Operating System

MODULE-4: Memory Management

Compiled by: Prof. Snehal Andhare

snehal.andhare@vit.edu.in

Vidyalankar Institute of Technology
Wadala (E), Mumbai
www.vit.edu.in

Vidyalankar Institute of Technology

Certificate

This is to certify that the e-book titled "OPERATING SYSTEM" comprises all elementary learning tools for a better understating of the relevant concepts. This e-book is comprehensively compiled as per the predefined eight parameters and guidelines.

Signature

Prof. Snehal Andhare

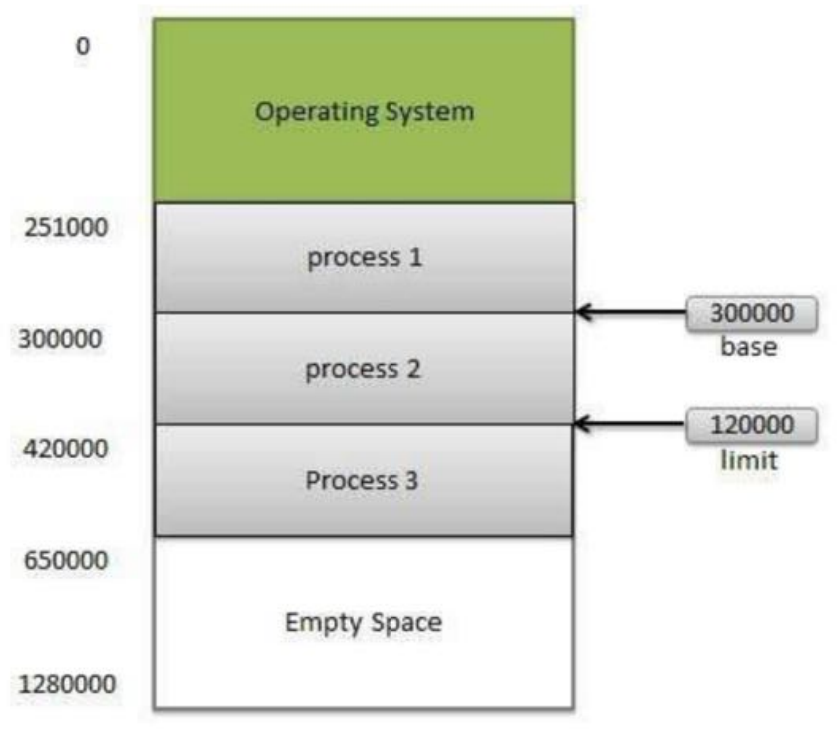
Assistant Professor

Department of Computer Engineering

Date: 12-10-2015

Memory Management Strategies:

Memory management is the functionality of an operating system which handles or manages primary memory. Memory management keeps track of each and every memory location either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.
Memory management provides protection by using two registers, a base register and a limit register. The base register holds the smallest legal physical memory address and the limit register specifies the size of the range. For example, if the base register holds 300000 and the limit register is 1209000, then the program can legally access all addresses from 300000 through 411999.



Instructions and data to memory addresses can be done in following ways
Compile time -- When it is known at compile time where the process will reside, compile time binding is used to generate the absolute code.
Load time -- When it is not known at compile time where the process will reside in memory, then the compiler generates re-locatable code.
Execution time -- If the process can be moved during its execution from one memory segment to another, then binding must be delayed to be done at run time

Dynamic Loading

In dynamic loading, a routine of a program is not loaded until it is called by the program. All routines are kept on disk in a re-locatable load format. The main program is loaded into memory and is executed. Other routines methods or modules are loaded on request. Dynamic loading makes better memory space utilization and unused routines are never loaded.

Dynamic Linking

Linking is the process of collecting and combining various modules of code and data into a executable file that can be loaded into memory and executed. Operating system can link system level libraries to a program. When it combines the libraries at load time, the linking is called static linking and when this linking is done at the time of execution, it is called as dynamic linking.

