Unit-5

Memory Management

Introduction

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.

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

Memory Hierarchy

The memory in a computer can be divided into five hierarchies based on the speed as well as use. The processor can move from one level to another based on its requirements. The five hierarchies in the memory are registers, cache, main memory, magnetic discs, and magnetic tapes. The first three hierarchies are volatile memories which mean when there is no power, and then automatically they lose their stored data. Whereas the last two hierarchies are not volatile which means they store the data permanently. A memory element is the set of storage devices which stores the binary data in the type of bits. In general, the storage of memory can be classified into two categories such as volatile as well as non- volatile.

The memory hierarchy design in a computer system mainly includes different storage devices. Most of the computers were inbuilt with extra storage to run more powerfully beyond the main memory capacity. The following memory hierarchy diagram is a hierarchical pyramid for computer memory. The designing of the memory hierarchy is divided into two types such as primary (Internal) memory and secondary (External) memory.

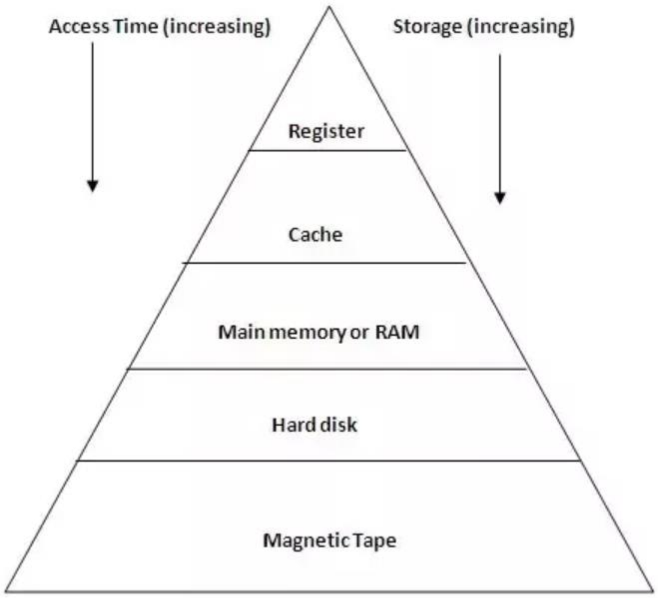


Fig: Memory Hierarchy

Registers

Usually, the register is a static RAM or SRAM in the processor of the computer which is used for holding the data word which is typically 64 or 128 bits. The program counter register is the most important as well as found in all the processors. Most of the processors use a status word register as well as an accumulator. A status word register is used for decision making, and the accumulator is used to store the data like mathematical operation. Usually, computers like complex instruction set computers have so many registers for accepting main memory, and RISC- reduced instruction set computers have more registers.

Cache Memory

Cache memory can also be found in the processor, however rarely it may be another IC (integrated circuit) which is separated into levels. The cache holds the chunk of data which are frequently used from main memory. When the processor has a single core then it will have two (or) more cache levels rarely. Present multi-core processors will be having three, 2-levels for each one core, and one level is shared.

Main Memory

The main memory in the computer is nothing but, the memory unit in the CPU that communicates directly. It is the main storage unit of the computer. This memory is fast as well as large memory used for storing the data throughout the operations of the computer. This memory is made up of RAM as well as ROM.

Magnetic Disks

The magnetic disks in the computer are circular plates fabricated of plastic otherwise metal by magnetized material. Frequently, two faces of the disk are utilized as well as many disks may be stacked on one spindle by read or write heads obtainable on every plane. All the disks in computer turn jointly at high speed. The tracks in the computer are nothing but bits which are stored within the magnetized plane in spots next to concentric circles. These are usually separated into sections which are named as sectors.

Magnetic Tape

This tape is a normal magnetic recording which is designed with a slender magnetizable covering on an extended, plastic film of the thin strip. This is mainly used to back up huge data. Whenever the computer requires to access a strip, first it will mount to access the data. Once the data is allowed, then it will be unmounted. The access time of memory will be slower within magnetic strip as well as it will take a few minutes for accessing a strip.

Advantages of Memory Hierarchy

The need for a memory hierarchy includes the following.

- Memory distributing is simple and economical.
- Removes external destruction.
- Data can be spread all over.
- Permits demand paging & pre-paging.
- Swapping will be more proficient.

Logical versus and Physical Address Space

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.

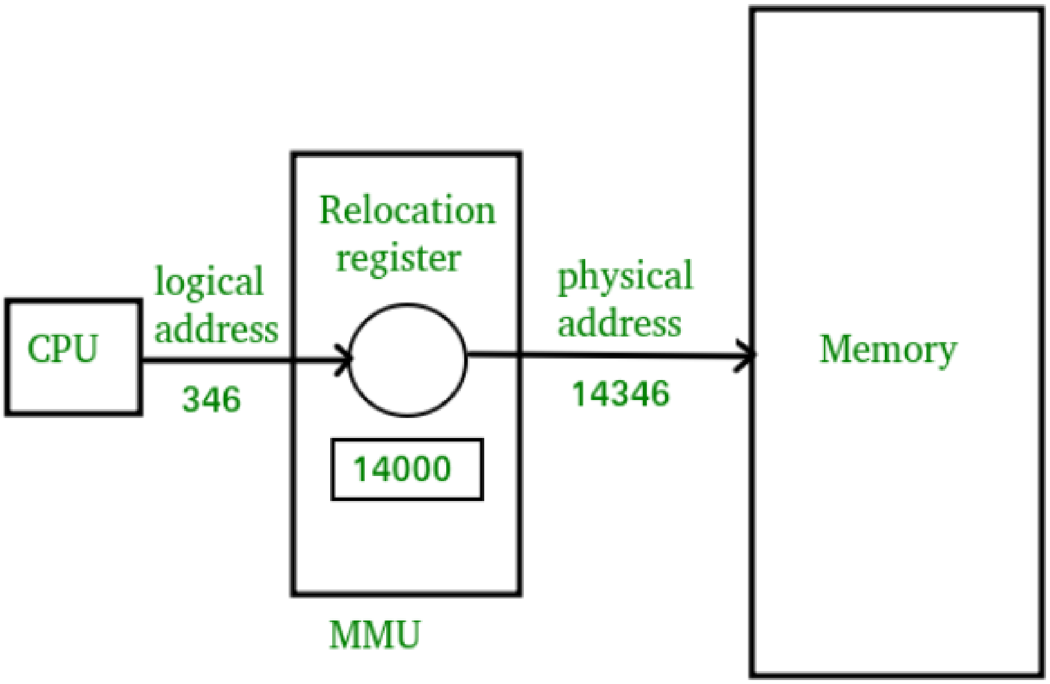


Fig: Mapping virtual address to physical address

Logical Address	Physical Address
1. It is the virtual address generated by CPU.	1. The physical address is a location in a memory unit.
2. Set of all logical addresses generated by CPU in reference to a program is referred as Logical Address Space.	2. Set of all physical addresses mapped to the corresponding logical addresses is referred as Physical Address.
3. The user can view the logical address of a program.	3. The user can never view physical address of program
4. The user uses the logical address to access the physical address.	4. The user can not directly access physical address.
5. The Logical Address is generated by the CPU	5. Physical Address is Computed by MMU.

Memory Management with Swapping

Swapping is a memory management scheme in which any process can be temporarily swapped from main memory to secondary memory so that the main memory can be made available for other processes. It is used to improve main memory utilization. In secondary memory, the place where the swapped-out process is stored is called swap space. The purpose of the swapping in operating system is to access the data present in the hard disk and bring it to RAM so that the application programs can use it. The thing to remember is that swapping is used only when data is not present in RAM. Although the process of swapping affects the performance of the system, it helps to run larger and more than one process. This is the reason why swapping is also referred to as memory compaction.

The concept of swapping has divided into two more concepts: Swap-in and Swap-out.

Swap-out is a method of removing a process from RAM and adding it to the hard disk.

Swap-in is a method of removing a program from a hard disk and putting it back into the main memory or RAM.

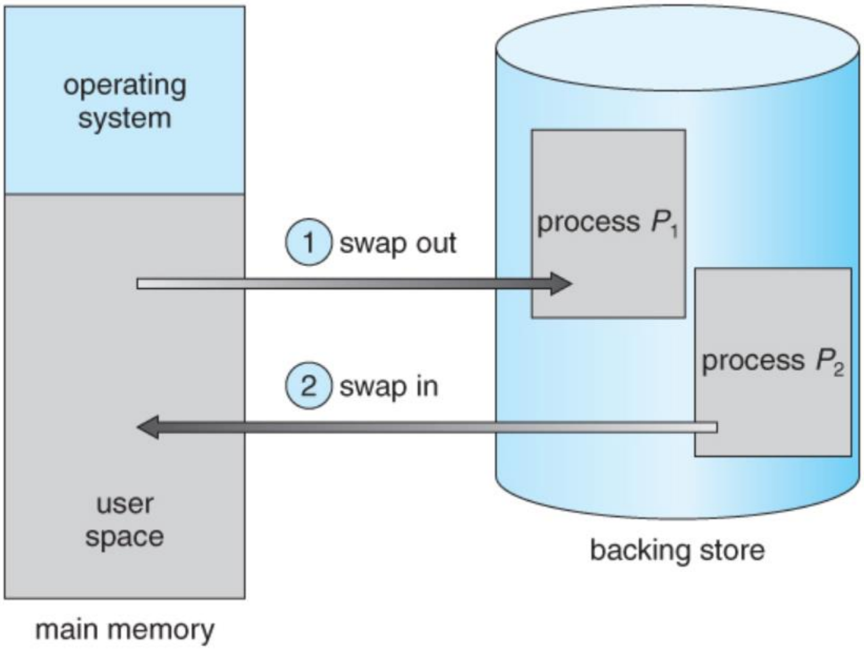
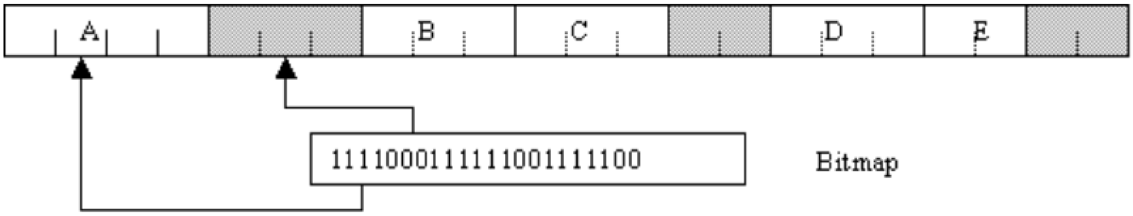


Fig: Swapping of two processes using a disk as a backing store

Memory Management with Bitmaps

The bitmap approach divides the memory space into small fixed sized allocation units. These units might be a few bytes/words in size or perhaps several kilobytes. The bitmap contains a single bit for each allocation unit which indicates the status of that unit, 1 for used, 0 for unused. The greater the number of allocation units, the larger the bit map. The size of memory and the size of allocation units determine the size of the bitmap. The size of the allocation unit can be an important design consideration. If the allocation unit is too large, the bit map will be smaller, but we will only be able to allocate memory space in multiples of large allocation units, so it is likely that some of the space within an allocated unit might not be required by a process.



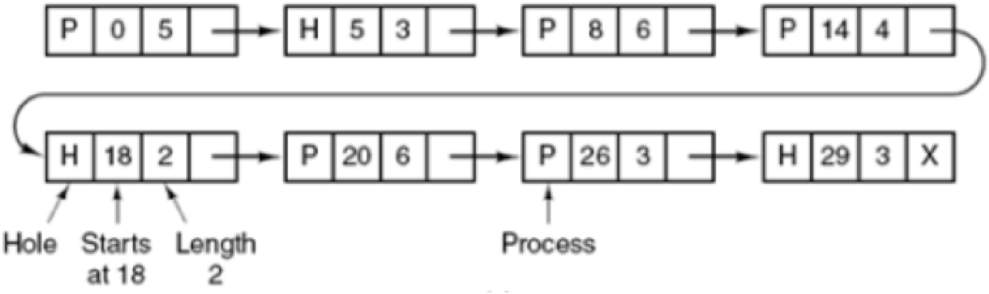
The diagram shows five processes in memory with three (shaded) free holes corresponding to 0 values in the bitmap. Initially, the memory is empty and the bitmap contains all 0 entries. To allocate a region of memory, the memory manager needs to search the bitmap for a string of 0's which represents a region large enough to accommodate the request. If the memory is large and the allocation units are relatively small, then searching the bitmap string maybe time consuming. To deallocate a region, the memory manager simply sets the appropriate bits in the bitmap back to 0.

Disadvantages of using Bitmap

- The OS has to assign some memory for bitmap as well since it stores the details about allocation units. That much amount of memory cannot be used to load any process therefore that decreases the degree of multiprogramming as well as throughput.
- To identify any hole in the memory, the OS need to search the string of 0s in the bitmap. This searching takes a huge amount of time which makes the system inefficient to some extent.

Memory Management with Linked Lists

Another way of keeping track of memory is to maintain a linked list of allocated and free memory segments, where a segment is either a process or a hole between two processes. Each entry in the list specifies a hole (H) or process (P), the address at which it starts, the length, and a pointer to the next entry. In this example, the segment list is kept sorted by address. Sorting this way has the advantage that when a process terminates or is swapped out, updating the list is straightforward. A terminating process normally has two neighbors (except when it is at the very top or very bottom of memory). These may be either processes or holes, leading to the four combinations shown in fig above.



