7...

Memory Management

Objectives...

- To learn basic concepts Memory Management.
- To get information of Swapping, Paging and Segmentation.
- To study about Virtual Memory Management.

7.1 BACKGROUND [S-23]

- Memory consists of a large array of words or bytes, each with its own address. Both the CPU and I/O system interact with memory. Interaction is achieved through a sequence of reads or writes to specific memory addresses.
- The CPU fetches instructions from memory according to the value of the program counter. These instructions may cause additional loading from and storing to specific memory addresses.
- Memory management is the process of controlling and coordinating computer memory. This process assigning portions called blocks to various running programs to optimize overall system performance.

Major activities of Memory Management:

- Keep track of which parts of memory are currently being used and by whom.
- Decide which processes are to be loaded into memory when memory space becomes available.
- Allocate and deallocate memory space as needed.

7.1.1 Basic Hardware

- Memory accesses to registers are very fast, generally one clock tick. A CPU may be able to execute more than one machine instruction per clock tick.
- A pair of base and limit registers define the logical address space. Every memory access made by a user process is checked against these two registers.

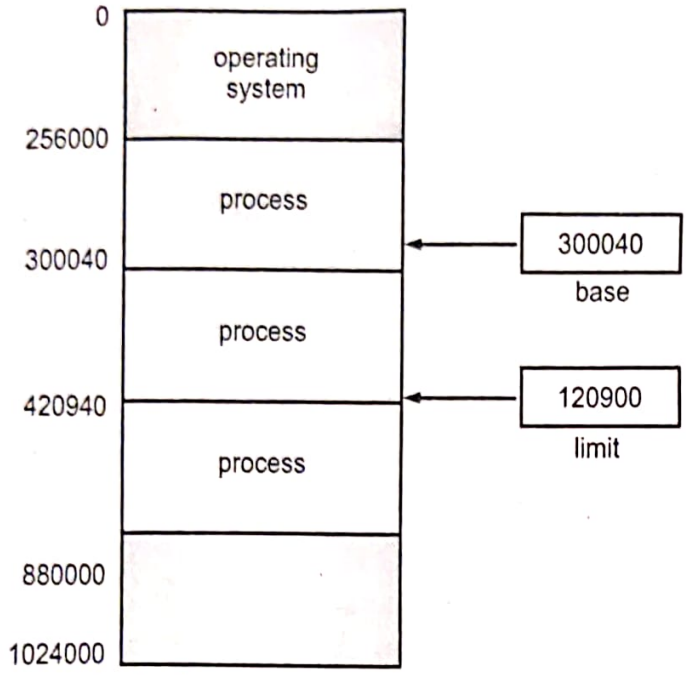

Fig. 7.1: Logical address space with Base and Limit Registers

- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user, otherwise a fatal error is generated.

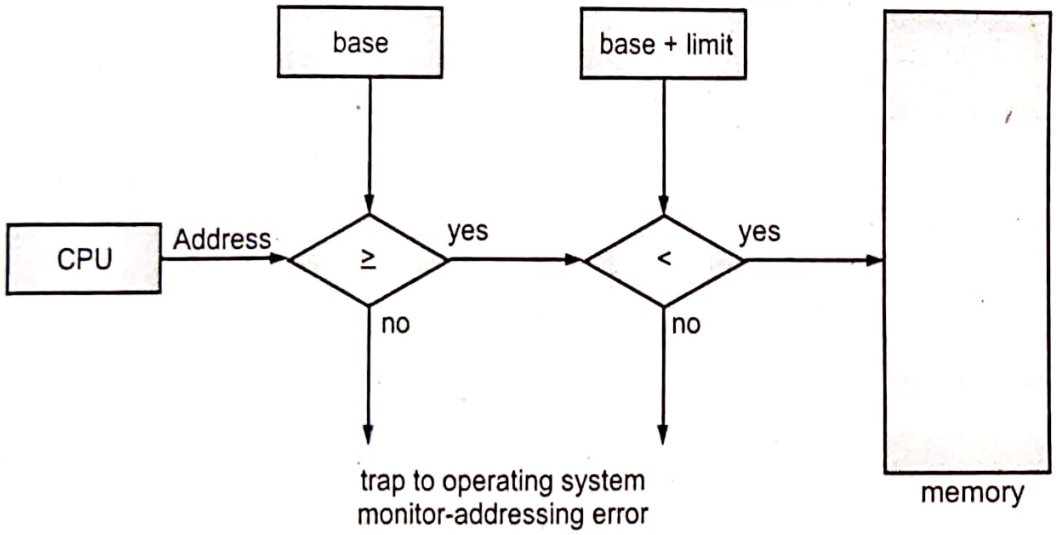

Fig. 7.2: Protection of Hardware Address

7.1.2 Address Binding

[W-18; S-22, 23]

- Programs are stored on the secondary storage disks as binary executable files. When the programs are to be executed they are brought in to the main memory and placed within a process.
- The collection of processes on the disk waiting to enter the main memory forms the Input Queue.
- One of the processes which are to be executed is fetched from the queue and placed in the main memory. During the execution it fetches instruction and data from main memory.
- After the process terminates it returns back the memory space. During execution the process will go through different steps and in each step the address is represented in different ways.

- In source program the address is symbolic. The compiler converts the symbolic address to re-locatable address. The loader will convert this re-locatable address to absolute address.
- Each binding is a mapping from one address space to another. These absolute addresses loaded into memory to be executed.
- The binding of instructions and data to memory addresses can be done at any step along the following way:
  - (a) **Compile time:** If we know whether the process resides in memory then absolute code can be generated. If the static address changes then it is necessary to re-compile the code from the beginning.
  - (b) **Load time:** If the compiler doesn't know whether the process resides in memory then it generates the re-locatable code. In this the binding is delayed until the load time.
  - (c) **Execution time:** If the process is moved during its execution from one memory segment to another then the binding is delayed until run time. Special hardware is used for this. Most of the general purpose operating system uses this method.

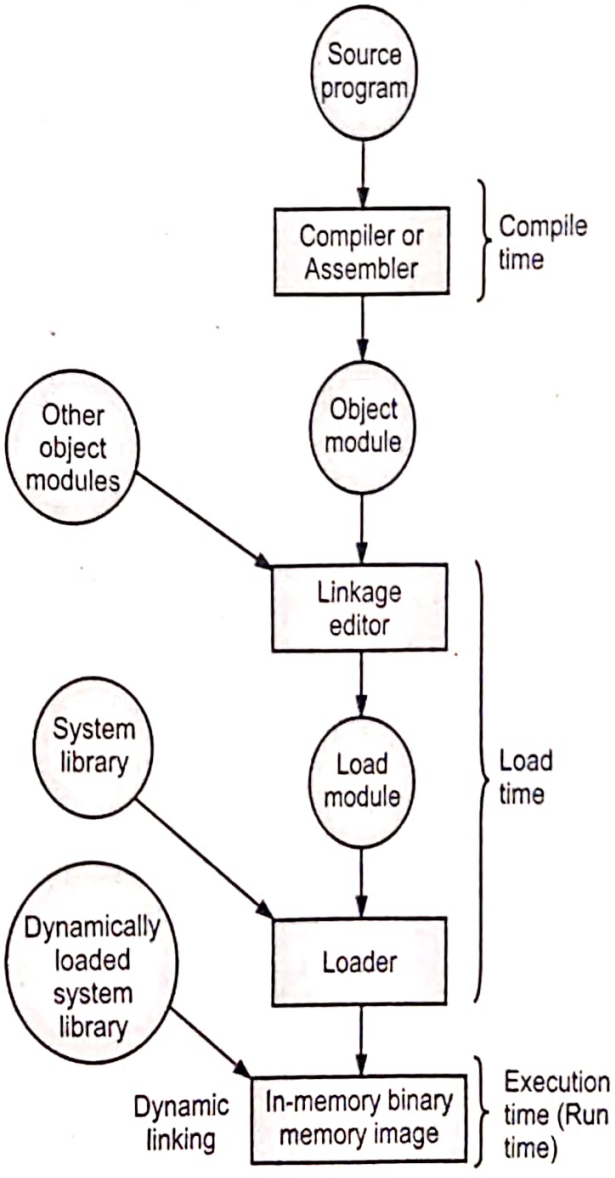

Fig. 7.3: Multistep processing of a user program

7.1.3 Logical versus Physical Address Space [W-18, 22; S-22]

- **Logical Address** is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically, therefore, it is also known as Virtual Address.
- This address is used as a reference to access the physical memory location by CPU. The term Logical Address Space is used for the set of all logical addresses generated by a program's perspective.
- The hardware device called Memory-Management Unit is used for mapping logical address to its corresponding physical address.
- **Physical Address** identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, the logical address must be mapped to the physical address by MMU before they are used.
- The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.

