OS

↳ Introduction :

↳ OS manages the computer hardware.

↓

Resources like cpu, memory I/O devices.

↳ acts as intermediary between computer hardware and user interface.

↳ It Allocates the Resources efficiently.

Goals:

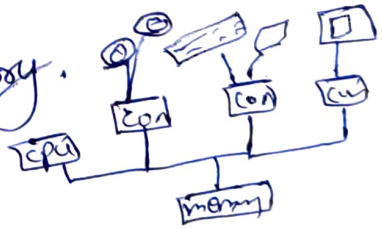
↳ It is very convenient to the user

↳ It provides an efficient usage of a system.

↳ It provides maximum throughput.

↳ Common bus :

↳ A computer system consists of one or more cpus and many device controllers which are connected to a common line or bus. that can give an access to share information between them and memory.



↳ device controllers :

↳ Controller consists of many I/o. devices connected to it, ~~makes~~ works Individually..

↳ memory is shared concurrently to avoid lag.

↳ memory controller :

↳ It synchronice memory in a smoothway to all the controllers.

↳ Bootstrap programs :

↳ program that run when we reboot @ poweron the computer.

↳ It will load the operating system.

↳ It is stored in the ~~memory~~ ROM.

↳ It will load OS kernel into main memory.

(heart)

↳ Interrupt :

↳ Hardware will Interrupt the cpu by sending the signal through system bus.

and the cpu to execute the task

It will be given presently.

↳ system call :

↳ software will Interrupt the cpu by sending the signal through system bus.

↳ Interrupt · hard. ~~Contains~~ a Service Routine · ~~Others~~
It contains · starting addrees and work that has to do. and then after completing the work it will Resumed with · first one.

↳ Computer · storage structure · consists · of.
main memory and secondary memory.
↳ In the main memory · programs & softwares are loaded.
↳ In the secondary memory · programs & softwares are stored.
↳ If the computer has more ram then it will be faster & it loads very fast.
↳ If · the power of · computer is on. then · main: memory and · secondary memory · will be active.
↳ If the power of Computer is off then main memory will not be actice. only secondary memory will be active.

↳ device controllers maintains · local Buffer storage and set of registers.
↳ device · controllers have device drives that will help to understand the nature of · device and provides uniform Interface.

↳ working of I/O operation.

↳ Device controllers have device drivers that will help to load the Registers.

↳ Those Registers contain the action that has to be done. and then device controller starts transferring the data to the local buffer

↳ After completing the transfer then device controller informs the device driver via an interrupt that it has completed its work.

↳ Then device driver returns the control to OS.

limitation:

↳ This is only applicable for small amount of data.

↳ To solve this problem. (DMA) Direct memory access is used. which is directly transfers the block of data. from our buffer to memory.

↳ It Interrupts only once when its work is completed and also at that time CPU is not have any work so it can be used for other I/O works.

↳ we can solve this issue by DMA

↳ Computer system consists of single processor system and multi processer system.

↳ single processer system has only one main cpu. It has to execute all the instructions that has given..

↳ multiprocesser system has more then one cpu.

↳ All cpu's are closly communicated and they share common system bus and memory, which will help the work easier and if one cpu not working then other can take over.

↳ performance increases

↳ convenience.

↳ Types of multiprocessor.

↳ symetric processors → Here all the cpu's are involved work. that is equally shared.

↳ Asymetric processers → Here every single processor has its own work. and that will assigned by a master cpu..

↳ clustered system → Here they are composed of two or more individual system coupled together.

↳ OS structure.

↳ OS is capable of doing multiprogramming and Time sharing (multitasking)

multiprogramming

↳ It Increases the cpu utilization. very efficiantly. It assigns the cpu. for all the work and it keep it busy.

↳ work like memory & peripheral devices. but their is no user Interaction with system.

multi-tasking

↳ It provides a direct communication b/w the user and the system. and cpu or executes multiple jobs by switching among them very frequently. (user does not feel any lag while using).

process.

↳ A program loaded into memory and executing is called process.

OS Services :-

- User Interface
  - Command line interface
  - Graphical user interface.
- program execution.
  - Execute & run the software, programs.
- I/O operation.
  - OS is present in b/w user and cpu. which will controls the usage of I/O devices. Ex : keyboard, mouse---
- File system manipulation.
  - create, edit, delete controls by OS.
- Communication :
  - communication b/w the process are controlled by OS. where those proces are present in same computer & diffent computer with is connect by a network.
- Error detection.
  - OS detects the error in the computer and saves the computer from Break down
- Resource Allocation.
  - It Allocate the Resource to all the processes sygnchronisly.

↳ Accounting
  ↳ It will keep record of every user usages in the system and what they used.

↳ protection and security
  ↳ Access to the system Resources are controlled.
  ↳ Access to the system resources are not allowed by outside users.

↳ OS Interface
  ↳ OS provides CLI which allows the user to enter some commands that are to be performed by OS.
  ↳ OS also provides GUI which allow the user to directly communicate by using keyboard & mouse etc.

⇒ multiple Command Interpretors are called shells.

↳ System Calls.

↳ system call provide an Interface to the services that are make available by an os.

↳ we have Two modes one is usermode and kernel mode, In user mode if the user wants to do some task then their will be no acesses to the memory & any other resources.

But In kernal mode we can acesses all the resources.

↳ system call is nothing but when user is doing some task in the user mode and if he wants to use kernel mode then a call is reacested for the service provided by os in the kernel mode the call is known as system call.

↳ Types of system calls.

↳ process control.

↳ If the processes wants to end, abort, load, execute, create process, teminat process, wait for time, allocate memory, free the memory we need system calls.

↳ file manipulation.

↳ To create file, To delete file, to open & close, To read & write & reposition we need system call.

↳ Device manipulation.
  ↳ To logically attach & detach the devices and read, write and reposition and setting attributes we need system calls.

↳ Information maintenance.
  ↳ setting and getting the time & date
  ↳ To ~~maintai~~ maintain the system upto date we need system calls.

↳ Communications.
  ↳ For the communication between the process we need system calls & send & receive messages and share status Information we need system calls.

↳ System programs:
  ↳ system programs helps the user with the OS by providing; programs that helps in managing files. by file management, programs that helps in showing the status information in the system, programs that helps in file modifications such modification that done inside the file, programs that helps in providing programming-language support like compilers, Assembels etc, and also Absolute loaders, Relocatable loaders ... etc

↳ system programs provides a virtual connection among processes, users with computer systems over webpages, mail ... etc.

↳ To Design an OS

↳ we need.

↳ we have to choose nice Hardware.

↳ Type of the system.

↳ Requirements

↳ User Goals

↳ System Goals.

↳ Implementation of OS

↳ OS have been written in assembly language

↳ But now we have C, C++, Java ... etc.

↳ process management.

process: when a program begins its execution at that time is known as process.

Thread:

A thread is the unit of execution. within the process, A process can have one to many threads.

process state :

A process can change its state according to their environment.

↳ New state → A process is being created.

↳ Running state → A process is executing.

↳ waiting state → A process is waiting for some event to occure.

↳ Ready state → A process is being created and it is Ready to execute.

↳ Termination state → A process is terminating from the execution.

process control block :

↳ process is represented in OS as process control block.

↳ process control block contains the related information of the process such as process state, process number etc.

↳ process number → It contains unique ID of process

↳ process state → It tells in which state the process is present.

↳ process Counter → It contains the address of the next instruction.

↳ cpu Registers → Registers that are being used by the process.

↳ cpu scheduling information:
↳ It contains the order in which the process has to be executed.

↳ memory management information
↳ It contains the memory related information of the process. such as memory used by the process
--- etc

↳ Accounting information:
↳ In this block it keep the record of certain things such as resource used, status information ---. etc.

↳ I/O states information:
↳ Information that are related to the I/O devices for the process.

↳ process scheduling:

↳ process scheduler selects the available process from in the system. and gives the job and assigns them in a queue.
↳ Ready.

↳ Job queue → when a process wants the job to run. any program then they are put into a job queue.

↳ Ready queue → Here when the proces got the Job and wants to execute they put in a Ready queue and keep them waiting for cpu to free for execution.

↳ Context switch

↳ First one proces is executing and it is using cpu. and suddenly an Interupt occured and it wants the cpu. → Here we have two cases. one is if the proces is less important then the current proces. then it will be waiting in the ready queue and second is if the proces is highly Important then the currectly running proces has to save the Context and it has to swap with new proces that is Important and after procesing is done then the old proces that has stopped has to Resume. and It will restore.

↳ This task is known as Context switching.

↳ save & restore.

↳while switching, the cpu won't do any work.

↳ It speed of switching varies from machine to machine.

↳ How process is created ?

↳ A processes are created by create-process system call during the execution time.

↳ Here we have Two types of process.

↳ parent process

↳ A process that create a another process is called parent process

↳ child process

↳ A process that is created by a preant process is called child process.

↳ This can grow like a Tree, any process can create another child process.

↳ when a process creates a new process then their exists a Two possibilities.

① execution time :-

↳ The parent continue to execute simultaneously with the child process.

↳ The parent continue its execution after child process completes.

② address space of process :-

↳ The child process is dupliate to parent process.

↳ The child process has new program.

↳ How processes are terminated.

↳ The process terminates by calling exit() system call after completing the execution.

↳ After terminating the process will return to it parent by a wait() system call.

↳ After terminating the process deallocates the resources which are allocated during the execution.

↳ And parent process can terminate the child process by a system call.
(That can be done by only parent process.

↳ Reasons for terminating the child process is
(1) It excessly using the resources.
(2) If task assigned to it completed or no longer required.
(3) If the parent is terminated then child should terminate.

↳ Inter process communication (IPC)

↳ The way of communication between the processes is known as IPC

↳ In OS if two processes are executing simultaneously then those processes can be

↳ Independent processes.

↳ Cannot effect (or) effected by other process.

↳ Cooperating processes

↳ can effect (or) effected by other process.

↳ If the Two processes share same data then they will effect each other.

↳ why processes has to cooperate :?

↳ Information sharing (user wants common work)

↳ Computation speed up (divide & conquer)

↳ modularity (dividing modules for solving. and finally we will combine them & for combining we use common processes)

↳ convenience (easy for user).

↳ For Inter process communication we have two fundamental communications.

(1) shared memory → A common region of memory is shared b/w processes.

(2) message passing model → By sending/receiving message b/w processes.

↳ shared memory system:

↳ A common shared memory region is required for communication b/w the processes.

↳ If their exists A processes 'A' and 'B' then, if process 'A' wants to communicate it writes in the shared memory. and B will Read from that shared memory.

↳ This shared memory will resides in the address space of the processes,

↳ The shared memory should create & attach ~~should be~~ is decided by the respective processes.

↳ message passing system:

↳ IPS mechanism processes must communicate and synchronize their actions by sending & receiving messages without any shared memory.

↳ By send message & receive message.

↳ message can be fixed & variable size.

↳ system level operation are Implemented

↳ programmers tasks are very simpler.

↳ For sending and receiving message we need a logically Implemented communication link.

For that we have.

↳ Direct & indirect Communication
↳ synchronous & asynchronous communication.
↳ Automatic & explicit buffering.

↳ Direct Communication is done by name of the processes.

↳ In Direct Communication is done by the common mailboxes & ports to the processes.

↳ synchronous communication
↳ The sending process will be blocked until receiver receive message
↳ The receiver process will be blocked until message is available.

↳ asynchronous communication
↳ The sending process will not be blocked.
↳ The receiving process will not be blocked.

↳ 2050 capacity Buffer.
↳ Buffer acts as path b/w the sender & receiver.

↳ Bounded capacity Buffer.
↳ Buffer is of finite length and if the Buffer is full sender should wait until the space is available to fill.

↳ Un Bounded Buffer capacity.
↳ The Buffer has infite length.

↳ Sockets:
↳ These are used for communication in client-server system
↳ Socket is an end point of the communication.
↳ A socket is identified by an Ip address and with port number
↳ server port no should be greater than 1024
http → 80
ftp → 21
telnet → 23.

Threads

↳ A thread is a Basic unit of cpu utilization.

↳ A executing program has multiple process and process consists of multiple threads and thread is Basic unit in it.

↳ A thread consists of → Thread ID
→ program counter
→ register set
→ stack.

→ If the threads consists of same process then it shares code section, data section, other OS resources such as files and signals.

↳ if a process has single thread then it cannot perform more than one task at a time.

↳ if a process has multiple threads then it can perform more than one task at a time.

↳ Benifits of multithreaded programming.

① multiple threads works simultaneously and multiple tasks will be done and user will, don't wait for one task to complete..

② Resources sharing is happening in the multi threading.

③ It is more economical to create and use.

④ In multiprocessor system the multiple threads will be more helpfull to complete the tasks very quickly. cpu is used efficently.

↳ Types of threads :

↳ user threads.
↳ which are handled by the user.

↳ kernel threads
↳ which are handled directly by the OS.

↳ connection b/w user thread and kernel thread is established in three different ways.

① many to one model.
↳ many user level connects to one kernal thread.
↳ It is very efficient because thread library is manages the threads in user level.

limitations.

↳ The entire process will be blocked if one thread makes a blocking system call.

↳ Only one kernel thread can run in only processor at a time.

one to one model :-

↳ Each thread is connected each kernel thread in a processor.

↳ if one thread is blocked then it won't effect other threads in the processes.

↳ multiple threads can run in multiple processor at the same time.

limitations :-

↳ we have to create kernel thread for every user thread.

↳ The Applicatn has to restrict the no of threads that are supported because performance decreases.

(3) many to many model.

↳ many user threads are connected to. smaller @ equal to the no of kernel threads.

