Operating System...

Saruha questions → Multithreading, Deadlock, Hashmap, ~~Indexing~~ & Normalization, when to use linked list & stack, tree all easy level questions.,
how to read a text file using cpp. ie how to find that word. json parsing, stringify

Logical address & physical address :-

→ Logical → cpu generates the logical address while the program is running.

→ Logical address does not exist physically in the memory therefore known as virtual address.

→ the logical addresses are used as a referenced in main memory to access physical addresses. a physical address can not be accessed directly.

→ The set of all logical addresses generated about a program by a CPU is called Logical address space.

physical address =>

→ The physical address is the location in the memory.

→ physical addresses located in memory unit.

→ all the physical addresses mapped to the logical addresses physical address space.

process Synchronisation :-

process synchronisation is basically a way to coordinate processes that use shared resources or data.

- The main aim of process synchronisation is to ensure that ~~that~~ multiple processes access shared resources without interfering with each other, i.e. to prevent possibility of inconsistent data due to concurrent access.

- In multiprocess system, synchronisation is necessary to ensure data consistency, integrity, and to avoid deadlock & to resolve the problems of race condition. etc.

- to use synchronisation techniques are: muterxes, semaphore, ~~critical sections~~, file locks, condition variable etc.

two types of processes.

→ independent processes :- execution of one dus not depends on another.

→ co-operative process :- A process that can affect or be affected by other processes.

process synchronisation problem arises in case of Co-operative process.

When more than one process is executing the same code or accessing the same memory or any shared variable so in that case there is the possibility that o/p of or value of shared var is wrong so for that all the processes doing the race that my out is correct this condition is called as ~~sha~~ race condition.

- Several process access & process the manipulations over the same data concurrently, then the output depends on the particular order in which the access takes place.
- Race cond'n ~~exists~~ may occurs in critical section.
- This happens when the result of multiple threads in critical section differs acc to order in which ~~they executes~~ access takes

place.

proper thread synchronisation using locks & or atomic variable can prevent the race conditions.

Critical section :- A critical section is a code segment that can be access by only one process at a time.

The critical section contains the shared variables that need to be synchronized to maintain the consistency of data variable...

So critical section problem means designing a way for co-operative processes to access shared resources without creating data inconsistencies

```
do {
    [entry section]
        critical section.
    [exit section]
        remainder section.
} while (TRUE);
```

In entry section, the process request for entry in the critical section.

Any soln to critical section problem must satisfy the three conditions.

① Mutual exclusion: if a process is executing in its critical section, then no other process is allowed to execute in critical section.

② Progress :- If no process is executing in critical section & other processes are waiting outside the critical section then the processes which are in remainder (their) section can participate in deciding which will enter in the critical section next.

eg:- Petersons soln is a soln for critical sect'n problem.

③ Bounded Waiting :- A bound must exist on the number of times that that other process are allowed to enter in their critical section after made request by process & before accepting request.

Read Mutex Before Semaphore in depth.

Semaphore :- A semaphore is a signaling mechanism & a thread that is waiting on a semaphore can be signaled by another thread.

- This is different than the mutex as mutex can only be signaled ~~called~~ by a thread who called the wait function.

- operations on semaphore - wait() & signal(). for process synchronisation.
- semaphore is an integer variable which can be access by above two operations only.
- 2 types → Binary & Counting semaphore -

Counting Semaphore :- they can have any value bet'n $[-\infty, \infty]$

- used to control access to resources that has the limitation on the simultaneous access. The semaphore can be initialized to no. of resources of instances of resources, if process wants that resource it checks whether ~~no~~ no. of remaining instances greater than 0. i.e process instance available if less than zero then that process goes to block / suspend state & info went to PCB.

Whenever any process comes out of ~~PCB~~ critical sect'n it adds one to semaphore value & if at time of exit value of semaphore ≥ 0 then it takes process from suspend list & put in ready queue again..

at entry section process runs code.

- this is called 'P' or Down operation or wait()

```
Down (semaphore S){
  S.value = S.value - 1;
  if( S.value < 0){
      put procer in (pcb) in
      suspended list , sleep().
  }
  else {
      return ;  → that is put in critical section
  }
```

But we know only one (1) process can be there in critical section, yes but it is just example consider value of semaphore as 1 then automatically mutual exclusion will occur.

- At exit there is Up or V() or signal operation.

```
Up (semaphore S) {
S.value = S.value + 1;
if( S.value ≤ 0){
    select a process from suspended list &
wake up() i.e. put in ready queue
  }
}
```

Advantage :- ① Machine independent ② No waste of resources due to busy waiting.

