UNIT V: Memory Management

Basic concept, Logical and Physical address, Memory allocation: Contiguous Memory allocation, Fixed and variable partition, Internal and External fragmentation, Compaction, Swapping, Segmentation.
Paging: Principle of operation, Page allocation, Hardware support for paging, Page table structuring technique, Protection and sharing, Advantages and Disadvantages of paging.
Virtual Memory: Basics of Virtual Memory, Demand paging, Page fault, Working Set, Dirty page/Dirty bit, Page Replacement algorithms: Optimal, First in First Out (FIFO) and Least Recently used (LRU), Thrashing.

Introduction

- This chapter mainly deals with Memory Management strategies which help in managing memory.
- It describes various contiguous and non-contiguous memory allocation schemes.
- It describes contiguous memory allocation schemes as fixed and variables.
- It also describes process of swapping in Operating System.
- It focuses on the concept of fragmentation along with its types.
- It also discusses memory management scheme called paging and segmentation along with different page table structures.
- It gives idea about usage of virtual memory in Operating System with their implementations in the form of demand paging.
- It also explains the concept of page fault with different page replacement algorithms.
- At the end of this chapter numerical based on page fault and segmentation are solved.

Memory Management Basics

- Memory is any physical device capable of storing information temporarily or permanently.
- Memory consist of large array of words or bytes each with it's own address.
- Memory is central to operation of modern computer system.
- The part of Operating System that manages memory hierarchy is called memory manager.
- Memory can be either volatile or non-volatile.
- Proper management of memory is vital for a computer system to operate properly.
- Modern operating system has complex system to properly manage memory.
- A computer programmer requires considering how to manage memory. Even storing number in memory requires programmer to specify how memory should store it.

- A memory hierarchy in computer distinguishes each level in hierarchy by response time. Memory hierarchy is shown below.

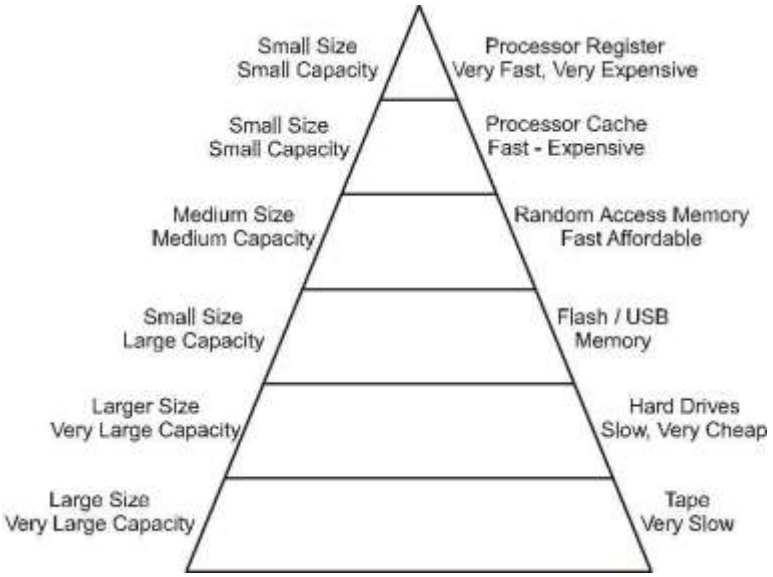


Fig. Computer Memory Hierarchy

Basic Hardware Mechanism

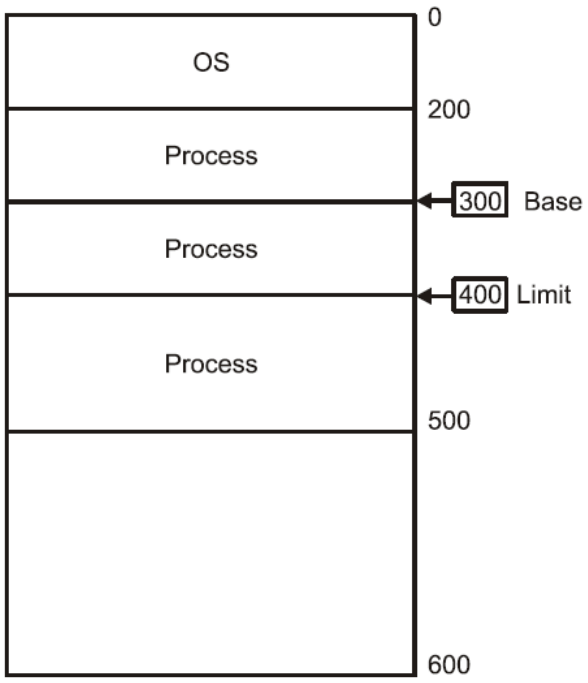


Fig. A base and limit register defines logical address space

- Main memory is required in order to execute a program.
- Main memory and registers built into processor itself are only storage that CPU can access directly.

- Any instruction in execution and any data being used by instruction must be in one of these direct access storage devices.
- Registers that are built into CPU are generally accessible within one cycle of CPU clock.
- We need to make sure that each process has separate memory space.
- To do this we need to determine range of legal address space that process may access and to ensure that process can access only this legal address.
- The above fig shows base register which holds smallest legal physical memory address.
- Here limit register specifies size of range i.e. 300 is base and 400 is limit.
- Above fig. also shows address space or process in which every program can lie between base and limit address value.

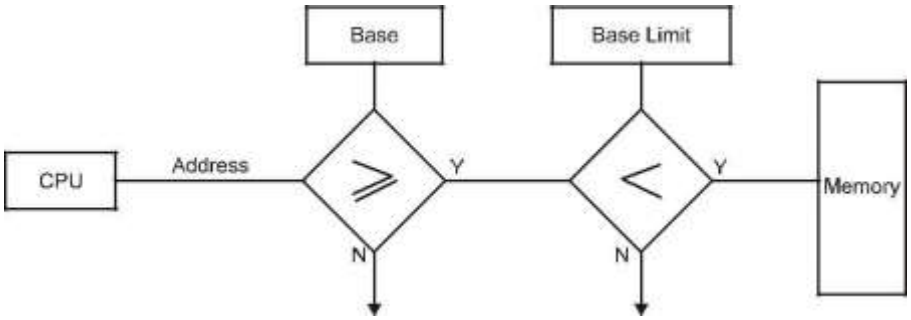


Fig. : Hardware Address Protection with Base and Limit Register

- Protection of memory space is accomplished by having CPU hardware.
- The base and limit register can be loaded only by Operating System, which uses a special privileged instruction.
- The operating system execution in kernel mode will get unrestricted access to both operating system memory and user memory.
- This provision allows operating system to load users program into users memory.

Address Binding

- Address binding is the process of binding the instruction and data to memory address.
- Generally program resides on secondary storage i.e. hard disk but in order to execute that program it should be brought into main memory.
- During execution process may be moved between disk and main memory.
- Collection or process waiting to be brought into memory for execution forms input queue.
- The normal procedure is to select one of the processes from the i/p queue and to load that process into memory.
- During execution process accesses instruction and data from memory.
- In most cases user program will go through several steps as shown in below fig.

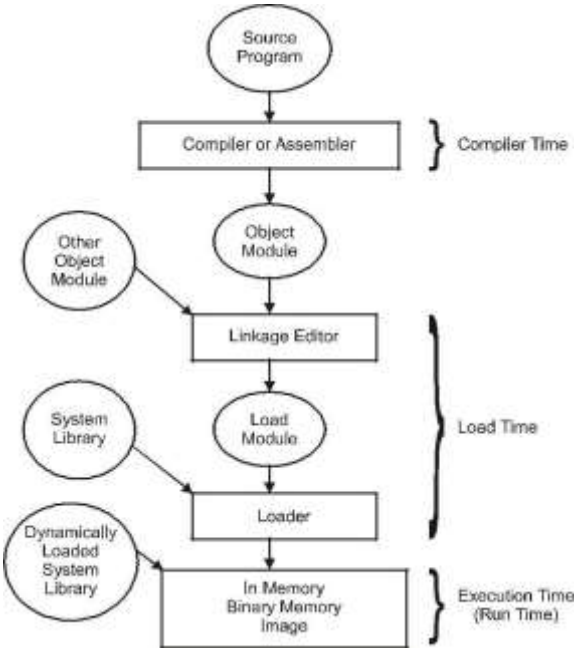


Fig.: Multistep processing of user program

There are three types of binding as:

1. Compile Time
2. Load Time
3. Execution Time

Compile Time: Absolute code can be generated if we know at compile time where process will reside in memory. If we know that user process resides at location R then generated compiler code will start at that location and extend up from there. After that if starting location changes then it will be necessary to recompile this code. It done by compiler Ex. Train ticket where we know everything's prior to journey

Load Time: Compiler must generate relocatable code if it is not known at compile time where process will reside. In this case final binding is delayed until load time. If starting address changes then we only need to reload user code. It done by OS. Ex. Airplan journey where we came to know the seat detail after boarding

Execution Time: If process can be moved during it's execution from one memory segment to another then binding must be delayed until run time, special hardware must be available for this scheme to work. It done by Processor itself .Ex. Exchange of seats if seats not allocated sequentially after boarding

Address Space

- Compile time and load time binding generate identical logical and physical addresses.
- An address generated by CPU is commonly referred to as logical address.
- An address of main memory where actually instruction resides is known as physical address.
- The logical and physical addresses differs due to execution time address binding, during which logical address refers to as virtual address.
- The set of all logical addresses are generated by program is a logical address space.
- The set of all physical addresses corresponding to these logical addresses is physical address space.
- The run time mapping from virtual to physical addresses is done by a hardware device called memory management unit.

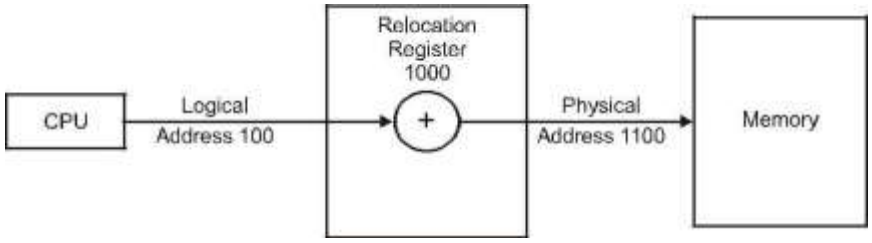


Fig. Dynamic Relocation Using Relocation Register

- The above figure shows Dynamic Relocation using relocation register which is nothing but a base register. For e.g. if user wants to access location 100 then it is mapped to location 1100.

Dynamic Loading and Linking

- Dynamic Loading is useful for better memory space utilization.
- In dynamic loading, routine is not loaded until it is called.
- All routines are kept on disk in relocatable load format and main program is loaded into memory and executed.

Advantages:

1. Unused routine is never loaded.
2. It is useful when large amount of code needs to handle infrequently occurring cases.
3. It does not require special support from operating system. Disadvantages:

As it is run time activity so if we need to load data from I/O device then it become time consuming as I/O read/write is mechanical activity.

Dynamic Linking

- In this case linking is postponed until run time i.e. linked at run time.
- With dynamic linking stub is included for each library routine reference.
- A stub is small piece of code which indicates how to locate appropriate memory resident library routine or how to load library if routine is not already present.
- During execution of process, stub is replaced by address of relevant library code.
- If library code is not in memory it is loaded at this time.

Advantages:

4. Less time needed to load program.
5. Less disk space needed to store libraries. Disadvantages:

1. Time consuming run time activity.
2. It results in slower execution of program

Difference of Static and dynamic

Difference Between Logical address and Physical Address

Parameter	LOGICAL ADDRESS	PHYSICAL ADDRESS
Basic	Generated by CPU	location in a memory unit
Address Space	Logical Address Space is set of all logical addresses generated by CPU in reference to a program.	Physical Address is set of all physical addresses mapped to the corresponding logical addresses.
Visibility	User can view the logical address of a program.	User can never view physical address of program.
Generation	generated by the CPU	Computed by MMU
Access	The user can use the logical address to access the physical address.	The user can indirectly access physical address but not directly.
Editable	Logical address can be change.	Physical address will not change.
Also called	virtual address.	real address.

Parameter	LOGICAL ADDRESS	PHYSICAL ADDRESS
Size	It is larger than physical address	Smaller than logical
Time	It is generated at compile time.	It is generated at load time.

Swapping

- Swapping is a mechanism in which process can be swapped temporarily out of main memory to backing store and then brought back into memory for continued execution.
- Backing store is usually a hard disk or any other secondary storage which is fast in access and large enough to accommodate copies of all memory images for all users.

