UNIT – III

Memory Management Strategies

Objectives:

- Students will be able To develop the concepts of memory management techniques

Syllabus:

Memory Management Strategies

- Swapping, contiguous memory allocation (memory mapping and protection, memory allocation, fragmentation), paging (basic method, hardware support, shared pages), Segmentation (basic method, Hardware).

Virtual-Memory Management:

- Demand paging (Basic concepts, Performance of Demand Paging), page replacement (FIFO, Optimal, LRU), Allocation of frames (Minimum number of frames, Allocation Algorithms), Thrashing (Cause of Thrashing, Working-Set model, Page fault frequency).

Subject Outcomes:

Students will be able to

- Describe the benefits of a virtual memory system
- Explain the concepts of demand paging, page-replacement algorithms, and allocation of page frames
- Discuss the principle of the working-set model
- Describe various ways of organizing memory hardware
- Discuss various memory-management techniques, including paging and segmentation
- Describe both pure segmentation and segmentation with paging

Learning Material

3.1. Swapping

- A process must be in memory to be executed.
- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
- **For example:**
    - Consider a **round –robin** CPU scheduling algorithm, when time quantum expires , the memory manager will start to swap out the process that just finished and to swap another process into memory space that has been freed.
    - Swapping policy is used for **priority based** scheduling algorithms. If a higher priority process arrives and wants service, the memory manager can swap out the lower priority processes and then load and execute the higher priority process.
- **Roll out–**Swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- **Roll in:** Swapping in the higher priority process is known as roll-in.
- Swapping requires a backing store.
- The backing store is commonly a fast disk.
- It must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images.
- When the CPU scheduler decides to execute a process, it calls the dispatcher.
- The dispatcher checks whether the next process in the queue is in memory.
- If not, and if there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process.

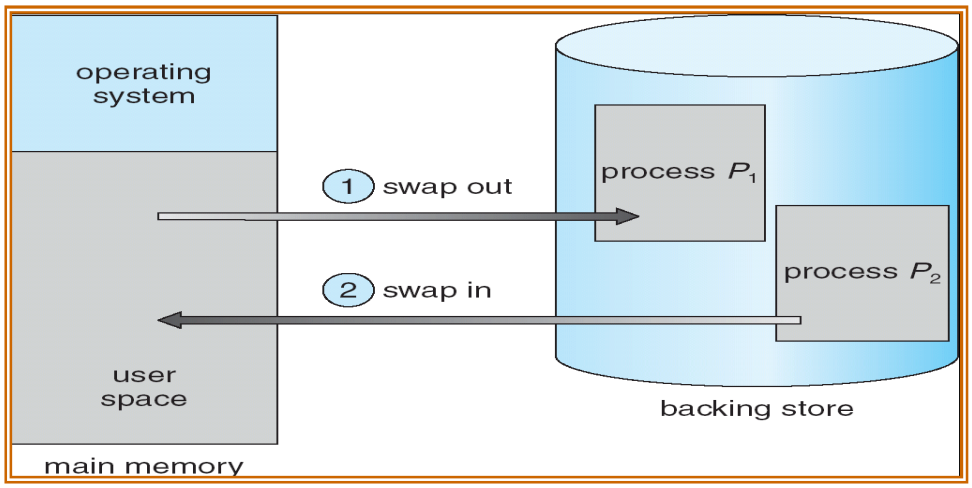


Fig: Swapping of two processes using a disk as a backing store

- The context-switch time in such a swapping system is fairly high.
- **For example**
  - If the user process is 10 MB in size and the backing store is a standard hard disk with a transfer rate of 40 MB per second.
  - The actual transfer of the 10-MB process to or from main memory takes

    10000 KB/40000 KB per second = 1/4 second

    = 250 milliseconds.

3.2. Contiguous Memory Allocation:

- The main memory must accommodate both the operating system and the various user processes.
- So the main memory must be allocated in an efficient way possible
- The memory is usually divided into two partitions:

  1) Operating system

  2) User processes.
- We can place the operating system in either low memory or high memory.
- The major factor affecting this decision is the location of the interrupt vector. Since the interrupt vector is often in low memory, so programmers place the operating system in low memory.

- We have to allocate all available memory to the processes.
- In contiguous memory allocation, each process is contained in a single contiguous section of memory.

3.2.1. Memory mapping and Protection:

- We can provide memory mapping and protection by using a relocation register and a limit register.
- The relocation register contains the value of the smallest physical address;
- the limit register contains the range of logical addresses
- For example, relocation = 100040 and limit = 74600.
- With relocation and limit registers, each logical address must be less than the limit register;
- The MMU maps the logical address *dynamically* by adding the value in the relocation register.

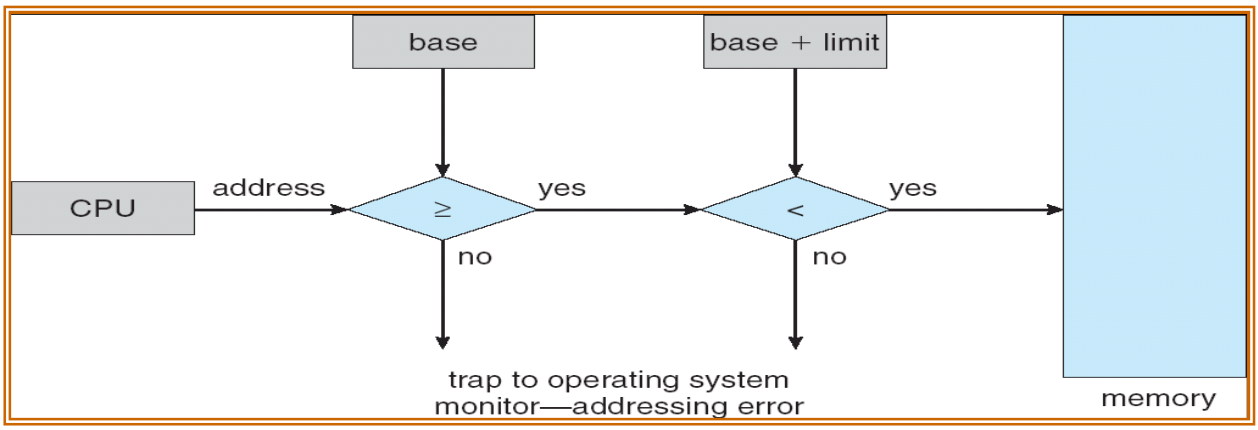


Fig: Hardware support for relocation and limit register

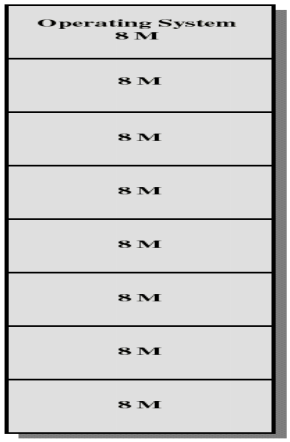
- When the CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context switch.
- Because every address generated by a CPU is checked against these registers.
- We can protect both the operating system and the other users' programs and data being modified by the running process.

- The relocation-register scheme provides an effective way to allow the operating system size to change dynamically.

3.2.2. Memory Allocation:

3.2.2.1. Multiprogramming with Fixed number of Tasks: (Fixed Partitioning)

- OS occupies some fixed portion of main memory
- Rest of **main memory is divided into several fixed-number of partitions of equal size**
- Each partition may contain **exactly one process**
- Any **process whose size is <= to the partition size** can be loaded into any available partition
- If all the partitions are full – swapping is done
- Difficulties with equal – sized fixed partitions:
  - A program may be too big to fit into a partition
  - Example:
    1. Partition size – 8 M
       Program size – 20 M
       Main memory utilization is extremely inefficient
    2. Program size – 2 M
       Partition size – 8 M = 6M wasted



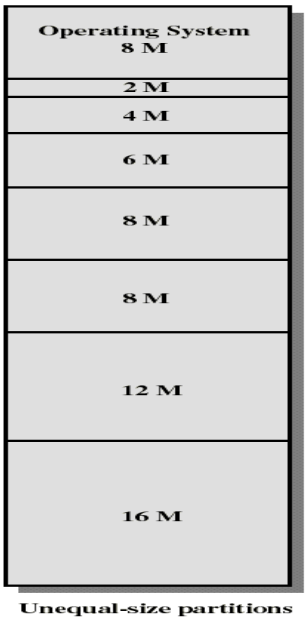
Equal-size partitions

- **Drawback: Internal fragmentation:**
  - Amount of space wasted inside a partition allocated to a process is called as Internal Fragmentation.
  - allocated memory may be slightly larger than requested memory
  - Example:
  - Program size –> 2 M

