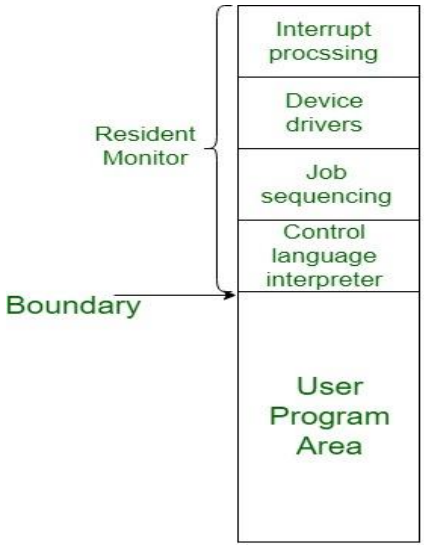
Unit-4th – Memory Management

Bare Machine: - Bare machine is logical hardware which is used to execute the program in the processor without using the operating system. We can't execute any process without the Operating system. But yes with the help of the Bare machine we can do that.
Initially, when the operating systems are not developed, the execution of an instruction is done by directly on hardware without using any interfering hardware, at that time the only drawback was that the Bare machines accepting the instruction in only machine language, due to this those person who has sufficient knowledge about Computer field are able to operate a computer. So after the development of the operating system Bare machine is referred to as inefficient.

Resident Monitor: - if we talk about how the code runs on Bare machines, then this component is used, so basically, the Resident Monitor is a code that runs on Bare Machines. The resident monitor works like an operating system that controls the instructions and performs all necessary functions. It also works like job sequencer because it also sequences the job and sends them to the processor.
After scheduling the job Resident monitors loads the programs one by one into the main memory according to their sequences. One most important factor about the resident monitor is that when the program execution occurred there is no gap between the program execution and the processing is going to be faster.

The Resident monitors are divided into 4 parts as:

1. Control Language Interpreter
2. Loader
3. Device Driver
4. Interrupt Processing



These are explained as following below.

1. Control Language Interpreter: - The first part of the Resident monitor is control language interpreter which is used to read and carry out the instruction from one level to the next level.

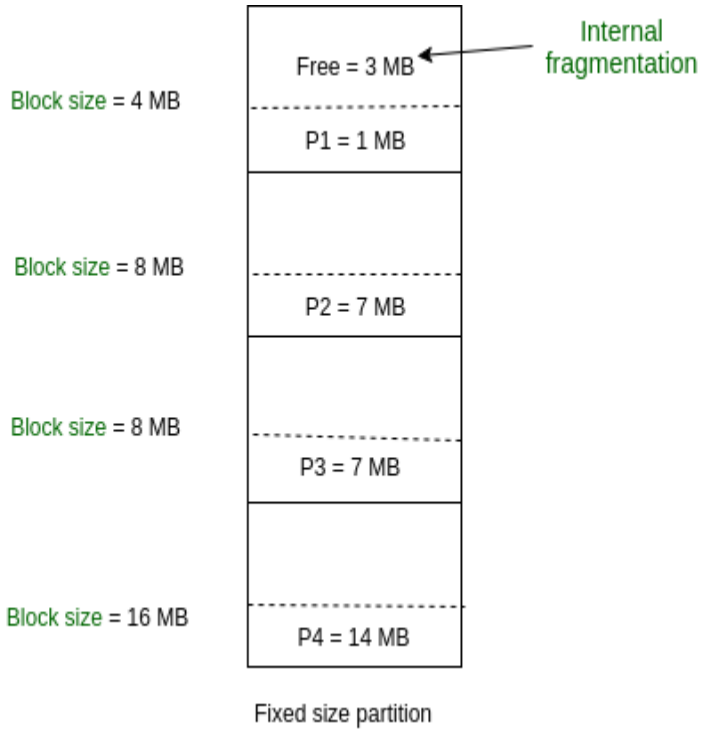
2. Loader: - The second part of the Resident monitor which is the main part of the Resident Monitor is Loader which Loads all the necessary system and application programs into the main memory.

3. Device Driver: - The third part of the Resident monitor is Device Driver which is used to managing the connecting input-output devices to the system. So basically it is the interface between the user and the system. It works as an interface between the request and response. Request which user made, Device driver responds that the system produces to fulfil these requests.

4. Interrupt Processing: - The fourth part as the name suggests, it processes the all occurred interrupt to the system.

Multiprogramming with Fixed Partitioning:

This is the oldest and simplest technique used to put more than one process in the main memory. In this partitioning, the number of partitions (non-overlapping) in RAM is **fixed but the size** of each partition may or **may not be the same**. As it is a **contiguous** allocation, hence no spanning is allowed. Here partitions 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 in spite 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 a certain partition without focusing on the emergence of Internal and External Fragmentation.

2. Little OS overhead: - Processing of Fixed Partitioning requires 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.

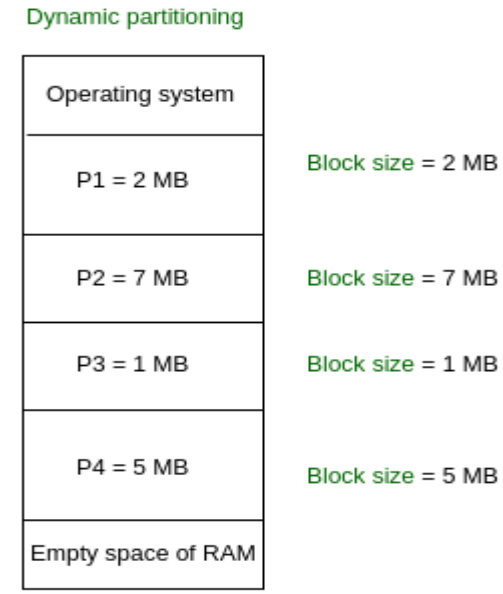
2. External Fragmentation: - The total unused space (as stated above) of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form (as spanning is not allowed).
3. Limit process size: - Process of size greater than the size of the partition in Main Memory cannot be accommodated. The partition size cannot be varied according to the size of the incoming process size. Hence, the process size of 32MB in the above-stated example is invalid.