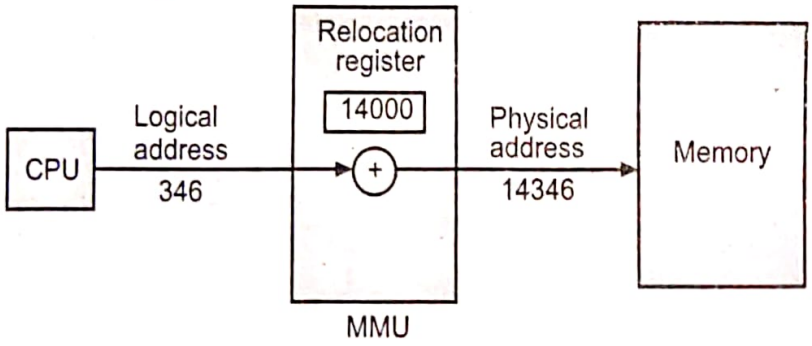

Fig. 7.4: Dynamic relocation using a Relocation Register

Differences between Logical and Physical Address in Operating System:

1. The basic difference between logical and physical address is that logical address is generated by CPU in perspective of a program whereas the physical address is a location that exists in the memory unit.
2. Logical Address Space is the set of all logical addresses generated by CPU for a program whereas the set of all physical address mapped to corresponding logical addresses is called Physical Address Space.
3. The logical address does not exist physically in the memory whereas physical address is a location in the memory that can be accessed physically.
4. Identical logical addresses are generated by Compile-time and Load time address binding methods whereas they differs from each other in run-time address binding method. Please refer this for details.
5. The logical address is generated by the CPU while the program is running whereas the physical address is computed by the Memory Management Unit (MMU).

7.1.4 Dynamic Loading

- For a process to be executed it should be loaded in to the physical memory. The size of the process is limited to the size of the physical memory. Dynamic loading is used to obtain better memory utilization.
- In dynamic loading, the routine or procedure will not be loaded until it is called. Whenever a routine is called, the calling routine first checks whether the called routine is already loaded or not.
- If it is not loaded it cause the loader to load the desired program in to the memory and updates the programs address table to indicate the change. Then the control is passed to newly called routine.

Advantages:

1. It gives better memory utilization.
2. Unused routine is never loaded.
3. It does not need special operating system support.
4. This method is useful when large amount of codes are needed to handle in frequently occurring cases.

7.1.5 Dynamic Linking and Shared Libraries

- Some operating systems support only static linking. In static linking system language libraries are treated like any other object module and are combined by the loader into the binary program image.
- Dynamic linking, in contrast, is similar to dynamic loading. Here, though, linking, rather than loading, is postponed until execution time.
- This feature is usually used with system libraries, such as language subroutine libraries. Without this facility, each program on a system must include a copy of its language library in the executable image. This requirement wastes both disk space and main memory.
- With dynamic linking, a **stub** is included in the image for each library-routine reference.
- The stub is a small piece of code that indicates how to locate the appropriate memory resident library routine or how to load the library if the routine is not already present.
- When the stub is executed, it checks to see whether the needed routine is already in memory. If it is not, the program loads the routine into memory. Either way, the stub replaces itself with the address of the routine and executes the routine.
- Thus, the next time the particular code segment is reached, the library routine is executed directly, incurring to cost for dynamic linking.
- Under dynamic linking, all processes that use a language library execute only one copy of the library code.
- A library may be replaced by a new version, and all programs that reference the library will automatically use the new version.
- Without dynamic linking, all such programs would need to be relinked to gain access the new library, for this reason programs will not accidentally execute new, incompatible versions of libraries, version information is included in both the program and the library.

7.2 SWAPPING [S-18]

- Swapping is a memory management scheme in which any process can be temporarily swapped from main memory to secondary memory so that the main memory can be made available for other processes. It is used to improve main memory utilization. In secondary memory, the place where the swapped-out process is stored is called swap space.
- The purpose of the swapping in operating system is to access the data present in the hard disk and bring it to RAM so that the application programs can use it. The thing to remember is that swapping is used only when data is not present in RAM.
- Although the process of swapping affects the performance of the system, it helps to run larger and more than one process. This is the reason why swapping is also referred to as memory compaction.
- The concept of swapping has divided into two more concepts: Swap-in and Swap-out.
  - Swap-out is a method of removing a process from RAM and adding it to the hard disk.
  - Swap-in is a method of removing a program from a hard disk and putting it back into the main memory or RAM.

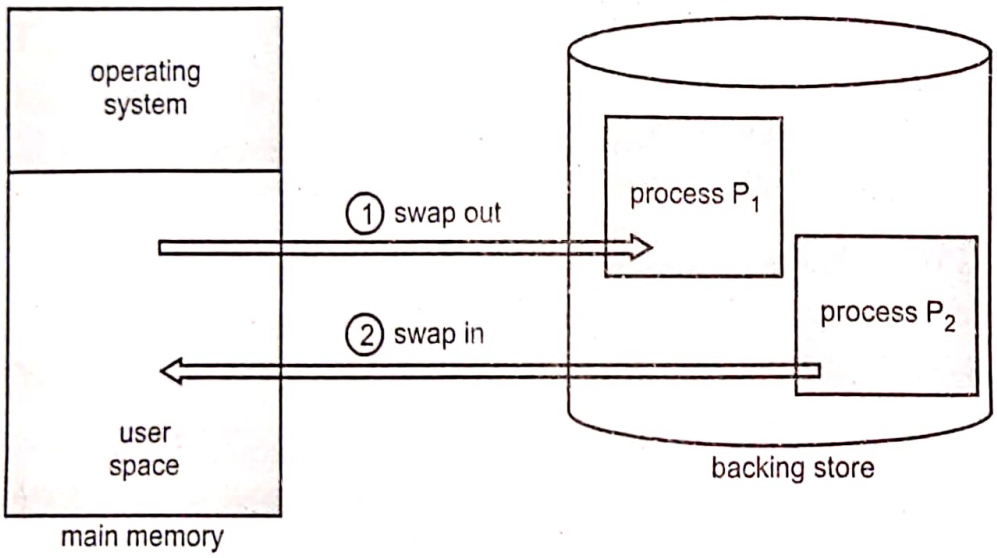

Fig. 7.5: Swapping Method

- **Backing Store:** Swapping requires a backing store. The backing store is commonly a fast drum or disk. It must be large enough to accommodate copies of all memory images for all users, and must provide direct access to these memory images. All memory images are on the backing stores, which are ready to run. Whenever, the CPU scheduler decides to execute a process, it calls the dispatcher. The dispatcher checks to see whether that process is in memory. If not, it swaps out the process currently in memory and swaps in the desired process.
- **Overlapped Swapping:** In this scheme, the objective is to overlap the swapping of one process with the execution of another. Thus, the CPU will not sit idle while swapping is going on.

Example: Suppose the user process's size is 2048KB and is a standard hard disk where swapping has a data transfer rate of 1Mbps. Now we will calculate how long it will take to transfer from main memory to secondary memory.

Solution: User process size is 2048 Kb

Data transfer rate is 1 Mbps = 1024 kbps

$$\text{Time} = \frac{\text{Process size}}{\text{Transfer rate}}$$

$$= 2048 / 1024$$

$$= 2 \text{ seconds}$$

$$= 2000 \text{ milliseconds}$$

Now taking swap-in and swap-out time, the process will take 4000 milliseconds.

Advantages of Swapping:

1. It helps the CPU to manage multiple processes within a single main memory.
2. It helps to create and use virtual memory.
3. Swapping allows the CPU to perform multiple tasks simultaneously. Therefore, processes do not have to wait very long before they are executed.
4. It improves the main memory utilization.

Disadvantages of Swapping:

1. If the computer system loses power, the user may lose all information related to the program in case of substantial swapping activity.
2. If the swapping algorithm is not good, the composite method can increase the number of Page Fault and decrease the overall processing performance.

7.3 CONTIGUOUS MEMORY ALLOCATION [S-23]

- The main memory must accommodate both the operating system and the various user processes. Therefore we need to allocate the parts of the main memory in the most efficient way possible. This section explains one common method, contiguous memory allocation.
- The memory is usually divided into two partitions:
  1. one for the resident operating system.
  2. one for the 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, programmers usually place the operating system in low memory as well. Here, we discuss only the situation where the operating system resides in low memory.
- We usually want several user processes to reside in memory at the same time. We therefore need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory. In this **Contiguous Memory Allocation**, each process is contained in a single contiguous section of memory.

7.3.1 Memory Mapping and Protection [S-19]

- Memory protection means protecting the OS from user process and protecting process from one another. Memory protection is provided by using a re-location register, with a limit register. Re-location register contains the values of smallest physical address and limit register contains range of logical addresses (Re-location = 100040 and limit = 74600).
- The logical address must be less than the limit register; the MMU maps the logical address dynamically by adding the value in re-location register. When the CPU scheduler selects a process for execution, the dispatcher loads the re-location and limit register with correct values as a part of context switch. Since every address generated by the CPU is checked against these register we can protect the OS and other users programs and data from being modified.