(2) Binary Semaphore :-

it can have either 0 or 1 value only. also known as mutex locks as locks can provide mutual exclusion. all processes can share the same mutex semaphore that is initialised to 1. when process enters critical section it makes semaphore as one & when it comes out of critical section it decrements value to 0.

if value is 0 other processes can enter in critical section.

What is IPC ? different IPC mechanisms - ?

- IPC allows different process to communicate with each other.
- It is simply used for exchanging data betn multiple threads in one or more programs or processes.
- IPC also helps in synchronising the actions of those communicating threads.
- Different IPC Mechanism ?

1) pipes 2) Message Queuing 3) Semaphores 4) Socket
5) Shared Memory 6) Signals.

pipes :- using pipes two or more processes can communicate with each other by creating bi-directional or unidirectional channel betn them.

→ implemented using system calls.

unidirectional means → only one process can send data at a time.

In bidirectional process must be synchronised to ensure data transmitted in correct order.
this is overhead & disadvantage.

By pipe output of one process is given to another process as a input. thus one way flow provided betn 2 related processes.

~~when one~~ pipe has two ends input & output (write into i/p & read from o/p)

pipe descriptor has array which has two pointers for ip&op.

- if two processes are communicating writting process should close his read end & vice versa..
- Limitation :- size of pipe is fix.

eg. keyboard.

Data from output buffered until until input process receives it as input.

2) Named pipes :- using this communication can be betn unrelated processes or devices
- it can be used in a processes that don't have a shared common proces origin.
- eg: fifo.
  Synchronisation is required for Named pipe. It ensure data transfered in correct order.
-

3) Message Queuing :- this allows messages to be passed betn processes using either a single queue or several message queues.
- this is managed by system kernel.
- this messages are co-ordinated using an API.

4) Semaphore :- this is used in solving problems associated with synchronisation and avoiding the racing condition.
  Default value of semaphore is ≥ 0.

5) Shared Memory :-
- it is OS feature that allows the the database server, threads & processes to share data by sharing access to the pools of memory.
- to communicate via shared memory, communicating processes needs to establish a region of shared memory.
- shared-memory region resides in the address space of the process creating a shared memory segment.
- other processes that with to communicate using this shared-memory segment must attatch it to their address space

6) Sockets :- This method is mostly used to communicate over a N/W bet'n client & server. It allows standard connection which is computer & OS independent.

Q What is Bootstrap program in OS ?.

⇒ it is a program that initialises the OS during startup i.e. the first code get's executed when computer system starts-up.

- it initialises all the aspects of the system, from CPU register to device controllers & the content of the main memory.

- this bootstrap program resides in ROM (for most of computers)
  - → ROM because ROM is read only & can not be affected by virus.
  - → ROM does not require initialization and moreover location here is fixed so that processor can start executing when powered up. or reset.
  - → ROM faster than hard drives, as it is directly stored on mother board of the system.

- to do his job bootstrap program finds the OS kernel on disk and then loads the kernel into memory after this, it jumps to the initial address to ~~jump~~ start operating system execution.
- The full bootstrap program is stored in the boot blocks at a fixed location on the disk.
- The code in the boot ROM basically instructs the read controller to read the boot block into memory & then start the execution of code.

Bootstraping :- Process of loading, set of instruction, when comp first turned on.

Q. Context switching :- context switching in os involves saving the context or state of running process so that it can be restored late on & then loading the context of another process & run it.

- it is a method used by system to change the ~~state of~~ process from one state to another ~~process~~ state using the CPUs present in the system to perform it's job.
- Context switching enables all the processes to share a single CPU to finish their execution & store the status of systems tasks.
- Context switching only allows single CPU to handle multiple process requests parallelly without the need for any additional processors or additional resources.

eg :- there are 'N' no. of processes stored in PCB, the process is running using CPU to do it's job so he is in running state now if high priority process will come from ready queue then our running process state will save to PCB & he will changed to suspend/ waiting state & High priority process state will change from ready to running. once high priority process complete it's work again state of previous process can be loaded from PCB.

Context switching triggel :-

① Interrupts -> I/o interrupt or need. } storing in PCB so late we can resume process from the state it was earlier at or left at.

② Multitasking.

③ user / kernel switch.

fork values

returned -ve value → child creation unsuccessful.
+ve value → Returned to parent or caller.
0 value → child creation successful & returned to newly created child process.

