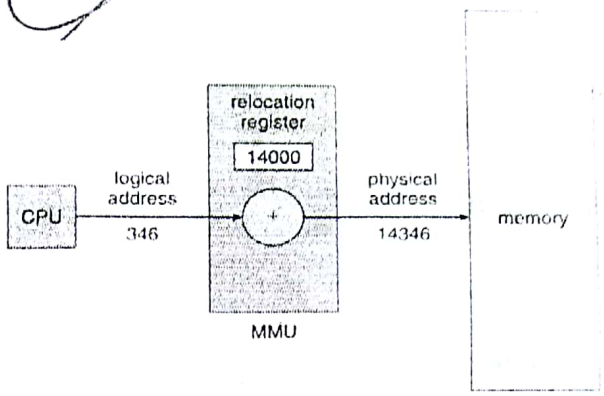
UNIT-III

D. Rupa
Roll no:-16851-A052

1) **Memory management** is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution.



Process logical Address Space vs physical address space:

The set of all logical addresses generated by the cpu 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 operating system takes care of mapping the logical addresses to physical addresses at the time of memory allocation to the program.

Logical and physical addresses are the same in compile-time and load-time, address-binding schemes.

Logical and physical addresses differ in execution-time, address-binding scheme. The runtime mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device.

- 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 14000, then an attempt by the user to use address location 346 will be dynamically reallocated to location 14346.

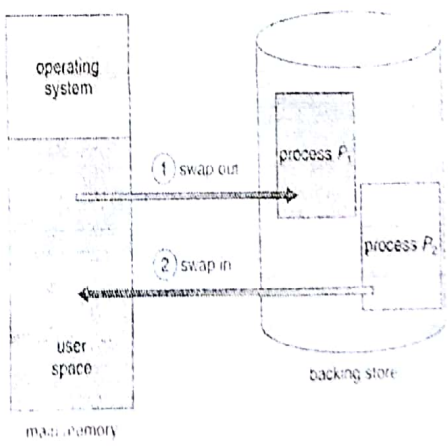
- The user program deals with virtual addresses; it never sees the real physical addresses.

2) Swapping: A process needs to be in memory for execution. But sometimes there is not enough main memory to hold all the currently active processes in a timesharing system. So, excess process are kept on disk and brought in to run dynamically. Swapping is the process of bringing in each process in main memory, running it for a while and then putting it back to the disk.

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason **Swapping is also known as a technique for memory compaction.**

The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.



Let us assume that the user process is of size 2048KB and on standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of 1000K process to or from memory will take a the

2048KB / 1024KB per second

= 2 seconds

= 2000 milliseconds

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.

3) Contiguous and Noncontiguous Memory Allocation

1. The basic difference between contiguous and noncontiguous memory allocation is that contiguous allocation allocates **one single contiguous block of memory** to the process whereas, the noncontiguous allocation divides the **process into several blocks** and place them in the **different address space of the memory** i.e. in a noncontiguous manner.
2. In contiguous memory allocation, the process is stored in contiguous memory space; so there is **no overhead of address translation** during execution. But in noncontiguous memory allocation, there is **an overhead of address translation** while the process execution, as the process blocks are spread in the memory space.
3. Process stored in contiguous memory executes **faster** in comparison to process stored in noncontiguous memory space.
4. The solution for contiguous memory allocation is to **divide the memory space into the fixed-sized partition** and allocate a partition to a single process only. On the other hands, in non contiguous memory allocation, **a process is divided into several blocks** and each block is placed at **different places in memory** according to the availability of the memory.
5. In contiguous memory allocation, operating system has to maintain a **table** which indicates which partition is available for the process and which is occupied by the process. In noncontiguous memory allocation, a **table** is maintained for **each process** which indicates the base address of each block of the process placed in the memory space.

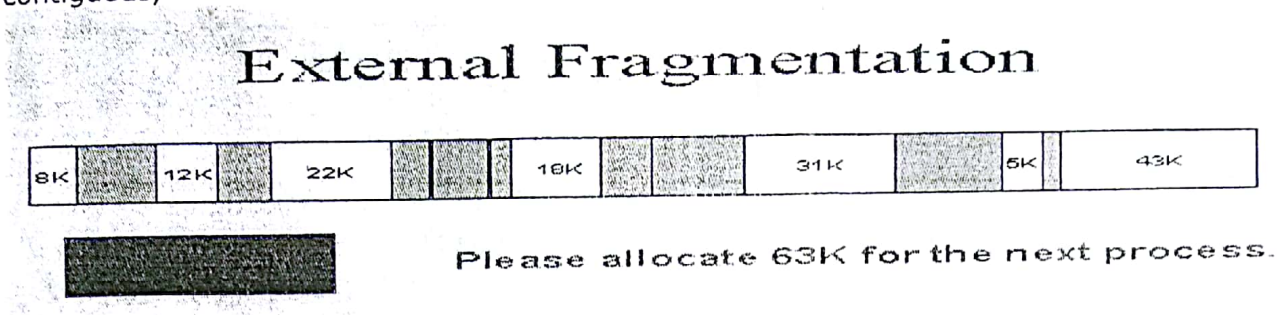
4) Fragmentation

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types:

External fragmentation

Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.



