3/02/2021
Wednesday

• 01. Operating System Overview •

* Operating system provide interaction between user and computer.

Booting sequence - starting a computer or computer embedded device.

STEPS : 1) Switching on power supply
2) Loading of OS into computer main memory
(BIOS will load OS)
3) keeping all application in a state of readiness in case needed by user.

BIOS = - Basic Input Output System (ROM (present in ROM))
- always on
- Firmware (hardware + Software)

• Cold booting - ① Switching on power
- ② loading BIOS

• Warm Booting - when system is already running and need to restart

warm booting is faster (bcz OS is ~~of~~ already loaded)

* Objectives of operating system :
- To make computer system more convenient.
- To hide hardware details from users.
- provide interface betn user and computer.

Scanned with CamScanner

* Functions of operating system :

Security
Control on system performance
job accounting
Error detecting
Coordination betn user & Computer
Memory manager
processor manager
Device Manager
file manager.

* Evolution of OS

1. Serial processing -

- Computer start in using from 1940-1950 era.
- programmer / user directly intracted with computer hardware.
- That machine are called as Bare machine (as they don't have OS)
- Every computer is programmed in its machine language.
- Uses punch card, paper tape and language translator

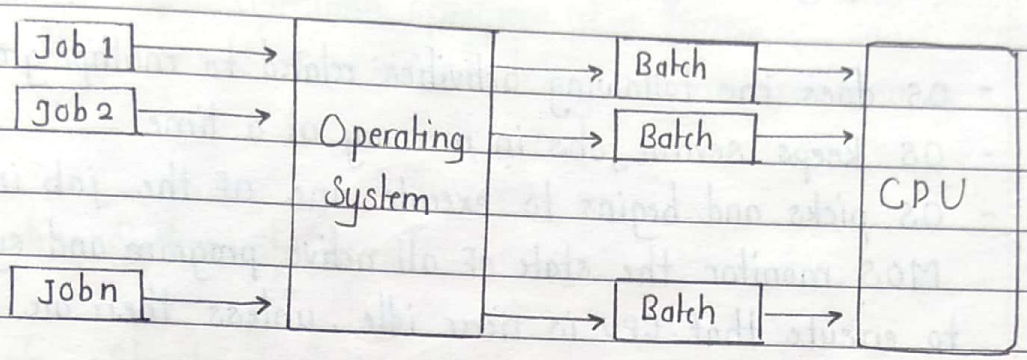
* Drawbacks of this system

1. Scheduling
2. Setup timing

Scanned with CamScanner

2. Batch processing -

it is a technique in which an OS collects data & program together before processing starts.



Operating System does following activities related to batch processing -

1. OS define job which has program, commands, data in single unit.
2. it keeps n number of jobs in memory and execute them.
3. jobs are processed in the order of submission i.e First come First served.
4. when job completes its execution, its memory is released and the output for job get copied into an output spool for later printing / processing.

Advantages -

- takes much work of the operator to the computer.
- Increased performance as a new job get started as soon previous get finished.

Disadvantages -

- Difficult to debug program
- job could enter an infinite loop
- due to lack of protection, one job can affect pending jobs.

Scanned with CamScanner

3. Multiprogramming OS -

sharing the processor, when two or more programs reside in memory at the time called multiprogramming.

Multiprogramming increase utilization of CPU.

- OS does the following activities related to multiprogramming:
- OS keeps several jobs in memory at a time
- OS picks and begins to execute one of the job in memory.
- MOS monitor the state of all active program and system resources to ensure that CPU is never idle, unless there are no jobs to process

Advantage -

High and efficient CPU utilization

Disadvantage -

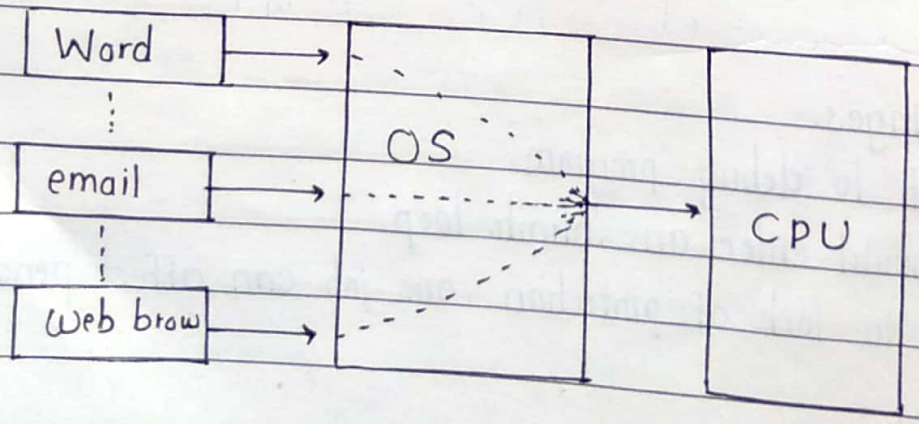
CPU scheduling is required

memory management is required to accommodate many jobs.

4. Multitasking or Time sharing OS

when multiple jobs are executed by CPU simultaneously by switching between them.

Switches occurs so frequently that user may interact with each program while it is running.



Scanned with CamScanner

OS does following activities related to multitasking :

- User give instructions to OS or to program directly and receive an immediate response.

- handle multiple operation / program at a time.
- Uses the concept of CPU scheduling and multiprogramming.

5/02/2021
Friday

Operating System Structures

1. Simple Structure -
   - small, simple and limited systems.

   eg - MS-DOS - written to provide more functionality in less space.
   - it does not break systems into subsystems.

2. Monolithic structures -
- File management, memory management, device management and process management are directly controlled within kernel (program)
- eg Linux
- Device drivers are loaded into the running kernel and become part of kernel.

Features -
- having simple structure. (all component that are needed are embended into kernel)

- Work for smaller task as it handle ~~mi~~limited resources.

- Components can communicate with each other.

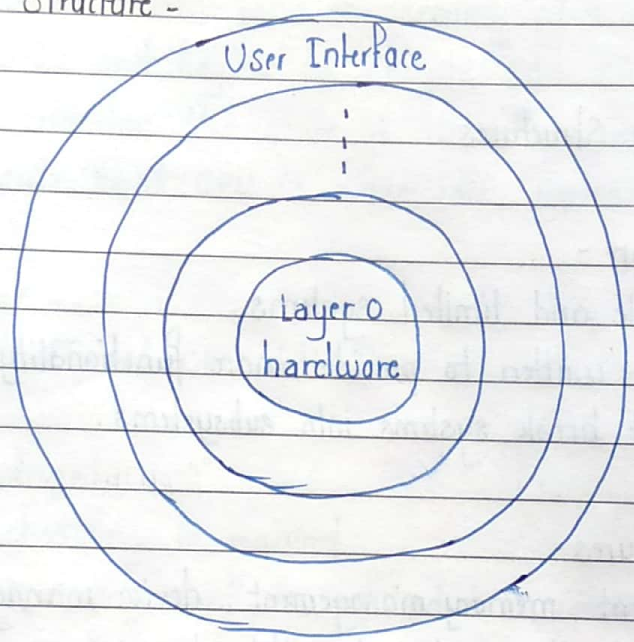
- Fast operating system.

Scanned with CamScanner

Limitations -

- Code written in this OS is difficult to port.
- it generate more errors and bugs bcz user use same address location as the kernel.
- adding and removing functions are difficult.

3. Layered Structure -



- each layer developed and debugged independently with asumption that all lower layers have already debugged.

Advantage - simple construction

Disadvantage - less efficient
difficult to define various layers

4. Microkernel Structure -

- basic idea → remove all non-essential services from the kernel and make kernel efficient and small as possible.

- provide basic process, memory management, message passing between other service.

- More secure
- easily expansion

eg - QNX.

Basis for Comparison	Microkernel	Monolithic kernel
Basic	user service and kernel service are kept in seperate address.	—"— kept in same address
size	smaller in size	larger in size
Execution	slow execution	Fast execution
Extendible	easily extendable	hard to extend
Security	if service crashes it doesn't effect on working	—"— does effect
Code	more code required	less code is required
eg	eg - QNX, Symbian	Linux etc.

Scanned with CamScanner

6/02/2021
Saturday

System calls - provide an interface to services made available by OS.
- Calls written in C & C++

* Types -

1) process Control -
load, execute, end, abort, termination process, get/set process attributes, wait for time, wait event, signal event, allocate, free memory.

2) File management -
Create file, delete file, open, close, read, write, reposition, get/set file attribution.

3) Device management -
request device, release device, read, write, reposition, get/set device attributes, logically attach or detach device.

4) Information maintenance -
get/set time or date, get/set system data.

5) Communication -
Create, delete communication connection, send, receive messages transfer status information, attach or detach remote devices.

Scanned with CamScanner

Linux kernel and shell -

- both kernel and shell are used to perform any operation on system.

- When user gives his command for performing any operation, then Request will goes to shell parts / interpreter which translate human program into machine language.

- And then request will transferred to kernel.

- kernel → heart of operating system. and every operation is performed by using kernel.

- Some operation like :

1) it checks whether process is running or waiting for the user request i.e Control process state.

2) allocate memory for running process

3) process scheduling and swap out function.

4) Virtual memory management

5) File management and permissions.

* process scheduling is the activity of the process manager that handles the removal of running process from CPU and selection of another process on basis of a particular strategy.

Scanned with CamScanner

10/02/2020
wednesday

02. Process and scheduling of process

- ~~program~~ Process - a program in execution load in file memory

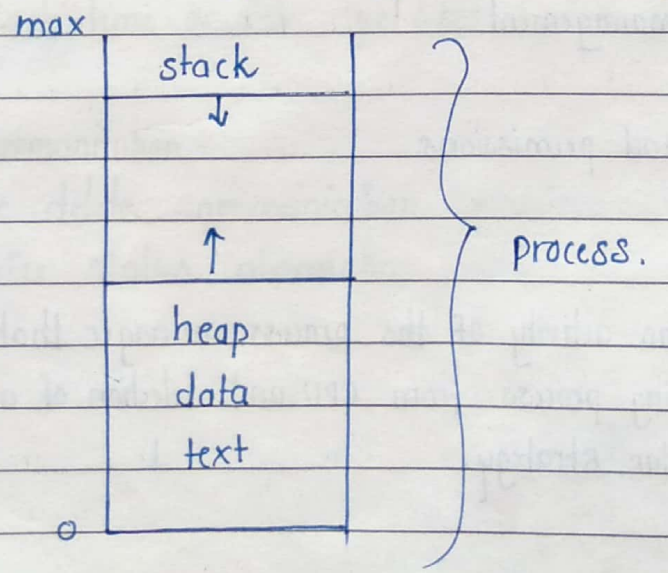
program is passive entity stored in executable file.
process is active entity.

program becomes process when executable File loaded into memory

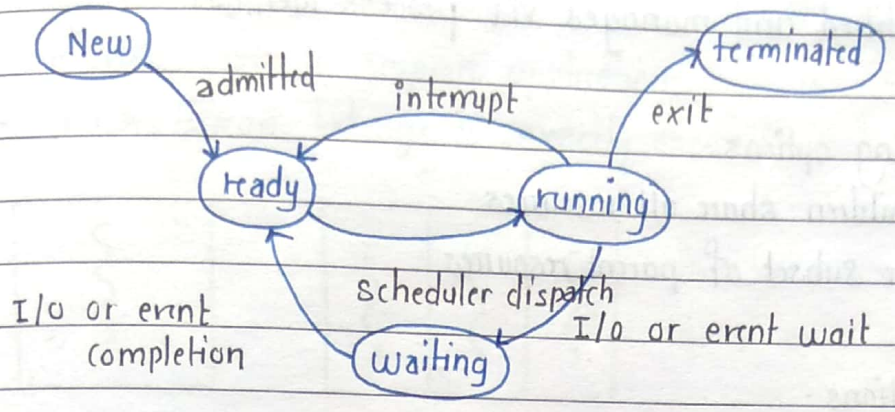
Execution of program started via GUI mouse click, command line entry of its name

* parts of process -

1. program code / text section.
2. current activity including program counter
3. stack - contain temporary data.
4. parameters, addresses, variables.
5. Data section containing global variable.
6. Heap containing memory allocate during runtime.



process state -



New = process is created
running - executed
waiting - waiting for some event to occur.
ready - waiting to be assigned to a processor
terminated - finished execution.

* Process Control Block (PCB)

Process state	→ running + waiting
Process number	
program counter	→ location of instructions to next execute.
registers	→ contains all process-centric registers.
memory limits	
list of open files	
. . .	

* process operations = it provides a mechanism for process creation and termination.

process creation =
- parent process creates children process, forming a tree of process.
- process identified and manages via process identifier

Resource sharing options -
- parent and children share all resources
- childrens are subset of parent resources

Execution options -
- parent and children execute concurrently
- parent waits until children terminate.

when process creates a new process, two possibilities for execution exist:
parent continues to execute concurrently with its children.
parent waits until some or all children have terminated.

address-space possibilities -
- child process is duplicate of the parent process.
- child has new program loaded into it.

Process Termination.

- exit()
- output data from child to parent
- resources are deallocated by OS.

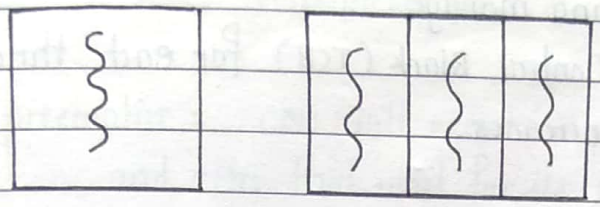
- parent may terminate execution of child process by abort()

Reason for that = child has exceeded allocated resources.
= Task assigned to child is no longer required
= parent is exiting and OS doesn't allow child to continue if if its parent terminate.

Scanned with CamScanner

• Threads •

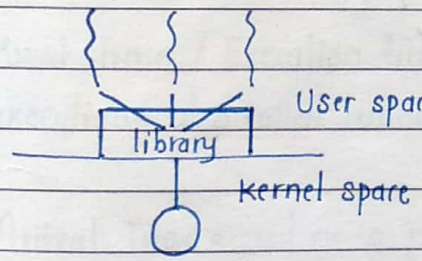
- light weight process
- It is flow of execution through the process code with its own program counter, registers and stack.
- provide way to improve application.
- each thread belongs to exactly one process.



Single-thread multi-thread.

* Types of threads -

① User level Threads (ULT)
- implement in user level libraries.



Advantages =
- Easy to represent and manage
- can run on any OS.
- Fast and efficient
- doesn't require modification to OS.

Disadvantage
- lack of coordination between threads & OS kernel.

② Kernel-level thread = (KLT)
handle by OS directly and management is done by kernel.

Scanned with CamScanner

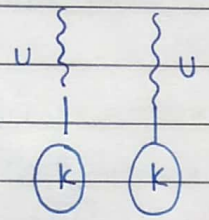
Advantages of KLT =
- kernel can simultaneously schedule multiple thread.
- if one thread block kernel can schedule another thread.

Disadvantages =
- generally slower to create and manage.
- kernel requires Thread Control Block (TCB) for each thread in pool, hence complexity increases.

