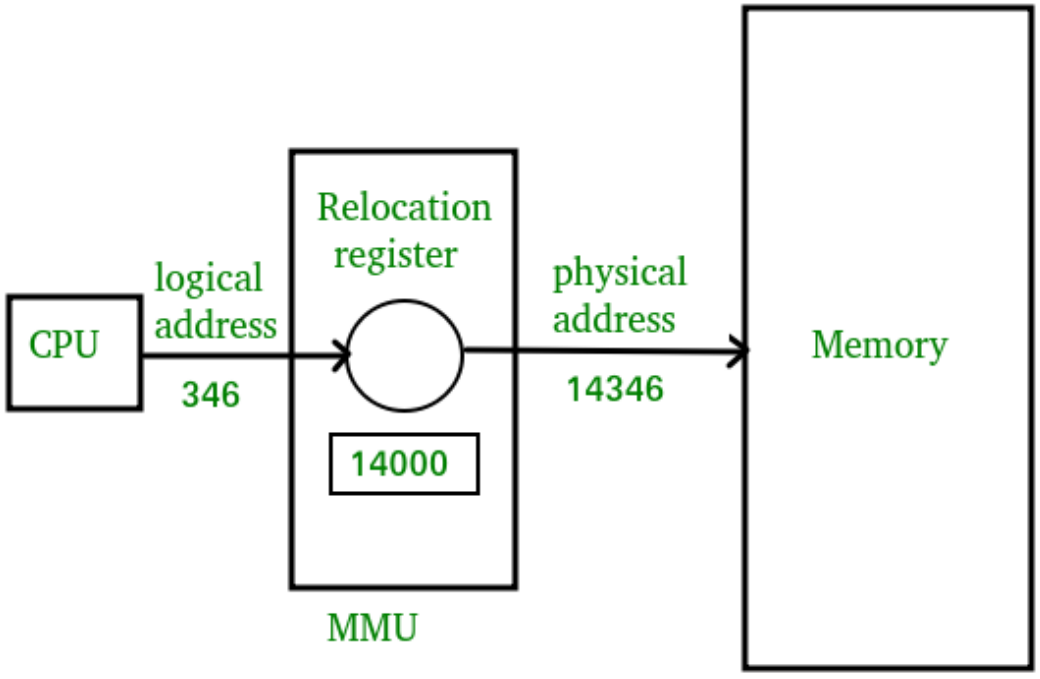
MODULE-5

Logical Address is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically, therefore, it is also known as Virtual Address. This address is used as a reference to access the physical memory location by CPU. The term Logical Address Space is used for the set of all logical addresses generated by a program's perspective.
The hardware device called Memory-Management Unit is used for mapping logical address to its corresponding physical address.
Physical Address identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, the logical address must be mapped to the physical address by MMU before they are used. The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.



Mapping virtual-address to physical-addresses
Differences Between Logical and Physical Address in Operating System

1. The basic difference between Logical and physical address is that Logical address is generated by CPU in perspective of a program whereas the physical address is a location that exists in the memory unit.
2. Logical Address Space is the set of all logical addresses generated by CPU for a program whereas the set of all physical address mapped to corresponding logical addresses is called Physical Address Space.
3. The logical address does not exist physically in the memory whereas physical address is a location in the memory that can be accessed physically.
4. Identical logical addresses are generated by Compile-time and Load time address binding methods whereas they differs from each other in run-time address binding method. Please refer this for details.
5. The logical address is generated by the CPU while the program is running whereas the physical address is computed by the Memory Management Unit (MMU).

Comparison Chart:

Parameter	LOGICAL ADDRESS	PHYSICAL ADDRESS
Basic	generated by CPU	location in a memory unit
Address Space	Logical Address Space is set of all logical addresses generated by CPU in reference to a program.	Physical Address is set of all physical addresses mapped to the corresponding logical addresses.
Visibility	User can view the logical address of a program.	User can never view physical address of program.
Generation	generated by the CPU	Computed by MMU
Access	The user can use the logical address to access the physical address.	The user can indirectly access physical address but not directly.
Editable	Logical address can be change.	Physical address will not change.
Also called	virtual address.	real address.

Contiguous memory allocation

Contiguous Memory Allocation is a type of memory allocation technique where processes are allotted a continuous block of space in memory. This block can be of fixed size for all the processes in a fixed size partition scheme or can be of variable size depending on the requirements of the process in a variable size partition scheme.

Scope:-

This article explains the concept of contiguous memory allocation in the operating system.

This article explains the two techniques of fixed size partition and variable size partition in contiguous memory allocation.

The overall article meets the defined scope.

Contiguous memory allocation in the operating system is a memory allocation technique. But what is memory allocation? When a program or process is to be executed, it needs some space in the memory. For this reason, some part of the memory has to be allotted to a process according to its requirements. This process is called memory allocation.

One such memory allocation technique is contiguous memory allocation. As the name implies, we allocate contiguous blocks of memory to each process using this technique. So, whenever a process wants to enter the main memory, we allocate a continuous segment from the totally empty space to the process based on its size.

Contiguous Memory Allocation Techniques

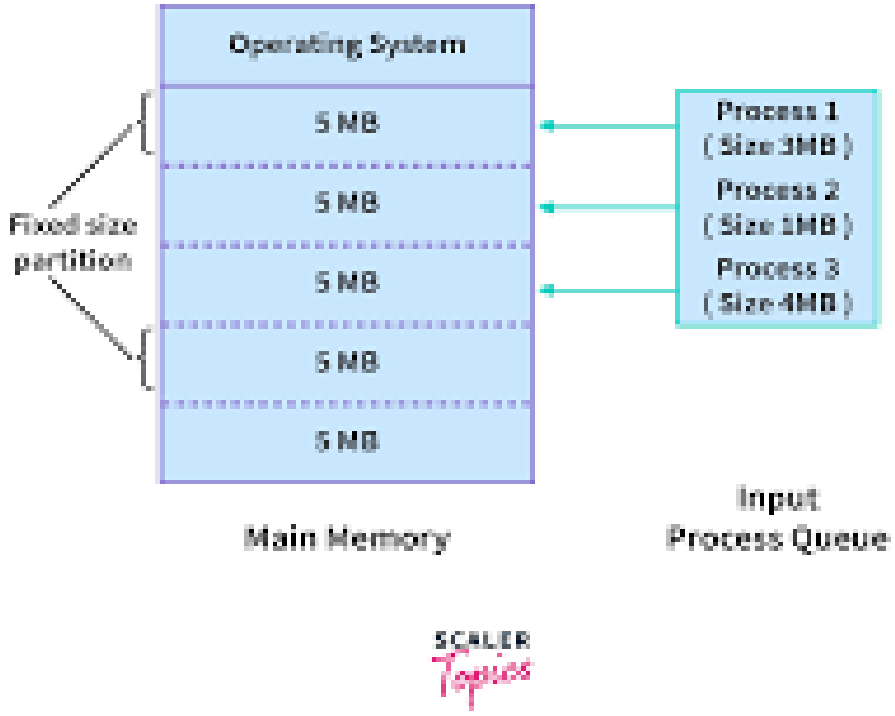
Whenever a process has to be allocated space in the memory, following the contiguous memory allocation technique, we have to allot the process a continuous empty block of space to reside. This allocation can be done in two ways:

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

Let us look at both of these schemes in detail, along with their advantages and disadvantages.