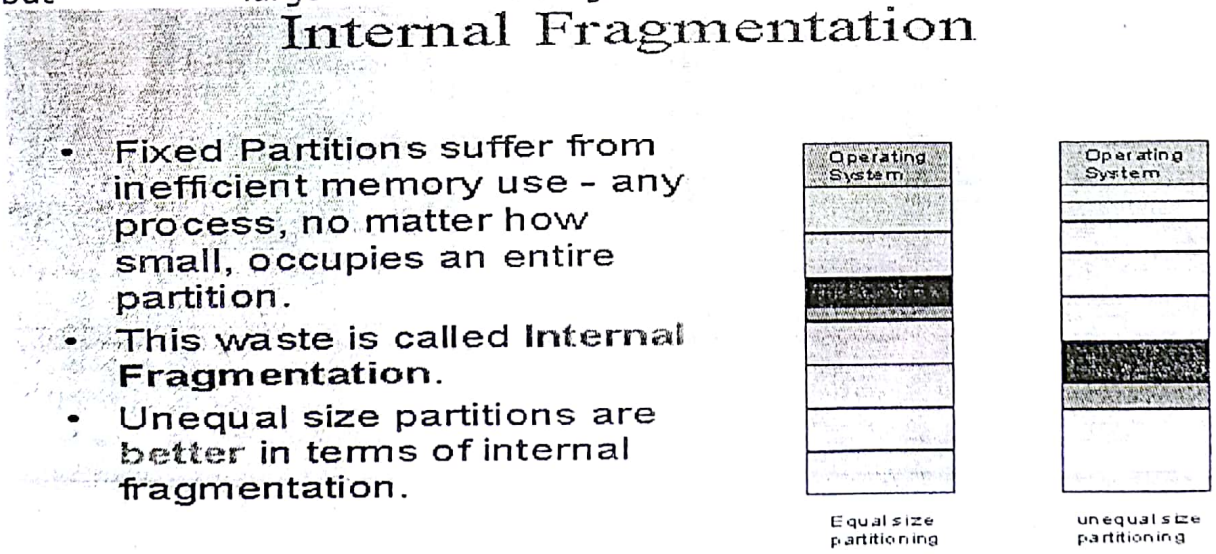
- Eventually, main memory forms holes too small to hold any process. This is external fragmentation.
- Total memory space may exist to satisfy a request but it is not contiguous.
- Compaction reduces external fragmentation by shuffling memory contents to place all free memory together in one large block.

Internal fragmentation

Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process. The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory –

External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.



1. **First fit** - Search the list of holes until one is found that is big enough to satisfy the request, and assign a portion of that hole to that process. Whatever fraction of the hole not needed by the request is left on the free list as a smaller hole. Subsequent requests

may start looking either from the beginning of the list or from the point at which this search ended.

2. **Best fit** - Allocate the *smallest* hole that is big enough to satisfy the request. This saves large holes for other process requests that may need them later, but the resulting unused portions of holes may be too small to be of any use, and will therefore be wasted. Keeping the free list sorted can speed up the process of finding the right hole.

3. **Worst fit** - Allocate the largest hole available, thereby increasing the likelihood that the remaining portion will be usable for satisfying future requests.

Segmentation:

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

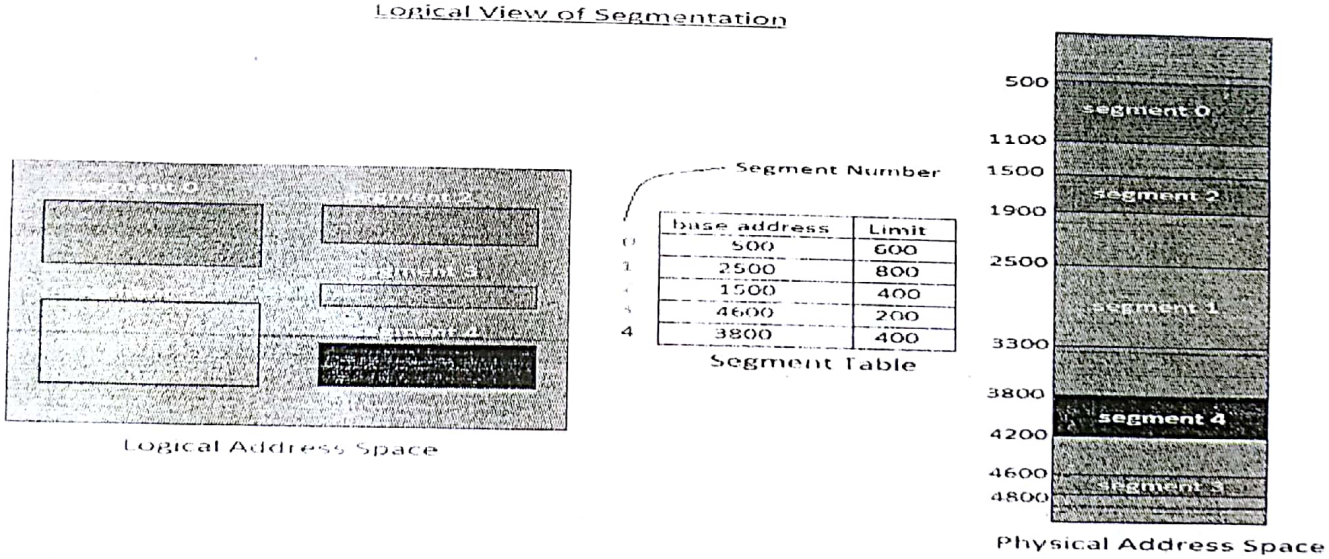
When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

Memory Management technique in which memory is divided into variable sized chunks which can be allocated to processes. Each chunk is called a**Segment**.
A table stores the information about all such segments and is called **Segment Table.**

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

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

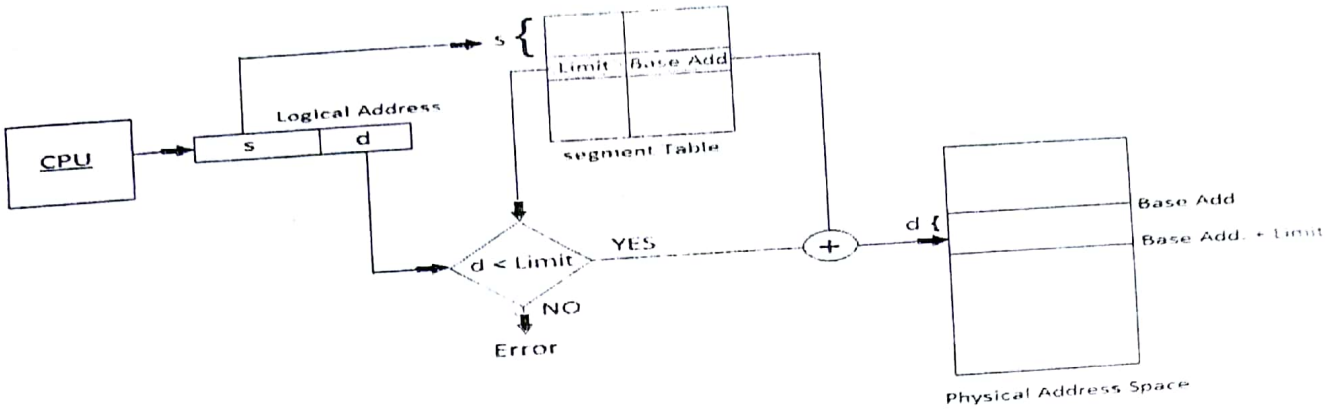


Translation of Two dimensional Logical Address to one dimensional Physical Address.

Address generated by the CPU is divided into:

- Segment number **(s):** Number of bits required to represent the segment.
- Segment offset **(d)**: Number of bits required to represent the size of the segment.

this



Advantages of Segmentation:

- No Internal fragmentation.
- Segment Table consumes less space in comparison to Page table in paging.

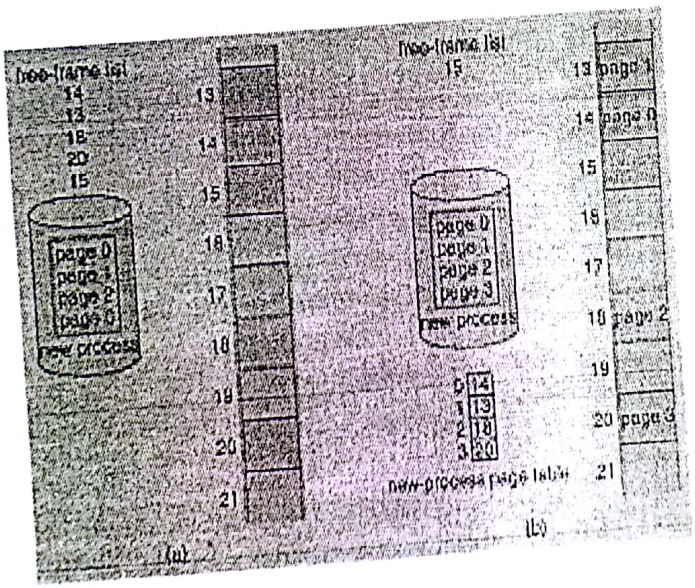
Disadvantage of Segmentation:

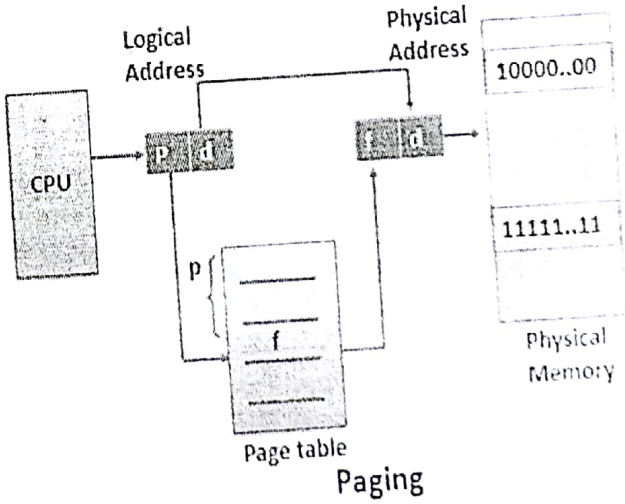
- As processes are loaded and removed from the memory, the free memory space is broken into little pieces, causing External fragmentation.

6) **Paging: "Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory."**

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

Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages**. The size of the process is measured in the number of pages. Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.





Paging

Address Translation

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

Logical Address = Page number + page offset

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

Physical Address = Frame number + page offset

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

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

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

Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging –

- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

7) Virtual Memory:

"Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as it were part of main memory."

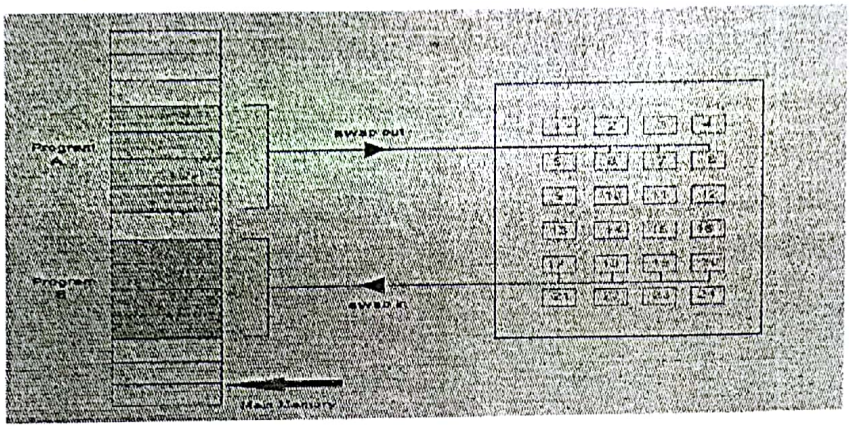
This technique is useful as large virtual memory is provided for user programs when a very small physical memory is available say 4GB only.

Benefits of having Virtual Memory :

1. Large programs can be written, as virtual space available is huge compared to physical memory.
2. Less I/O required, leads to faster and easy swapping of processes.
3. More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.
4. System libraries can be shared by mapping them into the virtual address space of more than one process.
5. Processes can also share virtual memory by mapping the same block of memory to more than one process.
6. Process pages can be shared during a fork( ) system call, eliminating the need to copy all of the pages of the original (parent) process.

8) *Demand Paging:*

The basic idea behind demand paging is that when a process is swapped in, its pages are not swapped in all at once. Rather they are swapped in only when the process needs them(On demand). This is termed as lazy swapper, although a pager is a more accurate term.



Initially only those pages are loaded which will be required the process immediately.

The pages that are not moved into the memory, are marked as invalid in the page table. For an invalid entry the rest of the table is empty. In case of pages that are loaded in the memory, they are marked as valid along with the information about where to find the swapped out page.

When the process requires any of the page that is not loaded into the memory, a page fault trap is triggered and following steps are followed,

1. The memory address which is requested by the process is first checked, to verify the request made by the process.
2. If its found to be invalid, the process is terminated.
3. In case the request by the process is valid, a free frame is located, possibly from a free-frame list, where the required page will be moved.
4. A new operation is scheduled to move the necessary page from disk to the specified memory location. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)
5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to valid.
6. The instruction that caused the page fault must now be restarted from the beginning.

There are cases when no pages are loaded into the memory initially, pages are only loaded when demanded by the process by generating page faults. This is called **Pure Demand Paging**.

