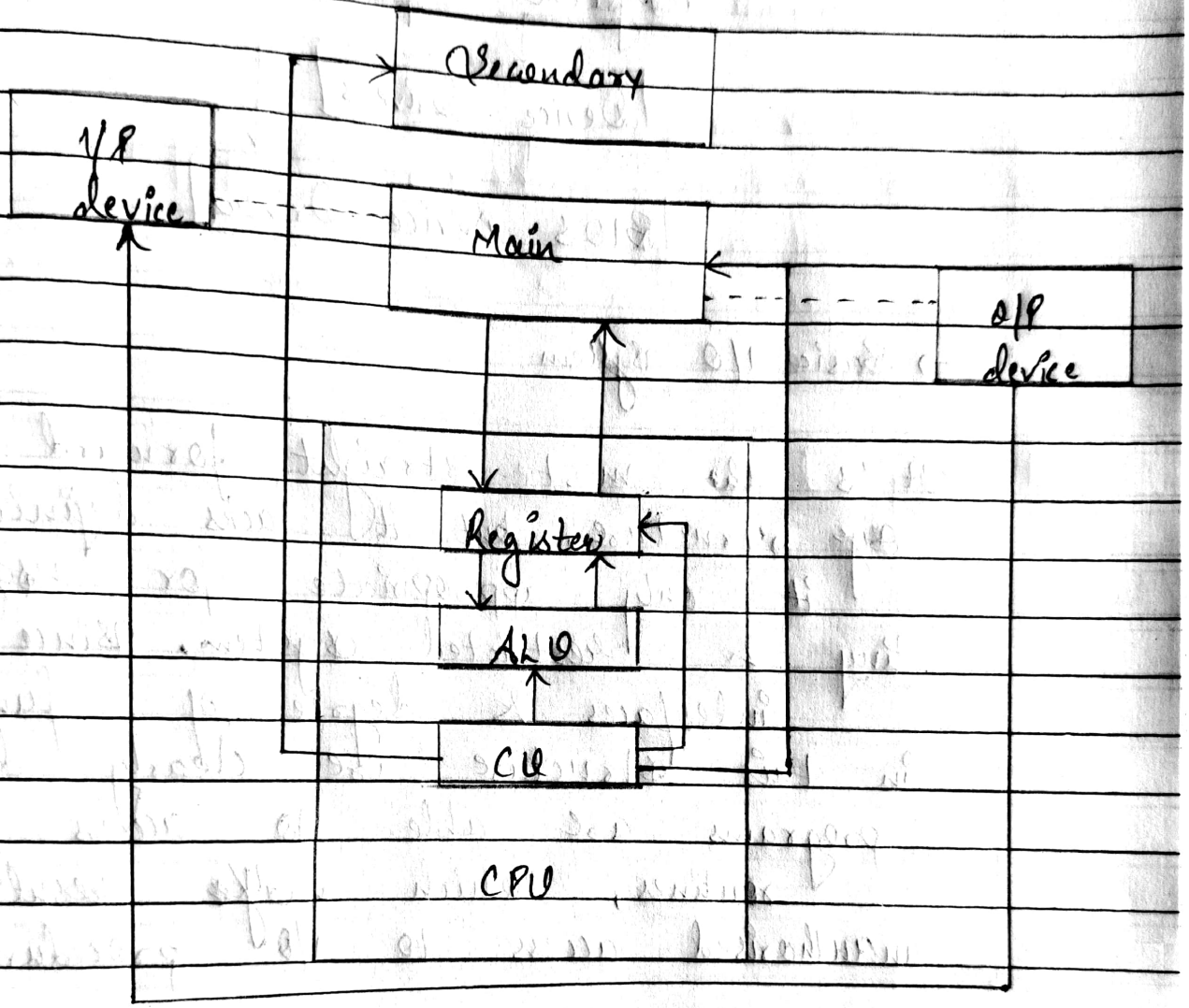
DATE: / /20
PAGE NO.