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.

Advantages

The advantages of a fixed-size partition scheme are:

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.

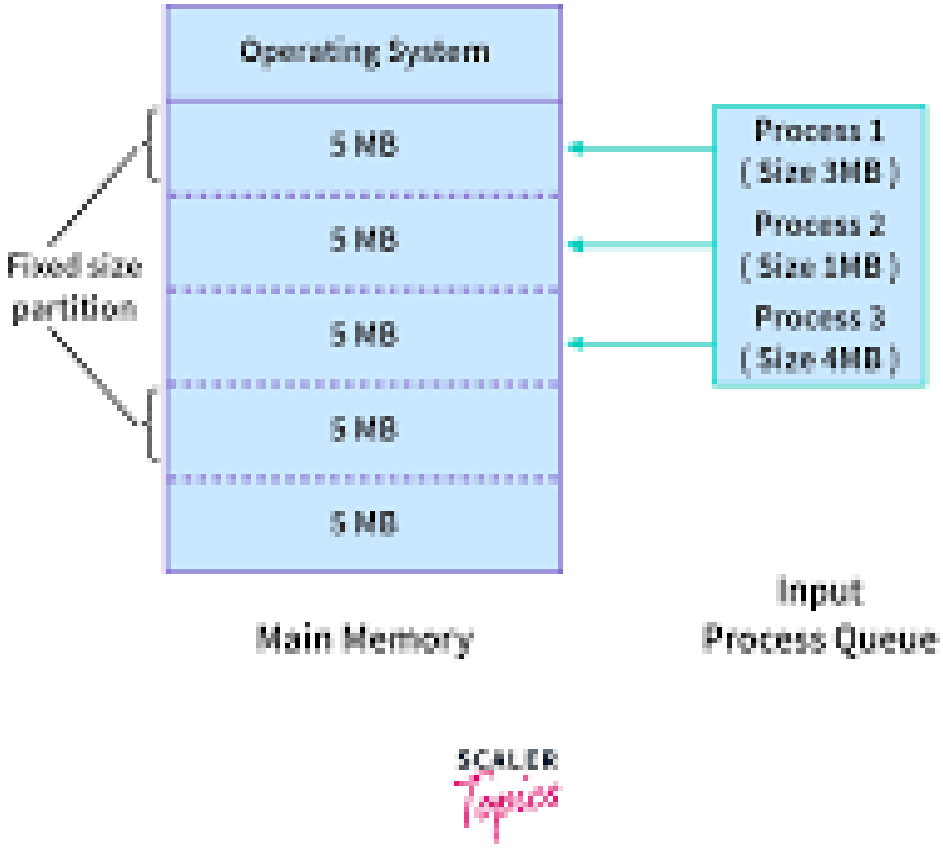
Disdvantages

Though the fixed-size partition scheme has many advantages, it also has some 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.

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, 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

The advantages of a variable-size partition scheme are:

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.

Disdvantages

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.

Conclusion

- In contiguous memory allocation, we allocate contiguous blocks of memory to each process when it is brought in the main memory to be executed.
- There are two techniques for contiguous memory allocation:
    - Fixed Size Partitioning: Each process is allotted to a fixed size continuous block in the main memory.
    - Variable Size Partitioning: Each process is allotted space depending upon its requirements. There is no defined fixed-size block.
- There are three strategies to allot a hole to an incoming process:
    - First-Fit: Allot the process to the first hole, which is big enough.
    - Best-Fit: Allot the smallest hole that satisfies the requirements of the process.
    - Worst-Fit: Allot the largest size hole among all to the incoming process.

Fragmentation

Fragmentation is an unwanted problem in the operating system in which the processes are loaded and unloaded from memory, and free memory space is fragmented. Processes can't be assigned to memory blocks due to their small size, and the memory blocks stay unused.

Contiguous memory allocation allocates space to processes whenever the processes enter **RAM**. These **RAM** spaces are divided either by fixed partitioning or by dynamic partitioning. As the process is loaded and unloaded from memory, these areas are fragmented into small pieces of memory that cannot be allocated to coming processes.

Fragmentation is an unwanted problem in the operating system in which the processes are loaded and unloaded from memory, and free memory space is fragmented. Processes can't be assigned to memory blocks due to their small size, and the memory blocks stay unused. It is also necessary to understand that as programs are loaded and deleted from memory, they generate free space or a hole in the memory. These small blocks cannot be allotted to new arriving processes, resulting in inefficient memory use.

The conditions of fragmentation depend on the memory allocation system. As the process is loaded and unloaded from memory, these areas are fragmented into small pieces of memory that cannot be allocated to incoming processes. It is called **fragmentation**.

Causes of Fragmentation

User processes are loaded and unloaded from the main memory, and processes are kept in memory blocks in the main memory. Many spaces remain after process loading and swapping that another process cannot load due to their size. Main memory is available, but its space is insufficient to load another process because of the dynamical allocation of main memory processes.

Types of Fragmentation

There are mainly two types of fragmentation in the operating system. These are as follows:

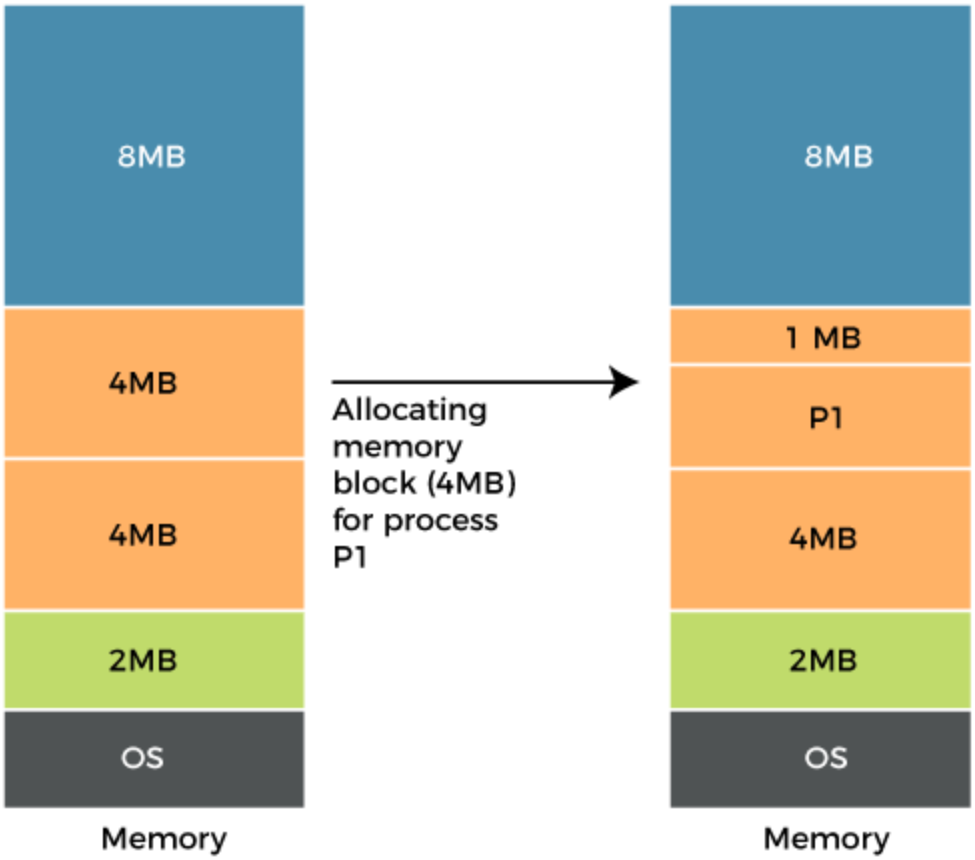
1. **Internal Fragmentation**
2. **External Fragmentation**

