L1:1 Introduction to Operating System and its functions.

- Operating System (OS)

An Operating system is system software that manages computer hardware and software resources and provide services for computer programs.

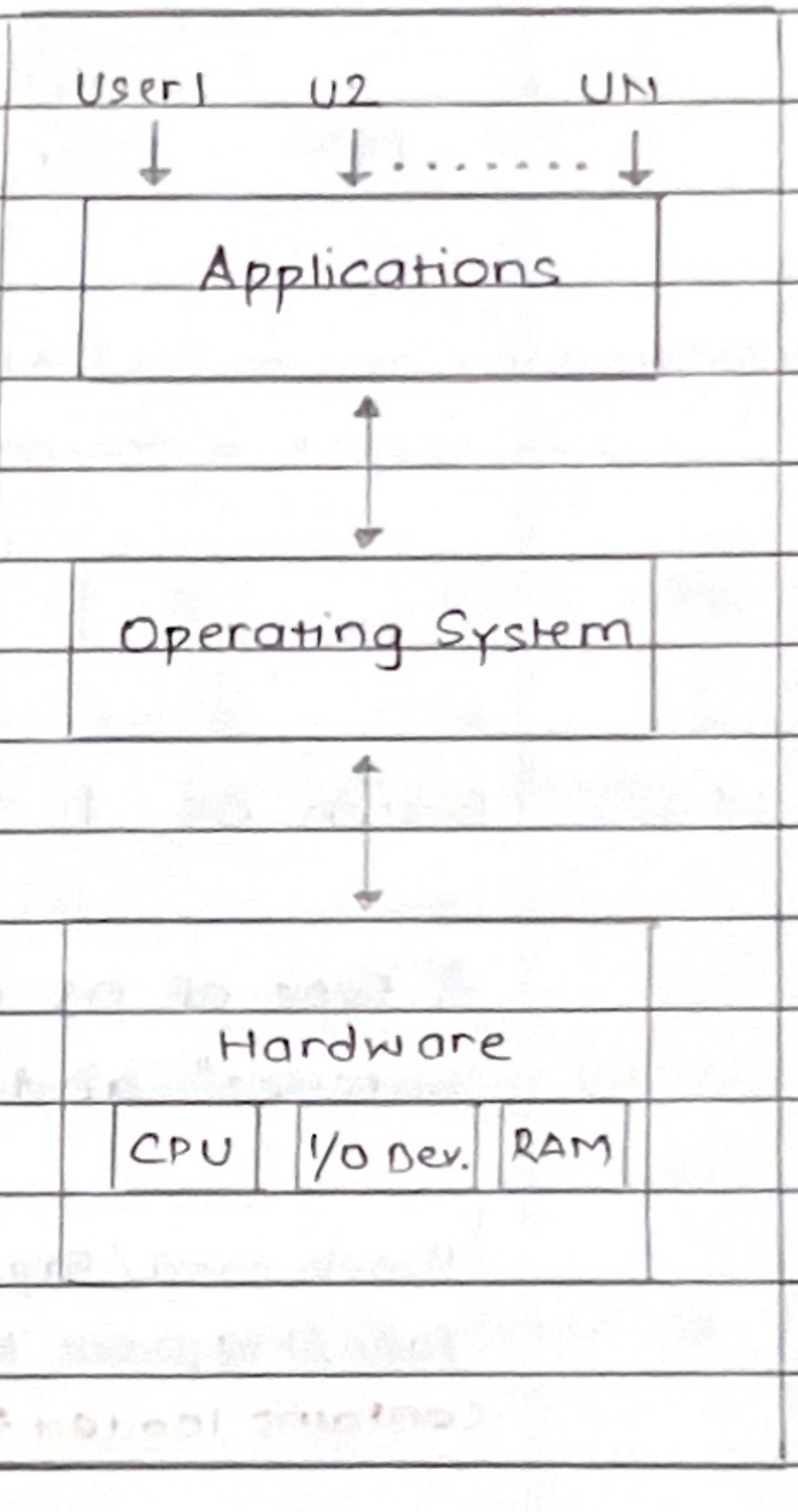
It acts as an intermediary between users and hardware.

- Functions of an OS:

1. Process Management

Handles process execution using CPU scheduling algorithms.

Manages process creation, termination, synchronization and prevent deadlocks



2. Resource Management

Manages hardware resources (CPU, memory, I/O) for efficent use, especially in multi-user or parallel processing environments.

3. Storage Management

Uses file system to organize, store and retrive data permanently on storage devices.

