32-bit logical address space and a page size of 4 KB. A logical address is divided into a page number consisting of 20 bits and a page offset consisting of 12 bits. Because we page the page table, the page number is further divided into a 10-bit page number and a 10-bit page offset.

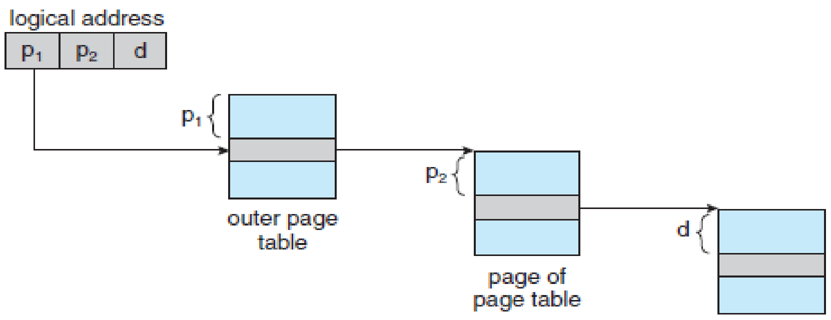


Thus, a logical address is as follows:
where

***p1*** *- an index into the outer page table*

***p2*** *- the displacement within the page of the inner page table.*

The address-translation method for this architecture is shown in the figure. Because address translation works from the outer page table inward, this scheme is also known as a forward-mapped page table.

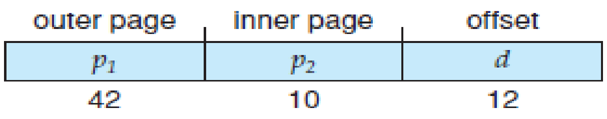


**Address translation for a two- level 32-bit paging architecture**

For a system with a 64-bit logical address space, a two-level paging scheme is no longer appropriate.

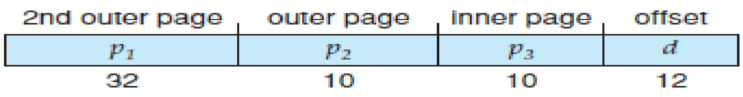
- Suppose that the page size in such a system is 4 KB (212).
- In this case, the page table consists of up to 252 entries.
- If a two-level paging scheme is used, then the inner page tables can conveniently be one page long, or contain 210 4-byte entries.

The addresses look like this:



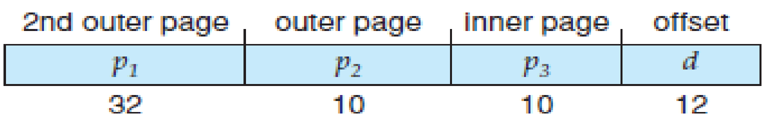
The outer page table consists of 242 entries, or 244 bytes. The obvious way to avoid such a large table is to divide the outer page table into smaller pieces.
The outer page table can be divided in various ways.

We can page the outer page table, giving us a three-level paging scheme. Suppose that the outer page table is made up of standard-size pages (210 entries, or 212 bytes).
In this case, a 64-bit address space is as follows:



The outer page table is still 234 bytes (16 GB) in size.

The next step would be a four-level paging scheme, where the second-level outer page table itself is also paged, and so forth.



The 64-bit UltraSPARC would require seven levels of paging—a prohibitive number of memory accesses - to translate each logical address.

**Hashed Page Tables**

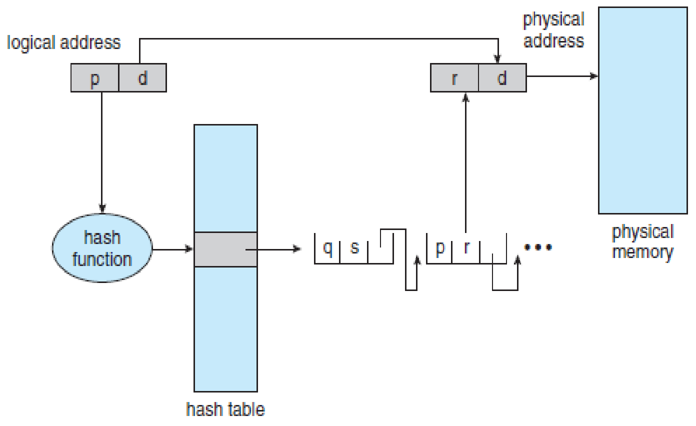
- A common approach for handling address spaces larger than 32 bits is to use a hashed page table, with the hash value being the virtual page number.
- Each entry in the hash table contains a linked list of elements that hash to the same location (to handle collisions).
- Each element consists of three fields:

  **the virtual page number**
  **the value of the mapped page frame**
  **a pointer to the next element in the linked list.**

**Algorithm:**

- The virtual page number in the virtual address is hashed into the hash table.
- The virtual page number is compared with field 1 in the first element in the linked list.
- If there is a match, the corresponding page frame (field 2) is used to form the desired physical address.
- If there is no match, subsequent entries in the linked list are searched for a matching virtual page number.

