Memory Management

1. Introduction

Memory is the physical device which is used to store programs or data on a temporary or permanent basis for use in a computer or other digital electronics device. There are various types of memories in a computer system and are accessed by various processes for their execution.

Memory management is an important and complex task of an operating system. Memory management involves treating main memory as a resource to be allocated and shared among number of active processes. It involves providing ways to allocate portions of memory to programs at their request, and freeing the memory for reuse when memory is no longer needed. As a role of memory management, an operating system is responsible for;

- Allocation and de-allocation of memory space as requested.
- Keep track of which parts of memory are currently being used and by whom.
- Efficient utilization when the memory resource is heavily used.

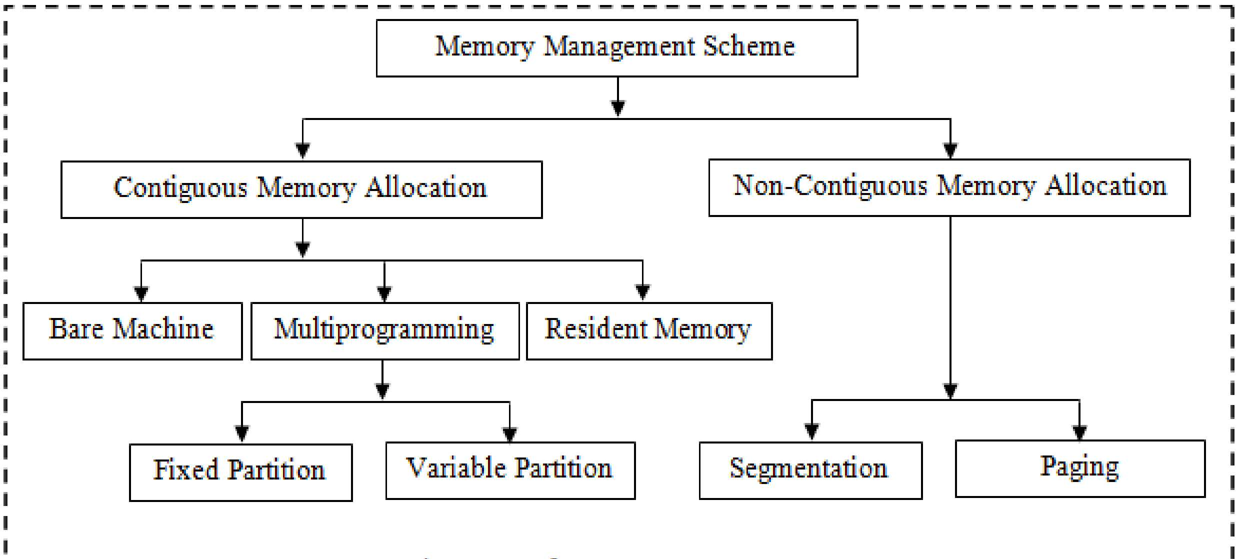
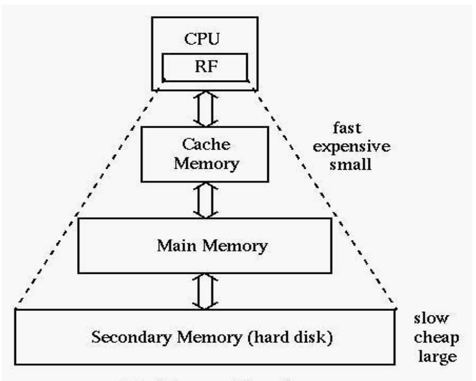


Fig: types of memory management

1.1. Storage organization and Memory Hierarchy



The Memory Hierarhcy

Memory hierarchy shows the access time and storage capacity of different computer memory devices. Devices having the fastest access time and smallest storage capacity are kept at the top of the hierarchy, and devices with slower access times but larger capacity and lower cost at lower levels. The memories in the hierarchy are as follows:

- At the top level of the memory hierarchy are the CPU's **general-purpose registers**. The registers provide the fastest access to data. The register file is also the smallest memory object in the memory hierarchy.
- The **Level One Cache (L1)** system is the faster than other memory but slower than CPU registers. The size is usually very small (typically between 4Kbytes and 32Kbytes), though much larger than the registers available on the CPU chip. The Level One Cache size is fixed on the CPU and we cannot expand it.
- The **Level Two Cache (L2)** is present on some CPUs. The Level Two Cache is generally much larger than the level one cache (e.g., 256KB or 512KBytes).
- Below the Level Two Cache system in the memory hierarchy there is the **main memory**. This is the general-purpose, relatively low-cost memory found in most computer systems. ***Main memory* or *internal memory*** is the only one directly accessible to the CPU. The CPU continuously reads instructions stored from them and executes them as required.
- **Secondary memory** also known as external memory or auxiliary storage is not directly accessible by the CPU. The data stored in such memories are permanently stored and are not lost even the power cut off i.e., non-volatile in nature. Such memories are relatively inexpensive than others. It has the largest storage capacity among other memory devices.

1.2. Storage allocation

Many files are stored on the same disk. The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly. **Storage allocation** is the assignment of particular areas of a magnetic disk to particular data or instructions. An allocation method refers to how disk block are allocated for files. There are various methods. Some major methods are:

- Contiguous memory allocation

- Linked allocation
- Indexed allocation, etc.

1.3. Contiguous versus Noncontiguous storage allocation

1.3.1. Contiguous storage allocation

Contiguous storage allocation is allocation technique in which each program occupied a single contiguous memory block i.e. each logical object is placed in a set of memory locations with strictly consecutive addresses. In these systems, the technique of multiprogramming was not possible. 1-D array is an example of contiguous memory allocation, where one array element is immediately followed by the next.

Contiguous memory allocation is one of the oldest memory allocation schemes. When a process needs to execute, memory is requested by the process. The size of the process is compared with the amount of contiguous main memory available to execute the process. If sufficient contiguous memory is found, the process is allocated memory to start its execution. Otherwise, it is added to a queue of waiting processes until sufficient free contiguous memory is available.

The contiguous memory allocation scheme can be implemented in operating systems with the help of two registers, known as the **base** and **limit registers**. When a process is executing in main memory, its base register contains the starting address of the memory location where the process is executing, while the amount of bytes consumed by the process is stored in the limit register. A process does not directly refer to the actual address for a corresponding memory location. Instead, it uses a relative address with respect to its base register. All addresses referred by a program are considered as virtual addresses. The CPU generates the logical or virtual address, which is converted into an actual address with the help of the memory management unit (MMU). The base address register is used for address translation by the MMU. Thus, a physical address is calculated as follows:

Physical Address = Base register address + Logical address/Virtual address.

