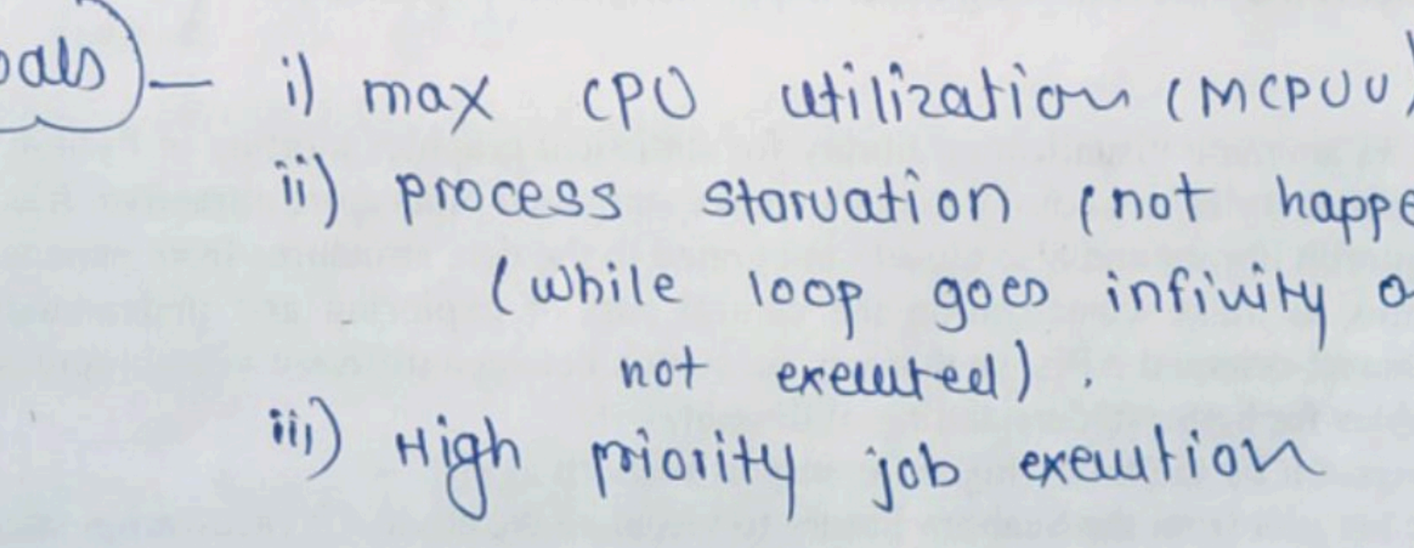
॥ श्री ॥ PCB - process control Block.

① operating system :-

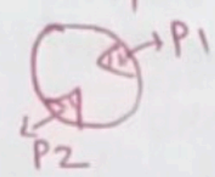
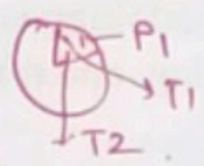
— piece of s/w that manages all resources of computer system both h/w & s/w.

— provides environment in which user can executes programs conveniently & efficiety by hiding underlying complexity of h/w & acting as resource manager.



Process :- program under execution. in RAM.
↓ compiled code.

Thread :- light weight process ← used to achieve parallelism by dividing process's task.
- small process that executes independently.
- multiprocessing ને multithreading
- e.g. multiple tab in browser

Multitasking	Multithreading
- more than one process	- more than threads એક કે process but એમાં multiple thread
- isolation or memory protection diff^th memory to each process 	- no such concept we same memory that of process 
- Process schedule	- threads schedule.
- execution of more than one task simultaneously.	- process divided into several diff^t subtask called thread. which has its own path of execution

Thread scheduling :-
- scheduled based on priority
- even though threads are executing within runtime, all threads are assigned processor time slices by OS.

Thread context switching	Process context switching
- OS saves current state of thread & switches to another thread of same process - doesn't includes switching of memory address space (But program counter, register & state included. - Fast switching - CPU cache state is preserved એક process ના thread માંથી બીજા thread પર use કરી શકાતો	- ~~switching~~ & switches to another process by restoring its state. - switching of memory address space. - slow switching. - flushed એક process ને માંથી બીજી process માટે પહેલા cache delete

Components of OS :-

kernel — user space (→ no h/w access)

Kernel
- interacts directly with h/w
- performs crucial tasks.
- Heart of OS
- very first part of OS to load on startup.

User space
- where appl^n. s/w runs
- interacts with kernel

Types:
- GUI — graphical user interface (direct interaction)
- CLI — command line interface (terminal)

Shell :- command interpreter
- receives command from user & gets them executed.

Functions :-

user ↓↑ kernel → OS
kernel ↓↑ H/W

Functions of kernel:

① process management
→ create, delete, suspend, resume process (both user & system)
→ mechanism - process synchronization, process communication

② memory management
- allocation / deallocation of memory
- keep track of currently using memory

③ File management
- creating, deleting files or directories
- mapping files into secondary storage

④ I/O management
- manage / control I/O operations & devices
- spooling → save करके रख लेते है data को पहले से
- Buffering → within 1 job
- caching → memory / web caching

Types :-

Types

Monolithic (सारे fun^c (4) kernel के अंदर centralized)
- all fun^c. in kernel
- disadvantages:
  - bulky
  - memory required high
  - less reliable (one module crash → whole kernel down)
- high performance (fast communicat^n.)
- e.g. Linux, Unix

s/w interrupt — एक process को signal, दूसरे work assign करना

Hybrid :-
- file mgmt → user space, rest in kernel
- speed & design → mono
- modularity & stability → micro
- e.g. macOS, windows

Micro (→ only puts major fun^c. in kernel)
- only fun^c. ① memory management → RAM
  ② process management → CPU
- File & I/O management in user space.
- smaller
- More Reliable
- More stable.
- performance slow - because
- overhead switching bet. user & kernel mode (um → km → um → km)
- e.g. L4 Linux

Nano — offer hlw abstraction without system services
↳ makes easy communication bet hlw & llw

Exo — follows end-to-end principle
allocates physical resources to applications

Problem Statement

- Build a machine learning model that predicts the type of people who survived the Titanic shipwreck using passenger data (i.e. name, age, gender, socio-economic class, etc.).
- Dataset Link: https://www.kaggle.com/competitions/titanic/data

Motivation

The motivation for the Titanic Survivor Prediction Model is multifaceted. It stems from the historical significance of the Titanic disaster, a moment that continues to captivate people's imagination and curiosity. The availability of a comprehensive dataset about Titanic passengers offers a unique opportunity for data exploration and analysis, merging historical events with modern data science techniques. Beyond historical interest, there is a human component; understanding the factors that influenced survival can help us empathize with the passengers' experiences. From a practical perspective, the project applies predictive analytics, estimating passenger survival based on various factors, which has implications for disaster management and risk assessment. It also has educational value, serving as an engaging way to introduce data science and machine learning. The study also seeks to reveal insights into human behavior in crisis situations, with potential relevance for disaster preparedness. Above all, it is a tribute to the memory of those affected by the Titanic disaster and highlights the ongoing advancements in data science and predictive modeling.

Objective

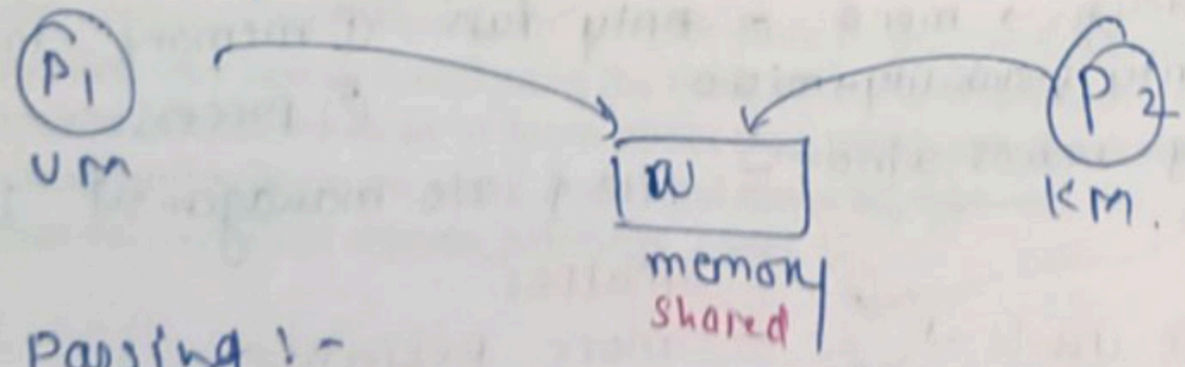
- To predict the survivors of the Testing Dataset by building a model using the Training Dataset of the Titanic Event.

