MODULE 4

MANAGEMENT

Background:

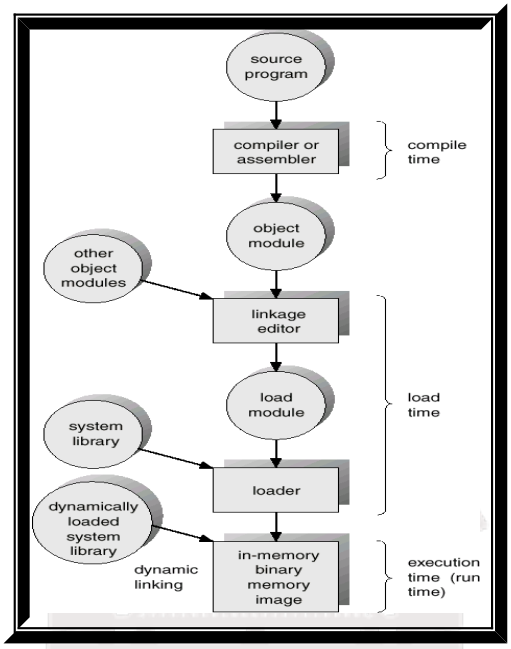
- Memory management is concerned with managing the primary memory.
- Memory consists of array of bytes or words each with their own address.
- The instructions are fetched from the memory by the cpu based on the value program counter.

Functions of memory management:-

- Keeping track of status of each memory location..
- Determining the allocation policy.
- Memory allocation technique.
- De-allocation technique.

Address Binding:-

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



Binding of instructions and data can be done at any step along the way:-

1. Compile time:-

   If we know whether the process resides in memory then absolute code can be generated. If the static address changes then it is necessary to re-compile the code from the beginning.

2. Load time:-

   If the compiler doesn't know whether the process resides in memory then it generates the re-locatable code. In this the binding is delayed until the load time.

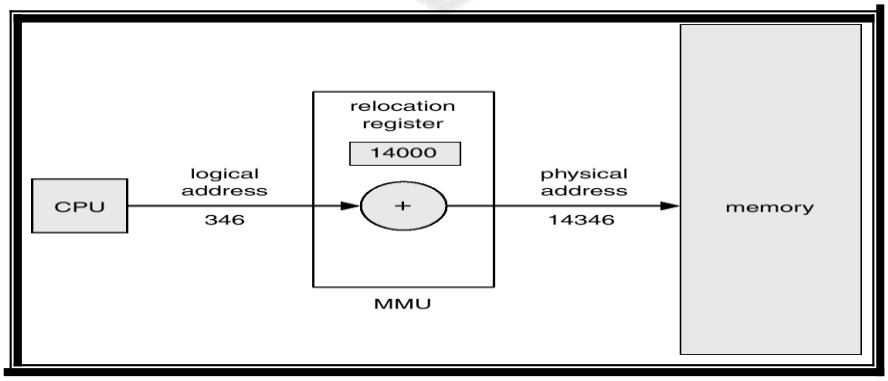
3. Execution time:-

   If the process is moved during its execution from one memory segment to another then the binding is delayed until run time. Special hardware is used for this. Most of the general purpose operating system uses this method.

Logical versus physical address:-

- The address generated by the CPU is called logical address or virtual address.
- The address seen by the memory unit i.e., the one loaded in to the memory register is called the physical address.

- Compile time and load time address binding methods generate some logical and physical address.
- The execution time addressing binding generate different logical and physical address.
- Set of logical address space generated by the programs is the logical address space.
- Set of physical address corresponding to these logical addresses is the physical address space.
- The mapping of virtual address to physical address during run time is done by the hardware device called memory management unit (MMU).
- The base register is also called re-location register.
- Value of the re-location register is added to every address generated by the user process at the time it is sent to memory.
- User program never sees the real physical address.
- The figure below shows the dynamic relation. Dynamic relation implies mapping from virtual address space to physical address space and is performed at run time usually with some hardware assistance.
- Relocation is performed by the hardware and is invisible to the user. Dynamic relocation makes it possible to move a partially executed process from one area of memory to another without affecting.



Dynamic re-location using a re-location registers The above figure shows that dynamic re-location which implies mapping from virtual addresses space to physical address space and is performed by the hardware at run time.

Re-location is performed by the hardware and is invisible to the user dynamic relocation makes it possible to move a partially executed process from one area of memory to another without affecting.

Dynamic Loading:-

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

Advantage:-

- Gives better memory utilization.
- Unused routine is never loaded.
- Do not need special operating system support.
- This method is useful when large amount of codes are needed to handle in frequently occurring cases.

Dynamic linking and Shared libraries:-

- Some operating system supports only the static linking.
- In dynamic linking only the main program is loaded in to the memory. If the main program requests a procedure, the procedure is loaded and the link is established at the time of references. This linking is postponed until the execution time.
- With dynamic linking a “stub” is used in the image of each library referenced routine. A “stub” is a piece of code which is used to indicate how to locate the appropriate memory resident library routine or how to load library if the routine is not already present.
- When “stub” is executed it checks whether the routine is present is memory or not. If not it loads the routine in to the memory.

- This feature can be used to update libraries i.e., library is replaced by a new version and all the programs can make use of this library.
- More than one version of the library can be loaded in memory at a time and each program uses its version of the library. Only the program that are compiled with the new version are affected by the changes incorporated in it. Other programs linked before new version is installed will continue using older libraries this type of system is called "shared library".

Overlays:-

- The size of the process is limited to the size of physical memory. If the size is more than the size of physical memory then a technique called overlays is used.
- The idea is to load only those instructions and data that are needed at any given time. When other instructions are needed, they are loaded in to memory apace that was previously occupied by the instructions that are no longer needed.

  Eg:-

  Consider a 2-pass assembler where pass-1 generates a symbol table and pass-2 generates a machine code.

  Assume that the sizes of components are as follows:

  Pass-1 = 70k

  Pass-2 = 80k

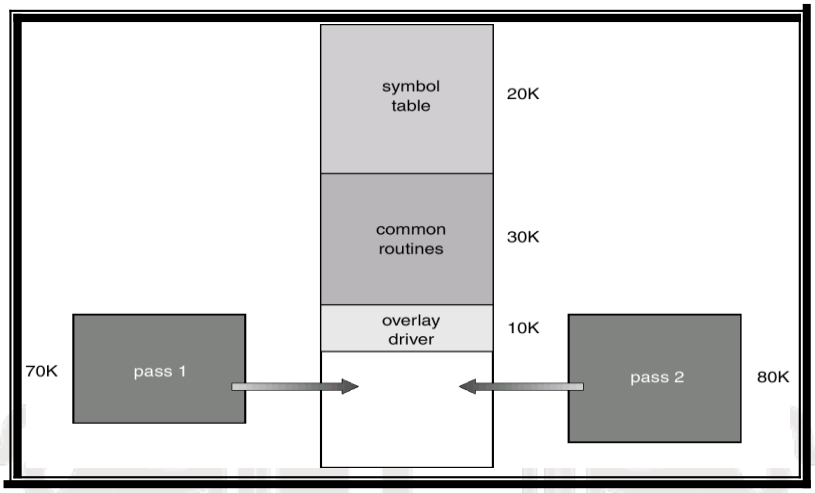
  Symbol table = 20k

  Common routine = 30k

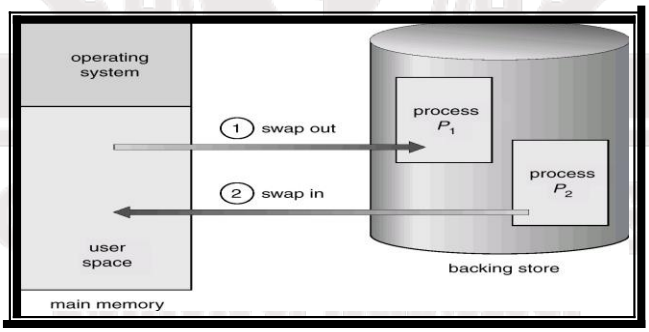
  To load everything at once, it requires 200k of memory. Suppose if 150k of memory is available, we can't run all the components at same time.

- Thus we define 2 overlays, overlay A which consist of symbol table, common routine and pass-1 and overlay B which consists of symbol table, common routine and pass-2.

- We add an overlay driver and start overlay A in memory. When we finish pass-1 we jump to overlay driver, then the control is transferred to pass-2.
- Thus we can run assembler in 150k of memory.
- The code for overlays A and B are kept on disk as absolute memory images. Special re-location and linking algorithms are needed to construct the overlays. They can be implemented using simple file structures.



Swapping:-



- Swapping is a technique of temporarily removing inactive programs from the memory of the system.
- A process can be swapped temporarily out of the memory to a backing store and then brought back in to the memory for continuing the execution. This process is called swapping.

Eg:- In a multi-programming environment with a round robin CPU scheduling whenever the time quantum expires then the process that has just finished is swapped out and a new process swaps in to the memory for execution.

- A variation of swap is priority based scheduling. When a low priority is executing and if a high priority process arrives then a low priority will be swapped out and high priority is allowed for execution. This process is also called as Roll out and Roll in.
- Normally the process which is swapped out will be swapped back to the same memory space that is occupied previously. This depends upon address binding.
- If the binding is done at load time, then the process is moved to same memory location.
- If the binding is done at run time, then the process is moved to different memory location. This is because the physical address is computed during run time.
- Swapping requires backing store and it should be large enough to accommodate the copies of all memory images.
- The system maintains a ready queue consisting of all the processes whose memory images are on the backing store or in memory that are ready to run.
- Swapping is constant by other factors:-
    - To swap a process, it should be completely idle.
    - A process may be waiting for an i/o operation. If the i/o is asynchronously accessing the user memory for i/o buffers, then the process cannot be swapped.

Contiguous Memory Allocation:-

Memory allocation

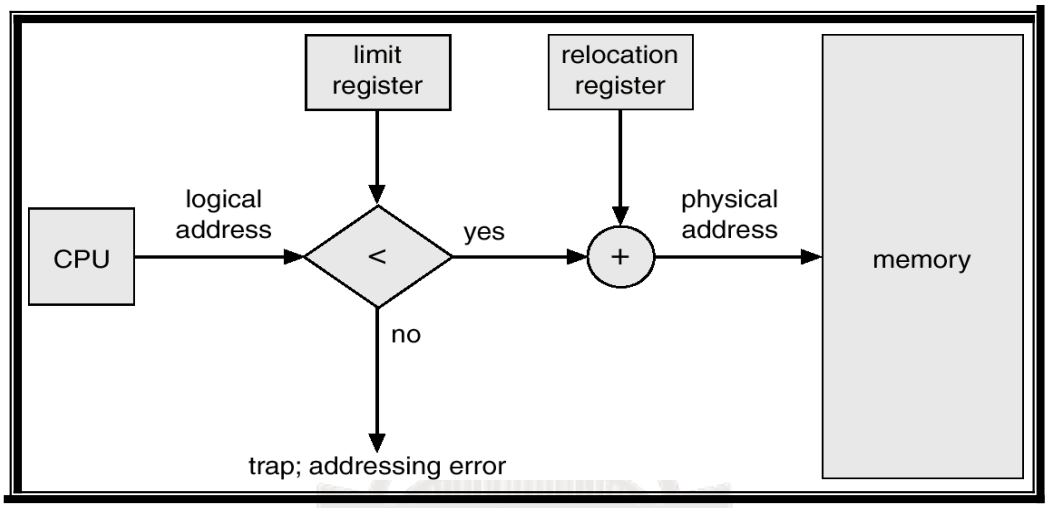
- One of the simplest method for memory allocation is to divide memory in to several fixed partition. Each partition contains exactly one process. The degree of multi-programming depends on the number of partitions.
- In multiple partition method, when a partition is free, process is selected from the input queue and is loaded in to free partition of memory.
- When process terminates, the memory partition becomes available for another process.
- Batch OS uses the fixed size partition scheme.
- The OS keeps a table indicating which part of the memory is free and is occupied.
- When the process enters the system it will be loaded in to the input queue. The OS keeps track of the memory requirement of each process and the amount of memory available and determines which process to allocate the memory.