The address of any memory location referenced by a process is checked to ensure that it does not refer to an address of a neighboring process. This processing security is handled by the underlying operating system.

Features of contiguous storage allocation:

- It is oldest technique.
- Contiguous disk space allocation is very simple to implement.
- The entire file can be read from the disk in a single operation i.e. data can be accessed more quickly.
- The degree of multiprogramming is reduced due to processes waiting for free memory.
- The method of ***contiguous storage allocation*** can cause disk fragmentation. If file is removed and blocks become free. It is also not possible to compact the disk on the spot to compress the hole.

1.3.2. Noncontiguous storage allocation

In non-contiguous storage allocation, a program is divided into several blocks that may be placed in different parts of main memory i.e. it implies that a single logical object may be placed in non-consecutive sets of memory locations. It is more difficult for an operating system to control non-contiguous storage allocation. The benefit is that if main memory has many small holes available instead of a single large hole, then operating system can often load and execute a program that would otherwise need to wait. There are two mechanisms that are used to manage non-contiguous memory allocation and they are:

- Paging (System view) and

- Segmentation (User view)

1.4. Logical versus Physical Address Space:

An address generated by the CPU is a logical address whereas address actually available on memory unit is a physical address. Logical address is also known a Virtual address. Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme. The set of all logical addresses generated by a program is referred to as a logical address space. The set of all physical addresses corresponding to these logical addresses is referred to as a physical address space. The run-time mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device. MMU uses following mechanism to convert virtual address to physical address;

- The value in the base register is added to every address generated by a user process which is treated as offset at the time it is sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.
- The user program deals with virtual addresses; it never sees the real physical addresses.

2. Fragmentation:

It is a technique of partition of memory into different blocks or pages while a process are loaded or removed from the memory. The free memory space is broken into little pieces; such types of pieces may or may not be of any use to be allocated individually to any process. This problem is known as memory waste or fragmentation.

In other words, Fragmentation refers to the condition of a disk in which files are divided into pieces scattered around the disk. Fragmentation occurs naturally when we use a disk frequently such as creating, deleting, and modifying files. Fragmentation occurs in a dynamic memory allocation system when many of the free blocks are too small to satisfy any request. There are two types of fragmentation and they are:

- **External fragmentation:** It happens when a dynamic memory allocation algorithm allocates some memory and small pieces are left over that cannot be effectively used. If too much external fragmentation occurs, the amount of usable memory is drastically reduced. Total memory space exists to satisfy a request, but it is not contiguous.
- **Internal fragmentation:** Internal fragmentation is the space wasted inside of allocated memory blocks because of restriction on the allowed sizes of allocated blocks. The phenomenon, in which there is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition, is referred to as internal fragmentation. So, it is also known as unused memory within an allocated partition.

Fragmentation can be done by two methods and they are:

- ➢ Fixed partition multiprogramming
- ➢ Variable partition multiprogramming

3. Fixed Partition Multiprogramming (Static):

Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size. Memory Manager allocates a region to a process that best fits it. The easiest way to achieve multiprogramming is to divide memory into several partitions. When a job arrives, it can be put into the input queue for the smallest partition enough to hold it.

Advantages:

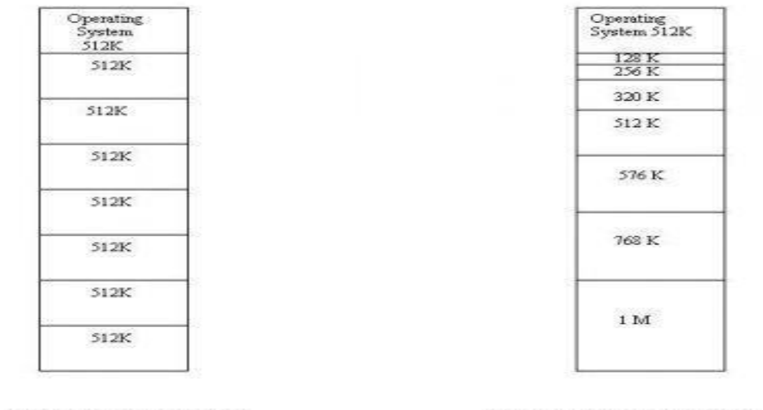
- It is simple to implement
- Overhead of operating system is small.

Disadvantages:

- Inefficient use of Memory due to internal fragmentation. Main memory utilization is extremely inefficient. Any program, no matter how small, occupies an entire partition. This creates internal fragmentation.

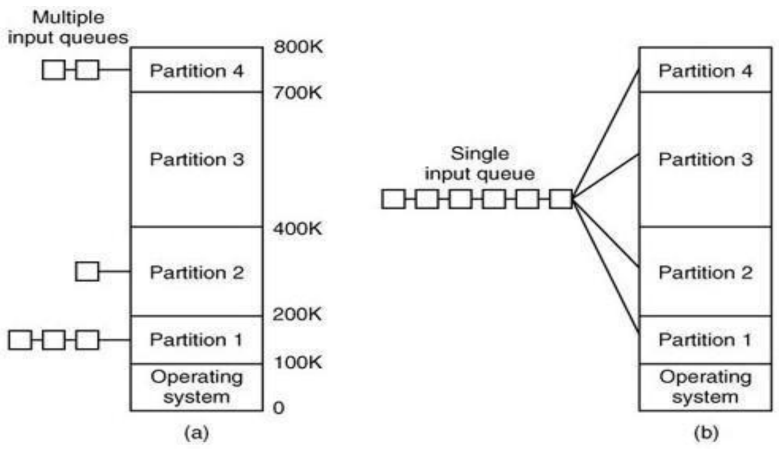
Fixed memory partitions are of two types**:**

a) **Equal size partitioning**: In this partition the size of partition of memory is always equal for all partition.
b) **Unequal size Partition:** In this partition the size of partition may differ from one another.



(a) Equal-size partitions (b) Unequal-size partitions

The numbers of programs (i.e; the degree of multiprogramming) are bounded by the number of partitions, whenever a partition become free the program nearest to the front of queue is loaded into the empty partition that fit it and run.



a) Fixed memory partitions with separate input queues for each partition.
b) Fixed memory partitions with a single input queue.

When the queue for a large partition is empty but the queue for a small partition is full, as is the case for partitions 1 and 3. Here small jobs have to wait to get into memory, even though plenty of memory is free.

An alternative organization is to maintain a single queue as in Fig.(b). Whenever a partition becomes free, the job closest to the front of the queue that fits in it could be loaded into the empty partition and run.

