STRUCTURE OF PAGE TABLE

The data structure that is used by the virtual memory system in the operating system of a computer in order to store the mapping between physical and logical addresses is commonly known as **Page Table**.

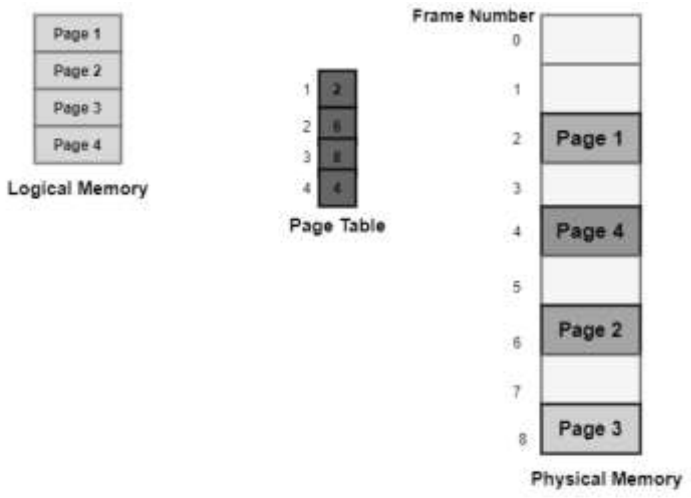
As we had already told you that the logical address that is generated by the CPU is translated into the physical address with the help of the page table.

- Thus page table mainly provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.

Characteristics of the Page Table

Some of the characteristics of the Page Table are as follows:

- It is stored in the main memory.
- Generally; the Number of entries in the page table = the Number of Pages in which the process is divided.
- **PTBR** means page table base register and it is basically used to hold the base address for the page table of the current process.
- Each process has its own independent page table.



The above diagram shows the paging model of Physical and logical memory.

Techniques used for Structuring the Page Table

Some of the common techniques that are used for structuring the Page table are as follows:

1. Hierarchical Paging
2. Hashed Page Tables
3. Inverted Page Tables

Let us cover these techniques one by one;

Hierarchical Paging

Another name for Hierarchical Paging is multilevel paging.

- There might be a case where the page table is too big to fit in a contiguous space, so we may have a hierarchy with several levels.
- In this type of Paging the logical address space is broke up into Multiple page tables.
- Hierarchical Paging is one of the simplest techniques and for this purpose, a two-level page table and three-level page table can be used.

Two Level Page Table

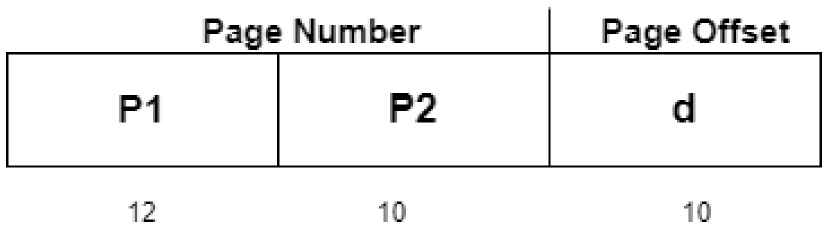
Consider a system having 32-bit logical address space and a page size of 1 KB and it is further divided into:

- Page Number consisting of 22 bits.
- Page Offset consisting of 10 bits.

As we page the Page table, the page number is further divided into :

- Page Number consisting of 12 bits.
- Page Offset consisting of 10 bits.

Thus the Logical address is as follows:



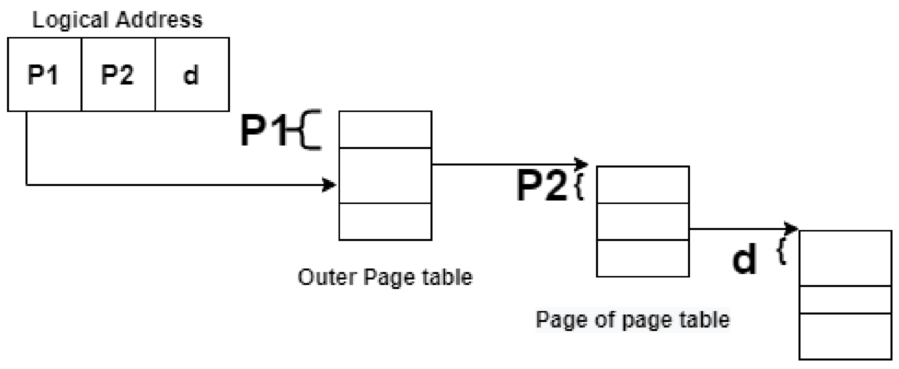
In the above diagram,

P1 is an index into the **Outer Page** table.

P2 indicates the displacement within the page of the **Inner page** Table.

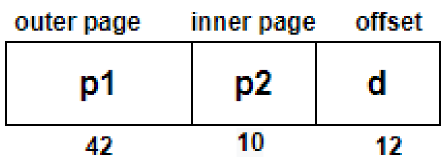
As address translation works from outer page table inward so is known as **forward-mapped Page Table**.