- When a process requests, the OS searches for large hole for this process, hole is a large block of free memory available.
- If the hole is too large it is split in to two. One part is allocated to the requesting process and other is returned to the set of holes.
- The set of holes are searched to determine which hole is best to allocate. There are three strategies to select a free hole:-
  - First bit:- Allocates first hole that is big enough. This algorithm scans memory from the beginning and selects the first available block that is large enough to hold the process.
  - Best bit:- It chooses the hole i.e., closest in size to the request. It allocates the smallest hole i.e., big enough to hold the process.
  - Worst fit:- It allocates the largest hole to the process request. It searches for the largest hole in the entire list.
- First fit and best fit are the most popular algorithms for dynamic memory allocation. First fit is generally faster. Best fit searches for the entire list to find the smallest hole i.e., large enough. Worst fit reduces the rate of production of smallest holes.
- All these algorithms suffer from fragmentation.

Memory mapping and Protection:-

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

- Since every address generated by the CPU is checked against these register we can protect the OS and other users programs and data from being modified.



Fragmentation:-

- Memory fragmentation can be of two types:-
  - Internal Fragmentation
  - External Fragmentation
- In Internal Fragmentation there is wasted space internal to a portion due to the fact that block of data loaded is smaller than the partition.

Eg:- If there is a block of 50kb and if the process requests 40kb and if the block is allocated to the process then there will be 10kb of memory left.

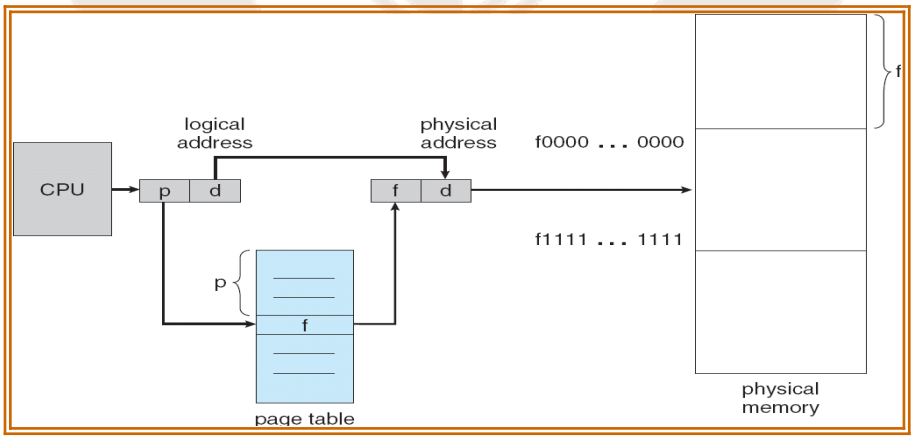
- External Fragmentation exists when there is enough memory space exists to satisfy the request, but it not contiguous i.e., storage is fragmented in to large number of small holes.
- External Fragmentation may be either minor or a major problem.
- One solution for over-coming external fragmentation is compaction. The goal is to move all the free memory together to form a large block. Compaction is not possible always. If the re-location is static and is done at load time then compaction is not possible. Compaction is possible if the re-location is dynamic and done at execution time.
- Another possible solution to the external fragmentation problem is to permit the logical address space of a process to be non-contiguous, thus allowing the process to be allocated physical memory whenever the latter is available.

Paging:-

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

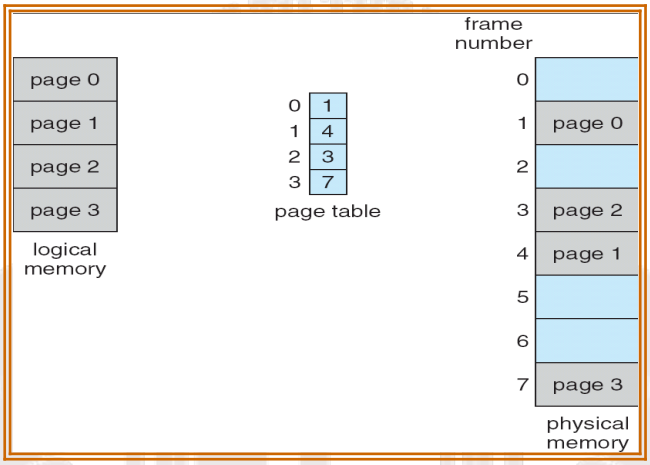
Basic Method:-

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



- Logical address generated by the CPU is divided in to two parts: page number (p) and page offset (d).

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



Eg:- To show how to map logical memory in to physical memory consider a page size of 4 bytes and physical memory of 32 bytes (8 pages).

a. Logical address 0 is page 0 and offset 0. Page 0 is in frame 5. The logical address 0 maps to physical address 20. [(5*4) + 0].
b. Logical address 3 is page 0 and offset 3 maps to physical address 23 [(5*4) + 3].
c. Logical address 4 is page 1 and offset 0 and page 1 is mapped to frame 6. So logical address 4 maps to physical address 24 [(6*4) + 0].
d. Logical address 13 is page 3 and offset 1 and page 3 is mapped to frame 2. So logical address 13 maps to physical address 9 [(2*4) + 1].

Hardware Support for Paging:-

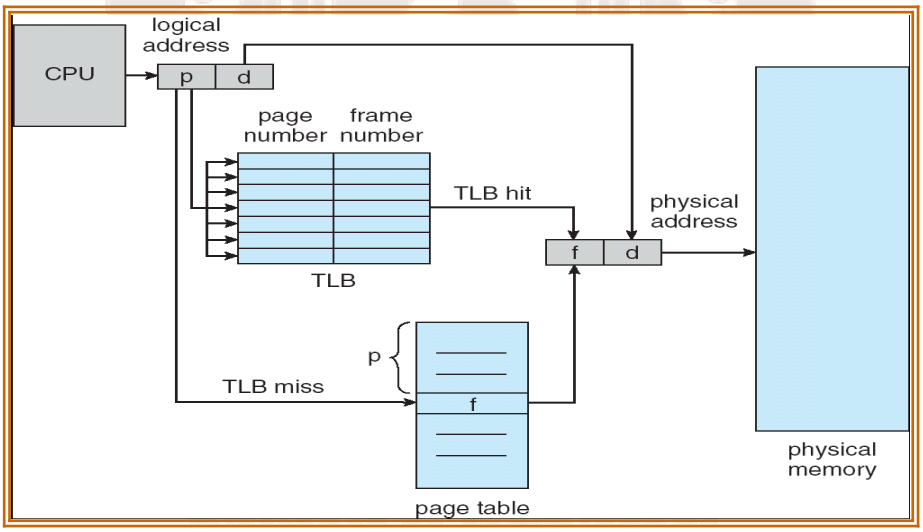
The hardware implementation of the page table can be done in several ways:-

1. The simplest method is that the page table is implemented as a set of dedicated registers. These registers must be built with very high speed logic for making paging

address translation. Every accessed memory must go through paging map. The use of registers for page table is satisfactory if the page table is small.

2. If the page table is large then the use of registers is not visible. So the page table is kept in the main memory and a page table base register [PTBR] points to the page table. Changing the page table requires only one register which reduces the context switching type. The problem with this approach is the time required to access memory location. To access a location [i] first we have to index the page table using PTBR offset. It gives the frame number which is combined with the page offset to produce the actual address. Thus we need two memory accesses for a byte.
3. The only solution is to use special, fast, lookup hardware cache called translation look aside buffer [TLB] or associative register.

- TLB is built with associative register with high speed memory. Each register contains two paths a key and a value.



- When an associative register is presented with an item, it is compared with all the key values, if found the corresponding value field is return and searching is fast.

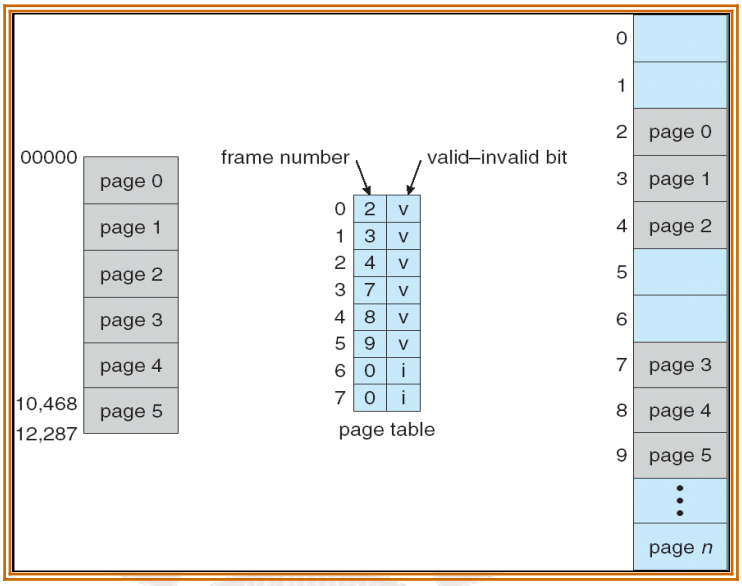
TLB is used with the page table as follows:-

- TLB contains only few page table entries.
- When a logical address is generated by the CPU, its page number along with the frame number is added to TLB. If the page number is found its frame memory is used to access the actual memory.

- If the page number is not in the TLB (TLB miss) the memory reference to the page table is made. When the frame number is obtained use can use it to access the memory.
- If the TLB is full of entries the OS must select anyone for replacement.
- Each time a new page table is selected the TLB must be flushed [erased] to ensure that next executing process do not use wrong information.
- The percentage of time that a page number is found in the TLB is called HIT ratio.

Protection:-

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



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

Structure of the Page Table:-

a. **Hierarchical paging**:-

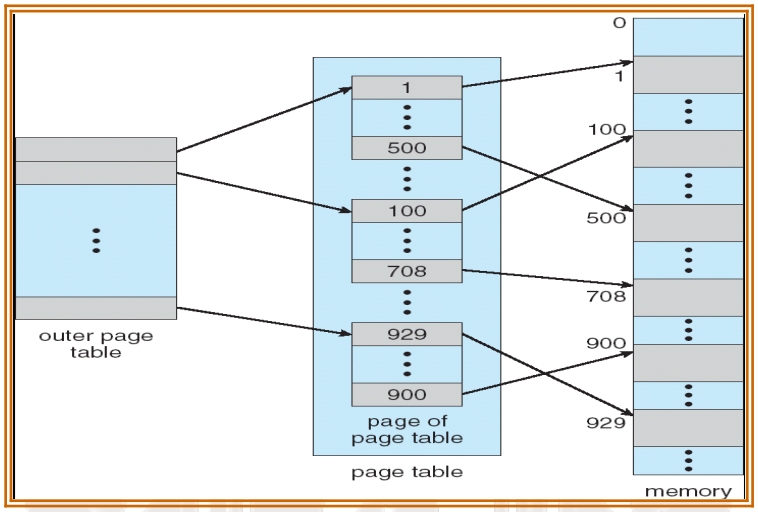
Recent computer system support a large logical address apace from 2^32 to 2^64. In this system the page table becomes large. So it is very difficult to allocate contiguous main memory for page table. One simple solution to this problem is to divide page table in to smaller pieces. There are several ways to accomplish this division.

One way is to use two-level paging algorithm in which the page table itself is also paged.

Eg:- In a 32 bit machine with page size of 4kb. A logical address is divided in to a page number consisting of 20 bits and a page offset of 12 bit. The page table is further divided since the page table is paged, the page number is further divided in to 10 bit page number and a 10 bit offset. So the logical address is

Page number page offset

P1	P2	d
10	10	12



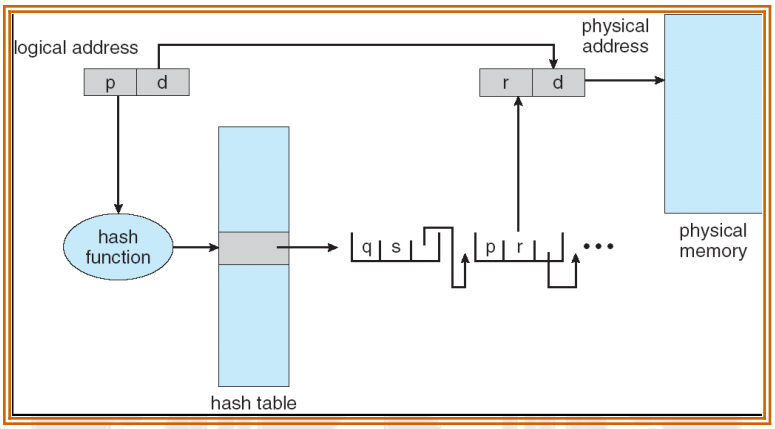
b. **Hashed page table**:-

- Hashed page table handles the address space larger than 32 bit. The virtual page number is used as hashed value. Linked list is used in the hash table which contains a list of elements that hash to the same location.
- Each element in the hash table contains the following three fields:-
  - Virtual page number
  - Mapped page frame value
  - Pointer to the next element in the linked list

Working:-

- Virtual page number is taken from virtual address.
- Virtual page number is hashed in to hash table.
- Virtual page number is compared with the first element of linked list.
- Both the values are matched, that value is (page frame) used for calculating the physical address.

