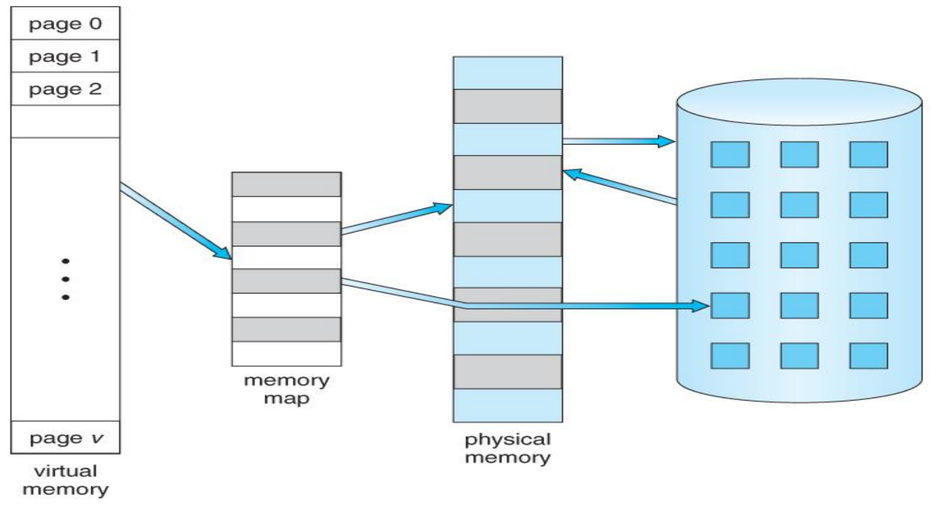
Virtual Memory

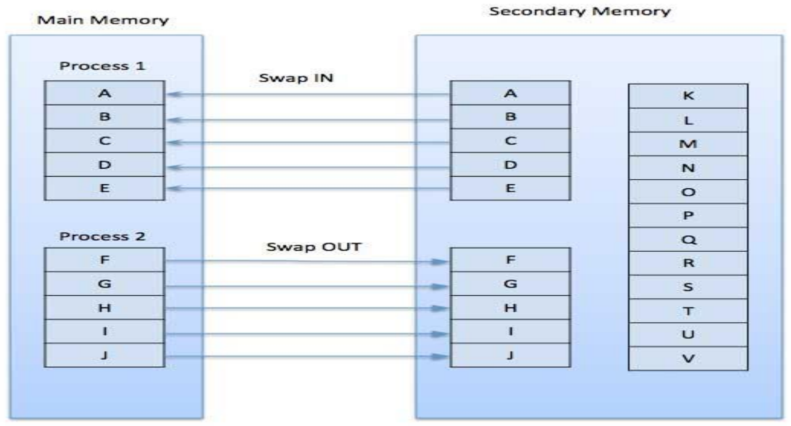
Computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.

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.



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.

Thus it avoids reading into memory pages that will not be used anyway , decreasing the swap time and the amount of physical memory needed.

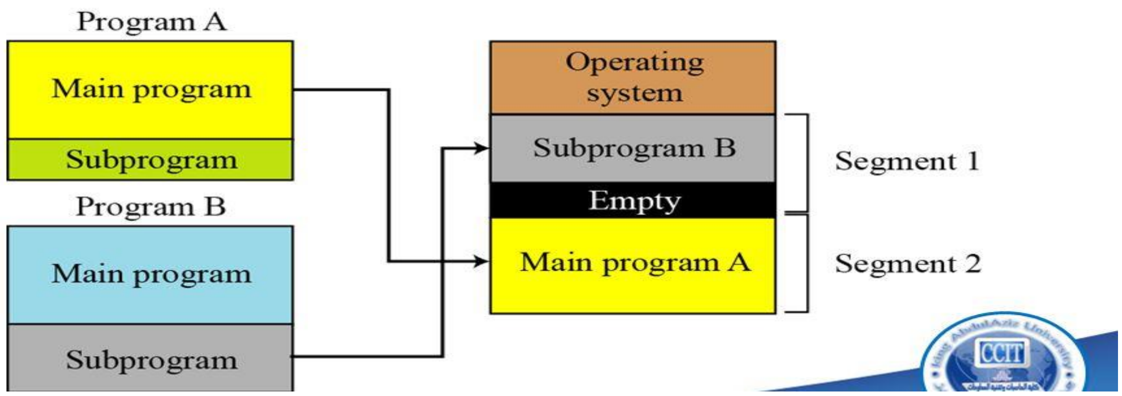
Advantages:

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

Demand Segmentation

•A technique similar to paging. In paging, a program is divided into equally sized pages. (not as programmer think)

•In demand segmentation, the program is divided into segments that match the programmer view (Modules).



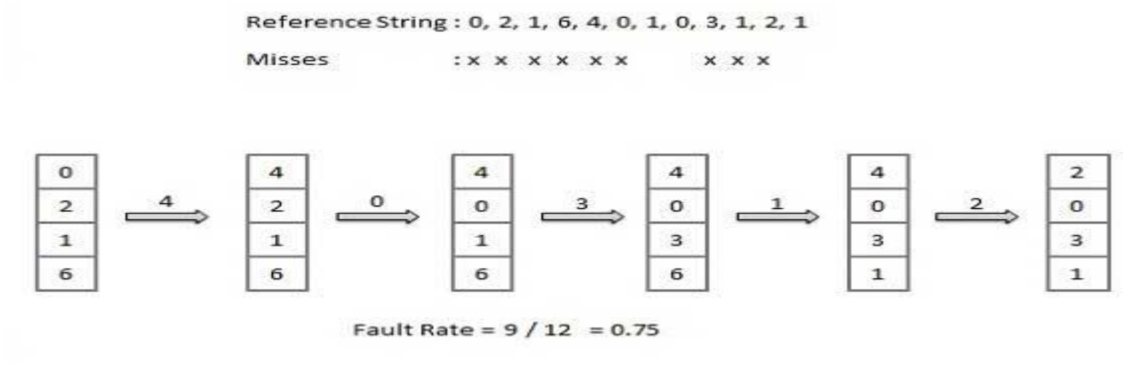
Operating system also uses demand segmentation, which is similar to demand paging.
It allocates memory to segments instead of pages.
The segment table has a valid bit to specify if the segment is already in physical memory or not.
If a segment is not in physical memory then segment fault results, which traps to the operating system and brings the needed segment into physical memory, mu ch like a page fault.
Demand segmentation allows for pages that are often referenced with each other to be brought into memory together, this decreases the number of page faults.
Another space server would be to keep some of a segment's page tables on disk and swap them into memory when needed.

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.
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.**

First In First Out (FIFO) algorithm

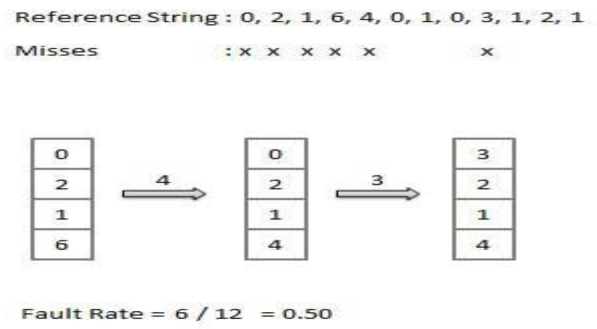
- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, **replace pages from the tail and add new pages at the head.**



Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1
Misses : x x x x x x x x x
0 2 1 6
4
4 2 1 6
0
4 0 1 6
3
4 0 3 6
1
4 0 3 1
2
2 0 3 1
Fault Rate = 9 / 12 = 0.75