    Partition size –> 8 M

    = 6M Internal fragmentation

- **MFT – Unequal size partitions**
  - **Example:**

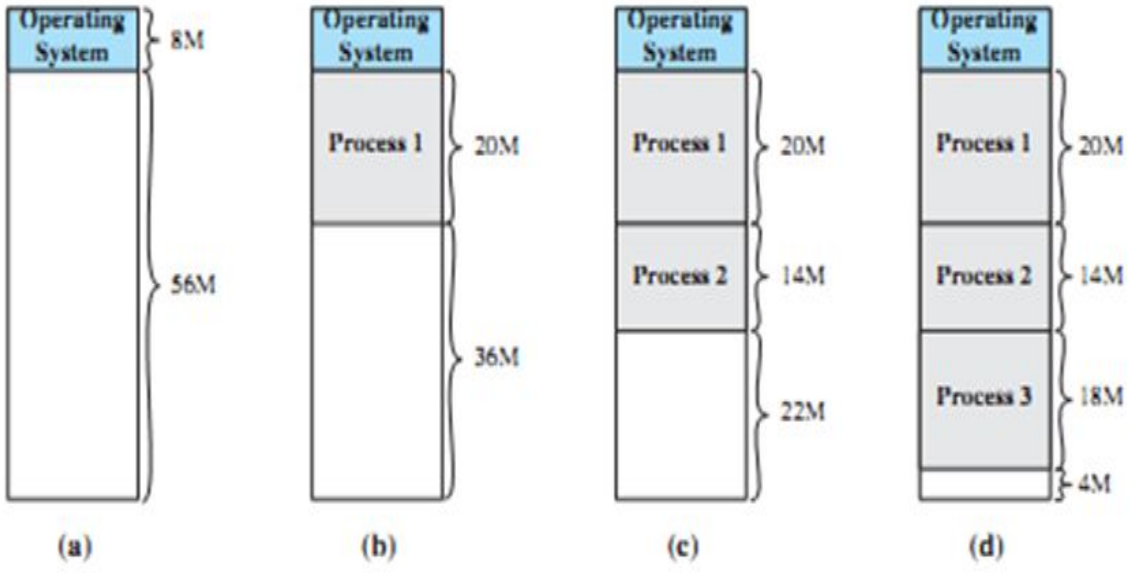


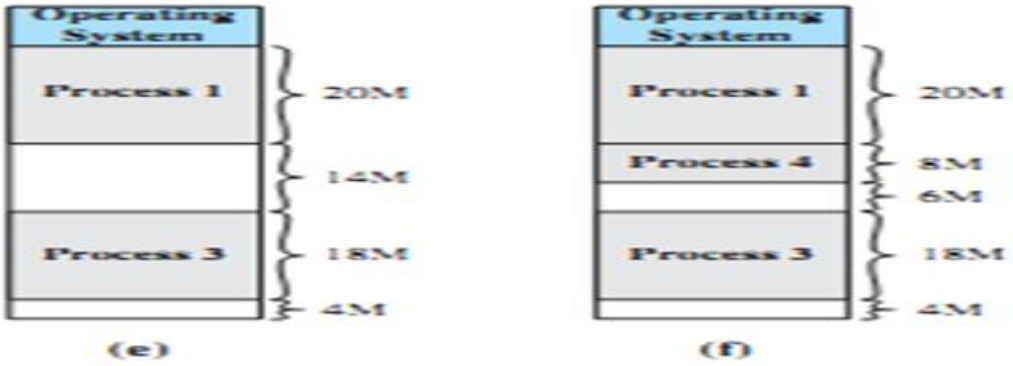
Unequal-size partitions

- **Advantages:**
  - Simple to implement
  - Little OS overhead
- **Disadvantages:**
  - Internal Fragmentation
  - Maximum number of active processes is fixed
  - Degree of multiprogramming is fixed.

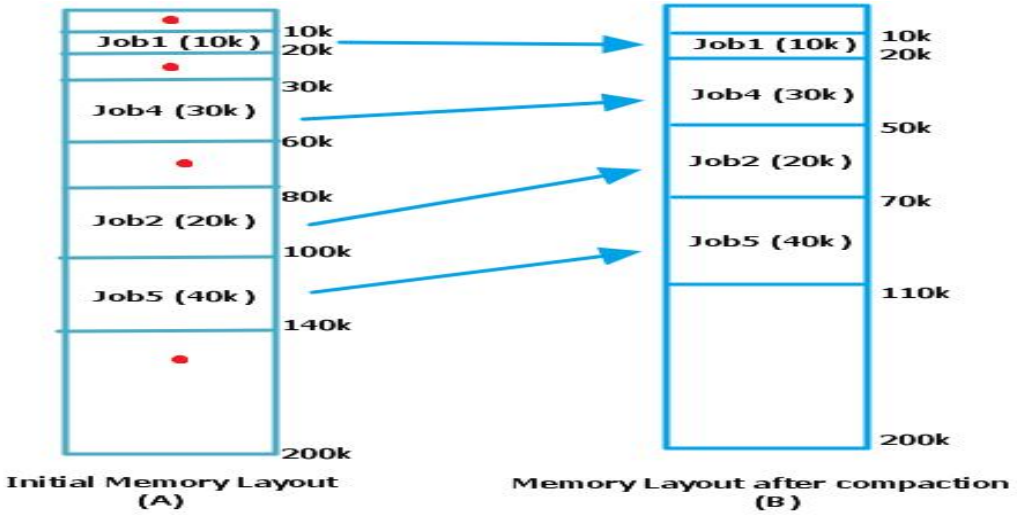
3.2.2.2. Multiprogramming with Variable number of Tasks (Dynamic partitioning):

- Partitions are of variable length and number
- Initially all memory available for user processes is considered as one large block of available memory (hole)
- When a process arrives and needs memory, search for a hole large enough for this process.
- If found – allocate only as much memory as is need keeping the rest available to satisfy future requests





- **Drawback: External Fragmentation**
  - Total memory space exists to satisfy a request, but it is not contiguous.
  - Storage space is fragmented into a large number of small holes
  - **Solution :**
    - Compaction: OS shifts the processes so that they are contiguous and so that all of the free memory is together in one block



- **Advantages:**
  - No Internal Fragmentation
  - More efficient use of main memory
- **Disadvantages:**
  - External Fragmentation

- Inefficient use of processor due to the need for compaction to counter external fragmentation.
- Compaction is very expensive scheme.

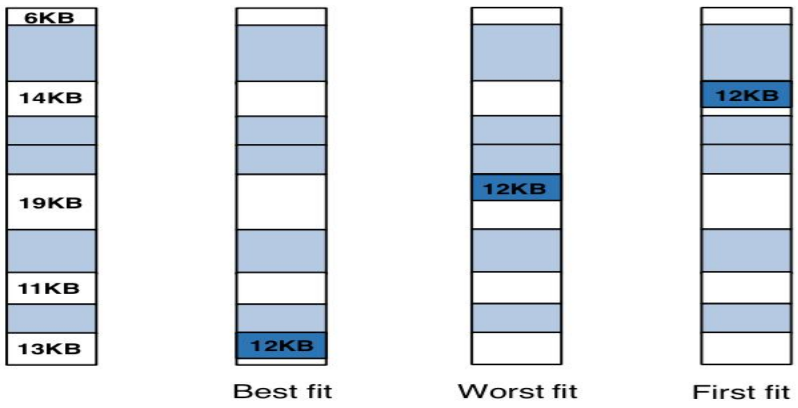
3.2.2.3. Dynamic Storage Allocation Problem:

- This problem deals with how to satisfy a request of size n from a list of free holes.
- This problem has many solutions and these strategies are used to select a free hole from a set of available holes.
  - First fit
  - Best fit
  - Worst fit

- **First fit:** Allocate the *first* hole that is big enough. Searching can start either at the beginning of the set of holes or where the previous first-fit ended. We can stop searching when we find a free hole that is large enough.
- **Best fit**. Allocate the *smallest* hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.
- **Worst fit:** Allocate the *largest* hole. We must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole.

Both first fit and best fit are better than worst fit in terms of decreasing time and storage utilization.

Example: New process size is 12 KB



- **50-percent rule**: If we are having *N* allocated blocks, another 0.5 *N* blocks will be lost to fragmentation. That is, one-third of memory may be unusable. This property is known as 50-percent rule.

3.3. Paging:

- Paging permits a program memory to be noncontiguous.
- Thus allowing a program to be allocated physical memory whenever it is available.
- Every address generated by the CPU is divided into two parts:
  1. Page number (p)
  2. Page offset (d).
- The page number is used as an index into a page table.
- The page table contains the base address of each page in physical memory.
- This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.

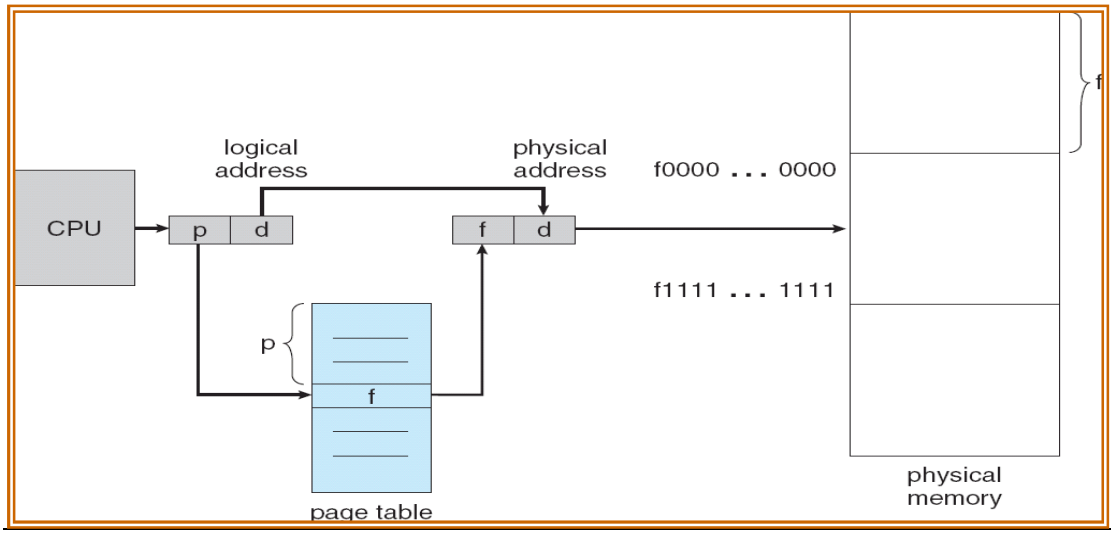


Fig: Paging Hardware

3.3.1. Basic method:

- Physical memory is broken into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages.
- When a program is to be executed, its pages are loaded into any available frames.

- The page table is defined to translate from user pages to memory frames (IBM 370 uses 2048 or 4096 bytes for page).

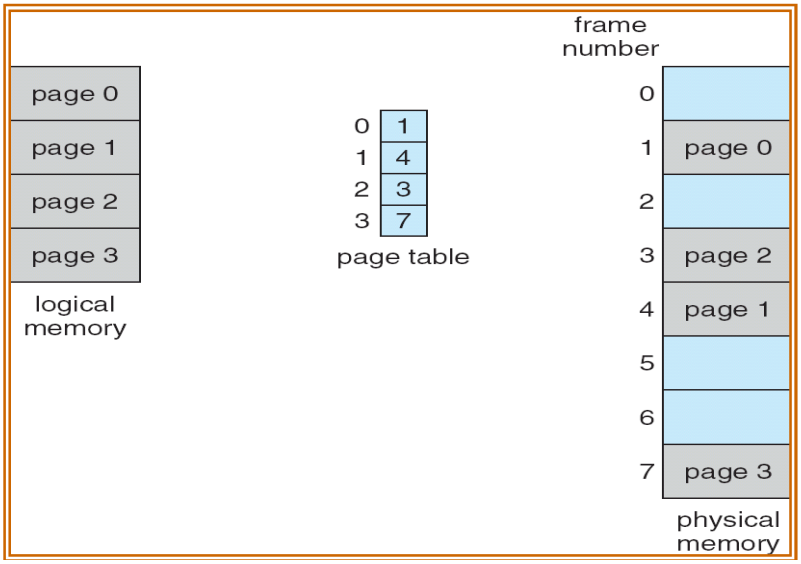


Fig: Paging model of logical and physical memory.

