SCHOOL OF COMPUTING

Common to : CSE , IT

UNIT IV

UNIT 4 MEMORY MANAGEMENT **9 Hrs.**

Storage Management Strategies - Contiguous Vs. Non-Contiguous Storage Allocation - Fixed & Variable Partition Multiprogramming - Paging - Segmentation - Paging/Segmentation Systems - Page Replacement Strategies - Demand & Anticipatory Paging - File Concept - Access Methods - Directory Structure - File Sharing - Protection - File - System Structure - Implementation.

STORAGE / MEMORY MANAGEMENT:

- Memory management is the functionality of an operating system which handles or manages primary memory.
- Memory management keeps track of each and every memory location either it is allocated to some process or it is free.
- It checks how much memory is to be allocated to processes. It decides which process will get memory at what time.
- It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.
- Memory management provides protection by using two registers, a base register and a limit register.
- The base register holds the smallest legal physical memory address and the limit register specifies the size of the range.
- For example, if the base register holds 300000 and the limit register is 1209000, then the program can legally access all addresses from 300000 through 411999

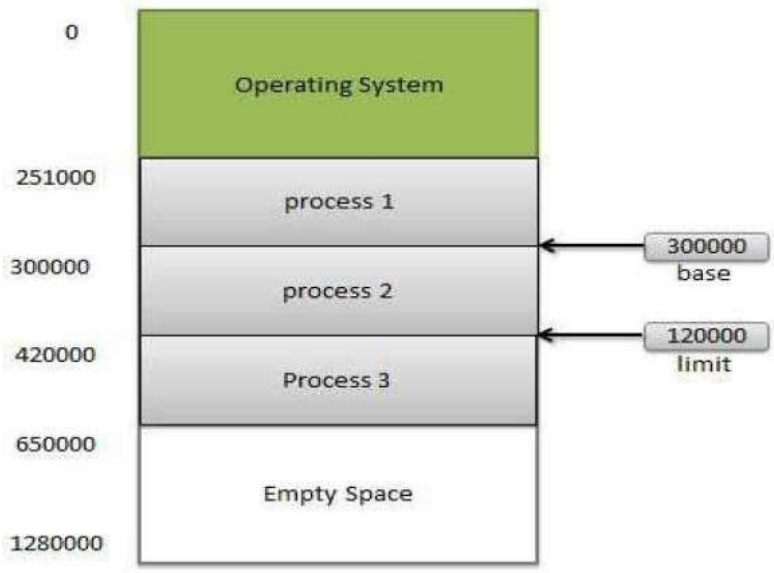


Fig 1 : Memory Management

GOALS OF MEMORY MANAGEMENT:

- Allocate available memory efficiently to multiple processes
- Main functions
- Allocate memory to processes when needed
- Keep track of what memory is used and what is free
- Protect one process's memory from another

MEMORY ALLOCATION STRATEGIES:

- **CONTIGUOUS ALLOCATION :**
  - In contiguous memory allocation each process is contained in a single contiguous block of memory.
  - Memory is divided into several fixed size partitions.
  - Each partition contains exactly one process.
  - When a partition is free, a process is selected from the input queue and loaded into it.
  - The free blocks of memory are known as holes.
  - The set of holes is searched to determine which hole is best to allocate.

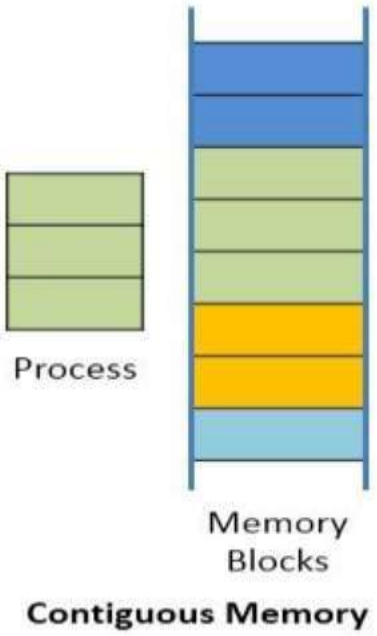


Fig 2 : Contiguous memory

- **NON-CONTIGUOUS ALLOCATION:**
  - Parts of a process can be allocated noncontiguous chunks of memory.

- In context to memory organization, non contiguous memory allocation means the available memory space is scattered here and there it means all the free available memory space is not together at one place.

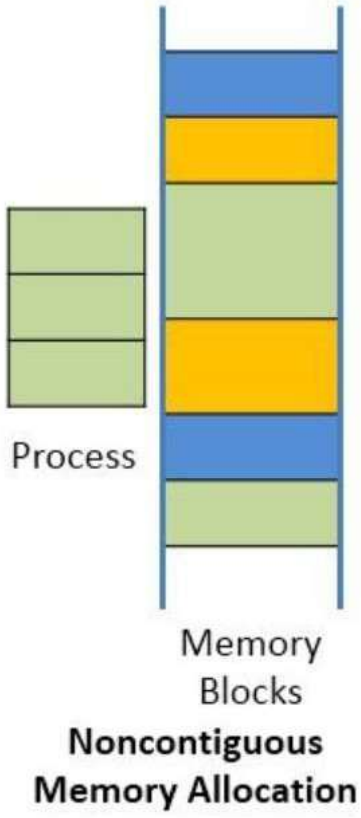
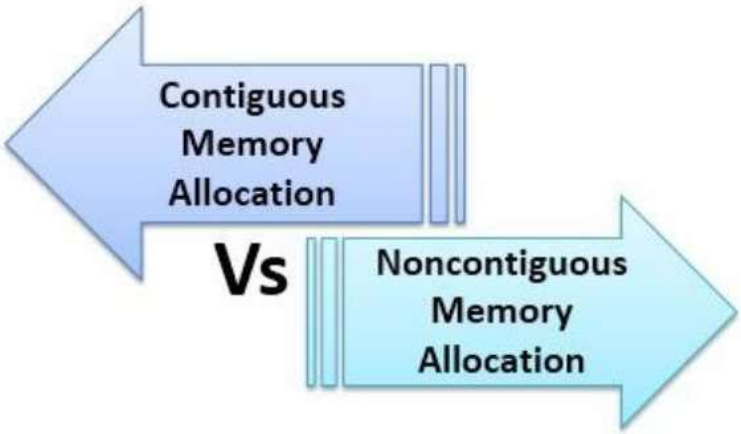


Fig 3 : Non-contiguous memory

CONTAGIOUS Vs NON CONTAGIOUS



CONTIGUOUS MEMORY ALLOCATION	NONCONTIGUOUS MEMORY ALLOCATION
Allocates consecutive blocks of memory to a process.	Allocates separate blocks of memory to a process.
Contiguous memory allocation does not have the overhead of address translation while execution of a process.	Noncontiguous memory allocation has overhead of address translation while execution of a process.
A process executes fatser in contiguous memory allocation	A process executes quite slower comparatively in noncontiguous memory allocation.
The memory space must be divided into the fixed-sized partition and each partition is allocated to a single process only.	Divide the process into several blocks and place them in different parts of the memory according to the availability of memory space available.

Table 1 : Contiguous vs Non-Contiguous

FIXED PARTITION SCHEME:

- Memory is broken up into fixed size partitions
- But the size of two partitions may be different
- Each partition can have exactly one process
- When a process arrives, allocate it a free partition
- Easy to manage
- Problems - Maximum size of process bound by max. partition size, Large internal fragmentation possible

VARIABLE PARTITION SCHEME

- Hole – block of available memory; holes of various size are scattered throughout memory
- When a process arrives, it is allocated memory from a hole large enough to accommodate it
- Operating system maintains information about:
  - allocated partitions
  - free partitions (hole)

MULTIPROGRAMMING:

- To overcome the problem of underutilization of CPU and main memory, the multiprogramming was introduced.
- The multiprogramming is interleaved execution of multiple jobs by the same computer.
- In multiprogramming system, when one program is waiting for I/O transfer, there is another program ready to utilize the CPU.
- SO it is possible for several jobs to share the time of the CPU.
- But it is important to note that multiprogramming is not defined to be the execution of jobs at the same instance of time.
- Rather it does mean that there are a number of jobs available to the CPU (placed in main memory) and a portion of one is executed then a segment of another and so on.

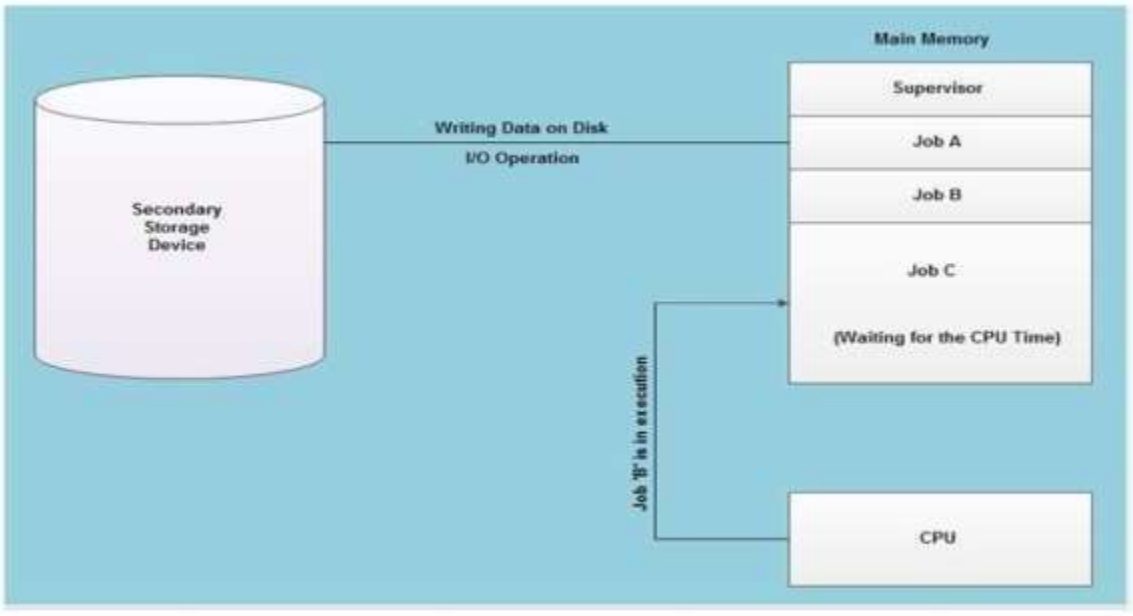


Fig 4 : Multiprogramming

- At the particular situation, job' A' is not utilizing the CPU time because it is busy in I/ 0 operations.
- Hence the CPU becomes busy to execute the job 'B'. Another job C is waiting for the CPU for getting its execution time.
- So in this state the CPU will never be idle and utilizes maximum of its time.
- A program in execution is called a "Process", "Job" or a "Task".
- The concurrent execution of programs improves the utilization of system resources and enhances the system throughput as compared to batch and serial processing.
- In this system, when a process requests some I/O to allocate; meanwhile the CPU time is assigned to another ready process.
- So, here when a process is switched to an I/O operation, the CPU is not set idle.
- Multiprogramming is a common approach to resource management.
- The essential components of a single-user operating system include a command processor, an input/ output control system, a file system, and a transient area.
- A multiprogramming operating system builds on this base, subdividing the transient area to hold several independent programs and adding resource management routines to the operating system's basic functions.

STORAGE HIERARCHY

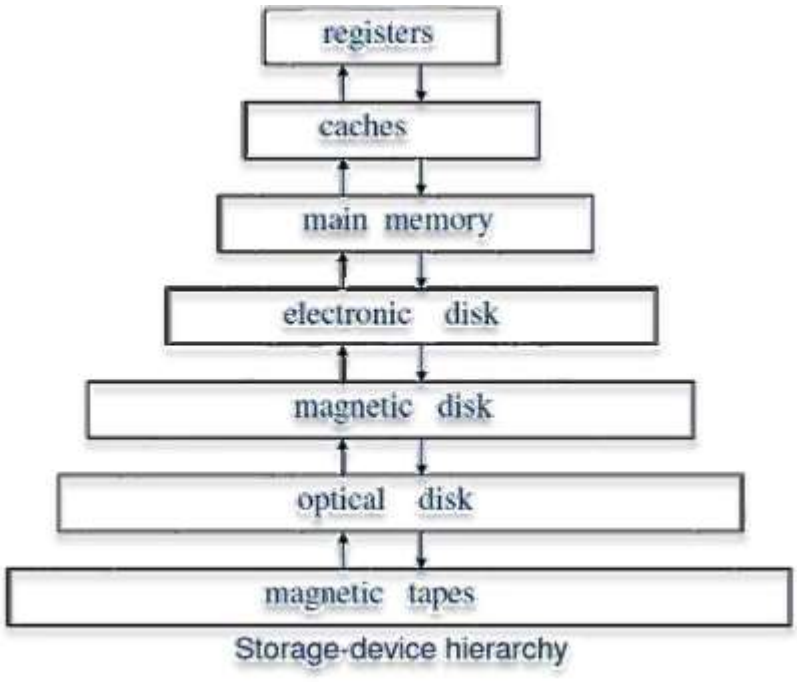


Fig 5 : Storage Hierarchy

- The wide variety of storage systems in a computer system can be organized in a hierarchy according to their speed and their cost.
- The higher levels are expensive but fast. As we move down the hierarchy, the cost per bit decreases, whereas the access time increases.
- The reason for using the slower memory decives is that they are cheaper than the faster ones.
- Many early storage devices, including paper tape and core memories, are found only in museums now that magnetic tape and semiconductor memory have become faster and cheaper.

DYNAMIC LOADING

- In dynamic loading, a routine of a program is not loaded until it is called by the program.
- All routines are kept on disk in a re-locatable load format.
- The main program is loaded into memory and is executed.
- Other routines methods or modules are loaded on request.
- Dynamic loading makes better memory space utilization and unused routines are never loaded.

DYNAMIC LINKING

- Linking is the process of collecting and combining various modules of code and data into a executable file that can be loaded into memory and executed.
- Operating system can link system level libraries to a program.
- When it combines the libraries at load time, the linking is called static linking and when this linking is done at the time of execution, it is called as dynamic linking.