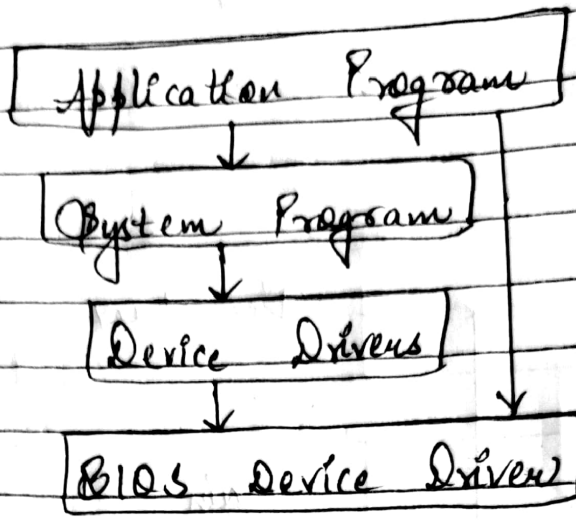
Q.Q.



* **O.S. Structure**

Structure of an O.S. defines how its components are organised, interact & function. Different architecture offers various approaches for managing hardware & software resources, each suited to specific use case.

DATE: / /20
PAGE NO.



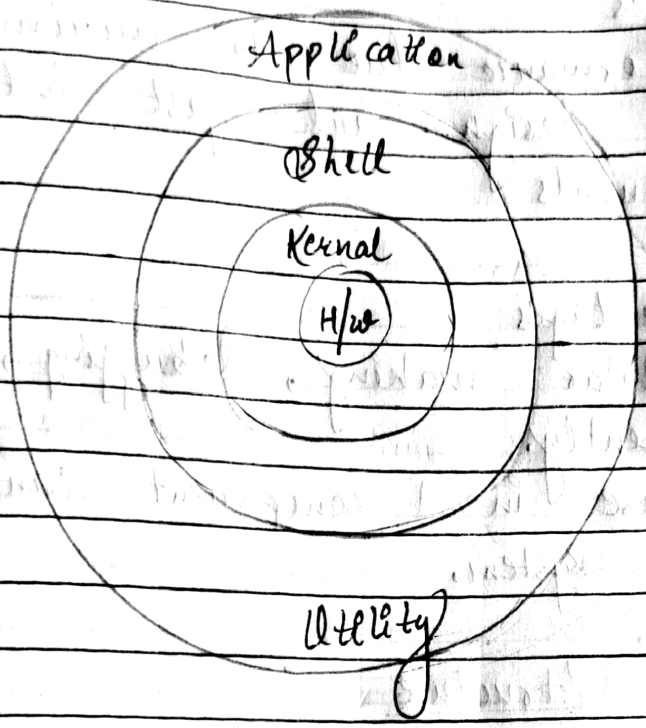
→ Basic I/O System

It's the most straight forward O.S. structure, but it lacks definition & it only appropriate for usage with tiny & restricted system. Since the interfaces & degree of functionality in this structure are clearly define, programs are able to access I/O routines, which may result in unauthorised access to I/O procedures.

Monolithic Systems

In a monolithic system, the entire O.S. runs in a single address space. All the basic services like process management, memory management drivers are directly integrate.

DATE : / /20
PAGE NO.



Linux
Unix
} OoS

Untrivileged mode

App^n
App^n
App^n

Privileged mode
Kernel

file system
Ntwk system
Memory Management
Process Management
Drivers
H/w

Advantages

① High performance due to minimal overhead.
② Simple design and fast interaction b/w components.

Disadvantages

① less modular making, debugging, update differently.
② A failure in 1 component can crash the entire system.

Layered Structure

The O.S. is divided into a series of layers, each build on top of the previous one. The bottom layer interacts with the hardware and top-most layer interacts with the use

Advantages

1. Modular design make debugging & maintainence easier.
2. Higher layers are extracted from H/w details

Disadvantages

1. Poor performance due to strict layer & additional overhead.
2. Challenging in defining clear boundary b/w layers.

DATE: / /20
PAGE NO.

Functions of O.S.

An O.S. acts as a communication interface b/w the user and computer hardware. Its purpose is to provide a platform on which a user can execute programs, conveniently & efficiently.

An O.S. is a software that manages the allocation of computer H/w. The coordination of the H/w must be appropriate to ensure the comp. system's correct operation & to prevent user programs from interfering with it.

The main goal of the O.S. is to make a comp. environment more convenient to use & the secondary goal it uses the resources more efficiently.