Interprocess communication (IPC) :- → user mode & kernel mode communication

— two process executing independently, having independent memory space need to communicate

— done by → shared memory
msg passing

① Shared memory :—



P1 communicate करण्यासाठी shared memory मध्ये write करतो & P2 तिथून read करतो.

② Msg Passing :-

— OS provides method to communicate through channel

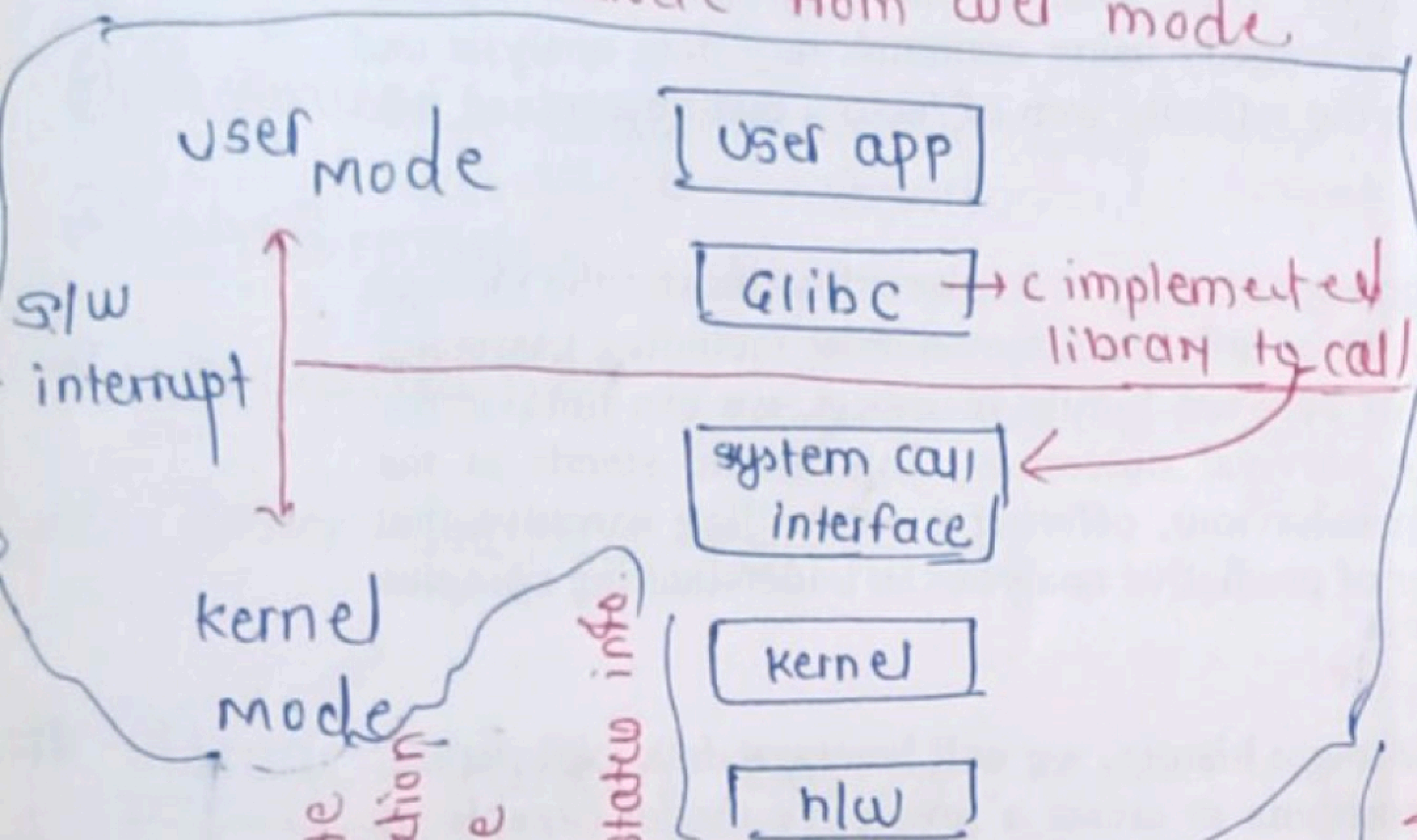
System calls in OS :-

① How apps interact with kernel?
→ using system calls. → implemented in C

Transition from user space to kernel space done by s/w interrupts.

system call :-

- user program can request a service from kernel which it does not have permission to perform.
- only way → process can go into kernel mode from user mode.



• Types

Types
- i) Process control
  - end, abort.
  - load, execute
  - create, terminate
  - set/get attributes,
  - wait for time
  - allocate & free memory
  - security
- ii) File mngmt
  - create, delete
  - open, close
  - read, write, reposition
  - // —
- iii) Device
  - request, release device
  - read, write, reposition
  - // —
  - logically attach / detach device
- iv) Info maintanance
  - get-time date
  - set-time / date
  - get/set system data
  - get/set process, file or device attributes
- v) Communication
  - create, delete comm. connection
  - send/receive msg
  - transfer status info
  - attach / detach remote device

How OS Boots up :-

i) PC on

ii) CPU utilizes & look for firmware prgm. (BIOS) in BIOS chip.

> Basic Input-output system chip -
> - Rom chip
> - found on mother board
> - allow access & set up computer system at basic level.

- in modern computer CPU loads: UEFI (Unified Extensible Firmware interface)

iii) POST : (Power on self-test)
- CPU runs BIOS which test & initializes system h/w → काही wrong असेल तर error देत.
- UEFI → do more than just initialize firmware.
  - tiny os.
  - Intel Mangmt Engine - provide variety of feature including IAMT - allow remote management of business PC.

iv) BIOS handoff responsibility for booting your PC to os's bootloader

BIOS

UEFI → efi ↓ partion मध्ये असते bootloader

v) Bootloader : MBR - हेच असते (Master Boot Record) old system मध्ये असत
- small program have large task of booting rest of OS (Boots kernel then, user).
  - window → use Window Boot Manager
  - Linux - GRUB
  - Macs - boot.efi.
- actual OS loads in this step

Difference Between 32-bit & 64-bi[t]

32 bit OS

- 32-bit register – (CPU के अंदर ऐसी location जहां पर actual computation होता है) – 64-b[it]
- can access 2^32 unique memory address i.e. 4 GB physical memory – 2^64 ... 17.17 ...
- process 32 bits data & info – 64 bi[ts] ...

> • **Stackoverflow Error:-**
> – when OS makes recursive call stack get floded & OS don't want to make entire memory bad
> – so it gives stackoverflow error
>
> • **out of memory / memory ensufficient error:-**
> – we goes on allocating heap without making it free when one process gets completed so it can be used for another process heaps get full. (C++ में)

Advantages of 64-bit over 32-bit:-

	64	32
① Addressable Memory	2^{64}	2^{32}
② Resource usage – upgrade system difference		doen't imapat
③ Performance	8 bytes in one instruction cycle	4 bytes
④ compatibility	run both 64-bit or 32-bit OS	only 32-bit
⑤ Better Graphic Performance		

How ...

→ conve[rt] ...

• **Steps**: ① load program & static data into memory (static → used for initialization; program under execution)
② Allocate runtime stack
③ Heap memory allocation
④ IO tasks ← i/p, o/p handling, error handling
⑤ OS handoffs control to main().