LOGICAL vs PHYSICAL ADDRESS SPACE:

- An address generated by the CPU is a logical address whereas address actually available on memory unit is a physical address.
- Logical address is also known a Virtual address.
- Virtual and physical addresses are the same in compile-time and load-time address-binding schemes.
- Virtual and physical addresses differ in execution-time address-binding scheme.

LOGICAL ADDRESS	PHYSICAL ADDRESS
It is the virtual address generated by CPU	The physical address is a location in a memory unit.
Set of all logical addresses generated by CPU in reference to a program is referred as Logical Address Space.	Set of all physical addresses mapped to the corresponding logical addresses is referred as Physical Address.
The user can view the logical address of a program.	The user can never view physical address of program
The user uses the logical address to access the physical address.	The user can not directly access physical address.
The Logical Address is generated by the CPU	Physical Address is Computed by MMU

Table 2 : Logical vs Physical Address

SWAPPING

- Swapping is a mechanism in which a process can be swapped temporarily out of main memory to a backing store, and then brought back into memory for continued execution.
- Backing store is a usually a hard disk drive or any other secondary storage which fast in access and large enough to accommodate copies of all memory images for all users.
- It must be capable of providing direct access to these memory images.
- Major time consuming part of swapping is transfer time.
- Total transfer time is directly proportional to the amount of memory swapped.
- Let us assume that the user process is of size 100KB and the backing store is a standard hard disk with transfer rate of 1 MB per second.
- The actual transfer of the 100K process to or from memory will take

- 100KB / 1000KB per second = 1/10 second = 100 milliseconds

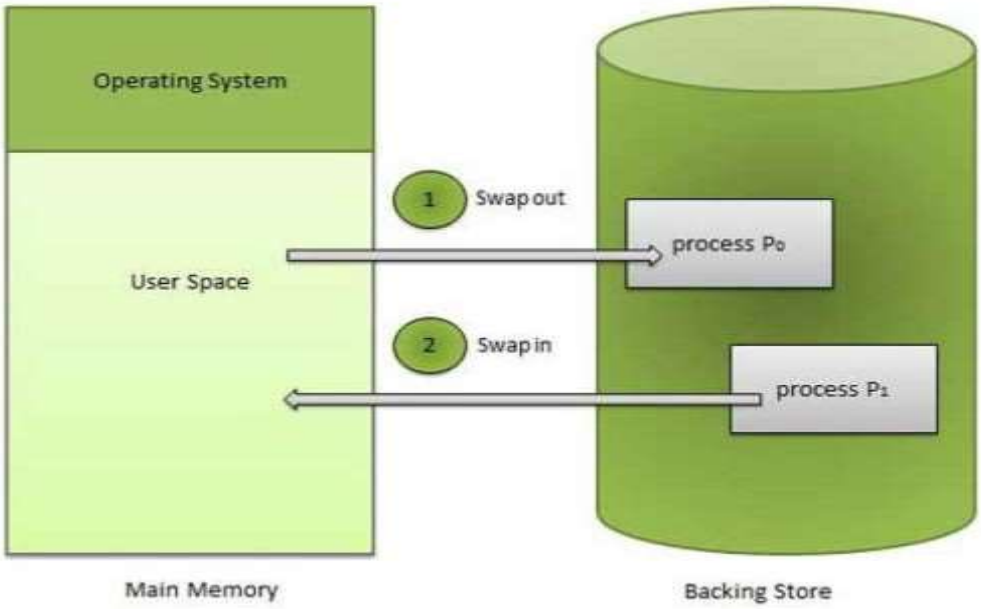


Fig 6 : Swapping

MULTIPLE-PARTITION ALLOCATION

FIXED PARTITION ALLOCATION

- One memory allocation method is to divide the memory into a number of fixed-size partitions.
- Each partition may contain exactly one process. Thus the degree of multiprogramming is bound by the number of partitions.
- When a partition is free, a process is selected from the input queue and is loaded into the free partition.
- When the process terminates, the partition becomes available for another process.

DYNAMIC ALLOCATION

- The operating system keeps a table indicating which parts of memory is available (called holes) are available and which are occupied.
- Initially, all memory is available for user processes (i.e., there is one big hole).
- When a process arrives, we select a hole which is large enough to hold this process.
- We allocate as much memory is required for the process and the rest is kept as a hole which can be used for later requests. Selection of a hole to hold a process can follow the following algorithms

- **FIRST-FIT:**
  - Allocate the first hole that is big enough. Searching can start at the beginning of the set of holes or where the previous first-fit search ended. There may be many holes in the memory, so the operating system, to reduce the amount of time it spends analyzing the available spaces, begins at the start of primary memory and allocates memory from the first hole it encounters large enough to satisfy the request.
- **BEST-FIT:**
  - Allocate the smallest hole that is big enough. We must search the entire list, unless the list is kept ordered by size. This strategy produces the smallest leftover hole.
- **WORST-FIT:**
  - Allocate the largest hole that is big enough. We must search the entire list, unless the list is kept ordered by size. This strategy produces the largest leftover hole.
  - The idea is that this placement will create the largest hold after the allocations, thus increasing the possibility that compared to best fit, another process can use the remaining space.

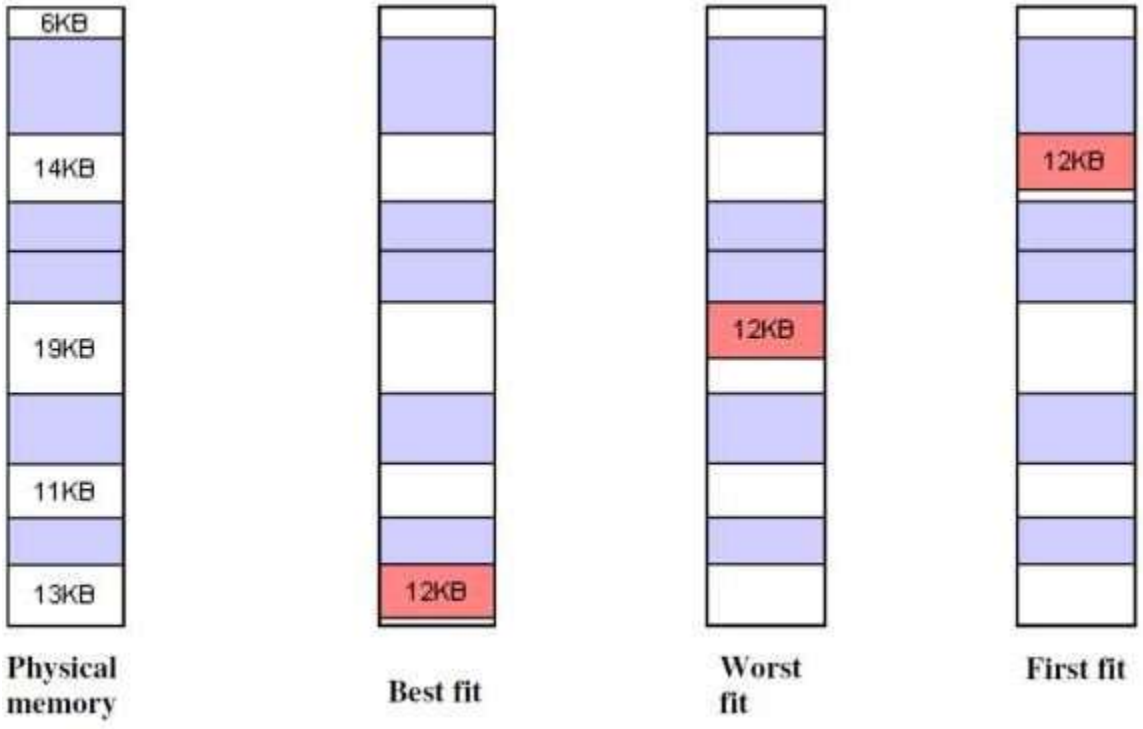


Fig 7 : Best, Worst, First Fit

FRAGMENTATION:

- As processes are loaded and removed from memory, the free memory space is broken into little pieces.
- It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

OR

The user of a computer continuously load and unload the processes from main memory. Processes are stored in blocks of the main memory. When it happens that there are some free memory blocks but still not enough to load the process, then this condition is called fragmentation.

Fragmentation is a condition that occurs when we dynamically allocate the RAM to the processes, then many free memory blocks are available but they are not enough to load the process on RAM.

TYPES:

- **External fragmentation**
- **Internal**

fragmentation EXTERNAL

FRAGMENTATION:

- External Fragmentation happens when a dynamic memory allocation algorithm allocates some memory and a small piece is left over that cannot be effectively used.
- If too much external fragmentation occurs, the amount of usable memory is drastically reduced.
- Total memory space exists to satisfy a request, but it is not contiguous.
- The problem of fragmentation can be solved by **COMPACTION.**
- The goal is to shuffle the memory contents to place all free memory together in one large block.
- For a relocated process to be able to execute in its new location, all internal addresses (e.g., pointers) must be relocated.
- If the relocation is static and is done at assembly or load time, compaction cannot be done.
- Compaction is possible only if relocation is dynamic and is done at execution time.
- If addresses are relocated dynamically, relocation requires only moving the program and data, and

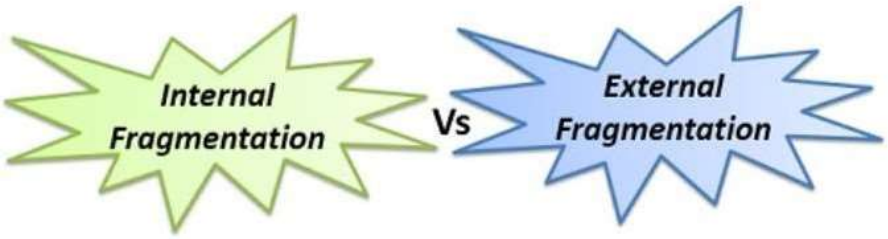
then changing the base register to reflect the new base address.

There may be many compaction algorithms:

- Simply move all processes toward one end of the memory; all holes move in the other direction producing one large hole of available memory.
- Create a large hole big enough anywhere to satisfy the current request.

INTERNAL FRAGMENTATION

- Internal fragmentation is the space wasted inside of allocated memory blocks because of restriction on the allowed sizes of allocated blocks.
- Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.



INTERNAL FRAGMENTATION	EXTERNAL FRAGMENTATION
It occurs when fixed sized memory blocks are allocated to the processes.	It occurs when variable size memory space are allocated to the processes dynamically.
When the memory assigned to the process is slightly larger than the memory requested by the process this creates free space in the allocated block causing internal fragmentation.	When the process is removed from the memory, it creates the free space in the memory causing external fragmentation.
The memory must be partitioned into variable sized blocks and assign the best fit block to the process.	Compaction, paging and segmentation.

Table 3 : Internal vs External fragmentation

PAGING

- Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory.
- This scheme permits the physical address space of a process to be non – contiguous.
- Logical Address or Virtual Address (represented in bits): An address generated by the CPU
- Logical Address Space or Virtual Address Space(represented in words or bytes): The set of all logical addresses generated by a program
- Physical Address (represented in bits): An address actually available on memory unit
- Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses
- Physical memory is broken into fixed-size blocks called **FRAMES**.
- Logical memory is also broken into blocks of the same size called **PAGES.**

- When a process is to be executed, its pages (which are in the backing store) are loaded into any available memory frames. Thus the pages of a process may not be contiguous.