Optimal Page algorithm

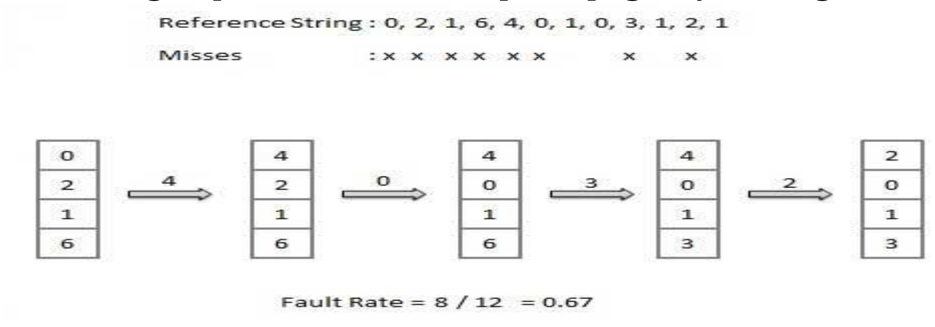
- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- **Replace the page that will not be used for the longest period of time.** Use the time when a page is to be used.



Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1
Misses : x x x x x x
0 2 1 6
4
0 2 1 4
3
3 2 1 4
Fault Rate = 6 / 12 = 0.50

Least Recently Used (LRU) algorithm

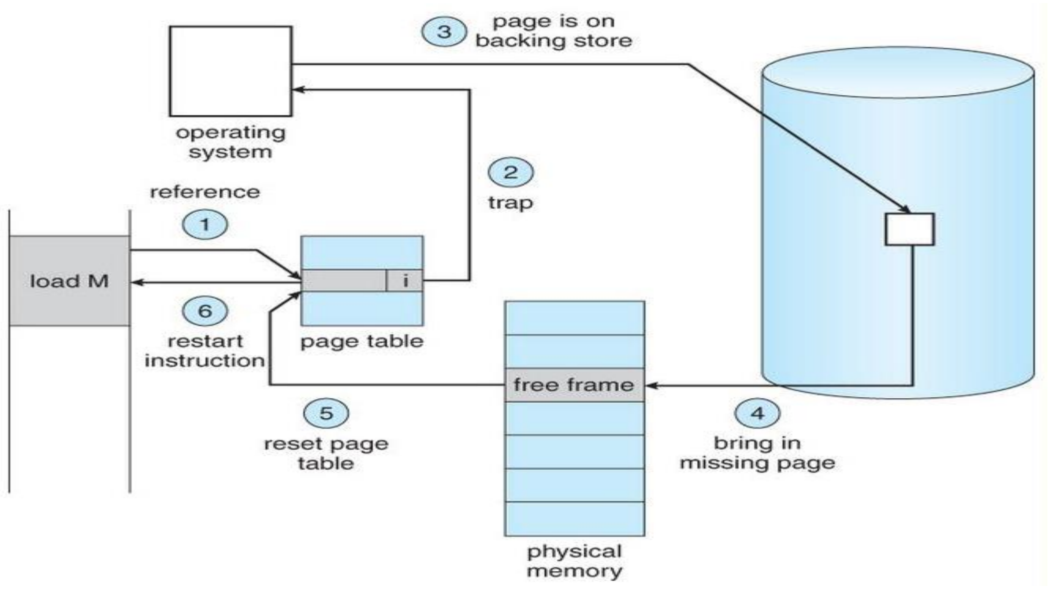
- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, **replace the page that has not been used for the longest period of time** or **replace pages by looking back into time.**



Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1
Misses : x x x x x x x x
0 2 1 6
4
4 2 1 6
0
4 0 1 6
3
4 0 1 3
2
2 0 1 3
Fault Rate = 8 / 12 = 0.67

Page Fault Handling

A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM. So when page fault occurs then following sequence of events happens :