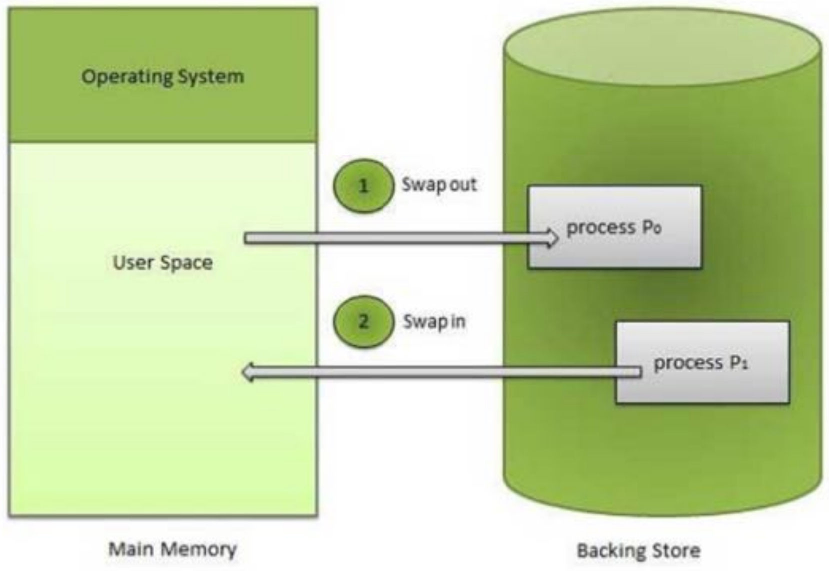
In static linking, libraries linked at compile time, so program code size becomes bigger whereas in dynamic linking libraries linked at execution time so program code size remains smaller.

Logical versus Physical Address Space

An address generated by the CPU is a logical address whereas address actually available on memory unit is a physical address. Logical address is also known a Virtual address.
Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme. The set of all logical addresses generated by a program 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 run-time mapping from virtual to physical address is done by the memory management unit *MMU* which is a hardware device. MMU uses following mechanism to convert virtual address to physical address.
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 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.
The user program deals with virtual addresses; it never sees the real physical addresses.

Swapping

Swapping is a mechanism in which a process can be swapped temporarily out of main memory to a backing store , and then brought back into memory for continued execution. Backing store is a usually a hard disk drive or any other secondary storage which fast in access and large enough to accommodate copies of all memory images for all users. It must be capable of providing direct access to these memory images. Major time consuming part of swapping is transfer time. Total transfer time is directly proportional to the amount of memory swapped. Let us assume that the user process is of size 100KB and the backing store is a standard hard disk with transfer rate of 1 MB per second. The actual transfer of the 100K process to or from memory will take 100KB / 1000KB per second
= 1/10 second
= 100 milliseconds



Memory Allocation

Main memory usually has two partitions

Low Memory -- Operating system resides in this memory.
High Memory -- User processes then held in high memory.

Operating system uses the following memory allocation mechanism.

Sr.No.	Memory Allocation	Description
1	**Single-partition allocation**	In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.
2	**Multiple-partition allocation**	In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

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

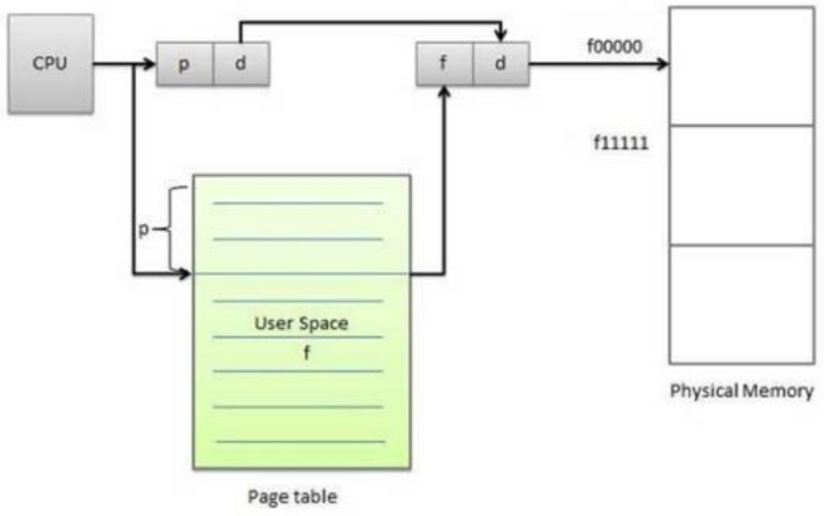
Sr.No.	Fragmentation	Description
1	**External fragmentation**	Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous so it cannot be used.
2	**Internal fragmentation**	Memory block assigned to process is bigger. Some portion of memory is left unused as it cannot be used by another process.