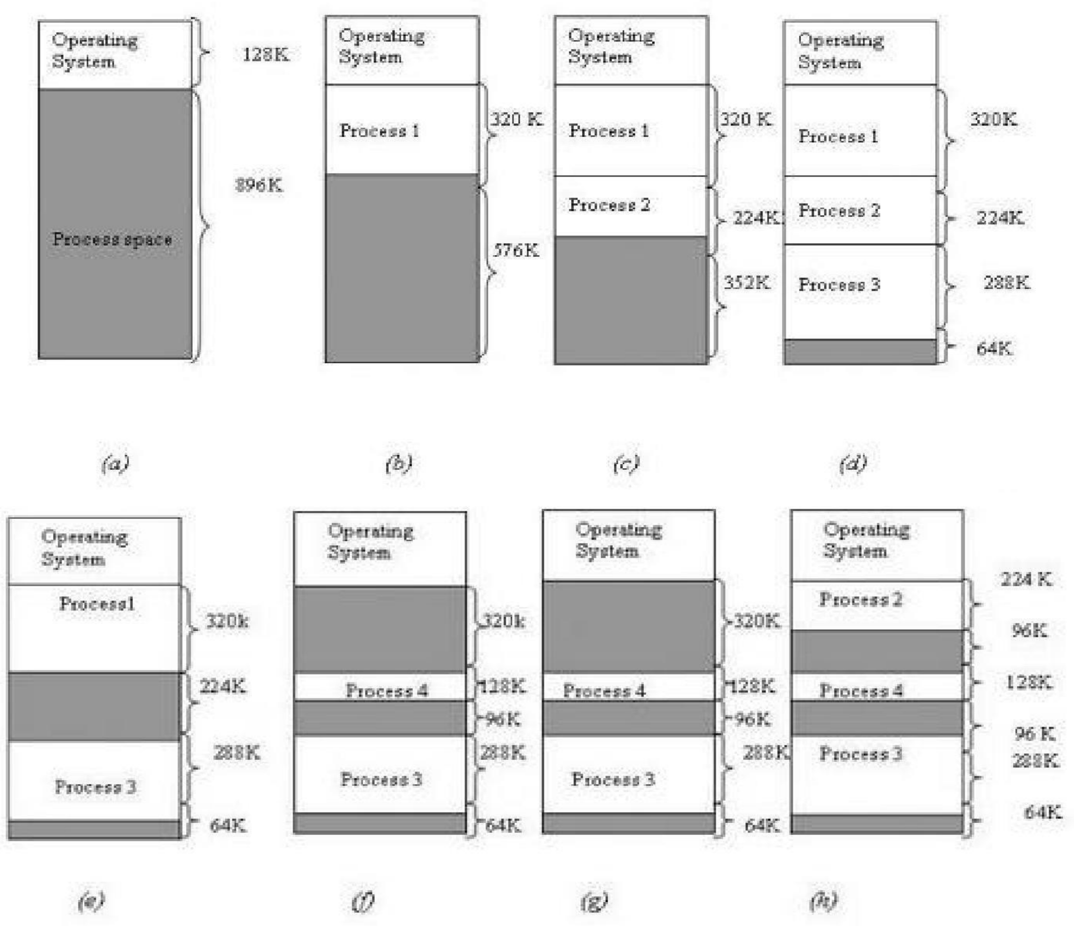
4. Variable Partition Multiprogramming (Dynamic):

In such scheme, partitions are of variable length and number. When a process is brought into main memory, it is allocated exactly as much memory as it requires and no more. Let us consider 1MB of memory is shown in following diagram.

- Initially, main memory is empty except for the operating system as shown in figure (a) below.
- The first three processes are loaded in, starting where the operating system ends, and occupy just enough space for each process (fig b, c and d). This leaves a "hole" at the end of memory that is too small for a fourth process. At some point, none of the processes in memory is ready.
- The operating system therefore swaps out process 2 (fig e), which leaves sufficient room to load a new

process, process 4 (fig f). Because process 4 is smaller than process 2, another small hole is created.

- Later, a point is reached at which none of the processes in main memory is ready, but process 2 in the Ready, suspend state is available. Because there is insufficient room in memory for process 2, the operating system swaps process 1 out (fig g) and swaps process 2 back in (fig h).
- This phenomenon is called ***external fragmentation*** i.e. the memory that is external to all partitions becomes increasingly fragmented.



5. Placement algorithm:

There are different placement algorithms such as;

5.1. First fit memory allocation:

- ❖ It find the first unused block of memory that can contain the process
- ❖ Faster in making allocation of memory space

5.2. Next fit:

- ❖ It works the same way as first fit, except that it keeps track of where it is whenever it finds a suitable hole. The next time it is called to find a hole, it starts searching the list from the place where it left off last time, instead of always at the beginning, as first fit does.

5.3. Best fit memory allocation:

- ❖ chooses the block that is closest in size to the request i.e. the smallest partition fitting the requirements
- ❖ Results in least wasted space
- ❖ Internal fragmentation reduced but not eliminated.
- ❖ Slower in making allocation

5.4. Worst fit:

- ❖ Allocates the largest free available block to the new job
- ❖ Opposite of best fit

Solved Numerical:

1. For the following partition of 100K, 500K, 200K, 300K and 600K (in order) place the processes 212K, 417K, 122K, and 426K in order according to the: Best fit and worst fit

Solution: Here, the given processes are;

P1 = 212K P2 = 417KP3 = 122K P4 = 426K

i. **Best fit**

Operating System	
	100K
P2	500K
P3	200K
P1	300K
P4	600K

Fig: Best Fit

ii. **Worst fit**

Operating System	
	100K
P2	500K
	200K
P3	300K
P1	600K

Fig: Worst Fit

2. For the following partition of 100K, 500K, 300K, 50K, and 600K in order place the process 212K, 417K, 122K and 40K in order to: First fit and Next fit.