↳ The no of kernels are specific for the application.

↳ Here user threads can run parallel on multiprocessor.

↳ if one thread is blocked then other can take over.

Hyper threading:

↳ The physical cores of the processors logically divided into multiple processors so that multiple threads can be supported at same time.

Fork system call:

↳ The fork system call is used to create a separate duplicate process.

exec():

↳ when exec system call is called then it will replace the entire process including threads with another process.

Threading Issues:

↳ if one thread in a program call fork() then the new process will duplicate all threads but in some UNIX systems it only duplicate the invoked thread.

↳ if exec() is called immediatly after forking then duplicating only the calling thread is appropriate.

↳ if exec() is not called immediately after forking then process should duplicate all threads.

↳ if thread is cancelled before its task completion then it is called terminating.

↳ if a thread that is to be cancelled is called target thread.

↳ we have two types of cancellation.

1) Asynchronous cancellation.

↳ A thread will ~~cancel immediately~~ terminate

↳ A target thread is terminated immediately by another thread.

2) Deffered cancellation.

↳ A target thread will terminate by itself in an orderly fashion.

↳ deffered cancellation is much better cancellation because. It is very safe and Resources will be deallocated safely.

↳ main() & runner function are used to create and control threads.

↳ CPU Scheduling :

↳ Here CPU is scheduled b/w the processes by OS. (By switching the CPU among the process)

↳ Here In single processor system, only one process can run at a time. If other process wants the cpu then it has to wait. still cpu is free.

↳ And In multiprocessor system if one process is running and if it wants to execute by I/O resource then it has to wait.

↳ So, At that time cpu is not doing any work.

↳ But our goal is to have some process has to run all the time so that the cpu is utilized properly.

↳ Here ↳ waiting time is wasted.
↳ If Important process wants cpu then that should wait until the cpu is free.

↳ So that why cpu scheduler came to use in the OS.
↳ It will keep all the process in the memory at the time.
↳ If one process is waiting then cpu scheduler allocate the cpu to other process at that time.

↳ cpu burst

↳ when a proces is begin. Its execution then it is cpu burst

↳ I/o burst

↳ when a proces is waiting for input or output operation it is I/O burst.

↳ processes will alternate b/w these two states.

cpu scheduler:

↳ If the processes are the ready queue then cpu scheduler selects the proces which are in ready queue and assigns the cpu to them, and increases the cpu utilization and cpu won't be idle.

Dispatcher:

The dispatcher is the module that gives control of the cpu to the proces ~~selected~~ selected by the cpu scheduler.

↳ Time taken by the dispatcher to switch the cpu from one proces to another is dispatch latency.

↳ preemptive scheduling.

↳ Here processes are not disturbed until termination (or) if they are ~~in~~ ~~waiting~~ switching from waiting state to

---

↳ what are reasons for making cpu scheduler to take decisions?

↳ if process switching from running state to waiting state.

↳ if process switches from running state to ready state

↳ if process switches from waiting state to ready state

↳ if process terminates.

↳ preemptive scheduling take place if the process switches from running state to ready state (or) waiting state to ready state.

↳ non preemptive scheduling take place if the process switches from running state to waiting state (or)

~~if process terminates~~ if process terminates.

↳ non preemptive scheduling is more better then preemtive scheduling.

Scheduling Criteria depends these terms ?

- ↳ Cpu utilization.
  - ↳ we have to keep cpu as busy as possible.
- ↳ Trough put:
  - ↳ work done by the number of process per unit time.
- ↳ Turnaround time.
  - ↳ sum of times that takes by the processes waiting to get into memory + waiting in the ready queue + executing on the cpu + doing I/O operation.
- ↳ waiting time.
  - ↳ time spent waiting in the ready queue.
- ↳ Response time
  - ↳ time from the submission of a request until the first response is produced.

First-come, First-serve scheduling Algorithm. (FCFS)

↳ The process that requests the cpu first is allocated the cpu first.

↳ In this algorithm the average waiting time is very long.

↳ Here if the processes having less burst time and if they requests cpu first then the average waiting time will be less.

↳ waiting times depends on burst time.

↳ FCFS algorithm is non preemptive.

limitation:-

↳ If the processes has higher burst time and if they arrive first and they will take more time and other processes after that has to wait for long time.

↳ In time sharing system the processes has to share the cpu frequently because the user will won't face any lag. but here it is not possible.

Turn Around time = Completion time - Arrival time.

waiting time = TAT - Burst time

Shortest - Job - first

↳ Here the processes with least burst time is assigned cpu first.

↳ If the cpu burst time are same then FCFS scheduling is used.

↳ It can be preemptive or non preemptive.

↳ Here scheduling depends on the length of the next cpu Burst of a process.

limitation:

↳ knowing the length of the next cpu burst of the process.

↳ There is no way to know the length of the next cpu burst.

waiting time = Total waiting time - burst time that executed - Arrival time.

Efficiency = useful time / Total time.

priority scheduling

↳ A process with the highest priority get the cpu first.

↳ cpu burst inversly proportional to the priority.

↳ priority scheduling can be preemptive or nonpreemptive.

limitation :-

↳ A low priority process will never get the cpu because higher priority processes will prevent them from getting the cpu.

↳ To solve this problem we will gradualy increase the priority of processes that wait in the system for long time, this technique is called aging.

↳ Round Robin scheduling :-

↳ It is designed for time sharing systems

↳ It similar to FCFS but preemptive.

↳ Have time quantum is assigned to every processes.

↳ Scheduler will allocate the cpu to each process for time quantum interval and then it interrupt and gives control to next process.

↳ we have Two scenarios :-

↳ if cpu burst time is less then the time quantum. then it will be execute till burst time completes and volentarly terminates.

↳ if cpu burst time is greater then the time quatum then scheduler interrupts the os to exit the proces and put that proces at the end of the tail of queue. and cpu will move to next.

↳ multilevel queue scheduling Algorithm :-

↳ Two types of processes

↳ fore groun proces (user interactive)

↳ Back groun proces (Batch)

↳ foregroun proces > backgroun proces.

↳ This Algorithm will divide the foregroun proces to one side and Backgroun proces to other side.

↳ Dependly on the proces type the proces will be assigna to different queue by memory size or proces priority.

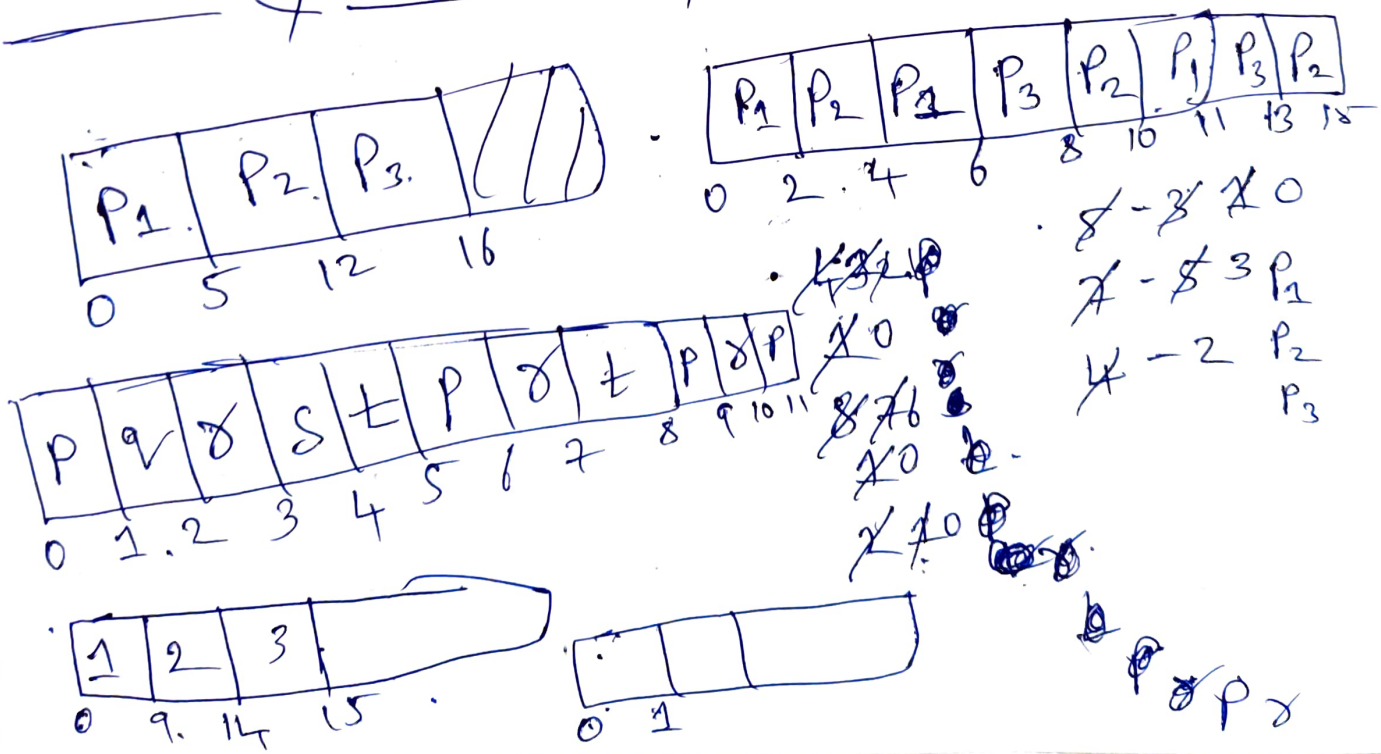
↳ multilevel feedback - queue scheduling Algorithm

↳ This Algorithm allows a proces to move b/w queues.

↳ Here if proces uses too much of CPU time then it can be moved to a lower priority queue.

↳ if the proces waith too long in a lower priority queue may be moved to higher priority queue.

↳ The processes belonging to the queues of lower priority will be executed only when the higher priority queue is empty.



process synchronization

↳ From Ipc we know that when two process execute simultaneously then those process are may be.
Independent process (or) cooperating process.

↳ process can affect (or) can be affected by other processes.

↳ These can directly share a logical address space. (both code & data)

↳ allowed to share data only through files (or) messages.

↳ If two (or) process concurrently access to shared data at time then that may lead to data inconsistency.

↳ For this problem we need synchronization.

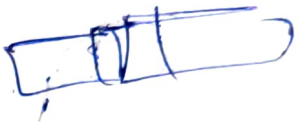
↳ Here cooperating processes execute orderly to maintaine inconsistency.

↳ To run concurrently we must have available a buffer of items and this buffer will reside in a region of memory. (The producer and consumer must be. (shared b/w them) synchronized)

↳ Race condition :

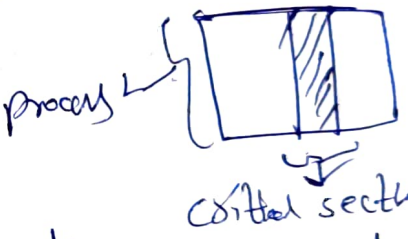
↳ If several processes access and manipulate the same data concurrently and the outcome of execution depends on the particular order in which the access takes place. is called.

Race condition.



↳ Critical section problem :-

↳ It is a segment of code of a process in which it is responsible for changing the data in a shared region memory that shared b/w the different process.



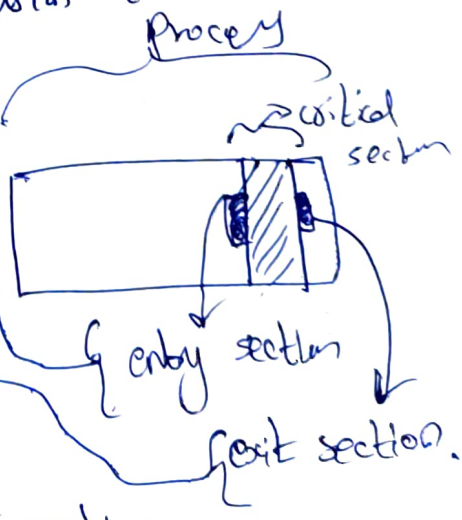
↳ Here if one process is executing in its critical section, no other process is to be allowed to execute in its critical section.

↳ Each process must request permission to enter its critical section.

↳ section of code giving the permission to enter.

↳ section of code giving the permission to exit.

↳ The remaining code is remainder section.



↳ To know the solution for critical section problem.

1) mutual exclusion.

↳ if process P_i is executing in its critical section then no other processes can be executing in their critical section.

2) progress.

↳ if no process is executing in critical section and not executing in their remainder section can be take the decision on which will enter its own critical section next.

3) Bounded waiting.

↳ if one processes has requested first to access the critical section but this process not accepting the request and if other process come and have permission to access, then the initial process will not get the chance to access so this leads to the starvation. so that there should be a bound or limit to enter there. critical section. so that all can access.

↳ These three requirements met the solution.

peterson's solution : (Turn and flag) (humble)

↳ peterson's solution is software based solution to the critical section problem which provides the requirements of mutual exclusion, progress and bounded waiting.

↳ Consider a two process P_i & P_j and we have two data items to be shared to get the solution

↳ It has two data items to be shared b/w the two processes.

↳ ~~turn~~ ~~If Int turn~~ is ~~equal~~ True ~~then the~~

~~P_i should be ready to enter~~

↳ turn := If int turn is equal to 'i' then its P_i turn to enter its critical section.

↳ flag := if boolean flag [i] = True then P_i should be ready to enter its critical section.

P_i

```
do{ flag[i]=true
    turn := j
    while (flag[j] && turn==[j]);
    critical section.
    flag [i]=false
    Remainder section
} while (True)
```

P_j

```
do { flag [j] := true
     turn = i
     while (flag [i] && turn==[i]);
     critical section.
     flag [j] = false
     remainder section
} while (True).
```

- Test and set lock. (washroom)

↳ It is hardware solution to the synchronization problem.

