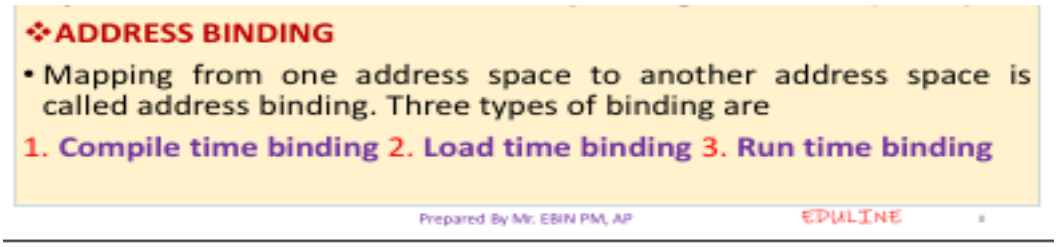
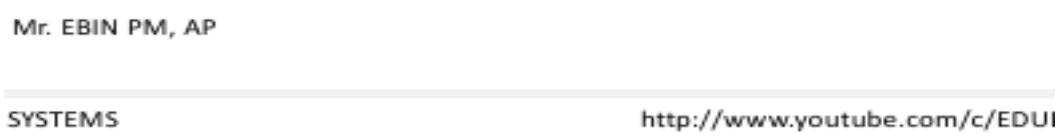
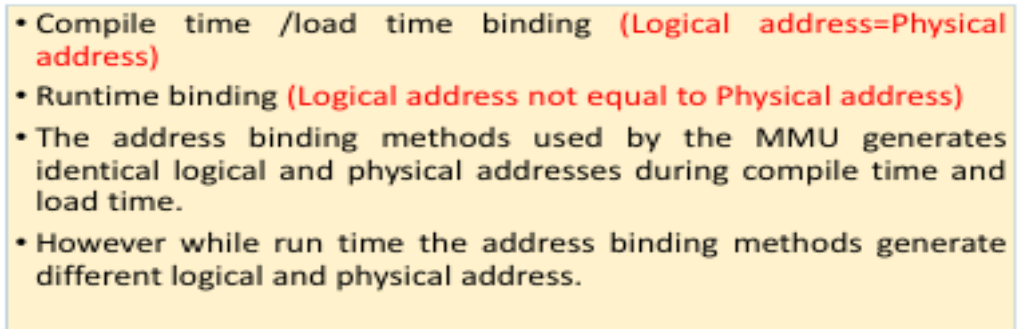
Solved Questions Module-4

1. Differentiate between compile time and load time address binding.







If you know at compile time where the process will reside in memory, then absolute code can be generated at compile time - compile time binding

If it is not known at compile time where the process will reside in memory, then the compiler must generate relocatable code. In this case, final binding is delayed until load time. - load time binding.

1.5 marks each

2. Consider a logical address space of 256 pages with a 4-KB page size, mapped onto a physical memory of 64 frames. Find the

(i) number of bits in the logical address

(ii) number of bits in the physical address

(iii) number of bits in the offset part of logical address.

Logical memory size = 256 * 4KB = 2^20 B. Logical address has 20 bits.

Physical memory size = 64 * 4KB = 2^18 B. Physical address has 18 bits.