Why O.S. used?

O.S. works as an intermediate b/w the system H/w & end user(external user). O.S. handles the following responsibility-

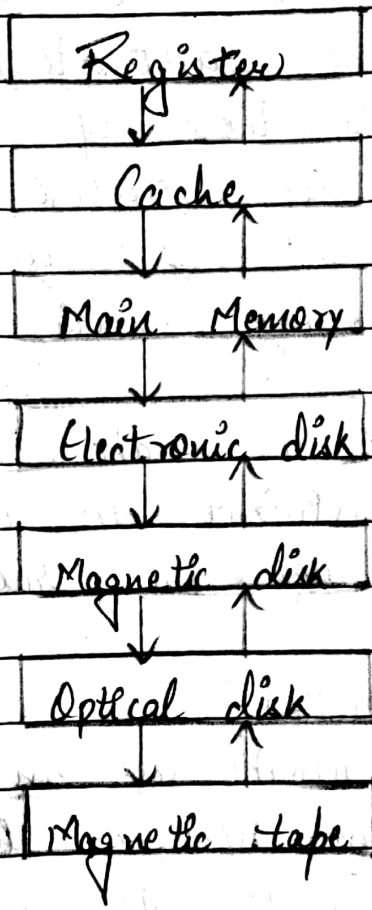
1. It controls all the comp. resources.
2. It provides valuable services to user programs
3. It co-ordinates the execution of user programs
4. It provides resources for user programs.
5. It provides an interface to the user.
6. It hides the complexity of software.
7. It supports multiple execution modes.

DATE : / /20
PAGE NO.

(8) It monitors the execution of users programs to prevent errors.

Functions

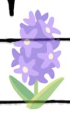
1. Memory Management



DATE: / /20
PAGE NO.

The O.S. manages the primary memory or the main memory. Main memory is made up of a large array of bytes or words where each byte or word is assigned a certain address. Main memory is fast storage & it can be accessed directly by the C.P.U. For a program to be executed it should be first loaded in the main memory. An O.S. manages the allocation & deallocation of memory to various process and ensure that the other process doesn't consume the memory allocated to one process. An O.S. performs many different types of activities for memory management.

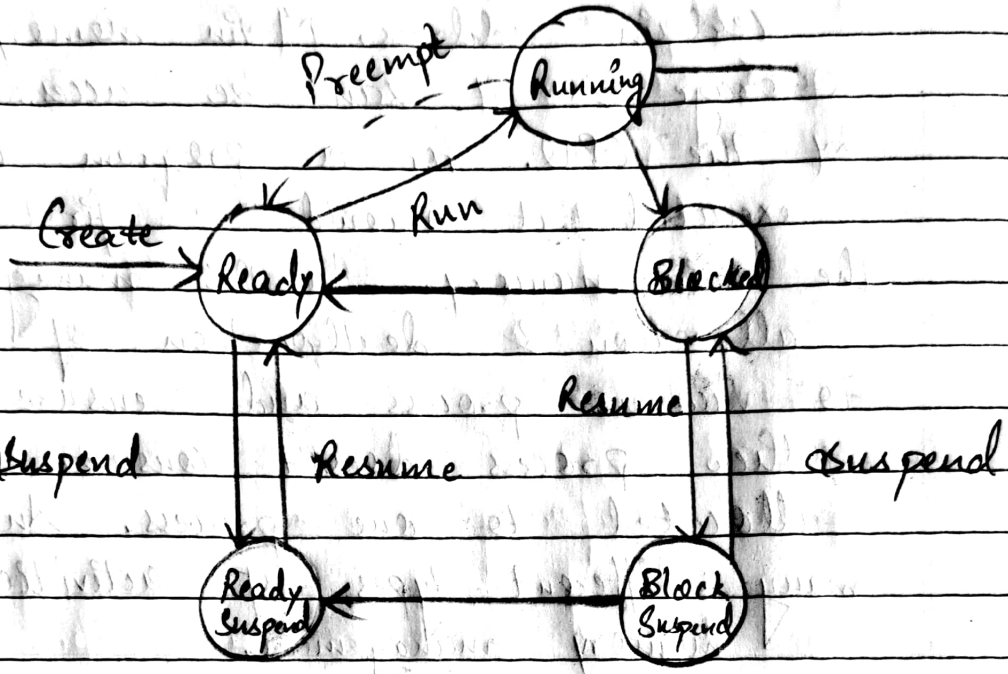
(1) It keeps track of primary memory.
(2) Multiprogramming O.S. decides the order in which the process are granted memory access & for how long.
(3) It allocates the memory to be processed when the process required to deallocate the memory, when the process has terminated & it performs I-O operations.