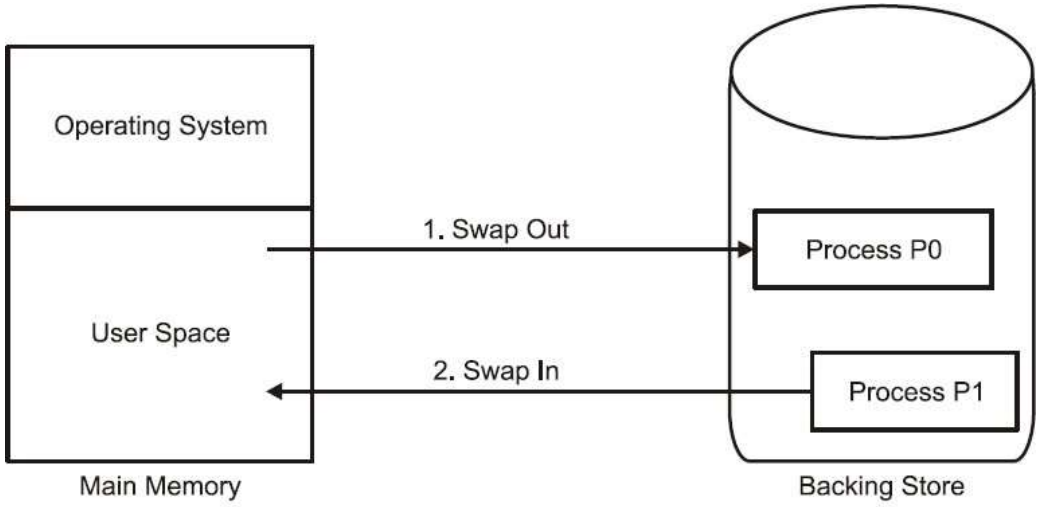


Fig.: Mechanism of Swapping

- The above figure shows swapping mechanism in OS where process P0 is not needed currently so it is rollout or swapout to backing store and process P1 from backing store swap in to main memory as it is necessary for process execution, it should be in main memory.
- Process that is swap out will be swapped back into same memory space that it occupied previously.
- Major time consuming part of swapping is transfer time. Total transfer time is directly proportional to amount of memory swapped.
- Let us assume that

Process size = 1MB Transfer rate = 4 MB/Sec.

Swapout time would be = 1/4 = 250 ms Avg. latency = 5 ms

Net swap out time = 255 ms

Swapout + Swap in time = 255+255 = 510 ms. Need of swapping:

1. To achieve multiprogramming environment.
2. Less amount of space available.
3. High priority process waiting for execution.
4. Nature of large context switching.

Contiguous Memory Allocation

- In contiguous memory allocation, each process is contained in a single contiguous section of memory.
- Here base i.e. relocation and limit registers are used to point smallest and largest memory addresses of process.

- We usually want several user processes to reside in memory at same time so we need to consider how to allocate available memory to the processes that are in input queue waiting to be brought into memory.
    - There are two techniques available to achieve contiguous allocation which are as below.

1. MFT (Multiprogramming with Fixed Task).
2. MVT (Multiprogramming with Variable Task).

- **Fixed Partition**
    - In this type of technique total space is partitioned into fixed partition at boot time.
    - Boundary for each partition is fixed and set at booting time which cannot be changed.
    - In this technique each partition has exactly one process.
    - This scheme was used by IBM for system 360 OS/MFT.

    - Consider above figure in which memory space is divided into 5 partitions at boot time, where for every partition one input queue is allocated and degree of multiprogramming is bound by number of partitions.
    - The above figure shows example where partition 1 has 1 process to execute, partition 4 and 5 have 2 and 3 processes to execute that will be executed one by one.
    - Partition 2 and 3 don't have any process so it's wastage of space as separate queue mechanism does not allow to allocate process to partitions 2 and 3.
    - This technique suffers from internal fragmentation so to overcome this problem single queue mechanism is implemented which is shown in below figure.

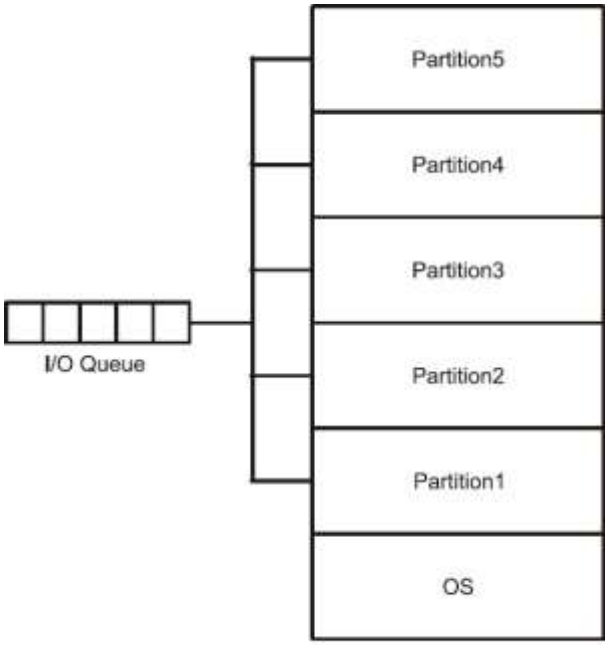


Fig. : Fixed Partition with Single Queue.

– The above figure shows fixed partition with single queue where as per capacity of partition, the I/O queue will assign process to partition. When partition becomes free it searches in queue for process.

Disadvantages:

1. It suffer from the problem of internal fragmentation.
2. No sharing of memory between process.
3. No dynamic address translation.
4. Load time binding needed.
5. Limit of process size
6. Limitation of degree of multiprogramming

Advantages of Fixed Partitioning –

- **Easy to implement:** The algorithms needed to implement Fixed Partitioning are straightforward and easy to implement.
- **Low overhead:** Fixed Partitioning requires minimal overhead, which makes it ideal for systems with limited resources.
- **Predictable:** Fixed Partitioning ensures a predictable amount of memory for each process.
- **No external fragmentation:** Fixed Partitioning eliminates the problem of external fragmentation.
- **Suitable for systems with a fixed number of processes:** Fixed Partitioning is well-suited for systems with a fixed number of processes and known memory requirements.
- **Prevents processes from interfering with each other:** Fixed Partitioning ensures that processes do not interfere with each other's memory space.
- **Efficient use of memory:** Fixed Partitioning ensures that memory is used efficiently by allocating it to fixed-sized partitions.
- **Good for batch processing:** Fixed Partitioning is ideal for batch processing environments where the number of processes is fixed.
- **Better control over memory allocation:** Fixed Partitioning gives the operating system better control over the allocation of memory.
- **Easy to debug:** Fixed Partitioning is easy to debug since the size and location of each process are predetermined.

Disadvantages of Fixed Partitioning –

1. **Internal Fragmentation:**
   Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This can cause internal fragmentation.

2. **External Fragmentation:**
   The total unused space (as stated above) of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form (as spanning is not allowed).

3. **Limit process size:**
   Process of size greater than the size of the partition in Main Memory cannot be accommodated. The partition size cannot be varied according to the size of the incoming process size. Hence, the process size of 32MB in the above-stated example is invalid.

4. **Limitation on Degree of Multiprogramming:**
   Partitions in Main Memory are made before execution or during system configure. Main Memory is divided into a fixed number of partitions. Suppose if there are partitions in RAM and are the number of processes, then condition must be fulfilled. Number of processes greater than the number of partitions in RAM is invalid in Fixed Partitioning.

Multiprogramming with Variable Task (Variable Partition)

- In order to overcome the problem of internal fragmentation in fixed partition, variable partition technique was introduced.

- In this technique both size and number of partition can vary with time.
- Process can load at anywhere in main memory and also can move from one partition to another.
- It requires dynamic address binding.
- Sizing of partition consider the size of process residing in queue.

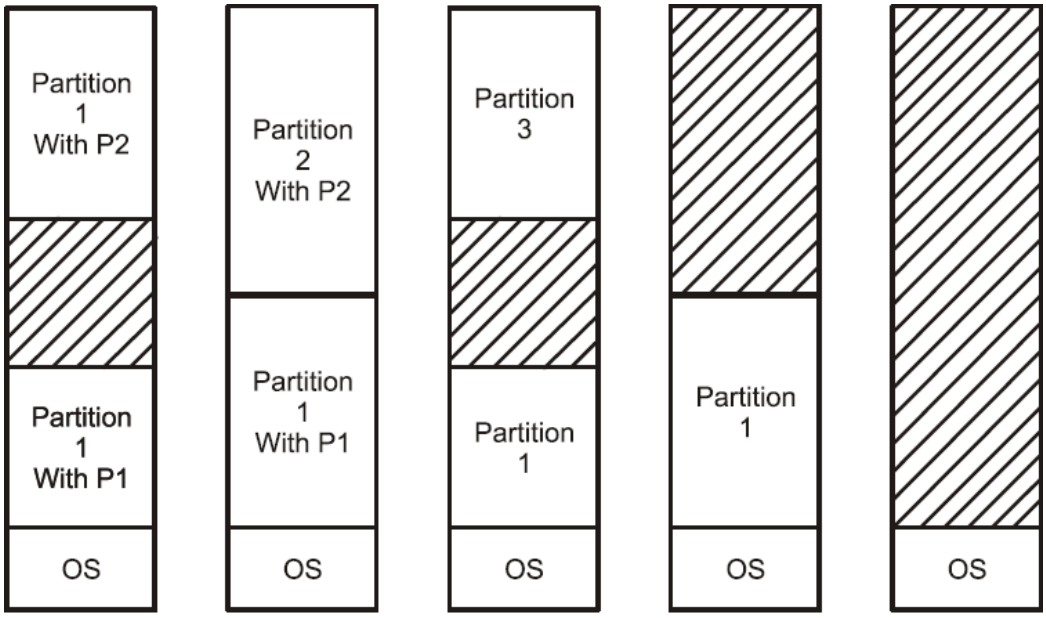


Fig.: Variable Partioning

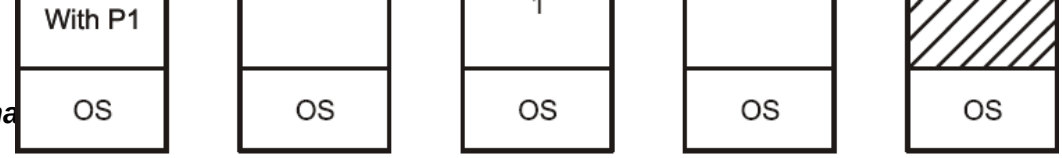


Fig.: Variable Partioning

- The above figure shows variable partitioning in which shaded blocks represent empty partition without process while other blocks contain process with same size.
- So we can say this technique overcomes the problem of internal fragmentation but may suffer from problem of external fragmentation.

1. No internal fragmentation
2. No limitation of no of processes
3. No limitation of process size

Memory Allocation Policy for Fixed Partition

- There are three memory allocation policies adopted for fixed parition as follows.
- First fit
- Best fit
- Worst fit

First Fit: It allocates first hole that is big enough. In this case searching starts in beginning of set of holes or where previous first fit search has ended. First fit allocation is faster in making allocation but leads to memory waste.

Best fit: This policy makes best use of memory space which allocates smallest hole that is big enough. In this case set of holes are first arranged in ascending order and then allocation is done. It is somehow slower in making allocation.

Worst fit: Worst fit memory allocation is opposite of best fit which allocates largest hole first. In this case we must search entire list unless it is sorted by size. This strategy produces largest leftover hole, which may be more useful than smaller leftover hole from best fit approach. It is not best choice for an actual system.

e.g. If given memory partitions are 100K, 500K, 200K, 300K and 600K. How would each of first fit, best fit and worst fit algorithm places processes of 212K, 417K, 112K, 426K.

First fit=>

212K □ 500K memory partition 417K □ 600K memory partition 112K □ 200K memory partition

426 => No allocation as no sufficient amount of memory available.

Best fit=>

212K □ 300K memory partition 417K □ 500K memory partition 112K □ 200K memory partition 426K □ 600K memory partition

Worst fit=>

212K □ 600K memory partition

417K □ 500K memory partition 112K □ 300K memory partition

426K □ No allocation as no sufficient amount of memory available.

In this case best fit turns out to be best policy as it allocates memory for all process.

4.2 Fragmentation

Que.: What is memory fragmentation? Differentiate between Internal and External Fragmentations. (S13-2M, S12-3M, S11-5M, S10-5M, S09-5M)

– Memory fragementation is a process in which there is a wastage of memory space as free blocks are too small to satisfy any request.

– Fragmentation occurs when dynamic memory allocation algorithm allocates some memory and a small piece is left over that can not be effectively used.

There are two types of fragmentations as

1. Internal Fragmentation
2. External Fragmentation

Internal Fragmentation:

It's a space wasted inside of allocated memory blocks because of restriction on allowed sizes of allcoated blocks. Allocated memory may be slightly larger than requested memory. This size difference is memory internal to a parittion but not being used.

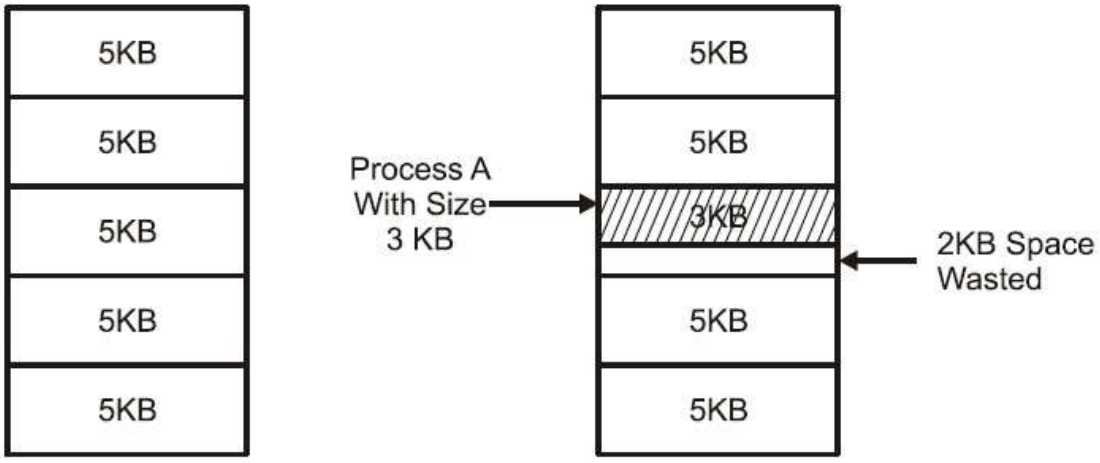


Fig.: Internal Fragmentation

– Above figure shows mechanism of Internal Fragementation, where all blocks are of size 5KB but process requests only 3KB so there is wastage memory of 2KB that is nothing but internal fragmentation.

