**Unit 3: Storage Management**

* **Basic concept of storage management**

**Storage Management** is defined as it refers to the management of the data storage equipment’s that are used to store the user/computer generated data. Hence it is a tool or set of processes used by an administrator to keep your data and storage equipment’s safe.

Storage management is a process for users to optimize the use of storage devices and to protect the integrity of data for any media on which it resides and the category of storage management generally contain the different type of subcategories covering aspects such as security, virtualization and more, as well as different types of provisioning or automation, which is generally made up the entire storage management software market.

* **Storage management key attributes:** Storage management has some key attribute which is generally used to manage the storage capacity of the system. These are given below:

1. *Performance*
2. *Reliability*
3. *Recoverability*
4. *Capacity*

* **Feature of Storage management:** There is some feature of storage management which is provided for storage capacity. These are given below:
* Storage management is a process that is used to optimize the use of storage devices.
* Storage management must be allocated and managed as a resource in order to truly benefit a corporation.
* Storage management is generally a basic system component of information systems.
* It is used to improve the performance of their data storage resources
* **Advantage of storage management:**

1. It becomes very simple to manage a storage capacity.
2. It generally reduces the time consumption.
3. It improves the performance of system.
4. In virtualization and automation technologies, it can help an organization improve its agility.

* ***Limitations of storage management:***

1. Limited physical storage capacity: Operating systems can only manage the physical storage space that is available, and as such, there is a limit to how much data can be stored.
2. Performance degradation with increased storage utilization: As more data is stored, the system’s performance can decrease due to increased disk access time, fragmentation, and other factors.
3. Complexity of storage management: Storage management can be complex, especially as the size of the storage environment grows.
4. Cost: Storing large amounts of data can be expensive, and the cost of additional storage capacity can add up quickly.
5. Security issues: Storing sensitive data can also present security risks, and the operating system must have robust security features in place to prevent unauthorized access to this data.
6. Backup and Recovery: Backup and recovery of data can also be challenging, especially if the data is stored on multiple systems or devices.

* **Logical and Physical address Space**

**What is a Logical Address?**

The logical address is a virtual address created by the CPU of the computer system. The logical address of a program is generated when the program is running. A group of several logical address is referred to a logical address space. The logical address is basically used as a reference to access the physical memory locations.

In computer systems, a hardware device named memory management unit (MMU) is used to map the logical address to its corresponding physical address. However, the logical address of a program is visible to the computer user.

**What is a Physical Address?**

The physical address of a computer program is one that represents a location in the memory unit of the computer. The physical address is not visible to the computer user. The MMU of the system generates the physical address for the corresponding logical address.

The physical address is accessed through the corresponding logical address because a user cannot directly access the physical address. For running a computer program, it requires a physical memory space. Therefore, the logical address has to be mapped with the physical address before the execution of the program.

* **Difference between Logical and Physical Address in Operating System**

The following table highlights all the major differences between logical and physical address in operating system −

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Logical Address** | **Physical Address** |
| 1. | This address is generated by the CPU. | This address is a location in the memory unit. |
| 2. | The address space consists of the set of all logical addresses. | This address is a set of all physical addresses that are mapped to the corresponding logical addresses. |
| 3. | These addresses are generated by CPU with reference to a specific program. | It is computed using Memory Management Unit (MMU). |
| 4. | The user has the ability to view the logical address of a program. | The user can’t view the physical address of program directly. |
| 5. | The user can use the logical address in order to access the physical address. | The user can indirectly access the physical address. |

* **Swapping**

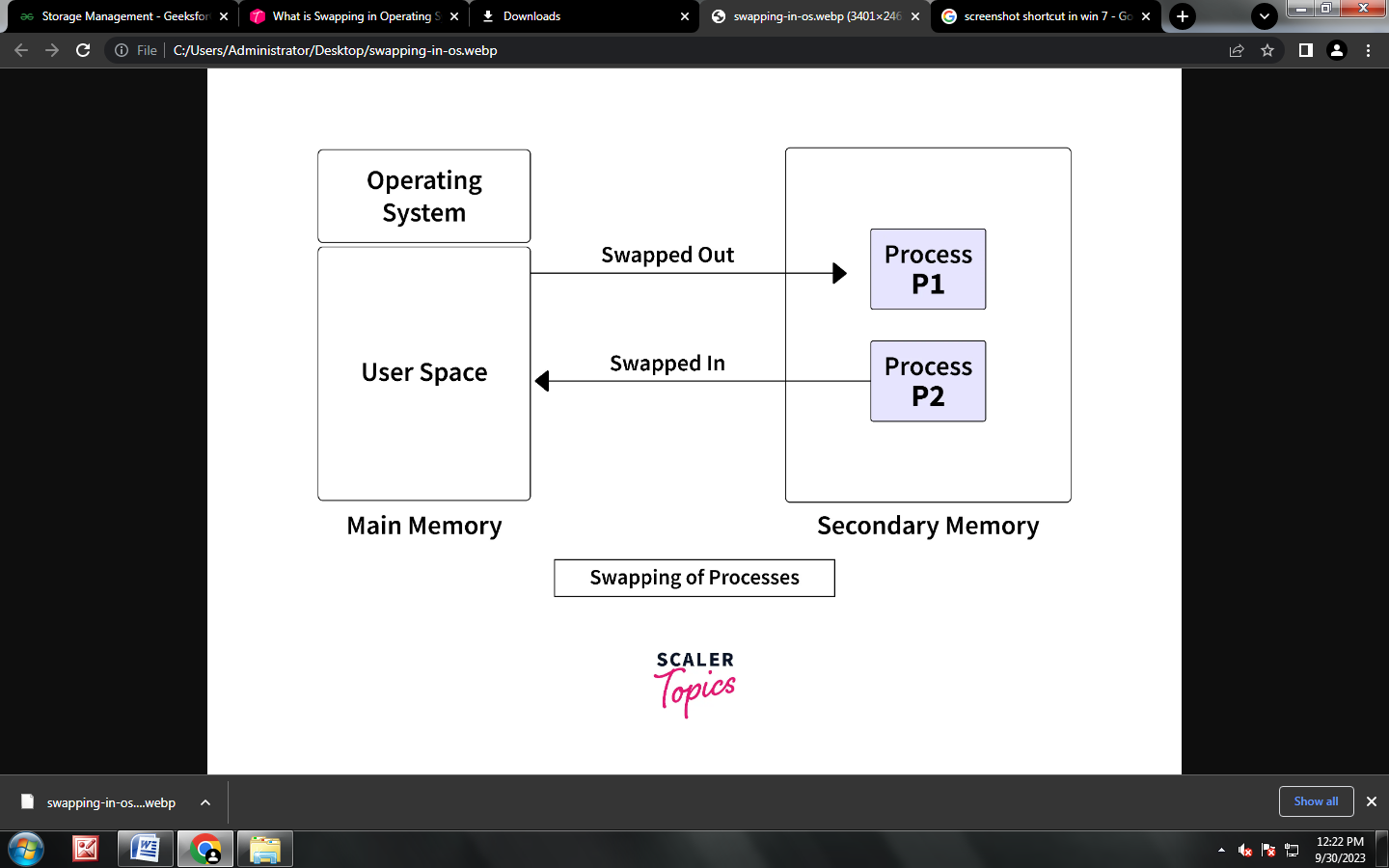
Swapping is a memory management scheme in which any process can be temporarily swapped from main memory to secondary memory so that the main memory can be made available for other processes. It is used to improve main memory utilization. In secondary memory, the place where the swapped-out process is stored is called swap space.

The purpose of the swapping in [operating system](https://www.javatpoint.com/os-tutorial) is to access the data present in the hard disk and bring it to [RAM](https://www.javatpoint.com/ram) so that the application programs can use it. The thing to remember is that swapping is used only when data is not present in [RAM](https://www.javatpoint.com/ram-full-form).

Although the process of swapping affects the performance of the system, it helps to run larger and more than one process. This is the reason why swapping is also referred to as memory compaction.