Internal Fragmentation

When a process is allocated to a memory block, and if the process is smaller than the amount of memory requested, a free space is created in the given memory block. Due to this, the free space of the memory block is unused, which causes **internal** fragmentation.

For Example:

Assume that memory allocation in RAM is done using fixed partitioning (i.e., memory blocks of fixed sizes). **2MB, 4MB, 4MB**, and **8MB** are the available sizes. The Operating System uses a part of this RAM.



Let's suppose a process **P1** with a size of **3MB** arrives and is given a memory block of **4MB**. As a result, the **1MB** of free space in this block is unused and cannot be used to allocate memory to another process. It is known as **internal fragmentation**.

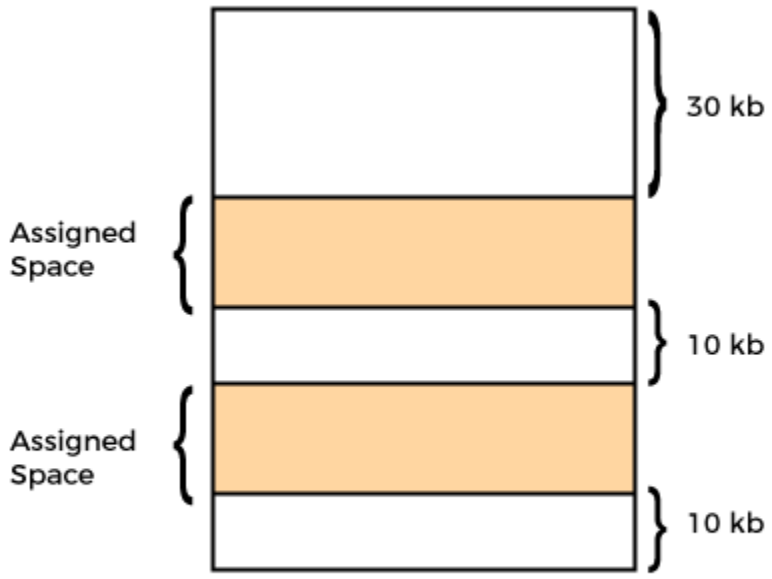
How to avoid internal fragmentation?

The problem of internal fragmentation may arise due to the fixed sizes of the memory blocks. It may be solved by assigning space to the process via dynamic partitioning. Dynamic partitioning allocates only the amount of space requested by the process. As a result, there is no internal fragmentation.

External Fragmentation

External fragmentation happens when a dynamic memory allocation method allocates some memory but leaves a small amount of memory unusable. The quantity of available memory is substantially reduced if there is too much external fragmentation. There is enough memory space to complete a request, but it is not contiguous. It's known as **external** fragmentation.

For Example:



Process 05 needs 45kb memory space

Let's take the example of external fragmentation. In the above diagram, you can see that there is sufficient space **(50 KB)** to run a process **(05) (need 45KB)**, but the memory is not contiguous. You can use compaction, paging, and segmentation to use the free space to execute a process.

How to remove external fragmentation?

This problem occurs when you allocate RAM to processes continuously. It is done in paging and segmentation, where memory is allocated to processes non-contiguously. As a result, if you remove this condition, external fragmentation may be decreased.

Compaction is another method for removing external fragmentation. External fragmentation may be decreased when dynamic partitioning is used for memory allocation by combining all free memory into a single large block. The larger memory block is used to allocate space based on the requirements of the new processes. This method is also known as defragmentation.

Advantages and disadvantages of fragmentation

There are various advantages and disadvantages of fragmentation. Some of them are as follows:

Advantages

There are various advantages of fragmentation. Some of them are as follows:

Fast Data Writes

Data write in a system that supports data fragmentation may be faster than reorganizing data storage to enable contiguous data writes.

Fewer Failures

If there is insufficient sequential space in a system that does not support fragmentation, the write will fail.

Storage Optimization

A fragmented system might potentially make better use of a storage device by utilizing every available storage block.

Disadvantages

There are various disadvantages of fragmentation. Some of them are as follows:

Need for regular defragmentation

A more fragmented storage device's performance will degrade with time, necessitating the requirement for time-consuming defragmentation operations.

Slower Read Times

The time it takes to read a non-sequential file might increase as a storage device becomes more fragmented.

Sr. No.	Internal Fragmentation	External Fragmentation
1.	Frames square measure designated for processing in internal fragmentation of fixed-sized storage.	Variable-sized memory frames square measure designated to the process during external fragmentation.
2.	When the system or procedure is greater than the storage, internal fragmentation occurs.	Whenever the system or procedure is withdrawn, external fragmentation occurs.
3.	The internal fragmentation approach is the frame with the perfect match.	Compression, paging, and differentiation are alternatives to external fragmentation.
4.	Internal fragmentation happens whenever the storage is split into fragments of a fixed length.	External fragmentation happens whenever the storage is split into segments of variable size depending on the process length.
5.	The distinction between the assigned memory and the storage or memory needed is considered as internal fragmentation.	The empty spaces created among non-contiguous pieces of storage are too tiny for a new system to operate, considered as external fragmentation.