Solution: Here, the given processes are:

P1 = 212K P2 = 417KP3 = 122KP4 = 40K

i. **First fit**

Operating System	
P4 (40K)	100K
P1 (212K)	500K
P3 (122K)	300K

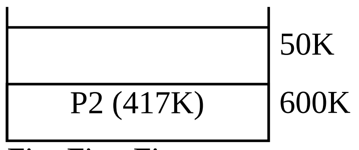


Fig: First Fit

ii. <u>Next fit</u>

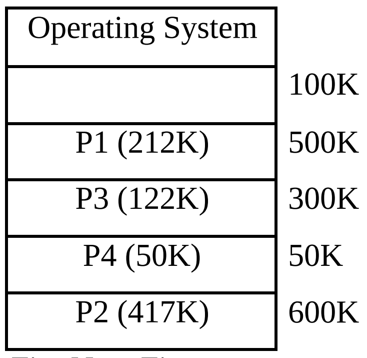


Fig: Next Fit

6. Relocation and Memory Protection:

Relocation is the ability to execute processes independently from their physical location in memory. It plays important role in memory management. The need for relocation is obvious when one considers that in a general purpose multiprogramming environment a program cannot know in advance (before execution i.e. at compiled time) what processes will be running in memory when it is executed nor how much memory the system has available for it, nor where it is located. Hence a program must be compiled and linked in such a way that it can later be loaded starting from an unpredictable address in memory, an address that can even change during the execution of the process itself, if any swapping occurs. So, relocation is the process of assigning load addresses to various parts of a program and adjusting the code and data in the program to reflect the assigned addresses.

Memory protection is a way to control memory access rights on a computer. It is a part of most modern processor 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. An attempt to access allocated 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.

7. Coalescing and Compaction:

Coalescing is the act of merging two adjacent free blocks of memory. When an application frees memory, gaps can fall in the memory segment that the application uses. Among other techniques, coalescing is used to reduce external fragmentation, but is not totally effective. Coalescing can be done as soon as blocks are freed, or it can be deferred until sometime later (known as deferred coalescing), or it might not be done at all. Coalescence and related techniques like heap compaction can be used in garbage collection.

Compaction: It is time consuming procedure. It involves shifting a program in memory in such a way that the program does not notice the change. This consideration requires that logical addresses be relocated dynamically, at execution time. If addresses are relocated only at load time we cannot support compact storage.

8. Virtual Memory

The main memory is considered as the physical memory in which many programs wants to reside. However, the size of such memory is limited and it cannot hold or load all the programs simultaneously. To overcome such types of problem, the overloaded inactive programs are swapped out from the main memory to hard disk for certain period of time. Such technique is known as virtual memory i.e. **Virtual memory** is a memory management system in a computer that temporarily stores inactive parts of the content of RAM on a disk, restoring it to RAM when quick access to it is needed.

Virtual memory permits software to use additional memory by utilizing the hard disc drive (HDD) as temporary storage. This technique involves the manipulation and management of memory by allowing the loading and execution of larger programs or multiple programs simultaneously. It is often considered more cost effective than purchasing additional RAM and enables the system to run more programs. However the process of mapping data back and forth between the hard drive and the RAM takes longer than accessing it directly from the memory i.e. depending upon virtual memory slows the computer.

8.1. Paging

Paging is one of the memory management schemes by which a computer can store and retrieve data from secondary storage for use in main memory. Paging is an important part of virtual memory implementation in most operating systems, allowing them to use disk storage for data that does not fit into physical RAM.

In this system, physical memory is broken down into fixed-sized blocks called **page frames** or simply **frames** and the logical (virtual) memory is broken down into the blocks of same sized blocks called **pages**. When a process is to be executed, its pages are loaded into any available memory frames form their source. The backing store is also divided into fixed-sized blocks that are of the same sizes as the memory frames.

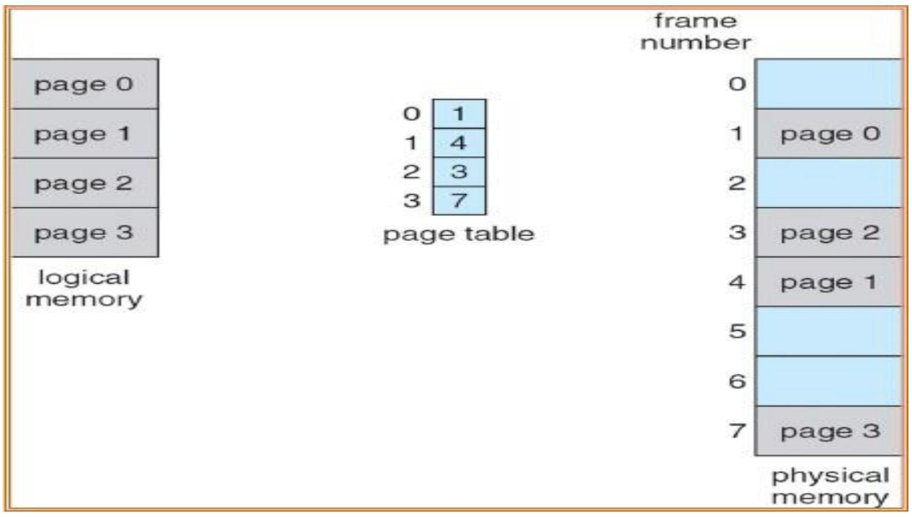


Fig: Paging concept

8.2. Page table:

A **page table** is the data structure used by a virtual memory system in a computer operating system to store the mapping between virtual address and physical address i.e. it is used to map virtual page number to physical page number. Every logical address is divided into page number (p) and page offset (d) and every physical address is divided into page frame number (f) and frame offset (d). The virtual page number (p) is used as an index into the page table to find the entry for that virtual page and page offset (d) is combined with base address to define the physical memory address that is sent to the memory unit. The content of page table represents the page frame number. The page table also indicates the status of each page i.e. whether space in the physical memory is allocated to it or not, if yes which frame is occupied currently. The mapping from virtual address to physical address is done by the mapping table (page table). If there are very large virtual address spaces, there would be problem. So, some methods to handle the large virtual address spaces are as follows;

- Inverted page table
- Multilevel page table
- Virtualized page table
- Nested page table

8.3. Block mapping:

A block allocation map is a data structure used to track disk blocks that are considered “in use”. Blocks may also be referred to as allocation units or clusters. Each directory entry could list 8 or 16 blocks (depending on disk format) that were allocated to a file. If a file used more blocks, additional directory entries would be needed. Thus, a single file could have multiple directory entries. A benefit of this method is the possibility to use sparse files by declaring a large file size but only allocating blocks that are actually used. A disadvantage of this method is that the disk may have free space (unallocated blocks) but data cannot be appended to a file because all directory entries are used.

8.4. Direct mapping:

It is a method of storing information for easy access on a computer. Recently used information is stored in a cache so the computer can quickly find the information the next time it is needed. With direct mapping, each piece of data in memory is assigned a space in the cache, which it shares with other pieces of data. Cache data is constantly being overwritten as new data is needed. It is one method of deciding where blocks of memory will be stored in the cache. Each block of memory is assigned a specific line in the cache. Since, the cache is smaller than the memory; multiple blocks will share a single line in the cache. If a line is already full when a new blocks needs to be written to it, an old block will be overwritten.