This scheme is shown below:



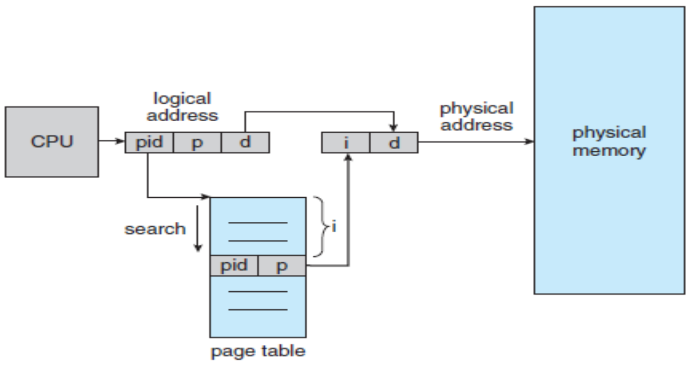
- A variation of this scheme that is useful for 64-bit address spaces has been proposed.
- This variation uses clustered page tables, which are similar to hashed page tables except that each entry in the hash table refers to several pages (such as 16) rather than a single page.Therefore, a single page-table entry can store the mappings for multiple physical-page frames.
- Clustered page tables are particularly useful for sparse address spaces, where memory references are noncontiguous and scattered throughout the address space.

**Inverted Page Tables**

- Each process has an associated page table.
- The page table has one entry for each page that the process is using. This table representation is a natural one, since processes reference pages through the pages' virtual addresses.
- The operating system must then translate this reference into a physical memory address.
- Since the table is sorted by virtual address, the operating system is able to calculate where in the table the associated physical address entry is located and to use that value directly.

**Drawbacks of this method**

- Each page table may consist of millions of entries. These tables may consume large amounts of physical memory just to keep track of how other physical memory is being used. To solve this problem, we can use an inverted page table.
- An inverted page table has one entry for each real page (or frame) of memory.
- Each entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns the page. Thus, only one page table is in the system, and it has only one entry for each page of physical memory.



The operation of an inverted page table is shown above:

- Inverted page tables often require that an address-space identifier be stored in each entry of the page table, since the table usually contains several different address spaces mapping physical memory.
- Storing the address-space identifier ensures that a logical page for a particular process is mapped to the corresponding physical page frame.

**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 a segment which can be allocated to a process.

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

Segment table contains mainly two information about segment:

**Base:** It is the base address of the segment

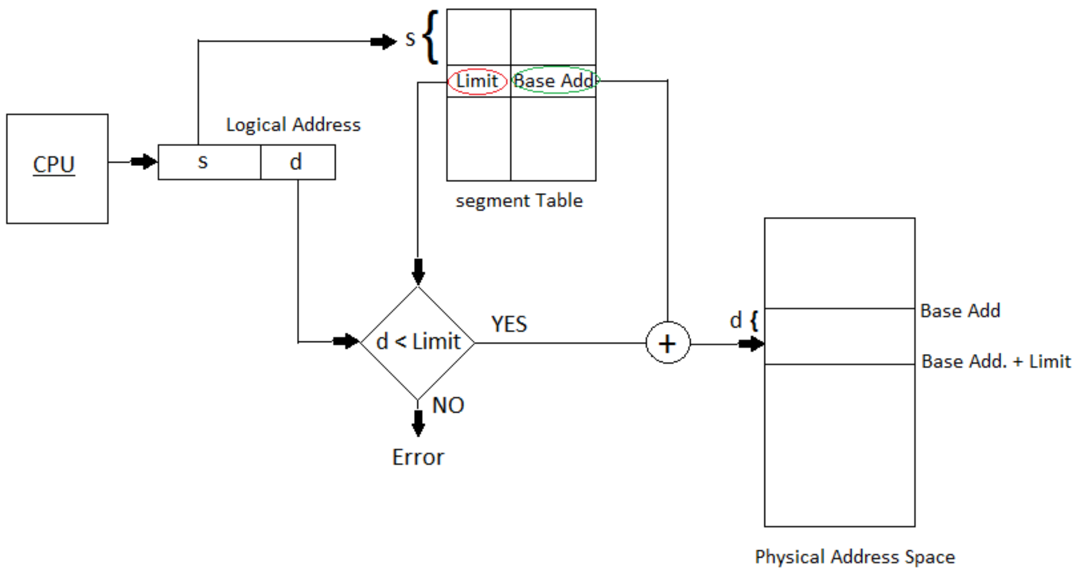
**Limit:** It is the length of the segment.