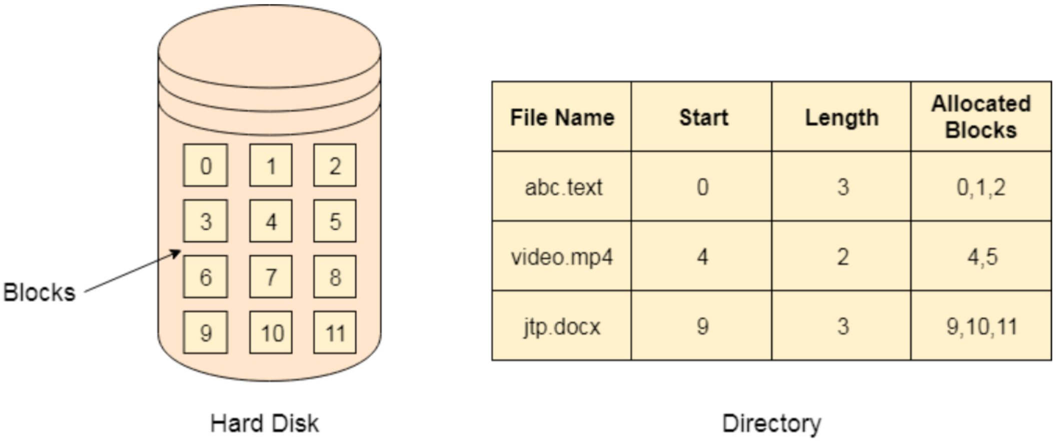
When the processes and holes are kept on a list sorted by address, several algorithms can be used to allocate memory for a newly created process (or an existing process being swapped in from disk). We assume that the memory manager knows how much memory to allocate.

Memory Management without Swapping

Memory management is the process of controlling and coordinating computer memory, assigning portions known as blocks to various running program to optimize the overall performance of the system. Here entire process remains in memory from start to finish and does not move. The sum of the memory requirements of all jobs in the system cannot exceed the size of physical memory.

Contiguous Memory Allocation

Contiguous memory allocation is basically a method in which a single contiguous section/part of memory is allocated to a process or file needing it. In the image shown below, there are three files in the directory. The starting block and the length of each file are mentioned in the table. We can check in the table that the contiguous blocks are assigned to each file as per its need.



File Name	Start	Length	Allocated Blocks
abc.text	0	3	0,1,2
video.mp4	4	2	4,5
jtp.docx	9	3	9,10,11

Fig: Contiguous Allocation

Advantages

1. It is simple to implement.
2. We will get Excellent read performance.
3. Supports Random Access into files.

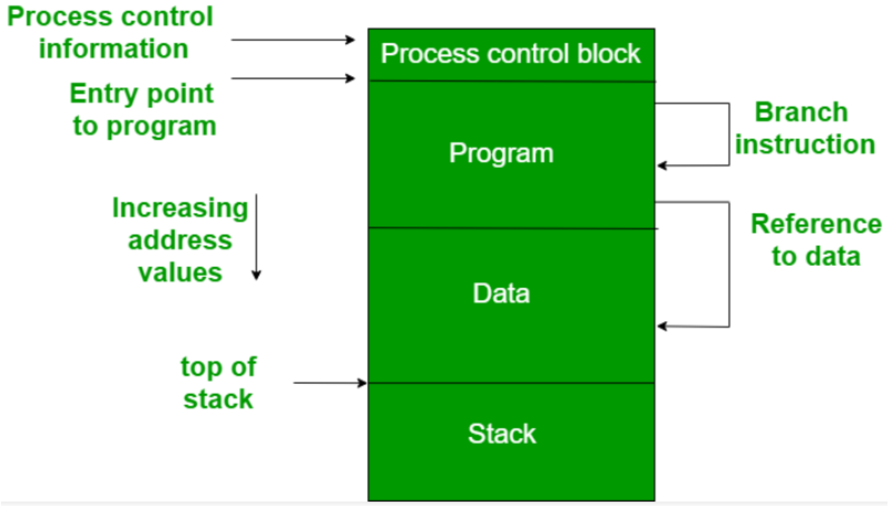
Disadvantages

1. The disk will become fragmented.

2. It may be difficult to have a file grow.

Relocation

The available memory is generally shared among a number of processes in a multiprogramming system, so it is not possible to know in advance which other programs will be resident in main memory at the time of execution of his program. Swapping the active processes in and out of the main memory enables the operating system to have a larger pool of ready-to-execute process. When a program gets swapped out to a disk memory, then it is not always possible that when it is swapped back into main memory then it occupies the previous memory location, since the location may still be occupied by another process. We may need to **relocate** the process to a different area of memory. Thus there is a possibility that program may be moved in main memory due to swapping.



The figure depicts a process image. The process image is occupying a continuous region of main memory. The operating system will need to know many things including the location of process control information, the execution stack, and the code entry. Within a program, there are memory references in various instructions and these are called logical addresses. After loading of the program into main memory, the processor and the operating system must be able to translate logical addresses into physical addresses. Branch instructions contain the address of the next instruction to be executed. Data reference instructions contain the address of byte or word of data referenced.

Memory Protection

Memory protection is a way to control memory access rights on a computer, and is a part of most modern instruction set architectures and operating systems. The main purpose of memory protection is to prevent a process from accessing memory that has not been allocated to it. This prevents a bug or malware within a process from affecting other processes, or the operating system itself. Protection may encompass all accesses to a specified area of memory, write accesses, or attempts to execute the contents of the area. An attempt to access unowned memory results in a hardware fault, called a segmentation fault or storage violation exception, generally causing abnormal termination of the offending process. Memory protection for computer security includes additional techniques such as address space layout randomization and executable space protection.

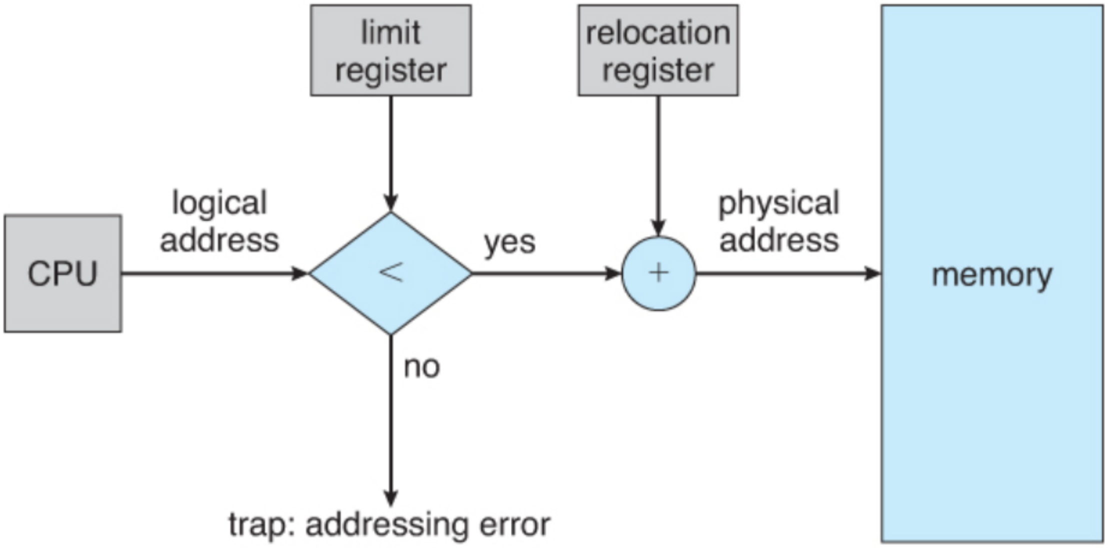


Fig: Memory protection

Memory Allocation

Memory allocation is an action of assigning the physical or the virtual memory address space to a process (its instructions and data). The two fundamental methods of memory allocation are static and dynamic memory allocation.

Static memory allocation method assigns the memory to a process, before its execution. On the other hand, the dynamic memory allocation method assigns the memory to a process, during its execution.

To get a process executed it must be first placed in the memory. Assigning space to a process in memory is called memory allocation. Memory allocation is a general aspect of the term binding.

Let us understand binding with the help of an example. Suppose, there is an entity in a program, with the set of attributes. Now, a variable of this entity will have values for these set of attributes. For storing these values, we must have memory allotted to these attributes.

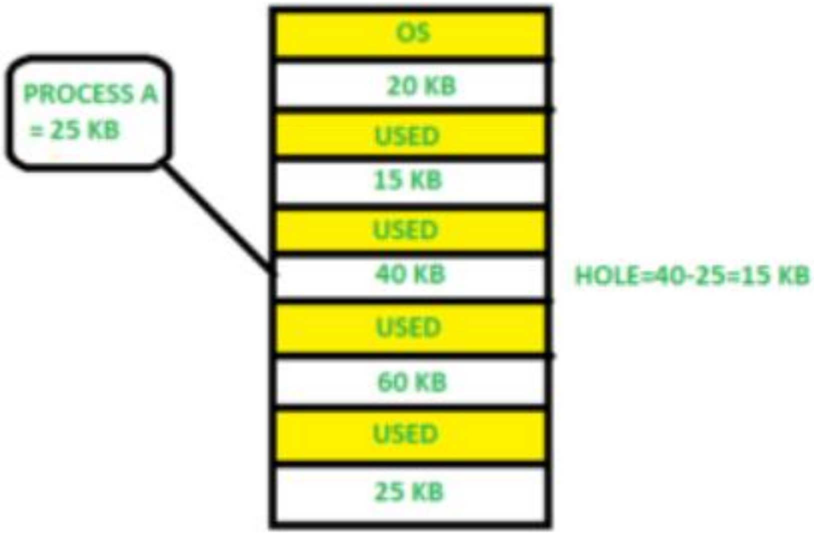
So, the act of assigning the memory address to the attribute of the variable is called memory allocation. And the act of specifying/binding the values to the attributes of the variable is called binding. This action of binding must be performed before the variable is used during the execution of the program.

Different allocation algorithms are:

a. First fit
b. Next fit
c. Best fit
d. Worst fit
e. Quick fit

a. First fit

In the first fit, the partition is allocated which is the first sufficient block from the top of Main Memory. It scans memory from the beginning and chooses the first available block that is large enough. Thus it allocates the first hole that is large enough.



Advantage

Fastest algorithm because it searches as little as possible.

Disadvantage

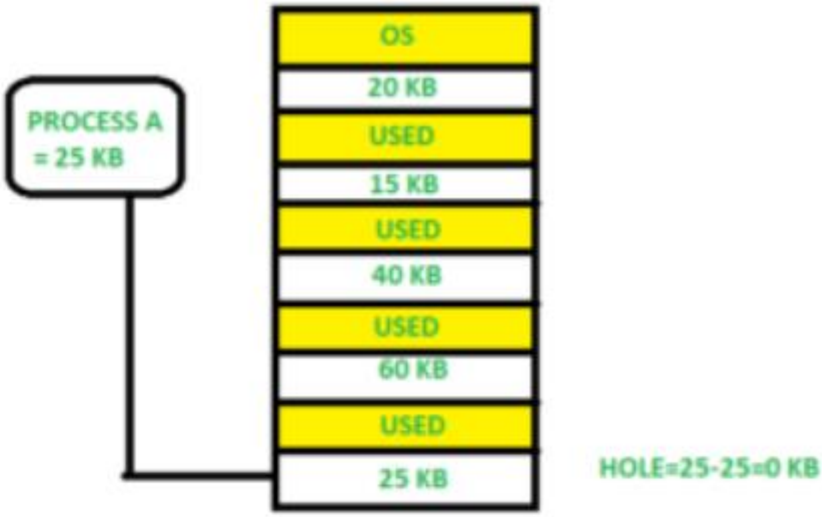
The remaining unused memory areas left after allocation become waste if it is too smaller. Thus request for larger memory requirement cannot be accomplished.

b. Next fit

Next fit is a modified version of first fit. It begins as first fit to find a free partition. When called next time it starts searching from where it left off, not from the beginning.

c. Best fit

Allocate the process to the partition which is the first smallest sufficient partition among the free available partition. It searches the entire list of holes to find the smallest hole whose size is greater than or equal to the size of the process.



Advantage

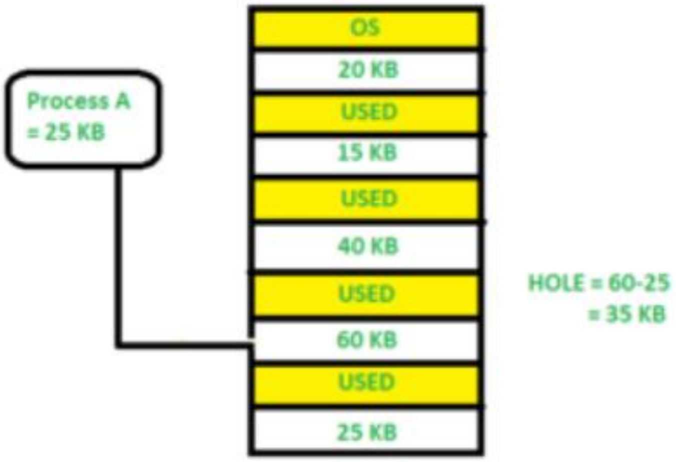
Memory utilization is much better than first fit as it searches the smallest free partition first available.

Disadvantage

It is slower and may even tend to fill up memory with tiny useless holes.

d. Worst fit

Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory. It is opposite to the best-fit algorithm. It searches the entire list of holes to find the largest hole and allocate it to process.



Advantage

Reduces the rate of production of small gaps.

Disadvantage

If a process requiring larger memory arrives at a later stage then it cannot be accommodated as the largest hole is already split and occupied.

e. Quick fit

The quick fit algorithm suggests maintaining the different lists of frequently used sizes. Although, it is not practically suggestible because the procedure takes so much time to create the different lists and then expending the holes to load a process.

Exercise

Given memory partitions of 100k, 500k, 200k, 300k and 600k (in order), how would each of the First-fit, Best-fit and Worst-fit algorithms place processes of 212k, 417k, 112k and 426k (in order)? Which algorithm makes the most efficient use of memory?

