**UNIT – V**

**Virtual Memory Systems**

Physical memory is the actual real memory used in RAM. Virtual memory as the name suggests is not real. The OS uses virtual memory as a memory management technique in which non-contiguous memory is presented to software as contiguous memory. If the RAM falls short of memory to accommodate more running processes, the OS allocates a portion of your hard drive to act as though it were RAM. That's what is referred to as **virtual memory**.

Physical memory is the only memory that is directly accessible to the CPU. CPU reads the instructions stored in the physical memory and executes them continuously. The data that is operated will also be stored in physical memory in uniform manner. Virtual memory is one classification of memory which was created by using the hard disk for simulating additional RAM, the addressable space available for the user. Virtual addresses are mapped into real addresses.

**Concept of Paging**

Paging is a storage structure that enables the operating framework to fetch processes from the secondary storage into the main memory in the form of pages. In the Paging method, the main memory is split into small fixed-size blocks of physical memory, which is known as frames. The size of a frame must be preserved the same as that of a page to have maximum use of the main memory and to prevent external fragmentation.

Paging changes pages from the swap disk to frames of the physical memory therefore data can be accessed by the processor. Any page can involve any frame. This leads to multiple issues that should be undertaken by a paging system:

* When should a page be changed into physical memory?
* How does the CPU find data in physical memory, especially if its logical address is not the same as its physical address?
* What arises when all the frames have pages and the CPU require to access data from a page not recently saved in physical memory?

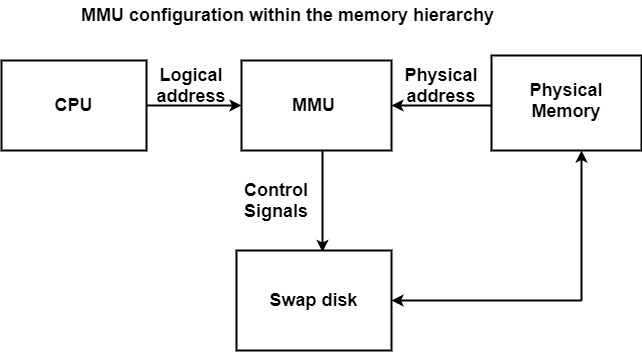
All of these issues are managed by the memory management unit (MMU). As displayed in the figure, the logical address is output from the CPU to the MMU. The MMU converts this address to a physical address, which is supplied to the cache and physical memory. If the data is not placed in physical memory, it creates a page fault and changes the page from the swap disk to a frame, eliminating another page if essential.

Paging is a storage structure that enables the operating framework to fetch processes from the secondary storage into the main memory in the form of pages. In the Paging method, the main memory is split into small fixed-size blocks of physical memory, which is known as frames. The size of a frame must be preserved the same as that of a page to have maximum use of the main memory and to prevent external fragmentation.

Paging changes pages from the swap disk to frames of the physical memory therefore data can be accessed by the processor. Any page can involve any frame. This leads to multiple issues that should be undertaken by a paging system −

* When should a page be changed into physical memory?
* How does the CPU find data in physical memory, especially if its logical address is not the same as its physical address?
* What arises when all the frames have pages and the CPU require to access data from a page not recently saved in physical memory?

All of these issues are managed by the memory management unit (MMU). As displayed in the figure, the logical address is output from the CPU to the MMU. The MMU converts this address to a physical address, which is supplied to the cache and physical memory. If the data is not placed in physical memory, it creates a page fault and changes the page from the swap disk to a frame, eliminating another page if essential.



* Paging is the process of swapping a page from the virtual memory space into the main memory.
* Processes are divided into pages that are stored in the virtual memory.
* These are fixed-size units.
* The main memory is divided into frames, that are of the same size as the page.
* Generally, the range of these frames and pages are from 512 bytes to 64 kb.

Different algorithms are used to perform **page replacement**. We will learn about some of the most common algorithms in this lesson.

Basic terminologies used in page replacement:

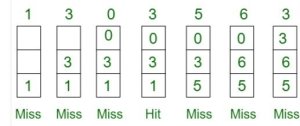
* **Page:** It is a fixed-length contiguous block of virtual memory.
* **Logical Address/Virtual Address:** It is an address generated by the CPU.
* **Logical Address Space:** It is the set of all logical addresses that are generated by a program.
* **Physical Address:** It is the address that is available on the memory unit.
* **Physical Address Space:** It is the set of all physical addresses that correspond to the logical addresses.
* **Page Fault:** When a running program accesses a memory page that is mapped into the virtual address space but is not loaded in physical memory, it is known as a page fault.

**First in first out (FIFO) page replacement algorithm**

Thisis the simplest page replacement algorithm. In this algorithm, the operating system keeps a track of all the pages present in the memory in a queue fashion. When a page needs to be replaced, the oldest page in the queue is selected and replaced with the new page.

For instance, let’s suppose we have a **page reference string as (1, 3, 0, 3, 5, 6, 3)** with **three-page frames**. Initially, all the slots are empty so the first three reference strings (1, 3, 0) will be allocated to the empty slots with 3 page-faults. Next in the reference string is (3) which is already in the memory, so no page fault. Then comes (5). Since there is no available memory space the newly arrived page will replace the oldest residing page in the memory, i.e., (1) with 1 page-fault. Likewise, now (6) will arrive and will replace (3) with another page fault. And then (3) will arrive and will replace (0) with another page fault.

