Unit-3

Memory Management

Memory management deals with allocation and de allocation of space in main memory or RAM for programs or processes.
Performance of computer system is high when the CPU is kept busy at all times. To keep the CPU busy at all times, number of processes or programs is loaded into main memory at a time. So that, when a process makes an input/output request during its execution then the CPU is allocated to some other process.

Logical address

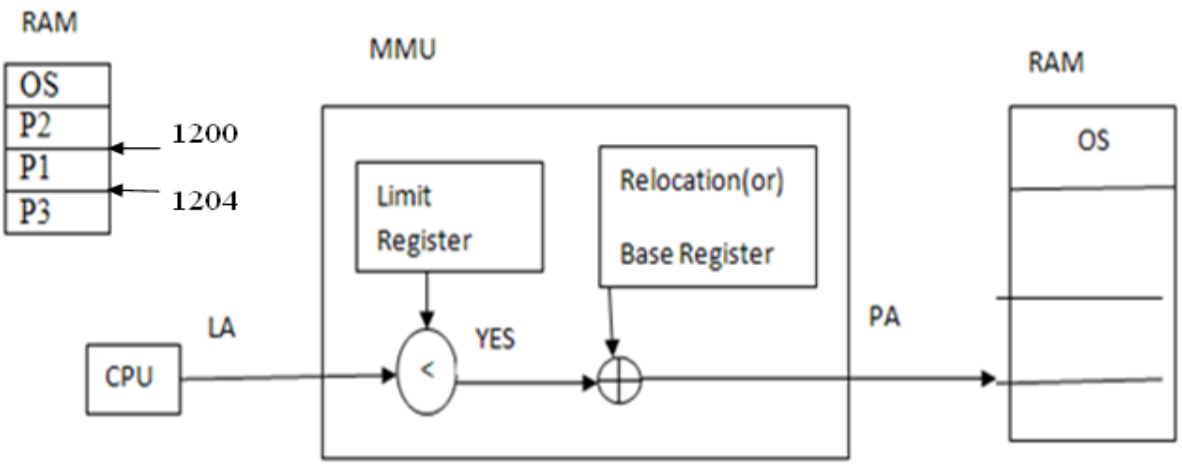
A program or process is collection of statements or instructions. Operating system loads the process in RAM when the process needs to be executed. CPU don't not know the location of process inside RAM. To execute the process, CPU generates addresses of statements in the process as 0, 1, 2, and so on. When CPU wants to execute 1st statement of the process, it generates the address 0. Similarly, when CPU wants to execute 2nd statement of the process, it generates the address 1. This generation of addresses is continued till the last statement of the process. The addresses (0, 1, 2, ….) generated by CPU for executing the statements of the program are called logical addresses.

Physical address

Physical addresses are addresses of locations in RAM where statements of the process are loaded. Physical address is generated by adding the logical address value to the starting location or position of the process in RAM. The starting location of process in the RAM is stored in a register called "Relocation Register".

Memory Management Unit (MMU)

MMU maps logical addresses to their corresponding physical addresses. This mapping is also called as relocation. Before starting the execution of a process, address of starting location of that process in RAM is loaded into Relocation or Base Register and number of statements in the process is loaded into Limit Register. For example, if a process P1 is loaded into RAM at address 1200 and the number of statements in the process P1 is 5 then when execution of process P1 is started, operating system loads address 1200 into Relocation Register and value 5 in Limit Register.
While executing any process, the logical address generated by CPU for executing a statement is compared with the value in Limit Register. If the logical address value is less than the value in Limit Register then the logical address is added to the value in Relocation Register to generate corresponding physical address. Otherwise, the MMU reports an error to the operating system.



Memory Management Techniques

Operating system can use any of the following techniques to load programs into main memory or RAM.

1. Contiguous Memory Allocation
2. Paging
3. Segmentation

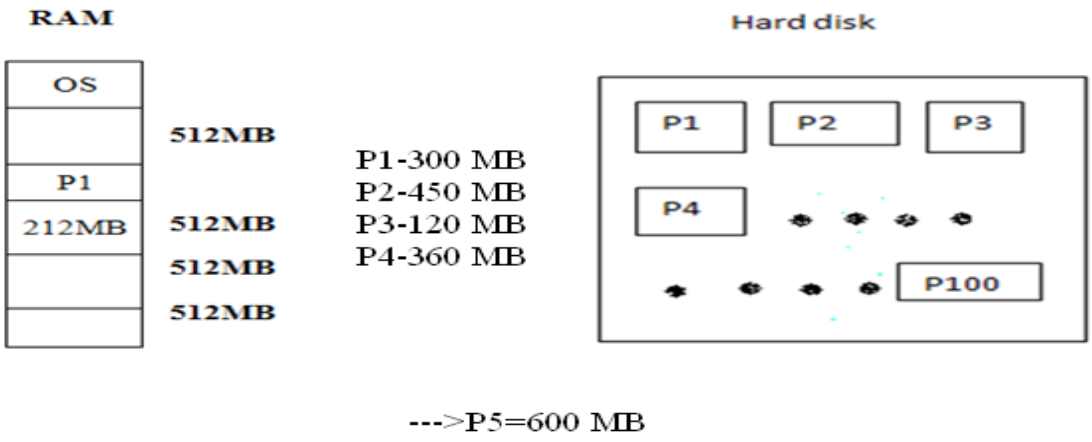
Contiguous Memory Allocation

With this technique, operating system loads entire program into RAM wherever required free space is available. Different contiguous memory allocation techniques are:

1) Equal size fixed partition
2) Unequal size fixed partition
3) Dynamic partition

Equal size fixed partition technique

Before loading any program into RAM, operating system divides RAM into number of equal size parts. Only one process can be loaded into each partition of RAM. Operating system can load a program into any free partition of RAM.



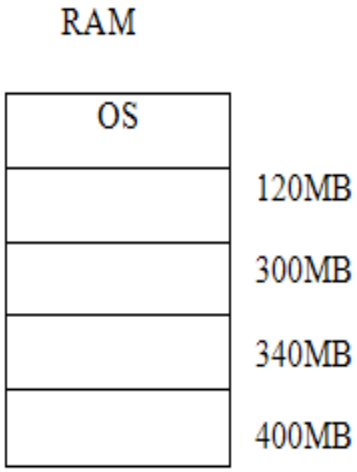
--->P5=600 MB

Advantage:
Operating system can select any free partition of RAM to load a program.

Disadvantages:
1) Some memory is wasted inside partition when size of program is less than size of partition. This wastage of memory is called "**internal fragmentation**". For example, when program P1 is loaded into 2nd partition of RAM as shown in above figure, 212 MB of memory in that partition is wasted.
2) If size of program to be loaded is greater than size of partition then it is not possible to load the process into RAM. For example, if the size of program P5 is 600 MB then it is not possible to load P5 into RAM.

Un-equal size fixed partition

Before loading any program into RAM, operating system divides RAM into number of parts of different sizes. Only one process can be loaded into each partition of RAM. Operating system has to select a suitable partition of RAM for loading a program.



2. What is fragmentation? Explain the differences between internal and external fragmentation.

Advantage

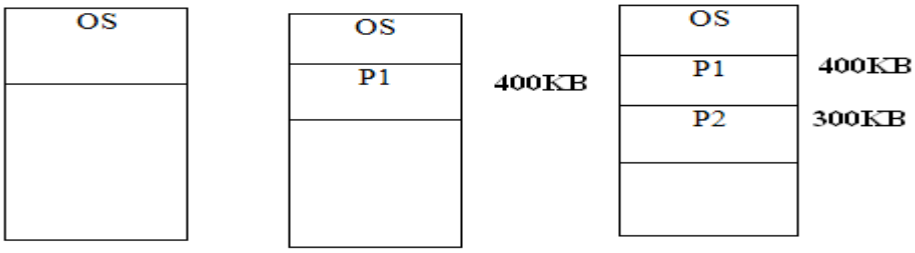
Internal Fragmentation is less compared to equal size fixed partition.

Disadvantages

1) Operating System has to select a suitable partition for loading a program into RAM.
2) Some memory is wasted inside partition when size of process is less than size of partition. This wastage of memory is called "**internal fragmentation**".
3) If size of process is greater than size of partition then it is not possible to load the process into RAM. For example, if the size of process P5 is 600 MB then it is not possible to load P5 into RAM.

Dynamic partitions

Operating system divides RAM into parts at the time of loading programs into RAM. Initially, RAM contains two parts. Into one part, Operating System is loaded and the other part is used for loading user programs. When a program P1 is loaded into RAM, RAM is divided into three parts. When a program say P2 is loaded, the RAM is divided into four parts as shown in below figure.



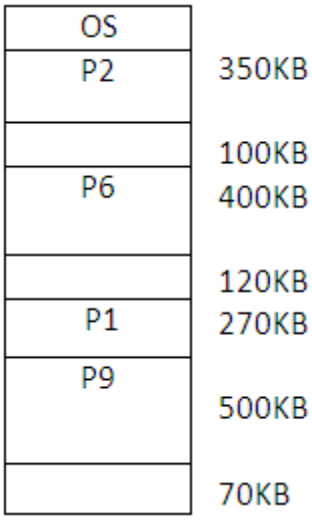
Advantage

There is no wastage of memory inside partitions. So, no internal fragmentation.

Disadvantages:

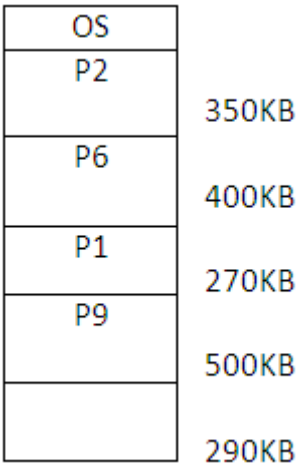
Some space is wasted outside of the partitions. This wastage of space is called as 'External Fragmentation'. Consider the following position of RAM which is the result of loading some programs into RAM and removing completed programs from RAM. If operating system wants to load a program say P20 with size 250 KB then it is not possible to load that program into RAM as there is no free part with size greater than or equal to 250KB in RAM.

To avoid external fragmentation **compaction** technique can be used.