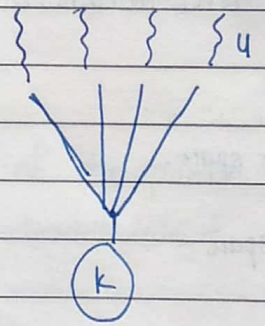
- Multi-Threading models =
  - Combination of ULT & KLT - eg = Solaris

Types =

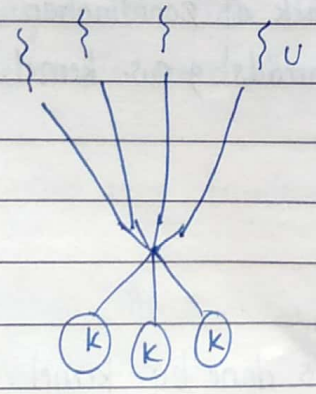
① One to one



② Many-to-one



③ many-to-many / two-level



* CPU Scheduling

process of determining which process ~~will~~ will own CPU for execution while another process is on hold.

- Types = pre-emptive
  Non-pre-emptive

preemptive = CPU will execute a process but for a limited period of time and after that wait for its next turn.

non-preemptive = if some resource is allocated to a process then that resource will not be taken back until completion of process. and other are waiting in queue.

Imp CPU scheduling Terminologies -

- Brust time / Execution time = time required by a process to complete execution. also called running time.
- Arrival Time = when a process enters in ready state.
- Jobs = type of program.
- process = program in execution
- Waiting time = time spend in waiting state.
- Turnaround time = total time from arrival to execution complete.

  Turnaround time = waiting time + Burst time.

Scanned with CamScanner

CPU Scheduling Criteria

- Maximize =

1) CPU utilization = CPU remains as busy as possible.
2) Throughput = The no. of process that finish their execution per unit time.

- Minimize =

1) Waiting time = time for which process needs to wait in ready queue.
2) Response time = time in which the request was submitted until the first response is produced.

3) Turnaround time = amount of time to execute specific process. It is the calculation of total time spent waiting to get into memory, waiting in the queue and executing on the cpu. It is time period between time of process submission to the completion time.

- Schedulers

-special system software which handle process scheduling.
- main task is to select the jobs to be submitted into the system and to decide which process to run.

* Types of scheduler

1) Long-Term Scheduler = / Job scheduler
It determines which program are admitted to the system for processing.

Scanned with CamScanner

- It selects processes from the queue and loads them into memory for CPU scheduling & execution.

2) Short-Term scheduler = / CPU scheduler / dispatchers
- change in ready state to running state.
- It selects a process among the process that are ready to execute and allocates CPU to one of them.

3) Medium Term scheduler =
- part of swapping
- removes process from memory and reduces degree of multiprogramming.

Swapping means = remove process from memory and make space for another processes. the suspended process is moved into the secondary storage device.

8+9 • CPU Scheduling Algorithms. (5)

1] First Come First Serve (FCFS)
- Jobs are executed on FCFS basis. (AT)
- It is non-preemptive, ~~preemptive~~ scheduling algorithm.
- easy to understand and implement.
- based of FIFO queue.
- poor in performance as avg waiting time is high.
- It assigns CPU to the process which comes first in ready state

eg =

Process-ID	Arrival time (AT)	Burst Time (BT)
P0	0	2
P1	1	6
P2	2	4
P3	3	9
P4	4	12

Scanned with CamScanner

* Gantt chart

P_0	P_1	P_2	P_3	P_4

0 2 8 12 21 33

* Waiting time = time in gantt time - arrival time

$P_0 = 0 - 0 = 0$

$P_1 = 2 - 1 = 1$

$P_2 = 8 - 2 = 6$

$P_3 = 12 - 3 = 9$

$P_4 = 21 - 4 = 17$

* Turn-around time = waiting time + Burst time

$P_0 = 0 + 2 = 2$

$P_1 = 1 + 6 = 7$

$P_2 = 6 + 9 = 15$

$P_3 = 9 + 4 = 13$

$P_4 = 17 + 12 = 29$

P_ID	AT	BT	WT	TAT
P_0	0	2	0	2
P_1	1	6	2	8
P_2	2	4	8	12
P_3	3	9	12	21
P_4	4	12	21	33
		Total =		

avg waiting time =

avg TAT =

* Convoy Effect in FCFS

meaning = if the burst time of the first job is highest among all. due to which processes / job of lower burst time may may get block

Advantages = simple and easy
Disadvantages = process will run to the completion (non-preemptive)
- problem of starvation may occur.
- poor in performance, since the avg waiting time is higher as compare to other scheduling algorithms.

2] Shortest Job First (SJF) scheduling

- Non-preemptive SJF
- preemptive SJF

- Non-pre-emptive SJF =
schedules according to burst time. (BT)
- lowest burst time among the list of available processes in the ready queue is going to be scheduled next.

eg =

PID	AT	BT
1	1	7
2	3	3
3	6	2
4	7	10
5	9	8

Scanned with CamScanner

Gantt chart -

	P_1	P_3	P_2	P_5	P_4

0 1 8 10 13 21 31

PID	AT	BT	WT	TAT
1	1	7	0	7
2	3	3	7	10
3	6	2	2	4
4	7	10	14	24
5	9	8	4	12
			27	57

avg WT = 27/5 = 5.4
avg TAT = 57/5 = 11.4

Advantages = maximum throughput
minimum avg WT & avg TAT

Disadvantages =
- may suffer with the problem of starvation.
- non implementable bcz the exact BT for process can't be known in advance.

3] Shortest Remaining Time Next (SRTN) = BT
- promptive version of SJF scheduling.
- execution stopped after certain amout of time.
- At the arrival of every process, the short term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.

Scanned with CamScanner

eg =

P-ID	AT	BT	Remaining BT
1	0	8	7
2	1	4	~~3~~ 2
3	2	2	~~1~~
4	3	1	~~1~~
5	4	3	
6	5	2	~~1~~

$P_1 = 8-1 = 7$

$P_2 = 3$

$\cancel{P_3} = 1$

$P_4 = 1$

P_5

$P_6 = 1$

Gantt chart =

P_1	P_2	P_3	P_3	P_4	P_6	P_6	P_2	P_5	P_1

0 1 2 3 4 5 6 7 10 13 20

P-ID	AT	BT	WT	TAT
P_1	0	8	20-8=12	20-0=20
P_2	1	4	9-4=5	10-1=9
P_3	2	2	2-2=0	4-2=2
P_4	3	1	2-1=1	5-3=2
P_5	4	3	9-3=6	13-4=9
P_6	5	2	2-2=0	7-5=2
		Total	24	44
			avg = 4	avg = 6.33

TAT = Completion time - Arrival time

WT = TAT - BT

Advantage = less waiting time
quite good response for short process.

Disadvantage = starvation is possible for long process.

Scanned with CamScanner

4] Priority Scheduling =

- method of scheduling process based on priority.
- higher priority carried out first and equal priority carried out on FCFS basis.

- preemptive priority scheduling =

If the new process arrived at ready queue has higher priority than currently running then CPU is preempted i.e current process stopped and incoming new process with higher priority gets CPU for execution.

- Non-preemptive priority scheduling =

then, after execution of current process it will be processed.

eg -

PID	BT	Priority
P_1	21	2
P_2	3	1 H
P_3	6	4 L
P_4	2	3

Gantt chart =

P_2	P_1	P_4	P_3

0 3 24 26 30

P.ID	P	BT	WT	TAT
P_1	2	21	3	24
P_2	1	3	0	3
P_3	4	6	26	32
P_4	3	2	24	26
			53	85

avg WT = 13.25

avg TAT = 21.25

Scanned with CamScanner

Advantages = easy to use
- higher priority doesn't need to wait for long which saves time
- suitable for applications with fluctuating time and resource requirment.

Disadvantage =
If higher priority will take lots of CPU timing then lower priority may starve.

5] Round Robin Scheduling

- promptive version of FCFS.
- focuses on time sharing
- process executed in cyclic way
- time slice is defined in system called quantum.
- Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of process is completed during that time the process will terminate else process will go back to ready queue and waits for next turn to complete execution.

eg-

P.ID	AT	BT	
1	0	5	✓
2	1	6	~~2~~
3	2	3	X
4	3	1	X
5	4	5	1
6	6	4	X

given time quantum = 4 unit

Gantt chart =

P_1	P_2	P_3	P_4	P_5	P_6	P_1	P_2	P_5

0 4 8 11 12 16 20 21 23 24

P-ID	AT	BT	WT	TAT
P_1	0	5	16	21
P_2	1	6	16	22
P_3	2	3	6	9
P_4	3	1	8	9
P_5	4	5	15	20
P_6	6	4	10	14
		avg	11.83	15.83

Adv = can be implemented in system bcz it is not depending on BT.
- doesn't suffer from starvation
- all the process gate fare allocation of CPU.

Dis-advantage -
- the higher time quantum – higher will be response timing
- lower the time quantum, higher the context switching overhead in the system.
- Deciding a perfect time quantum is difficult.

eg =

P-ID	BT	Priority
P_1	2	2
P_2	1	1
P_3	8	4
P_4	4	5
P_5	5	3

Calculate avg waiting time and TAT for FCFS, SJF (Non-preemptive), priority & RR (Quantum=2).

Scanned with CamScanner

① FCFS =

Gantt chart

P_1	P_2	P_3	P_4	P_5

0 2 3 11 15 20

P-ID	CT	BT	WT	TAT
P_1	2	2	0	2
P_2	3	1	2	3
P_3	11	8	3	11
P_4	15	4	11	15
P_5	20	5	15	20
			31	51

avg WT = 6.2

avg TAT = 10.2

② SJF (BT)

Gantt chart

P_2	P_1	P_4	P_5	P_3

0 1 3 7 12 20

P-ID	CT	BT	WT	TAT
P_1	3	2	1	3
P_2	~~3~~ 1	1	0	1
P_3	20	8	12	20
P_4	7	4	3	7
P_5	12	5	7	12
			23	43

avg WT = 4.6

avg TAT = 8.6

③ Priority

Gantt chart

P_2	P_1	P_5	P_3	P_4

0 1 3 8 16 20

Scanned with CamScanner

PID	BT	CT	WT	TAT
P_1	2	3	1	3
P_2	1	1	0	1
P_3	8	16	8	16
P_4	4	20	16	20
P_5	5	8	3	8
			28	48

avg WT = 5.6

avg TAT = 9.6

(4) R.R -

Quantum = 2

Gantt chart

P_1	P_2	P_3	P_4	P_5	P_3	P_4	P_5	P_3	P_5	P_3

0 2 3 5 7 9 11 13 15 16 18 20

P-ID	BT	CT	WT	TAT
P_1	2	2	0	2
P_2	1	3	2	3
P_3	8	20	12	20
P_4	4	13	9	13
P_5	5	18	13	18
			36	56

avg WT = 7.2

avg TAT = 11.2

Scanned with CamScanner

15

31/03/2021 ~~Thursday~~ Wednesday

Process synchronization and Deadlock

* principle of concurrency -
Execution of ~~concurrenc~~ multiple process at a time.

Types of Concurrency -

① Independent process -
- The process that neither affect working of other process nor affected by other processes.
- doesn't share any information or data with other process.

② Cooperating process -
- process that affect working of other process and also affected by other process
- Communicate and share data and information with other process.

- Concurrent access to shared data may result in data inconsistency
- maintaining data consistency requirs mechanisms to ensure orderly execution of cooperating processes.

* Race Condition -
Multiple processes read & write data items
- Mutual exclusion = way to prohibit more than one process from accessing share data at a time.

- Critical - Section problem - is to design a protocol that the processes can use to cooperate.

```
do {
    entry section
    critical section
    exit section
    Reminder section
} while (TRUE);
```

- Software-based solution to the critical-section = peterson's solution.

Scanned with CamScanner

16

5/03/2021
Friday

• Mutual exclusion

- To avoid race around condition we need mutual exclusion.
- It makes sure that if one process is using a share data / file the other process will be executed from doing the same thing without disturbing that other process

Hardware Support - TSL (Test and Set Lock)

- Synchronization mechanism.
- provide synchronization among concurrently running process.

TSL Instructions -

- That returns the old value of a memory location & sets the memory location value to 1. (single operation)
- only one TSL instruction can run at a time.

Implementation -

While (TSL);	Entry
critical Section	
Lock = 0	Exit

Lock = 0 → Critical section is vaccant i.e no process inside it
Lock = 1 → Critical Section occupied and process is present in it.

16 Working :-

Characteristics =
- ensures mutual exclusion.
- doesn't guarantee bounded waiting and may cause starvation.

* Operating System support - Semaphore

• Semaphore = Integer variables that are used to solve critical section problem by using following two operations.

1] Wait (s) :
decrements the value of its argument 's', if it is +ve &
if it is -ve/0 then no operation is performed.

2] Signal (s) :
Increments value of argument 's'.

• Types of Semaphore =

① Counting Semaphore =
- It count no of resources available.
- Integer value having unrestricted domain.
- Count ~~ate~~ automatically increment & decrement after addition & deletion of resources.

② Binary Semaphore =
- are counting semaphores but value is restricted to 0 & 1
- when semaphore = 1 ⟶ wait() perform
- — " — = 0 ⟶ signal() — " —

* advantages & Disadvantages

06/03/2021
Saturday

17

- Classical problem of Synchronization =

1] Bounded Buffer (producer- Consumer) problem -

- In this problem finite buffer pool is used to exchange msgs between producer and consumer.

- Sol^n = creating two counting semaphores "Full" & "Empty" to keep track of current no. of full & empty buffers respectively.

2] Reader Writer problem -

3] Dining philosophers problem -

- problem → Five philosophers sitting around a circular table and their job is to think and eat alternatively.