Every time the OS has to replace a page, a page-fault is counted.

[](https://i0.wp.com/technobyte.org/wp-content/uploads/2019/12/sym.jpg?ssl=1)

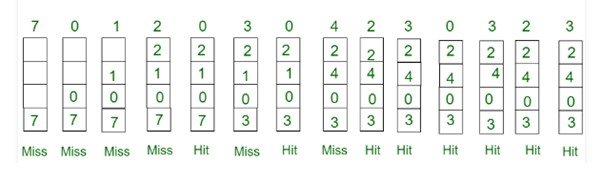
Total Page Fault: 6

**Optimal page replacement algorithm**

Optimal Page Replacement algorithm says that the newly arrived page will replace a page in memory which wouldn’t be used for the longest period of time in the future.

Let’s understand this through an example. Let’s consider a page reference string (7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2) with 4-page frames.

Initially all the memory slots will be empty so (7, 0, 1, 2) will be allocated to the memory with 4 page-faults. As (0) is already there, there’s no page fault. Next in the string is (3), it’ll replace (7) as it’s not used for the longest period of time in the future, with one page fault. Again, 0 is already there, so no page fault. 4 will replace 1 with one page fault. And for the rest of the string, there will be no page fault as all the arriving pages are already there in the memory.

[](https://i0.wp.com/technobyte.org/wp-content/uploads/2019/12/sym-1.jpg?ssl=1)

Total Page Fault = 6

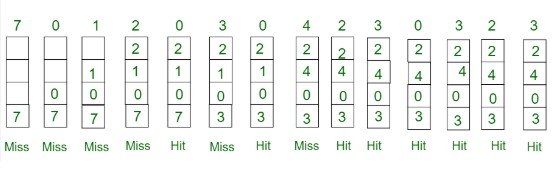
Now, as we understand this algorithm, we tend to realize that this algorithm can be implemented in practice as an operating system cannot know the future page requests in advance. So, this algorithm is just an instance of what can be the optimal solution.

**Least Recently Used (LRU) page replacement algorithm**

In this algorithm, the new page is replaced with the existing page that has been used the least recently. In other words, the page which has not been referred for a long time will be replaced with the newly arrived page.

This algorithm is just opposite to the Optimal Page Replacement algorithm.

So, if we have a page reference string (7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2) with four page frames, then the page replacement will take place as shown below:

[](https://i0.wp.com/technobyte.org/wp-content/uploads/2019/12/sym-2.jpg?ssl=1)

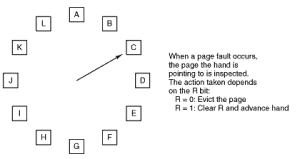
Steps:

1. (7): It’s inserted into the empty frame, as the first four frames are empty initially. +1 page fault.
2. (0): This page string also finds an empty frame. +1 page fault.
3. (1): This page string also finds an empty frame. +1 page fault.
4. (2): This page string also finds an empty frame. +1 page fault.
5. (0): As this string is already available in the page frames, it hits. Page fault remains the same.
6. (3): Page string (7) has been used the least recently, so it gets replaced by (3). +1 page fault.
7. (0): We get another hit as (0) is already in the frame. Page fault remains the same.
8. (4): Page string (1) has been used the least recently, so it gets replaced by (4). +1 page fault.
9. (2): This string is already in the frame.
10. (3): This string is already in the frame.
11. (0): This string is already in the frame.
12. (3): This string is already in the frame.
13. (2): This string is already in the frame.
14. (3): This string is already in the frame.

Total Page Fault = 6

**Clock page replacement algorithm**

The Second Chance Page Replacement algorithm has a drawback that it constantly moves the pages around on its list. To solve this, the OS keeps all the page frames on a circular list in the form of a clock. The hand of the clock (pointer) points to the oldest page.

[](https://i0.wp.com/technobyte.org/wp-content/uploads/2019/12/sym.png?ssl=1)

When a page fault takes place, the page being pointed is inspected. If its R bit is 0, the page is evicted and the new page is inserted into its place. The pointer moves one position ahead.

On the other hand, if R is 1, then it’s cleared and the hand is advanced to the next page until a page with a false R bit is found. Since this works the way a clock works, it is called Clock Page Replacement Algorithm.

**Working Set page replacement algorithm**

In its truest form, the processes are started up with no pages in the memory. One by one, as the pages are required, they are loaded into the memory. This strategy is called **demand paging**as the pages are loaded when needed and not in advance.

The set of pages required to execute a process is called its **working set**. Typically, if the entire working set is in memory, then the process is executed without any page faults until it moves into a new execution phase.

If the working set is not entirely in the memory, many page faults may occur and this situation is known as **thrashing**. This is really bad for the system as it wastes considerable resources.

To prevent thrashing, many paging systems keep track of each process’ working set and try to load the working set-in memory before the process is run. This approach is called the working set model. This model emphasizes on greatly reducing the page faults. This method is also called as prepaging.