Q. Many to one model ?

→

Q. How mutex & turn variable works together.

→ - when process wants to enter the CR section it checks turn variable to see if it is it's turn, if not it waits until the turn becomes it's identifier.
- Once the process determines it's turn, it check mutex variable if the mutex is locked, process waits until mutex is unlock -
- when process enters the critical section it aquires lock & perform critical operation releases the mutex lock, and then updates the turn variable to allow the next process in line to enter the critical section.

RTOS :- RTOS is an os used for real time applications, where data processing should be done in fixed time & in short time.

- it performs better on task who need to be done in short time - -
- occupies less memory.
- consumes fewer resources,
- but less no. of jobs can be executed online.

Hard, soft & firm RT. OS.

Components of RTOS :-

1) The scheduler 2) function Library 3) fast Dispatch Latency 4) user defined data objects & classes.

5) memory Management 6) Symetric Multiprocessing.

top 10 OS examples :-

- Android . Centos , chrome os . Debian . Fedora
- free BSD . MS-OS , MS-windows . Solaris . Ubuntu.
- Dos

Re-entrancy : it is a function in which various clients can use the single copy of program during a single period

- it does not deals with the concurrency.

it has a two major functions :-

- program code can not change or modify itself.
- Local data for every client need to be stored in different disk.

fragmentation :- processes stored & removed from the memory, which makes the free memory space., which is too little even to consider utilizing by different processes. Suppose processes is not ready to despense to memory blocks since its little size & that memory consistently staying unused this is called as memory fragmentation.

- this occurs during dynamic memory allotement framework when free blocks are small.

Memory Management :-

- In multiprogramming, the os resides in part of memory & rest memory is used by multiple processes - -
- the task of subdividing the memory among different proces is called memory management.
- required to allocate & deallocate to process.
- minimize fragmentation.
- proper utilization of Main Memory.
- to keep track of used memory space by process.

two memory allocation techniques :-

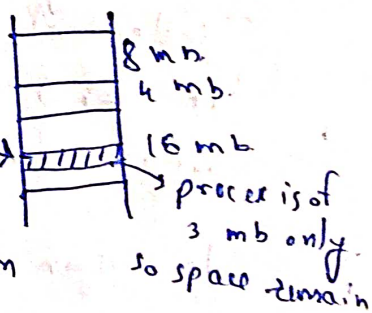
① Contiguous memory allocation :- In it, executing proces must be loaded entirely / completely into partition.

Contiguous divided into.

☒ fixed / static partitioning :-
- No. of partitions fixed, size may or maynot be fixed.
- spanning not allowed.
- partition made before execution during system configuration.

Advantage :- easy to implement / little overhead for os.

Disadvantage :- internal fragmentation.



when size of proces is lesser than partition size because of which unused space remain this is called internal fragmentation.

~~external fragmentation~~ :-

Limit in proces size.

Limitation on Degree of multiprogramming.

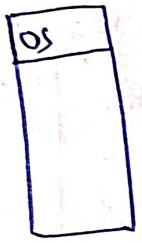
Variable partitioning :-

- partitions generated in run-time.
- initially ram is empty & partitions are made at run time a/c to program process need. & not during system configuration.

advantage :-

- size of partition will be equal to incoming process.
- Avoid internal fragmentation.
- No. of partitions depends on. no. of. processes come.

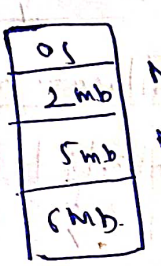
Consider example :-



↑ empty Ram

Now processes are coming of size 2MB, 5MB, 6MB

So =>



Now let say Ram is full Now, let, say 6 MB 2mb process is removed from RAM & new process came which is of 8 MB. but we can not accomodate that process even though we have 8 MB of space because that space is not contiguous Now, this is called external fragmentation.

=> Although we are having available size, i.e. space collectively but still we are not able to put process in that area because we dont have contiguous space it is called as external fragmentation.

We can Remove this disadvantage using compaction means moving processes to one side & space to one side but moving such addresses is costlier & time consuming therefore it is not convinient way.

Disadvantage :- ① External fragmentation.
② Allocation & Deallocation is complecated.

to do allocation & deallocation we use Bitmap & linklist to find holes & process.

(2) Non-Contiguous memory allocation :-

In it, different parts of a process are allocated to different places in Main Memory.

- spanning is allowed which is not possible in static & dynamic partitioning.
- Non contiguous allocation is little bit similar to dynamic partitioning
- Here, process is divided into no. of parts into ~~memory~~ sec called as pages.
- Our RAM also divided into no. of parts same as process no. & each partion size same as page size partion in main memory is called as frames.

no. of. pages = no. of. frames for a given process.