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

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

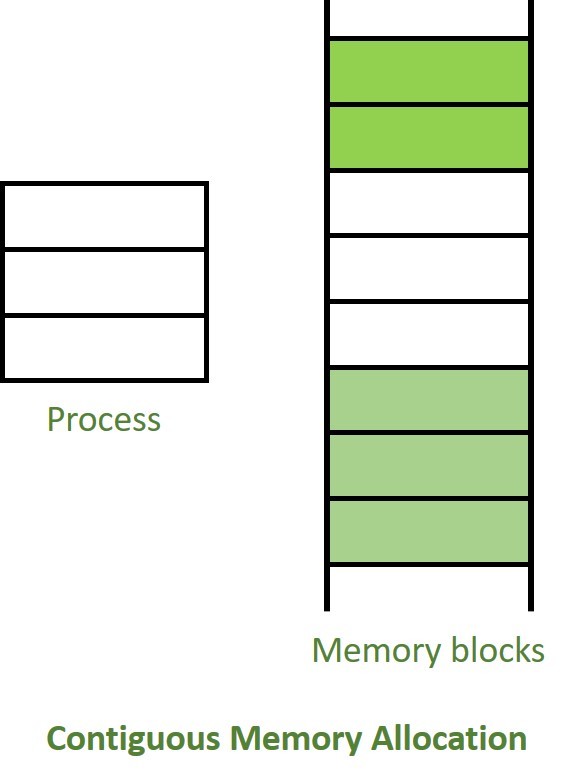


***Advantages of Swapping***

* It helps the CPU to manage multiple processes within a single main memory.
* It helps to create and use virtual memory.
* Swapping allows the CPU to perform multiple tasks simultaneously. Therefore, processes do not have to wait very long before they are executed.
* It improves the main memory utilization.

***Disadvantages of swapping***

* If there is low main memory resource and user is executing too many processes and suddenly the power of system goes off there might be a scenario where data get erase of the processes which are took parts in swapping.
* Chances of number of page faults occur
* Low processing performance
* **What is Contiguous Memory Allocation?**
* It is the type of *memory allocation method.* When a process requests the memory, a single contiguous section of memory blocks is allotted depending on its requirements.
* It is completed by partitioning the memory into fixed-sized partitions and assigning every partition to a single process. However, it will limit the degree of multiprogramming to the number of fixed partitions done in memory.
* This allocation also leads to internal fragmentation. For example, suppose a fixed-sized memory block assigned to a process is slightly bigger than its demand. In that case, the remaining memory space in the block is referred to as internal fragmentation. When a process in a partition finishes, the partition becomes accessible for another process to run.
* The OS preserves a table showing which memory partitions are free and occupied by processes in the variable partitioning scheme. Contiguous memory allocation speeds up process execution by decreasing address translation overheads.

****

***Advantages***

* It is simple to keep track of how many memory blocks are left, which determines how many more processes can be granted memory space.
* The read performance of contiguous memory allocation is good because the complete file may be read from the disk in a single task.
* The contiguous allocation is simple to set up and performs well.

***Disadvantages***

* Fragmentation isn't a problem because every new file may be written to the end of the disk after the previous one.
* When generating a new file, it must know its eventual size to select the appropriate hole size.
* When the disk is filled up, it would be necessary to compress or reuse the spare space in the holes.
* **What is Non-Contiguous Memory Allocation?**

## non contigios.jpg

* It allows a process to obtain multiple memory blocks in various locations in memory based on its requirements. The non-contiguous memory allocation also reduces memory wastage caused by *internal* and *external* fragmentation because it uses the memory holes created by internal and external fragmentation.
* The two methods for making a process's physical address space non-contiguous are paging and segmentation. Non-contiguous memory allocation divides the process into blocks *(pages or segments)* that are allocated to different areas of memory space based on memory availability.
* Non-contiguous memory allocation can decrease memory wastage, but it also raises address translation overheads. As the process portions are stored in separate locations in memory, the memory execution is slowed because time is consumed in address translation.

***Advantages***

* It has the advantage of reducing memory waste, but it increases overhead because of the address translation.
* It slows down the memory execution because time is consumed in address translation.

***Disadvantages***

* The downside of this memory allocation is that the access is slow because you must reach the other nodes using pointers and traverse them.
* ***Difference between Contiguous and Non-contiguous Memory Allocation :***

| S.NO. | **Contiguous Memory Allocation** | **Non-Contiguous Memory Allocation** |
| --- | --- | --- |
| 1. | Contiguous memory allocation allocates consecutive blocks of memory to a file/process. | Non-Contiguous memory allocation allocates separate blocks of memory to a file/process. |
| 2. | Faster in Execution. | Slower in Execution. |
| 3. | It is easier for the OS to control. | It is difficult for the OS to control. |
| 4. | Overhead is minimum as not much address translations are there while executing a process. | More Overheads are there as there are more address translations. |
| 5. | Both Internal fragmentation and external fragmentation occurs in Contiguous memory allocation method. | Only External fragmentation occurs in Non-Contiguous memory allocation method. |
| 6. | It includes single partition allocation and multi-partition allocation. | It includes paging and segmentation. |
| 7. | Wastage of memory is there. | No memory wastage is there. |
| 8. | In contiguous memory allocation, swapped-in processes are arranged in the originally allocated space. | In non-contiguous memory allocation, swapped-in processes can be arranged in any place in the memory. |
| 9. | It is of two types:   1. Fixed(or static) partitioning 2. Dynamic partitioning | It is of five types:   1. Paging 2. Multilevel Paging 3. Inverted Paging 4. Segmentation 5. Segmented Paging |
| 10. | It could be visualized and implemented using Arrays. | It could be implemented using Linked Lists. |
| 11. | Degree of multiprogramming is fixed as fixed partitions | Degree of multiprogramming is not fixed |

* **What is Fragmentation?**

Fragmentation is an unwanted problem in the [operating system](https://www.javatpoint.com/os-tutorial) in which the processes are loaded and unloaded from memory, and free memory space is fragmented. Processes can't be assigned to memory blocks due to their small size, and the memory blocks stay unused. It is also necessary to understand that as programs are loaded and deleted from memory, they generate free space or a hole in the memory. These small blocks cannot be allotted to new arriving processes, resulting in inefficient memory use.

The conditions of fragmentation depend on the memory allocation system. As the process is loaded and unloaded from memory, these areas are fragmented into small pieces of memory that cannot be allocated to incoming processes. It is called **fragmentation**.

## Causes of Fragmentation

User processes are loaded and unloaded from the main memory, and processes are kept in memory blocks in the main memory. Many spaces remain after process loading and swapping that another process cannot load due to their size. Main memory is available, but its space is insufficient to load another process because of the dynamical allocation of main memory processes.

## *Types of Fragmentation*

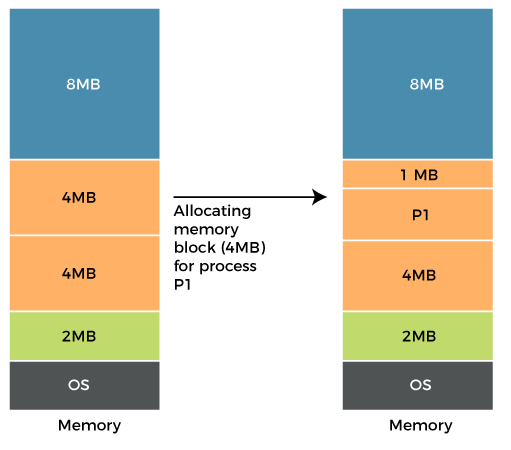
There are mainly two types of fragmentation in the operating system. These are as follows:

1. **Internal Fragmentation**
2. **External Fragmentation**

### Internal Fragmentation

When a process is allocated to a memory block, and if the process is smaller than the amount of memory requested, a free space is created in the given memory block. Due to this, the free space of the memory block is unused, which causes **internal** fragmentation.

**For Example:**

****

Assume that memory allocation in RAM is done using fixed partitioning (i.e., memory blocks of fixed sizes). **2MB, 4MB, 4MB**, and **8MB** are the available sizes. The Operating System uses a part of this RAM.

Let's suppose a process **P1** with a size of **3MB** arrives and is given a memory block of **4MB**. As a result, the **1MB** of free space in this block is unused and cannot be used to allocate memory to another process. It is known as **internal fragmentation**.

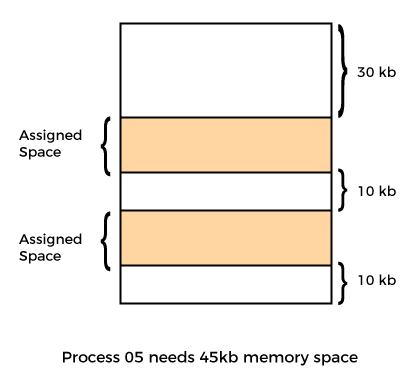
* **How to avoid internal fragmentation?**

The problem of internal fragmentation may arise due to the fixed sizes of the memory blocks. It may be solved by assigning space to the process via dynamic partitioning. Dynamic partitioning allocates only the amount of space requested by the process. As a result, there is no internal fragmentation.