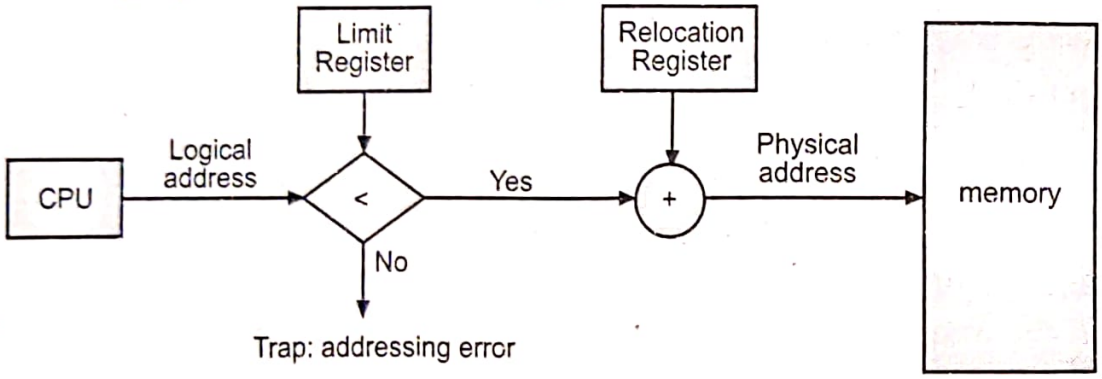

Fig. 7.6: Hardware support for relocation and limit registers

7.3.2 Memory Allocation

- One of the simplest methods for memory allocation is to divide memory in to several fixed partition. Each partition contains exactly one process. The degree of multi-programming depends on the number of partitions.
- In multiple partition method, when a partition is free, process is selected from the input queue and is loaded in to free partition of memory.
- When process terminates, the memory partition becomes available for another process. Batch OS uses the fixed size partition scheme.
- An alternate method is to keep a list of unused (free) memory blocks (holes), and to find a hole of a suitable size whenever a process needs to be loaded into memory.
- The OS keeps a table indicating which part of the memory is free and is occupied. When the process enters the system it will be loaded in to the input queue.
- The OS keeps track of the memory requirement of each process and the amount of memory available and determines which process to allocate the memory.
- When a process requests, the OS searches for large hole for this process, hole is a large block of free memory available.
- If the hole is too large it is split in to two. One part is allocated to the requesting process and other is returned to the set of holes. The set of holes are searched to determine which hole is best to allocate.

- There are three strategies to select a free hole:
  - (i) **First bit:** Allocates first hole that is big enough. This algorithm scans memory from the beginning and selects the first available block that is large enough to hold the process.
  - (ii) **Best bit:** It chooses the hole i.e., closest in size to the request. It allocates the smallest hole i.e., big enough to hold the process.
  - (iii) **Worst fit:** It allocates the largest hole to the process request. It searches for the largest hole in the entire list.

7.3.3 Fragmentation [S-18, 23; W-22]

- Memory fragmentation can be of two types:

1. **Internal Fragmentation :**
   - The phenomenon, in which there is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition, is referred to as Internal Fragmentation.
   - For Example: If there is a block of 50 Kb and if the process requests 40 Kb and if the block is allocated to the process then there will be 10 Kb of memory left.
2. **External Fragmentation:**
   - External Fragmentation exists when there is enough memory space exists to satisfy the request, but it not contiguous i.e., storage is fragmented in to large number of small holes.
   - External Fragmentation may be either minor or a major problem.
   - One solution for over-coming external fragmentation is compaction. The goal is to move all the free memory together to form a large block. Compaction is not possible always. If the relocation is static and is done at load time then compaction is not possible. Compaction is possible if the re-location is dynamic and done at execution time.
   - Another possible solution to the external fragmentation problem is to permit the logical address space of a process to be non-contiguous, thus allowing the process to be allocated physical memory whenever the latter is available.

7.4 PAGING [S-19, 23; W-18]

- Paging is non-contiguous memory allocation technique that permits the physical address space of a process to be non-contiguous. Support for paging is handled by hardware.
- It is used to avoid external fragmentation. Paging avoids the considerable problem of fitting the varying sized memory chunks on to the backing store. When some code or date residing in main memory need to be swapped out, space must be found on backing store.

7.4.1 Basic Method [S-23]

- Physical memory is broken in to fixed sized blocks called frames (f). Logical memory is broken in to blocks of same size called pages (p). When a process is to be executed its pages are loaded in to available frames from backing store. The blocking store is also divided in to fixed-sized blocks of same size as memory frames.

- The following figure shows paging hardware:

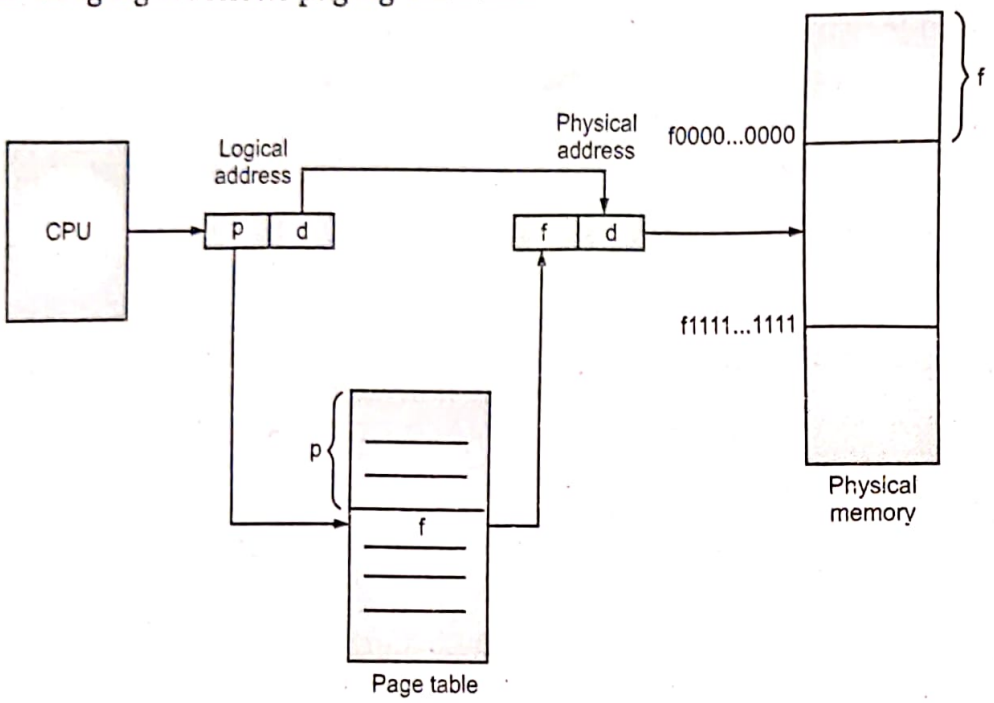

Fig. 7.7: Paging Hardware

- Logical address generated by the CPU is divided in to two parts: page number (p) and page offset (d). The page number (p) is used as index to the page table.
- The page table contains base address of each page in physical memory. This base address is combined with the page offset to define the physical memory i.e., sent to the memory unit.
- The page size is defined by the hardware. The size of a power of 2, varying between 512 bytes and 10Mb per page. If the size of logical address space is 2^m address unit and page size is 2^n, then high order m-n designates the page number and n low order bits represents page offset.

Example:

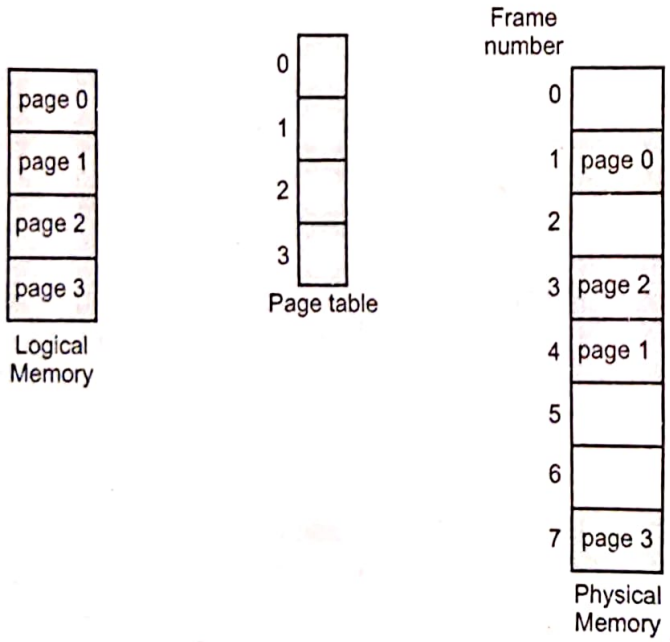

Fig. 7.8: Example for Paging

- To show how to map logical memory into physical memory; consider a page size of 4 bytes and physical memory of 32 bytes (8 pages).
- Logical address 0 is page 0 and offset 0. Page 0 is in frame 5. The logical address 0 maps to physical address 20. [(5*4) + 0].
- Logical address 3 is page 0 and offset 3 maps to physical address 23 [(5*4) + 3].
- Logical address 4 is page 1 and offset 0 and page 1 is mapped to frame 6. So logical address 4 maps to physical address 24 [(6*4) + 0].
- Logical address 13 is page 3 and offset 1 and page 3 is mapped to frame 2. So logical address 13 maps to physical address 9 [(2*4) + 1].