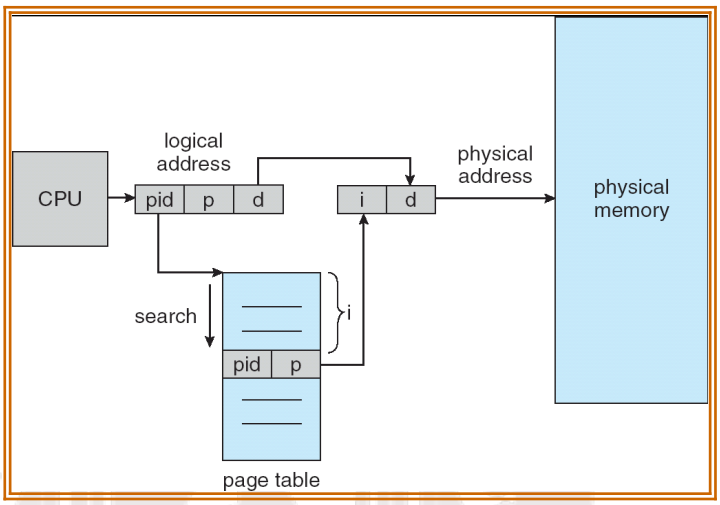
- If not match then entire linked list is searched for matching virtual page number.
- Clustered pages are similar to hash table but one difference is that each entity in the hash table refer to several pages.



c. **Inverted Page Tables**:-

- Since the address spaces have grown to 64 bits, the traditional page tables become a problem. Even with two level page tables. The table can be too large to handle.
- An inverted page table has only entry for each page in memory.
- Each entry consisted of virtual address of the page stored in that read-only location with information about the process that owns that page.
- Each virtual address in the Inverted page table consists of triple <process-id , page number , offset >.
- The inverted page table entry is a pair <process-id , page number>. When a memory reference is made, the part of virtual address i.e., <process-id , page number> is presented in to memory sub-system.
- The inverted page table is searched for a match.
- If a match is found at entry I then the physical address <i , offset> is generated. If no match is found then an illegal address access has been attempted.

- This scheme decreases the amount of memory needed to store each page table, it increases the amount of time needed to search the table when a page reference occurs. If the whole table is to be searched it takes too long.



Advantage:-

- Eliminates fragmentation.
- Support high degree of multiprogramming.
- Increases memory and processor utilization.
- Compaction overhead required for the re-locatable partition scheme is also eliminated.

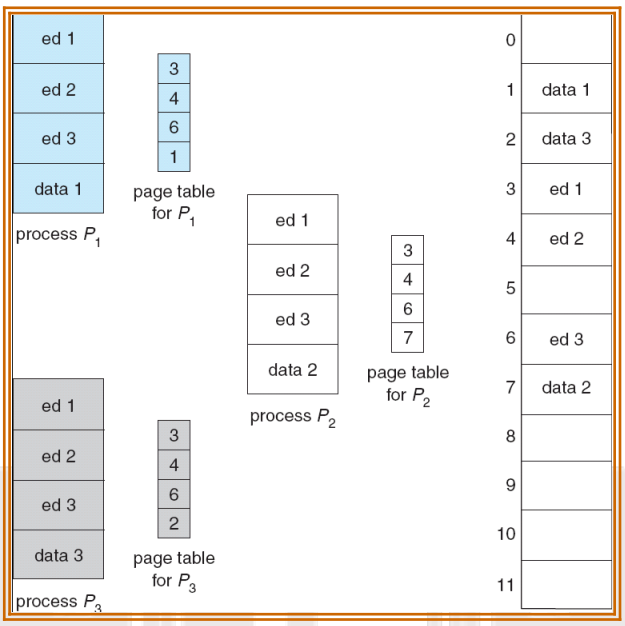
Disadvantage:-

- Page address mapping hardware increases the cost of the computer.
- Memory must be used to store the various tables like page tables, memory map table etc.
- Some memory will still be unused if the number of available block is not sufficient for the address space of the jobs to be run.

Shared Pages:-

Another advantage of paging is the possibility of sharing common code. This is useful in time-sharing environment.

Eg:- Consider a system with 40 users, each executing a text editor. If the text editor is of 150k and data space is 50k, we need 8000k for 40 users. If the code is reentrant it can be shared. Consider the following figure



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

Segmentation:-

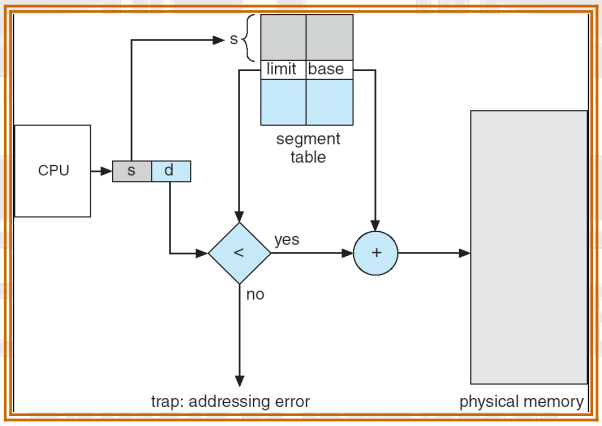
Basic method:-

- Most users do not think memory as a linear array of bytes rather the users thinks memory as a collection of variable sized segments which are dedicated to a particular use such as code, data, stack, heap etc.

- A logical address is a collection of segments. Each segment has a name and length. The address specifies both the segment name and the offset within the segments.
- The users specifies address by using two quantities: a segment name and an offset.
- For simplicity the segments are numbered and referred by a segment number. So the logical address consists of <segment number, offset>.

Hardware support:-

- We must define an implementation to map 2D user defined address in to 1D physical address.
- This mapping is affected by a segment table. Each entry in the segment table has a segment base and segment limit.
- The segment base contains the starting physical address where the segment resides and limit specifies the length of the segment.



The use of segment table is shown in the above figure:-

- Logical address consists of two parts: segment number‘s’ and an offset‘d’ to that segment.
- The segment number is used as an index to segment table.
- The offset ’d’ must bi in between 0 and limit, if not an error is reported to OS.

- If legal the offset is added to the base to generate the actual physical address.
- The segment table is an array of base limit register pairs.

Protection and Sharing:-

- A particular advantage of segmentation is the association of protection with the segments.
- The memory mapping hardware will check the protection bits associated with each segment table entry to prevent illegal access to memory like attempts to write in to read-only segment.
- Another advantage of segmentation involves the sharing of code or data. Each process has a segment table associated with it. Segments are shared when the entries in the segment tables of two different processes points to same physical location.
- Sharing occurs at the segment table. Any information can be shared at the segment level. Several segments can be shared so a program consisting of several segments can be shared.
- We can also share parts of a program.

Advantages:-

- Eliminates fragmentation.
- Provides virtual growth.
- Allows dynamic segment growth.
- Assist dynamic linking.
- Segmentation is visible.

Differences between segmentation and paging:-

Segmentation:-

- Program is divided in to variable sized segments.
- User is responsible for dividing the program in to segments.
- Segmentation is slower than paging.
- Visible to user.
- Eliminates internal fragmentation.

- Suffers from external fragmentation.
- Process or user segment number, offset to calculate absolute address.

Paging:-

- Programs are divided in to fixed size pages.
- Division is performed by the OS.
- Paging is faster than segmentation.
- Invisible to user.
- Suffers from internal fragmentation.
- No external fragmentation.
- Process or user page number, offset to calculate absolute address.

Virtual memory

Virtual memory is a technique that allows for the execution of partially loaded process.

There are many advantages of this:-

- A program will not be limited by the amount of physical memory that is available user can able to write in to large virtual space.
- Since each program takes less amount of physical memory, more than one program could be run at the same time which can increase the throughput and CPU utilization.
- Less i/o operation is needed to swap or load user program in to memory. So each user program could run faster.

Virtual memory is the separation of users logical memory from physical memory. This separation allows an extremely large virtual memory to be provided when these is less physical memory.

Separating logical memory from physical memory also allows files and memory to be shared by several different processes through page sharing.

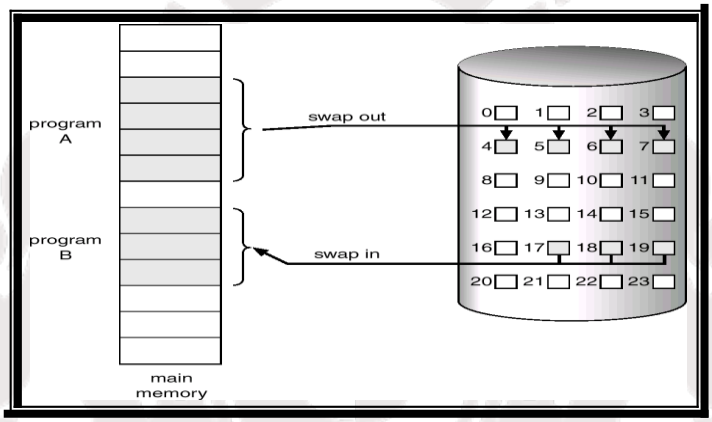
Virtual memory is implemented using Demand Paging.

Demand Paging:-

- A demand paging is similar to paging system with swapping when we want to execute a process we swap the process the in to memory otherwise it will not be loaded in to memory.
- A swapper manipulates the entire processes, where as a pager manipulates individual pages of the process.

Basic concept:-

- Instead of swapping the whole process the pager swaps only the necessary pages in to memory. Thus it avoids reading unused pages and decreases the swap time and amount of physical memory needed.



- The valid-invalid bit scheme can be used to distinguish between the pages that are on the disk and that are in memory.
  - If the bit is valid then the page is both legal and is in memory.
  - If the bit is invalid then either page is not valid or is valid but is currently on the disk.

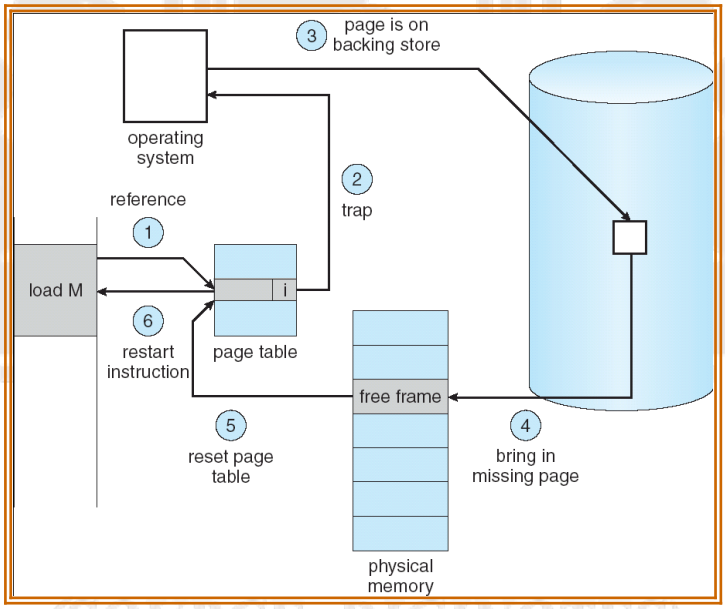
Marking a page as invalid will have no effect if the processes never access to that page. Suppose if it access the page which is marked invalid, causes a page fault trap. This may result in failure of OS to bring the desired page in to memory.