Below given figure below shows the Address Translation scheme for a two-level page table

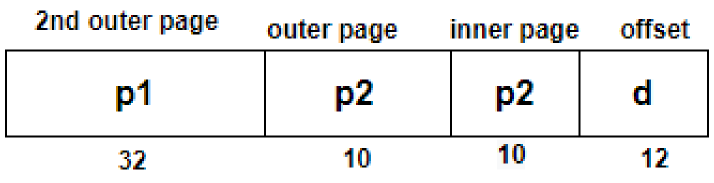


Three Level Page Table

For a system with 64-bit logical address space, a two-level paging scheme is not appropriate. Let us suppose that the page size, in this case, is 4KB.If in this case, we will use the two-page level scheme then the addresses will look like this:



Thus in order to avoid such a large table, there is a solution and that is to divide the outer page table, and then it will result in a **Three-level page table:**



Hashed Page Tables

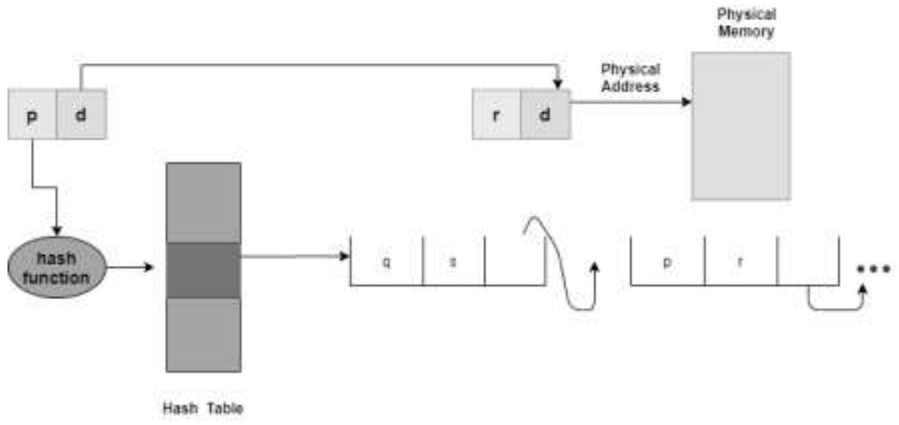
This approach is used to handle address spaces that are larger than 32 bits.

- In this virtual page, the number is hashed into a page table.
- This Page table mainly contains a chain of elements hashing to the same elements.

Each element mainly consists of :

1. The virtual page number
2. The value of the mapped page frame.
3. A pointer to the next element in the linked list.

Given below figure shows the address translation scheme of the Hashed Page Table:



The above Figure shows Hashed Page Table

The Virtual Page numbers are compared in this chain searching for a match; if the match is found then the corresponding physical frame is extracted.

In this scheme, a variation for 64-bit address space commonly uses **clustered page tables**.

Clustered Page Tables

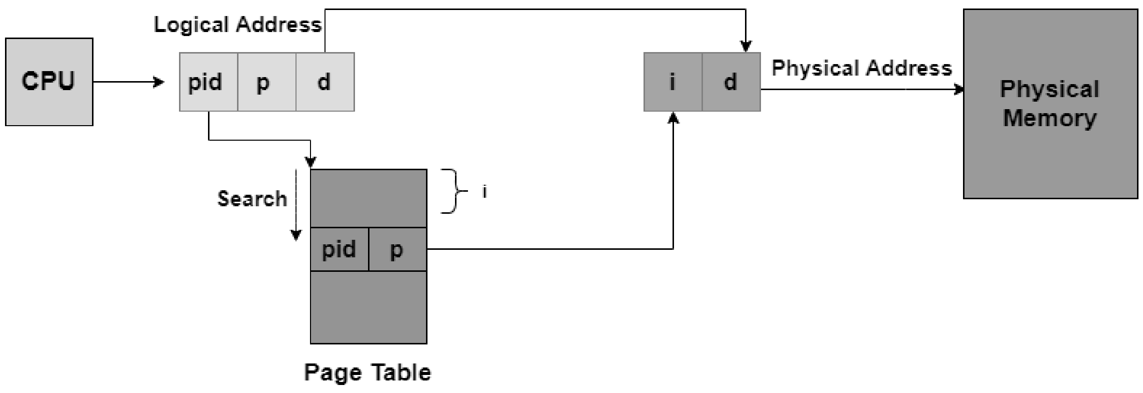
- These are similar to hashed tables but here each entry refers to several pages (that is 16) rather than 1.
- Mainly used for sparse address spaces where memory references are non-contiguous and scattered

Inverted Page Tables

The Inverted Page table basically combines A page table and A frame table into a single data structure.

- There is one entry for each virtual page number and a real page of memory
- And the entry mainly consists of the virtual address of the page stored in that real memory location along with the information about the process that owns the page.
- Though this technique decreases the memory that is needed to store each page table; but it also increases the time that is needed to search the table whenever a page reference occurs.

Given below figure shows the address translation scheme of the Inverted Page Table:



In this, we need to keep the track of process id of each entry, because many processes may have the same logical addresses.

Also, many entries can map into the same index in the page table after going through the hash function. Thus chaining is used in order to handle this.

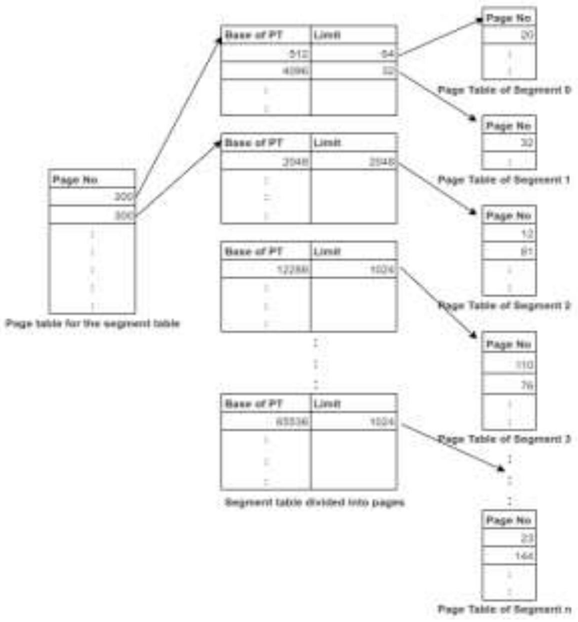
PAGED SEGMENTATION

In segmented paging, not every process has the same number of segments and the segment tables can be large in size which will cause external fragmentation due to the varying segment table sizes. To solve this problem, we use **paged segmentation** which requires the segment table to be paged. The logical address generated by the CPU will now consist of page no #1, segment no, page no #2 and offset.