**Why Segmentation is required?**

Till now, we were using Paging as our main memory management technique. Paging is more close to the Operating system rather than the User. It divides all the processes into the form of pages regardless of the fact that a process can have some relative parts of functions which need 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 contains the same type of functions such as the main function can be included in one segment and the library functions can be included in the other segment.



**Segmentation Hardware**

A logical address consists of two parts: a segment number, s and an offset into that segment, d
The segment number is used as an index to the segment table. The offset d of the logical address must be between 0 and the segment limit. If it is not, we trap to the OS (logical addressing attempt beyond end of segment).
When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte. The segment table is thus essentially an array of base-limit register pairs.

**Advantages of Segmentation**

- No internal fragmentation
- Average Segment Size is larger than the actual page size.
- Less overhead
- It is easier to relocate segments than entire address space.
- The segment table is of lesser size as compared to the page table in paging.

**Disadvantages**

- It can have external fragmentation.
- It is difficult to allocate contiguous memory to variable sized partition.
- Costly memory management algorithms.

**Difference between Paging and Segmentation**

| S. No. | Paging | Segmentation |
|---|---|---|
| 1 | Non-Contiguous memory allocation | Non-Contiguous memory allocation |
| 2 | Paging divides program into fixed size pages. | Segmentation divides program into variable size segments. |
| 3 | OS is responsible | Compiler is responsible. |
| 4 | Paging is faster than segmentation | Segmentation is slower than paging |
| 5 | Paging is closer to Operating System | Segmentation is closer to User |
| 6 | It suffers from internal fragmentation | It suffers from external fragmentation |
| 7 | There is no external fragmentation | There is no external fragmentation |
| 8 | Logical address is divided into page number and page offset | Logical address is divided into segment number and segment offset |
| 9 | Page table is used to maintain the page information. | Segment Table maintains the segment information |
| 10 | Page table entry has the frame number and some flag bits to represent details about pages. | Segment table entry has the base address of the segment and some protection bits for the segments. |

**Segmented Paging**

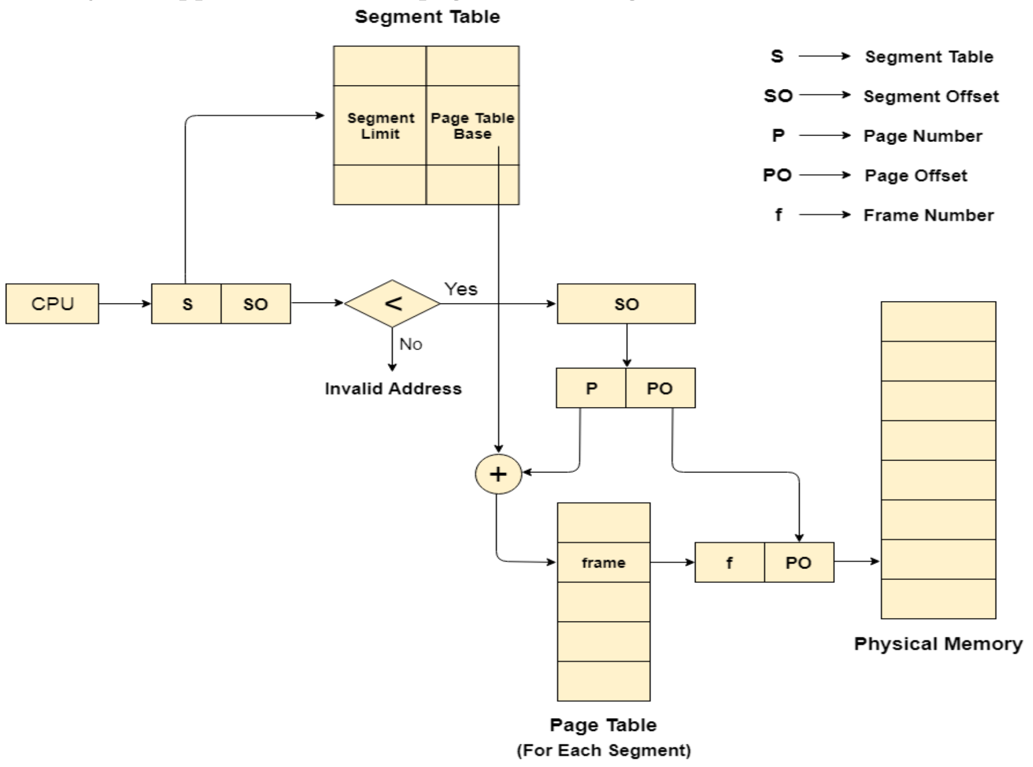
Pure segmentation is not very popular and not being used in many of the operating systems. However, Segmentation can be combined with Paging to get the best features out of both the techniques.