Compaction:-

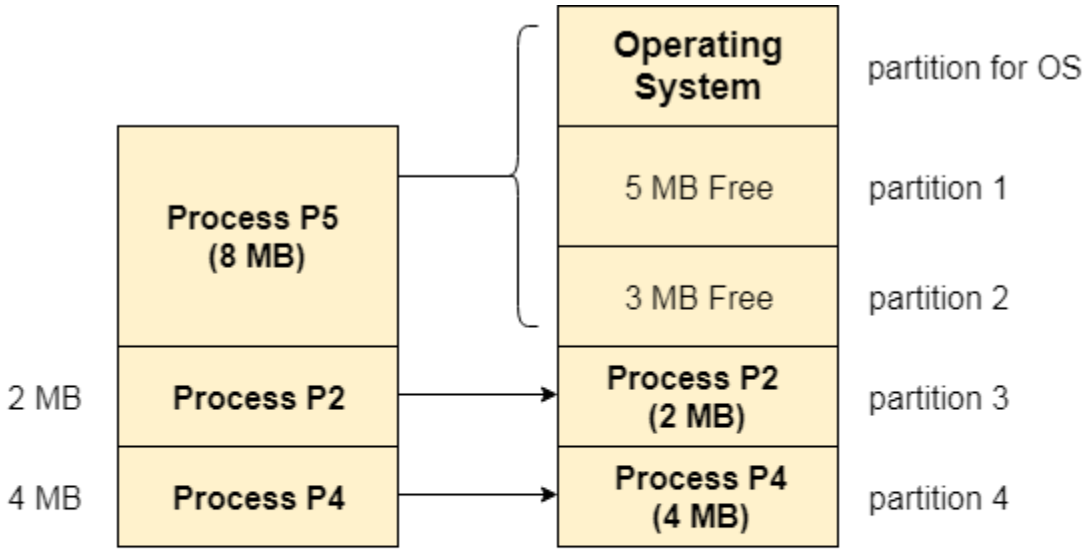
Compaction

We got to know that the dynamic partitioning suffers from external fragmentation. However, this can cause some serious problems.

To avoid compaction, we need to change the rule which says that the process can't be stored in the different places in the memory.

We can also use compaction to minimize the probability of external fragmentation. In compaction, all the free partitions are made contiguous and all the loaded partitions are brought together.

By applying this technique, we can store the bigger processes in the memory. The free partitions are merged which can now be allocated according to the needs of new processes. This technique is also called defragmentation.



Compaction

As shown in the image above, the process P5, which could not be loaded into the memory due to the lack of contiguous space, can be loaded now in the memory since the free partitions are made contiguous.

Problem with Compaction

The efficiency of the system is decreased in the case of compaction due to the fact that all the free spaces will be transferred from several places to a single place.

Huge amount of time is invested for this procedure and the CPU will remain idle for all this time. Despite of the fact that the compaction avoids external fragmentation, it makes system inefficient.

Let us consider that OS needs 6 NS to copy 1 byte from one place to another.

1. 1 B transfer needs 6 NS
2. 256 MB transfer needs 256 X 2^20 X 6 X 10 ^ -9 secs

hence, it is proved to some extent that the larger size memory transfer needs some huge amount of time that is in seconds.

Paging:-

Paging is a storage mechanism used in OS to retrieve processes from secondary storage to the main memory as pages. The primary concept behind paging is to break each process into individual pages. Thus the primary memory would also be separated into frames.

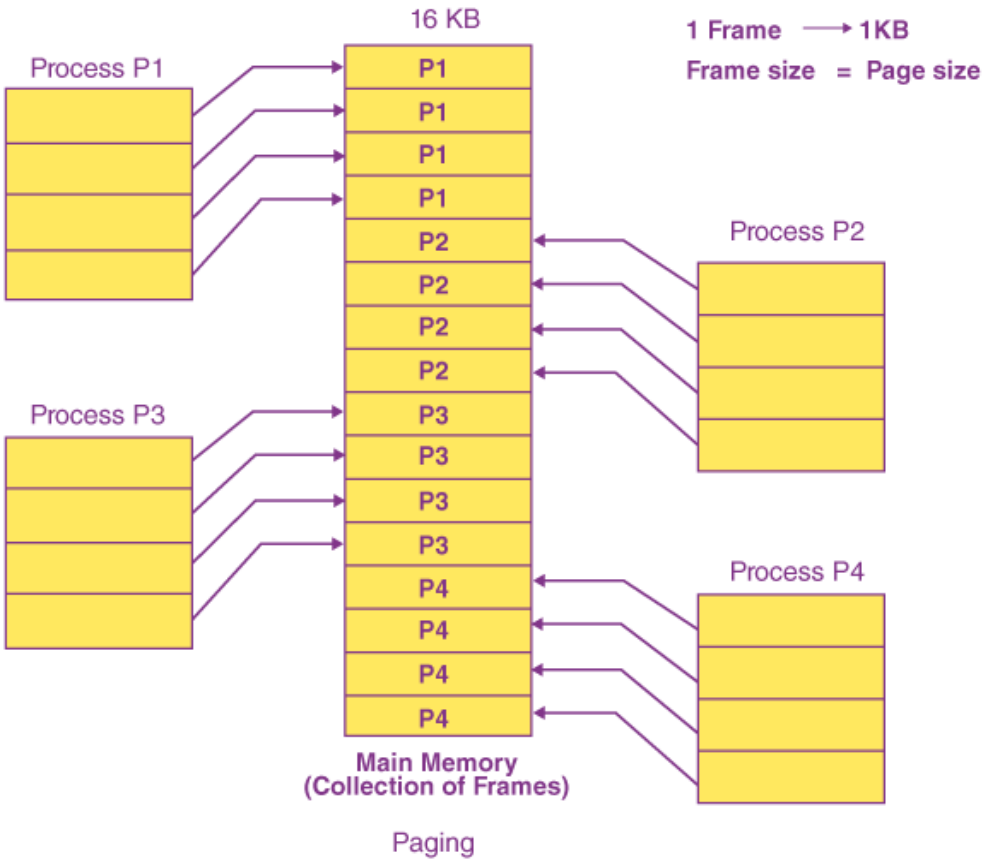
One page of the process must be saved in one of the given memory frames. These pages can be stored in various memory locations, but finding contiguous frames/holes is always the main goal. Process pages are usually only brought into the main memory when they are needed; else, they are stored in the secondary storage.

The frame sizes may vary depending on the OS. Each frame must be of the same size. Since the pages present in paging are mapped on to the frames, the page size should be similar to the frame size.