External Fragmentation

- It generally occurs when dynamic memory allocation algorithm allocates some memory

and small piece is left over that can not be effectively used.

– If too much external fragmentation occurs, the amount of usable memory is drastically reduced.

– Total memory space exists to satisfy request but it is not contiguous.

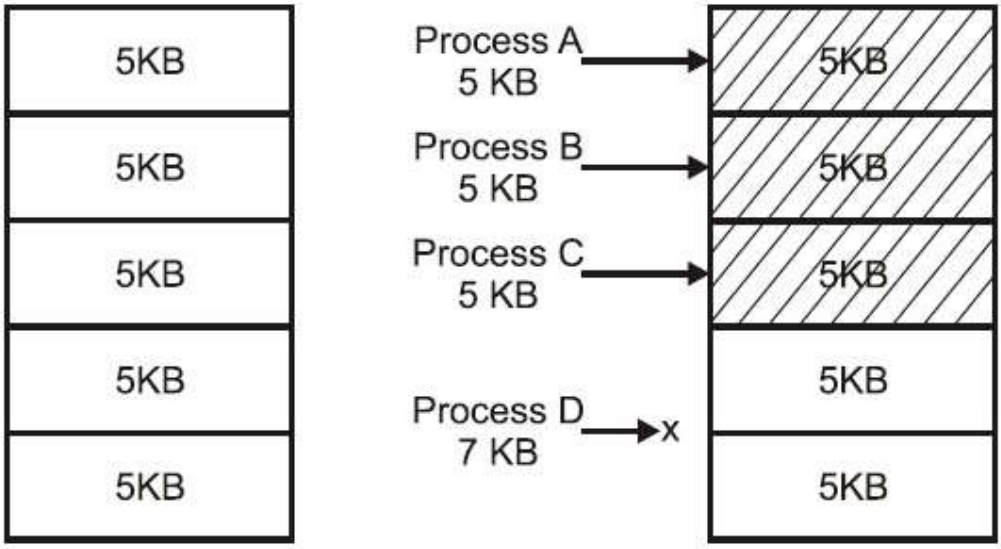


Fig.: External Fragmentation

– Above figure shows external fragmentation, where all blocks are of size 5KB and four request comes out of which 3 request of process A, B and C can be satisfied but request of process D cannot be satisfied because total memory space exists to satisfy request but it is not contiguous and external fragmentation occurs.

Compaction:

Que.: Explain the concept of Compaction.

Que.: How to solve problem of external fragmentation.

– External fragmentation occurs when enough total memory space is available to satisfy but that total memory is not contiguous that means, it is fragmented into small chunks, so we can not satisfy request.

– The solution to this problem is compaction, which combines all small chunks into single block by moving all allocated blocks to one end of memory.

– But there are some limitations to compaction as any pointer to block needs to be updated when block is moved and compaction is not possible till all such pointers are detected.

– Compaction is a shuffling of memory content to place all free memories together in one large block.

Difference between Internal fragmentation and External fragmentation

S.NO	Internal fragmentation	External fragmentation
1.	In internal fragmentation fixed-sized memory, blocks square measure appointed to process.	In external fragmentation, variable-sized memory blocks square measure appointed to the method.
2.	Internal fragmentation happens when the method or process is smaller than the memory.	External fragmentation happens when the method or process is removed.
3.	The solution of internal fragmentation is the best-fit block.	The solution to external fragmentation is compaction and paging.
4.	Internal fragmentation occurs when memory is divided into fixed-sized partitions.	External fragmentation occurs when memory is divided into variable size partitions based on the size of processes.
5.	The difference between memory allocated and required space or memory is called Internal fragmentation.	The unused spaces formed between non-contiguous memory fragments are too small to serve a new process, which is called External fragmentation.
6.	Internal fragmentation occurs with paging and fixed partitioning.	External fragmentation occurs with segmentation and dynamic partitioning.
7.	It occurs on the allocation of a process to a partition greater than the process's requirement. The leftover space causes degradation system performance.	It occurs on the allocation of a process to a partition greater which is exactly the same memory space as it is required.
8.	It occurs in worst fit memory allocation method.	It occurs in best fit and first fit memory allocation method.

Paging

- Page is a memory space where process related information is stored.
- Page is considered as a smallest unit of data for memory allocation.
- Size of page may vary as per architecture of processor.
- System with smaller page size uses more pages.
- Paging is memory management scheme that permits physical address space of process to be noncontiguous.
- Operating system retrieves data from secondary storage in same size block called pages.
- Paging is an important part of virtual memory implementation.
- One process can have number of pages.
- When process is to be executed it's pages are loaded into any available memory frames from backing store.
- Size of page is typically a power of 2, varying between 512 bytes and 16 MB per page.
- Paging keeps track of all free frames.
- To run process of size n pages, we require n frames but it is not required to be contiguous.

Hardware for paging:

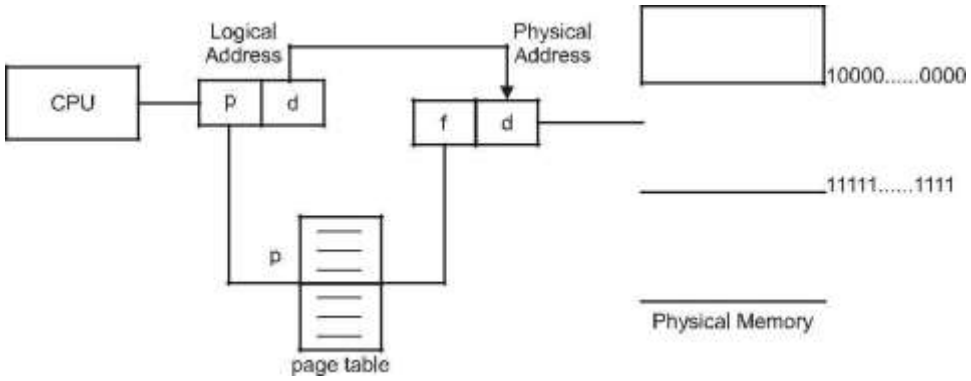


Fig.: Paging Hardware

- Above figure shows hardware support for paging, where every address generated by CPU is divided into 2 parts a page number (p) and page offset (d).
- Page number is used as an index into a page table.
- Page table contains base address of each page in physical memory.
- The base address is combined with page offset to define physical memory address that is sent to memory unit.

Paging model of memory:

Paging model of memory:

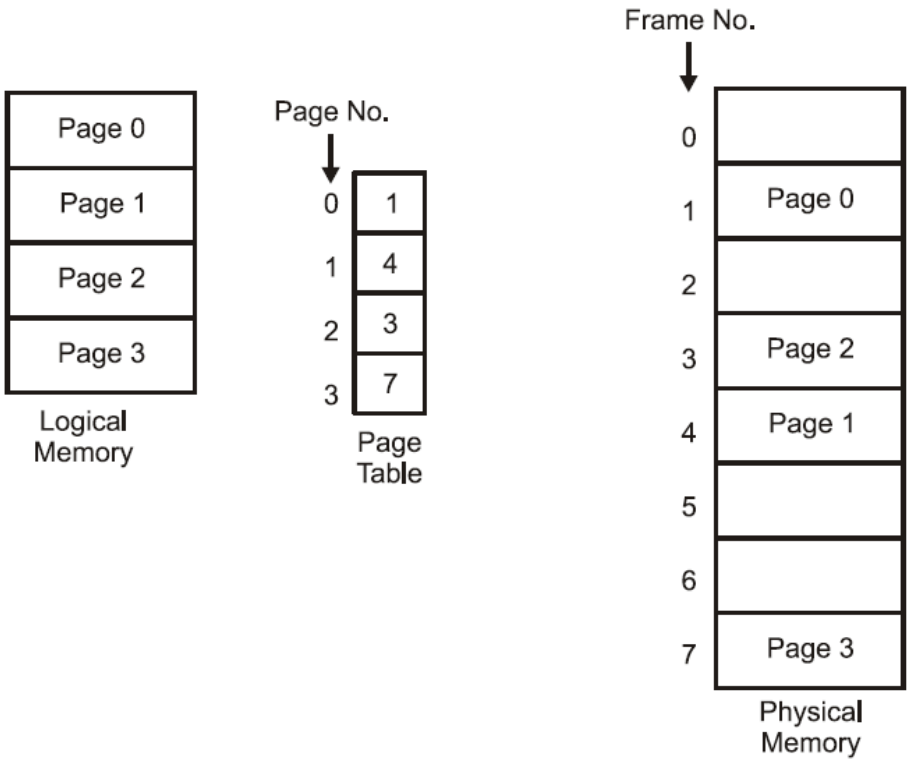


Fig.: Paging model of logical and physical memory

- Above figure shows paging model of logical and physical memory.
- Here logical memory is divided into blocks called pages.
- Physical memory is divided into blocks called frames.
- Page table contains base address of physical memory for page.
- Here page 0 is stored in frame 1 while page 1, 2, 3 are stored in frames 4, 3 and 7.

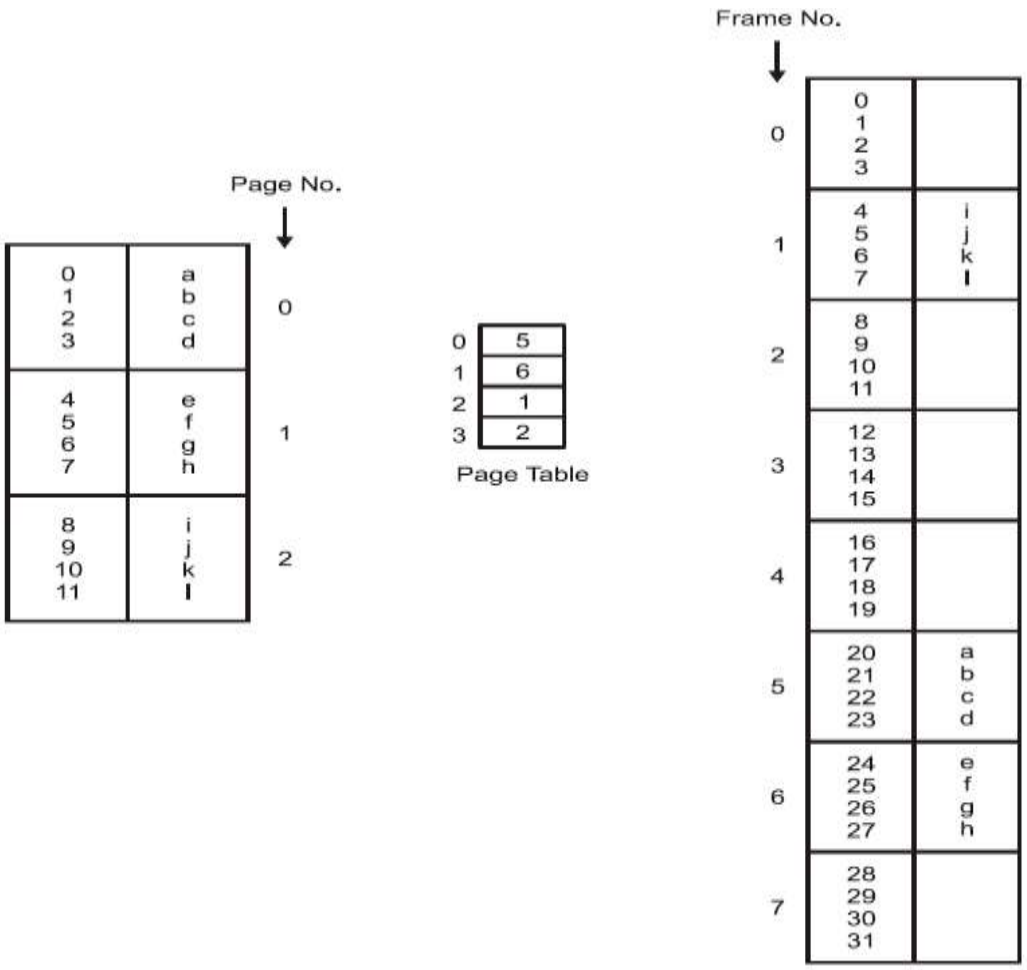


Fig. Paging Example

- Above figure shows example for paging where page is of size 4 bytes and physical memory of 32 bytes.
- Here logical Address 0 with Page 0 having Offset 0 can be map to physical address 20=[(5x4)+0]. Similarly other logical address with page0 can map to physical address 21, 22, 23.
- Same procedure is followed to store remaining pages.

Translation look aside buffer

- The main problem that we face with paging hardware is time required to access an user memory location. It requires two memory accesses to access byte.
- Therefore translation look aside buffer can be used which supports and reduces memory access time which consists of small, special and fast lookup hardware cache.
- TLB is associative and high speed memory.
- Each entry in TLB consists of two parts a key (or tag) and value.
- WIth TLB search is fast but hardware is expensive.
- Number of entries in TLB is small.
- When logical address is generated by CPU then page number is presented to TLB.
- If page number is found then it's frame number is immediately available and is used to access memory.
- Thus use of TLB saves most of time in paging than without TLB.