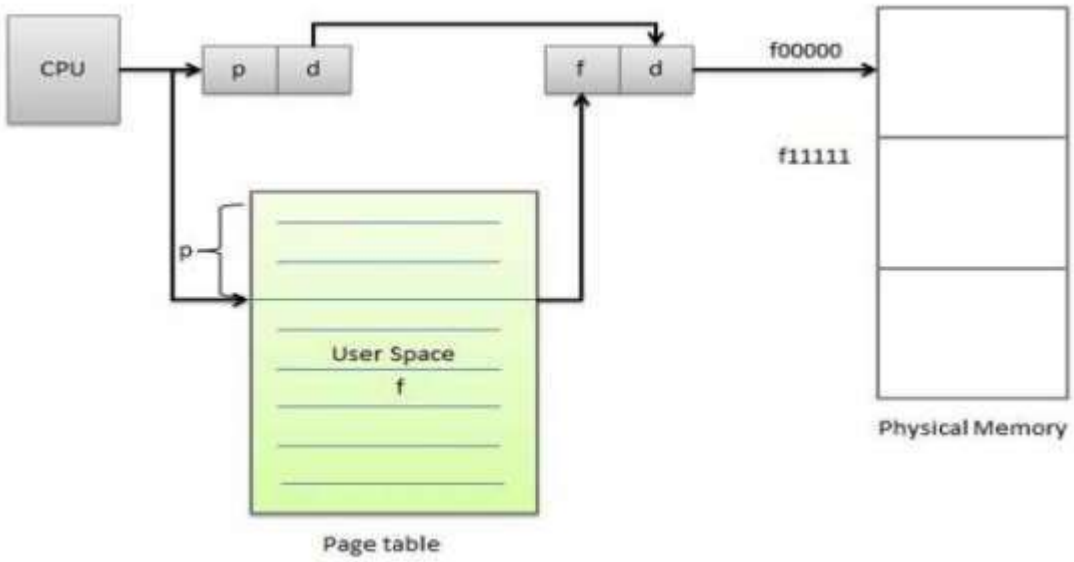


Fig 8 : Paging

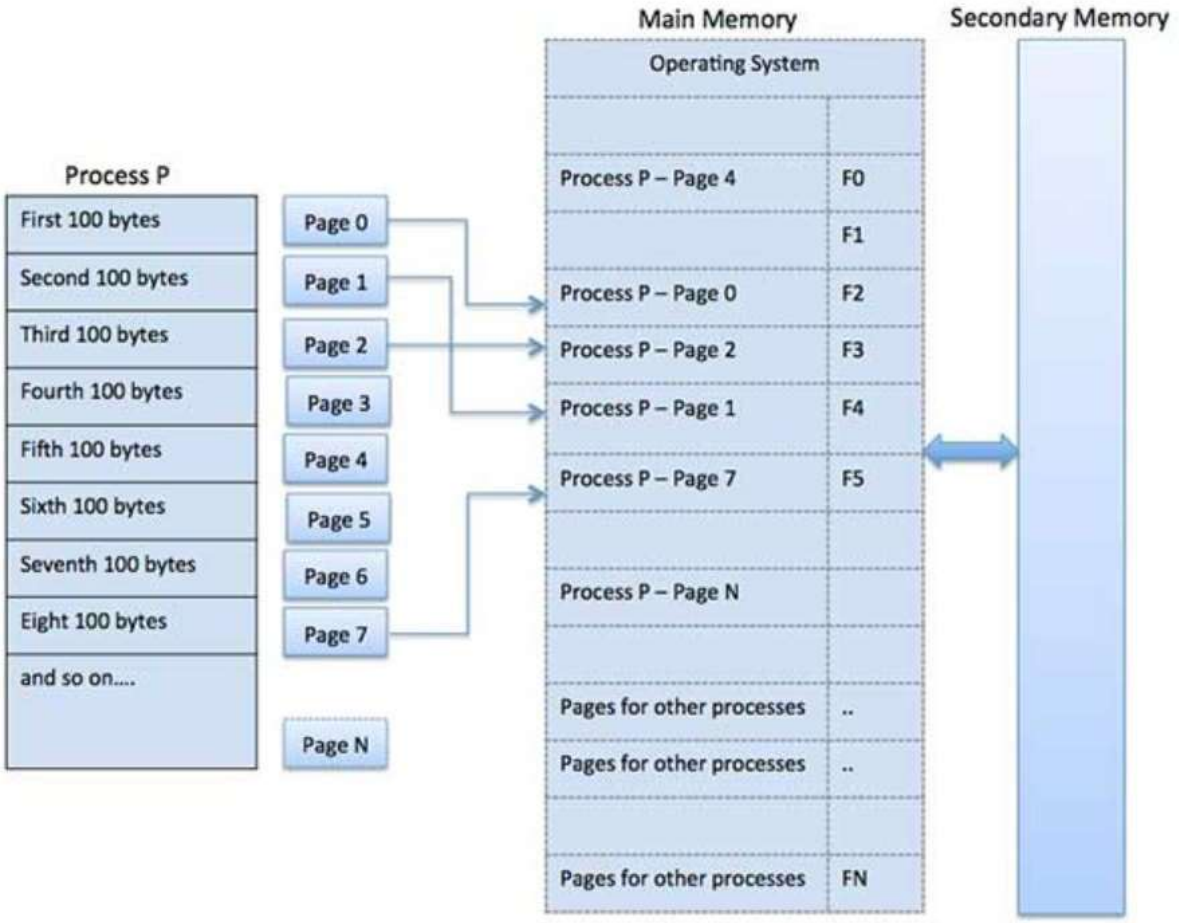


Fig 9 : Paging

ADDRESS TRANSLATION

- Page address is called **logical address** and represented by **page number** and the **offset**.
  - Logical Address = Page number + page offset
- Frame address is called **physical address** and represented by a **frame number** and the **offset**.
  - Physical Address = Frame number + page offset
- Paging eliminates external fragmentation altogether but there may be a little internal fragmentation.

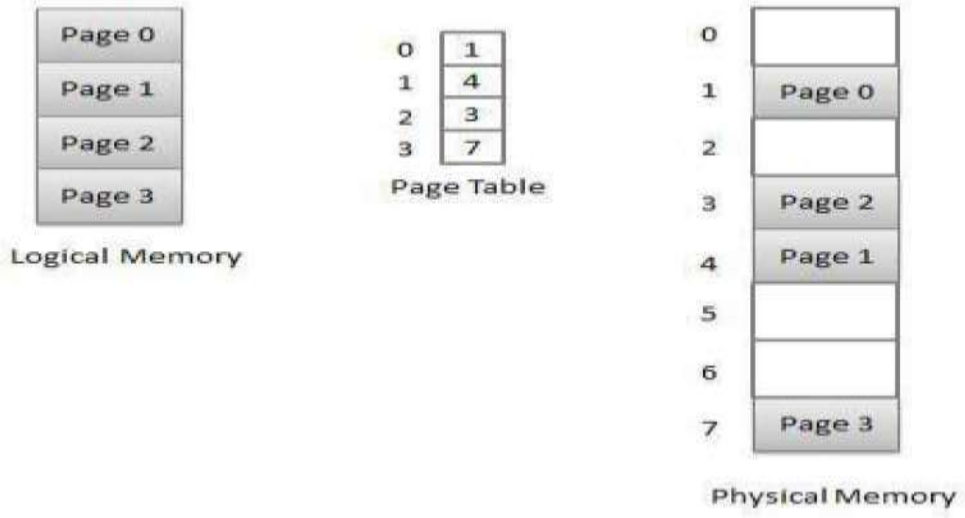


Fig 10 : Address Translation

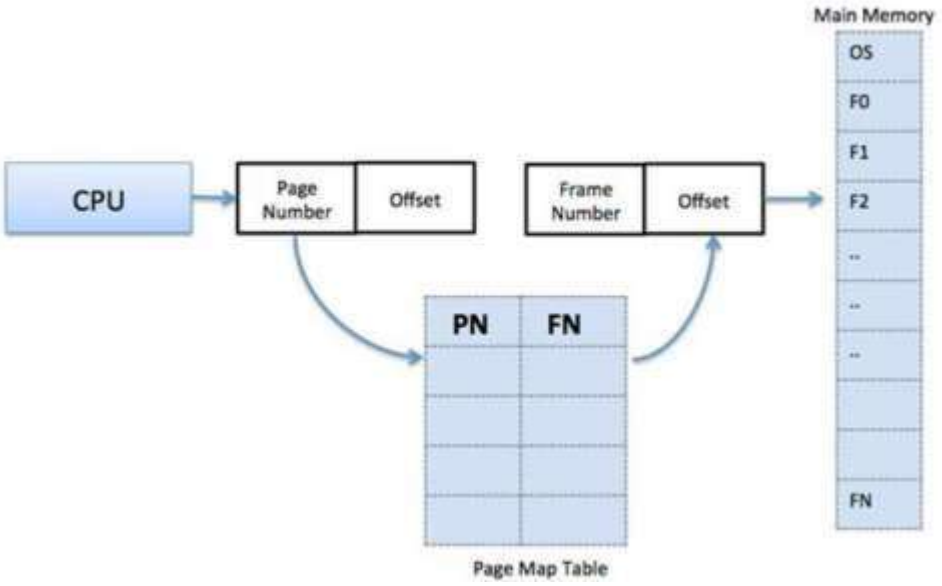


Fig 11 : Address translation

ADVANTAGES AND DISADVANTAGES OF PAGING

- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

MULTILEVEL PAGING

- Most computer systems support a very large logical address space (232 to 264). In such a case, the page table itself becomes very very large.
- E.g., consider a 32-bit logical address space. If the page size is 4K bytes (212), then a page table may consist of up to (232 / 212) = 1 million entries. If each entry consists of 4 bytes, each process may need 4 megabytes of physical address alone for the page table
- **DISADVANTAGE**: Page tables consume a large amount of physical memory because each page table can have millions of entries.

INVERTED PAGE TABLE

- To overcome the disadvantage of page tables given above, an inverted page table could be used. An inverted page table has one entry for each (frame) of memory.
- Each entry consists of the logical (or virtual) address of the page stored in that memory location, with information about the process that owns it.
- Thus there is only one inverted page table in the system, and it has only one entry for each frame of physical memory.
- **DISADVANTAGE:**
    - The complete information about the logical address space of a process, which is required if a referenced page is not currently in memory is no longer kept.
    - To overcome this, an external page table (one per process) must be kept. Each such table looks like the traditional per-process page table, containing information on where each logical page is located.
    - These external page tables need not be in the memory all the time, because they are needed only when a page fault occurs.

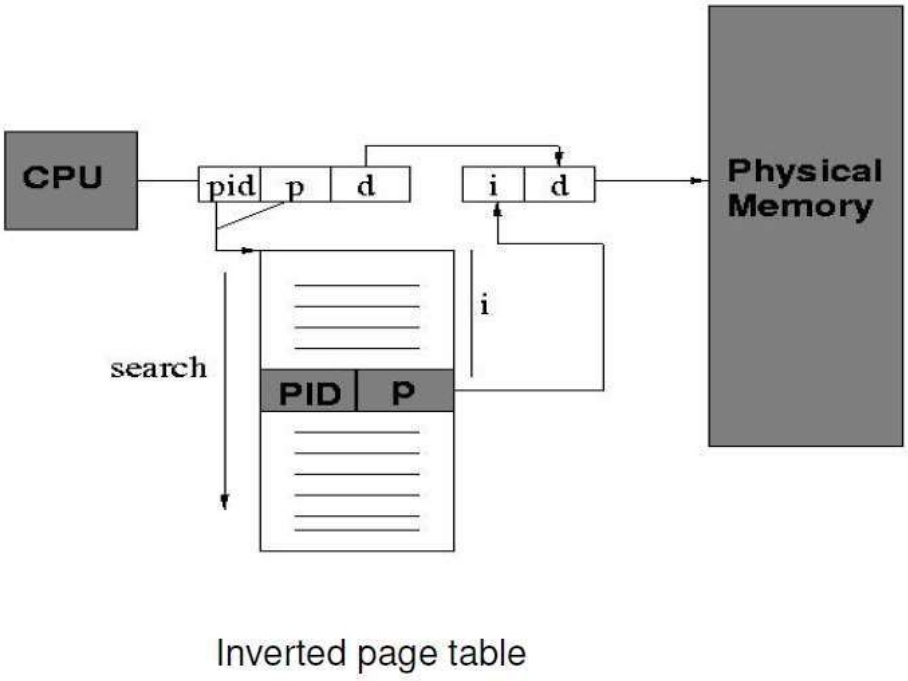


Fig 12 : Inverted page table

PROTECTION

- Memory protection in a paged environment is accomplished by protection bits that are associated with each frame. Normally, they are kept in the page table.
- One bit can define a page to be read-and-write or read-only.

SHARED PAGES

- Another advantage of paging is the possibility of sharing common code.
- Consider a system that supports 40 users, each of which executes a text editor.
- If the text editor consists of 150K of code and 50K of data space, then we would need 8000K to support the 40 users.
- If the code is reentrant (non-self-modifying), then it can be shared between the 40 users.

SEGMENTATION:

- Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions.
- Each segment is actually a different logical address space of the program.
- When a process is to be executed, its corresponding segmentations are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.
- Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.
- A program segment contains the program's main function, utility functions, data structures, and so on.
- The operating system maintains a segment map table for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory.

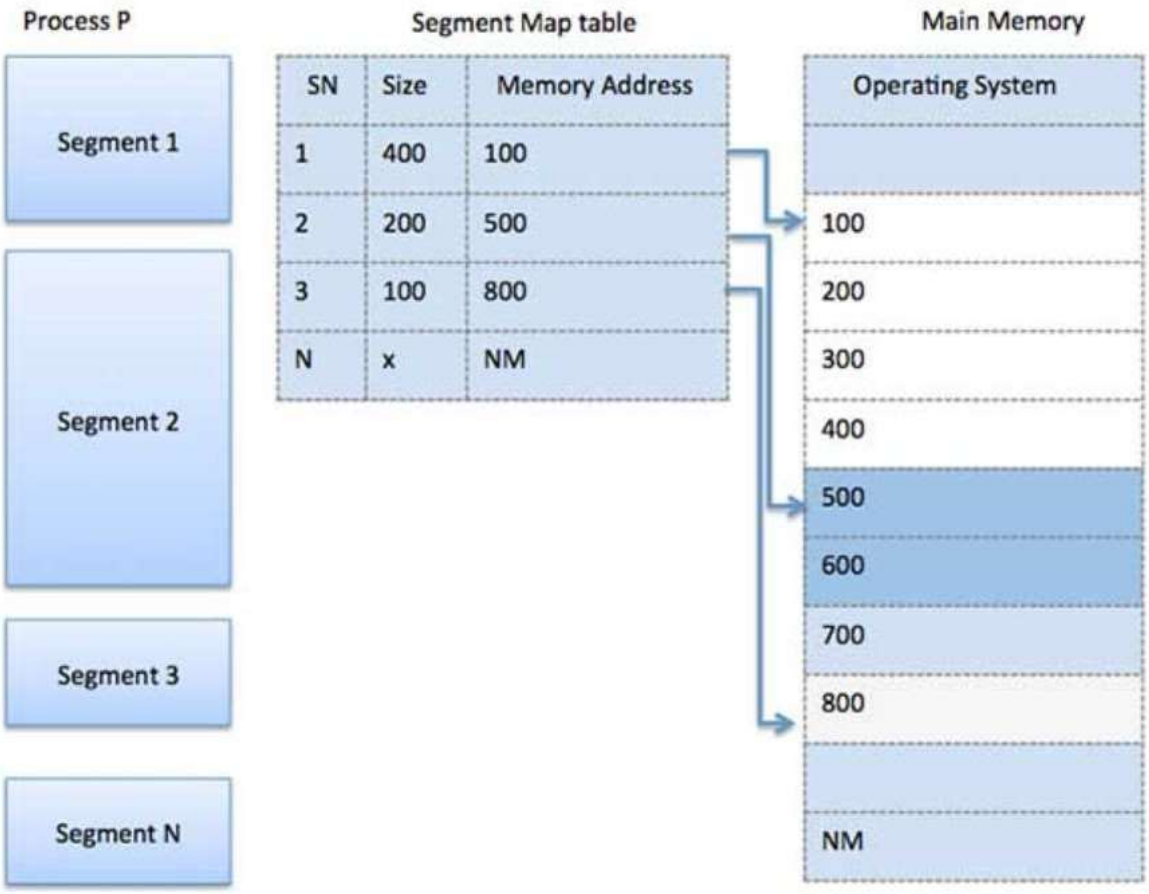


Fig 13 : Segmentation

- Segmentation is a memory-management scheme that suggests that a logical address space be divided into a collection of segments. Each segment has a name and a length.
- Addresses specify both the segment name and the offset within the segment.
- The user therefore specifies each address by two quantities: a segment name (or segment number) and an offset.
- **SEGMENT NUMBER (S)** -- segment number is used as an index into a segment table which contains base address of each segment in physical memory and a limit of segment.
- **SEGMENT OFFSET (O)** -- segment offset is first checked against limit and then is combined with base address to define the physical memory address.
- Therefore logical addresses consist of **two tuples**:
    - **<segment-number, offset>**

TUPLES:

- In the context of relational databases, a tuple is one record (one row).
- The information in a database can be thought of as a spreadsheet, with columns (known as fields or attributes) representing different categories of information, and tuples (rows) representing all the information from each field associated with a single record.