Example

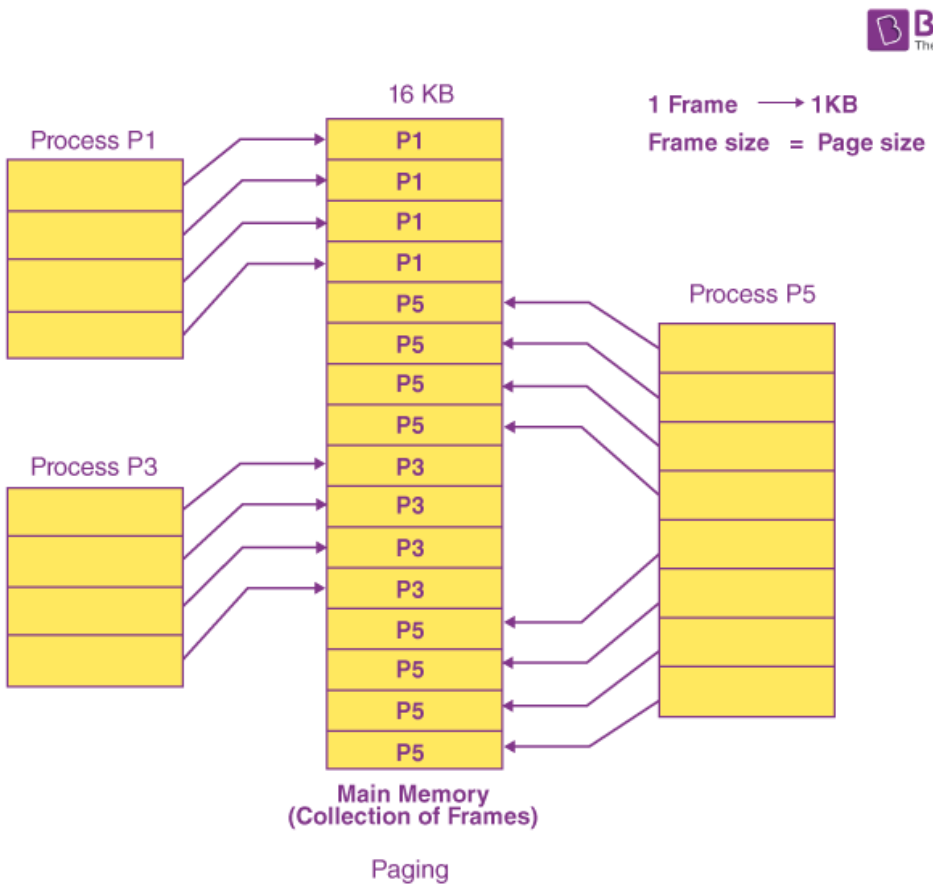
Assuming that the main memory is 16 KB and the frame size is 1 KB, the main memory will be partitioned into a collection of 16 1 KB frames. P1, P2, P3, and P4 are the four processes in the system, each of which is 4 KB in size. Each process is separated into 1 KB pages, allowing one page to be saved in a single frame.

Because all of the frames are initially empty, the pages of the processes will be stored in a continuous manner. The graphic below depicts frames, pages, and the mapping between them.



Consider the case when P2 and P4 are shifted to the waiting state after a period of time. Eight frames are now empty, allowing other pages to be loaded in their stead. Inside the ready queue is the process P5, which is 8 KB (8 pages) in size.

Given that we have 8 noncontiguous frames accessible in memory, paging allows us to store the process in many locations. As a result, we can load the process P5 page instead of P2 and P4.



Memory Management Unit

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.

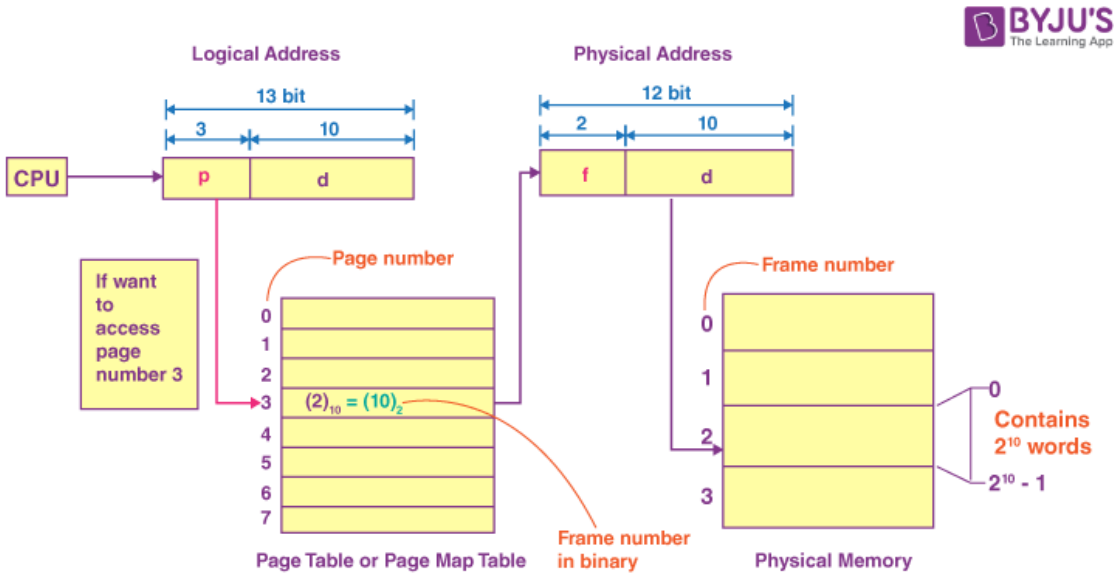
Examples

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:

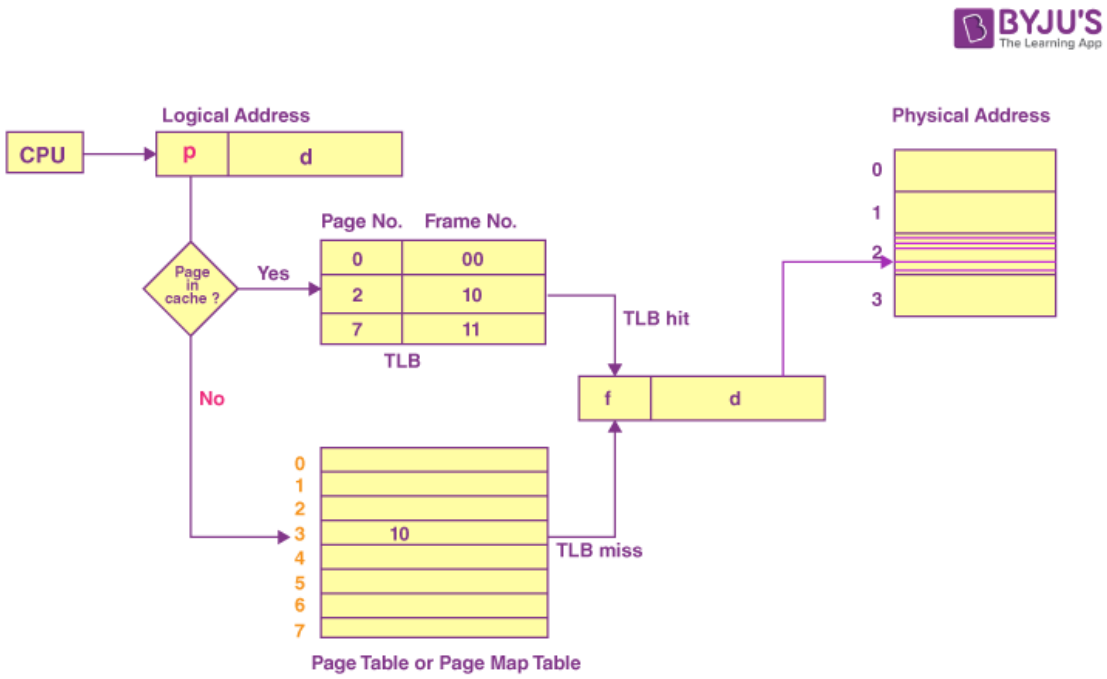
- Page offset(d): It refers to the number of bits necessary to represent a certain word on a page, page size in Logical Address Space, or page word number or page offset.
- Page number(p): It is the number of bits needed to represent the pages in the Logical Address Space or the page number.