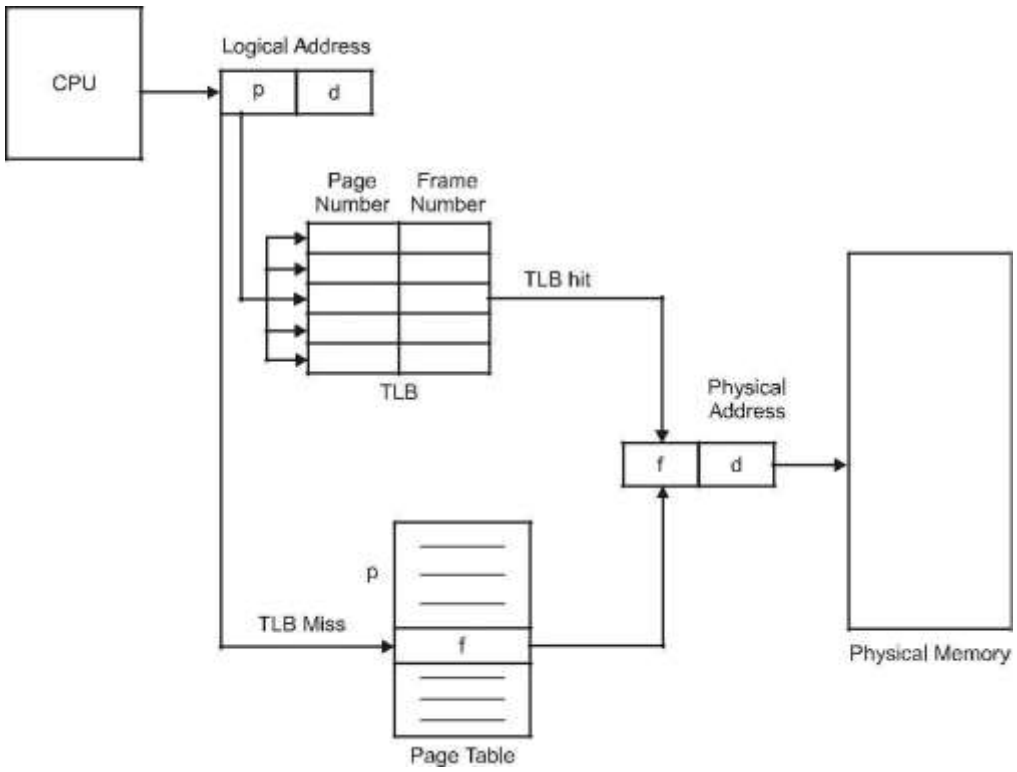


Fig.: Paging Hardware with TLB

- Above figure shows paging hardware with TLB, here TLB helps in paging to save time to access memory.
- If page number is not in TLB then it is called TLB miss.
- If TLB is already full of entries, the OS must select one for replacement policy to choose proper one.
- The percentage of times that a particular page number is found in TLB is called hit ratio.

Advantages :

- Allocating memory is easy and cheap
- Any free page is ok, OS can take first one out of list it keeps
- Eliminates external fragmentation
- Data (page frames) can be scattered all over PM
- Pages are mapped appropriately anyway
- Allows demand paging and prepaging
- More efficient swapping
- No need for considerations about fragmentation
- Just swap out page least likely to be used

Disadvantages :

- Longer memory access times (page table lookup)
- Can be improved using TLB
- Guarded page tables
- Inverted page tables
- Memory requirements (one entry per VM page)
- Improve using Multilevel page tables and variable page sizes (super-pages)
- Guarded page tables
- Page Table Length Register (PTLR) to limit virtual memory size
- Internal fragmentation

Structure of Page Table

As page size and page numbers increase for process so in order to manage that, three types of page structures are introduced which are as follow.

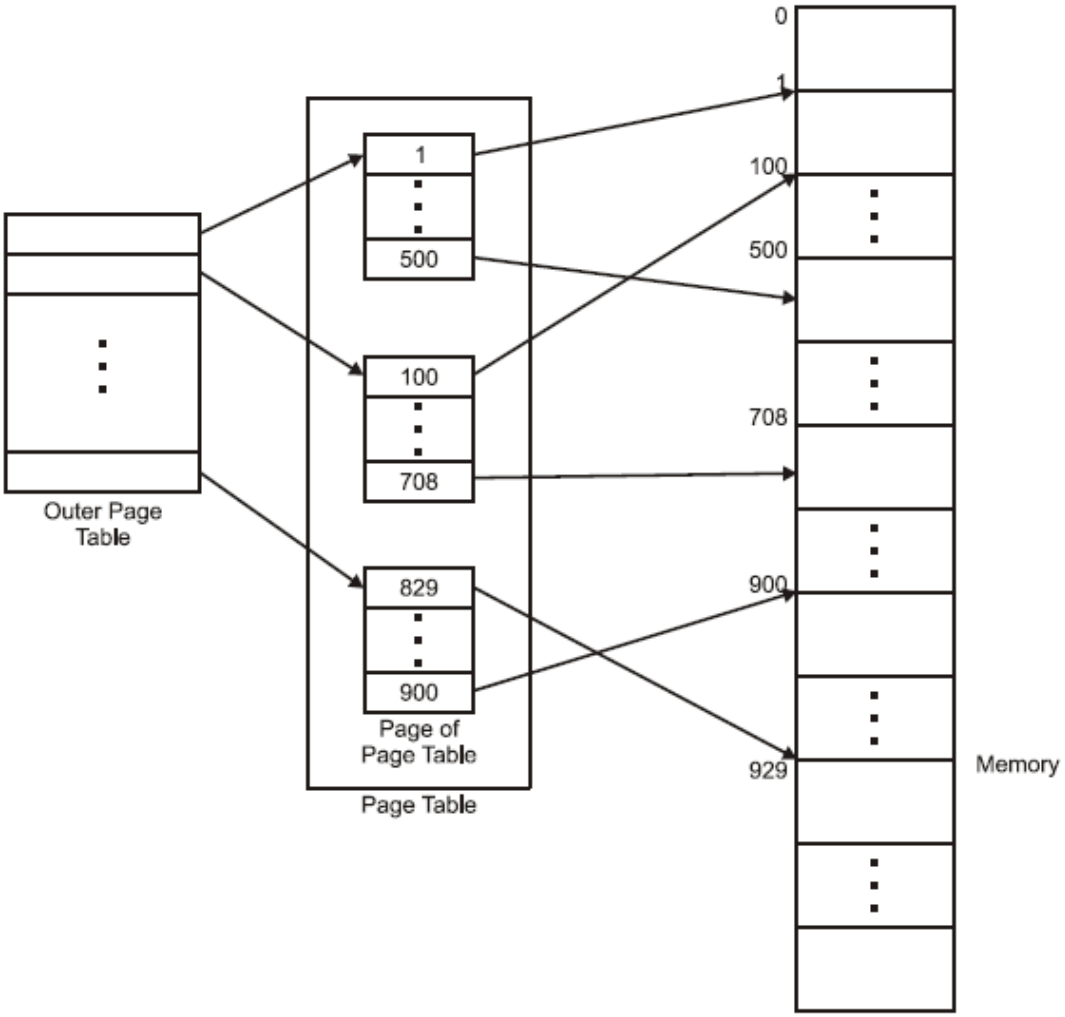
1. Hierarchical Paging

2. Hashed Page Table
3. Inverted Page Table

1. Hierarchical Paging

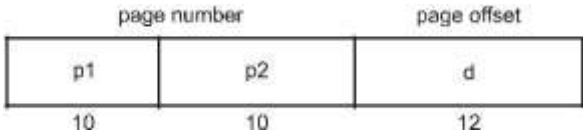
- Modern computer system supports a large logical address space(2^{32} to 2^{64}), in this environment page table itself becomes large.
- If page size in such a system is 4KB (2^{12}) then page table may consist of up to 1 million entries ($2^{32}/2^{12}$).
- Assuming that such entry consists of 4 bytes, each process may need up to 4MB of physical address for page table alone.
- We would not want to allocate page table space for page table alone.
- We would not want to allocate page table contiguously in main memory.
- So one of solutions to this problem is to divide a page table into smaller pieces, which can be done with help of two level paging algorithm, in which page table is paged.
- Below figure shows two level page table scheme in which there is 32 bit logical address space with page size of 4KB.
- Here logical address is divided into page number consisting of 20 bits and page offset of 12 bits.

– Page number is divided into 10 bit page number and 10 bits page offset, so logical address is as



given below.

Fig.: Two level page table scheme



– Here p1 is an index into outer page table and p2 is displacement within page of outer page table.

– The address transaction method for this architecture is shown in figure below.

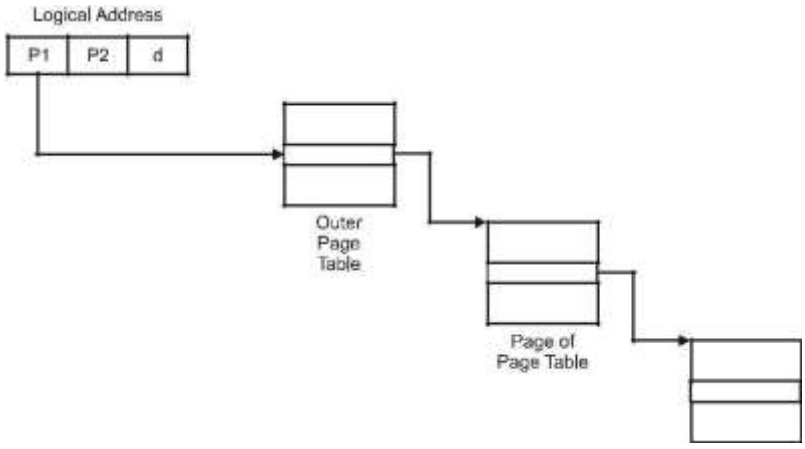


Fig.: Address transaction for 2 level 32 bit paging architecture.

Hashed Page Table

- The hashed page table is used for handling address space larger than 32 bits, in which hash value is virtual page number.
- Each entry in hash table contains linked list of elements that hash to same location.
- Each element consist of 3 fields as virtual page number, value of mapped page frame and pointer to next element in linked list.

Algorithm:

- Virtual page number in virtual address is hashed into page table.
- Virtual page number is compared with (field 1) in first element in linked list.
- If there is match, corresponding page frame (field 2) is used to form desired physical address.
- If there is no match, subsequent entries in linked list are searched for matching virtual page number.
- This algorithm can be shown in figure below.

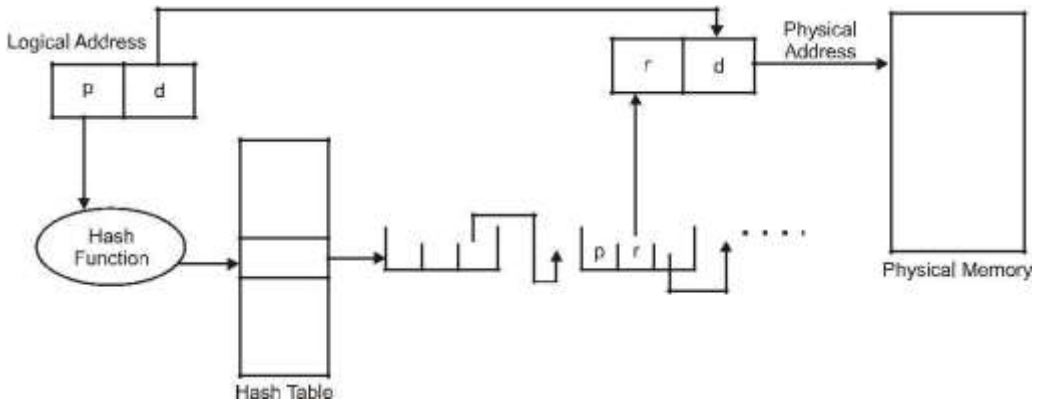


Fig.: Hash Page Table

Inverted Page Table

- Generally each process has an associated page table with one entry for each page that the process is using.
- This table representation is natural one, as process reference pass through the pages virtual address.
- Operating system must then translate this reference into physical memory address, as table is sorted by virtual address.
- Operating system then calculates where in table the associated physical address entry is located.
- One of drawbacks of this method is that each page table may consist of millions of entries.
- To overcome this drawback an inverted page table is used which has one entry for each real page of memory.
- Each entry consists of virtual address of page stored in that real memory location.
- Thus only one page table is in the system and it has only one entry for each page of physical memory.

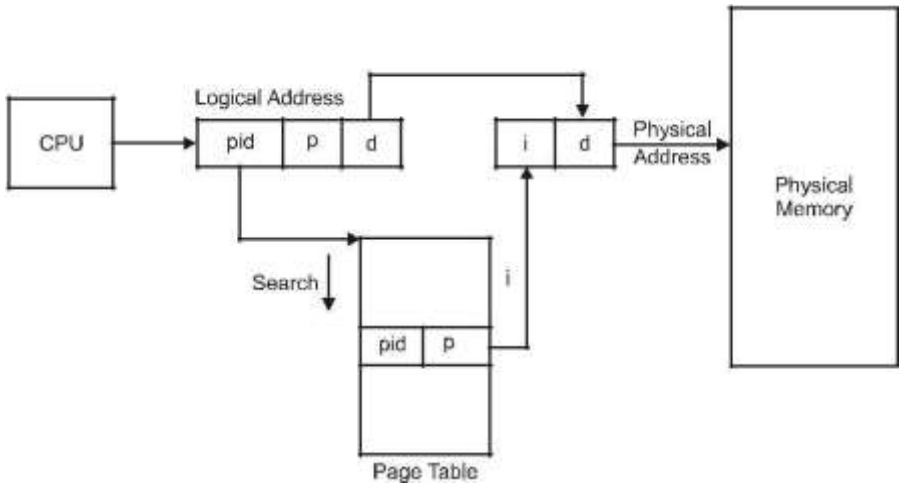


Fig.: Inverted Page Table

– The above figure shows operation of an inverted page table.

