UNIT – 2 : MEMORY MANAGEMENT

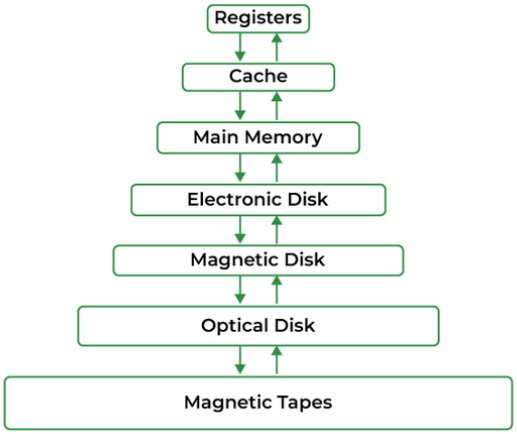
Introduction / Background:

The term **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 purpose 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?

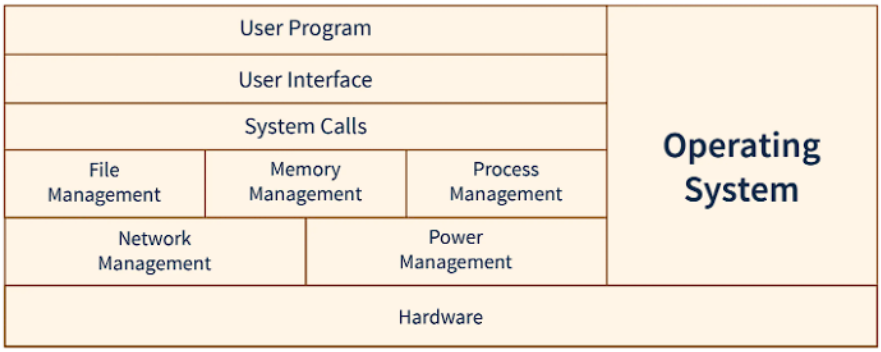
The main memory is central to the operation of a Modern Computer. Main Memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions. Main memory is a repository of rapidly available information shared by the CPU and I/O devices. Main memory is the place where programs 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 volatile. RAM loses its data when a power interruption occurs.



What is Memory Management?

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

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



Memory Management in OS

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 Address Space and Physical Address Space:

In the realm of memory management, distinguishing between logical address space and physical address space is paramount.

Logical Address Space encompasses the range of addresses that a CPU can generate. This is the perspective from which a program "sees" its memory. For instance, if a computer has 4 GB of RAM, the logical address space might span from 0 to 4 billion. However, it's important to note that this space is not entirely occupied by physical memory. Instead, it serves as a convenient and abstract representation.

Physical Address Space, on the other hand, is the actual physical location in the memory hardware where data is stored. It constitutes the tangible address of a storage cell in RAM. In our previous example, if the RAM modules consist of 4 billion cells, the physical address space would correspond to each of these individual cells.

The **Memory Management Unit (MMU)** plays a pivotal role in this interplay. It acts as an intermediary, translating logical addresses to physical addresses. This enables programs to operate in a seemingly large logical address space, while efficiently utilizing the available physical memory.

Example: Consider a scenario where a program attempts to access memory address 'x' in its logical address space. The MMU translates this to the corresponding physical address 'y' and retrieves the data from the actual RAM location. This abstraction allows for efficient multitasking and memory allocation

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 involves loading all the necessary program components into the main memory before the program's execution begins. This means that both the executable code and data are loaded into predetermined memory locations. This allocation is fixed and does not change during the program's execution. While it ensures direct access to all required resources, it may lead to inefficiencies in memory usage, especially if the program doesn't utilize all the loaded components.

Dynamic Loading, on the other hand, takes a more flexible approach. In this scheme, a program's components are loaded into the main memory only when they are specifically requested during execution. This results in a more efficient use of memory resources as only the necessary components occupy space. Dynamic loading is particularly advantageous for programs with extensive libraries or functionalities that may not be used in every session.

Static and Dynamic Linking:

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

Static Linking involves incorporating all necessary libraries and modules into the final executable at compile time. This means that the code from libraries is copied into the final executable file. The result is a self-contained executable that doesn't rely on external resources during runtime. While this ensures portability and guarantees that the program will run on any system, it can lead to larger file sizes and potential redundancy if multiple programs use the same libraries.

Dynamic Linking, on the other hand, takes a more dynamic approach. In this method, the necessary libraries are not included in the final executable. Instead, the program dynamically links to the required libraries at runtime. This means that multiple programs can share a single copy of a library, reducing redundancy and conserving memory. However, it does introduce a dependency on the availability of the required libraries at runtime.

Example: Consider a scenario where multiple programs use a common math library. With static linking, each program would contain its own copy of the library, potentially leading to larger file sizes. With dynamic linking, all programs can use the same shared instance of the library, saving disk space

Swapping:

Let's suppose there are several processes like P1, P2, P3, and P4 that are ready to be executed inside the ready queue, and processes P1 and P2 are very memory consuming so when the processes start executing there may be a scenario where the memory will not be available for the execution of the process P3 and P4 as there is a limited amount of memory available for process execution.

Swapping in the operating system is a memory management scheme that temporarily swaps out an idle or blocked process from the main memory to secondary memory which ensures proper memory utilization and memory availability for those processes which are ready to be executed.

When that memory-consuming process goes into a termination state means its execution is over due to which the memory dedicated to their execution becomes free Then the swapped-out processes are brought back into the main memory and their execution starts.

The area of the secondary memory where swapped-out processes are stored is called **swap space**. The swapping method forms a temporary queue of swapped processes in the secondary memory.

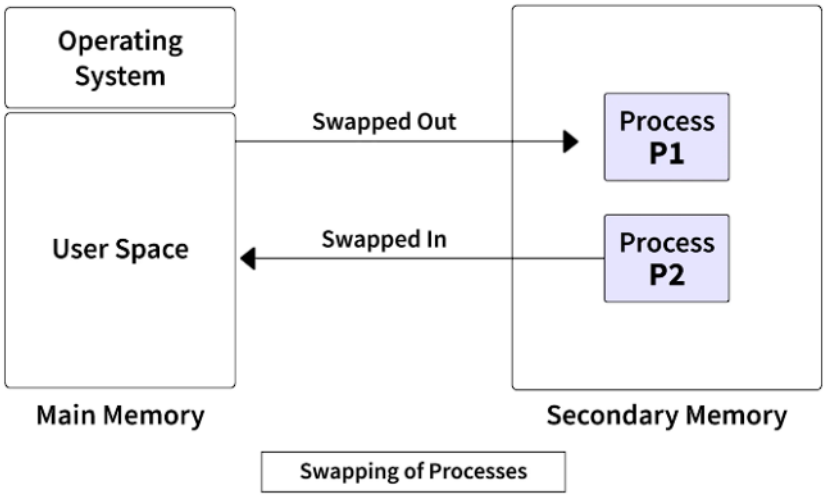
In the case of high-priority processes, the process with low priority is swapped out of the main memory and stored in swap space then the process with high priority is swapped into the main memory to be executed first.

The main goals of an operating system include **Maximum utilization of the CPU**. This means that there should be a process execution every time, the **CPU** should never stay idle and there should not be any **Process starvation** or **blocking**.

Different process management and memory management schemes are designed to fulfill such goals of an operating system.

Swapping in **OS** is done to get access to data present in secondary memory and transfer it to the main memory so that it can be used by the application programs.

It can affect the performance of the system but it helps in running more than one process by managing the memory. Therefore, swapping in OS is also known as the **memory compaction technique**.



Example:

Let's understand the concept of **swapping** with an example:

Suppose we have a user whose process size is 4096KB. Here the user is having a standard hard disk drive in which the swapping has a transfer rate of 4Mbps. Now we will compute how long it takes to transfer the data from the main memory which is **RAM** to the secondary memory which is the **hard disk**.

Solution:

process size of the user is 4096Kb

the transfer rate of data is 4Mbps

Now,

4 Mbps is equal to 4096 kbps

Time taken to transfer the data = process size of user/ transfer rate of data

Now, coming to the Calculation part:

Time Taken = 4096 / 4096 = 1 second = 1000 milliseconds

Now, taking both swap in and swap out time into account the total time taken for transferring the data from **main memory** to **secondary memory** is equal to 2000 milliseconds which is 2 Seconds.

There are two important concepts in the process of swapping which are as follows:

Swap In: The method of removing a process from secondary memory (**Hard Drive**) and restoring it to the main memory (**RAM**) for execution is known as the Swap In method.

Swap Out: It is a method of bringing out a process from the main memory (**RAM**) and sending it to the secondary memory (**hard drive**) so that the processes with higher priority or more memory consumption will be executed known as the Swap Out method.

Note: Swap In and Swap Out method is done by **Medium Term Scheduler(MTS)**.

Advantages of Swapping:

1. It helps the CPU to manage multiple processes within a single main memory.
2. It helps to create and use virtual memory.
3. Swapping allows the CPU to perform multiple tasks simultaneously. Therefore, processes do not have to wait very long before they are executed.
4. It improves the main memory utilization.

Disadvantages of Swapping:

1. If the computer system loses power, the user may lose all information related to the program in case of substantial swapping activity.
2. If the swapping algorithm is not good, the composite method can increase the number of Page Fault and decrease the overall processing performance.

Note:

- In a single tasking operating system, only one process occupies the user program area of memory and stays in memory until the process is complete.
- In a multitasking operating system, a situation arises when all the active processes cannot coordinate in the main memory, then a process is swap out from the main memory so that other processes can enter it.

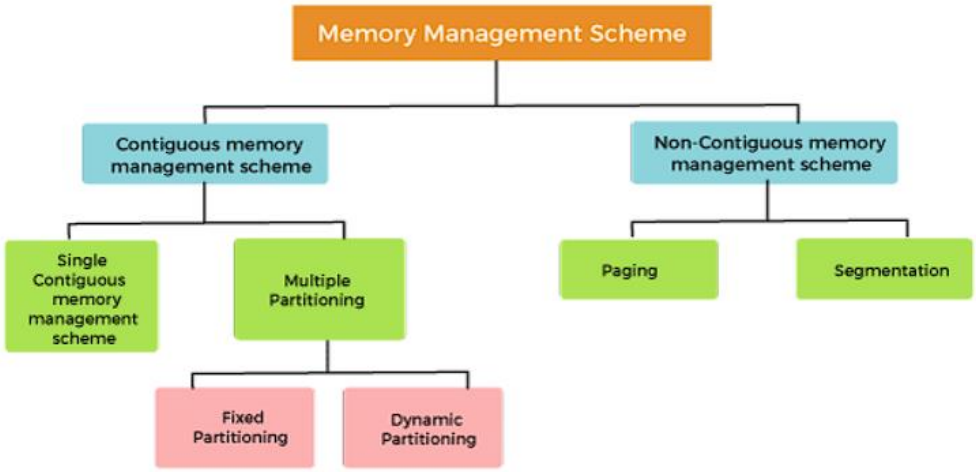
Memory Management/Allocation Techniques:

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

- Contiguous memory management schemes
- Non-Contiguous memory management schemes

Main memory usually has two partitions −

- **Low Memory** − Operating system resides in this memory.
- **High Memory** − User processes are held in high memory.



Classification of memory management schemes

Contiguous Memory Allocation:

Contiguous memory allocation is a memory management technique that involves allocating a process to the entire contiguous block of the main memory it requires to execute. This means that the process is loaded into a single continuous chunk of memory. While it's straightforward and efficient in terms of execution, it can lead to issues with fragmentation, where smaller blocks of memory remain unused between allocated processes.

Memory Allocation:

Memory allocation is the process of reserving a portion of the computer's memory for a specific application or program. It's a crucial aspect of memory management, ensuring that each running process has enough space to execute efficiently. Effective memory allocation strategies are essential for optimizing system performance.

Single Contiguous Memory Management Schemes:

The Single contiguous memory management scheme is the simplest memory management scheme used in the earliest generation of computer systems. In this scheme, the main memory is divided into two contiguous areas or partitions. The operating systems reside permanently in one partition, generally at the lower memory, and the user process is loaded into the other partition.

Advantages of Single contiguous memory management schemes:

- Simple to implement.
- Easy to manage and design.
- In a Single contiguous memory management scheme, once a process is loaded, it is given full processor's time, and no other processor will interrupt it.

Disadvantages of Single contiguous memory management schemes:

- Wastage of memory space due to unused memory as the process is unlikely to use all the available memory space.
- The CPU remains idle, waiting for the disk to load the binary image into the main memory.
- It cannot be executed if the program is too large to fit the entire available main memory space.
- It does not support multiprogramming, i.e., it cannot handle multiple programs simultaneously.

Multiple Partitioning Schemes:

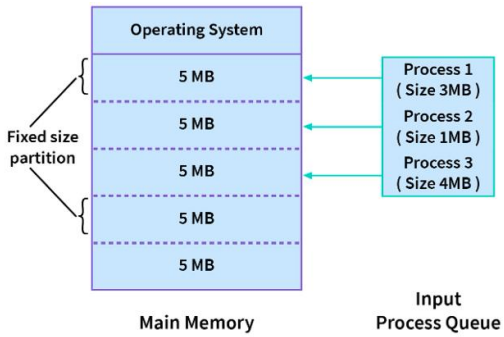
The single Contiguous memory management scheme is inefficient as it limits computers to execute only one program at a time resulting in wastage in memory space and CPU time. The problem of inefficient CPU use can be overcome using multiprogramming that allows more than one program to run concurrently. To switch between two processes, the operating systems need to load both processes into the main memory. The operating system needs to divide the available main memory into multiple parts to load multiple processes into the main memory. Thus, multiple processes can reside in the main memory simultaneously.

The multiple partitioning schemes can be of two types:

1. Fixed-size Partition Scheme
2. Variable-size/Dynamic Partition Scheme

1. Fixed-size Partition Scheme:

In this type of contiguous memory allocation technique, **each process is allotted a fixed-size continuous block in the main memory.** That means there will be continuous blocks of fixed size into which the complete memory will be divided, and each time a process comes in, it will be allotted one of the free blocks. Because irrespective of the size of the process, each is allotted a block of the same size memory space. This technique is also called static partitioning.



In the diagram above, we have 3 processes in the input queue that have to be allotted space in the memory. As we are following the fixed-size partition technique, the memory has fixed-sized blocks. The first process, which is of size 3MB is also allotted a 5MB block, and the second process, which is of size 1MB, is also allotted a 5MB block, and the 4MB process is also allotted a 5MB block. So, the process size doesn't matter. Each is allotted the same fixed-size memory block.