7.4.2 Hardware Support

- The hardware implementation of the page table can be done in several ways:

(i) Page table implemented as dedicated fast registers:

- The simplest method is that the page table is implemented as a set of dedicated registers. These registers must be built with very high speed logic for making paging address translation. Every access to memory must go through the paging map.
- The use of registers for page table is satisfactory if the page table is small.

(ii) Page table implemented in the Main Memory:

- If the page table is large then the use of registers is not visible. So the page table is kept in the main memory and a Page Table Base Register (PTBR) points to the page table. Changing the page table requires only one register which reduces the context switching type.
- The problem with this approach is the time required to access memory location. To access a location [i], first we have to index the page table using PTBR offset. It gives the frame number which is combined with the page offset to produce the actual address. Thus we need two memory accesses for a byte.
- The only solution is to use special, fast, lookup hardware cache called Translation Look- aside Buffer (TLB) or associative register. LB is built with associative register with high speed memory. Each register contains two paths a key and a value.

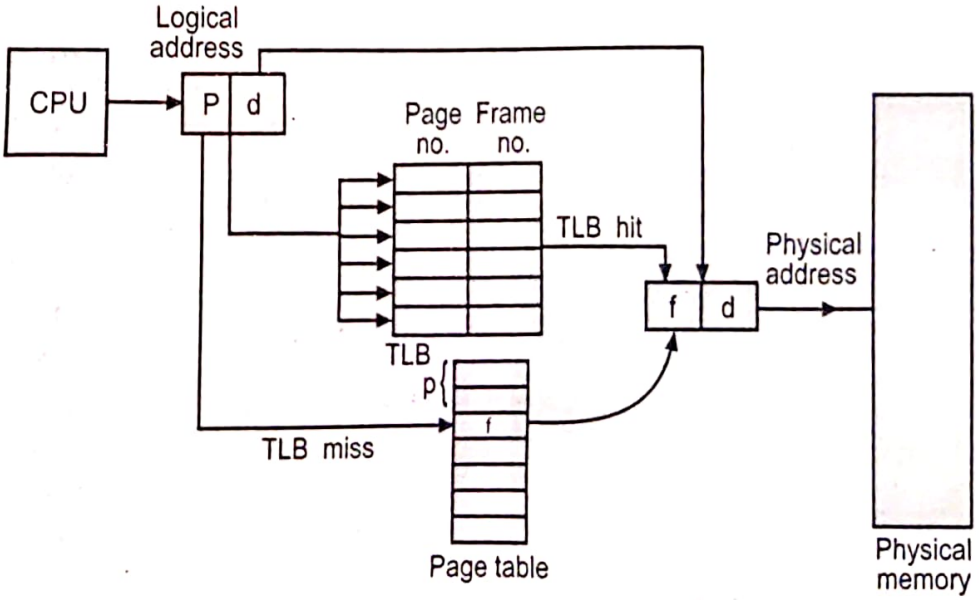

Fig. 7.9: Paging hardware with TLB

- When an associative register is presented with an item, it is compared with all the key values, if found the corresponding value field is return and searching is fast.
- TLB is used with the page table as follows:
  - TLB contains only few page table entries. When a logical address is generated by the CPU, its page number along with the frame number is added to TLB.
  - If the page number is found, its frame memory is used to access the actual memory.
  - If the page number is not in the TLB (TLB miss) the memory reference to the page table is made. When the frame number is obtained, we can use it to access the memory.
  - If the TLB is full of entries the OS must select anyone for replacement. Each time a new page table is selected the TLB must be flushed /erased to ensure that next executing process do not use wrong information.
- The percentage of time that a page number is found in the TLB is called HIT ratio.

7.4.3 Protection

- Memory protection in paged environment is done by protection bits that are associated with each frame these bits are kept in page table.
- One bit can define a page to be read-write or read-only. To find the correct frame number every reference to the memory should go through page table. At the same time physical address is computed. The protection bits can be checked to verify that no writers are made to read-only page. Any attempt to write in to read-only page causes a hardware trap to the OS.
- This approach can be used to provide protection to read-only, read-write or execute-only pages.
- One more bit is generally added to each entry in the page table: a valid-invalid bit.

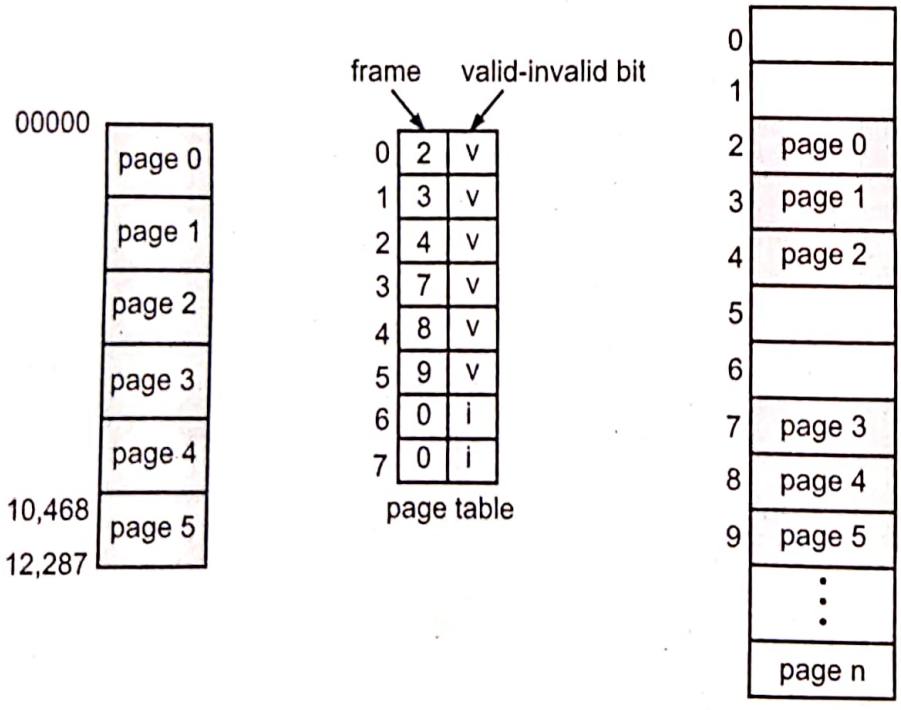

Fig. 7.10: Memory Protection using Valid-invalid bits

- A valid bit indicates that associated page is in the processes logical address space and thus it is a legal or valid page.
- If the bit is invalid, it indicates the page is not in the processes logical addressed space and illegal. Illegal addresses are trapped by using the valid-invalid bit.
- The OS sets this bit for each page to allow or disallow accesses to that page.

7.4.4 Shared Pages

- An advantage of paging is the possibility of sharing common code. This is useful in Timesharing environment.
- For Example, consider a system with 40 users, each executing a text editor. If the text editor is of 150k and data space is 50k, we need (150k+50k)*40 users = 8000k for 40 users.
- If the code is re-entrant, it can be shared. Consider the following Fig. 7.11.

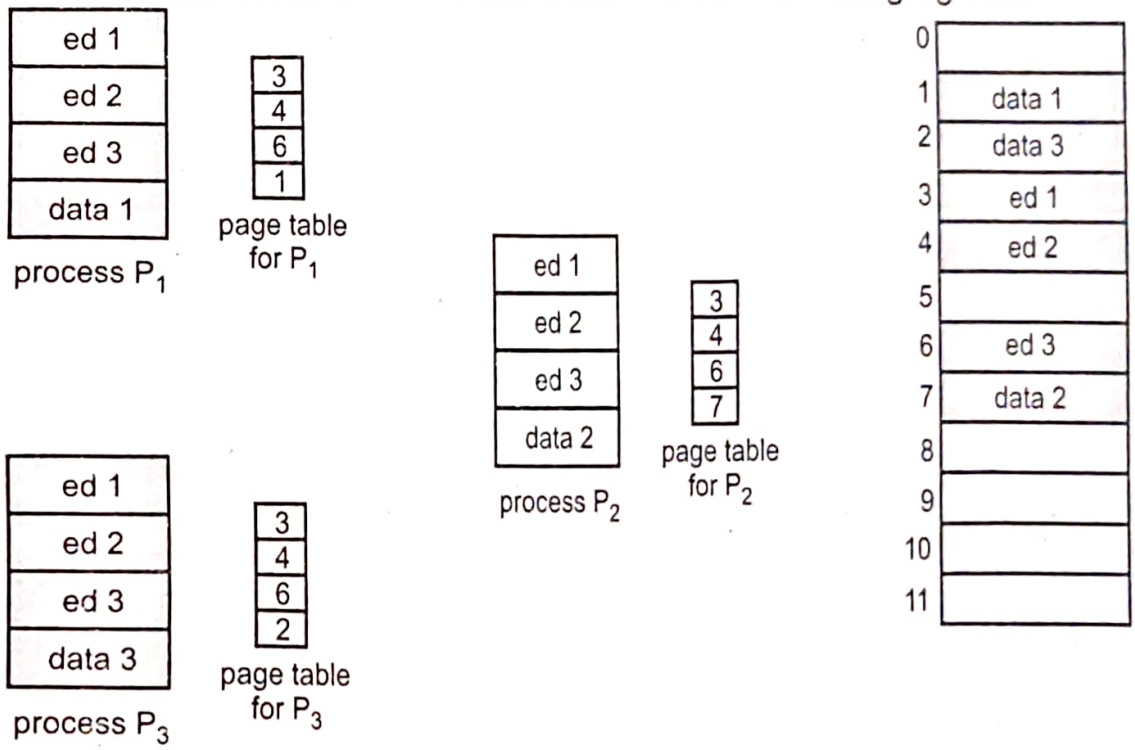

Fig. 7.11: Sharing of code in a Paging

- If the code is re-entrant then it never changes during execution. Thus two or more processes can execute same code at the same time.
- Each process has its own copy of registers and the data of two processes will vary. Only one copy of the editor is kept in physical memory.
- Each user's page table maps to same physical copy of editor but date pages are mapped to different frames. So to support 40 users we need only one copy of editor (150k) plus 40 copies of 50k of data space i.e., only 2150k instead of 8000k.