– Inverted page table often requires that an address space identifier to be stored in each entry of page table.

– Each virtual address in system consists of triple <process-id, page-number, offset>

– Each inverted page table entry is a pair of <process-id, page number> where process id assumes role of address space identifier.

Segmentation

- Segmentation is one of memory management techniques in operating system, which supports users view of memory.
- As paging suffers from internal fragmentation so to overcome this issue segmentation is introduced.
- A logical address space is a collection of segments with each segment has it's own segment name and length.
- Segment is considered as logical unit such as main program, procedure and function.
- The program is also a collection of segments.
- Generally user prefers to view memory as collection of variable size segments as shown in below figure.

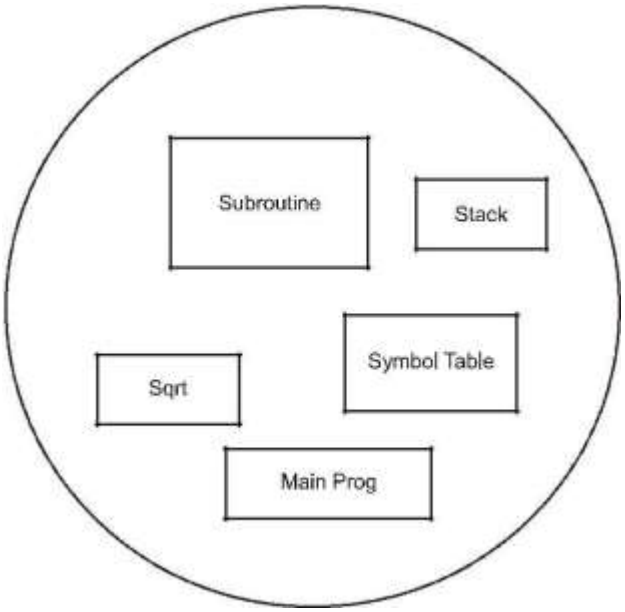


Fig.: User View of Program

— For simplicity of implementation, segments are numbered and referred by segment number than by segment name. Thus logical address consists of two tuple.

<segment number, offset>

Segmentation Hardware

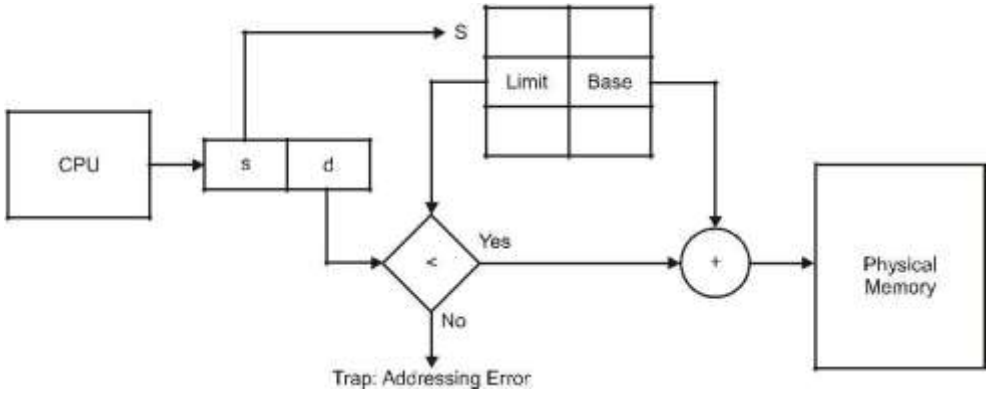
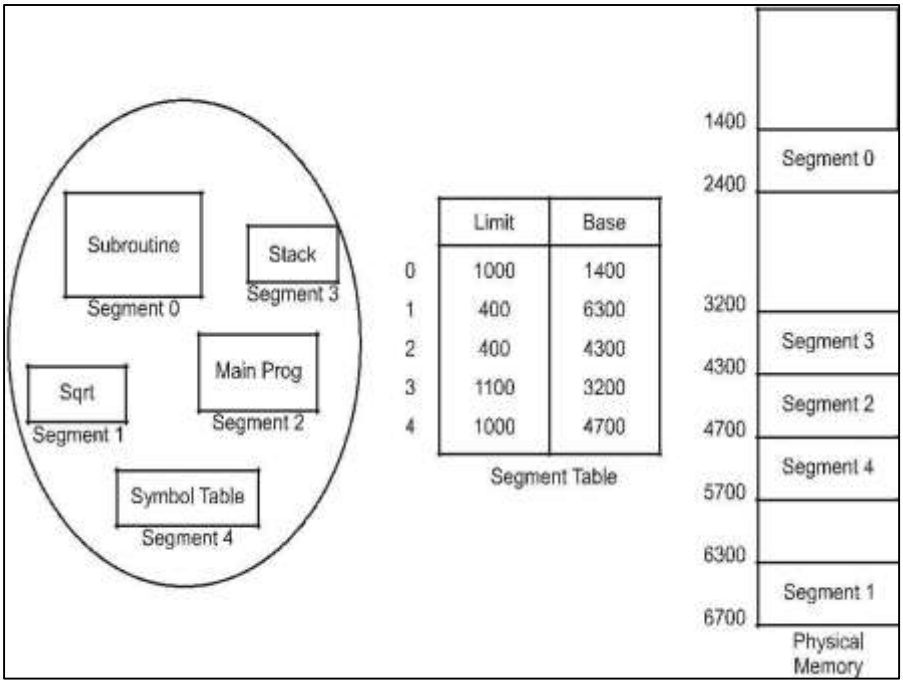


Fig.: Segmentation Hardware

- Above figure shows segmentation hardware. Here each entry in segment table has segment base and segment limit. The segment base contains starting physical address where segment resides in memory while segment limit specifies length of segment.
- The logical Address consists of 2 parts segment number and an offset into that segment d.
- When offset is legal, it is to be added to segment base to produce address in physical memory of desired byte.
- The offset d of logical address must be between 0 and segment limit. Example of Segmentation



- The above figure shows example of segmentation in which we have 5 segments stored in physical memory.
- Here segment table has separate entry for each segment storing the starting address and length for each segment.
- For example segment 0 is 1000 bytes long and begins at location 1400. So reference to byte 50 of segment 0 is mapped to location 1400+50=1450.

S.NO	Paging	Segmentation
1.	In paging, the program is divided into fixed or mounted size pages.	In segmentation, the program is divided into variable size sections.
2.	For the paging operating system is accountable.	For segmentation compiler is accountable.
3.	Page size is determined by hardware.	Here, the section size is given by the user.
4.	It is faster in comparison to segmentation.	Segmentation is slow.
5.	Paging could result in internal fragmentation.	Segmentation could result in external fragmentation.
6.	In paging, the logical address is split into a page number and page offset.	Here, the logical address is split into section number and section offset.
7.	Paging comprises a page table that encloses the base address of every page.	While segmentation also comprises the segment table which encloses segment number and segment offset.
8.	The page table is employed to keep up the page data.	Section Table maintains the section data.
9.	In paging, the operating system must maintain a free frame list.	In segmentation, the operating system maintains a list of holes in the main memory.

S.NO	Paging	Segmentation
10.	Paging is invisible to the user.	Segmentation is visible to the user.
11.	In paging, the processor needs the page number, and offset to calculate the absolute address.	In segmentation, the processor uses segment number, and offset to calculate the full address.
12.	It is hard to allow sharing of procedures between processes.	Facilitates sharing of procedures between the processes.
13	In paging, a programmer cannot efficiently handle data structure.	It can efficiently handle data structures.
14.	This protection is hard to apply.	Easy to apply for protection in segmentation.
15.	The size of the page needs always be equal to the size of frames.	There is no constraint on the size of segments.
16.	A page is referred to as a physical unit of information.	A segment is referred to as a logical unit of information.
17.	Paging results in a less efficient system.	Segmentation results in a more efficient system.

Virtual Memory Management

- When our program that we want to execute in main memory having size larger than main memory, so at that time some part of secondary is used as primary memory that is called virtual memory.
- 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 user from physical memory.
- This technique frees programmers from concerns of memory storage limitations.

- Virtual memory also allows processes to share files easily and to implement shared memory.
- Virtual memory is not easy to implement also it may decrease performance if it is used carelessly.
- User would be able to write programs for an extremely large virtual address space as each user program could take less physical memory. So more programs could be run at the same time, with corresponding increase in CPU utilization and throughput.

- Less I/O would be needed to load or swap each user program into memory, so such user program would run faster.

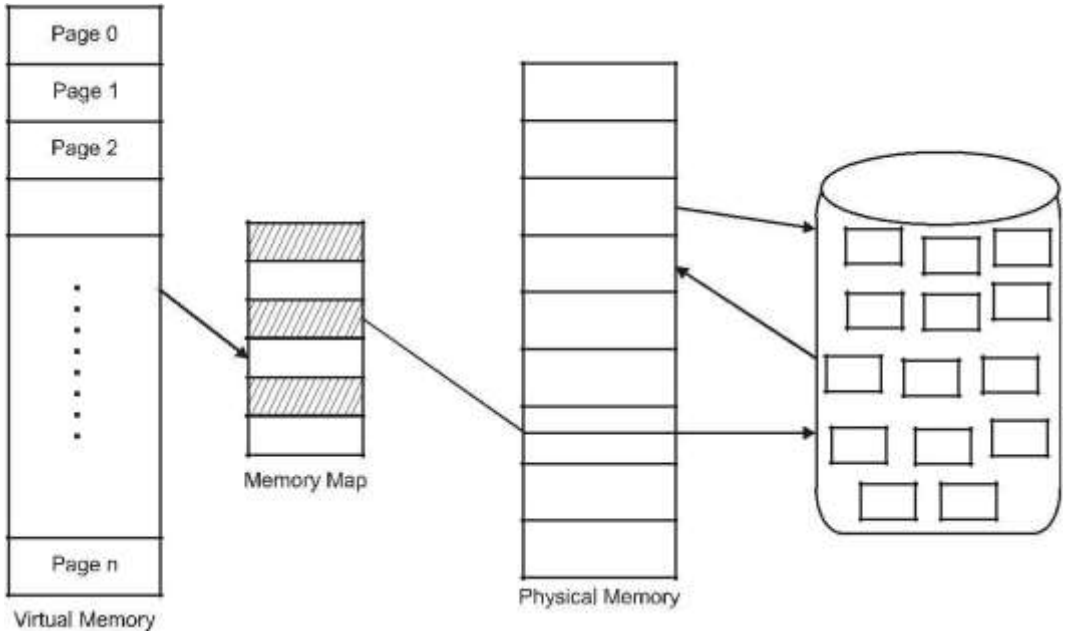


Fig.: Working of Virtual Memory

- Above figure shows how virtual memory works, in which there is separation of logical memory as perceived by users from physical memory.
- This separation allows programmer to have large virtual memory though only smaller physical memory is available.
- Virtual memory also provides mechanism for process creation using fork() and vfork().
- Virtual memory is generally implemented by demand paging and demand segmentation.

Prepaging in Virtual Memory System

As suggested by the concept of virtual memory, it is not necessary that the entire process should be loaded into the main memory at the given time. The process can be executed efficiently if only some of the pages are present in the main memory at a particular time. Now the question which arises here is what is the basis of the selection of pages to be loaded into the main memory for execution of a process beforehand?

The concept of **Prepaging** is used as an attempt to reduce the large number of page faults that occur at the start of a process where the basic strategy is to bring all the pages into the memory that will be needed at the same time before they are actually referenced by the

process.

Prepaging is used to overcome a major drawback of demand paging. A major drawback of **Demand Paging** is a significantly large number of page faults which may occur as soon as a process starts to execute. The situation is an outcome of an effort to load the initial locality into memory. The same situation may arise repeatedly. For example, when a process is restarted after being swapped-out, all its pages are present on the disk and hence each one of the pages must be brought back to the main memory for execution of the process by its own page fault in the worst case.

Advantages:

1. Overlapping the paging I/O with computation.
2. Reducing total run time of the application.
3. Avoiding the overhead of predictable page faults.

Disadvantages:

1. The prepaging system call is advisory only.

2. There is no indication when pages have arrived.
3. There is no indication of which page has arrived.
4. **Difference between Demand Paging and Prepaging :**

Demand Paging	Pre-Paging
Any page is not loaded into the main memory unless it is being referenced by the process at the present instant.	All the pages are loaded into the memory that will be needed at the same time in near future but before they are actually referenced by the process.
The number of page faults is significantly high.	The number of page faults may be reduced in certain specific cases.
The time taken to load the pages can not decrease in any situation.	The time taken to load the pages decreases when consecutive addresses are referenced by a process.
The pages loaded in the main memory are certainly used.	The pages loaded in the main memory might or might not be used.
There is no wastage of resources as a page is loaded as and when needed.	There is a wastage of resources as there is a high chance that the pages might be unused.

Benefits of Virtual Memory with user and system perspective.

There are many advantages of virtual memory, some of which listed below. Advantages of VM with system point of view.

- It is possible to run large application with RAM having less capacity.
- Degree of multiprogramming can be achieved due to more than one process running at once.
- Execution time of application is less due to file mapping.
- Efficient use of memory.

Advantages of VM with user point of view:

- User satisfaction as more number of applications are running at a time.