8.5. TLB (Translation Look Aside Buffers)

A **translation look aside buffer** (**TLB**) is a cache that memory management hardware uses to improve virtual address translation speed i.e. it acts as a cache for the page table. It is also referred to as the *address translation cache*. A TLB has a fixed number of slots that contain page table entries, which map virtual addresses to physical addresses. A typical TLB contains anywhere from 64 to 256 entries.

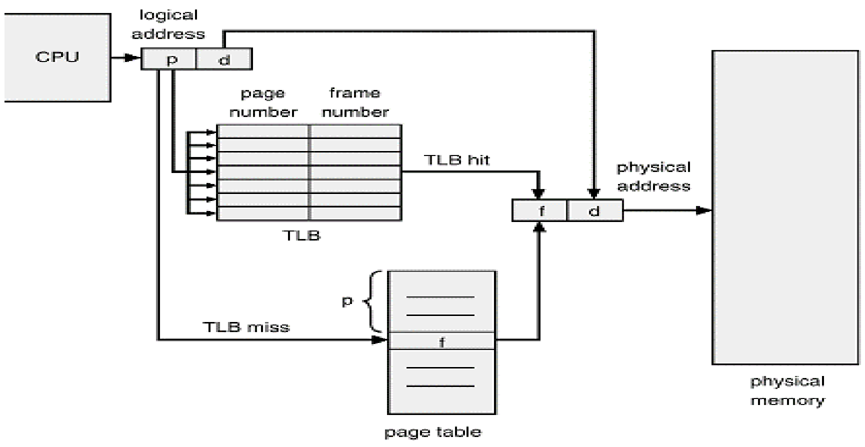
TLB is a table used in a virtual memory system that lists the physical address page number associated with each virtual address page number. A TLB is used in conjunction with a cache whose tags are based on virtual addresses. The virtual address is presented simultaneously to the TLB and to the cache so that cache access and the virtual-to- physical address translation can proceed in parallel. If the requested address is not cached then the physical address is used to locate the data in main memory. The alternative would be to place the translation table between the cache and main memory so that it will only be activated once there was a cache miss. A translation lookaside buffer (TLB) has a fixed number of slots containing the following entries:

Page table entries, which map virtual addresses to

- Physical addressees
- intermediate table addresses

Segment table entries, which map virtual addresses to

- segment addresses
- intermediate table addresses
- Page table addresses.



The organization of hardware of paging with TLB is shown in above diagram. Whenever a program performs a memory reference, the virtual address sent to the TLB to determine if it contains a translation for the address i.e. the program will first examine the TLB. If the desired page table entry is present or mapped i.e. a TLB hit occurs, then the frame number is retrieved and the real address is formed. If desired page table entry is not present in TLB i.e. a TLB miss occurs, then the processor uses the page number to index the process page table and examine the corresponding page entry. Hence TLB is a memory cache which as a fast execution processing element.

8.6. Page Fault

The main functions of paging are performed when a program tries to access pages that are not currently mapped to physical memory (RAM). This situation is known as a **page fault**. The operating system must then take control and handle the page fault, in a manner invisible to the program. Therefore, the operating system must:

- Determine the location of the data in secondary storage.
- Obtain an empty page frame in RAM to use as a container for the data.
- Load the requested data into the available page frame.
- Update the page table to refer to the new page frame.
- Return control to the program, transparently retrying the instruction that caused the page fault.

If there is not enough available RAM when obtaining an empty page frame, a page replacement algorithm is used to choose an existing page frame for eviction.

8.7. Thrashing

Thrashing is a condition in which excessive paging operations are taking place i.e. it refer to any situation in which multiple processes are competing for the same resource, and the excessive swapping back and forth between connections causes a slowdown. This causes the performance of the computer to degrade or collapse. Thrashing is often caused:

- when the system does not have enough memory,
- the system swap file is not properly configured, or
- Too much is running on the computer and it has low system resources.

To resolve hard drive thrashing, a user can do any of the below.

- Increase the amount of RAM in the computer.
- Decrease the amount of programs being run on the computer.

Adjust the size of the swap file.

9. Swapping:

Each and every process must be in main memory for execution. However, a process can be transferred temporarily out of memory to a backing storage device for certain time period and then brought back into main memory for continued execution. This process of transferring a process from main memory to backing storage and vice-versa is known as **Swapping.** Transformation of process from main memory to backing memory is called *swapped* out and from backing memory to main memory is known as *swapped in.* This scheme allows more processes to be run than can be fit into memory at one time.

For example:

Let us consider a multiprogramming environment with a round-robin CPU-scheduling algorithm. When a quantum expires, the memory management will start to swap out the process that just finished and to swap another process into the memory space that has been freed. Normally, a process that is swapped out will be swapped back into the same memory space it occupied previously.

Swapping policy is used for priority-based scheduling algorithms. If a higher-priority process arrives and wants services, the memory manager can swap out the lower-process and then load and execute the higher-priority process. When the higher-priority process finishes, the lower-priority process can be swapped backed in and continued. Sometimes this variant of swapping is called ***roll out, roll in.***

However, all the process cannot be swapped. If we want to swap a process, we must be sure that the process is completely idle. For example, if a process is waiting for an I/O operation then it cannot be swapped to free up memory. Swapping process can be shown in following figure.

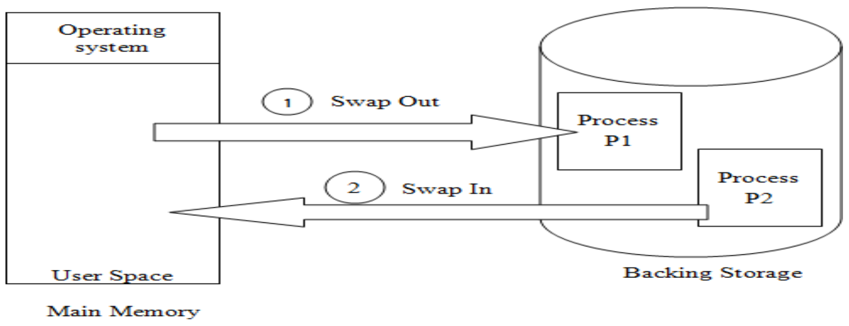


Fig: Swapping of two processes using a disk as a backing storage

10. Page Replacement Algorithm

When a page fault occurs, the operating system has to choose a page to remove from memory to make room for the page that has to be brought in. The page replacement is done by swapping the required pages from backup storage to main memory and vice-versa. A page replacement algorithm is evaluated by running the particular algorithm on a string of memory references and computes the page faults.