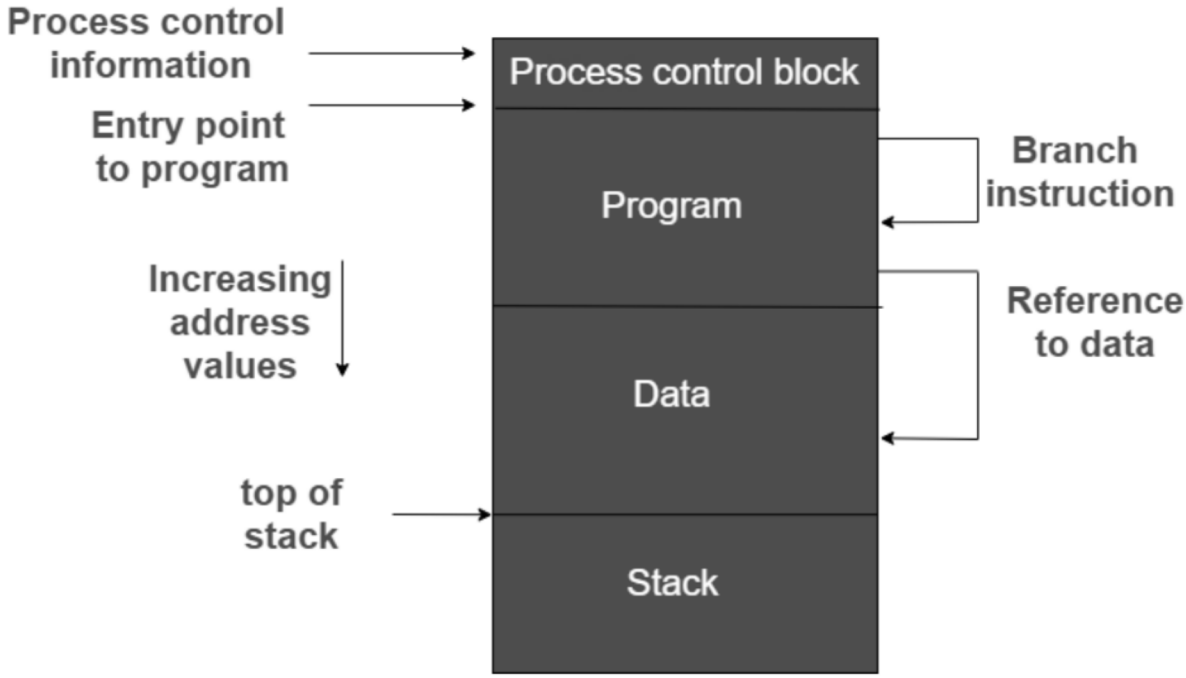
Scanned with CamScanner

4. Memory Management

4.1.1 Memory Management Requirements

Memory management keeps track of the status of each memory location, whether it is allocated or free. It allocates the memory dynamically to the programs at their request and frees it for reuse when it is no longer needed. Memory management should satisfy some requirements. These Requirements of memory management are:

1. **Relocation**
   a. The available memory is generally shared among a number of processes in a multiprogramming system, so it is not possible to know in advance which other programs will be resident in main memory at the time of execution of his program.
   b. Swapping the active processes in and out of the main memory enables the operating system to have a larger pool of ready-to-execute process. When a program gets swapped out to a disk memory, then it is not always possible to occupy the previous memory location when it is swapped back into main memory. Because the location may still be occupied by another process. The process must be relocated to a different area of memory.
   c. There is a possibility that program may be moved in main memory due to swapping.
   d. The figure depicts an image of a process. The process is occupying a continuous region of main memory. The operating system keeps track of many things including the location of process control information, the execution stack, and the code entry. Within a program, there are memory references in various instructions and these are called logical addresses.



e. After loading of the program into main memory, the processor and the operating system must be able to translate logical addresses into physical addresses. Branch instructions contain the address of the next instruction to be executed. Data reference instructions contain the address of byte or word of data referenced. Data reference instructions contain the address of byte or word of data referenced.

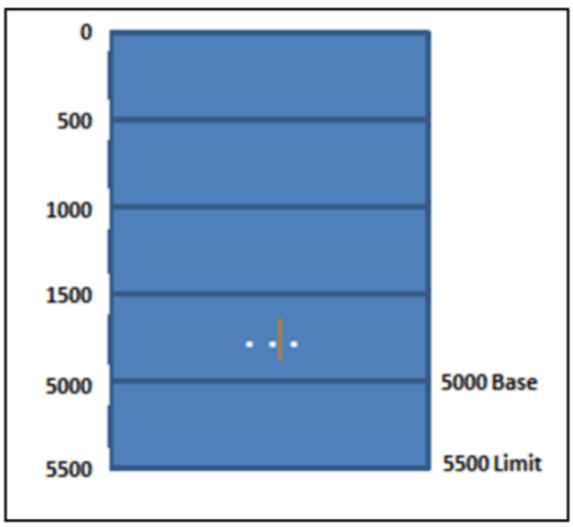
2. Protection ,logical and Physical Address Space

a. An address generated by CPU is called as logical address or virtual address and an address generated by Memory management unit is called as physical address.

b. If size of program is 1000K and loaded in memory from address 5000 to 6000, then physical address is from 5k to 6k and logical address is 0 to 999.

c. The logical address space is collection of all logical addresses generated by a program and the physical address space is collection of physical addresses corresponding to these logical addresses.

d. If binding of instructions and data to memory addresses are at compile time or load time, then logical and physical addresses are same.
e. If same binding occurs at run time then logical and physical addresses are different.
f. Relocation register (base register) contains address 5000. Physical address (5200) can be calculated as follows.
g. Consider the logical address 200 of the same program.
h. Physical address(5200) = logical address(offset)(200) + contents of base register(5000)
i. Mapping from virtual to physical address is done by MMU at run time.
j. User programs deals with logical addresses. Memory mapping hardware converts logical address into physical address.
k. Memory protection is done by base and limit registers.
l. The base register holds the smallest legal physical memory address.
m. The limit register gives the size of the range of physical address. If base address is 5000 and limit register is 500 , then program can access all addresses from 5000 to 5499. Memory protection with base and limit registers is depicted in the following figure.



3. Sharing

a. A protection mechanism must have to allow several processes to access the same portion of main memory.
b. Allowing each processes access to the same copy of the program rather than have their own separate copy has an advantage. For example, multiple processes may use the same system file and it is natural to load one copy of the file in main memory and let it shared by those processes.
c. It is the task of Memory management to allow controlled access to the shared areas of memory without compromising the protection. Mechanisms are used to support relocation supported sharing capabilities.

4. Logical organization

a. Main memory is organized as linear or it can be a one-dimensional address space which consists of a sequence of bytes or words.
b. Most of the programs can be organized into modules, some of those are read-only or execute only and some of those contain data that can be modified.
c. To effectively deal with a user program, the operating system and computer hardware must support a basic module to provide the required protection and sharing. It has the following advantages:
    - Modules are written and compiled independently and all the references from one module to another module are resolved by the system at run time.
    - Different modules are provided with different degrees of protection.
    - There are mechanisms by which modules can be shared among processes.
    - Sharing can be provided on a module level that allows the user to specify the sharing that is desired.

5. Physical organization

a. Computer memory has main memory and secondary memory. Main memory is relatively very fast and costly as compared to the secondary memory.
b. Main memory is volatile. Secondary memory is provided for storage of data on a long-term basis while the main memory holds currently used programs.
c. The major system concern between main memory and secondary memory is the flow of information and it is impractical for programmers to understand this for two reasons:
    - The programmer may engage in a practice known as overlaying when the main memory available for a program and its data may be insufficient. It allows different modules to be assigned to the same region of memory. One disadvantage is that it is time-consuming for the programmer.
    - In a multiprogramming environment, the programmer does not know how much space will be available at the time of coding and where that space will be located inside the memory.

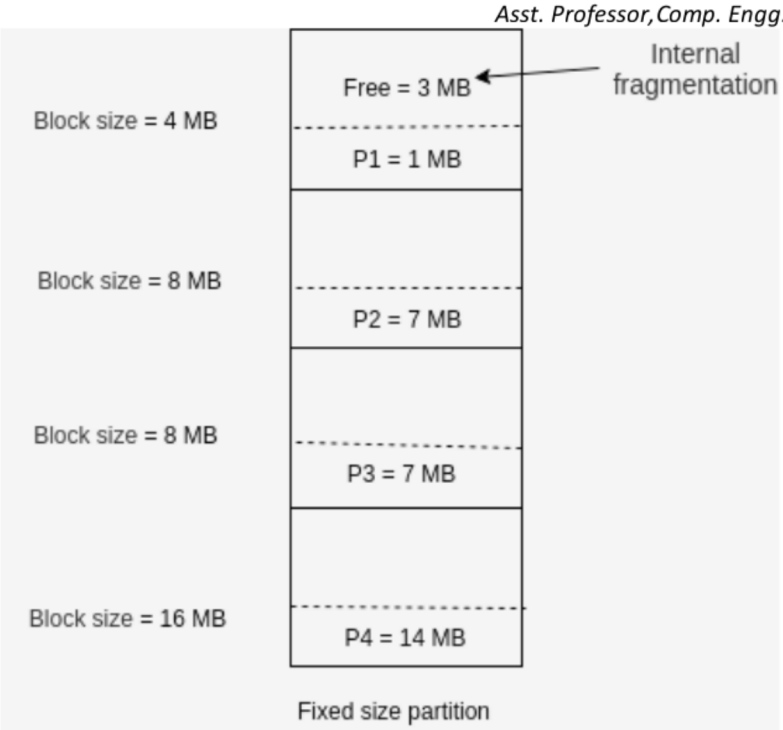
4.1.2 Memory Partitioning

a. Memory Management is responsible for allocation and management of computer's main memory.
b. Memory Management function keeps track of the status of each memory location, either allocated or free to ensure effective and efficient use of primary Memory.
c. There are two Memory Management Techniques:
    - Contiguous Memory Allocation
    - Non-Contiguous Memory Allocation

d. In Contiguous Technique, executing process must be loaded entirely in primary memory.
e. Contiguous Technique can be divided into:
   - Fixed (or static) partitioning
   - Variable (or dynamic) partitioning

Fixed (or static) partitioning

- Memory is divided into several fixed sized partitions.
- Each partition contains exactly one process. Thus degree of multiprogramming is bound by number of partitions.
- In this method, when a partition is free, a process is selected from the input queue and loaded into the free partition.
- When the process terminates, partition becomes available for another process.
- Operating system keeps a table indicating which part of memory is available and which are occupied.
- Figure below shows the concept of fixed partitioning.
- There are some advantages and disadvantages of variable partitioning over fixed partitioning as given below.
- First process is consuming 1MB out of 4MB in the main memory. Hence, Internal Fragmentation in first block is (4-1) = 3MB.
- Sum of Internal Fragmentation in every block = (4-1)+(8-7)+(8-7)+(16-14)= 3+1+1+2 = 7MB



Advantages of Fixed Partitioning –

1. Easy to implement:

Algorithms needed to implement Fixed Partitioning are easy to implement. It simply requires putting a process into certain partition without focussing on the emergence of Internal and External Fragmentation.

2. Little OS overhead:

Processing of Fixed Partitioning require lesser excess and indirect computational power.

Disadvantages of Fixed Partitioning –

1. Internal Fragmentation:

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

2. External Fragmentation:

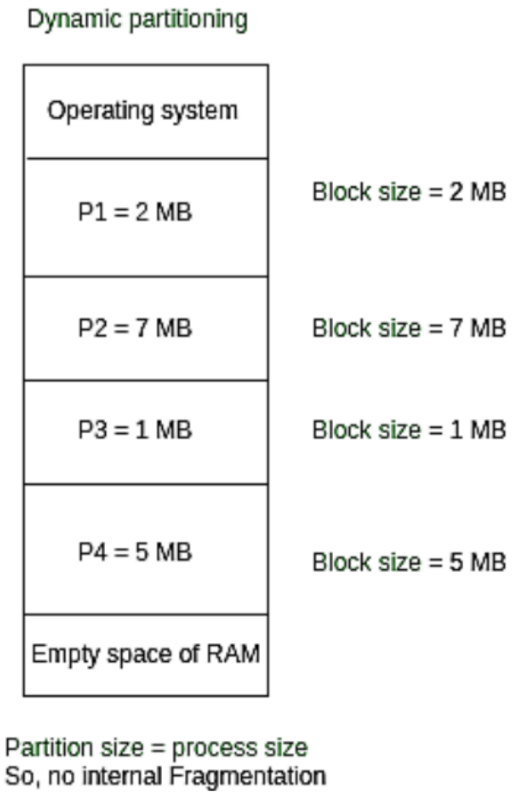
The total unused space (as stated above) of various partitions cannot be

used to load the processes even though there is space available but not in the contiguous form.

3. Limit process size:
Process of size greater than size of partition in Main Memory cannot be accommodated. Partition size cannot be varied according to the size of incoming process's size. Hence, process size of 32MB in given example above is invalid.
4. Limitation on Degree of Multiprogramming:
Partition in Main Memory are made before execution or during the system configure. Main Memory is divided into fixed number of partition. Suppose if there are n1 partitions in RAM and n2 are the number of processes, then n2<=n1 condition must be fulfilled. Number of processes greater than number of partitions in RAM is invalid in Fixed Partitioning.

Variable (or dynamic) partitioning

- Initially all memory is available for user processes and considered as one large block called as hole.
- When a process arrives and needs memory, a hole which is large enough is searched for this process. If such a hole is found, needed memory is allocated and remaining memory is available for another requests.



Advantages of Variable Partitioning –

1. No Internal Fragmentation:
   In variable Partitioning, space in main memory is allocated according to the need of process, hence there is no internal fragmentation. There will be no unused space left in the partition.
2. No restriction on Degree of Multiprogramming:
   More number of processes can be accommodated due to absence of internal fragmentation. A process can be loaded until the memory is available.
3. No Limitation on the size of the process:
   In Fixed partitioning, the process with the size greater than the size of the largest partition cannot be loaded in memory. In variable partitioning, the process size can't be restricted since the partition size is decided according to the process size.

Disadvantages of Variable Partitioning –

1. Difficult Implementation
   implementing variable Partitioning is difficult as compared to Fixed Partitioning as it involves allocation of memory during run-time rather than during system configure.
2. External Fragmentation:
   There will be external fragmentation in spite of absence of internal fragmentation.
   For example, suppose in above example- process P1(2MB) and process P3(1MB) completed their execution. Hence two spaces are left i.e. 2MB and 1MB. Let's suppose process P5 of size 3MB comes. The empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation. Process must be contiguously present in main memory to get executed. Hence it results in External Fragmentation.

4.1.3 Memory Allocation Strategies

If there are number of free holes in memory and process requests the free hole, then question is which hole is allocated to this process. Following are the strategies used to allocate the free hole to the requesting process.

1. **First-Fit –** Allocate first hole that is big enough. Searching can start either at the beginning of the set of holes or where the previous first-fit search ended. Searching can be stopped as soon as a free hole is found that is large enough.
2. **Best-Fit –** Allocate the smallest hole that is big enough. Entire list must be searched if the list is not kept ordered by size. This strategy produces the smallest leftover hole.
3. **Worst-Fit –** Allocate the largest hole. Entire list must be searched if the list is not kept ordered by size.

Example: Given memory partition of 150K, 500K, 200K, 300K and 550K (in Order) how would each of the first fit, best fit, and worst fit algorithm place the processes of 220K, 430K, 110K, 425K (in order). Which algorithm makes the most efficient use of memory?

Solution:

First Fit

1. 220K is put in 500K partition.
2. 430K is put in 550K partition.
3. 110K is put in 150K partition.
4. 425K must wait.

Best Fit

1. 220K is put in 300K partition.
2. 430K is put in 500K partition.
3. 110K is put in 150K partition.
4. 425K is put in 550K partition.