The step for handling page fault is straight forward and is given below:-

1. We check the internal table of the process to determine whether the reference made is valid or invalid.

2. If invalid terminate the process,. If valid, then the page is not yet loaded and we now page it in.
3. We find a free frame.
4. We schedule disk operation to read the desired page in to newly allocated frame.
5. When disk reed is complete, we modify the internal table kept with the process to indicate that the page is now in memory.
6. We restart the instruction which was interrupted by illegal address trap. The process can now access the page.

In extreme cases, we start the process without pages in memory. When the OS points to the instruction of process it generates a page fault. After this page is brought in to memory the process continues to execute, faulting as necessary until every demand paging i.e., it never brings the page in to memory until it is required.



Hardware support:-

For demand paging the same hardware is required as paging and swapping.

1. Page table:- Has the ability to mark an entry invalid through valid-invalid bit.
2. Secondary memory:- This holds the pages that are not present in main memory. It's a high speed disk.

Performance of demand paging:-

Demand paging can have significant effect on the performance of the computer system.

Let P be the probability of the page fault (0<=P<=1)

Effective access time = (1-P) * ma + P * page fault.

Where P = page fault and ma = memory access time.

Effective access time is directly proportional to page fault rate. It is important to keep page fault rate low in demand paging.

A page fault causes the following sequence to occur:-

1. Trap to the OS.
2. Save the user registers and process state.
3. Determine that the interrupt was a page fault.
4. Checks the page references were legal and determine the location of page on disk.
5. Issue a read from disk to a free frame.
6. If waiting, allocate the CPU to some other user.
7. Interrupt from the disk.
8. Save the registers and process states of other users.
9. Determine that the interrupt was from the disk.
10. Correct the page table and other table to show that the desired page is now in memory.
11. Wait for the CPU to be allocated to this process again.
12. Restore the user register process state and new page table then resume the interrupted instruction.

Comparison of demand paging with segmentation:-

Segmentation:-

- Segment may of different size.

- Segment can be shared.
- Allows for dynamic growth of segments.
- Segment map table indicate the address of each segment in memory.
- Segments are allocated to the program while compilation.

Demand Paging:-

- Pages are of same size.
- Pages can't be shared.
- Page size is fixed.
- Page table keeps track of pages in memory.
- Pages are allocated in memory on demand.

Process creation

Copy-on-write:-

Demand paging is used when reading a file from disk in to memory. Fork () is used to create a process and it initially bypass the demand paging using a technique called page sharing. Page sharing provides rapid speed for process creation and reduces the number of pages allocated to the newly created process.

Copy-on-write technique initially allows the parent and the child to share the same pages. These pages are marked as copy-on-write pages i.e., if either process writes to a shared page, a copy of shared page is created.

Eg:- If a child process try to modify a page containing portions of the stack; the OS recognizes them as a copy-on-write page and create a copy of this page and maps it on to the address space of the child process. So the child process will modify its copied page and not the page belonging to parent.

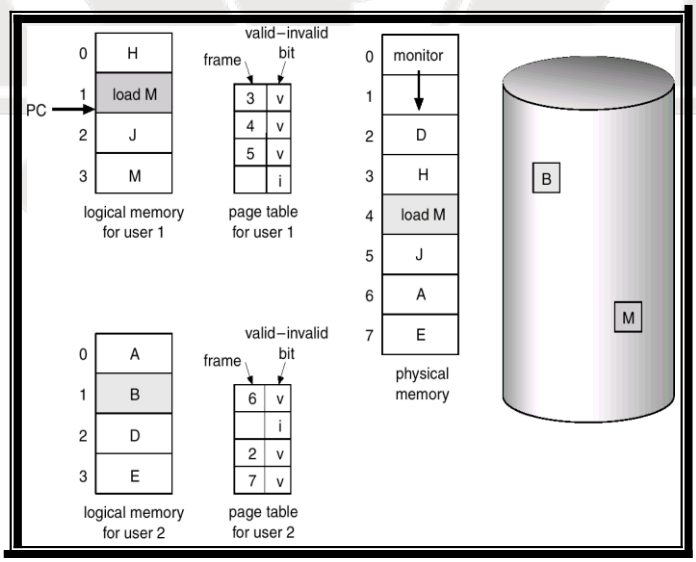
The new pages are obtained from the pool of free pages.

b. **Memory Mapping**:- Standard system calls i.e., open (), read () and write () is used for sequential read of a file. Virtual memory is used for this. In memory mapping a file allows a part

of the virtual address space to be logically associated with a file. Memory mapping a file is possible by mapping a disk block to page in memory.

Page Replacement

- Demand paging shares the I/O by not loading the pages that are never used.
- Demand paging also improves the degree of multiprogramming by allowing more process to run at the some time.
- Page replacement policy deals with the solution of pages in memory to be replaced by a new page that must be brought in. When a user process is executing a page fault occurs.
- The hardware traps to the operating system, which checks the internal table to see that this is a page fault and not an illegal memory access.
- The operating system determines where the derived page is residing on the disk, and this finds that thee are no free frames on the list of free frames.
- When all the frames are in main memory, it is necessary to bring a new page to satisfy the page fault, replacement policy is concerned with selecting a page currently in memory to be replaced.
- The page i,e to be removed should be the page i,e least likely to be referenced in future.

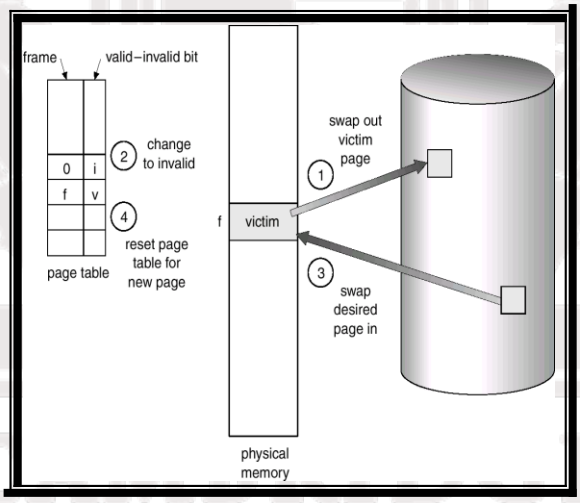


- When a user process is executing a page fault occurs. The OS determines where the desired pages is residing on disk but finds that there are no free frames on the free frame list i.e. all memory is in use as shown in the above figure.

- The OS has several options at this point. It can terminate the process.
- The OS could instead swap out a process by freeing all its frames and reducing the degree of multiprogramming.

Working of Page Replacement Algorithm

1. Find the location of derived page on the disk.
2. Find a free frame
   - If there is a free frame, use it.
   - Otherwise, use a replacement algorithm to select a victim.
   - Write the victim page to the disk; change the page and frame tables accordingly.
3. Read the desired page into the free frame; change the page and frame tables.
4. Restart the user process.



Victim Page

The page that is supported out of physical memory is called victim page.

- If no frames are free, the two page transforms come (out and one in) are read. This will see the effective access time.
- Each page or frame may have a dirty (modify) bit associated with the hardware. The modify bit for a page is set by the hardware whenever any word or byte in the page is written into, indicating that the page has been modified.

- When we select the page for replacement, we check its modify bit. If the bit is set, then the page is modified since it was read from the disk.
- If the bit was not set, the page has not been modified since it was read into memory. Therefore, if the copy of the page has not been modified we can avoid writing the memory page to the disk, if it is already there. Sum pages cannot be modified.

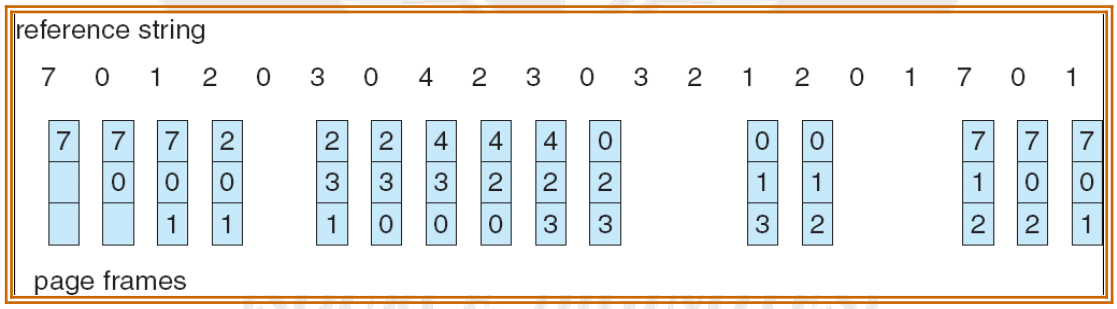
We must solve two major problems to implement demand paging: we must develop a frame allocation algorithm and a page replacement algorithm. If we have multiple processors in memory, we must decide how many frames to allocate and page replacement is needed.

Page replacement Algorithms

FIFO Algorithm:

- This is the simplest page replacement algorithm. A FIFO replacement algorithm associates each page the time when that page was brought into memory.
- When a Page is to be replaced the oldest one is selected.
- We replace the queue at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue.

  Example: Consider the following references string with frames initially empty.



- The first three references (7,0,1) cases page faults and are brought into the empty frames.
- The next references 2 replaces page 7 because the page 7 was brought in first.
- Since 0 is the next references and 0 is already in memory e has no page faults.
- The next references 3 results in page 0 being replaced so that the next references to 0 causer page fault.

This will continue till the end of string.There are 15 faults all together.

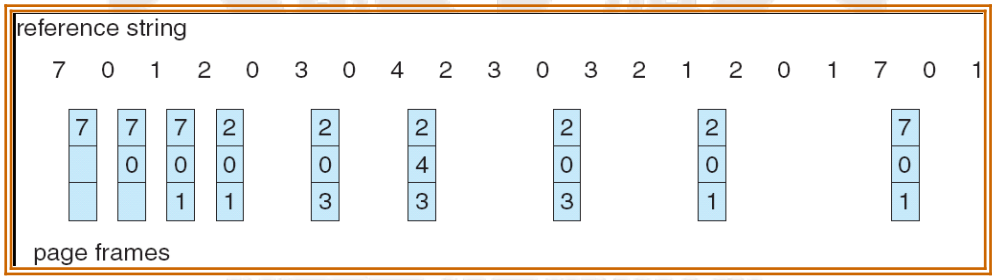
Belady's Anamoly

For some page replacement algorithm, the page fault may increase as the number of allocated frames increases. FIFO replacement algorithm may face this problem.

Optimal Algorithm

- Optimal page replacement algorithm is mainly to solve the problem of Belady's Anamoly.
- Optimal page replacement algorithm has the lowest page fault rate of all algorithms.
- An optimal page replacement algorithm exists and has been called OPT. The working is simple "Replace the page that will not be used for the longest period of time"

Example: consider the following reference string



- The first three references cause faults that fill the three empty frames.
- The references to page 2 replaces page 7, because 7 will not be used until reference 18.
- The page 0 will be used at 5 and page 1 at 14.
- With only 9 page faults, optimal replacement is much better than a FIFO, which had 15 faults.

This algorithm is difficult t implement because it requires future knowledge of reference strings.

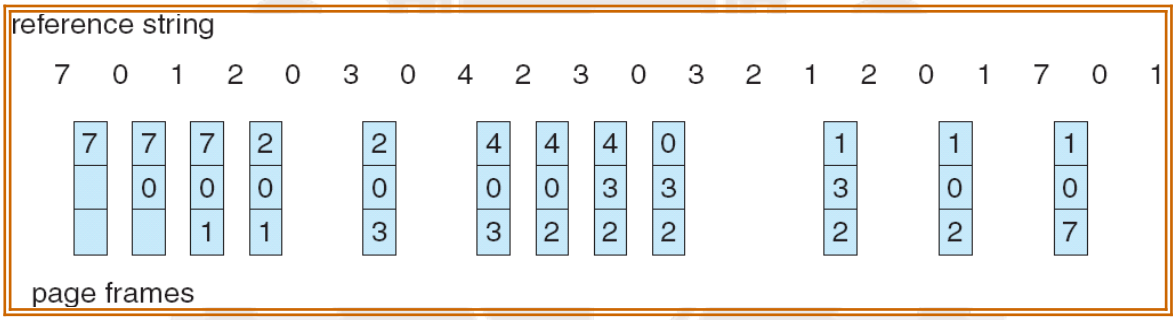
Least Recently Used (LRU) Algorithm

If the optimal algorithm is not feasible, an approximation to the optimal algorithm is possible.