↳ There exists a shared lock variable between the processes which can be either 0 or 1.

0 → unlock . 1 → lock.

↳ First every process has to inquire about the lock.

↳ if lock is 1 then the process has to wait until it become 0 to enter critical section

↳ if lock is not locked then it will take the lock and assign the lock to 1 and executes in the critical section.

↳ when the lock value is '0' then the process is allowed to enter its critical section and do its execution and while execution the lock value is set to one hence it is not allowing any other proces to critical section until execution completes.

Function

```
boolean TestAndSet(boolean *target){
boolean rv = *target;
*target = TRUE;
return rv;
}
```

Algorithm

```
do{ while(TestAndSet(&lock));
// Critical section.
lock := FALSE;
// remainder section.
}while TRUE;
```

Semaphores (wait & signal) (positive & class)

↳ We found a hardware solution for critical section. but It is difficult for application programmes to implement to avoid this problem we have software based solution to the synchronization.

↳ It is a technique to manage concurrent process by using a simple Integer value. and It is Introduced by Dijkstra.

↳ semaphore is simply a variable which is non-negative and shared b/w threads. and A semaphore 's' is a integer variable that is accessed only by wait() and signal().

↳ wait() ⇒ P ⇒ To test the condition and enter critical section

↳ signal() ⇒ V ⇒ To increment and signal other processes about that critical section is free.

↳ when one process modifies the semaphore value then no other process can simultaneously modify that same semaphore value.

wait()

```
P(semaphore S) {
    while (S<=0);
    S--; enter critical section
}
```

signal()

```
V(semaphore S) {
    S++;
}
```

↳ Types of semaphores :-

↳ Binary semaphore :-

↳ The value of binary semaphore can range btw 0 & 1.

↳ Some systems binary semaphores are known as mutex locks. locks that provide mutual exclusion.

↳ Counting semaphores :-

↳ It is used for multiple processes Its value can range over an unrestricted domain.

↳ Dis-Advantages of semaphores :-

↳ processes will wait for so long. (stuck in while loop)

↳ Busy waiting wastes cpu cycles, that some other process might be able to use productively.

↳ This type of semaphore is also called spinlock (spins in while loop waiting for the lock.

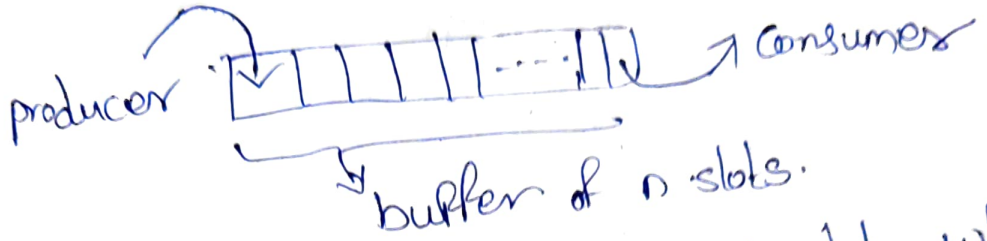
↳ To overcome busy waiting, the process can block itself and moves to waiting queue. and state is changed to waiting state.

↳ Then dispatcher will selects the process in waiting queue and again assigns cpu.

↳ This whole implementation of semaphore with waiting queue may results Deadlock & starvation.

The Bounded Buffer problem (producer consumer problem)

↳ There is a buffer of n slots and each slot is capable of storing one unit of data.



↳ The producer must not insert data when the buffer is full.

↳ The Consumer must not remove data when the buffer is empty.

↳ producer & consumer should not insert & remove data simultaneously.

↳ For this problem we use three semaphores.

↳ m (mutex) : binary semaphore which is used to acquire and release the lock.

↳ empty : A Counting semaphore whose value is the number of slots in buffer.
empty.

↳ full : A Counting semaphore whose initial value is "0".

Producer

```
do { wait (empty);
     wait (mutex);
     // add data
     signal (mutex);
     signal (full);
} while True
```

Consumer

```
do { wait (full);
     wait (mutex);
     // remove data
     signal (mutex);
     signal (empty);
} while True.
```

↳ The Readers writers problem;

↳ A data is shared among several concurrent processes.

↳ Here some process will read & some process will read & write.

↳ If two processes will only read then they are readers.

↳ If processes will both read & write then they are writers.

↳ If two process reads data at same time then their will be no effect in data.

↳ If two process writes data & one process reads & one proces writes also effects the data. and readers can not read.

↳ So this type of problem we require this Algorithm.

↳ for this problem we use three semaphores.

↳ mutex :- A semaphore which is used to update read count.

↳ wrt :- A semaphore common to both reader & writer processes.

↳ readcount :- An integer variable that keeps track of how many process are currently reading.

writer process :

```
do { wait (wrt);
   // it writes
   signal (wrt);
} while (True)
```

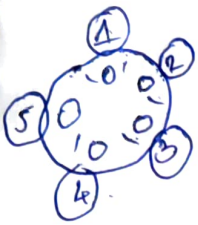
Reader process :

```
do { wait (mutex);
     Read cnt++;
     if (readcnt == 1)
        wait (wrt); -> no writer should enter
        signal (mutex) -> other reader can enter
     // current reader performs reading.
     wait (mutex);
     readcnt--;
     if (readcnt == 0)
        signal (wrt); -> writer can enter
        signal (mutex); -> reader leaves
   } while (true).
```

↳ This ensures mutual exclusion is satisfied in this problem.

↳ The Dining philosophers problem:

↳ IN This problem we have philosophes and forks and they can be in two state one is thinking and eating state. while eating they have to make use forks (2) so that they can eat properly.



↳ And if philosopher have limited forks so at when he hungry they have to pick up only one fork at a time. and other should take fork from the other's hand.

↳ so here forks are the resources and philosophes are processes. we have limited resources so they have to share the resources b/w processes in a synchronize manner.

↳ only one processer has acces to have same resources.

↳ philosopher grab a fork by executing wait() operation

↳ if philosopher releases his fork. the it can be done by excecuting signal() operation.

↳ The shared data are semaphore chopstick [5]

```
do {
    wait(chopstick[i]);
    wait(chopstick[(i+1) % 5]);

    // eat
    signal(chopstick[i]);
    signal(chopstick[(i+1) % 5]);
    // think
} while (TRUE);
```

↳ we can solve this problem by not allowing the processes (philosopher) adjacent to them to grab the fork at same time or they can not have the fork.

↳ If all the processes will grab the fork simultaneously then it could lead to a deadlock. (no one can eat and they will wait & leads to starvation).

↳ Allow at most 4 processes to grab the fork.

↳ Allow processes to pick if are only if the both forks are available.

↳ Add processes have to pick up left fork and even processes has to pick up right forks so that we can prevent deadlock.

monitors:

↳ It provides high level abstraction of convenient & effective mechanism.

↳ It preset a set of programmer-defined operations that provide mutual exclusion.

↳ It contains declaration of variables and functions that operate on these variables.

↳ monitor can have access over the locally declared variables.

↳ local variables can be accessed only be local procedures.

↳ monitor ensus that one process at a time is active.

~~↳ It can wait() & signal() monitor operations.~~

be use ~~on the con~~

Condition construct: (condition x, y);

↳ wait() & signal() operations can only be used on condition variables.

↳ x.wait(): operation mean the process invoking this operation and it should be waiting until another process invokes x.signal().

↳ x.signal() operation resumes one waiting process:

↳ Dining philosophers solution using monitors :

```
monitor dp {
    enum { Thinking, Hungry, Eating } state[5];
    condition self[5];

    void pickup(int i) {
        state[i] = Hungry;
        test[i];
        if (state[i] != Eating)
            self[i].wait; }

    void putdown(int i) {
        state[i] = Thinking;
        test((i+4) % 5);
        test((i+1) % 5); }

    void test(int i) {
        if ((state[(i+4) % 5] != Eating) &&
            (state[i] == Hungry &&
            (state[(i+1) % 5] != Eating)) {
            state[i] = Eating;
            self[i].signal(); }

    initialization_code() {
        for(int i=0; i<5; i++)
            state[i] = Thinking; }
}
```

Deadlocks.

↳ In multiprogramming environment, several processes have to compete for a limited number of resources.

↳ If a process requests & and if resource are not available at that time the process enters waiting state.

↳ This waiting process never gets resources and, it will be waiting for ever because those resources are held by other waiting processes which are waiting for other resources ; so these Each other waiting for each other resources and this may lead deadlock.

↳ system model :-

↳ A system consists of finite number of resources to be distributed among a number of competing processes.

↳ These resources are portition into several types, these resources consisting of some number of identical instance.

↳ Here each Instance can be used by each processes.

↳ process has to utilize the resources in only the squence.

1) Request ⇒ process has to request for cpu. If the request is cannot be granted immediatly than it has to wait util it gets permision to accesses.

2) Use ⇒ The process can operate on the resource.

3) Release ⇒ The process releases the resource.

↳ Deadlock characterization :

↳ Deadlocks can arise by the following four conditions hold simultaneously :

1) mutual exclusion,
   ↳ only one process at a time can use the resource. so deadlock occurs.
   ↳ if other process request resource then requesting process must be wait until the resource is free.

2) Hold and wait :
   ↳ process must be holding at least one resource and waiting to acquie additional resource that are held by other processes cosrently running.

3) No preemption.
↳ If the resource released only voluntarily by the holding process then deadlock occurs.

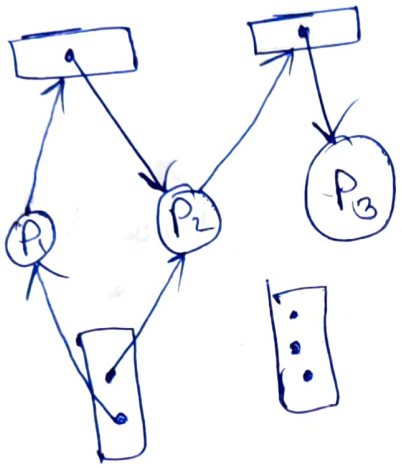
4) circular wait:
↳ A circular waiting may leads to deadlock.
Ex:- RR.

↳ Resource - Allcation graph.
↳ The graph consists of a set of vertices V and edges E.
↳ Set of vertices V.
↳ $P = \{P_1, P_2 -- P_n\}$ → active processes.
↳ $R = \{R_1, R_2 -- R_m\}$ → all resources types.
↳ set of edges E
↳ Request Edge.
↳ A directed edge from process P_i to resource type R_j : $P_i \rightarrow R_j$ (instance of Resource R_j) and is waiting for the resource.
↳ Assignment Edge.
↳ A directed edge from instance of Resource R_j to P_i which has been allocated to.
process P_i. $R_j \rightarrow P_i$

↳ process are represented in circles ○

↳ Resource are denoted using rectangles [▪▪▪] and dots are instances.

Graph



↳ P = {P1, P2, P3}

↳ R = {R1, R2, R3, R4}

↳ E = {P1 → R1, P2 → R3, R1 → P2, R2 → P2, R2 → P1, R3 → P3}

↳ If cycle exists in the graph then their is chance is occur deadlock.

↳ But it is not conform that deadlock is occur only cycle exists, sometimes cycle can be broken.

↳ methods for Handling Deadlocks.

↳ we have three ways to deal with deadlock.

(1) we can use some protocols and prevent (or) avoid deadlocks at first place. Ideal.

(2) we can allow the system to enter a deadlock and detect it and recover.

(3) we can ignore the problem altogether and pretend that deadlocks never ~~ever~~ occur.

method ①

↳ To ensure deadlocks never occur, the system can use either a deadlock prevention or a deadlock avoidance.

↳ deadlock prevention ensures at least one of the necessary conditions cannot hold.

① mutual exclusion ② Hold & wait

③ no preemption ④ circular wait.

↳ deadlock avoidance.

↳ knowing Addition Information about processes that require resources during its lifetime than they can wait at first place.

method ②

↳ If a system does not employ deadlock prevention or deadlock avoidance then the system has to provide a detector and cure (recover) for deadlock occurance.

method ③

↳ If a system neither ensures that a deadlock will never occur nor provide a mechanism of deadlock detection & cure then the system may still in deadlock. so system performance may ↓, so we have to ignore and restart the system.

↳ Deadlock prevention :

↳ we ensure that atleast one condition cannot hold from ① mutual exclusion
② Hold & wait
③ no preemption
④ circular wait.

↳ mutual exclusion.

↳ shareable resources do not require mutually exclusive.
↳ non shareable resource require mutually exclusive because they cannot be shared simultaneously.

↳ Hold & wait.

↳ if the process requests a resource then it should not hold any other resources.

↳ protocol ①
↳ all resources has to allocate before its execution begins.

↳ protocol ②
↳ if process request a additional resource then it must deallocate one then it should request.

↳ But these two protocols leads to.

(1) Starvation of process

(2) Resources will be not utilized efficiently.

↳ No preemption :-

↳ If a process is holding some resources and requests another resource. and if these are held by another process then these resource should be preempted and allocate to the requesting process.

↳ Here if we preempt then we can only save the state of cpu register & memory space but we cannot save & recover some other resources such as printer, tape drives.

↳ circular wait :-

↳ Here we can ensure this condition by changing the order of all resource types.

↳ Each process requests resources in an increasing order of enumeration.

↳ Here $R_1 \longrightarrow P_1$ and if $P_1 \longrightarrow R_4 \& R_3$

here $R_4 \& R_3 < R_5$ so only if Resources $> R_5$ are

↳ Application developers should develop only quoted.