Compaction

Compaction technique moves all free spaces in the RAM together. After applying compaction technique to the above position of RAM, the position of RAM becomes



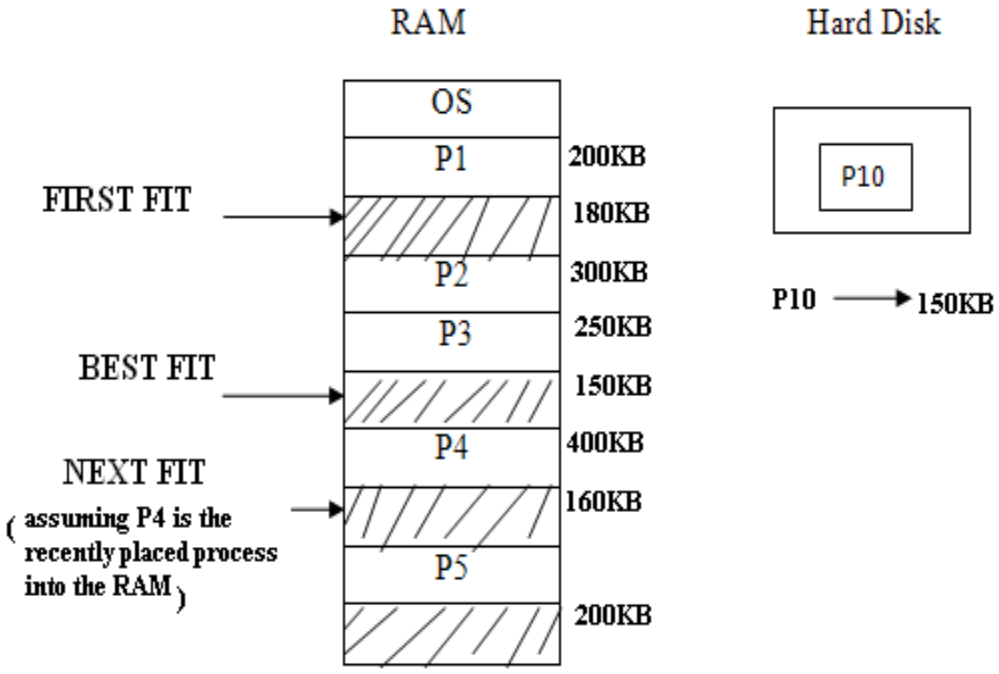
Now, operating system can load program P20 into RAM.

With unequal size fixed partition and dynamic partition, operating system has to select suitable partition in RAM for loading a program. Operating System uses any of the following placement algorithms to select suitable partition.

1. FIRST FIT
2. BEST FIT
3. NEXT FIT

- **First fit** algorithm scans RAM from starting to ending and identifies the first free partition whose size is greater than or equal to size of program.
- **Best fit** algorithm scans RAM from starting to ending and selects the free partition whose size is very close to the size of program.
- **Next fit** algorithm scans RAM from the partition where recent placement has occurred and selects the first free partition whose size is greater than or equal to size of the program.

Consider following position of RAM. To load a program P10 with size 150 KB, the partition selected by operating system with different algorithms is indicated in below diagram.

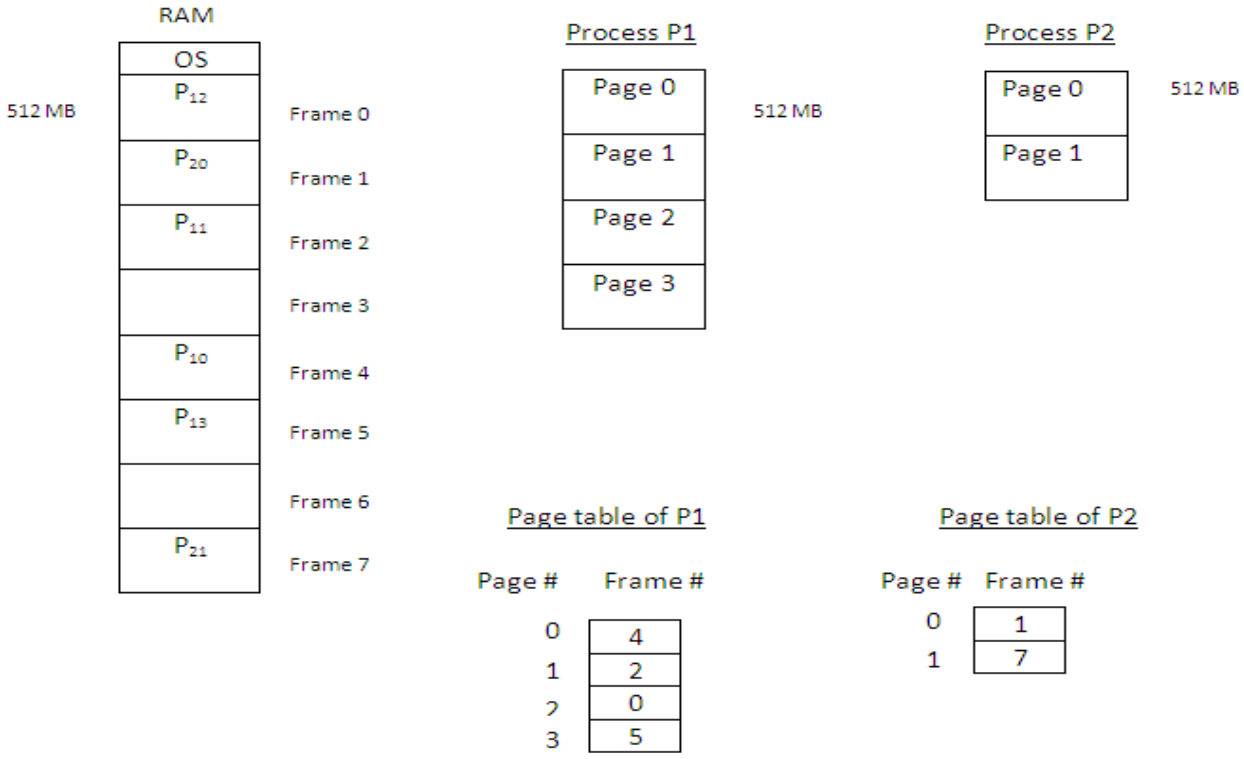


Paging

b. What is paging? Explain the hardware support given for paging.

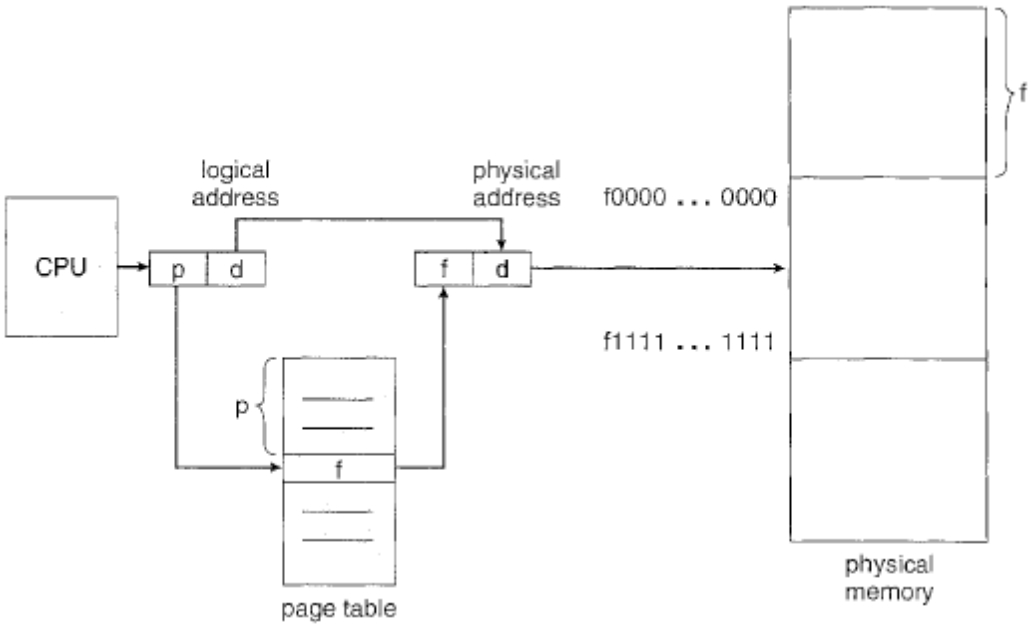
Before loading any program into RAM, RAM is divided into number of equal size frames. To load a program into RAM, operating system divides program into number of equal size pages. Size of page is equal to size of frame. Then, operating system loads pages of the program into RAM wherever free frames are available. Later, operating system creates a page table for the program. The number of entries in page table is equal to number of pages in the program. The entries of page table are indexed with page numbers. Operating system stores frame numbers of RAM into the entries of page table. Generally, the page size or frame size is selected as a power of 2.

The following diagram shows how programs are loaded into RAM with paging technique.



Conversion of Logical Address to Physical Address in Paging (or) Hardware support for Paging

To start execution of any process or program, operating system divides the program into equal size pages, loads these pages into free frames of RAM, create a page table for the program and stores frame numbers into entries of the page table. To execute a statement of the program, CPU generates logical address of that statement. This logical address is divided into two parts: page number (p) and offset (d). The page number is used as an index into the page table of the process and the corresponding frame number is identified. The identified frame number is appended with offset to get physical address of the statement. The statement in RAM at the generated physical address is transferred to CPU for execution. The following diagram depicts this conversion procedure. In below diagram, physical memory indicates RAM. Offset identifies the statement inside page or frame.