The main difference b/w OPTS and FIFO is that;

- FIFO algorithm uses the time when the pages was built in and OPT uses the time when a page is to be used.
- The LRU algorithm replaces the pages that have not been used for longest period of time.
  - The LRU associated its pages with the time of that pages last use.
  - This strategy is the optimal page replacement algorithm looking backward in time rather than forward.

Ex: consider the following reference string



- The first 5 faults are similar to optimal replacement.
- When reference to page 4 occurs, LRU sees that of the three frames, page 2 as used least recently. The most recently used page is page 0 and just before page 3 was used.

  The LRU policy is often used as a page replacement algorithm and considered to be good. The main problem to how to implement LRU is the LRU requires additional h/w assistance.

Two implementation are possible:

1. **Counters:** In this we associate each page table entry a time -of -use field, and add to the cpu a logical clock or counter. The clock is incremented for each memory reference.

When 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.

In this way we have the time of last reference to each page we replace the page with smallest time value. The time must also be maintained when page tables are changed.

2. **Stack:** Another approach to implement LRU replacement is to keep a stack of page numbers when a page is referenced it is removed from the stack and put on to the top of stack. In this way the top of stack is always the most recently used page and the bottom in least recently used page. Since the entries are removed from the stack it is best implement by a doubly linked list. With a head and tail pointer.

Neither optimal replacement nor LRU replacement suffers from Belady's Anamoly. These are called stack algorithms.

LRU Approximation

- An LRU page replacement algorithm should update the page removal status information after every page reference updating is done by software, cost increases.
- But hardware LRU mechanism tend to degrade execution performance at the same time, then substantially increases the cost. For this reason, simple and efficient algorithm that approximation the LRU have been developed. With h/w support the reference bit was used. A reference bit associate with each memory block and this bit automatically set to 1 by the h/w whenever the page is referenced. The single reference bit per clock can be used to approximate LRU removal.
- The page removal s/w periodically resets the reference bit to 0, write the execution of the users job causes some reference bit to be set to 1.
- If the reference bit is 0 then the page has not been referenced since the last time the reference bit was set to 0.

Count Based Page Replacement

There is many other algorithms that can be used for page replacement, we can keep a counter of the number of references that has made to a page.

a) LFU (least frequently used) :

This causes the page with the smallest count to be replaced. The reason for this selection is that actively used page should have a large reference count.

This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process but never used again. Since it was used heavily, it has a large count and remains in memory even though it is no longer needed.

b) Most Frequently Used(MFU) :

This is based on the principle that the page with the smallest count was probably just brought in and has yet to be used.

Allocation of Frames

- The allocation policy in a virtual memory controls the operating system decision regarding the amount of real memory to be allocated to each active process.
- In a paging system if more real pages are allocated, it reduces the page fault frequency and improved turnaround throughput.
- If too few pages are allocated to a process its page fault frequency and turnaround times may deteriorate to unacceptable levels.
- The minimum number of frames per process is defined by the architecture, and the maximum number of frames is determined by the available physical memory.
- The easiest way is to split m frames among n process by giving each process equal number of frames. This scheme is called equal allocation.
- With multiple processes competing for frames, we can classify page replacement into two broad categories

a) Local Replacement: requires that each process selects frames from only its own sets of allocated frame.

b). Global Replacement: allows a process to select frame from the set of all frames. Even if the frame is currently allocated to some other process, one process can take a frame from another.

In local replacement the number of frames allocated to a process do not change but with global replacement number of frames allocated to a process do not change global replacement results in greater system throughput.

Thrashing

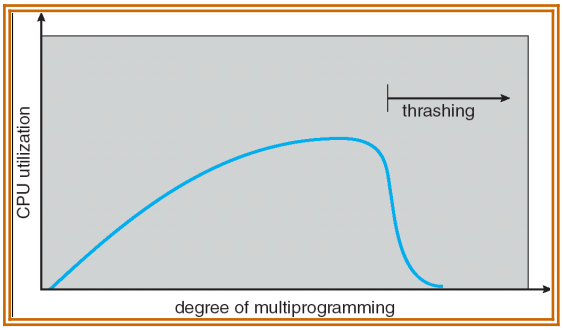
- If the number of frames allocated to a low-priority process falls below the minimum number required by the computer architecture then we suspend the process execution.
- A process is thrashing if it is spending more time in paging than executing.
- If the processes do not have enough number of frames, it will quickly page fault. During this it must replace some page that is not currently in use. Consequently it quickly faults again and again. The process continues to fault, replacing pages for which it then faults and brings back. This high paging activity is called thrashing. The phenomenon of excessively moving pages back and forth b/w memory and secondary has been called thrashing.

Cause of Thrashing

- Thrashing results in severe performance problem.
- The operating system monitors the cpu utilization is low. We increase the degree of multi programming by introducing new process to the system.
- A global page replacement algorithm replaces pages with no regards to the process to which they belong.

The figure shows the thrashing

- As the degree of multi programming increases, more slowly until a maximum is reached. If the degree of multi programming is increased further thrashing sets in and the cpu utilization drops sharply.



At this point, to increases CPU utilization and stop thrashing, we must increase degree of multi programming. We can limit the effect of thrashing by using a local replacement algorithm. To prevent thrashing, we must provide a process as many frames as it needs.

Locality of Reference:

- As the process executes it moves from locality to locality.
- A locality is a set of pages that are actively used.
- A program may consist of several different localities, which may overlap.
- Locality is caused by loops in code that find to reference arrays and other data structures by indices.

The ordered list of page number accessed by a program is called reference string.

Locality is of two types

1) spatial locality
2) temporal locality

Working set model

Working set model algorithm uses the current memory requirements to determine the number of page frames to allocate to the process, an informal definition is

"the collection of pages that a process is working with and which must be resident if the process to avoid thrashing".

The idea is to use the recent needs of a process to predict its future reader.

The working set is an approximation of programs locality.

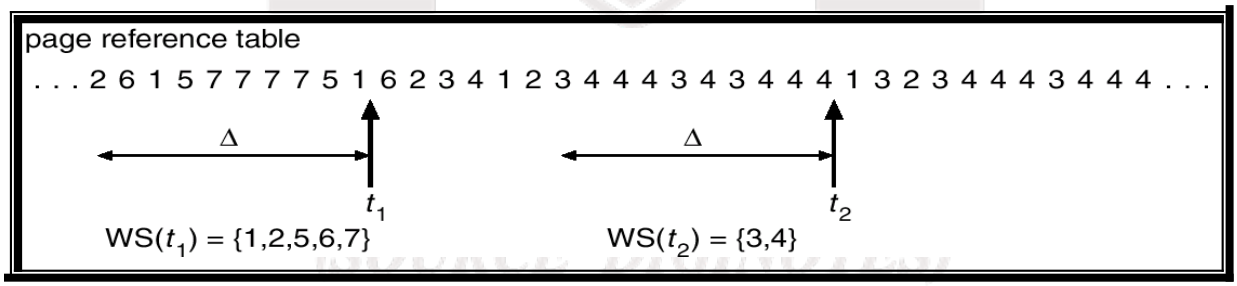
Ex: given a sequence of memory reference, if the working set window size to memory references, then working set at time t1 is {1,2,5,6,7} and at t2 is changed to {3,4}

- At any given time, all pages referenced by a process in its last 4 seconds of execution are considered to compromise its working set.
- A process will never execute until its working set is resident in main memory.
- Pages outside the working set can be discarded at any movement.

Working sets are not enough and we must also introduce balance set.

a) If the sum of the working sets of all the run able process is greater than the size of memory the refuse some process for a while.
b) Divide the run able process into two groups, active and inactive. The collection of active set is called the balance set. When a process is made active its working set is loaded.
c) Some algorithm must be provided for moving process into and out of the balance set.

As a working set is changed, corresponding change is made to the balance set. Working set presents thrashing by keeping the degree of multi programming as high as possible. Thus if optimizes the CPU utilization. The main disadvantage of this is keeping track of the working set.

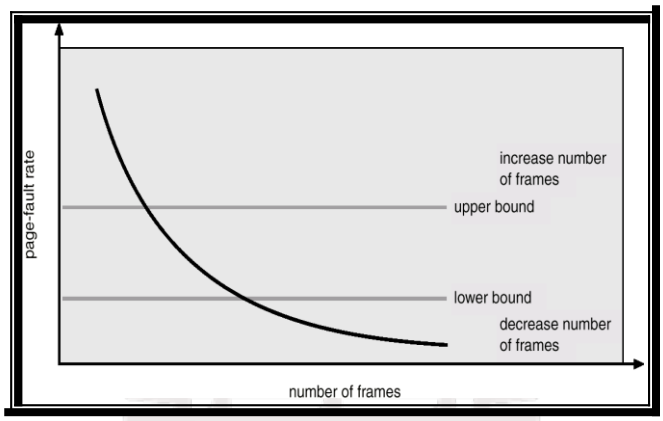


Page fault frequency

- The working set model is successful and knowledge of working set can be useful for pre paging but it is a difficult way to control thrashing. A strategy that uses page fault frequency takes a direct approach.
- Thrashing has a high page fault rate. Thus we control page fault rate, when too high the process is provided with more frames and when page fault rate is too low then it has more

frames. So we can establish a lower bound and upper bound on desired page fault rate. If actual page fault exceeds upper limit, we allocate the process another frame.

- If page fault rate is less than lower bound then we remove a frame.
- Thus we can directly control and measure page fault rate to prevent thrashing.



FILE SYSTEM INTERFACE

- A file is a collection of similar records.
- The data can't be written on to the secondary storage unless they are within a file.
- Files represent both the program and the data. Data can be numeric, alphanumeric, alphabetic or binary.
- Many different types of information can be stored on a file ---Source program, object programs, executable programs, numeric data, payroll recorder, graphic images, sound recordings and so on.
- A file has a certain defined structures according to its type:-
    1. Text file:- Text file is a sequence of characters organized in to lines.
    2. Object file:- Object file is a sequence of bytes organized in to blocks understandable by the systems linker.
    3. Executable file:- Executable file is a series of code section that the loader can bring in to memory and execute.
    4. Source File:- Source file is a sequence of subroutine and function, each of which are further organized as declaration followed by executable statements.

File Attributes

File attributes varies from one OS to other. The common file attributes are:

1. Name:- The symbolic file name is the only information kept in human readable form.
2. Identifier:- The unique tag, usually a number, identifies the file within the file system. It is the non-readable name for a file.
3. Type:- This information is needed for those systems that supports different types.
4. Location:- This information is a pointer to a device and to the location of the file on that device.

5. Size:- The current size of the file and possibly the maximum allowed size are included in this attribute.
6. Protection:-Access control information determines who can do reading, writing, execute and so on.
7. Time, data and User Identification:- This information must be kept for creation, last modification and last use. These data are useful for protection, security and usage monitoring.

File Operation

File is an abstract data type. To define a file we need to consider the operation that can be performed on the file.

Basic operations of files are:-

1. Creating a file:- Two steps are necessary to create a file. First space in the file system for file is found. Second an entry for the new file must be made in the directory. The directory entry records the name of the file and the location in the file system.
2. Writing a file:- System call is mainly used for writing in to the file. System call specify the name of the file and the information i.e., to be written on to the file. Given the name the system search the entire directory for the file. The system must keep a write pointer to the location in the file where the next write to be taken place.
3. Reading a file:- To read a file system call is used. It requires the name of the file and the memory address. Again the directory is searched for the associated directory and system must maintain a read pointer to the location in the file where next read is to take place.
4. Delete a file:- System will search for the directory for which file to be deleted. If entry is found it releases all free space. That free space can be reused by another file.

5. Truncating the file:- User may want to erase the contents of the file but keep its attributes. Rather than forcing the user to delete a file and then recreate it, truncation allows all attributes to remain unchanged except for file length.
6. Repositioning within a file:- The directory is searched for appropriate entry and the current file position is set to a given value. Repositioning within a file does not need to involve actual i/o. The file operation is also known as file seeks.

In addition to this basis 6 operations the other two operations include appending new information to the end of the file and renaming the existing file. These primitives can be combined to perform other two operations.

Most of the file operation involves searching the entire directory for the entry associated with the file. To avoid this OS keeps a small table containing information about an open file (the open table). When a file operation is requested, the file is specified via index in to this table. So searching is not required.