Worst Fit

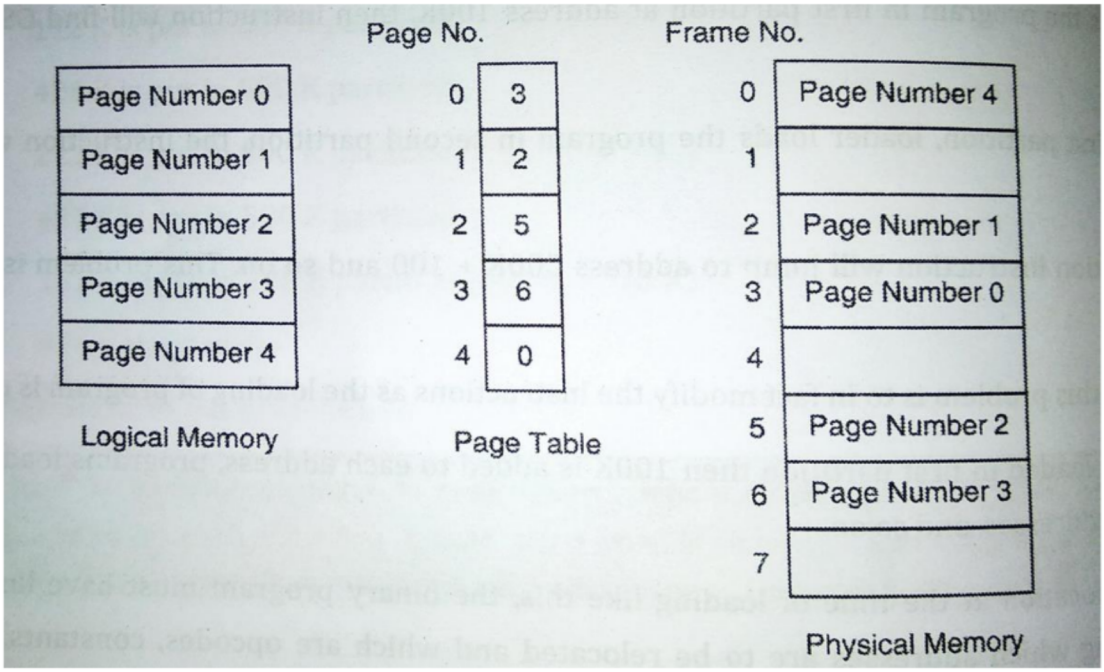
1. 220K is put in 550K partition.
2. 430K is put in 500K partition.
3. 110K is put in 330K partition.(550K-220K=330K)
4. 425K must wait.

4.1.4 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 noncontiguous.

Basic Operation

- The problem of external fragmentation is avoided using paging.
- Paging is implemented by integrating the paging hardware and operating system.
- In this technique, logical memory is divided into blocks of the same size called pages.
- The physical memory is divided into fixed sized blocks called frames (size is power of 2, between 512 bytes and 8192 bytes, also larger size possible in practice). The page size and frame size is equal.
- Initially all pages remains on secondary storage. When a process is to be executed, its pages are loaded into any available memory frames from secondary storage.
- Figure below shows the paging model of physical and logical memory.



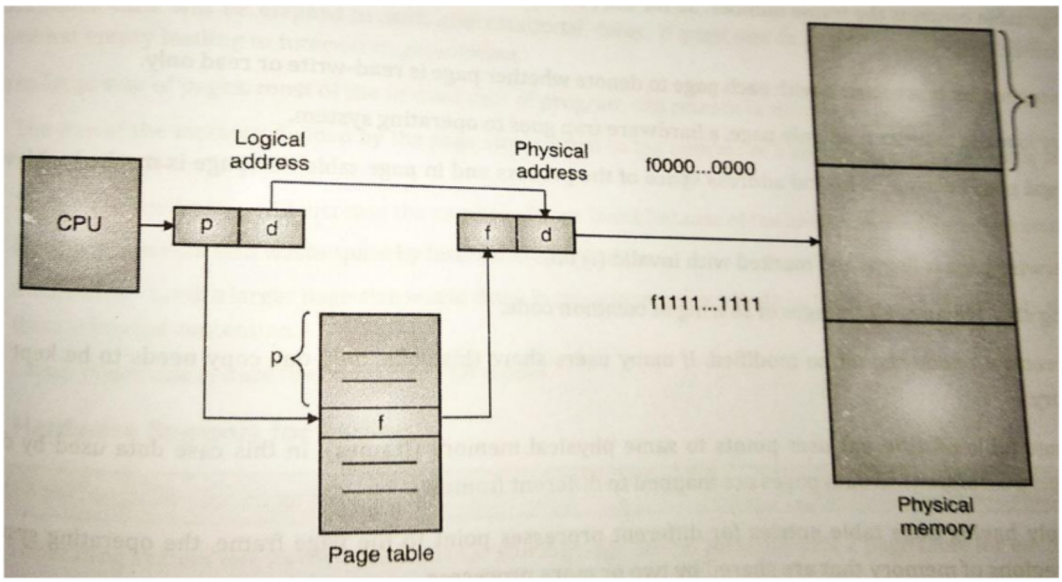
Following basic operations are done in paging.

1. CPU generates logical address and it is divided into two parts. A page number(p) and page offset(d)
2. The page number is used as an index into a page table.
3. The base address of each page in physical memory is maintained by page table.
4. The combination of base address with page offset defines the physical memory address that is sent to the memory unit.
5. The physical location of the data in memory is therefore at offset ***d*** in page frame ***f.***

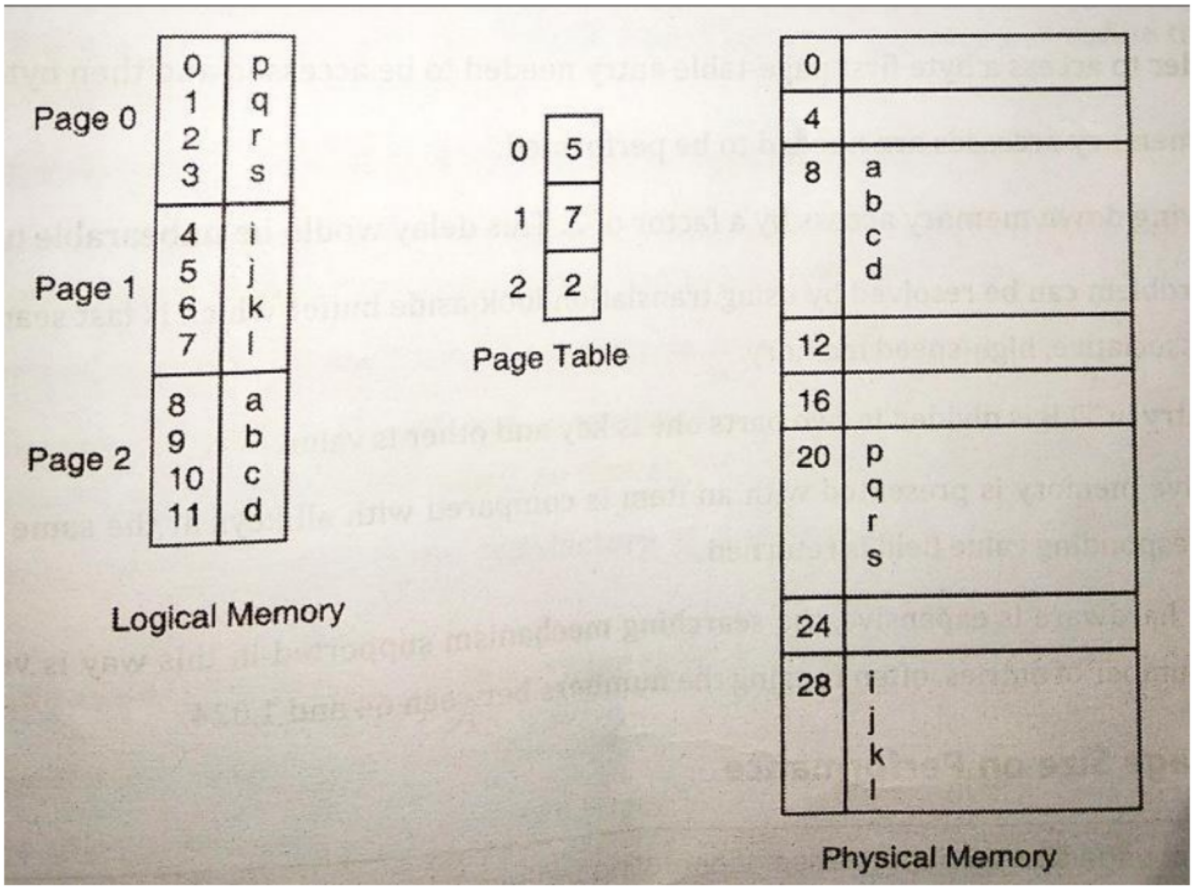
6. Because of the paging the user program sights the memory as one single contiguous space, it gives the illusion that memory contains only one program.

7. But the user program is spread throughout main memory(physical memory). The logical addresses are translated into physical addresses.

8. Figure below shows the operation of the paging hardware.



Example: Consider the following figure. Let page size is 4K and memory available is 32K.



- Page 0 is in frame number 5. Page 1 is in frame number 7 . Logical address 4 is in page 1 and offset is 0. Page 1 is mapped to to frame 7. So physical address is (7X4+0)=28K

- logical address 10 is in page 2. offset is 2. Page 2 is mapped to frame 2.
- So physical address is (2X4+2)=10K

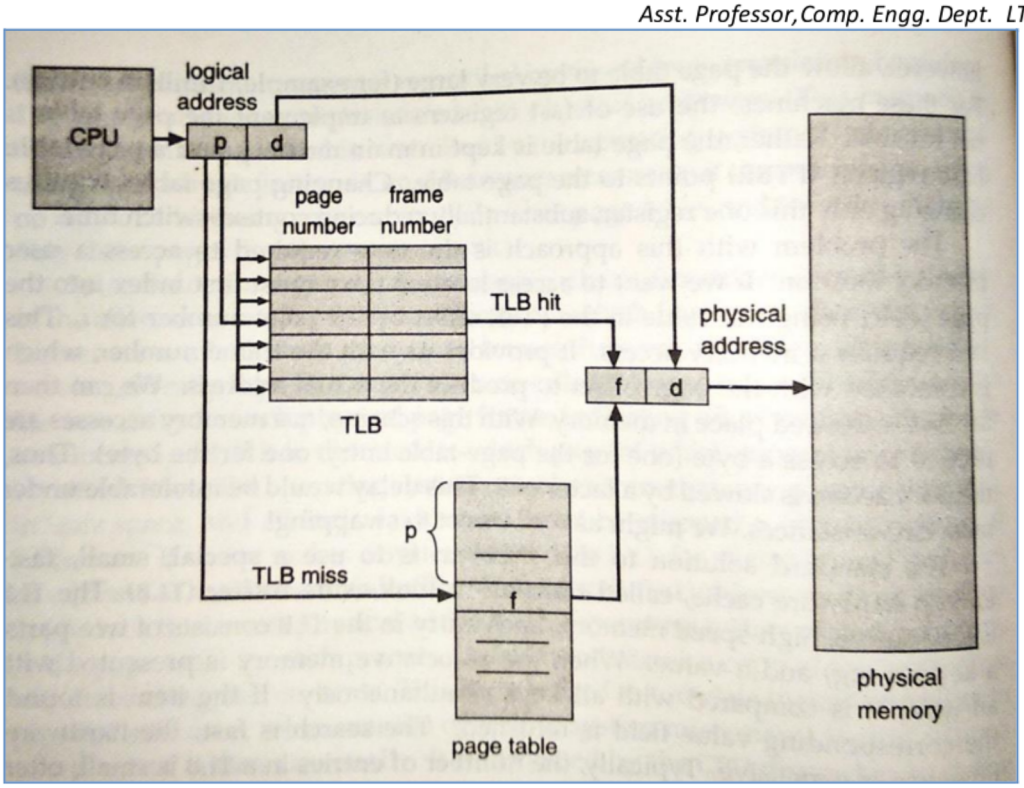
Hardware Support for Paging

1. Each operating system has its own method for storing page table. Most operating system allocates a page table for each process.
2. A pointer to the page table is stored with other register values in the process control block.
3. When the dispatcher is told to start a process, it must reload the user registers and define correct hardware page table values from the stored user page table.
4. The hardware implementation of the page table can be done in several ways.
5. In the simplest case, the page table is implemented as a set of dedicated registers. These registers should be built with very high speed logic to make the paging address translation efficient.
6. Every access to memory must go through the paging map so efficiency is a major consideration.
7. The CPU dispatcher reloads these registers just as it reloads the other registers. Instructions to load or modify the page table the registers are privileged so only OS can change the memory map.
8. The use of registers for the page table is reasonable if page table is reasonably small.
9. Most modern computers allow the page table to be very large.
10. For these machines the use of fast registers to implement the page table is not feasible. Rather, page table is kept in main memory and a page table base register (PTBR) points to the page table. Changing page tables requires only this one registers which reduces context switch time.
11. The problem with this approach is the time required to access a user memory location.
12. If we want to access location, we must first index into page table using the value in the PTBR offset by the page number.

13. With this scheme two memory accesses are required. One for page table entry and one for byte. Thus memory factor is slowed by factor 2.
14. Solution to this is use of Translation Look aside Buffer (TLB).
15. TLB contains a few of the page table entries. When a logical address is generated by the CPU, its page nuber is presented to the TLB.
16. If page number is found, its frame number is immediately found and used to access memory.
17. If page number is not found, a memory reference to the page table must be made and frame number is obtained. Then page number and frame number to the TLB so that they will be found quickly on the next reference.
18. If TLB is full OS must select one of the page replacement policies.

Translation Look aside Buffer (TLB)

1. Translation look aside buffer is a special small fast lookup hardware cache.
2. TLB is associative, high speed memory.
3. Each entry in TLB consists of two parts. A key and value.
4. When associative memory is presented with an item, it is compared with all keys simultaneously. If the item is found, the corresponding value field is returned. The search is fast. Hardware is expensive.
5. Paging hardware with TLB is shown in the following figure.



Example: For a paged system with TLB hit ratio is 0.9. Let RAM access time T is 100 ns. TLB access time t is 20ns. Find out

1. Effective memory access time with TLB
2. Effective memory access time without TLB
3. Reduction in effective access time

Solution:

1. Effective memory access time with TLB (Et)

2T: (One T for page table n frame number and one T for required byte in memory)

Et=(TLB hit ratio)*(T + t) + (1-TLB hit Ratio) * (2T + t)

= 0.9*(100+20) + (1-0.9) * (2*100+20)

= 0.9*(120) + (0.1) * (220)

= 108 + 22

= 130ns

2. Effective memory access time without TLB

Ewt = 2T=2*100=200ns

3. Reduction in effective access time

= (Ewt-Et) X (T/Ewt)

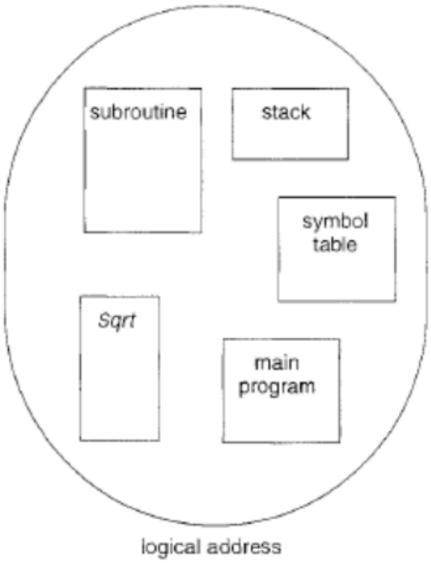
=(200-130)*(100/200)

=35 ns

Segmentation

Basic Method:

1. User view of a main program is a set of methods, procedures, or functions. It may also include various data structures: objects, arrays, stacks, variables, and so on. Each of these modules or data elements is referred to by name.
2. Each of these segments is of variable length and the length is defined by the purpose of the segment in the program. Elements within a segment are identified by their offset from the beginning of the segment: the first statement of the program, the seventh stack frame entry in the stack, the fifth instruction of the Sqrt (), and so on.