8. Processor Management

In a multiprogramming environment, the O.S. decides the order in which the process have access to the processor, and how much processing time each process has.

The function of OS is called process scheduling. An OS perform the following activities for processor management.



3. Device Management

An O.S. manages device communication via its respective drivers. It performs the following activities for device management -

(1) Keep track of all devices connected to the system & design program responsible for every device known as I-O device & I-O controller.

(2) Decides which process get access to a certain device & for how long.

(3) Allocates device effectively & efficiently & deallocate devices when they're no longer required.

DATE : / /20
PAGE NO.

(4) There are various I/P & O/P devices; an O.S. controls the working of these I-O devices.

(5) It receives the request from these devices, perform a specific task & communicates back to the requesting process.

6-2-25

Operating System Services

An O.S is a software that acts as an intermediatory b/w the user & computer Hardware. It's a program with the help of which are able to run various applications. It's a one program that is running all the time. Every computer must have an O.S. which smoothly execute the other programs. The O.S. coordinates the use of hardware & application programs for various users. It provides a platform for other application programs to work.

The O.S. is a set of the special programs that run on a comp. System that allows it to work properly. It controls I-O devices, execution of programs managing files etc. There are many different types of services of O.S -

(1) Program Execution (2) I-O operations
(3) Communication b/w process (4) File management

DATE: / /20
PAGE NO.

⑤ Memory management
⑦ Resource management
⑨ Networking
⑪ Time management
⑥ Security & Privacy
⑧ User Interface
⑩ Error Handling

7-2-25

Process Management

Process Management is an O.S. refers to the way of O.S. handles process program in execution by allocating resources, scheduling tasks & managing their execution.

Creating, scheduling & terminating

The key objective of process management are to ensure that process run efficiently, without interference & that system resources are used optimally.

There are many different key concepts in process management:- ① Process

① Process → It's a program in execution. It includes the program called data & system resources required to execute that program. It's process has a unique identifier called process P'd.

② Process Control Block → It's a data structure used by O.S. to store an info. about each process. It typically includes - ① Process P'd
② Process State

DATE : / /20
PAGE NO.

(2) Program
- (3) Program Counter
- (4) CPU registers
- (5) Memory Management Info.
- (6) I/O status info.
- (7) Scheduling info.

(3) Process states → A process can be in one of several states during its lifecycle.
- (1) NEW
- (2) READY
- (3) RUN
- (4) TERMINATE

11-2-25

(4) Process creation →

(a) Creation of a process
When a new program is executed, the O.S. creates a new process. This involves setting up various data structures to track the process such as its I'd or process I'd, memory allocation & other relevant info.

(b) Process control block (PCB)
The O.S. maintains PCB ~~wh~~ for each process which contains imp info. like a process state, program counter, CPU registers, memory management info. and scheduling info.

DATE: / /20
PAGE NO.

(5) Process Scheduling →

(a) Scheduler

The O.S. used a scheduler to manage the execution of process. It decides which process should run at any given time based on certain policies.

(b) Scheduling Queues

Process are placed in various queues such as Ready Queues, waiting Queues.

Ready Q → waiting for CPU time.
Waiting Q → waiting for an event like I/O completion

(c) Scheduling Algorithm

These algo., determines the order in which the process to be executed.

4 types of scheduling algorithms

- First come, first serve
- Shortest Job Next (SJN)
- Round Robin (RR)
- Priority Scheduling

→ **Shortest Job Next**

A CPU Scheduling Algorithm that selects the waiting process with the shortest execution time.

→ Round Robin

It's a scheduling algo. that gives an each process an equal amount of CPU time.

→ Priority Scheduling

The process executes based on the priority.