It is clear that in this scheme, the number of continuous blocks into which the memory will be divided will be decided by the amount of space each block covers, and this, in turn, will dictate how many processes can stay in the main memory at once.

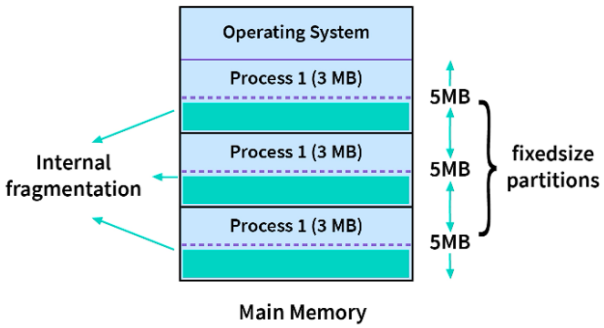
Note: The number of processes that can stay in the memory at once is called the **degree of multiprogramming**. Hence, the degree of multiprogramming of the system is decided by the number of blocks created in the memory.

Advantages:

1. Because all of the blocks are the same size, this scheme is simple to implement. All we have to do now is divide the memory into fixed blocks and assign processes to them.
2. It is easy to keep track of how many blocks of memory are left, which in turn decides how many more processes can be given space in the memory.
3. As at a time multiple processes can be kept in the memory, this scheme can be implemented in a system that needs multiprogramming.

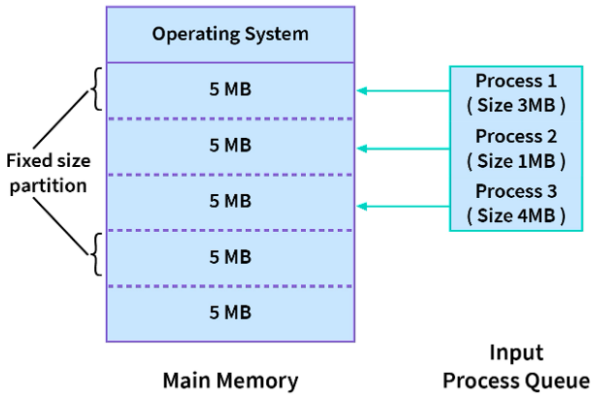
Disadvantages:

1. As the size of the blocks is fixed, we will not be able to allot space to a process that has a greater size than the block.
2. The size of the blocks decides the degree of multiprogramming, and only that many processes can remain in the memory at once as the number of blocks.
3. If the size of the block is greater than the size of the process, we have no other choice but to assign the process to this block, but this will lead to much empty space left behind in the block. This empty space could've been used to accommodate a different process. This is called internal fragmentation. Hence, this technique may lead to space wastage.



2. Variable-size Partition Scheme:

In this type of contiguous memory allocation technique, **no fixed blocks or partitions are made in the memory.** Instead, each process is allotted a variable-sized block depending upon its requirements. That means, that whenever a new process wants some space in the memory, if available, this amount of space is allotted to it. Hence, the size of each block depends on the size and requirements of the process which occupies it.



In the diagram above, there are no fixed-size partitions. Instead, the first process needs 3MB memory space and hence is allotted that much only. Similarly, the other 3 processes are allotted only that much space that is required by them.

As the blocks are variable-sized, which is decided as processes arrive, this scheme is also called Dynamic Partitioning.

Advantages:

1. As the processes have blocks of space allotted to them as per their requirements, there is no internal fragmentation. Hence, there is no memory wastage in this scheme.
2. The number of processes that can be in the memory at once will depend upon how many processes are in the memory and how much space they occupy. Hence, it will be different for different cases and will be dynamic.
3. As there are no blocks that are of fixed size, even a process of big size can be allotted space.

Disadvantages:

Though the variable-size partition scheme has many advantages, it also has some disadvantages:

1. Because this approach is dynamic, a variable-size partition scheme is difficult to implement.
2. It is difficult to keep track of processes and the remaining space in the memory.

Strategies Used for Contiguous Memory Allocation Input Queues:

So far, we've seen the two types of schemes for contiguous memory allocation. But what happens when a new process comes in and has to be allotted a space in the main memory? How is it decided which block or segment it will get?

Processes that have been assigned continuous blocks of memory will fill the main memory at any given time. However, when a process completes, it leaves behind an empty block known as a hole. This space could also be used for a new process. Hence, the main memory consists of processes and holes, and any one of these holes can be allotted to a new incoming process.

We have three strategies to allot a hole to an incoming process:

First-Fit: This is a very basic strategy in which we start from the beginning and allot the first hole, which is big enough as per the requirements of the process. The first-fit strategy can also be implemented in a way where we can start our search for the first-fit hole from the place we left off last time.

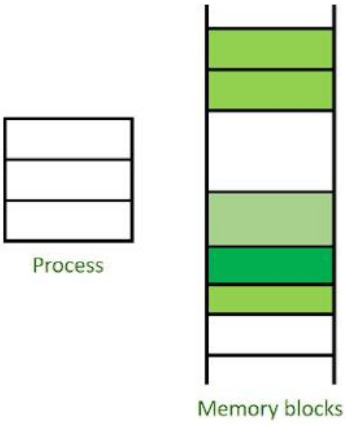
Best-Fit: This is a greedy strategy that aims to reduce any memory wasted because of internal fragmentation in the case of static partitioning, and hence we allot that hole to the process, which is the smallest hole that fits the requirements of the process. Hence, we need to first sort the holes according to their sizes and pick the best fit for the process without wasting memory.

Worst-Fit: This strategy is the opposite of the Best-Fit strategy. We sort the holes according to their sizes and choose the largest hole to be allotted to the incoming process. The idea behind this allocation is that as the process is allotted a large hole, it will have a lot of space left behind as internal fragmentation. Hence, this will create a hole that will be large enough to accommodate a few other processes.

Non-Contiguous Memory Allocation:

Non-contiguous allocation, also known as dynamic or linked allocation, is a memory allocation technique used in operating systems to allocate memory to processes that do not require a contiguous block of memory. In this technique, each process is allocated a series of non-contiguous blocks of memory that can be located anywhere in the physical memory.

Non-contiguous allocation involves the use of pointers to link the non-contiguous memory blocks allocated to a process. These pointers are used by the operating system to keep track of the memory blocks allocated to the process and to locate them during the execution of the process.



Noncontiguous Memory Allocation

Fundamental Approaches:

There are two fundamental approaches to implementing non-contiguous memory allocation:

- **Paging** − In paging, the memory is divided into fixed-size pages, and each page is assigned to a process. This technique is more efficient as it allows the allocation of only the required memory to the process.
- **Segmentation** − In segmentation, the memory is divided into variable-sized segments, and each segment is assigned to a process. This technique is more flexible than paging but requires more overhead to keep track of the allocated segments.

There are several advantages to non-contiguous allocation.

First, it reduces internal fragmentation since memory blocks can be allocated as needed, regardless of their physical location.

Second, it allows processes to be allocated memory in a more flexible and efficient manner since the operating system can allocate memory to a process wherever free memory is available.

However, non-contiguous allocation also has some disadvantages.

It can lead to external fragmentation, where the available memory is broken into small, non-contiguous blocks, making it difficult to allocate large blocks of memory to a process.

Additionally, the use of pointers to link memory blocks can introduce additional overhead, leading to slower memory allocation and deallocation times.

Difference between Contiguous and Non-contiguous Memory Allocation:

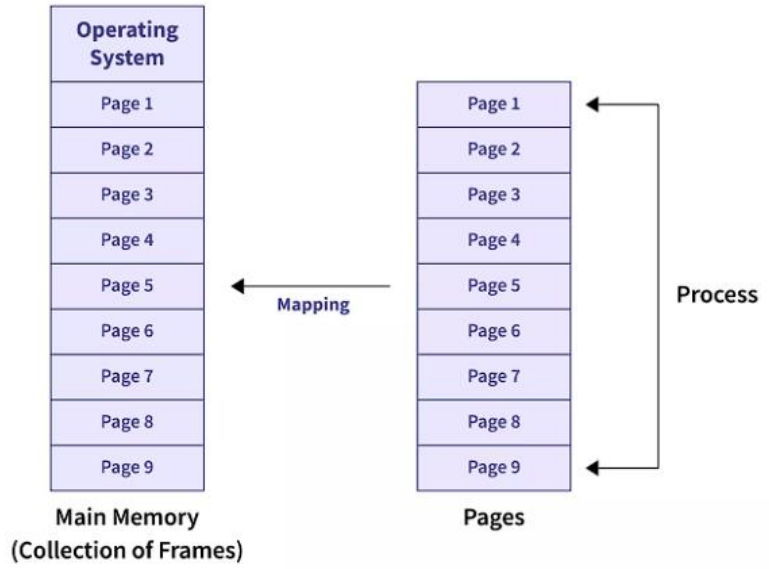
S.NO.	Contiguous Memory Allocation	Non-Contiguous Memory Allocation
1.	Contiguous memory allocation allocates consecutive blocks of memory to a file/process.	Non-Contiguous memory allocation allocates separate blocks of memory to a file/process.
2.	Faster in Execution.	Slower in Execution.
3.	It is easier for the OS to control.	It is difficult for the OS to control.
4.	Overhead is minimum as not much address translations are there while executing a process.	More Overheads are there as there are more address translations.
5.	Both Internal fragmentation and external fragmentation occurs in Contiguous memory allocation method.	Only External fragmentation occurs in Non-Contiguous memory allocation method.
6.	It includes single partition allocation and multi-partition allocation.	It includes paging and segmentation.
7.	Wastage of memory is there.	No memory wastage is there.
8.	In contiguous memory allocation, swapped-in processes are arranged in the originally allocated space.	In non-contiguous memory allocation, swapped-in processes can be arranged in any place in the memory.
9.	It is of two types: 1. Fixed (or static) partitioning 2. Dynamic partitioning	It is of five types: 1. Paging 2. Multilevel Paging 3. Inverted Paging 4. Segmentation 5. Segmented Paging
10.	It could be visualized and implemented using Arrays.	It could be implemented using Linked Lists.
11.	Degree of multiprogramming is fixed as fixed partitions	Degree of multiprogramming is not fixed

Paging:

Paging is a technique that divides memory into fixed-sized blocks. The main memory is divided into blocks known as **Frames** and the logical memory is divided into blocks known as **Pages**. Paging requires extra time for the address conversion, so we use a special hardware cache memory known as **TLB**.

This concept of Paging in OS includes dividing each process in the form of pages of equal size and also, the main memory is divided in the form of frames of fixed size. Now, each page of the process when retrieved into the main memory, is stored in one frame of the memory, and hence, it is also important to have the pages and frames of equal size for mapping and maximum utilization of the memory.

Its main advantage is that the pages can be stored at different locations of the memory and not necessarily in a contiguous manner, though priority is always set to firstly find the contiguous frames for allocating the pages.



If a process has n pages in the secondary memory, then there must be n frames available in the main memory for mapping.

Frames vs Pages:

Frames are basically the sliced up physical memory blocks of equal size. Example: 512kb of memory can be divided into 4 parts for 128kb each

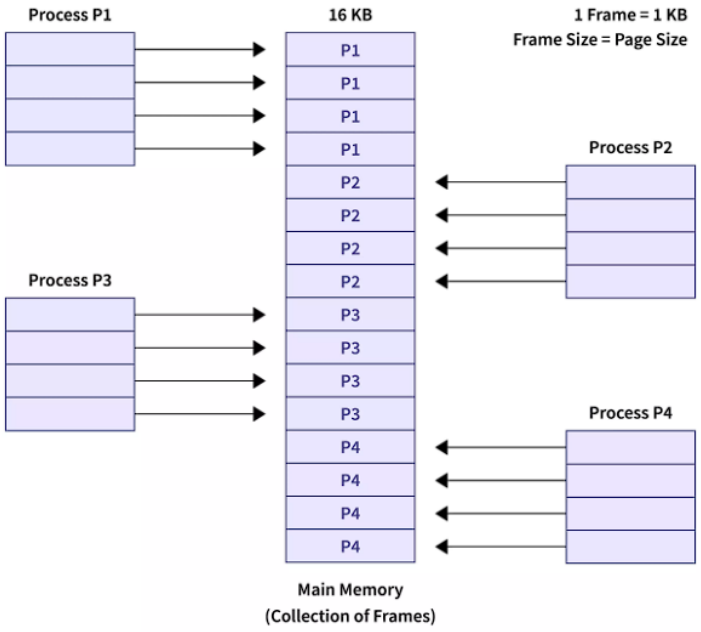
While, **pages** are sliced up logical memory blocks of equal size. While solving any problem always the page size should be equal to frame size.

Some Formulas –

- No. of Frames = Physical address space/ Frame size
- No. of pages = Logical address space/ Page size

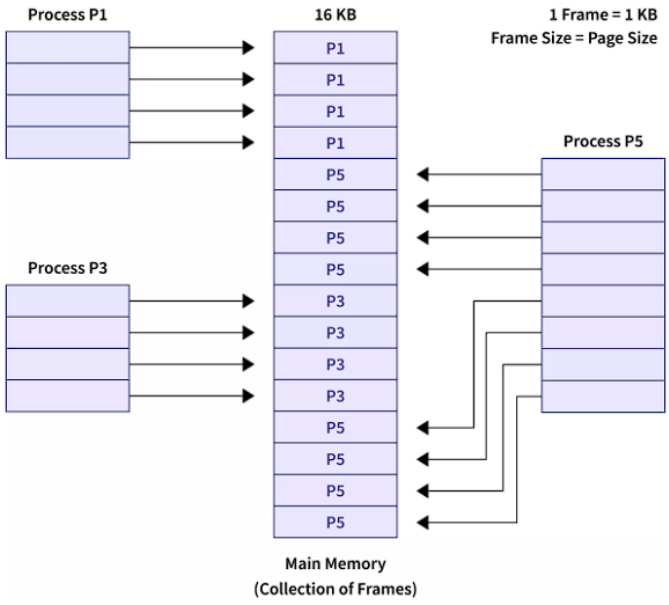
Example to understand Paging in OS -

CASE-1 (Contiguous Allocation of Pages)



As we can see in the above image, we have main memory divided into 16 frames of the size of 1KB each. Also, there are 4 processes available in the secondary (local) memory: P1, P2, P3, and P4 of a size of 4KB each. Clearly, each process needs to be further subdivided into pages of size of 1KB each, so that one page can be easily mapped to one frame of the main memory. This divides each process into 4 pages and the total for 4 processes gives 16 pages of 1KB each. Initially, all the frames were empty and therefore, pages will be allocated here in a contiguous manner.