User View of Program

3. Segmentation is a memory-management scheme that supports this user view of memory. A logical address space is a collection of segments.
4. Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment. The user therefore specifies each address by two quantities: a segment name and an offset.
5. Segments are numbered and are referred to by a segment number, rather than by a segment name. Thus, a logical address consists of a two tuple:

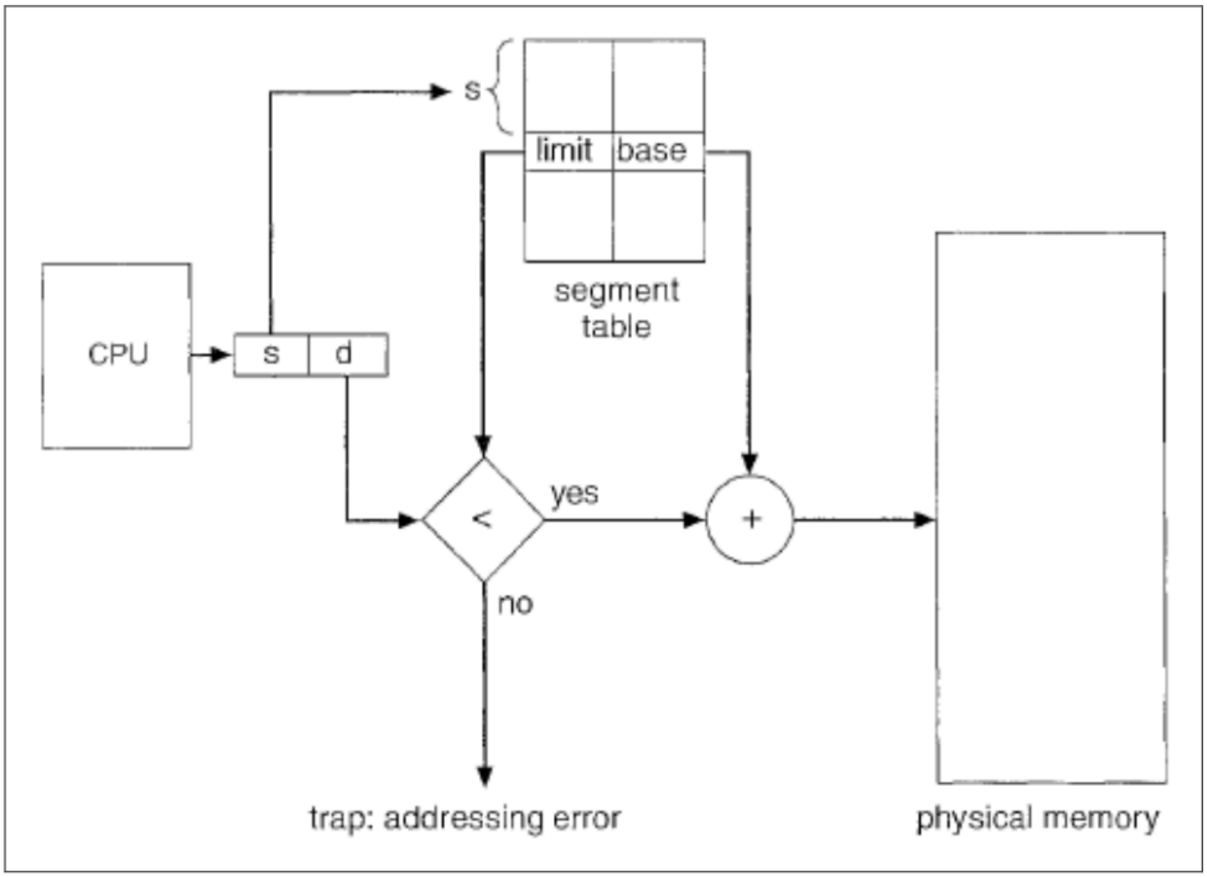
<Segment-number, offset>

6. User program is compiled, and the compiler automatically constructs segments reflecting the input program. A C compiler might create separate segments for the following:

- The code
- Global variables
- The heap, from which memory is allocated
- The stacks used by each thread
- The standard C library

7. Libraries that are linked in during compile time might be assigned separate segments. The loader would take all these segments and assign them segment numbers.

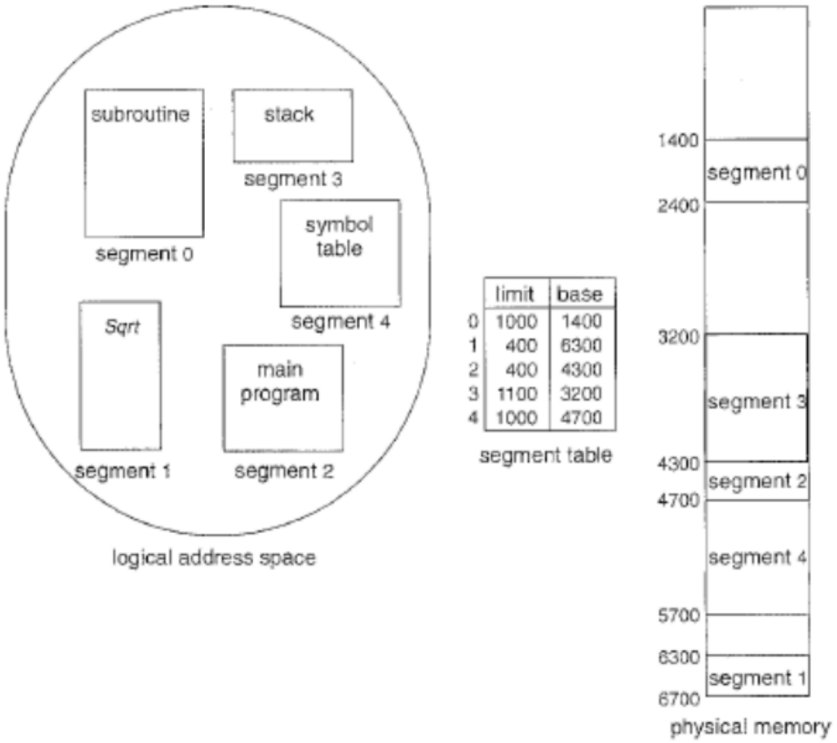
Segmentation Hardware



1. Segment table has a segment base and a segment limit.
2. The segment base contains the starting physical address where the segment resides in memory , and the segment limit specifies the length of the segment.
3. A logical address consists of two parts: a segment number, s, and an offset into that segment, d.

4. The segment number is used as an index to the segment table. The offset d of the logical address must be between 0 and the segment limit. If it is not, we trap to the operating system (logical addressing attempt beyond end of segment).

5. When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte. The segment table is thus essentially an array of base-limit register pairs.



Segmentation Example

6. As an example, consider the situation shown in above Figure. There are five segments numbered from 0 through 4. The segments are stored in physical memory as shown.

7. The segment table has a separate entry for each segment, giving the beginning address of the segment in physical memory (or base) and the length of that segment (or limit).

For example, segment 2 is 400 bytes long and begins at location 4300. Thus, a reference to byte 53 of segment 2 is mapped onto location 4300 +53= 4353.

A reference to segment 3, byte 852, is mapped to 3200 (the base of segment 3) + 852 = 4052. A reference to byte 1222 of segment 0 would result in a trap to the operating system, as this segment is only 1OOO bytes long.

Virtual Memory

1. Virtual memory involves the separation of logical memory from physical memory. This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available.
2. Virtual memory makes the task of programming much easier, because the programmer no longer needs to worry about the amount of physical memory available.
3. The address space of a process refers to the logical (or virtual) view of how a process is stored in memory. This view is that a process begins at a certain logical address, for example, addresses 0 and exists in contiguous memory. Physical memory may be organized in page frames and that the physical page frames assigned to a process may not be contiguous. It is up to the memory management unit (MMU) to map logical pages to physical page frames in memory.
4. Figure below shows virtual address space. Heap can grow upward in memory as it is used for dynamic memory allocation. Stack can grow downward in memory through successive function calls.
5. The large blank space (or hole) between the heap and the stack is part of the virtual address space but will require actual physical pages only if the heap or stack grows.
6. Virtual address spaces that include holes are known as sparse address spaces. Using a sparse address space is beneficial because the holes can be filled as the stack or heap segments grow or if we wish to dynamically link libraries or other shared objects during program execution.

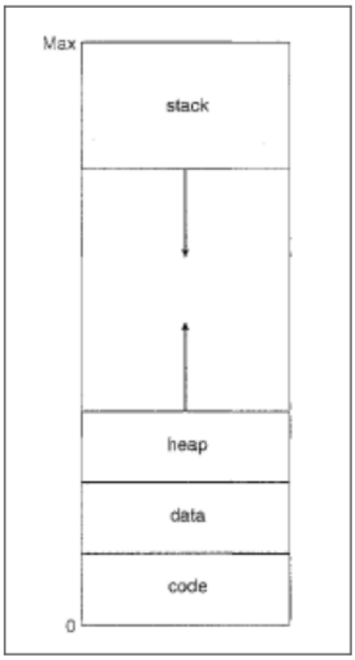


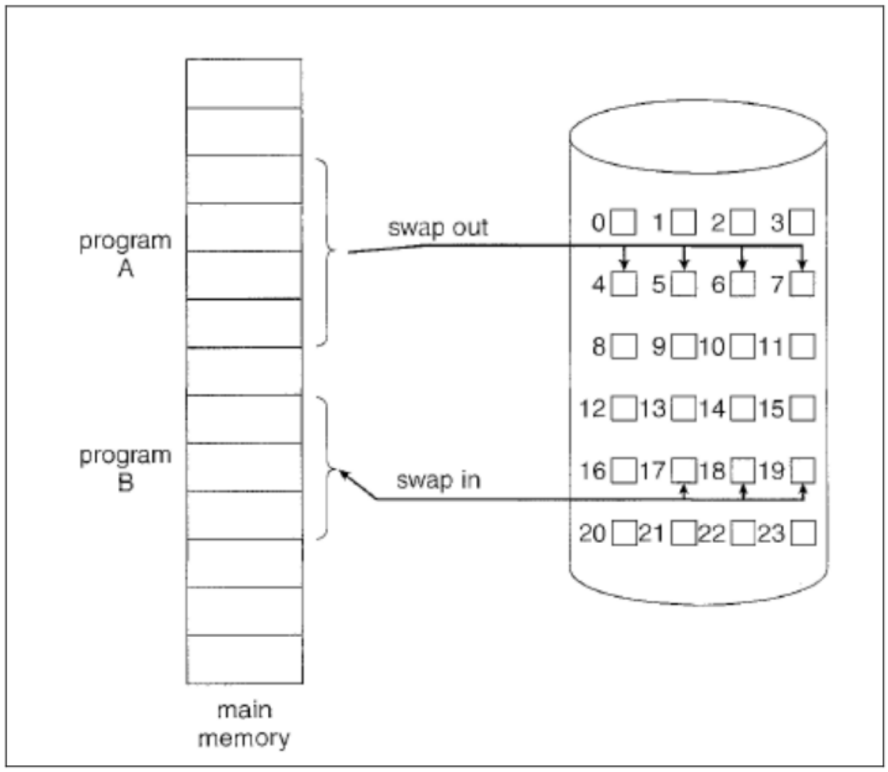
Figure: Virtual Address Space

Demand Paging:

1. An executable program can be loaded from disk into memory by following ways.
2. One way is to load the entire program in physical memory at program execution time. Here the problem is we may not need the entire program in memory as some of the modules of that program may not require. Loading the entire program into memory results loading the executable code for program including all modules regardless of whether the module is required to be executed or not.
3. An Alternative option is to load pages only as they are needed. This technique is known as paging and is commonly used in virtual memory systems.
4. With demand paging, pages are only loaded when they are demanded during program execution. Pages that are never accessed are never loaded into physical memory.
5. A demand-paging system is similar to a paging system with swapping

where processes reside in secondary memory (usually a disk). When we want to execute a process, we swap it into memory.

6. In demand paging, a lazy swapper swaps a page into memory only if that page is needed. A swapper manipulates entire process. Rather than swapper, pager term is used in connection with demand paging.
7. Figure below shows Transfer of a paged memory to contiguous disk space.

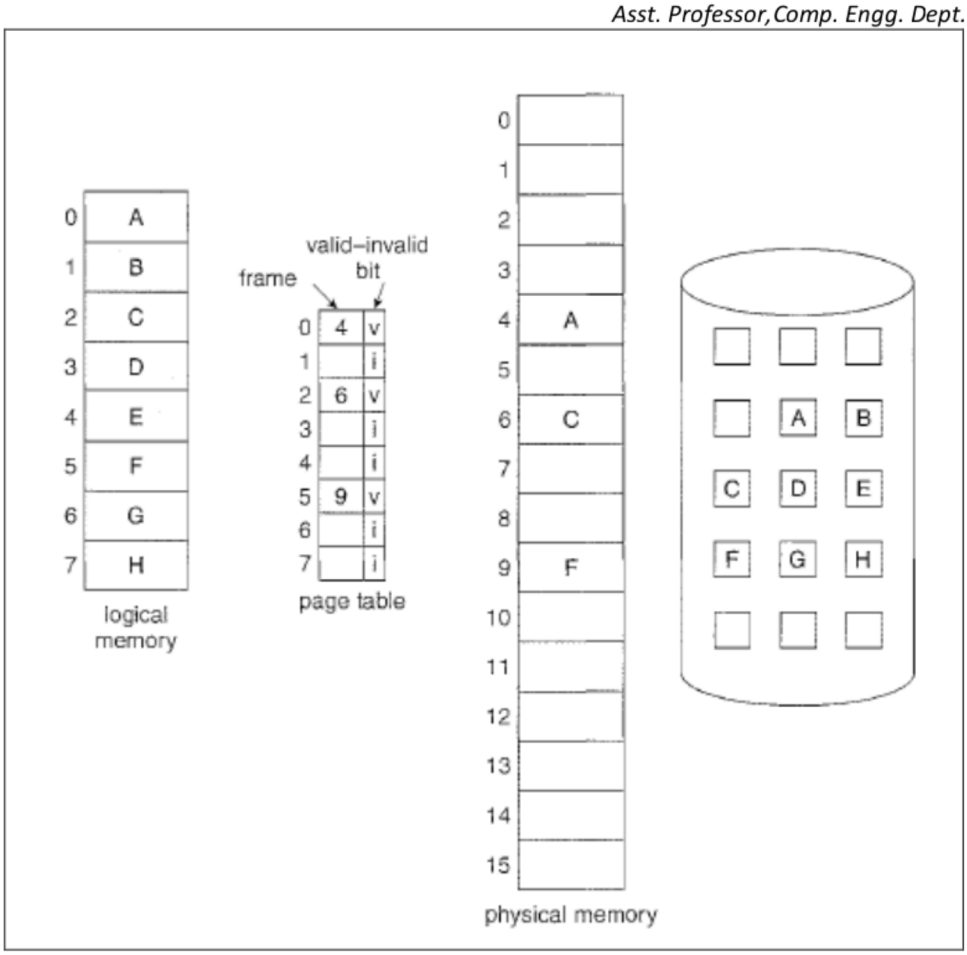


Transfer of a paged memory to contiguous disk space.