The page table even with segmented paging can have a lot of invalid pages. Instead of using multi level paging along with segmented paging, the problem of larger page table can be solved by directly applying multi-level paging instead of segmented paging.

Advantages of Paged Segmentation

1. No external fragmentation
2. Reduced memory requirements as no. of pages limited to segment size.
3. Page table size is smaller just like segmented paging,
4. Similar to segmented paging, the entire segment need not be swapped out.
5. Increased flexibility in memory allocation: Paged Segmentation allows for a flexible allocation of memory, where each segment can have a different size, and each page can have a different size within a segment.
6. Improved protection and security: Paged Segmentation provides better protection and security by isolating each segment and its pages, preventing a single segment from affecting the entire process’s memory.
7. Increased program structure: Paged Segmentation provides a natural program structure, with each segment representing a different logical part of a program.
8. Improved error detection and recovery: Paged Segmentation enables the detection of memory errors and the recovery of individual segments, rather than the entire process’s memory.
9. Reduced overhead in memory management: Paged Segmentation reduces the overhead in memory management by eliminating the need to maintain a single, large page table for the entire process’s memory.
10. Improved memory utilization: Paged Segmentation can improve memory utilization by reducing fragmentation and allowing for the allocation of larger blocks of contiguous memory to each segment.



Paged Segmentation

Disadvantages of Paged Segmentation

1. Internal fragmentation remains a problem.
2. Hardware is complexer than segmented paging.
3. Extra level of paging at first stage adds to the delay in memory access.
4. Increased complexity in memory management: Paged Segmentation introduces additional complexity in the memory management process, as it requires the maintenance of multiple page tables for each segment, rather than a single page table for the entire process's memory.
5. Increased overhead in memory access: Paged Segmentation introduces additional overhead in memory access, as it requires multiple lookups in multiple page tables to access a single memory location.
6. Reduced performance: Paged Segmentation can result in reduced performance, as the additional overhead in memory management and access can slow down the overall process.
7. Increased storage overhead: Paged Segmentation requires additional storage overhead, as it requires additional data structures to store the multiple page tables for each segment.

8. Increased code size: Paged Segmentation can result in increased code size, as the additional code required to manage the multiple page tables can take up valuable memory space.
9. Reduced address space: Paged Segmentation can result in a reduced address space, as some of the available memory must be reserved for the storage of the multiple page tables.

VIRTUAL MEMORY

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

What is Virtual Memory?

Virtual memory is a memory management technique used by operating systems to give the appearance of a large, continuous block of memory to applications, even if the physical memory (RAM) is limited. It allows the system to compensate for physical memory shortages, enabling larger applications to run on systems with less RAM.

A memory hierarchy, consisting of a computer system's memory and a disk, enables a process to operate with only some portions of its address space in memory. A virtual memory is what its name indicates- it is an illusion of a memory that is larger than the real memory. We refer to the software component of virtual memory as a virtual memory manager. The basis of virtual memory is the non-contiguous memory allocation model. The virtual memory manager removes some components from memory to make room for other components.
The size of virtual storage is limited by the addressing scheme of the computer system and the amount of secondary memory available not by the actual number of main storage locations.

Working of Virtual Memory

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

- All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of the main memory such that it occupies different places in the main memory at different times during the course of execution.
- A process may be broken into a number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time address translation and the use of a page or segment table permits this.

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

Types of Virtual Memory

In a computer, virtual memory is managed by the Memory Management Unit (MMU), which is often built into the CPU. The CPU generates virtual addresses that the MMU translates into physical addresses.

There are two main types of virtual memory:

- Paging
- Segmentation

Virtual Memory vs Physical Memory

When talking about the differences between virtual memory and physical memory, the biggest distinction is speed. RAM is much faster than virtual memory, but it is also more expensive.

When a computer needs storage for running programs, it uses RAM first. Virtual memory, which is slower, is used only when the RAM is full.

Feature	**Virtual Memory**	**Physical Memory (RAM)**
Definition	An abstraction that extends the available memory by using disk storage	The actual hardware (RAM) that stores data and instructions currently being used by the CPU
Location	On the hard drive or SSD	On the computer's motherboard
Speed	Slower (due to disk I/O operations)	Faster (accessed directly by the CPU)
Capacity	Larger, limited by disk space	Smaller, limited by the amount of RAM installed
Cost	Lower (cost of additional disk storage)	Higher (cost of RAM modules)
Data Access	Indirect (via paging and swapping)	Direct (CPU can access data directly)
Volatility	Non-volatile (data persists on disk)	Volatile (data is lost when power is off)

Performance in Virtual Memory

- Let p be the page fault rate($0 <= p <= 1$).
- if $p = 0$ no page faults
- if $p =1$, every reference is a fault.

