VIRTUAL MEMORY

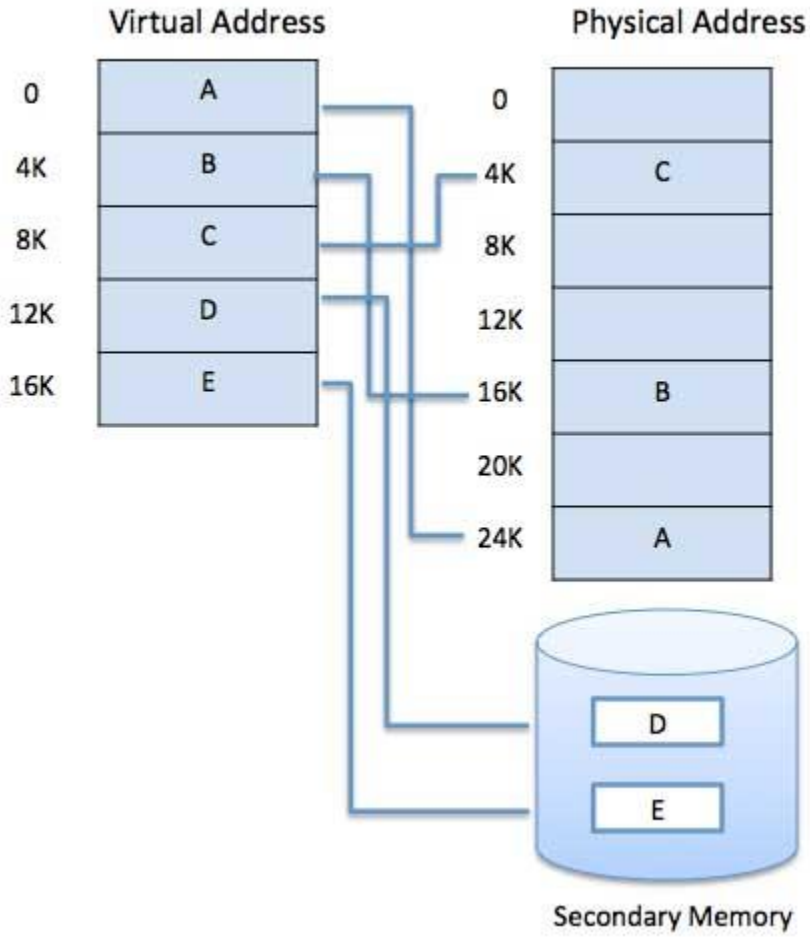
A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.

The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available.
- Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below –



Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

External fragmentation occurs because we allocate memory continuously to the processes. Due to this space is left and memory remains unused hence, cause external fragmentation. So to tackle this problem the concept of paging was introduced where we divide the process into small pages and these pages are allocated memory non-contiguously into the RAM. So, let's get started and learn more about it.

Non-Contiguous Memory Allocation Technique

In the non-contiguous memory allocation technique, different parts of the same process are stored in different places of the main memory. Types:

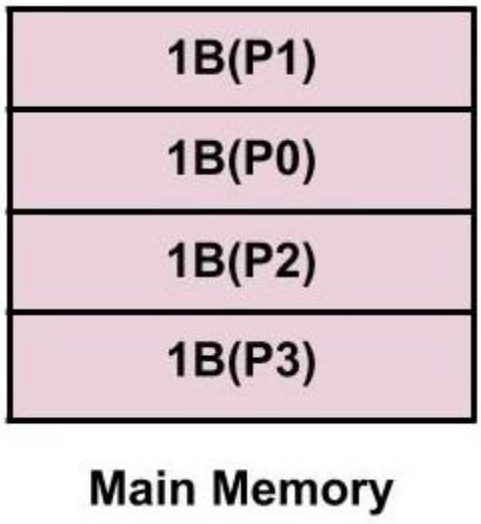
1. Paging

2. Segmentation

Paging

Paging is a non-contiguous memory allocation technique in which secondary memory and the main memory is divided into equal size partitions. **The partitions of the secondary memory are called pages while the partitions of the main memory are called frames.** They are divided into equal size partitions to have maximum utilization of the main memory and avoid external fragmentation.

Example: We have a process P having process size as 4B, page size as 1B. Therefore there will we four pages(say, P0, P1, P2, P3) each of size 1B. Also, when this process goes into the main memory for execution then depending upon the availability, it may be stored in non-contiguous fashion in the main memory frame as shown below:



This is how paging is done.