- If a page size is 2n addressing units (bytes or words) long, then the lower - order n bits of a logical addresses designates the page offset and the remaining higher order bits designates the paging number.

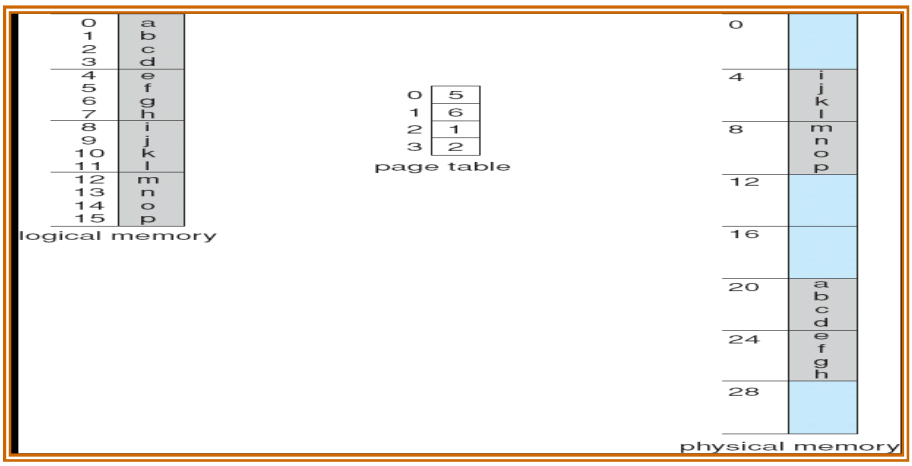


Fig: Paging example for a 32-byte memory with 4-byte pages.

- Logical address 0 is page 0, offset 0. Indexing into the page table, we find that page 0 is in frame 5. This logical address 0 maps to physical address 20 [= (5 x 4) + 0].
- Logical address 3 (page 0, offset 3) maps to physical address 23 [= (5 x 4) + 3].
- Every Logical address is mapped by the passing hardware to same physical address.
- physical address of word = (frame number x page size offset)

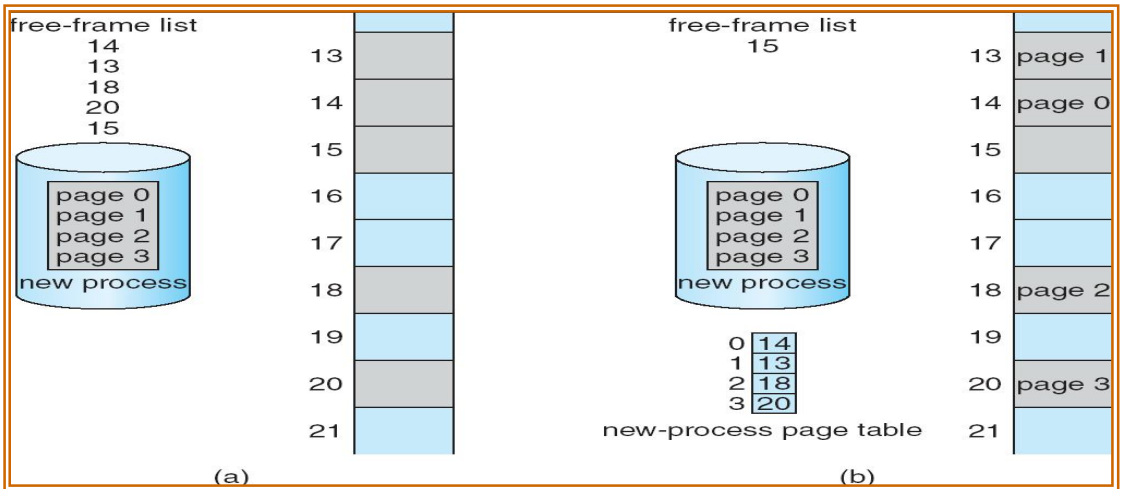


Fig: Free frames (a) before allocation and (b) after allocation.

3.3.2. Hardware support:

❖ **Implementation of page table:**

a. Registers
b. PTBR
c. TLB

a. Registers:

- Page table is implemented as a set of dedicated registers
- Very high speed- to make the translation efficient.
- Load and modification of these registers are controlled only by the operating system.
- Example: DEC PDP-11

- If page table is small-It is satisfactory.
- Most of the computers allow page table to be very large(1 Million entries).

b. PTBR

- Page table is placed in main memory
- PTBR points to the page table.
- Changes in page table-changing in PTBR register.
- **Drawback**: If we want to access location *i,* we must first index into the page table, using the value in the PTBR offset by the page number for 'i'.
- This task requires a memory access.
- It provides us with the frame number, which is combined with the page offset to produce the actual address.
- We can then access the desired place in memory.

With this scheme, *two* memory accesses are needed to access a byte

i. one for the page-table entry
ii. One for the byte).

- The memory access is slowed by a factor of 2. This delay would be intolerable under most circumstances.

c. The solution for this problem is **TLB** (Translation Look –aside Buffer),

- It is a special, small, fast lookup hardware cache.
- The TLB is used with page tables; it contains only a few of the page-table entries.
- When a logical address is generated by the CPU, its page number is presented to the TLB.

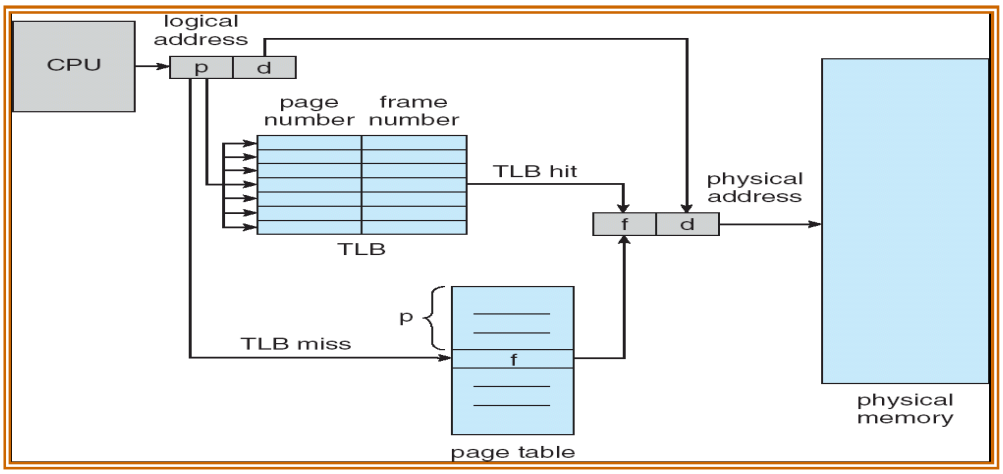


Fig: Paging hardware with TLB.

TLB Miss: If the page number is not in the TLB

TLB Hit: If the page number is in the TLB

Hit Ratio:

- The percentage of times that a particular page number is found in the TLB is called the hit ratio.
- **Example:** 80-percent hit ratio means that we find the desired page number in the TLB 80 percent of the time.
    - If it takes 20 nanoseconds to search the TLB and 100 nanoseconds to access memory, then the mapped-memory access takes 120 nanoseconds when the page number is in the TLB.
    - If we fail to find the page number in the TLB (20 nanoseconds), then we must first access memory for the page table and frame number (100 nanoseconds) and then access the desired byte in memory (100 nanoseconds), for a total of 220 nanoseconds.
    - To find the effective we weight the case by its probability:
    - Effective access time = 0.80 x 120 + 0.20 x 220

      = 140 nanoseconds.
    - In this example, we suffer a 40-percent slowdown in memory-access time (from 100 to 140 nanoseconds).

- For a 98-percent hit ratio,
  - effective access time = 0.98 x 120 + 0.02 x 220
    =122 nano seconds

3.3.3. Protection:

- Memory protection in a paged environment is accomplished by protection bits associated with each page.
- These bits are kept in the page table. One bit can define a page to be read/write or read-only.
- Every reference to memory goes through the page table to find the correct frame number.
- At the same time physical address is computed.
- The protection bits are checked to verify that no writes are being made on read-only page.
- An attempt is treated as a trap to the operating system.
- One mode bit is added to the page table. Valid/ Invalid bit.
- The OS sets this bit for each page to allow or disallow access to that page. Ex: A 14-bit address space (0 to16383).
- Addresses in page 0, 1, 2, 3, -----5 are mapped normally through the page table.
- Any attempt to generate an address in page 6 or 7 is a trap to operating system.

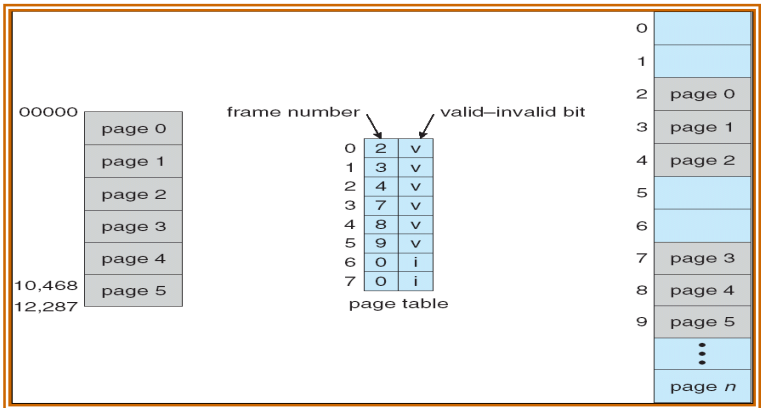


Fig: Valid (v) or invalid (i) bit in a page table.

3.3.4. Shared Pages:

- Another advantage of paging is the possibility of sharing common code (Time sharing system).