The only major issue with Demand Paging is, after a new page is loaded, the process starts execution from the beginning. Its is not a big issue for small programs, but for larger programs it affects performance drastically.

Page Replacement:

As studied in Demand Paging, only certain pages of a process are loaded initially into the memory. This allows us to get more number of processes into the memory at the same time. but what happens when a process requests for more pages and no free memory is available to bring them in. Following steps can be taken to deal with this problem :

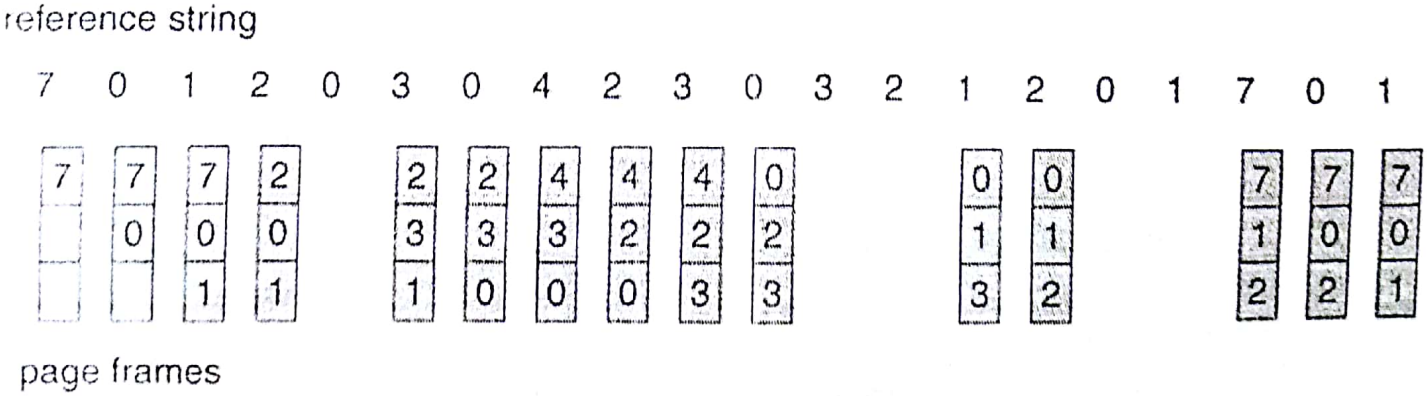
1. Put the process in the wait queue, until any other process finishes its execution thereby freeing frames.
2. Or, remove some other process completely from the memory to free frames.
3. Or, find some pages that are not being used right now, move them to the disk to get free frames. This technique is called **Page replacement** and is most commonly used. We have some great algorithms to carry on page replacement efficiently.

Basic Page Replacement

- Find the location of the page requested by ongoing process on the disk.
- 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 any existing frame to be replaced, such frame is known as **victim frame.**
- Write the victim frame to disk. Change all related page tables to indicate that this page is no longer in memory.
- Move the required page and store it in the frame. Adjust all related page and frame tables to indicate the change.
- Restart the process that was waiting for this page.

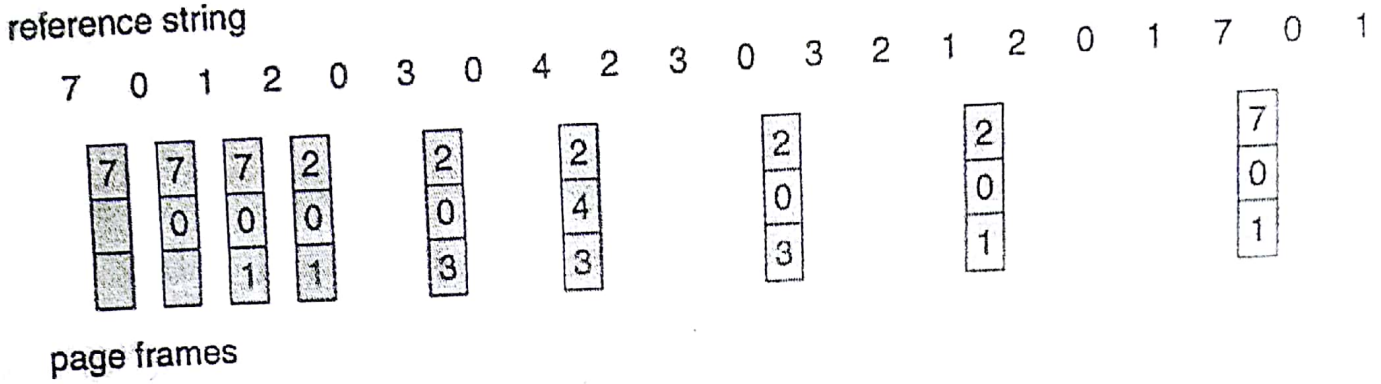
FIFO Page Replacement

- A very simple way of Page replacement is FIFO (First in First Out)
- As new pages are requested and are swapped in, they are added to tail of a queue and the page which is at the head becomes the victim.
- Its not an effective way of page replacement but can be used for small systems.