↳ Deadlock Avoidance

↳ Here we have to know the Additional information about the processes and how processes will request the resources. to Avoide deadlock.

↳ System should know in Advance about what processes will request what resources and when they require these processes in the future..

↳ Safe state:

↳ If a system allocate resource to each processes in some order and avoid deadlock then it is safe state.

↳ Safe sequence:

↳ If safe state is satisfied if only processes that are in that sequence then that sequence is known as safe sequence.

↳ A safe state is not a deadlocked state.

↳ A deadlocked state is an unsafe state.

↳ Not all unsafe states are deadlocks.

↳ The behavior of the processes controls unsafe states.

↳ Resource Allocation graph Algorithm.

↳ In this Algorithm we have new type of edge called claim edge.

↳ A claim edge $P_i \rightarrow R_j$ indicates that process P_i may request Resource R_j, at some time in the future.

↳ when process P_i request resource R_j the claim edge $P_i \rightarrow R_j$ is converted to a request edge.

↳ claim edge → request edge → Assignment edge.

↳ This will be done if they do not form cycle in the resource allocation graph.

↳ if no cycle exists then system will be in safe state

↳ if cycle exits then system will be in an unsafe state.

↳ Resource Allocatin graph Algorithm works only in single instance of Resource

↳ Banker's algorithm :-

↳ This algorithm works on multiple instances of Resources.

Available :

↳ : available Instances of each resource ~~each~~

~~process~~ Currently ~~holds~~ available in system.

↳ ~~available~~ · Available [i] = k (Instances)

Max :

↳ Instances of each resource each process can max request.

↳ Max [i, j] = k.

Allocation :

↳ Instances of each resource each process currently hold.

↳ · Allocation [i, j] = k.

Need :

↳ Instances of each resource need to each processes.

↳ · Need [i, j] = Max [i, j] - Allocation [i, j]

↳ Resource Request Algorithm:-

↳ Here also we have to apply Bankers algorithm.

↳ suppose now that Process P_1 requests one additional instance of resource type A and two instance type C, so $Request_1 = (1,0,2)$.

↳ Is it granted or not

↳ Here

↳ Available = Available - Request

Allocation = Allocation + Request

$Need_1$ = $Need_1$ - Request

↳ After Apply the Bankers algorithm on the new table and we have to check whether the new system state is safe or unsafe.

↳ We find that granting the request of process P_1 still keeps the system in a safe state and hence will not lead to a deadlock.

Deadlock detection :-

↳ In some cases we cannot prevent @ avoid .. Deadlocks so we have to do something.

↳ for that we have to detect and recove the deadlock.

↳ deadlock detection in single Instance.

↳ we use wait-for-graph which is an algorithm for deadlock detection. Here we obtain this graph from the resource allocation graph by removing the resource nodes and collapsing the appropriate edges.

↳ $P_i \rightarrow R_k$ & $R_k \rightarrow P_j \Rightarrow P_i \rightarrow P_j$

↳ wait-for graph used for detecting the deadlock and which should be formed after regular intervals of time. and it should check every time if their exists a cycle @ not.

↳ Deadlock detection for multiple instance.

↳ This algorithm employs several time varying data structures that are similar to those used in Bankers algorithm.

↳ Available.

↳ A vector length → the no of available resources

↳ Allocation.

↳ The current request of each process.

↳ Request.

↳ the current request of each process.

↳ If the resources are not sufficient then the system go to unsafe state and deadlocked.

↳ Deadlock Recovery :-

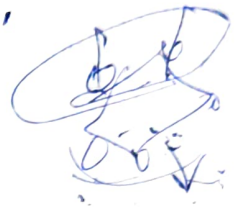
↳ After detecting the deadlock we have to inform the operator that deadlock has occurred and let operator deal with it.

↳ (or) Let the system recover from the deadlock manually.

↳ How to break from the deadlock.

① Abort one or more processes.

② preempt some resource from one or more of deadlocked processes.



↳ process termination:

↳ Abort all deadlocked processes.

↳ It may cost great expense.

↳ process may executed for long time.

↳ partial executed processes must discard & reexecuted from previous state.

↳ Abort one process at a time until the deadlock is eliminated.

↳ many times the deadlock detection algorithm has to perform → leads low performance.

↳ If the partial terminate method is used.

↳ we have to choose the deadlock.

↳ we have to choose the process which may cause minimum cost.

↳ How to choose process ? (factor).

↳ know the priority of process?

↳ know the completion time and know how long it will take to complete?

↳ know the resource that process has used?

↳ How many more resources the process needs in order to complete.

↳ How many processes will need to be terminated?

↳ check whether the processes are user level (or) batch level.

↳ Resource preemption :

↳ Here we preempt some resources from processes and give these resources to other processes until deadlock cycle is broken.

↳ For that we need to select victim and Rollback, if the process Rollbacks then sometimes leads to starvation. (1) selecting a victim

(2) Rollback

(3) starvation.

↳ selecting a victim.

↳ which resources and which processes are to be preempted?

↳ Determine the order of preemption to minimize cost.

↳ no of Resources a deadlocked process is holding.

↳ Amount of time. It consumed.

↳ Roll back.

↳ what should be done with a process if we prempt its resources.

As we must rollback the process to some safe state and restart it from that state.

↳ If the safe state is not known that we have to total rollback the process

↳ Abort the process and then restart it.

↳ If we prempt the resources then it cannot continue it normal execution because its resources are missing. So it has to roll back.

↳ starvation.

↳ if we prempt the same process again and again then that leads to starvation.

↳ how do we ensure that starvation will not occurs?

↳ we must ensure that process can be picked as a victim only a finite number of times.

(fix the no of rollbacks).

---

problems

	Allo A B C	Max A B C	Avail A B C	need
P_0	1 0 1	4 3 1	3 3 0	3 3 0
P_1	1 1 2	2 1 4	4 3 1	1 0 2
P_2	1 0 3	1 3 3	5 3 4	0 3 0
P_3	2 0 0	5 4 1	7 3 4	3 4 0

5 1 6

3 3 0

8 4 6

8 4 6

$T = 8\,4\,6$

$P_0 \rightarrow P_2 \rightarrow P_1 \rightarrow P_3$

R = 100

100 → deadlock

OS

↳ Memory management

- main memory
- virtual memory

↳ Computer system has to execute programs.

↳ These programs has to load in to the main memory (Ram) with the data they access.

↳ Ram is active when power is on.

↳ All the data will be store in the secondary memory when they are executed then they will be loaded into main memory.

↳ To use cpu efficiently then we use cpu scheduling, so that time many processes will be executed and loaded into the main memory.

So, we must share memory.

↳ If main memory size is more than cpu will be more fast.

↳ Here self → acts as secondary memory & books of books. as data.

↳ Table → acts as main memory & Here when user wants to read, they will load and execute.

↳ Basic hardware of main memory.

↳ memory consists of large array of words or bytes, each with its own address.

↳ cpu fetches instruction from memory according to the value of program counter.

↳ A typical Instruction execution cycle :

↳ first fetches an instruction from memory.

↳ second Instruction will be decoded.

↳ Third Instruction will be executed on the operands

↳ Forth results will be stored back in memory.

↳ cpu can directly access only on, Registers built into the processor and main memory.

↳ Instructions must take memory addresses as arguments.

↳ executions of Instructions must be in one of these direct-access storage devices.

↳ If the executions are not done in main memory then they must be moved there before cpu can operate.

↳ cpu cycle Time for accessing memories.

↳ Registers
  ↳ Access may take place one cycle of cpu clock to complete.

↳ main memory.
  ↳ Access may take place many cycles of the cpu clock to complete.
  ↳ In this case processor has to wait or incurs stall.
  ↳ for this we have cache memory which acts as memory buffer present in b/w cpu and main memory which is very fast to access.

↳ protection of os from unauthorized access.
  ↳ os has to be protected from access by multiple user processes.
  ↳ user processes must be protected from one another
    ↳ This will be done by hardware, By restricting the range of legal addresses of every processes to access data. and can access only that legal address called memory space.

↳ For this we use base & limit.

↳ logical address space. have base & a limit register.

↳ Base register holds the smallest legal memory address.

↳ limit register specifies the size of the range @ legal address range of individual process.

↳ Base & limit registers can only be loaded by os & these will use kernal instructions can be executed in kernel mode.

↳ Here os does not allow other user programs from changing the register contents.

Address Binding

↳ Every program resides on a disk as a binary executable file.

↳ so, To execute we must bring that executable file to main memory. that placed within a process.

↳ To manage the memory b/w process, the process has to move b/w disk & main memory.

↳ The process that are waiting to enter into main memory for execution will be in input queue.

↳ Input queue → loads into memory → execute → terminates
↳ memory space will be available

↳ user program has to compile → load → execute.
↳ Compile time (creates Interaction b/w as memory)
↳ load time (
↳ execution time (done by processor)

↳ Addresses are symbolic in source program.
↳ while compiling, the symbolic Address will be relocatable addresses.
↳ while loading, the relocatable addresses will be absolute addresses.
↳ These address binding are the mapping from one address space to another.

↳ Compile time → if compiler is responsible for performing address binding then it is called compile time.
↳ load time
↳ it will be done after loading the program.
↳ execution time → The program will be kept on changing the locations in memory until time of program execution.

↳ Logical Address : An address generated by CPU to process.

↳ physical Address : Address loaded into the memory address register of the main memory.

↳ At Compile time & Load time address Binding methods generate identical logical & physical addresses.

↳ At execution time address Binding generate two different logical & physical address.

↳ At this time we call logical address as virtual address.

↳ logical address space :-

↳ In this space the set of all logical addresses generated by a program.

↳ physical address space :-

↳ In this space the set of all physical address corresponding to these logical address.

↳ So At execution time logical & physical address space are different.

↳ The runtime mapping from virtual to physical address is done by a hardware device called memory management unit (MMU)

Dynamic loading:

↳ we know that every process must be in physical memory for the process to execute.

↳ The size of physical memory is limited, so, we cannot execute the processes of larger size.

↳ dynamic loading will help in obtaining better memory space. utilization.

↳ How dynamic loading works?

↳ A routine is called to physical memory.

↳ chunks of the program.

↳ All routines are kept on disk in a relocatable load format. Here one routine will execute only checks whether the other routine has been loaded or not, if not then calls with the help of relocatable linking loader (that passes control to newly loaded routine).

↳ Advantage of dynamic loading.

↳ unused routine is never loaded

↳ larger programs are handled by dynamic loading.

↳ This is a advantage to the users who wants to use dynamic loading.

↳ Dynamic linking & shared libraries.

↳ when program runs, it has its own modules to run and some times it also need system libraries.

↳ static linking.

↳ system libraries are treated has some other modules and they are combined with own modules in the program.

↳ This requirement wastes both disk & main memory space.

↳ dynamic linking.

↳ system libraries are loaded into main memory when routines are loaded.

↳ dynamic linking libraries.

↳ dynamic linking contains a stub (segment of code). That helps in locating a library routine and helps in loading it into main memory.

↳ stub contains the address of the system library routine and stub replaces itself with the address of the routine.

↳ Swapping.

↳ A process must be in memory to be executed.

↳ A process can be swapped temporarily out of main memory to a backing store and then after some time it will be brought back into memory and continue its execution.

↳ Ex: Round Robin cpu scheduling.

↳ Here backing store is a fast storage location in disk.

↳ Normally A process that is swapped out will be swapped back into the same memory space it occupied previously.

↳ Here it depends on the address binding methods.

↳ If binding is done at assembly or load time then process has same memory location.

↳ If binding is done at execution time then the process may be swapped into a different memory space.

↳ Requirement of backing store:

↳ It must be large enough to accommodate.

↳ It must provide direct access to memory.

↳ It should maintain ready queue.

↳ If process wants to execute in main memory then dispatcher will assigns the cpu to that process that is present in the backing store.

↳ If process has no free memory region in main memory to execute then dispatcher will swap out the process to backing store and swap in other process to main memory.

↳ Swap time:

↳ Time taken to swap the process from backing store to main memory & vice versa.

↳ Swap time will be both swap in and swap out.

$$\text{swap time} = \frac{\text{size of process}}{\text{Transfer rate of Backing store}} \times 2$$

↳ execution time should be greater than swap time.

↳ Swapping time $\propto$ size of memory.

↳ Factors affect swapping:

↳ A process must be idle in order to be swapped.

memory Allocation :

↳ How main memory is allocated ?

↳ Divide memory into several fixed sized partitions.

↳ Each partition may contain exactly one process.

↳ when partition is free, a process selected from the input queue and loaded into that free partition.

↳ when process terminates the partition becomes free @ available.

↳ Dynamic storage allocation problem :

↳ How to satisfy a request of size n from list of a free partitions in main memory.

↳ First fit.

↳ Allocate the first partition that is big enough.

↳ we can Allocate the partitions by searching from start either at the beginning of the set of partition @ where the previous first fit search ended.

↳ we can stop searching as soon as we find a free partition that is large enough.

↳ Best fit:

↳ Here we allocate the smallest partition that is big enough to fit the process.

↳ Here we have to search the entire list, and fit the process.

↳ Here we can see that we can minimize the main memory.

↳ Worst fit:

↳ Here we allocate the largest partition to process.

↳ Here also we has to search the entire list and fit the process into largest partition.

↳ From this we can see that Best fit is the Best solution for memory allocation problem.

↳ Fragmentation:

↳ When processes are loaded and from the main memory, the free memory space is broken into little pieces which results in fragmentation.

↳ External fragmentation :

↳ It exists when there is enough total memory space to satisfy a request, but the available spaces are not adjacent.