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

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

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

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



**Translation of logical address to physical address**

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

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

**Advantages of Segmented Paging**

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

**Disadvantages of Segmented Paging**

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

**Virtual Memory**

A storage method known as virtual memory gives the user the impression that their main memory is quite large.
By considering a portion of secondary memory as the main memory, this is accomplished.
By giving the user the impression that there is memory available to load the process, this approach allows them to load larger size programs than the primary memory that is accessible.
The Operating System loads the many components of several processes in the main memory as opposed to loading a single large process there.
By doing this, the level of multiprogramming will be enhanced, which will increase CPU consumption.

**Demand Paging**

The Demand Paging is a condition which is occurred in the Virtual Memory.
We know that the pages of the process are stored in secondary memory.
The page is brought to the main memory when required.
We do not know when this requirement is going to occur.
So, the pages are brought to the main memory when required by the Page Replacement Algorithms.
So, the process of calling the pages to main memory to secondary memory upon demand is known as Demand Paging.

The important jobs of virtual memory in Operating Systems are two. They are:

- Frame Allocation
- Page Replacement.

**Page Replacement**

A page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.
There are three types of Page Replacement Algorithms. They are:

- First In First Out Page Replacement Algorithm
- Optimal Page Replacement Algorithm
- Least Recently Used (LRU) Page Replacement Algorithm

If the page to be searched is found among the frames then, this process is known as **Page Hit**.
If the page to be searched is not found among the frames then, this process is known as **Page Fault.**

**1. First In First Out (FIFO):**
The First In First Out (FIFO) Page Replacement Algorithm removes the Page in the frame which is allotted long back.
This means the useless page which is in the frame for a longer time is removed and the new page which is in the ready queue and is ready to occupy the frame is allowed by the First in First out Page Replacement.

**2. OPTIMAL Page Replacement Algorithm**
The OPTIMAL Page Replacement Algorithms works on a certain principle. The principle is:
Replace the Page which is not used in the Longest Dimension of time in future
This principle means that after all the frames are filled then, see the future pages which are to occupy the frames. Go on checking for the pages which are already available in the frames. Choose the page which is at last.

**3. Least Recently Used (LRU) Replacement Algorithm**
The Least Recently Used (LRU) Page Replacement Algorithms works on a certain principle. The principle is:
Replace the page with the page which is less dimension of time recently used page in the past.

**Consider the following page preferences string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1. How many page faults would occur for the following replacement algorithm? Assume three frames and all frames are initially empty.**
a. FIFO replacement
b. Optimal replacement
c. LRU replacement

**a. FIFO replacement**
**Reference string**

| | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Frame0** | 7 | 7 | 7 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 7 | 7 |
| **Frame1** | | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| **Frame2** | | | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 1 |
| **Page Fault :** | * | * | * | * | | * | * | * | * | * | * | | | * | * | | | * | * | * |

**Page Fault: 15**

**b. Optimal replacement (check future) --- >**

| | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Frame0** | 7 | 7 | 7 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 7 | 7 | 7 |
| **Frame1** | | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Frame2** | | | 1 | 1 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **Page Fault :** | * | * | * | * | | * | | * | | | * | | | * | | | | * | | |

**Page Fault: 09**

**c. LRU replacement (< ---)**

| | 7 | 0 | 1 | 2 | 0 | 3 | 0 | 4 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 0 | 1 | 7 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Frame0** | 7 | 7 | 7 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **Frame1** | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 0 |
| **Frame2** | | | 1 | 1 | 1 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 7 | 7 | 7 |
| **Page Fault :** | * | * | * | * | | * | | * | * | * | * | | | * | | * | | * | | |

**Page Fault: 12**

**Allocation of frames in OS**

The main memory of the system is divided into frames.