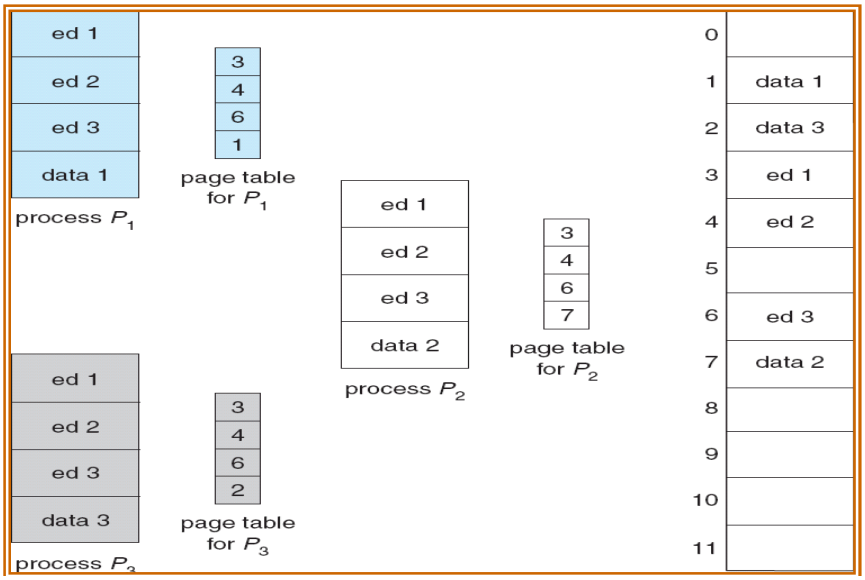


Fig: Sharing of code in a paging environment.

- Consider a system that supports 40 users, each of which executes a text editor 30K and 5K for data space. We would need 1400K (35*40).
- If the code is reentrant code it could be shared, three-page editor being shared among three processes. Each process has its own data page.
- If the code is "reentrant" (pure code) then it never changes during execution.
- Two or more processes can execute the same code at the same time.
- Each process has its own copy of registers and data storage to hold the data for its execution.
- So we need a copy of the editor (30 K), plus 40 copies of the 5 K of data space for user, total space required is now 230 K.
- Compilers, assemblers, database systems can also be shared.

3.4. Segmentation:

- The user's view of memory is not the same as the actual physical memory.
- The user's view is mapped onto physical memory segmentation is a memory management scheme which supports the user view of memory.

3.4.1. Basic method

- A logical address space is a collection of segments. Each segment has a name and offset within the segment.

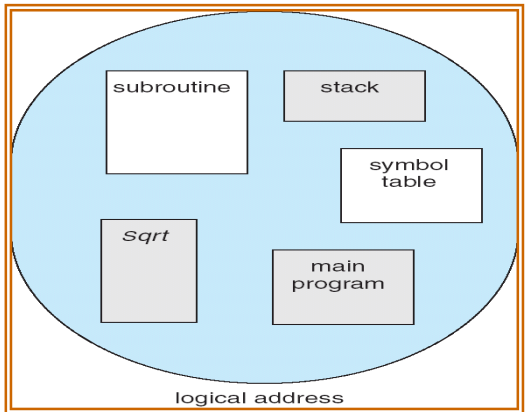


Fig: User's view of a program.

- We must define an implementation to map two dimensional user-defined addresses into one-dimensional physical addresses. This mapping is affected by a segment table.

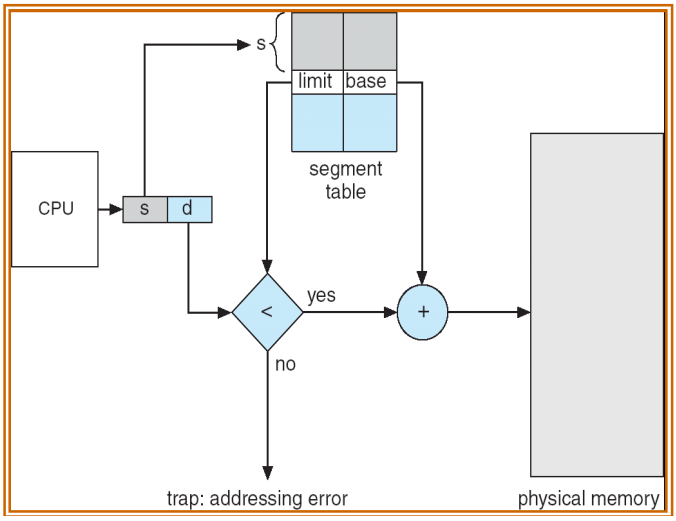


Fig: Segmentation hardware.

- Logical address consists of two parts: a segment number, s, and an offset into that segment, d.
- The segment numbers are used as an index to the segment table.
- Each entry of the segment table has a segment base and a segment limit.
- The offset *d* of the logical addresses must be between 0 and the segment limit. If it is not, we trap to the operating system.

3.4.2. Hardware:

- If it is legal, it is added to the segment base address to produce the addresses in physical memory of the desired word.
- The segment table is array of base/limit register pairs.

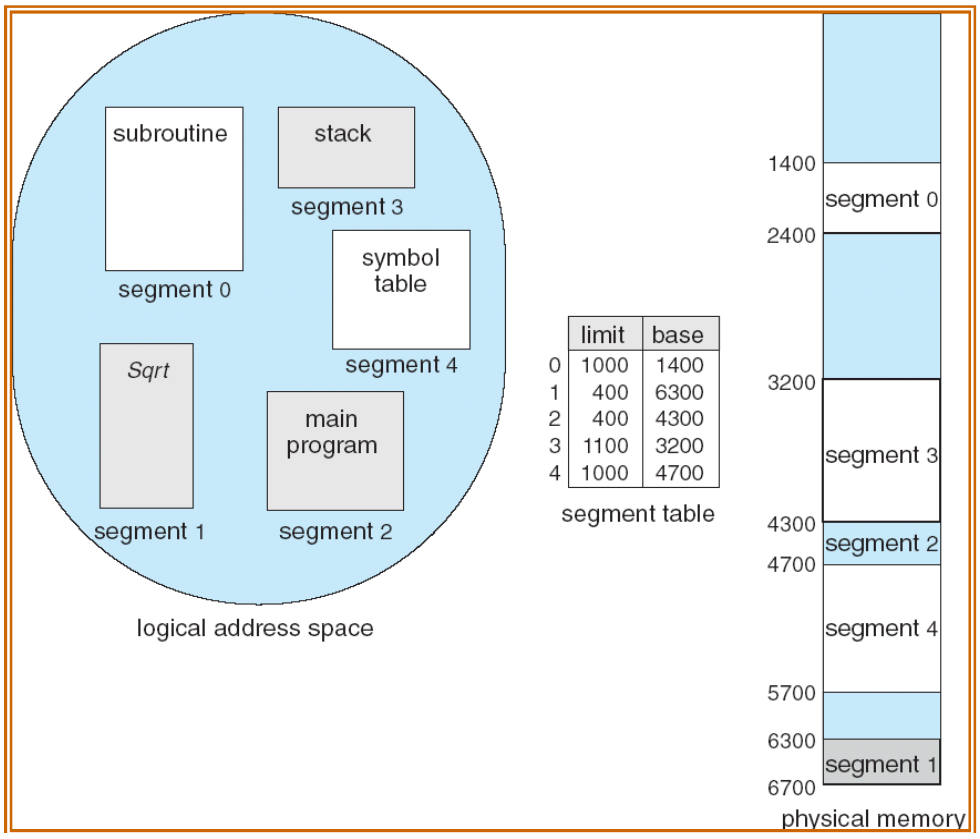


Fig: Example of segmentation.

- We have five segments the segment table has separate entry for each segment.
- The segment table contains the beginning address and the limiting address.

- F**or example,** segment 2 is 400 words long a beginning at 4300. So a reference word 53 of segment 2 is mapped on the physical address as 4300+53=4353.

PART II: <u>Virtual-Memory Management:</u>

- Virtual memory is a technique that allows the execution of processes that are not completely in memory.
- One major advantage of this scheme is that programs can be larger than physical memory.
- Virtual memory abstracts main memory into an extremely large , uniform array of storage, separating logical memory as viewed by the user from physical memory.
- This technique frees programmers from the concerns of memory storage limitations.
- Virtual memory also allows processes to share files easily and to implement shared memory.
- The instructions that are currently executing must be in physical memory.
- In order to meet this requirement the entire logical address space should placed in physical memory.
- Dynamic loading can help to ease this requirement.
- In many cases the entire program is not needed in main memory. For instance consider the following.
  - Certain options and features of a program may be used rarely.
  - Arrays, lists, and tables are often allocated more memory than they actually need. Example an array may be declared 100 by 100 elements, even though it uses 10 by 10 elements.

Benefits of virtual memory:

- A program would no longer be constrained by the amount of physical memory that is available.

- CPU utilization and throughput will be increased and response time or turnaround time will be decreased.
- Less I/O would be needed to load or swap each user program into memory, so each user program would run faster.
- Thus, running a program that is not entirely in memory would benefit both the system and the user:

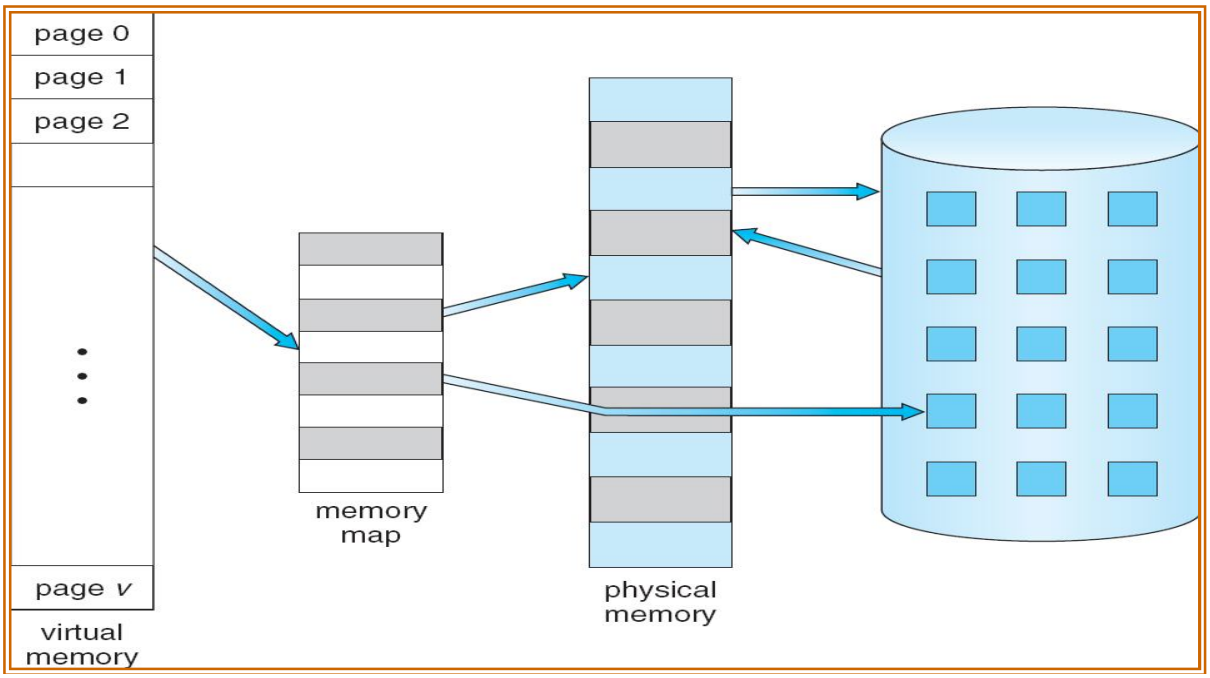


Fig: Diagram showing virtual memory that is larger than physical memory.