7.4.5 Advantages and Disadvantages of Paging

Advantages of Paging:

1. It permits physical address space of process to be non-contiguous and support virtual memory concept.
2. It maintains clear separation between user's view of memory and actual physical memory.

3. It does not require any support for dynamic relocation because paging itself is a form of dynamic relocation. Every logical address is bound by paging hardware to physical address.
4. It does not suffer from external fragmentation. Any free frames can be allocated to process that it needs.
5. We can share common code, so memory space is properly utilized.

Disadvantages of paging:

1. Paging suffers from internal fragmentation. Internal fragmentation especially on last page of process; average of 50% on last page.
2. Smaller frames create less fragmentation but increase number of frames and size of page table.
3. Overhead to maintain and update page table.
4. Paging hardware increases the cost of operating system.

7.5 SEGMENTATION

- Like Paging, Segmentation is another non-contiguous memory allocation technique.
- The user's view is mapped onto the physical storage. This mapping permits differentiation between logical memory and physical memory.
- Segmentation is a memory management technique which supports user's view of memory.

7.5.1 Basic Concept

- Most users do not think memory as a linear array of bytes rather the users thinks memory as a collection of variable sized segments which are dedicated to a particular use such as code, data, stack, heap etc.
- A logical address is a collection of segments. Each segment has a name and length. The address specifies both the segment name and the offset within the segments.
- The users specify address by using two quantities: a segment name and an offset. For simplicity the segments are numbered and referred by a segment number. So the logical address consists of <segment number, offset>.

7.5.2 Hardware

- Although the programmer can now refer to objects in the program by a two dimensional address, the actual physical memory is still a one dimensional sequence of bytes. Thus, we must define an implementation to map 2D user defined address in to 1D physical address. This mapping is affected by a segment table. Each entry in the segment table has a segment base and segment limit. The segment base contains the starting physical address where the segment resides and limit specifies the length of the segment.

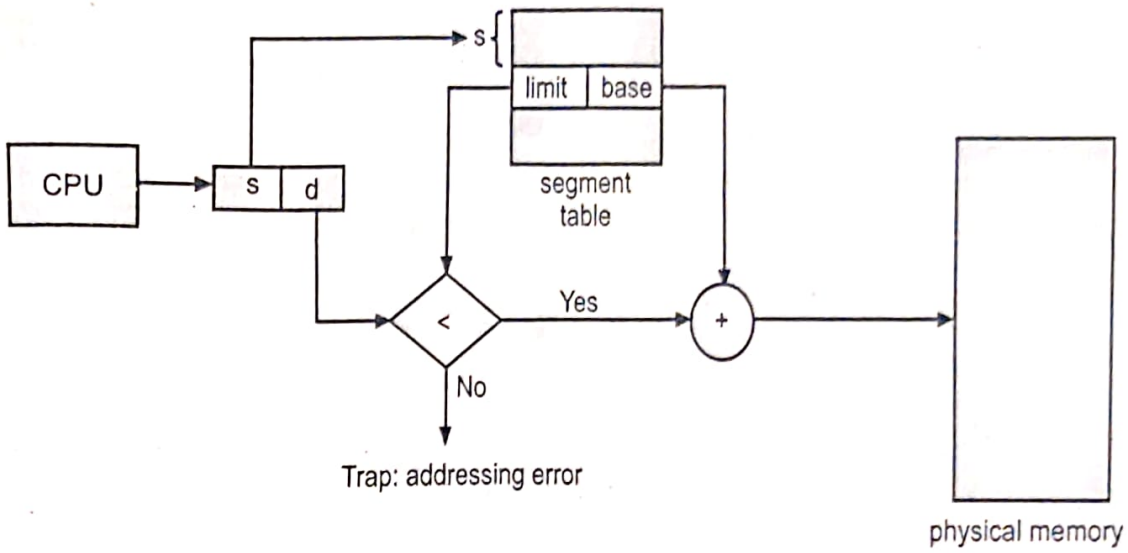

Fig. 7.12: Segmentation Hardware

- The use of segment table is shown in Fig. 7.12: Logical address consists of two parts: segment number 's' and an offset 'd' to that segment.
- The segment number is used as an index to segment table.
- The offset 'd' must be in between 0 and limit, if not an error is reported to OS. If legal the offset is added to the base to generate the actual physical address.
- The segment table is an array of base limit register pairs.

7.6 VIRTUAL MEMORY MANAGEMENT

- Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory.

7.6.1 Background [S-19]

- Virtual memory is a technique that allows the execution of processes that may not be completely in the physical memory.
- In this technique, the operating system loads only those parts of program in memory that are currently needed for execution of the process. The rest part of the disk and isloaded into the memory only when needed.
- Virtual memory is commonly implemented by demand paging or segmentation. Out of these two ways, demand paging is commonly used as it is easier to implement.
- Virtual memory provides following benefits:
  - Only part of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Allows address spaces to be shared by several processes.
  - Allows for more efficient process creation.

7.6.2 Demand Paging [W-22]

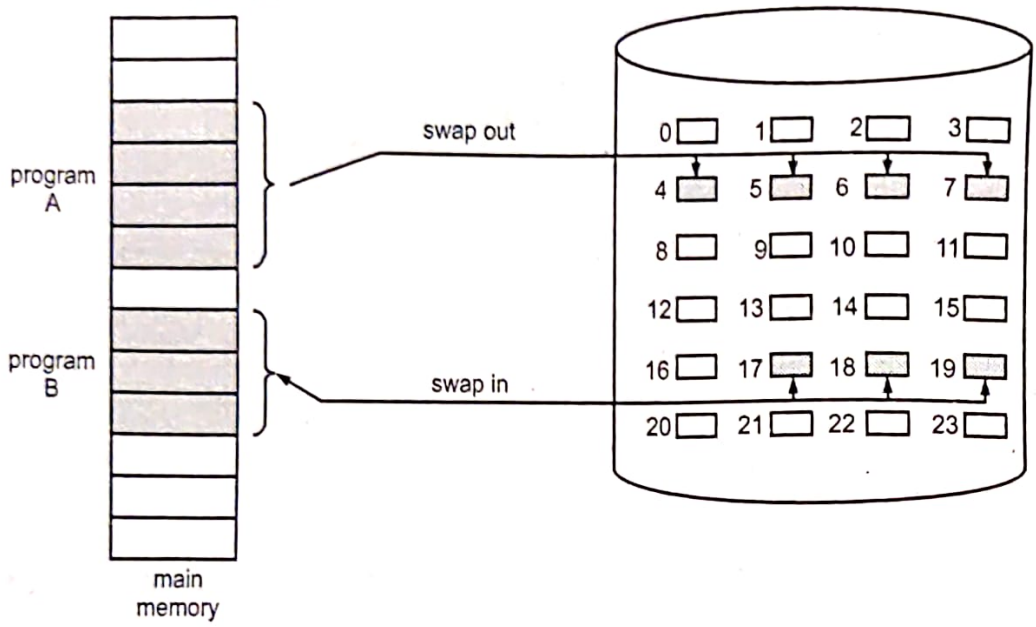

Fig. 7.13: Transfer of a paged memory to contiguous disk space

- The basic idea behind paging is that when a process is swapped in, the pager only loads into memory those pages that it expects the process to need (right away).
- Pages that are not loaded into memory are marked as invalid in the page table, using the invalid bit. (The rest of the page table entry may either be blank or contain information about where to find the swapped-out page on the hard drive).
- If the process only ever accesses pages that are loaded in memory (memory resident pages), then the process runs exactly as if all the pages were loaded in to memory.