Effective access time (EAT) = (1-p)* Memory Access Time + p * Page fault time.

Page fault time = page fault overhead + swap out + swap in +restart overhead

The performance of a virtual memory management system depends on the total number of page faults, which depend on paging policies and frame allocation.

Applications of Virtual memory

Virtual memory has the following important characteristics that increase the capabilities of the computer system. The following are five significant characteristics of Lean.

- **Increased Effective Memory:** One major practical application of virtual memory is, virtual memory enables a computer to have more memory than the physical memory using the disk space. This allows for the running of larger applications and numerous programs at one time while not necessarily needing an equivalent amount of DRAM.

- **Memory Isolation:** Virtual memory allocates a unique address space to each process and that also plays a role in process segmentation. Such separation increases safety and reliability based on the fact that one process cannot interact with and or modify another's memory space through a mistake, or even a deliberate act of vandalism.

- **Efficient Memory Management:** Virtual memory also helps in better utilization of the physical memories through methods that include paging and segmentation. It can transfer some of the memory pages that are not frequently used to disk allowing RAM to be used by active processes when required in a way that assists in efficient use of memory as well as system performance.

- **Simplified Program Development:** For case of programmers, they don't have to consider physical memory available in a system in case of having virtual memory.

They can program ‘as if’ there is one big block of memory and this makes the programming easier and more efficient in delivering more complex applications.

How to Manage Virtual Memory?

Here are 5 key points on how to manage virtual memory:

1. Adjust the Page File Size

- **Automatic Management**: All contemporary operating systems including Windows contain the auto-configuration option for the size of the empirical page file. But depending on the size of the RAM, they are set automatically, although the user can manually adjust the page file size if required.
- **Manual Configuration:** For tuned up users, the setting of the custom size can sometimes boost up the performance of the system. The initial size is usually advised to be set to the minimum value of 1. To set the size of the swap space equal to 5 times the amount of physical RAM and the maximum size 3 times the physical RAM.

2. Place the Page File on a Fast Drive

- **SSD Placement:** If this is feasible, the page file should be stored in the SSD instead of the HDD as a storage device. It has better read and write times, and the virtual memory may prove beneficial in an SSD.
- **Separate Drive:** Regarding systems having multiple drives involved, the page file needs to be placed on a different drive than the os and that shall in turn improve its performance.

3. Monitor and Optimize Usage

- **Performance Monitoring:** Employ the software tools used in monitoring the performance of the system in tracking the amounts of virtual memory. High page file usage may signify that there is a lack of physical RAM or that virtual memory needs a change of settings or addition in physical RAM.

- **Regular Maintenance:** Make sure there is no toolbar or other application running in the background, take time and uninstall all the tool bars to free virtual memory.

4. Disable Virtual Memory for SSDs (with Sufficient RAM)
   - **Sufficient RAM:** If for instance your system has a big physical memory, for example 16GB and above then it would be advised to freeze the page file in order to minimize SSD usage. But it should be done, in my opinion, carefully and only if the additional signals that one decides to feed into his applications should not likely use all the available RAM.

5. Optimize System Settings
   - **System Configuration:** Change some general properties of the system concerning virtual memory efficiency. This also involves enabling additional control options in Windows such as adjusting additional system setting option on the operating system, or using other options in different operating systems such as Linux that provides different tools and commands to help in adjusting how virtual memory is utilized.
   - **Regular Updates:** Ensure that your drivers are run in their newest version because new releases contain some enhancements and issues regarding memory management.

Advantages of Virtual Memory

- **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 the 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 the main memory as required.

- It allows greater multiprogramming levels by using less of the available (primary) memory for each process.
- It has twice the capacity for addresses as main memory.
- It makes it possible to run more applications at once.
- Users are spared from having to add memory modules when RAM space runs out, and applications are liberated from shared memory management.
- When only a portion of a program is required for execution, speed has increased.
- Memory isolation has increased security.
- It makes it possible for several larger applications to run at once.
- Memory allocation is comparatively cheap.
- It doesn't require outside fragmentation.
- It is efficient to manage logical partition workloads using the CPU.
- Automatic data movement is possible.

Disadvantages of Virtual Memory

- It can slow down the system performance, as data needs to be constantly transferred between the physical memory and the hard disk.
- It can increase the risk of data loss or corruption, as data can be lost if the hard disk fails or if there is a power outage while data is being transferred to or from the hard disk.
- It can increase the complexity of the memory management system, as the operating system needs to manage both physical and virtual memory.

DEMAND PAGING

What is Demand Paging?

Demand paging is a technique used in virtual memory systems where pages enter main memory only when requested or needed by the CPU. In demand paging, the operating system loads only the necessary pages of a program into memory at runtime, instead of loading the entire program into memory at the start. A page fault occurred when the program needed to access a page that is not currently in memory.

The operating system then loads the required pages from the disk into memory and updates the page tables accordingly. This process is transparent to the running program and it continues to run as if the page had always been in memory.

What is Page Fault?

The term “page miss” or “page fault” refers to a situation where a referenced page is not found in the main memory.

When a program tries to access a page, or fixed-size block of memory, that isn’t currently loaded in physical memory (RAM), an exception known as a page fault happens. Before enabling the program to access a page that is required, the operating system must bring it into memory from secondary storage (such a hard drive) in order to handle a page fault.

In modern operating systems, page faults are a common component of virtual memory management. By enabling programs to operate with more data than can fit in physical memory at once, they enable the efficient use of physical memory. The operating system is responsible for coordinating the transfer of data between physical memory and secondary storage as needed.