- Provides support for fault tolerance.
- No need to purchase more new memory.
- Multiple OS environment can exist on same computer.

Common Advantages:

- **More processes may be maintained in the main memory:** Because we are going to load only some of the pages of any particular process, there is room for more processes. This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in the ready state at any particular time.
- **A process may be larger than all of the main memory:** One of the most fundamental restrictions in programming is lifted. A process larger than the main memory can be executed because of demand paging. The OS itself loads pages of a process in the main memory as required.
- It allows greater multiprogramming levels by using less of the available (primary) memory for each process.
- It has twice the capacity for addresses as main memory.
- It makes it possible to run more applications at once.
- Users are spared from having to add memory modules when RAM space runs out, and applications are liberated from shared memory management.
- When only a portion of a program is required for execution, speed has increased.
- Memory isolation has increased security.
- It makes it possible for several larger applications to run at once.
- Memory allocation is comparatively cheap.
- It doesn't require outside fragmentation.
- It is efficient to manage logical partition workloads using the CPU.
- Automatic data movement is possible.

Demand Paging:

- During the execution time of program, entire program is loaded into main memory from secondary memory, but there may be the case that we may not initially need entire program in main memory, but require only some part of that program.
- So in order to solve above problem the concept of demand paging was introduced
- The demand paging is a technique which allows to load only some selected pages on demand to main memory from secondary memory.
- It is commonly used in virtual memory system.

- A demand paging system is similar to paging system with swapping shown in figure below.

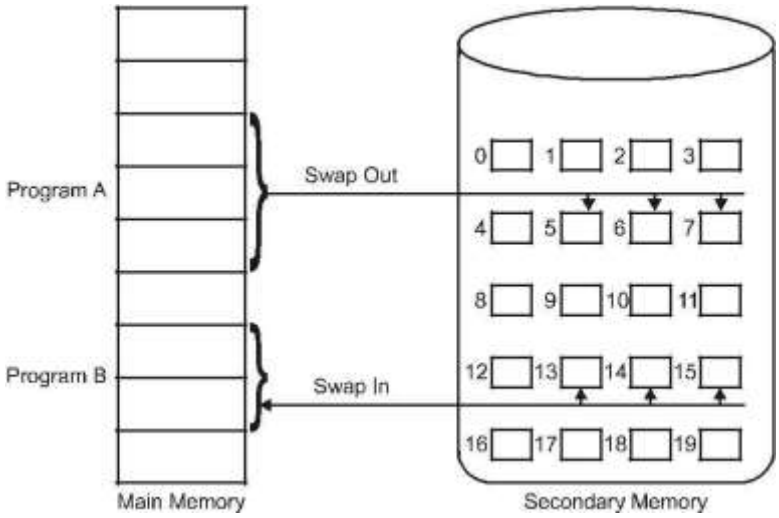


Fig.: Concept of Demand Paging with Swapping

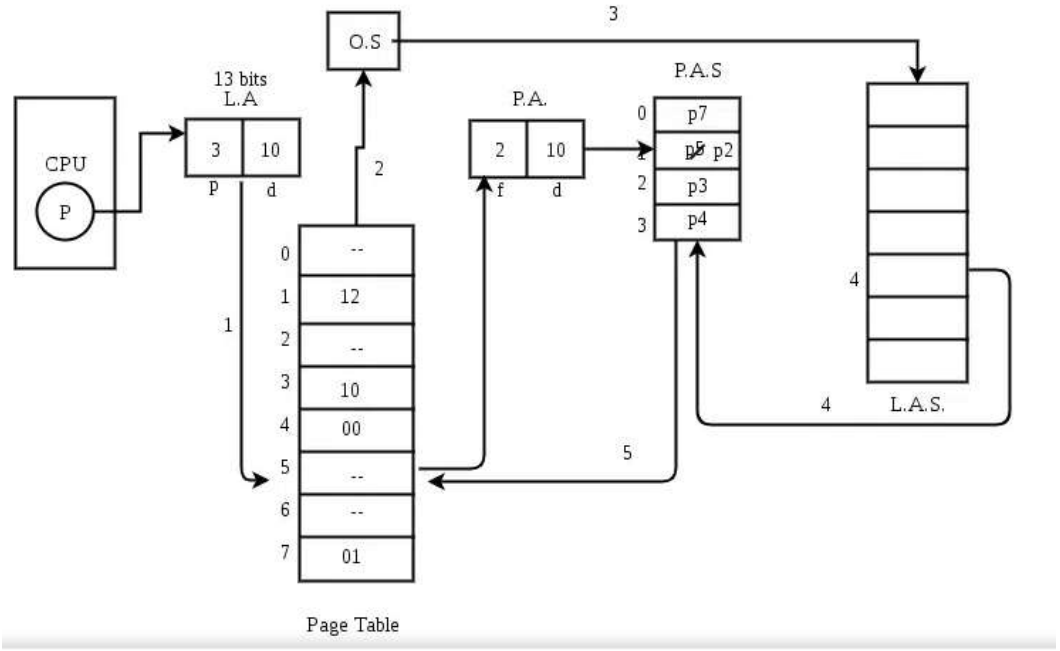
- As shown in above processes reside in secondary memory, when we want to execute process, we swap it into main memory.
- Instead of swapping entire process into main memory, we make use of lazy swapper to load only selected pages in to main memory when it is needed.
- As swapper manipulates entire processes, where as pager concerned with individual pages of process.
- Thus we use pager than swapper in connection with demand paging.

Advantages:

1. Less memory is needed.
2. Faster response.
3. Higher degree of multiprogramming.

Demand Paging

The process of loading the page into memory on demand (whenever a page fault occurs) is known as demand paging. The process includes the following steps are as follows:



Demand Paging

1. If the CPU tries to refer to a page that is currently not available in the main memory, it generates an interrupt indicating a memory access fault.
2. The OS puts the interrupted process in a blocking state. For the execution to proceed the OS must bring the required page into the memory.
3. The OS will search for the required page in the logical address space.
4. The required page will be brought from logical address space to physical address space. The page replacement algorithms are used for the decision-making of replacing the page in physical address space.
5. The page table will be updated accordingly.
6. The signal will be sent to the CPU to continue the program execution and it will place the process back into the ready state.

Page Fault Mechanism

- Page fault is a situation where process tries to access a page that was not brought into memory.
- In page fault there is failure of paging hardware in translating the address through the page table due to the invalid bit is set, so it causes trap to OS.
- This trap is the result of failure of OS to bring desired page into memory.

- Procedure for handling page fault:

1. Initially we check an internal table to determine whether reference was a valid or invalid memory access for particular process.
2. If we found reference was invalid then we terminate the process. Else if it was valid but we have not yet brought that page so we have to bring that page into memory.
3. We find a free frame.
4. We schedule disk operation to read desired page into newly allocated frame.
5. When disk read is complete, we modify internal table kept with process and page table to indicate that the page is now in memory.
6. We restart instruction that was interrupted by trap. The process can now access the page as though it had always been in memory.

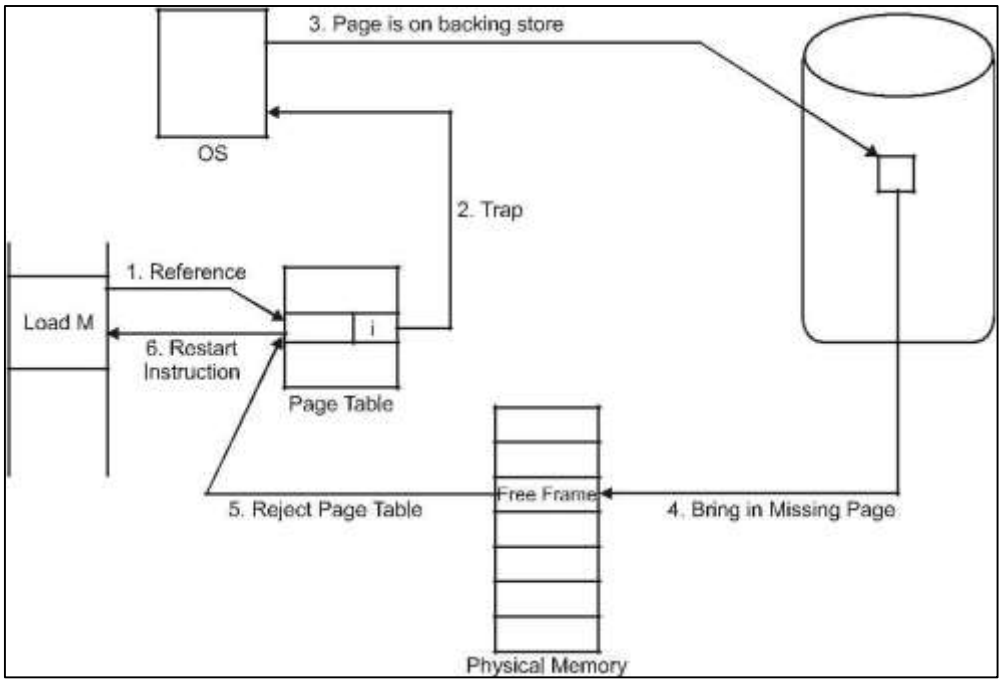


Fig.: Steps in handling page fault

Detailed Procedure:-

- The computer hardware traps to the kernel and program counter (PC) is saved on the stack. Current instruction state information is saved in CPU registers.
- An assembly program is started to save the general registers and other volatile information to keep the OS from destroying it.

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

Page Replacement

- It's a technique in which there is replacement of page from main memory to free the frame for new pages, when total memory requirements exceed the capacity of main memory.
- If no frame is free, we find one that is not currently being used and free it.
- We can free frame by writing its contents to swap space and changing the page table to indicate that the page is no longer in memory.

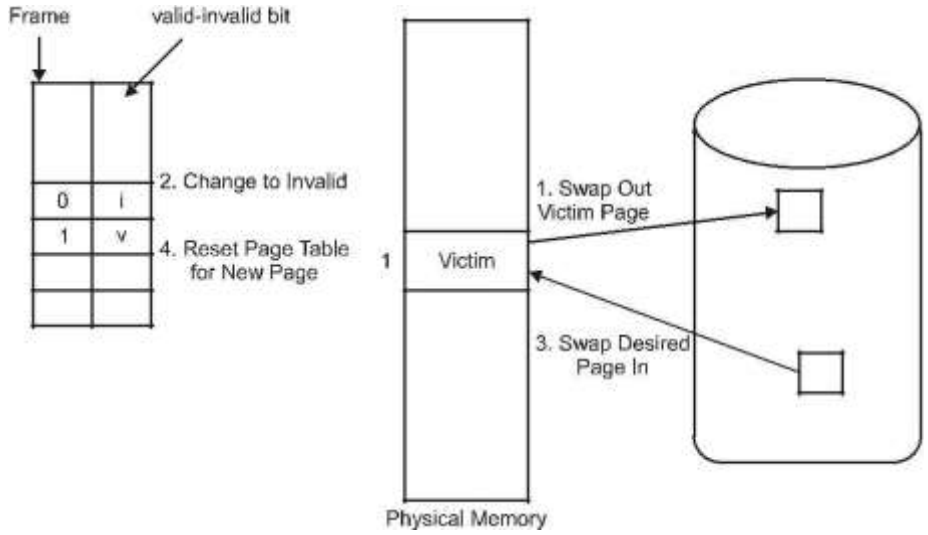


Fig.: Page Replacement

– The above figure shows page replacement mechanism which is explained below.

1. Find the location of desired page on disk.
2. Find a free frame
   - If there is a free frame, use it.
   - If there is no free frame, use page replacement algorithm to select a victim frame.
   - Write victim frame to disk, change the page and frame tables accodingly.
3. Read the desired page into newly freed frame, change the page and frame tables.
4. Restart the user process.

Page Fault: A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page faults happen. In case of a page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

Page Replacement Algorithms

- There are three basic algorithms that are frequently used for the selection of page to replace.
- Page replacement algorithm decides which memory page to swap out when a page of memory needs to be allocated.
- Algorithms
  1. FIFO
  2. Optimal
  3. LRU

FIFO (First In First Out):

- This is one of simplest pages replacement algorithm.
- According to this algorithm oldest page is replaced with new page.
- FIFO algorithm associates with each page, the time when that page was brought into memory.
- It is not necessary in this algorithm to record time when page is brought in.
- We can create a FIFO queue to hold all pages in memory.
- Generally we replace the page at head of queue, when page is brought into memory and insert it at tail of the queue.
- For example the reference string given to us is 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0,
- 1, 7, 0, 1 and frame size is three, which are initially empty.
- The first three references (7, 0, 1) cause page fault and are brought into these empty frames.
- Next Reference 2 replaces 7 as it was brought in first then next is 0 which does not cause page fault as 0 is already in memory.
- Reference to 3 results in replacement of page 0 as it is first in then next reference 0 will cause page fault and replace page 1, similarly the procedure is continued for rest of string.
- This algorithm will cause total 15 page faults which is shown in figure below. Reference String

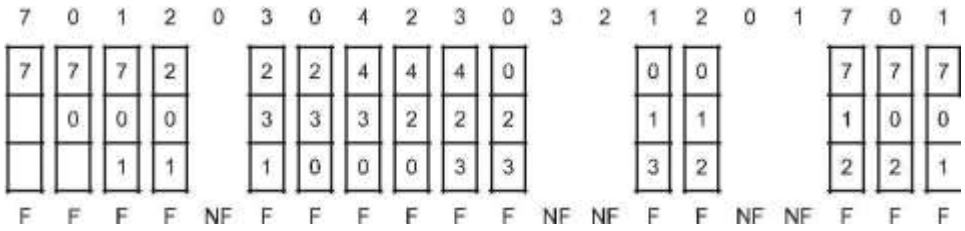


Fig.: FIFO Page Replacement Algorithm

- FIFO page replacement algorithm is easy to understand and program but its performance is not always good.
- This algorithm suffers from the problem of Belady's Anomaly as number of frames increases then number of faults may or may not be increase.