Translation of logical Address into physical Address

As a CPU always generates a logical address and we need a physical address for accessing the main memory. This mapping is done by the MMU(memory management Unit) with the help of the page table. Lets first understand some of the basic terms then we will see how this translation is done.

- **Logical Address:** The logical address consists of two parts

  page number and page offset.

1. **Page Number:** It tells the exact page of the process which the CPU wants to access.

2. **Page Offset:** It tells the exact word on that page which the CPU wants to read.

Logical Address = Page Number + Page Offset

- Physical Address: The physical address consists of two parts frame number and page offset.

1. Frame Number: It tells the exact frame where the page is stored in physical memory.

2. Page Offset: It tells the exact word on that page which the CPU wants to read. It requires no translation as the page size is the same as the frame size so the place of the word which CPU wants access will not change.

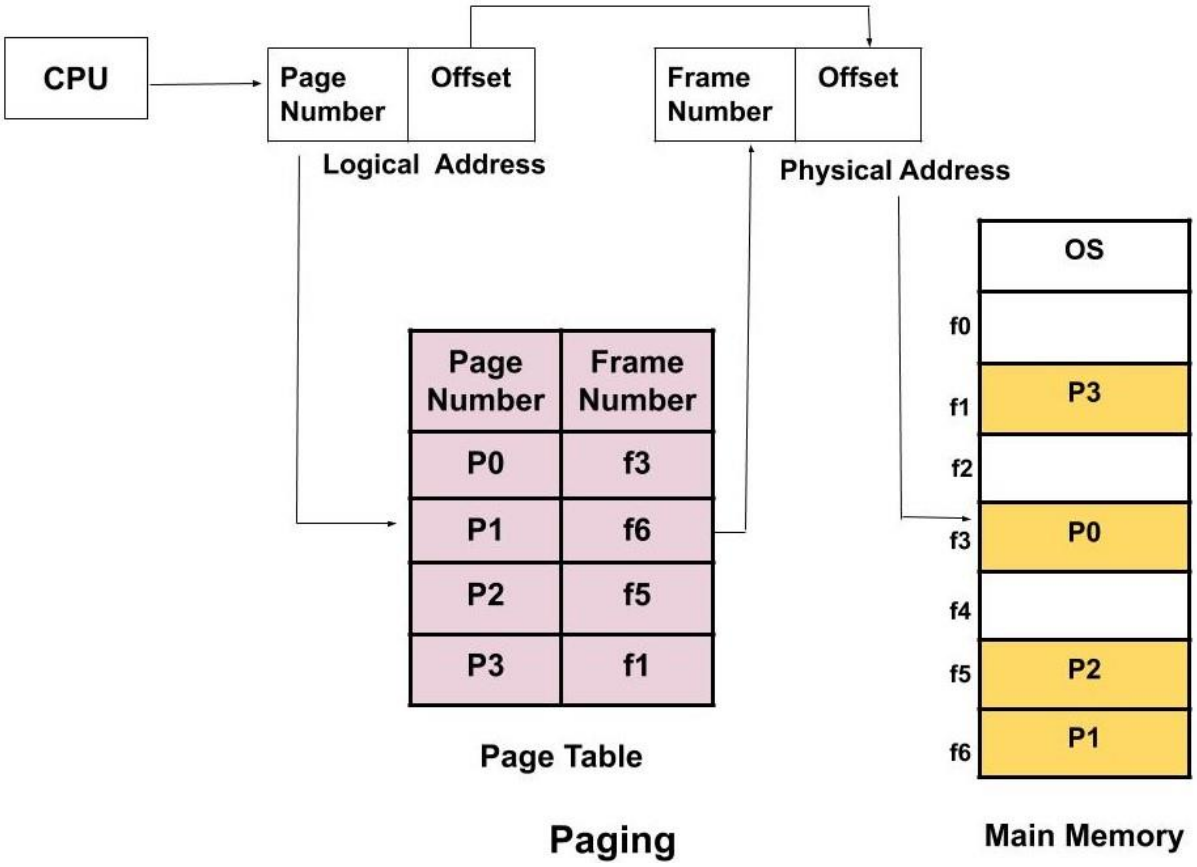
Physical Address = Frame Number + Page Offset

- Page table: A page stable contains the frame number corresponding to the page number of some specific process. So, each process will have its own page table. A register called Page Table Base Register(PTBR) which holds the base value of the page table.