Solution

First-fit

100K	500K	200K	300K	600K

Primary memory

1. 212K is put in 500k partition

100K	212k	▨	200K	300K	600K

2. 417 is put in 600k partition

100K	212k	▨	200K	300K	417k	▨

3. 112k is put in 288k partition (new partition 500k – 212k = 288k)

100K	212k	112k	▨	200K	300K	417k	▨

4. 426k must wait

Best-fit

100K	500K	200K	300K	600K

Primary memory

1. 212K is put in 300k partition

100K	500k	200K	212K	▨	600K

2. 417 is put in 500k partition

100K	417k	▨	200K	212K	▨	600K

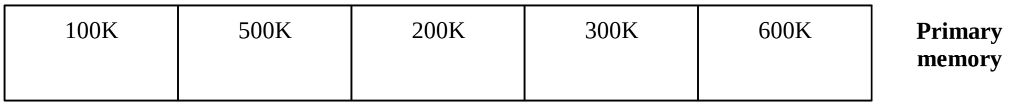
3. 112k is put in 200k partition

100K	417k	▨	112K	▨	212K	▨	600K

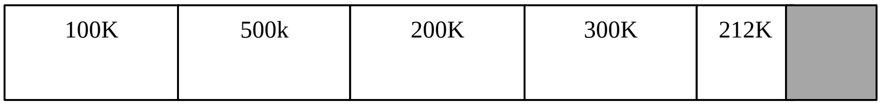
4. 426k is put in 600k partition

100K	417k	▨	112K	▨	212K	▨	426K	▨

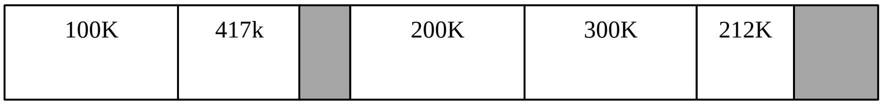
Worst-fit



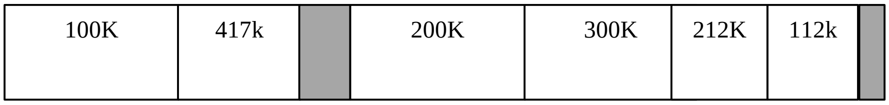
1. 212K is put in 600k partition



2. 417 is put in 500k partition



3. 112k is put in 388k partition (new partition 600k – 212k = 388k)



4. 426k must wait

In this example, Best-fit turns out to be the best.

Fragmentation

Fragmentation is an unwanted problem where the memory blocks cannot be allocated to the processes due to their small size and the blocks remain unused. It can also be understood as when the processes are loaded and removed from the memory they create free space or hole in the memory and these small blocks cannot be allocated to new upcoming processes and results in inefficient use of memory. Basically, there are two types of fragmentation:

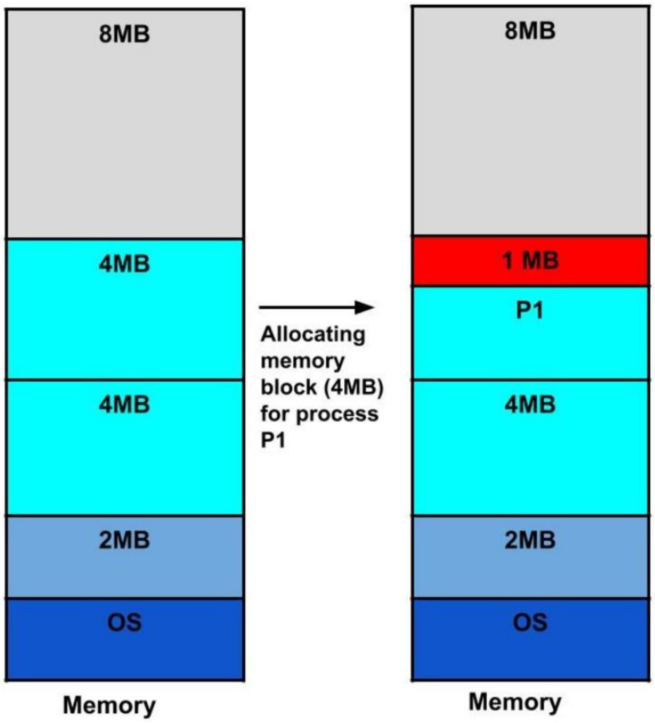
- Internal Fragmentation
- External Fragmentation

Internal fragmentation

In this fragmentation, the process is allocated a memory block of size more than the size of that process. Due to this some part of the memory is left unused and this cause internal fragmentation.

Example: Suppose there is fixed partitioning (i.e. the memory blocks are of fixed sizes) is used for memory allocation in RAM. These sizes are 2MB, 4MB, 4MB, 8MB. Some part of this RAM is occupied by the Operating System (OS).

Now, suppose a process P1 of size 3MB comes and it gets memory block of size 4MB. So, the 1MB that is free in this block is wasted and this space can't be utilized for allocating memory to some other process. This is called internal fragmentation.



How can we remove internal fragmentation?

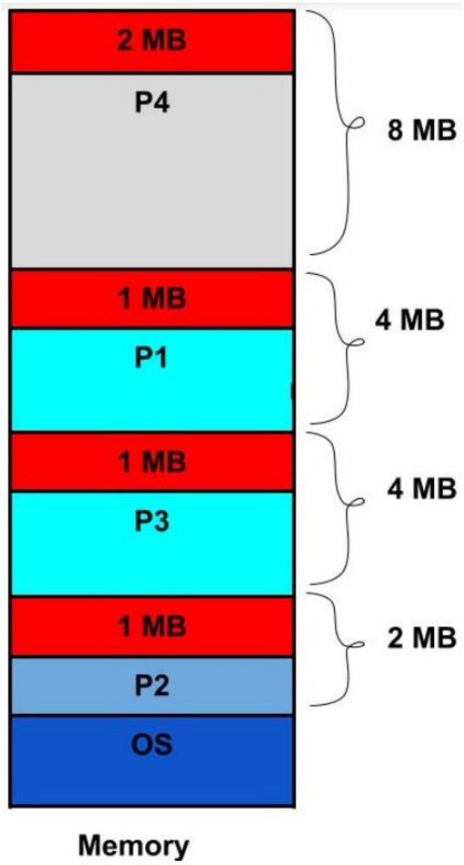
This problem is occurring because we have fixed the sizes of the memory blocks. This problem can be removed if we use dynamic partitioning for allocating space to the process. In dynamic partitioning, the process is allocated only that much amount of space which is required by the process. So, there is no internal fragmentation.

External fragmentation

In this fragmentation, although we have total space available that is needed by a process still we are not able to put that process in the memory because that space is not contiguous. This is called external fragmentation.

Example: Suppose in the above example, if three new processes P2, P3, and P4 come of sizes 1MB, 3MB, and 6MB respectively. Now, these processes get memory blocks of size 2MB, 4MB and 8MB respectively allocated.

So, now if we closely analyze this situation then process P3 (unused 1MB) and P4 (unused 2MB) are again causing internal fragmentation. So, a total of 4MB (1MB (due to process P1) + 1MB (due to process P3) + 2MB (due to process P4)) is unused due to internal fragmentation.



Now, suppose a new process of 4MB comes. Though we have a total space of 4MB still we can't allocate this memory to the process. This is called external fragmentation.

How can we remove external fragmentation?

This problem is occurring because **we are allocating memory continuously** to the processes. So, if we remove this condition external fragmentation can be reduced. This is what done in **paging & segmentation**(non-contiguous memory allocation techniques) where memory is allocated non-contiguously to the processes. We will learn about paging and segmentation in the next blog.
Another way to remove external fragmentation is **compaction**. When dynamic partitioning is used for memory allocation then external fragmentation can be reduced by **merging all the free**

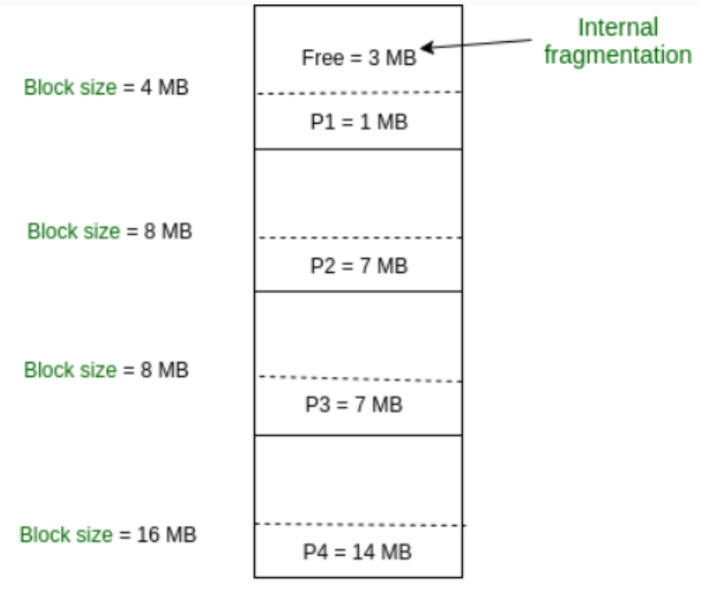
memory together in one large block. This technique is also called **defragmentation.** This larger block of memory is then used for allocating space according to the needs of the new processes.

Difference between Internal fragmentation and External fragmentation

Internal fragmentation	**External fragmentation**
1. In internal fragmentation fixed-sized memory, blocks square measure appointed to process.	1. In external fragmentation, variable-sized memory blocks square measure appointed to method.
2. Internal fragmentation happens when the method or process is larger than the memory.	2. External fragmentation happens when the method or process is removed.
3. The solution of internal fragmentation is best-fit block.	3. Solution of external fragmentation is compaction, paging and segmentation
4. Internal fragmentation occurs when memory is divided into fixed sized partitions.	4. External fragmentation occurs when memory is divided into variable size partitions based on the size of processes.
5. The difference between memory allocated and required space or memory is called Internal fragmentation.	5. The unused spaces formed between non-contiguous memory fragments are too small to serve a new process, is called External fragmentation .

Fixed (static) partitioning

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



Fixed size partition

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

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

There are some advantages and disadvantages of fixed partitioning.

Advantages of Fixed Partitioning –

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

Disadvantages of Fixed Partitioning –

1. **Internal Fragmentation:**
   Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This can cause internal fragmentation.
2. **External Fragmentation:**
   The total unused space (as stated above) of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form (as spanning is not allowed).
3. **Limit process size:**
   Process of size greater than size of partition in Main Memory cannot be accommodated.

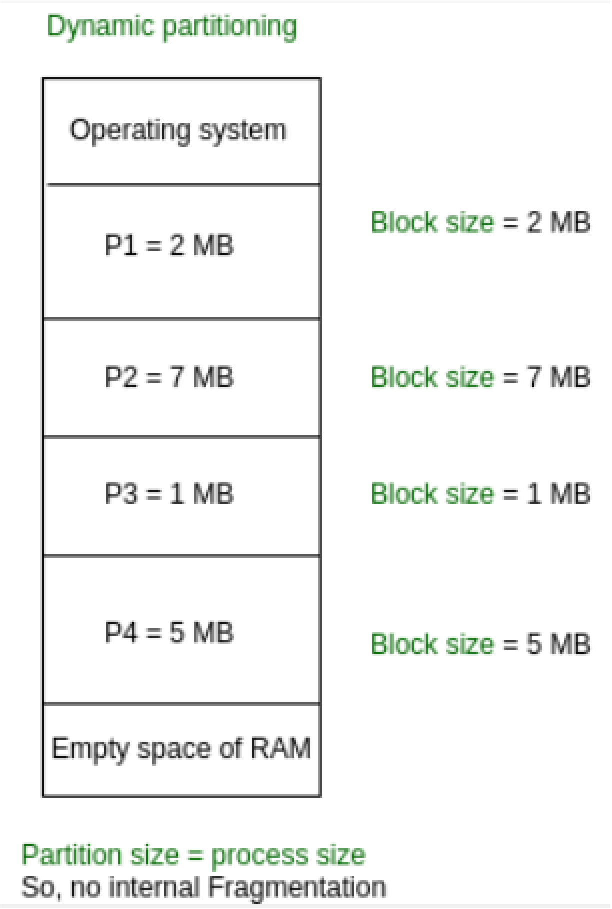
Partition size cannot be varied according to the size of incoming process’s size. Hence, process size of 32MB in above stated example is invalid.

4. **Limitation on Degree of Multiprogramming:**
   Partition in Main Memory are made before execution or during system configure. Main Memory is divided into fixed number of partition. Suppose if there are n1 partitions in RAM and n2 are the number of processes, then n2<=n1 condition must be fulfilled. Number of processes greater than number of partitions in RAM is invalid in Fixed Partitioning.

Variable (Dynamic) partitioning

It is a part of Contiguous allocation technique. It is used to alleviate the problem faced by Fixed Partitioning. In contrast with fixed partitioning, partitions are not made before the execution or during system configure. Various **features** associated with variable Partitioning-

1. Initially RAM is empty and partitions are made during the run-time according to process’s need instead of partitioning during system configure.
2. The size of partition will be equal to incoming process.
3. The partition size varies according to the need of the process so that the internal fragmentation can be avoided to ensure efficient utilisation of RAM.
4. Number of partitions in RAM is not fixed and depends on the number of incoming process and Main Memory’s size.



Advantages of Variable Partitioning –

1. **No Internal Fragmentation:**
   In variable Partitioning, space in main memory is allocated strictly according to the need of process, hence there is no case of internal fragmentation. There will be no unused space left in the partition.
2. **No restriction on Degree of Multiprogramming:**
   More number of processes can be accommodated due to absence of internal fragmentation. A process can be loaded until the memory is empty.
3. **No Limitation on the size of the process:**
   In Fixed partitioning, the process with the size greater than the size of the largest partition could not be loaded and process can not be divided as it is invalid in contiguous allocation technique. Here, In variable partitioning, the process size can't be restricted since the partition size is decided according to the process size.