Offset - depends upon page size. Hence 4KB = 2^12 B. So offset = 12 bits

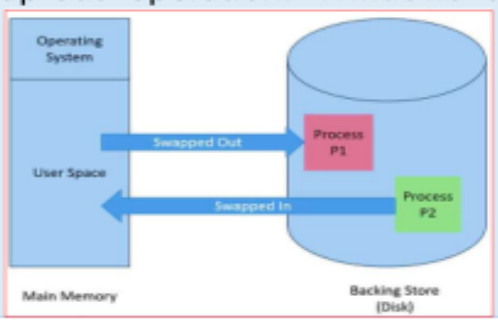
1 mark each

3. How does swapping result in better memory management?

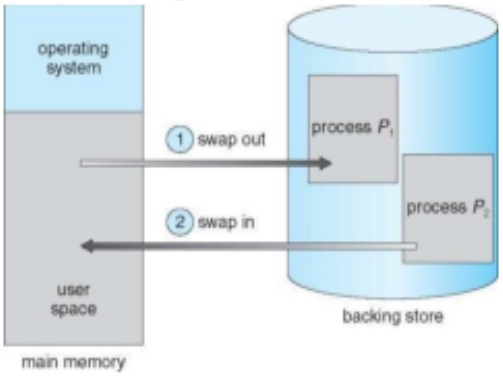
SWAPPING

- Swapping is a mechanism in which a process can be swapped temporarily out of main memory or move to secondary storage(disk) and make that memory available to other processes.
- At some later time, the system swaps back the processes from the secondary storage to main memory.
- Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel
- Swapping may happen in the case of Round Robin scheduling. A process is swapped out when its time quantum finishes and later it is brought in to the memory for continued execution

- This may also happen when it is desired to place a high priority process in the memory. A lower priority process may be swapped out so that higher priority process may be loaded and executed.
- Swapping increases the OS overhead due to the need to perform swap-in and swap-out operation. Windows and UNIX perform 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
- Swapping variant used for priority-based scheduling algorithms are called roll out, roll in- lower-priority process is swapped out so higher-priority process can be loaded and executed.
- 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.
- The system maintains a ready queue consisting of all processes whose memory images are on the backing store or in memory and are ready to run.
- Whenever the CPU scheduler decides to execute a process, it calls the dispatcher.
- The dispatcher checks to see whether the next process in the queue is in memory. If it is not, and if there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process. It then reloads registers and transfers control to the selected process.



Swapping of two processes using a disk as a backing store.

- The context-switch time in such a swapping system is fairly high.
- The major part of the swap time is transfer time. The total transfer time is directly proportional to the amount of memory swapped.
- Other constraints as well on swapping
  - Pending I/O – can't swap out as I/O would occur to wrong process
  - Or always transfer I/O to kernel space, then to I/O device
    - Known as **double buffering**, adds overhead
- Standard swapping not used in modern operating systems, but modified version is common
  - Swap only when free memory extremely low

4. Explain the concept of virtual memory. Write one memory management scheme which supports virtual memory.

VIRTUAL MEMORY

- A computer can address more memory than the amount of physically installed on the system. This extra memory is actually called virtual memory and it is a section of hard disk that has setup to emulate the computers RAM
- Virtual memory is implemented using secondary storage to augment the main memory.
- Programs can be larger than the physical memory. Instead of holding the entire program , main memory hold only a portion of the program which is currently being executed.

- Virtual Memory enhances the CPU utilization and through put by executing as many programs as possible, simultaneously.

➢ Virtual memory serves two purposes:

❖ First it allows us to extend the use of physical memory by using Disk.

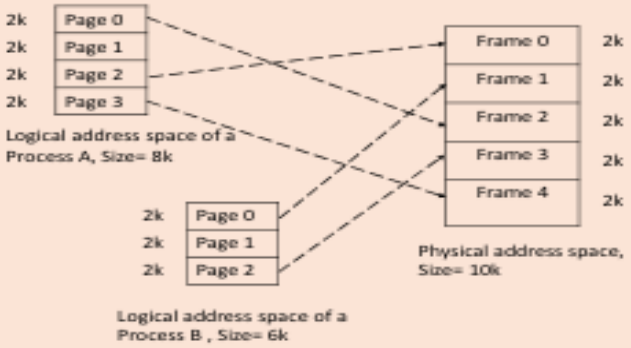
❖ Second , it allows us to have memory protection, because each virtual address is translated to a physical address.