As we know a/c to overlay concept, load only a part of program what we need to execute right now & then unload it & load other part to execute.

Here, we load desired pages into main memory ie. into a frames.

Why we need paging :-

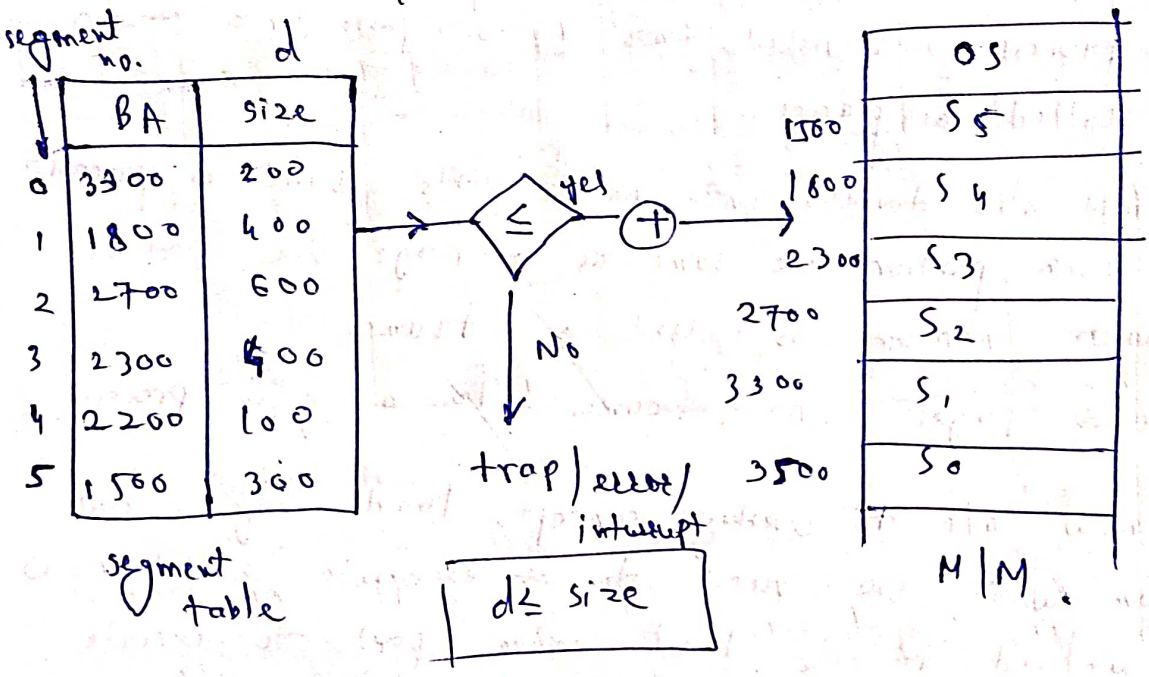
Let say there is a process in secondary memory & we went to load it now process is of size 6 MB & in RAM we have holes of size 3MB & 3MB. So in non-contiguous allocation we break process into two halfs ie. 3MB & 3MB & load into memory. But everytime while loading we need to break the

process & then load into available place.
this is very time consuming therefore we divide process in advance in sec memory & also we divide RAM in advance.

∴ paging is important..

pages are present in physical address space.

Segmentation :- In os, segmentation is a memory management technique in which memory is divided into the variable size parts. each part is known as segments which can be allocated to a process.



CPU generates the logical address for code of sec memory that logical address has segment no. ie. starting address & segment size ie. from starting address how many bits he can store then os finds that base address in m.m. & then checks whether size of d ≤ size for that starting address in m.m. or not if yes then it allocates that segment for our segment of sec memory & our segment of sec. memory will be loaded into main memory.

Need of segmentation :-

Let say there are pages in memory & 1 process has two functions to complete addition of 2 vars. But in one page only one function is there & as all pages are of same size so we can't take both ~~prog~~ functions in one page so. only page allocated into m.m. So we quickly need to allocate other page having remaing addition function of program. or sometimes it fails to find the page of remaining program which leads to process termination or error.

So to overcome this issue we use segmentation as size of segments may vary so we can accomodate both functions in single segment & load into m.m.

Advantage :- - No internal fragmentation.
- segment table consumes less space than page table.
- As complete module loaded all at once, it improves CPU utilization.
- The user specifies the segment size whereas in paging Hardware determines the paging size.
- flexibility
- sharing :- segmentation allows for sharing of memory segments betn processes. this can be helpful in inter process communication or to share code libraries.