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.

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.

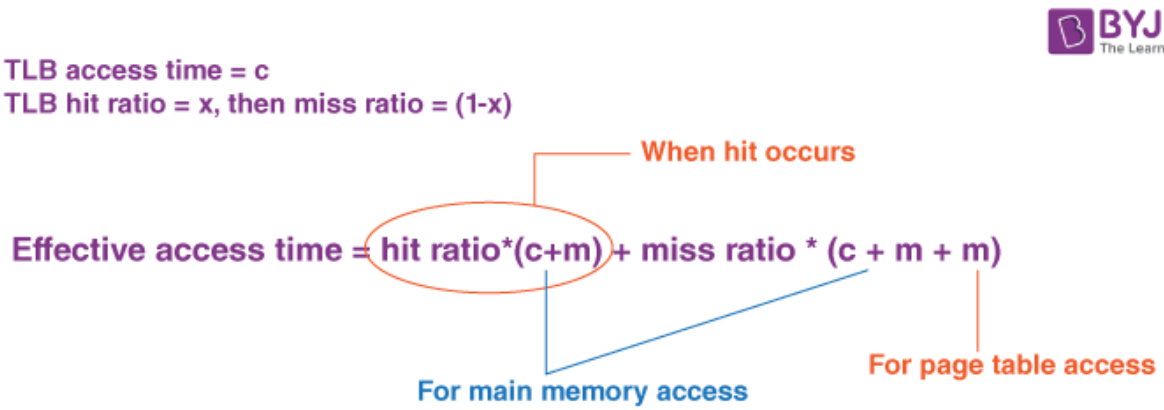
- The TLB is a high-speed, associative memory.
- LB entries are made up of two parts: a value and a tag.
- When this memory is accessed, an item is compared to all tags at the same time.
- If the object is located, the value associated with it is returned.



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)



Hardware support for paging

For the sake of efficiency, linear addresses are grouped in fixed-length intervals called *pages*; contiguous linear addresses within a page are mapped into contiguous physical addresses. In this way, the kernel can specify the physical address and the access rights of a page instead of those of all the linear addresses included in it. Following the usual convention, we shall use the term "page" to refer both to a set of linear addresses and to the data contained in this group of addresses.

The paging unit thinks of all RAM as partitioned into fixed-length *page frames* (sometimes referred to as *physical pages*). Each page frame contains a page — that is, the length of a page frame coincides with that of a page. A page frame is a constituent of main memory, and hence it is a storage area. It is important to distinguish a page from a page frame; the former is just a block of data, which may be stored in any page frame or on disk.

The data structures that map linear to physical addresses are called *Page Tables* ; they are stored in main memory and must be properly initialized by the kernel before enabling the paging unit.

Paging Protection

The paging process should be protected by using the concept of insertion of an additional bit called Valid/Invalid bit. Paging Memory protection in paging is achieved by associating protection bits with each page. These bits are associated with each page table entry and specify protection on the corresponding page.

Advantages of Paging in OS

- Memory management is effective.
- Simplicity in partitioning (non-contiguous memory allocation).
- Allocating memory is simple and inexpensive.
- Pages are simple to share.
- No compaction is necessary.
- No external fragmentation.
- More efficient swapping.

Disadvantages of Paging in OS

- Internal fragmentation (only at the last page of the process).
- Address translation necessitates the use of specialized hardware.
- Page Table is stored in the main memory.
- Address translation lengthens memory cycle times.
- Memory reference overhead is caused by multi-level paging.
- Memory access time is longer.

Virtual Memory

Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory.

In this scheme, User can load the bigger size processes than the available main memory by having the illusion that the memory is available to load the process.

Instead of loading one big process in the main memory, the Operating System loads the different parts of more than one process in the main memory.

By doing this, the degree of multiprogramming will be increased and therefore, the CPU utilization will also be increased.

How Virtual Memory Works?

In modern word, virtual memory has become quite common these days. In this scheme, whenever some pages needs to be loaded in the main memory for the execution and the memory is not available for those many pages, then in that case, instead of stopping the pages from entering in the main memory, the OS search for the RAM area that are least used in the recent times or that are not referenced and copy that into the secondary memory to make the space for the new pages in the main memory.

Since all this procedure happens automatically, therefore it makes the computer feel like it is having the unlimited RAM.

Advantages of Virtual Memory

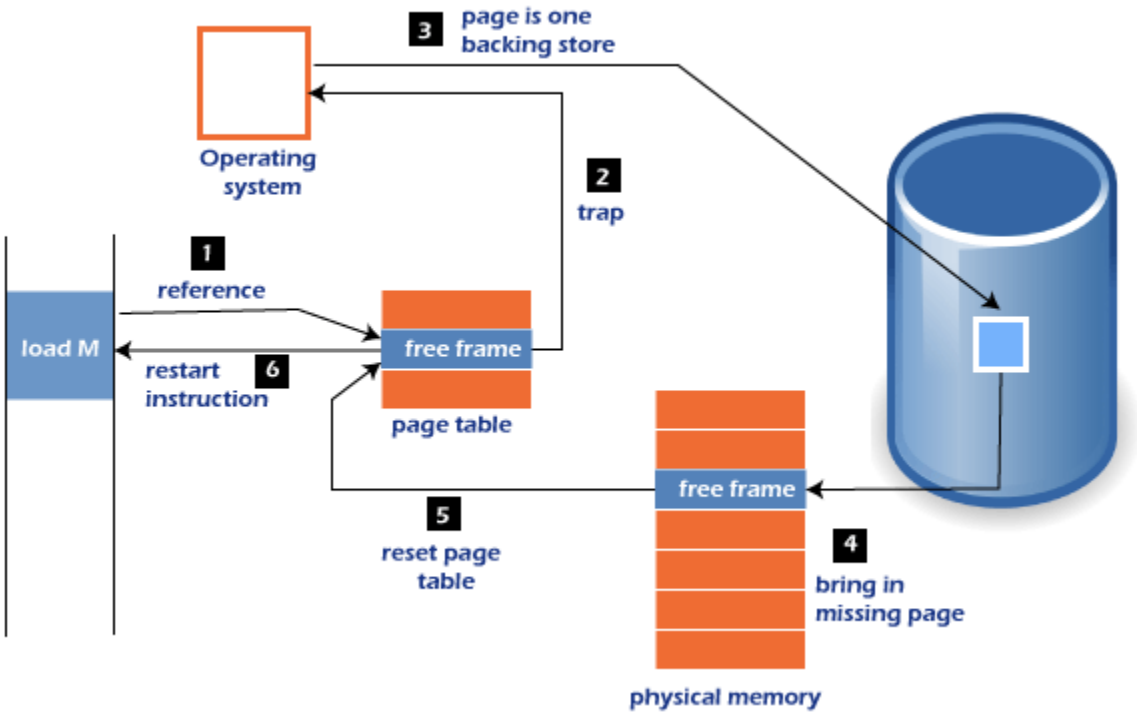
1. The degree of Multiprogramming will be increased.
2. User can run large application with less real RAM.
3. There is no need to buy more memory RAMs.