* process Architecture ↓

Section	used for
Stack	local variables, function arguments, return values
Heap	dynamically allocated variable
Data	Global & static data
Text	compiled code

• **Attributes:-**
① Identify process uniquely
② Process table
  ① all process tracked by OS
  ② each entry is PCB.
③ **PCB**: stores info/attributes of process

Storage Devices Basics:-

Primary memory:
- **Register** (close to CPU) – smallest unit storage; hold – instruction, storage address, any data; ↳ used – quickly access, store & transfer data.
- **Cache** → temporarily stores frequently used instruction – quick access less cost
- **Main memory** → RAM

Secondary memory:
- Electronic Disk
- Magnetic Disk
- Optical Disk
- Magnetic tapes

} storage media ↔ store data & programs.

① cost
- Primary storage costly ∵ ↗ registers most expensive due to semiconductors & labour (labour)
- secondary – " – cheaper

② Access speed:
① primary > secondary
② Register > cache > main memory

③ storage size:
- secondary has more space

④ volatility:- जो process computer का power off होने के बाद चली जाती है.
① primary more
② secondary non-volatile.

PCB: Process control Block.

PCB	
Process ID	→ unique identifier
Program counter	→ next instruction address of program
Process state	→ stores process state
Priority	→ based on priority process gets CPU time
Registers	→ process time slice expires current value of process specific register stored in PCB when process schedule to run value read from PCB & written to CPU registers
List of open files	
List of open devices	

Hard — offer hlw abstraction ....

Abstract

The "Titanic Survivor Prediction Model" represents a data-driven approach to examine the historical tragedy of the RMS Titanic and its passengers. This model is motivated by a profound interest in understanding the factors that influenced survival and, by extension, the implementation of predictive analytics in the context of disaster scenarios.

The objectives of this model encompass the analysis of a dataset containing passenger information, including demographics and ticket class, to identify patterns that could predict a passenger's likelihood of survival during the Titanic disaster. It seeks to derive insights into the critical factors that determined survival rates, ultimately shedding light on the human dynamics at play during this tragic event.

Leveraging machine learning techniques and data exploration, this model aims to make predictions about passenger survival, thus contributing to a broader understanding of historical events and the application of predictive analytics in disaster management. This abstract offers a glimpse into the intriguing journey of utilizing data science to revisit the past and extract valuable lessons that may inform future decision-making in critical situations.

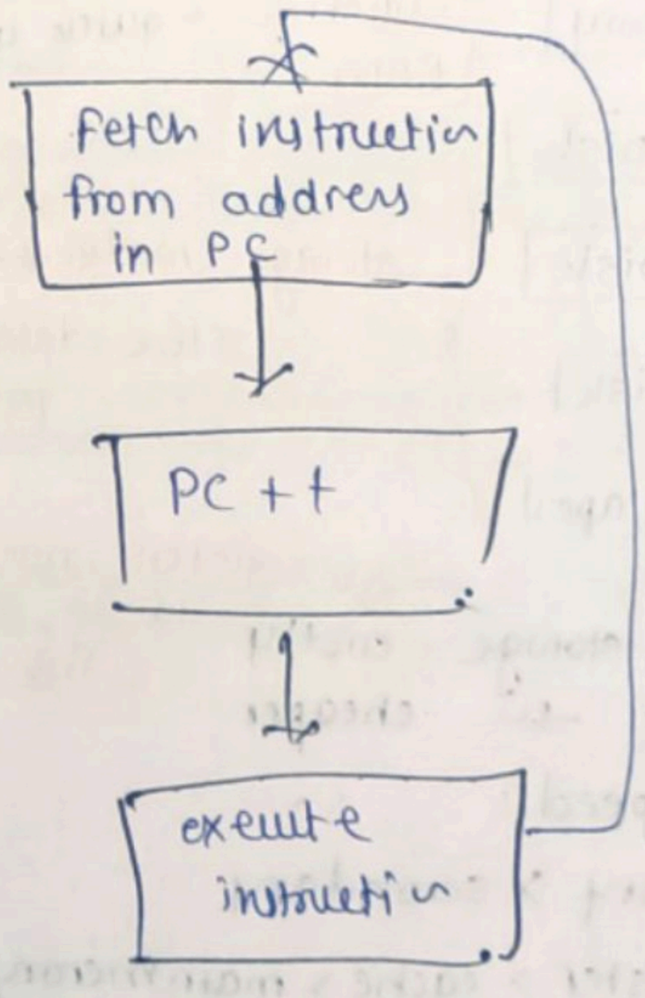
Sol :-

Stack overflow - unbinding stale

out of memory - disallocate unnecessary process

* PCB Register :- → Program counter (PC)

P_1 is executing & suddenly its time quantum expires it will save registers as it is then & after some when it gets time to execute it will come in action.



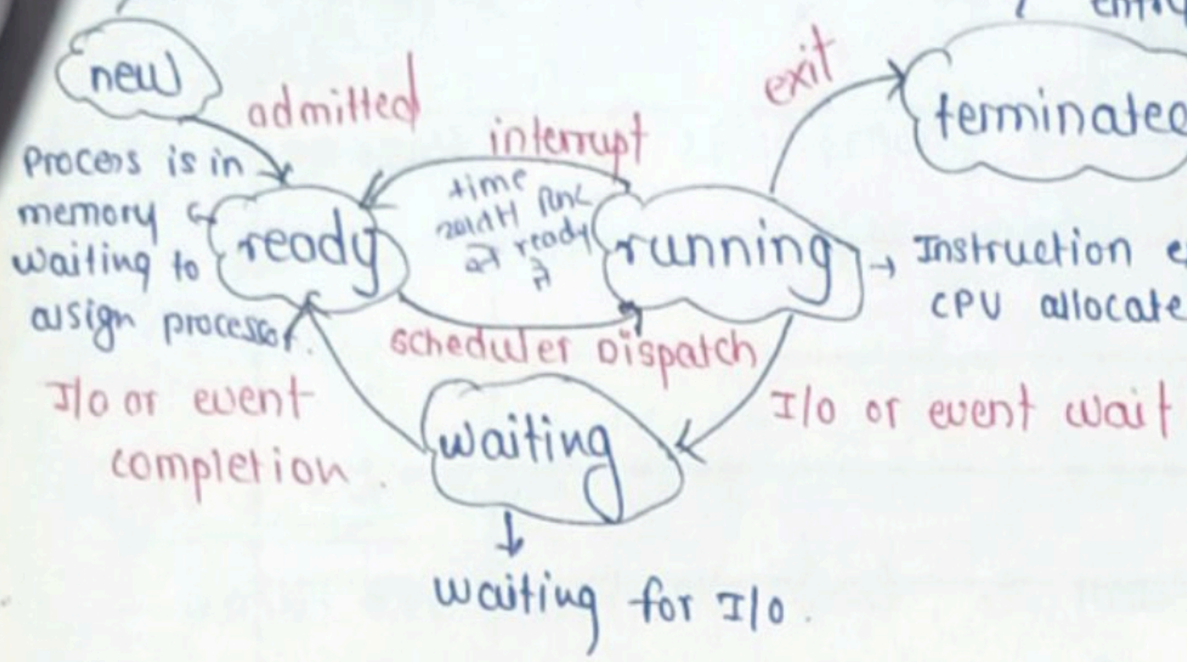
PC पुढिले भागाला कुठे आहे तिथ काय instruction आहे ती घेवून PC next instruction वर जावून आणि execute करवून आणी ची →

Process states :- / Process Queues :-

1) Process state :-

- as process executes it changes state.

process is being created.



The process finished execution PCB ↑ entry removed from process table

Instruction executing CPU allocated

2) Process Queue :-

1) Job Queue :

- process in new state
- present → secondary memory
- Job scheduler (long term scheduler (LTS)) picks process from pool & load in memory for execution.

2) Ready Queue :-

- process in ready state
- in main memory
- CPU scheduler (short Term scheduler) picks process from ready queue & dispatch to CPU.

③ Waiting Queue :-

- process in wait state.

- **Degree of multiprogramming :-**
  - no. of process in memory
  - LTS controls it.
- **Dispatcher :-**
  - gives control of CPU to process selected by STS (CPU assign)

process table called reaping

- entry removed only if parent process reads exit status of child process.