- Two most common methods of implementing virtual storage are **Paging** and **Segmentation**

PAGING

- Paging is a memory management technique.
- In this approach, physical memory is divided in to fixed sized block called frames and logical memory is also divided in to the fixed sized blocks called pages.
- The size of the page is same as that of frame.
- The key idea of this method is to place the pages of a process in to the available frames of memory , whenever this process is to be executed.

❖Paging Concept



❖ADDRESS TRANSLATION

- It is done by mapping. Logical address is mapped to physical address using a page table
- Every address generated by the CPU is divided into two parts
  page number (p) and pageoffset (d).
- Physical address is represented by two parts:
  Frame number(f) and 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.

Mr. EBIN PM, AP

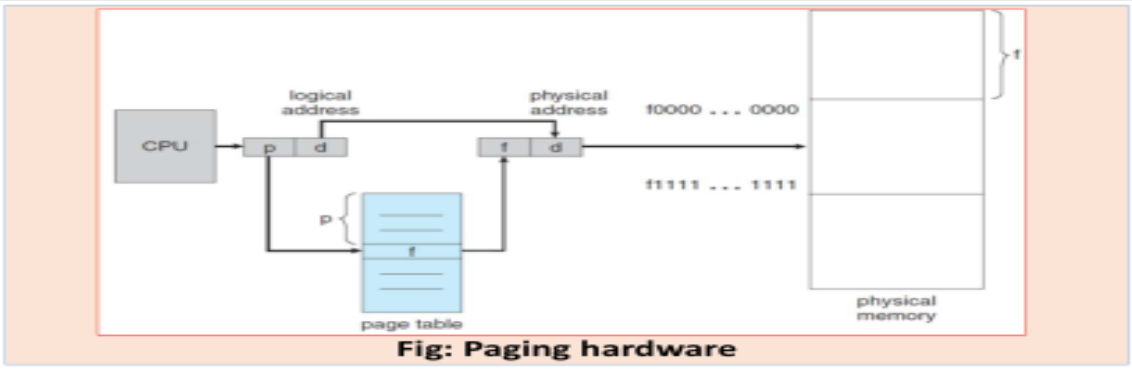


Fig: Paging hardware

5. Does paging suffer from fragmentation? Justify your answer.

Paging is another memory-management scheme. Any fixed-size partition based memory allocation scheme suffers from internal fragmentation. This problem arises when the size of a process is not a whole multiple of the partition size. In such case, for the last portion of the process, we need to allocate an entire partition whose part is only used by the process. Paging is also a fixed-size partition scheme, hence, suffers from internal fragmentation problem.

6. Consider a logical address space of 64 pages with 1024 bytes per page mapped

to a physical memory of 256 frames. Calculate the

(i) Number of bits required for logical address.

(ii) Number of Bits required for physical address.

(i) Bits required for logical address (ii) Bits required for physical address

Logical address space (/size) = 2^m

Logical address space (/size) = # of pages × page size = 64 × 1024

Logical address space (/size) = 2^6 x 2^{10}

m = 16bits(Answer)

Let x be the number of physical addresses

Physical address space (/size) = 2^x

Physical address space (size) = # of frames × frame size = 256 × 1024

Physical address space (size) = 2^8× 2^{10} = 2^{18}

Number of required bits in the physical address=x =18 bits

7. Differentiate between internal and external fragmentation.

8. What do you understand by external and internal fragmentation in case of memory management schemes.

External fragmentation- 1.5 marks, Internal fragmentation- 1.5 marks

External Fragmentation

 As process are loaded and removed from memory, the free memory space is broken into little pieces. This is known as external fragmentation.

 External fragmentation exists when there is enough total memory space to satisfy a request but the available spaces are not contiguous: storage is fragmented into a large number of small holes.

 Both the first-fit and best-fit strategies for memory allocation suffer from external fragmentation.

Internal fragmentation

 Unused memory that is internal to a partition.