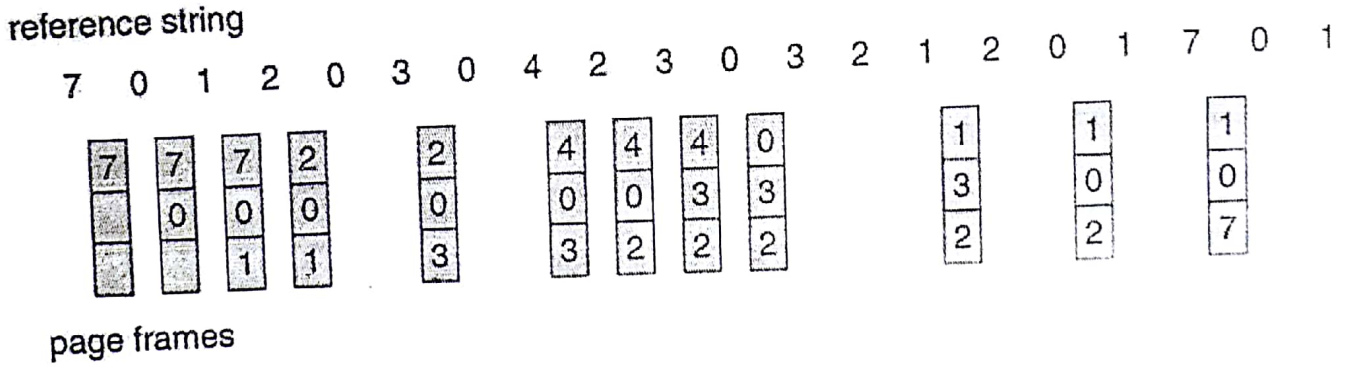
(ii) Optimal Page algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- "Replace the page that will not be used for the longest period of time. Use the time when a page is to be used."



(iii) Least Recently Used (LRU) algorithm

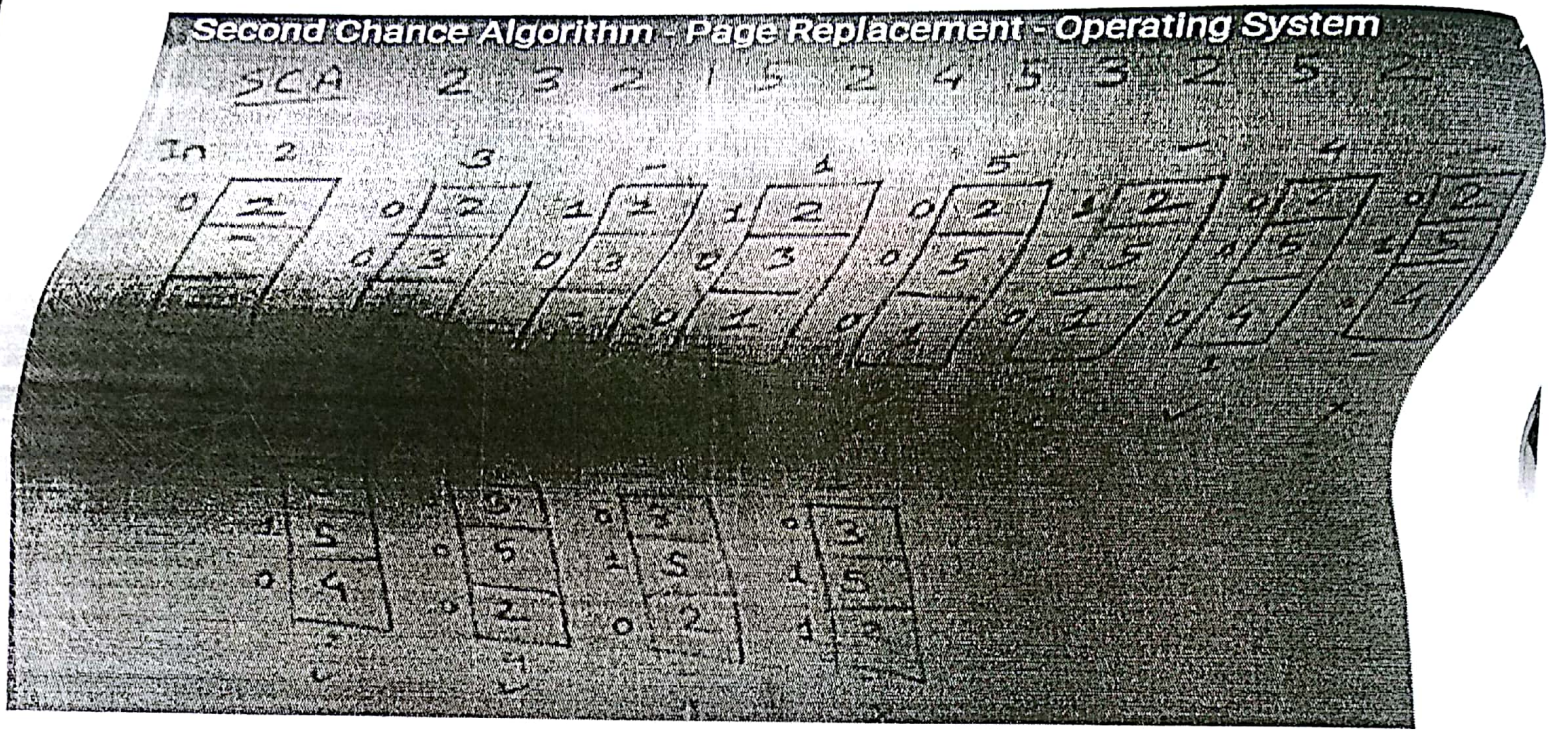
- "Replace the Page which has not been used for the longest time in main memory is the one which will be selected for replacement."
- Easy to implement, keep a list, replace pages by looking back into time.