CASE-2 (Non-Contiguous Allocation of Pages)



Let us assume that in Case-1, processes P2 and P4 are moved to the waiting state after some time and leave behind the empty space of 8 frames. **In Case-2, we have another process P5 of size 8KB (8 pages) waiting inside the ready queue to be allocated.** We know that with Paging, we can store the pages at different locations of the memory, and here, we have8non-contiguous frames available. Therefore, we can easily load the 8 pages of the process P5 in the place of P2 and P4 which we can observe in the above image.

Terminologies Associated with Memory Control:

- **Logical Address or Virtual Address:** This is a deal that is generated through the CPU and used by a technique to get the right of entry to reminiscence. It is known as a logical or digital deal because it isn't always a physical vicinity in memory but an opportunity for a connection with a place inside the device's logical address location. **Logical Address = Page number + page offset**
- **Logical Address Space or Virtual Address Space:** This is the set of all logical addresses generated via a software program. It is normally represented in phrases or bytes and is split into regular-duration pages in a paging scheme.
- **Physical Address:** This is a cope that corresponds to a bodily place in reminiscence. It is the actual cope with this that is available on the memory unit and is used by the memory controller to get admission to the reminiscence. **Physical Address = Frame number + page offset**
- **Physical Address Space:** This is the set of all bodily addresses that correspond to the logical addresses inside the way's logical deal with place. It is usually represented in words or bytes and is cut up into fixed-size frames in a paging scheme.

Important Features of Paging in PC Reminiscence Management:

- **Mapping from logical to physical addresses:** In the context of paging, the logical address space of a system is divided into fixed-sized pages, with each page associated with a specific physical frame in the main memory. This approach grants the operating system greater flexibility in memory management as it can allocate and release frames as needed.
- **Uniform page and frame dimensions:** Paging adopts a constant page size, typically matching the size of a frame in the main memory. This simplifies memory management and enhances system efficiency.
- **Entries within the page table:** Each page within a process's logical address space is represented by a page table entry (PTE) containing details about the corresponding physical frame in the main memory. This information includes the frame number and various control bits used by the operating system for memory management.

- **Matching page table entries:** The quantity of page table entries in a process's page table is equivalent to the number of pages within the logical address space of the process.
- **Page table stored in primary memory:** Typically, the page table for each process is stored in the main memory to facilitate efficient access and manipulation by the operating system. However, this approach can introduce some overhead, as the page table must be updated whenever a process is swapped in or out of the main memory.

Memory Management Unit (Basic Method):

The Memory Management Unit (MMU) is **responsible for converting logical addresses to physical addresses.** The physical address refers to the actual address of a frame in which each page will be stored, whereas the logical address refers to the address that is generated by the CPU for each page.

When the CPU accesses a page using its logical address, the OS must first collect the physical address in order to access that page physically. There are two elements to the logical address:

- Page number
- Offset

The OS's memory management unit must convert the page numbers to the frame numbers.

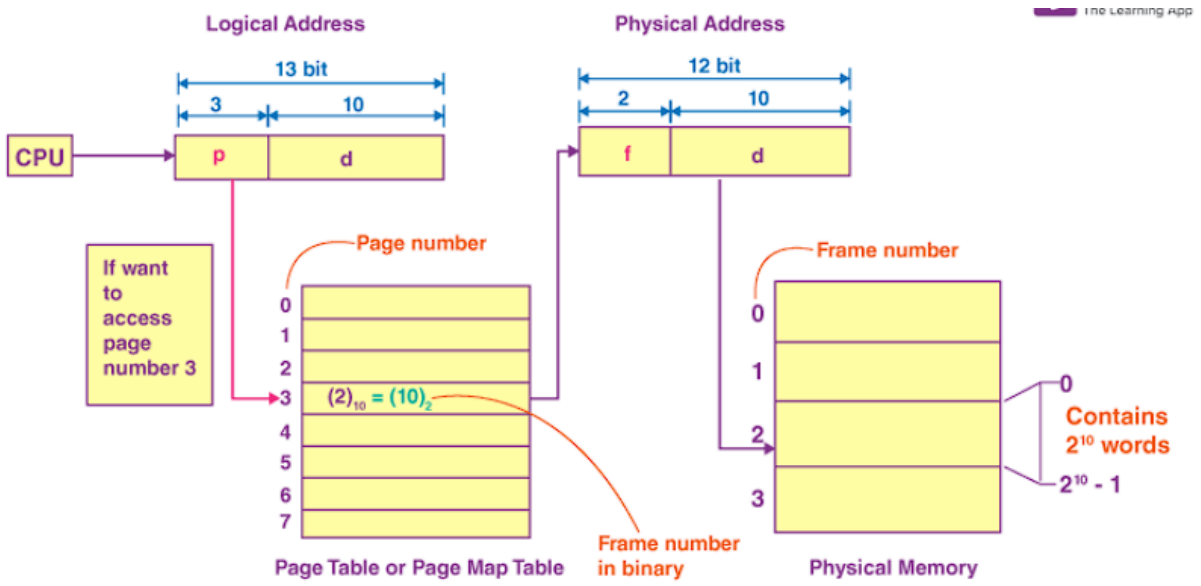
Examples:

Let's say the CPU requests the 10th word of the 4th page of process P3 in the image above. Because page number 4 of process P1 is stored at frame number 9, the physical address will be returned as the 10th word of the 9th frame.

Let's consider another example:

- If the physical address is 12 bits, then the physical address space would be 4 K words
- If the logical address is 13 bits, then the logical address space would be 8 K words
- If the page size is equal to the frame size, which is equal to 1 K words (assumption),

Then:



The address generated by the CPU is divided into the following:

- **Page offset(d):** It refers to the number of bits necessary to represent a certain word on a page, page size in Logical Address Space, or page word number or page offset.
- **Page number(p):** It is the number of bits needed to represent the pages in the Logical Address Space or the page number. Page number sometimes is also called as **VPN (Virtual Page Number).**

The Physical Address is divided into the following:

- **Frame offset(d):** It refers to the number of bits necessary to represent a certain word in a frame, or the Physical Address Space frame size, the word number of a frame, or the frame offset.
- **Frame number(f):** It's the number of bits needed to indicate a frame of the Physical Address Space or a frame number. Frame number is also sometimes referred as **PFN (Physical Frame Number).**

Hardware Support for Paging:

Dedicated registers can be used to implement the page table in hardware. However, using a register for the page table is only useful if the page table is tiny. We can employ **TLB (translation look-aside buffer),** a particular, tiny, fast look-up hardware cache if the page table has a significant number of entries.

- The TLB is a high-speed, associative memory.
- TLB entries are made up of two parts: a value and a tag.
- When this memory is accessed, an item is compared to all tags at the same time.
- If the object is located, the value associated with it is returned.

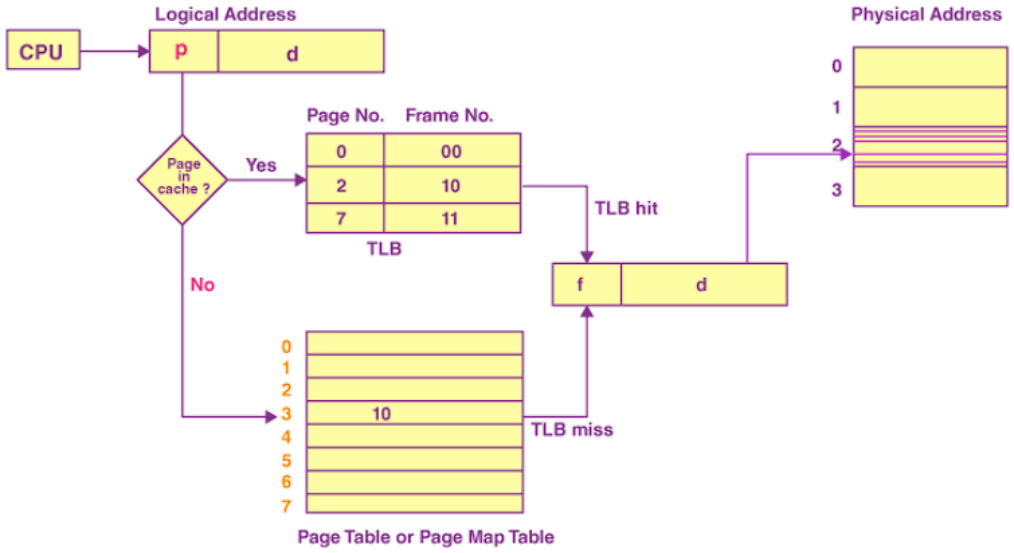
Working:

- When a logical-address is generated by the CPU, its page-number is presented to the TLB.
- If the page-number is found (TLB hit), its frame-number is

  → immediately available and

  → used to access memory.

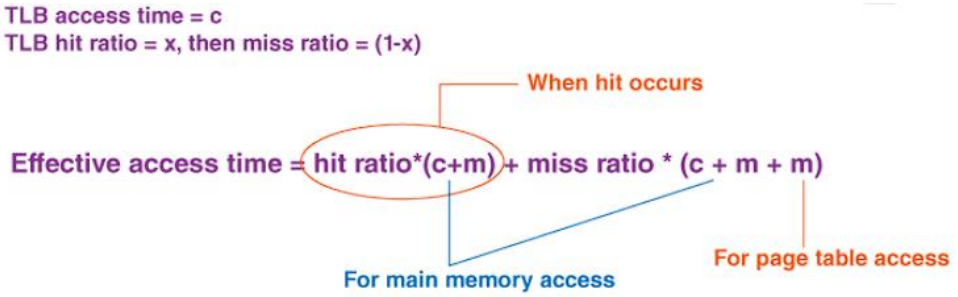
- If page-number is not in TLB (TLB miss), a memory-reference to page table must be made.
- The obtained frame-number can be used to access memory (Below figure).
- In addition, we add the page-number and frame-number to the TLB, so that they will be found quickly on the next reference.
- If the TLB is already full of entries, the OS must select one for replacement.
- Percentage of times that a particular page-number is found in the TLB is called **hit ratio.**



m = main memory access time

In case the page table is kept in the main memory,

then the effective access time would be = m(page table) + m(page in page table)



Paging Protection:

- Memory-protection is achieved by protection-bits for each frame.
- The protection-bits are kept in the page-table.
- One protection-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.
- Firstly, the physical-address is computed. At the same time, the protection-bit is 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 OS (or memory-protection violation).

Valid Invalid Bit:

- This bit is attached to each entry in the page-table (Figure 3.20).
  1. **Valid bit:** The page is in the process' logical-address space.
  2. **Invalid bit:** The page is not in the process' logical-address space.
- Illegal addresses are trapped by use of valid-invalid bit.
- The OS sets this bit for each page to allow or disallow access to the page.

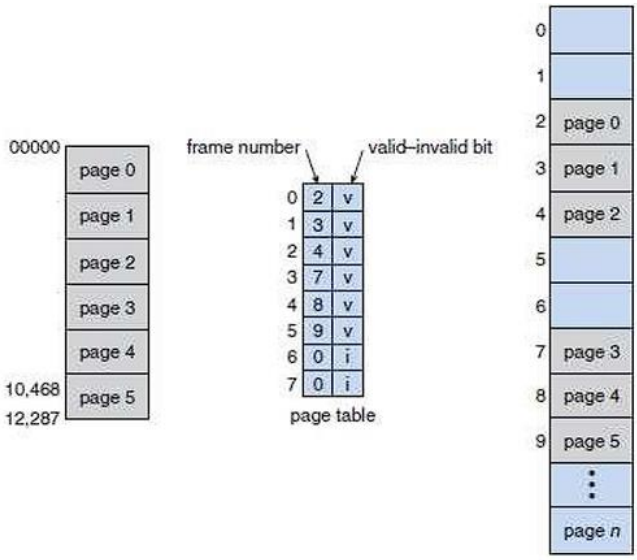


Figure 3.20 Valid (v) or invalid (i) bit in a page-table

What is a Page Fault?

The term "page miss" or "page fault" refers to the concept of a miss that occurs if the referred page is not existent in the main memory.

The missing page must be accessed by the CPU from secondary memory. The system's effective access time will increase if the number of page faults is quite high.

Shared Pages:

- An advantage of paging is the possibility of sharing common code.

• Re-entrant code is non-self-modifying code; it never changes during execution.

• Two or more processes can execute the same code at the same time.

• Each process has its own copy of registers and data-storage to hold the data for the process's execution.

• The data for 2 different processes will be different.

• Only one copy of the editor need be kept in physical-memory (Figure 3.21).

• Each user's page-table maps onto the same physical copy of the editor, but data pages are mapped onto different frames.

• Disadvantage: Systems that use inverted page-tables have difficulty implementing shared-memory.