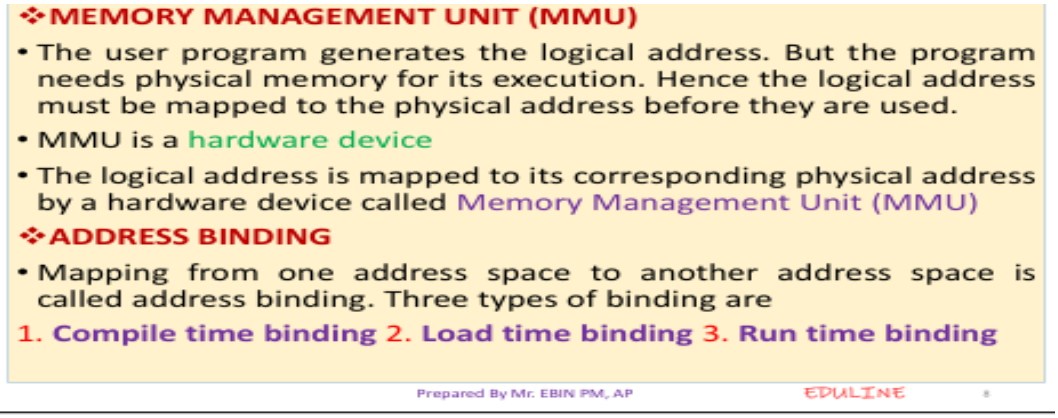
 Occurs when memory is partitioned on fixed size basis.

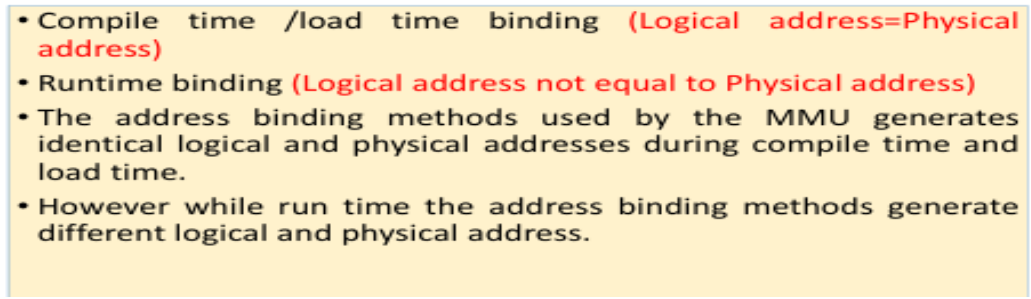
Consider multiple-partition allocation scheme with a hole of 18,464 bytes. Suppose that the next process requests 18,462 bytes. If we allocate exactly the requested block, we are left with a hole of 2 bytes.

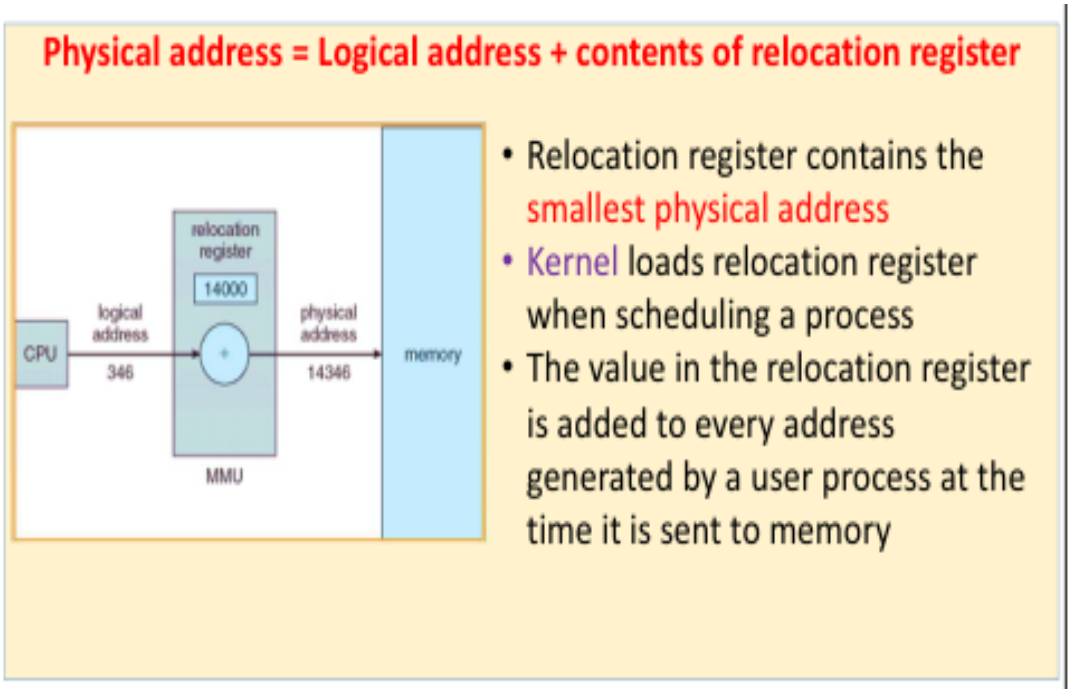
9. Explain the function of memory management unit.