(iv) **LRU-Approximation Page Replacement:**

- The second chance algorithm is essentially a FIFO, except the reference bit is used to give pages a second chance at staying in the page table.
- When a page must be replaced, the page table is scanned in a FIFO (circular queue) manner.
- If a page is found with its reference bit not set, then that page is selected as the next victim.
- If, however, the next page in the FIFO does have its reference bit set, then it is given a second chance:
- The reference bit is cleared, and the FIFO search continues.
- If some other page is found that did not have its reference bit set, then that page will be selected as the victim, and this page (the one being given the second chance) will be allowed to stay in the page table.
- If , however, there are no other pages that do not have their reference bit set, then this page will be selected as the victim when the FIFO search circles back around to this page on the second pass.

- If all reference bits in the table are set, then second chance degrades to FIFO, but als complete search of the table for every page-replacement.
- As long as there are some pages whose reference bits are not set, then any page referenced t enough gets to stay in the page table indefinitely.



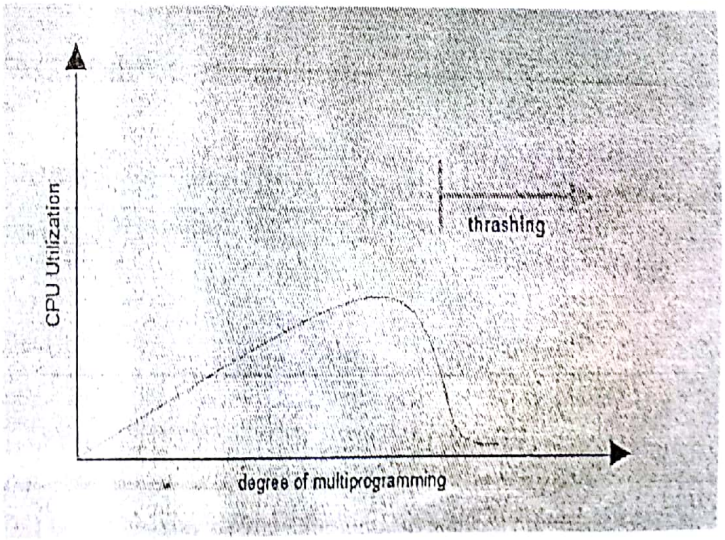
Thrashing:

A process that is spending more time paging than executing is said to be thrashing. In other words it means, that the process doesn't have enough frames to hold all the pages for its execution, so it is swapping pages in and out very frequently to keep executing. Sometimes, the pages which will be required in the near future have to be swapped out.

Initially when the CPU utilization is low, the process scheduling mechanism, to increase the level of multiprogramming loads multiple processes into the memory at the same time, allocating a limited amount of frames to each process. As the memory fills up, process starts to spend a lot of time for the required pages to be swapped in, again leading to low CPU utilization because most of the processes are waiting for pages. Hence the scheduler loads more processes to increase CPU utilization, as this continues at a point of time the complete system comes to a stop.