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.

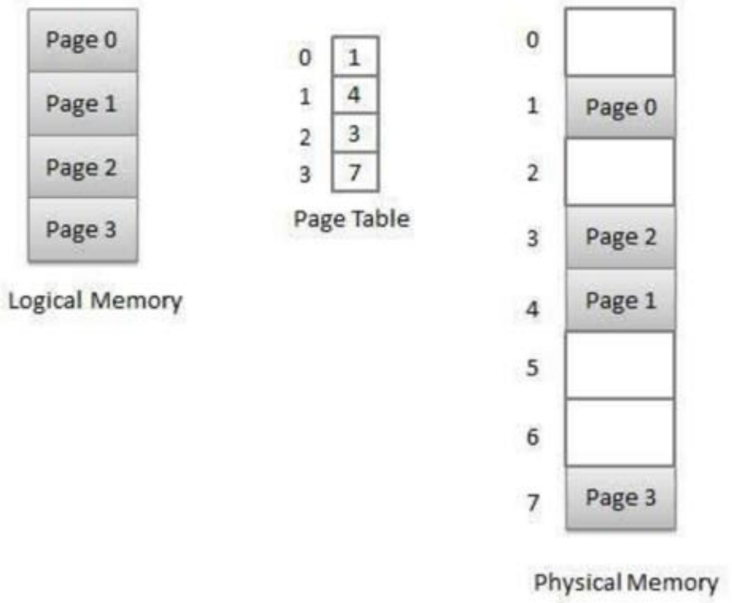
Paging

External fragmentation is avoided by using paging technique. Paging is a technique in which physical memory is broken into blocks of the same size called pages sizeispowerof2, between512bytesand8192bytes. When a process is to be executed, it's corresponding pages are loaded into any available memory frames.
Logical address space of a process can be non-contiguous and a process is allocated physical memory whenever the free memory frame is available. Operating system keeps track of all free frames. Operating system needs n free frames to run a program of size n pages.

Address generated by CPU is divided into
Page number *p* -- page number is used as an index into a page table which contains base address of each page in physical memory.
Page offset *d* -- page offset is combined with base address to define the physical memory address.



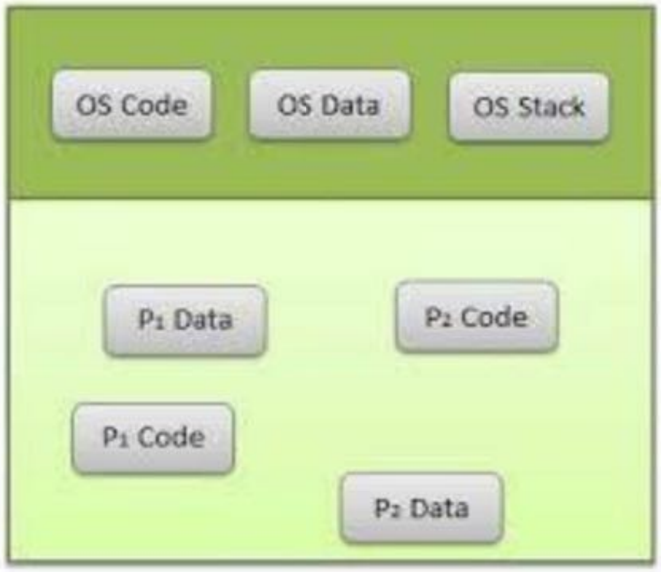
Following figure show the paging table architecture.