Hence child process remains zombie till it is removed from process table.

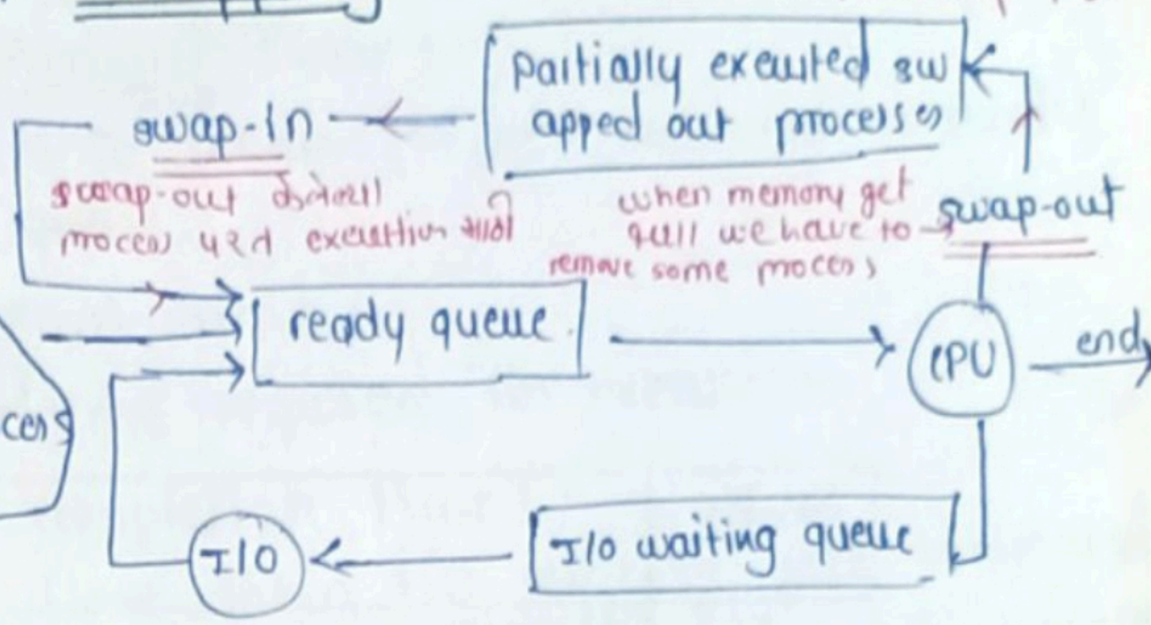
swapping :-

Time sharing system - medium term scheduler

reduce degree of multiprograming

swap-out / swap in

MTS



- remove process from memory to reduce degree of multiprogramming.
- This removed get back into memory & its execution continued where it left off. called swapping.
- done < swap-out / swap-in > by MTS
- necessary : ① to improve process mix
  ② changes in memory requirement
  → overcommited memory requiring to freed up.
- process swapped temporarily out of main memory to secondary storage & make memory available to other process
- after some time swap back process from secondary to main

context switching :-

- switching requires < saving state of current process in PCB / restore state of diff process.

kernel does it

orphan process :-

- process < whose parent process terminated / it is still running
- adopted by init process. (making parent)
  - ↳ first process of OS.

zombie / Defunct Process :-

- execution completed but still have entry in process table.
- occur for child process, as parent process still needs to read its child exit status once this done using wait system call zombie process eliminated from

Convoy effect :- Intro to process scheduling :-

1) Process scheduling :-
- the process of picking process from ready queue (determining which to pick) then scheduling it at CPU
- by switching CPU among processes OS can make computer more productive.

2) CPU scheduler :-
- when CPU becomes ideal, os must select one process from ready queue to be executed
- done by STS

Non-primtive
- once CPU allocated to process, process keeps CPU until it releases CPU by: (i) terminating (ii) switching to wait state. મરે કોલી 12
- starvation (long BT process starve less BT process)
- Low CPU utilization

primtive scheduling
- CPU taken away from process after time quantum expires along with (i) terminating & (ii) switching to wait state
- less starvation
- High CPU utilization

CPU ideal નહીં રહે.

Goals of CPU scheduling
- max CPU utilization
- min Turnaround time (TAT) = CT - AT = (CT - AT)
  time taken from 1st time process enters ready queue till it terminates
- min wait time (WT)
  (WT = TAT - BT)
- max through put
  No. of process completed per unit time
- min Response time
  duration bet. process getting into ready queue & CPU first time

i) ~~Throu~~.
i) Arrival Time :-
- time when process arrived at ready queue.

ii) Brust Time :-
- time required for execution

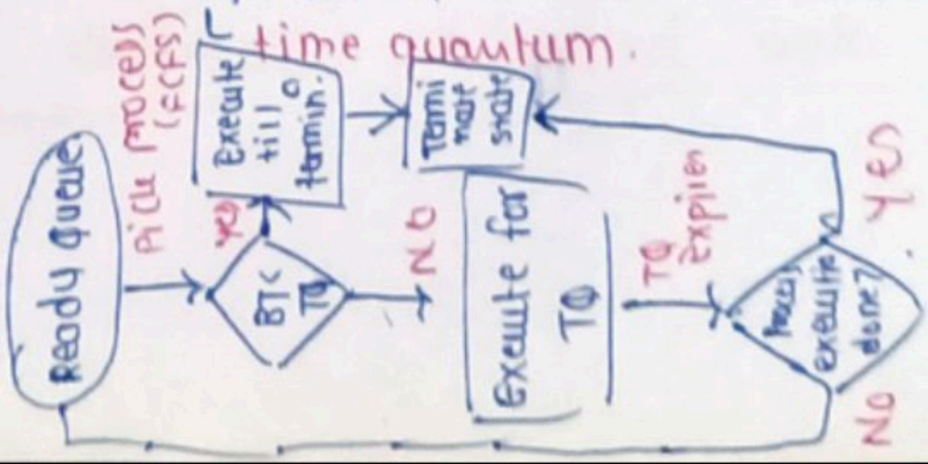
iii) completion time :-
- time taken till process gets terminated

Convoy effect :-
- situation where many processes, who need to use a resource for a short time, are blocked by one process holding that resource for long time.
- this cause poor resource management
- longer BT → effect on average WT of diff. process

(1) SJF :- (Non-preemptive) → convoy effect if 1st process has large BT
- process with least BT get CPU.
- process starvation
(2) primtive - less starvation
- no convoy effect

(2) Priority scheduling :-
- Assign priority to each job
- starvation - કાઢી
  (1) Ageing
  (2) gradually increase priority of process