### External Fragmentation

External fragmentation happens when a dynamic memory allocation method allocates some memory but leaves a small amount of memory unusable. The quantity of available memory is substantially reduced if there is too much external fragmentation. There is enough memory space to complete a request, but it is not contiguous. It's known as **external** fragmentation.

**For Example:**

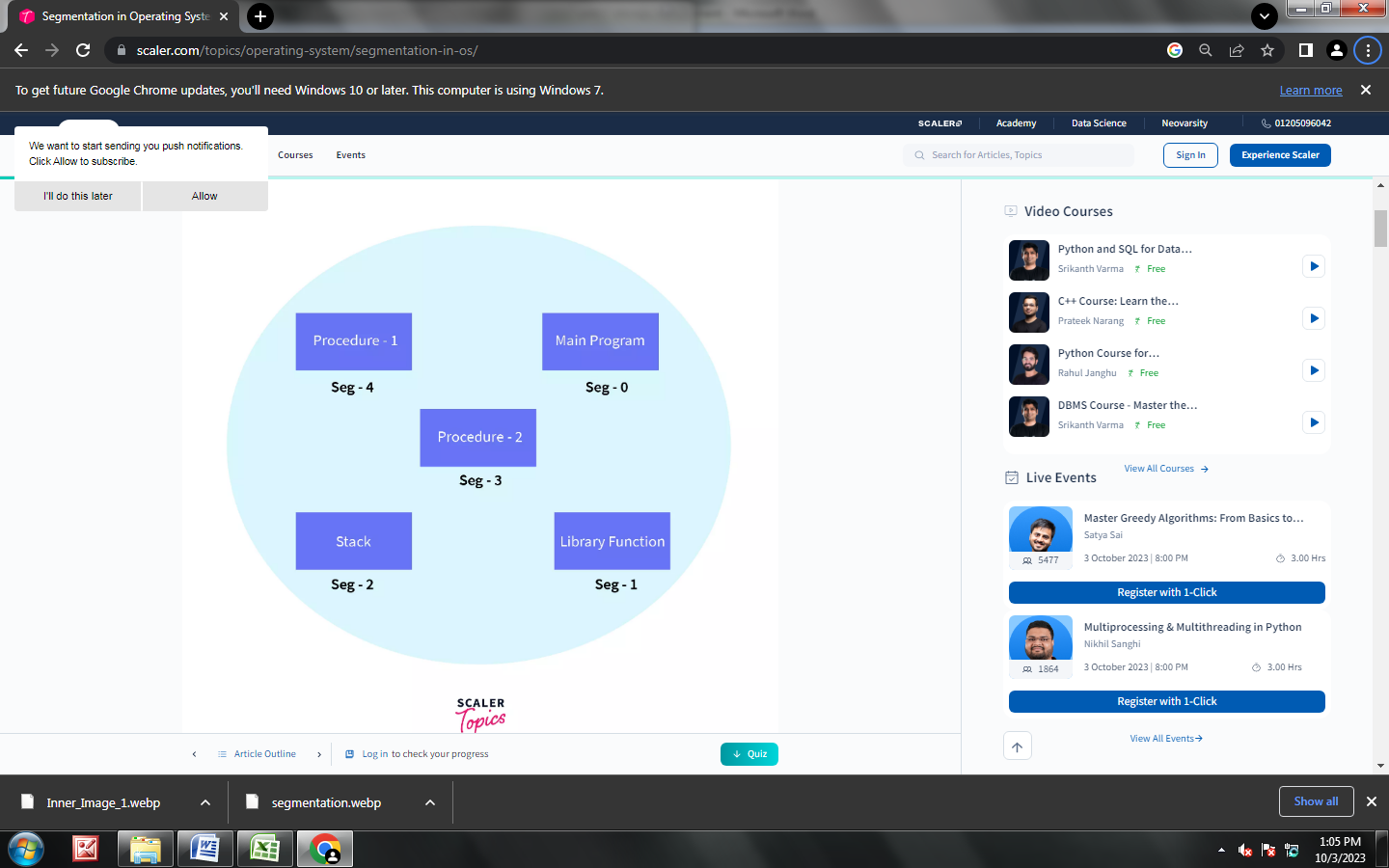
****

* **Segmentation**

Let's take the example of external fragmentation. In the above diagram, you can see that there is sufficient space (50 KB) to run a process (05) (need 45KB), but the memory is not contiguous. You can use compaction, paging, and segmentation to use the free space to execute a process.

Segmentation divides processes into smaller subparts known as modules. The divided segments need not be placed in contiguous memory. Since there is no contiguous memory allocation, internal fragmentation does not take place. The length of the segments of the program and memory is decided by the purpose of the segment in the user program.

We can say that logical address space or the main memory is a collection of segments.



**Types of Segmentation**

1. **Virtual Memory Segmentation**: Virtual Memory Segmentation divides the processes into n number of segments. All the segments are not divided at a time. Virtual Memory Segmentation may or may not take place at the run time of a program.
2. **Simple Segmentation**: Simple Segmentation also divides the processes into n number of segments but the segmentation is done all together at once. Simple segmentation takes place at the run time of a program. Simple segmentation may scatter the segments into the memory such that one segment of the process can be at a different location than the other(in a noncontinuous manner).

* ***Why Segmentation is required?***

Segmentation came into existence because of the problems in the paging technique. In the case of the paging technique, a function or piece of code is divided into pages without considering that the relative parts of code can also get divided. Hence, for the process in execution, the CPU must load more than one page into the frames so that the complete related code is there for execution. Paging took more pages for a process to be loaded into the main memory. Hence, segmentation was introduced in which the code is divided into modules so that related code can be combined in one single block.

Other memory management techniques have also an important drawback - the actual view of physical memory is separated from the user's view of physical memory. Segmentation helps in overcoming the problem by dividing the user's program into segments according to the specific need.

***Advantages of Segmentation in OS***

* No internal fragmentation is there in segmentation.
* Segment Table is used to store the records of the segments. The segment table itself consumes small memory as compared to a page table in paging.
* Segmentation provides better CPU utilization as an entire module is loaded at once.
* Segmentation is near to the user's view of physical memory. Segmentation allows users to partition the user programs into modules. These modules are nothing but the independent codes of the current process.
* The Segment size is specified by the user but in Paging, the hardware decides the page size.
* Segmentation can be used to separate the security procedures and data.

***Disadvantages of Segmentation in OS***

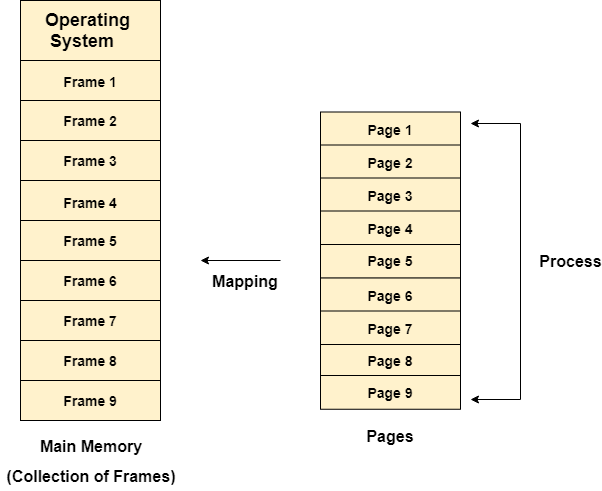
* During the swapping of processes the free memory space is broken into small pieces, which is a major problem in the segmentation technique.
* Time is required to fetch instructions or segments.
* The swapping of segments of unequal sizes is not easy.
* There is an overhead of maintaining a segment table for each process as well.
* **Paging in OS (Operating System)**

In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.

The main idea behind the paging is to divide each process in the form of pages. The main memory will also be divided in the form of frames.

One page of the process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes.

Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.