The basic page replacement method:

a) Find the location of the desired page on the disk.
b) Find a free frame:
   - If there is a free frame, use it.
   - If there is no free frame, use a page replacement algorithm to select a victim frame.
   - Write the victim frame to the disk; change the page and frame tables accordingly.
c) Read the desired page into the newly free frame; change the page and frame tables.
d) Restart the user process.

Example:

Let us consider the reference as (for a memory with 3 frames):

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

10.1. First in First out (FIFO) Algorithm

It is the simplest page replacement algorithm. The oldest page which has spent the longest time in memory is chosen and replaced. This algorithm is implemented with the help of FIFO queue to hold the page in memory. A page is inserted at the rear (tail) of queue and is replaced at the front (head) of queue. It is easy to implement and understand. But its performance is not always good. It may replace an active page to bring a new page and thus may cause a page fault of that page immediately.

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*
7	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
7	7	7	2		2	2	4	4	4	0			0	0			7	7	7
	0	0	0		3	3	3	2	2	2			1	1			1	0	0
		1	1		1	0	0	0	3	3			3	2			2	2	1
P	P	P	P		P	P	P	P	P	P			P	P			P	P	P

Fig: FIFO page replacement algorithm

There are 15 faults.

10.2. Optimal Page (OPT) Replacement Algorithm

It replaces the page that will not be used for the longest period of time i.e. it removes the page that will not be needed in the most distance in future. It has lowest page fault rate among all of the algorithms. It is difficult to implement because it requires future knowledge of program behavior.

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*
7	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
7	7	7	2		2		2			2			2				7		
	0	0	0		0		4			0			0				0		
		1	1		3		3			3			1				1		
P	P	P	P		P		P			P			P				P		

Fig: Optimal page replacement algorithm

There are 9 page faults.

Difference between FIFO and OPT algorithm:

The key distinction between the FIFO and OPT algorithms is that

- The FIFO algorithm uses the time when a page was brought into memory.
- Whereas the OPT algorithm uses the time when a page is to be used.

10.3. Least Recently Used (LRU) Replacement Algorithm

This algorithm replaces the page that has *not been used* for the longest period of time. It is based on the observation that the pages that have not been used for long time will remain unused and will be replaced. In this algorithm, it is necessary to maintain a linked list of all pages in memory, with the most recently used page at the front and the least recently used page at the rear.

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*
7	**0**	**1**	**2**	**0**	**3**	**0**	**4**	**2**	**3**	**0**	**3**	**2**	**1**	**2**	**0**	**1**	**7**	**0**	**1**
7	7	7	2		2		4	4	4	0			1		1		1		
	0	0	0		0		0	0	3	3			3		0		0		
		1	1		3		3	2	2	2			2		2		7		
P	P	P	P		P		P	P	P	P			P		P		P		

Fig: LRU replacement algorithm

There are 12 page faults.

10.4. Not Recently Used (NRU) Page Replacement Algorithm

Two status bit associated with each page. **R** is set whenever the page is referenced (read or written). **M** is set when the page is written to (i.e., modified).
When a page fault occurs, the operating system inspects all the pages and divides them into four categories based on the current values of their R and M bits:

- Class 0: not referenced, not modified.
- Class 1: not referenced, modified.
- Class 2: referenced, not modified.
- Class 3: referenced, modified.

The NRU algorithm removes a page at random from the lowest numbered non-empty class.

10.5. Second Chance Page (SCP) Replacement Algorithm

It is a simple modification to FIFO that avoids the problem of heavily used page. It inspects the R bit if it is 0, the page is both old and unused, and so it is replaced immediately. If the R bit is 1, the bit is cleared and the page is skipped. The page is put onto the end of the list of pages, and its load time is updated as though it had just arrived in memory. Then the search continues.

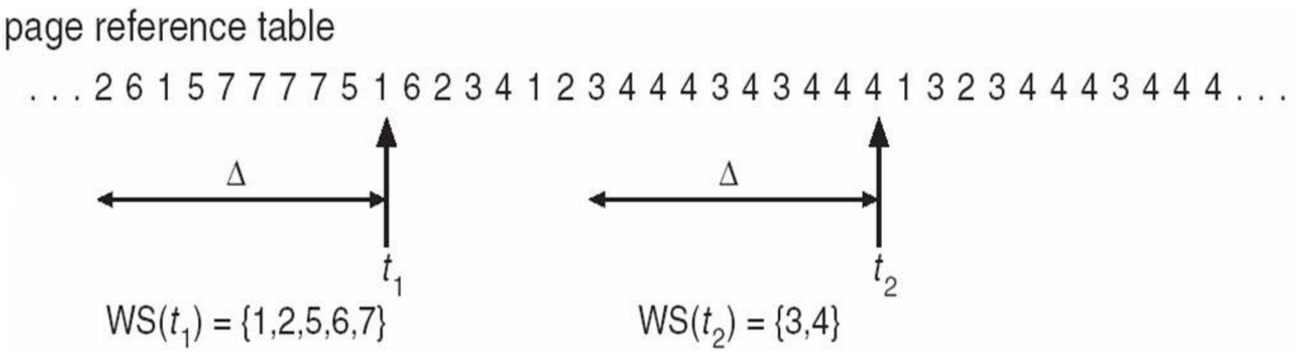
10.6. Working set (WS) page replacement algorithm:

In multiprogramming, processes are frequently moved to disk to let other process have a turn at the CPU. What to do when a process just swapped out and another process has to load? The set of pages that a process is currently using is called its working-set window.

$\Delta \equiv$working-set window $\equiv$ a fixed number of page references

For example: let us assume that there are 10,000 instructions. If the entire working set is in memory, the process will run without causing many faults until it moves into another execution. Otherwise, excessive page fault might occur called thrashing. Many paging systems try to keep track of each process' working set and make sure that it in memory before the process run- working set model or pre-paging. The working set of pages of process, *WS(t, Δ)* at time t, is the set of pages referenced by the processes in time interval t-k to t.

Ex: Working-set-model with Δ = 10.



10.7. WS Clock Page Replacement Algorithm

It is also known as improved WS algorithm. The hand of the clock points to the oldest page. When a page fault occurs, the page being pointed to by the hand is inspected. If its R bit is 0, the page is expelled, the new page is inserted into the clock in its place, and the hand is advanced one position. If R is 1, it is cleared and the hand is advanced to the next page. This process is repeated until a page is found with R = 0.

This algorithm keeps all the page frames on a circular list in the form of a clock, as shown in following figure.