Disadvantage :-
- ⊙ external fragmentation.
- ⊙ overhead of maintaining segment table with each process.

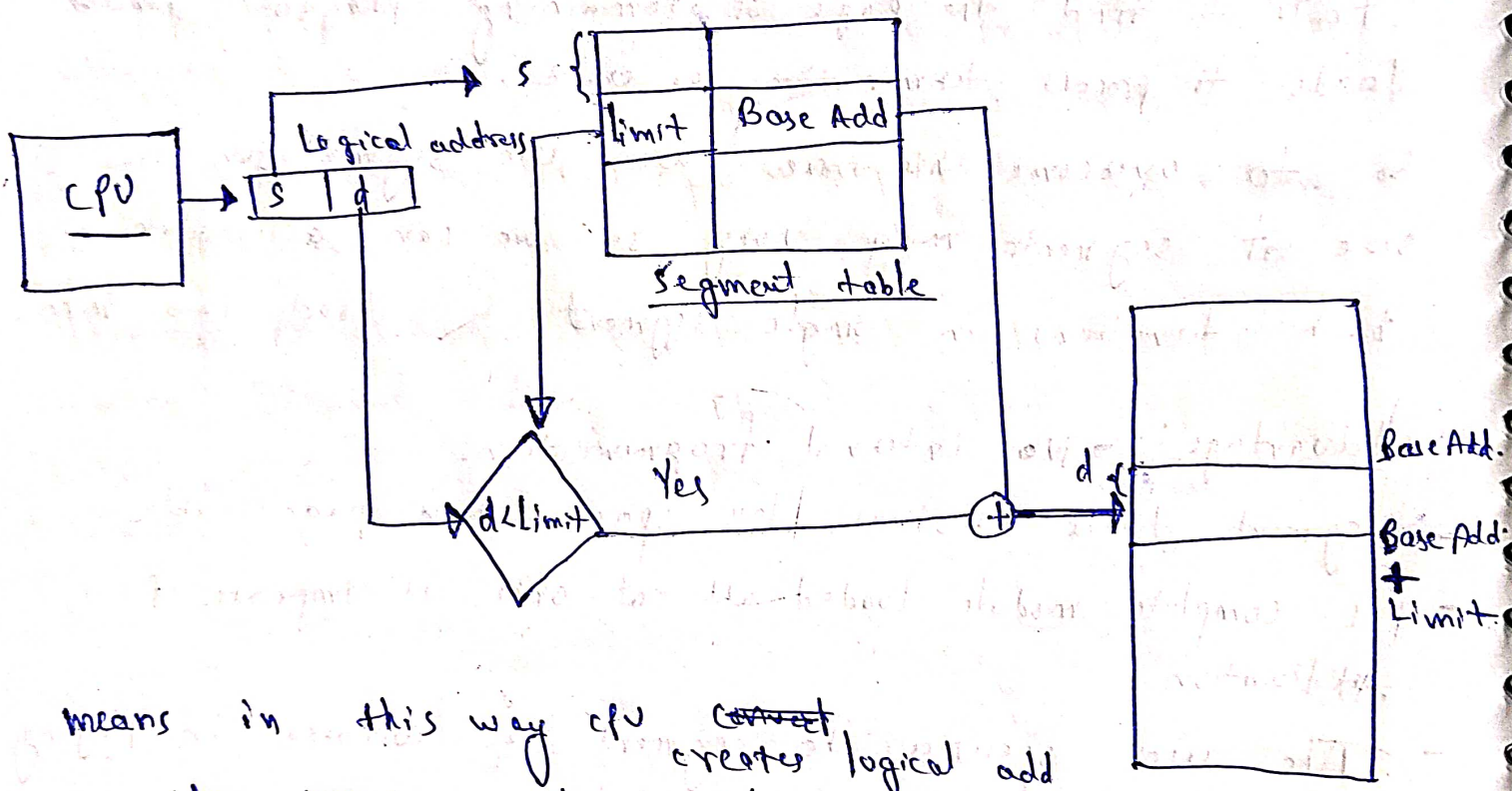
Segment table :-

it maps 2 dimensional logical address into 1D physical address.

it's each table has 2 entries.

① **Base Adress** :- starting physical address address where segment reside in memory.