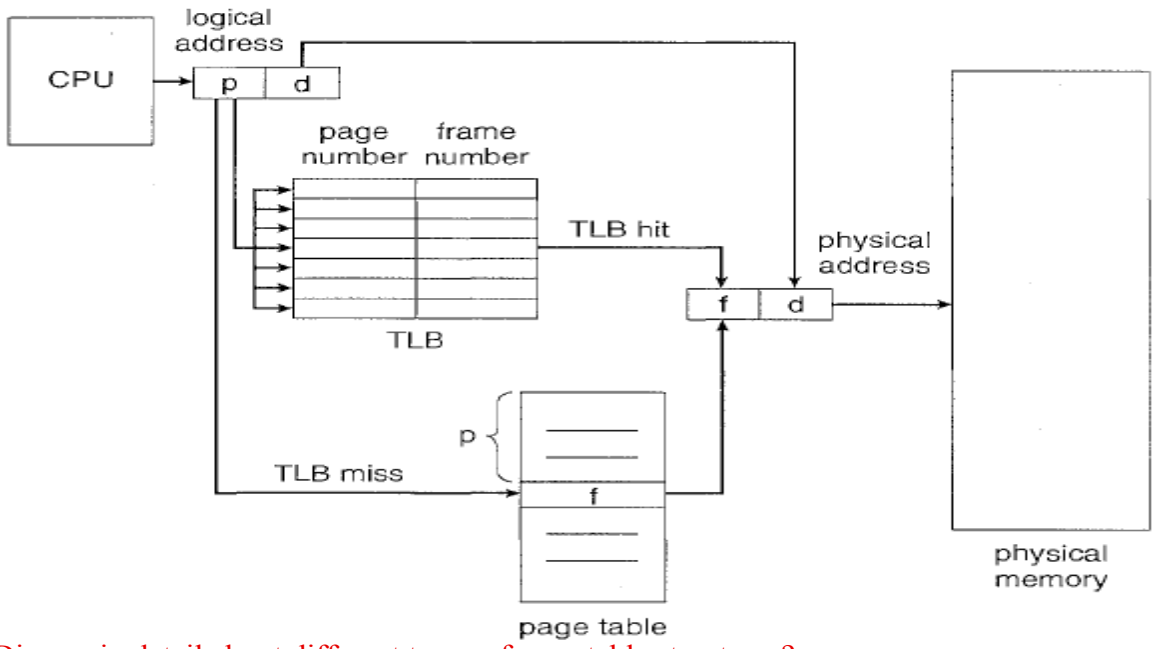
Translation Look-aside Buffer (TLB)

After creating page table of a process or program, operating system can store entries of that page table in either registers or RAM. Operating system stores entries of page table in registers of computer system when number of entries in page table of the process is less. Otherwise, stores entries of page table in RAM. If entries of page table are stored in registers then they can be accessed quickly. One memory access is enough for getting a statement of the process from RAM. If the entries of page table are stored in RAM then more time is required to access statements of the process from RAM. Two memory accesses are required for accessing each statement of the process. To get any statement from RAM, frame number of that statement needs to be obtained first. To get frame number of the statement, page table of process stored in RAM has to be accessed. For this, one memory access is required. One more memory access is for getting the statement from the identified frame of RAM.

To get statements of the process from RAM with one memory access, information about frequently used pages is stored in translation look-aside buffer (TLB). TLB contains number of entries. Each entry of TLB contains a page number and its corresponding frame number. TLB is associative in nature. All entries of TLB are searched at a time in parallel.

When CPU generates a logical address, that logical address is divided into two parts: page number and offset. The page number is first searched in TLB. If the page number is found in TLB (TLB hit) then corresponding frame number is appended with offset of logical address to generate corresponding physical address. Otherwise (TLB miss), the page number of logical address is used as an index into page table and corresponding frame number is identified and appended with offset of logical address to generate physical address.

The following diagram shows usage of TLB in paging.



3. Discuss in detail about different types of page table structures?

Different Structures for page table

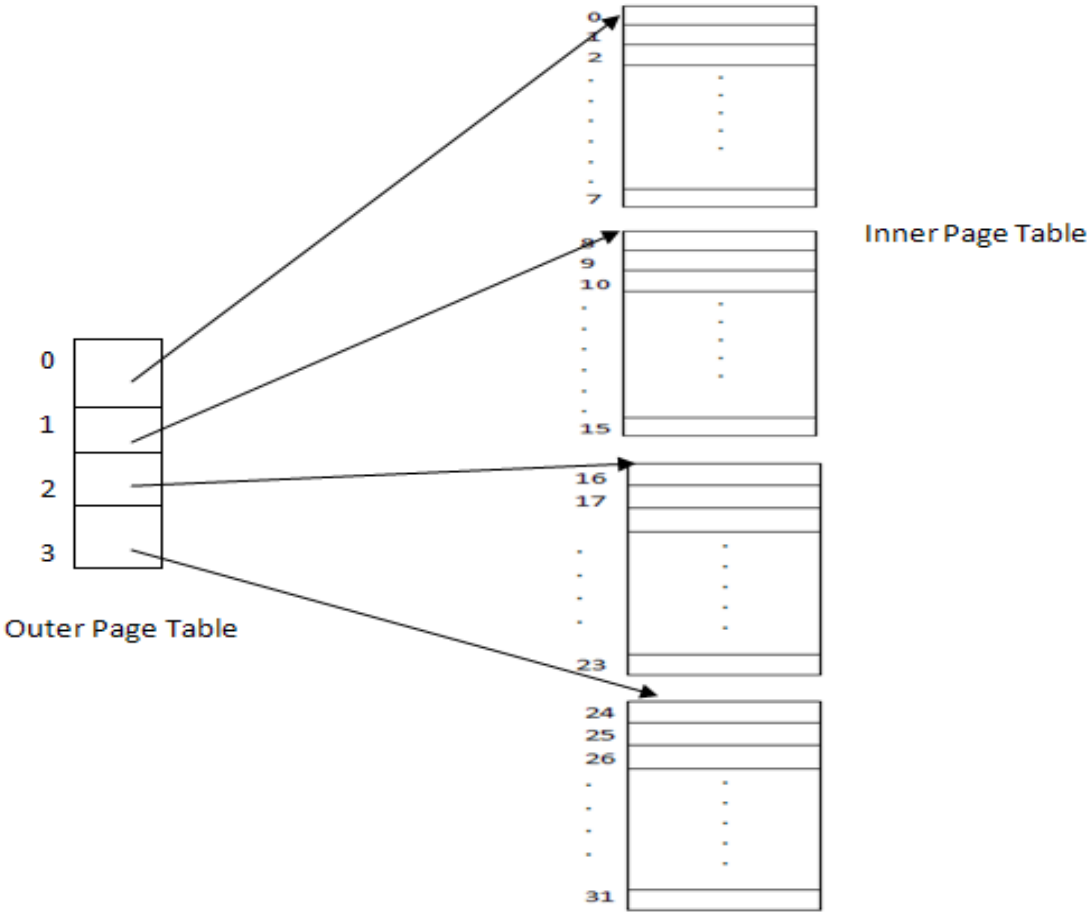
Operating system can use the following structures for maintaining the page table of a program.

1. Hierarchical structure
2. Hashed structure
3. Inverted structure

Hierarchical Page Table

The page table of a process occupies more space in RAM when number of entries in the page table of process is very high. To reduce space requirement in RAM, the page table of process is maintained in hierarchical structure. In hierarchical structure, the entries of page table are divided into number of page tables called inner page tables. To store addresses of inner page tables, an outer page table is created. The number of entries in outer page table is equal to number of inner page tables.

For example, if the page table of a process is containing 32 entries then these entries can be divided into four inner page tables (4x8=32). The addresses of these four inner page tables are stored in outer page table with four entries as shown in below diagram.



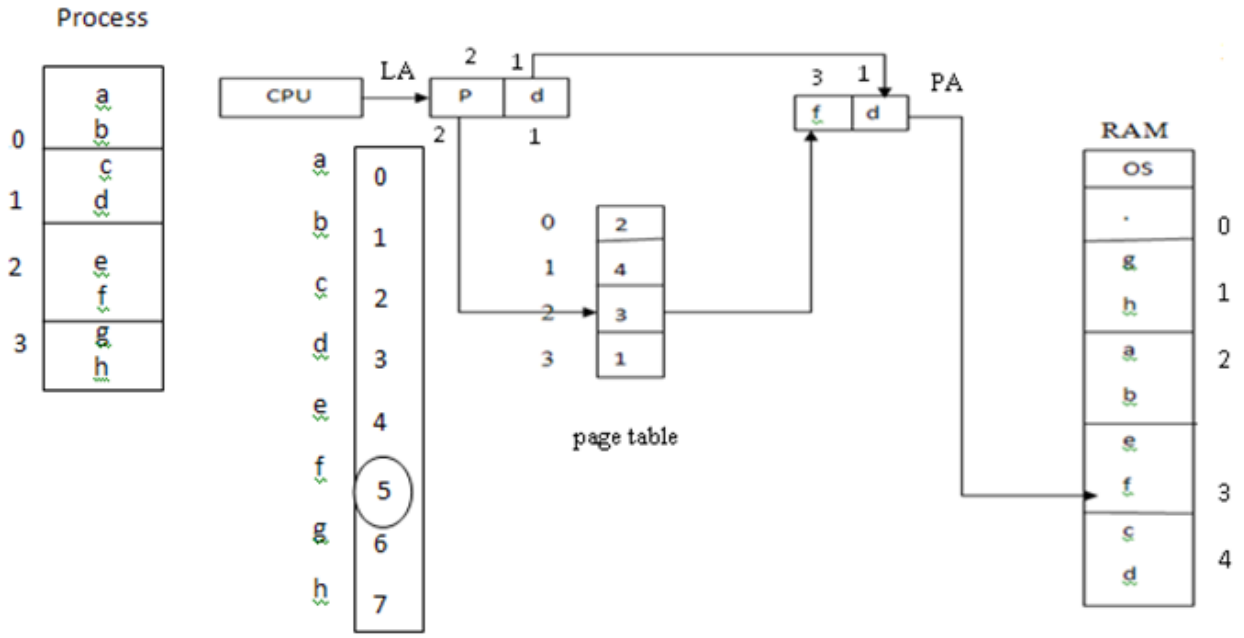
While executing the process, when CPU generates a logical address, that logical address is divided into three parts as shown below