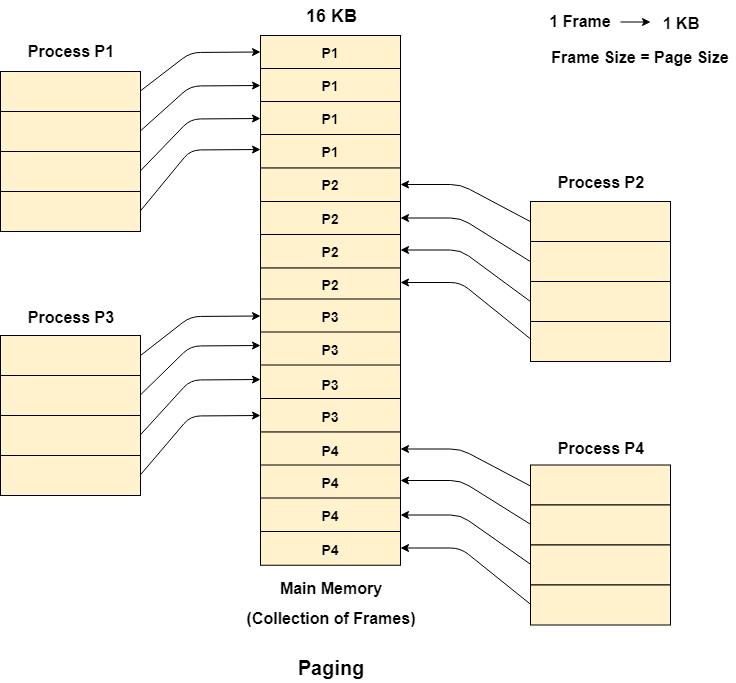
Different operating system defines different frame sizes. The sizes of each frame must be equal. Considering the fact that the pages are mapped to the frames in Paging, page size needs to be as same as frame size.

### Example :

Let us consider the main memory size 16 Kb and Frame size is 1 KB therefore the main memory will be divided into the collection of 16 frames of 1 KB each.

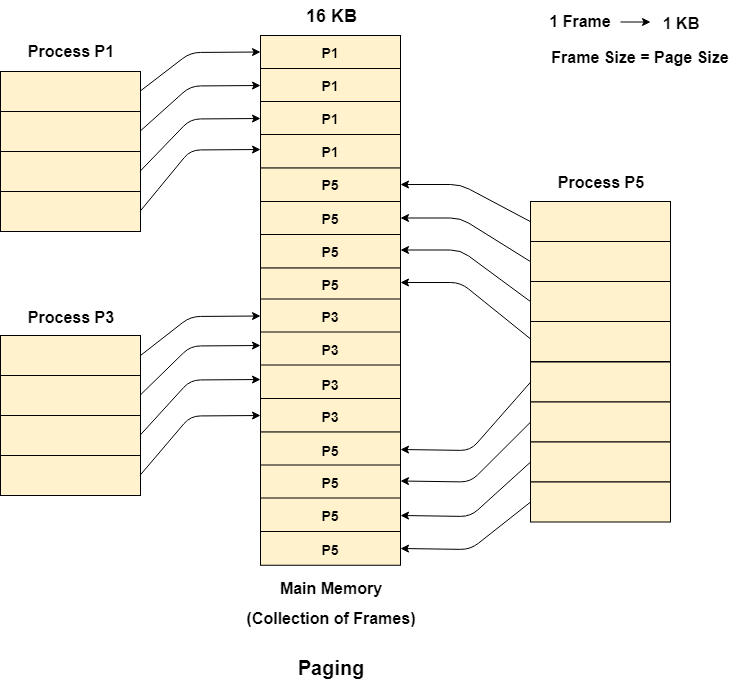
There are 4 processes in the system that is P1, P2, P3 and P4 of 4 KB each. Each process is divided into pages of 1 KB each so that one page can be stored in one frame.

Initially, all the frames are empty therefore pages of the processes will get stored in the contiguous way.

Frames, pages and the mapping between the two is shown in the image below.

Let us consider that, P2 and P4 are moved to waiting state after some time. Now, 8 frames become empty and therefore other pages can be loaded in that empty place. The process P5 of size 8 KB (8 pages) is waiting inside the ready queue.

Given the fact that, we have 8 non contiguous frames available in the memory and paging provides the flexibility of storing the process at the different places. Therefore, we can load the pages of process P5 in the place of P2 and P4.

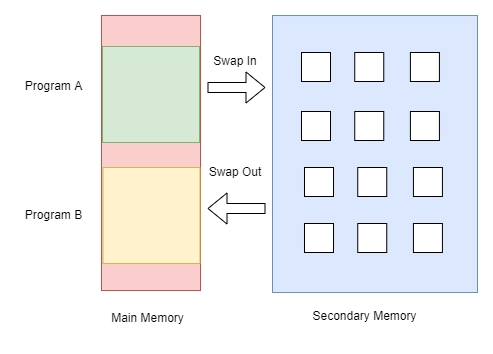


* **What is Demand Paging in OS?**

The demand paging system is similar to the swapping paging system in that processes are mostly stored in the main memory (usually on the hard disk). As a result, demand paging is a procedure that addresses the problem above just by shifting pages on demand. Lazy swapper is another name for this ( It never swaps the page into the memory unless it is needed).

A pager is a kind of swapper that deals with the individual pages of a process.

Demand Paging is a method in which a page is only brought into main memory when the CPU requests it. At first, just those pages are loaded directly required by the operation. Pages that are never accessed are thus never loaded into physical memory.



* ***How does Demand Paging in OS work?***

When the CPU requests access to any page, the page table is used to find the page in the main memory. If the page is found on the main memory, it is good, and if it is not, then a page fault occurs.

Page fault is when the CPU wants to access the page from the main memory, but it is not present in the main memory. Then, how to overcome this?

For this swapped-in is used, the swapped-in is used to swap from the secondary memory. Swapped-in refers to moving a program back to the hard drive from the main memory, or RAM. However, if the page is already in the main memory, it is retrieved from there. The secondary memory is loaded with other pages. There are also valid and invalid bits, which are used to check whether the page is present in the main memory. Valid bits mean that the page is legal and present in the memory, and invalid bits mean the page is not valid or it is not present in the memory.

* ***Common Terms in Demand Paging Operating System***

Following are some common terms in demand paging operating systems:

1. **Page Fault**
2. **Swapping**
3. **Thrashing**

1**.Page Fault**

There will be a miss if the referenced page is not present in the main memory; this is known as a page miss or page fault.

The CPU must look up the missing page in secondary memory. When the number of page faults is significant, the system's effective access time increases dramatically.

**2. Swapping**

Swapping comprises either erasing all of the process's pages from memory or marking the pages so that we can remove them via the page replacement method.

When a process is suspended, it indicates it is unable to run. However, we can change the process for a while. The system can swap the process from secondary memory to primary memory over a period of time. Thrashing describes a condition in which a process is busy, and the pages are swapped in and out of it.

**3. Thrashing**

The effective access time will be the time needed by the CPU to read one word from the secondary memory if the number of page faults is equal to the number of referred pages or if the number of page faults is so high that the CPU is only reading pages from the secondary memory. This is known as Thrashing.

If the page fault rate is PF%, the time spent retrieving a page from secondary memory and resuming is S (service time), and the memory access time is “ma”, the effective access time may be calculated as follows:

EAT = PF x S + (1 - PF) x (ma)

***Advantages***

Following are the advantages of Demand Paging −

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

***Disadvantages***

* Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.
* **What is virtual memory?**