SEGMENTATION HARDWARE:

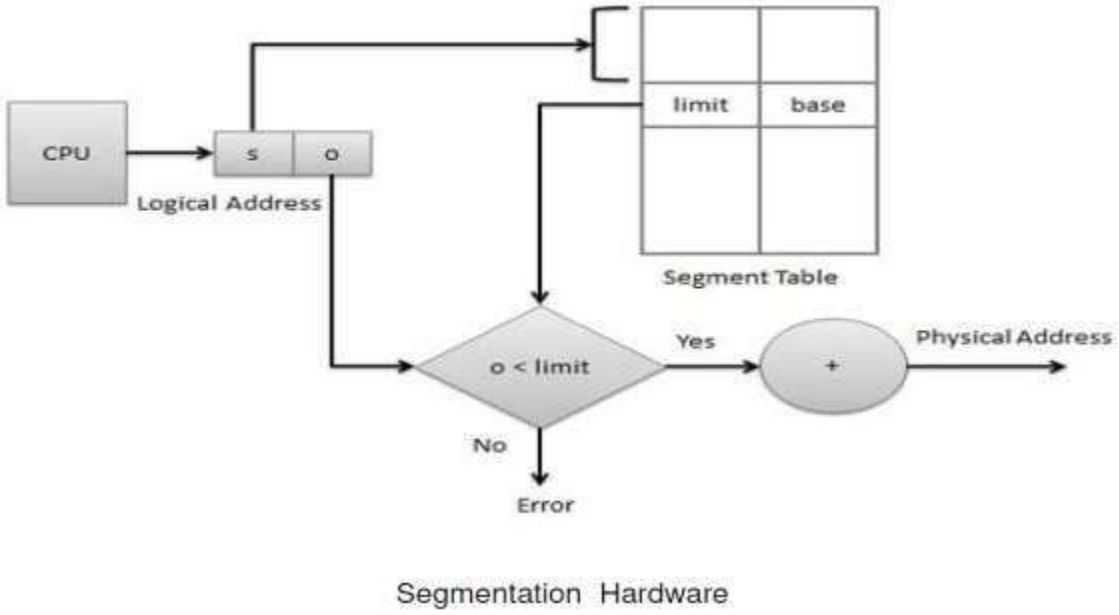


Fig 14 : Segmentation Hardware

- Each entry of the segment table has a segment base and segment limit.
- The segment base contains the starting physical address where the segment resides in the main memory, whereas the segment limit specifies the length of the segment.
- The main difference between the segmentation and multi-partition schemes is that one program may consist of several segments.
- The segment table can be kept either in fast registers or in memory.
- In case a program consists of several segments, we have to keep them in the memory and a **segment-table base register (STBR)** points to the segment table. Moreover, because the number of segments used by a program may vary widely, a **segment-table length register (STLR)** is used.

- One advantage of segmentation is that it automatically provides protection of memory because of the segment-table entries (base and limit tuples).
- Segments also can be shared in a segmented memory system. Segmentation may cause external fragmentation.

PAGED-SEGMENTATION

Paged Segmentation

In paged segmentation, we divide every segment in a process into fixed size pages. We need to maintain a page table per segment CPU's memory management unit must support both segmentation and paging. The following snapshots illustrate these points.

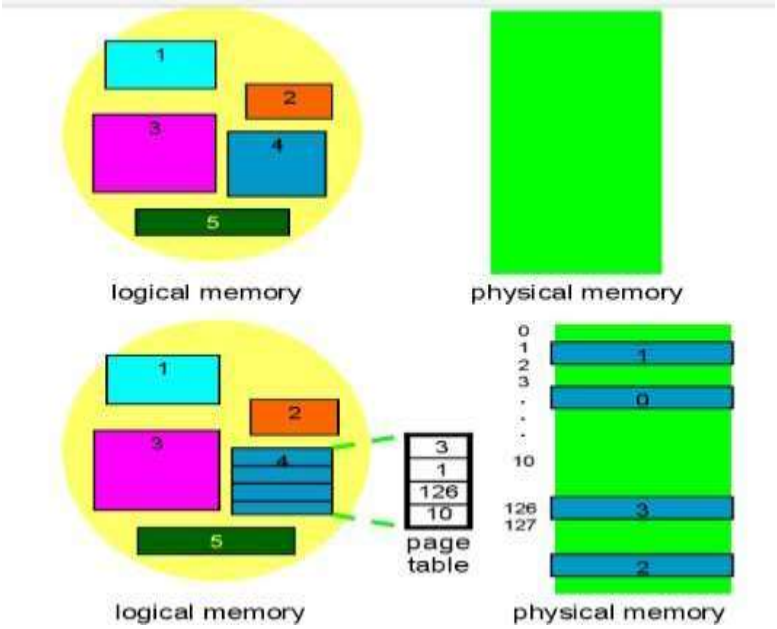
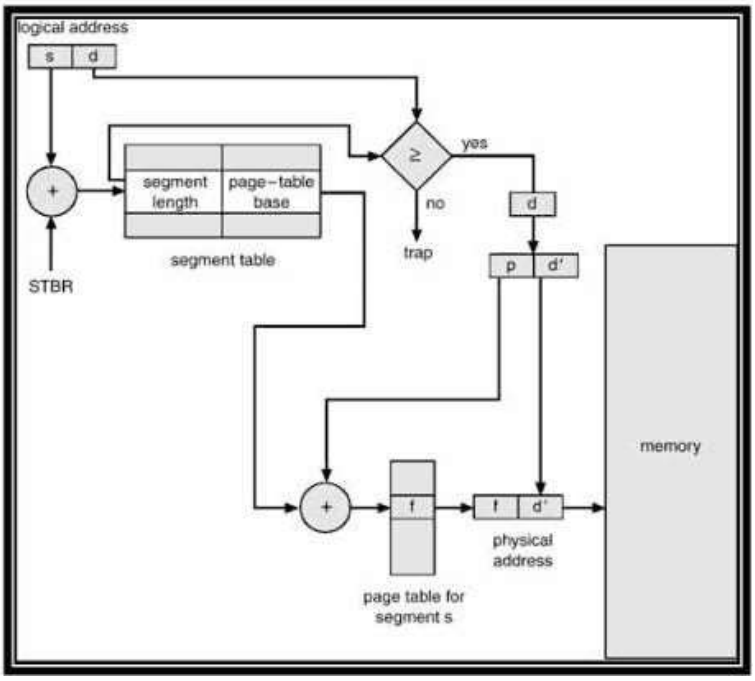


Fig 15 : Paged Segmentation



MULTICS:

We now take the example of one of the finest operating systems of late 1960s and early 1970s, known as the MULTICS operating system. Here are the specifications of the CPU supported by MULTICS and calculation of its various parameters such as the largest segment size supported by MULTICS.

- GE 345 processor
- Logical address = 34 bits

ADVANTAGES OF SEGMENTATION

- No Internal fragmentation.
- Segment Table consumes less space in comparison to Page table in paging.

DISADVANTAGE OF SEGMENTATION

- As processes are loaded and removed from the memory, the free memory space is broken into little pieces, causing External fragmentation.

PAGE REPLACEMENT

- In a operating systems that use paging for memory management, page replacement algorithm are needed to decide which page needed to be replaced when new page comes in.
- Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page.
- Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.

PAGE FAULT

- A page fault is a type of interrupt, raised by the hardware when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory

PAGE REPLACEMENT ALGORITHMS

- There are many page-replacement algorithms. We select a particular replacement algorithm based on the one with the lowest page-fault rate.
- An algorithm is evaluated by running it on a particular string of memory references (called a reference string) and computing the number of page-faults.
- Reference strings are generated artificially (by a random-number generator, for example) or by tracing a given system and recording the address of each memory reference.

TYPES OF REPLACEMENT ALGORITHMS

- **FIFO –** First In First Out
- **LRU –** Least Recently Used
- **Optimal Algorithm**
- **LFU -** Least Frequently Used
- **MFU -** Least Frequently Used

PAGE REPLACEMENT ALGORITHMS:

* The Longest time period repeated pages are to be replaced first.

* page fault: The CPU will demand for the particular page number; If it is not present in the reference string (RAM / main memory). Then it is going to be Page Fault.

* Page Hit: The CPU will demand for the particular page number; If it is present in the RAM/ main memory. Then it is going to be Page Hit.

TYPES: → FIFO: First In First Out.
→ LRU: Least Recently Used.
→ Optimal Algorithm.

} with the help of this 3, we can find page fault, Page Hit, Hit ratio.

FIFO:

* First In First Out.

* From the name itself we can recognize that which page has to be replaced first and which page has to be Deleted first.

* Replace the pages that has been in the RAM/ main memory for the longest time period.

* The Example as follows,

Reference String :

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

* These page numbers are present in the RAM / memory
* Because the main memory size is lesser than the logical memory.
* The main memory can't store all the pages at a time.
* If the particular demanded page number not present in main memory. It has to be borrowed from secondary memory, by deleting the longest time repeated page number.
* If the page faults are more than RAM gets slower.

(left side).

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

Page Frames															
F1	(7)	7	7	(2)	2	2	2	(4)	4	4	0	0	0	0	Present 0
F2		(0)	0	0	Present 0	(3)	3	3	(2)	2	2	2	1	1	1
F3			(1)	1	1	1	(0)	0	0	(3)	3	Present 3	3	2	2
	*	*	*	*	Hit	*	*	*	*	*	*	Hit	*	*	Hit

Page Hit : Hits → 3

Pag faults : * → 12

Reference string → 15.

Explanation for FIFO:

* Starting RAM / Main memory is free. It does not contains any page numbers.

* It contains (RAM) only three frames for Replacing the longest time period page numbers.

* The CPU is demanding for Page no: 7..

The RAM doesn't contains any page no in the staiting. So CPU will borrow the particular page number from RAM. So page no. 7 is not there in RAM. Then it is going to be 'Page fault *'

* The process continuous like this.....

* Now the CPU is demanding for page '0'.

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

⇒ It is present in the given Reference String Then it is going to be 'page Hit - Hit'.

$$\text{Hit ratio} = \frac{\text{No. of Hits}}{\text{total no. of. Reference String}}$$

$$= \frac{3}{15} \times 100 = 20\% \text{ (approx)}$$

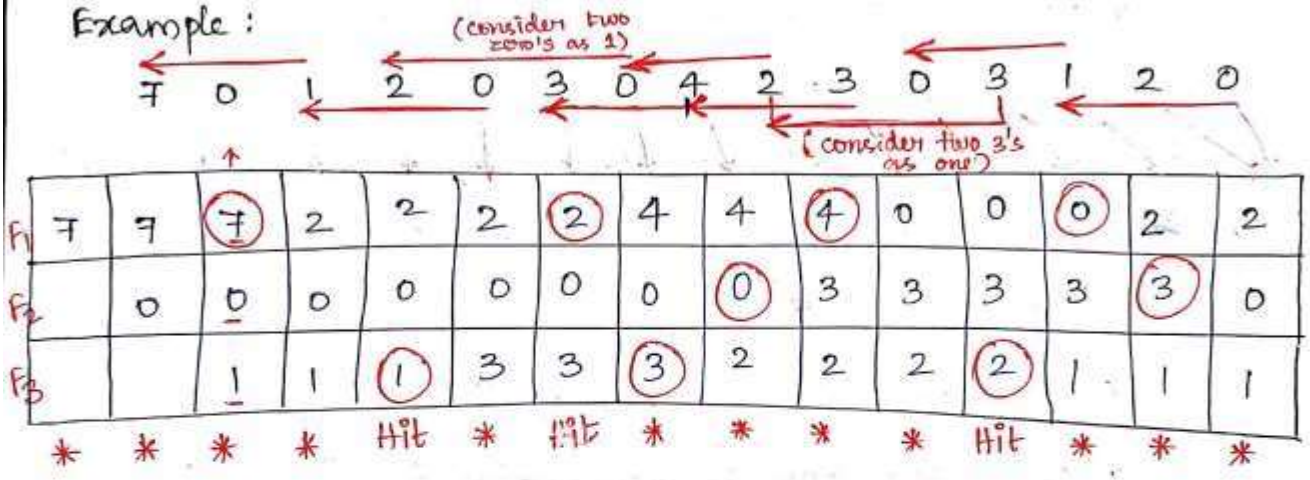
$$\text{page fault} = \frac{\text{No. of fault}}{\text{Ref. string}}$$

$$= \frac{12}{15} \times 100 = 80\% \text{ (approx)}.$$

LRU:

* Least Recently Used.
* Note : 3 Frames = 3 page numbers. (compare)
* Left side page numbers are replaced with another.

Example :



* Step 1 : 7 0 1 (frame) = 7 0 1 (page no). check the Arrow [For your Reference].
* Step 2 : compare the page numbers and frames in left direction.
* Don't compare the Frame F2 for the particular time period.
* Check the frames with given page numbers.

For example : 7 0 1 (1 is most recently used. 7 is Least recently used. so select '7' and replace with next page number '2'.)

The process continuous.....