② **segment Limit** :- Also known as segment offset, it specifies the length of the segment --



means in this way cpu ~~cannot~~ creates logical add -ress then mmu converts it into physical address & maps logical address to physical address thereby creating physical address space & whenever we need segment we can load it via logical address from address space of sec. memory.

address generated by CPU - ~~no of~~

- **segment No(s)**:- No. of bits required to represent segment ~~(s)~~.
- **segment offset (d)** :- No. of bits required to represent size of the bits.

difference Bet'n paging & segmentation :-

- invisible to programmer.
- page size is fixed.
- It is faster for memory access than segmentation
- In this os needs to maintain a free frame..
- The size of pages determined by available memory.

- visible to programmer.
- segment size is not fixed.
- it is slower than segmentation.
- In this os mains a list of holes in main memory.
- The size of segment determined by user.

Memory allocation methods :-

In static partitioning :-

when it is time to load a process into the main ~~the~~ memory & if there is more than one free block of memory of sufficient size then the os decides which free block to allocate.

there are 4 placements algorithms :-
A. first fit. B. Best fit C. Worst first D. Next fit.

① first → partition is allocated which is first allocated & sufficient block from top of memory.

② Best fit → allocates the process to the partition which is first smallest sufficient partition.

③ Worst fit → allocate partition which is largest sufficient among freely available partitions in main memory.

④ Next fit → it is similar to first fit But it will search for the first sufficient partition from the last allocation point..

functionality of memory management -

→ CPU utilization.

→ ~~to carried out multiprogramming~~

→ memory management ensures that the block of memory space are properly managed & allocated so the OS, or other running processes have the memory they need to carry out their operations.

What is the kernel & write its main functions ?

→ The kernel is basically a computer program usually considered as a central component or the modules of OS. it is responsible for handling, managing & controlling all operations of computer system & hardware.

When are the system starts, the kernel is loaded first & remain in main memory. It also acts as interface between user application & hardware.

Diagram

functions of Kernel :-

- it is responsible for managing all computer resources such as CPU, memory, files & processes etc.
- it facilitates or initiates the interaction betn components of hardware and software.
- It manages the RAM memory so that all running processes & programs can work effectively & efficiently.
- It also controls & manages all the primary tasks of OS. as well as manages access to & use of various peripherals connected to the computer.
- It schedules the work done by the CPU so that the work of each user is executed as efficiently as possible.

types of kernels :-

5 types

1. Monolithic
2. Microkernel
3. Hybrid kernel
4. Nano kernel
5. exo kernel.

Microkernel → Microkernel is a type of OS kernel that is designed to provide only most basic services required for an operating system to function, such as memory management and process scheduling. other services such as device drivers & file systems are executed as user level process. that communicates with microkernel via message passing

it is flexible than monolithic because monolithic executes all OS services in kernel space.

- it makes os more secure as kernel level services are reducing. less so attack surface of os reduces.
- it is flexible because if user level process crashed it wont affect stability of os. since kernel is responsible for managing process & memory.

eg:- Mac os.

(2) Monolithic kernel :-

- Like micro kernel it also manages system resources bet'n application & hardware but user services & kernel services are are implemented under the same address space.. it increases the size of kernel thus increases the size of os as well.
- This kernel provides CPU scheduling, memory management, file management, and the other os functions through system calls.

as both services are executing under same address space, it makes execution faster.

- here all operating system service runs in kernel space..

eg:- Linux, unix etc.

- if any one service crashes entire system crashes.
- if user has to add new program or functionality he has to modify entire os.

tight integration of os service.

Difference Bet'n Monolithic & micro kernel :-

Micro

(1) Kernel services present in different address space.

(2) Smaller in size.

(3) If any service fails it does not affect the microkernel.

(4) Uses message queues to achieve IPC.

Monolithic

(1) In same address space.

(2) Larger in size.

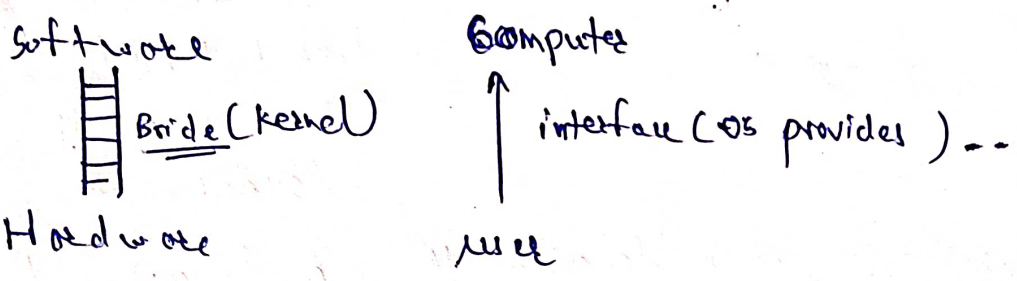
(3) On failure entire system crashes.