The OS has to allocate a sufficient number of frames for each process and to do so, the OS uses various algorithms.

The five major ways to allocate frames are as follows:

**Proportional frame allocation**

The proportional frame allocation algorithm allocates frames based on the size that is necessary for the execution and the number of total frames the memory has.

The only disadvantage of this algorithm is it does not allocate frames based on priority. This situation is solved by Priority frame allocation.

**Priority frame allocation**

Priority frame allocation allocates frames based on the priority of the processes and the number of frame allocations.

If a process is of high priority and needs more frames then the process will be allocated that many frames. The allocation of lower priority processes occurs after it.

**Global replacement allocation**

When there is a page fault in the operating system, then the global replacement allocation takes care of it.

The process with lower priority can give frames to the process with higher priority to avoid page faults.

**Local replacement allocation**

In local replacement allocation, the frames of pages can be stored on the same page.

It doesn’t influence the behavior of the process as it did in global replacement allocation.

**Equal frame allocation**

In equal frame allocation, the processes are allocated equally among the processes in the operating system.

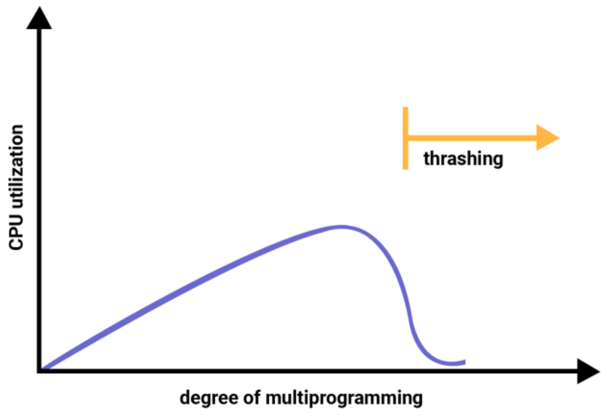
The only disadvantage in equal frame allocation is that a process requires more frames for allocation for execution and there are only a set number of frames.

**Thrashing**

In case, if the page fault and swapping happens very frequently at a higher rate, then the operating system has to spend more time swapping these pages. This state in the operating system is termed thrashing. Because of thrashing the CPU utilization is going to be reduced.

Let's understand by an example, if any process does not have the number of frames that it needs to support pages in active use then it will quickly page fault. And at this point, the process must replace some pages. As all the pages of the process are actively in use, it must replace a page that will be needed again right away. Consequently, the process will quickly fault again, and again, and again, replacing pages that it must bring back in immediately. This high paging activity by a process is called thrashing.

During thrashing, the CPU spends less time on some actual productive work spend more time swapping.



**Causes of Thrashing**

Thrashing affects the performance of execution in the Operating system. Also, thrashing results in severe performance problems in the Operating system.

When the utilization of CPU is low, then the process scheduling mechanism tries to load many processes into the memory at the same time due to which degree of Multiprogramming can be increased. Now in this situation, there are more processes in the memory as compared to the available number of frames in the memory. Allocation of the limited amount of frames to each process.

Whenever any process with high priority arrives in the memory and if the frame is not freely available at that time then the other process that has occupied the frame is residing in the frame will move to secondary storage and after that this free frame will be allocated to higher priority process.

We can also say that as soon as the memory fills up, the process starts spending a lot of time for the required pages to be swapped in. Again the utilization of the CPU becomes low because most of the processes are waiting for pages.

Thus a high degree of multiprogramming and lack of frames are two main causes of thrashing in the Operating system.

**Effect of Thrashing**

At the time, when thrashing starts then the operating system tries to apply either the Global page replacement Algorithm or the Local page replacement algorithm.

**Global Page Replacement**

The Global Page replacement has access to bring any page, whenever thrashing found it tries to bring more pages. Actually, due to this, no process can get enough frames and as a result, the thrashing will increase more and more. Thus the global page replacement algorithm is not suitable whenever thrashing happens.

**Local Page Replacement**

Unlike the Global Page replacement, the local page replacement will select pages which only belongs to that process. Due to this, there is a chance of a reduction in the thrashing. As it is also proved that there are many disadvantages of Local Page replacement. Thus local page replacement is simply an alternative to Global Page replacement.