Logical Address

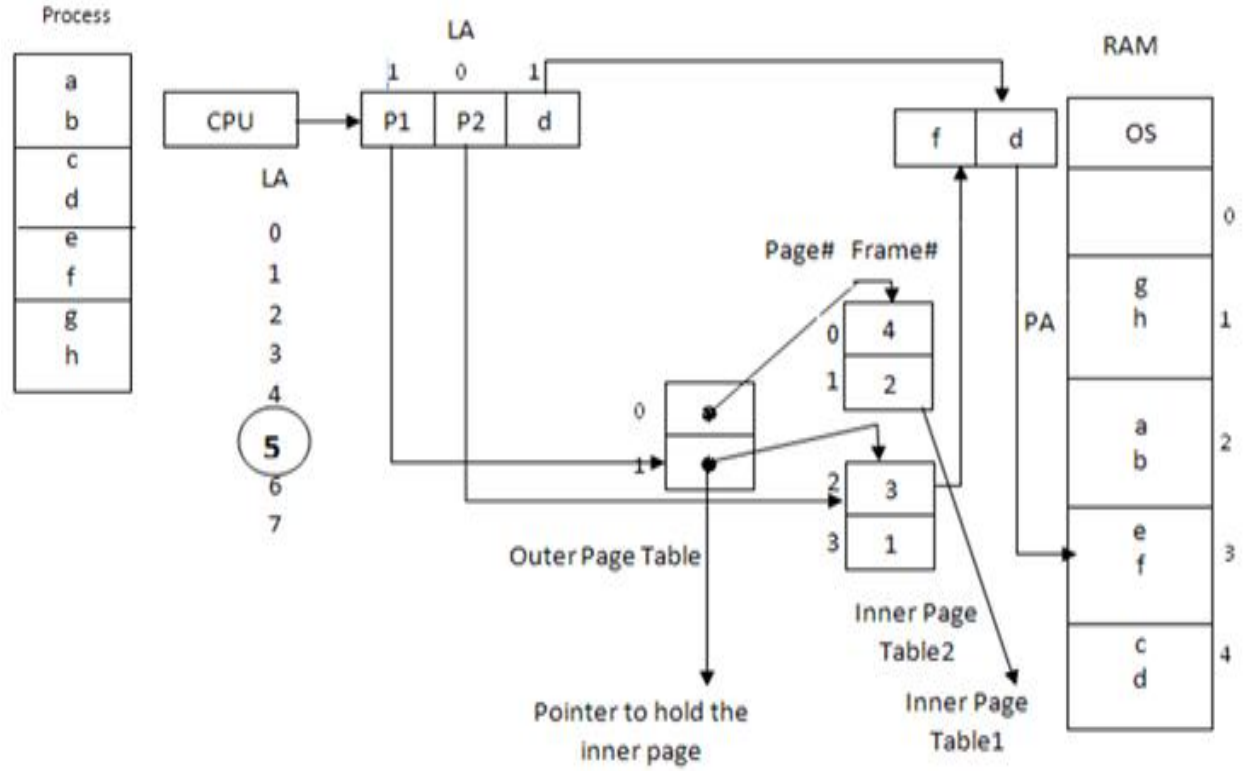
p1	p2	d

p1 is used as index into outer page table to identify the inner page table. p2 is used as index into the identified inner page table to identify the frame number. The identified frame number is appended with offset (d) of logical address to generate the corresponding physical address. The generated physical address identifies the statement in the RAM. Hierarchy can be extended to any number of levels.

For example, the below diagram shows a process containing four pages, loading of the four pages into RAM and mapping of logical address to corresponding physical address with single level page table.



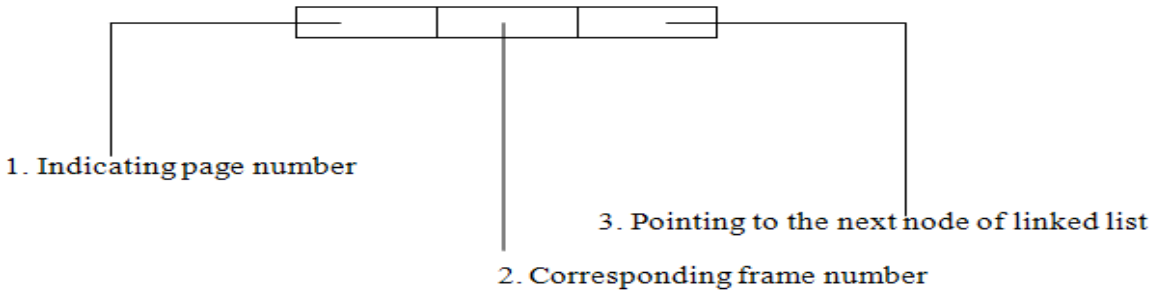
The below diagram shows the same process with four pages, loading of four pages into RAM and mapping of logical address to its corresponding physical address with hierarchical page table.



With hierarchical page table structure, the space requirement for the page table is reduced by loading the outer page table and one of the inner page tables instead of all inner page tables at any particular time during execution of the process.

Hashed Page Table

In this structure, the entries of page table of the process are distributed into number of linked lists. Each linked list contains number of nodes and each node contains three fields as shown below



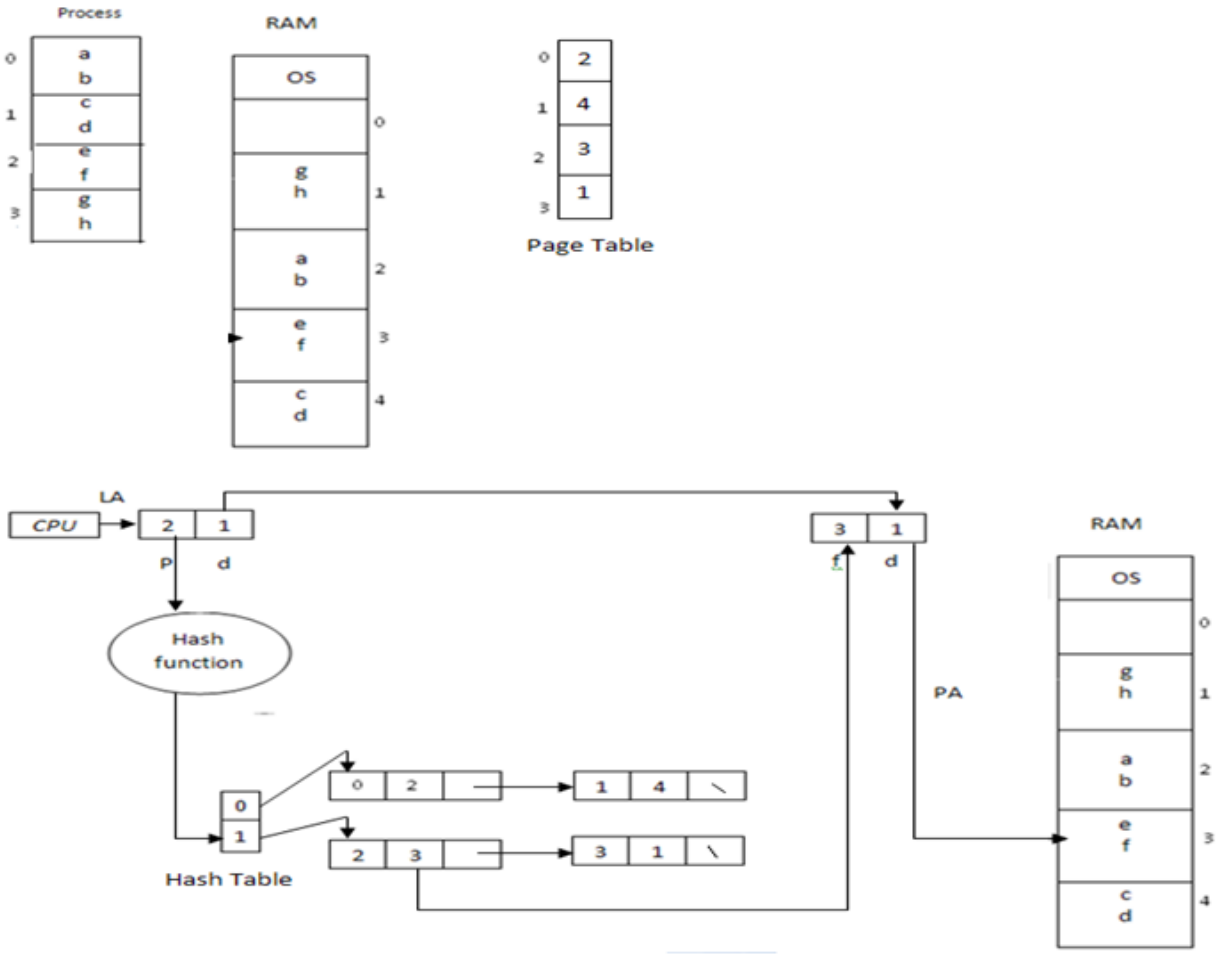
A hash table containing number of entries is maintained. Each entry of the hash table points to one of the linked lists.

While executing the process, when CPU generates a logical address, that logical address is divided into two parts called page number and offset as shown below

Logical Address

p	d

The page number is passed to the hash function. The hash function returns a value. This value is used as an index into the hash table to identify the entry of hash table. From the identified entry of hash table, address of the corresponding linked list is obtained. In this linked list, the page number value in each node is compared with the page number value of the logical address. Whenever a match occurs then the corresponding frame number is appended with offset of logical address to get the physical address. The following diagram shows this mapping.