Pure Demand Paging

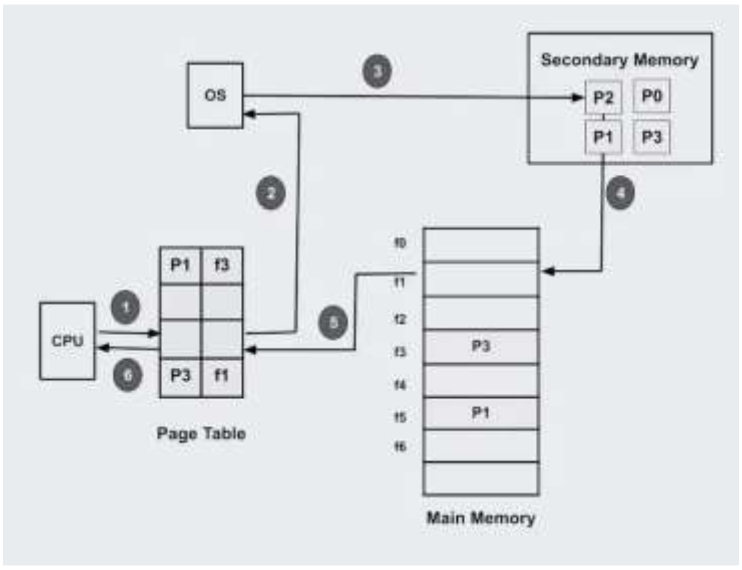
Pure demand paging is a specific implementation of demand paging. The operating system only loads pages into memory when the program needs them. In on-demand paging only, no pages are initially loaded into memory when the program starts, and all pages are initially marked as being on disk.

Operating systems that use pure demand paging as a memory management strategy do so without preloading any pages into physical memory prior to the commencement of a task. Demand paging loads a process’s whole address space into memory one step at a time, bringing just the parts of the process that are actively being used into memory from disc as needed.

It is useful for executing huge programs that might not fit totally in memory or for computers with limited physical memory. If the program accesses a lot of pages that are not in memory right now, it could also result in a rise in page faults and possible performance overhead. Operating systems frequently use caching techniques and improve page replacement algorithms to lessen the negative effects of page faults on system performance as a whole.

Working Process of Demand Paging

Let us understand this with the help of an example. Suppose we want to run a process P which have four pages P0, P1, P2, and P3. Currently, in the page table, we have pages P1 and P3.



The operating system's demand paging mechanism follows a few steps in its operation.

- **Program Execution:** Upon launching a program, the operating system allocates a certain amount of memory to the program and establishes a process for it.
- **Creating Page Tables:** To keep track of which program pages are currently in memory and which are on disk, the operating system makes page tables for each process.

- **Handling Page Fault:** When a program tries to access a page that isn't in memory at the moment, a page fault happens. In order to determine whether the necessary page is on disk, the operating system pauses the application and consults the page tables.
- **Page Fetch:** The operating system loads the necessary page into memory by retrieving it from the disk if it is there.
- The page's new location in memory is then reflected in the page table.
- **Resuming The Program:** The operating system picks up where it left off when the necessary pages are loaded into memory.
- **Page Replacement:** If there is not enough free memory to hold all the pages a program needs, the operating system may need to replace one or more pages currently in memory with pages currently in memory. on the disk. The page replacement algorithm used by the operating system determines which pages are selected for replacement.
- **Page Clean up:** When a process terminates, the operating system frees the memory allocated to the process and cleans up the corresponding entries in the page tables.

How Demand Paging in OS Affects System Performance?

Demand paging can improve system performance by reducing the memory needed for programs and allowing multiple programs to run simultaneously. However, if not implemented properly, it can cause performance issues. When a program needs a part that isn't in the main memory, the operating system must fetch it from the hard disk, which takes time and pauses the program. This can cause delays, and if the system runs out of memory, it will need to frequently swap pages in and out, increasing delays and reducing performance.

Common Algorithms Used for Demand Paging in OS

Demand paging is a memory management technique that loads parts of a program into memory only when needed. If a program needs a page that isn't currently in memory, the system fetches it from the hard disk. Several algorithms manage this process:

- **FIFO (First-In-First-Out):** Replaces the oldest page in memory with a new one. It's simple but can cause issues if pages are frequently swapped in and out, leading to thrashing.
- **LRU (Least Recently Used):** Replaces the page that hasn't been used for the longest time. It reduces thrashing more effectively than FIFO but is more complex to implement.
- **LFU (Least Frequently Used):** Replaces the page used the least number of times. It helps reduce thrashing but requires extra tracking of how often each page is used.
- **MRU (Most Recently Used):** Replaces the page that was most recently used. It's simpler than LRU but not as effective in reducing thrashing.
- **Random:** Randomly selects a page to replace. It's easy to implement but unpredictable in performance.

What is the Impact of Demand Paging in Virtual Memory Management?

With demand paging, the operating system swaps memory pages between the main memory and secondary storage based on need. When a program needs a page not currently in memory, the operating system retrieves it from secondary storage, a process called a page fault.

Demand paging significantly impacts virtual memory management by allowing the operating system to use virtual memory efficiently, improving overall system performance. Its main advantage is reducing the physical memory required, enabling more applications to run at once and allowing larger programs to run.

However, demand paging has some drawbacks. The page fault mechanism can delay program execution because the operating system must retrieve pages from secondary storage. This delay can be minimized by optimizing the page replacement algorithm.