**Techniques used to handle the thrashing**

As we have already told you the Local Page replacement is better than the Global Page replacement but local page replacement has many disadvantages too, so it is not suggestible. Thus given below are some other techniques that are used:

**Working-Set Model**

This model is based on the assumption of the locality. It makes the use of the parameter ? in order to define the working-set window. The main idea is to examine the most recent? page reference. What locality is saying, the recently used page can be used again, and also the pages that are nearby this page will also be used?

**1. Working Set**

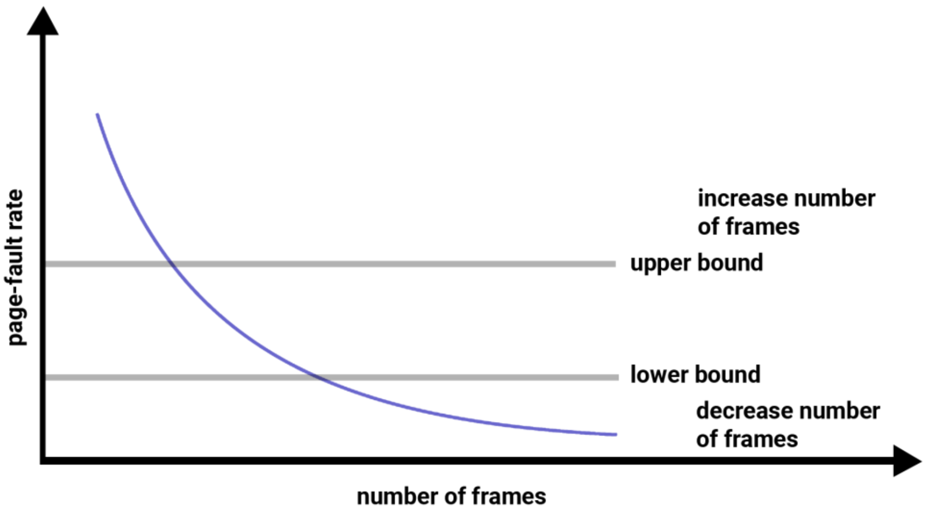
The set of the pages in the most recent? page reference is known as the working set. If a page is in active use, then it will be in the working set. In case if the page is no longer being used then it will drop from the working set ? times after its last reference.

The working set mainly gives the approximation of the locality of the program.

The accuracy of the working set mainly depends on? what is chosen?

This working set model avoids thrashing while keeping the degree of multiprogramming as high as possible.

**2. Page Fault Frequency**



The working-set model is successful and its knowledge can be useful in preparing but it is a very clumpy approach in order to avoid thrashing. There is another technique that is used to avoid thrashing and it is Page Fault Frequency(PFF) and it is a more direct approach.

The main problem is how to prevent thrashing. As thrashing has a high page fault rate and also we want to control the page fault rate.

When the Page fault is too high, then we know that the process needs more frames. Conversely, if the page fault-rate is too low then the process may have too many frames.

We can establish upper and lower bounds on the desired page faults. If the actual page-fault rate exceeds the upper limit then we will allocate the process to another frame. And if the page fault rate falls below the lower limit then we can remove the frame from the process.

Thus with this, we can directly measure and control the page fault rate in order to prevent thrashing.

**Copy-on-Write**

Copy-on-Write(CoW) is mainly a resource management technique that allows the parent and child process to share the same pages of the memory initially. If any process either parent or child modifies the shared page, only then the page is copied.

The CoW is basically a technique of efficiently copying the data resources in the computer system. In this case, if a unit of data is copied but is not modified then "copy" can mainly exist as a reference to the original data.

But when the copied data is modified, then at that time its copy is created(where new bytes are actually written )as suggested from the name of the technique.

The main use of this technique is in the implementation of the fork system call in which it shares the virtual memory/pages of the Operating system.

Recall in the UNIX(OS), the fork() system call is used to create a duplicate process of the parent process which is known as the child process.

The CoW technique is used by several Operating systems like Linux, Solaris, and Windows XP.

The CoW technique is an efficient process creation technique as only the pages that are modified are copied.

Free pages in this technique are allocated from a pool of zeroed-out pages.

**The Copy on Write(CoW) Technique**

The main intention behind the CoW technique is that whenever a parent process creates a child process both parent and child process initially will share the same pages in the memory.

These shared pages between parent and child process will be marked as copy-on-write which means that if the parent or child process will attempt to modify the shared pages then a copy of these pages will be created and the modifications will be done only on the copy of pages by that process and it will not affect other processes.

Now its time to take a look at the basic example of this technique:

Let us take an example where Process A creates a new process that is Process B, initially both these processes will share the same pages of the memory.