* Now, check the frame : 4 0 2 and page number (0 & 2). In frames 2 is very most recently used ; 4 is mostly recently used (only 2 times repeated. we should replace the frame with another page number only if particular page number repeats 3 is above times). So In this condition you just delete the frame (F_2) - '0'. with '3'.

The process continuous .....

page Hit = 3

Page fault = 12.

OPTIMAL ALGORITHM:

* This algorithm represents the longest time period page number in future (Right Direction).

Example:

longest time in future. →

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7
7	7	(7)	2	2	2	2	2	2	2	2	2	2	2	2	2	(2)	7
	0	0	0	0	0	(0)	4	4	(4)	0	0	0	0	0	0	0	0
		1	1	(1)	3	3	3	3	3	3	3	(3)	1	1	1	1	1
*	*	*	*	Hit	*	Hit	*	Hit	Hit	*	Hit	Hit	*	Hit	Hit	Hit	*

Explanation:

* The process starts as usual ;

* Now, check for longest time in future.

⇒ 7 0 1 (frames) ; comparing these numbers in given Reference string. 7 is most longest traveled page number (i.e., Repeated in future).

These processes are continues .....

* Now, page number 4 is not there in future reference string. In this condition just replace the longest time repeated frame with another page number (i.e) '0' is replaced with 4.

* Then process repeats. After that 4 is not there in future reference string. So if it is in frames also no use. So just delete that page number and replace '0'.

2 (4) 3 replace with 2 0 3

* The process continuous. Last 7 is not there in future. So just replace with longest traveled page number (2) with 7.

Page Hit – 9

Page fault – 9.

LFU (Least Frequently Used) ALGORITHM

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

MFU (Least Frequently Used) ALGORITHM

- Replace the most frequently used page
- **Disadvantage:**
  - This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used. Neither MFU nor LFU replacement is common. Their implementation is fairly expensive.

VIRTUAL MEMORY

- Virtual memory is a technique that allows the execution of processes that may not be completely in memory.
- In many cases, in the course of execution of a program, some part of the program may never be executed.
- The advantages of virtual memory are:
  - Users would be able to write programs whose logical address space is greater than the physical address space.
  - More programs could be run at the same time thus increasing the degree of multiprogramming.
  - Less I/O would be needed to load or swap each user program into memory, so each program would run faster.

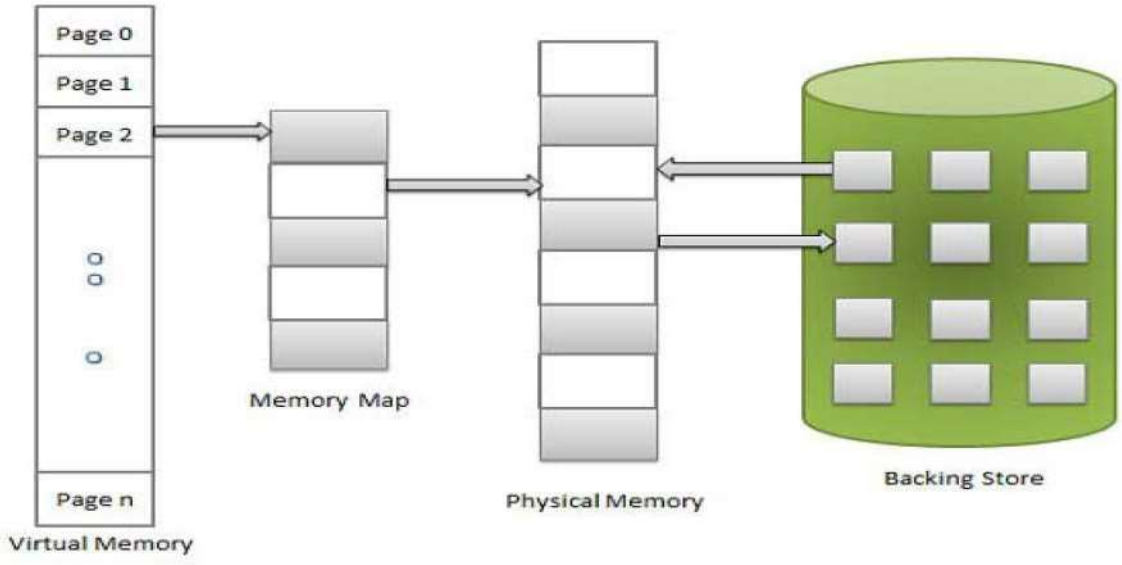


Fig 16 : Virtual Memory

- Virtual memory is the separation of logical memory from physical memory.
- Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system.

DEMAND PAGING

- A demand-paging system is similar to a paging system with swapping.
- When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again.
- Instead of swapping in a whole process, the pager brings only those necessary pages into the memory.
- Thus, it avoids reading into memory pages that will not be used anyway, decreasing the swap time and the amount of physical memory needed.

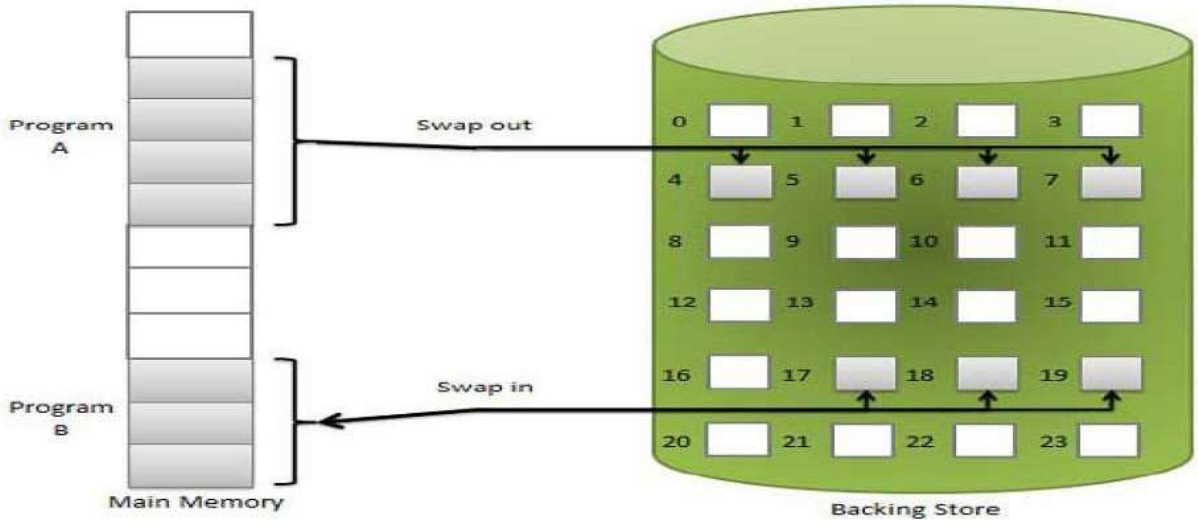


Fig 17 : Demand paging

- To distinguish between those pages that are in memory and those that are on disk, we use an invalid-valid bit which is a field in each page-table entry.
- When this bit is set to “valid”, this value indicates that the associated page is both legal and in memory.
- If the bit is set to “invalid”, it indicates that the page is either not valid (i.e., not in logical address space of the process), or is valid but is currently on the disk.
- The page-table entry for a page that is not currently in memory is simple marked invalid, or contains the address of the page on disk.
- Access to a page marked invalid causes a page-fault trap. The procedure for handling page fault is given below:

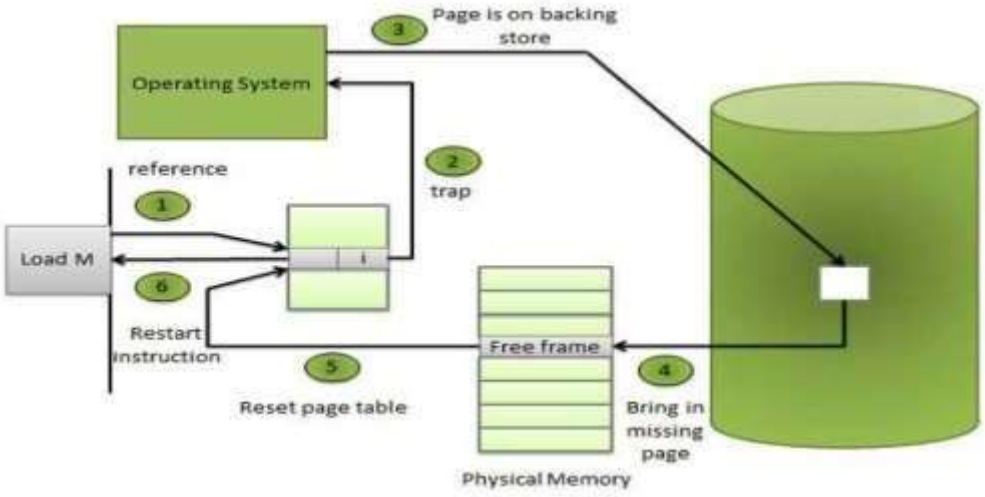


Fig 17 : Demand paging

ADVANTAGES

- Large virtual memory.
- More efficient use of memory.
- Unconstrained multiprogramming.
- There is no limit on degree of multiprogramming.

DISADVANTAGES

- Number of tables and amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.
- Due to the lack of explicit constraints on jobs address space size.

PURE DEMAND PAGING

- In the extreme case, we could start executing a process with no pages in memory.
- When the operating system set the instruction pointer to the first instruction of the process which is on a non-memory-resident page, the process would immediately fault for the page.
- After this page was brought into memory, the process would continue to execute, faulting as necessary until every page that is needed was actually in memory.
- Then, it could execute with no more faults. This scheme is called **pure demand paging.**

DEMAND PAGING vs ANTICIPATORY PAGING

DEMAND PAGING

- When a process first executes, the system loads into main memory the page that contains its first instruction
- After that, the system loads a page from secondary storage to main memory only when the process explicitly references that page
- Requires a process to accumulate pages one at a time

ANTICIPATORY PAGING

- Operating system attempts to predict the pages a process will need and preloads these pages when memory space is available
- Anticipatory paging strategies must be carefully designed so that overhead incurred by the strategy does not reduce system performance.

FILE CONCEPT

- A file is a named collection of related information that is recorded on secondary storage.
- Data can NOT be written to secondary storage unless they are within a file.

FILE STRUCTURE

- A text file is a sequence of characters organized into lines.
- A source file is a sequence of subroutines and function.
- An object file is a sequence of bytes organized into blocks understandable by the system's linker
- An executable file is a series of code sections that the loader can bring into memory and execute.

FILE ATTRIBUTES

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring

- Information about files are kept in the directory structure, which is maintained on the disk

FILE OPERATIONS

- The operating system provides system calls to create, write, read, reposition, delete, and truncate files. We look at what the operating system must do for each of these basic file operations.
- **CREATING A FILE**
  - Firstly, space in the file system must be found for the file. Secondly, 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 system.
- **WRITING A FILE.**
  - In the system call for writing in a file, we have to specify the name of the file and the information to be written to the file.
  - The operating system searches the directory to find the location of the file.
  - The system keeps a write pointer to the location in the file where the next write is to take place.
  - The write pointer must be updated whenever a write occurs.
- **READING A FILE**
  - To read a file, we make a system call that specifies the name of the file and where (in memory) the next block of the file should be put.
  - As in writing, the system searches the directory to find the location of the file, and the system keeps a read pointer to the location in the file where the next read is to take place.
  - The read pointer must be updated whenever a read occurs.
- **REPOSITIONING WITHIN A FILE**
  - In this operation, the directory is searched for the named file, and the current-file-position is set to a given value. This file operation is called a **FILE SEEK**.
- **DELETING A FILE**
  - We search the directory for the appropriate entry.
  - Having found it, we release all the space used by this file and erase the directory entry**.**
- **TRUNCATING A FILE**
  - This operation is used when the user wants to erase the contents of the file but keep the attributes the file intact

MEMORY-MAPPED FILES

- Some operating systems allow mapping sections of file into memory on virtual-memory systems.
- It allows part of the virtual address space to be logically associated with a section of a file.
- Reads and writes to that memory region are then treated as reads and writes to the file, greatly simplifying the file use.
- Closing the file results in all the memory-mapped data being written back to the disk and removed from the virtual memory of the process.