12-12-25

⑥ Process Synchronisation

17-2-25

Threads

(1) User level thread
- Manage entirely by the user application or threading library.
- The O.S is unaware of the threads & it treats them as a single process.
- Pros - thread creation, synchronisation & management are faster as they do require system calls.
- Cons - if the thread blocks, the entire process is blocked since the OS is unaware of individual threads.

(2) Kernel level thread
- Manage by the O.S, with the O.S Kernel aware of each thread.
- Pros - the O.S can schedule each thread independently, so if one thread blocked other can continue executing.
- Cons - Over head due to context switching of threads at the Kernal level. Thread creation is slower compared to user level thread.

(3) Thread synchronisation in an O.S.

Synchronisation ensures that threads do not interfere with each other when accessing shared resources, such as memory or

I/O devices. Preventing problems like race condition, data corruption. O.S. typically provides several synchronization primitives numbers.

(1) • Mutual Exclusion
It allows only 1 thread to access a shared resources at a time.

(2) • Semaphores
Counting semaphores control access to a pool resources.
for eg. limiting a no. of threads & accessing resources simultaneously.

(3) Conditional Variables
Allow threads to wait for a condition to be meet before proceeding.

(4) Monitors
A higher level abstraction that combines mutual exclusion & conditional variables.

(5) Deadlock and Starvation in multithreaded systems

Deadlock → It occures when 2 or more threads are blocked indefinitely, each waiting for a resource that the other holds.
This can be prevented by careful design, using technique like avoiding circular wait condition or applying timeouts.

DATE: / /20
PAGE NO.

Starvation → It happens when a thread denied access to resources because other threads keep consuming them.

Fair scheduling algo. like round robin can help in preventing starvation.

→ Thread Communication

- Message passing
  Thread send msg to each other through a mailbox & other communication channels.

- Shared Memory
  Threads within the same process can communicate through shared memory regions although synchronisation is needed to avoid race conditions.

Concurrency & Parallelism

↳ The ability to handle multiple task at once, potentially on a single CPU core by switching b/w tasks repeatedly.

P→ The simultaneous execution of multiple threads on different CPU cores. Modern ml make parallel execution possible, significantly improving performance.

DATE : / /20
PAGE NO.

Real-time Systems or O.S.

In real time O.S, thread management is particularly important as certain threads need to meet stripped timing constraints. Real time O.S. often support priority scheduling to ensure that critical thread is executed in a timely manner.

Multithreading Models

O.S. implements different models for managing multithreaded process.

- One to one
- Many to one
- One to many
- Many to many
- Hybrid

(1) Many to one

Multiple user level threads are mapped to a single Kernal level thread, while this is efficient the single thread blogging that the process can cause all thread to be blocked

(2) ~~Multiple user level~~ One to one

An each user level thread are mapped to a seperate Kernal thread.

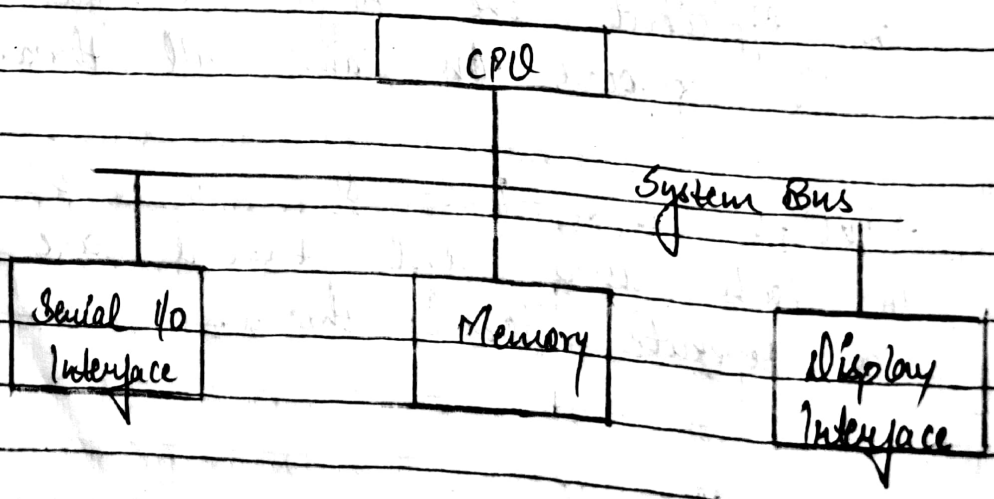
③ Many to Many
Multiple user level threads are mapped to multiple Kernal level threads.