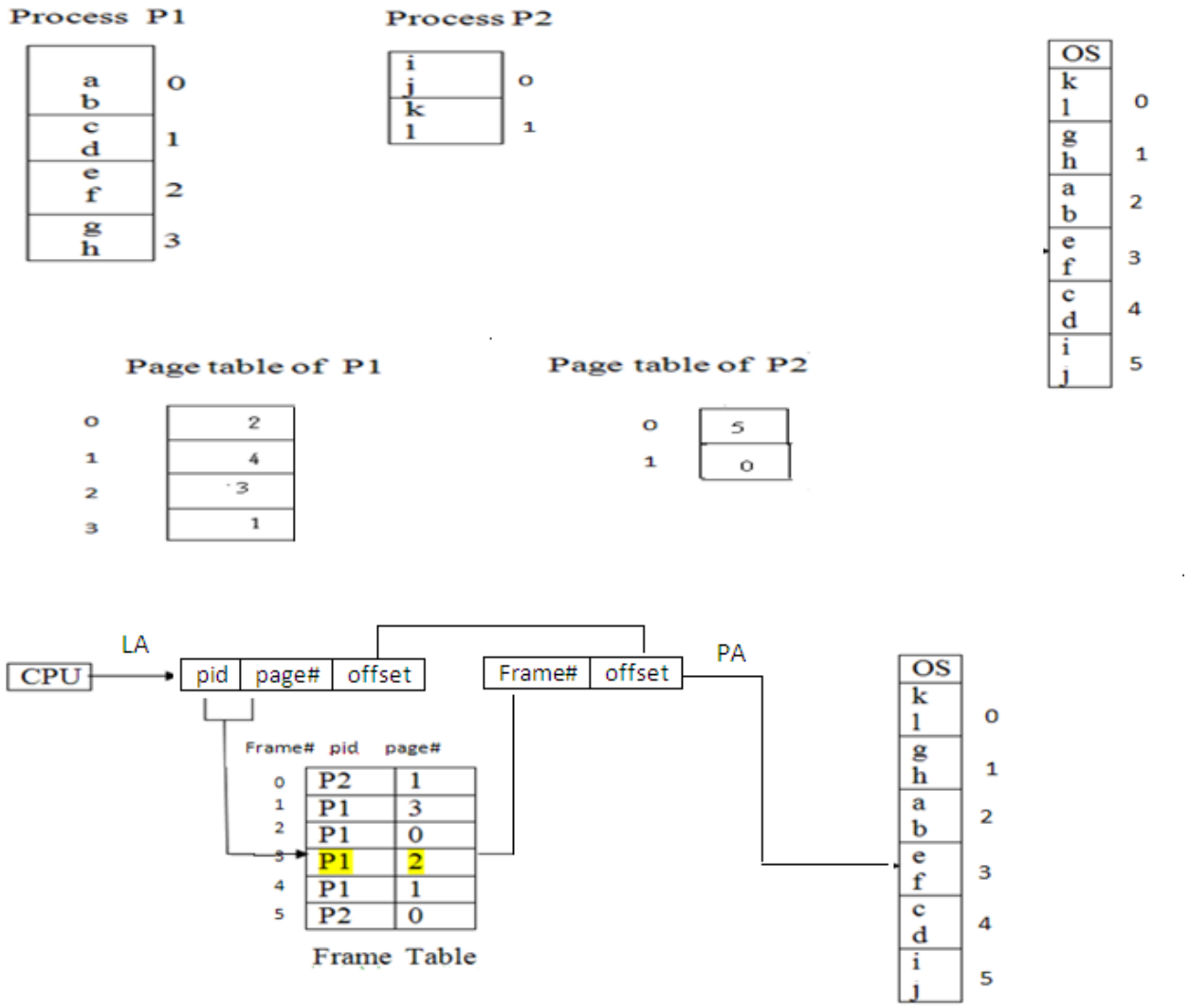
With hashed page table structure also, the space requirement for the page table is reduced by loading the hash table and one of the linked lists instead of all entries of page table at any particular time during execution of the process.

Inverted Page table

In this structure, instead of maintaining a separate page table for each process, one table is maintained for all processes. This table is called as "Frame Table". The number of entries in frame table is equal to number of frames in RAM. Each entry in the frame table contains a process number and a page number.

While executing a process, when CPU generates a logical address, that logical address is divided into 3 parts: process id (pid), page number and offset. The pid and page number are searched in the frame table. If they are present in the frame table then the corresponding frame number is appended with offset of logical address to get the physical address.

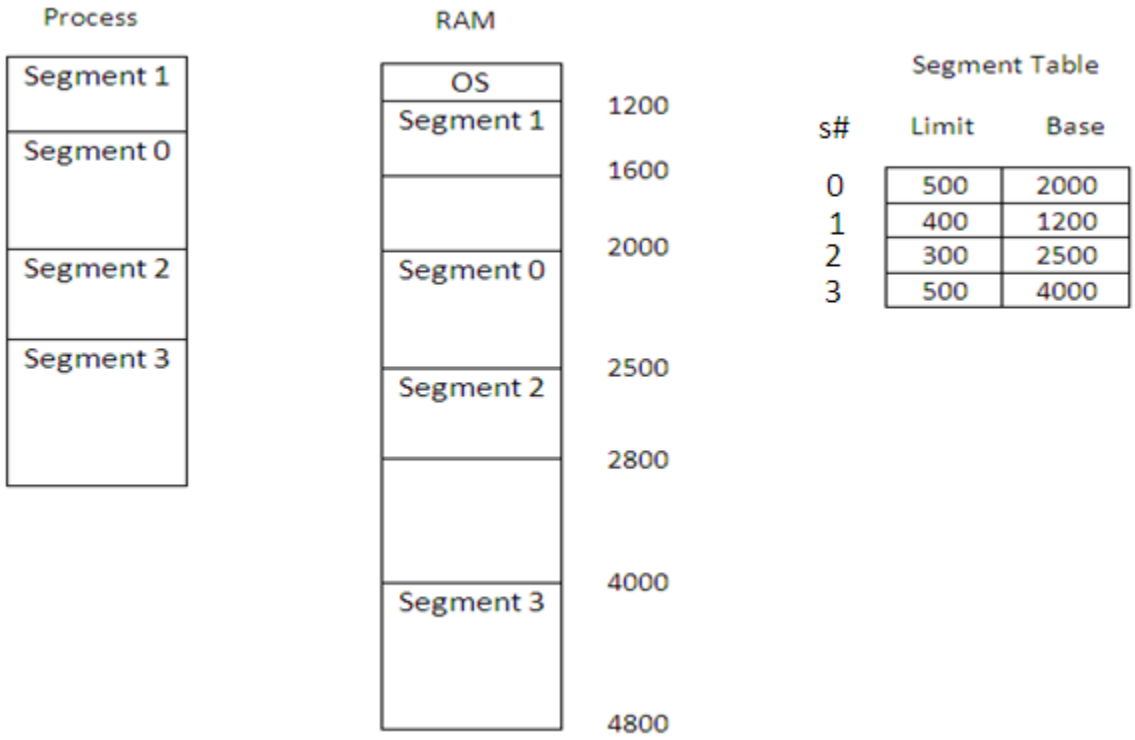
The following diagram shows usage of inverted page table.



Segmentation

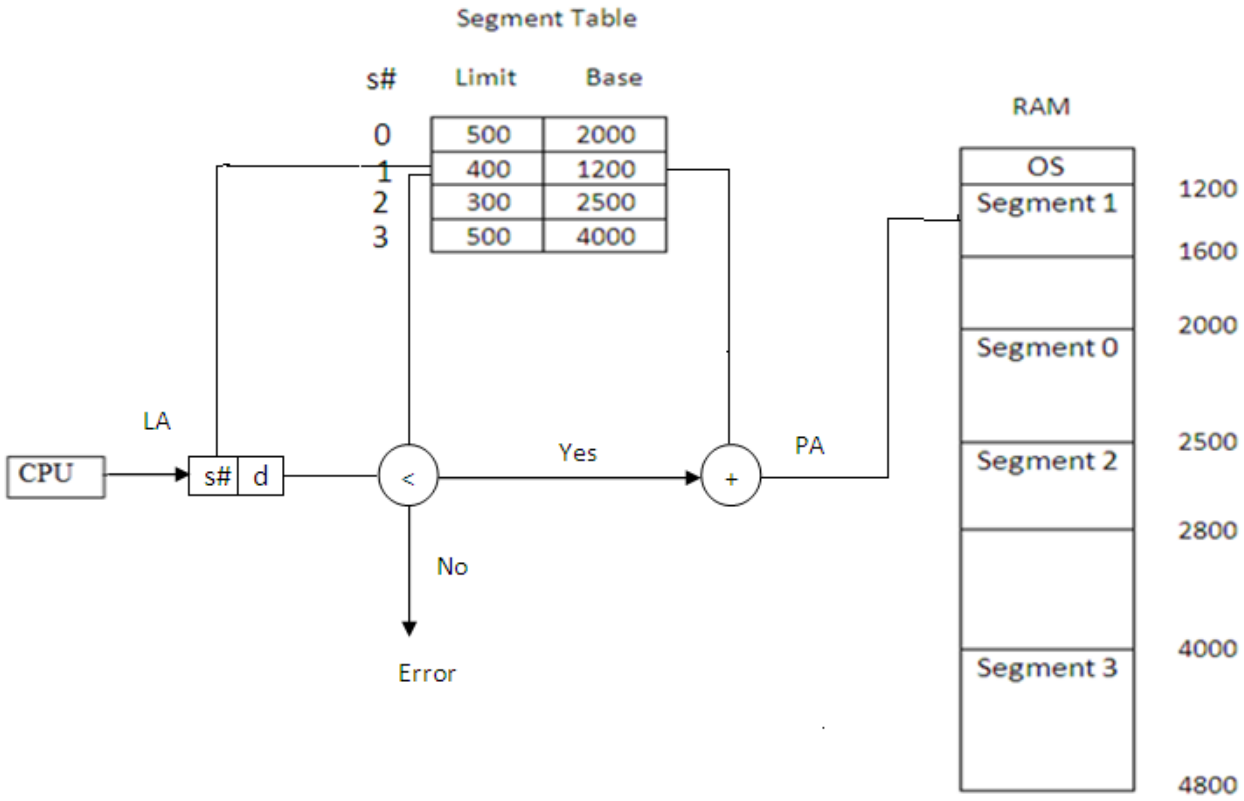
In this technique, each process is divided into number of parts of different sizes. Each part of process is called a segment. RAM is not divided into parts before loading any process. RAM is divided into number of parts at the time of loading segments of process. Segments of process can be loaded into RAM at any position and in any order. After loading segments of process into RAM, a segment table is created for the process by Operating System. Number of entries in segment table is equal to number of segments in the process. Each entry of segment table contains a base which indicates the starting address of the segment in RAM, and limit which indicates the number of instructions in that segment.

In the following figure, a process is divided into four segments. These segments are loaded into RAM and then a segment table is created for the process. This segment table contains four entries. Each entry of the table contains starting address and number statements of the corresponding segment.



Conversion of Logical Address to Physical Address

While executing a process, when CPU generates a logical address, that logical address is divided into two parts: segment number and offset. Segment number is used as an index into segment table to identify the entry of segment table. Offset value of Logical address is compared with limit value in the identified entry. If offset value is less than the limit value then offset value is added to the base value in the entry to get physical address. Otherwise, an error is reported. The following diagram shows this conversion.



Virtual memory

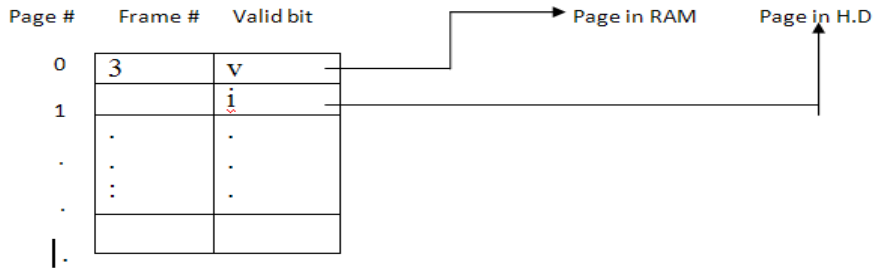
When Operating system uses contiguous allocation or paging or segmentation technique to manage main memory, to start the execution of a process, operating system has to load entire process into memory. With virtual memory, the execution of a process can be started by loading a part of the process into RAM.