Demand Paging in OS vs Pre-Paging

Demand paging and pre-paging are two memory management techniques used in operating systems.

Demand paging loads pages from disk into main memory only when they are needed by a program. This approach saves memory space by keeping only the required pages in memory, reducing memory allocation costs and improving memory use. However, the initial access time for pages not in memory can delay program execution.

Pre-paging loads multiple pages into main memory before they are needed by a program. It assumes that if one page is needed, nearby pages will also be needed soon. Pre-paging can speed up program execution by reducing delays caused by demand paging but can lead to unnecessary memory allocation and waste.

Advantages of Demand Paging

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

- **Efficient use of physical memory**: Query paging allows for more efficient use because only the necessary pages are loaded into memory at any given time.
- **Support for larger programs:** Programs can be larger than the physical memory available on the system because only the necessary pages will be loaded into memory.
- **Faster program start:** Because only part of a program is initially loaded into memory, programs can start faster than if the entire program were loaded at once.
- **Reduce memory usage:** Query paging can help reduce the amount of memory a program needs, which can improve system performance by reducing the amount of disk I/O required.

Disadvantages of Demand Paging

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

Demand paging can significantly affect the performance of a computer system. Let's compute the effective access time for a demand-paged memory.

For most computer systems, the memory-access time, denoted ma, ranges from 10 to 200 nanoseconds.
As long as we have no page faults, the effective access time is equal to the memory access time.
If, however a page fault occurs, we must first read the relevant page from disk and then access the desired word.
Let p be the probability of a page fault ($0 <= p <= 1$). We would expect p to be close to zero -that is, we would expect to have only a few page faults.

The effective access time is then
effective access time = (1 – p)*ma + p*page fault time

We are faced with three major components of the page-fault service time:

- Service the page-fault interrupt.
- Read in the page.

- Restart the process.

The first and third tasks can be reduced, with careful coding, to several hundred instructions. These tasks may take from 1 to 100 microseconds each.

The page-switch time, however, will probably be close to 8 milliseconds.

A typical hard disk has an average latency of 3 milliseconds, a seek of 5 milliseconds, and a transfer time of 0.05 milliseconds.

Thus, the total paging time is about 8 milliseconds, including hardware and software time.

If we take an average page-fault service time of 8 milliseconds and a memory-access time of 200 nanoseconds, then the effective access time in nanoseconds is

effective access time = (1 - p)*(200) + p*(8 milliseconds)

= (1 - p)*200 + p*8,000,000

= 200 + 7,999,800 x p.

We see, then, that the effective access time is directly proportional to the page-fault rate.

If one access out of 1,000 causes a page fault, the effective access time is 8.2 microseconds. The computer will be slowed down by a factor of 40 because of demand paging!

It is important to keep the page-fault rate low in a demand-paging system. Otherwise, the effective access time increases, slowing process execution dramatically.

An additional aspect of demand paging is the handling and overall use of swap space.

Disk I/O to swap space is generally faster than that to the file system. It is faster because swap space is allocated in much larger blocks, and file lookups and indirect allocation methods are not used.

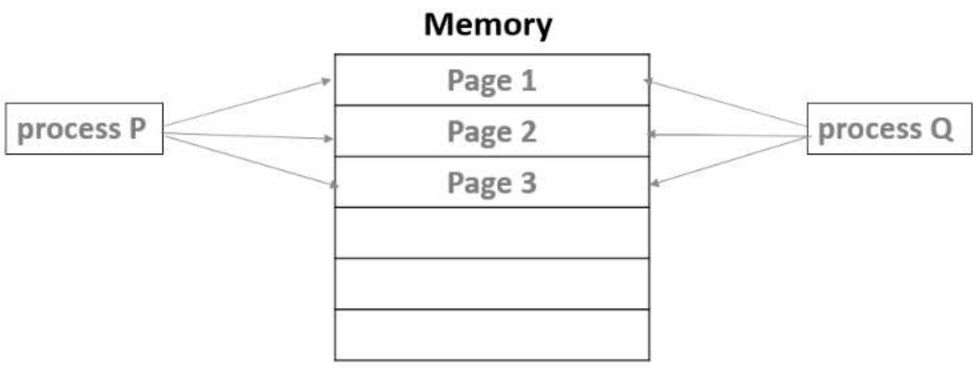
The system can therefore gain better paging throughput by copying an entire file image into the swap space at process startup and then performing demand paging from the swap space.

Another option is to demand pages from the file system initially but to write the pages to swap space as they are replaced.

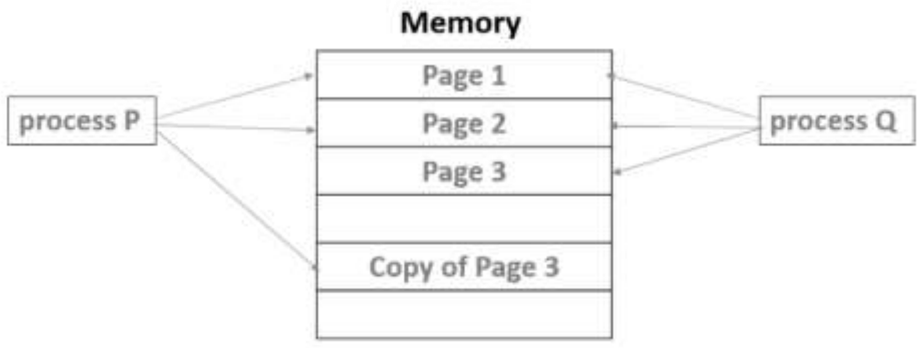
Copy on Write or simply COW is a resource management technique. One of its main use is in the implementation of the fork system call in which it shares the virtual memory(pages) of the OS.