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.

Page Fault in Operating System?

Page faults dominate more like an **error**. A page fault will happen if a program tries to access a piece of memory that does not exist in physical memory (main memory). The fault specifies the operating system to trace all data into virtual memory management and then relocate it from secondary memory to its primary memory, such as a hard disk.



A page fault trap occurs if the requested page is not loaded into memory. The page fault primarily causes an exception, which is used to notify the operating system to retrieve the **"pages"** from virtual memory to continue operation. Once all of the data has been placed into physical memory, the program resumes normal operation. The Page fault process occurs in the background, and thus the user is unaware of it.

1. The computer's hardware track to the kernel and the program counter is often saved on the stack. The CPU registers hold information about the current state of instruction.
2. An assembly program is started, which saves the general registers and other volatile data to prevent the Operating system from destroying it.

Page Fault Handling

A Page Fault happens when you access a page that has been marked as invalid. The paging hardware would notice that the invalid bit is set while translating the address across the page table, which will cause an operating system trap. The trap is caused primarily by the OS's failure to load the needed page into memory.

Now, let's understand the procedure of page fault handling in the OS:

1. Firstly, an internal table for this process to assess whether the reference was valid or invalid memory access.
2. If the reference becomes invalid, the system process would be terminated. Otherwise, the page will be paged in.
3. After that, the free-frame list finds the free frame in the system.
4. Now, the disk operation would be scheduled to get the required page from the disk.
5. When the I/O operation is completed, the process's page table will be updated with a new frame number, and the invalid bit will be changed. Now, it is a valid page reference.
6. If any page fault is found, restart these steps from starting.

Why do page faults occur

Page faults occur when a program tries to access a page of memory that is not currently in RAM. This can happen due to various reasons, such as the page being swapped out to disk, the page not being allocated yet, or the page being paged out to make room for other pages.

Page Fault Terminology

There are various page fault terminologies in the operating system. Some terminologies of page fault are as follows:

1. Page Hit

When the CPU attempts to obtain a needed page from main memory and the page exists in **main memory (RAM)**, it is referred to as a **"PAGE HIT"**.

2. Page Miss

If the needed page has not existed in the **main memory (RAM)**, it is known as **"PAGE MISS"**.

3. Page Fault Time

The time it takes to get a page from secondary memory and recover it from the main memory after loading the required page is known as **"PAGE FAULT TIME"**.

4. Page Fault Delay

The rate at which threads locate page faults in memory is referred to as the **"PAGE FAULT RATE"**. The page fault rate is measured per second.

5. Hard Page Fault

If a required page exists in the hard disk's page file, it is referred to as a **"HARD PAGE FAULT"**.

6. Soft Page Fault

If a required page is not located on the hard disk but is found somewhere else in memory, it is referred to as a **"SOFT PAGE FAULT"**.

7. Minor Page Fault

If a process needs data and that data exists in memory but is being allotted to another process at the same moment, it is referred to as a **"MINOR PAGE FAULT"**.

Dirty bit/ Dirty Page

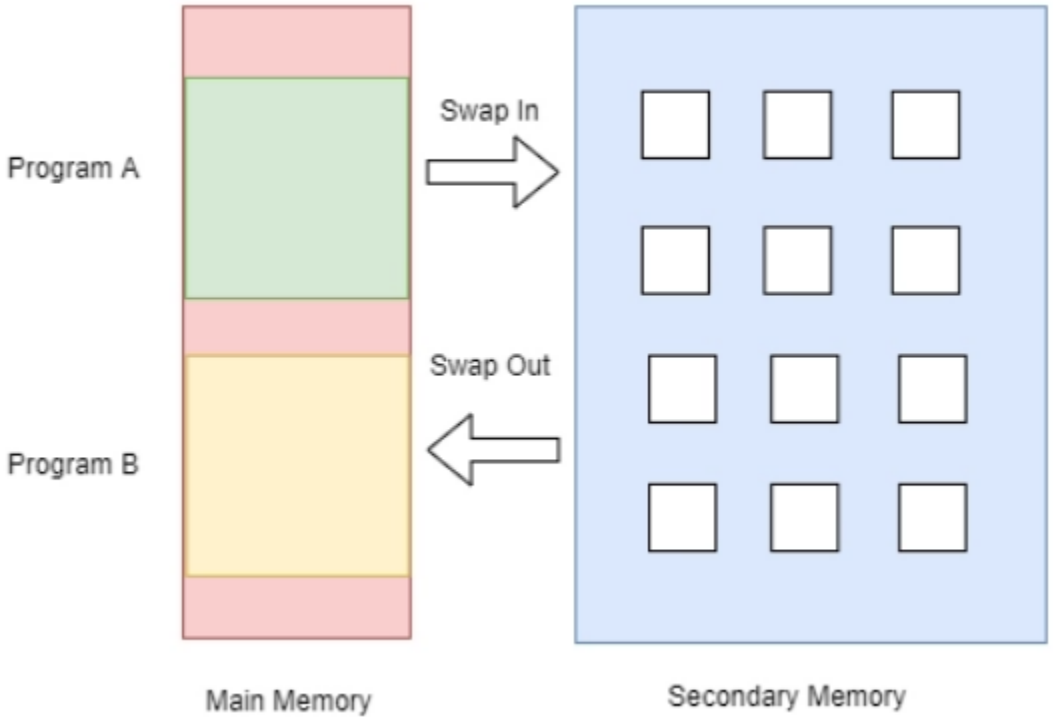
A dirty bit is a flag that indicates whether an attribute needs to be updated. Such situations usually occur when a bit in a memory cache or virtual memory page that has been changed by a processor but has not been updated in the storage.

How does the operating system handle a page fault

When a page fault occurs, the operating system typically follows a series of steps to handle it. First, it checks if the page being accessed is in the process's virtual memory. If it is not, the operating system tries to fetch the required page from secondary storage (e.g., hard disk) into RAM. Once the page is loaded, the operating system updates the page table to reflect its new location in memory and resumes the execution of the program that encountered the page fault.