Segmentation

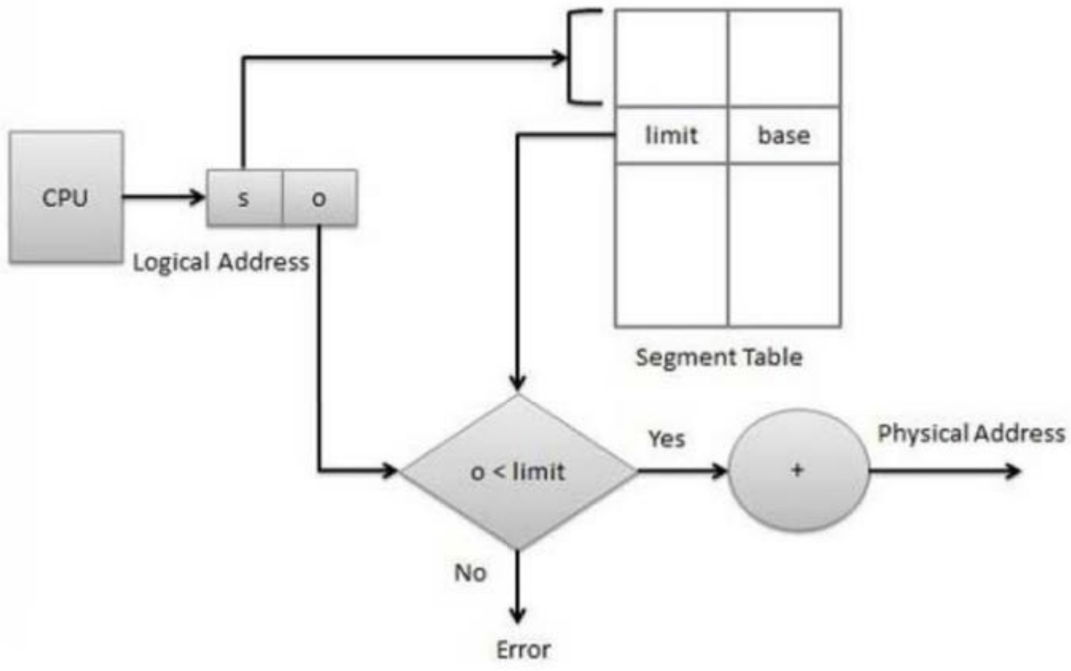
Segmentation is a technique to break memory into logical pieces where each piece represents a group of related information. For example ,data segments or code segment for each process, data segment for operating system and so on. Segmentation can be implemented using or without using paging.

Unlike paging, segment are having varying sizes and thus eliminates internal fragmentation.
External fragmentation still exists but to lesser extent.



Logical Address Space

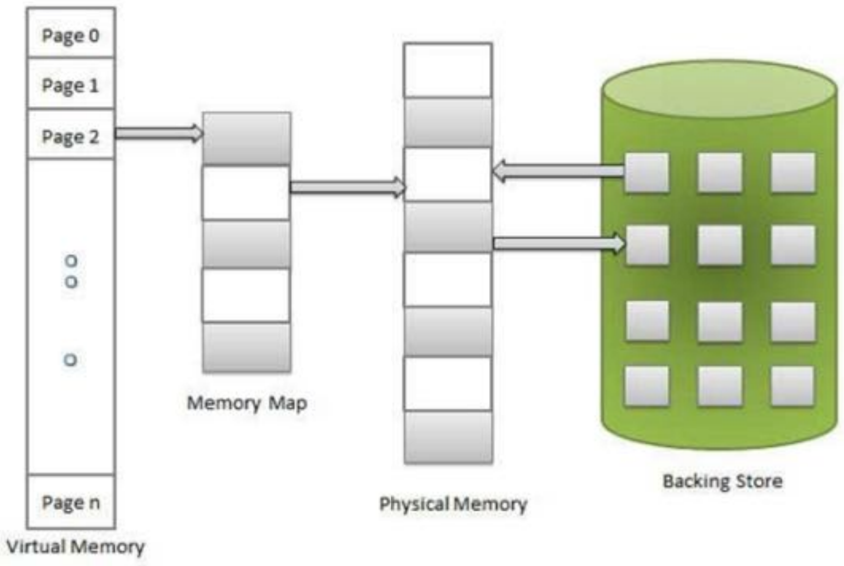
Address generated by CPU is divided into
Segment number *s* -- segment number is used as an index into a segment table which contains base address of each segment in physical memory and a limit of segment.
Segment offset *o* -- segment offset is first checked against limit and then is combined with base address to define the physical memory address.