④ Hybrid
Combination of many to many and one to one.

Concurrency

Concurrency in an O.S. is the ability to execute/run multiple process or threads at the same time. It is a key feature of modern O.S.
There are different concurrency work-

① Multithreading



DATE : / /20
PAGE NO.

There are 4 parts of concurrency control-
① Physical ② logical ③ Computation ④ Modularity

It improves source utilisation & system efficiency. It allows several ~~process~~ tasks to be processed at a same time either by running on seperate processor or through context switching on a single processor.
Concurrency is essential in modern O.S. design to handle multitasking, increase system responsiveness & optimize performance for user & application.

There are several motivations for allowing concurrent execution

① Physical Resource Sharing
Multiuser environment since H/w resources are limited.

② logical Resource Sharing
Share files.

③ Computation Speed up
More than 1 file execute in a single time.

④ Modularity
Divide system functions into seperation process.

4. Relationship b/w Process of O.S:

There are 2 types of processes —

OR

The process executing in an O.S. is one of the following two types:-

(1) **Independent Process**

Its states are not shared with any other process.

(i) The result of execution depends only on the I/P state.

(ii) The result of execution will always be the same for the same I/P.

(iii) The termination of the independent process will not terminate any other.

(2) **Co-operating Process**

Its state is shared along other process.

(i) The result of the execution depends on relative execution sequence & can not be predicted in advance.

(ii) The result of the execution will not always be the same for the same I/P.

(iii) The termination of the co-operating process may affect (terminate) any other process.

DATE: / /20
PAGE NO.

Critical Section Problem

The critical section problem is an essential concept in O.S. & refers to a situation in which multiple process or threads excess a shared resource like a variable, file or H/w device concurrently.

If these process doesn't synchronise their excess properly, it can lead to data inconsistency & unpredictable behaviour which is undesirable.

Integral Section Problem

2 key concepts: → Critical Section
→ Race condition

① Critical Section

It's a part of a program that accessed shared resources. Only 1 process can be in the critical section at a time to prevent race condition.

② Race condition

This occurs when multiple process try to modify shared data concurrently and outcome depends on the non-deterministic order of execution which can lead to inconsistent or erroneous result.

There are 2 key concepts of critical section → ① Race condition ② Critical Section

Requirement for a solution to critical section problem to be effective -

It must satisfy 3 conditions -

① Mutual Exclusion → Only 1 process can be inside the critical section at any time. If 1 process is executing in its critical section, all other process must be prevented from entering the critical section.

② Progress → If no process is executing in its critical section and multi-process wants to enter, one of them should be able to enter then its critical section in a finite amount of time. If the critical section is empty then some process must be allowed to enter.

③ Bounded waiting → After a process has requested entry to the critical section, there must be a limit on the no. of times other process can enter the critical section before the requesting process is allowed in. This avoid starvation, where a process could be indefinately delayed from accessing the critical

7-3-25

DATE : / /20
PAGE NO

* Hardware Synchronisation
This refers to a mechanism provided by H/w to manage concurrent access to shared resources in multiprocessor & multithreaded environments. It's crucial for ensuring data consistency & preventing race conditions. Hence, multiple process or threads access shared memory or variables.

* 3 H/w Synchronisation mechanisms
(1) Atomic operations
(i) Perform as a single, uninterruptable unit
(2) (ii) locks & mutexes

(ii) a) Spin locks
threads continuously check a variable until they require the lock.

(2) Test and Set (TAS)
A single atomic I/S to test & modify a lock variable.

(3) Special CPU I/s.
(i) Assembly → Test and set

TAS R_1, (lock); Tests & set the lock atomically

(ii) C

```
int compare & swap (int *add, int expected,
                    int new_value)
{
int old = *add;
if (old == expected) *add = new value
return old;
}
```

(ii) a) fetch & increement
used in ~~counting~~ managing shared counters in multiprocessor systems

b) load link or store conditions
prevents race conditions by ensuring that a store operation only succeed if no other modifications have occured since the load.

c) Memory barrier & fences
① Ensure current ordering of memory operations across different processors.
② Prevent out of order execution that could lead the race condition.