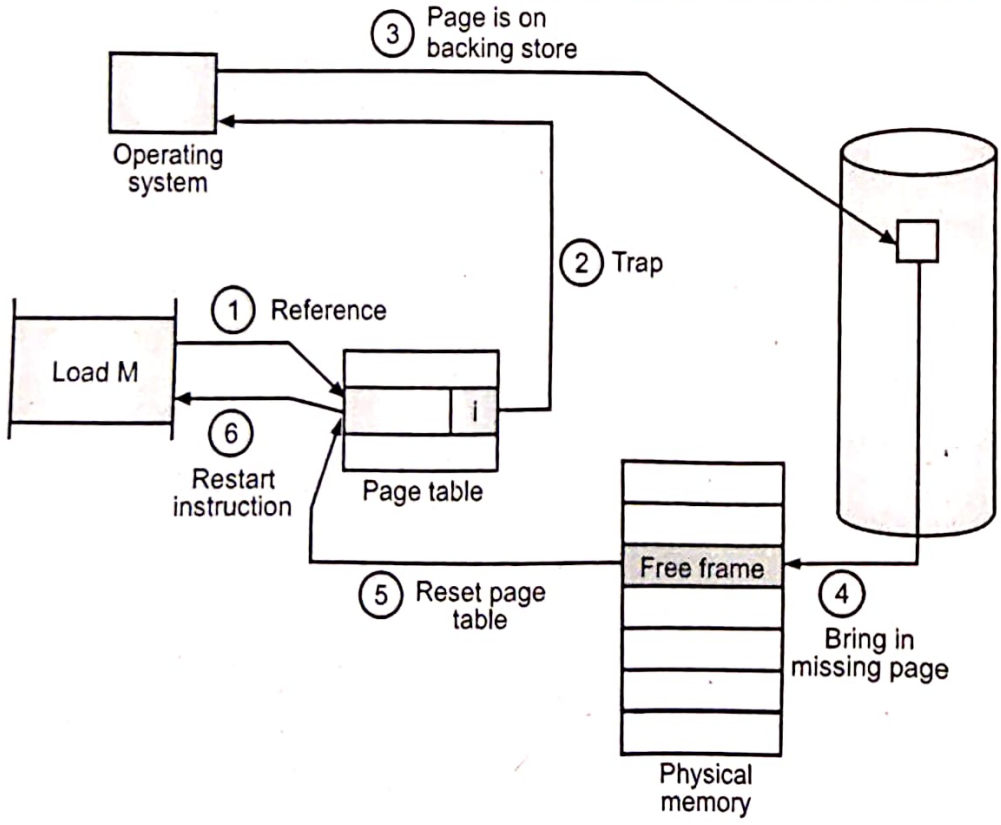

Fig. 7.14: Steps for handling Page Fault

- **Page fault:** 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. The fault notifies the operating system that it must locate the data in virtual memory, then transfer it from the storage device, such as an HDD or SSD, to the system RAM.
- **Steps for handling Page Fault:**
  1. Check the location of the referenced page in the Page Map Table(PMT).
  2. If a page fault occured, call on the operating system to fix it.
  3. Using the frame replacement algorithm, find the frame location.
  4. Read the data from disk to memory.
  5. Update the page map table for the process.The instruction that caused the page fault is restarted when the process resumes execution.

7.6.3 Performance of Demand Paging [S-19]

- Obviously there is some slowdown and performance hit whenever a page fault occurs and the system has to go get it from memory, but just how big a hit is it exactly?
- There are many steps that occur when servicing a page fault and some of the steps are optional or variable. Suppose that a normal memory access requires 200 nano-seconds, and that servicing a page fault takes 8 milliseconds. (8,000,000 nanoseconds, or 40,000 times a normal memory access).
- With a **page fault rate** of p, (on a scale from 0 to 1), the effective access time is now:
- Effective access time = p * time taken to access memory in page fault+ (1-p)* time taken to access memory

  = p * 8000000 + (1 – p) * (200)

  = 200 + 7,999,800 * p

  which depends heavily on p!
- Even if only one access in 1000 causes a page fault, the effective access time drops from 200 nanoseconds to 8.2 microseconds, a slowdown of a factor of 40 times. In order to keep the slowdown less than 10%, the page fault rate must be less than 0.0000025, or one in 399,990 accesses.
- The swap space is faster to access than the regular file system, because it does not have to go through the whole directory structure. For this reason some systems will transfer an entire process from the file system to swap space before starting up the process, so that future paging all occurs from the (relatively) faster swap space.

7.6.4 Page Replacement Algorithms

- When a page fault occurs, the operating system has to choose a page to remove from memory to make room for the page that has to be brought in. This is known as **Page Replacement**.
- Page replacement algorithms should have the lowest page fault rate.
- 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 memory after the first reference; hence, the immediately following references will not cause a page fault.

- As the number of frames increases, the numbers of page faults are minimized.
- Page replacement is basic to demand paging. It completes the separation between logical memory and physical memory. With this mechanism an enormous virtual memory can be provided for programmers on a smaller physical memory.
- Every operating system usually has its own page replacement scheme.

7.6.4.1 FIFO (First In First Out) [S-18, 22, 23; W-18, 22]

- The simplest page replacement algorithm is a FIFO.
- A FIFO replacement algorithm associates with each page the time when that page was brought into the memory. When a page must be replaced, the oldest page is chosen.
- FIFO queue is created to hold all pages in the 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.
- The FIFO page replacement algorithm is easy to understand and program.
- Its performance is not always good.
- For example consider the following reference string:

  7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0,1,7,0,1
- Our three frames are initially empty. The first three references (7, 0, 1) cause page faults and are brought into these empty frames.
- The next reference (2) replaces page 7, because page 7 was brought in first. Since, 0 is the next reference and 0 is already in memory, we have no fault for this reference.
- The first reference to 3 results in page 0 being replaced, since it was the first of the three pages in memory (0, 1 and 2) to be brought in.
- Because of this replacement, the next reference 0 will cause page fault. Page 1 is then replaced by page 0. This process continues. There are 15 faults altogether.

Reference String:

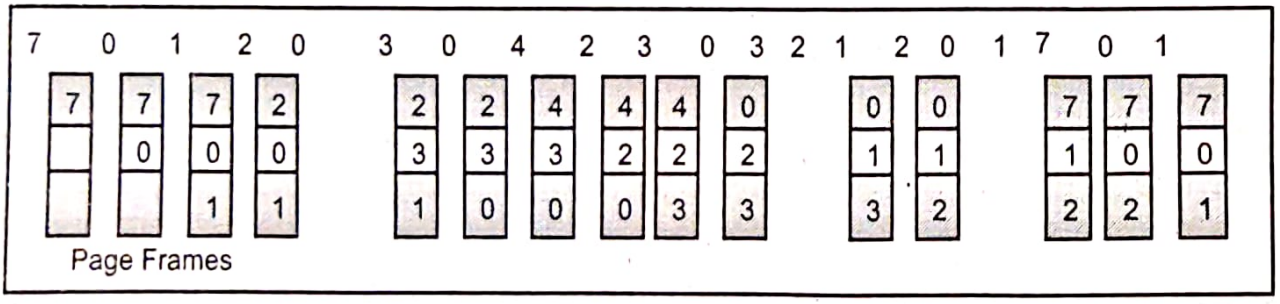

Fig. 7.15: Reference string in FIFO

- The number of faults for four frames (10) is greater than the number of faults for three frames (9).
- This most unexpected result is known as Belady's anomaly. For some page replacement algorithms, the page fault rate may increase as the number of allocated frames increases.

7.6.4.2 Optimal Page Replacement (OPT) [W-18; S-23]

- An optimal page replacement algorithm has the lowest page fault rate of all algorithms and does not suffer from Belady's anomaly.

- Optimal replacement algorithm state that replaces the page which will not be used for the longest period of time.
- For example, consider the following reference string.

  7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
- The first three reference cause faults which fill the three empty frames.
- The reference to page 2 replaces page 7, because 7 will 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.
- With only nine page faults, optimal replacement is much better than a FIFO algorithm, which had 15 faults. The optimal page replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.

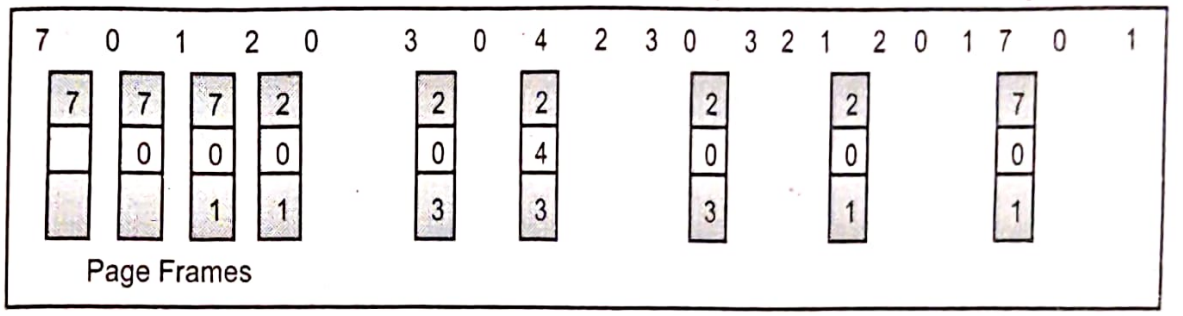

Fig. 7.16: Optimal Page Replacement Algorithm

7.6.4.3 LRU (Last Recently Used) [S-22, 23]

- If we use the recent past as an approximation of the near future, then LRU replaces the page which has not been used for the longest period of time. This is the least recently used algorithm.
- LRU replacement associates with each page the time of its last use. When a page is to be replaced, LRU chooses that page which has not been used for the longest period or time.
- For example, consider the following reference string

  7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
- By applying LRU, the first five faults are same as the optimal replacement. When the reference to page 4 occurs, LRU sees out of the three frames in memory, page 2 was used least recently.
- The most recently used page is page 0, and just before that page 3 was used. Thus, LRU replaces page 2, not knowing that it is about to be used.
- When fault for page 2 occurs, LRU replaces page 3. Since of the three pages in memory (0, 3, 4) page 3 is the least recently used. No. of page faults = 12.