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

Multiprogramming with Variable 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.



There are some advantages and disadvantages of variable partitioning over fixed partitioning as given below.

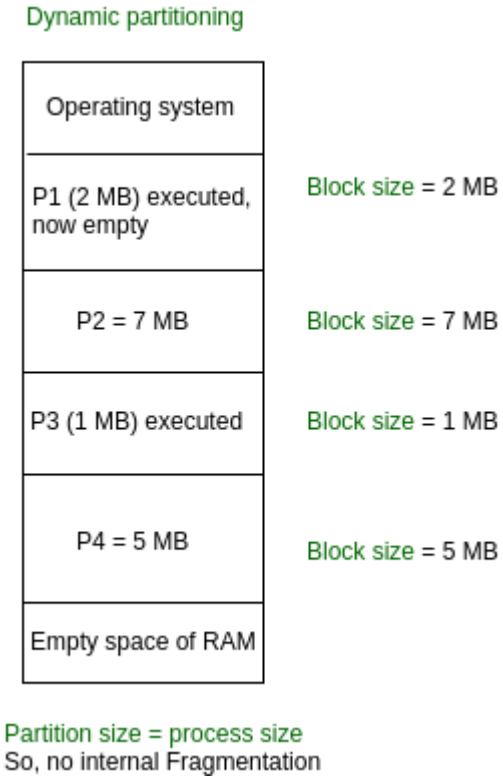
Advantages of Variable Partitioning –

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 cannot 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 in spite of absence of internal fragmentation.

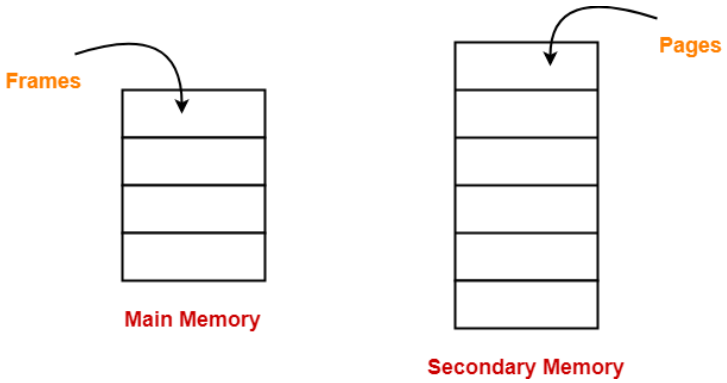
For example, suppose in above example- process P1 (2MB) and process P3(1MB) completed their execution. Hence two spaces are left i.e. 2MB and 1MB. Let's suppose process P5 of size 3MB comes. The empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation. The rule says that process must be contiguously present in main memory to get executed. Hence it results in External Fragmentation.



Now P5 of size 3 MB cannot be accommodated in spite of required available space because in contiguous no spanning is allowed.

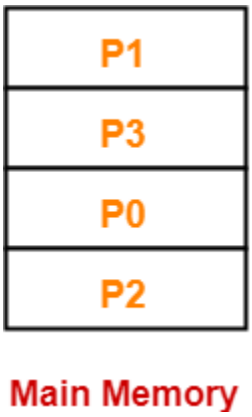
Paging

- Paging is a fixed size partitioning scheme.
- In paging, secondary memory and main memory are divided into equal fixed size partitions.
- The partitions of secondary memory are called as **pages**.
- The partitions of main memory are called as **frames**.



- Each process is divided into parts where size of each part is same as page size.
- The size of the last part may be less than the page size.
- The pages of process are stored in the frames of main memory depending upon their availability.

Example- Consider a process is divided into 4 pages P_0, P_1, P_2 and P_3. Depending upon the availability, these pages may be stored in the main memory frames in a non-contiguous fashion as shown-



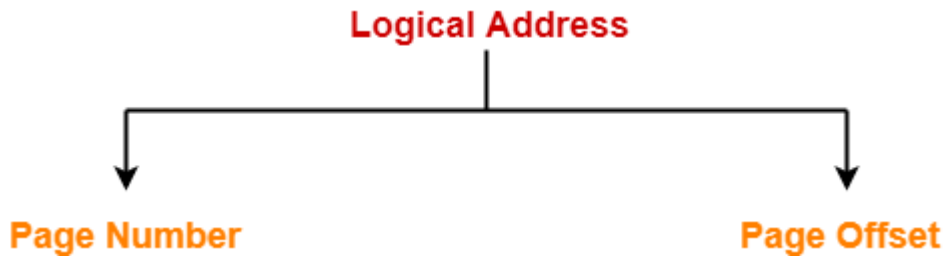
Translating Logical Address into Physical Address:-

- CPU always generates a logical address.
- A physical address is needed to access the main memory.

Following steps are followed to translate logical address into physical address-

Step-1: - CPU generates a logical address consisting of two parts-

1. Page Number
2. Page Offset



- Page Number specifies the specific page of the process from which CPU wants to read the data.
- Page Offset specifies the specific word on the page that CPU wants to read.