5) Interrupt disabling
① Temporary disabled CPU interrupts to prevent context switching & ensure atomicity.
② Suitable for small critical sections but not scalable in multiprocessor environments.

6 Mutex locks & Semaphores

They are both synchronisation mechanism used in O.S. to prevent race conditions & ensure proper access to shared resources in multi-threaded & multiprocess environments.

Mutex lock

① This is a locking mechanism that allows only 1 thread or process to access a shared resources at a time.

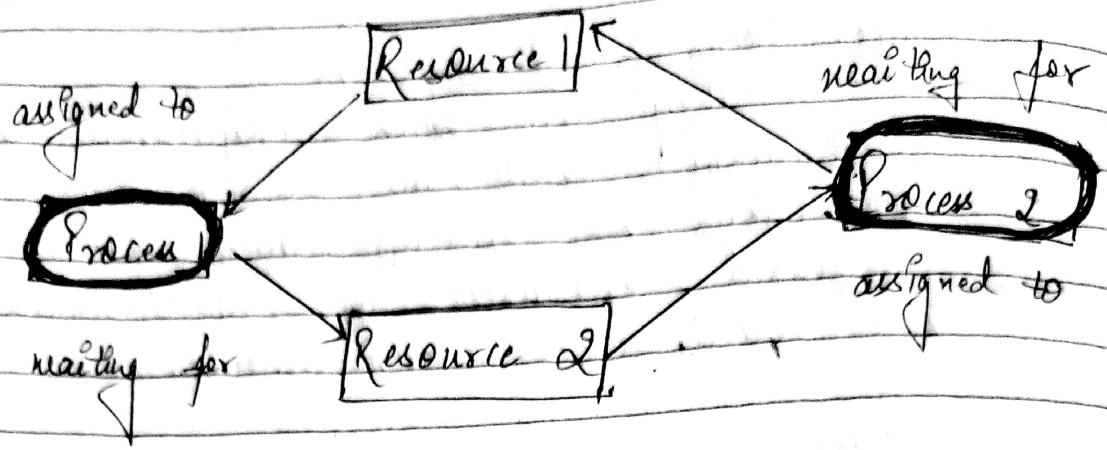
② Once a thread acquires the mutex, no other thread can access the resource until the mutex is

13-25

11 Deadlock

It's a situation in computer systems particularly in O.S. where a set of process are blocked, for each waiting for resources held by another process in the set, creating a cycle of dependency that can not be resolved without external intervention.

Deadlock is a situation in computing where 2 or more process are enable to process because each is waiting for the other to release resources.



P_0	P_1
wait A;	wait B
wait B;	wait A

1) # System Model

In a computer system, particularly in a multitasking O.S., process requires resources to complete their task I/P, O/P, CPU, CPU time, memory or data. The system model for deadlock consider the following -

(i) PROCESS

Entity that requests and releases resources (real-world object)

(ii) RESOURCES (tools)

These are finite and include CPU, memory, I/O devices etc.

(iii) RESOURCE ALLOCATION

Resources are allocated to process &

they hold these resources while executing.

(iv) REQUEST AND RELEASE
Process requests resources when needed & releases them when they are done.

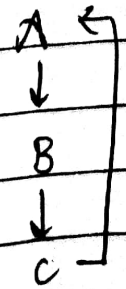
→ A deadlock occurs when -

① Mutual Exclusion
Atleast 1 resource is held in a non-sharable mode.

② Hold and wait
A process holding atleast 1 resource is waiting for additional resources that are being held by other resources.

③ No Preemption
Resources can't be forcibly taken from process. They must be release voluntary.

④ Circular wait
A set of process are waiting for resources in a circular chain, where each process holds atleast 1 resource & is waiting for a resource held by a next process in a chain.



DATE: / /20
PAGE NO.