Belady's Anomaly:

In Operating System, process data is loaded in fixed sized chunks and each chunk is referred to as a page. The processor loads these pages in the fixed sized chunks of memory called frames. Typically the size of each page is always equal to the frame size.

A page fault occurs when a page is not found in the memory, and needs to be loaded from the disk. If a page fault occurs and all memory frames have been already allocated, then replacement of a page in memory is required on the request of a new page. This is referred to as demand-paging. The choice of which page to replace is specified by a page replacement algorithms. The commonly used page replacement algorithms are FIFO, LRU, optimal page replacement algorithms etc.

Generally, on increasing the number of frames to a process' virtual memory, its execution becomes faster as less number of page faults occur. Sometimes the reverse happens, i.e. more number of page faults occur when more frames are allocated to a process. This most unexpected result is termed as **Belady's Anomaly**.

Béládý's anomaly is the name given to the phenomenon where increasing the number of page frames results in an increase in the number of page faults for a given memory access pattern.
This phenomenon is commonly experienced in following page replacement algorithms:

1. First in first out (FIFO)
2. Second chance algorithm
3. Random page replacement algorithm

Reason of Belady's Anomaly –
The other two commonly used page replacement algorithms are Optimal and LRU, but Belady's Anamoly can never occur in these algorithms for any reference string as they belong to a class of stack based page replacement algorithms.
A **stack based algorithm** is one 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. For LRU replacement, the set of pages in memory would be the *n* most recently referenced pages. If the number of frames increases then these *n* pages will still be the most recently referenced and so, will still be in the memory. While in FIFO, if a page named *b* came into physical memory before a page – *a* then priority of replacement of *b* is greater than that of *a*, but this is not independent of the number of page frames and hence, FIFO does not follow a stack page replacement policy and therefore suffers Belady's Anomaly.
Example: Consider the following diagram to understand the behaviour of a stack-based page replacement algorithm

The diagram illustrates that given the set of pages i.e. {0, 1, 2} in 3 frames of memory is not a subset of the pages in memory – {0, 1, 4, 5} with 4 frames and it is a violation in the property of stack based algorithms. This situation can be frequently seen in FIFO algorithm.

Belady's Anomaly in FIFO –
Assuming a system that has no pages loaded in the memory and uses the FIFO Page replacement algorithm. Consider the following reference string:

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

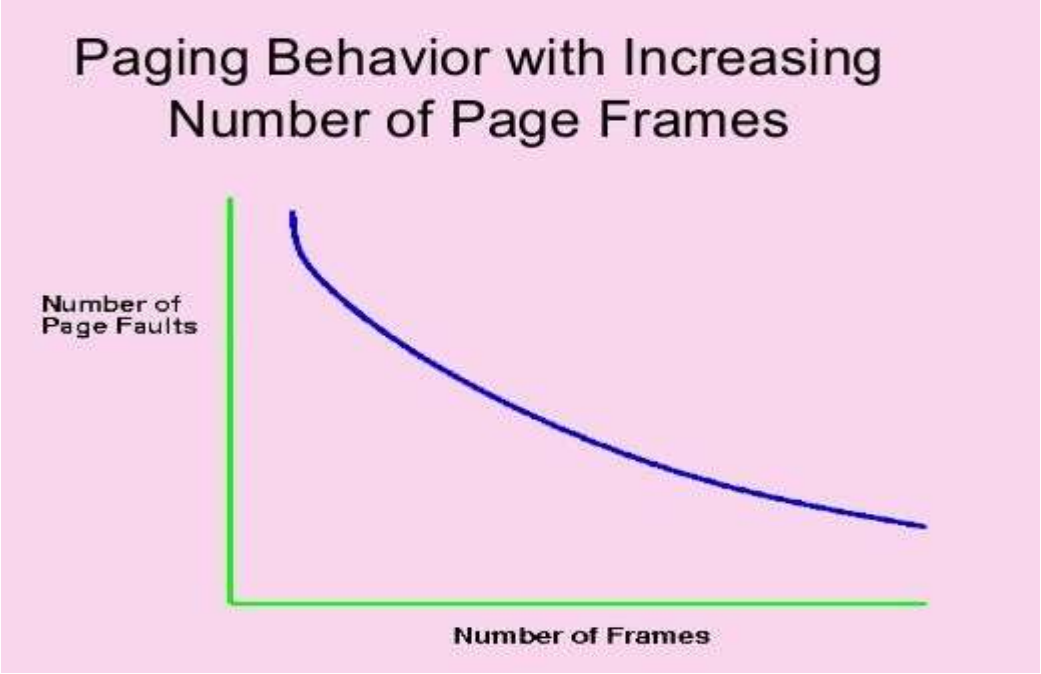
Case-1: If the system has 3 frames, the given reference string on using FIFO page replacement algorithm yields a total of 9 page faults. The diagram below illustrates the pattern of the page faults occurring in the example.

1	1	1	2	3	4	1	1	1	2	5	5
	2	2	3	4	1	2	2	2	5	3	3
		3	4	1	2	5	5	5	3	4	4
PF	PF	PF	PF	PF	PF	PF	X	X	PF	PF	X

Case-2: If the system has 4 frames, the given reference string on using FIFO page replacement algorithm yields a total of 10 page faults. The diagram below illustrates the pattern of the page faults occurring in the example.

1	1	1	1	1	1	2	3	4	5	1	2
	2	2	2	2	2	3	4	5	1	2	3
		3	3	3	3	4	5	1	2	3	4
			4	4	4	5	1	2	3	4	5
PF	PF	PF	PF	X	X	PF	PF	PF	PF	PF	PF

It can be seen from the above example that on increasing the number of frames while using the FIFO page replacement algorithm, the number of **page faults increased** from 9 to 10.
Note – It is not necessary that every string reference pattern cause Belady anomaly in FIFO but there are certain kind of string references that worsen the FIFO performance on increasing the number of frames



Optimal Page Replacement Algorithm

- This is one of the best page replacement algorithm which has lowest page fault rate compared to other algorithm.
- This algorithm never suffers from Belady's Anomaly.
- This algorithm replaces the page that will not be used for longest period of time.
- This algorithm ensures lowest possible page fault rate for fixed number of frames.
- For example consider the same string as 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 with frame size 3.
- First three references causes fault that will fill 3 empty frames.
- Reference to page 2 replaces page 7 as it will not be used for longer period of time as compare to other page 0 and 1.
- Then page 0 will not cause fault as it is already available in memory.
- Then for page 3, it will replace with page 1, at it will not be used for longer period of time compare to 2 and 0.
- Same procedure will be continue for rest of string.
- This algorithm causes 9 page faults as shown in figure below.

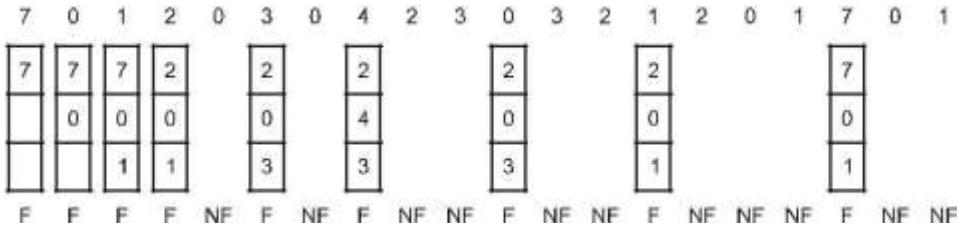


Fig.: Optimal page replacement algorithm

- The optimal page replacement is twice as good as FIFO algorithm.
- No any other page replacement algorithm can process this reference string in 3 frames, with fewer than 9 faults.
- This algorithm is difficult to implement as it requires future knowledge of reference string.

LRU Page Replacement

- [Replace a page which is not in use for a long time]
- This page replacement algorithm will replace the page that has not been used for the longest period of time.
- It associates with each page at the time of that page's last use.
- This algorithm looks backward in time as compared to optimal which looks forward in time.
- For example consider same reference string as referred for FIFO and Optimal as 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 with frame size 3.
- In this case first 5 faults are same as those for optimal replacement.
- Here first 3 references cause fault that will fill 3 empty frames.
- Then page 2 will replace page 7 as it has not been used for longest period of time compared to 0 and 4, then page 0 will not cause fault as it is already present in

memory.

- Next page 3 replaces page 1 as it has not been used for longest period of time compared to 2 and 0, then again for page 0, no fault will occur.

- Next page 4 will replace page 2 as it has not been used for longest period of time compared to 0 and 3.
- Same procedure will be continued for rest of string.
- This algorithm produces 12 faults for above reference string, as shown in figure below.

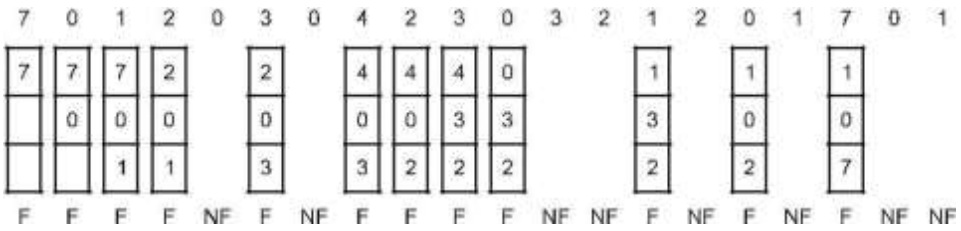


Fig.: LRU Page Replacement Algorithm

- LRU replacement with 12 faults is much better than FIFO with 15 faults.
- LRU page replacement may require substantial hardware assistance.
- The problem in LRU is to determine order for frames defined by time of last use.
- Like optimal replacement, LRU does not suffer from Belady's anomaly.
- Both optimal and LRU belong to class of algorithm called stack algorithm, that can never exhibit Belady's anomaly.

Example: Example: Consider the Pages referenced by the CPU in the order are 6, 7, 8, 9, 6, 7, 1, 6, 7, 8, 9, 1

Ans:-

FIFO:- Number of Page Faults = 9 [3 frames] for 4 frames 10 page faults

Example: Consider the Pages referenced by the CPU in the order are 6, 7, 8, 9, 6, 7, 1, 6, 7, 8, 9, 1, 7, 9, 6

Ans:- Optimal:- The number of Page Faults = 8 [3 frames]

Example: Consider the Pages referenced by the CPU in the order are 6, 7, 8, 9, 6, 7, 1, 6, 7, 8, 9, 1, 7, 9, 6

Ans:- The number of Page Faults = 12 [3 frames]

Thrashing

- Thrashing is a Phenomenon in which processor spends most of time in swapping the pages only rather than executing instructions.
- Thrashing results in severe performance problem.
- It's a very high paging activity which leads to low CPU utilization.
- Consider a situation when process does not have pages, then it leads to page fault, so in this case only disk usage increases but CPU usage decreases.
- In order to maintain degree of multiprogramming generally more processes are added into main memory however each process gets less pages so it leads to page fault which results in high disk usage low CPU utilization.
- This activity is also called as thrashing where only page operation as swap out and swap in is carried out unnecessarily without any proper outcome.

Fig.: Thrashing

- The above figure shows the mechanism of thrashing whereas degree of multiprogramming increases then CPU utilization also increases initially up to maximum is reached.
- But after maximum is reached, if degree of multiprogramming increases, thrashing occurs and CPU utilization drops sharply.
- So in order to increase CPU utilization and stop thrashing, we must decrease degree of multiprogramming.
- We can limit the effects of thrashing by using local replacement algorithm.
- To prevent thrashing we must provide a process with as many frames as it needs.
- But problem is how to determine how many frames it needs.
- There are few techniques as working set strategy and locality model are available to determine frame needs.
- The system with thrashing can either be very slow or comes to halt.
- We can say thrashing is a condition in which excessive paging operation takes place.

Working Set Model

- The working set model is based on the assumption of locality.
- This model uses parameter to define working set window.
- It examines the most recent D page references where set of pages in the most recent D page references in the working set.
- If page is in active use, it will be in the working set. If it is no longer being used, it will drop from working set D time unit after last reference.
- The working set is an approximation of the programs locality.
- The most important property of working set is its size. If we compute working set size, WSSi for each process in the system, we can consider that

D= S WSSi

Here Demand,
D > M ----- Not allowed
D< M ------- is allowed

- Where D is total demand for frames.
- Consider the string and use working set model of window size=10.
- Page reference table is

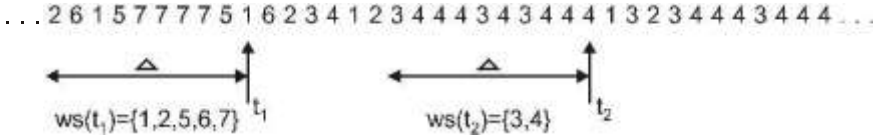


Fig.: Working set model

- The above figure shows mechanism of working set model in which working set at time t1 {1,2,5,6,7} then by time t2, working set has changed to {3,4}.

Working set represented by Delta (Δ) Here there are three conditions

1. If Delta is too small then it will not cover all the locality of reference(pages) so it will leads to larger page fault
2. If Delta is too large then large space is required and other inactive page also saves.
3. If Delta is infinity then entire program pages are store.