OPEN FILES

- **File pointer:** pointer to last read/write location, per process that has the file open
- **File-open count:** the counter tracks the number of opens and closes, and reaches zero on the last close. The system can then remove the entry.
- **Disk location of the file**: the info needed to locate the file on disk.
- **Access rights**: per-process access mode information so OS can allow or deny subsequent I/O request

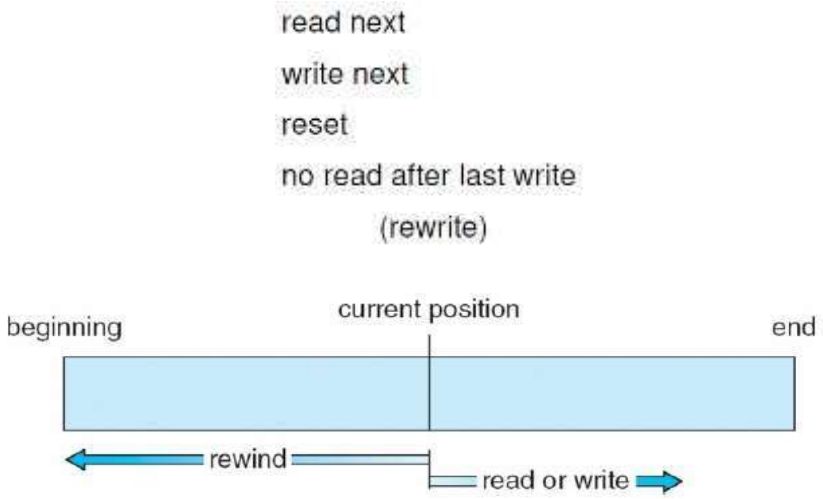
FILE TYPES – NAME, EXTENSION

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Table 4 : File Concepts and types

ACCESS METHODS

- **Sequential Access**



- **Direct Access**

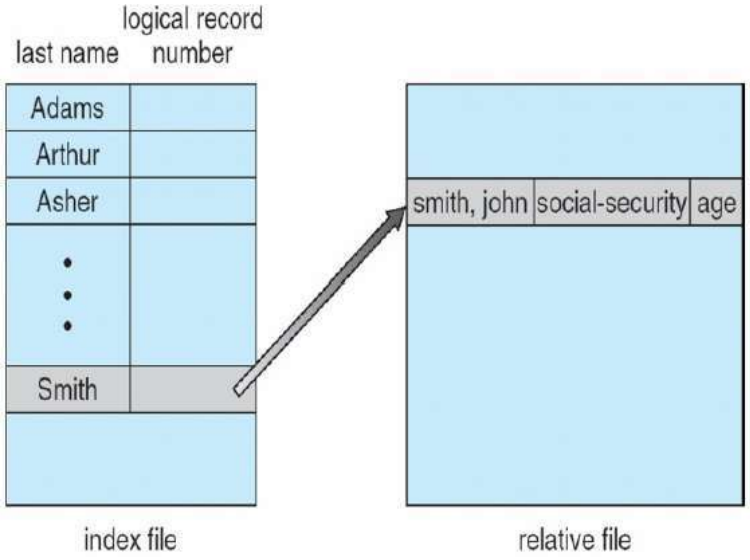
read n
write n
position to n
read next
write next
rewrite n
n = relative block number

- **Simulation of Sequential Access on Direct-access File**

sequential access	implementation for direct access
reset	*cp* = 0;
read next	*read cp;* *cp* = *cp* + 1;
write next	*write cp;* *cp* = *cp* + 1;

Cp=> current position

- **Index and Relative Files**



  - The index contains pointers to the various blocks.
  - To find a record in the file, we first search the index and then use the pointer to access the file directly and to find the desired record

DISK STRUCTURE

- Disk can be subdivided into partitions
- Disks or partitions can be **Redundant Arrays of Independent Disks (RAID)** protected against failure

- Disk or partition can be used raw – without a file system, or formatted with a file system
- Partitions also known as minidisks, slices
- Entity containing file system known as a volume
- Each volume containing file system also tracks that file system's info in device directory or volume table of contents
- There are five commonly used directory structures:
    - **Single-Level Directory**
    - **Two-Level Directory**
    - **Tree-Structure Directories**
    - **Acyclic-Graph Directories**
    - **General Graph**

Directories A TYPICAL FILE-SYSTEM

ORGANIZATION

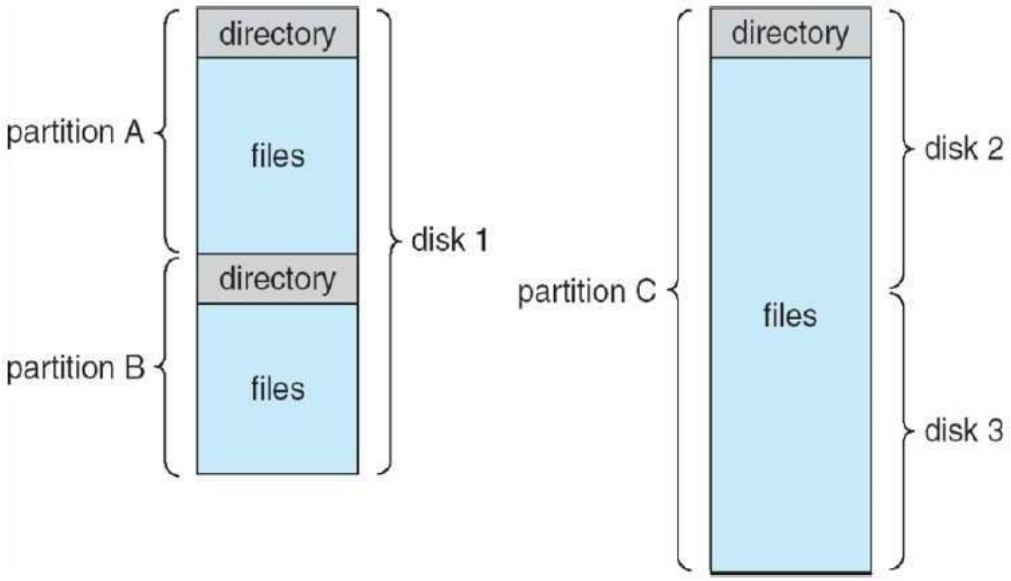


Fig 18 : File system

Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory

- Rename a file

- Traverse the file system

SINGLE-LEVEL DIRECTORY

- All files are contained in the same directory.
- It is difficult to maintain file name uniqueness.
- CP/M-80 and early version of MS-DOS use this directory structure.

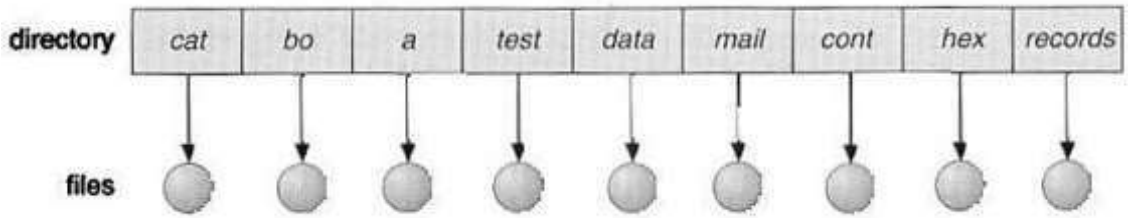


Fig 19 : Single level directory

TREE-STRUCTURED DIRECTORIES

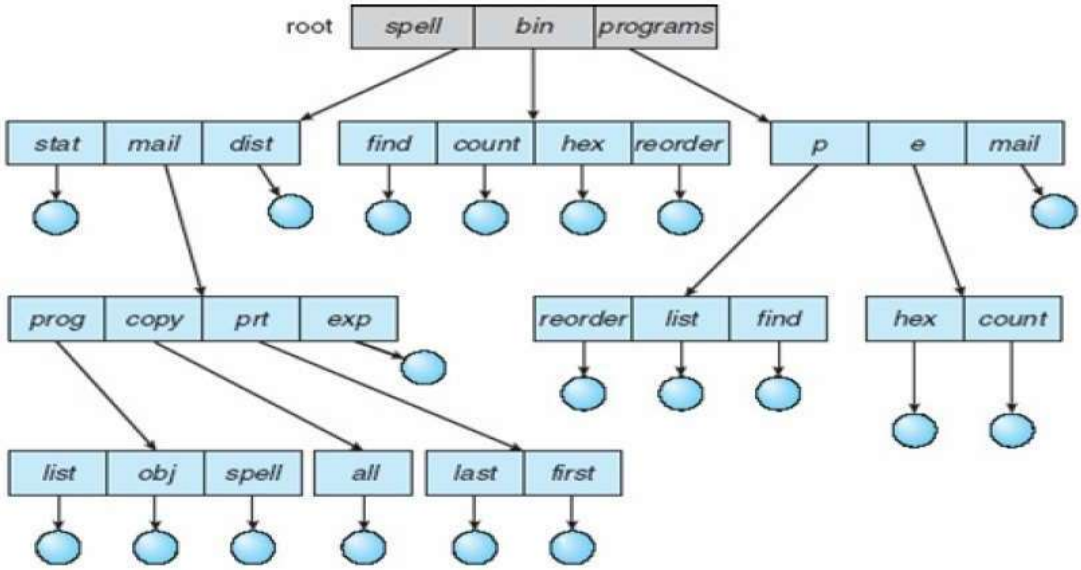


Fig 20 : Multi-level directory , Tree structured directory

- **ABSOLUTE PATH:** begins at the root and follows a path down to the specified file.
    - root/spell/mail/prt/first
- **RELATIVE PATH:** defines a path from the current directory.
    - prt/first given root/spell/mail as current path

ACYCLIC-GRAPH DIRECTORY

- This type of directories allows a file/directory to be shared by multiple directories.
- This is different from two copies of the same file or directory.
- An acyclic-graph directory is more flexible than a simple tree structure.

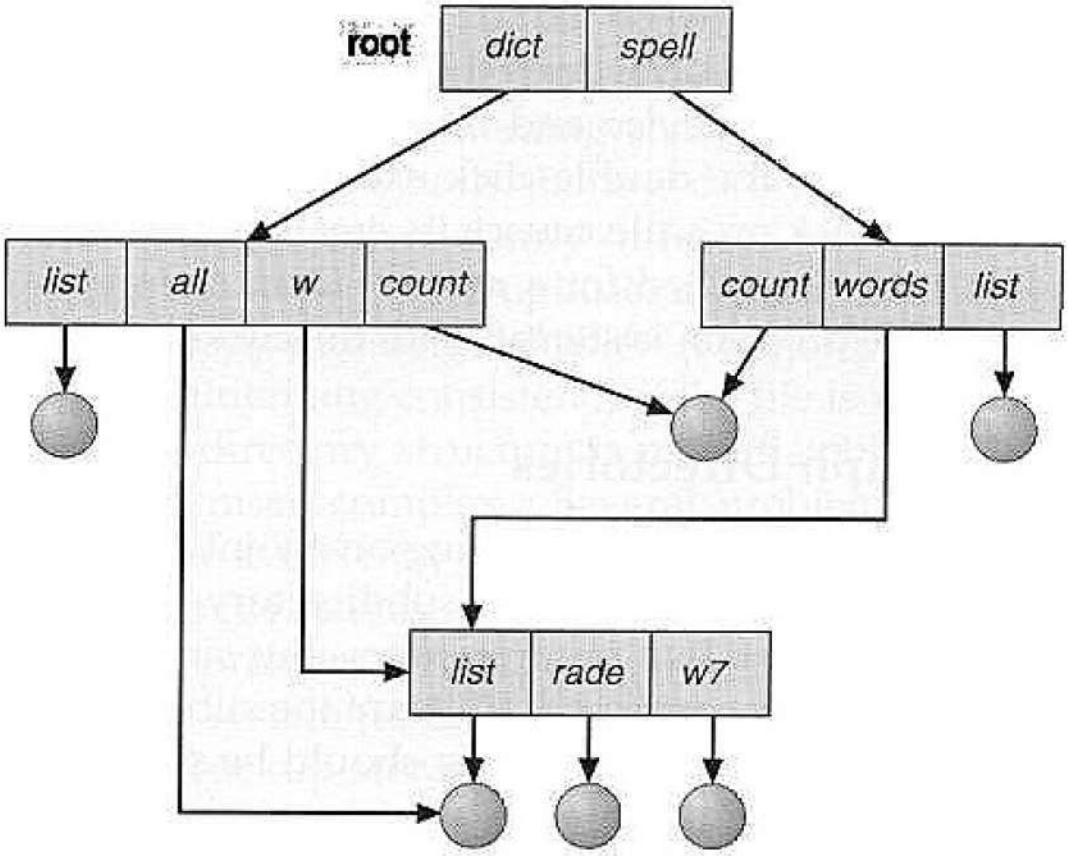


Fig 21 : Acyclic graph directory

GENERAL GRAPH DIRECTORY

- It is easy to traverse the directories of a tree or an acyclic directory system.
- However, if links are added arbitrarily, the directory graph becomes arbitrary and may contain cycles

- Creating a new file is done in current directory
    - Delete a file
        - rm <file-name>
    - Creating a new subdirectory is done in current directory
        - mkdir <dir-name>

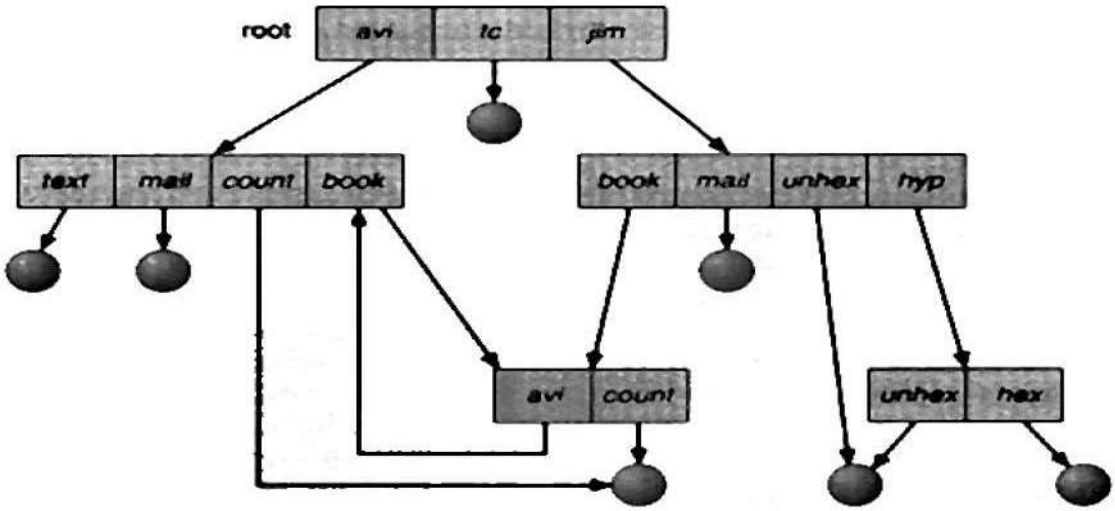
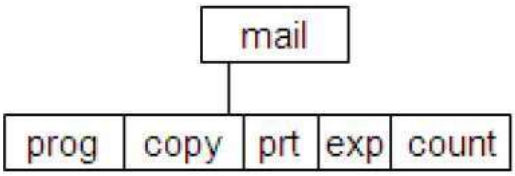