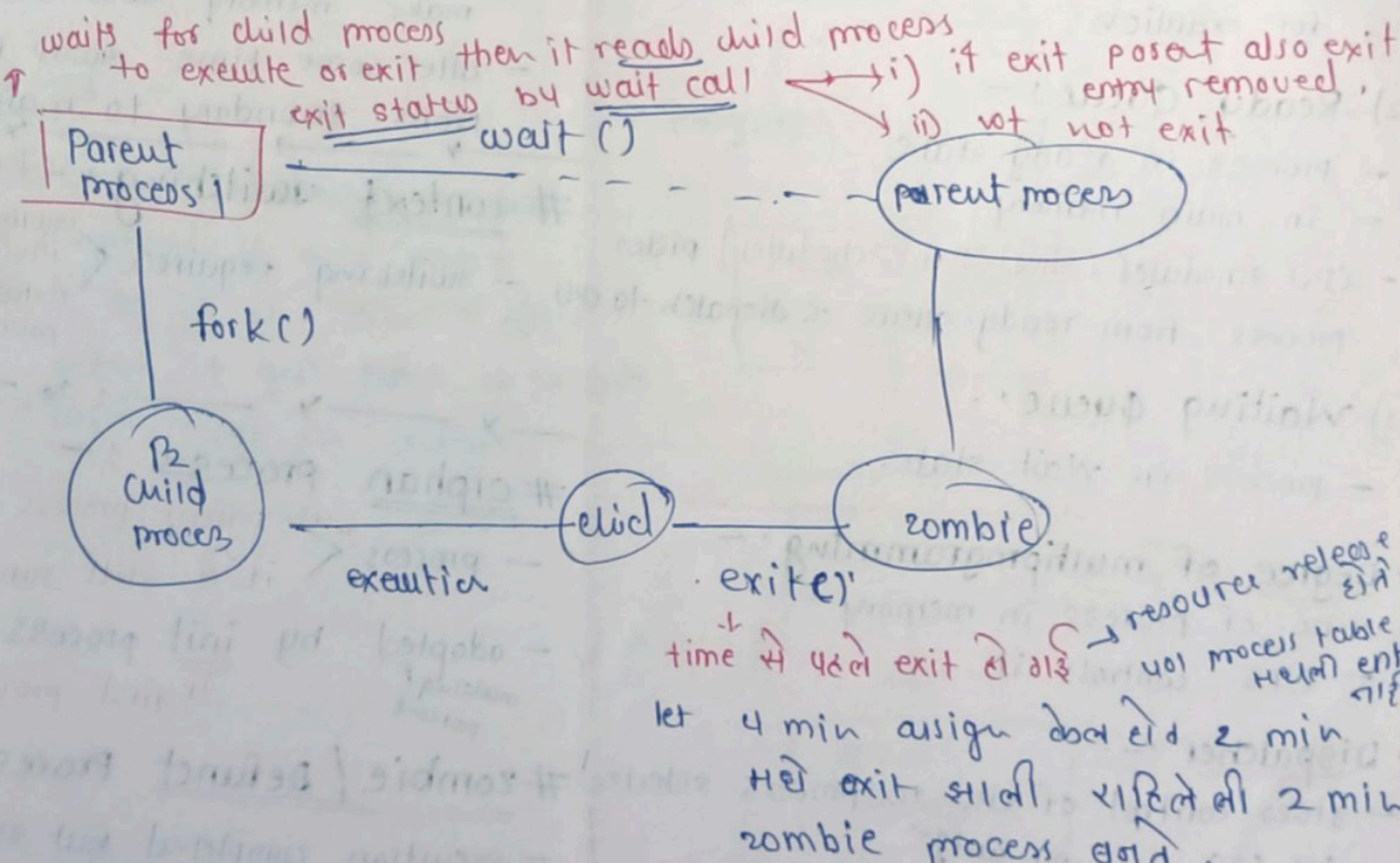
(3) Round Robin :-
- like FCFS but preemptive.
- designed for time sharing system
- low starvation as no process goes to wait forever.
- if TQ small more context switch



CONTENTS

Sr. No	TITLE	Page no
1.	Abstract	5
2.	Introduction	6
3.	Problem Statement	7
4.	Motivation	7
5.	Objectives	7
6.	Theory	8
7.	Conclusion	18
8.	References	18

zombie process :-

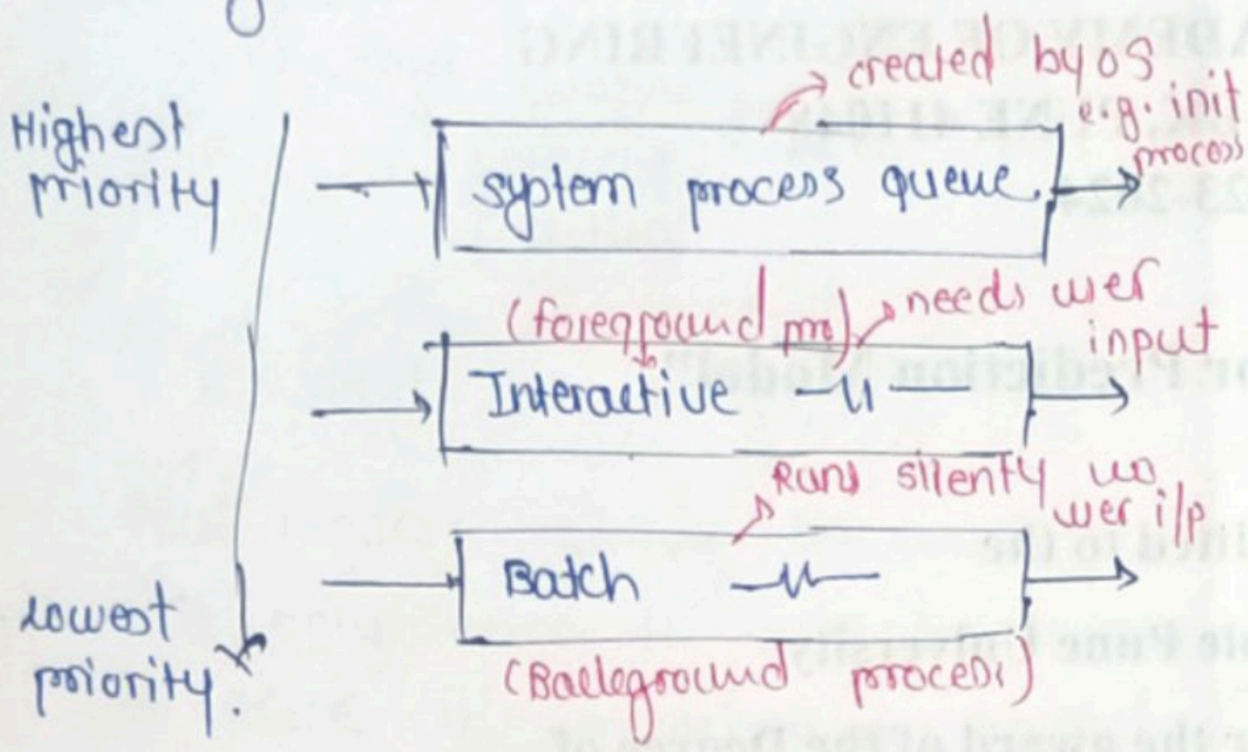


Multi-level queue scheduling :- (MLQ) :- fixed priority scheduling.

① ready queue → divided into multiple queue based on priority

② process permanently assigned to one of the queues

③ each queue has its own scheduling algo.



- problem : ① only after completion of all processes from top level ready queue, the further will be scheduled.
② convoy effect present.

multi-level feedback queue sched. :-
- multiple sub-queues
- all process to move bet. queues
- less starvation
- flexible
- separating process based on BT

BT ↑ → moved lower queue

- Aging used to prevent starvation.

concurrency :- execution of multiple instruction sequences at a time.

will single CPU system would gain by multithreading technique?
→ i) never
ii) as two threads have to context switch
iii) won't give any gain.

Thread control Block (TCB)

How each thread get access to CPU?
① thread → own program counter
② OS schedule thread based on thread scheduling algo
③ OS fetch instruction using PC.
→ ज्यातले thread, त्याचे state