Step-2:- For the page number generated by the CPU,

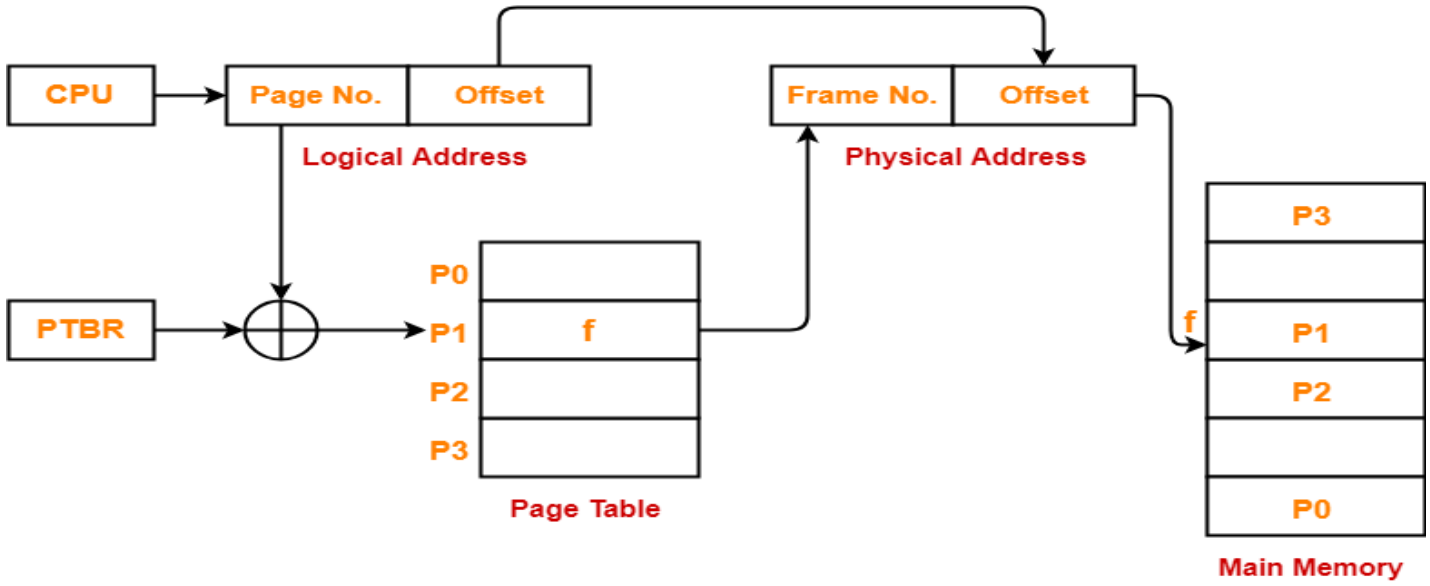
- **Page Table** provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.

Step-3: - The frame number combined with the page offset forms the required physical address.

- Frame number specifies the specific frame where the required page is stored.
- Page Offset specifies the specific word that has to be read from that page.

Diagram-

The following diagram illustrates the above steps of translating logical address into physical address-



Translating Logical Address into Physical Address

Advantages-

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

Disadvantages-

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

Important Formulas

For Main Memory-

- Physical Address Space = Size of main memory
- Size of main memory = Total number of frames x Page size
- Frame size = Page size

- If number of frames in main memory = 2^X, then number of bits in frame number = X bits
- If Page size = 2^X Bytes, then number of bits in page offset = X bits
- If size of main memory = 2^X Bytes, then number of bits in physical address = X bits

For Process-

- Virtual Address Space = Size of process
- Number of pages the process is divided = Process size / Page size
- If process size = 2^X bytes, then number of bits in virtual address space = X bits

For Page Table-

- Size of page table = Number of entries in page table x Page table entry size
- Number of entries in pages table = Number of pages the process is divided
- Page table entry size = Number of bits in frame number + Number of bits used for optional fields if any

NOTE-

- In general, if the given address consists of 'n' bits, then using 'n' bits, 2^n locations are possible.
- Then, size of memory = 2^n x Size of one location.
- If the memory is byte-addressable, then size of one location = 1 byte.
- Thus, size of memory = 2^n bytes.
- If the memory is word-addressable where 1 word = m bytes, then size of one location = m bytes.
- Thus, size of memory = 2^n x m bytes.

PRACTICE PROBLEMS BASED ON PAGING AND PAGE TABLE-

Problem-01:- Calculate the size of memory if its address consists of 22 bits and the memory is 2-byte addressable.

Solution-

- Number of locations possible with 22 bits = 2^{22} locations
- It is given that the size of one location = 2 bytes

Thus, Size of memory

= 2^{22} x 2 bytes

= 2^{23} bytes

= 8 MB

Problem-02:- Calculate the number of bits required in the address for memory having size of 16 GB. Assume the memory is 4-byte addressable.

Solution-

Let ‘n’ number of bits are required. Then, Size of memory = 2^n x 4 bytes.

Since, the given memory has size of 16 GB, so we have-

2^n x 4 bytes = 16 GB

2^n x 4 = 16 G

2^n x 2^2 = 2^{34}

2^n = 2^{32}

∴ n = 32 bits

Problem-03:- Consider a system with byte-addressable memory, 32 bit logical addresses, 4 kilobyte page size and page table entries of 4 bytes each. The size of the page table in the system in megabytes is _____.

