VIRTUAL MEMORY

It is a technique that allows the execution of processes that may not be completely in main memory.

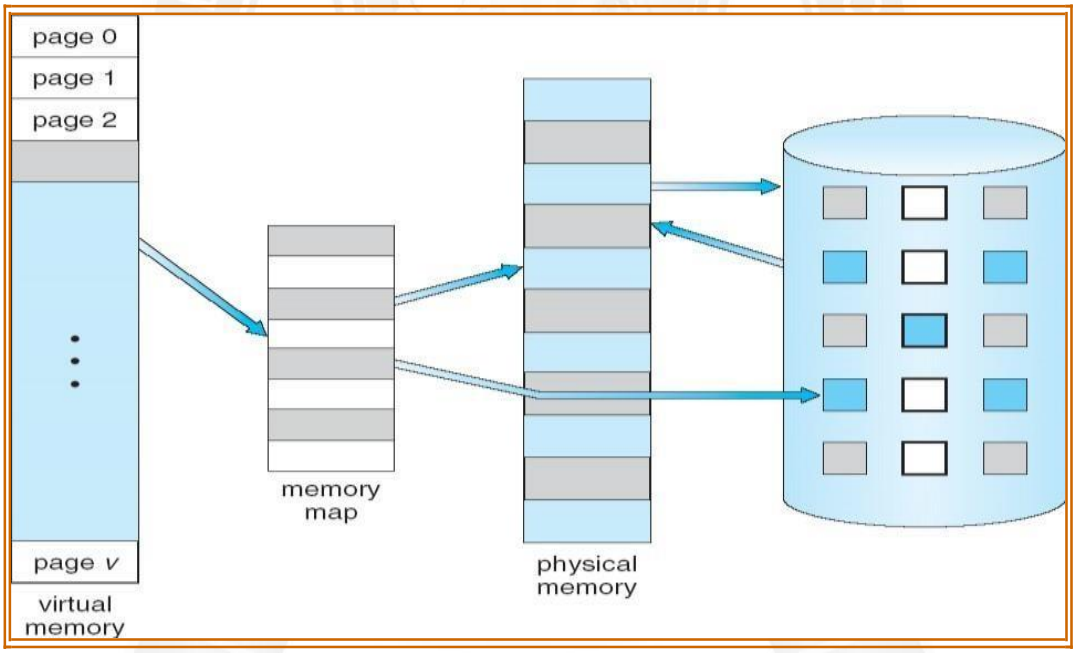
Advantages:

- Allows the program that can be larger than the physical memory.
- Separation of user logical memory from physical memory
- Allows processes to easily share files & address space.
- Allows for more efficient process creation.

Virtual memory can be implemented using

- Demand paging
- Demand segmentation

Virtual Memory that is Larger than Physical Memory



Demand Paging

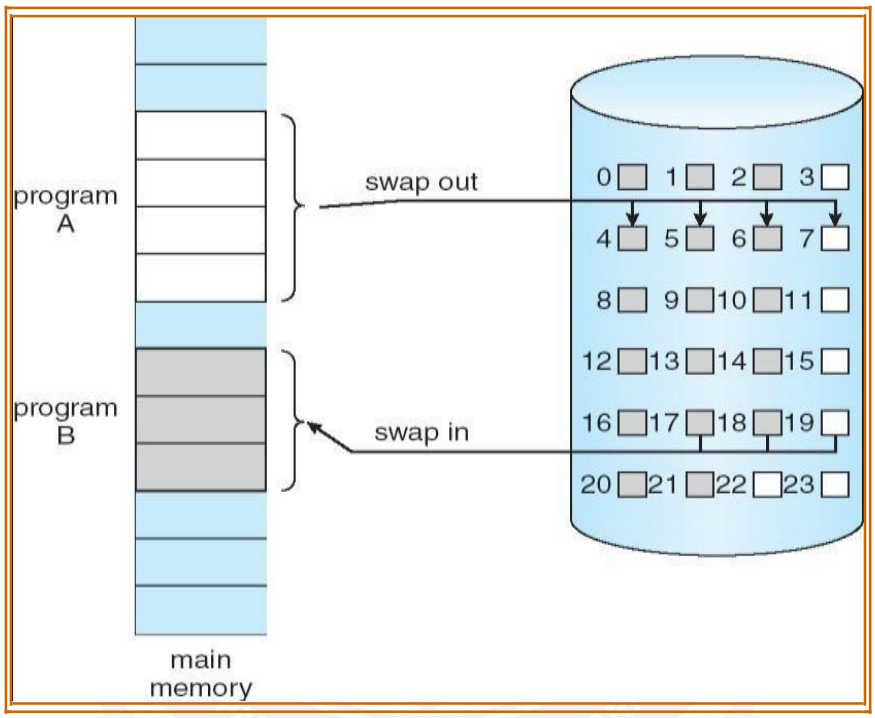
- It is similar to a paging system with swapping.
- Demand Paging - Bring a page into memory only when it is needed
- To execute a process, swap that entire process into memory. Rather than swapping the entire process into memory however, we use —Lazy Swapper

Lazy Swapper - Never swaps a page into memory unless that page will be needed.