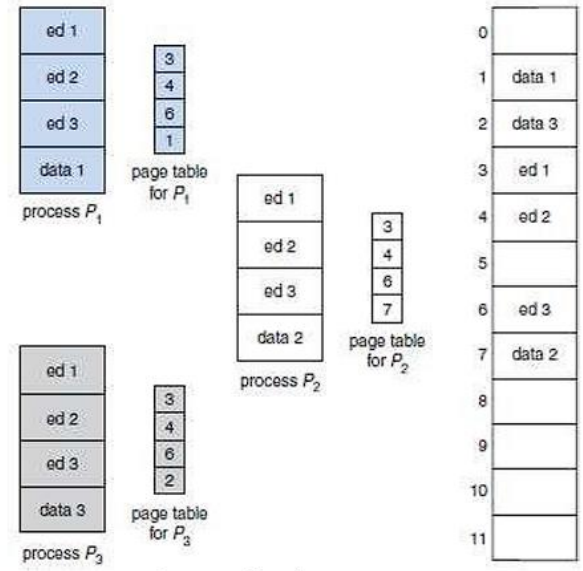
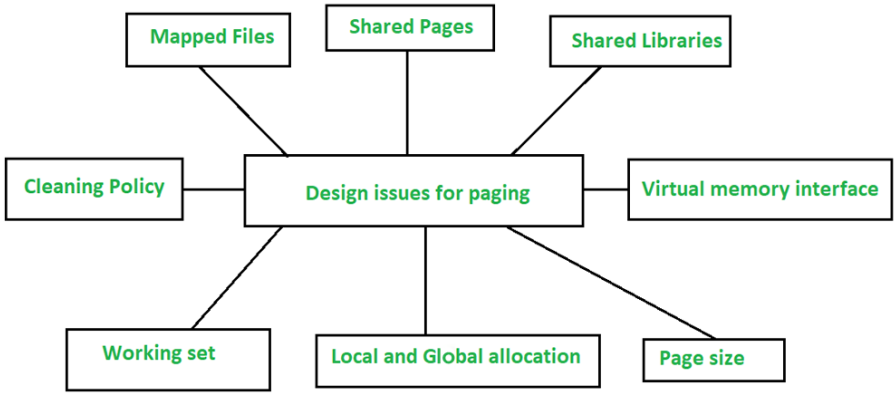


Figure 3.21 Sharing of code in a paging environment

Design Issues for Paging:



1. **Working Set:** The working set represents the set of pages a process is actively using. Determining an appropriate working set size is crucial for optimizing page replacement policies. If the working set is too small, it may lead to frequent page faults, while a too-large working set can result in unnecessary page-in operations.
2. **Local and Global Allocation:** Local allocation refers to allocating frames for a process's pages from its own set of frames, while global allocation allows frames to be allocated from the entire pool of free frames. The choice between local and global allocation affects process isolation and can impact system performance. Global allocation may lead to better utilization but requires careful management to avoid interference between processes.
3. **Page Size:** The size of pages directly affects internal fragmentation and the efficiency of memory usage. Choosing an appropriate page size involves balancing factors like reducing internal fragmentation, minimizing page table size, and optimizing I/O performance during page transfers.
4. **Shared Pages:** Efficiently managing shared pages among processes is essential. Shared pages, such as code segments, should be identified and shared to avoid unnecessary duplication of memory. This involves implementing mechanisms for detecting and handling shared pages while maintaining data consistency.
5. **Shared Libraries:** Shared libraries present a specific case of shared pages. Designing mechanisms for sharing code segments among multiple processes using the same library can significantly reduce memory consumption. This requires coordination to ensure that updates or modifications to shared libraries do not impact the stability of processes using them.
6. **Mapped Files:** Mapped files involve mapping portions of files directly into the virtual address space of a process. Efficiently managing mapped files requires coordination between the file system and the paging system. Decisions related to caching, synchronization, and updates to mapped files impact system performance and data consistency.

7. **Cleaning Policy:** The cleaning policy determines when and how dirty pages (modified pages that need to be written back to disk) are cleaned. Policies such as demand cleaning or background cleaning affect I/O performance and system responsiveness. Effective cleaning policies are crucial for maintaining a balance between minimizing page-out operations and ensuring timely updates to disk.
8. **Virtual Memory Interface:** The virtual memory interface defines the system calls and interactions between the operating system and user processes regarding virtual memory management. Designing a clear and efficient interface is essential for providing flexibility to applications while allowing the operating system to manage virtual memory effectively. This includes aspects such as memory allocation, deallocation, and protection mechanisms.

Structure of the Page Table:

1) Hierarchical Paging

2) Hashed Page-tables

3) Clustered Page Tables

4) Inverted Page-tables

1. Hierarchical Paging:

Problem: Most computers support a large logical-address space (232 to 264). In these systems, the page-table itself becomes excessively large.

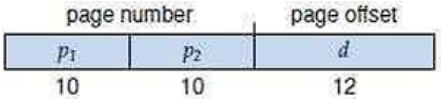
Solution: Divide the page-table into smaller pieces.

Two Level Paging Algorithm:

• The page-table itself is also paged (Figure 3.22).

• This is also known as a **forward-mapped page-table** because address translation works from the outer page-table inwards.

• For example (Figure 3.23): Consider the system with a 32-bit logical-address space and a page-size of 4 KB. A logical-address is divided into 20-bit page-number and 12-bit page-offset. Since the page-table is paged, the page-number is further divided into 10-bit page-number and 10-bit page-offset. Thus, a logical-address is as follows:



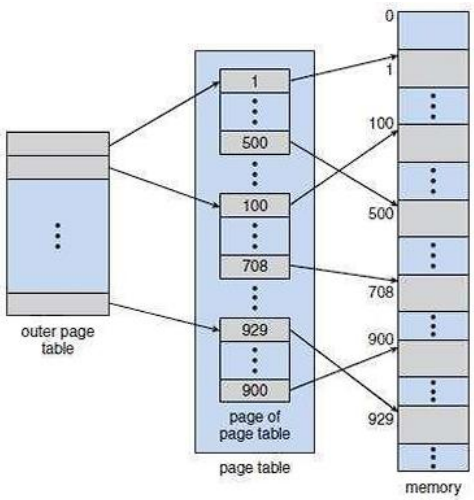


Figure 3.22 A two-level page-table scheme

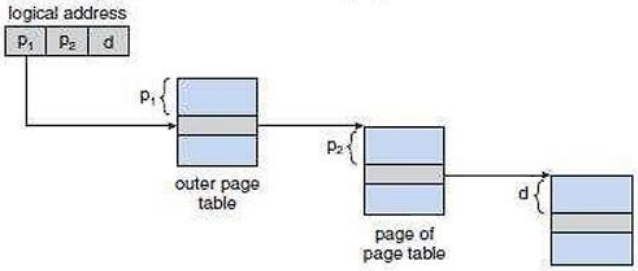


Figure 3.23 Address translation for a two-level 32-bit paging architecture

2. Hashed Page Tables:

• This approach is used for handling address spaces larger than 32 bits.

• The hash-value is 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 3 fields:

1) Virtual page-number

2) Value of the mapped page-frame and

3) Pointer to the next element in the linked-list.

• The algorithm works as follows (Figure 3.24):

1) The virtual page-number is hashed into the hash-table.

2) The virtual page-number is compared with the first element in the linked-list.

3) If there is a match, the corresponding page-frame (field 2) is used to form the desired physical-address.

4) If there is no match, subsequent entries in the linked-list are searched for a matching virtual page-number.

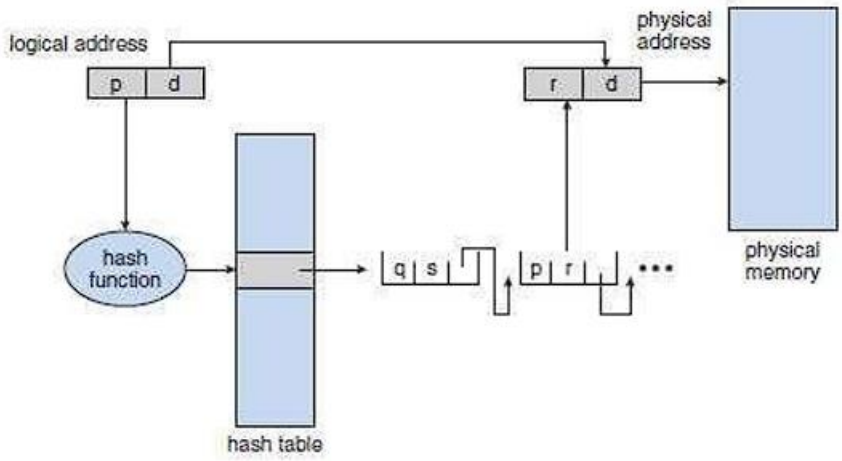


Figure 3.24 Hashed page-table

3. Clustered Page Tables:

• These are similar to hashed page-tables except that each entry in the hash-table refers to several pages rather than a single page.

• **Advantages:**

1) Favorable for 64-bit address spaces.

2) Useful for address spaces, where memory-references are noncontiguous and scattered throughout the address space.

4. Inverted Page Tables:

• Has one entry for each real page of memory.

• Each entry consists of

→ virtual-address of the page stored in that real memory-location and

→ information about the process that owns the page.

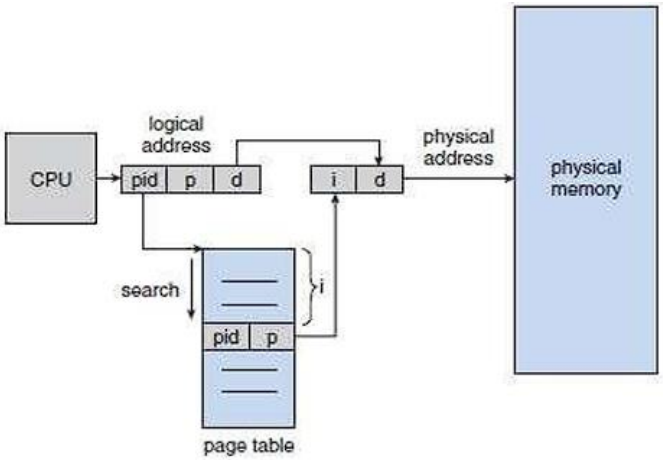


Figure 3.25 Inverted page-table

• Each virtual-address consists of a triplet (Figure 3.25).

• Each inverted page-table entry is a pair

• The algorithm works as follows:

1) When a memory-reference occurs, part of the virtual-address, consisting of , is presented to the memory subsystem.

2) The inverted page-table is then searched for a match.

3) If a match is found, at entry i-then the physical-address is generated.

4) If no match is found, then an illegal address access has been attempted.

• Advantage: Decreases memory needed to store each page-table

• Disadvantages:

1) Increases amount of time needed to search table when a page reference occurs.

2) Difficulty implementing shared-memory.

Advantages of Paging:

- It is a memory management technique as we can store the pages of a single process in a non-contiguous manner as well which saves the memory.
- The problem of external fragmentation is solved with the help of the Paging technique.
- Allocating the pages within equal and fixed-size frames is easy and simple. Swapping is also easy between the pages and the page frames.

Disadvantages of Paging:

- Internal Fragmentation may occur especially during the allocation of the last page of the process.
- Page tables that are separate for each process in the secondary memory may consume extra memory.
- If we are not using TLB (Translation of look-aside buffer), then the time taken to fetch an instruction and the element byte is high because we need to access the memory two times (for page entry and for actual element within the frame).
- Address translation of logical to physical address lengthens the memory cycle times. Also, it needs specialized hardware.
- Though memory access time can be improved by using TLB, but again there is a limited size of page entries within the TLB up to 1024.

What is Thrashing?

The effective access time will be as long as it takes the CPU to read one word from secondary memory if there are as many page faults as referred pages or if there are so many page faults that the CPU is constantly reading pages from secondary memory. Thrashing is the name given to the idea.

When the memory access time is ma, the page fault rate is PF%, and it takes S (service time) to retrieve a page from secondary memory before resuming, the effective access time can be calculated as follows:

EAT = PF * S + (1 - PF) * (ma)

Numerical Example:

Consider a machine with 64 MB physical memory and a 32-bit virtual address space. If the page size is 4KB, what is the approximate size of the page table?

Solution:

Physical Address Space = 64MB = 2^{26}B
Virtual Address = 32-bits,

∴ Virtual Address Space = 2^{32}B
Page Size = 4KB = 2^{12}B
Number of pages = $2^{32}/2^{12} = 2^{20}$ pages.

Number of frames = $2^{26}/2^{12} = 2^{14}$ frames.

∴ Page Table Size = 2^{20}×14-bits ≈ 2^{20}×16-bits ≈ 2^{20}×2B = 2MB.

Segmentation:

Segmentation is a memory management technique in operating systems. It divides memory into variable-sized parts called **segments**. Each segment can be allocated to a process. The details about each segment are stored in a table called a **segment table**. It is a non-contiguous memory allocation technique. It allows users to partition their programs into modules that operate independently. Segmentation can lead to memory fragmentation. This occurs when available memory becomes inefficiently divided. Segmentation also requires complex hardware and software support.

Types of Segmentation:

Segmentation can be divided into two types:

1. **Virtual Memory Segmentation**: Virtual Memory Segmentation divides the processes into **n** number of segments. All the segments are not divided at a time. Virtual Memory Segmentation may or may not take place at the run time of a program.
2. **Simple Segmentation**: Simple Segmentation also divides the processes into **n** number of segments but the segmentation is done all together at once. Simple segmentation takes place at the run time of a program. Simple segmentation may scatter the segments into the memory such that one segment of the process can be at a different location than the other (in a noncontinuous manner).

Why Segmentation is required?

Segmentation came into existence because of the problems in the paging technique. In the case of the paging technique, a function or piece of code is divided into pages without considering that the relative parts of code can also get divided. Hence, for the process in execution, the CPU must load more than one page into the frames so that the complete related code is there for execution. Paging took more pages for a process to be loaded into the main memory. Hence, segmentation was introduced in which the code is divided into modules so that related code can be combined in one single block.

Other memory management techniques have also an important drawback - the actual view of physical memory is separated from the user's view of physical memory. Segmentation helps in overcoming the problem by dividing the user's program into segments according to the specific need.

Characteristics of Segmentation in OS:

Some of the characteristics of segmentation are discussed below:

- Segmentation partitions the program into variable-sized blocks or segments.
- Partition size depends upon the type and length of modules.
- Segmentation is done considering that the relative data should come in a single segment.
- Segments of the memory may or may not be stored in a continuous manner depending upon the segmentation technique chosen.
- Operating System maintains a segment table for each process.

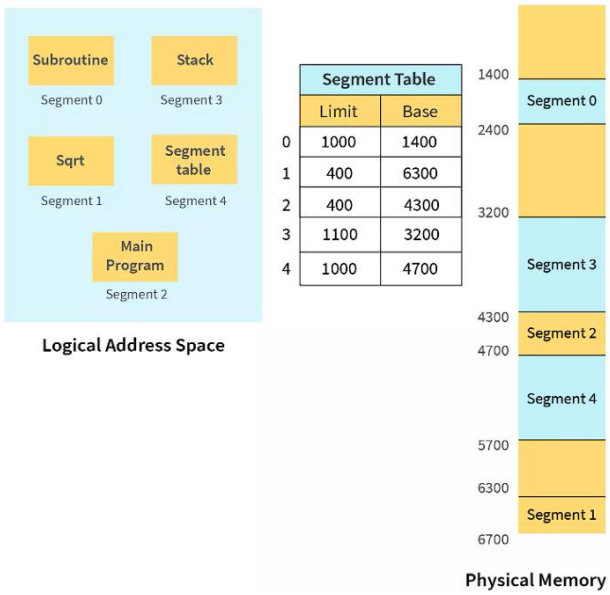
Example of Segmentation:

Let's take the example of segmentation to understand how it works.

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

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

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



What is Segment Table?