Demand Paging in OS

The demand paging system is similar to the swapping paging system in that processes are mostly stored in the main memory (usually on the hard disk). As a result, demand paging is a procedure that addresses the problem above just by shifting pages on demand. Lazy swapper is another name for this (It never swaps the page into the memory unless it is needed).
A pager is a kind of swapper that deals with the individual pages of a process.
Demand Paging is a method in which a page is only brought into main memory when the CPU requests it. At first, just those pages are loaded directly required by the operation. Pages that are never accessed are thus never loaded into physical memory.



Why Demand Paging

There are a lot of pages in one particular process. But to put all the pages of that particular process in the memory will only consume the memory. Now for this problem, the concept of demand paging came into execution. Demand Paging is a method in which a page is only brought into main memory when the CPU requests it.
Let's understand this with the help of an example. Suppose the user did a program to calculate the overall cost of the fencing required for the field. For this, there will be different pages for different codes, like:
Page 1: The overall cost of the fence
Page 2: The labour cost
Page 3: The surface area of the field.
The user wants to calculate the overall cost of the fence. Then there is no need for loading all three pages, and this is where the demand paging comes into role. Then only Page 1 will be loaded, not all the pages.

How does Demand Paging in OS work?

When the CPU requests access to any page, the page table is used to find the page in the main memory. If the page is found on the main memory, it is good, and if it is not, then a page fault occurs.
Page fault is when the CPU wants to access the page from the main memory, but it is not present in the main memory. Then, how to overcome this?

For this swapped-in is used, the swapped-in is used to swap from the secondary memory. Swapped-in refers to moving a program back to the hard drive from the main memory, or RAM. However, if the page is already in the main memory, it is retrieved from there. The secondary memory is loaded with other pages. There are also valid and invalid bits, which are used to check whether the page is present in the main memory. Valid bits mean that the page is legal and present in the memory, and invalid bits mean the page is not valid or it is not present in the memory.

Common Terms in Demand Paging Operating System

Following are some common terms in demand paging operating systems:

- Page Fault
- Swapping
- Thrashing

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)

Common Algorithms used for Demand Paging in OS

The target for all algorithms is to reduce the number of page faults. Here are some common algorithms used for Demand Paging in OS:

1. **First In First Out (FIFO):** This algorithm is used to manage resources in operating systems like CPU time, memory, and input/output operations. The first process that enters the queue gets executed first, and the rest wait till they are executed. Here, algorithms are used to manage the queue to make sure that the processes are added and removed correctly. When a new process arrives, it is added to the end of the queue using the enqueue operation

2. **Optimal Page Algorithm:** As the name suggests, the Optimal page replacement algorithm is used to replace pages. This works by replacing the page whose demand in the future is the least as compared to other pages from the frame. Belady's Anomaly does not occur in this algorithm as it uses a stack based algorithm.

3. **Least Recently Used (LRU) Algorithm:** Whenever a page fault occurs, the least recently used page will be replaced with a new page. Hence the page not utilised for the longest time in the memory gets replaced. The Least Recently Used (LRU) Algorithm gives fewer page faults than any other algorithm and is capable of complete analysis.

4. **Page Buffering Algorithm:** This algorithm is used to get a process to start quickly and keep a pool of free frames.

5. **Least Frequently Used (LFU) Algorithm:** In this algorithm, the least frequently used pages are replaced. The page with the least visits in a given period of time is removed.The page that comes first is replaced first, given that the frequency of the page remains constant.
6. Second Chance (or Clock) Page Replacement Policy:- Apart from LRU, OPT and FIFO page replacement policies, we also have the second chance/clock page replacement policy. In the Second Chance page replacement policy, the candidate pages for removal are considered in a round robin matter, and a page that has been accessed between consecutive considerations will

not be replaced. The page replaced is the one that, when considered in a round robin matter, has not been accessed since its last consideration.
It can be implemented by adding a “second chance” bit to each memory frame-every time the frame is considered (due to a reference made to the page inside it), this bit is set to 1, which gives the page a second chance, as when we consider the candidate page for replacement, we replace the first one with this bit set to 0 (while zeroing out bits of the other pages we see in the process). Thus, a page with the “second chance” bit set to 1 is never replaced during the first consideration and will only be replaced if all the other pages deserve a second chance too!

Example –
Let’s say the reference string is **0 4 1 4 2 4 3 4 2 4 0 4 1 4 2 4 3 4** and we have **3** frames. Let’s see how the algorithm proceeds by tracking the second chance bit and the pointer.

- Initially, all frames are empty so after first 3 passes they will be filled with {0, 4, 1} and the second chance array will be {0, 0, 0} as none has been referenced yet. Also, the pointer will cycle back to 0.

- **Pass-4:** Frame={0, 4, 1}, second_chance = {0, 1, 0} [4 will get a second chance], pointer = 0 (No page needed to be updated so the candidate is still page in frame 0), pf = 3 (No increase in page fault number).

- **Pass-5:** Frame={2, 4, 1}, second_chance= {0, 1, 0} [0 replaced; it’s second chance bit was 0, so it didn’t get a second chance], pointer=1 (updated), pf=4

- **Pass-6:** Frame={2, 4, 1}, second_chance={0, 1, 0}, pointer=1, pf=4 (No change)

- **Pass-7:** Frame={2, 4, 3}, second_chance= {0, 0, 0} [4 survived but it’s second chance bit became 0], pointer=0 (as element at index 2 was finally replaced), pf=5

- **Pass-8:** Frame={2, 4, 3}, second_chance= {0, 1, 0} [4 referenced again], pointer=0, pf=5

- **Pass-9:** Frame={2, 4, 3}, second_chance= {1, 1, 0} [2 referenced again], pointer=0, pf=5
- **Pass-10:** Frame={2, 4, 3}, second_chance= {1, 1, 0}, pointer=0, pf=5 (no change)
- **Pass-11:** Frame={2, 4, 0}, second_chance= {0, 0, 0}, pointer=0, pf=6 (2 and 4 got second chances)
- **Pass-12:** Frame={2, 4, 0}, second_chance= {0, 1, 0}, pointer=0, pf=6 (4 will again get a second chance)
- **Pass-13:** Frame={1, 4, 0}, second_chance= {0, 1, 0}, pointer=1, pf=7 (pointer updated, pf updated)
- **Page-14:** Frame={1, 4, 0}, second_chance= {0, 1, 0}, pointer=1, pf=7 (No change)
- **Page-15:** Frame={1, 4, 2}, second_chance= {0, 0, 0}, pointer=0, pf=8 (4 survived again due to 2nd chance!)
- **Page-16:** Frame={1, 4, 2}, second_chance= {0, 1, 0}, pointer=0, pf=8 (2nd chance updated)
- **Page-17:** Frame={3, 4, 2}, second_chance= {0, 1, 0}, pointer=1, pf=9 (pointer, pf updated)
- **Page-18:** Frame={3, 4, 2}, second_chance= {0, 1, 0}, pointer=1, pf=9 (No change)

Advantages of Demand Paging in OS

Demand paging has the following benefits:

- 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 in OS

Demand paging has the following drawbacks:

- 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