↳ That memory space is fragmented into large number of small fragments.

↳ This External fragmentation is more in worst fit.

↳ if the memory spaces are big & adjacent then they can be used by other processes.

↳ Internal fragmentation :

↳ It occurs when memory blocks assigned to processes are bigger than what the process actually need. At that time some portion of memory is left unused as it cannot be used by another process.

↳ So, that space will fragmented.

↳ solution for Internal fragmentation is to follow Best fit approch.

↳ solution for External fragmentation is to follow

• Compaction.

↳ Compaction :-

↳ we have to shuffle the memory contents so as to place all free memory together in one large block.

↳ It is not always possible.

↳ It is only possible at execution time.

↳ paging

↳ It is a solution for External fragmentation.

↳ paging is a memory management scheme that helps to combaine the physical address space that are not adjacent also.

↳ It means :

↳ By dividing the physical memory into fixed-sized blocks called frames.

↳ By dividing the logical memory into fixed-sized blocks called pages

↳ Here the pages are loaded into the frames even they are of large size.

↳ backing store is divided into fixed sized blocks that are of same size of frames.

↳ Page Table.

↳ How to know which pages are present in which frames.?

↳ With help of page table we are having the record of which pages are preset in which frames.

↳ How logical address generated by cpu is divided into two pages. (1) page number (p) (2) page offset (d)

page number (p) → index into a page table

page offset (d) → displacement with in the page (position)

↳ page table contains base address of each page and page offset is combined with base address. that define the physical memory address in the physical memory.

4) How to Translate of logical address to physical address. using page table.

((Frame x pagesize) + offset))

↳ Hardware Implementation of page table is

case ①
↳ dedicate every page table for every register.

case ②
↳ keep the page table in main memory, so that pagetable bus register can points to the pagetable.

↳ drawbacks :-
case ① ⇒ we need so many registers for large data.
case ② ⇒ The time required to access a user memory location is increased. (one for pagetable & one for the byte).

↳ solution for this drawbacks are to use (TLB) Translation Lookaside Buffer.
↳ TLB is high speed memory. It consists of two parts ① A key
② A value.
↳ TLB takes Input as item, if that item is matched with keys then corresponding value is returned.

↳ How to use TLB with page table ?

↳ The TLB contains only few of the page table entries.

↳ when logical address is generated by cpu, it page number is presented to the TLB.

↳ If the page number available in TLB then its corresponding frame number is available. This is called TLB Hit.

↳ If the page number is not available in TLB then its corresponding frame number is not available this called TLB Miss.

↳ If page number is not available then we add that page number & corresponding frame number. but Here we said TLB contains only few page table entries & if we want to add one and there is no more place to add.

↳ At that we just replace it with the least used page number and frame number of page table.

↳ Page table Entries :-

↳ It contains Information about pages and corresponding frame numbers.

↳ Information means :-

(1) Frame number.
↳ frame where the page is present.

(2) present/absent bit (valid or invalid bit)
↳ specifies whether the frame no is present or absent :
↳ If frame is not present then it is called page fault.
↳ present/absent is referred as 1/0.

(3) protection bit
↳ which control the read & write operations.
↳ bit is set to 0 means read
↳ bit is set to 1 means read/write.

(4) Reference bit
↳ specifies whether the page is accessed in the last clock cycle.
↳ accessed/not accessed means 1/0

(5) caching bit

↳ used to enabling & disabling caching.

↳ when we need new data then we disable it & we enable it.

↳ disable/enable means 1/0

(6) Dirty bit (modified bit)

↳ Specifies whether the page has modified & not

↳ modified/not modified means 1/0.

↳ shared pages

↳ Here In paging we can share common code.

↳ This is beneficial to time-sharing environment.

↳ If that shared code is reentrant code.

& pure code then it can be shared among processes.

↳ The data for every process will be different.

memory.

↳ Hierarchical paging

↳ modern computer system supports large logical address space (like 32 bit).

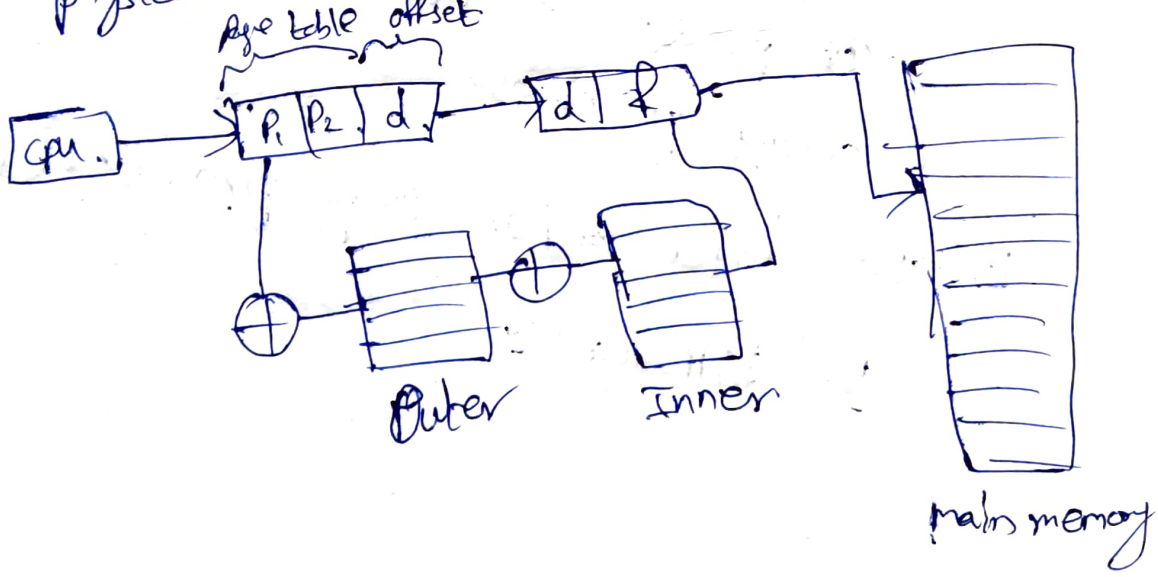
↳ so, page table becomes excessively large.

↳ so, for this problem we need to divide the page table into smaller sizes.

↳ Multilevel paging:-

↳ page table itself is also paged.

↳ Here page table divides into outer page table & Inner page table which combines gives the frame number and By combining the frame number and offset we get the physical address of the main memory.



↳ Hashed page Tables

↳ These are used for handling address spaces larger than 32 bits.

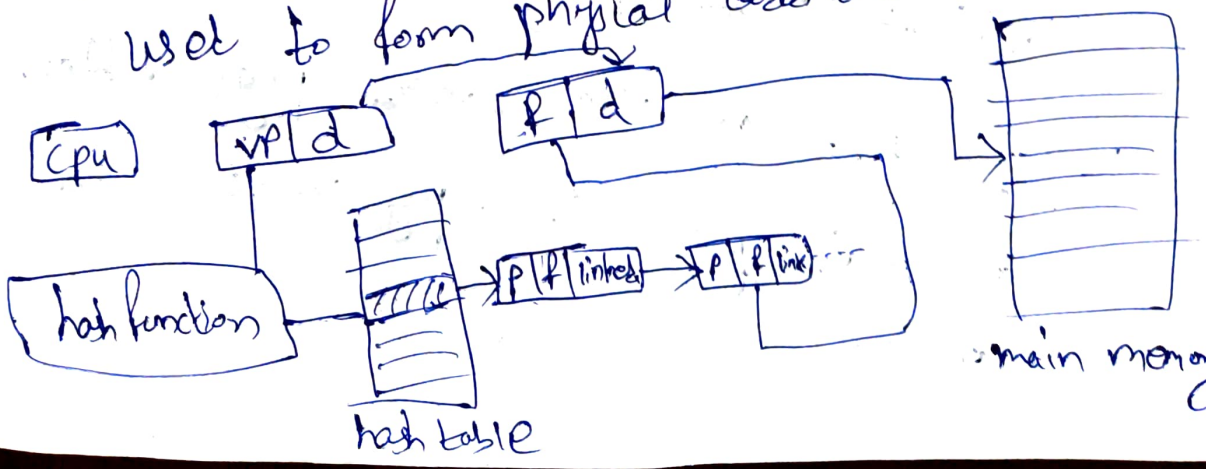
↳ The virtual page number in the virtual address is hashed into a the hash table.

↳ hash table contains virtual page number, corresponding frame number and A pointer to the next element in the linked list.

↳ The virtual page number in hash table is matched then frame number will be used to form physical address.

↳ If the virtual page numbr in hash table is not matched then that point the next element in the linked list.

↳ If that is matched then that will be used to form physical address.



clustered page tables :-

↳ This hashed page table is most favorable for 64-bit address.

↳ Here each entry in the hash table refers to several pages rather than a single page.

↳ clustered page tables are useful for sparse address spaces where memory references are scattered throughout the address space.

Inverted page Tables :

↳ A separate page table is maintained for each process.

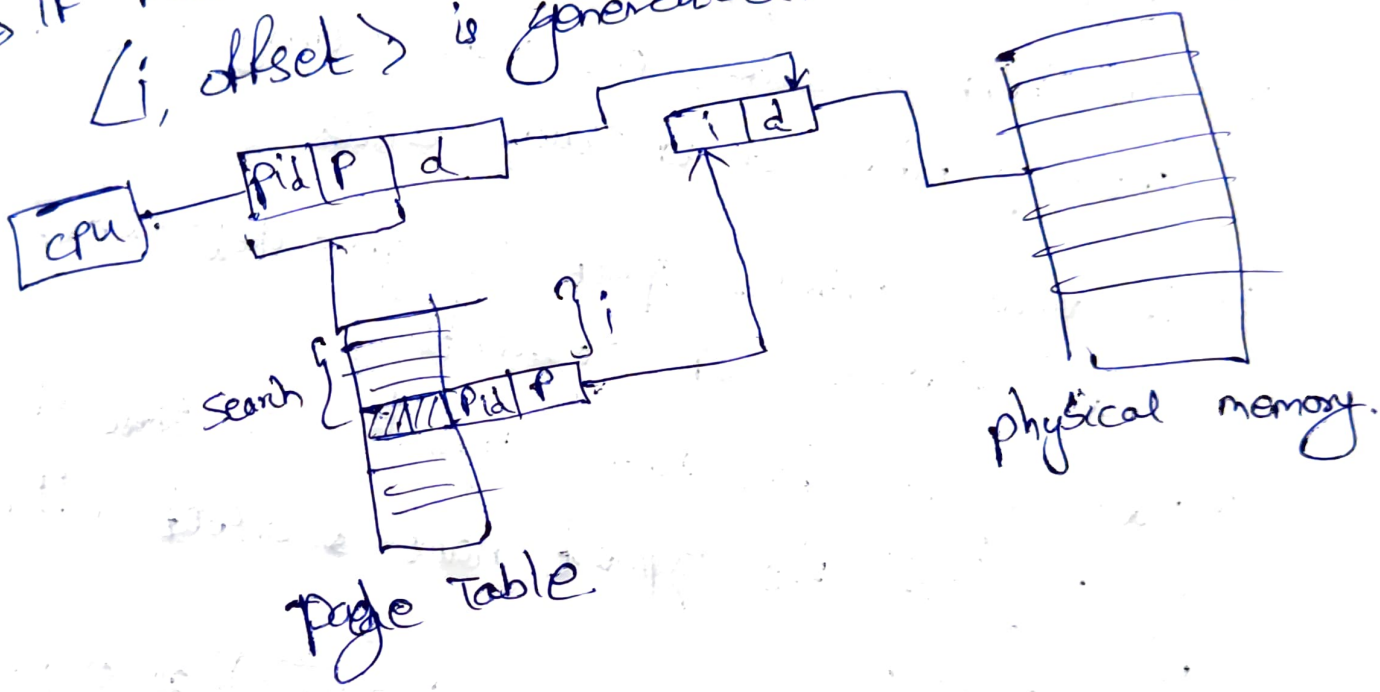
↳ So, for n no of processes we have n number of page tables.

↳ for large processes there would be many pages so we need lot of memory hence memory utilization is not efficient.

↳ So, for that we use inverted page tables. It contains process id, page number and offset.

↳ Page table contains process id and page number. Here inverted page table is then searched for a match.

↳ If match (i) is found then physical address <i, offset> is generated.



Advantages:

↳ Reduces memory usage.

Disadvantages:

↳ Search time increases.

↳ difficulty in implementing shared memory.

↳ segmentation:

↳ It is a non-contiguous memory allocation technique (not adjacent)
like paging.

↳ Here the processes are divided into several modules called segments.

↳ Here both secondary and main memory are divided into partitions of unequal sizes.

↳ A logical address space is generated which is a collection of segments.

↳ Each segment has name & length. and these addresses specify both the segment name & offset within the segment.

↳ For simplicity segment name is referred as segment number.

segment number	offset.

↳ segment Table:

↳ In segment table we map the two dimensional user-defined addresses into one dimensional physical addresses.

↳ Each entry in segment table has segment base and segment limit.

↳ segment base contals the starting address of the physical address.