- Virtual memory involves the separation of 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.
- Virtual memory makes the task of programming much easier.

Virtual address space:

- The virtual address space of a process refers to the logical (or virtual) view of how a process is stored in memory.

- Here a process begins at a certain logical address say, address 0—and exists in contiguous memory.
- Physical memory may be organized in page frames, that the physical page frames assigned to a process may not be contiguous.
- Memory management unit (MMU) to map logical pages to physical page frames in memory.

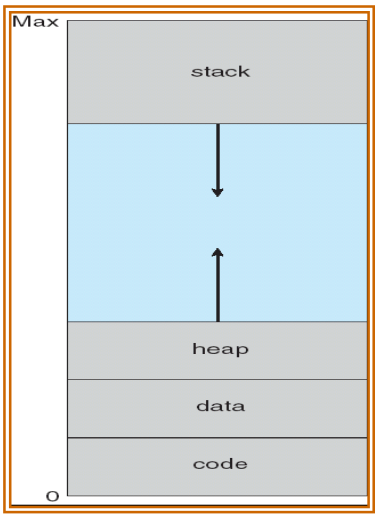


Fig: Virtual address space

- Here the heap to grow upward direction memory as it is used for dynamic memory allocation.
- Similarly, we allow for the stack to grow downward in memory through successive function calls.
- The large blank space (or hole) between the heap and the stack is part of the virtual address space but will require actual physical pages only if the heap or stack grows.
- Virtual address spaces that include holes are known as sparse address spaces.
- Virtual memory also allows files and memory to be shared by two or more processes through page sharing. This leads to the following benefits:

- System libraries can be shared by several processes through mapping of the shared object into a virtual address space.
- Virtual memory enables processes to share memory. Virtual memory allows one process to create a region of memory that it can share with another process.
- Virtual memory can allow pages to be shared during process creation with the fork ( ) system call, thus speeding up process creation.

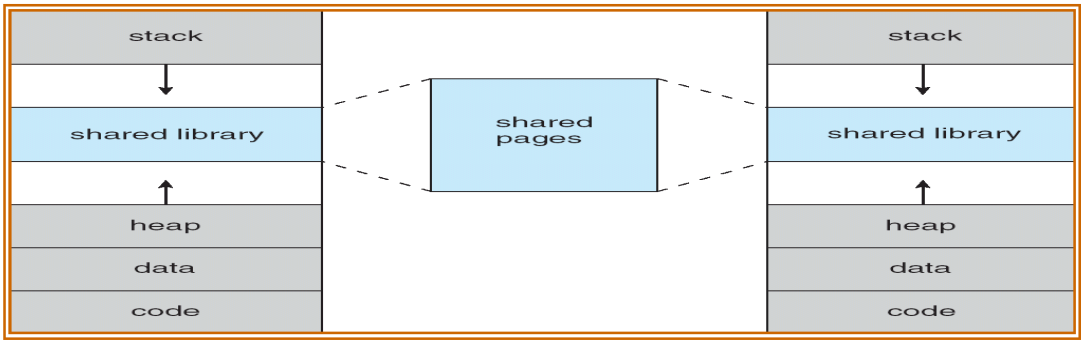


Fig: Shared library using virtual memory.

3.5. Demand paging:

- Initially load pages only as they are needed. This technique is known as demand paging and is commonly used in virtual memory systems.
- With demand-paged virtual memory, pages are only loaded when they are demanded during program execution; pages that are never accessed are thus never loaded into physical memory.

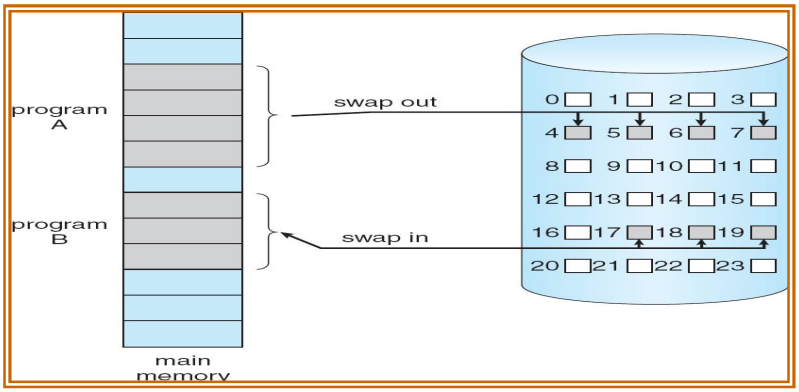


Fig: Transfer of a paged memory to contagious disk space.

- Demand-paging system is similar to a paging system with swapping where processes reside in secondary memory (usually a disk).
- 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.
- **Lazy swapper:** It never swaps a page into memory unless that page will be needed.
- A swapper manipulates entire processes, whereas a pager is concerned with the individual pages of a process. Here the word ***pager*** is used, rather than *swapper,* in connection with demand paging.

3.5.1. Basic Concepts:

- What happens if the process tries to access a page that was not brought into memory?
- Access to a page marked invalid causes a paging hardware, in translating the address through the page table, will notice that the invalid bit is set, causing a trap to the operating system.
- This trap is the result of the operating system's failure to bring the desired page into memory.

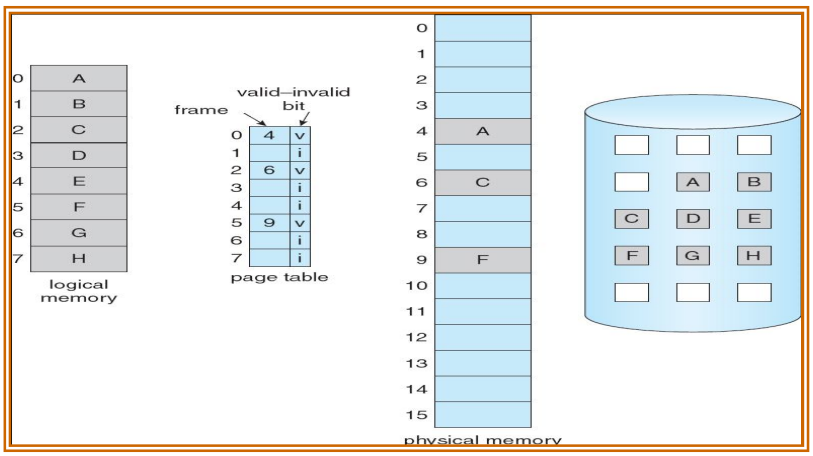


Fig: Page table when some pages are not in memory.

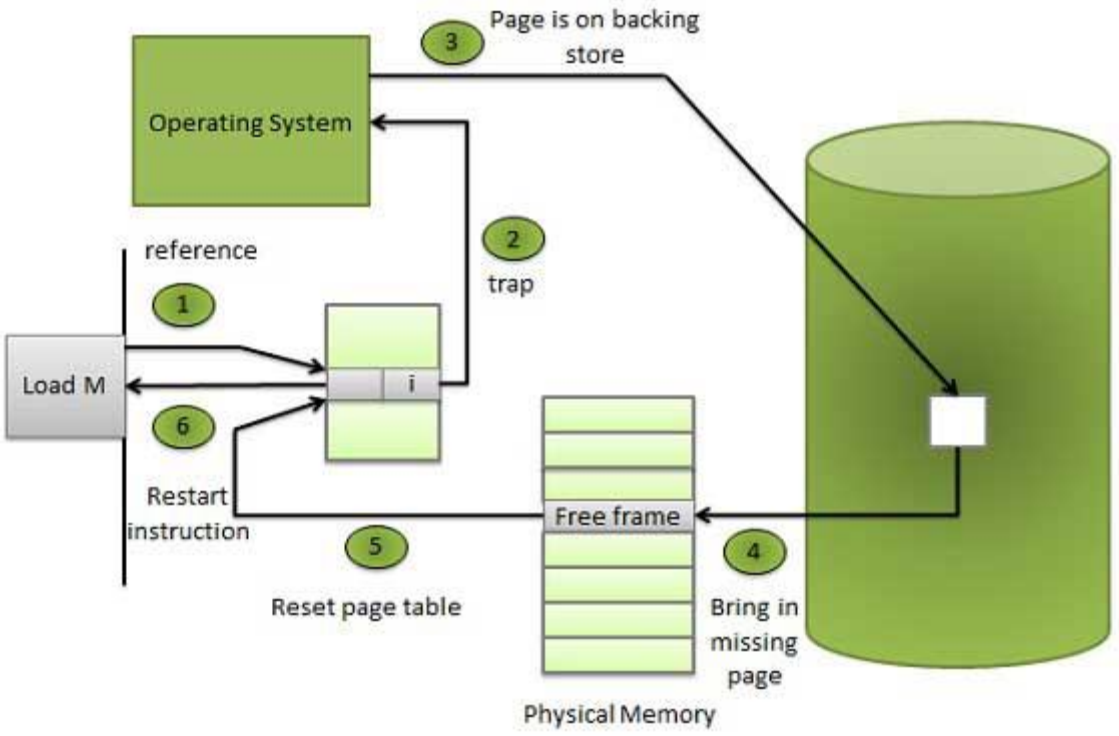


Fig: Steps in handling a page fault.

The procedure for handling this page fault is straightforward:

1) We check an internal table (usually kept with the process control block) for this process to determine whether the reference was a valid or an Invalid memory access.
2) If the reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.
3) We find a free frame (by taking one from the free-frame list, for example). We schedule a disk operation to read the desired page into the newly allocated frame.
4) When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
5) We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

6) Restart the instruction that was interrupted. By the illegal address –trap. The process can now access the page as if it had always been in the memory.

The hardware is same for paging and swapping

- **A page table** with the ability to mark an entry invalid through a valid/invalid bit.
- **Secondary memory:** This memory holds those pages that are not present in main memory. The secondary memory is usually a high-speed disk. It is known as the swap device, and the section of disk used for this purpose is known swap space.