Now, let's see how the translation is done.

How is the translation done?

The CPU generates the logical address which contains the page number and the page offset. The PTBR register contains the address of the page table. Now, the page table helps in determining the frame number corresponding to the page number. Now, with the help of frame number and the page offset the physical address is determined and the page is accessed in the main memory.



Advantages of Paging

1. There is no external fragmentation as it allows us to store the data in a non-contiguous way.
2. Swapping is easy between equal-sized pages and frames.

Disadvantages of Paging

1. As the size of the frame is fixed, so it may suffer from internal fragmentation. It may happen that the process is too small and it may not acquire the entire frame size.
2. The access time increases because of paging as the main memory has to be now accessed two times. First, we need to access the page table which is also stored in the main memory and second, combine the frame number with the page offset and then get the physical address of the page which is again stored in the main memory.
3. For every process, we have an independent page table and maintaining the page table is extra overhead.

Segmentation

In paging, we were blindly diving the process into pages of fixed sizes but in segmentation, we divide the process into modules for better visualization of the process. Here each segment or module consists of the same type of functions. For example, the main function is included in one segment, library function is kept in other segments, and so on. As the size of segments may vary, so memory is divided into variable size parts.

Translation of logical Address into physical Address

As a CPU always generates a logical address and we need a physical address for accessing the main memory. This mapping is done by the MMU(memory management Unit) with the help of the segment table.

Lets first understand some of the basic terms then we will see how this translation is done.

- **Logical Address:** The logical address consists of two parts

  segment number and segment offset.

1. Segment Number: It tells the specific segment of the process from which the CPU wants to read the data.

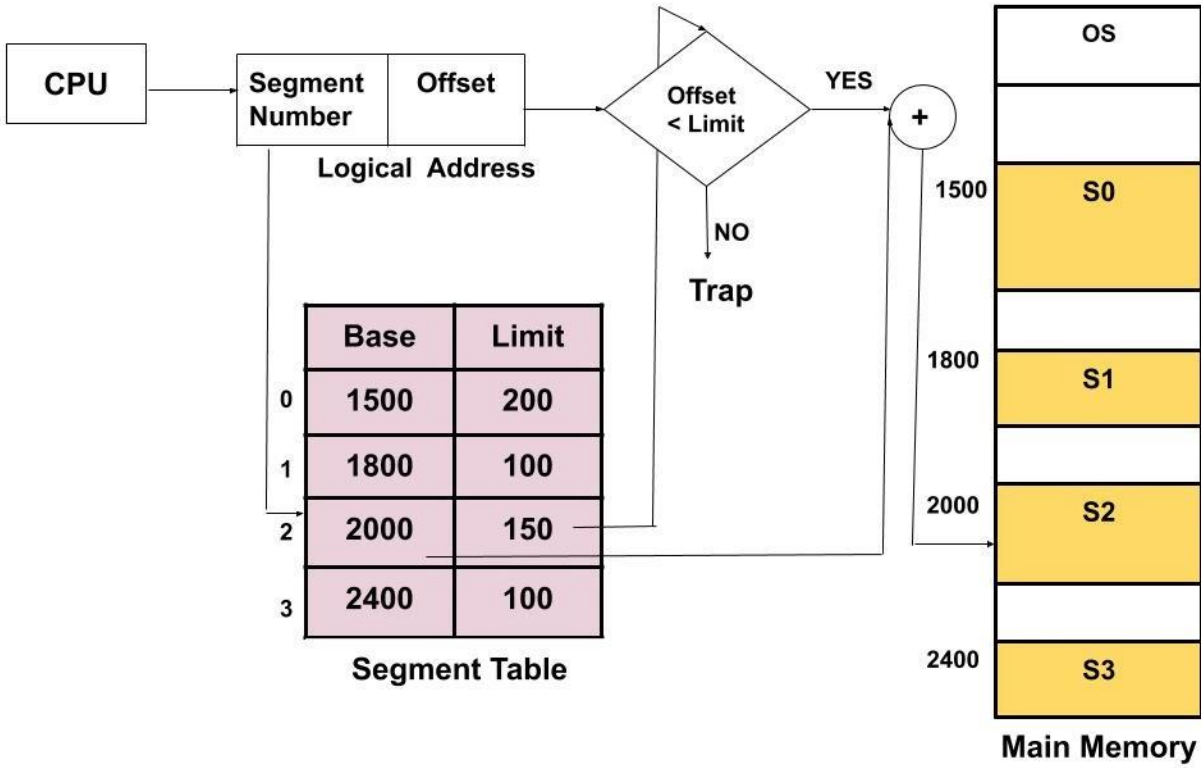
2. Segment Offset: It tells the exact word in that segment which the CPU wants to read.