↳ segment limit contals the length of segment.

	base	limit
seg0	1400	400
seg1	1700	300
seg2	1520	600
seg3	2000	800

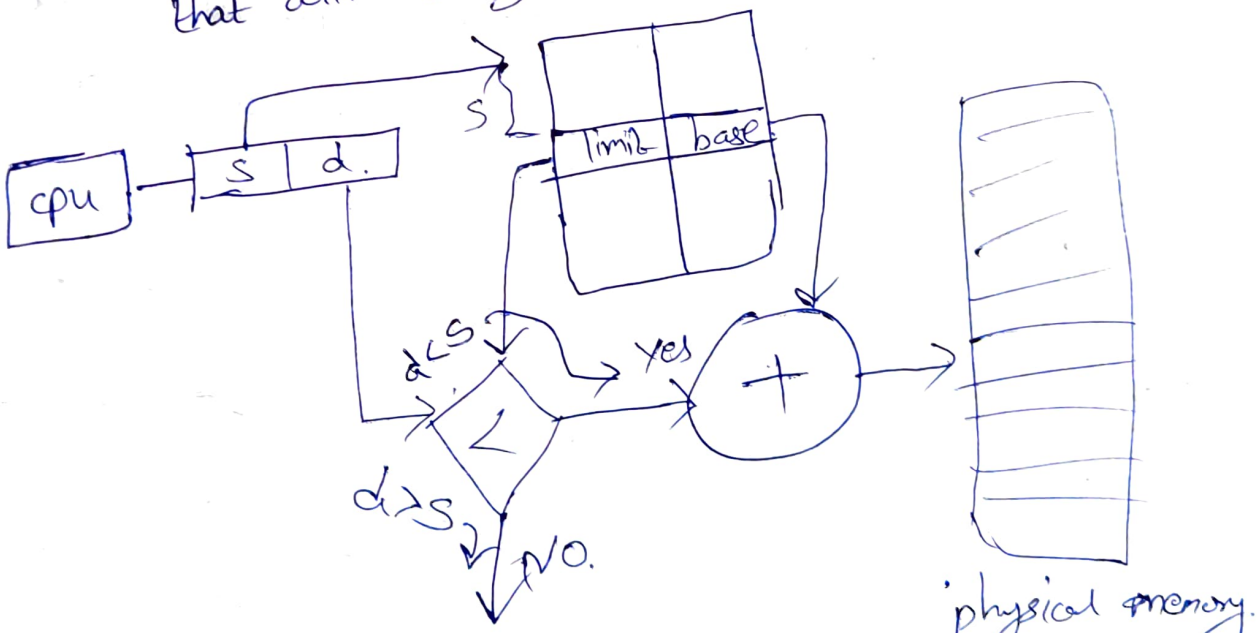
↳ segmentation Hardware.

↳ A logical address consists of two parts.

① segment number 's' → Index of segment table

② offset (d) that must be b/w 0 & segment limit

↳ If the offset greater then segment limit then that will thrugh error.



Address error

File system

↳ programs has to be loaded to main memory for execution.

↳ Here main memory is too small to accommodate data so Computer system has secondary memory. has a backup.

↳ so, we use disks as secondary storage device.

↳ Here file system provides the mechanism for

↳ storage of data & programs on the disk.

↳ Access to data and programs on the disk.

Files :- A collection of related information that is recorded on secondary storage which is defined by creator.

↳ Different storage devices vary in many aspects.

Eg: devices transfer a character or block of characters. at a time.

↳ some can access sequentially & randomly

↳ some can access synchronously & asynchronously.

↳ some can access read only & read-write.

↳ That's file system provides the functionalities & access to data on the disk.

↳ Files.

↳ we have different types of storage devices.
Ex: magnetic disks, magnetic tapes, optical disks etc.

↳ The files are mapped by the OS onto physical devices.

↳ These physical devices are non-volatile.

↳ files represent both data and programs.

↳ file is a sequence of bits, bytes, lines (or) records.
data can be numeric, alphabetic (or) binary..

↳ Information like, source program, text, video Recording, graphic images etc may be stored in a file.

↳ File Attributes:-

↳ Naming a file:-

↳ A file is named for convenience of its human users.

↳ name is string characters Eg: myfile.c

↳ when a file is named it become independent of process, users and systems.

(1) Name :-
↳ A name (characters) is the information kept in human readable form.

(2) Identifier :
↳ It is unique tag number with in the file system.
↳ It is non readable by human.

(3) Type :- ↳ information needed for systems that supports different types.

(4) Location :- ↳ Information is a pointer to a device that locates the file.

(5) size :- ↳ current size of file (bytes)

(6) protection :-
↳ Access-control Information will be present

(7) create data :
↳ The date on which the file is created

(8) Last modified data :-
↳ The date on which the file was last modified.

(9) owner :-
↳ Information contain about who is the owner of the file.

↳ These all attributes of the file are stored in the File control Block (FCB)

↳ File operations :-

↳ OS provides system calls to perform various file operations.

↳ creating a file.
  ↳ check if space is available in file system to create file.
  ↳ make entry for the new file in the directory

↳ writing a file.
  ↳ we make a system call specifying both the name of information to be written in a file.
  ↳ The file system searches for a file in directory & locates it
  ↳ A write pointer is maintained which points to the location in the file, so that next write takeplace.
  ↳ After writing is done, the write pointer is updated.

↳ Reading a file.

↳ We make system call specifying both name and where the next block of the file should be read.

↳ File system searches for file in a directory.

↳ A read pointer is maintained which locates the point in the file from where the next read must take place.

↳ After reading is done then read pointer is updated.

↳ Repositioning within a file (File seek)

↳ The directory is searched for the appropriate file.

↳ The current file-position pointer is repositioned to a given value.

↳ For that we do not need any actual I/O

↳ Deleting a file.

↳ The directory is searched for the named file

↳ After directory is found then file space occupied by file is released. & deleted.

↳ Truncating a file.

↳ Here if the user wants to erase the contents of file but keeps its attributes.

↳ Here we keep all the attributes of the file.

↳ we reset the size of file.

↳ Release the file space.

↳ To avoid this constant searching, many file systems require an open() system call be made if the file is first used actively.

↳ Here OS keep a small table called open-file table contals information about all open files.

↳ File types :-

↳ file systems must be able to recognize the file types.

↳ Here extention of file specifies the file type. while helps OS to recognize. Ex: file.txt

↳ In windows we have many types of file types.
Ex: pdf, .c, ---, .zip, .lib, .a, .exe ---- etc.

↳ In mac os :-
↳ Here file has a type specified.
Ex: For text → TEXT ---- etc

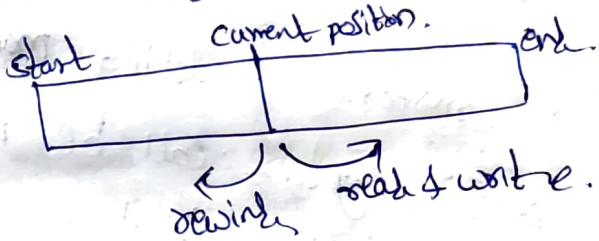
↳ In unix systems :-
↳ Could magic number at beginning of files.

↳ Access methods:-

↳ Here we know that the Information in files can be accessed in several ways.

(1) Sequential Access

↳ Information in the file is processed in order one record after the other. Ex:- compilers, editors -- etc.



↳ Read :- (read next)

↳ reads the next portion of the file - and pointer is advanced automatically.

↳ write :- (write next)

↳ appends to the end of the file & advance to the end of newly written material.

(2) Direct Access.

↳ A file is made up of fixed length logical records that allows programs to read & write in no particular order.

↳ Direct Access file is viewed as a numbered sequence.

↳ direct access files are great use for ~~[illegible]~~.

Immediate access to large amounts of information.

Ex:- databases.

↳ Read (Read n)
  ↳ Reads from block number 'n' from the file.

↳ write (write n)
  ↳ write from block number 'n' from the file.

↳ Here user has to provide a relative block number 'n'. corresponding physical address is found in the file.

↳ Directory structure.
  ↳ Directories are used for organizing the file data.

  ↳ storage structure.
    ↳ A disk is a large storage device and these disk is divided into parts, so we can have multiple file system on a disk.

    ↳ These parts are called.
      (1) partitions.
      (2) slices.
      (3) Minidisk.

    ↳ Here a file system can be created on each of these parts of the disk.

    ↳ Combination of these parts form volumes and file system can created in these also.

↳ volume can be thought as virtual disk.

↳ Each volume can store multiple operating systems.

↳ Each volume contains a device directory and file systems are present in device directory.

↳ directory records information such as name, location, size and type for all files on that volume.

↳ operation performed on a directory.

- ↳ search a file in a directory.
- ↳ create a file in a directory
- ↳ delete a file in a directory
- ↳ List a files in a directory
- ↳ Rename a file in a directory
- ↳ we can move one file to other in directory.

↳ Single-Level Directory.

↳ Here all the files are contained in the same directory.

↳ Advantages:

- ↳ Implementation will be easy.
- ↳ file operations are done very easy.
- ↳ searching for files will be very fast.

limitations

↳ if files are more @ the system has more than one uses.

↳ user cannot remember all the names of files.

↳ Here all files must have unique names for files in directory.

↳ searching will difficult if no of files increases.

↳ Two - level directory:

↳ Here each user has his/her own directory called user file directory (UFD)

↳ Information about (UFD) is stored in a system directory called master file directory (MFD).

↳ This MFD is indexed by user name and each entry points to the UFD for that user.

↳ Here if user wants a particular file then he has to search his own UFD.

↳ problems:

↳ Here users are isolated from each other, But if two users wants to access the same file, then it becomes to difficult because system does not permit this.

↳ If other users wants to access the files then the files must be named with their complete paths.

Eg: ./user/test. (or) c:\user1\test..etc

↳ Here if the user wants to access a system file, then he should be first check in UFD and if it is not present, then those will be present in special user directory present in MFD. as (user 0).

↳ Tree level directories :-

↳ Here users will create their own sub directories to organize their files accordingly.

↳ The tree has a root directory and every file in system has a unique path name.

↳ Here we have to specify a path name to find a file. and if you are in current directory then directly we can search there.

↳ path names :

① Absolute path name :

↳ Here we have to start the path from root.

Ex: c:/root/spell/mail/emp/file.

② Relative path name:

↳ starts the path from current directory.

Ex: ./pt/first..

↳ How to delete a directory in tree level directory.?

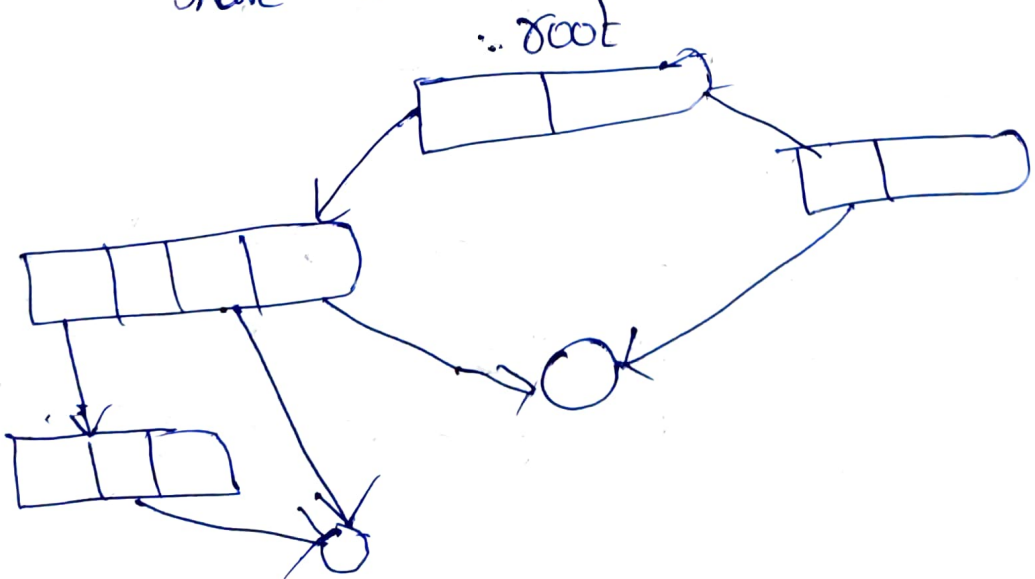
↳ if the directory is empty it can be simply deleted.

↳ if the directory is not empty then we have to check if any sub directory is present are not and first we have to delete all sub directories and then we have to delete the root directory.

↳ Acyclic - graph directories:

↳ If the multiple users wants to share a file@ directory then it is not possible in any of level directories.

↳ So, the Acyclic graph that allow directories to share sub directories and files with no cycles.



↳ Implementation of shared files & subdirectories.

↳ links:
↳ A link is a pointer to another file or subdirectory that tells that it is sharing.

↳ duplicating:
↳ Duplicate all information about shared files in both sharing directories.

↳ problems in Acyclic graph
↳ same file may have many path names.
↳ if the file is deleted by one user, it will leak to dangling pointers
↳ so here we have to preserve the file until all references are deleted.

↳ General graph directory:
↳ we know that in Acyclic-graph directory there should be no cycles.
↳ sometimes it may contain cycles, so for Avoiding the cycles we just add links to an existing tree structured directory that results simple graph structure.

↳ so here we can traverse the graph.

↳ travel accorse.

problems :-

↳ Here if the cycles are allowed then the searching in cycles will decrese the performance.

↳ if a poorly designed algorithm result an infinite loop.

↳ Here we can not delete the cyclic graph file.

↳ For that we need Garbage collection.

↳ It is a scheme that determins the last reference has been deleted or not.

steps :-

↳ Travel accorse the entire file system and mark accessed one.

↳ Travel accorse the entire file system and unmark the one that is not accessed. and then we will delete.

↳ Garbage collection is very time consuming scheme and It is rarely attempted in system.

↳ File system mounting :-

↳ A file must be opened before it is used. But to open, file must be available.

↳ So, Mounting is a process by which OS makes the files and directories on a storage device available for users to access in file system.