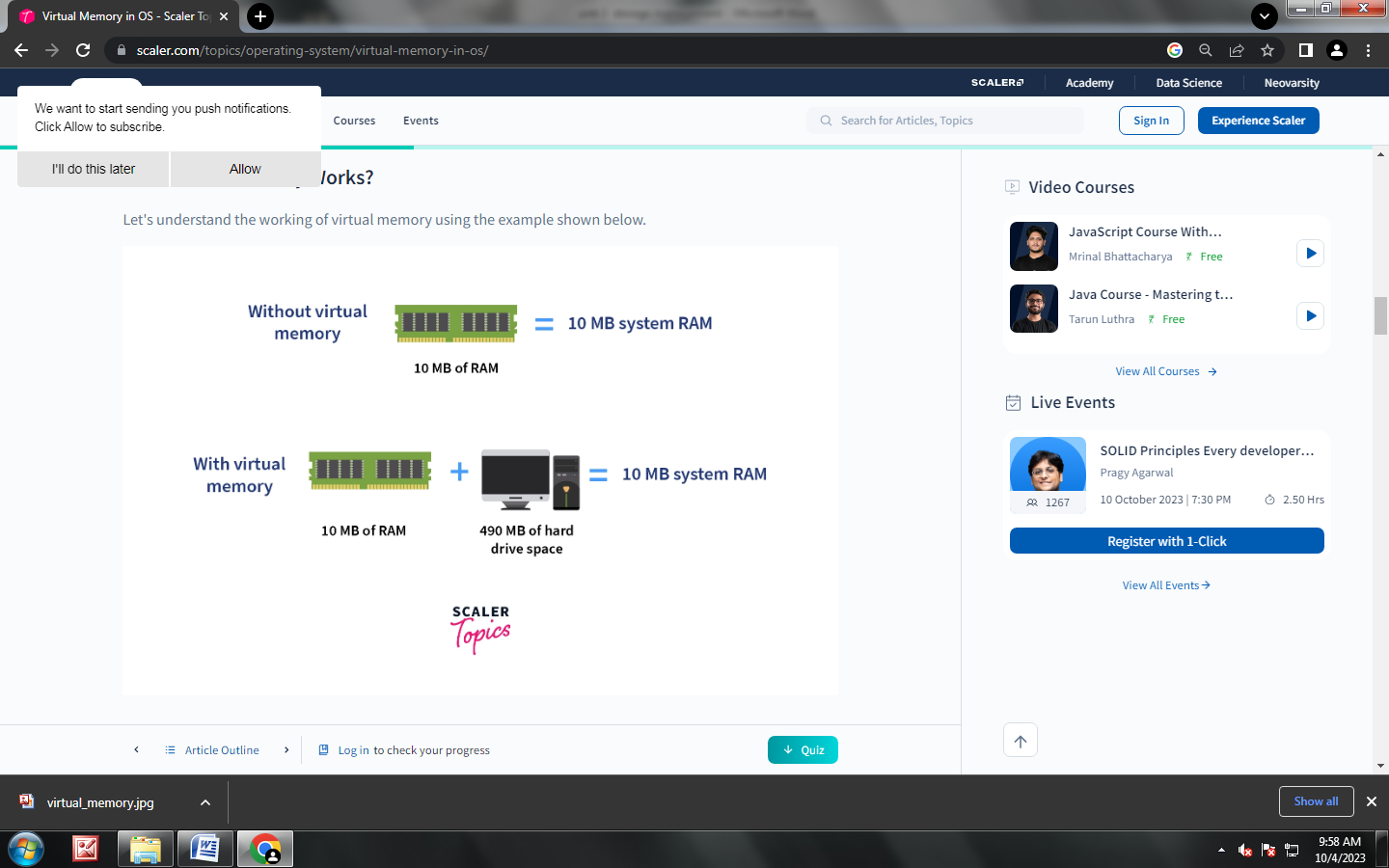
Virtual memory is a memory management technique where secondary memory can be used as if it were a part of the main memory. Virtual memory is a common technique used in a computer's operating system (OS).

Virtual memory uses both hardware and software to enable a computer to compensate for physical memory shortages, temporarily transferring data from random access memory ([RAM](https://www.techtarget.com/searchstorage/definition/RAM-random-access-memory)) to disk storage. Mapping chunks of memory to disk files enables a computer to treat secondary memory as though it were main memory.

Today, most personal computers (PCs) come with at least 8 GB (gigabytes) of RAM. But, sometimes, this is not enough to run several programs at one time. This is where virtual memory comes in. Virtual memory frees up RAM by swapping data that has not been used recently over to a storage device, such as a hard drive or solid-state drive ([SSD](https://www.techtarget.com/searchstorage/definition/SSD-solid-state-drive)).

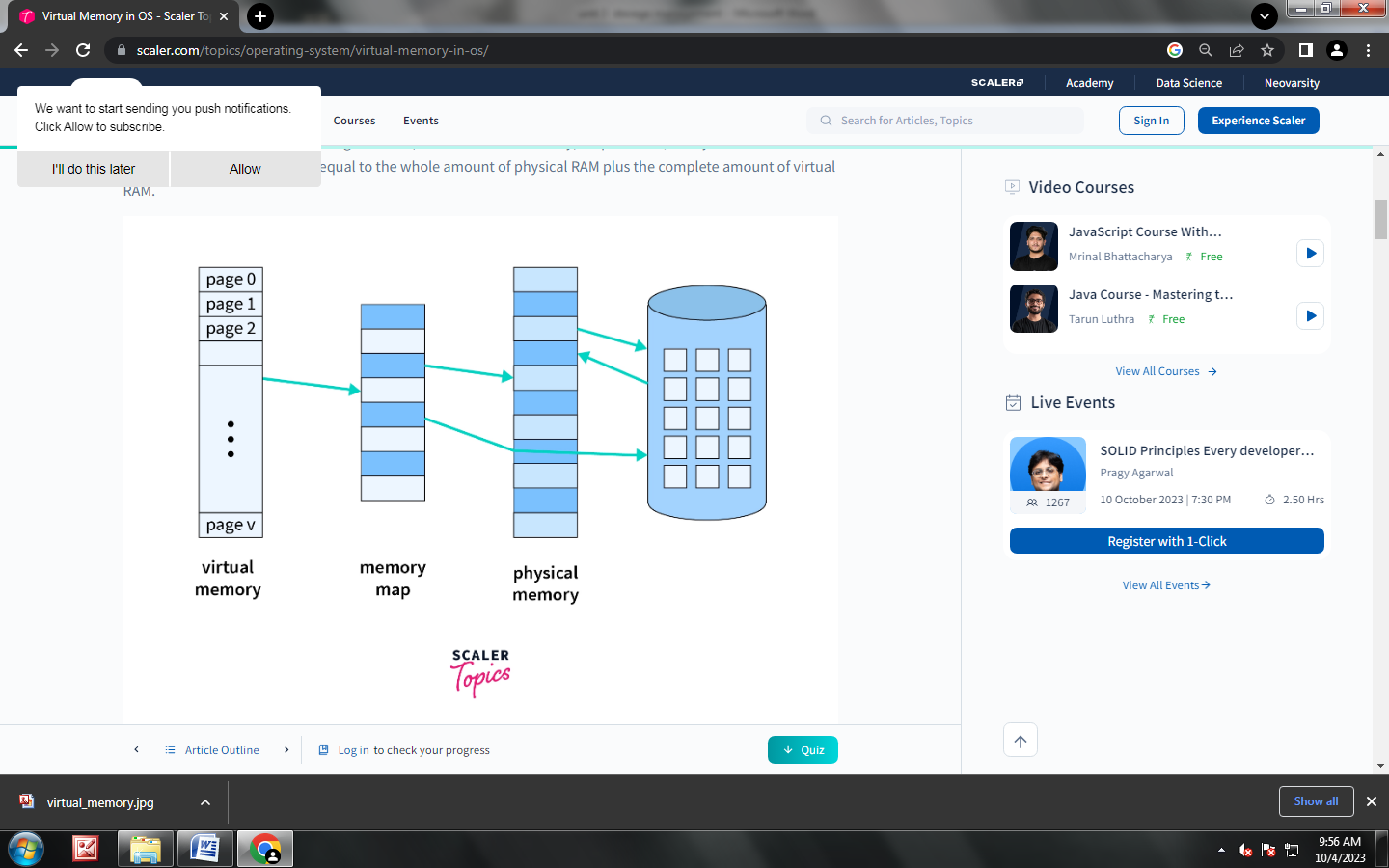
Virtual memory is important for [improving system performance](https://searchservervirtualization.techtarget.com/feature/Memory-management-techniques-you-should-remember), multitasking and using large programs. However, users should not overly rely on virtual memory, since it is considerably slower than RAM. If the OS has to swap data between virtual memory and RAM too often, the computer will begin to slow down -- this is called [thrashing](https://www.techtarget.com/whatis/definition/thrashing).

Virtual memory was developed at a time when physical memory -- also referenced as RAM -- was expensive. Computers have a finite amount of RAM, so memory will eventually run out when multiple programs run at the same time. A system using virtual memory uses a section of the hard drive to emulate RAM. With virtual memory, a system can load larger or multiple programs running at the same time, enabling each one to operate as if it has more space, without having to purchase more RAM.



* **How Virtual Memory Works?**

Let's understand the working of virtual memory using the example shown below.



Assume that an operating system uses 500 MB of RAM to hold all of the running processes. However, there is now only 10 MB of actual capacity accessible on the RAM. The operating system will then allocate 490 MB of virtual memory and manage it with an application called the Virtual Memory Manager (VMM). As a result, the VMM will generate a 490 MB file on the hard disc to contain the extra RAM that is necessary. The OS will now proceed to address memory, even if only 10 MB of space is available because it considers 500 MB of actual memory saved in RAM. It is the VMM's responsibility to handle 500 MB of memory, even if only 10 MB is available.

***Advantages of Virtual Memory in OS***

* Increases the degree of multiprogramming: You can run many applications at once without buying more memory RAMs.
* **Data Sharing:** Data that is common can be shared between memory.
* **Avoids Relocation:** The code in the physical memory can be accessed whenever required without relocation of the code.
* **Increases memory:** Users can run larger applications in systems that cannot support further RAM. Thus, it can run applications that are larger than the physical memory.
* **Increases effective use of CPU:** Since more processes can be maintained in the memory, the CPU is more effectively used. Each page is stored on the disk while it's in use, and further removed.