(4) It uses signal & sockets to achieve IPC.

Q. What is the difference Betn kernel & OS.

Kernel :- Kernel is a system program that controls all the programs running on the computer. The kernel is basically a bridge betn software & hardware of the system.

OS => OS is a system program that runs on the computer to provide an interface to the computer user so that they can easily operate on the computer.



Kernel..	OS..
- It is considered as a central component of OS.	- It is considered as a system software.
- It converts user commands into machine level commands.	- It manages the resources of the system.
- Interface betn hardware & software/application..	- Interface betn hardware & user.

- it also performs functions like process management, file management, device management, I/O communication, etc.

- it also provides functions like providing security to data & files in the system, providing access controls to users, maintaining the systems privacy. etc.

scheduling algorithms :-

① fcfs.

(2) priority scheduling.

③ SJF (shortest job first scheduling) :-

④ STN (shortest time next scheduling) :-

⑤ Round Robin scheduling :-

(6) Multilevel queue scheduling :-

Spooling :- Symultaneous peripheral operations online.

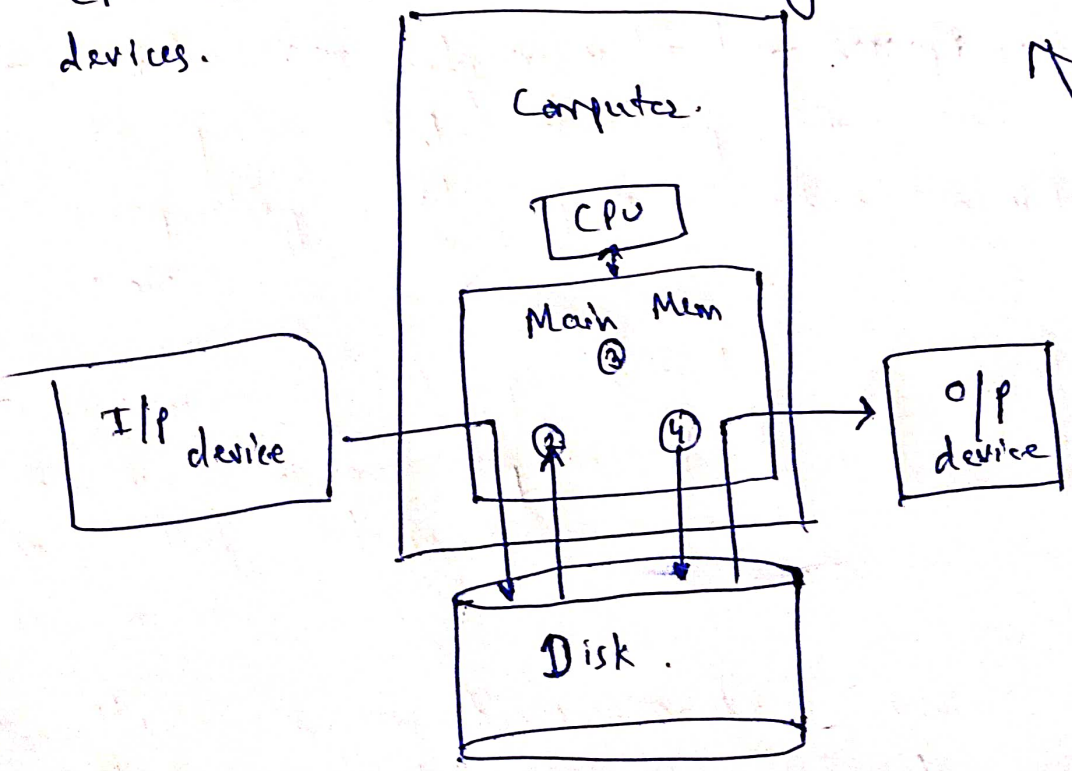
peripheral means → I/O operations.
operations

Our digital devices like RAM & disk are relatively very very fast as compared to I/O devices.

I/O devices very slow so if we make interaction of I/O devices directly to CPU & assigning jobs to CPU via I/O it would be extremly slow.

therefore our I/O devices put processes into disk via RAM i.e. Main memory & our CPU access those resources (Processes) from disk via RAM. & CPU process those processes & put into disk again via m.m. & then again m.m. takes those processes from disk & give to output devices.

as multiple processes can simultaneasly get stored into disk our processing speed increases rapidly. because processes stored in disk & whenever CPU wants processes he can take it by interacting with RAM & CPU - RAM interaction is very fast as both are digital devices.



all about spooling.

earlier CPU used to directly interact with I/O's.