Memory management unit- Mapping from virtual address to physical address is done with Memory Management Unit (MMU)

 Base register is also called relocation register







10. Differentiate Logical address Space and Physical Address Space

LOGICAL ADDRESS & PHYSICAL ADDRESS

❖**LOGICAL ADDRESS**

- It is the virtual address generated by the CPU that can be viewed by the user
- Logical address is generated by the CPU during a program execution
- The logical address is virtual as it does not exist physically. Hence it is also called Virtual address
- This address is used as a reference to access the physical memory location(physical address)

➢**Logical address space:-** set of all logical addresses generated by the CPU in reference to a program is referred as logical address space.

❖**PHYSICAL ADDRESS**

- Physical address is a location in a memory unit.
- The user can never view the physical address of program.
- The user cannot directly access the physical address. Instead, the physical address is accessed by its corresponding logical address by the user

➢**Physical address space:-** set of all physical addresses corresponding to the logical address is called physical address space.

11. Explain the terms (i) Dynamic Loading (ii) Dynamic Linking

8.1.4 Dynamic Loading

In our discussion so far, the entire program and all data of a process must be in physical memory for the process to execute. The size of a process is thus limited to the size of physical memory. To obtain better memory-space utilization, we can use **dynamic loading**. With dynamic loading, a routine is not loaded until it is called. All routines are kept on disk in a relocatable load format. The main program is loaded into memory and is executed. When a routine needs to call another routine, the calling routine first checks to see whether the other routine has been loaded. If not, the relocatable linking loader is called to load the desired routine into memory and to update the program's address tables to reflect this change. Then control is passed to the newly loaded routine.

The advantage of dynamic loading is that an unused routine is never loaded. This method is particularly useful when large amounts of code are

needed to handle infrequently occurring cases, such as error routines. In this case, although the total program size may be large, the portion that is used (and hence loaded) may be much smaller.

Dynamic loading does not require special support from the operating system. It is the responsibility of the users to design their programs to take advantage of such a method. Operating systems may help the programmer, however, by providing library routines to implement dynamic loading.

8.1.5 Dynamic Linking and Shared Libraries

Figure 8.3 also shows **dynamically linked libraries**. Some operating systems support only **static linking**, in which system language libraries are treated like any other object module and are combined by the loader into the binary program image. The concept of dynamic linking is similar to that of 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 (or at least the routines referenced by the program) 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 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 that particu'ar code segment is reached, the library routine is executed directly, incurring no cost for dynamic linking. Under this scheme, all processes that use a language library execute only one copy of the library code.

This feature can be extended to library updates (such as bug fixes). 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 to the new library. So that programs will not accidentally execute new, incompatible versions of libraries, version information is included in both the program and the library. More than one version of a library may be loaded into memory, and each program uses its version information to decide which copy of the library to use. Minor changes retain the same version number, whereas major changes increment the version number. Thus, only programs that are compiled with the new library version are affected by the incompatible changes incorporated in it. Other programs linked before the new library was installed will continue using the older library. This system is also known as **shared libraries**.