3.5.2. Performance of Demand Paging:

A page fault causes the following sequence to occur:

1. Trap to the operating system.
2. Save the user registers and process state.
3. Determine that the interrupt was a page fault.
4. Check that the page reference was legal and determine the location of the page on the disk
5. Issue a read from the disk to a free frame:
    a. Wait in a queue for this device until the read request is serviced.
    b. Wait for the device seek and/ or latency time.
    c. Begin the transfer of the page to a free frame.
6. While waiting, allocate the CPU to some other user (CPU scheduling, optional).
7. Receive an interrupt from the disk I/0 subsystem (I/0 completed).
8. Save the registers and process state for the other user (if step 6 is executed).
9. Determine that the interrupt was from the disk
10. Correct the page table and other tables to show that the desired page is now in memory.
11. Wait for the CPU to be allocated to this process again.
12. Restore the user registers, process state, and new page table, and then resume the interrupted instruction.

3.6. Page Replacement:

- While executing a user process, a page fault occurs.
- The hardware traps to the operating system; which checks its internal tables to see that this page fault and not an illegal memory access.
- This operating system determines where the desired page is residing on the backing store, but then finds that there are *no* free frames on the free-frame list; all memory is in use.

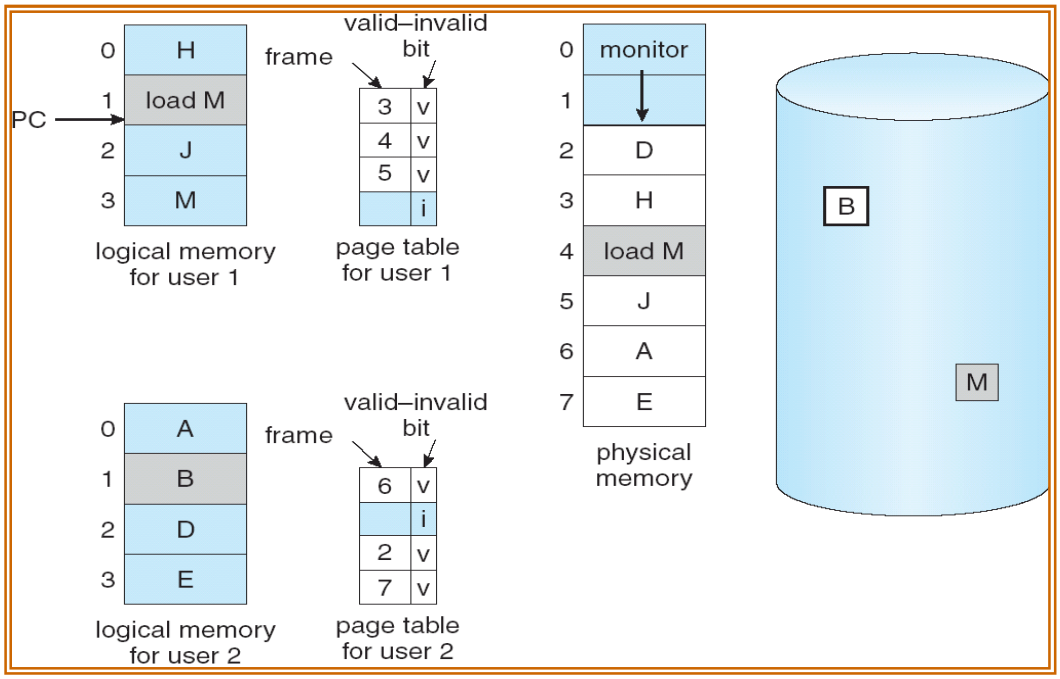


Fig: Need for Page Replacement

3.6.1. Basic Page Replacement:

- If no frame is free, find one which is not currently being used and free it.
- We can free a frame by writing its contents to the backing store, and changing the page table (and all other tables) to indicate that the page is no longer in memory.
- The freed frame can now be used to hold the page for which the process faulted.

1. Find the location of the desired page on disk

2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a victim frame

3. Bring the desired page into the (newly) free frame; update the page and frame tables

4. Restart the process

- This overhead by the use of a dirty **bit**.
- Each page or frame may have a dirty bit associated with it in the hardware.
- The modify bit for a page is set by the hardware.
- When we select a page for replacement, we examine its dirty bit.
- If the bit is set, we know that the page has been modified since it was read in from the baking store.
- In this case, we must write that page to the backing store.

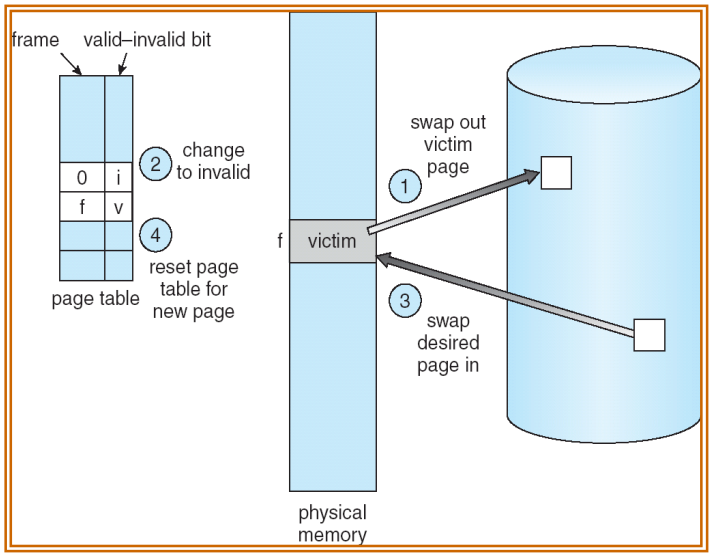


Fig: Page Replacement

- With demand paging, the size of the logical address space is no longer constrained by physical memory.

- If we have a user process of twenty pages, we can execute it in ten frames simply by using **demand paging**
- To implement demand paging two problems to be solved ***frame-allocation algorithm*** and a ***page-replacement algorithm.***
- If page replacement is required, we must select the frames that are to be replaced.

3.6.2. Page Replacement Algorithms:

- We have to select the algorithm which the lowest page-fault rate.
- An algorithm is evaluated by running it on a particular string of memory references and computing the number of page faults.
- The string of memory references is called a **reference string**.
- We can generate reference strings artificially (by using a random-number generator

3.6.2.1. FIFO Page Replacement:T

- The simplest page-replacement algorithm is a first-in, first-out (FIFO) algorithm.
- A FIFO replacement algorithm associates with each page the time when that page was brought into memory.
- When a page must be replaced, the oldest page is chosen.
- We can create a FIFO queue to hold all pages in memory.
- We replace the page at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue.

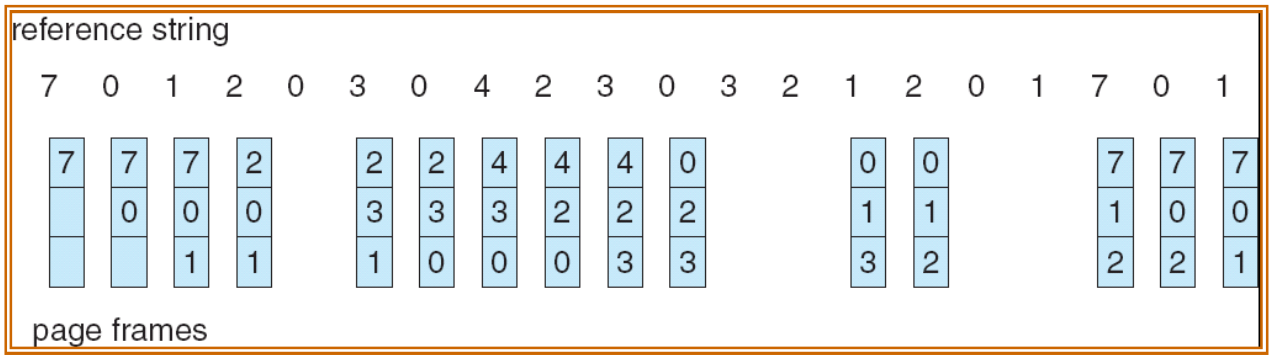


Fig: FIFO Page-Replacement algorithm

Belady's Anomaly:

- The page-fault rate may *increase* as the number of allocated frames increases. This phenomenon is called Belady's Anomaly.
- To illustrate the problems that are possible with a FIFO page-replacement algorithm, we consider the following reference string:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

  - 3 frames (3 pages can be in memory at a time per process)-9 page faults
  - 4 frames-10 page faults

3.6.2.2. Optimal Page Replacement:

- Replace the page that will not be used for the longest period of time.
- Optimal Page Replacement has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly.

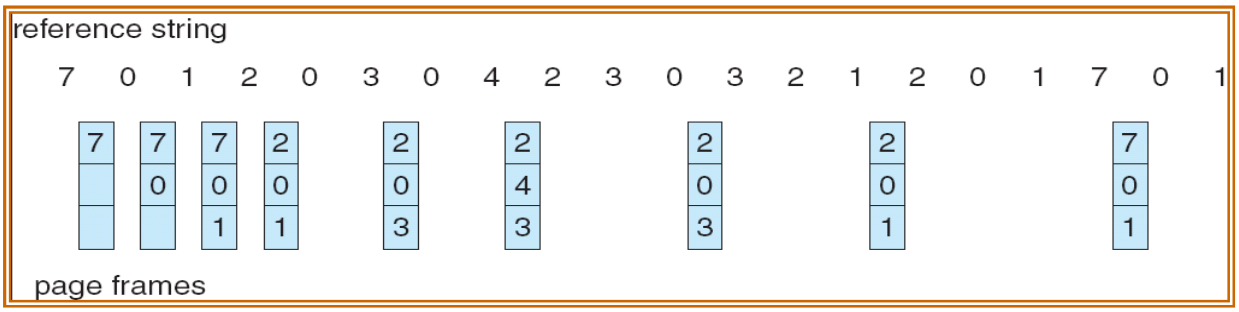


Fig: Optimal Page-Replacement algorithm

- For example, on our sample reference string, the optimal page-replacement algorithm would yield nine page faults.
- The first three references cause faults that fill the three empty frames. The reference to page 2 replaces page 7, because page 7 will not be used until reference 18, whereas page 0 will be used at 5, and page 1 at 14.
- The reference to page 3 replaces page 1, as page 1 will be the last of the three pages in memory to be referenced again.

3.6.2.3. LRU Page Replacement:

- If we use the recent past as an approximation of the near future, then we can replace then that *has not been used* for the longest period of time.
- This approach is the approach is known as LRU Page Replacement.