Advantage

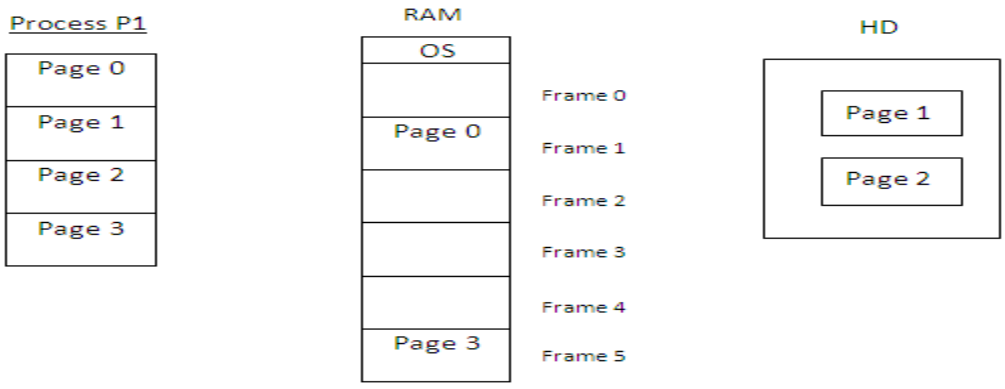
When Operating system uses contiguous allocation or paging or segmentation technique to manage main memory, if the size of the process is greater than the size of RAM then it is not possible to load the entire process into RAM and start the execution of process. But, with virtual memory the operating system can load the process into the RAM even if the size of process is greater than the size of RAM.

Demand Paging Technique

It is combination of paging and swapping techniques. In this technique, the RAM is divided into number of equal size frames before loading any process into RAM. Each process is divided into number of equal size pages. To execute a process, either all or some of the pages of the process are loaded into free frames of RAM. A page table for the process is created after loading pages of the process into RAM. The number of entries in the page table is equal to number of pages in the process. The structure of the page table is as shown below



If the page is currently loaded in the RAM then the corresponding entry in the page table contains the frame number and v. If the page is not loaded into RAM then the corresponding entry in the page table does not have any frame number and it contains bit i. In the following figure, the process P1 is divided into 4 pages, two pages are loaded into RAM remaining two pages are in the HD. The page table of process P1 is containing four entries. Two entries of page table are containing frame numbers and remaining two entries are empty.



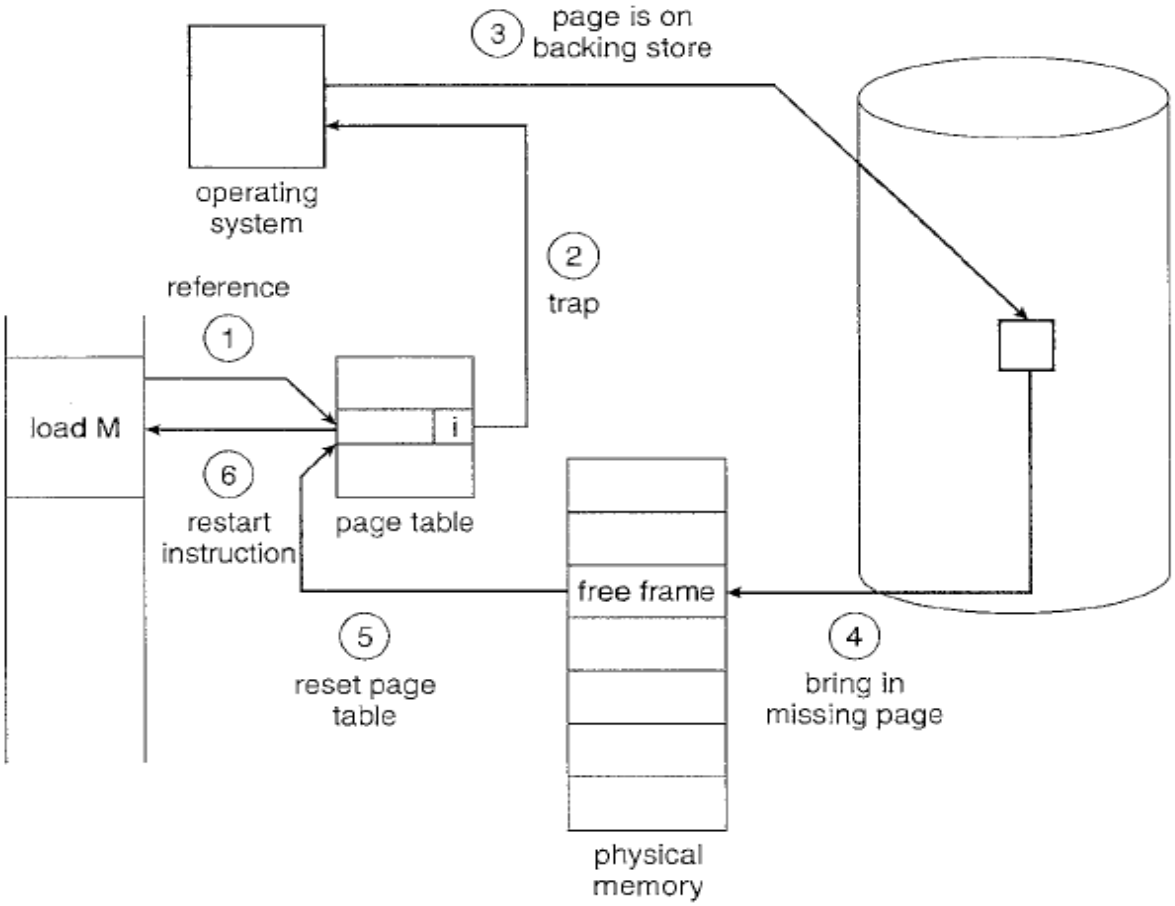
Page table of P1

Page #	Frame #	Valid bit
0	1	v
1		i
2		i
3	5	v

1. a. What is a Page Fault? Explain the steps involved in handling a page fault with a neat sketch?

While executing a process, when CPU generates a logical address, that logical address is divided into two parts: page number and offset as in case of paging. The page number is used as an index into the page table of the process. If the entry of page table is containing the frame number then that frame number is appended with offset of logical address to generate physical address. If the entry is not containing frame number then that situation is called a page fault. **The operating system performs the following activities when a page fault occurs**.

1. The operating system checks whether RAM is containing a free frame or not.
2. If RAM is containing a free frame then OS loads the required page into the free frame and updates page table of the process.
3. If there is no free frame in RAM then OS replaces one of the pages in RAM with the required page and updates page table of the process. For replacing one of the pages, OS uses a page replacement algorithm.
4. Instructs CPU to generate the same logical address again.



4. What is the need of Page replacement? Discuss about different types of Page replacement algorithms.

Page Replacement

If RAM is currently full and if OS wants to load a new page into RAM then OS moves one of the pages in RAM to Hard disk and loads the new page into the free frame. For selecting a page for replacement, OS uses one of the following page replacement algorithms.

Page Replacement Algorithms

1. First in first out (FIFO)
2. Optimal
3. Least Recently Used (LRU)

First in first out

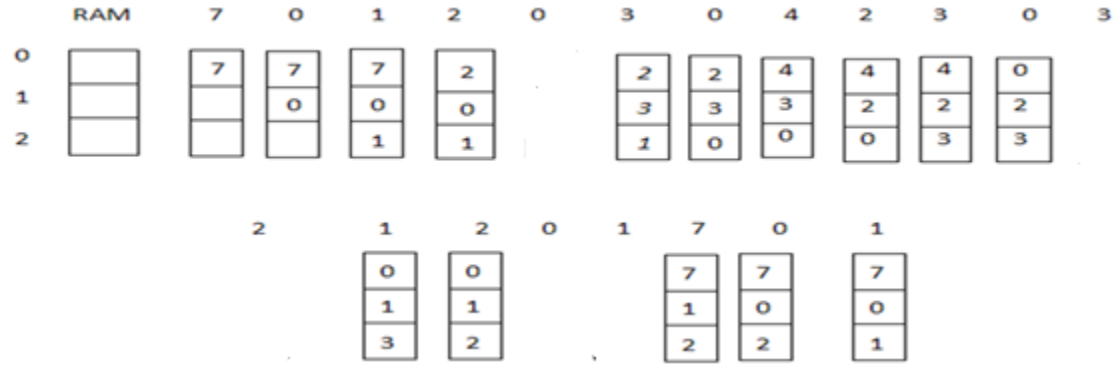
The page that was loaded for the first time into the RAM will be replaced by requested page.

Reference string

Reference string indicates the order in which pages of a process are executed or refereed.

Ex:-
Number of pages in process=8 (0 to 7)
Number of frames in RAM=3
Reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

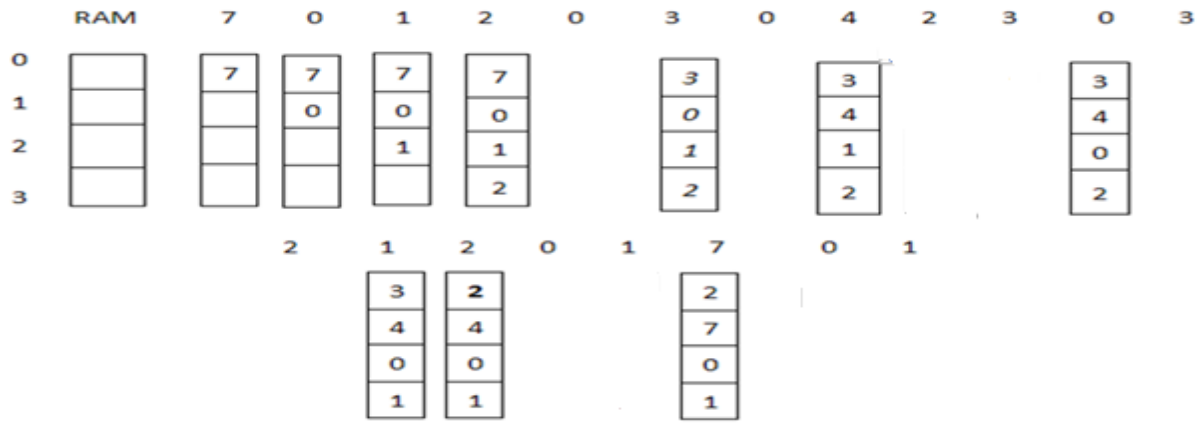
The replacement of pages in RAM is shown in below figure



Number of page faults=15

➢ To compare the performance of different page replacement algorithms we calculate number of page faults for each algorithm.