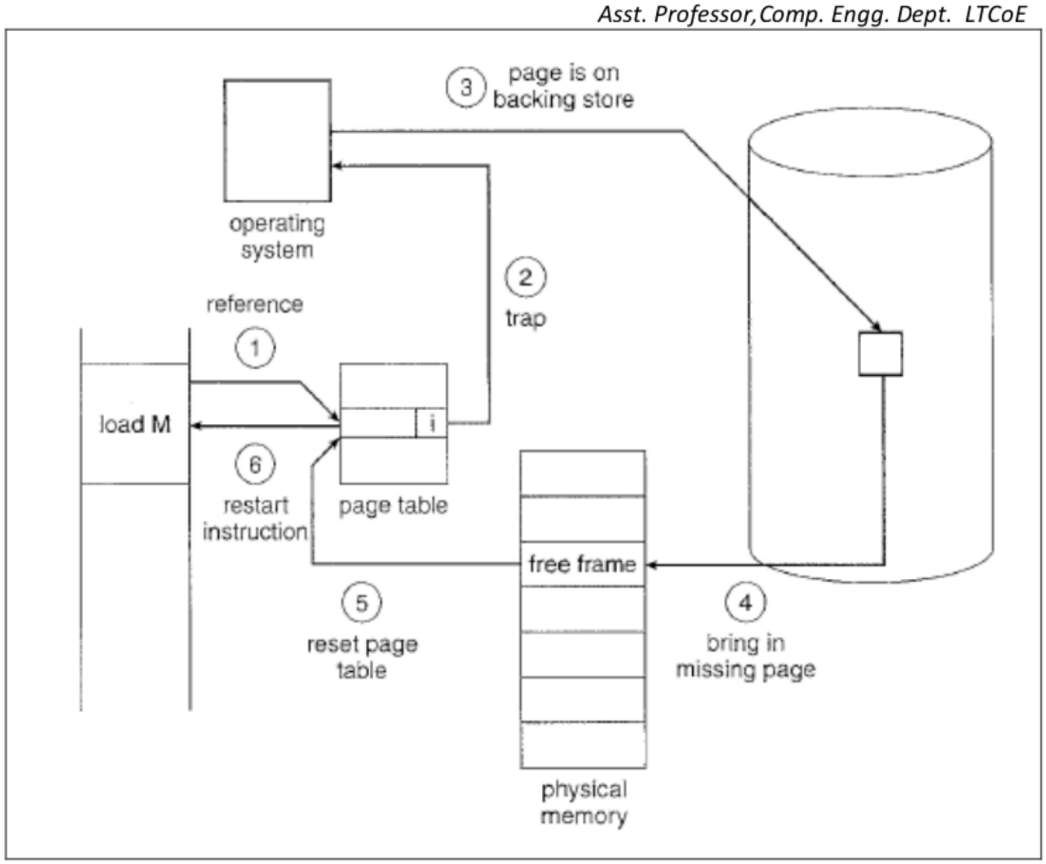
Basic Concepts

1. When a process is to be swapped in, the pager finds which pages will be used before the process is swapped out again.
2. Instead of swapping in a whole process, the pager brings only those pages into memory.

3. Thus, it avoids reading into memory pages that will not be used anyway, decreasing the swap time and the amount of physical memory needed.
4. With this scheme, hardware support is required to distinguish between the pages that are in memory and the pages that are on the disk. The valid -invalid bit scheme is used for this purpose.
5. This time, however, when this bit is set to "valid/' the associated page is both legal and in memory. If the bit is set to "invalid/' the page is not in the logical address space of the process or is currently on the disk.
6. The page table entry for a page that is brought into memory is set as usual but the page-table entry for a page that is not currently in memory is either simply marked invalid or contains the address of the page on disk. This situation is depicted in Figure.



7. Access to a page marked invalid causes a paging hardware a trap to the operating system. This trap is the result of the operating system's failure to bring the desired page into memory.
8. The procedure for handling this page fault is shown in figure.



1) We check an internal table (usually kept with the process control block) for this process to determine whether the reference was a valid or an invalid memory access.

2) If the reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.

3) We find a free frame (by taking one from the free-frame list, for example).

4) We schedule a disk operation to read the desired page into the newly allocated frame. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.

5) We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

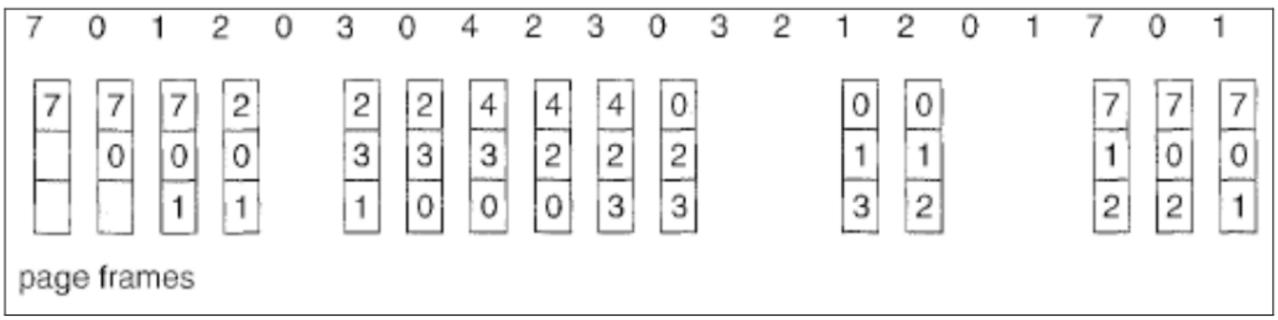
Pure Demand Paging

1. In the extreme case, we can start executing a process with no pages in memory.
2. When the operating system sets the instruction pointer to the first instruction of the process, immediately page fault occurs.
3. After this page is brought into memory, the process continues to execute. Page faults occur until every page that it requires is in memory. At this point it can execute with no more page faults. With this scheme a page is never brought into memory until it is required.

Page Replacement Strategies:

1. First In First Out(FIFO) page replacement algorithm

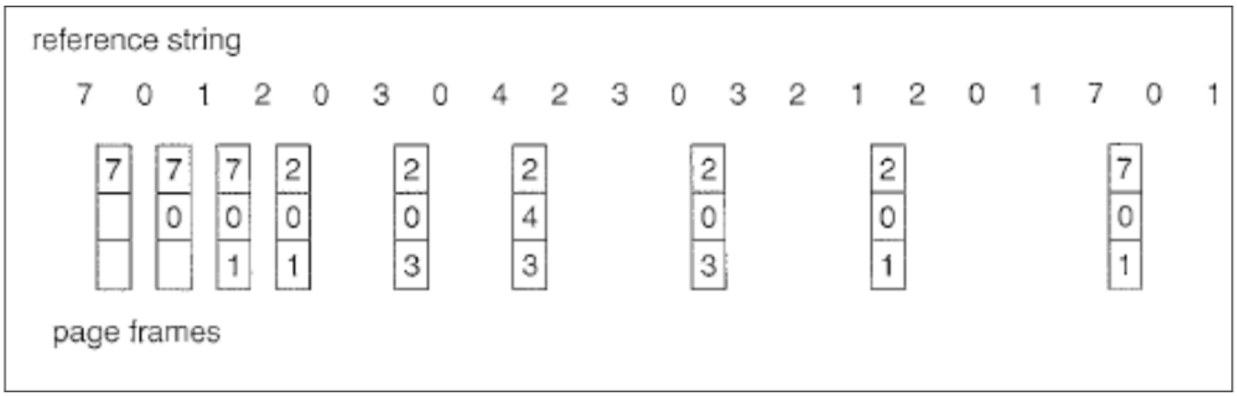
1. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen.
2. We can create a FIFO queue to hold all pages in memory.
3. We replace the page at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue.
   Example:
   Consider a reference string for a memory with three frames.
   7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



4. There are fifteen faults altogether.

2. Optimal Page Replacement algorithm

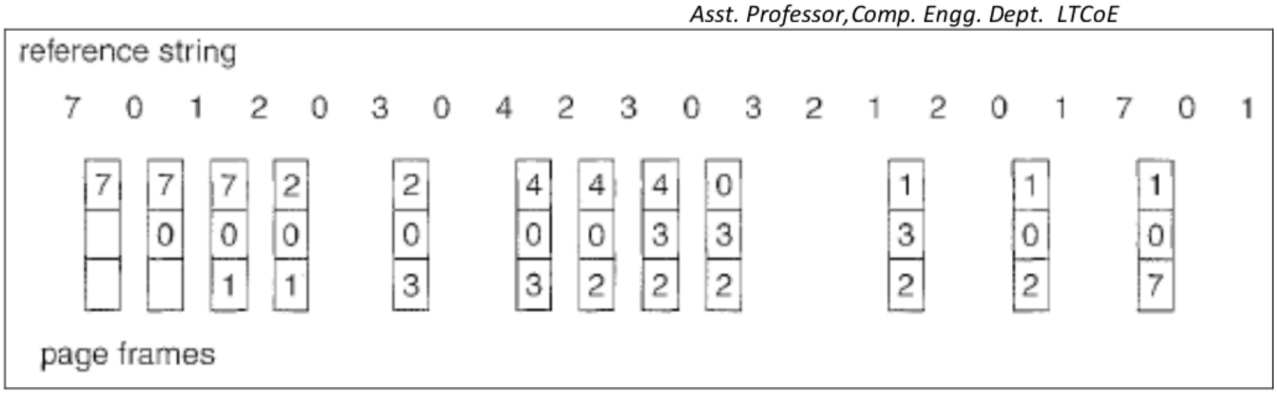
1. Replace the page that will not be used for the longest period of time.
2. **Example:**
   Consider a reference string for a memory with three frames.
   7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



3. There are **nine** faults altogether.

LRU Page Replacement algorithm

1. LRU replacement associates with each page the time of that page's last use.
2. When a page must be replaced, LRU chooses the page that has not been used for the longest period of time.
3. **Example:**
   Consider a reference string for a memory with three frames.
   7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



4. There are twelve faults altogether.

Example: Compare the performance of FIFO, LRU and Optimal based on number of page hit for the following string. Frame size=3
String(pages)= 1 2 3 4 5 2 1 3 3 2 4 5
Solution:

FIFO

Frame	1	2	3	4	5	2	1	3	3	2	4	5
0	1	1	1	4	4	4	1	1	1	1	1	5
1		2	2	2	5	5	5	3	3	3	3	3
2			3	3	3	2	2	2	2	2	4	4
Page fault	y	y	y	y	y	y	y	y	n	n	y	y

No of Hits=2
No of Miss=10

LRU

Frame	1	2	3	4	5	2	1	3	3	2	4	5
0	1	1	1	4	4	4	1	1			4	4

1		**2**	**2**	**2**	**5**	**5**	**5**	**3**			**3**	**5**
2			**3**	**3**	**3**	**2**	**2**	**2**			**2**	**2**
Page fault	**y**	**y**	**y**	**y**	**y**	**y**	**y**	**y**	**n**	**n**	**y**	**y**

No of Hits=2
No of Miss=10

Optimal

Frame	**1**	**2**	**3**	**4**	**5**	**2**	**1**	**3**	**3**	**2**	**4**	**5**
0	**1**	**1**	**1**	**1**	**1**			**3**			**4**	
1		**2**	**2**	**2**	**2**			**2**			**2**	
2			**3**	**4**	**5**			**5**			**5**	
Page fault	**y**	**y**	**y**	**y**	**y**	**n**	**n**	**y**	**n**	**n**	**y**	**n**

No of Hits=5
No of Miss=7

Thrashing

1. Consider any process that does not have "enough" frames. If the process does not have the number of frames it needs to support pages in active use, it will quickly page-fault.

2. At this point, it must replace some page.
3. However, since all its pages are in active use, it must replace a page that will be needed again right away. Consequently, it quickly faults again, and again, and again, replacing pages that it must back in immediately.
4. This high paging activity is called thrashing. A process is thrashing if it is spending more time on paging than executing.

Notes Compiled By Archana Naware
Asst. Professor ,Comp. Engg. Dept. LTCoE

5. File Management

1. Overview

1.1 Definition of File

A file is named collection of related information that is recorded on secondary storage. File represents programs and data. Data files are numeric ,alphabetic, alphanumeric, or binary. File can be free forms such as text files, bytes, lines or records. Information in the file is defined by the creator of the file. Information can be of type source programs, object programs, executable programs, numeric data, text, payroll records, graphics images, sound recordings and so on.
File has certain defined structure according to its type.

- A text file is a sequence of characters organized into lines.
- Source file is a sequence of subroutines and functions.
- Object file is sequence of bytes organized into blocks and understandable by system's linker.
- Executable file is a series of code sections that the loader can bring into memory and execute.

1.2 File Attributes

File is named for the convenience of the user. A name is string of characters. Some systems differentiate between upper case lower case characters in names where as some systems consider it equivalent. When a file is named, it becomes independent of the process, the user, even the system that created the file. A file typically consists of the following attributes.

1. **Name** - Human readable form.
2. **Identifier** - Unique tag usually a number. It is non-human readable form.
3. **Type-** this information is for the systems that support different types.
4. **Location**- it's a pointer to a device and the location of the file on that device.
5. **Size** – current size of the file it is in bytes, words or blocks.
6. **Protection** - Access control information such as read , write , execute permissions.
7. **Time, date and user identification** - This information may be kept for creation, last modification and last use. The information is needed for protection, security and usage monitoring.

1.3 File Operations

File is abstract data type. The operating system provides system calls to perform file operations. The file operations are as follow.

1. **Creating a file** –Two steps are necessary to create a file.
    a. Space must be found for the file.
    b. An entry for new file must be made in the directory.

2. **Writing a file**- to write to a file system call is made to specify the name of the file and information to be written to the file. With this name of the file system searches the directory to find the location of the file. System keeps write pointer to the location in the file where next write is to take place.
3. **Reading a file-** to read from a file system call is made which specifies name of the file and where the next block of the file should be put. Directory is searched for the specified file name and system keeps read pointer to the location in the file where the next read is to take place.
   After reading the file read pointer is updated.
4. **Repositioning within a file-**the directory is searched for the specific file name and the current file position s set to the given value. Repositioning within a file is also called file seek.
5. **Deleting a file-**The directory is searched for the specified file name and after the file is found, all file space is released so that it can be reused by other files.
6. **Truncating a file-** To erase some contents of the file , user truncate the file instead of deleting it. In this operation only length is reduced keeping all its attributes same.

1.4 File Types

File type is implemented as a apart of file name. The name is split into two parts- name and extension separated by a period character. User and operating system can tell the type of the file from the name of the file.

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

Common File types

2. File Organization and Access Methods

2.1 File Structure

A File Structure should be according to a required format that the operating system can understand. An object file is a sequence of bytes organized into blocks that are understandable by the system. When operating system defines different file structures, it also contains the code to support these file structure.