26-3-25

(3) ## Methods for Handling Deadlock

There are several strategies to handle deadlocks. These methods are based on detecting, preventing & avoiding the concurrence of deadlock

(4) ## Deadlock Prevention

It involves ensuring that atleast 1 of the coffman condition is never meet. The goal is to structure the system in such a way that a deadlock can not be occur.

a) Mutual Exclusion

Since, some resources must be non-sharable, this condition can not be generally avoided.

b) Hold & wait

This can be prevented by requiring process to request all the resources they need before starting the execution. This method may use resource utilisation & result in longer waiting times but ensure that deadlock doesn't occur.

c) No preemption

This condition can be avoided by

allowing the system to forcibly take resources away from process.
eg. if a process holding some resources is waiting for additional resources, the system can preempt some resources & allocate them to another process.

d) Circular weight
This can be prevented by defining a linear ordering of resource types & requiring each process to request resources in increasing order of enumeration.

6) Deadlock Avoidance
It allows the system to make decisions dynamically about whether resource allocation can processed without causing a deadlock.

e) Safe v/s Unsafe

DATE : / /20
PAGE NO.

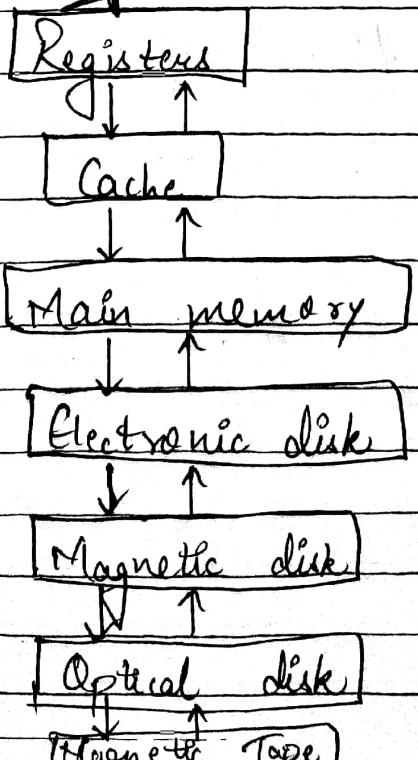
UNIT-3

* Memory Management

→ **MAIN MEMORY**

The main memory is a central to the operation of a modern computer. It is a large array of words or bytes, ranging in size from 100s to 1000s to Billions. It's a repository of rapidly available info. shared by CPU & I-O devices. It's a place where programs & info. are kept when the processor is effectively utilising them. It's associated with the processor so moving I/s & info. into & out of the processor with extremely fast speed. Also known as RAM. This is volatile. RAM stores the data when the power obstacle occurs.

diagram



DATE: / /20
PAGE NO.

* Why memory management is required?

(1) Allocate & deallocate memory before & after process execution.
(2) To keep track of used memory space by process.
(3) To minimal fragmentation issue.
(4) To proper utilization of main memory.
(5) To maintain data integrity while executing the process.

* Logical & Physical Address Space

Address space - ① logical Address Space
② Physical Address Space

(1) An address generated by CPU is known as logical address or virtual address. It can be defined as the size of the process. It can be changed.

(2) An address seen by the memory unit. eg. - I loaded into the memory address register of the memory is commonly known as physical address. It also known as real address. The set of all physical address corresponding to these logical address is known as physical address space. It's computed by MMU (Memory Management Unit).

DATE : / /20
PAGE NO.

The run time mapping from virtual to physical address is done by a hardware device MMU. The physical address always remain constant.

Static & Dynamic loading

Loading → Static
↳ Dynamic

loading is a process into a main memory is done by a loader. There are 2 different types of loading - Static & Dynamic.

① Static

It is basically loading the entire program into a fixed address. It requires more memory space.

② Dynamic

The entire program & all data of a process must be in physical memory for the process to execute. So, the size of a process is limited to the size of physical memory. To gain proper memory utilization, dynamic loading is used. In this, a routine is not loaded until it's called all routines are residing on this in a

DATE: / /20
PAGE NO.

relocatable load format.

Linking

→ Static → Dynamic

To perform a linking task, a linker is used. Linker is a program that takes one or more object files generated by compiler & combines them into a single executable file.

① Static → The linker combines all necessary program modules into a single executable program, so there is no routine dependency.

② Dynamic → The basic concept of dynamic linking is similar to dynamic loading. It stubs is including each appropriate library routine reference. Stub is a small piece of code, when stub is executed, it checks whether the needed routine is already in memory or not. If not available then the program loads the routine into memory.