In order to know which segment is present at which address in the physical memory, every process has its segment table. The segment table is one of the most important parts of the segmentation. It helps in mapping the logical address generated by the CPU to the physical address of the memory.

segment number	Base	Limit
0	6000	600
1	1500	300
2	400	500
3	2837	100

Segment table

The segment table consists of:

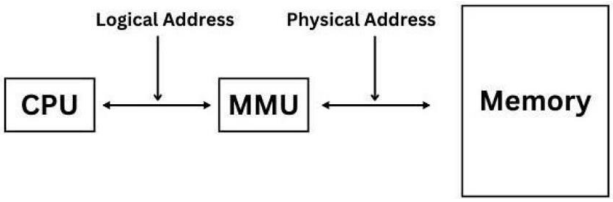
1. Segment Base- The base represents the starting address of the segment in the physical memory.

2. Segment Limit- Limit determines the size of the segment. Suppose the limit of a segment is 600 bytes, and the base address of the segment in the memory is 4000. We have assumed the memory to be byte-addressable. Therefore, the last byte of the segment will be at the 4599th byte in the physical memory.

Translation of Logical address into physical address by segment table:

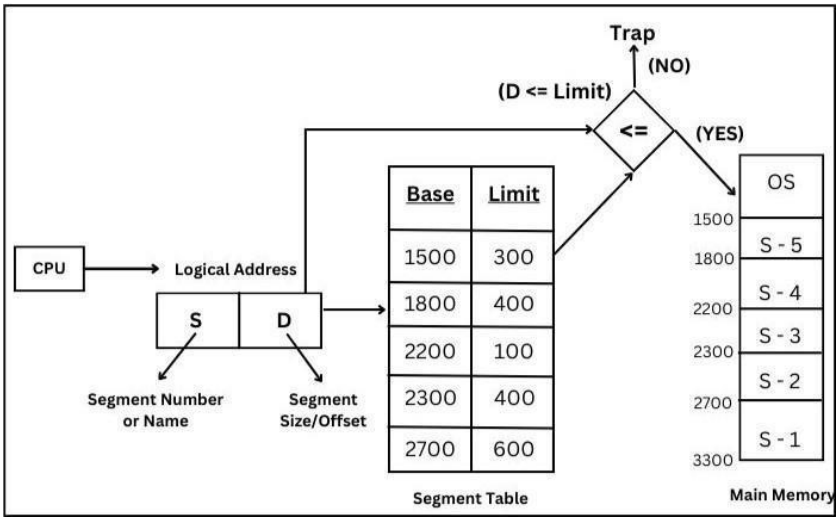
CPU (Central Processor Unit) produces the logical address, but this is just a virtual address that is not present in our memory, so we need to translate the logical to a physical address. Translation of logical (or virtual) address into physical address is done by a unit called MMU (Memory Management Unit) which checks the range and permissions, and this process is programmed by OS.

Here is the diagram below to understand the process:



In the above diagram, The CPU produces the logical address, which then converts to a physical address using MMU (Memory Management Unit). Now this physical address can be used in memory for the assignment of processes.

Here is the diagram of the architecture of the segmentation that shows how logical address is used:



CPU produces the logical address, which has 2 parameters that are segment number, shows the specific segment, and the segment offset shows the size of that segment. Then OS searches this segment number in the segment table, and MMU does computation to check the offset, where offset should be lesser than or equal to the limit of that segment because only then that segment can be accessed. If the condition gets true, the segment can be accessed. Here the physical address will be (base + D).

For example, if we search for base 1500, which is a segment (S - 5), and D is 200. Then MMU does a computation where D should be lesser than or equal to the limit; here, D is 200, which is lesser than the limit, which is 300. So this segment can be accessed, and the physical address will be (base + D), which is 1500 + 200 = 1700. So the accessing will start from the memory index 1500 to 1700.

Advantages of Segmentation:

- No internal fragmentation is there in segmentation.
- Segment Table is used to store the records of the segments. The segment table itself consumes small memory as compared to a page table in paging.
- Segmentation provides better CPU utilization as an entire module is loaded at once.
- Segmentation is near to the user's view of physical memory. Segmentation allows users to partition the user programs into modules. These modules are nothing but the independent codes of the current process.
- The Segment size is specified by the user but in Paging, the hardware decides the page size.
- Segmentation can be used to separate the security procedures and data.

Disadvantages of Segmentation:

- During the swapping of processes, the free memory space is broken into small pieces, which is a major problem in the segmentation technique.
- Time is required to fetch instructions or segments.
- The swapping of segments of unequal sizes is not easy.
- There is an overhead of maintaining a segment table for each process as well.
- When a process is completed, it is removed from the main memory. After the execution of the current process, the unevenly sized segments of the process are removed from the main memory. Since the segments are of uneven length it creates unevenly sized holes in the main memory. These holes in the main memory may remain unused due to their very small size.

Differences between Paging and Segmentation:

Paging	Segmentation
Paging is a memory management technique where memory is partitioned into fixed-sized blocks that are commonly known as **pages.**	Segmentation is also a memory management technique where memory is partitioned into variable-sized blocks that are commonly known as **segments**.
With the help of Paging, the logical address is divided into a **page number** and **page offset**.	With the help of Segmentation, the logical address is divided into **section number** and **section offset**.
This technique may lead to **Internal Fragmentation**.	Segmentation may lead to **External Fragmentation**.
In Paging, the page size is decided by the hardware.	While in Segmentation, the size of the segment is decided by the user.
In order to maintain the page data, the page table is created in the Paging	In order to maintain the segment data, the segment table is created in the Paging
The page table mainly contains the base address of each page.	The segment table mainly contains the segment number and the offset.
This technique is faster than segmentation.	On the other hand, segmentation is slower than paging.
In Paging, a list of free frames is maintained by the Operating system.	In Segmentation, a list of holes is maintained by the Operating system.
In this technique, in order to calculate the absolute address page number and the offset both are required.	In this technique, in order to calculate the absolute address segment number and the offset both are required.

Segmentation with Paging:

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.

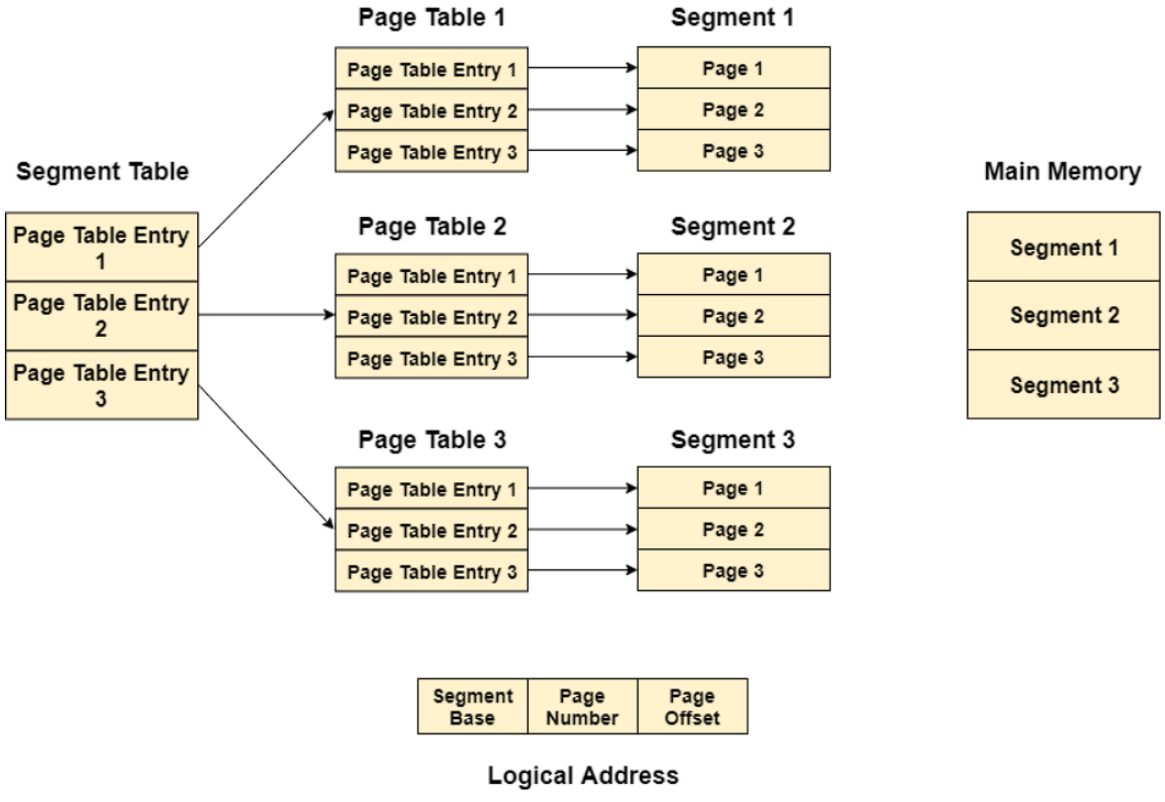
1. Pages are smaller than segments.
2. Each Segment has a page table which means every program has multiple page tables.
3. The logical address is represented as Segment Number (base address), Page number and page offset.

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

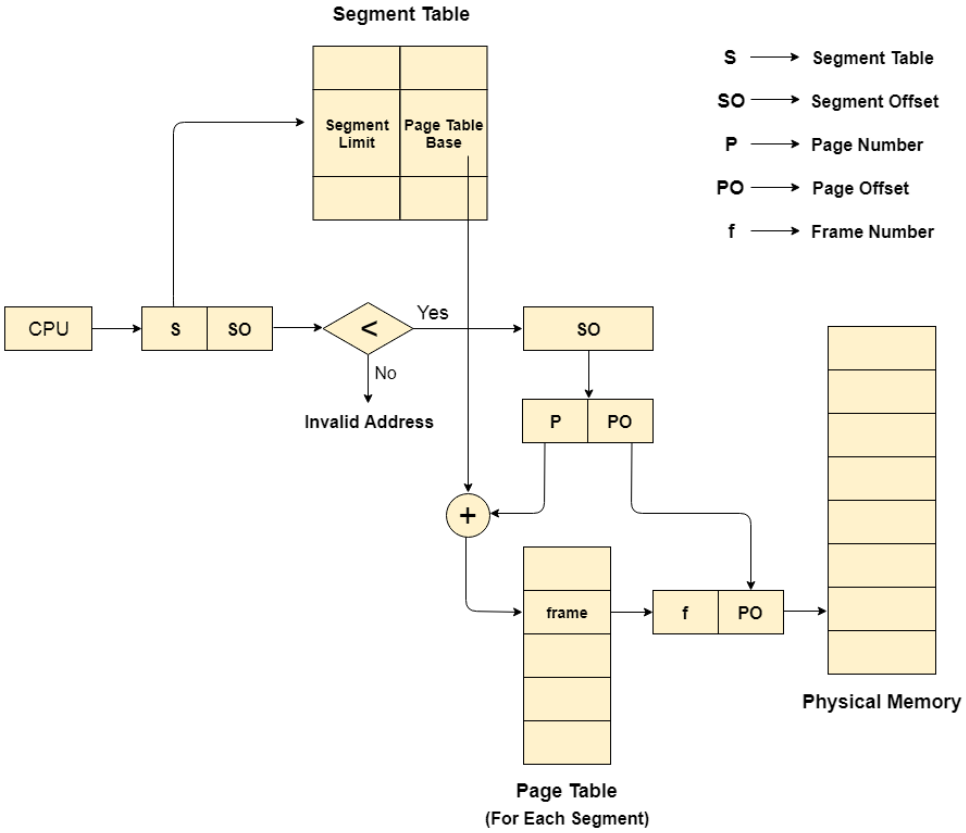
Each Page table contains the various information about every page of the segment. The Segment Table contains the information about every segment. Each segment table entry points to a page table entry and every page table entry is mapped to one of the page within a segment.



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:

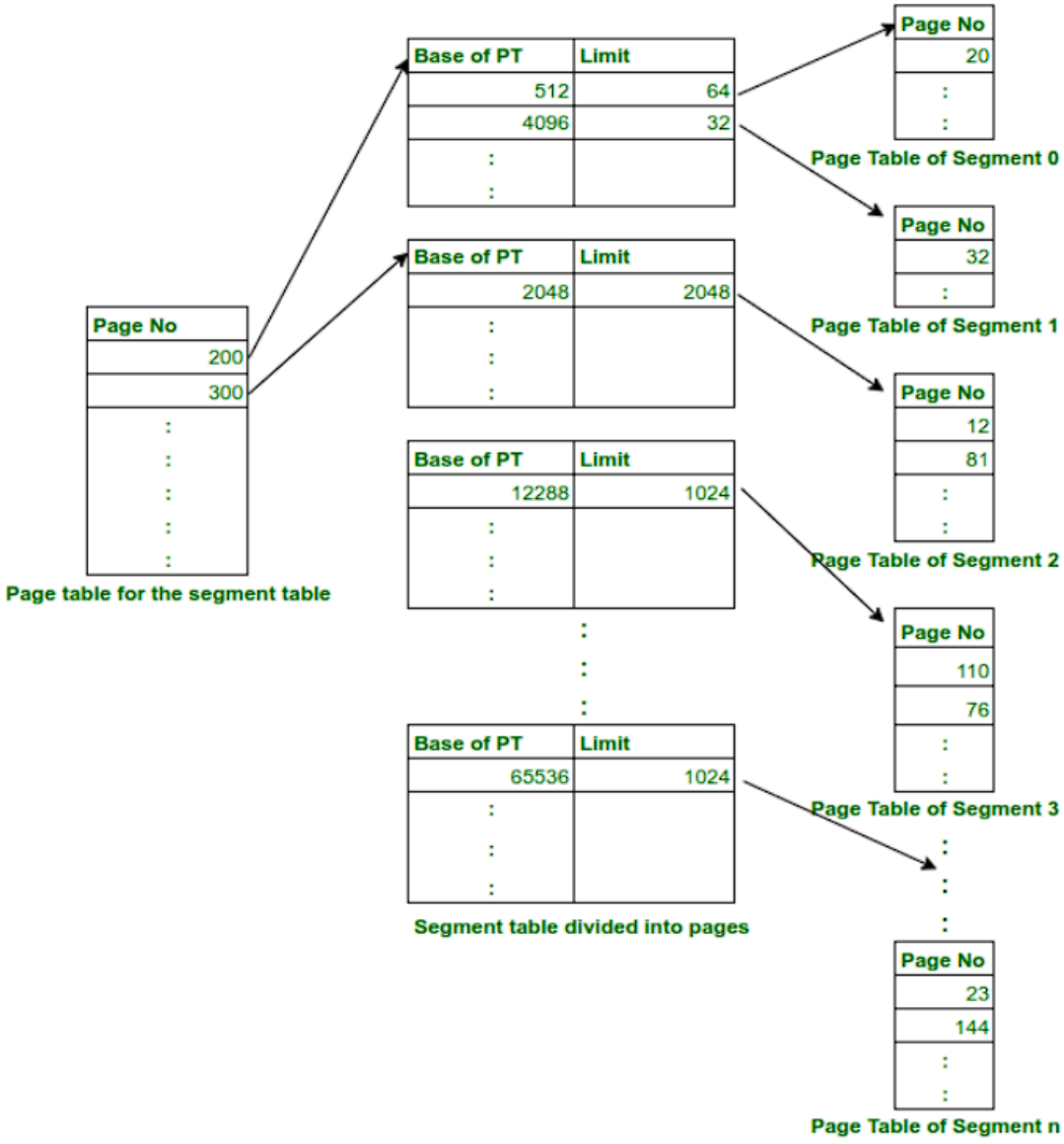
1. It reduces memory usage.
2. Page table size is limited by the segment size.
3. Segment table has only one entry corresponding to one actual segment.
4. External Fragmentation is not there.
5. It simplifies memory allocation.