↳ mounting procedure :-

↳ OS provides name of device & mount point

↳ A mount point is a directory or file at which a new file or directory or a file is made accessible.

↳ File sharing :-

↳ It is very Important for users who want to Collaborate & work and to reduce the effort.

↳ So, their should be a user - oriented OS must be accommodate to share files.

↳ File sharing will be done by multiple users.

↳ If it is done then some Issues will be arises. like File sharing, File naming, file protection.

↳ So, To allow file sharing then these issues should be solved.

↳ we have to allow multiple users to access files

↳ if one user wants to access other users files then that should give permission to access.

↳ how to Implement file sharing:

↳ we know that we have file and directory attributes.

↳ likewise file system follow a concept of owner and group.

↳ owner - who can change attributes and gives permission to other users.

↳ group - the users that are subset of owner. who gets permission to access.

↳ so, we have other attributes like owner id and group id for both file & directory.

↳ Remote file systems:

↳ networking allows the sharing of resources spread across the world.

↳ we can share data in the form of files.

↳ methods of remote file-sharing:

↳ we can transfer files by manually transferring files like (FTP) file transfer protocol.

↳ we can transfer files by using distributed file system. Here data is stored in the server and with help of network we can access data from server to a local machine.

↳ world wide web:-

↳ By using Internet we can gain access to the remote file ... etc.

↳ The client server model:-

↳ Remote file system allow a computer to mount one (or) more file system from one (or) more remote machines.

↳ Here machine containing the file is server machine ~~Containing~~ seeking access to files is client.

↳ The server tells that resource is available to clients

↳ specifies which is the resource. & which clients they are available.

↳ Server can serve multiple clients & clients can access multiple servers.

client Identification :-

↳ client can be specified by a network name (A) other Identifiers like Ip addresses.

↳ Here the unauthorized client can have accessy to sever, for that we can use encryption, but both client & server should have same encryption. It is very difficult.

↳ once authorization is done the user sends a request to mount the file system via DFS then sever checks if user has the access (B) not, if it has then server sends files over websites.

↳ Failure modes :-

↳ Here we have two types of file systems.

(1) In local file system..

↳ Failure of disk.
↳ corruption of the directory structure.
↳ Disk controller failure.
↳ cable failure.
↳ host adaptor failure.

↳ These failures need to Repair by a human Intervention.

② In remote file systems :

↳ Hardware failure.
↳ poor hardware configuration
↳ networking Implementation issues.

↳ some systems have built in capability to recover its but some not → This leads to break (DFS).

↳ The system can either terminate (or) delay operations.

↳ To Recover from this failure.
↳ we should maintain a state Information so that we can recover after.
↳ when DFS breaks then. NFS (network file system) is recognizes the mounted exported file system and carries out all the information that needed to send to client.

↳ Consistency semantics :-

↳ This semantics represents in helping file systems that supports file sharing.
↳ This semantics will tell how multiple users of system or to access a shared file.

↳ we have UNIX semantics.

↳ when user writes then no other user should write. (because it has single Image)

↳ if user is writing then the pointer of current location is shown in the file.

Ext google Docs --- etc.

↳ Andrew file system semantics (AFS)

↳ Here if user is writing then if the file is shared to other users they can not see the updated contents at that time.

↳ They only able to see if the file is closed after writing. then they will visible to others. at next sessions.

↳ Here multiple users are allowed to read & write at same time.

↳ Immutable shared files semantics:

↳ once a file is shared by its creator. then it can not be modified.

protection :-

① physical damage :

↳ keep the computer system from physical damage.

↳ can done by

↳ Hardware problems

↳ power failures

↳ Head crashe

↳ Dirt

↳ Temparature extems

↳ human Interaction.

② Improper access :

↳ unauthorized user accesses.

↳ can done by

↳ give access to authorized users.

↳ multiuser system it is difficult.

↳ In single user system it is possible to protect.

↳ Complete protection :

↳ prohibiting access to every one - except user

↳ Free access :

↳ give world wide access with no protection.

↳ Controlled access-

↳ give access to just some ~~edd~~ classes of the file types ~~to~~ to different users.

↳ user permissions :-

↳ Read → Read from file (or) not

↳ write → write (or) rewrite the file (or) not

↳ Execute → load the file and execute (or) not

↳ Append → add new information to file (or) not

↳ Delete → free the space in file (or) not.

↳ list → list the attributes of file (or) not.

↳ renaming & copying & editing the file (or) not

↳ these operation has to controlled to protect.

↳ Access Control :-

↳ Here we give the multiple access rights to multiple users depending on the identity.

↳ For that we use (ACL) Access control list.

↳ This list contains the name & right to access.

↳ if the user wats to access the particular file the os checks the list and if that matchs then user gets access to the file.

problems:

↳ we have to know all the users in the system.
↳ if we add more users then size also Increses.

↳ sol to the problems are;

↳ Here we keep all users into three classifications

① owner → who created the file.
② group → set of users sharing the file
③ universe → All users apart from owner & group

↳ separate rights will be allocated to these three classifications.

Virtual memory

↳ We know that any process has to load into main memory for executing. but if the size of main memory = 2GB and if size of process = 3GB then we can not accomodate for that we use virtual memory.

↳ Here secondary memory is treated as though it is a part of main memory & large process are stored in it.

↳ Only the part of the process that is actually needed for execution will be loaded into the actual main memory.

↳ Benefits of virtual memory.

① user can write large size of programs.
② we can save our main memory space.
③ less I/O operations are needed to load.
④ It is benefit for user & system.

↳ Demand paging

↳ How can an executable program be loaded from disk to memory?

① Load entire program in physical memory. (wasting of main memory)

② pages are loaded when they are demanded during program execution. (This is called Demand paging).

↳ with help of lazy swapper we will swap the pages to physical memory when they are needed (or) demanded.

↳ swap is actual pager (we have to use pager term)

↳ Implementation:-

↳ we use valid/invalid Bit.

↳ valid → This page is both legal & in memory

↳ invalid → This page is either not in logical address space of the process (or) in the secondary memory.

1) If a process tries to access a page that is not present in memory (or marked as invalid) This is known as pagefault.

↳ If the page fault occurs on the instruction fetch then we can restart by fetching the instruction again.

↳ If the page fault occurs while we are fetching the operand then we must fetch and decode the instruction again.

↳ How to handle page faults?

(1) check an internal table (PCB) whether the page is valid or invalid.

(2) If it is invalid then terminate.

(3) If it is valid but we have not yet brought in that page then we have to page it.

(4) we have to find a free frame.

(5) Now we have to schedule a disk operation to read the page in the frame.

(6) we update the internal table.

(7) we have to restart the instruction cycle.

↳ performance of demand paging.

↳ ~~If a computer sys~~

↳ Demand paging effects the computer performance significantly.

↳ If the Effective Access time = memory access time then page fault occurs.

Effective Acces time = $(1-p) \times$ ma $+ p \times$ page fault time

ma → memory Access time.

↳ p is probability of page fault (0 ≤ p ≤ 1)

↳ page fault effects the computer system performance.

↳ EAT $\propto$ page fault rate.

↳ Copy on write.

↳ when a process going to write (or) modify a page only at that time we have to create duplicate a copy and other original page will be modified this technique is called copy on write.

↳ problems in demand paging.

↳ over-allocation of memory, hence in multiprogramming system we are facilitating the possibility of having more process loaded in to the memory.

↳ For this

(1) we could terminate the process

(2) OS can swap out the process.

(3) (OR) we can make use of page replacement.

↳ page replacement:

↳ It is technique of swapping out pages from physical memory when there are no more free frames available in order make room for other pages which has to be loaded to physical memory.

↳ Here if we use modify (dirty) Bit we can reduce the effective access time. that decreases the page fault service time.

↳ Fifo page Replacement Algorithm:-

↳ Here oldest page in memory is chosen to be swapped out.

↳ Here we have to maintain the fifo queue that can hold all the pages that are in memory.

↳ Here head of the queue is swapped out and tail of the queue is swapped in.

↳ Belady's Anomaly:-

↳ normally more number of page frames there will be lesser page faults.

↳ At no of frames increases the no of page faults drops to some minimal level.

↳ Adding physical memory increases the no of frames.

↳ In some rare cases we see that on no of frames were increased the no of page faults also increases, this unexpected result is known as Belady's anomaly.

↳ optimal page replacement algorithm;

↳ It will never suffer from Belady's anomaly.

↳ It guarantees the lowest possible page fault rate for a fixed no of frames.

↳ Here it checks for the page which is not used for long time in the queue and it is replaced with tail of queue.

↳ optimal page replacement algo is difficult to implement because it requires future knowledge of reference string.

↳ LRU (least Recently used Algorithm)

↳ It is better than the fifo ...

↳ when a page must be replaced LRU chooses the page that has not been used for the longest period of time.

↳ It is like optimal page Replacement Algorithm looking backwards in time rather than backwards.

↳ LRU (Implementation)

↳ By using counters.

↳ In each page table entry we have time of use field & counter to the cpu.

↳ Here a clock is incremented for every memory reference.

↳ When a reference to a page is made the contents of the clock is copied to time of use field.

↳ Here we have time of last reference, so there we can replace the page with smallest time value.

↳ By using stack.

↳ Here we maintain a stack of page no.

↳ When page is referenced, it is removed from stack and put on the top.

↳ Most recently used page is always on Top & least recently used page is always at the bottom.

↳ LRU - Approximation page Replacement Algorithm.

(oo) ↳ Additional Reference bits Algorithm.

↳ some systems does not provide LRU support.

↳ Here systems support for LRU in the form of a reference bit.

↳ Here reference bit is used pages are initially to '0'.

↳ if the page is referenced then bit set to '1'.

↳ here we can not find the order, But in additional reference bits we can.

↳ Here we keep 8 bit byte for every page table and at regular intervels and timer interpets and transfers control to os.

↳ The os shifts the reference bit for each page into high order bit of its 8-bit byte.

↳ Second chance Algorithm :

↳ Here we check its reference bit.

↳ If Reference bit 0 → we replace it

↳ If Reference bit 1 → we give second chance.

↳ Here when page gets second chance its reference bit is set to '0'. & arrival time is now current time.

↳ If a page is frequently used its reference bit will mostly set '1' and it will never be replaced.

↳ Enhanced second-chance algorithm.

↳ Here we use Reference bit & modify bit.

Reference bit → when it is referenced

modify bit → when it is modified.

Ref	mod	
↳ 0	0	→ Best page to replace
↳ 0	1	→ here we have to write this page in disk, so it is not best page.
↳ 1	0	→ here we have know that it will be used again, so not best page to replace.
↳ 1	1	→ here we have know that it will be used again and it has to write this page in disk, so not best page to replace.

↳ Counting Based page Replacement:

↳ Here we maintain a counter that keeps a count of no of references made to each page.

① Least frequently used (LFU)

↳ pages which are least used - will have lesser count.

↳ pages which are frequently used - will have greater count.

↳ Here we replace the page with least count.

↳ If processes of higher count are present in page may not be needed further. that pages count is ~~replaced~~ shifted right by ① bit in regular Intervels.

② Most frequently used (MFU)

↳ (Here we use the same points)

↳ Here we do not replace the page with the least count.

↳ These algorithms are very expensive to implement.

↳ do not approximate optimum page Replacement algorithm.

↳ page - Buffering Algorithms:

↳ Here we maintain a pool of free frames.

↳ when a page fault occurs:

↳ A victim frame is chosen. and desired page is loaded into a frame from the free frames pool.

↳ This happens when the victim page is swapped out and after that victim frame will be added to the free frames pool.

↳ Hee process can restart its execution as soon as possible without waiting.

↳ Hee when the paging device is ~~idel~~ idle then we ~~dirty~~ write the page to the disk from the pages that are present in the pool of free frames.

↳ we modify the modified bit.

↳ By this we can save time that will be wasted in future to write the page to disk.

Allocation of frames:

↳ How do we allocate to various processes the fixed amount of free memory that is available?

↳ If page sharing is available then we can allocate more than the total no of available frames.

↳ minimum no of frames should be allocated to each process.

Allocation Algorithm:

↳ how many frames should be allocated to different process.

① Equal Allocation.

↳ The frames are equally distributed among the processes.

↳ m → no of frames & n → no of processes

↳ we allocate m/n frames to each process.

problems:

↳ The memory requirement of process are not same.

② proportional Allocation :-

↳ The frames are distributed among the processes according to their sizes.

↳ no of frames allocated to $P_i = a_i$:

where $a_i = \frac{S_i}{S} \times m$

$S_i \rightarrow$ size of process P_i

$S \rightarrow \sum S_i$

③ priority Allocation :-

↳ Here the frames are distributed among the processes according to their priorities.

↳ A process of higher priority get more frames.

↳ A process of lesser priority get less frames.

↳ Global versus local Allocation :-

↳ If we are replacing the ~~pages~~ pages in global way then we follow a global Allocation of frames.

↳ If we are replacing the pages in local way then we follow a local Allocation of frames.

↳ In Global Replacement
  ↳ The selection of replacement frames are done from the set of all frames even if the frames are are allocated to some other process.

problem :-
  ↳ process cannot control their own page fault rate.

↳ In local Replacement
  ↳ The selection of replacement frame are done from the set of frames that is particularly allocated to them.

problem :-
  ↳ Memory utilization not good as compared to Global replacement.

Trashing :-
  ↳ If a process ↳ spends more time in paging rather than in execution. the we say it that process is Thrashing.

problems:-

↳ Have all the processes are busy for the paging device and waiting in queue that decreases cpu utilization.

↳ Then cpu scheduler by to increase the cpu utilization by brings new processes. → this causes Thrashing

↳ Working set model:

↳ Here working set model checks how many frames a process is actually using to provide a process with as many frames as it need.