Several piece of information are associated with an open file:-

- File pointer:- on systems that does not include offset an a part of the read and write system calls, the system must track the last read-write location as current file position pointer. This pointer is unique to each process operating on a file.
- File open count:- As the files are closed, the OS must reuse its open file table entries, or it could run out of space in the table. Because multiple processes may open a file, the system must wait for the last file to close before removing the open file table entry. The counter tracks the number of copies of open and closes and reaches zero to last close.
- Disk location of the file:- The information needed to locate the file on the disk is kept in memory to avoid having to read it from the disk for each operation.
- Access rights:- Each process opens a file in an access mode. This information is stored on per-process table the OS can allow OS deny subsequent i/o request.

Access Methods:-

The information in the file can be accessed in several ways.

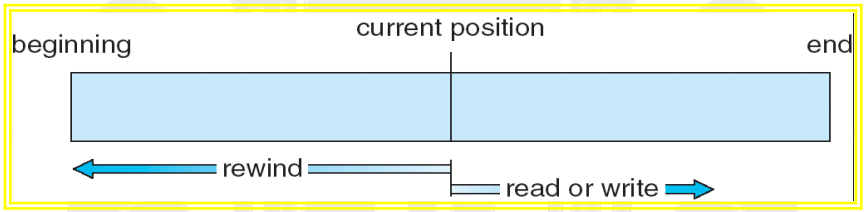
Different file access methods are:-

1. **Sequential Access**:-

Sequential access is the simplest access method. Information in the file is processed in order, one record after another. Editors and compilers access the files in this fashion.

Normally read and write operations are done on the files. A read operation reads the next portion of the file and automatically advances a file pointer, which track next i/I track.

Write operation appends to the end of the file and such a file can be next to the beginning.



Sequential access depends on a tape model of a file.

2. **Direct Access OR relative access**

Direct access allows random access to any file block. This method is based on disk model of a file.A file is made up of fixed length logical records. It allows the program to read and write records rapidly in any order.A direct access file allows arbitrary blocks to be read or written.

Eg:-User may need block 13, then read block 99 then write block 12.

For searching the records in large amount of information with immediate result, the direct access method is suitable. Not all OS support sequential and direct access. Few OS use sequential access and some OS uses direct access. It is easy to simulate sequential access on a direct access but the reverse is extremely inefficient.

Direct access is based on disk model.

Indexing Method:-

- The index is like an index at the end of a book which contains pointers to various blocks.
- To find a record in a file, we search the index and then use the pointer to access the file directly and to find the desired record.

With large files index file itself can be very large to be kept in memory. One solution to create an index to the index files itself. The primary index file would contain pointer to secondary index files which would point to the actual data items.

Two types of indexes can be used:-

a. Exhaustive index:- Contain one entry for each of record in the main file. An index itself is organized as a sequential file.
b. Partial index:- Contains entries to records where the field of interest exists with records of variable length, some record will not contain an fields. When a new record is added to the main file, all index files must be updated.

Directory Structure

The files systems can be very large. Some systems stores millions of files on the disk.

To manage all this data we need to organize them. This organization is done in two parts:-

1. Disks are split in to one or more partition also known as minidisks.
2. Each partition contains information about files within it. This information is kept in entries in a device directory or volume table of contents.

The device directory or simple directory records information as name, location, size, type for all files on the partition.

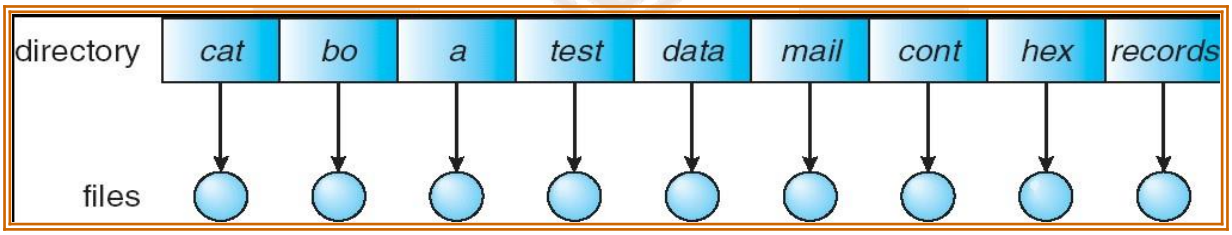
The directory can be viewed as a symbol table that translates the file names in to the directory entries. The directory itself can be organized in many ways.

When considering a particular directory structure, we need to keep in mind the operations that are to be performed on a directory.

- Search for a file:- Directory structure is searched for finding particular file in the directory. Files have symbolic name and similar name may indicate a relationship between files, we may want to be able to find all the files whose name match a particular pattern.
- Create a file:- New files can be created and added to the directory.
- Delete a file:- when a file is no longer needed, we can remove it from the directory.
- List a directory:- We need to be able to list the files in directory and the contents of the directory entry for each file in the list.
- Rename a file:- Name of the file must be changeable when the contents or use of the file is changed. Renaming allows the position within the directory structure to be changed.
- Traverse the file:- it is always good to keep the backup copy of the file so that or it can be used when the system gets fail or when the file system is not in use.

1. **Single-level directory**:-

   This is the simplest directory structure. All the files are contained in the same directory which is easy to support and understand.



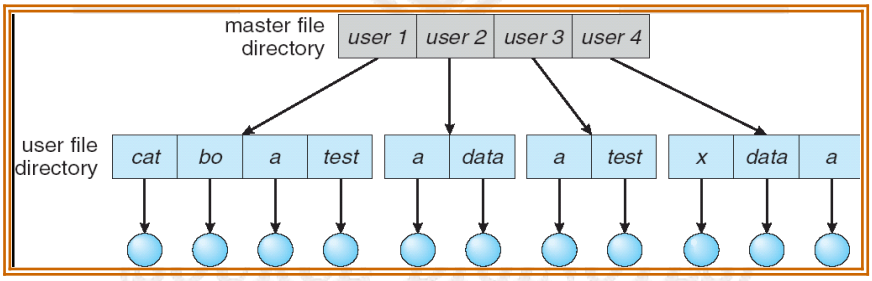
Disadvantage:-

- Not suitable for a large number of files and more than one user.
- Because of single directory files, files require unique file names.
- Difficult to remember names of all the files as the number of files increases.

MS-DOS OS allows only 11 character file name where as UNIX allows 255 character.

2. **Two-level directory**:-

- A single level directory often leads to the confusion of file names between different users. The solution here is to create separate directory or each user.
- In two level directories each user has its own directory. It is called User File Directory (UFD). Each UFD has a similar structure, but lists only the files of a single user.
- When a user job starts or users logs in, the systems Master File Directory (MFD) is searched. The MFD is indexed by the user name or account number and each entry points to the UFD for that user.
- When a user refers to a particular file, only his own UFD is searched. Thus different users may have files with the same name.
- To create a file for a user, OS searches only those users UFD to as certain whether another file of that name exists.
- To delete a file checks in the local UFD so that accidentally delete another user's file with the same name.



Although two-level directories solve the name collision problem but it still has some disadvantage.

This structure isolates one user from another. This isolation is an advantage. When the users are independent but disadvantage, when some users want to co-operate on some table and to access one another file.

3. **Tree-structured directories**:-

MS-DOS use Tree structure directory. It allows users to create their own subdirectory and to organize their files accordingly. A subdirectory contains a set of files or subdirectories. A directory is simply another file, but it is treated in a special way.

The entire directory will have the same internal format. One bit in each entry defines the entry as a file (0) and as a subdirectory (1). Special system calls are used to create and delete directories.

In normal use each uses has a current directory. Current directory should contain most of the files that are of the current interest of users. When a reference to a file is needed the current directory is searched. If file is needed i.e., not in the current directory to be the directory currently holding that file. The user must search by specifying path name.

Path name can be of two types:-

a. Absolute path name:- Begins at the root and follows a path down to the specified file, giving the directory names on the path.
b. Relative path name:- Defines a path from the current directory.

One important policy in this structure is low to handle the deletion of a directory.

a. If a directory is empty, its entry can simply be deleted.
b. If a directory is not empty, one of the two approaches can be used.
    i. In MS-DOS, the directory is not deleted until it becomes empty.
    ii. In UNIX, RM command is used with some options for deleting directory.

4. **Acyclic graph directories**:-

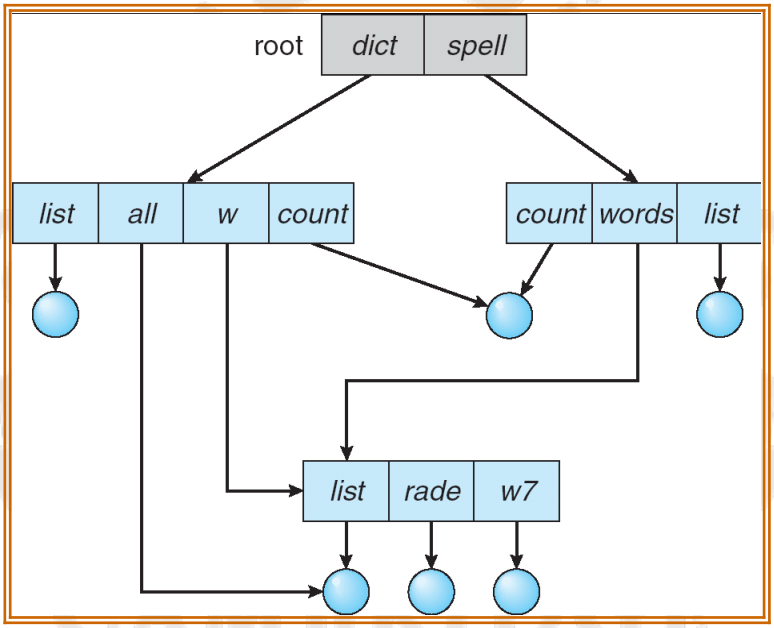
- It allows directories to have shared subdirectories and files.
- Same file or directory may be in two different directories.
- A graph with no cycles is a generalization of the tree structure subdirectories scheme.
- Shared files and subdirectories can be implemented by using links.

- Alink is a pointer to another file or a subdirectory.
- A link is implemented as absolute or relative path.
- An acyclic graph directory structure is more flexible then is a simple tree structure but sometime it is more complex. Several problems must be considered carefully.

Disadvantages

1. A file can have multiple absoluter path names. So distinct file path refer to same file.

2. Another problem is deletion. If anyone deletes a file then it leads to a problem. So instead of deleting a file only the link should be deleted. Another method to deletion is to preserve the file until all references to it are deleted.



File System Mounting:-

The file system must be mounted before it can be available to processes on the system

The procedure for mounting the file is:

a. The OS is given the name of the device and the location within the file structure at which to attach the file system (mount point).A mount point will be an empty directory at which the mounted file system will be attached.

Eg:- On UNIX a file system containing users home directory might be mounted as /home then to access the directory structure within that file system. We must precede the directory names as /home/jane.

b. Then OS verifies that the device contains this valid file system. OS uses device drivers for this verification.
c. Finally the OS mounts the file system at the specified mount point.

To show file system mounting consider figure a showing existing file system and figure b shows unmounted volume. At this point only existing file system can be accessed.

The figure c shows the effect of mounting. On unmounting the file system is restored as shown in figure a.

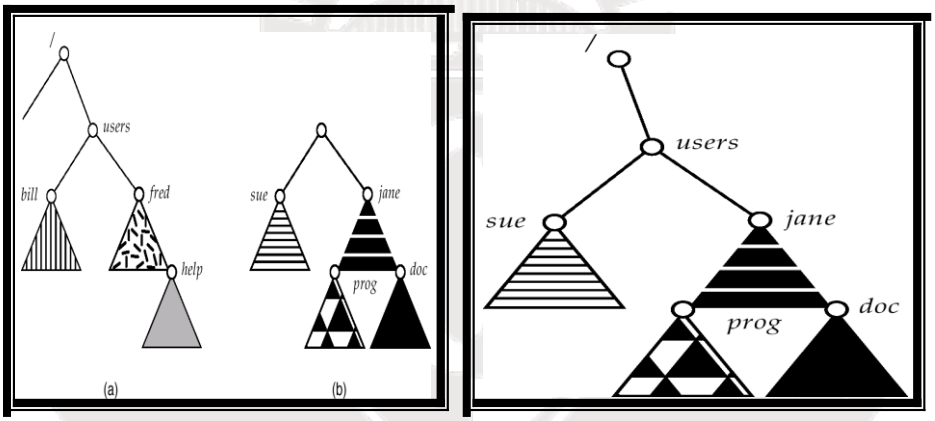


Figure c

File System Structure:-

Disks provide bulk of secondary storage on which the file system is maintained. Disks have two characteristics:-

a. They can be rewritten in place i.e., it is possible to read a block from the disk to modify the block and to write back in to same place.
b. They can access any given block of information on the disk. Thus it is simple to access any file either sequentially or randomly and switching from one file to another.