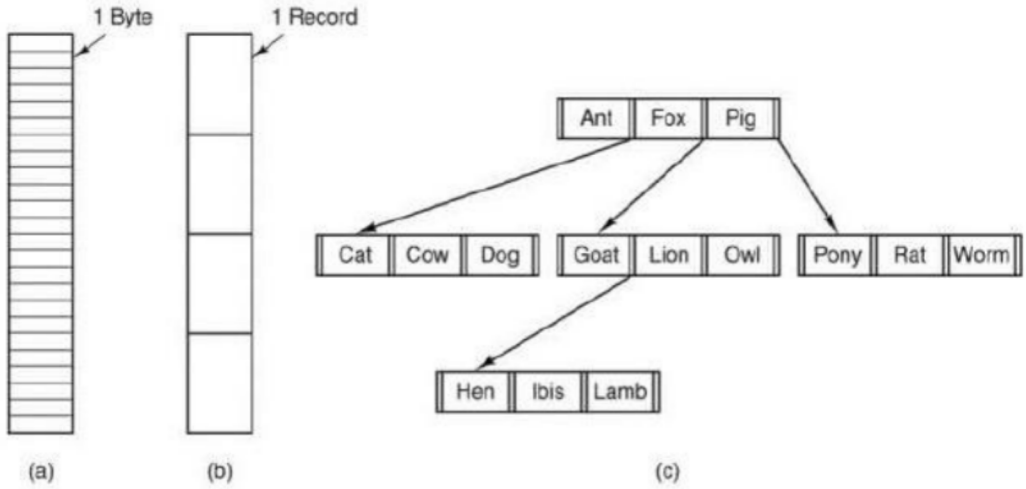
Internal File Structure

Files can be structured in several ways. Three common possibilities are depicted in the following Figure.

a) *Sequence of bytes*

b) *Sequence of records*

c) *Tree structure*



Internal File Structure

1. Internally, locating an offset within a file can be complicated for the OS.
2. Disk systems typically have a well-defined block size determined by the size of a sector. All disk I/O is performed in units of one block (physical record), and all blocks are the same size.
3. It is unlikely that the physical record size will exactly match the length of the desired logical record. Packing a number of logical records into physical blocks is a common solution to this problem. For example, the UNIX OS defines all files to be simply streams of bytes. Each byte is individually addressable by its offset from the beginning (or end) of the file. In this case, the logical record size is 1 byte. The file system automatically packs and unpacks bytes into physical disk blocks -say, 512 bytes per block- as necessary.
4. The file may be considered to be a sequence of blocks. All the basic I/O operations occur in terms of blocks.
5. Because disk space is always allocated in blocks, some portion of the last block of each file is generally wasted. If each block were 512 bytes, for example, then a file of 1,949 bytes would be allocated four blocks (2,048 bytes); the last 99 bytes would be wasted.

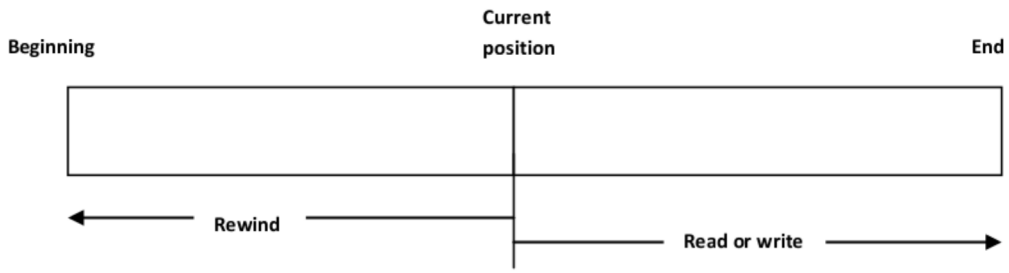
2.2 Access Methods

Information in the file is accessed in several ways and thoses are discussed below.

1. Sequential access

i) It is the simplest access method. Information in the file is processed in order, one record after the other.

ii) This mode of access is the most common access method. for example, editor and compiler usually access the file in this fashion.
iii) A read operation reads next portion of the file and automatically advance a file pointer which keeps track I/O location.
iv) Write appends to the end of the file and advances to the end of the newly written contents. Sequential access is based on a tape model of a file and also work on sequential access devices as it does on random access devices.
v) Sequential access is as shown in the following figure.



Sequential – access file

2. Direct Access

i) Another method is direct access method also known as relative access method. A file is made of fixed-length logical record that allows the program to read and write record rapidly in no particular order. The direct access is based on the disk model of a file since disk allows random access to any file block.
ii) For direct access, the file is viewed as a numbered sequence of block or record. Thus, we may read block 14 then block 59 and then we can write block 17.
iii) There is no restriction on the order of reading and writing for a direct access file.
iv) Direct access is useful for immediate access of large amount of information and databases are often of this type of access.
v) A block number provided by the user to the operating system is normally a relative block number. The first relative block of the file is 0 and then 1 and so on.
vi) For direct access method file operations must be modified to include block number as a parameter. Thus ***read next*** is modified as ***read n*** and ***write next*** is modified as ***write n*** where ***n*** is a block number.
vii) Block number is usually a relative block number. Relative block number is an index relative to the beginning of the file. Thus first relative block is 0, second is 1 and so on. Thus given a logical record length L, a request for record N is turned into I/O request for L bytes starting at location L * (N – 1) within the file (assume N=1).

3. Index files

i) These methods involve the construction of an index for the file.
ii) Index contains pointers to the various blocks.

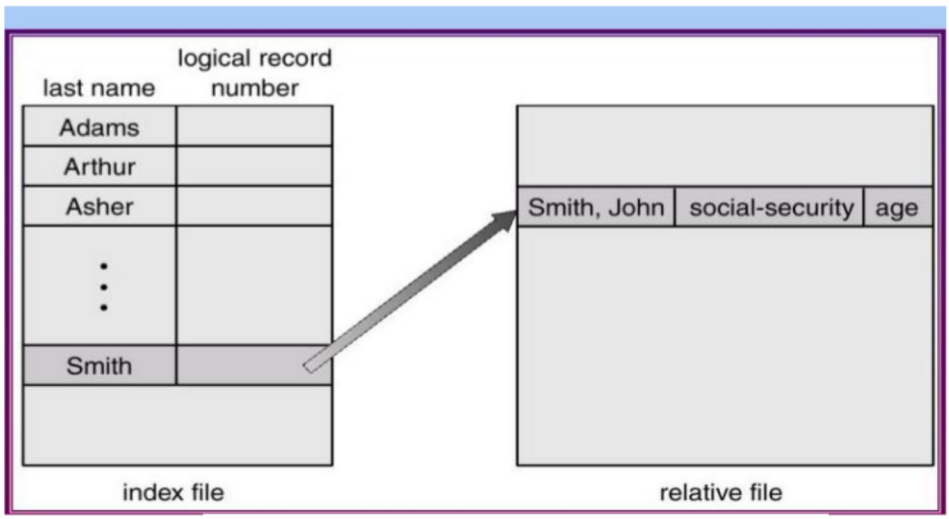
iii) To find a record in a file, index is searched and then pointer is used to access the file directly and to find the desired record.

iv) Example: Consider a retail-price file containing Universal Product Code (UPC) for items with associated price. UPC is 10 digits, price is 6 digit. Each record is of 16 Bytes. Consider a disk containing 1024 Bytes per block. Each block contains 64 records. A file containing 120000 records occupy about 2000 blocks (2M bytes).

v) If the file is sorted by UPC, index can be defined consisting of the first UPC in each block. Thus index would have 2000 entries each of which is 10 digits. To find price of a particular item, index is searched and the block containing the desired record if found.

vi) With large file index files becomes to large. Solution is to create index to index file. Primary index files contains pointers to secondary index files which points to the actual data items.

vii) Figure below is the example of index and relative files.



Example of index and relative files

3. File Directories and Structure

A directory is a container that is used to contain folders and file. It organizes files and folders into a hierarchical manner.

There are several logical structures of a directory, these are given below.

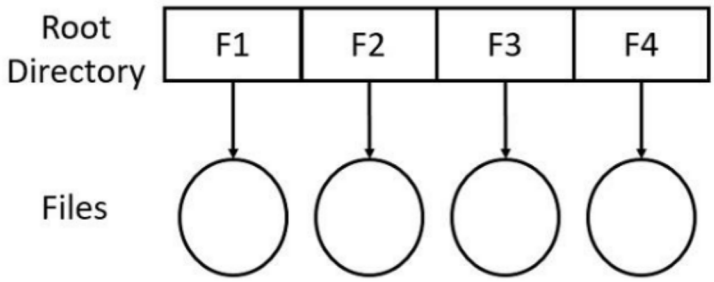
1. Single-level directory

i) Single level directory is simplest directory structure. In it all files are contained in same directory which make it easy to support and understand.

ii) A single level directory has a significant limitation, however, when the number of files increases or when the system has more than one user. Since all the files are in the same directory, they must have the unique name. If two users call their data files test, then the unique name rule is violated.

iii) Single-level directory structure is as shown in the following figure.

iv)



Single level directory

Advantages:

a. Since it is a single directory, so its implementation is very easy.
b. If the files are smaller in size, searching will become faster.
c. The operations like file creation, searching, deletion, updating are very easy in such a directory structure.

Disadvantages:

a. There may chance of name collision because two files can not have the same name.
b. Searching will become time taking if the directory is large.
c. In this cannot group the same type of files together.

2. Two-level directory

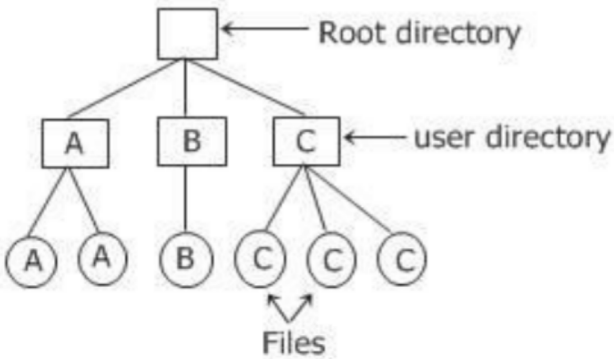
i) as we have seen, a single level directory often leads to confusion of files names among different users. The solution to this problem is to create a separate directory for each user.

i) In the two-level directory structure, each user has their own user files directory (UFD).

ii) The UFDs has similar structures, but each lists only the files of a single user.

iii) System's master file directory (MFD) is searches whenever a new user id=s logged in. The MFD is indexed by username or account number, and each entry points to the UFD for that user.

iv) The two level directory structure is as shown in the following figure.



Two-level directory

Advantages:

a. We can give full path like **/User-name/directory-name/.**
b. Different users can have same directory as well as file name.
c. Searching of files become more easy due to path name and user-grouping.

Disadvantages:

a. A user is not allowed to share files with other users.
b. Still it is not very scalable; two files of the same type cannot be grouped together in the same user.

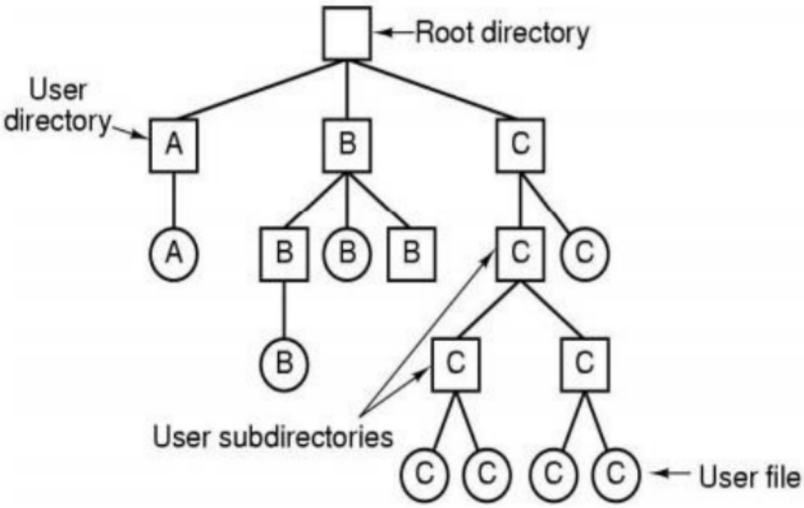
3. Tree structured directory

i) Once we have seen a two-level directory as a tree of height 2, the natural generalization is to extend the directory structure to a tree of arbitrary height.

ii) This generalization allows the user to create their own subdirectories and to organize on their files accordingly.

v) A tree structure is the most common directory structure. The tree has a root directory, and every file in the system have a unique path.

vi) Tree structured directory is as shown in figure below.



Advantages:

i) Very generalize, since full path name can be given.
ii) Very scalable, the probability of name collision is less.
iii) Searching becomes very easy, we can use both absolute path as well as relative.

Disadvantages:

i) Every file does not fit into the hierarchical model; files may be saved into multiple directories.
ii) We cannot share files.
iii) It is inefficient, because accessing a file may go under multiple directories.

4. Acyclic graph directory

i) An acyclic graph is a graph with no cycle and allows sharing subdirectories and files.

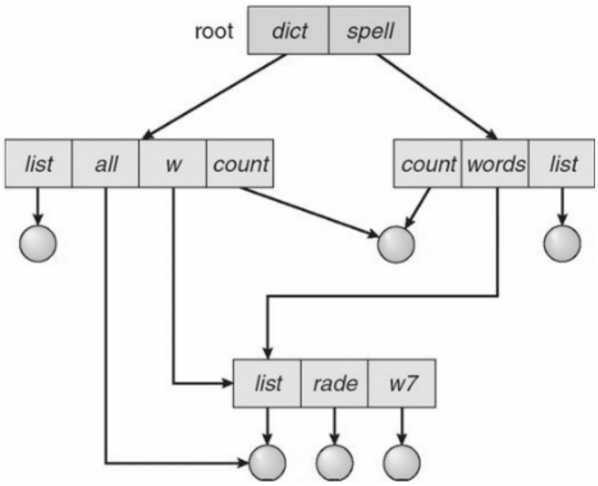
ii) The same file or subdirectories may be in two different directories.

iii) It is a natural generalization of the tree-structured directory.

iv) It is used in the situation like when two programmers are working on a joint project and they need to access files. The associated files are stored in a subdirectory, separating them from other projects and files of other programmers, since they are working on a joint project so they want the subdirectories to be into their own directories. The common subdirectories should be shared. So here we use acyclic directories.

v) It is the point to note that shared file is not the same as copy file. If any programmer makes some changes in the subdirectory it will reflect in both subdirectories.

vi) Acyclic graph directory is as shown in figure below.



Advantages:

a. We can share files.

b. Searching is easy due to different-different paths.

Disadvantages:

a. We share the files via linking, in case of deleting it may create the problem,

b. If the link is soft link then after deleting the file we left with a dangling pointer.

c. In case of hard link, to delete a file we have to delete all the reference associated with it.

5. General graph directory

i) In general graph directory structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory.

ii) The main problem with this kind of directory structure is to calculate total size or space that has been taken by the files and directories.

iii) General graph directory is as shown in figure.

Advantages

i) It allows cycles.

ii) It is more flexible than other directories structure.

Disadvantages

i) It is more costly than others.
ii) It needs garbage collection.

4. File Allocation Methods:

File allocation methods define how the files are stored in the disk blocks. There are three methods.