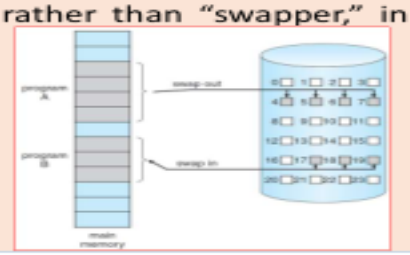
Unlike dynamic loading, dynamic linking generally requires help from the operating system. If the processes in memory are protected from one another, then the operating system is the only entity that can check to see whether the needed routine is in another process's memory space or that can allow multiple processes to access the same memory addresses. We elaborate on this concept when we discuss paging in Section 8.4.4.

12. Explain the concept of demand paging?

DEMAND PAGING

- A 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.
- Any page execution is started on a page fault.
- In demand paging firstly no programs are in memory.
- When CPU generate an address, a page fault will occur. When a page fault occurs, we can load the entire program in to main memory or we can load only the needed program.

- A lazy swapper never swaps a page into memory unless that page will beneeded.
- A swapper manipulates entire processes, whereas a **pager** is concerned with the individual pages of a process.
- We thus use "**pager**," rather than "swapper," in connection with demand paging.



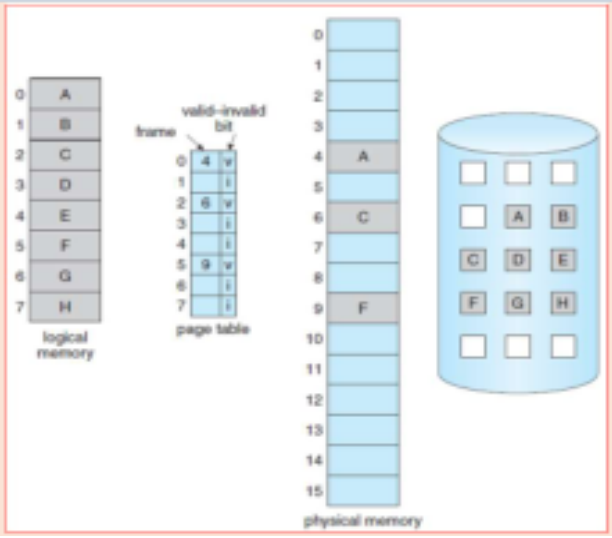
❖Basic Concepts

- With this scheme, we need some form of hardware support to distinguish between the pages that are in memory and the pages that are on the disk.
- The valid–invalid bit scheme can be used for this purpose.
- When this bit is set to "valid," the associated page is both legal and in memory.
- If the bit is set to "invalid," the page either is not valid (that is, not in the logical address space of the process) or is valid but is currently on the disk.

Mr. EBIN PM, AP

SYSTEMS http://www.youtube.com/c/EDULI

Page table when some pages are not in main memory



- Access to a page marked invalid causes a **page fault**.
- The 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.
- 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 aninvalid 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).