1. 2
2. 4
3. 8
4. 16

Solution-

Given-

- Number of bits in logical address = 32 bits
- Page size = 4KB
- Page table entry size = 4 bytes

Process Size-

Number of bits in logical address = 32 bits

Thus,

Process size

= 2^{32} B

= 4 GB

Number of Entries in Page Table-

Number of pages the process is divided

= Process size / Page size

= 4 GB / 4 KB

= 2^{20} pages

Thus,

Number of entries in page table = 2^{20} entries

Page Table Size-

Page table size

= Number of entries in page table x Page table entry size

= 2^{20} x 4 bytes

= 4 MB

Thus, Option (B) is correct.

Problem-04:- Consider a machine with 64 MB physical memory and a 32 bit virtual address space. If the page size is 4 KB, what is the approximate size of the page table?

1. 16 MB
2. 8 MB
3. 2 MB
4. 24 MB

Solution-

Given-

- Size of main memory = 64 MB
- Number of bits in virtual address space = 32 bits
- Page size = 4 KB

We will consider that the memory is byte addressable.

Number of Bits in Physical Address-

Size of main memory

= 64 MB

= 2^{26} B

Thus, Number of bits in physical address = 26 bits

Number of Frames in Main Memory-

Number of frames in main memory

= Size of main memory / Frame size

= 64 MB / 4 KB

= 2^{26} B / 2^{12} B

= 2^{14}

Thus, Number of bits in frame number = 14 bits

Number of Bits in Page Offset-

We have,

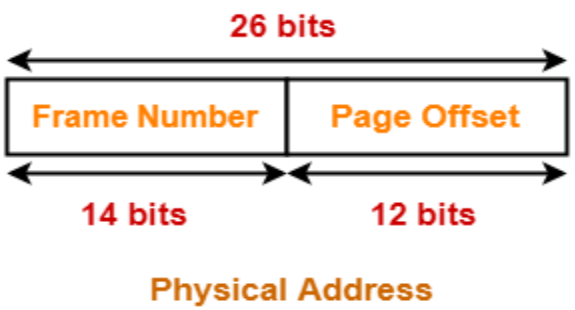
Page size

= 4 KB

= 2^{12} B

Thus, Number of bits in page offset = 12 bits

So, Physical address is-



Process Size-

Number of bits in virtual address space = 32 bits

Thus,

Process size

= 2^{32} B

= 4 GB

Number of Entries in Page Table-

Number of pages the process is divided

= Process size / Page size

= 4 GB / 4 KB

= 2^{20} pages

Thus, Number of entries in page table = 2^{20} entries

Page Table Size-

Page table size

= Number of entries in page table x Page table entry size

= Number of entries in page table x Number of bits in frame number

= 2^{20} x 14 bits

= 2^{20} x 16 bits (Approximating 14 bits ≈ 16 bits)

= 2^{20} x 2 bytes

= 2 MB

Thus, Option (C) is correct.

Problem-05:- In a virtual memory system, size of virtual address is 32-bit, size of physical address is 30-bit, page size is 4 Kbyte and size of each page table entry is 32-bit. The main memory is byte addressable. Which one of the following is the maximum number of bits that can be used for storing protection and other information in each page table entry?

1. 2
2. 10
3. 12
4. 14

Solution-

Given-

- Number of bits in virtual address = 32 bits
- Number of bits in physical address = 30 bits
- Page size = 4 KB
- Page table entry size = 32 bits

Size of Main Memory-

Number of bits in physical address = 30 bits

Thus,

Size of main memory

$= 2^{30}$ B

= 1 GB

Number of Frames in Main Memory-

Number of frames in main memory

= Size of main memory / Frame size

= 1 GB / 4 KB

= 2^{30} B / 2^{12} B

= 2^{18}

Thus, Number of bits in frame number = 18 bits

Number of Bits used for Storing other Information-

Maximum number of bits that can be used for storing protection and other information

= Page table entry size – Number of bits in frame number

= 32 bits – 18 bits

= 14 bits

Thus, Option (D) is correct.

Segmentation-

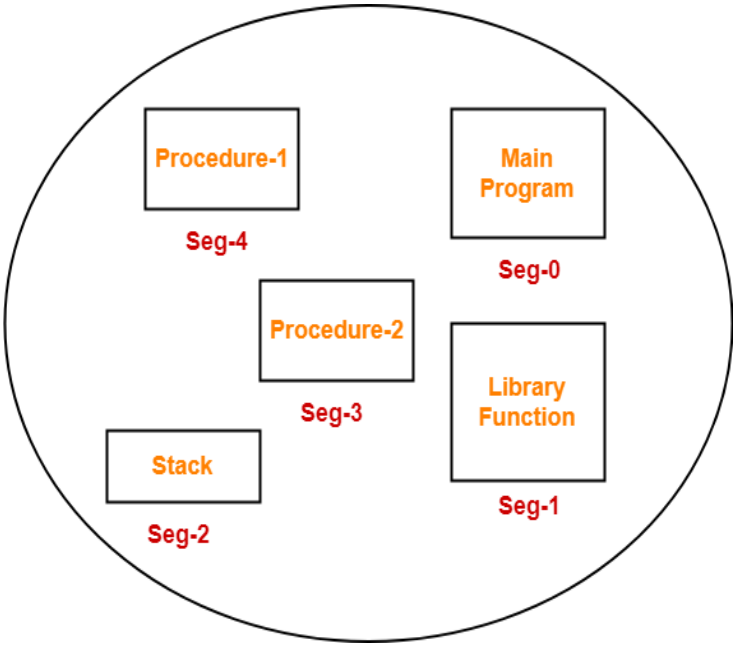
- Like Paging, Segmentation is another non-contiguous memory allocation technique.
- In segmentation, process is not divided blindly into fixed size pages.
- Rather, the process is divided into modules for better visualization.