***Disadvantages of Virtual Memory in OS***

1. **Makes the system slower:** Swapping takes time affecting the system's speed and making it slower. This happens as it takes more time to switch between applications.
2. **Lesser Hard Disk Space:** Virtual Memory takes up storage space that could otherwise be used for long-term data storage.
3. **Reduces System Stability:** It negatively affects the overall performance of a system

* **Page Replacement Algorithms in Operating Systems**

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.

**Page Fault:** A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page faults happen. In case of a page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

**Page Hit**: When we want to load the page on the memory, and the page is already available on memory, then it is called page hit.

### (FIFO)First in First Out Algorithm

This is the first basic algorithm of Page Replacement Algorithms. This algorithm is basically dependent on the number of frames used. Then each frame takes up the certain page and tries to access it. When the frames are filled then the actual problem starts. The fixed number of frames is filled up with the help of first frames present. This concept is fulfilled with the help of Demand Paging

After filling up of the frames, the next page in the waiting queue tries to enter the frame. If the frame is present then, no problem is occurred. Because of the page which is to be searched is already present in the allocated frames.

If the page to be searched is found among the frames then, this process is known as Page Hit.

If the page to be searched is not found among the frames then, this process is known as Page Fault.

When Page Fault occurs this problem arises, then the First In First Out Page Replacement Algorithm comes into picture.

The First In First Out (FIFO) Page Replacement Algorithm removes the Page in the frame which is allotted long back. This means the useless page which is in the frame for a longer time is removed and the new page which is in the ready queue and is ready to occupy the frame is allowed by the First In First Out Page Replacement.

Let us understand this First In First Out Page Replacement Algorithm working with the help of an example.

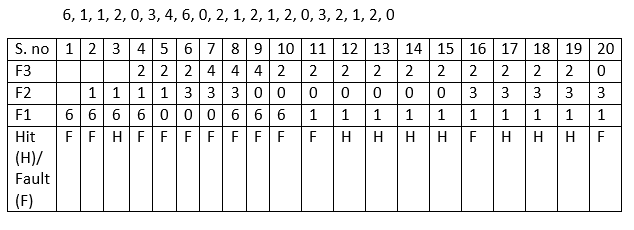
**Example:**

Consider the reference string **6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0** for a memory with three frames and calculate number of page faults by using FIFO (First In First Out) Page replacement algorithms.

**Points to Remember**

Page Not Found - - - > Page Fault

Page Found - - - > Page Hit

**Reference String:**

Number of **Page Hits = 8**

Number of **Page Faults = 12**

The Ratio of Page Hit to the Page Fault = 8 : 12 - - - > 2 : 3 - - - > 0.66

The Page Hit Percentage = 8 \*100 / 20 = 40%

The Page Fault Percentage = 100 - Page Hit Percentage = 100 - 40 = 60%

**Explanation**

First, fill the frames with the initial pages. Then, after the frames are filled we need to create a space in the frames for the new page to occupy. So, with the help of First in First Out Page Replacement Algorithm we remove the frame which contains the page is older among the pages. By removing the older page we give access for the new frame to occupy the empty space created by the First in First out Page Replacement Algorithm.

***FOR EXPLANING THIS ALGORITHAM CLICK TO BELOW LINK***

[***https://youtu.be/8rcUs5RutX0?si=WJ9Jmgw69YNXg1bw***](https://youtu.be/8rcUs5RutX0?si=WJ9Jmgw69YNXg1bw)

### Optimal Page Replacement Algorithm

This is the second basic algorithm of Page Replacement Algorithms. This algorithm is basically dependent on the number of frames used. Then each frame takes up the certain page and tries to access it. When the frames are filled then the actual problem starts. The fixed number of frames is filled up with the help of first frames present. This concept is fulfilled with the help of Demand Paging

After filling up of the frames, the next page in the waiting queue tries to enter the frame. If the frame is present then, no problem is occurred. Because of the page which is to be searched is already present in the allocated frames.

If the page to be searched is found among the frames then, this process is known as Page Hit.

If the page to be searched is not found among the frames then, this process is known as Page Fault.

When Page Fault occurs this problem arises, then the OPTIMAL Page Replacement Algorithm comes into picture.

The OPTIMAL Page Replacement Algorithms works on a certain principle. The principle is:

Replace the Page which is not used in the Longest Dimension of time in future

This principle means that after all the frames are filled then, see the future pages which are to occupy the frames. Go on checking for the pages which are already available in the frames. Choose the page which is at last.

Let us understand this Least Recently Used (LRU) Page Replacement Algorithm working with the help of an example.

**Example:**

Consider the reference string **7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1** for a memory with three frames and calculate number of page faults by using Least Recently Used (LRU) Page replacement algorithms.

**Points to Remember**

Page Not Found - - - > Page Fault

Page Found - - - > Page Hit

**Reference String:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **F4** |  |  |  | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| **F3** |  |  | 1 | 1 | 1 | 1 | 1 | 4 | 4 | 4 | 4 | 4 | 4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| **F2** |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **F1** | 7 | 7 | 7 | 7 | 7 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 7 | 7 | 7 |
| Ref. string | **7** | **0** | **1** | **2** | **0** | **3** | **0** | **4** | **2** | **3** | **0** | **3** | **2** | **1** | **2** | **0** | **1** | **7** | **0** | **1** |
| Fault(F)/Hit(H) | F | F | F | F | H | F | H | F | H | H | H | H | H | F | H | H | H | F | H | H |

Number of **Page Hits = 12**

Number of **Page Faults = 8**

The Ratio of Page Hit to the Page Fault = 12 : 8 - - - > 1.5

The Page Hit Percentage = 12 \* 100 / 20 = 60%

The Page Fault Percentage = 100 - Page Hit Percentage = 100 - 60 = 40%

***Explanation***

First, fill the frames with the initial pages. Then, after the frames are filled we need to create a space in the frames for the new page to occupy

Here, we would fill the empty spaces with the pages we and the empty frames we have. The problem occurs when there is no space for occupying of pages. We have already known that we would replace the Page which is not used in the **Longest Dimension of time in future.**

There comes a question what if there is absence of page which is in the frame.

***FOR EXPLANING THIS ALGORITHAM CLICK TO BELOW LINK***

[***https://youtu.be/q2BpMvPhhrY?si=fwORRtsmjbo\_F7Y0***](https://youtu.be/q2BpMvPhhrY?si=fwORRtsmjbo_F7Y0)

### Least Recently Used (LRU) Replacement Algorithm

This is the last basic algorithm of Page Replacement Algorithms. This algorithm is basically dependent on the number of frames used. Then each frame takes up the certain page and tries to access it. When the frames are filled then the actual problem starts. The fixed number of frames is filled up with the help of first frames present. This concept is fulfilled with the help of Demand Paging

After filling up of the frames, the next page in the waiting queue tries to enter the frame. If the frame is present then, no problem is occurred. Because of the page which is to be searched is already present in the allocated frames.

If the page to be searched is found among the frames then, this process is known as Page Hit.

If the page to be searched is not found among the frames then, this process is known as Page Fault.

When Page Fault occurs this problem arises, then the Least Recently Used (LRU) Page Replacement Algorithm comes into picture.

The Least Recently Used (LRU) Page Replacement Algorithms works on a certain principle. The principle is:

Replace the page with the page which is less dimension of time recently used page in the past.

Let us understand this Least Recently Used (LRU) Page Replacement Algorithm working with the help of an example.

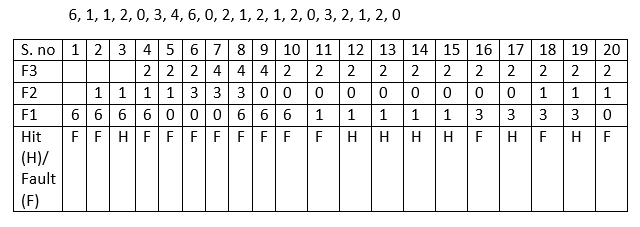
***Example:***

Consider **the reference string 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0** for a memory with three frames and calculate number of page faults by using Least Recently Used (LRU) Page replacement algorithms.

Points to Remember

Page Not Found - - - > Page Fault

Page Found - - - > Page Hit

***Reference String:***

Number of **Page Hits = 7**

Number of **Page Faults = 13**

The Ratio of Page Hit to the Page Fault = 7 : 12 - - - > 0.5833 : 1

The Page Hit Percentage = 7 \* 100 / 20 = 35%

The Page Fault Percentage = 100 - Page Hit Percentage = 100 - 35 = 65%

***Explanation***

First, fill the frames with the initial pages. Then, after the frames are filled we need to create a space in the frames for the new page to occupy.