Virtual Memory Management:

Virtual memory is a technique that allows the execution of processes which are not completely available in memory. The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory is the separation of user logical memory from physical memory.
This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available. Following are the situations, when entire program is not required to be loaded fully in main memory.

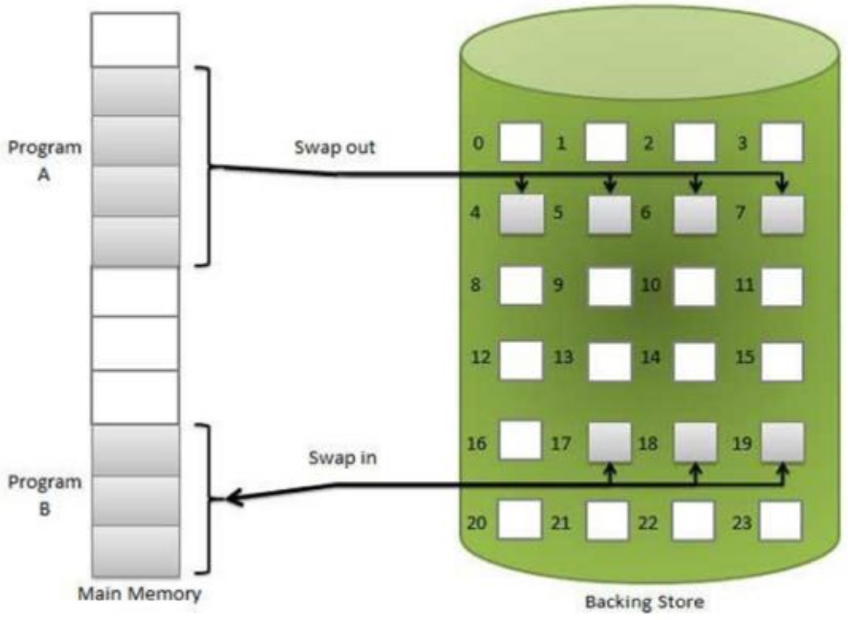
- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability 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.



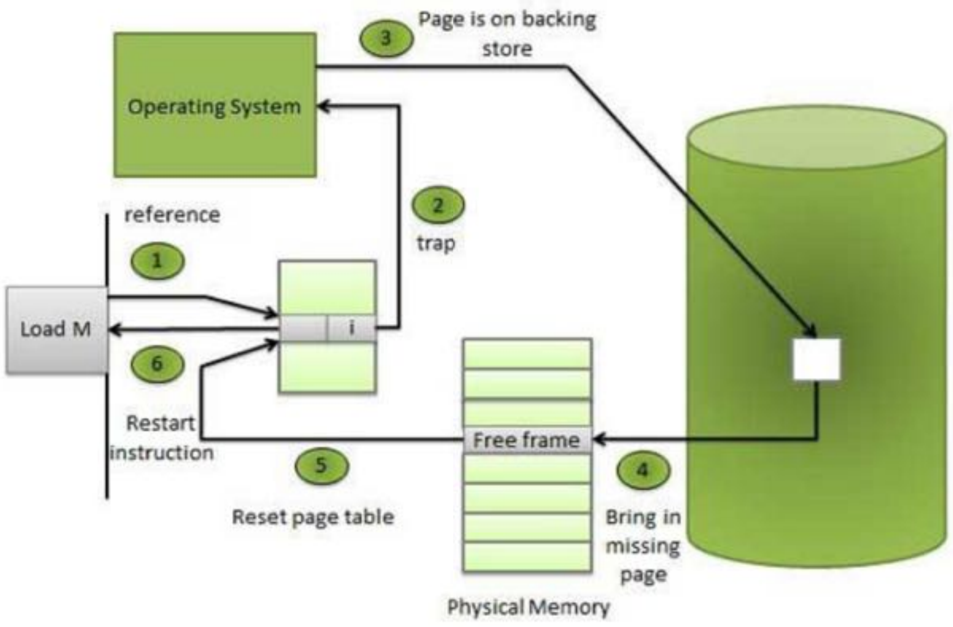
Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Demand Paging

A demand paging system is quite similar to a paging system with swapping. When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however, we use a lazy swapper called pager.
When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again. Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed. Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the valid-invalid bit scheme. Where valid and invalid pages can be checked by checking the bit. Marking a page will have no effect if the process never attempts to access the page. While the process executes and accesses pages that are memory resident, execution proceeds normally.