- The computer hardware traps to the kernel and program counter (PC) is saved on the stack. Current instruction state information is saved in CPU registers.
- Operating system finds that a page fault has occurred and tries to find out which virtual page is needed. If not, the operating system must retrieve PC, fetch instruction and find out what it was doing when the fault occurred.
- Once virtual address caused page fault is known, system checks to see if address is valid and checks if there is no protection access problem.
- If the virtual address is valid, the system checks to see if a page frame is free. If no frames are free, the page replacement algorithm is run to remove a page.
- If frame selected is dirty, page is scheduled for transfer to disk, context switch takes place, fault process is suspended and another process is made to run until disk transfer is completed.
- As soon as page frame is clean, operating system looks up disk address where needed page is, schedules disk operation to bring it in.
- When disk interrupt indicates page has arrived, page tables are updated to reflect its position, and frame marked as being in normal state.
- Faulting instruction is backed up to state it had when it began and PC is reset. Faulting is scheduled, operating system returns to routine that called it.
- Reloads register and other state information, returns to user space to continue execution.

Allocation of Frames using Allocation Algorithms

There are two techniques for allocation of frames to user process.
Assume that there are m frames, *n* processes.

1. **Equal allocation**: give each process m/n frames. And the alternative is to recognize that various processes will need different amounts of memory

consider
1k frame size
62 free frames
Student process requiring: 10k
Interactive database requiring: 127k
it makes no sense to give each process as m/n frames the student process needs no more than 10 frames,
For example: so allocating 31 frames means, the additional 21 frames are wasted for student process.

Proportional allocation: when *m* frames available, in which we allocate available memory to each process according to its size.

- Let the size of virtual memory for process p_i is s_i, ai is allocation.
- S = Sigma s_i
- $a_i = (s_i/S) * m$

Example: if one process p1 needs 10 frames and another process p2 needs 127 frames and total number of available frames(m) is 62.

P1 process = (10/137)*62= gets 4 frames.
P2 process = (127/137)*62= gets 57 frames.

Global vs. Local Allocation

1) ***local page replacement*** - When process A page faults, consider only replacing pages allocated to process A. Processes are allocated a fixed fraction of the page frames.

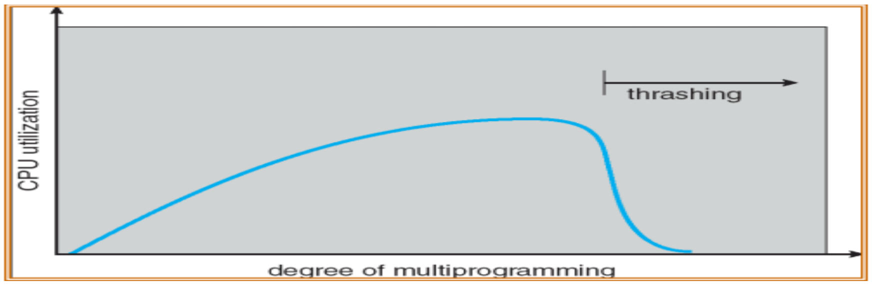
2) ***global-page replacement*** - When a page fault occurs, consider replacing pages from any process in memory. Page frames are dynamically allocated among processes, i.e., # of page frames of a process varies.

Thrashing

If it happens that your system has to swap pages at such a higher rate that major chunk of CPU time is spent in swapping **then this state is known as thrashing.**

- So effectively during thrashing, the CPU spends less time in some actual productive work and more time in swapping.
- Memory thrashing is a problem which arises when the memory is located more than the physical memory and it is not available.

Causes of Thrashing



1) If CPU utilization is too low then we increase the degree of multiprogramming by introducing a new process to the system. A global page replacement algorithm is used. The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming.
2) CPU utilization is plotted against the degree of multiprogramming.
3) As the degree of multiprogramming increases, CPU utilization also increases.
4) If the degree of multiprogramming is increased further, thrashing sets in and CPU utilization drops sharply.
5) So, at this point, to increase CPU utilization and to stop thrashing, we must decrease the degree of multiprogramming.