Here, we would fill the empty spaces with the pages we and the empty frames we have. The problem occurs when there is no space for occupying of pages. We have already known that we would replace the Page which is not used in the Longest Dimension of time in past or can be said as the Page which is very far away in the past.

***FOR EXPLANING THIS ALGORITHAM CLICK TO BELOW LINK***

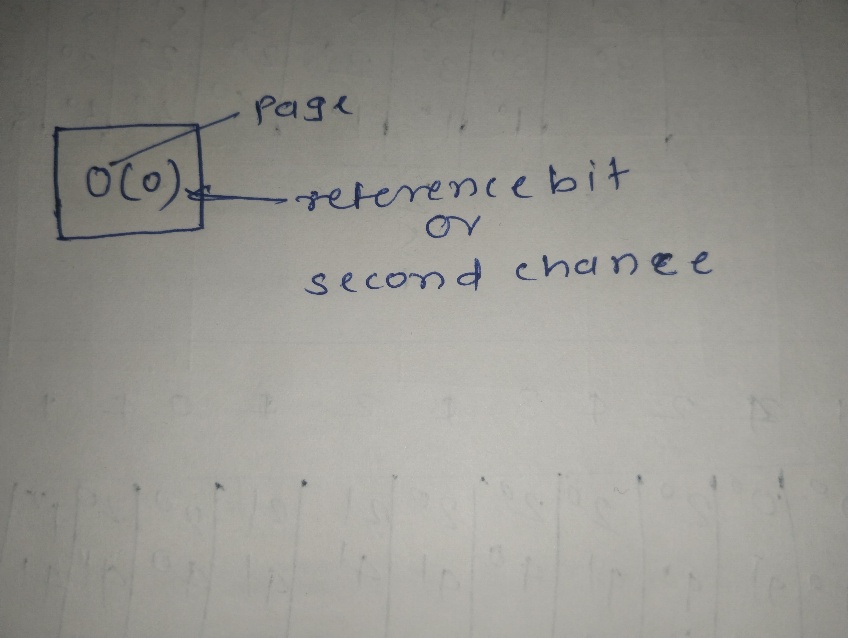
[**https://youtu.be/dYIoWkCvd6A?si=S7KaCPpNxsSmZKwJ**](https://youtu.be/dYIoWkCvd6A?si=S7KaCPpNxsSmZKwJ)

### Clock (or Second Chance) Page Replacement

In the Second Chance page replacement policy, the candidate pages for removal are considered in a round robin matter, and a page that has been accessed between consecutive considerations will not be replaced. The page replaced is the one that, when considered in a round robin matter, has not been accessed since its last consideration.

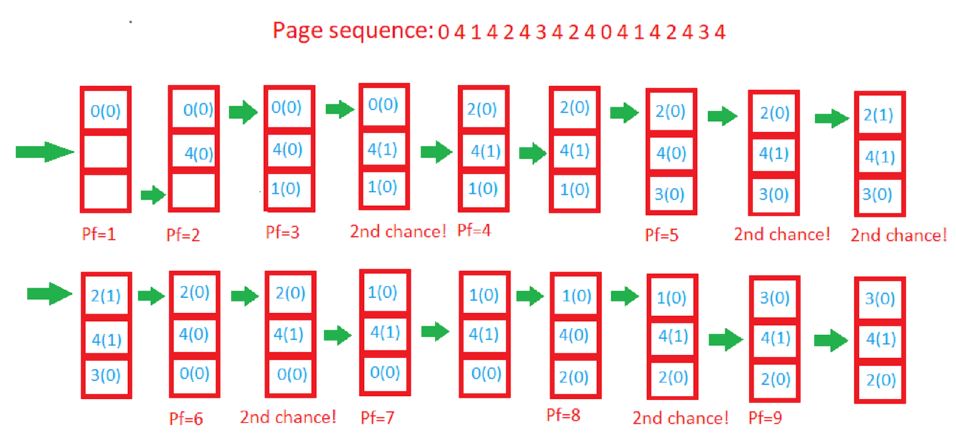
It can be implemented by adding a “Reference Bit ”to each memory frame-every time the frame is considered (due to a reference made to the page inside it), this bit is set to 1, which gives the page a second chance, as when we consider the candidate page for replacement, we replace the first one with this bit set to 0 (while zeroing out bits of the other pages we see in the process). Thus, a page with the **“Reference Bit/Second chance” set to 1 is never replaced** during the first consideration and will only be replaced if all the other pages deserve a second chance too!

Traditionally Second Chance and Clock are believed to be less efficient than LRU (having a higher miss ratio). Recent research from Carnegie Mellon University finds that Second Chance and Clock are more efficient than LRU with a lower miss ratio. Because Second Chance and Clock is faster and more scalable than LRU, this means LRU has no advantage over Second Chance and Clock anymore.



**Example –**

Let’s say the **reference string is 0 4 1 4 2 4 3 4 2 4 0 4 1 4 2 4 3 4** and we have **3 frames**. Let’s see how the algorithm proceeds by tracking the second chance bit and the pointer.



***Explanation***

***No of pages = 18***

Initially, all frames are empty so after first 3 passes they will be filled with {0, 4, 1} and the **second chance/ Reference bit** array will be {0, 0, 0} as none has been referenced yet. Also, the pointer will cycle back to 0.

**Pass-4**: ***Frame={0, 4, 1}, Reference bit = {0, 1, 0}*** [4 will get a second chance], pointer = 0 (No page needed to be updated so the candidate is still page in frame 0), pf = 3 (No increase in page fault number).

**Pass-5:** ***Frame={2, 4, 1}, second\_chance= {0, 1, 0}*** [0 replaced; it’s second chance bit was 0, so it didn’t get a second chance], pointer=1 (updated), pf=4

**Pass-6:** ***Frame={2, 4, 1}, second\_chance={0, 1, 0},*** pointer=1, pf=4 (No change)

**Pass-7*: Frame={2, 4, 3}, second\_chance= {0, 0, 0}*** [4 survived but it’s second chance bit became 0], pointer=0 (as element at index 2 was finally replaced), pf=5

**Pass-8:** ***Frame={2, 4, 3}, second\_chance= {0, 1, 0}*** [4 referenced again], pointer=0, pf=5

**Pass-9:** ***Frame={2, 4, 3}, second\_chance= {1, 1, 0}*** [2 referenced again], pointer=0, pf=5

**Pass-10:** ***Frame={2, 4, 3}, second\_chance= {1, 1, 0}***, pointer=0, pf=5 (no change)

**Pass-11:** ***Frame={2, 4, 0}, second\_chance= {0, 0, 0},*** pointer=0, pf=6 (2 and 4 got second chances)

**Pass-12:** ***Frame={2, 4, 0}, second\_chance= {0, 1, 0},*** pointer=0, pf=6 (4 will again get a second chance)

**Pass-13:** ***Frame={1, 4, 0}, second\_chance= {0, 1, 0},*** pointer=1, pf=7 (pointer updated, pf updated)

**Page-14**: ***Frame={1, 4, 0}, second\_chance= {0, 1, 0},*** pointer=1, pf=7 (No change)

**Page-15:** ***Frame={1, 4, 2}, second\_chance= {0, 0, 0},*** pointer=0, pf=8 (4 survived again due to 2nd chance!)

**Page-16: *Frame={1, 4, 2}, second\_chance= {0, 1, 0},*** pointer=0, pf=8 (2nd chance updated)

**Page-17:** ***Frame={3, 4, 2}, second\_chance= {0, 1, 0},*** pointer=1, pf=9 (pointer, pf updated)

**Page-18:** ***Frame={3, 4, 2}, second\_chance= {0, 1, 0},*** pointer=1, pf=9 (No change)

* **What is Thrashing?**

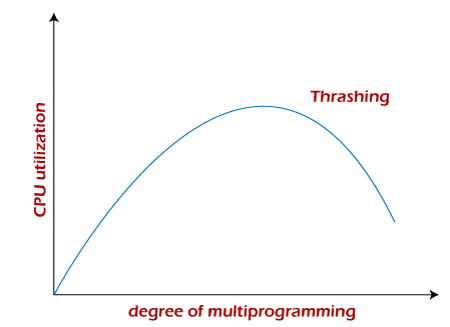
In computer science, **thrash** is the poor performance of a virtual memory (or paging) system when the same pages are being loaded repeatedly due to a lack of main memory to keep them in memory. Depending on the configuration and algorithm, the actual throughput of a system can degrade by multiple orders of magnitude.

In computer science, **thrashing** occurs when a computer's virtual memory resources are overused, leading to a constant state of paging and page faults, inhibiting most application-level processing. It causes the performance of the computer to degrade or collapse. The situation can continue indefinitely until the user closes some running applications or the active processes free up additional virtual memory resources.