Ex2:-
Number of pages in process=8
Number of frames in RAM=4

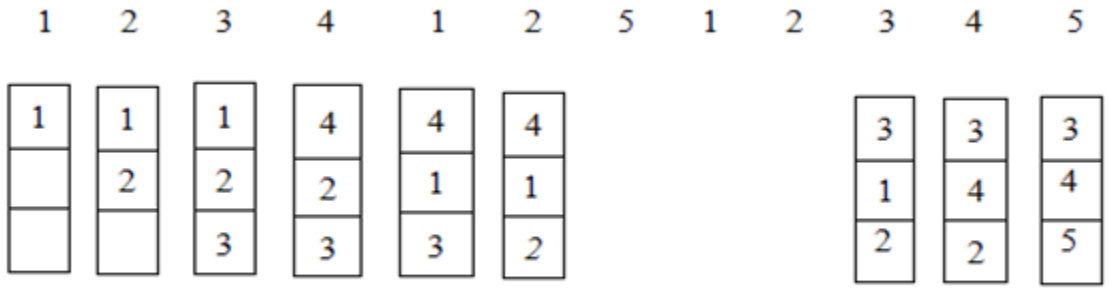


Number of page faults=10

By increasing the number of frames in RAM the number of page faults can be reduced.

Ex3:-
Number of pages in process=5
Number of frames in RAM=3

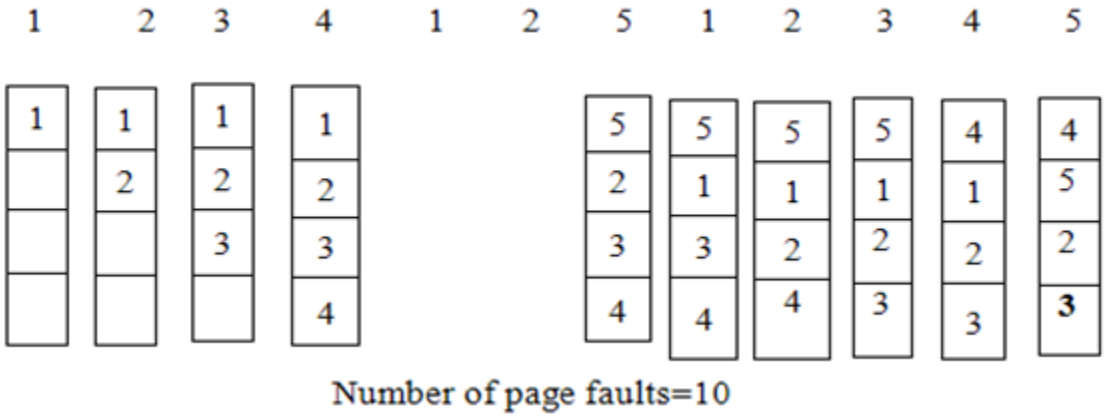
Reference String: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



Number of pageFaults=9

Ex4:-
Number of pages in process=5
Number of frames in RAM=4

Reference String: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

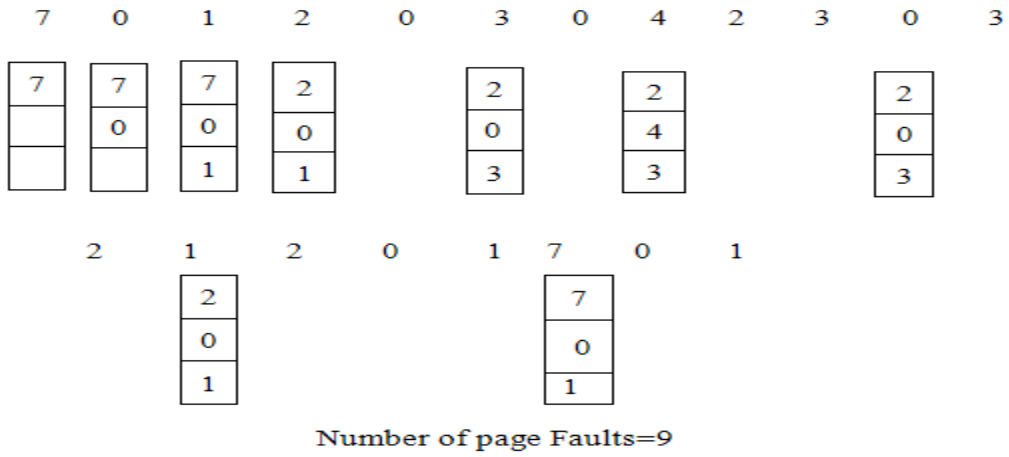


- Number of page faults will be decreased by increasing number of frames in RAM.
- With FIFO algorithm, in some cases number of page faults increases when the number of frames in RAM increases. This situation is called as **"Belady's Anamoly".**
- This is the major drawback with FIFO.
- To avoid Belady's Anamoly, optimal page replacement algorithm is used.

Optimal page replacement algorithm

The page that will not be used for the longest period of time will be replaced by the requested page.

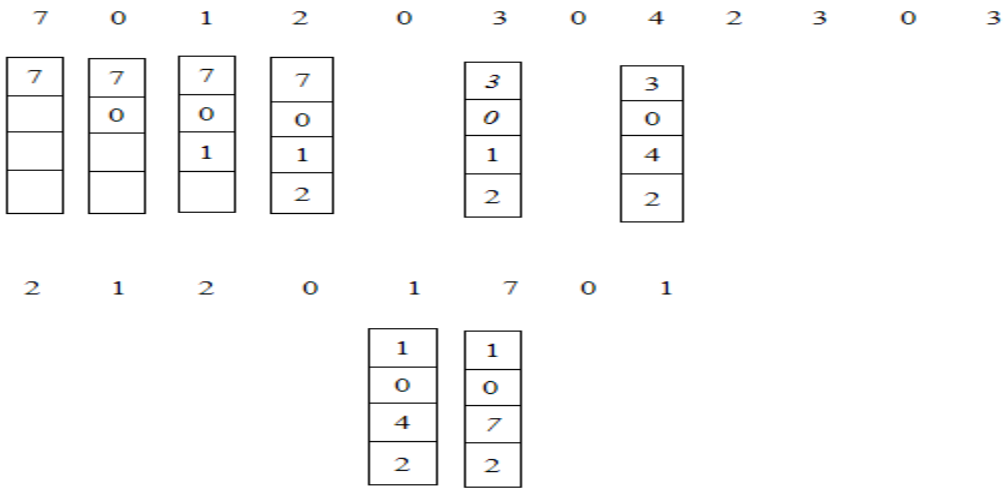
Ex1:
Number of pages in process=8
Number of frames in RAM=3
Reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



Ex2:
Number of pages in process=8
Number of frames in RAM=4

Reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

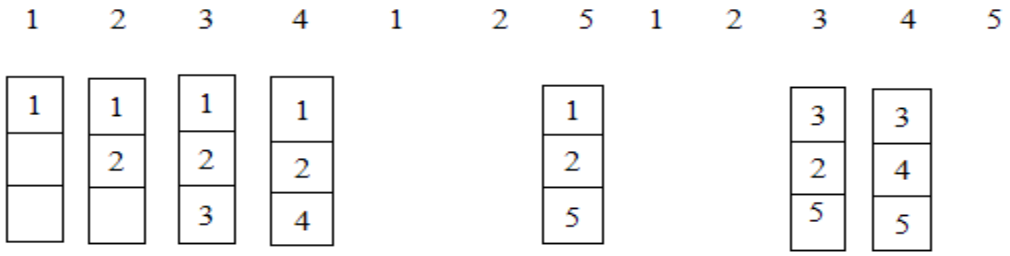
Number of frames=4



Number of page faults=8

Ex3:
Number of pages in process=5(1 to 5)
Number of frames in RAM=3
Reference String: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

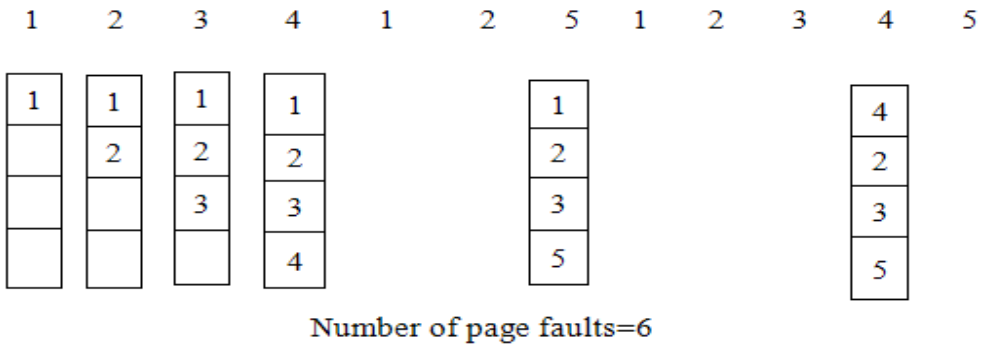
Number of frames=3



Number of page Faults=7

Ex4:
Number of pages in the process=5(1 to 5)
Number of frames in RAM=4
Reference String: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Number of frames=4



Number of page faults=6

Advantage:
Less number of page faults.

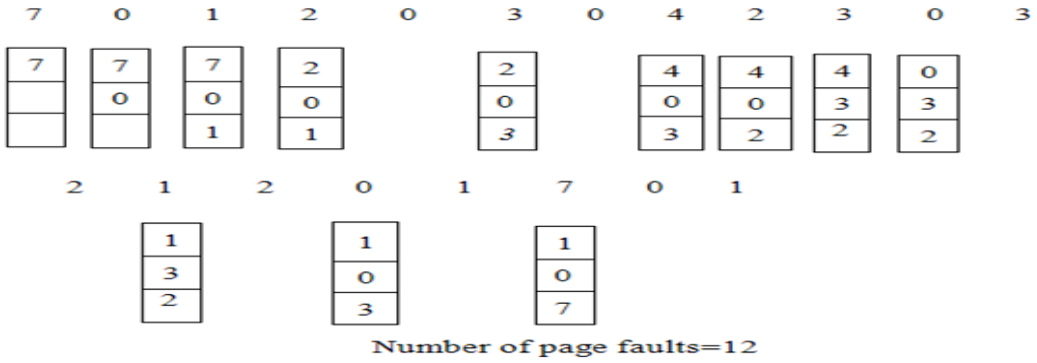
Disadvantage:
This algorithm is working based on future references of the pages. If the operating system don't know the order in which the pages will be requested then it is not possible to calculate future references for the pages and not possible to use this algorithm.