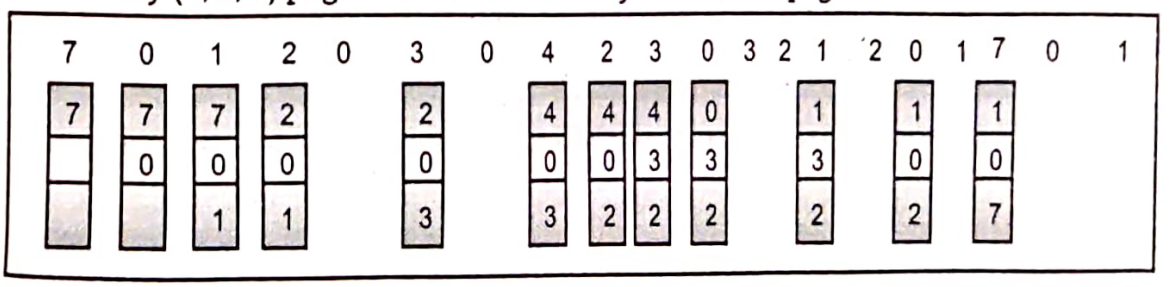

Fig. 7.17: LRU

7.6.4.4 Second-Chance Replacement Algorithm

- The basic algorithm of second-chance replacement is a FIFO replacement algorithm.
- When a page has been selected, however, we inspect its reference bit. If the value is 0, we proceed to replace this page, but if the reference bit is set to 1, we give the page a second chance and move on to select the next FIFO page.
- When a page gets a second chance, its reference bit is cleared and its arrival time is reset to the current time for this reason, a page that is given a second chance will not be replaced until all other pages have been replaced.
- If a page is used often enough to keep its reference bit set, it will never be replaced.
- One way to implement the second-chance algorithm is as a circular queue.
- A pointer indicates which page is to be replaced next. When a frame is needed, the pointer advances until it finds a page with a 0 reference bit.
- Following Fig. 7.18 shows Second-chance (clock) Page-Replacement Algorithm.

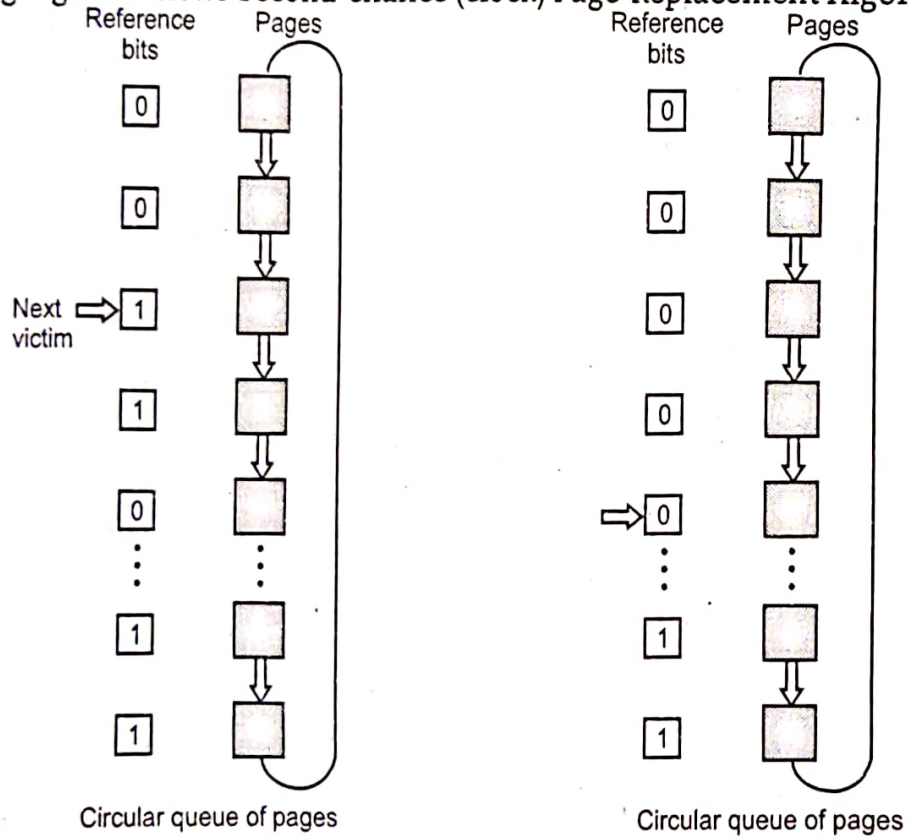

Fig. 7.18: Second-chance (clock) Page-Replacement Algorithm

- As it advances, it clears the reference bits as shown in Fig. 7.18.
- Once, a victim page is found, the page is replaced, and the new page is inserted in the circular queue in that position.
- Second-chance replacement degenerates to FIFO replacement if all bits are set.

Enhanced Second-Chance Algorithm:

- We can enhance the second-chance algorithm by considering the reference bit and the modify bit as an ordered pair. With these two bits, we have the following four possible classes:
  1. **(0, 0) neither recently used nor modified:** best page to replace.
  2. **(0, 1) not recently used but modified:** Not quite as good, because the page will need to be written out before replacement.

3. **(1, 0) recently used but clean:** Probably will be used again soon.
4. **(1, 1) recently used and modified:** Probably will be used again soon and the page will need to be written out to disk before it can be replaced.

7.6.4.5 MFU

[S-18, W-22]

- The **Most Frequently Used (MFU) page-replacement algorithm** is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.
- As you might expect, neither MFU nor LFU page replacement is common. The implementation of these (MFU and LFU) algorithms is expensive and they do not approximate OPT replacement well.

7.6.4.6 LFU

- The **Least Frequently Used (LFU) page-replacement algorithm** requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count.
- A problem arises, however, when a page is used heavily during the initial phase of a process but then it is never used again.
- Since, it was used heavily; it has a large count and remains in memory even though it is no longer needed. One solution of this problem is to shift the counts right by 1 bit at regular intervals, forming an exponentially decaying average usage count.

Solved Examples of Previous Exam

Example 1: Consider the following page reference string.

7,5,4,9,4,7,8,5,2,3,4,7,9,7,4

Find the number of page fault for the following algorithm with 3 frames.

(i) LFU, (ii) FIFO.

Solution:

(i) **LFU:** Initially three memory frame slots are empty, so the first three memory references (7, 5, 4) cause page faults and are brought into these empty frames.

In Least Frequently used memory technique the page which has least memory count is replaced with the new page which has to brought in.

So initially 7 has page count 1, 5 has page count 1 also 4 has page count 1 when new page 9 comes it will be replaced with 7 because all has same count so according to first in first out 7 will be replaced with 9, again 4 is the page references which is already in memory so no page fault will occur. And page 4 has count 2 now. Similarly 7 page reference will get replace with 5 because it is the page which is not accessed recently.

7	5	4	9		7	8	5	2	3	4	7	9
7	7	7	9		9	8	5	5	5	5		9
	5	5	5		7	7	7	7	7	7		7
		4	4		4	4	4	2	3	4		4
1	2	3	4		5	6	7	8	9	10	11	12

So on all pages will be mapped to memory frames and total 11 page faults will occur using LFU page replacement Technique

(ii) **FIFO:** Our three frames are initially empty. The first three references (7, 5, 4) cause page faults and are brought into these empty frames.
The next reference (9) replaces page 7, because page 1 was brought in first. Since, 4 is the next reference and 4 is already in memory, we have no fault for this reference.
Page 7 will be replaced with 5 since 5 is the oldest page in frame.
So this process continues. There are 12 faults altogether.

Reference String:

7	5	4	9		7	8	5	2	3	4	7	9
7	7	7	9		9	9	5	5	5	4	4	4
	5	5	5		7	7	7	2	2	2	7	7
		4	4		4	8	8	8	3	3	3	9
1	2	3	4		5	6	7	8	9	10	11	12

Using FIFO page fault algorithm total 12 page faults will be occurred.

Example 2: Consider the following page reference string.
1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 3
Find the number of page fault for the following algorithm with 3 frames.
(i) FIFO, (ii) LRU.

Solution: (i) FIFO:

1	2	3	4	2	1	5	6	2	1	3
1	1	1	4	4	4	6	6	6	6	3
	2	2	2	1	1	1	2	2	2	2
		3	3	3	5	5	5	1	1	1
1	2	3	4		1	5	6	2	1	3

Using FIFO page fault algorithm the page will be replace with the new page. So total 10 page faults will be occurred.
Our three frames are initially empty. The first three references (1, 2, 3) cause page faults and are brought into these empty frames.
The next reference (4) replaces page 1, because page 1 was brought in first. Since, 2 is the next reference and 2 is already in memory, we have no fault for this reference.
Page 1 will be replaced with 2 since 2 is the oldest page in frame.
So this process continues. There are 10 faults altogether.
Reference String: Using FIFO page fault algorithm total 10 page faults will be occurred.