To provide efficient and convenient access to the disks, the OS provides the file system to allow the data to be stored, located and retrieved.

A file system has two design problems:-

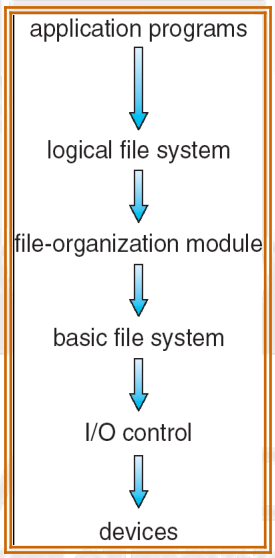
a. How the file system should look to the user.

b. Selecting algorithms and data structures that must be created to map logical file system on to the physical secondary storage devices.

The first task is mainly concerned with defining a file, its attributes, operations on file, the directory structure.

The file system itself is composed of different levels. Each level uses the feature of the lower levels to create new features for use by higher levels.

The following structures shows an example of layered design



The lowest level is the i/o control consisting of device drivers and interrupt handless to transfer the information between memory and the disk system. The device driver is like a translator. Its input is a high level command and the o/p consists of low level hardware specific instructions, which are used by the hardware controllers which interface I/O device to the rest of the system.

The basic file system needs only to issue generic commands to the appropriate device drivers to read and write physical blocks on the disk.

The file organization module knows about files and their logical blocks as well as physical blocks. By knowing the type of file allocation used and the location of the file, the file organization module can translate logical block address to the physical block address. Each logical block is numbered 0 to N. Since the physical blocks containing the data usually do not match the logical numbers, So a translation is needed to locate each block. The file allocation modules also include free space manager which tracks the unallocated blocks and provides these blocks when requested.

The logical file system uses the directory structure to provide the file organization module with the information, given a symbolic file name. The logical file system also responsible for protection and security.

Logical file system manages metadata information. Metadata includes all the file system structures excluding the actual data.

The file structure is maintained via file control block (FCB). FCB contains information about the file including the ownership permission and location of the file contents.

File System Implementation:-

- File system is implemented on the disk and the memory.
- The implementation of the file system varies according to the OS and the file system, but there are some general principles.

If the file system is implemented on the disk it contains the following information:-

a. **Boot Control Block**:- can contain information needed by the system to boot an OS from that partition. If the disk has no OS, this block is empty. It is the first block of the partition. In UFS◇boot block, In NTFS◇partition boot sector.
b. **Partition control Block**:- contains partition details such as the number of blocks in partition, size of the blocks, number of free blocks, free block pointer, free FCB count and FCB pointers. In NTFS◇master file tables, In UFS◇super block.

c. Directory structure is used to organize the files.
d. An FCB contains many of the files details, including file permissions, ownership, size, location of the data blocks. In UFS◇inode, In NTFS this information is actually stored within master file table.

Structure of the file system management in memory is as follows:-

a. An in-memory partition table containing information about each mounted information.
b. An in-memory directory structure that holds the directory information of recently accessed directories.
c. The system wide open file table contains a copy of the FCB of each open file as well as other information.
d. The per-process open file table contains a pointer to the appropriate entry in the system wide open file table as well as other information.

A typical file control blocks is shown below

File permission
File dates (create, access, write)
File owner, group, Acc
File size
File data blocks

Partition and Mounting:-

- A disk can be divided in to multiple partitions. Each partition can be either raw i.e., containing no file system and cooked i.e., containing a file system.
- Raw disk is used where no file system is appropriate. UNIX swap space can use a raw partition and do not use file system.
- Some db uses raw disk and format the data to suit their needs. Raw disks can hold information need by disk RAID (Redundant Array of Independent Disks) system.

- Boot information can be stored in a separate partition. Boot information will have their own format. At the booting time system does not load any device driver for the file system. Boot information is a sequential series of blocks, loaded as an image in to memory.
- Dual booting is also possible on some Pc's, more than one OS are loaded on a system.

  A boot loader understands multiple file system and multiple OS can occupy the boot space once loaded it can boot one of the OS available on the disk. The disks can have multiple portions each containing different types of file system and different types of OS.

  Root partition contains the OS kernel and is mounted at a boot time. Microsoft window based systems mount each partition in a separate name space denoted by a letter and a colon. On UNIS file system can be mounted at any directory.

Directory Implementation:-

Directory is implemented in two ways:-

1. **Linear list**:-
   - Linear list is a simplest method.
   - It uses a linear list of file names with pointers to the data blocks.
   - Linear list uses a linear search to find a particular entry.
   - Simple for programming but time consuming to execute.
   - For creating a new file, it searches the directory for the name whether same name already exists.
   - Linear search is the main disadvantage.
   - Directory implementation is used frequently and uses would notice a slow implementation of access to it.
2. **Hash table**:-
   - Hash table decreases the directory search time.

- Insertion and deletion are fairly straight forward.
- Hash table takes the value computed from that file name.
- Then it returns a pointer to the file name in the linear list.
- Hash table uses fixed size.

Allocation Methods:-

The space allocation strategy is closely related to the efficiency of the file accessing and of logical to physical mapping of disk addresses.

A good space allocation strategy must take in to consideration several factors such as:-

1. Processing speed of sequential access to files, random access to files and allocation and de-allocation of blocks.

2. Disk space utilization.

3. Ability to make multi-user and multi-track transfers.

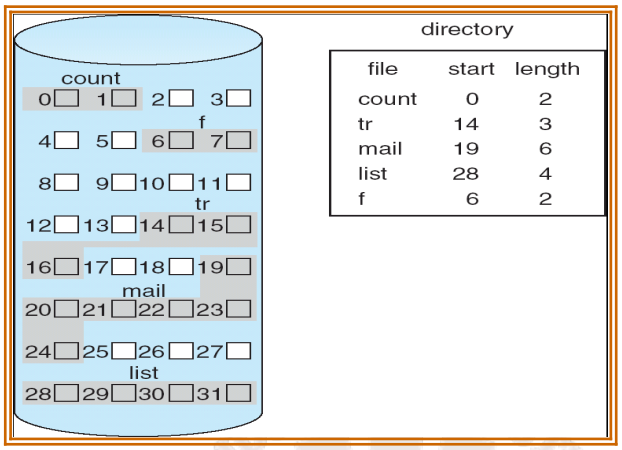
4. Main memory requirement of a given algorithm.

Three major methods of allocating disk space is used.

1. **Contiguous Allocation**:-

   A single set of blocks is allocated to a file at the time of file creation. This is a pre-allocation strategy that uses portion of variable size. The file allocation table needs just a single entry for each file, showing the starting block and the length of the file.

   The figure shows the contiguous allocation method.



If the file is n blocks long and starts at location b, then it occupies blocks b, b+1, b+2…….b+n-1. The file allocation table entry for each file indicates the address of starting block and the length of the area allocated for this file.

Contiguous allocation is the best from the point of view of individual sequential file. It is easy to retrieve a single block. Multiple blocks can be brought in one at a time to improve I/O performance for sequential processing. Sequential and direct access can be supported by contiguous allocation.

Contiguous allocation algorithm suffers from external fragmentation. Depending on the amount of disk storage the external fragmentation can be a major or minor problem. Compaction is used to solve the problem of external fragmentation.

The following figure shows the contiguous allocation of space after compaction. The original disk was then freed completely creating one large contiguous space.

If the file is n blocks long and starts at location b, then it occupies blocks b, b+1, b+2…….b+n-1. The file allocation table entry for each file indicates the address of starting block and the length of the area allocated for this file. Contiguous allocation is the best from the point of view of individual sequential file. It is easy to retrieve a single block. Multiple blocks can be brought in one at a time to improve I/O performance for sequential processing. Sequential and direct access can be supported by contiguous allocation. Contiguous allocation algorithm suffers from external fragmentation. Depending on the amount of disk storage the

external fragmentation can be a major or minor problem. Compaction is used to solve the problem of external fragmentation.

The following figure shows the contiguous allocation of space after compaction. The original disk was then freed completely creating one large contiguous space.

Another problem with contiguous allocation algorithm is pre-allocation, i.e., it is necessary to declare the size of the file at the time of creation.

Characteristics:-

- Supports variable size portion.
- Pre-allocation is required.
- Requires only single entry for a file.
- Allocation frequency is only once.

Advantages:-

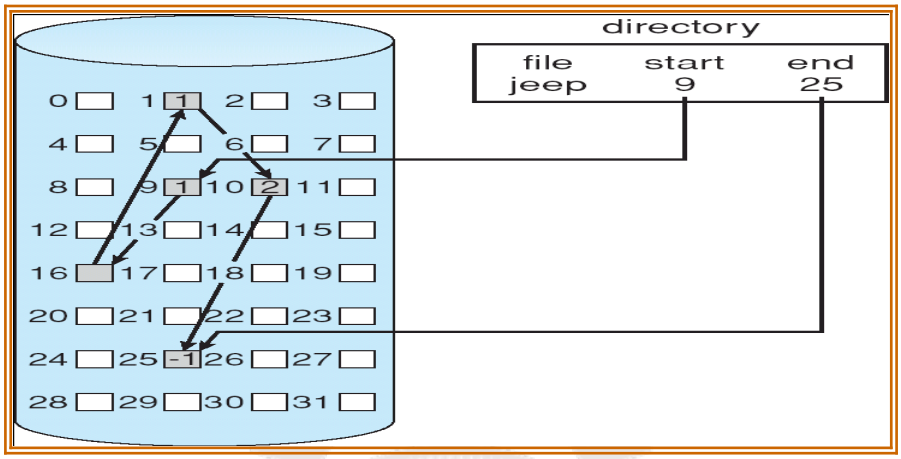
- Supports variable size problem.
- Easy to retrieve single block.
- Accessing a file is easy.
- It provides good performance.

Disadvantage:-

- Pre-allocation is required.
- It suffers from external fragmentation.

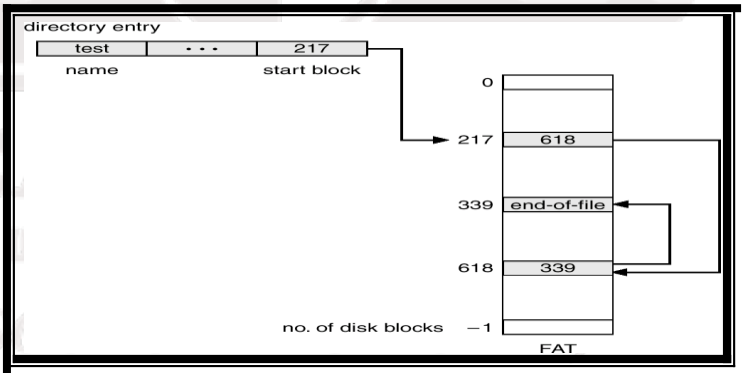
2. **Linked Allocation**:-
   - It solves the problem of contiguous allocation. This allocation is on the basis of an individual block. Each block contains a pointer to the next block in the chain.
   - The disk block can be scattered anywhere on the disk.
   - The directory contains a pointer to the first and the last blocks of the file.

- The following figure shows the linked allocation. To create a new file, simply create a new entry in the directory.



- There is no external fragmentation since only one block is needed at a time.
- The size of a file need not be declared when it is created. A file can continue to grow as long as free blocks are available.
- An important variation of linked allocation is the use of file allocation table(FAT).

Example:-



Advantages:-

- No external fragmentation.
- Compaction is never required.
- Pre-allocation is not required.