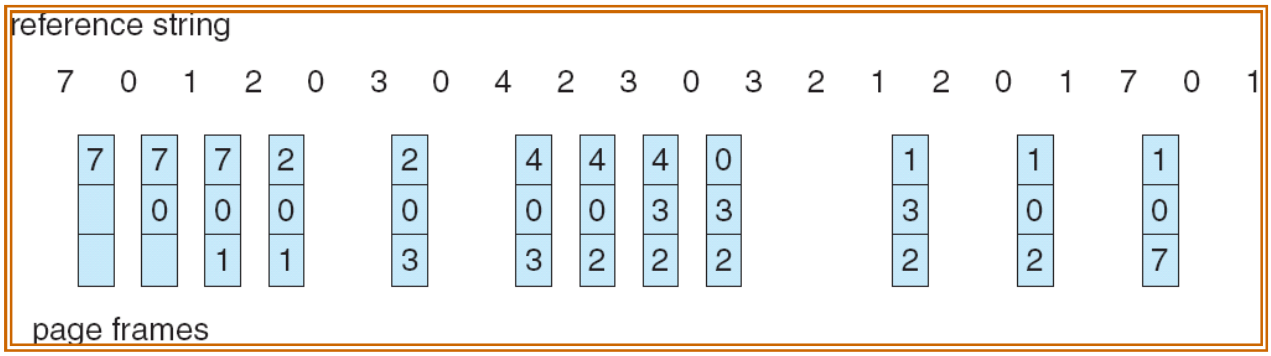


Fig: LRU Page-Replacement algorithm

- The LRU algorithm produces twelve faults.
- The first five faults are the same as those for optimal replacement. When the reference to page 4 occurs.LRU replacements sees that, of the three frames in memory, page 2 was used least recently.
- Thus, the LRU algorithm replaces page 2, not knowing that page 2 is about to be used. When it then faults for page 2, the LRU algorithm replaces page 3, since it is now the least recently used of the three pages in memory.
- Despite these problems, LRU replacement with twelve faults is much better than FIFO replacement with fifteen.

3.7. ALLOCATION OF FRAMES:

- Normally, there are fixed amounts of free memory with various processes at different time in a system.
- The question is how this fixed amount of free memory is allocated among the different processes.
- The simplest case is the single process system.
- All available memory for user programs can initially be put on the free frame list (pure demand paging).

- When the user program starts its execution, it will generate a sequence of page faults.
- The user program would get all free frames from the free frame list.
- As soon as this list was exhausted, and the more free frames are required, the page replacement algorithm can be used to select one of the in-used pages to be replaced with the next required page and so on.
- After the program was terminated, all used pages are put on the free frame list again.
- The frame allocation procedure is more complicated when there are two of more programs in memory at the same time.

3.7.1. MINIMUM NUMBER OF FRAMES:

- We cannot allocate more than the total number of available frames in the system.
- On the other hand, there is a minimum number of frames which must be allocated.
- This minimum number is determined by the instruction architecture.
- It is obvious that we must provide enough frames to hold all the different pages that any single instruction can reference.
- For example, all memory reference instructions of-a machine have only one memory address.
- So we need at least one frame for the instruction code and one frame for the memory reference.
- If one level indirect addressing is allowed, a load instruction on page m can refer to an address on page. It is an indirect reference to page k.
- We need three pages.

3.7.2. ALLOCATION ALGORITHM:

- The simplest way is to divide m available frames among n processes to give everyone an equal share, m/n frames.
- This is called **equal allocation**.

- Various processes will need different amounts of memory. If the equal allocation is applied, there can be some frames wasted.
- Therefore, other allocation scheme can be used to give available memory to each process according to its size. This is called, proportional allocation.
- Let the size of the virtual memory for process p_i be s_i, the number of frames allocated to the process p_i be a_i, and define

$$S = \sum s_i$$

- If the total number of available frames is m, then a_i can be calculated:

$$a_i = (s_i/S) * m.$$

- Of course ai must be adjust to be a integer, greater than the minimum number of frames required by the instruction set with a sum not exceeding m.
- In both of these cases, the number of frames allocated to each process may vary according to the multiprogramming level.

Global versus Local Allocation:

- When it's necessary to find free page frames, what set of pages should become candidates for replacement?
- **Local replacement** policies replace pages that belong to the process that needs the new frame.
- **Global policies** consider all unlocked frames. Most systems use global replacement because it is easy to implement, has minimal overhead, and performs reasonably well.

	Local Replacement	**Global Replacement**
Fixed Allocation	Rarely used -A process is given a fixed number of frames. Page faults are satisfied from this set.	This combination isn't possible
Variable Allocation	The process is given a fixed allocation and pages to be replaced are chosen from this set. Periodically, the	Replacement pages are chosen from any page in memory. Resident set size varies, although

	resident, set size is re-evaluated. Pages can be added or subtracted.	by a blind process. This is the most common approach

3.8. THRASHING:

- Consider a process which does not have enough frames. It is possible to reduce the no of allocated frames to the minimum.
- There are some no: of pages that are in active use. if the process does not have no of frames ,it will very quickly page fault since all of the pages are in active use.
- It must replace a page which will be needed again a right way,
- Consequently if very quickly faults arrive again and again.
- This high paging activity is called “Trashing”. (A process is trashing if it is spending more time in paging than executing).

3.8.1. Causes of thrashing

- The OS monitors CPU utilization, if CPU utilization is too low;
- the degree of multiprogramming is increased by introducing a new process to the system.
- A global page replacement algorithm is used.
- It will create page faults.
- If the graph is drawn between CPU utilization and multiprogramming as the degree of multiprogramming increases CPU utilization also increases until maximum reached.
- If degree of multi programming increased further trashing sets and CPU utilization decreases slowly.

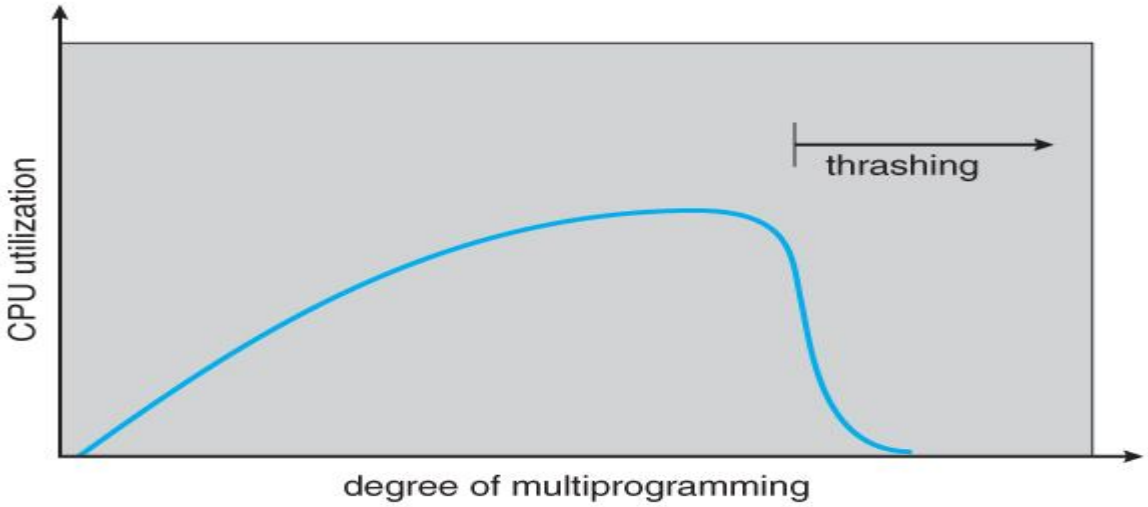


Fig: showing trashing between CPU utilization and degree of multiprogramming.

- The effect of thrashing can be limited by using a locator priority replacement algorithm.
- With local replacement if one process starts trashing it cannot steal frames from another process and cause it to trash also.
- If process are trashing they will be in the queue for paging device most of the time average service time for page fault increases there by effective access time increases.

3.8.2. Working-Set Model

- Δ ≡ working-set window ≡ a fixed number of page references Example: 10,000 instruction.
- WSS_i (working set of Process P_i) = total number of pages referenced in the most recent Δ (varies in time)
  1. if Δ too small will not encompass entire locality
  2. if Δ too large will encompass several localities
  3. if $\Delta = \infty \Rightarrow$ will encompass entire program
  4. $D = \Sigma\ WSS_i \equiv$ total demand frames
- if $D > m \Rightarrow$ Thrashing
- Policy if $D >$ m, then suspend one of the processes

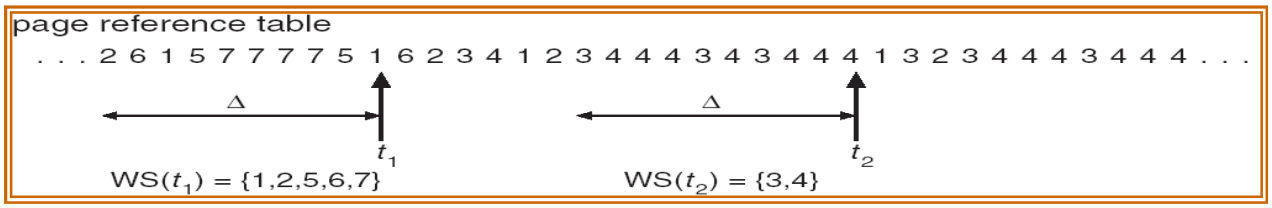


Fig: Working-set model.

- If a page is in active use, it will be in the working set.
- If it is no longer being used, it will drop from the working set 6 time units after its last reference.
- Thus, the working set is an approximation of the program's locality.
- For example, given the sequence of memory references. If 6 = 10 memory references, then the working set at time t1 is {1, 2, 5, 6, and 7}.
- By time *t2,* the working set has changed to {3, 4}.
- The accuracy of the working set depends on the selection of 6. If 6 are too small, it will not encompass the entire locality; if 6 are too large, it may overlap several localities

3.8.3. Page-Fault Frequency:

- The specific problem is how to prevent thrashing.
- Thrashing has a high page-fault rate. Thus, we want to control the page-fault rate.
- When it is too high, we know that the process needs more frames. Conversely, if the page-fault rate is too low, then the process may have too many frames.
- We can establish upper and lower bounds on the desired page-fault rate
- If the actual page-fault rate exceeds the upper limit, we allocate the process another frame;
- If the page-fault rate falls below the lower limit, we remove a frame from the process.
- Thus, we can directly measure and control the page-fault rate to prevent thrashing.