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

PAGE REPLACEMENT

Page Replacement Algorithms

Page replacement algorithms are techniques used in **operating systems** to manage **memory** efficiently when the **virtual memory** is full. When a new page needs to be loaded into **physical memory** , and there is no free space, these algorithms determine which existing page to replace.

If no page frame is free, the virtual memory manager performs a page replacement operation to replace one of the pages existing in memory with the page whose reference caused the page fault. It is performed as follows: The virtual memory manager uses a page replacement algorithm to select one of the pages currently in memory for replacement, accesses the page table entry of the selected page to mark it as “not present” in memory, and initiates a page-out operation for it if the modified bit of its page table entry indicates that it is a dirty page.

Common Page Replacement Techniques

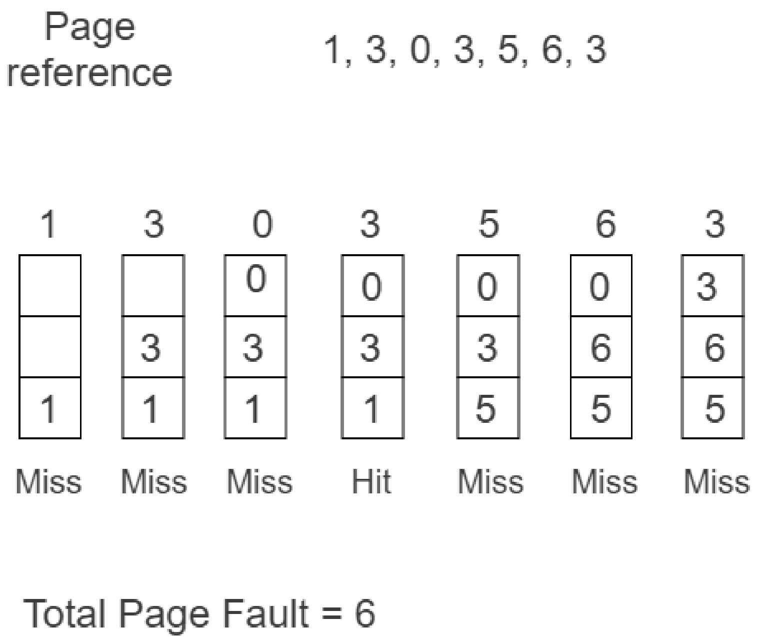
- First In First Out (FIFO)
- Optimal Page replacement
- Least Recently Used
- Most Recently Used (MRU)

FIFO PAGE REPLACEMENT

First In First Out (FIFO) Page Replacement

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Example 1: Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3 page frames. Find the number of page faults.



Initially, all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.**

when 3 comes, it is already in memory so —> **0 Page Faults.** Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —> **1 Page Fault.** 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —> **1 Page Fault.** Finally, when 3 come it is not available so it replaces 0 **1 page fault.**

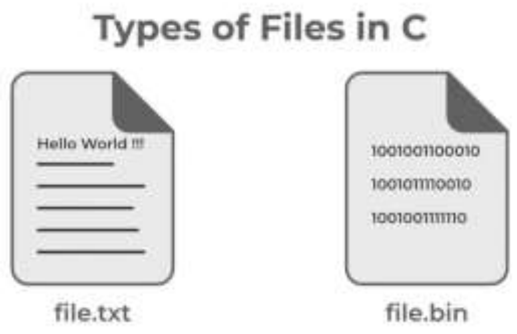
Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out page replacement algorithm. For example, if we consider reference strings 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4, and 3 slots, we get 9 total page faults, but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

PROGRAM USING FILE SYSTEM

Types of Files in C

A file can be classified into two types based on the way the file stores the data. They are as follows:

- **Text Files**
- **Binary Files**



1. Text Files

A text file contains data in the **form of ASCII characters** and is generally used to store a stream of characters.

- Each line in a text file ends with a new line character ('\n').
- It can be read or written by any text editor.
- They are generally stored with **.txt** file extension.
- Text files can also be used to store the source code.

2. Binary Files

A binary file contains data in **binary form (i.e. 0's and 1's)** instead of ASCII characters. They contain data that is stored in a similar manner to how it is stored in the main memory.

- The binary files can be created only from within a program and their contents can only be read by a program.
- More secure as they are not easily readable.

- They are generally stored with **.bin** file extension.

C File Operations

C file operations refer to the different possible operations that we can perform on a file in C such as:

1. Creating a new file – **fopen() with attributes as “a” or “a+” or “w” or “w+”**
2. Opening an existing file – **fopen()**
3. Reading from file – **fscanf() or fgets()**
4. Writing to a file – **fprintf() or fputs()**
5. Moving to a specific location in a file – **fseek(), rewind()**
6. Closing a file – **fclose()**

The highlighted text mentions the C function used to perform the file operations.
Functions for C File Operations

File Pointer in C

A file pointer is a reference to a particular position in the opened file. It is used in file handling to perform all file operations such as read, write, close, etc. We use the **FILE** macro to declare the file pointer variable. The FILE macro is defined inside **<stdio.h>** header file.

Syntax of File Pointer

```
FILE* pointer_name;
```

File Pointer is used in almost all the file operations in C.

Open a File in C

For opening a file in C, the fopen() function is used with the filename or file path along with the required access modes.