To know more clearly about thrashing, first, we need to know about page fault and swapping.

* **Page fault:** We know every program is divided into some pages. A page fault occurs when a program attempts to access data or code in its address space but is not currently located in the system RAM.
* **Swapping:** Whenever a page fault happens, the operating system will try to fetch that page from secondary memory and try to swap it with one of the pages in RAM. This process is called swapping.

***Thrashing*** is when the page fault and swapping happens very frequently at a higher rate, and then the operating system has to spend more time swapping these pages. This state in the operating system is known as thrashing. Because of thrashing, the CPU utilization is going to be reduced or negligible.



The basic concept involved is that if a process is allocated too few frames, then there will be too many and too frequent page faults. As a result, no valuable work would be done by the CPU, and the CPU utilization would fall drastically.

The long-term scheduler would then try to improve the CPU utilization by loading some more processes into the memory, thereby increasing the degree of multiprogramming. Unfortunately, this would result in a further decrease in the CPU utilization, triggering a chained reaction of higher page faults followed by an increase in the degree of multiprogramming, called thrashing.

* **Causes of Thrashing**

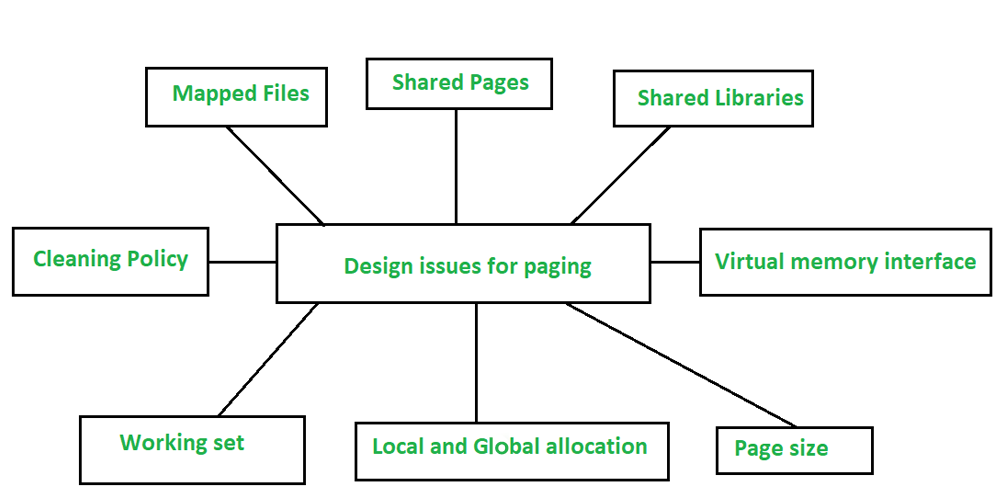
Programs or workloads may cause thrashing, and it results in severe performance problems, such as:

* If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new system. A global page replacement algorithm is used. The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming.
* CPU utilization is plotted against the degree of multiprogramming.
* As the degree of multiprogramming increases, CPU utilization also increases.
* If the degree of multiprogramming is increased further, thrashing sets in, and CPU utilization drops sharply.
* So, at this point, to increase CPU utilization and to stop thrashing, we must decrease the degree of multiprogramming.
* **Design Issues for Paging**

Paging is an important concept in memory management. In Paging, the operating system divides each incoming program into pages of equal sizes of blocks. The section of disks is called blocks. The section of main memory is called page frames. Fixed-size blocks are called frames and the breaking of logical memory into blocks of the same size is called **pages**. Each page can be stored in an available page frame anywhere in the main memory. Memory manager keeps track of pages of programs in memory. The relation between virtual addresses and physical memory addresses is given by the page table. The paging system has made multiprogramming very effective.

To set good performance for the paging system, the issues are :

* **Working set**
* **Local and Global allocation**
* **Page size**
* **Shared pages**
* **Shared libraries**
* **Mapped Files**
* **Cleaning Policy**
* **Virtual memory interface**



### Working Set :

The set of pages whose process is currently in use or it is in execution is called a **working set**. If the entire working set of a process is in the memory, it will execute quickly. The execution of a program follows the principle of **locality of reference**. The locality of reference is a phase of execution where the process references particular pages. The working set can result in page fault if available memory given to the process cannot be handled by the working set. If a program causes page faults every few instructions are called **thrashing**. The working set issue is designed to reduce the page fault rate. The operating system keeps track of the working set of a process. The operating system uses the data of the working set process to reduce the page fault rate.

### Local and Global Allocation:

A **local allocation algorithm** assigns a fixed number of frames to every running process. In the case of a page, a fault occurs in this algorithm, it considers only the pages allocated to the fixed number of frames for replacement.  If the working set size grows, then page fault will increase in case of local allocation. A **global allocation algorithm** dynamically allocates page frames in runnable processes. The number of page frames assigned to each process varies over time. In case of page, a fault occurs in this algorithm, it considers all pages assigned to different processes for page replacement. Global allocation performs better when the working set size can vary greatly over lifetime of a process.

### Page Size :

Page size is an important parameter in designing a paging system. Determining an optimum page size depends on various factors :

* Large page sizes will reduce the number of pages allocated to a process. It is called a **Small page allocation table**.
* Large page sizes will increase the waste of space. On average, the last page allocated to the process is half utilized due to internal fragmentation.
* A smaller page size will increase the size of the page allocation table but the wastage of the last page half utilization is reduced. It is called a **Large page allocation table**.
* **Optimum Page Size = √2\*Se,**where Se =Average process size in ‘e’ bytes.

### Shared Pages :

Shared pages are used to improve the performance of the paging system. There can be a scenario where multiple users might be executing the same jobs at a time. To avoid the duplication of the same pages in the same memory, it is preferable to share the pages. Shared pages are used in order to avoid having two copies of a page in memory at once. Shared pages can only be read by the software that has requested the page, and the data written to a shared page is not visible to other applications. A shared page is a shared memory page that can be used by multiple processes at the same time. Shared pages can be used in place of physical RAM when more memory is needed.  The most advantage of shared pages is that only one copy of a shared file exists in memory, reducing the overhead of pages and allowing more efficient use of RAM.

### Shared Libraries:

A shared library is a file whose contents are divided into blocks, and each block is loaded into memory in units of pages. Shared libraries are a special type of software that provides more than one function for one program. For example, a shared library is loaded into the virtual space when a program starts up. Some functions included in this library can be used by other programs as well. When the user wants to use these functions, then the user needs to load this library into memory again. This process occurs independently of the other programs and usually takes longer than loading individual functions separately. The advantage of a shared library is that the user can replace its original file with another version if the user wants.

With shared libraries, programs can access more libraries and functions by using only one shared library. Another advantage of using the shared library is that if the user modifies the function in the shared library without recompiling the program, it will still work as before. An example of a program that uses a shared library is a graphics application that calls graphics routines in the graphics library (such as GDI). If you upgrade your graphics library, there is no need to recompile your GDI program because it uses references to functions in the new version instead.

### Mapped files :

A memory-mapped file is a shared memory object that allows processes to share its memory. The hardware is responsible for allocating the space needed on the page used to store a mapped file. Mapped files are created by using a mapping process called a “mapping table”. A process can map a file to a part of its virtual address space by making a system call. Mapped files are specialized shared memory structures that support the sharing of a file among many processes. Essentially, they allow one process to access another’s portion of the file as its own virtual address space. When writing to a structure, the mapped page always refers to the same physical page in the file system. Mapped pages thus can more efficiently use available memory and storage space on a hard drive than regular files. Mapped files are also used for communication between processes.

### Cleaning Policy :

The paging daemon is an important component of the paging system. The paging system continuously keeps track of the number of free page frames in the system and ensures that they are used before they are no longer needed. If there are too few free page frames, the paging daemon selects pages that it needs to use through a page replacement algorithm.

### Virtual Memory Interface:

A virtual memory interface (VMI) is a type of hardware that allows a program to access system memory, irrespective of whether memory actually exists. Memory mapping is the translation of a virtual address within the local address space to an actual memory location. This type of virtual memory provides off-hardware access to different processes using separate pages. When a process requests memory mapping, a virtual memory table is created that contains information about where each page exists in physical memory.

Virtual memory is used in operating systems to allow multiple processes to share the same physical memory storage space. This means that the accesses are done through a technique called blocking, which means that when an access occurs, the CPU must wait until all other accesses have been completed. The benefits of virtual memory include ease of use, can run multiple processes on a computer at once, and sharing of memory between them.