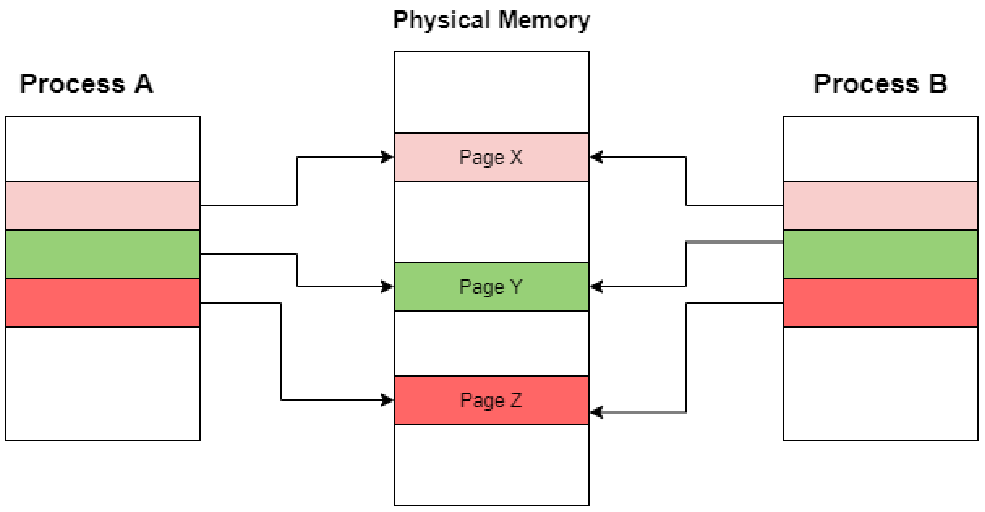


**Figure: Above figure indicates parent and child process sharing the same pages**

Now, let us assume that process A wants to modify a page in the memory. When the Copy-on-write(CoW) technique is used, only those pages that are modified by either process are copied; all the unmodified pages can be easily shared by the parent and child process.

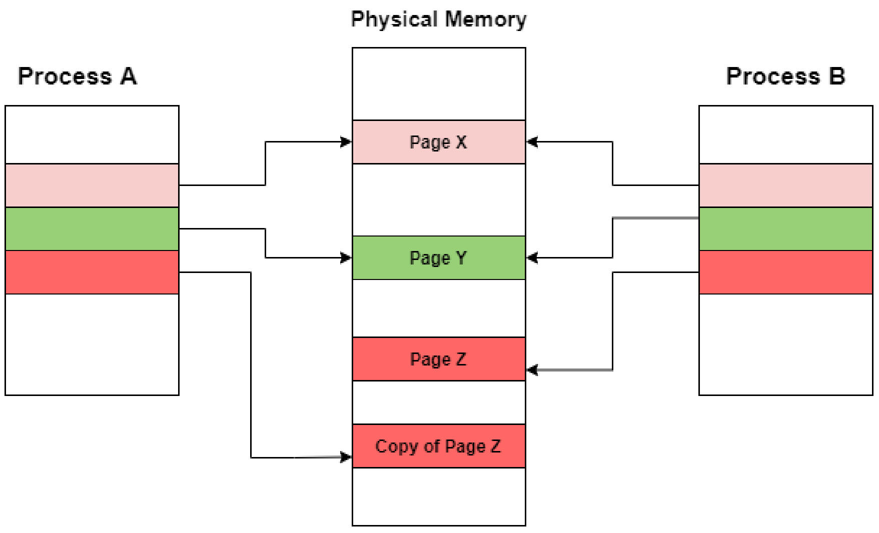


**Figure: After Page Z is modified by Process A**

Whenever it is determined that a page is going to be duplicated using the copy-on-write technique, then it is important to note the location from where the free pages will be allocated. There is a pool of free pages for such requests; provided by many operating systems.

And these free pages are allocated typically when the stack/heap for a process must expand or when there are copy-on-write pages to manage.

These pages are typically allocated using the technique that is known as Zero-fill-on-demand. And the Zero-fill-on-demand pages are zeroed-out before being allocated and thus erasing the previous content.

## TWO MARKS QUESTIONS WITH ANSWERS

**1.What is known as Dynamic loading?**

With Dynamic loading, a routine is not loaded until it is called. All routines are kept on disk in a re-locatable load format. The main program is loaded into memory and is executed. When a routine needs to call another routine, the calling routine first checks to see whether the another routine has been loaded. If not, the re-locatable linking loader is called to load the desired routine into memory and to update the program's address tables to reflect this change. Then, Control is passed to the newly loaded routine.