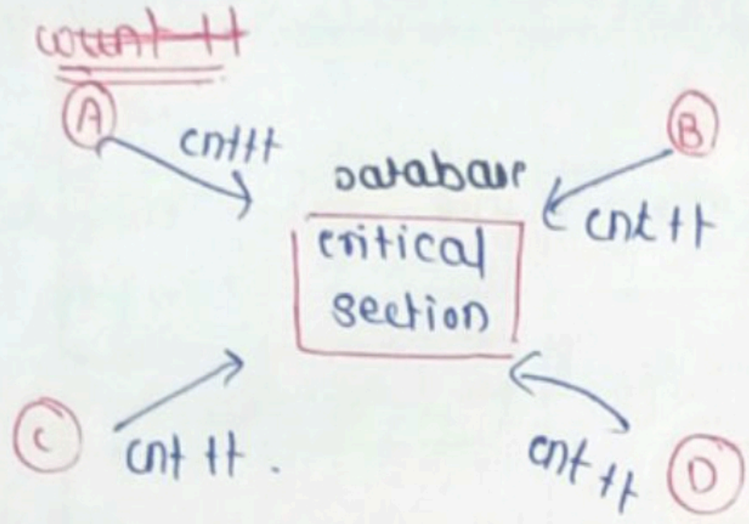
e.g.
```
#include <thread>
using namespace std;
void taskA(){
  for(int i=0; i<10; i++)
  {
    sleep(1);
    printf("TaskA: %d\n", i);
    fflush(stdout);
  }
}
void taskB(){
  ~~~
}
int main(){
  thread t1(taskA);
  t2(taskB);
  t1.join();
  t2.join();
  return 0;
}
```
join() - याचमुळे wait करतोय main thread ला नाहीतर main thread exit होईल नाहीतर error देईल.

critical section :-
- process synchronization → maintain consistency of shared data.

i) Critical section (C.S)

- segment of code where process / thread accessed shared resources & perform write operations on them.
- since they execute concurrently, any process can be interrupted mid-execution.

- e.g. ~~count++~~



- four request coming for count++ but only once it will be increased so there is inconsistency. called race condition

Thread scheduling issue:-

a) Race condition:-

- when two or more threads can access shared data & try to change it at same time.
- as thread scheduling algo can swap bet. threads at any time

cond → i) Mutual exclusion iii) Bonded wait
ii) progress

• Solution :-

att → atomic<int>

① Atomic operation - execute in one CPU cycle.

② Mutual Exclusion using locks or mutex. → no simultaneous execution (sequential)

③ Semaphores.

→ simple flag variable is not used to solve race cond. → mutual exclusion होता पण progress नाही

→ peterson's sol. used to avoid race condition but only for 2 process or threads.

Mutex / Locks :-

- to implement mutual exclusion & avoid race condition.
- allow only one process / thread access to C.S.

• Disadvantages :-

① contention - infinite waiting of thread if one dies who acquired lock.
② Deadlock
③ Debugging
④ starvation of high priority threads

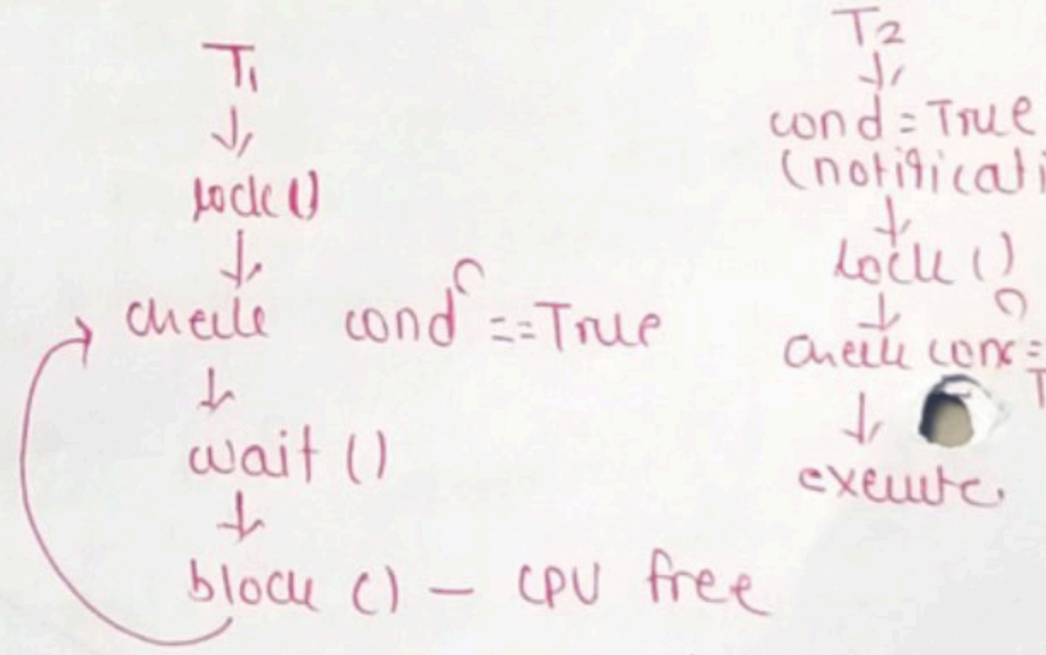
CS मध्ये असे low priority thread ने lock acquire केला आहे तर तो जो पर्यंत unlock होत नाही तो पर्यंत high priority waiting

conditional variable :-

- works with lock
- let thread wait until certain cond. occur.

→ to avoid busy waiting

- no contention



semaphore :- (no synchronization)

- synchronization method.
- integer = no. of resource.
- multiple threads execute CS concurr entry

Type

Binary
- value either 0/1
- i) Ako, locks

counting
- range over unrestricted domain
- control access to given resource

Producer consumer Problem :-

① produer thread → produces data

② consumer thread → consumes data.

① synchronization bef. producer & consumer (જેવો producer insert કરે ત્યારે consumer ને કાઢવો નહીં પડીએ)

② producer not invert data → buffer full.

③ consumer not pick/remove → buffer empty.

Sol^c :- ① semaphore :-

i) m, mutex → binary sema.
→ used acquire lock on buffer.

ii) empty → counting sema.
initial value n
↳ track empty slots.

iii) full → track filled slots
initial : 0

- producer

```
do {
wait (empty); wait until empty>0
wait (mutex); then empty--
//CS add data to buffer
signal (mutex);
signal (full); //increment full→value
} while(1)
```

- consumer

```
do{
wait(full); until full >0, then full--
wait(mutex);
//remove data from buffer  → C.S.
signal(mutex);
signal(empty); //increment empty
} while (1)
```

Reader Writer problem :-

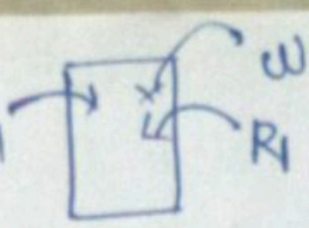
→ synchronization :- एक बार में एक ही writer write कर सके लेकिन read सब कर सके

① Reader Thread

② writer Thread

③ if >1 readers are reading no issue

④ if >1 writers OR 1 writer or some other thread (R/W) → Parallel execution
↳ race cond^n occur
data inconsistent ↳ data inconsistency



both writing at same position data become inconsistent

→ if there is W, writing & R_1 reading then also problem occur as after reading if writer updates something (inconsistent reading)

solution :-

- wing semaphore :-

① mutex - binary sema
- to ensure mutual exclusion when read count (RC) updated.

② - no two threads modify RC at same time

③ wrt - binary sema
- common for both reader/writer.
- database access mutually

④ RC - integer initialize = 0
- track how many readers are reading

① Writer sol^c :-

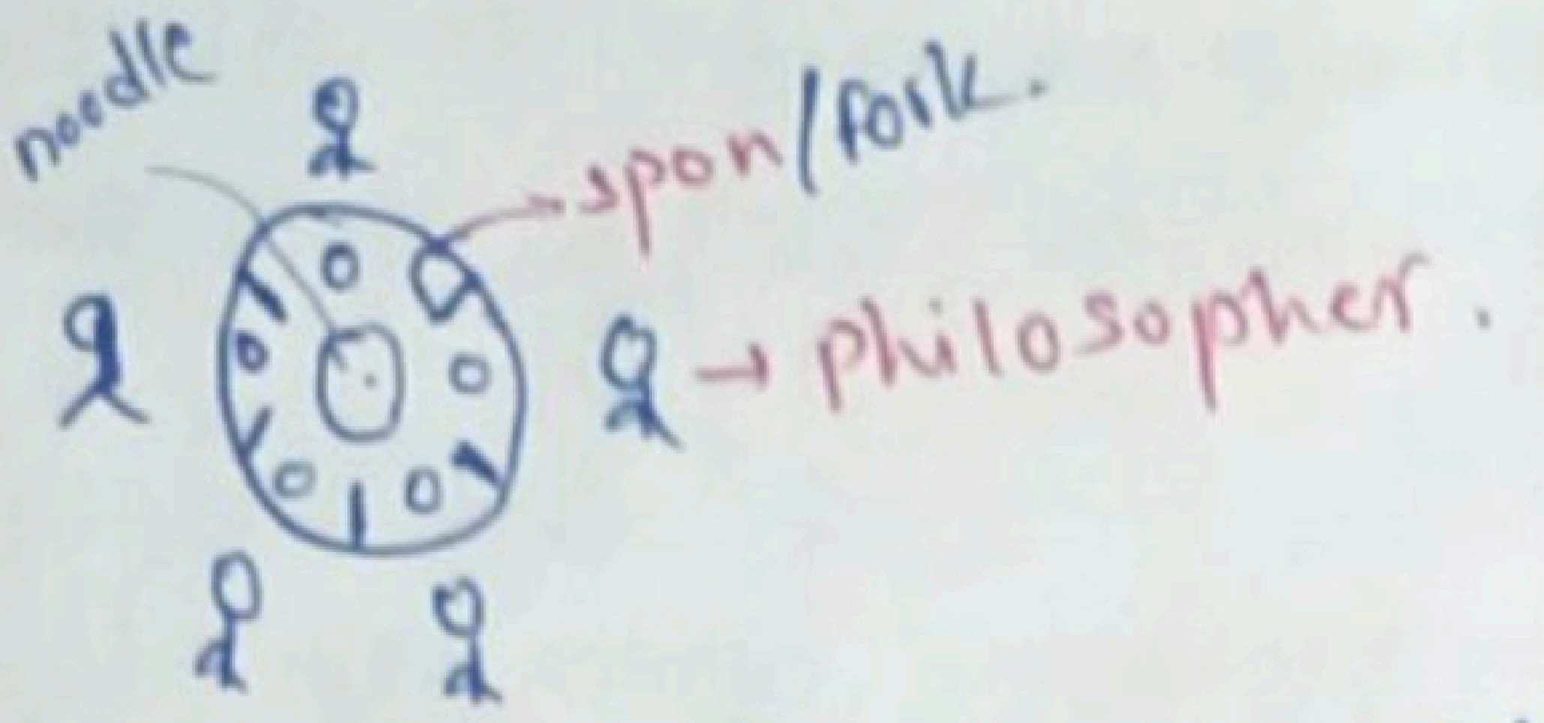
```
do{
wait (wrt);   wait → then lock
//do write operation → C.S.
signal (wrt);   signal it
} while (true);
```

write operation

② Reader sol^c :- to avoid race cond^n. when RCtt.