Mr. EBIN PM, AP

---

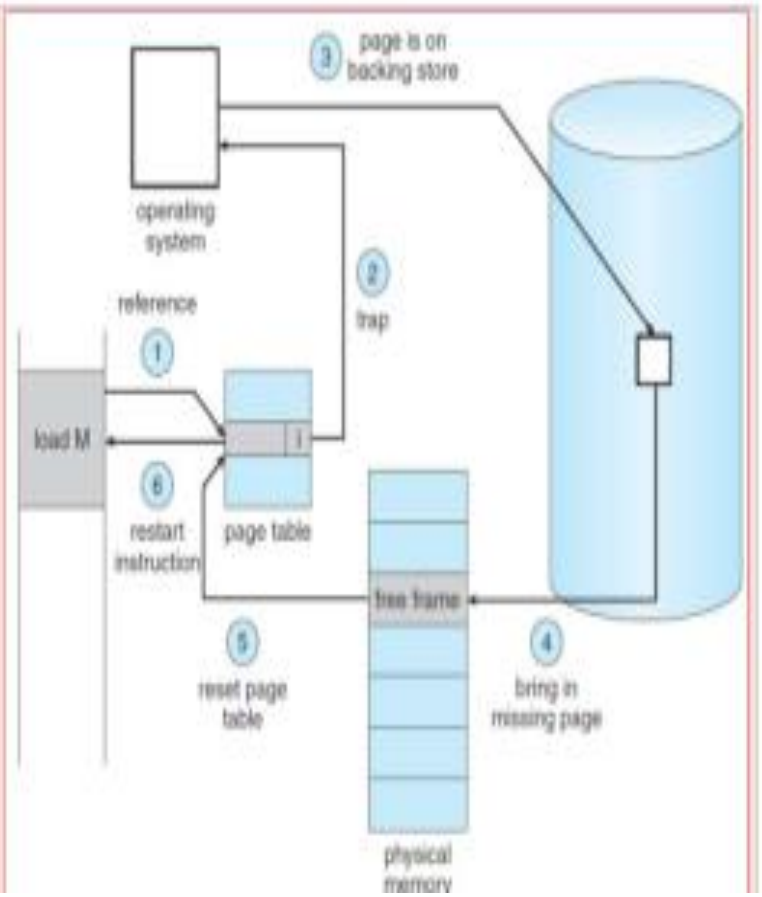
4. We schedule a disk operation to read the desired page into the newlyallocated frame.
5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
6. We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

- The efficiency of demand paging is increased by using locality of reference, because continuous hit is occurred. Hardware support for demand paging is

✓The page table must have valid/invalid bit

✓A swap area must need for performing swap out/swap in

❖Steps in handling a page fault

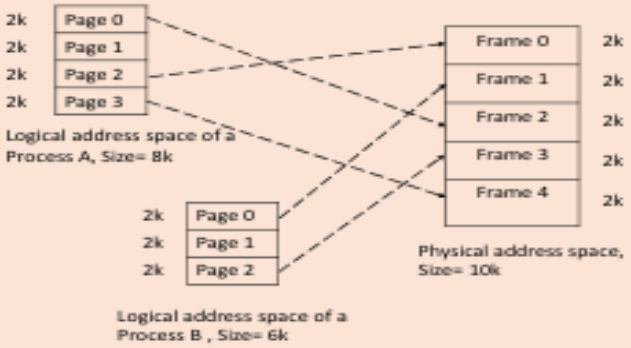


13. Explain the concept of paging.

PAGING

- Paging is a memory management technique.
- In this approach, physical memory is divided in to fixed sized block called frames and logical memory is also divided in to the fixed sized blocks called pages.
- The size of the page is same as that of frame.
- The key idea of this method is to place the pages of a process in to the available frames of memory , whenever this process is to be executed.

❖Paging Concept



❖**ADDRESS TRANSLATION**

- It is done by mapping. Logical address is mapped to physical address using a page table
- Every address generated by the CPU is divided into two parts
  page number (p) and **pageoffset (d).**
- Physical address is represented by two parts:
  Frame number(f) and **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.