Disadvantages of Segmented Paging:

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

Paged Segmentation:

1. 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.
2. 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.



Paged Segmentation

Advantages of Paged Segmentation:

- It offers protection in specific segments
- It uses less memory than paging
- There is no external fragmentation
- It optimizes resource allocation
- Easiness in adding or removing segments and pages
- It also facilitates multi-programming
- It supports large programs

Disadvantages of Paged Segmentation:

- It is costly
- In the case of swapping, segments with different sizes are not good
- Programmer intervention is required
- introduce additional overhead in terms of memory and processing
- adds complexity to memory management
- fragmentation can still occur over time
- The learning curve is steep

Virtual Memory:

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

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

Why Need Virtual Memory?

Here, are reasons for using virtual memory:

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

How Virtual Memory Works?

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

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

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

For example:

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

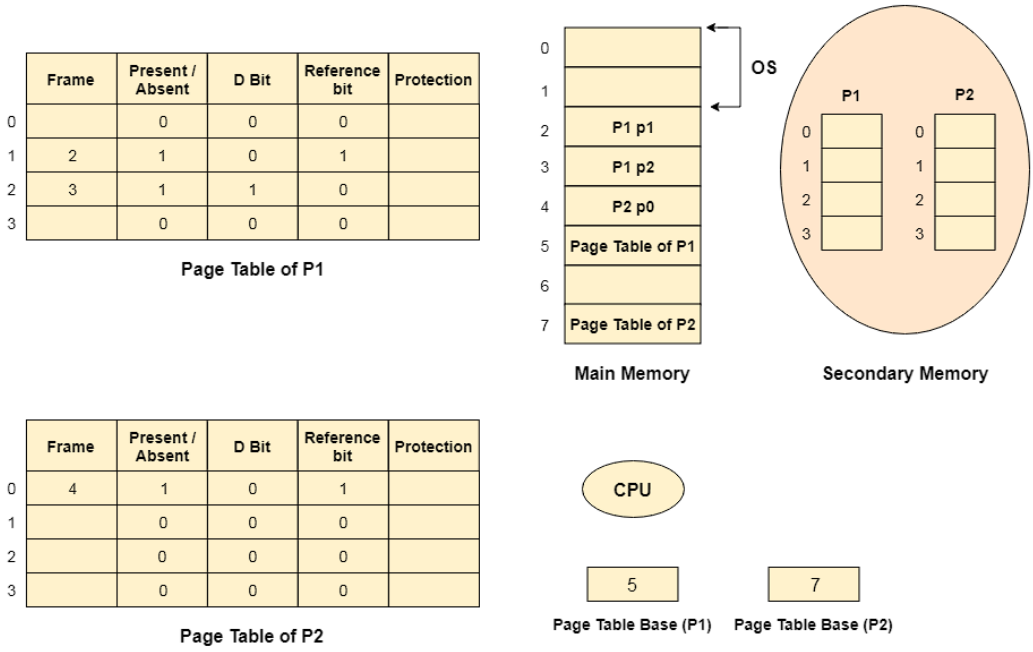
- The OS will then set up 250 MB of virtual memory and use a program called the Virtual Memory Manager (VMM) to manage that 250 MB.
- So, in this case, the VMM will create a file on the hard disk that is 250 MB in size to store extra memory that is required.
- The OS will now proceed to address memory as it considers 300 MB of real memory stored in the RAM, even if only 50 MB space is available.
- It is the job of the VMM to manage 300 MB memory even if just 50 MB of real memory space is available.

Snapshot of a virtual memory management system:

Let us assume 2 processes, P1 and P2, contains 4 pages each. Each page size is 1 KB. The main memory contains 8 frame of 1 KB each. The OS resides in the first two partitions. In the third partition, 1st page of P1 is stored and the other frames are also shown as filled with the different pages of processes in the main memory.

The page tables of both the pages are 1 KB size each and therefore they can be fit in one frame each. The page tables of both the processes contain various information that is also shown in the image.

The CPU contains a register which contains the base address of page table that is 5 in the case of P1 and 7 in the case of P2. This page table base address will be added to the page number of the Logical address when it comes to accessing the actual corresponding entry.



Types of virtual memory:

The two ways computers handle virtual memory are through paging and segmentation.

Paging: It is a technique of memory management that breaks the process address space into various blocks of similar sizes, known as pages. Here, we measure the size of a process in the total number of pages.

Segmentation: In this technique every job gets divided into various blocks of varied sizes, known as segments. This way we get one segment for every module with pieces performing related functions. These segments act as different spaces of the logical address of any program.

Virtual memory is commonly implemented using demand paging.

Advantages of Virtual Memory:

1. The degree of Multiprogramming will be increased.
2. User can run large application with less real RAM.
3. There is no need to buy more memory RAMs.

Disadvantages of Virtual Memory:

1. The system becomes slower since swapping takes time.
2. It takes more time in switching between applications.
3. The user will have the lesser hard disk space for its use.

Demand Paging:

Demand paging is a memory management technique used by operating systems to optimize the use of memory resources. **In demand paging, we only load the required pages of a process into the main memory instead of loading the entire process.**

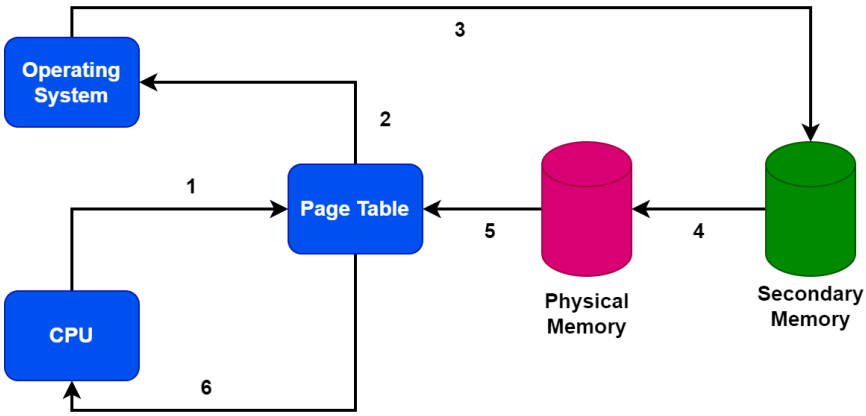
When we first load a process into memory, only the pages that are necessary for the initial execution of the program are loaded. As the program runs, we bring additional pages into memory on demand as needed. Hence, this allows the operating system to optimize memory usage. Additionally, it doesn't have to load all the pages of a program into memory at once.

Demand paging allows the system to swap out pages that are not currently in use, freeing up memory for other processes. When a page that has been swapped out is needed again, the system can bring it back into memory. Therefore, the main motivation behind demand paging is to reduce the time taken for process initialization. Additionally, it also helps to reduce the memory requirements for a process.

The demand paging technique is commonly used in modern operating systems, including Windows, Linux, and macOS.

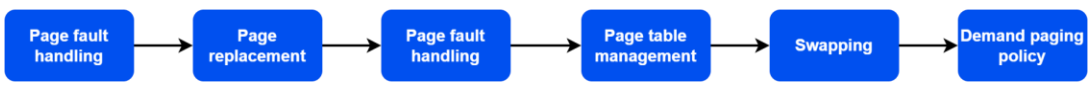
Working Procedure:

Let's discuss how the demand paging technique works:



Let's say the CPU wants to access a page for a specific process. **The first step is to look for the required page in the page table.** Suppose we find the required page in the page table. Therefore, the CPU access the page and forwards it to the process. However, if we can't find the target page, the page table generates a trap or page fault signal. Furthermore, the page table sends the signal to the operating system. This's where we apply the demand paging technique.

The demand paging technique involves six steps:



The first step is to handle the page faults. **A page fault is generated when a process attempts to access a memory address or a page that isn't currently available on the main memory.** Hence, the operating system must handle this page fault and load the required page from secondary storage into memory.

Additionally, if the system doesn't have enough free memory to load the requested page, it must select a page to evict from memory to make room for the requested page. Furthermore, the system uses a page replacement algorithm in order to select the page to evict.

The next step is to manage the page table. Hence, the operating system must maintain a page table for each process, which maps virtual pages to physical pages. Additionally, the operating system must update the page table whenever a page is loaded or evicted from memory. Furthermore, we must allocate memory for each process and manage the allocation of physical memory to virtual pages.

When the operating system evicts a page from memory, it's typically swapped to disk to make room for other pages. Additionally, the operating system must manage the swapping of pages between memory and disk. Finally, The OS needs to determine when to load pages into memory and when to evict them based on the current workload.

Common Terms in Demand Paging:

1.Page Fault: There will be a miss if the referenced page is not present in the main memory; this is known as a page miss or page fault.

The CPU must look up the missing page in secondary memory. When the number of page faults is significant, the system's effective access time increases dramatically.

2. Swapping: Swapping comprises either erasing all of the process's pages from memory or marking the pages so that we can remove them via the page replacement method.

When a process is suspended, it indicates it is unable to run. However, we can change the process for a while. The system can swap the process from secondary memory to primary memory over a period of time. Thrashing describes a condition in which a process is busy, and the pages are swapped in and out of it.

3. Thrashing: The effective access time will be the time needed by the CPU to read one word from the secondary memory if the number of page faults is equal to the number of referred pages or if the number of page faults is so high that the CPU is only reading pages from the secondary memory. This is known as Thrashing.

If the page fault rate is PF%, the time spent retrieving a page from secondary memory and resuming is S (service time), and the memory access time is "ma", the effective access time may be calculated as follows: EAT = PF x S + (1 - PF) x (ma)

Advantages of Demand Paging:

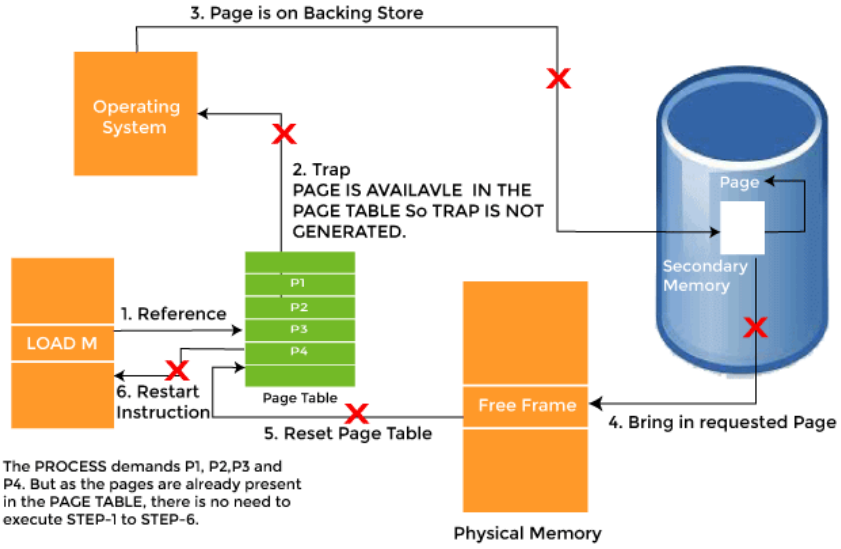
- Memory can be put to better use.
- If we use demand paging, then we can have a large virtual memory.
- By using demand paging, we can run programs that are larger than physical memory.
- In demand paging, there is no requirement for compaction.
- In demand paging, the sharing of pages is easy.
- Partition management is simple in demand paging because of the fixed partition size and the discontinuous loading.

Disadvantages of Demand Paging:

- Internal fragmentation is a possibility with demand paging.
- It takes longer to access memory (page table lookup).
- Memory requirements
- Guarded page tables
- Inverted page tables

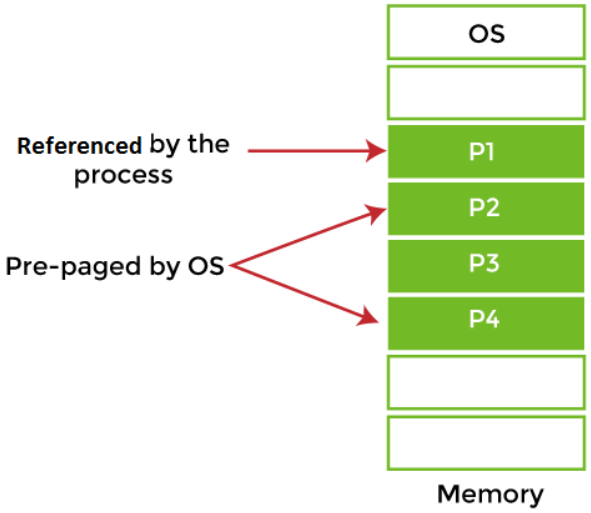
What is Pre-paging in OS?

Pre-paging is used to overcome a major drawback of demand paging. A major drawback of ***demand paging*** is many page faults, which may occur as soon as a process starts to execute. The situation results from an effort to load the initial locality into memory, and the same situation may arise repeatedly.