Disadvantages of Variable Partitioning –

1. **Difficult Implementation:**
   Implementing variable Partitioning is difficult as compared to Fixed Partitioning as it involves allocation of memory during run-time rather than during system configure.
2. **External Fragmentation:**
   There will be external fragmentation inspite of absence of internal fragmentation.
   For example, suppose in above example- process P1(2MB) and process P3(1MB) completed their execution. Hence two spaces are left i.e. 2MB and 1MB. Let's suppose process P5 of size 3MB comes. The empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation. The rule says that process must be contiguously present in main memory to get executed. Hence it results in External Fragmentation.

Non-contiguous memory allocation

The Non-contiguous memory allocation allows a process to acquire the several memory blocks at the different location in the memory according to its requirement. The non-contiguous memory allocation also reduces the memory wastage caused due to internal and external fragmentation. As it utilizes the memory holes, created during internal and external fragmentation.

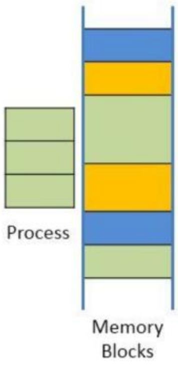


Fig: Non-contiguous memory allocation

Non-contiguous memory allocation is of different types,

1. Paging
2. Segmentation
3. Segmentation with paging

i) Paging

A non-contiguous policy with a fixed size partition is called paging. A computer can address more memory than the amount of physically installed on the system. This extra memory is actually called virtual memory. Paging technique is very important in implementing virtual memory. Secondary memory is divided into equal size partition (fixed) called pages. Every process will have a separate page table. The entries in the page table are the number of pages a process. At each entry either we have an invalid pointer which means the page is not in main memory or we will get the corresponding frame number. When the frame number is combined with instruction of set D than we will get the corresponding physical address. Size of a page table is generally very large so cannot be accommodated inside the PCB, therefore, PCB contains a register value PTBR(page table base register) which leads to the page table.

Advantages: It is independent of external fragmentation.

Disadvantages:

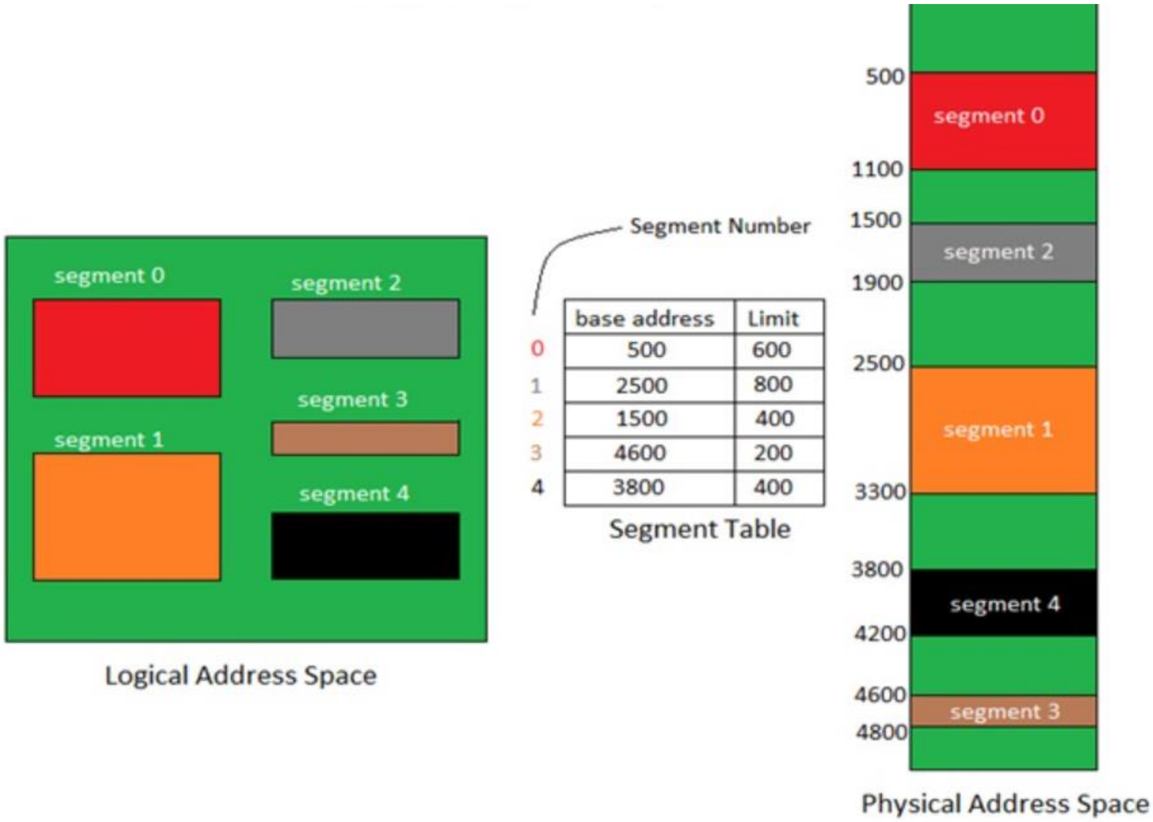
1. It makes the translation very slow as main memory access two times.
2. A page table is a burden over the system which occupies considerable space.

ii) Segmentation

Segmentation is a programmer view of the memory where instead of dividing a process into equal size partition we divided according to program into partition called segments. The translation is the same as paging but paging segmentation is independent of internal fragmentation but suffers from external fragmentation. Reason of external fragmentation is program can be divided into segments but segment must be contiguous in nature.

iii) Segmentation with paging

In segmentation with paging, we take advantages of both segmentation as well as paging. It is a kind of multilevel paging but in multilevel paging, we divide a page table into equal size partition but here in segmentation with paging, we divide it according to segments. All the properties are the same as that of paging because segments are divided into pages



	base address	Limit
0	500	600
1	2500	800
2	1500	400
3	4600	200
4	3800	400

Difference between contiguous and non-contiguous memory allocation

Contiguous memory allocation	Non-contiguous memory allocation
1. Contiguous memory allocation allocates consecutive blocks of memory to a file/process.	1. Non-Contiguous memory allocation allocates separate blocks of memory to a file/process.
2. Faster in Execution	2. Slower in Execution
3. It is easier for the OS to control.	3. It is difficult for the OS to control.
4. Overhead is minimum as not much address translations are there while executing a process.	4. More Overheads are there as there are more address translations.

5. Internal fragmentation occurs in Contiguous memory allocation method.	5. External fragmentation occurs in Non-Contiguous memory allocation method.
6. It includes single partition allocation and multi-partition allocation.	6. It includes paging and segmentation.
7. Wastage of memory is there.	7. No memory wastage is there
8. In contiguous memory allocation, swapped-in processes are arranged in the originally allocated space	8. In non-contiguous memory allocation, swapped-in processes can be arranged in any place in the memory.

Coalescing and Compaction

The process of merging two adjacent holes to form a single larger hole is called **coalescing**.

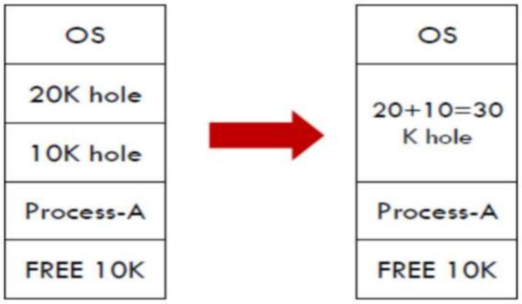


Figure: coalescing

Even when holes are coalesced, no individual hole may be large enough to hold the job, although the sum of holes is larger than the storage required for a process. It is possible to combine all the holes into one big one by moving all the processes downward as far as possible; this technique is called **memory 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.

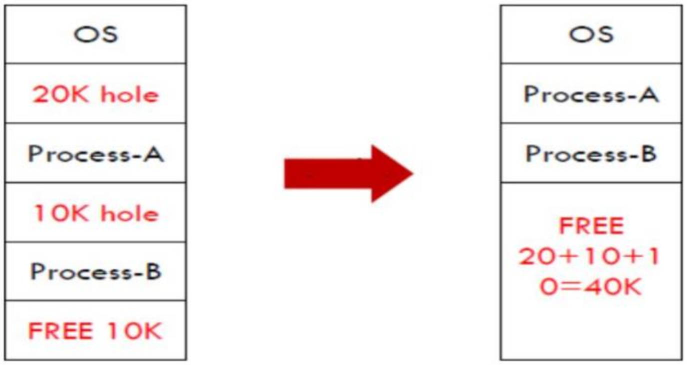


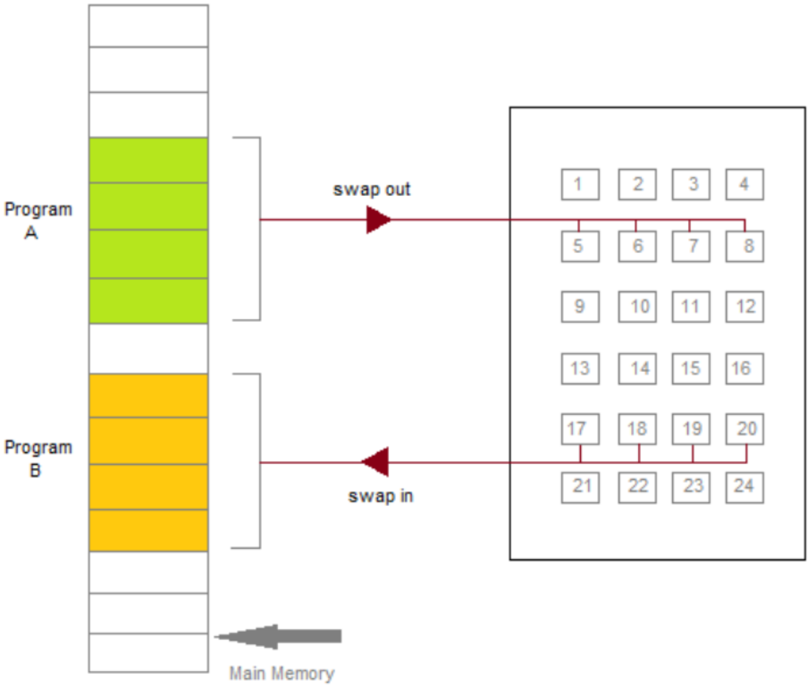
Fig: Compaction

Virtual Memory

Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there. 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.

Benefits of having Virtual Memory

1. Large programs can be written, as virtual space available is huge compared to physical memory.
2. Less I/O required, leads to faster and easy swapping of processes.
3. More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.

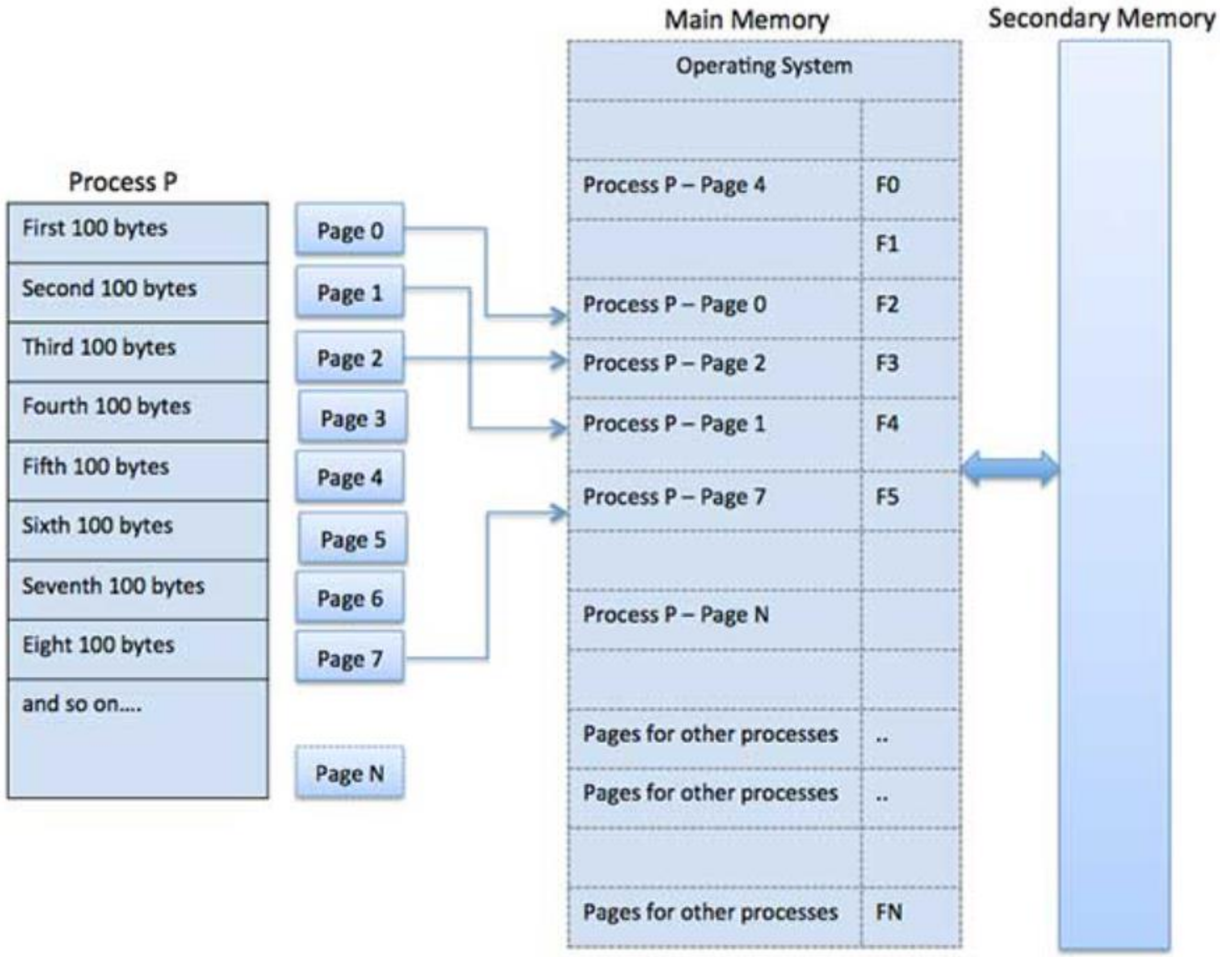


Paging

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.