Characteristics-

- Segmentation is a variable size partitioning scheme.
- In segmentation, secondary memory and main memory are divided into partitions of unequal size.
- The size of partitions depend on the length of modules.
- The partitions of secondary memory are called as **segments**.

Example-

Consider a program is divided into 5 segments as-



Segment Table-

- Segment table is a table that stores the information about each segment of the process.
- It has two columns.
- First column stores the size or length of the segment.
- Second column stores the base address or starting address of the segment in the main memory.
- Segment table is stored as a separate segment in the main memory.
- Segment table base register (STBR) stores the base address of the segment table.

For the above illustration, consider the segment table is-

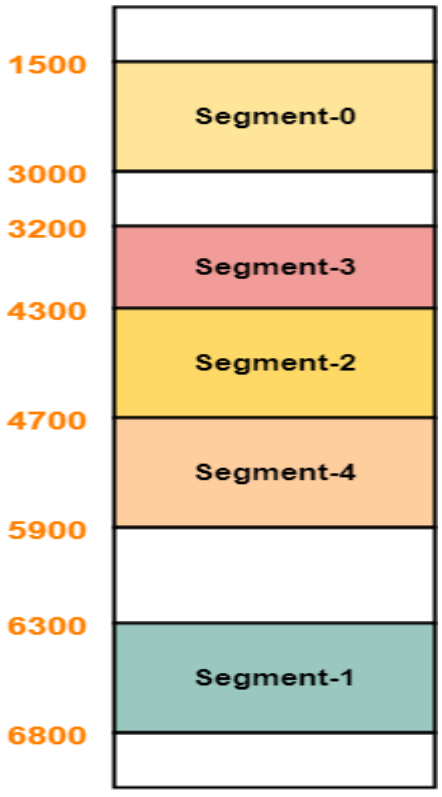
	Limit	Base
Seg-0	1500	1500
Seg-1	500	6300
Seg-2	400	4300
Seg-3	1100	3200
Seg-4	1200	4700

Segment Table

Here,

- Limit indicates the length or size of the segment.
- Base indicates the base address or starting address of the segment in the main memory.

In accordance to the above segment table, the segments are stored in the main memory as-



Main Memory

Translating Logical Address into Physical Address-

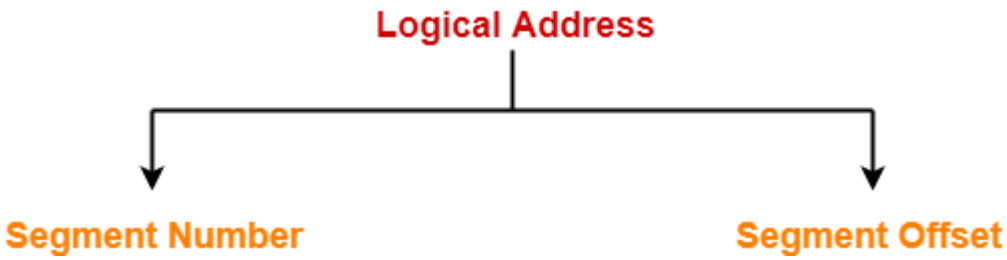
- CPU always generates a logical address.
- A physical address is needed to access the main memory.

Following steps are followed to translate logical address into physical address-

Step-01:

CPU generates a logical address consisting of two parts-

1. Segment Number
2. Segment Offset



- Segment Number specifies the specific segment of the process from which CPU wants to read the data.
- Segment Offset specifies the specific word in the segment that CPU wants to read.

Step-02:

- For the generated segment number, corresponding entry is located in the segment table.
- Then, segment offset is compared with the limit (size) of the segment.

Now, two cases are possible-

Case-01: Segment Offset >= Limit

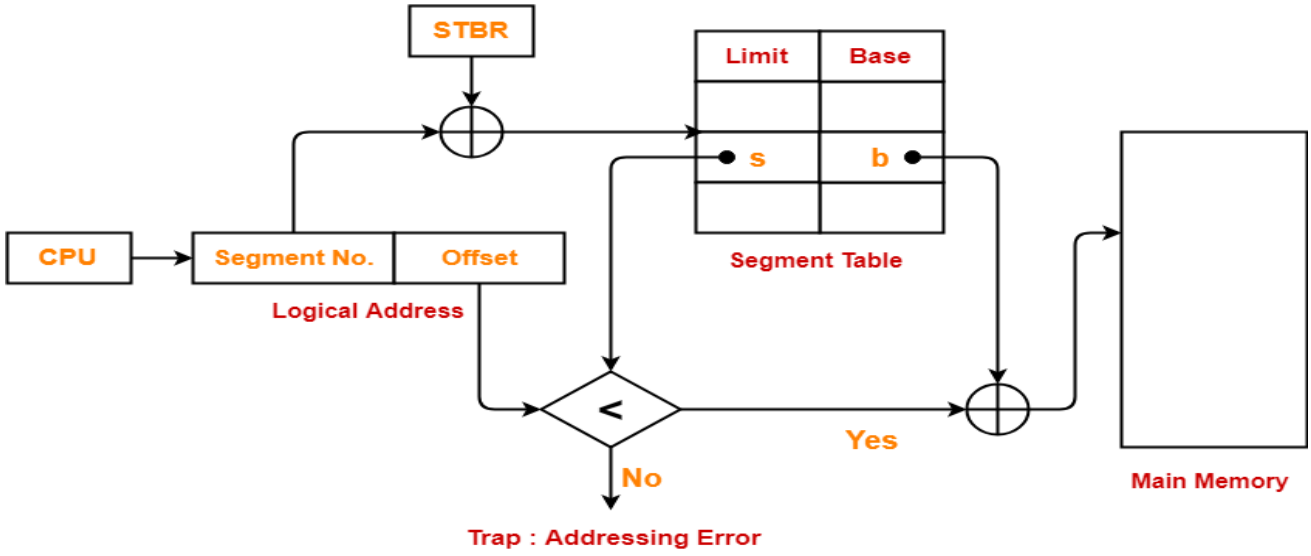
- If segment offset is found to be greater than or equal to the limit, a trap is generated.