Logical Address = Segment Number + Segment Offset

- Physical Address: The physical address is obtained by adding the base address of the segment to the segment offset.
- Segment table: A segment table stores the base address of each segment in the main memory. It has two parts i.e. Base and Limit. Here, base indicates the base address or starting address of the segment in the main memory. Limit tells the size of that segment. A register called Segment Table Base Register(STBR) which holds the base value of the segment table. The segment table is also stored in the main memory itself.

How is the translation done?

The CPU generates the logical address which contains the segment number and the segment offset. STBR register contains the address of the segment table. Now, the segment table helps in determining the base address of the segment corresponding to the page number. Now, the segment offset is compared with the limit corresponding to the Base. If the segment offset is greater than the limit then it is an invalid address. This is because the CPU is trying to access a word in the segment and this value is greater than the size of the segment itself which is not possible. If the segment offset is less than or equal to the limit then only the request is accepted. The physical address is generated by adding the base address of the segment to the segment offset.



Segmentation

Advantages of Segmentation

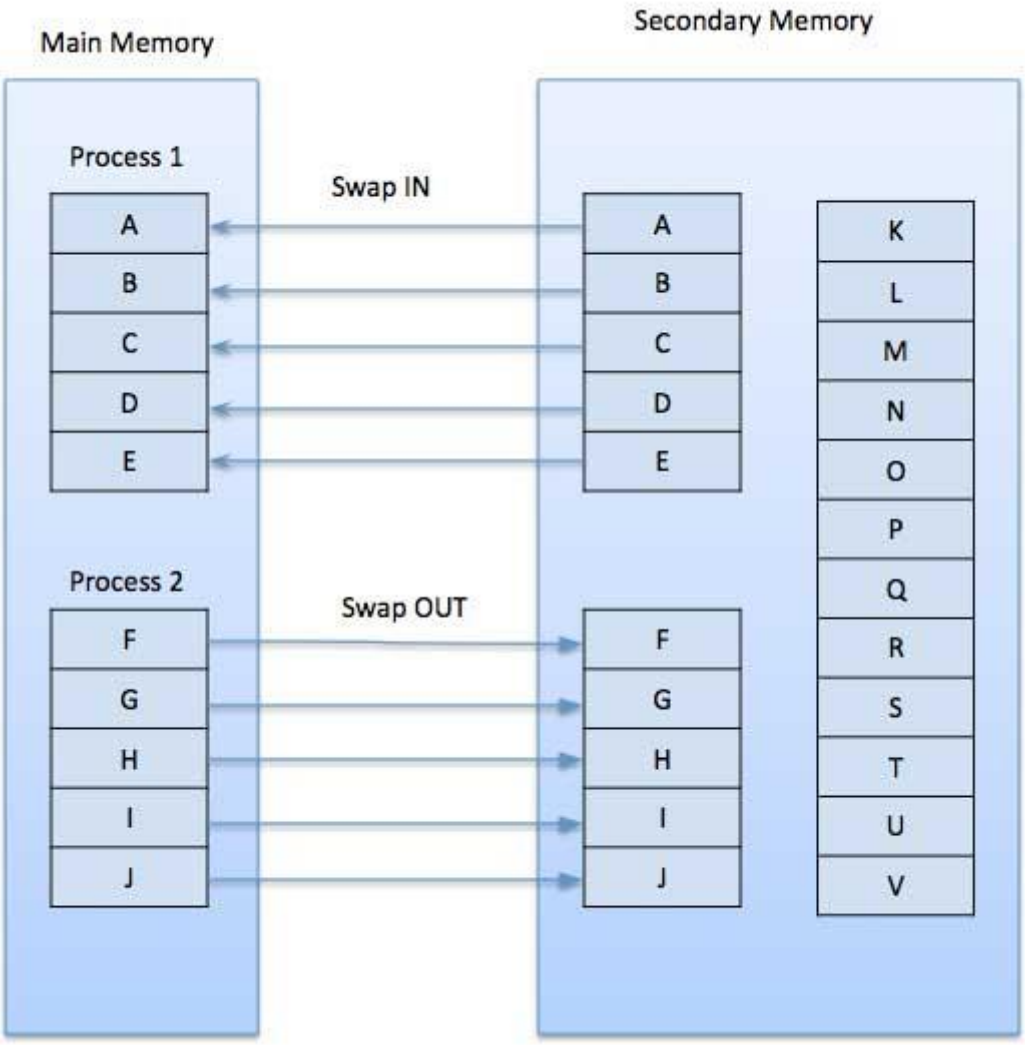
1. The size of the segment table is less compared to the size of the page table.
2. There is no internal fragmentation.