Syntax of fopen()

```
FILE* fopen(const char *file_name, const char *access_mode);
```

Parameters

- file_name: name of the file when present in the same directory as the source file. Otherwise, full path.
- access_mode: Specifies for what operation the file is being opened.

Return Value

- If the file is opened successfully, returns a file pointer to it.
- If the file is not opened, then returns NULL.

File opening modes in C

File opening modes or access modes specify the allowed operations on the file to be opened. They are passed as an argument to the fopen() function. Some of the commonly used file access modes are listed below:

Opening Modes	**Description**
r	Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer that points to the first character in it. If the file cannot be opened fopen( ) returns NULL.
rb	Open for reading in binary mode. If the file does not exist, fopen( ) returns NULL.
w	Open for writing in text mode. If the file exists, its contents are overwritten. If the file doesn't exist, a new file is created. Returns NULL, if unable to open the file.
wb	Open for writing in binary mode. If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer that points to the last character in it. It

Opening Modes	Description
	opens only in the append mode. If the file doesn't exist, a new file is created. Returns NULL, if unable to open the file.
ab	Open for append in binary mode. Data is added to the end of the file. If the file does not exist, it will be created.
r+	Searches file. It is opened successfully fopen( ) loads it into memory and sets up a pointer that points to the first character in it. Returns NULL, if unable to open the file.
rb+	Open for both reading and writing in binary mode. If the file does not exist, fopen( ) returns NULL.
w+	Searches file. If the file exists, its contents are overwritten. If the file doesn't exist a new file is created. Returns NULL, if unable to open the file.
wb+	Open for both reading and writing in binary mode. If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Searches file. If the file is opened successfully fopen( ) loads it into memory and sets up a pointer that points to the last character in it. It opens the file in both reading and append mode. If the file doesn't exist, a new file is created. Returns NULL, if unable to open the file.
ab+	Open for both reading and appending in binary mode. If the file does not exist, it will be created.

As given above, if you want to perform operations on a binary file, then you have to append ‘b’ at the last. For example, instead of “w”, you have to use “wb”, instead of “a+” you have to use “a+b”.

Example of Opening a File

```
// C Program to illustrate file opening
#include <stdio.h>
#include <stdlib.h>

int main()
{
   // file pointer variable to store the value returned by
   // fopen
   FILE* fptr;

   // opening the file in read mode
   fptr = fopen("filename.txt", "r");

   // checking if the file is opened successfully
   if (fptr == NULL) {
      printf("The file is not opened. The program will "
             "now exit.");
      exit(0);
   }
   return 0;
}
```

Create a File in C

The fopen() function can not only open a file but also can create a file if it does not exist already. For that, we have to use the modes that allow the creation of a file if not found such as w, w+, wb, wb+, a, a+, ab, and ab+.

```
        FILE *fptr;
        fptr = fopen("filename.txt", "w");
```

Example of Opening a File

```
// C Program to create a file
#include <stdio.h>
#include <stdlib.h>

int main()
{
    // file pointer
    FILE* fptr;

    // creating file using fopen() access mode "w"
    fptr = fopen("file.txt", "w");

    // checking if the file is created
    if (fptr == NULL) {
        printf("The file is not opened. The program will "
               "exit now");
        exit(0);
    }
    else {
        printf("The file is created Successfully.");
    }

    return 0;
}
```

Reading From a File

The file read operation in C can be performed using functions fscanf() or fgets(). Both the functions performed the same operations as that of scanf and gets but with an additional parameter, the file pointer. There are also other functions we can use to read from a file. Such functions are listed below:

Function	**Description**
fscanf()	Use formatted string and variable arguments list to take input from a file.
fgets()	Input the whole line from the file.
fgetc()	Reads a single character from the file.
fgetw()	Reads a number from a file.
fread()	Reads the specified bytes of data from a binary file.

So, it depends on you if you want to read the file line by line or character by character.

Example:

```
FILE * fptr;
fptr = fopen("fileName.txt", "r");
fscanf(fptr, "%s %s %s %d", str1, str2, str3, &year);
char c = fgetc(fptr);
```

The getc() and some other file reading functions return EOF (End Of File) when they reach the end of the file while reading. EOF indicates the end of the file and its value is implementation-defined.

Note: One thing to note here is that after reading a particular part of the file, the file pointer will be automatically moved to the end of the last read character.

Write to a File

The file write operations can be performed by the functions fprintf() and fputs() with similarities to read operations. C programming also provides some other functions that can be used to write data to a file such as:

Function	**Description**
fprintf()	Similar to printf(), this function use formatted string and varible arguments list to print output to the file.
fputs()	Prints the whole line in the file and a newline at the end.
fputc()	Prints a single character into the file.
fputw()	Prints a number to the file.
fwrite()	This functions write the specified amount of bytes to the binary file.

Example:

```
FILE *fptr ;
fptr = fopen(“fileName.txt”, “w”);
fprintf(fptr, "%s %s %s %d", "We", "are", "in", 2012);
fputc("a", fptr);
```

Closing a File

The fclose() function is used to close the file. After successful file operations, you must always close a file to remove it from the memory.

Syntax of fclose()

```
fclose(file_pointer);
```

where the file_pointer is the pointer to the opened file.

Example:

```
FILE *fptr ;
fptr= fopen("fileName.txt", "w");
---------- Some file Operations -------
fclose(fptr);
```

<u>Conclusion</u>

This week we learned about the Structure of Page Table, Paged Segmentation, Virtual Memory, Demand Paging, Performance of Demand Paging and Copy-on-Write, FIFO Page replacement and Program using file system