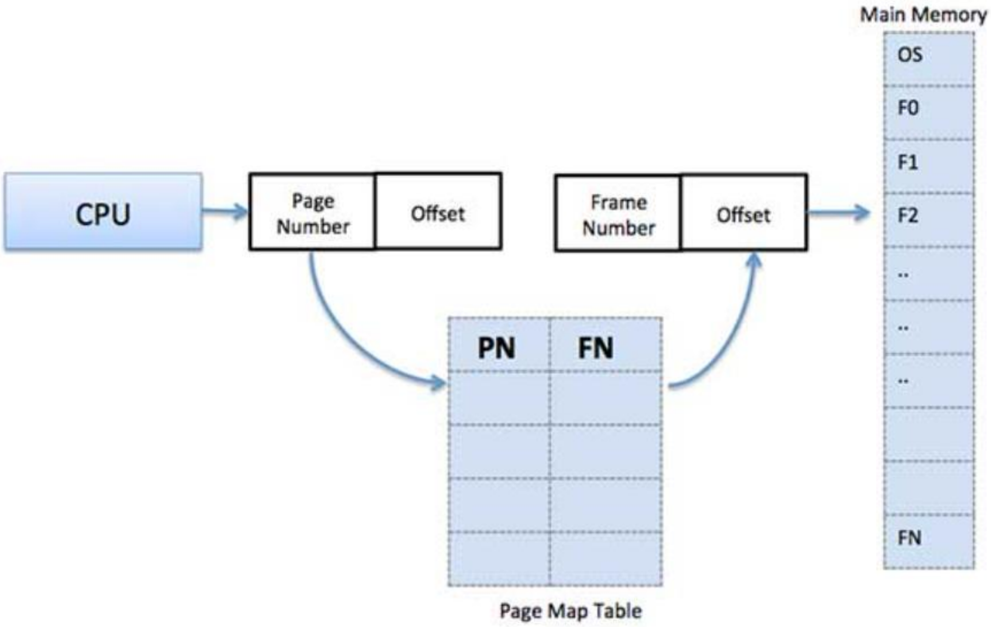
Page address is called logical address and represented by page number and the offset.

`Logical Address = Page number + page offset`

Frame address is called physical address and represented by a frame number and the offset.

`Physical Address = Frame number + page offset`

A data structure called page map table is used to keep track of the relation between a page of a process to a frame in physical memory.



When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Advantages of paging

- It allows to store parts of a single process in a non-contiguous fashion.
- It solves the problem of external fragmentation

Disadvantages of paging

- It suffers from internal fragmentation.
- There is an overhead of maintaining a page table for each process.
- The time taken to fetch the instruction increases since now two memory accesses are required

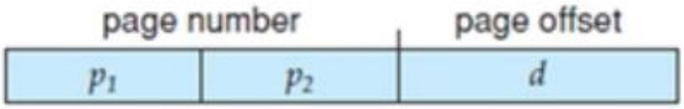
Structure of page table

There are several types of page tables, which are optimized for different requirements. Some of major page tables are listed below:

a. Hierarchical page table
b. Hashed page table
c. Inverted page table
d. Shared page table

a. Hierarchical page table

It is also called multilevel page table. The multilevel page table method is to avoid keeping all the pages tables in memory all the time. The top level page table with 1024 entries corresponding to the 10 bits PTL (Page Table Length Filed). When a virtual address is presented to the memory management unit (MMU). it first extracts the PTL. (Page Table Length Filed), and uses this value as an index into the top level page table.



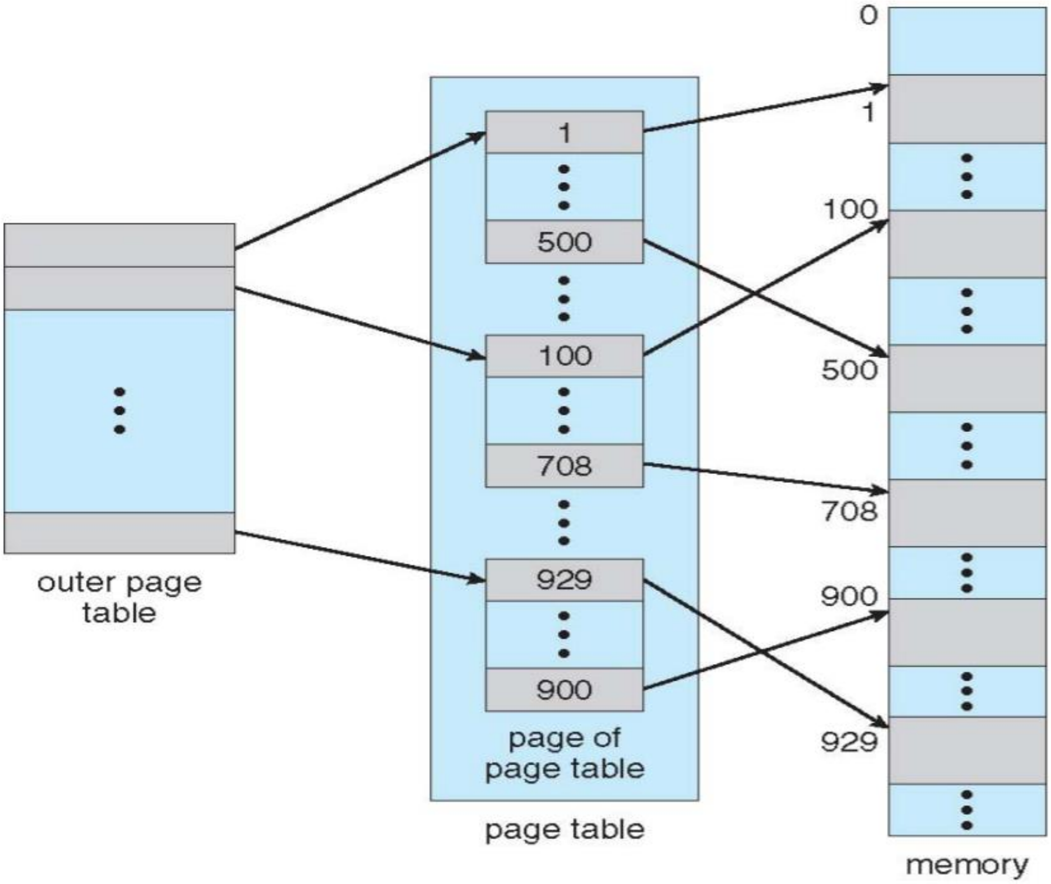
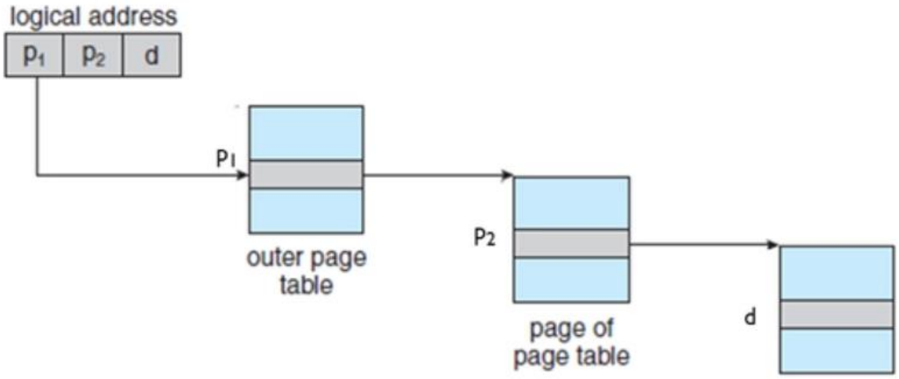


Fig: Two level page table

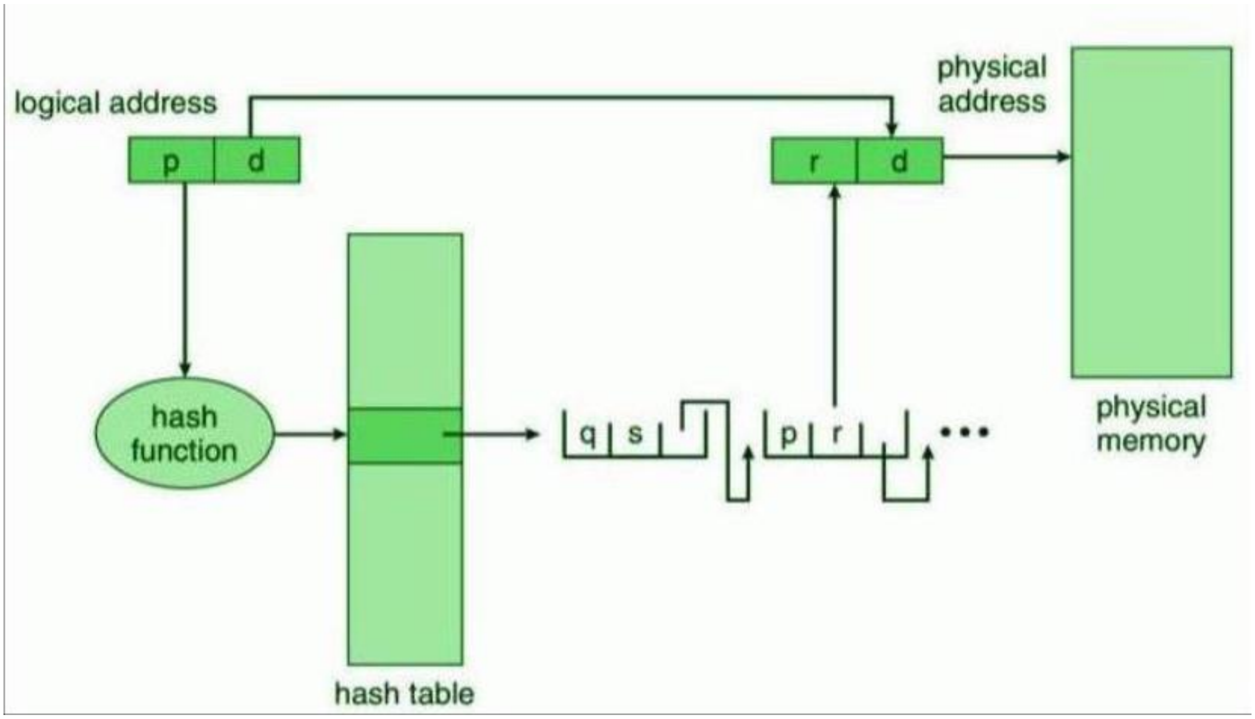


b. Hashed page table

In hashed page tables, the virtual page number in the virtual address is hashed into the hash table. They are used to handle address spaces higher than 32 bits. Each entry in the hash table has a linked list of elements hashed to the same location (to avoid collisions – as we can get the same value of a hash function for different page numbers). The hash value is the virtual page number. The Virtual Page Number is all the bits that are not a part of the page offset.

For each element in the hash table, there are three fields –

1. Virtual Page Number (which is the hash value).
2. Value of the mapped page frame.
3. A pointer to the next element in the linked list.



c. Inverted page table

Inverted page table is the global page table which is maintained by the operating system for all the processes. In inverted page table, the number of entries is equal to the number of frames in the main memory. It can be used to overcome the drawbacks of page table. There is always a space reserved for the page regardless of the fact what whether it is present in the main memory or not. However, this is simply the wastage of the memory if the page is not present. We can save this wastage by just inverting the page table. We can save the details only for the pages which are present in the main memory. Frames are the indices and the information saved inside the block will be Process ID and page number.

It has only one-page table for all processes. This table has an entry for each real page (block of physical memory). Each element contains the virtual address of the page stored in that physical location, with information about the process that owns that page.

frame no	Process id	Page no
0	P1	p0
1	P2	p1
2	P1	p2
3	P3	p1
4	P2	p3
5	P3	p2

Inverted page table

Page no	Frame no
0	f0
1	X
2	f2
3	X

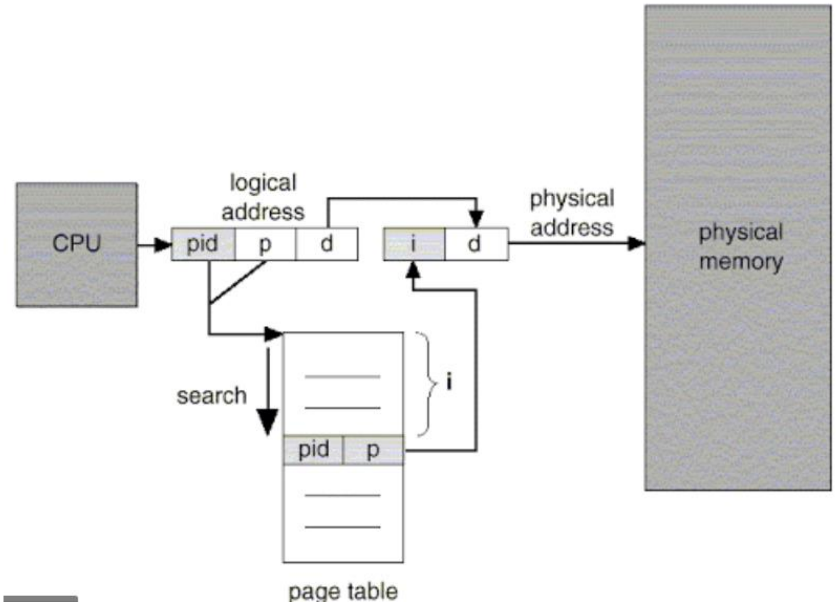
Page table of p1

Page no	Frame no
0	X
1	f3
2	f5
3	X

Page table of p3

Page no	Frame no
0	X
1	f1
2	X
3	f4

Page table of p2



d. Shared page table

The primary purpose of sharing page tables is improved performance for large applications that share big memory areas between multiple processes. It eliminates the redundant page tables and significantly reduces the number of minor page faults. Here pages may be shared by multiple processes.