```
do{
wait (mutex); - to mutex readcount variable
rctt;
if (rc==1)   check writer આવે કે નહીં C.S. માં જો હોય તો reader block થઈ જાય
wait (wrt); ensure no writer can enter if there is even one reader.
signal (mutex);
// C.S.: Reader is reading
wait (mutex);
rc--;   → a reader leaves
if (rc==0) → no reader left
signal (wrt); → writer can enter
signal (mutex); → reader leaves
} while (1);
```

Dining Philosopher Prblm :-



- 5 philosopher → spend life in two states
- i) Thinking ii) Eating
- sit on circular table with 5 chairs bowl at centre of table & 5 single forks / spoon

i) Thinking :- at this time he doesn't interact with other.

ii) Eating :- when hungry – tries to pick up 2 spoons adjacent (left & right)
- he can pick one at time.

- one can't pick a fork if already taken
- when – has both fork at same time.
  ↳ he eates without releasing forks.

Solution :-

1) using semaphore :-

① each fork – binary sema.
semaphore fork [5] = {1}

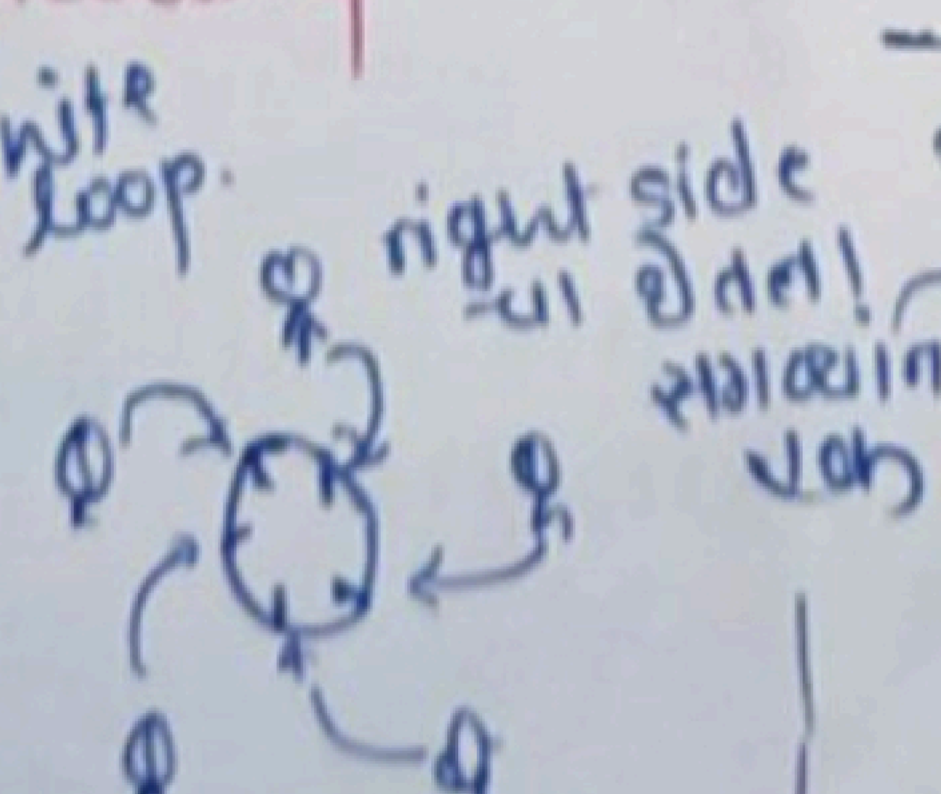
② wait () → fork[i] = ph[i] → to acquire fork

③ Release () → fork [i] → particular fork
↳ by calling signal() release.