Least Recently Used Page Replacement Algorithm (LRU)

In this algorithm, the page that has not been used for the longest period of time is replaced by the requested page.

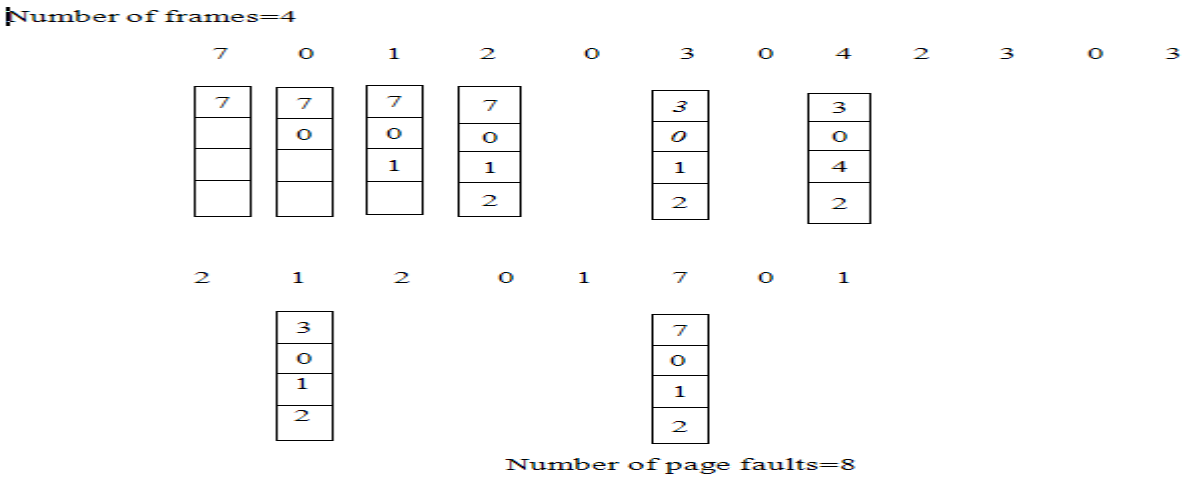
Ex1:
Number of pages in process=8
Number of frames in RAM=3
Reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Number of frames=3



Number of page faults=12

Ex2:
Number of pages in process=8
Number of frames in RAM=4
Reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



Advantages:

- Less number of page faults compared to FIFO.
- Its implementation is easy as it depends only on the previous references of pages.

Variants of LRU page replacement algorithm

1. Additional-Reference-Bits algorithm
2. Second-Chance algorithm
3. Enhanced Second-Chance algorithm

Additional-Reference-Bits algorithm

For each page loaded in RAM, an 8-bit byte is maintained. The operating system shifts the reference bit for each page into the high-order bit of its 8-bit byte, shifts other bits to right by 1 bit and discard the low-order bit. The page with lowest 8-bit byte value is selected for replacement.

Second-Chance algorithm

For each page loaded in RAM, a reference bit is maintained. The reference bit value of a page is '0' if the page is not referenced. Otherwise, reference bit value is '1'. When a page fault occurs, the reference bits of pages in the frames of RAM are checked. This algorithm moves over the pages with reference value '1'. While moving, sets the reference bit value of pages to '0'. When a page with reference bit value '0' is found then that page is replaced by the required page.

Enhanced Second-Chance algorithm

For each page loaded in RAM, a reference bit and a modify bit are maintained as an ordered pair. 4 situations are possible:
(0, 0) - neither recently used nor modified - best page to replace.
(0, 1) - not recently used, but modified - not quite as good, because the page will need to be written out before replacement.

(1, 0) - recently used but clean - probably will be used again soon.
(1, 1) - probably will be used again, will need to write out before replacement.

Counting based page replacement algorithm

A reference count is maintained for each page loaded in RAM. Reference count indicates the number of times a page is referenced. If a page is not referenced after loading it into RAM then its reference count is '0'. Two variants of counting based page replacement are

1. Least Frequently Used (LFU) page replacement algorithm
2. Most Frequently Used (MFU) page replacement algorithm

Least Frequently Used (LFU) page replacement algorithm

The page with least reference count is replaced.

Most Frequently Used (LFU) page replacement algorithm

The page with highest reference count is replaced.

5. a. Discuss various issues related to the allocation of frames to processes?

Allocation of frames

To increase performance of computer system, the CPU of system should be kept busy at all times. To keep the CPU busy, many processes or programs needs to be loaded into RAM. To load many processes into RAM, the frames in RAM needs to be allocated to processes. To allocate frames of RAM to processes, the following points need to be considered.

1. Minimum number of frames allocated to each process.
2. Allocation algorithms.
3. Global versus Local Allocation.
4. Non-Uniform Memory Access (NUMA).

Minimum number of frames

When less number of frames is allocated to a process then more number of page faults occurs. When more number of frames is allocated to a process then less number of page faults occurs. For each process, minimum two frames are required in order to start the execution of the process. One frame for loading at least one page of the process and another frame for loading page table of the process. Specifically, the minimum number of frames required by a process depends on the architecture of the computer system and the method of addressing.

Allocation Algorithms

Two types of algorithms are used for allocating frames to processes.

1. Equal allocation
2. Proportional allocation

Equal allocation

If there are 'm' number of frames in RAM and 'n' number of processes then 'm/n' number of frames is allocated to each process.

Two disadvantages with equal allocation are:

1. Some frames are unused when number of pages in a process is less than the allocated 'm/n' number of frames. For example, if a process requires 4 frames but allocated 10 frames then 6 frames are unused.
2. More page faults occur when number of pages in a process is far more than the allocated 'm/n' number of frames. For example, if a process requires 20 frames but allocated only 3 frames then more number of page faults occurs.

Proportional allocation

Depending on size or priority or both of processes, different number of frames is allocated to different processes. More number of frames is allocated to large processes and less number of frames is allocated to small processes.

Advantages:

1. Less number of page faults as adequate number of pages is allocated to each process.
2. No wastage of frames.

Global versus Local Allocation

Local allocation

While executing a process, if a page fault occurs then one of the frames allocated to the same process is selected to load the required page.

Global allocation

While executing a process, if a page fault occurs then one of the frames allocated to other processes is selected to load the required page.

Global allocation is the commonly used algorithm.

Non-Uniform Memory Access (NUMA)

In a computer system, processors and RAM components are placed either on same system board or on different system boards. If the RAM component in which the process is loaded and the processor to which the process is assigned for execution are on the same system board then the processor can access the process in less time. Otherwise, more time is required to access the process by the processor.

To execute a process in less time, the operating system has to load the process in the RAM component placed on system board on which the processor is placed.

b. What is the cause of Thrashing? How does the system detect Thrashing? How to eliminate this problem?

Thrashing

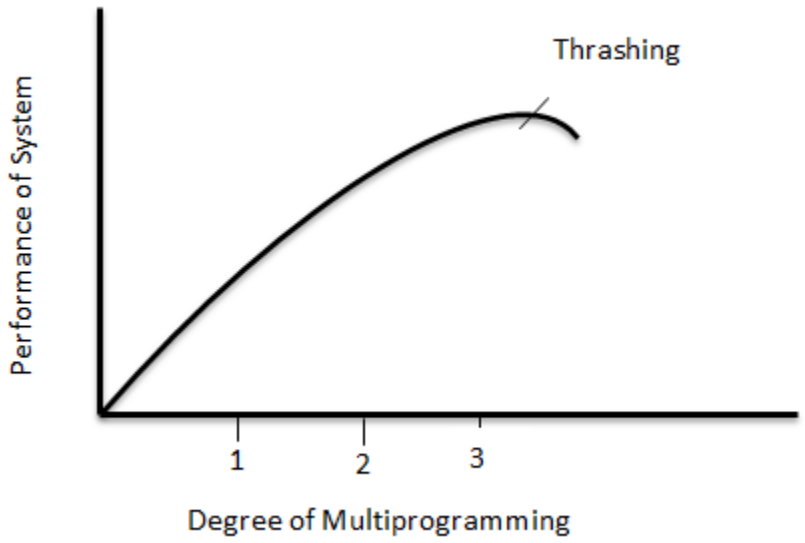
Multi Programming: indicates placing more number of programs in the RAM. Utilization of the CPU and also performance of the system will be increased by placing more number of programs in the RAM.

Degree of Multiprogramming: indicates size for multiprogramming.

Increase in degree of multiprogramming increases number of page faults.

If number of page faults increases then most of the time will be spent on moving the pages from Hard Disk to RAM instead of executing the pages.

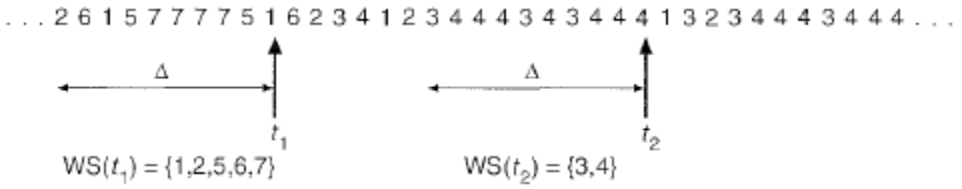
If number of page faults increases to a maximum level then that situation is called *"thrashing"* and it leads to less performance of the system.



To avoid thrashing, working set model is used.

Working-set model

In working-set model, a window called working-set window is maintained. This working-set window moves on the set of page references.



At any particular time, the set of active pages in the working-set window is indicated through a set called working-set. The above diagram shows the pages in working-set at time t1 and at time t2. The number of pages in working set determines the minimum number of frames required by the process at any particular time. In above diagram, the process requires 5 frames at time t1 and 2 frames at time t2.

Based on working-set, operating system from time to time determines the number of frames required by the process and allocates that required number of frames to the process.