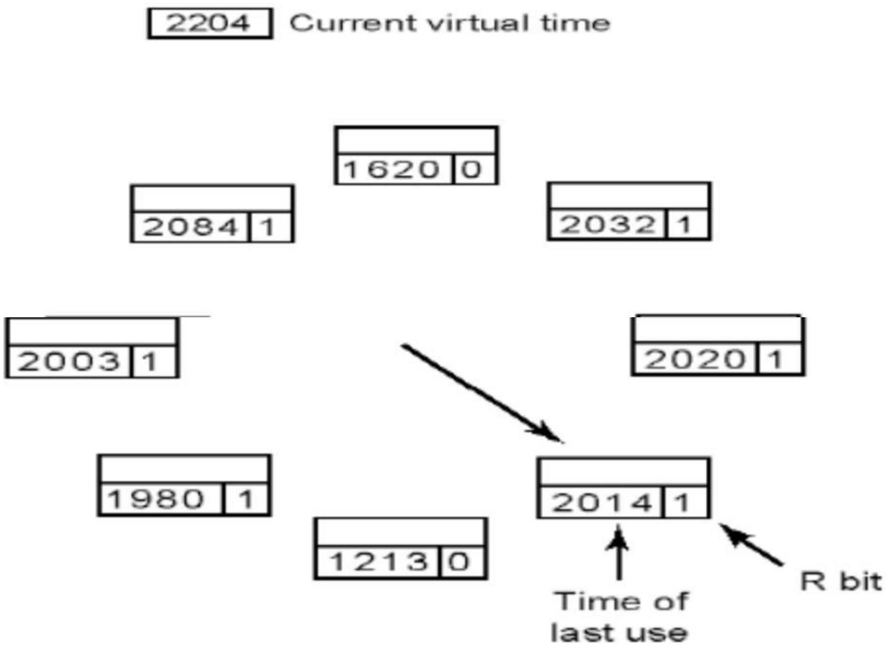


Fig: WS clock page replacement algorithm

11. Segmentation:

Segmentation is a memory management technique that breaks memory into logical pieces where each piece represents a group of related information. It is different from paging as pages are physical in nature and hence are fixed in size, whereas the segments are logical in nature and hence are variable size. Unlike paging, segments are having varying sizes and thus eliminate internal fragmentation. External fragmentation still exists but to lesser extent.

It supports the user view of the memory rather than system view as supported by paging. In segmentation we divide the logical address space into different segments. Each segment has a name and length which is loaded into physical memory as it is. For simplicity, the segments are referred by a segment number, rather than a segment name. Each segment consists of linear sequence of addresses starting from 0 to maximum value depending upon the size of segment. A table stores the information about all such segments and is called Global Descriptor Table (GDT). A GDT entry is called Global Descriptor.

Address generated by CPU is divided into **segment number(s)** and **segment offset (o).** The **segment number** is used as an index into a segment table. The segment table contains base address of each segment in physical memory and a limit of segment. The s**egment offset** is first checked against limit and then is combined with base address to define the physical memory address.

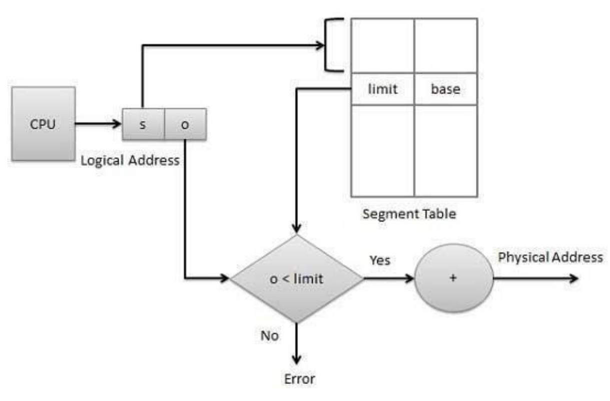


Fig: Segmentation Hardware

A memory management unit (MMU) is responsible for translating a segment and offset within that segment into a memory address, and for performing checks to make sure the translation can be done and the reference to that segment and offset is permitted. The size of a segment can increase or decrease depending upon the data stored or nature of operation performed on that segment. This change of size doesn't affect other segments in the memory. Segmentation can be implemented with or without paging