```
do{
wait(fork[i]);        → ①  → ②
wait( for[(i+1) %5]);
//eat
signal ( fork [i]);
signal (fork [(i+1) %5]);
//think
} dowhile (1)
```

(i+1) → ... आने के लिए

- इसमे दो philosopher simultaneously eating नहीं कर सकते
- major problem : deadlock ← infinite loop.
- પણ prblm થાય છે તર પ્રત્યેકોલા wait કરવા છે કુસરા ક્યારે ફ્રી કરતોય તેટલી તો ચાલાક



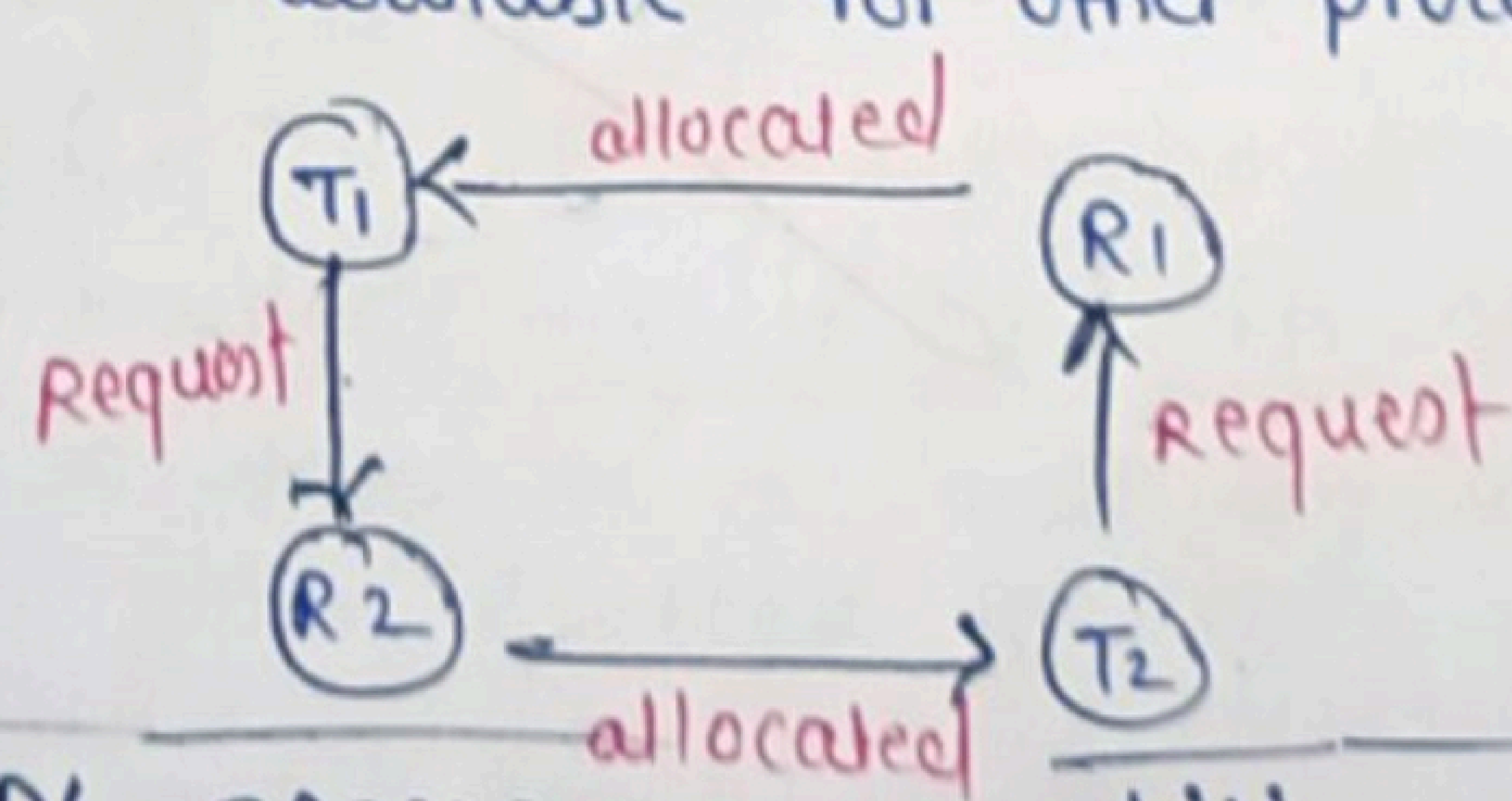
- avoiding deadlock :- methods:
  i) allow at most 4 philosopher to sit simultaneously
  ii) allow philosopher to pick up his fork only if both are available & to do this → he must pick them up in critical section (automatically)

③ odd-even rule :-
- odd phi – lift first left fork. then his right fork
- even phi – first right then left

Deadlock :-

- several processes competing for finite no. of resource
- waiting process never able to change its state because the resource it has requested is busy (forever). called Deadlock (DL).
- eg. of resource :- memory space, CPU cycles, files, locks, I/O devices

• How process | thread utilizes resource
→ a) Request :-
  - request resource (R), if R free lock it else wait till it is available
  b) Use
  c) Release :-
  - release resource instance & make it available for other processes.



• DL necessary condition :- – to happen DL
  a) Mutual Exclusion :-
  - only one person at a time can use resource
  - another process request that resource it have to wait until it get released

b) Hold and wait :-

- process hold at least one resource & waiting to acquire additional currently held by other

c) No preemption :-

- resource voluntarily released by process after completion

d) circular wait :-

- i.e. {P_0, P_1, P_2} P_0 waiting for P_1's resource, P_1 → P_2's & so on.

- Methods to handle deadlock :-

a) use protocol to prevent & avoid

b) allow system to enter DL state detect it & recover.

c) deadlock ignorance (ostrich algo).

- ignore problem & pretend DL never occur.

- Deadlock Prevention :- (→ 4 cond. માંથી કોઈપણ એક ગમે તે fail કરાવી)

a) Mutual Exclusion :-

i) we lock (no sharable resource માટે ચાલે)

ii) sharable resource (Read-only files). no lock used can be accessed by multiple process.

iii) we can't prevent DLs by denying mutual exclusion.

b) Hold and wait :-

- to prevent we ensure that whenever process request resource it doesn't hold by any other

① allocate resource before execution

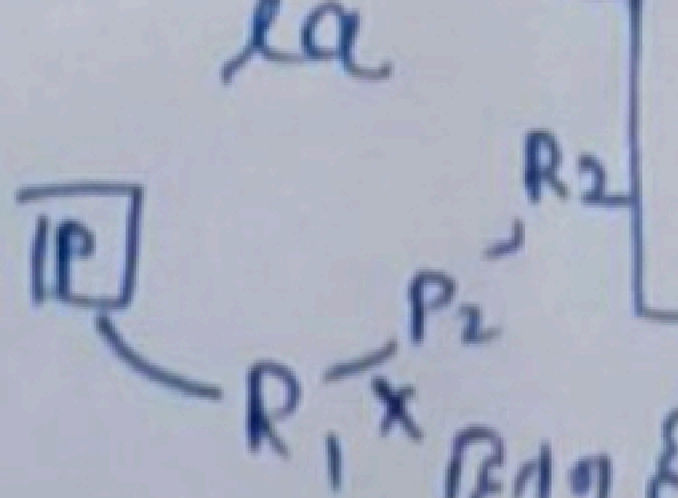
② allow process to request resources only when it has none

c) No preemption :-

- live lock. → બધા એક lock ની રાહ જોઈ resource લઈ

- check allocated wait

not → allocate.



ક્યાંથી preempt કરાવવાની ખાતરી P_1 ના દેવાની કામ આવતું પડે

d) circular wait :-

- locking on resource should be like both try to lock R1 then R2.

- by this way which ever process first locks R1 will get R2.

Deadlock Avoidance :-

- Kernel has info. about :-

- current state of process :-

① no. of process

② max. need of resources for each process

③ currently allocated amt of resource

④ max amt of each resource

- schedule process & its resource allocation → DL never occur.

• safe state	• unsafe state
- allocate resource to each process to avoid DL (up to max)	- OS cannot prevent process from requesting resource that DL occur

Banker Algorithm :-

- check resource allocation ⟶ safe state

i) if yes resource allocated to process

ii) if not → process wait till resource get released.

Deadlock Detection :- (→ જ્યારે prevention & avoidance implement નથી કરેલ ત્યારે)

a) single instance of each resource type (wait-for graph method) :-

- deadlock → if cycle in graph.

- to detect DL → maintain wait-for graph & look for cycle in it.

b) multiple instances for each resource type :-

① Banker Algo.

Recovery from DL :-

a) Process Termination :-

i) abort all DL process.

ii) abort one process at a time until DL cycle is eliminated.

b) Resource preemption :-

① successively preempt some resource from processes & give to other process until DL cycle is broken

```
class FizzBuzz{
private:
    int n;
    mutex m;
    conditional_variable cv;
    int i;
public:
    FizzBuzz(int n){
        this->n = n;
        this->i = 1;
    }
void fizz(function<void()> printfizz){
    while(i<=n){
        unique_lock<mutex> lock(m);
        while(i<=n && (((i%3==0) &&
            ((i%5==0))==0){
            c.wait(lock);
        }
    if(i<=n){
        printfizz();
        ++i;
    }
    cv.notify_all();
}
```

Memory Management :-

logical	physical
- generated by CPU	- loaded into physical memory.
- address of an instruction or data used by process	
- we access it	- user never
-	- computed by memory mang unit. (MMU).

How to satisfy a request of a of n size from list of free holes?

① First Fit :-
- allocate 1st hole i.e. big enough
- simple & easy
- Fast & less time complexity

② Next Fit :-
- start search from last allocated hole
- same advantage of First fit

③ Best Fit :-
- allocate smallest hole that is big enough
- lesser internal fragmentation.
- major external
- slow - iterate whole free holes list

④ Worst Fit :-
- allocate largest hole i.e. big enough.
- same point
- leave larger holes.

Paging :-
- permits physical address space of a process to be non-contiguous
- avoid external fragmentation & need of compaction.
- divides physical memory into fixed-sized blocks called "Frames"
- divide logical memory into same size block "Pages" (Page size = Frame size)