Fig 22 : General directory

Example:

if in current directory /mail

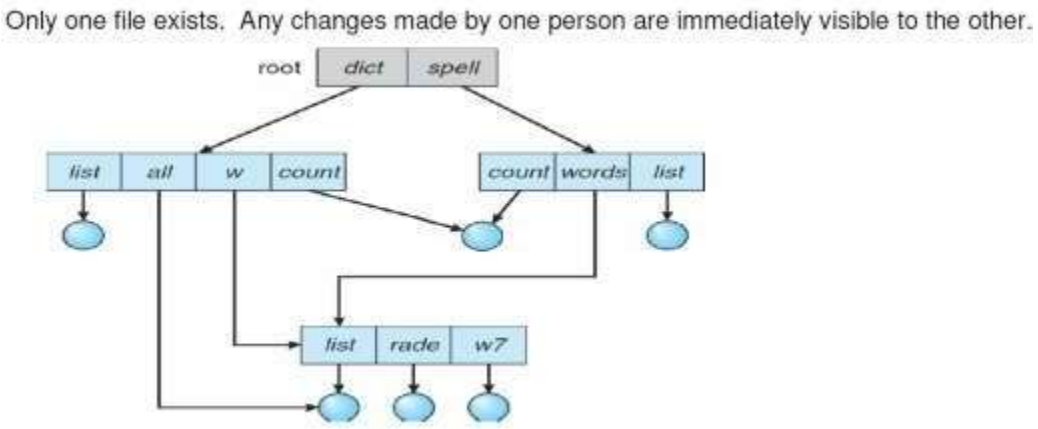
mkdir count



Deleting "mail" ⇒ deleting the entire subtree rooted by "mail"

Option1: do not delete a directory unless it is empty, such as MS-DOS

Option2: delete all files in that directory, such as UNIX rm command with r option



- Efficient searching
- Grouping Capability
    - pwd
    - cd /spell/mail/prog
    - New directory entry type
    - Link – another name (pointer) to an existing file
    - Resolve the link – follow pointer to locate the file

FILE SHARING

- When a file is shared by multiple users, how can we ensure its consistency
- If multiple users are writing to the file, should all of the writers be allowed to write? Or, should the operating system protect the user actions from each other?
- This is the file consistency semantics

FILE CONSISTENCY SEMANTICS

- Consistency semantics is a characterization of the system that specifies the semantics of multiple users accessing a shared file simultaneously.
- Consistency semantics is an important criterion for evaluating any file system that supports file sharing.
- There are three commonly used semantics
    - Unix semantics
    - Session Semantics
    - Immutable-Shared-Files Semantics

Unix Semantics

- Writes to an open file by a user are visible immediately to other users have the file open at the same time. All users share the file pointer.
- A file has a single image that interleaves all accesses, regardless of their origin

Session Semantics

- Writes to an open file by a user are not visible immediately to other users that have the same file open simultaneously
- Once a file is closed, the changes made to it are visible only in sessions started later.
- Already-open instances of the file do not affect these changes
- Multiple users are allowed to perform both read and write concurrently on their image of the file without delay.
- The Andrew File System (AFS) uses this semantics.

Immutable-Shared-Files Semantics

- Once a file is declared as shared by its creator, it cannot be modified.
- An immutable file has two important properties**:**
    - **Its name may not be used**
    - **Its content may not be**

altered FILE PROTECTION

- We can keep files safe from physical damage (i.e., reliability) and improper access (i.e., protection).
- Reliability is generally provided by backup.
- The need for file protection is a direct result of the ability to access files.
- Access control may be a complete protection by denying access. Or, the access may be controlled.

TYPES OF ACCESS

- **Read:** read from the file
- **Write:** write or rewrite the file
- **Execute**: load the file into memory and execute it
- **Append**: write new info at the end of a file

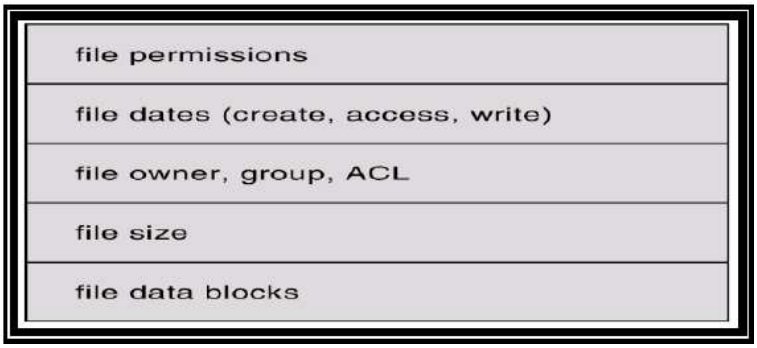
- **Delete:** delete a file
- **List:** list the name and attributes of the file

FILE-SYSTEM STRUCTURE

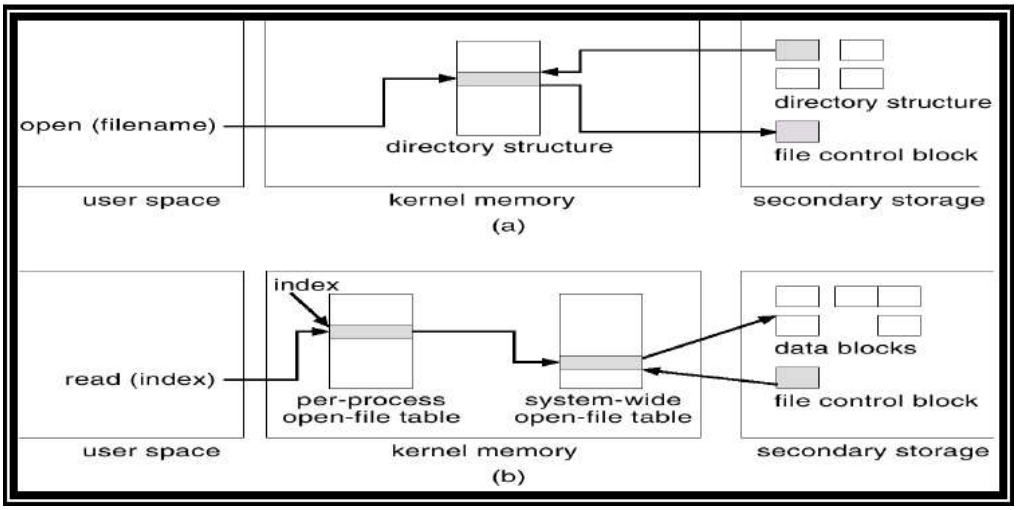
- Logical storage unit
- Collection of related information

Layered File System

A Typical File Control Block



In-Memory File System Structures



VIRTUAL FILE SYSTEMS

- Virtual File Systems (VFS) provide an object-oriented way of implementing file systems.
- VFS allows the same system call interface (the API) to be used for different types of file systems.
- The API is to the VFS interface, rather than any specific type of file system.

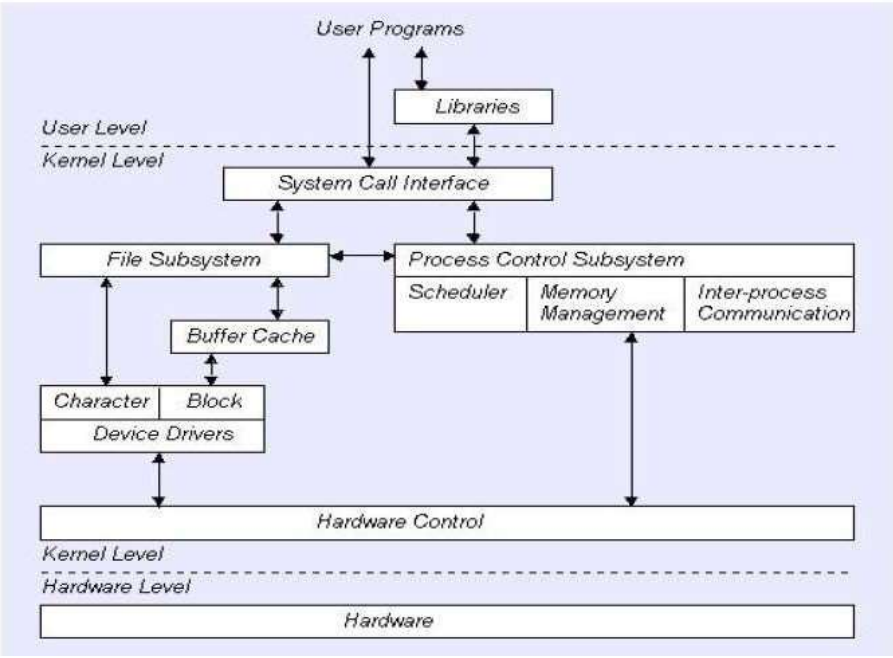
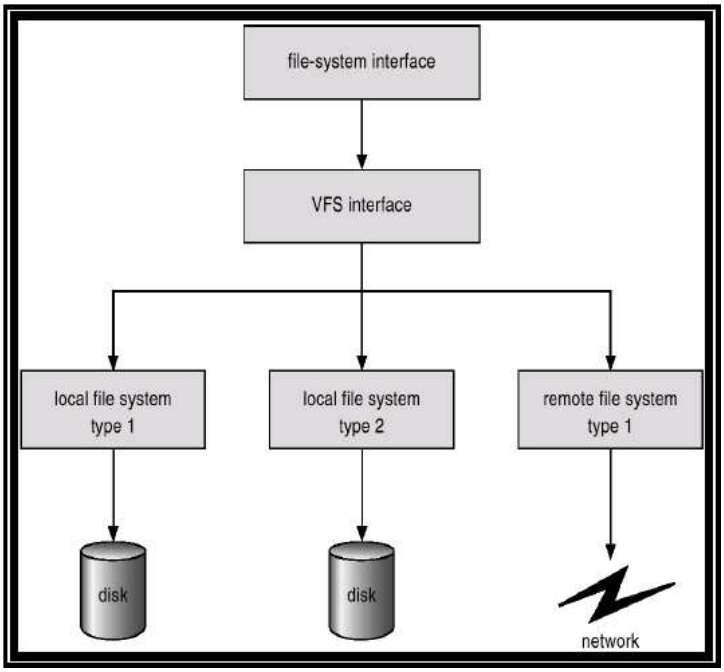


Fig 23 : Virtual file system

Schematic View of Virtual File System



ALLOCATION METHODS

- An allocation method refers to how disk blocks are allocated for files:
  - Contiguous allocation
  - Linked allocation
  - Indexed allocation

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk.
- Simple – only starting location (block #) and length (number of blocks) are required.
- Random access.
- Wasteful of space (dynamic storage-allocation problem).
- Files cannot grow.

Contiguous Allocation of Disk Space

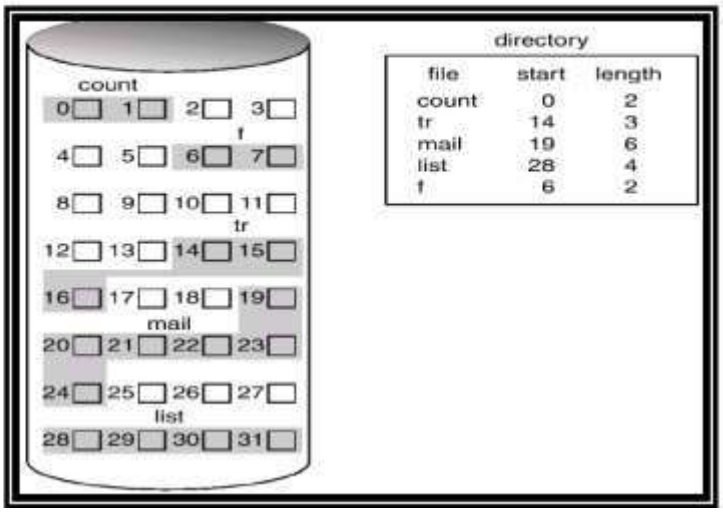


Fig 24 : Disk space

Linked Allocation

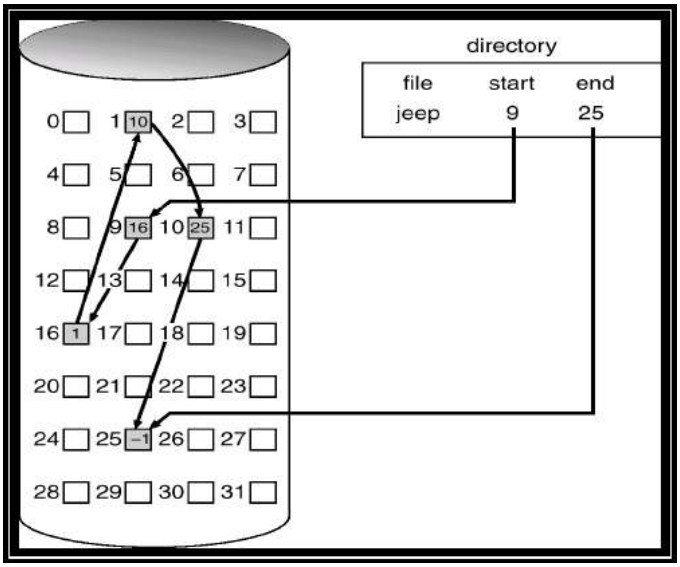
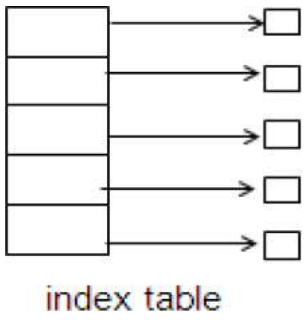


Fig 25 : linked allocation

- Simple – need only starting address
- Free-space management system – no waste of space

- No random access
- Space waste for pointer (e.g. 4byte of 512 B)
- Reliability

Indexed Allocation



- Brings all pointers together into the index block.