**2.What is meant by Swapping?**

It is a process of bringing in each process in its entirety, running it for a while and then putting it back on the disk.

**3.What is the advantage of Dynamic Loading?**

The advantage of Dynamic Loading is that an unused routine is never loaded.(i.e) when large amounts of code are needed to handle infrequently occurring cases, such as error routines. Here although program size may be large, the portion that is used may be much smaller and better memory space utilization.

**4.What is known as Dynamic Linking?**

In this Dynamic Linking, a stub is included in the image for each library-routine reference. This Stub is a small piece of code that indicates how to locate the appropriate memory-resident library routine or how to load the library if the routine is not already present.

**5.What is meant by External Fragmentation and Internal Fragmentation?**

External Fragmentation exists when enough total memory space exists to satisfy a request, but it is not contiguous and storage is fragmented into a large number of small holes.

- The memory allocated to a process may be slightly larger than the requested memory.
- The difference between these two numbers is called as Internal Fragmentation.

**6.What is meant by Paging? Give its advantages.**

Paging is a Memory-management scheme that permits the physical -address space of a process to be Non-contiguous.

**Advantages:**

- Avoids the considerable problem of fitting the varying -sized memory chunks onto the baking store
- Fragmentation problems are also prevalent baking store, except that access is much slower, so compaction is impossible.

**7.What is TLB and Hit-Ratio?**

Translation Lookaside Buffer (TLB) is a small, special and fast cache which is associated with high speed memory.

The Percentage of times that a particular page number is found in the Translation Lookaside Buffer (TLB) is called as Hit- Ratio.

**8.What is meant by Segmentation?**

Segmentation is a memory-management scheme that supports the user-view memory. Blocks of different size is called as Segments and its associative virtual storage Organization is called as Segmentation.

**9.What is meant by Memory Compaction?**

When swapping creates multiple holes in memory, it is possible to combine them all into one big one by moving all the processes downward as far as possible.

**10.What is meant by overlay?**

The idea of overlays is to keep in memory only those instructions and data that are needed at any given time. So, to enable a process to be larger than the amount of memory allocated to it.

**11.What is meant by Demand Paging?**

Whenever the CPU tries to fetch the first instruction, it gets a page fault causing the 0S to bring in the page containing that instruction. Thus the pages are loaded only on demand is called as Demand Paging.

**12.What is meant by Locality of reference?**

During any phase of execution, the page references only a relative small fraction of its pages. This reference of fraction of all pages is called as Locality of Reference.

**13.What are the principal events of Process Creation?**

- System Initialization.
- Execution of a System call by a running process. A user request to create a new process.
- Initiation of a batch job.

**14.What is meant by Page Fault?**

Whenever memory management unit notices that the page is unmapped and causes the CPU to trap to the Operating System. This trap is called as Page Fault.

**15.What is meant by Thrashing?**

A Program which is causing page faults every few instructions to occur is called as Thrashing.

**16.What is meant by Page Table?**

Page Table is a table which has the ability to mark an entry invalid through a Valid-Invalid bit or special value of protection bits.

## SIXTEEN MARK QUESTIONS WITH ANSWERS

**1.Explain in detail about Contiguous Memory Allocation.**

- Memory Protection
- Memory Allocation
- Fragmentation

**2.Explain about Segmentation.**

- Basic Methods
- Hardware

**3.Explain the concept of Paging**

- Basic Method
- Hardware
- Protection
- Shared pages

**4.Describe the following allocation algorithms: a.First fit b.Best fit c.Worst fit**

- First fit - example
- Best fit - example
- Worst fit - example

**5.Explain Demand Paging.**

- Basic concepts
- Hardware
- Example
- Performance

**6.Consider the following page preferences string 1,2,3,4,5,3,4,5,3,4,1,6,7,8,7,8,9,7,8,9,5,4,4,5,3. How many page faults would occur for the following replacement algorithm? Assume four frames and all frames are initially empty.**

a. LRU replacement
b. FIFO replacement
c. Optimal replacement

**7. Consider the reference string 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 4, 0 for a memory with three frames and calculate number of page faults for the following replacement algorithm.**

a. LRU replacement
b. FIFO replacement
c. Optimal replacement

**8.Explain in detail about Thrashing.**

- Thrashing
- Cause of Thrashing
- Working set model
- Page-fault frequency