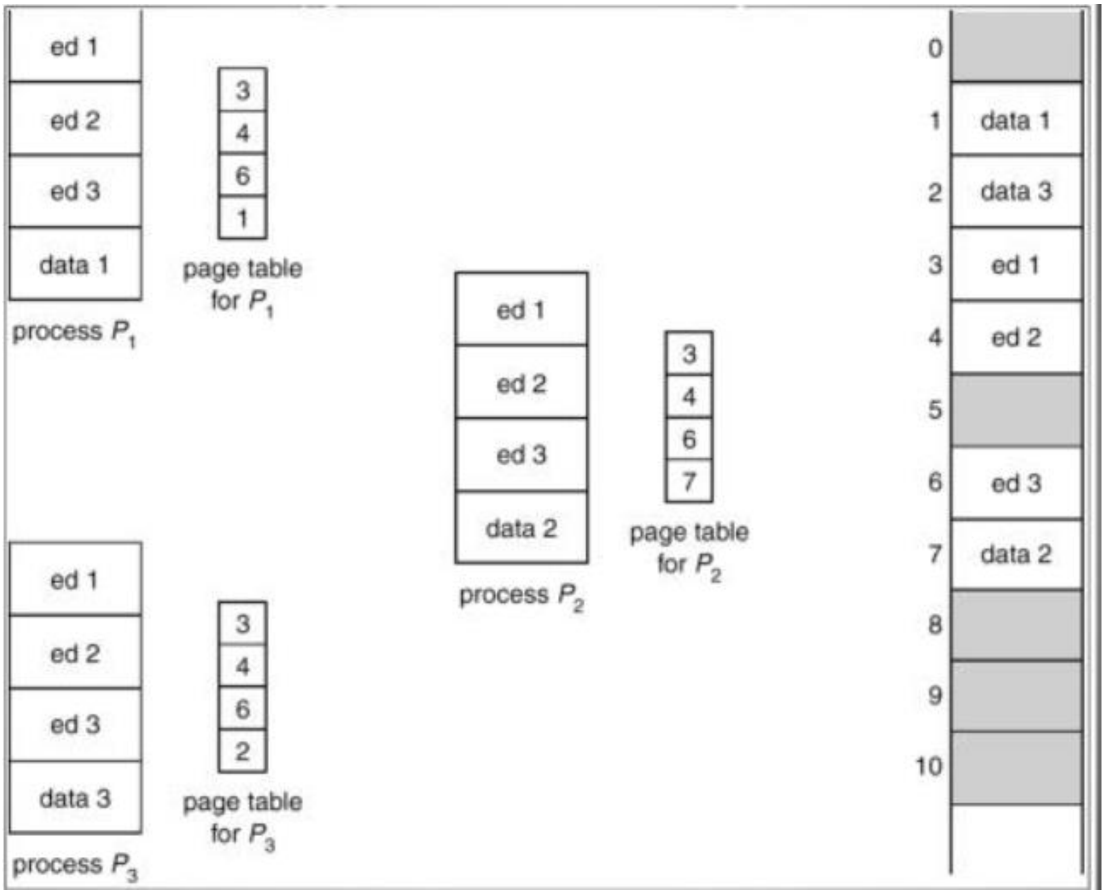
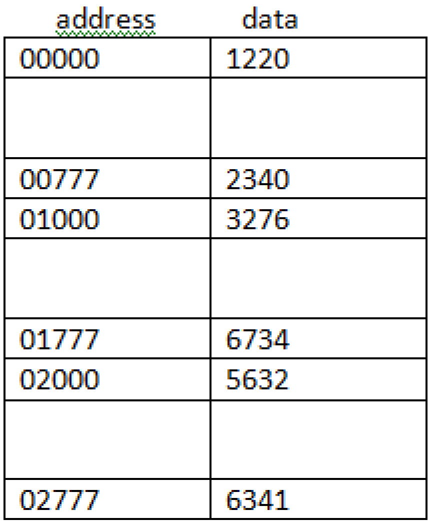


Fig: Shared page table

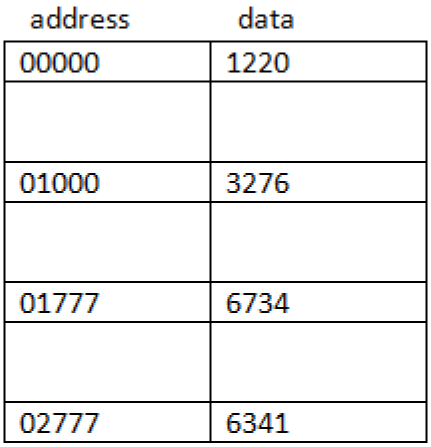
Mapping

The transformation of data from main memory to cache memory is referred to as a mapping. There are three types of mapping

1. Associative mapping: The fastest and most flexible cache organization uses an associative memory. The associative memory stores both the address and content (data) of the memory word. This permits any location in cache to store any word from main memory. A CPU address is placed in the argument register and the associative memory is searched for a matching address. If the address is found, the corresponding data is read and sent to the CPU. If no match occurs, the main memory is accessed for the word. The address- data pair is then transferred to the associative cache memory.



address	data
00000	1220
00777	2340
01000	3276
01777	6734
02000	5632
02777	6341

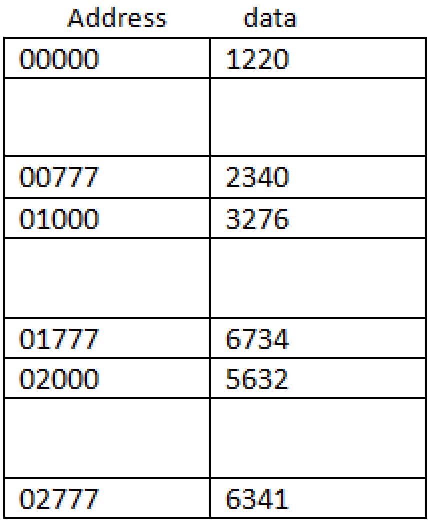


address	data
00000	1220
01000	3276
01777	6734
02777	6341

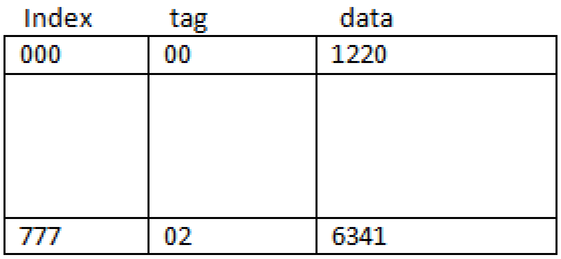
Cache memory

Main memory

2. Direct mapping: Associative memory is expensive compared to random-access memories because of the added logic associated with each cell. The CPU address is divided into two fields index and tag field. Least significant bits constitute the index field and the remaining form the tag field. The main memory needs an address that includes both the tag and the index bits. Each word in cache consists of the data word and its associated tag. When a new word is first brought into the cache, the tag bits are stored alongside the data bits. When the CPU generates a memory request, the index field is used for the address to access the cache. The tag field of the CPU address is compared with the tag in the word read from the cache. If the two tags match, there is a hit and the desired data word is in cache. If there is no match, there is a miss and the required word is read from main memory. The disadvantage of direct mapping is that the hit ratio can drop considerably if two or more words whose address have the same index but different tags are accessed repeatedly.



Address	data
00000	1220
00777	2340
01000	3276
01777	6734
02000	5632
02777	6341



Index	tag	data
000	00	1220
777	02	6341

Cache memory

Main memory

3. Set Associative mapping: Disadvantage of direct mapping is that two words with the same index in their address but with different tag values cannot reside in cache memory at the same time.

In set associative mapping each word of cache can store two or more words of memory under the same index address. Each data word is stored together with its tag and the number of tag-data items in one word of cache is said to form a set. When the CPU generates a memory request, the index value of the address is used to access the cache. The tag field of the CPU address is then compared with both tags in the cache to determine if a match occurs. The hit ratio will improve as the set size increases because more words with the same index but different tags can reside in cache. However, an increase in the set size increases the number of bits in words of cache and requires more complex comparison logic.

Address	Data
00000	1220
00777	2340
01000	3276
01777	6734
02000	5632
02777	6341

Main memory

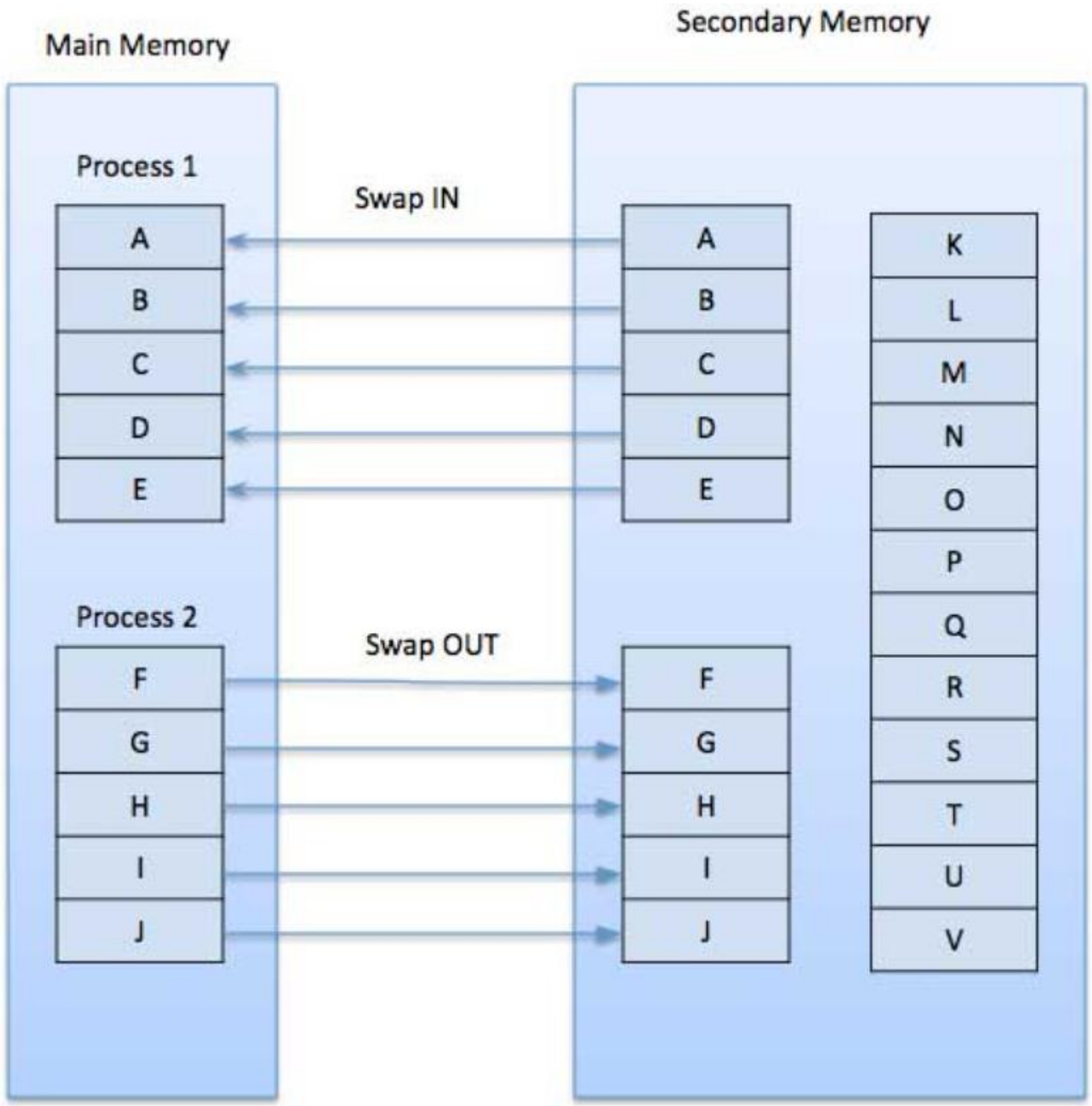
Index	Tag	Data	Tag	Data
000	01	3276	02	5632
777	02	6341	00	2340

Cache memory

Demand paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.