Working Sets: conceptual model proposed by Peter Denning to prevent thrashing.

- Informal definition: the collection of pages a process is using actively, and which must thus be memory-resident to prevent this process from thrashing.
- If the sum of all working sets of all runnable threads exceeds the size of memory, then stop running some of the threads for a while.
- Divide processes into two groups: active and inactive:
  - ✓ When a process is active its entire working set must always be in memory: never execute a thread whose working set is not resident.
  - ✓ When a process becomes inactive, its working set can migrate to disk.
  - ✓ Threads from inactive processes are never scheduled for execution.
  - ✓ The collection of active processes is called the *balance set.*
  - ✓ The system must have a mechanism for gradually moving processes into and out of the balance set.
  - ✓ As working sets change, the balance set must be adjusted.

Garbage collection

Garbage collection (GC) in an operating system (OS) is a dynamic memory

management process that finds stranded blocks of memory and reallocates them. Like the noisy garbage trucks that rumble down our streets, a GC algorithms' job is to find stranded addresses, unused addresses, and clean them up. How this is performed can vary, as there are different strategies, or algorithms, to achieve this purpose.

In computer science, **garbage collection** (**GC**) is a form of automatic memory management. The *garbage collector*, or just *collector*, attempts to reclaim *garbage*, or memory occupied by objects that are no longer in use by the program.

Garbage collection is an automatic memory management feature in many modern programming languages, such as Java and languages in the .NET framework. Languages that use garbage collection are often interpreted or run within a virtual machine like the JVM. In each case, the environment that runs the code is also responsible for garbage collection.

Advantages

Garbage collection frees the programmer from manually dealing with memory deallocation. As a result, certain categories of bugs are eliminated or substantially reduced:

- Dangling pointer bugs, which occur when a piece of memory is freed while there are still pointers to it, and one of those pointers is dereferenced. By then the memory may have been reassigned to another use, with unpredictable results.
- Double free bugs, which occur when the program tries to free a region of memory that has already been freed, and perhaps already been allocated again.
- Certain kinds of memory leaks, in which a program fails to free memory occupied by objects that have become unreachable, which can lead to memory exhaustion. (Garbage collection typically does not deal with the unbounded accumulation of data that is reachable, but that will actually not be used by the program.)
- Efficient implementations of persistent data structures

Disadvantages

Typically, garbage collection has certain disadvantages,

- Including consuming additional resources,
- Performance impacts,
- Possible stalls in program execution,
- Incompatibility with manual resource management

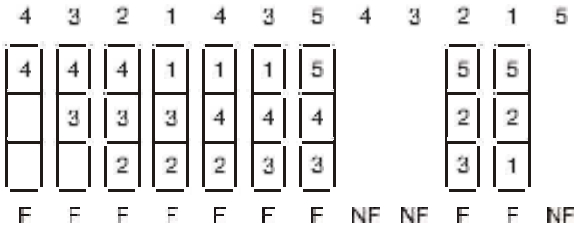
In order to improve performance and reduce fragmentation, GC can be accomplished through a few key algorithms. Let's look at mark-and-sweep, semi-space, and mark- claim-compact.

Numericals based on page replacement policy

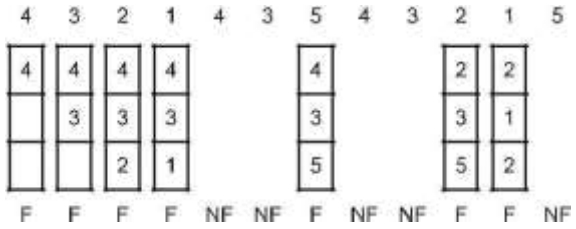
Q.1 Consider following page reference string 4 3 2 1 4 3 5 4 3 2 1 5

Assume frame size = 3. How many page fault would occur for FIFO, Optimal and LRU algorithm? (W08-5M)

For FIFO

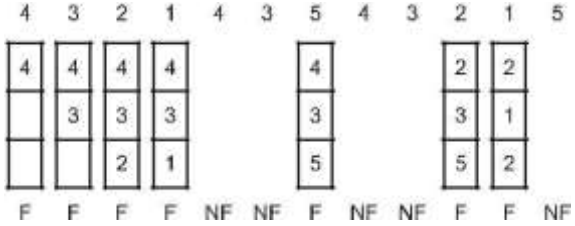


Total number of page faults = 9. For Optimal



Total number of page fault = 7.

For LRU



Total number of page fault = 10

Here F = Page Fault NF = Number of page fault.

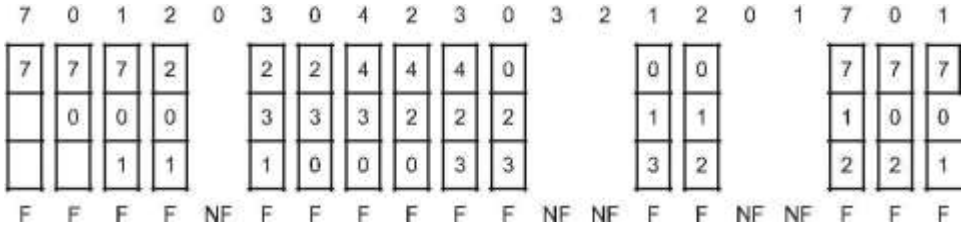
Q.2 Consider page reference string and find out page fault rate for each algorithm given below

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

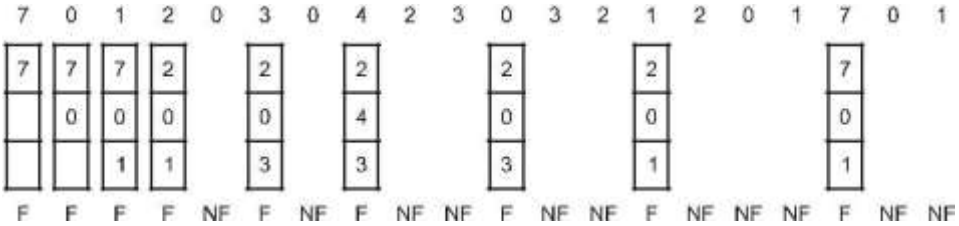
(Use 3 and 4 frames)

1. FIFO, 2. Optimal, 3. LRU (S10-9M, W12-6M)

For Frame Size = 3 For FIFO

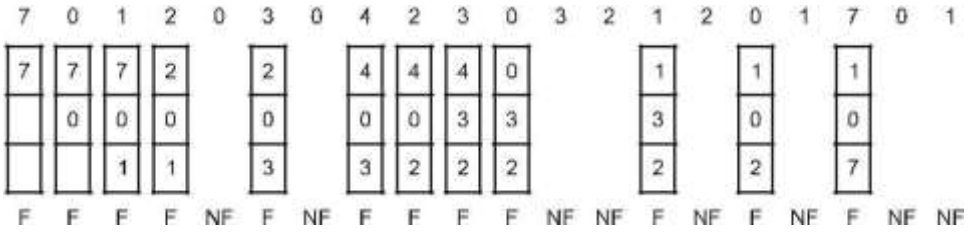


Total number of page fault=15. For Optimal



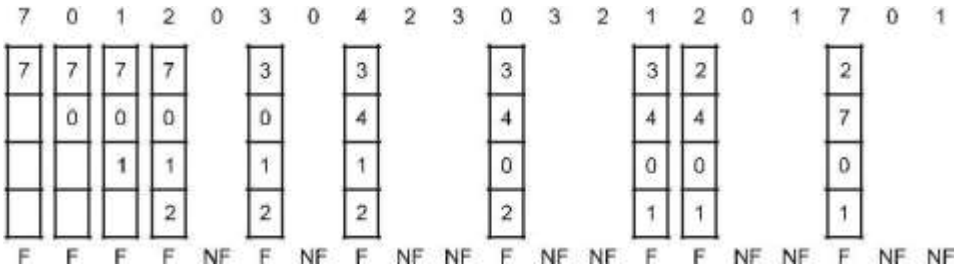
Total number of page faults=9

For LRU

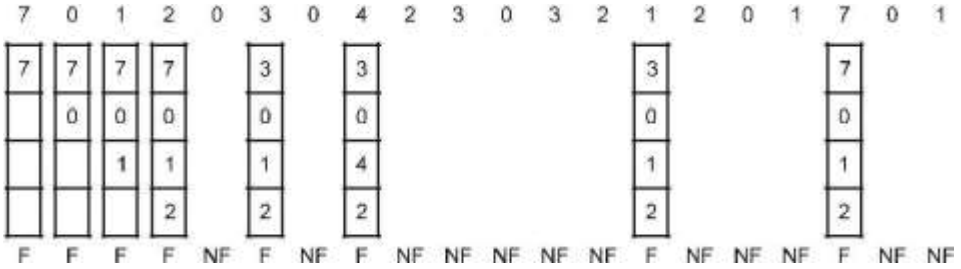


Total number of page faults=12 For Frame Size = 4

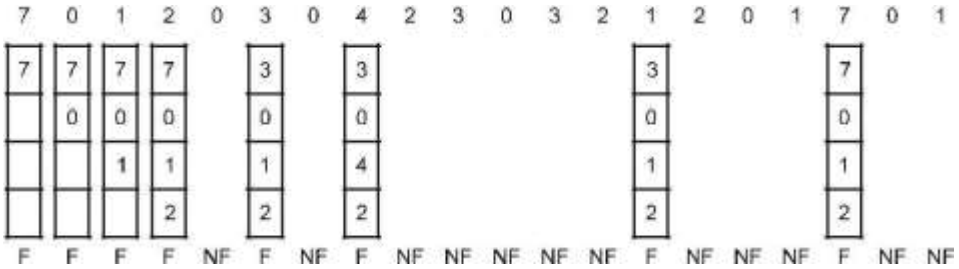
For FIFO



Total number of page faults=10. For Optimal



Total number of page faults= 8. For LRU



Total number of page faults = 8.

Numerical Based on Segmenta tion

Q.1 Consider following segment table (W09-5M)

Seg	Base	Length/Limit
0	219	600
1	2300	14
3	1327	580
4	1952	96

What are physical addresses of following logical addresses?

1. 0,430 2. 1, 10 3. 2,500 4. 3,400 5. 4, 112

1. 0,430

Here given logical address is <0,430> Seg No. = 0 and Offset = 430

Step 1 – Check availability of seg number in segment table Here Seg number=0 is present in segment table.

Step 2 – Check whether given offset is less than limit or not i.e. offset<limit. Here offset=430<limit=600

Step 3 – Calculate physical address using formula PA = Base + offset

Here PA = 219+430=649

2. 1,10

Step 1: Seg No. = 1 Present in segment table

Step 2: Offset=10 < Limit=14 Step 3: PA = Base+Offset

= 2300+10

= 2310

3. 2,500

– Here as seg no.=2 is not present in segment table so we can not calculate physical address for above given logical address.

– In this case trap is generated and control transfer to OS.

4. 3,400

Step 1 : Seg No.=3 is present in seg table Step 2 : Offset=400 < Limit=580

Step 3 : PA =Base+Offset

=1327+400

1727

5. 4,112

Step 1 : Seg No.=4 is present in seg table Step 2 : Offset=112<Limit=96

Here as offset is not less than limit so trap is generated and physical address can not be calculated.

Q. 2 Consider the following segment table

Segment	Base	Length/Limit
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are physical address for following logical addresses

1. 0,430 2. 1,10 3. 1,11 4. 2,500 5. 3,400 (W10-5M)

1. 0,430

Step 1 : Seg No. = 0 is present in seg table. Step 2 : Offset=430<Limit=600

Step3 : PA = Base + Offset

= 219+430

= 649

2. 1,10

Step 1 : Seg No.=1 is present in Seg table Step 2 : Offset=10<Limit=14

Step 3 : PA=Base+Offset

=2300+10

=2310

3. 1,11

Step 1 : Seg No.=1 is present in seg table Step 2 : Offset=11<Limit=14

Step 3 : PA=Base+Offset

=2300+11

=2311

4. 2,500

Step 1 : Seg No.=2 is present in seg table Step 2 : Offset=500<Limit=100

Step 3 : Trap is generated and physical address can not be calculated.

5. 3,400

Step 1 : Seg No.=3 is present in seg table Step 2 : Offset=400<580

Step 3 : PA=Base+Offset

=1327+400

=1727

Q. 3 On a system using simple segmentation compute physical address for each of following logical address, if address generates segment fault then indicate so,

Segement	Base	Legth/ Limit
0	330	124
1	876	125
2	111	99
3	49	302

	8	

1. 0,99	2 . 2 , 7 8	3. 1,26 8	4. 3,2 22	5. 0,111	(S12-4M, S13-4M)

1. 0,99

Step 1 : Seg No.=0 is present in table Step 2 : Offset=99<Limit=124

Step 3 : PA=Base+Offset

=330+99

=429

2. 2,78

Step 1 : Seg No.=2 is present in table Step 2 : Offset=78<Limit=99

Step 3 : PA=Base+Offset

=111+78

=189

3. 1,268

Step 1 : Seg No.=1 is present in table Step 2 : Offset=268<Limit=99

Trap is generated and physical address can not be calculated.

4. 3,222

Step 1 : Seg No.=3 is present in table Step 2 : Offset=222<302

Step 3 : PA=Base+Offset

=498+222

=720

5. 0,111

Step 1 : Seg No.=0 is present in table Step 2 : Offset=111<Limit=124

Step 3 : PA=Base+Offset

=330+111

=441