Case-02: Segment Offset < Limit

- If segment offset is found to be smaller than the limit, then request is treated as a valid request.
- The segment offset must always lie in the range [0, limit-1],
- Then, segment offset is added with the base address of the segment.
- The result obtained after addition is the address of the memory location storing the required word.

Diagram-

The following diagram illustrates the above steps of translating logical address into physical address-



Translating Logical Address into Physical Address

Advantages-

- It allows to divide the program into modules which provides better visualization.
- Segment table consumes less space as compared to **Page Table** in paging.
- It solves the problem of internal fragmentation.

Disadvantages-

The disadvantages of segmentation are-

- There is an overhead of maintaining a segment table for each process.
- The time taken to fetch the instruction increases since now two memory accesses are required.
- Segments of unequal size are not suited for swapping.
- It suffers from external fragmentation as the free space gets broken down into smaller pieces with the processes being loaded and removed from the main memory

Problem-

Consider the following segment table-

Segment No.	Base	Length
0	1219	700
1	2300	14
2	90	100
3	1327	580
4	1952	96

Which of the following logical address will produce trap addressing error?

1. 0, 430
2. 1, 11
3. 2, 100
4. 3, 425
5. 4, 95

Calculate the physical address if no trap is produced.

Solution-

In a segmentation scheme, the generated logical address consists of two parts-

1. Segment Number
2. Segment Offset

We know-

- Segment Offset must always lie in the range [0, limit-1].
- If segment offset becomes greater than or equal to the limit of segment, then trap addressing error is produced.

Option-A: 0, 430-

Here,

- Segment Number = 0
- Segment Offset = 430

We have,

- In the segment table, limit of segment-0 is 700.
- Thus, segment offset must always lie in the range = [0, 700-1] = [0, 699]

Now,

- Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced.
- Physical Address = 1219 + 430 = 1649

Option-B: 1, 11-

Here,

- Segment Number = 1
- Segment Offset = 11

We have,

- In the segment table, limit of segment-1 is 14.
- Thus, segment offset must always lie in the range = [0, 14-1] = [0, 13]

Now,

- Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced.
- Physical Address = 2300 + 11 = 2311

Option-C: 2, 100-

Here,

- Segment Number = 2
- Segment Offset = 100

We have,

- In the segment table, limit of segment-2 is 100.
- Thus, segment offset must always lie in the range = [0, 100-1] = [0, 99]

Now,

- Since generated segment offset does not lie in the above range, so request generated is invalid.
- Therefore, trap will be produced.

Option-D: 3, 425-

Here,

- Segment Number = 3
- Segment Offset = 425

We have,

- In the segment table, limit of segment-3 is 580.
- Thus, segment offset must always lie in the range = [0, 580-1] = [0, 579]

Now,

- Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced.
- Physical Address = 1327 + 425 = 1752

Option-E: 4, 95-

Here,

- Segment Number = 4
- Segment Offset = 95

We have,

- In the segment table, limit of segment-4 is 96.
- Thus, segment offset must always lie in the range = [0, 96-1] = [0, 95]

Now,

- Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced.
- Physical Address = 1952 + 95 = 2047

Thus, Option-(C) is correct.

What is Virtual Memory?

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

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

Why Need Virtual Memory?

Here, are reasons for using virtual memory:

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

How Virtual Memory Works?

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

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

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

For example:

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

- The OS will then set up 250 MB of virtual memory and use a program called the Virtual Memory Manager(VMM) to manage that 250 MB.
- So, in this case, the VMM will create a file on the hard disk that is 250 MB in size to store extra memory that is required.

- The OS will now proceed to address memory as it considers 300 MB of real memory stored in the RAM, even if only 50 MB space is available.
- It is the job of the VMM to manage 300 MB memory even if just 50 MB of real memory space is available.

Advantages of Virtual Memory

Here, are pros/benefits of using Virtual Memory:

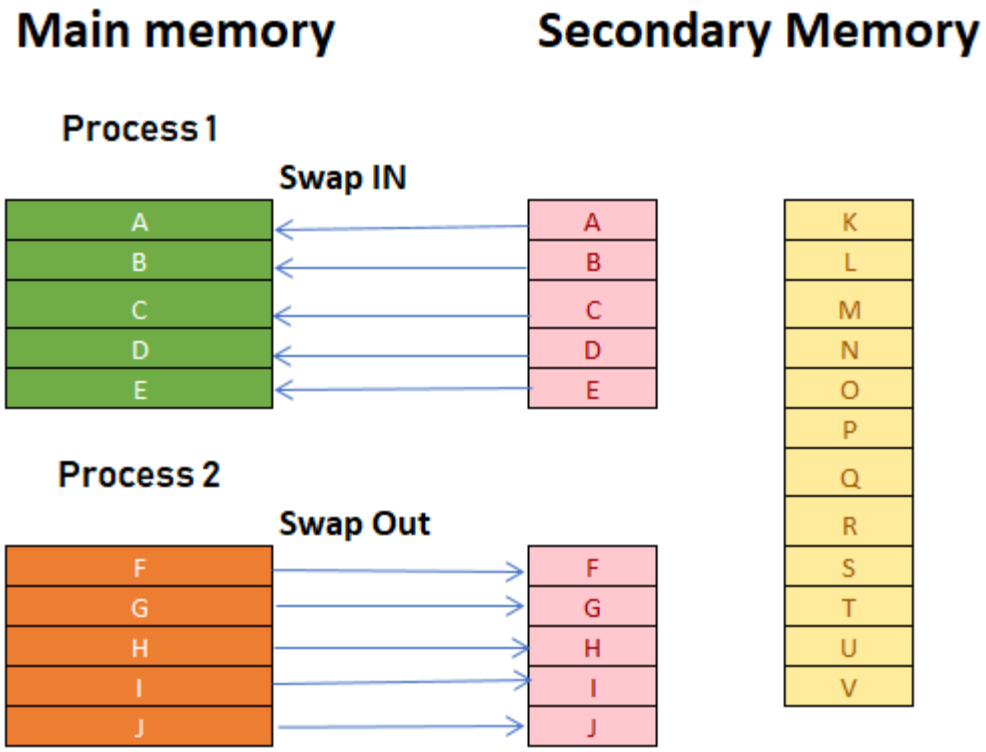
- Virtual memory helps to gain speed when only a particular segment of the program is required for the execution of the program.
- It is very helpful in implementing a multiprogramming environment.
- It allows you to run more applications at once.
- It helps you to fit many large programs into smaller programs.
- Common data or code may be shared between memories.
- Process may become even larger than all of the physical memory.
- Data / code should be read from disk whenever required.
- The code can be placed anywhere in physical memory without requiring relocation.
- More processes should be maintained in the main memory, which increases the effective use of CPU.
- Each page is stored on a disk until it is required after that, it will be removed.
- It allows more applications to be run at the same time.
- There is no specific limit on the degree of multiprogramming.
- Large programs should be written, as virtual address space available is more compared to physical memory.

Disadvantages of Virtual Memory

Here, are drawbacks/cons of using virtual memory:

- Applications may run slower if the system is using virtual memory.
- Likely takes more time to switch between applications.
- Offers lesser hard drive space for your use.
- It reduces system stability.
- It allows larger applications to run in systems that don't offer enough physical RAM alone to run them.
- It doesn't offer the same performance as RAM.
- It negatively affects the overall performance of a system.
- Occupy the storage space, which may be used otherwise for long term data storage.

What is Demand Paging?



A demand paging mechanism is very much similar to a paging system with swapping where processes stored in the secondary memory and pages are loaded only on demand, not in advance.

So, when a context switch occurs, the OS never copy any of the old program's pages from the disk or any of the new program's pages into the main memory. Instead, it will start executing the new program after loading the first page and fetches the program's pages, which are referenced.

During the program execution, if the program references a page that may not be available in the main memory because it was swapped, then the processor considers it as an invalid memory reference. That's because the page fault and transfers send control back from the program to the OS, which demands to store page back into the memory.

Page Replacement Algorithms:-

Types of Page Replacement Methods

Here, are some important Page replacement methods

- FIFO
- Optimal Algorithm
- LRU Page Replacement

FIFO Page Replacement

FIFO (First-in-first-out) is a simple implementation method. In this method, memory selects the page for a replacement that has been in the virtual address of the memory for the longest time.

Features:

- Whenever a new page loaded, the page recently comes in the memory is removed. So, it is easy to decide which page requires to be removed as its identification number is always at the FIFO stack.
- The oldest page in the main memory is one that should be selected for replacement first.

Optimal Algorithm

The optimal page replacement method selects that page for a replacement for which the time to the next reference is the longest.

Features:

- Optimal algorithm results in the fewest number of page faults. This algorithm is difficult to implement.
- An optimal page-replacement algorithm method has the lowest page-fault rate of all algorithms. This algorithm exists and which should be called MIN or OPT.
- Replace the page which unlike to use for a longer period of time. It only uses the time when a page needs to be used.

LRU Page Replacement

The full form of LRU is the Least Recently Used page. This method helps OS to find page usage over a short period of time. This algorithm should be implemented by associating a counter with an even- page.

How does it work?

- Page, which has not been used for the longest time in the main memory, is the one that will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

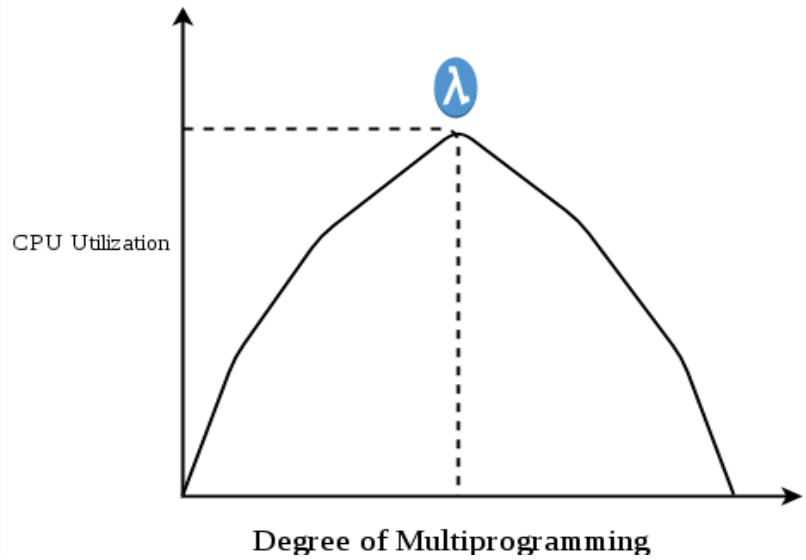
Features:

- The LRU replacement method has the highest count. This counter is also called aging registers, which specify their age and how much their associated pages should also be referenced.
- The page which hasn't been used for the longest time in the main memory is the one that should be selected for replacement.
- It also keeps a list and replaces pages by looking back into time.