1. Contiguous Allocation
2. Linked Allocation
3. Indexed Allocation

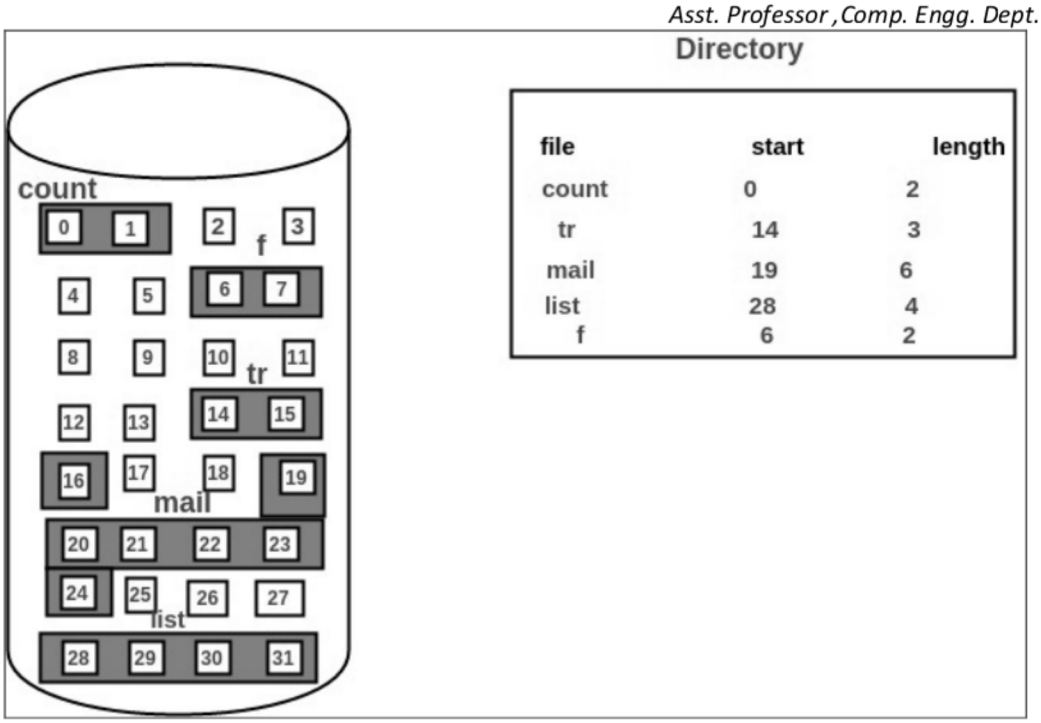
The main idea behind these methods is to provide:

a. Efficient disk space utilization.
b. Fast access to the file blocks.

4.1. Contiguous Allocation

1. Each file occupies a contiguous set of blocks on the disk. For example, if a file requires 5 blocks and is given a block b as the starting location, then the blocks assigned to the file will be: b, b+1, b+2,b+3,b+4.
2. This means that given the starting block address and the length of the file we can determine the blocks occupied by the file. The directory entry for a file with contiguous allocation contains
    - Address of starting block
    - Length of the allocated portion.

The file list in the following figure starts from the block 28 with length = 4 blocks. Therefore, it occupies 28,29,30,31 blocks.



Advantages:

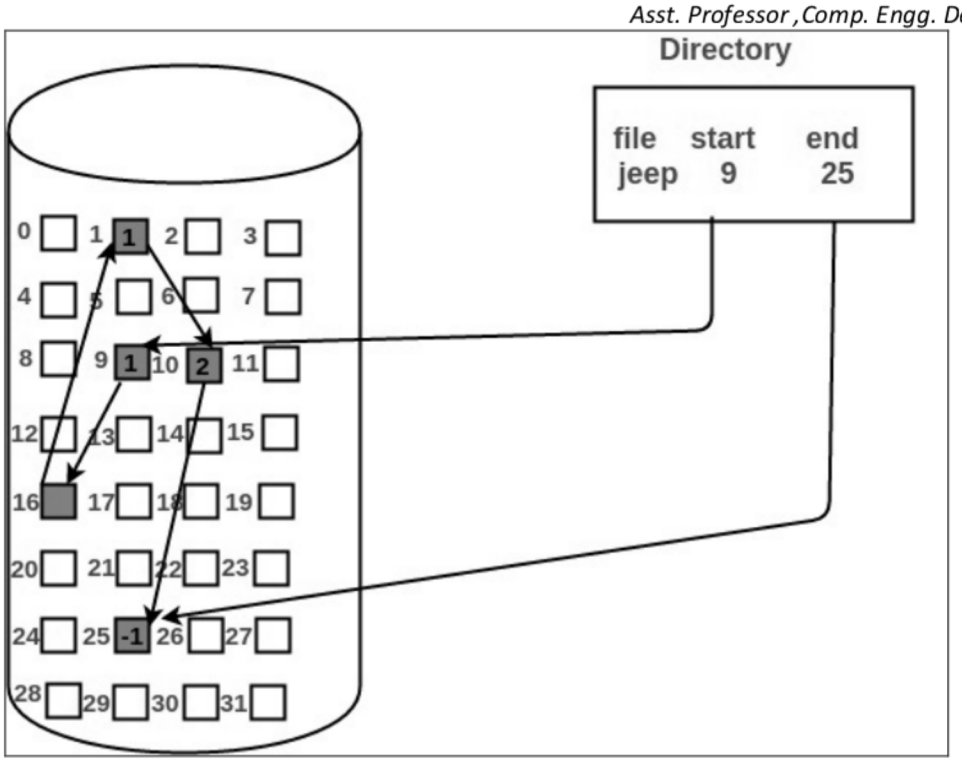
1. Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
2. This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.
3.

Disadvantages:

1. This method suffers from internal and external fragmentation. This makes it inefficient in terms of memory utilization.
2. Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

4.2 Linked List Allocation

In this scheme, each file is a linked list of disk blocks which can be scattered anywhere on the disk.
The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.
Example: The file 'jeep' in following diagram shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.



Advantages:

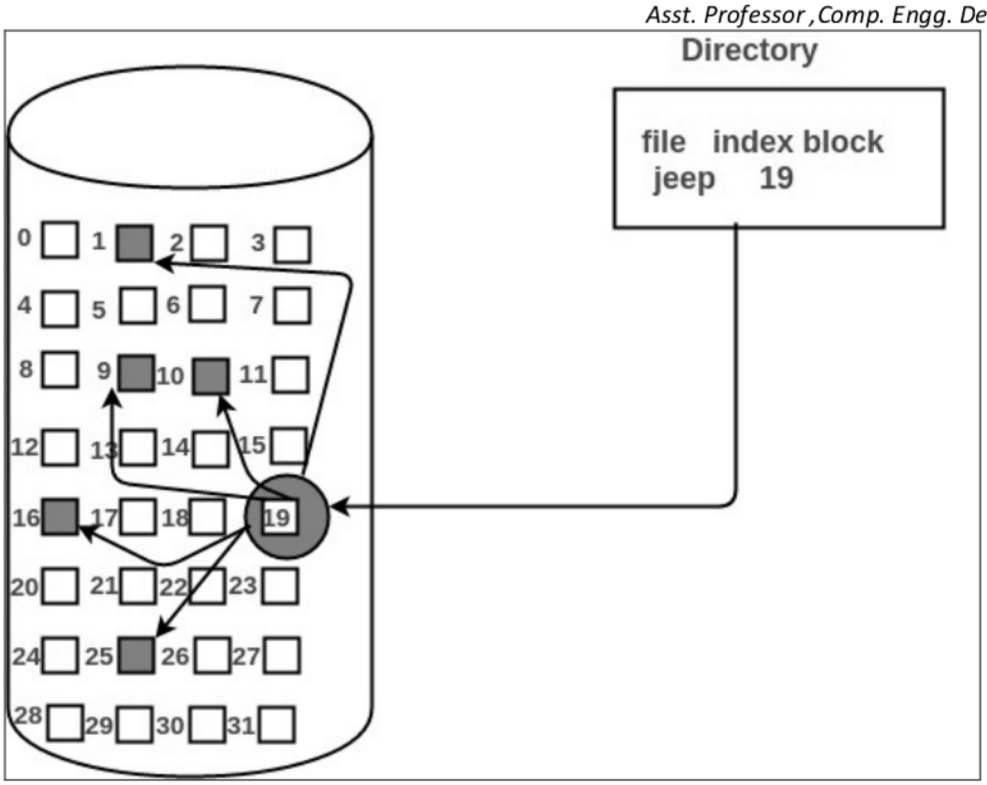
1. This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for availability of contiguous memory.
2. This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages:

1. Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
2. It does not support random or direct access. We cannot directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
3. Pointers required in the linked allocation cause some extra overhead.

4.3. Indexed Allocation

1. In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file.
2. Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block.
3. The directory entry contains the address of the index block as shown in the following figure.



Advantages:

1. This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
2. It overcomes the problem of external fragmentation.

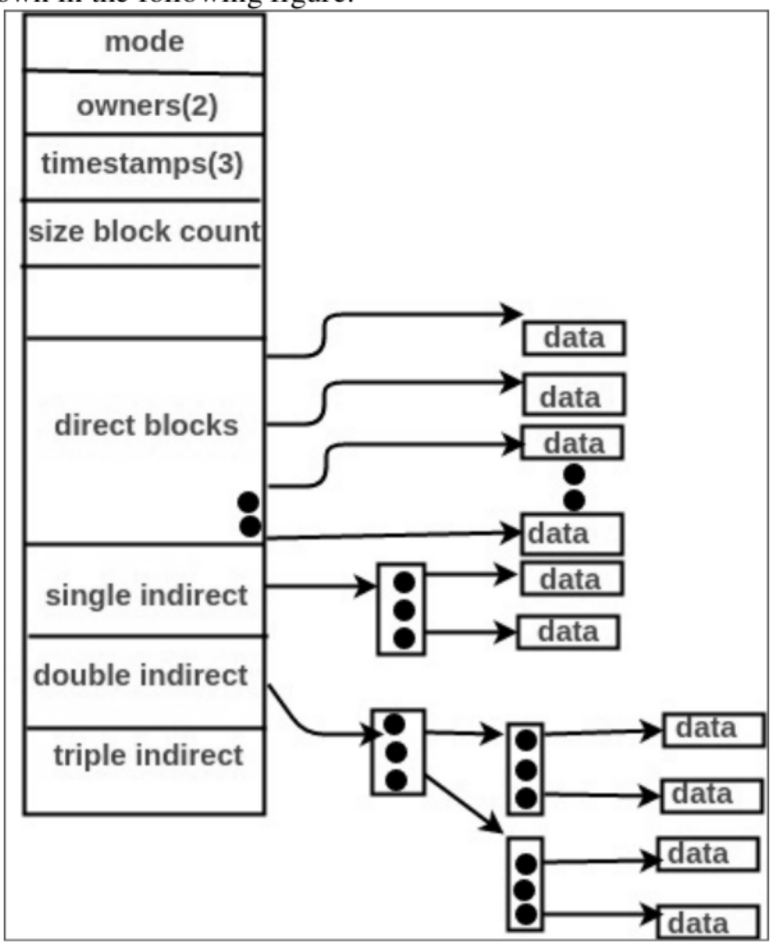
Disadvantages:

1. The pointer overhead for indexed allocation is greater than linked allocation.
2. For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

4.4 For files that are very large, single index block may not be able to hold all the pointers.
Following mechanisms can be used to resolve this:

1. **Linked scheme:** This scheme links two or more index blocks together for holding the pointers. Every index block would then contain a pointer or the address to the next index block.
2. **Multilevel index:** In this policy, a first level index block is used to point to the second level index blocks which in turn points to the disk blocks occupied by the file. This can be extended to 3 or more levels depending on the maximum file size.
3. **Combined Scheme:**

1. In this scheme, a special block called the **Inode (information Node)** contains all the information about the file such as the name, size, authority, etc and the remaining space of Inode is used to store the Disk Block addresses which contain the actual file as shown in the following figure.



2. The first few of these pointers in Inode point to the direct blocks i.e the pointers contain the addresses of the disk blocks that contain data of the file.
3. The next few pointers point to indirect blocks. Indirect blocks may be single indirect, double indirect or triple indirect.
4. Single Indirect block is the disk block that does not contain the file data but the disk address of the blocks that contain the file data. Similarly, double indirect blocks do not contain the file data but the disk address of the blocks that contain the address of the blocks containing the file data.

5. File Sharing

File sharing is very important for the users who want to collaborate and to reduce the efforts required to achieve a computing goal. There are many more aspects of the file sharing discussed below.

5.1 Multiple Users

i) In multiuser systems, system can allow a user to access the files of other users by default or it may require that a user grant access to the files.
ii) To implement sharing and protection system must maintain more file and directory attributes than single user system.
iii) Most systems use the concept of file/directory owner and group. Owner can change attributes, grant access and has most control over the file or directory. Group attribute defines subset of users who has share access to the file.
iv) To implement owner attributes a list of user names and associated user identifiers (User IDs) is maintained. These numerical UIDs are unique per user.
v) UID is associated with all users' processes and threads. When user reads the process, UID is translated back to the user name via the user name list.
vi) Group functionality is implemented as a system wide group names and group identifiers. Every user can be in one or more groups. The user's group IDs is included in every associated process and thread.
vii) When a user requests an operation on a file, UID is compared with owner attribute. Likewise group IDs can be compared. Like this permissions are determined which are applicable. Those permissions are applied to the requested operation and the requested operation is accepted or denied.

5.2 Remote File Systems

i) Networking allows sharing of resources within a campus or around the world. Data in the form of files is one of the resource types.
ii) File sharing methods have changed in the evolution of network. In the first method users were manually transferring files between machines via programs like ftp. Ftp is used for both anonymous and authenticated access. Anonymous access allows user to transfer files without having an account on the remote system.
iii) Second method is Distributed File System in which remote directories are accessible from local machines. There is tighter integration between the machines that is accessing the remote files and the machine providing the files.
iv) Third method is World Wide Web in which browser is needed to gain access to remote files and separate operations are used to transfer files. WWW uses anonymous file exchange exclusively.

.

6. Linux Virtual File system

1) Linux virtual file system is based on object oriented principles. It has two components. First component is set of definitions that define what a file object is allowed to look alike and second component is layer of software to manipulate those objects.
2) Three main object types are inode-object, file-object structure which represents individual files and file-system object which represents an entire file system.

3) For each object VFS defines a set of operations that must be implemented by that structure. Each object contains a pointer to a function table. The function table lists the addresses of the actual functions that implement those operations for that particular object.
4) VFS does not need to know whether an inode represents a networked file, a disk file, network socket or a directory file.
5) The file system object is connected as a set of files that forms a self-contained directory hierarchy. The kernel maintains single file system object for each disk mounted as file system and for each networked file system currently connected.
6) The file system object's main responsibility is to give access to inode. File system object returns the inode with a particular inode number to VFS and VFS identifies every inode by a unique file-system-inode number pair.
7) The inode and file-object are used to access files. Inode object represents file as a whole whereas file –object represents a point of access in the file.
8) A process cannot access an inode's data contents without obtaining a file-object pointing to the inode.
9) File object keeps track of the location in the file where the process is currently reading or writing. It remembers whether the process asked for write permission when the file was opened and also keeps track of process's activity.
10) File objects belong to a single process but inode-objects do not. All cached file data are linked onto a list are linked onto a list in the file's inode object. Inode also maintains file's standard information such as owner, size, time most recently modified.
11) Directory is treated differently than files. UNIX programming interface defines a number of operations on directories such as creating, deleting, and renaming. Directory operations do not need that the user should open concerned files.
12) VFS defines these operations in inode objects rather than in the file objects.