Disadvantages of Segmentation

1. When the processes are loaded and removed (during swapping) from the main memory then free memory spaces are broken into smaller pieces and this causes external fragmentation.
2. Here also the time to access the data increases as due to segmentation the main memory has to be now accessed two times. First, we need to access the segment table which is also stored in the main memory and second, combine the base address of the segment with the segment offset and then get the physical address which is again stored in the main memory.

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program’s pages out to the disk or any of the new program’s pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program’s pages as they are referenced.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

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.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults,

Reference String

The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data, where we note two things.

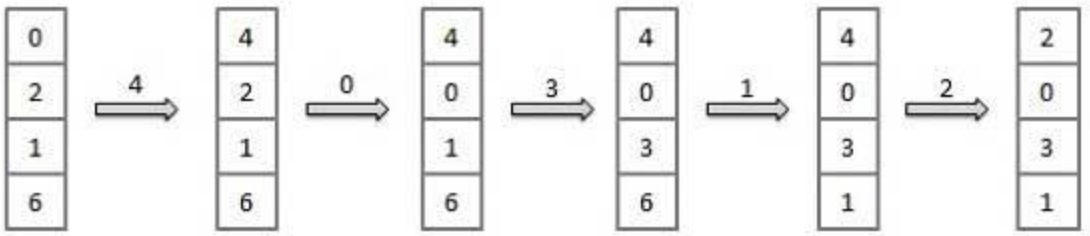
- For a given page size, we need to consider only the page number, not the entire address.
- If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.
- For example, consider the following sequence of addresses – 123,215,600,1234,76,96
- If page size is 100, then the reference string is 1,2,6,12,0,0

First In First Out (FIFO) algorithm

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x x



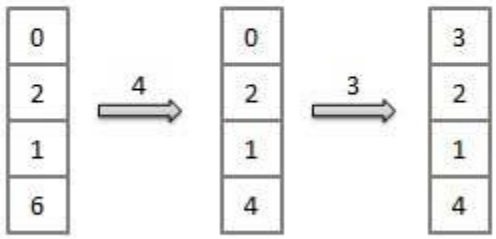
Fault Rate = 9 / 12 = 0.75

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.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x



Fault Rate = 6 / 12 = 0.50

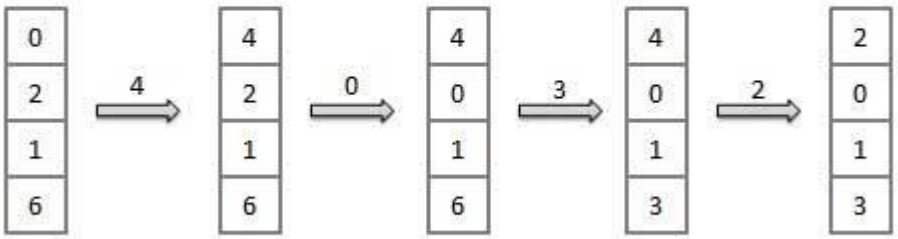
Least Recently Used (LRU) algorithm

- 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.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x



Fault Rate = 8 / 12 = 0.67

Page Buffering algorithm

- To get a process start quickly, keep a pool of free frames.
- On page fault, select a page to be replaced.
- Write the new page in the frame of free pool, mark the page table and restart the process.
- Now write the dirty page out of disk and place the frame holding replaced page in free pool.

Least frequently Used(LFU) algorithm

- The page with the smallest count is the one which will be selected for replacement.
- This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

Most frequently Used(MFU) algorithm

- This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.