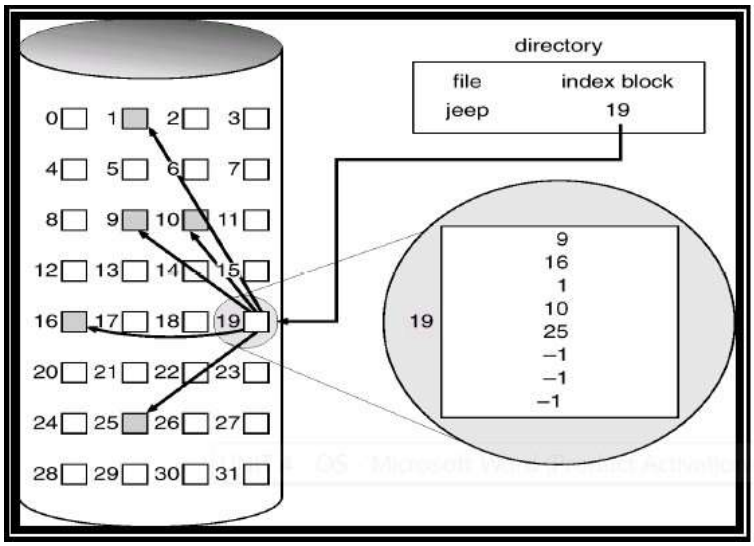
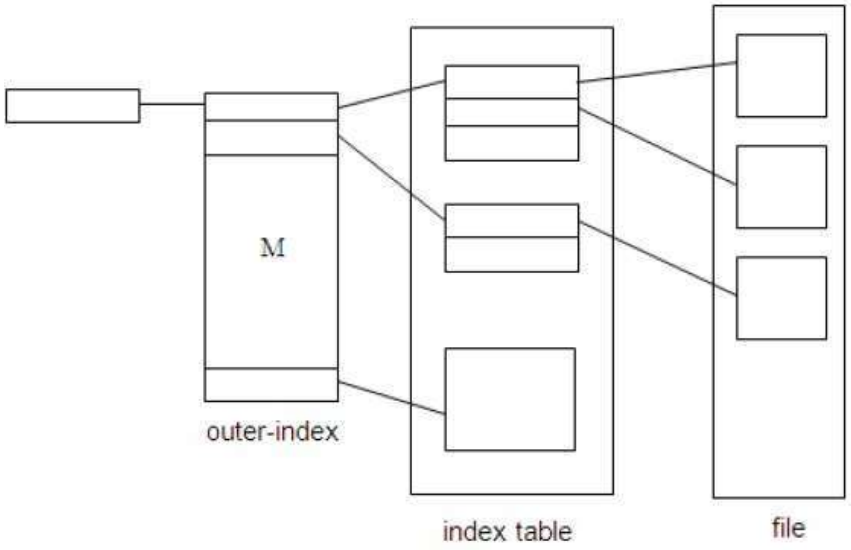


Fig 26 : Indexed allocation

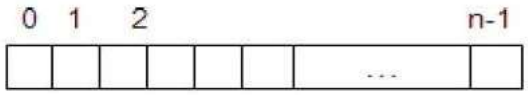
- Need index table
- Random access
- Dynamic access without external fragmentation, but have overhead of index block

Indexed Allocation – Mapping



Free-Space Management

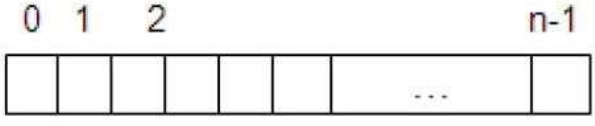
Bit vector (*n* blocks)



$$bit[i] = \begin{cases} 0 \Rightarrow block[i] \text{ free} \\ 1 \Rightarrow block[i] \text{ occupied} \end{cases}$$

Block number calculation

(number of bits per word) *
(number of 0-value words) +
offset of first 1 bit



$$bit[i] = \begin{cases} 0 \Rightarrow block[i] \text{ free} \\ 1 \Rightarrow block[i] \text{ occupied} \end{cases}$$

- Bit map requires extra space. Example:

  block size = 2^{12} bytes (4 K)

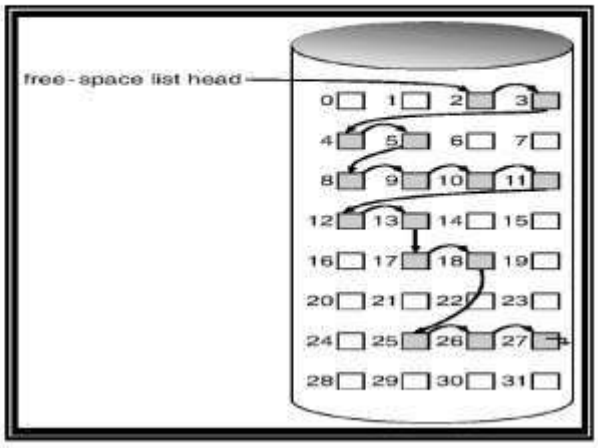
  disk size = 2^{30} bytes (1 gigabyte)

  $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)

  - Easy to get contiguous files
- Linked list (free list)
  - Cannot get contiguous space easily
  - No waste of space
- Grouping
  - Large free blocks can be quickly found
- Counting
- Need to protect:
  - Pointer to free list
  - Bit map

- Must be kept on disk
- Copy in memory and disk may differ.

- Linked Free Space List on Disk



EFFICIENCY AND PERFORMANCE

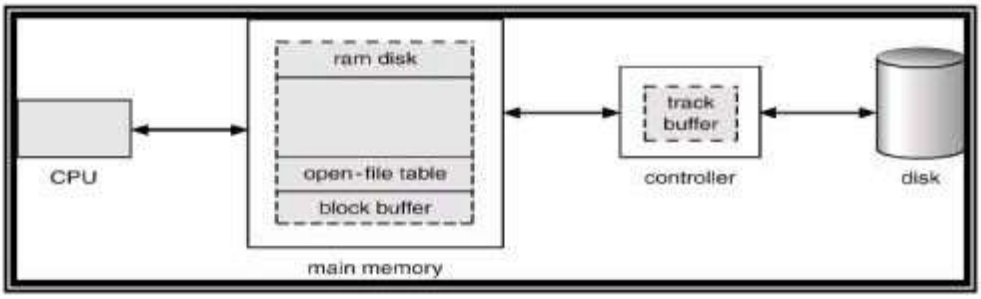
- **Efficiency dependent on:**

  disk allocation and directory algorithms

  types of data kept in file's directory entry

- **Performance**
  - disk cache – separate section of main memory for frequently used blocks
  - free-behind and read-ahead – techniques to optimize sequential access
  - improve PC performance by dedicating section of memory as virtual disk, or RAM disk.

Various Disk-Caching Locations



PAGE CACHE

- A page cache caches pages rather than disk blocks using virtual memory techniques.
- Memory-mapped I/O uses a page cache.
- Routine I/O through the file system uses the buffer (disk) cache.
- This leads to the following figure.

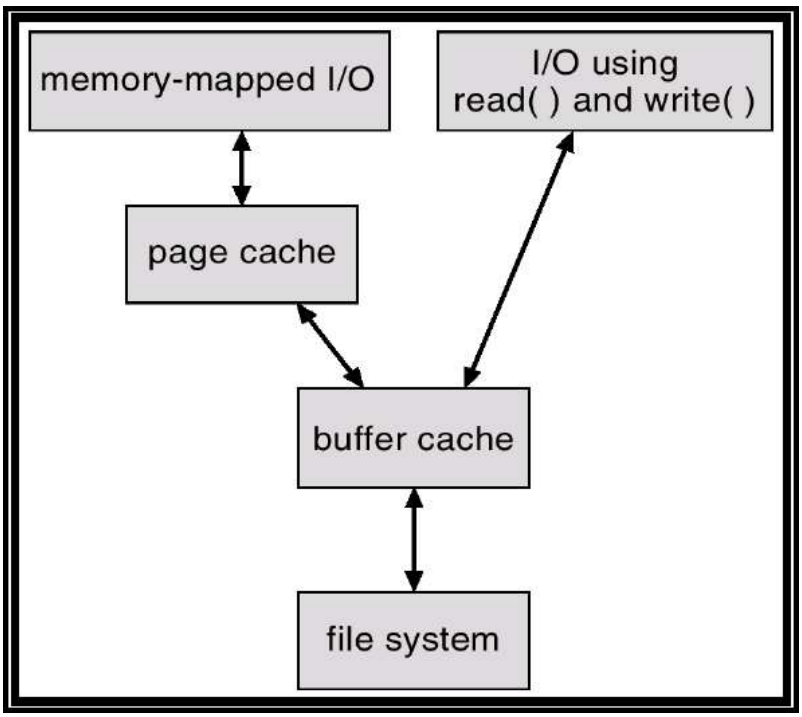


Fig 25 : Paged cache

Unified Buffer Cache

- A unified buffer cache uses the same page cache to cache both memory-mapped pages and ordinary file system I/O.

I/O Using a Unified Buffer Cache

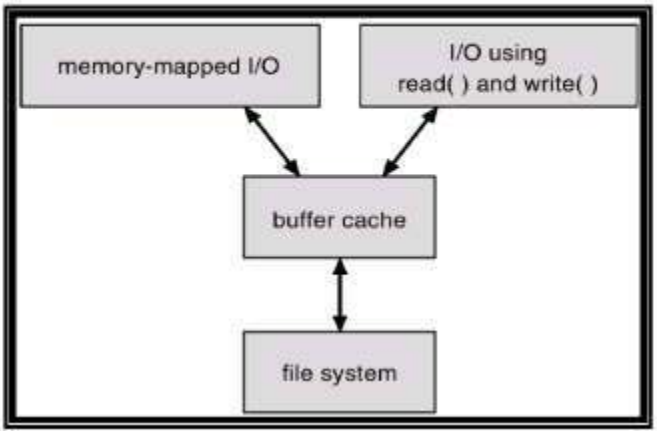


Fig 26 : Buffer cache

RECOVERY

- Consistency checking – compares data in directory structure with data blocks on disk, and tries to fix inconsistencies.
- Use system programs to back up data from disk to another storage device (floppy disk, magnetic tape).
- Recover lost file or disk by restoring data from backup.

LOG STRUCTURED FILE SYSTEMS

- Log structured (or journaling) file systems record each update to the file system as a transaction.
- All transactions are written to a log. A transaction is considered committed once it is written to the log.
- The transactions in the log are asynchronously written to the file system. When the file system is modified, the transaction is removed from the log.
- If the file system crashes, all remaining transactions in the log must still be performed

THE SUN NETWORK FILE SYSTEM (NFS)

- An implementation and a specification of a software system for accessing remote filesacross LANs (or WANs).
- The implementation is part of the Solaris and SunOS operating systems running on Sun workstations using an unreliable datagram protocol (UDP/IP protocol and Ethernet.
- NFS is designed to operate in a heterogeneous environment of different machines operating systems, and network architectures; the NFS specifications independent of these media.
- The NFS specification distinguishes between the services provided by a mount mechanism and the actual remote-file-access services.

Three Major Layers of NFS Architecture

- UNIX file-system interface (based on the open, read, write, and close calls, and file descriptors).
- Virtual File System (VFS) layer – distinguishes local files from remote ones, and local files are further distinguished according to their file-system types.
- The VFS activates file-system-specific operations to handle local requests according to their file-system types.
- Calls the NFS protocol procedures for remote requests.
- NFS service layer – bottom layer of the architecture; implements the NFS protocol

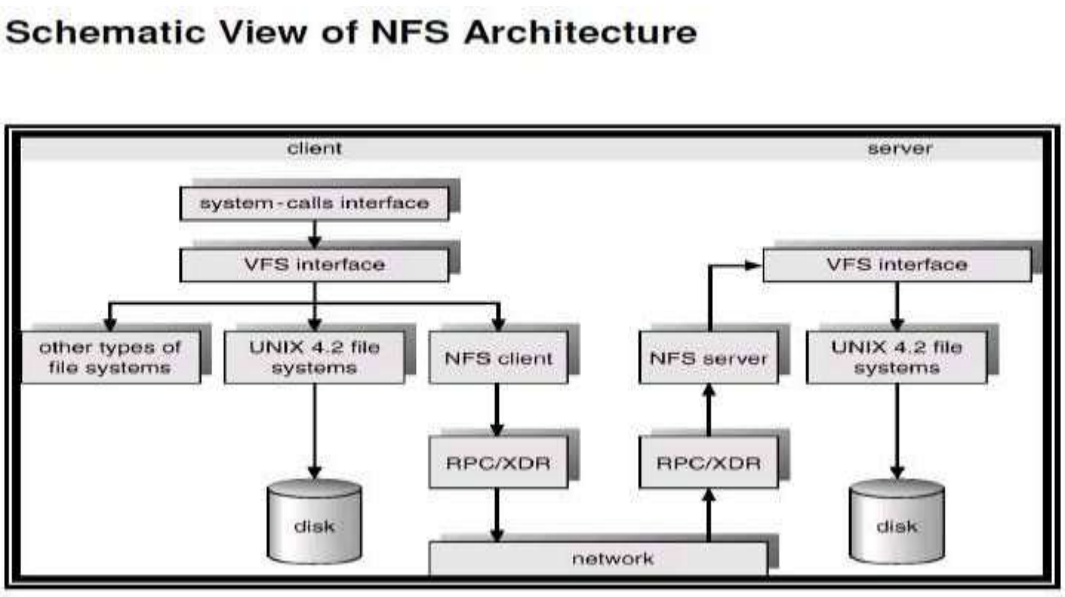


Fig 27 : NFS Architecture

NFS PROTOCOL

- Provides a set of remote procedure calls for remote file operations. The procedures support the following operations:
    - searching for a file within a directory
    - reading a set of directory entries
    - manipulating links and directories
    - accessing file attributes
    - reading and writing files
- NFS servers are stateless; each request has to provide a full set of arguments.
- Modified data must be committed to the server's disk before results are returned to the client (lose advantages of caching). The NFS protocol does not provide concurrency control mechanisms

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING
Common to : CSE , IT

UNIT V

UNIT 5 I/O SYSTEM,LINUX & SHELL PROGRAMMING 10 Hrs.

Mass Storage Structure - Disk Structure- Disk Scheduling - Disk Management - Swap Space Management - RAID Structure - Shell Operation Commends - Linux File Structure - File Management Operation - Internet Service - Telnet - FTP - Filters & Regular Expressions - Shell Programming - Variable, Arithmetic Operations, Control Structures, Handling Date, Time & System Information.