While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a page fault and transfers control from the program to the operating system to demand the page back into the memory



Advantages

Following are the advantages of Demand Paging –

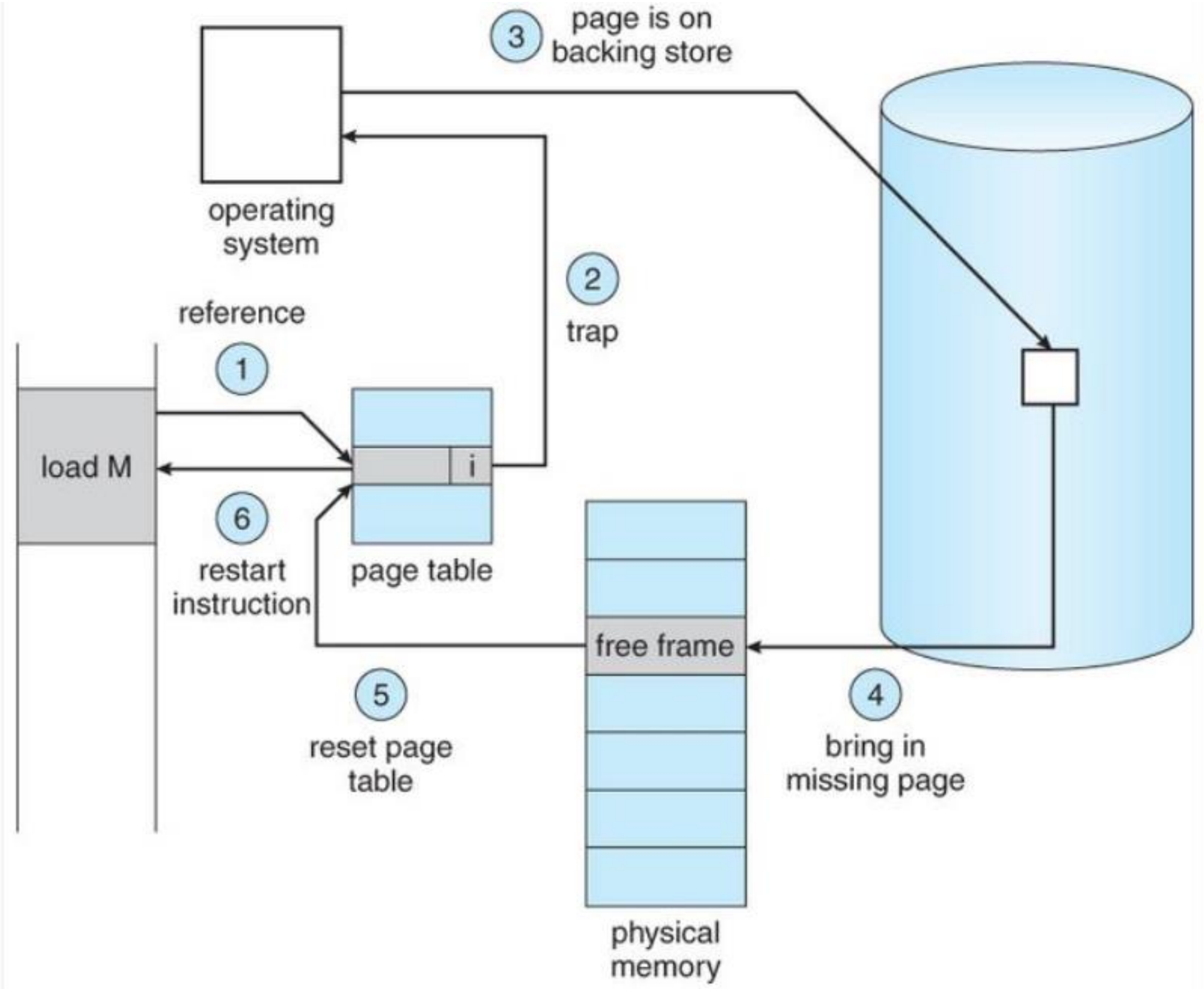
- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques

Page fault

A page fault occurs when a program attempts to access a block of memory that is not stored in the physical memory, or RAM. The fault notifies the operating system that it must locate the data in virtual memory, then transfer it from the storage device, such as an HDD or SSD, to the system RAM. So when page fault occurs then following sequence of events happens:



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

- If the virtual address is valid, the system checks to see if a page frame is free. If no frames are free, the page replacement algorithm is run to remove a page.
- If frame selected is dirty, page is scheduled for transfer to disk, context switch takes place, fault process is suspended and another process is made to run until disk transfer is completed.
- As soon as page frame is clean, operating system looks up disk address where needed page is, schedules disk operation to bring it in.
- When disk interrupt indicates page has arrived, page tables are updated to reflect its position, and frame marked as being in normal state.
- Faulting instruction is backed up to state it had when it began and PC is reset. Faulting is scheduled, operating system returns to routine that called it.
- Assembly Routine reloads register and other state information, returns to user space to continue execution.

Page Replacement Algorithm

Page Replacement Algorithm decides which page to remove, also called swap out when a new page needs to be loaded into the main memory. Page Replacement happens when a requested page is not present in the main memory and the available space is not sufficient for allocation to the requested page.

A page replacement algorithm tries to select which pages should be replaced so as to minimize the total number of page misses. There are many different page replacement algorithms. These algorithms are evaluated by running them on a particular string of memory reference and computing the number of page faults. The fewer is the page faults the better is the algorithm for that situation. Some Page Replacement Algorithms :

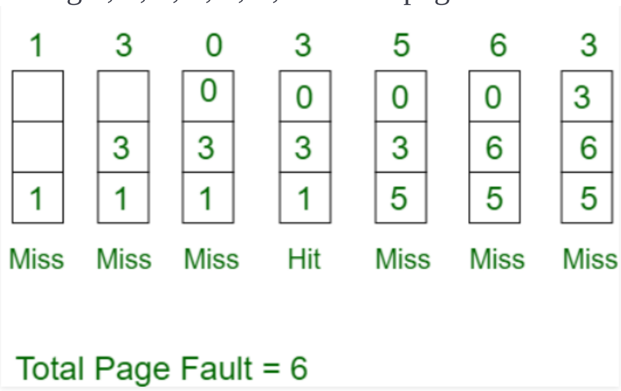
- First In First Out (FIFO)
- Least Recently Used (LRU)
- Optimal Page Replacement (OPR)

First In First Out (FIFO)

This is the simplest page replacement algorithm. In this algorithm, the OS maintains a queue that keeps track of all the pages in memory, with the oldest page at the front and the most recent page at the back. When there is a need for page replacement, the FIFO algorithm, swaps out the page at the front of the queue, that is the page which has been in the memory for the longest time.

For Example:

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



1	3	0	3	5	6	3
		0	0	0	0	3
	3	3	3	3	6	6
1	1	1	1	5	5	5
Miss	Miss	Miss	Hit	Miss	Miss	Miss

Total Page Fault = 6

Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.**
when 3 comes, it is already in memory so —> **0 Page Faults.**
Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.**
6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 Page Fault.**
Finally when 3 come it is not avilable so it replaces 0 **1 page fault**

Advantages

- Simple and easy to implement.
- Low overhead.

Disadvantages

- Poor performance.
- Doesn’t consider the frequency of use or last used time, simply replaces the oldest page.
- Suffers from Belady’s Anomaly (i.e. more page faults when we increase the number of page frames).

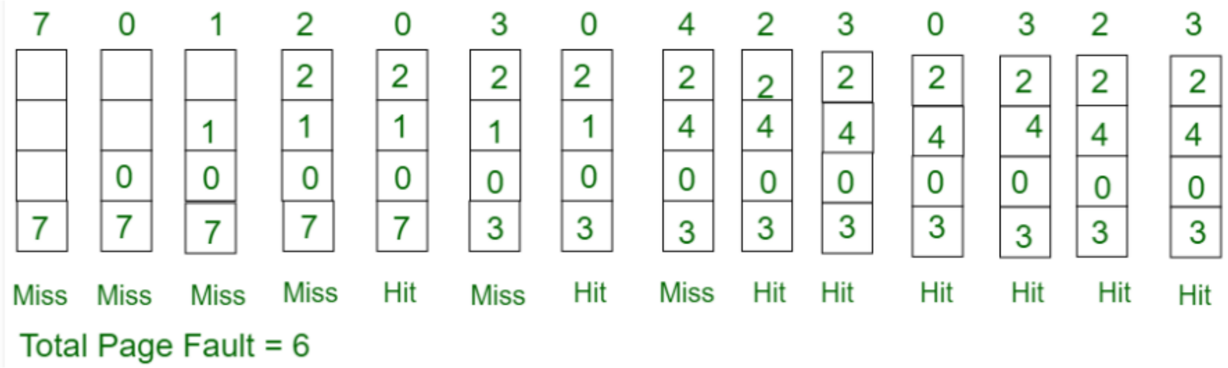
Least Recently Used (LRU)

Least Recently Used page replacement algorithm keeps track of page usage over a short period of time. It works on the idea that the pages that have been most heavily used in the past are most likely to be used heavily in the future too.

In LRU, whenever page replacement happens, the page which has not been used for the longest amount of time is replaced.

For Example:

Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frames. Find number of page faults.



Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already their so —> **0 Page fault.**
when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**
0 is already in memory so —> **0 Page fault**.
4 will takes place of 1 —> **1 Page Fault**
Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

Advantages

- Efficient.
- Doesn't suffer from Belady's Anomaly.

Disadvantages

- Complex Implementation.
- Expensive.
- Requires hardware support.

Optimal Page Replacement

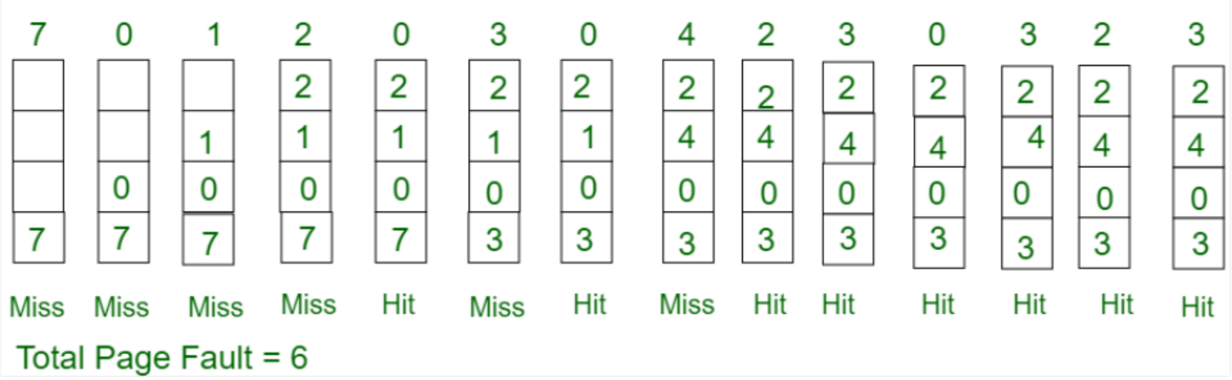
Optimal Page Replacement algorithm is the best page replacement algorithm as it gives the least number of page faults. It is also known as OPT, clairvoyant replacement algorithm, or Belady's optimal page replacement policy.

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future, i.e., the pages in the memory which are going to be referred farthest in the future are replaced.

This algorithm was introduced long back and is difficult to implement because it requires future knowledge of the program behaviour. However, it is possible to implement optimal page replacement on the second run by using the page reference information collected on the first run.

For Example

Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frame. Find number of page fault.



7	0	1	2	0	3	0	4	2	3	0	3	2	3
			2	2	2	2	2	2	2	2	2	2	2
		1	1	1	1	1	4	4	4	4	4	4	4
	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	3	3
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit

Total Page Fault = 6

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already there so —> **0 Page fault.**
when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.**
0 is already there so —> **0 Page fault.**.
4 will takes place of 1 —> **1 Page Fault.**

Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

Advantages

- Easy to Implement.
- Simple data structures are used.
- Highly efficient.

Disadvantages

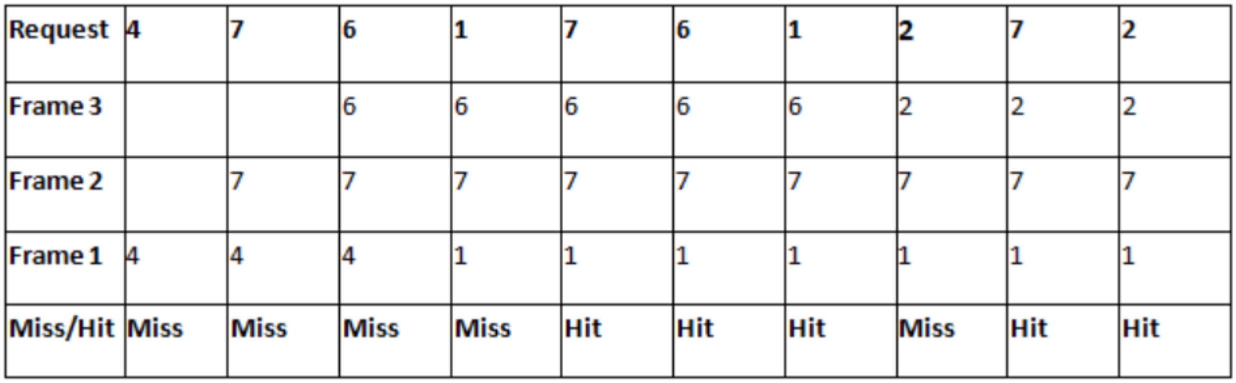
- Requires future knowledge of the program.

- Time-consuming.