Advantages

- Less I/O needed
- Less memory needed
- Faster response
- More users

Transfer of a paged memory to contiguous disk space



Basic Concepts:

Instead of swapping in the whole processes, the pager brings only those necessary pages into memory. Thus,

1. It avoids reading into memory pages that will not be used anyway.
2. Reduce the swap time.
3. Reduce the amount of physical memory needed.

To differentiate between those pages that are in memory & those that are on the disk we use the **Valid-Invalid bit**

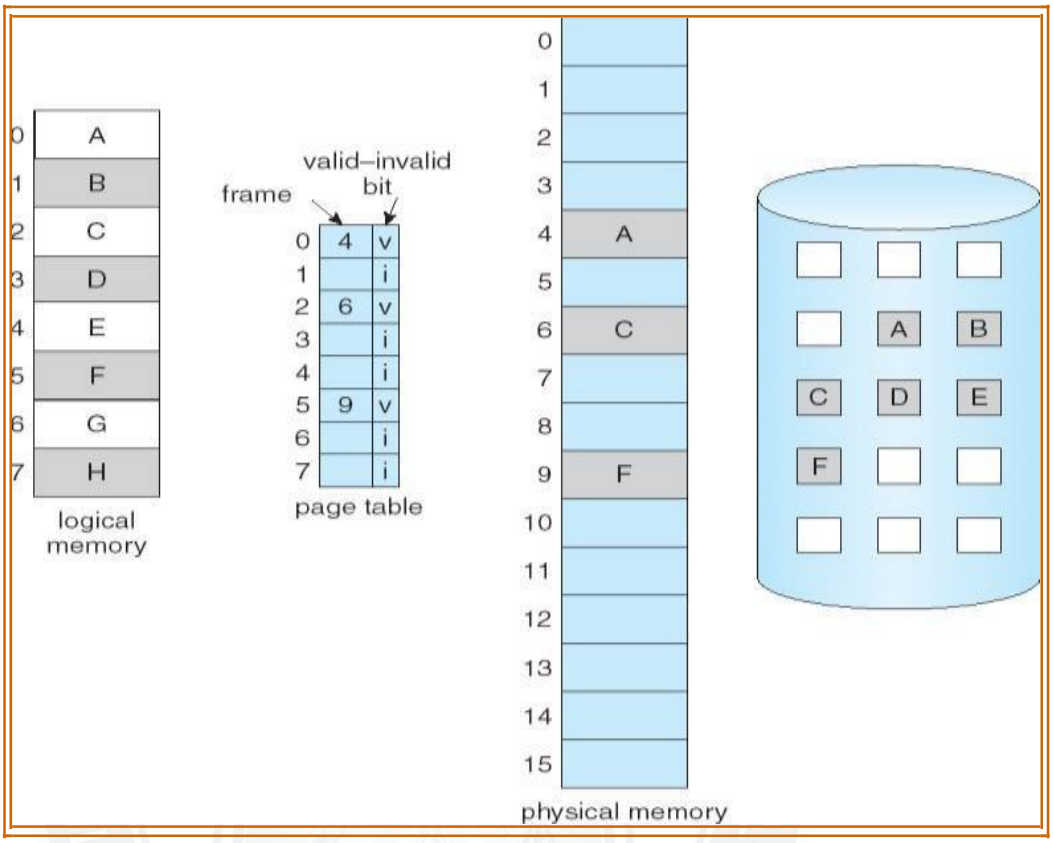
Valid-Invalid bit

A valid – invalid bit is associated with each page table entry.

Valid - > associated page is in memory.