Fault rate

Fault rate is a frequency with which a designed system or component fails. It is expressed in failures per unit of time. It is denoted by the Greek letter ? (lambda).

Thrashing:



At any given time, only few pages of any process are in main memory and therefore more processes can be maintained in memory. Furthermore time is saved because unused pages are not swapped in and out of memory. However, the OS must be clever about how it manages this scheme. In the steady state practically, all of main memory will be occupied with process's pages, so that the processor and OS has direct access to as many processes as possible. Thus when the OS brings one page in, it must throw another out. If it throws out a page just before it is used, then it will just have to get that page again almost immediately. Too much of this leads to a condition called Thrashing. The system spends most of its time swapping pages rather than executing instructions. So a good page replacement algorithm is required.

In the given diagram, initial degree of multi programming upto some extent of point(lamda), the CPU utilization is very high and the system resources are utilized 100%. But if we further increase the degree of multi programming the CPU utilization will drastically fall down and the system will spent more time only in the page replacement and the time taken to complete the execution of the process will increase. This situation in the system is called as thrashing.

Causes of Thrashing:

1. High degree of multiprogramming: If the number of processes keeps on increasing in the memory than number of frames allocated to each process will be decreased. So, less number of frames will be available to each process. Due to this, page fault will occur more frequently and more CPU time will be wasted in just swapping in and out of pages and the utilization will keep on decreasing.
For example:
Let free frames = 400

Case 1: Number of process = 100
Then, each process will get 4 frames.

Case 2: Number of process = 400
Each process will get 1 frame.

Case 2 is a condition of thrashing, as the number of processes is increased, frames per process are decreased. Hence CPU time will be consumed in just swapping pages.

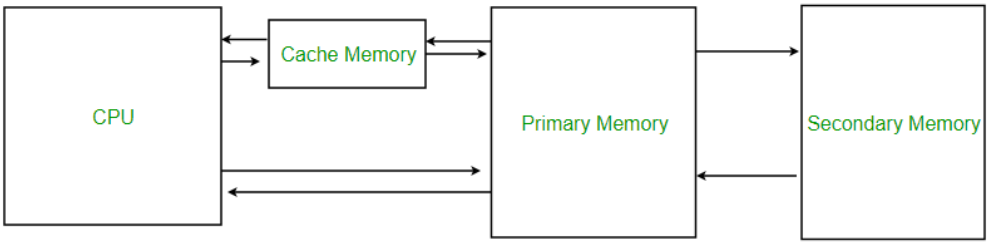
2. Lacks of Frames: If a process has less number of frames then less pages of that process will be able to reside in memory and hence more frequent swapping in and out will be required. This may lead to thrashing. Hence sufficient amount of frames must be allocated to each process in order to prevent thrashing.

Recovery of Thrashing:

- Do not allow the system to go into thrashing by instructing the long term scheduler not to bring the processes into memory after the threshold.
- If the system is already in thrashing then instruct the mid-term scheduler to suspend some of the processes so that we can recover the system from thrashing.

Cache Memory Organization-

Cache Memory is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Cache memory is costlier than main memory or disk memory but economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.
Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which stored instruction and data.



Levels of memory:

- **Level 1 or Register –** It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Most commonly used register is accumulator, Program counter, address register etc.
- **Level 2 or Cache memory –** It is the fastest memory which has faster access time where data is temporarily stored for faster access.
- **Level 3 or Main Memory –** It is memory on which computer works currently it is small in size and once power is off data no longer stays in this memory
- **Level 4 or Secondary Memory –** It is external memory which is not fast as main memory but data stays permanently in this memory

Cache Performance: When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.

- If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from cache
- If the processor **does not** find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

Hit ratio = hit / (hit + miss) = no. of hits/total accesses

We can improve Cache performance using higher cache block size, higher associativity, reduce miss rate, reduce miss penalty, and reduce Reduce the time to hit in the cache.

Application of Cache Memory –

1. Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory.
2. The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

Types of Cache –

- **Primary Cache –** A primary cache is always located on the processor chip. This cache is small and its access time is comparable to that of processor registers.
- **Secondary Cache –** Secondary cache is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2) cache. Often, the Level 2 cache is also housed on the processor chip.

Locality of reference – Since size of cache memory is less as compared to main memory. So to check which part of main memory should be given priority and loaded in cache is decided based on locality of reference.

Types of Locality of reference

1. **Spatial Locality of reference: -** This says that there is a chance that element will be present in the close proximity to the reference point and next time if again searched then more close proximity to the point of reference.
2. **Temporal Locality of reference: -** In this Least recently used algorithm will be used. Whenever there is page fault occurs within a word will not only load word in main memory but complete page fault will be loaded because spatial locality of reference rule says that if you are referring any word next word will be referred in its register that's why we load complete page table so the complete block will be loaded