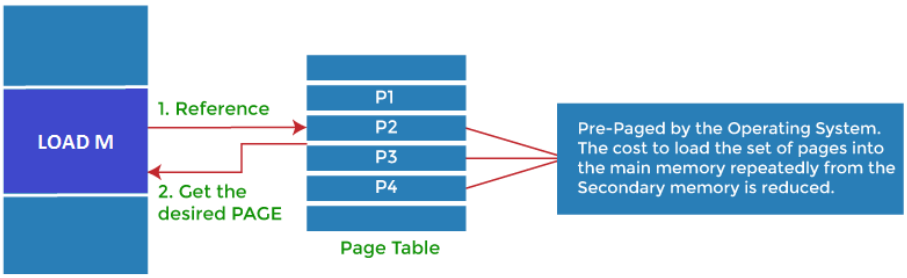
For example, when a process is restarted after being swapped out, all its pages are present on the disk, and hence each of the pages must be brought back to the main memory for execution of the process by its own page fault the worst case.

If a system uses a ***Working Set Model***, a list of pages is maintained with each process in its working set. If a process is suspended due to a lack of free frames or an I/O wait, the working set of the process is not lost. When a process is resumed, the entire working set is brought back into the memory before the process begins to execute again.



The above diagram shows that only one page was referenced or demanded by the CPU, but three more pages were ***pre-paged*** by the OS. The OS tries to predict which page would be next required by the processor and brings that page proactively into the main memory.

The major advantage of Pre-paging is that it might save time when a process references consecutive addresses. In this case, it is easy for the operating system to guess and load the appropriate pages, and, as there is a high probability of the guess being right for many pages, fewer page faults will occur.



Pre-paging may not always be beneficial. The advantage of Pre-paging is based on the answer to a simple question: whether the cost of implementing Pre-paging is less than the cost of servicing the corresponding page faults. It may be the case that a considerably large number of pages brought back into the memory by Pre-paging are not used. The disadvantage of the concept is that there is wastage of resources like time and memory if the pre-loaded pages are unused.

Advantages of Pre-paging:

In an operating system, pre-paging has the following advantages, such as:

- It saves time when large contiguous structures are used. Consider anexample where the process requests consecutive addresses. So, in such cases, the operating system can guess the next pages. And, if the guesses are right, fewer page faults will occur, and the effective memory access time will increase.

Disadvantages of Pre-paging:

Pre-paging also has the following disadvantages, such as:

- There is wastage of time and memory if those pre-paged pages are unused.

Difference between Demand Paging and Pre-paging:

Demand Paging	Pre-paging
Any page is not loaded into the main memory unless the process is referencing it at the present instant.	All the pages are loaded into the memory that will be needed simultaneously but before the process actually references them.
The number of page faults is significantly high.	The number of page faults may be reduced in certain specific cases.
The time taken to load the pages cannot decrease in any situation.	The time taken to load the pages decreases when a process references consecutive addresses.
The pages loaded in the main memory are certainly used.	The pages loaded in the main memory might or might not be used.
There is no wastage of resources as a page is loaded as and when needed.	There is wastage of resources as there is a high chance that the pages might be unused.

Pure Demand Paging:

In some cases when initially no pages are loaded into the memory, pages in such cases are only loaded when are demanded by the process by generating page faults. It is then referred to as **Pure Demand Paging**.

- In the case of pure demand paging, there is not even a single page that is loaded into the memory initially. Thus pure demand paging causes the page fault.

- When the execution of the process starts with no pages in the memory, then the operating system sets the instruction pointer to the first instruction of the process and that is on a non-memory resident page and then in this case the process immediately faults for the page.
- After that when this page is brought into the memory then the process continues its execution, page fault is necessary until every page that it needs is in the memory.
- And at this point, it can execute with no more faults.
- This scheme is referred to as Pure Demand Paging: means never bring a page into the memory until it is required.

Difference Between Paging and Swapping:

Paging	**Swapping**
Paging is a memory management technique in which the computer stores and retrieves data for usage in the main memory from secondary storage.	Swapping is a technique for temporarily removing inactive applications from the computer system's main memory.
More processes can be stored in the main memory using this strategy.	Swapping reduces the number of processes in the main memory.
Non-contiguous memory management is followed by paging.	Swapping is possible without the use of any memory management techniques.
Paging is more adaptable because it allows for the relocation of process pages.	Swapping is less flexible because the entire operation goes back and forth between the main memory and the back store.
Paging occurs when a portion of a process is written to disk.	When the entire operation is transferred to the disk, this is known as swapping.
For fewer workloads, the paging technique is used.	For vast workloads swapping technique is used.
This method allows a process memory address area to be non-contiguous.	Multiple processes can operate in parallel in the operating system with the help of Swapping.
This technique helps to implement virtual memory.	Swapping helps the CPU to access processes faster.

Page Replacement Policies/Algorithms:

A page replacement algorithm is a systems management tool that helps optimize and manage the web server. It helps eliminate the need to restart the web server after large updates or changes to the web pages. A page replacement algorithm usually erases old pages and creates new ones with new information.

Why is a page replacement algorithm needed?

Actual **RAM** is much smaller than virtual memory. So, it is not possible to keep all the data in RAM, so it is needed to load the pages when required (**demand paging**). If the page is found in RAM, then good otherwise, page faults will occur. Therefore, when a page fault occurs, the operating system must replace the existing page in RAM with the newly requested page. In this scenario, the page replacement algorithm helps the operating system determine which page to replace.

Belady's Anamoly in page replacement algorithm:

The percentage of page faults depends directly on the number of frames allocated to each process. Increasing the number of frames will significantly reduce the number of page faults. However, the opposite action can occur as the number of frames increases and the page faults increase. This exception is known as **Balady's anomaly**. **Optimal** and **LRU** are the two mainly used page replacement algorithms, but Balady's anomaly does not occur with these algorithms in the reference string because they belong to a class of stack-based page replacement algorithms.

Types of Page Replacement Algorithm:

1. First in, first out (FIFO):

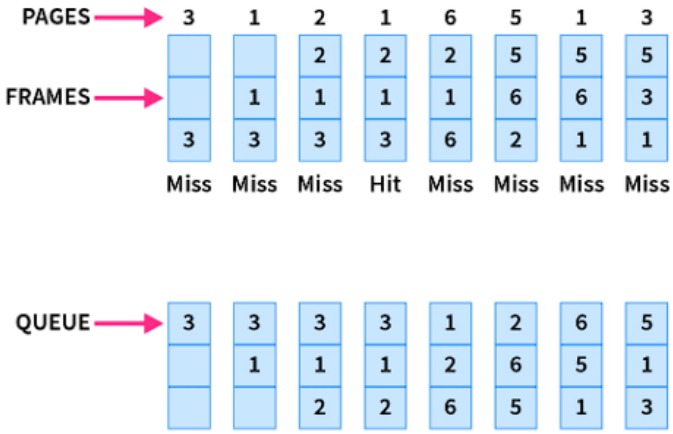
A first in, first out (FIFO) Page Replacement Algorithm assigns pages sequentially according to the order in which they were accessed most recently. The algorithm looks for the latest accessed page and then assigns the next page in the sequence to that location.

If a page is not found, the algorithm will search for the page with the oldest access time and assign it to the location. Pages are never reassigned to a location if they have already been assigned to that location by another 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**.

Example: Consider the page reference string as 3, 1, 2, 1, 6, 5, 1, 3 with 3-page frames. Let's try to find the number of page faults:



- Initially, all of the slots are empty so page faults occur at 3,1,2.

Page faults = 3

- When page 1 comes, it is in the memory so no page fault occurs.

Page faults = 3

- When page 6 comes, it is not present and a page fault occurs. Since there are no empty slots, we **remove the front of the queue, i.e 3.**

Page faults = 4

- When page 5 comes, it is also not present, and hence a page fault occurs. The front of the queue i.e. **1 is removed.**

Page faults = 5

- When page 1 comes, it is not found in memory and again a page fault occurs. The front of the queue i.e. **2 is removed.**

Page faults = 6

- When page 3 comes, it is again not found in memory, a page fault occurs, and **page 6 is removed** being on top of the queue

Total page faults = 7

Advantages:

- Simple to understand and implement
- Does not cause more overhead

Disadvantages:

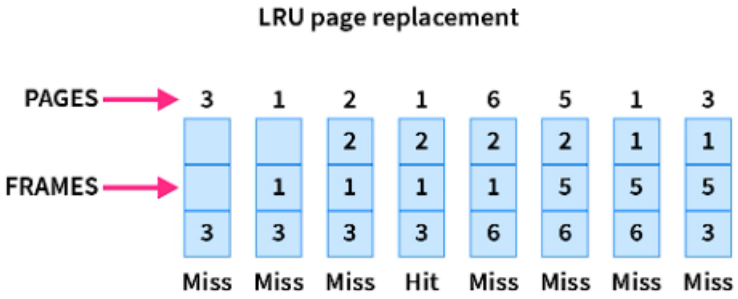
- Poor performance
- Doesn't use the frequency of the last used time and just simply replaces the oldest page.
- Suffers from Belady's anomaly.

2. Least Recently Used (LRU):

The least recently used page replacement algorithm keeps the track of usage of pages over a period of time. This algorithm works on the basis of the **principle of locality of a reference** which states that a program has a tendency to access the same set of memory locations repetitively over a short period of time. So pages that have been used heavily in the past are most likely to be used heavily in the future also.

In this algorithm, **when a page fault occurs, then the page that has not been used for the longest duration of time is replaced by the newly requested page**.

Example: Let's see the performance of the LRU on the same reference string of 3, 1, 2, 1, 6, 5, 1, 3 with 3-page frames:



- Initially, since all the slots are empty, **pages 3, 1, 2** cause a page fault and take the empty slots.

Page faults = 3

- When page 1 comes, it is in the memory and no page fault occurs.

Page faults = 3

- When page 6 comes, it is not in the memory, so a page fault occurs and the least recently used page **3 is removed**.

Page faults = 4

- When page 5 comes, it again causes a page fault, and page **1 is removed** as it is now the least recently used page.

Page faults = 5

- When page 1 comes again, it is not in the memory, and hence page **2 is removed** according to the LRU.

Page faults = 6

- When page 3 comes, the page fault occurs again and this time page **6 is removed** as the least recently used one.

Total page faults = 7

Now in the above example, the LRU causes the same page faults as the FIFO, but this may not always be the case as it will depend upon the series, the number of frames available in memory, etc. In fact, on most occasions, LRU is better than FIFO.

Advantages

- It is open for full analysis
- Doesn't suffer from Belady's anomaly
- Often more efficient than other algorithms

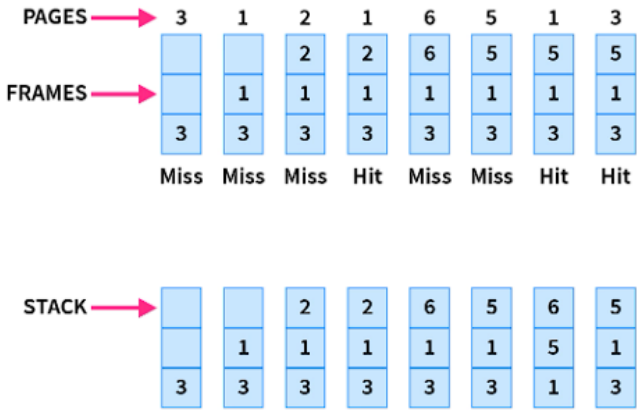
Disadvantages

- It requires additional data structures to be implemented
- More complex
- High hardware assistance is required

3. Last in First Out (LIFO):

This is the Last in First Out algorithm and works on LIFO principles. In this algorithm, the newest page is replaced by the requested page. Usually, this is done through a stack, where we maintain a stack of pages currently in the memory with the newest page being at the top. Whenever a page fault occurs, the page at the top of the stack is replaced.

Example: Let's see how the LIFO performs for our example string of 3, 1, 2, 1, 6, 5, 1, 3 with 3-page frames:



- Initially, since all the slots are empty, **page 3,1,2** causes a page fault and takes the empty slots.

Page faults = 3

- When page 1 comes, it is in the memory and no page fault occurs.

Page faults = 3

- When page 6 comes, the page fault occurs and page **2 is removed** as it is on the top of the stack and is the newest page.

Page faults = 4

- When page 5 comes, it is not in the memory, which causes a page fault, and hence page **6 is removed** being on top of the stack.

Page faults = 5

- When page 1 and page 3 come, they are in memory already, hence no page fault occurs.

Total page faults = 5

As you may notice, this is the same number of page faults as the Optimal page replacement algorithm. So we can say that for this series of pages, this is the best algorithm that can be implemented without the prior knowledge of future references.

Advantages

- Simple to understand
- Easy to implement
- No overhead

Disadvantages

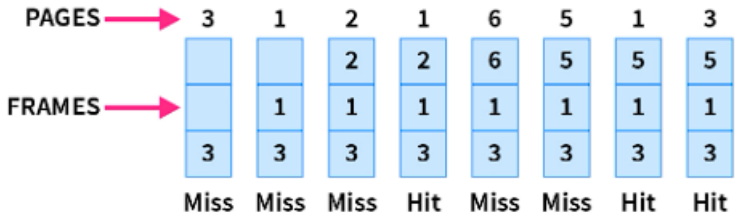
- Does not consider Locality principle, hence may produce worst performance
- The old pages may reside in memory forever even if they are not used

4. Optimal Page Replacement:

Optimal page replacement is the best page replacement algorithm as this algorithm results in the least number of page faults. In this algorithm, the pages are replaced with the ones that will not be used for the longest duration of time in the future. In simple terms, the pages that will be referred to farthest in the future are replaced in this algorithm.

Example:

Let's take the same page reference string 3, 1, 2, 1, 6, 5, 1, 3 with 3-page frames as we saw in FIFO. This also helps you understand how Optimal Page replacement works the best.



- Initially, since all the slots are empty, **pages 3, 1, 2** cause a page fault and take the empty slots.

Page faults = 3

- When page 1 comes, it is in the memory and no page fault occurs.

Page faults = 3

- When page 6 comes, it is not in the memory, so a page fault occurs and **2 is removed** as it is not going to be used again.

Page faults = 4

- When page 5 comes, it is also not in the memory and causes a page fault. Similar to above **6 is removed** as it is not going to be used again.

page faults = 5

- When page 1 and page 3 come, they are in the memory so no page fault occurs.

Total page faults = 5

Advantages

- Excellent efficiency
- Less complexity
- Simple data structures can be used to implement
- Used as the benchmark for other algorithms

Disadvantages

- More time consuming
- Difficult for error handling
- Need future awareness of the programs, which is not possible every time

5. Random Page Replacement:

This algorithm, as the name suggests, chooses any random page in the memory to be replaced by the requested page. This algorithm can behave like any of the algorithms based on the random page chosen to be replaced.