Mr. EBIN PM, AP

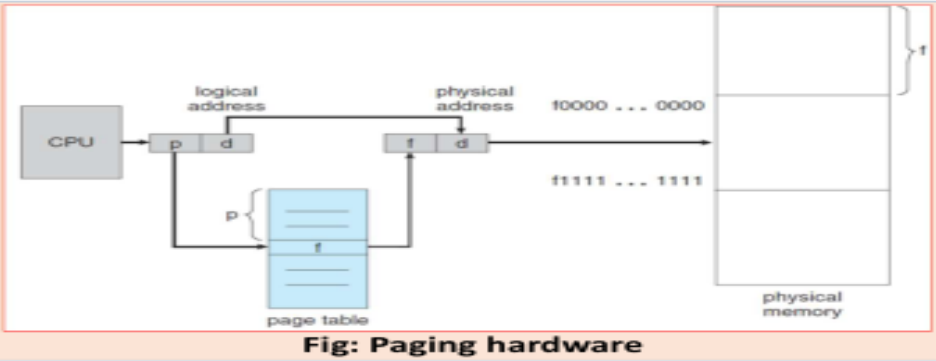


Fig: Paging hardware

14. Explain Optimal page replacement and LRU algorithms for page replacement.

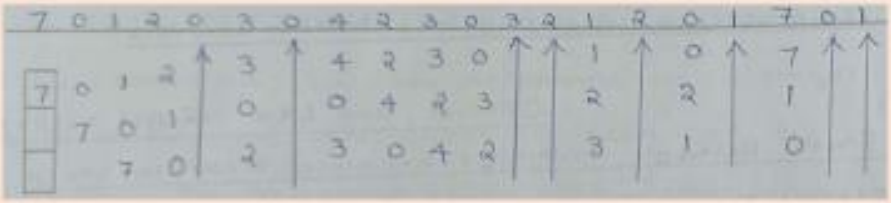
2. Optimal Replacement algorithm

- Optimal page replacement algorithm has the lowest page fault rate of all algorithms, and will never suffer from Belady's anomaly.
- It is simply, replace the page that will not be used for the longest period of time.

- Hit=11, Page fault=9. Difficult to implement because it requires future knowledge of the reference string

3. LRU (Least Recently Used) Page Replacement

- LRU chooses the page that has not been used for the longest period of time.
- This strategy looking backward in time rather than forward.



Hit=8 and page fault=12

❖Hardware Support for LRU implementation

➢Counters

- we associate with each page-table entry a time-of-use field and add to the CPU a logical clock or counter.
- The clock is incremented for every memory reference.
- Whenever a reference to a page is made, the contents of the clock register are copied to the time-of-use field in the page-table entry for that page.
- We replace the page with the smallest time value.

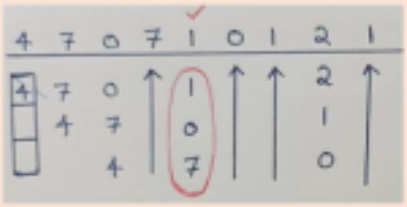
➢Stack

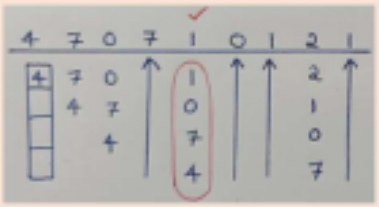
- Another approach to implementing LRU replacement is to keep a stack of page numbers.
- Whenever a page is referenced, it is removed from the stack and put on the top.
- In this way, the most recently used page is always at the top of the stack and the least recently used page isalways at the bottom.
- Here doubly linked list is used for implementing stack

  Most recently used page number – Top of stack

  Least recently used page number - Bottom of stack

- Like optimal replacement, LRU replacement does not suffer from Belady's anomaly. Both algorithms are called **stack algorithms.**
- A stack algorithm is an algorithm for which it can be shown that the set of pages in memory for *n* frames is always a ***subset*** of the set of pages that would be in memory with *n*+ 1 frames.





- Consider a particular instant. We can see that LRU with frame number 3 is a subset of LRU with frame number 4.

15. Why are page sizes always powers of 2?

- Page size is defined by the hardware. The size of the page is typically a power of 2.
- At physical level addresses are represented in binary form
- In binary system it is easy to handle various activities if everything is expressed in power of 2 and this is to make the division of a logical address into page number and page offset easy.
- To perform multiplication and division in power of 2, we only need to shift left or right the value to be multiplied. Thus the address computation becomes easy.