- Spooling is the process of temporary storage of data for use & execution by a device, program or system. Data is sent to & stored in main memory or other volatile storage until it is requested for execution by a program or computer.
- spooling makes the use of Disk for large buffer to send data to printers & other devices.
  ↳ can also be use as I/P.
- primary function of spooling is to prevent two users from printing on the same page at the same time, resulting in their output being completely mixed together.
- it prevents this uses fifo strategy to retrieve the stored jobs in the spool. & creates synchronisation.
- example:- printing is the most obvious application of spooling. The document to be printed are added in the ~~queue~~ spool & then adding to the printing queue, during this time many processes can run & use CPU without waiting while the printer runs the printing process on each document one by one.

Advantages:-

- The spooling operation makes use of a disk as a very large buffer.
- It enables applications to be run at CPUs speed while I/O devices operates at their full speed.

Disadvantages:-

Depending on volume of requests received & the no. of input devices connected, spooling needs a lot of storage.

- spool is created in secondary mem. so there will be disc traffic can be created due to increase in no of devices connected. result in slow performance.

① Difference Betn Semaphore & mutex.

Mutex :- Mutex is a kind of binary semaphore that is used to provide a locking mechanism.

- it stands for mutual exclusion object.
- it is used to provide mutual exclusion to a part of code.

Mutex	Semaphore
→ A mutex is an object.	→ it is an integer.
- mutex works upon locking Mechanism.	- uses signaling mechanism.
- Operations on mutex. — Lock — unlock	- wait - signal
- doesn't have subtypes	- Binary & counting.
- mutex can only be modified by process that is requesting or releasing a resource.	- semaphore works with 2 atomic operations wait, signal which can modify it.
- if mutex is lock process need to wait in process queue. & once lock release mutex can be access	

* Mutex & semaphores are not same. mutex is a locking Mechanism while semaphore is the signaling mechanism.

- Mutex allows only one process to use critical section at a time.

What is a deadlock & different conditions to achieve a deadlock.

A process in os use the resources in following ways.

→ Request a resources.

→ use the resources

→ Release the resources.

deadlock is the situation where set of processes are blocked because each process is holding a resource & ~~a~~ waiting for another resource aquired by some other process.

① Deadlock can arise if following four conditions arises simultaneously.

1) Mutual exclusion : two or more resources are non-shareable (only one process can use at a time).

2) Hold & wait : A process is holding at least one resource & waiting for another resource.

3) No-preemption : A resource can not be taken from a process unless process releases the resources.

4) circular wait :-

A set of processes waiting for each other in circular form.

There are 4 ways to handle the deadlock :-

① Deadlock prevention or avoidance :-

prevention can be done in 4 ways :-

① eliminate mutual exclusion ② Allow preemption.

③ Solve hold & wait ④ Circular wait solution.

Avoidance :- we need to ensure that all info about ~~processes~~ resources that processes will need is known to us before the execution of processes.

Banker's algorithm given by Dijkstra is used to avoid deadlock.

Bankers algorithm :-

Some terminologies :-

1) **Response time** : it is the amount of time it takes for CPU to respond to a request made by process. it is the duration betn arival of process & first time it runs.

2) **Arrival time** :- it refers to moment in time, when process enters the ready queue & awaiting for getting CPU.

3) **waiting time** :- this is process duration in ready queue before it begins executing.

4) **Burst time** :- also known as execution time, it is the amount of processing time required by a process to execute a specific task or unit of job. ~~fact~~
   ⇒ it is the amount of CPU time the process required to complete its execution.

5) **Completion time** :- it is a time when process stops its execution means process has completed its burst time & executed completely....

6) **Turn-around time** :- The time elapsed betn arival of a process and it's completion is known as turnaround time
   t. around time ⇒ Tcpu + Tsleep + Twait.
   t. around time ⇒ Completion time - arrival time.

7) **Throughput** :- the no. of processes completed in a unit time.

8) **Latency** :- In os, latency is the time when an interrupt occurs, & when the processor starts to run the code to process the interrupt.
   ie. time delay ~~betn~~ from occurence of interrupt ~~to~~ until the ISR is entered.

Jitter :- it is considered as the minor differences in latencies bet'n two or more processes. / events.