(ii) LRU:

1	2	3	4	2	1	5	6	2	1	3
1	1	1	4	4	4	5	5	5	1	1
	2	2	2	2	2	2	6	6	6	3
		3	3	1	1	1	1	2	2	2
1	2	3	4		1	5	6	2	1	3

Total number of page faults = 10.

Reference String: Our three frames are initially empty. The first three references (1, 2, 3) cause page faults and are brought into these empty frames.

The next reference (4) replaces page 1, because page 1 was brought in first. Since, 2 is the next reference and 2 is already in memory, we have no fault for this reference.

Page 1 will be replaced with 3 since 2 is the page which has been accessed in frame. So this process continues. There are 10 faults altogether.

Example 3: Consider the following page reference string: [W-22]

4, 6, 7, 8, 4, 6, 9, 6, 7, 8, 4, 6, 7, 9

The number of frames is 3. Show page trace and calculate page fault for the following page replacement schemes:

(i) FIFO, (ii) LRU

Solution: (i) FIFO

Reference String: 4, 6, 7, 8, 4, 6, 9, 6, 7, 8, 4, 6, 7, 9

4	6	7	8	4	6	9	6	7	8	4	6	7	9
4	4	4	8	8	8	9		9	9	4	4	4	9
	6	6	6	4	4	4		7	7	7	6	6	6
		7	7	7	6	6		6	8	8	8	7	7
1	2	3	4	5	6	7		8	9	10	11	12	13

So on all pages will be mapped to memory frames and total 13 page faults will occurs using FIFO page replacement Technique.

(ii) LRU:

Reference String:

4	6	7	8	4	6	9	6	7	8	4	6	7	9
4	4	4	8	8	8	9		9	8	8	8	7	7
	6	6	6	4	4	4		7	7	7	6	6	9
		7	7	7	6	6		6	6	4	4	4	4
1	2	3	4	5	6	7		8	9	10	11	12	13

Using LRU Page Replacement Algorithm page fault algorithm total 13 page faults will be occurred.

Summary

- Computer memory can be defined as a collection of some data represented in the binary format.
- Memory management is the process of controlling and coordinating computer memory. This process assign portions called blocks to various running programs to optimize overall system performance.
- The memory-management algorithms (contiguous allocation, paging, segmentation, and combinations of paging and segmentation) differ in many aspects.
- In comparing different memory-management strategies, we use the following considerations:
- **Hardware support:** A simple base register or a base – limit register pair is sufficient for the single- and multiple-partition schemes, whereas paging and segmentation need mapping tables to define the address map.
- **Performance:** If memory-management algorithm becomes more complex, the time required to map a logical address to a physical address increases.
- **Fragmentation** is a multi-programmed system that performs more efficiently if it has a higher level of multiprogramming. For a given set of processes, we can increase the multiprogramming level only by packing more processes into memory. To accomplish this task, we must reduce memory waste, or fragmentation. Systems with fixed-sized allocation units, such as the single-partition scheme and paging, suffer from internal fragmentation. Systems with variable-sized allocation units, such as the multiple-partition scheme and segmentation, suffer from external fragmentation.
- **Relocation:** One solution to the external-fragmentation problem is compaction. Compaction involves shifting a program in memory in such a way that the program does not notice the change.
- **Swapping:** Processes are copied from main memory to a backing store and later are copied back to main memory.
- **Sharing:** Another means of increasing the multiprogramming level is to share code and data among different processes. Sharing generally requires that either paging or segmentation be used to provide small packets of information (pages or segments) that can be shared. Sharing is a means of running many processes with a limited amount of memory, but shared programs and data must be designed carefully.
- **Protection:** If paging or segmentation is provided, different sections of a user program can be declared execute-only, read-only, or read – write.
- **Virtual Memory** is a storage scheme that provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory.
- The basic idea behind paging is that when a process is swapped in, the pager only loads into memory those pages that it expects the process to need.

- When a page fault occurs, the operating system has to choose a page to remove from memory to make room for the page that has to be brought in. This is known as Page Replacement.
- The process of replacement is sometimes called swap out or write to disk.
- **Types of Page Replacement Algorithms:** Optimal Page Replacement (OPT) Algorithm, First-in-First-Out (FIFO), Least recent used (LRU), Second chance page replacement, MFU, LFU.

Check Your Understanding

1. In which of the following page replacement policies Belady's anomaly occurs?
   (a) FIFO (b) OPT
   (c) MFU (d) program status word
2. _____ allocates the largest hole (free fragment) available in the memory.
   (a) Best fit (b) Worse Fit
   (c) First fit (d) None of the above
3. Which one of the following is the address generated by CPU?
   (a) physical address (b) absolute address
   (c) logical address (d) none of the mentioned
4. Run time mapping from virtual to physical address is done by _____.
   (a) Memory management unit (b) CPU
   (c) PCI (d) None of the mentioned
5. Memory management technique in which system stores and retrieves data from secondary storage for use in main memory is called?
   (a) fragmentation (b) paging
   (c) mapping (d) none of the mentioned
6. The address of a page table in memory is pointed by _____.
   (a) stack pointer (b) page table base register
   (c) page register (d) program counter
7. Program always deals with _____.
   (a) logical address (b) absolute address
   (c) physical address (d) relative address
8. The page table contains _____.
   (a) base address of each page in physical memory
   (b) page offset
   (c) page size
   (d) none of the mentioned

9. What is compaction?
   (a) a technique for overcoming internal fragmentation.
   (b) a paging technique.
   (c) a technique for overcoming external fragmentation.
   (d) a technique for overcoming fatal error.
10. Operating System maintains the page table for _____.
   (a) each process (b) each thread
   (c) each instruction (d) each address

Answers

1. (a)	2. (b)	3. (c)	4. (a)	5. (b)	6. (b)	7. (a)	8. (a)	9. (c)	10. (a)

Practice Questions

Q.I Answer the following questions in short:

1. What is meant by memory management?
2. What is swapping?
3. Describe the basic concept of paging.
4. What is segmentation?
5. What is external fragmentation?
6. Define Belady's Anomaly.

Q.II Answer the following questions:

1. Explain page replacement algorithms with their advantages and disadvantages.
2. Define NRU, MFU, LFU page replacement algorithms.
3. Explain the term virtual memory in detail.
4. Explain FIFO, LRU page replacement algorithms for the reference string 7 0 1 2 0 3 0 4 2 3 1 0 3.
5. With the help of diagram describe demand paging.
6. Describe second chance algorithm with example.
7. State advantages and disadvantages of segmentation.
8. Differentiate between Internal and External fragmentation.
9. Explain optional page replacement algorithm with example.
10. Consider the given reference string: 2 3 2 1 5 2 4 5 3 2 5 2
    Calculate the page fault for number of frames = 3 for the following algorithms.
    (i) LRU
    (ii) FIFO
11. Compare paging and segmentation.
12. What is meant by memory partition?

13. Consider the given page reference string: 4, 3, 2, 1, 4, 3, 5, 4, 3, 2, 1, 5.
The number of frames is 3. Show page trace and calculate page faults for the following page replacement schemes:
(i) FIFO
(ii) MFU

Q.III Define terms:

1. Fragmentation.
2. Page fault.
3. Page table.
4. Dynamic loading.
5. Dynamic linking.
6. Swap time.
7. Logical address.
8. Physical address.

Previous Exam Questions

Summer 2018

1. Define Swap Time. [2 M]

Ans. Refer to Section 7.2.

2. Consider the following page reference string: [4 M]
7, 5, 4, 9, 4, 7, 8, 5, 3, 4, 7, 9, 7, 4
Find the number of page fault for the following algorithm with 3 frames:
(i) FIFO
(ii) MFU

Ans. Refer to Section 7.6. 3.1 and 7.6.3.5.

3. What is fragmentation? Explain types of fragmentation in details. [4 M]

Ans. Refer to Section 7.3.3.

Winter 2018

1. What is meant by Address Binding? [2 M]

Ans. Refer to Section 7.1.2.

2. What do you mean by Paging? List the advantages and disadvantages of Paging. [4 M]

Ans. Refer to Section 7.4.

3. Define the terms: [4 M]
(i) Logical Address
(ii) Physical Address.

Ans. Refer to Section 7.3.1.

4. Consider the following page reference string:

   9, 2, 3, 4, 2, 5, 2, 6, 4, 5, 2, 5, 4, 3, 4, 2, 3, 9, 2, 3

   The number of frames is 4. Calculate the page faults for the following page replacement schemes:

   (i) FIFO

   (ii) Optimal **[4 M]**

Ans. Refer to Section 7.6.3.1 and 7.6.3.2.

Summer 2019

1. What is demand paging? **[2 M]**

Ans. Refer to Section 7.6.1.

2. Write the steps of calculate the physical address by operating system. Explain with example. **[4 M]**

Ans. Refer to Section 7.3.1.

3. What is a page fault? Explain the different steps in handling a page fault. **[4 M]**

Ans. Refer to Section 7.6.2.

■■■