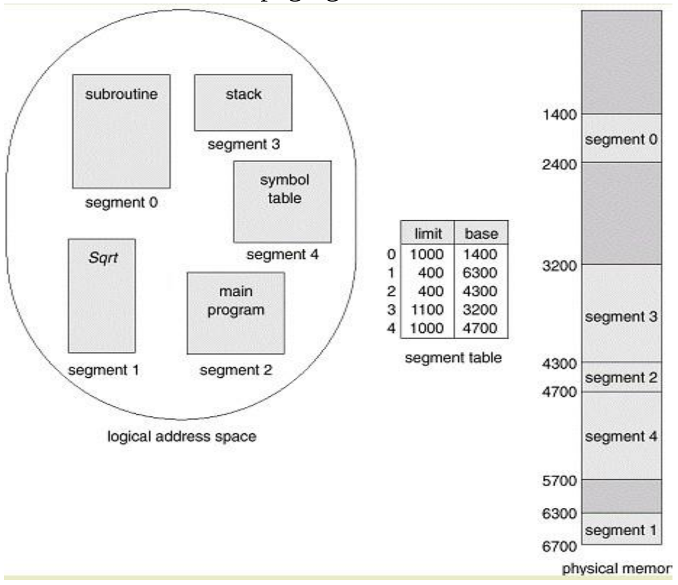


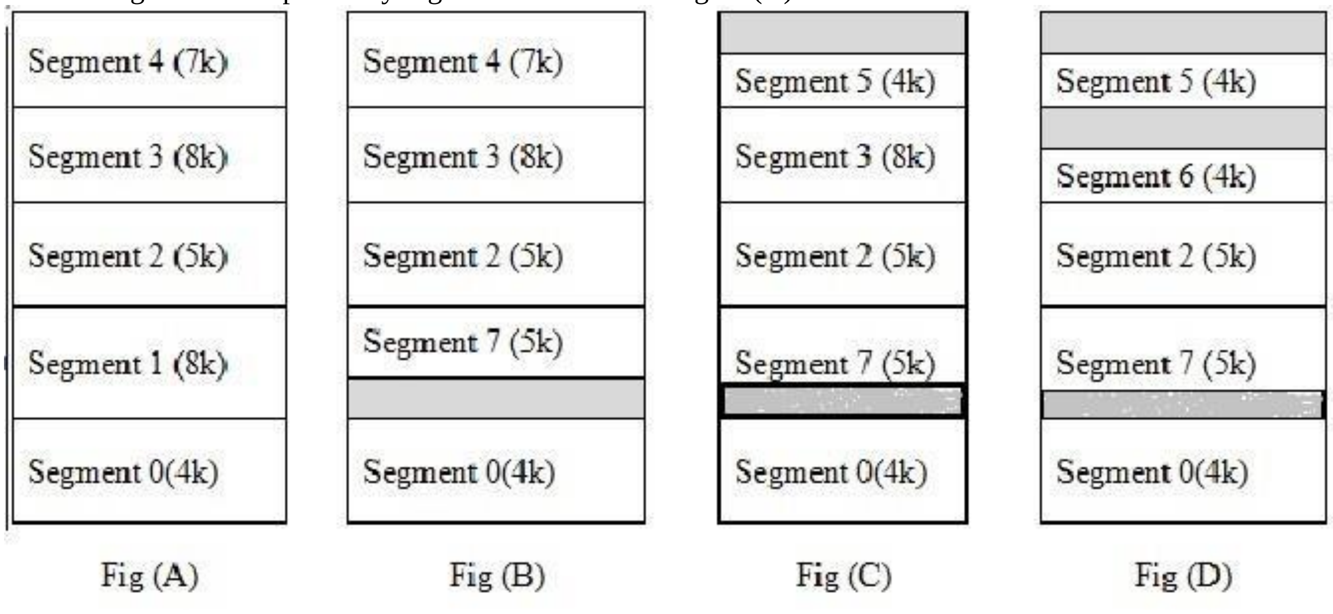
Fig: Segmentation

11.1. Implementation of Pure Segmentation

The implementation of segment differs from paging. Paging is fixed size and segment is not. Following figure shows the example of segmentation.

- Initially there are 5 segments.
- Now segment 1 is removed and segment 7 which is smaller is put in the place, as shown in the figure (B), between segment 0 and segment 2 and there is an unused area i.e. holes.
- Then segment 4 is replaced by segment 5 as shown in figure (C).

- Segment 3 is replaced by segment 6 as shown in figure (D).



11.2. Segmentation with Paging

Segments can be of different lengths, so it is harder to find a place for a segment in memory than a page. If the segments are large it may be inconvenient or even impossible to keep them in main memory entirely. This leads to idea of paging them so that only those pages that are actually needed have to be brought in main memory. With segmented virtual memory, we get the benefits of virtual memory but we still have to do dynamic storage allocation of physical memory. In order to avoid this, it is possible to combine segmentation and paging into a two-level virtual memory system. Each segment descriptor points to page table for that segment. This gives some of the advantages of paging (easy placement) with some of the advantages of segments (logical division of the program).

Some operating systems allow for the combination of segmentation with paging. If the size of a segment exceeds the size of main memory, the segment may be divided into equal size pages. The virtual address consists of three parts: (1) segment number (2) the page within the segment and (3) the offset within the page. The segment number is used to find the segment descriptor and the address within the segment is used to find the page frame and the offset within that page.

Advantages of segmentation:

- It can share data in a controlled way with appropriate protection mechanisms.
- It can move segments independently.
- It can put segments on disk independently.
- No internal fragmentation

Disadvantages of segmentation:

- Fragmentation and complicated memory management
- The maximum size of segment is limited by the size of memory.

12. Comparison between paging and segmentation:

Paging	**Segmentation**
Block replacement is easy and blocks are of fixed-length.	Block replacement is hard and blocks are of variable-length.
It is invisible to application programmer	It is visible to application programmer.
No external fragmentation but there is internal fragmentation in unused portion of page.	No Internal Fragmentation but there is external fragmentation in unused portion of main memory.
It is a physical unit invisible to the user's program and is of fixed size	It is a logical unit visible to the user's program and is of arbitrary size

It maintains one address space.	It maintains multiple address spaces per process
Units of code and data are broken into separate pages.	It keeps the block of code or data as a single unit.
It facilitates sharing of procedures between users.	No sharing of procedures between users.

Numericals:

1. Consider a swapping system in which memory consists of the following hole sizes in memory order: 10K, 4K, 20K, 18K, 7K, 9K, 12K, and 15K. Which hole is taken for successive segment requests of;

(a) 12K (b) 10K (c) 9K

For first fit, best fit, worst fit, and next fit.

Ans: First fit: (a) 20K (b) 10K (c) 18K
Best fit: (a) 12K (b) 10K (c) 9K
Worst fit: (a) 20K (b) 18K (c) 15K
Next fit: (a) 20 KB (b) 18K (c) 9K

2. A computer has four page frames. The time of loading, time of last access, and the R and M bits for each page are as shown below (the times are in clock ticks):

Page	Loaded	Last Ref.	R	M
0	126	280	1	0
1	230	265	0	1
2	140	270	0	0
3	110	285	1	1

(a) Which page will NRU replace?
(b) Which page will FIFO replace?
(c) Which page will LRU replace?
(d) Which page will second chance replace?

Solution: (a) Page 2 because R M = 0 0
(b) Page 3 because it is loaded at 110 (First In)
(c) Page 1 because is referenced at 265 (Least Recently)
(d) Page 2 because it is loaded at 140 and R M = 0 0.

3. If FIFO page replacement is used with three page frames and eight virtual pages, how many page faults will occur with the reference string 01723331005423 if the three frames are initially empty? Now repeat the problem for LRU and Optimal.

SOLUTION:

I. FIFO:

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*
0	**1**	**7**	**2**	**3**	**3**	**3**	**1**	**0**	**0**	**5**	**4**	**2**	**3**
0	0	0	2	2			2	0		0	0	2	2
	1	1	1	3			3	3		5	5	5	3
		7	7	7			1	1		1	4	4	4

P	P	P	P	P			P	P		P	P	P	P

11 Page Faults

II. LRU:

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*
0	**1**	**7**	**2**	**3**	**3**	**3**	**1**	**0**	**0**	**5**	**4**	**2**	**3**
0	0	0	2	2			2	0		0	0	2	2
	1	1	1	3			3	3		5	5	5	3
		7	7	7			1	1		1	4	4	4
P	P	P	P	P			P	P		P	P	P	P

11 Page Faults

III. Optimal:

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*
0	**1**	**7**	**2**	**3**	**3**	**3**	**1**	**0**	**0**	**5**	**4**	**2**	**3**
0	0	0	0	0	0	0	0	0	0	5	5	2	2
	1	1	1	1	1	1	1	1	1	1	4	4	4
		7	2	3	3	3	3	3	3	3	3	3	3
P	P	P	P	P						P	P	P	

8 Page Faults