Example: Suppose we choose to replace the middle frame every time a page fault occurs. Let's see how our series of 3, 1, 2, 1, 6, 5, 1, 3 with 3-page frames perform with this algorithm:

- Initially, since all the slots are empty, **page 3,1,2** causes a page fault and takes the empty slots

Page faults = 3

- When page 1 comes, it is in the memory and no page fault occurs.

Page faults = 3

- When page 6 comes, the page fault occurs, we replace the middle element i.e **1 is removed.**

Page faults = 4

- When page 5 comes, the page fault occurs again and middle element **6 is removed**

Page faults = 5

- When page 1 comes, there is again a page fault, and again the middle element **5 is removed**

Page faults = 6

- When page 3 comes, it is in memory, hence no page fault occurs.

Total page faults = 6

As we can see, the performance is not the best, but it's also not the worst. The performance in the random replacement algorithm depends on the choice of the page chosen at random.

Advantages

- Easy to understand and implement
- No extra data structure needed to implement
- No overhead

Disadvantages

- Can not be analyzed, may produce different performances for the same series
- Can suffer from Belady's anomaly

Calculating Hit Ratio:

Total number of page hits = Total number of references – Total number of page misses or page faults

Thus, **Hit ratio** = Total number of page hits / Total number of references

Calculating Miss Ratio:

Miss ratio = Total number of page misses / Total number of references

Alternatively,

Miss ratio = 1 – Hit ratio

Allocation of Frames:

The main memory of the operating system is divided into various frames. The process is stored in these frames, and once the process is saved as a frame, the CPU may run it. As a result, the operating system must set aside enough frames for each process. As a result, the operating system uses various algorithms in order to assign the frame.

Demand paging is used to implement virtual memory, an essential operating system feature. It requires the development of a page replacement mechanism and a frame allocation system. If you have multiple processes, the frame allocation techniques are utilized to define how many frames to allot to each one.

A number of factors constrain the strategies for allocating frames:

1. You cannot assign more frames than the total number of frames available.
2. A specific number of frames should be assigned to each process. This limitation is due to two factors. The first is that when the number of frames assigned drops, the page fault ratio grows, decreasing the process's execution performance. Second, there should be sufficient frames to hold all the multiple pages that any instruction may reference.

Frame Allocation Constraints:

- The Frames that can be allocated cannot be greater than total number of frames.
- Each process should be given a set minimum number of frames.
- When fewer frames are allocated then the page fault ration increases and the process execution become less efficient.
- There ought to be sufficient frames to accommodate all the many pages that a single instruction may refer to.

There are mainly five ways of frame allocation algorithms in the OS. These are as follows:

1. **Equal Frame Allocation**
2. **Proportional Frame Allocation**
3. **Priority Frame Allocation**
4. **Global Replacement Allocation**
5. **Local Replacement Allocation**

Equal Frame Allocation:

In equal frame allocation, the processes are assigned equally among the processes in the OS. For example, if the system has 30 frames and 7 processes, each process will get 4 frames. The 2 frames that are not assigned to any system process may be used as a free-frame buffer pool in the system.

Disadvantage:

In a system with processes of varying sizes, assigning equal frames to each process makes little sense. Many allotted empty frames will be wasted if many frames are assigned to a small task.

Proportional Frame Allocation:

The proportional frame allocation technique assigns frames based on the size needed for execution and the total number of frames in memory.

The allocated frames for a process **pi** of size **si** are **ai = (si/S)*m**, in which **S** represents the total of all process sizes, and **m** represents the number of frames in the system.

Disadvantage: The only drawback of this algorithm is that it doesn't allocate frames based on priority. Priority frame allocation solves this problem.

Priority Frame Allocation:

Priority frame allocation assigns frames based on the number of frame allocations and the processes. Suppose a process has a high priority and requires more frames that many frames will be allocated to it. Following that, lesser priority processes are allocated.

Global Replacement Allocation:

When a process requires a page that isn't currently in memory, it may put it in and select a frame from the all frames sets, even if another process is already utilizing that frame. In other words, one process may take a frame from another.

Advantages: Process performance is not hampered, resulting in higher system throughput.

Disadvantages: The process itself may not solely control the page fault ratio of a process. The paging behavior of other processes also influences the number of pages in memory for a process.

Local Replacement Allocation:

When a process requires a page that isn't already in memory, it can bring it in and assign it a frame from its set of allocated frames.

Advantages: The paging behavior of a specific process has an effect on the pages in memory and the page fault ratio.

Disadvantages: A low priority process may obstruct a high priority process by refusing to share its frames.

Global Vs. Local Replacement Allocation: The number of frames assigned to a process does not change using a local replacement strategy. On the other hand, using global replacement, a process can choose only frames granted to other processes and enhance the number of frames allocated.

Thrashing:

Thrashing in OS is a phenomenon that occurs in computer operating systems when the system spends an **excessive amount** of time swapping data between physical memory (RAM) and virtual memory (disk storage) due to **high memory demand** and **low available resources.**

Thrashing can occur when there are too many processes running on a system and not enough physical memory to accommodate them all. As a result, the operating system must **constantly** swap pages of memory between physical memory and virtual memory. This can lead to a significant **decrease** in system performance, as the CPU is spending more time swapping pages than it is actually executing code.

Symptoms and How to Detect it?

The following are some of the symptoms of thrashing in an operating system:

1. High CPU utilization: When a system is thrashing, the CPU is spending a lot of time swapping pages of memory between physical memory and disk. This can lead to high CPU utilization, even when the system is not performing any significant work.

2. Increased Disk Activity: When the system is Thrashing in OS, the disk activity increases significantly as the system tries to swap data between physical memory and virtual memory.

3. High page fault rate: A page fault is an event that occurs when the CPU tries to access a page of memory that is not currently in physical memory. Thrashing can cause a high page fault rate, as the operating system is constantly swapping pages of memory between physical memory and disk.

4. Slow Response Time: When the system is Thrashing in OS, its response time slows significantly.

If you are experiencing any of these symptoms, it is possible that your system is **thrashing**. You can use a **system monitoring tool** to check the CPU utilization, page fault rate, and disk activity to confirm this.

Algorithms during 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.

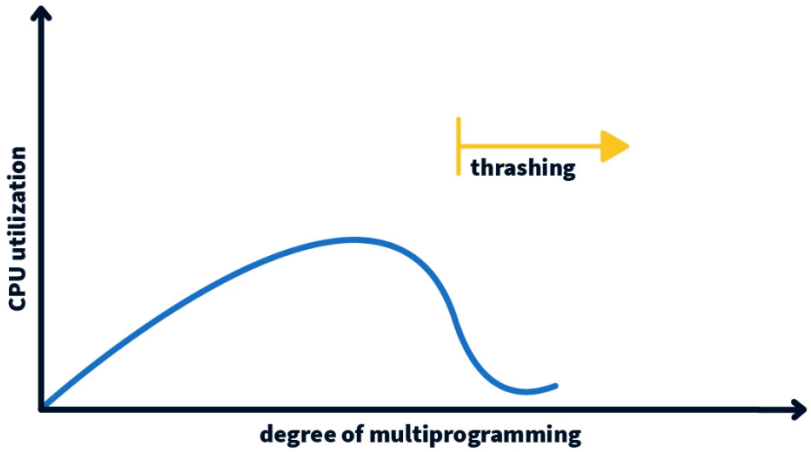
Local Page Replacement: Unlike the Global Page replacement, the local page replacement will select pages which only belongs to that process. Due to this, there is a chance of a reduction in the thrashing. As it is also proved that there are many disadvantages of Local Page replacement. Thus, local page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

Causes of Thrashing:

The main causes of thrashing in an operating system are:

1. High degree of multiprogramming:

When too many processes are running on a system, the operating system may not have enough physical memory to accommodate them all. This can lead to thrashing, as the operating system is constantly swapping pages of memory between physical memory and disk.



2. Lack of frames: Frames are the units of memory that are used to store pages of memory. If there are not enough frames available, the operating system will have to swap pages of memory to disk, which can lead to thrashing.

3. Page replacement policy: The page replacement policy is the algorithm that the operating system uses to decide which pages of memory to swap to disk. If the page replacement policy is not effective, it can lead to thrashing.

4. Insufficient physical memory: If the system does not have enough physical memory, it will have to swap pages of memory to disk more often, which can lead to thrashing.

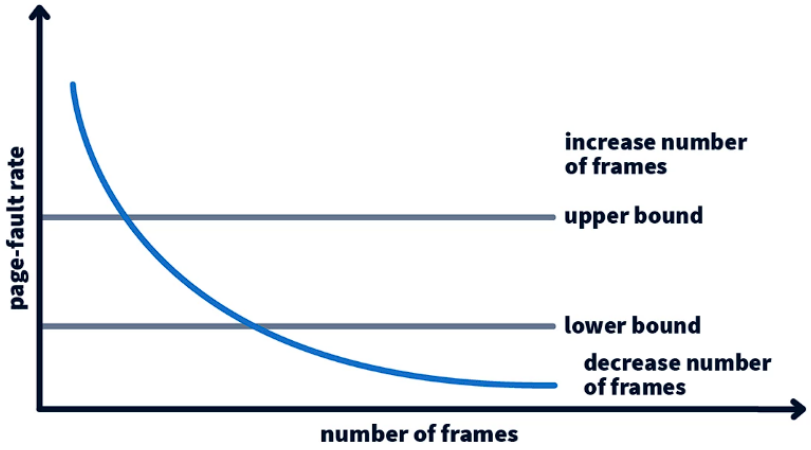
5. Inefficient memory management: If the operating system is not managing memory efficiently, it can lead to fragmentation of physical memory, which can also lead to thrashing.

6. Poorly designed applications: Applications that use excessive memory or that have poor memory management practices can also contribute to thrashing.

Techniques to Prevent Thrashing:

There are a number of ways to eliminate thrashing, including:

- **Increase the amount of physical memory:** This is the most effective way to eliminate thrashing, as it will give the operating system more space to store pages of memory in physical memory.



- **Reduce the degree of multiprogramming:** This means reducing the number of processes that are running on the system. This can be done by terminating or suspending processes, or by denying new processes from starting.
- **Use an effective page replacement policy:** The page replacement policy is the algorithm that the operating system uses to decide which pages of memory to swap to disk. An effective page replacement policy can help to minimize the number of page faults that occur, which can help to eliminate thrashing.
- **Optimize applications:** Applications should be designed to use memory efficiently and to avoid memory management practices that can lead to thrashing. For example, applications should avoid using excessive memory or using inefficient data structures.
- **Monitor the system's resource usage:** Monitor the system's CPU utilization, memory usage, and disk activity. If you notice that any of these resources are being overutilized, you may need to take steps to reduce the load on the system.
- **Use a system monitoring tool:** A system monitoring tool can help you to identify bottlenecks in your system and to track the system's resource usage over time. This information can help you to identify potential problems before they cause thrashing.

Effects on System Performance and User Experience:

Thrashing has a significant negative impact on system performance and user experience. Here are some of the specific effects of thrashing on system performance and user experience:

- **Slow application response times:** Thrashing can cause applications to take longer to load and respond to user input. This is because the operating system is spending a lot of time swapping pages of memory between physical memory and disk, rather than executing the application's code.
- **Increased system load:** Thrashing can cause the system to become more overloaded. This is because the CPU is spending a lot of time swapping pages of memory between physical memory and disk, rather than executing code. This can lead to increased CPU utilization, memory usage, and disk activity.
- **System crashes:** In severe cases, thrashing can cause the system to crash. This is because the operating system may not be able to allocate enough memory to all of the running processes, or it may not be able to swap pages of memory between physical memory and disk quickly enough.

Thrashing vs. Swapping:

Thrashing	Swapping
Here, the CPU spends most of its time swapping pages in and out of the main memory. It results in a decrease in the system's performance.	It is a technique that includes moving an entire process or a part of it from the main memory to the secondary memory and vice versa.
It is caused by overcommitment of the physical memory where the system system tries to manage many processes with only limited RAM.	It is done by the OS to manage the memory efficiently and mainly to free up the space in RAM when not required by the process.
It degrades the system's performance.	It improves the overall performance of the system.
To resolve it, the system can reduce the number of running processes or can increase the physical memory(RAM).	It can be controlled by many memory management techniques to optimize the memory usage.
If not taken care of, it can lead to delays and crashes.	It prevents memory exhaustion leading to efficient memory usage.

Real-life Examples of Thrashing and their Impact on Applications:

Here are some real-life examples of thrashing and their impact on applications:

1. **A Web Server:** A web server may thrash if it is overloaded with requests and does not have enough memory to handle them all. This can cause the server to become unresponsive and even crash.

   Impact: A web server that is thrashing may be unable to respond to requests from users, resulting in down time and lost revenue.

2. **A database server:** A database server may thrash if it is trying to process too many queries at the same time and does not have enough memory to store all of the data that it needs. This can cause the server to become slow and unresponsive.

   Impact: A database server that is thrashing may be unable to process transactions quickly enough, resulting in delays for users and businesses.

3. **A video editing application:** A video editing application may thrash if it is trying to edit a large video file and does not have enough memory to store the entire file in memory. This can cause the application to become slow and unresponsive, and it may even crash.

   Impact: A video editing application that is thrashing may be unable to edit videos smoothly, resulting in unusable footage.

4. **A video game:** A video game may thrash if it is trying to render a large scene and does not have enough memory to store all of the textures and other resources that it needs. This can cause the game to become slow and laggy.

   Impact: A video game that is thrashing may be unplayable, due to lag and stuttering.

Thrashing in Virtual Memory Systems:

Thrashing in operating systems with virtual memory is a condition where the system spends so much time swapping pages between main memory and secondary storage (such as a hard disk) that it cannot efficiently execute any user processes. This can lead to a significant decrease in system performance, and in some cases, the system may become unresponsive or even crash.

Thrashing is typically caused when the system has insufficient physical memory (RAM) to support the workload. When the system runs out of physical memory, it must start swapping pages to disk. This frees up physical memory for other processes, but it can also lead to a vicious cycle, where the system spends more and more time swapping pages and less and less time executing actual instructions.

Future Trends:

Thrashing is a problem that has been around since the early days of operating systems, but it is still a relevant issue today. With the increasing complexity of workloads and the growing memory demands of applications, the risk of thrashing is still present.

However, there are a number of trends in the future of operating systems and computing that could help to mitigate thrashing. These trends include:

1. The increasing availability of memory.
2. The rise of non-volatile memory.
3. The development of new memory management techniques.
4. Using machine learning to predict and prevent thrashing.
5. Using distributed memory management techniques.