WSS_i → working set size.

total demand for frames → $D = \sum WSS_i$

if $D > m$ then. Thrashing occur

File system Implementation :

↳ To Implement file system we have several on-disk and in-memory structures

↳ on-disk structures :

↳ It contains Boot Control Block.

↳ There is a boot control Block for each volume.

↳ It contains Information needed by the system to boot an OS from that volume.

↳ In unix file system we have boot block

↳ In NTFS It is called partition boot sector.

↳ It contains a volume control Block

↳ It contains volume or partition details.

↳ The no of blocks in the partition.

↳ Size of the blocks

↳ Free block count.

↳ Free block pointers.

↳ Free FCB (File control Block) count and FCB pointers.

↳ unix file system → super block

↳ NTFS → stored → master file table

↳ Directory structure per file system.
  ↳ this is used to organize the files.
  ↳ In unix file system it includes file names and associated inode numbers.
  ↳ In NTFS → master file table.

↳ per-File (FCB)
  ↳ It contains many details about the file.
    ↳ File permissions.
    ↳ ownerships
    ↳ size
    ↳ Location of data blocks.
  ↳ In unix file system it is called inode.
  ↳ In NTFS it is stored in the master file table.

↳ In-Memory structures:
  ↳ The in-memory information is used for both file-system management and performance improvement via caching.
    ↳ At mount time the data is loaded & discarded at dismount time.

↳ This structure Includes:-

(1) In memory mount table:
↳ Contain information about each mounted volume.

(2) In memory directory-structure cache:
↳ Contains information of recently accessed directories.

(3) system-wide open file table.
↳ Contains a copy of the FCB of each open file and other information also.

(4) per-process open-file table:
↳ Contains a pointer to the appropriate entry in the system wide open file table.

↳ Virtual file systems:-

↳ It allows multiple file system types to be implemented within the same structure.

↳ user can also access file from both the file systems.

↳ This can be done by some object-oriented techniques.

↳ object-oriented technique consists of three major layers in file system implementation.

(1) The file system Interface (Layer 1) :
(2) The virtual file system (VFS)
(3) The remote file system protocol.

↳ It provides virtually many file systems but user thinks it is my own file system.
↳ It will represent a file uniquely by a vnode.

↳ Directory Implementation :
↳ These directory-allocation & directory management algorithms used in a system affect the efficiency, performance and reliability of the file system.

↳ Two method to implement a directory.

(1) Linear list.
↳ A linear list of file names is used which points to the data blocks.

↳ It is very time consuming to execute.

Disadvantage :

↳ Linear search is required for every operations.

(2) Hash Table.

↳ The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list.

↳ This decreases the search time.

Disadvantage :

↳ Some times two file names hash to same location.

↳ hash function depends on that size.

↳ Contiguous disk space Allocation :-

↳ It has to provide efficient disk space utilization.

↳ Fast access to the file blocks.

↳ This scheme provides starting block address and length of the file. So that we can determine the blocks occupied by the file. (sequential & Direct access is possible)

Advantages

↳ with the starting address we can access any block.

↳ This is very fast access scheme.

Disadvantages :-

↳ It suffers from Internal, external fragmentation.

↳ files with larger size can not accessed.

(b) Linked List Allocation :- (only sequential possible)

↳ In this scheme, each file is a linked list of disk blocks which need not be contiguous.

↳ If directory entry contains a pointer to the starting & the ending file block. Then we can access the next block of the file.

Advantages :-

↳ This is very flexible in terms of size.

↳ Does not suffer from external fragmentation.

Advantages:-

↳ file blocks are distributed randomly.

↳ we can access the file block randomly @ directly.

↳ Indexed Allocation:- (Sequential & direct access is possible)

↳ In this scheme, a special block known as Index block contains the pointers to all the blocks occupied by a file.

Advantages:

↳ supports direct access to the blocks.

↳ does not suffer from external fragmentation.

disadvantages:-

↳ pointers are required in the linked list It is overhead.

↳ For longer blocks we can not specify Index block (that of small size), we may want more Indexes.

↳ for that we have linked scheme, multilevel Index & Combined scheme.

↳ Unix Inode:

↳ Here each file is Indexed by an inode.

mass-storage structure :-

↳ we have three parts

(1) File system Interface.

↳ The user & file system Interface.

(2) File system Implementation.

↳ Done By - internal data structure and algorithms used by the os to Implement Interface.

(3) mass-storage structure.

↳ study of Internal secondary storage structure.

↳ It the lowest level of file system.

↳

↳ we use magnetic disk and magnetic tapes as the secondary storage device.

↳ magnatic disk

↳ povides bulk of secondary storage.

↳ disk platter look place circular shape (CD)

↳ Its diameters are 1.8 to 5.25 inches.

↳ These platter are covered with magnetic material.

↳ with help of these magnetic material. we record information.

↳ The read - write head flies above the each surface of every platter.

↳ These heads are attached to a disk arm.

↳ These above surfaces have several tracks and these tracks are sub divided into sectors.

↳ A cylinder is any set of all tracks of equal diameter that are at one arm position.

↳ These disk contain thousands of cylinders.

↳ In order to store a data from the disk to file system, the disk has to rotate 60 to 200 times per second with help of drive motor.

↳ Transfer rate → The rate at which data flow b/w disk & computer.

↳ positioning time / Random access time :-

↳ The time to move the disk arm to the desired cylinder (seek time) + The time for the desired sector to rotate to the disk head. (rotational latency)

Head crash:

↳ The read-write head flies on an the surface of disk with little tiny gap.

↳ Here head may contact with the disk surface and damage the magnetic surface.

↳ This is called head crash.

↳ This can not be repaired (entire disk has to replaced.

↳ The disk drive is attached to computer by a set of wires called an I/O bus.

magnetic tapes:

↳ These are tapes of thin layer which is coated with magnetic material, and this tape is attached to two cylinders where the Read-write head is present at same place b/w the cylinders.

↳ Random access time to magnetic tape is more than magnetic disk.

↳ we don not use this magnatic tapes?

↳ The tape has to move to correct spot to Read-write head.

↳ These tapes store 20GB to 200 GB only.

↳ These tapes are designed based width & technlgy.

↳ These tapes are used for backup and devices which are not frequntly used. are used to trasfer data from one system to another.

Disk structure:-

↳ os has logical blocks which is mapped on the sectors of the disk:-

↳ Here sector 0 is the first sector of first track on the outermost cylinder.

↳ This mapping proceeds from outermost cylinders to innermost cylinders.

↳ The amount of data that is transfer from disk to computer must constant.

(1) Constant linear velocity (CLV)

↳ Here the drive increases its rotation speed as the head moves from the outer to inner tracks to keep the same data moving under the head.

↳ rate

↳ outer tracks → more sectors.

↳ Inner tracks → less sectors.

(2) Constant Angular velocity (CAV)

↳ The disk rotates - speed can stay constant

↳ The density of bits decreases from inner tracks to outer tracks to keep the data rate constant.

Disk Attachment:

↳ Disk storage can be accessed by computers By host-attached storage & network attached storage.

(1) via I/O ports.

↳ Here storage accessed through I/O ports

↳ PC uses I/O bus architecture

↳ IDE (Integrated drive electronics)

↳ ATA (Advanced Technology Attachment)

↳ For newer pc's (serial ATA)

↳ For high end work stations.

↳ SCSI (small computer system interface)

(2) network Attached storage.

↳ A network attached storage device is special-purpose system that is accessed remotely over a data network.

↳ we access this NAS by remote procedure call Interface.

(3) Storage Area network

↳ The storage I/O operations consume bandwidth on the data network. Thereby increasing the latency of network communication.

↳ It uses storage protocols rather than network protocols by (SAN) storage area network.

Disk scheduling:-

↳ OS has to use the hardware efficiently.

↳ For this we have.

↳ Fast access time

↳ Large disk bandwidth.

↳ Access Time = seek time + Rotational latency.

↳ Disk Bandwidth = $\frac{\text{Total no of bytes transferred}}{\text{time b/w first request and completion time.}}$

↳ we can Improve both by scheduling the servicing of disk I/O request in a good order.

↳ if the drive @ controller is busy, the newly requests for service will be placed in queue of pending requests.

↳ FCFS algorithms :-

The one who come first will get the chance - one next second.

↳ The algorithm is fair
↳ no starvation.
↳ The performance is not good.

↳ SSTF algorithms (shortest seek time)

↳ The SSTF algorithm selects the request with the minimum seek time from the current head position.

↳ It is better than FCFS algorithm
↳ Can cause starvation.
↳ It is not optimal.

↳ SCAN algorithm :- (elevator).

↳ In scan algorithm the disk arm starts at one end of the disk and moves towards the other end. and servicing requests at it reach each cylinder.

↳ And at other end the direction of head is reversed and servicing continues.

↳ It is better than FCFS algorithm.

↳ No starvation.

↳ If the request that arrives behind the head position has to wait for the head to reach one end and come back.

↳ C-SCAN algorithm :-

↳ like SCAN algorithm C-SCAN moves the head from one end of the disk to the other servicing request along the way.

↳ when head reaches the other end it should Immediately return to the beginning of the disk without servicing any request on the return trip.

↳ This algorithm provides a more uniform wait time.

↳ Total head movement is more than SCAN algorithm.

↳ look scheduling algorithms:

↳ Here the arm goes only as far as the final request in each direction, then it reverse direction immediately without going all the way to the end of the disk.

↳ The seek time is better than all the algorithms discussed ~~for~~ before.

↳ C- look scheduling algorithm:-

↳ In -C- look algorithm moves the head in one direction services requests along the way.

↳ When the head reaches the last request it immediately jetus to the ~~farthest~~ farthest request on the other end without servicing.

↳ This algorithm provides a more uniform wait time.

↳ Total head movement is more as compared to look algorithm.

↳ How do we choose the best algorithm to perform?

Ans:

↳ SSTF algorithm is natural appeal because it increases performance over FCFS

↳ SCAN & C-SCAN perform better for system that place a heavy load on the disk. because they do not cause starvation.

↳ Here performance depend heavily on the number and types of requests.

↳ Depending on the situation we may have to choose which Algorithm should execute.

↳ SSTF (or) LOOK is a reasonable choice for the default algorithm.

Disk Formatting

↳ The process ~~formatting~~ dividing the ~~sec~~ sectors that the disk controller can read & write is called low-level formatting.

↳ This low level formatting fills the disk with special data structure for each sector.

↳ data structure consists of

- ↳ A header
- ↳ A data area
- ↳ A trailer

↳ A header & A trailer contain information used by disk controller. (such as sector no and ECC (Error-correcting code))

↳ The logical formatting is the process places a file system on the disk and allows an OS to store & retrieve files.

Boot Block:

↳ The program that helps the computer to start running when it is powered up or rebooted.

↳ This is done by Bootstrap program. By Initializes all aspects of the system from CPU registers to device controllers and contents of main memory.

↳ Bootstrap program will be reside in the Rom (where we can write & manipulate)

↳ Bootstrap program will be reside in the Rom of Boot Block.

Bad Blocks:

↳ Some this disk is damaged by head. so here one or more sectors become defective. this defective blocks are called Bad Blocks.

↳ This Bad Blocks are handled by manual handling → sector sparing & sector slipping.

manual handling

↳ By performing the logical formatting. we just find the bad blocks and FAT entry tells not to use that block.

↳ when execute if we found bad block then we run chkdsk and locks it away.

Sector Sparing:-

↳ Spare sectors are maintained by the low level formatting process, these are not visible to os.

↳ when bad block or found there will be replaced by these spare sectors

Sector slipping:

↳ Here we just make a free sector just beside the sector that is damaged

↳ so that sequential order is maintained.

→ swap-space needed.

↳ depends on the physical memory.

↳ depends on the virtual memory it is backing.

↳ The way in which the virtual memory is used.

↳ Swap-space is located in filesystem
  ⓐ separate disk partition.

↳ swap map is the array of Integer
  counters each corresponds to a page
  slot in the swap area.

  ↳ counter is 0 → slot is available

  ↳ counter is ~~0~~ → number of mappings to the swapped page.

↳ Raid structure:
  (Redundant arrays of inexpensive disks)

↳ It is a way of storing the same data in different places on multiple hard disks ⓐ SSD's to protect data in case of failure.

↳ RAID places data on multiple disks and allows I/O operations to overlap in a balanced way.

↳ The techniques used by RAID

(1) disk mirroring.
↳ Copies Identical data on more than one drive.

(2) disk striping
↳ spread data over multiple disk drives.

↳ The data will be lost, if the replaced disk also fails before the first disk replaced.

↳ we can Improve the performance of fault tolerance by strips data across the disk.
↳ By splits the bits of each byte across multiple disks → bit level striping.

Raid level → 0 → · stripping · $B_1 \to B_2$

Raid level → 1 → · mirroring · $B_1 \to B_1$

Raid level → 2 → Blocks & parity : $B_1 B_2 P_1 P_2$

Raid level → 3 → Block & parity for individual block sections. $A_1 A_2 A_3 A_{P_{1-3}}$

Raid level → 4 → Blocks & seperate Block parity. $A_1 A_2 A_3 P_A$

Raid level - 5 → · $A_1 A_2 A_3 P_A$
$B_1 B_2 P_B B_3$

Raid level - 6 → — — —

↳ Stable-storage Implemation :-

↳ · Information residing in stable storge is never lost.

↳ replicate the information in multiple storge device

① successfull Completion.
↳ data return correctly

② partial failure.
↳ data partially written

③ Total failure
↳ data failure is started at beginning