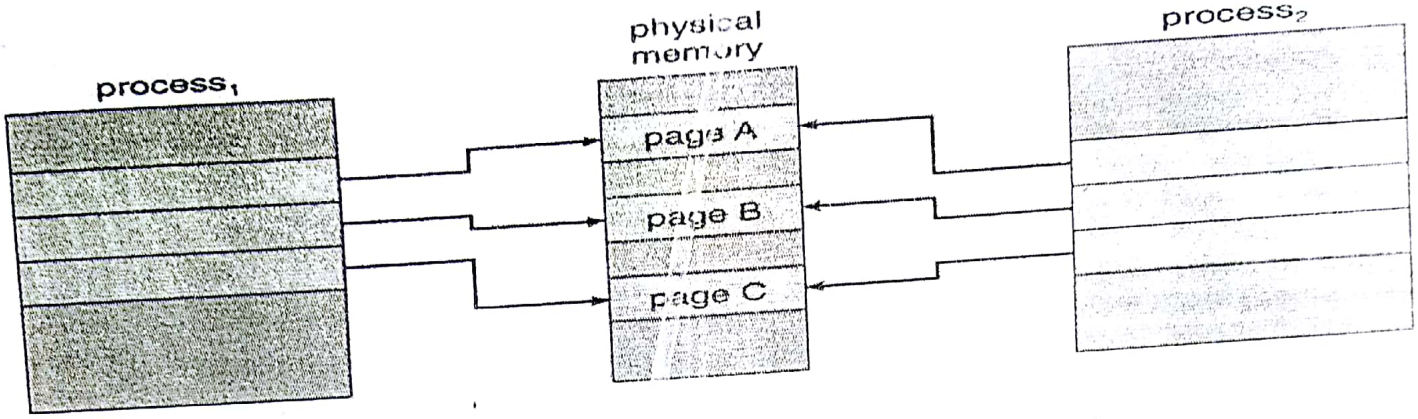
Causes of Thrashing :

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.
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.



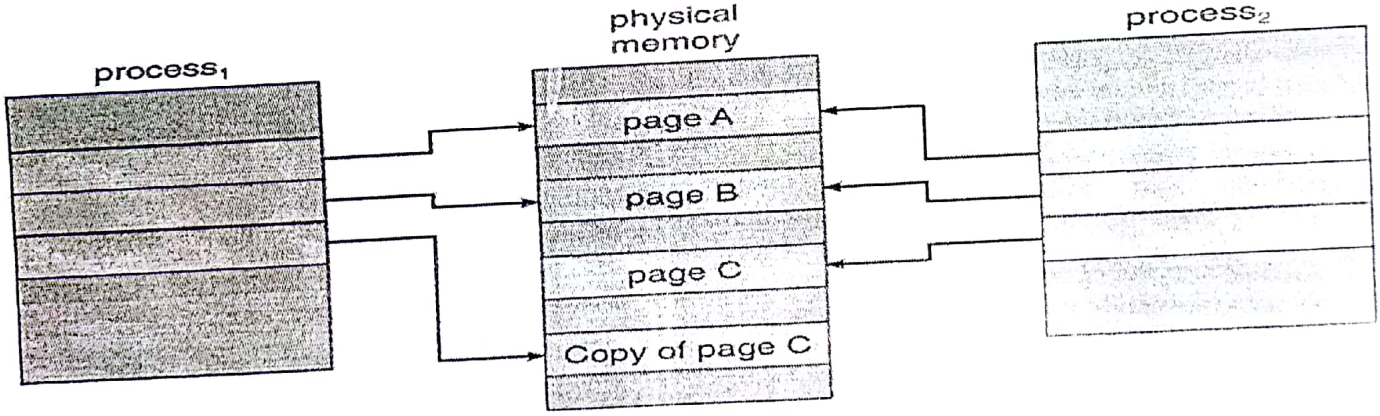
Copy-on-Write

- The idea behind a copy-on-write fork is that the pages for a parent process do not have to be actually copied for the child until one or the other of the processes changes the page. They can be simply shared between the two processes in the meantime, with a bit set that the page needs to be copied if it ever gets written to. This is a reasonable approach, since the child process usually issues an exec( ) system call immediately after the fork.



Before process 1 modifies page C.

- Obviously only pages that can be modified even need to be labeled as copy-on-write. Code segments can simply be shared.



After process 1 modifies page C.

Comparison of Paging and Segmentation

Paging	Segmentation
Main memory is partitioned into frames or blocks	Main memory is partationed into segments
The logical address space is divided into pages by MMU	The logical address space is divided into segments as specified by the program
The scheme suffers from internal fragmentation or page breaks	The scheme suffers from External fragmentation
OS maintains a free frame list , so that searching of free frame is not necessary	OS maintains the particulars of available memory
OS maintains a page map table for mapping between frames and pages	OS maintains a segment map table for mapping
This scheme does not support the users view of memory	This scheme support the users view of memory
Processor use the page number and displacement to calculate absolute address (p, d)	Processor use the segment number and displacement to calculate the absolute address (p, d)