UNIT-III
Assignment-Cum-Tutorial Questions
SECTION-A

Objective Questions

1. An address generated by the CPU is commonly referred as _____
2. _____holds smallest legal physical memory address [ ]
   A) Limit register B) Relocation register
   C) Segment offset D) Segment register
3. _____specifies size of the range of addresses [ ]
   A) Limit register B) Relocation register
   C) Segment offset D) Segment register
4. User program never sees the real physical addresses (T/F)
5. _____ is a solution to external fragmentation. [ ]
   A) Compaction B) Blocks C) Thrashing D) frames
6. Breaking physical memory into fixed-sized blocks called [ ]
   A) Pages B) Frames C) Partitions D) None
7. Paging Suffers from _____ [ ]
   A) Internal Fragmentation B) External Fragmentation
   C) Both Internal and External Fragmentation D) None
8. Breaking logical memory into blocks of the same size called_____
9. Segmentation Suffers from _____ [ ]
   A) Internal Fragmentation B) External Fragmentation
   C) Both Internal and External Fragmentation D) None
10. A routine is not loaded until it is called is referred as [ ]
    A) Dynamic loading B) Dynamic linking
    C) Shared libraries D) Dynamic binding
11. Effective Access Time= _____
12. In paging logical address is divided into two parts_____,_____
13. In segmentation logical address is divided into_____,_____

14. Which of the following page replacement algorithms suffers from Belady's anomaly? [ ]

A) FIFO B) LRU C) OPTIMAL D) LFU

15. Consider a virtual memory system with FIFO page replacement policy. For an arbitrary page access pattern, increasing the number of page frames in main memory will [ ]

A) always decrease the number of page faults

B) always increase the number of page faults

C) sometimes increase the number of page faults

D) never affect the number of page faults **(GATE-2001)**

16. The optimal page replacement algorithm will select the page that

A) Has not been used for the longest time in the past. [ ]

B) Will not be used for the longest time in the future.

C) Has been used least number of times.

D) Has been used most number of times. (**GATE-2002)**

17. A virtual memory system uses First In First Out (FIFO) page replacement policy and allocates a fixed number of frames to a process. Consider the following statements. [ ]

P: Increasing the number of page frames allocated to a process sometimes increases the page fault rate.

Q: Some programs do not exhibit locality of reference.

Which one of the following is TRUE?

A) Both P and Q are true, and Q is the reason for P

B) Both P and Q are true, but Q is not the reason for P.

C) P is false, but Q is true

D) Both P and Q are false (**GATE-2007)**

18. The essential content(s) in each entry of a page table is / are

A) Virtual page number [ ]

B) Page frame number

C) Both virtual page and page frame number

D) Access right information **(GATE-2009)**

19. Dirty bit for a page in a page table [ ]

A) helps avoid unnecessary writes on a paging device

B) helps maintain LRU information

C) allows only read on a page

D) None of the above **(ISRO 2015)**

20. Consider a 32-bit machine where four-level paging scheme is used. If the hit ratio to TLB is 98%, and it takes 20 nanosecond to search the TLB and 100 nanoseconds to access the main memory what is effective memory access time in nanoseconds? [ ]

A) 126 B) 128 C) 122 D) 120

(ISRO 2011)

21. A page fault [ ]

A) Occurs when a program accesses an available page on memory

B) is an error in a specific page

C) is a reference to a page belonging to another program

D) occurs when a program accesses a page not currently in memory

(ISRO2009)

22. The page replacement algorithm which gives the lowest page fault rate is [ ]

A) LRU B) FIFO C) Optimal page replacement D) Second chance algorithm

(ISRO 2008)

23. Which of the following statements are true? [ ]

a) External Fragmentation exists when there is enough total memory space to satisfy a request but the available space is contiguous.

b) Memory Fragmentation can be internal as well as external.

c) One solution to external Fragmentation is compaction. **(NET 2018)**

A) (a) and (b) only B) (a) and (c) only

C) (b) and (c) only D) (a), (b) and (c)

24. Consider the following segment table in segmentation scheme:

Segment	Base	Limit
0	200	200
1	500	12510
2	1527	498
3	2500	50

What happens if the logical address requested is -Segment Id 2 and offset 1000? [ ]

A) Fetches the entry at the physical address 2527 for segment Id2

B) A trap is generated

C) Deadlock

D) Fetches the entry at offset 27 in Segment Id 3 **(ISRO2015)**

SECTION-B

Descriptive Questions

1. Compare need of swap-in and swap-out operations?
2. Explain about MVT and MFT in detail?
3. Briefly explain the concept of contiguous memory allocation.
4. Classify two Counting-Based page replacement algorithms.
5. Explain paging scheme for memory management, discuss the paging hardware and paging model.
6. Differentiate Internal and External fragmentation.
7. With a neat diagram explain how segmentation works?
8. What is the necessity of Demand Paging?
9. Illustrate the concepts of demand paging? Why it is called as lazy swappers?
10. Demonstrate in detail Copy-on-Write technique?
11. Summarize various page replacement algorithms?
    a) FIFO b) LRU c) LFU d) OPTIMAL
12. Define thrashing. Explain working set window model to handle thrashing problem.
13. Compare and Contrast First Fit, Best Fit and Worst Fit.
14. Illustrate the concept of Segmentation with neat Sketch.

Problems:

1. Find the number of page faults in FIFO and LRU page replacement algorithms for the following reference string;
   7 0 2 1 3 4 2 1 0 2 1 4 3 2 1 0 0 1 2 1 (no. of frames=3)
2. Make use of the reference string **7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1.** Identify number of page faults using (Assume that there are 3 page frames which are initially empty) LRU, Optimal page replacement algorithms.

3. Make use of the reference string **7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1.** Identify number of page faults using FIFO page replacement algorithm. Assume that there are 3 page frames which are initially empty.

4. Explain Optimal page replacement algorithm. Apply the same to find out page faults for the reference string **1,2,3,4,5,3,2,1,6,7,8,7,6,9,1,2,4,3,5** by assuming frame size as 4.

5. Consider the following reference **1,2,3,4,5,3,2,1,6,7,8,7,6,9,1,2,4,3,5** String, How many Page Faults would occur for LRU and FIFO Page Replacement Algorithms for frame size of 3.

6. Consider a logical address space of 8 pages of 1024 words mapped into memory of 32 frames. How many bits are there in the logical address?

7. Consider the following page reference string : 1**, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6**. Which of the following options, gives the correct number of page faults related to LRU, FIFO, and optimal page replacement algorithms respectively, assuming 05 page frames and all frames are initially empty ?

8. A computer has 16 pages of virtual address space but the size of main memory is only four frames. Initially the memory is empty. A program references the virtual pages in the order **0, 2, 4, 5, 2, 4, 3, 11, 2, 10.** How many page faults occur if LRU page replacement algorithm is used?

9. Consider a virtual page reference string **1, 2, 3, 2, 4, 2, 5, 2, 3, 4.** Suppose LRU page replacement algorithm is implemented with 3 page frames in main memory. Then the number of page faults are___.

10. A system uses 3 page frames for storing process pages in main memory. It uses the Least Recently Used (LRU) page replacement policy. Assume that all the page frames are initially empty. What is the total number of page faults that will occur while processing the page reference string given below? **4, 7, 6, 1, 7, 6, 1, 2, 7, 2**

SECTION-C

I. QUESTIONS AT THE LEVEL OF GATE

1. Suppose that the virtual Address space has eight pages and physical memory with four page frames. If LRU page replacement algorithm is used, _____ number of page faults occur with the reference string.0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 2 3 4 1 [ ]
A) 13 B) 12 C) 11 D) 10 **(NET 2016)**

2. Consider the data given in above question. Least Recently Used (LRU) page replacement policy is a practical approximation to optimal page replacement. For the reference string 1, 2, 1, 3, 7, 4, 5, 6, 3, 1, how many more page faults occur with LRU than with the optimal page replacement policy? [ ]
A) 0 B) 1 C) 2 D) 3 **(GATE 2017)**

3. Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB, and 250 KB, where KB refers to kilobyte. These partitions need to be allotted to four processes of sizes 357 KB, 210 KB, 468 KB and 491 KB in that order. If the best fit algorithm is used, which partitions are NOT allotted to any process? [ ]
A) 200 KB and 300 KB B) 200 KB and 250 KB
C) 250 KB and 300 KB D) 300 KB and 400 KB **(GATE 2015)**

4. Assume that there are 3 page frames which are initially empty. If the page reference string is 1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6, the number of page faults using the optimal replacement policy is_____. [ ]
A) 5 B) 6 C) 7 D) 8 **(GATE 2014)**

5. Consider the virtual page reference string 1, 2, 3, 2, 4, 1, 3, 2, 4, 1 On a demand paged virtual memory system running on a computer system that main memory size of 3 pages frames which are initially empty. Let LRU, FIFO and OPTIMAL

denote the number of page faults under the corresponding page replacements policy. Then [ ]

A) OPTIMAL < LRU < FIFO B) OPTIMAL < FIFO < LRU

C) OPTIMAL=LRU D) OPTIMAL=FIFO **(GATE 2012)**

6. Assume that a main memory with only 4 pages, each of 16 bytes, is initially empty. The CPU generates the following sequence of virtual addresses and uses the Least Recently Used (LRU) page replacement policy. 0, 4, 8, 20, 24, 36, 44, 12, 68, 72, 80, 84, 28, 32, 88, 92. How many page faults does this sequence cause? What are the page numbers of the pages present in the main memory at the end of the sequence? [ ]

A) 6 and 1, 2, 3, 4 B) 7 and 1, 2, 4, 5

C) 8 and 1, 2, 4, 5 D) 9 and 1, 2, 3, 5 **GATE2008)**

7. A process has been allocated 3 page frames. Assume that none of the pages of the process are available in the memory initially. The process makes the following sequence of page references (reference string): 1, 2, 1, 3, 7, 4, 5, 6, 3, 1. If optimal page replacement policy is used, how many page faults occur for the above reference string? [ ]

A) 7 B) 8 C) 9 D) 10 **(GATE-2007)**

8. Consider a fully associative cache with 8 cache blocks (numbered 0-7) and the following sequence of memory block requests: 4, 3, 25, 8, 19, 6, 25, 8, 16, 35, 45, 22, 8, 3, 16, 25, 7. If LRU replacement policy is used, which cache block will have memory block 7? [ ]

A) 4 B) 5 C) 6 D) 7 **(GATE 2004)**