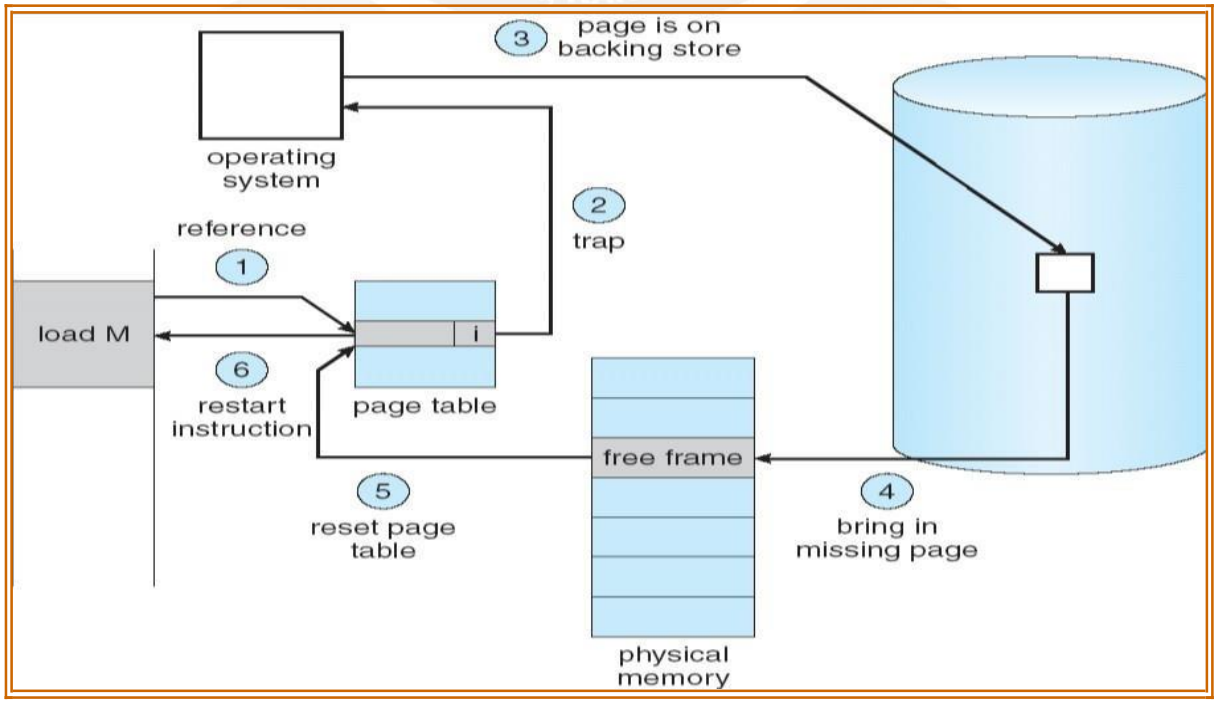
In-Valid -> invalid page, valid page but is currently on the disk

Page table when some pages are not in main memory



Page Fault

Access to a page marked invalid causes a page fault trap.



Steps in Handling a Page Fault

1. Determine whether the reference is a valid or invalid memory access
2. a) If the reference is invalid then terminate the process.

   b) If the reference is valid then the page has not been yet brought into main memory.
3. Find a free frame.
4. Read the desired page into the newly allocated frame.
5. Reset the page table to indicate that the page is now in memory.
6. Restart the instruction that was interrupted .

Pure demand paging

- Never bring a page into memory until it is required.
- We could start a process with no pages in memory**.**
- When the OS sets the instruction pointer to the 1st instruction of the process, which is on the non-memory resident page, then the process immediately faults for the page.
- After this page is bought into the memory, the process continue to execute, faulting as necessary until every page that it needs is in memory.

Performance of demand paging

Let p be the probability of a page fault 0 <*p* >1

Effective Access Time (EAT)

EAT = (1 – *p*) x ma + *p* x page fault time.

Where ma -> memory access, p ->Probability of page fault (0≤ p ≤ 1)

The memory access time denoted ma is in the range 10 to 200 ns.

If there are no page faults then EAT = ma.

To compute effective access time, we must know how much time is needed to service a page fault.

A page fault causes the following sequence to occur:

1. Trap to the OS
2. Save the user registers and process state.
3. Determine that the interrupt was a page fault.
4. Check whether the reference was legal and find the location of page on disk.
5. Read the page from disk to free frame.

a. Wait in a queue until read request is serviced.
b. Wait for seek time and latency time.
c. Transfer the page from disk to free frame.

6. While waiting ,allocate CPU to some other user.
7. Interrupt from disk.
8. Save registers and process state for other users.
9. Determine that the interrupt was from disk.
7. Reset the page table to indicate that the page is now in memory.
8. Wait for CPU to be allocated to this process again.
9. Restart the instruction that was interrupted .

Process Creation

Virtual memory enhances the performance of creating and running processes:

- Copy-on-Write
- Memory-Mapped Files

a) Copy-on-Write

fork()

Creates a child process as a duplicate of the parent process & it worked by creating copy of the parent address space for child, duplicating the pages belonging to the parent.

Copy-on-Write (COW)

Allows both parent and child processes to initially *share* the same pages in memory. These shared pages are marked as Copy-on- Write pages, meaning that if either process modifies a shared page, a copy of the shared page is created.

vfork():

With this the parent process is suspended & the child process uses the address space of the parent.

- Because vfork() does not use Copy-on-Write, if the child process changes any pages of the parent‘s address space, the altered pages will be visible to the parent once it resumes.
- Therefore, vfork() must be used with caution, ensuring that the child process does not modify the address space of the parent.

(b) Memory – mapped files:

Sequential read of a file on disk uses open() , read() and write()

Every time a file is accessed it requires a system call and disk access.

Alternative method: **"Memory – mapped files"**

Allowing a part of virtual address space to be logically associated with file

Mapping a disk block to a page in memory.