Consider a reference string: 4, 7, 6, 1, 7, 6, 1, 2, 7, 2. the number of frames in the memory is 3. Find out the number of page faults respective to:

1. Optimal Page Replacement Algorithm
2. FIFO Page Replacement Algorithm
3. LRU Page Replacement Algorithm

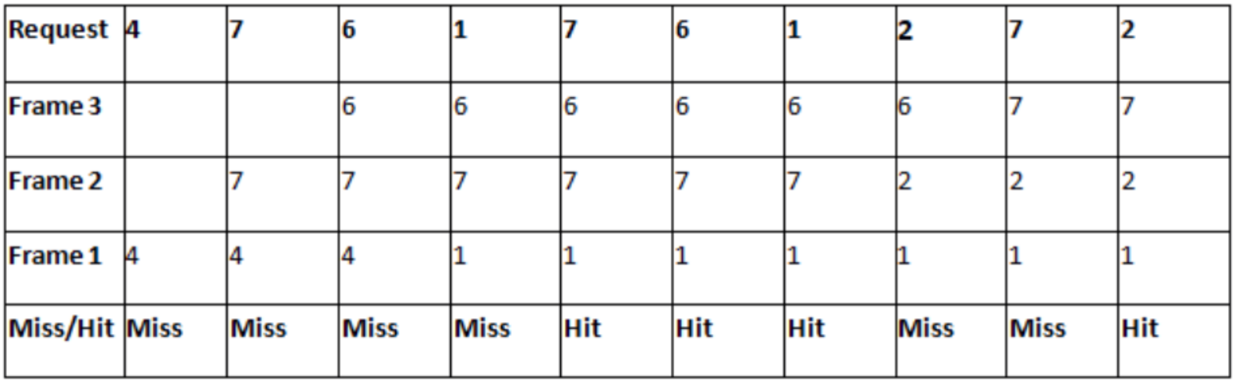
Optimal Page Replacement Algorithm



Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	2	2	2
Frame 2		7	7	7	7	7	7	7	7	7
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Hit	Hit

Number of Page Faults in Optimal Page Replacement Algorithm = 5

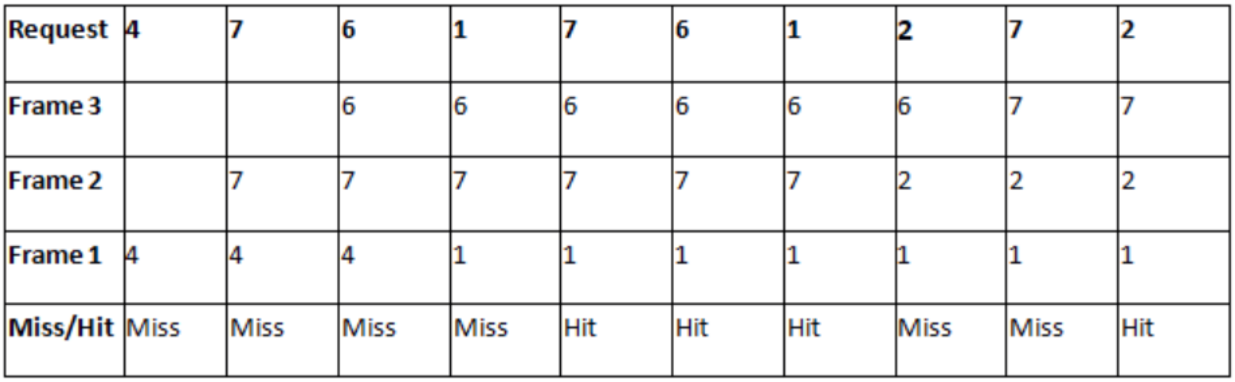
LRU Page Replacement Algorithm



Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	6	7	7
Frame 2		7	7	7	7	7	7	2	2	2
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Number of Page Faults in LRU = 6

FIFO Page Replacement Algorithm



Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	6	7	7
Frame 2		7	7	7	7	7	7	2	2	2
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Number of Page Faults in FIFO = 6

Thrashing

Thrashing in computing is an issue caused when virtual memory is in use. It occurs when the virtual memory of a computer is rapidly exchanging data for data on hard disk, to the exclusion of most application-level processing. As the main memory gets filled, additional pages need to be swapped in and out of virtual memory. The swapping causes a very high rate of hard disk access. Thrashing can continue for a long duration until the underlying issue is addressed. Thrashing can potentially result in total collapse of the hard drive of the computer. Thrashing is also known as disk thrashing.

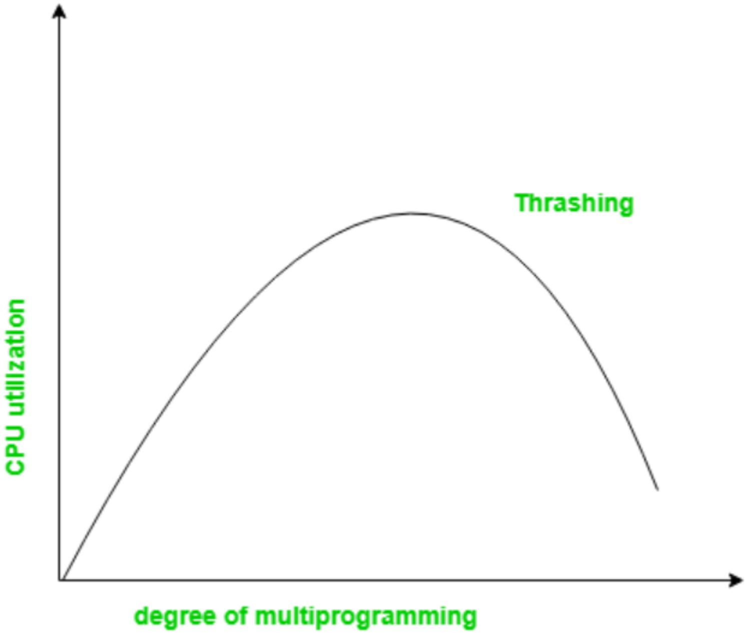


Fig: Thrashing

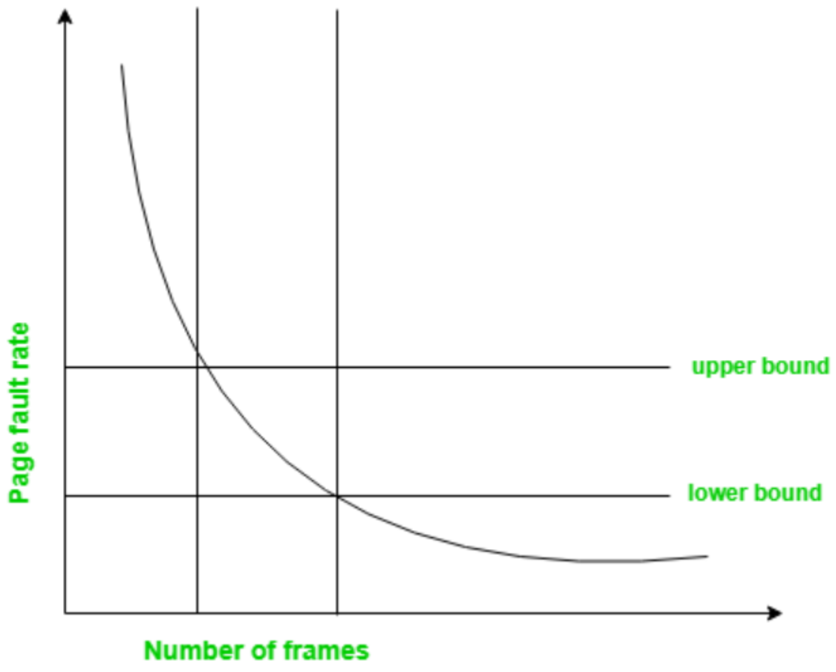
Techniques to Handle Thrashing

We already seen Local replacement is better than Global replacement to avoid thrashing. But it also has disadvantages and not suggestable. Some more techniques are

Working Set Model

This model is based on locality. What locality is saying, the page used recently can be used again and also the pages which are nearby this page will also be used. Working set means set of pages in the most recent D time. The page which completed its D amount of time in working set automatically dropped from it. So accuracy of working set depends on D we have chosen. This working set model avoid thrashing while keeping the degree of multiprogramming as high as possible.

Page Fault Frequency



It is some direct approach than working set model. When thrashing occurring we know that it has few number of frames. And if it is not thrashing that means it has too many frames. Based on this property we assign an upper and lower bound for the desired page fault rate. According to page fault rate we allocate or remove pages. If the page fault rate become less than the lower limit, frames can be removed from the process. Similarly, if the page fault rate become more than the upper limit, more number of frames can be allocated to the process. And if no frames available due to high page fault rate, we will just suspend the processes and will restart them again when frames available.

Segmentation

Segmentation is a technique of memory management. It is just like the Paging technique except the fact that in segmentation, the segments are of variable length but, in Paging, the pages are of fixed size. In segmentation, the memory is split into variable-length parts. Each part is known as segments. The information which is related to the segment is stored in a table which is called a segment table.

There are types of segmentation:

1. **Virtual memory segmentation –**
   Each process is divided into a number of segments, not all of which are resident at any one point in time.
2. **Simple segmentation –**
   Each process is divided into a number of segments, all of which are loaded into memory at run time, though not necessarily contiguously.

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

Segment Table – It maps two-dimensional Logical address into one-dimensional Physical address. It's each table entry has:

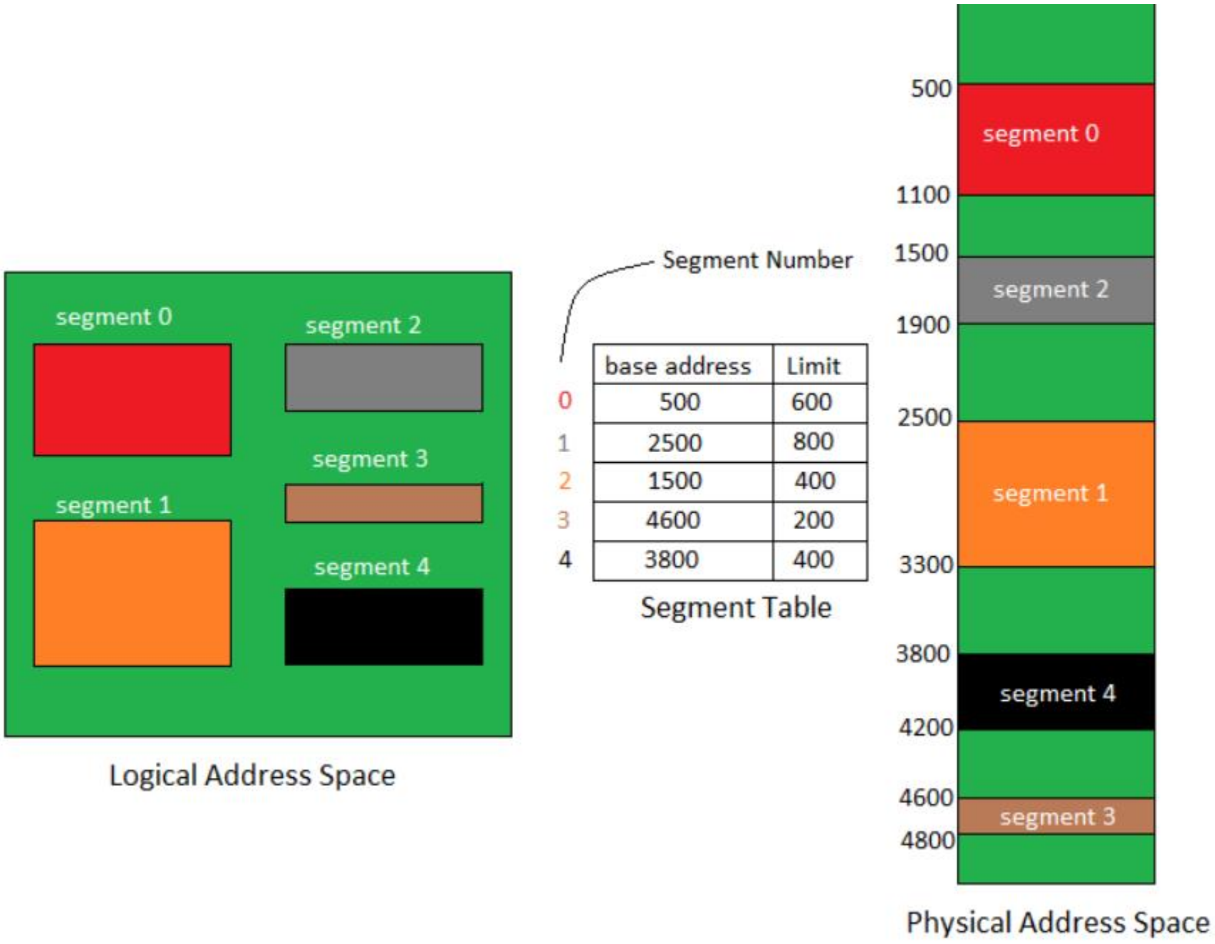
- **Base Address:** It contains the starting physical address where the segments reside in memory.
- **Limit:** It specifies the length of the segment.

Advantages of Segmentation

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

Disadvantages of Segmentation

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



	base address	Limit
0	500	600
1	2500	800
2	1500	400
3	4600	200
4	3800	400

Fig: Logical view of segmentation

Difference between Paging and Segmentation

Paging	Segmentation
1. In paging, program is divided into fixed or mounted size pages	1. In segmentation, program is divided into variable size sections.
2. For paging operating system is accountable.	2. For segmentation compiler is accountable.
3. Page size is determined by hardware.	3. Here, the section size is given by the user.
4. It is faster in the comparison of segmentation.	4. Segmentation is slow.
5. Paging could result in internal fragmentation.	5. Segmentation could result in external fragmentation.
6. In paging, logical address is split into page number and page offset.	6. Here, logical address is split into section number and section offset.

7. Paging comprises a page table which encloses the base address of every page.	7. While segmentation also comprises the segment table which encloses segment number and segment offset.
8. Paging is invisible to the user.	8. Segmentation is visible to the user.

Segmentation with paging (MULTICS)

Instead of an actual memory location the segment information includes the address of a page table for the segment. When a program references a memory location the offset is translated to a memory address using page table. A segment can be extended simply by allocating another memory page and adding it to the segment’s page table.

An implementation of virtual memory on a system using segmentation with paging usually only moves individual pages back and forth between main memory and secondary storage, similar to a paged non-segmented system. Pages of the segment can be located anywhere in main memory and need not be contiguous. This usually results in a reduced amount of input/output between primary and secondary storage and reduced memory fragmentation.

The MULTICS (Multiplexed Information and Computing Service) operating system was one of the most influential operating systems ever, having had a major influence on topics as disparate as UNIX, the x86 memory architecture, TLBs, and cloud computing. It was started as research project at MIT and went live in 1969. The MULTICS system solved problems of external fragmentation and lengthy search times by paging the segments.