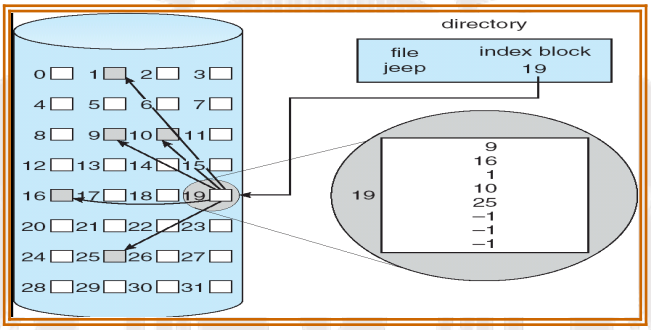
Disadvantage:-

- Files are accessed sequentially.
- Space required for pointers.
- Reliability is not good.
- Cannot support direct access.

3. **Indexed Allocation**:-

The file allocation table contains a separate one level index for each file. The index has one entry for each portion allocated to the file. The i th entry in the index block points to the i th block of the file.

The following figure shows indexed allocation.



The indexes are not stored as a part of file allocation table rather than the index is kept as a separate block and the entry in the file allocation table points to that block.

Allocation can be made on either fixed size blocks or variable size blocks. When the file is created all pointers in the index block are set to nil. When an entry is made a block is obtained from free space manager.

Allocation by fixed size blocks eliminates external fragmentation where as allocation by variable size blocks improves locality.

Indexed allocation supports both direct access and sequential access to the file.

Advantages:-

- Supports both sequential and direct access.
- No external fragmentation.

- Faster then other two methods.
- Supports fixed size and variable sized blocks.

Disadvantage:-

- Suffers from wasted space.
- Pointer overhead is generally greater.

Every block has a indexed file and this must be as small as possible. If index block is small then it cannot hold enough pointers for large file then we need a mechanism to deal with this. The only one of the following mechanism can be used for this purpose.

1. Linked Scheme: An index block is only one disk block. To allow for large files we link several index blocks.

2. Multilevel Index: A variant of linked representation uses a first level index block to point to a set of second level index which in turn point to file blocks.

The first block index is used to find second level index and this can be continued further depending on maximum size.

3. Combined Scheme: Another alternative used in the UFS is to keep the first say 15 pointers of index block in files inode. The first 12 of these pointers points to direct blocks i.e. this contain addresses of blocks that contain data of the file.

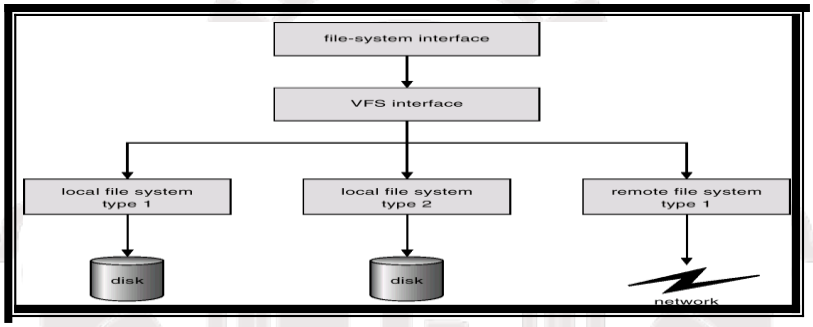
The next three pointers points to indirect blocks.

- The first points to single indirect block which is an index block containing the addresses of block that contain data.
- The 2nd points to double indirect block which contains the address of block that contains pointers to actual data blocks.
- The last pointer points to triple indirect block.

VIRTUAL FILE SYSTEM

Modern OS should concurrently support multiple file types of file system concurrently.

- The optimal method is to write directory and file routine to each type.
- Instead most of the OS including UNIX supports object oriented technique to simplify organize and modularize the implementation.
- The use of these methods allows dissimilar file system types to be implemented within the same structure including network file system (NFS).
- Users can access multiple files on same disk or the other system through network.
- The following figure shows schematic view of virtual file system



- The first layer is file system interface based on read(), write(), open(), close() calls on file description.
- The second layer is VFS interface, it serves 2 important functions

a. It separate generic file operations from their implementation.

b. The VFS provides a mechanism to represent the file throughout the network uniquely. The VFS is based on file representation structure called Vnode that contain a unique code i.e. a number for network wide unique file. The kernel maintains one Vnode for each active node.

The UFS distinguishes local files from remote ones.

- The UFS activates file system specific operations to handle local requests according to their file types and even calls NFS protocol procedure for remote request.

3. The third layer implements the file system type or remote file system protocol.

Linux VFS defines 4 main objects

a. inode object- which represents an individual file.

b. File object- which represents an open file.

c. Superblock object- represents entire file system.

d. Entry object- represents an individual directory entry.

CONSISTENCY SEMANTICS

- Consistency semantics represents an important criterion for evaluating any file system that supports file sharing.
- These semantics specify how multiple users of a system are to access shared files simultaneously. They also specify how the modification to the file by one is visible to the other users.
- These semantics are implemented a code with the file system.
- Consistency semantics are closely related to the process synchronization algorithms.
- We assume that series of file accesses attempted by users to some file are always enclosed between open() and close() operations. The series of accesses between open() and close() make a file session.
- The following are some of examples of consistency semantics.

a. UNIX SEMANTICS

- Unix File System (UFS) uses following consistency semantics

i. Write to an open file is visible to other users that have this file open.

ii. one node of file sharing allows users to share the pointers of current location into the file. Advancing the pointer by one user affects all sharing users.

b. SESSION SEMANTICS

- Andrew file system (AFS) supports following consistency semantics

i. writes to an open file is not immediately available to other users that have the same file open.

ii. once the file is closed, the changes made to it are visible only in next session.

c. IMMUTABLE _SHARED FILE SEMANTICS

- In this once a file is declared as shared by its creator, it can't be changed.
- An immutable file has two key properties:

i. its name may not be reused.

ii. its contents may not be altered.

- The name of the immutable file indicates that the contents of file are fixed.

PROTECTION

- When information is stored on a computer system, it should be safe from physical damage (Reliability) and improper access (Protection).
- Reliability is generally provided by duplicate copies of files.
- File system may be damaged by hardware problems, power failures, head crashes, dirt, and temperature extremes.
- Files may be deleted accidently.
- Bugs in the file system software can also cause contents of file to be lost.

a. Types of Access

- System that do not permit access to files of other users do not need protection.
- We can provide complete protection by prohibiting access and we can provide controlled access to the files.
- Protection mechanism gives controlled access by limiting the types of access.
- Access can be permitted or denied depending on several factors. Several different types of operations may be controlled.
    - Read- read from file
    - Write- write or rewrite the file
    - Execute- load file into memory and execute.
    - Append- write new information at the end of the file.
    - Delete- deletes the file and frees its space.
    - List- lists the names and attributes of the file.

  Other operations like renaming, copying and editing the file may also be controlled.

b. Access Control

- The most common approach to protection is to make access dependent on identity of user.
- Different users require different access to file and directory.
- Each file is associated with access-control list (ACL), which specifies the user name and types of access allowed for each.
- When a user requests access to a particular file, the OS checks the access list associated with that file. If user is listed for requested access, he is allowed else not.
- The main problem with this is, if we allow everyone to read a file, we must list all users along with access. This technique has two undesirable consequences.

  i. constructing such a list is tedious task.

  ii. if the directory size to be varied requires more complicated space management.

These problems can be solved by using condensed version of access list. To do this file is associated with three classifications of users.

1. owner- the user who created the file is the owner.
2. Group- a set of users who are sharing the file and need similar access.
3. Universe- all other users in the system constitute the universe.

FREE SPACE MANAGEMENT

- Since the disk space is limited, we need to reuse the space from deleted files for new files.
- To keep track of free disk space, the system maintains a free space list.
- The free space list records all the free disk space i.e. not allocated to any directory.
- To create a file, we search free space list for the required amount of space and allocate new file to that space.
- When the file is deleted the disk space is taken back.
- The free space management can be implemented as follows

1. BIT VECTOR:-

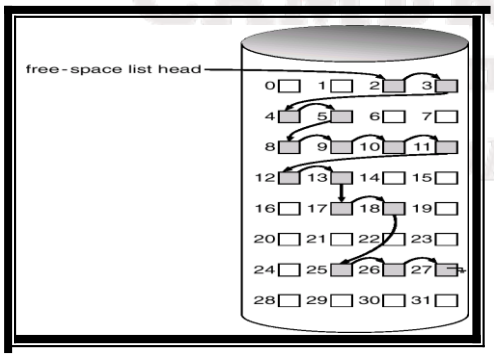
- Frequently the free space list is implemented as bit map or bit vector.
- If the bit is 1, then block is free and if bit is 0, then block is allocated.

Example: consider a disk where blocks 2, 3, 4. 5, 8, 9, 10, 11, 12, 13 are free and rest of the others are allocated. The free space bit map will be 001111001111110000-------- .

- The main advantage of this method is its relative simplicity and its efficiency in finding the first free n blocks on the disk.
- Many computers supply bit-manipulation instructions that can be used effectively for that purpose.
- One method for finding the first free block on a system that implement bit vector is to sequentially search each word in bit map to see whether block is free or not.
- Bit vector are inefficient unless the entire vector is kept in main memory.

2. LINKED LIST:-

- In this approach all the free disk blocks are linked together, by keeping a pointer to the first block in a special location on disk and caching it in memory.
- But this scheme is not efficient, to traverse the list we must read each block which consumes substantially input/output time.
- The following figurer shows block 2 contain pointer to block 3 which contain pointer to block 4 which contain pointer to block 5.



3. GROUPING:-

- A modification of free space list approach is to store the addresses of n free blocks in the first free block. The first n-1 blocks are free and the n^{th} block contain the addresses of next n free blocks and so on.
- The addresses of large number of blocks can be found quickly.

4. COUNTING:-

- In this rather than keeping the addresses of n free disk, we keep the address of first free block and number n of free contiguous blocks that follow the first block. Each entry in free space list consists of disk addresses and a count. Each entry requires more space.

FILE SHARING

1. Multiple Users

- If an OS supports multiple users, the issues of file naming, sharing and protection becomes important.
- If a directory contains file and allows file to be shared by various users, the system must mediate file sharing.
- The system can also a user to access the file of other users by default or requires a user to specifically grant access to the file. These are issues related to protection and access control.
- To implement sharing and protection, the system must maintain more files and directory attribute. Most systems evolved to use the concept of file owner and group.
- The owner is the one who grant access to other users or change attributes and has control over file.
- The group attribute defines a subset of users who can share access to the file.

2. REMOTE FILE SYSTEM

- Network allows remote computers to communicate. Network allows the sharing or resources across a campus or around the world.
- Through the evolution of network and file technology, remote file sharing methods have changed.

1. I^{st} implemented method involves manually transferring files between machine through FTP.

2. The 2nd method uses distributed file system (DFS) in which remote directories are visible on local machine.

3. The 3rd method is WWW. A browser is needed to gain access to remote access to remote files.

a. Client-Server Model

- In this the machine containing the file is a server and the machine access these files is the client.
- The client-server relation is common in the network machines.
- The servers indicate the resources available to the clients and servers can serve multiple clients and clients can use multiple servers depending on implementation detail of client-server facility.
- The client identification is difficult. A client can be specified by a network name or IP addresses or an identifier. But these can be duplicated. So more secure solutions include secure authentication of client via encrypted keys.
- Once the remote file system is mounted, file operations requests are sent to users across the network to server via DFS protocol.

b. Distributed Information System

- To make client-server easier to manage distributed information system also called distributed naming services provide unified access to the information needed for remote computing.
- The domain name system (DNS) provides host-name to network address translation for entire internet.
- Other distributed information system provides user name/password/user id/group id space for a distributed facility.
- UNIX system uses variety of distributed information methods.

3. Failure Modes

- Local file system can fail for variety of reasons, including failure of disk containing the file system, corruption of directory structures or other disk management information (metadata), disk controller failures, cable failure and host-adopter failure.

- Uses or system administrator failure can also cause files to be lost or entire directory to deleted. Most of these failures will cause a host to crash and an error condition to be displayed and human intervention may be required to repair the damage.
- Remote file systems have more failure modes. The complexity of the network may interface with operation with proper operation of remote file system.
- In case of network, the network can be interrupted between two hosts. Such interruption is because of hardware failures, poor hardware configuration or network implementation issues.
- To recover from failures, some kind of state information may be maintained on both client and the server.
- If both server and client maintain knowledge of current activities and open files, then they can recover from failures.