Access to a page marked invalid causes a **page-fault trap**. This trap is the result of the operating system's failure to bring the desired page into memory. But page fault can be handled as following



Step	Description
Step 1	Check an internal table for this process, to determine whether the reference was a valid or it was an invalid memory access.
Step 2	If the reference was invalid, terminate the process. If it was valid, but page have not yet

	brought in, page in the latter.
Step 3	Find a free frame.
Step 4	Schedule a disk operation to read the desired page into the newly allocated frame.
Step 5	When the disk read is complete, modify the internal table kept with the process and the page table to indicate that the page is now in memory.
Step 6	Restart the instruction that was interrupted by the illegal address trap. The process can now access the page as though it had always been in memory. Therefore, the operating system reads the desired page into memory and restarts the process as though the page had always been in memory.

Advantages

Following are the advantages of Demand Paging

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

Disadvantages

Following are the disadvantages of Demand Paging

- Number of tables and amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.
- Due to the lack of an explicit constraint on jobs address space size.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which 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 then 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

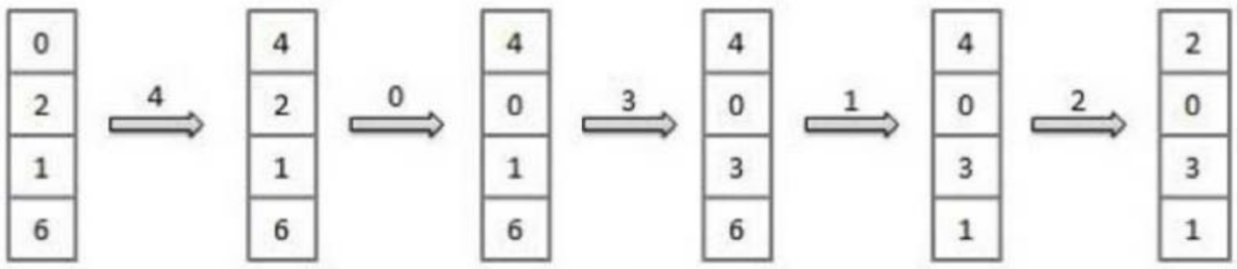
The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data, where we note two things. For a given page size we need to consider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



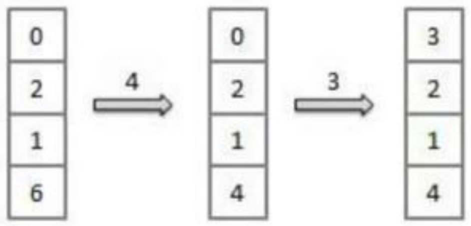
Fault Rate = 9 / 12 = 0.75

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

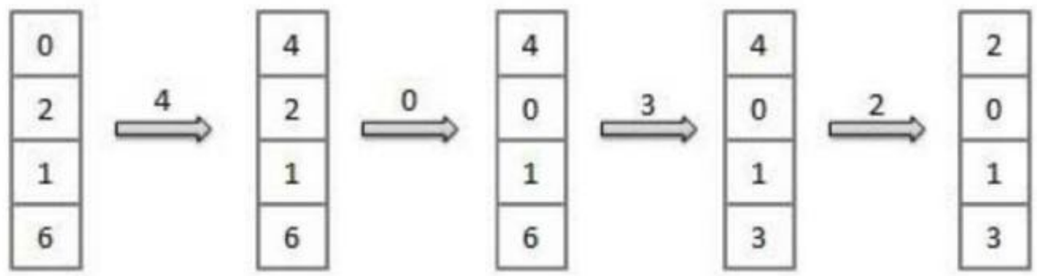
Least Recently Used *LRU* algorithm

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 process start quickly, keep a pool of free frames.
- On page fault, select a page to be replaced.
- Write 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

- 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.

References:

- https://en.wikipedia.org/wiki/Memory_management_(operating_systems)
- https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/8_MainMemory.html
- http://www.wiley.com/college/silberschatz6e/0471417432/slides/pdf2/mod9.2.pdf
- http://www.eecg.toronto.edu/~jacobsen/os/2007s/memory.pdf