Managing file directories, access permissions and ensure data integrity

4. Memory Management

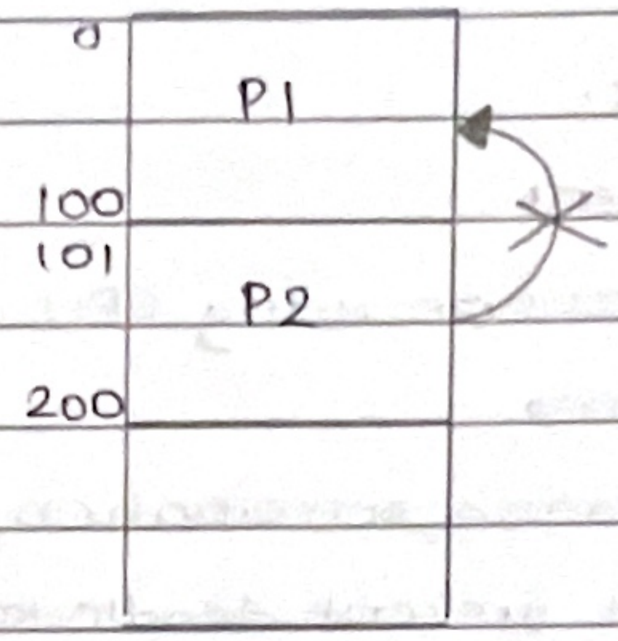
Allocates and deallocates RAM efficently

Using technique like paging, segmentation and virtual memory to optimize memory usage.

6. Security and Privacy

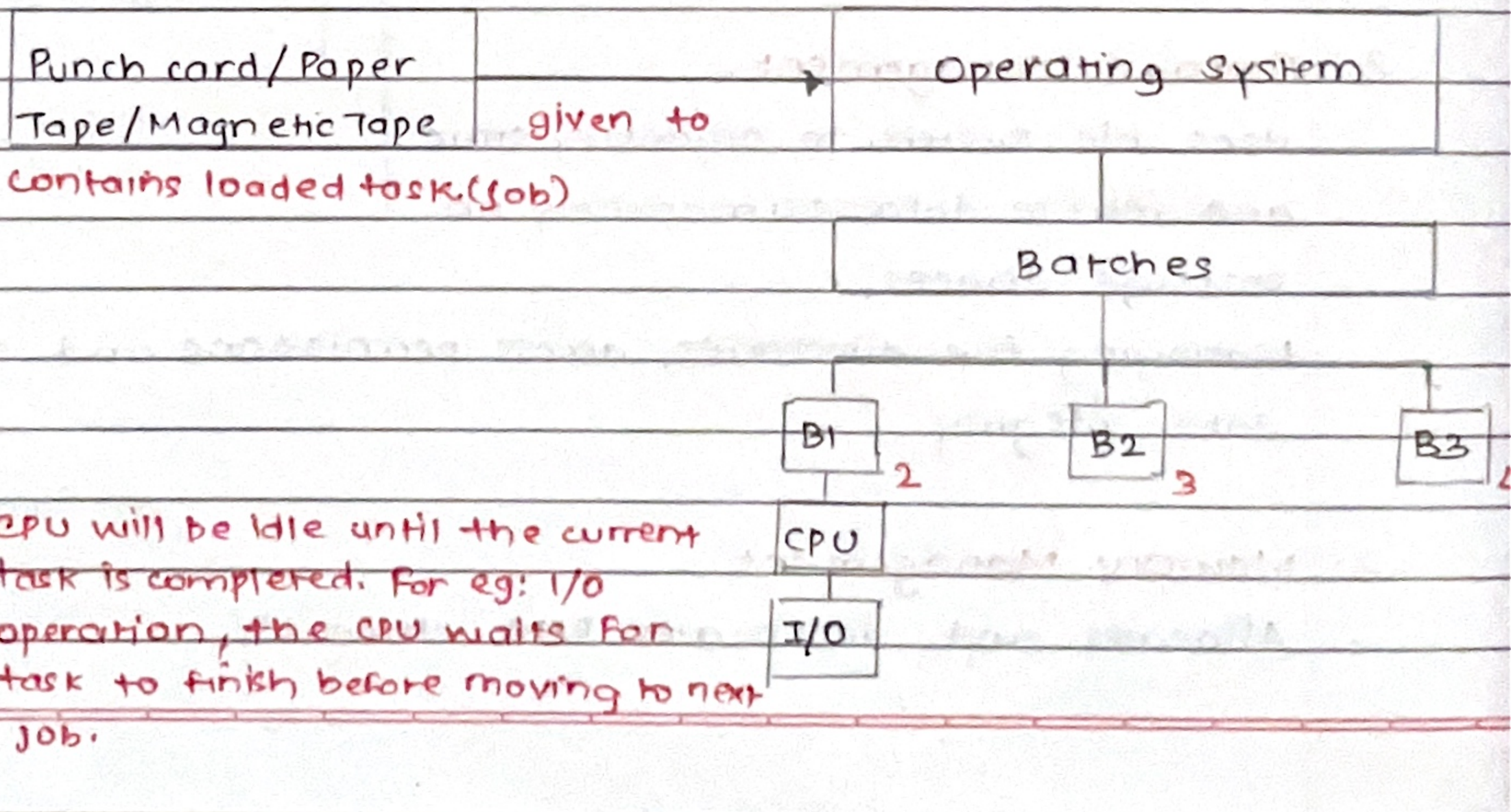
Authenticates user and protects data from unauthorized access

Ensures proper isolation to prevent interference between process (eg: if block P2 tries to access P1's data)



1:2 Batch OS | Types of Operating System

A type of OS where similar tasks (jobs) are grouped into "batches" and executed together without user interaction



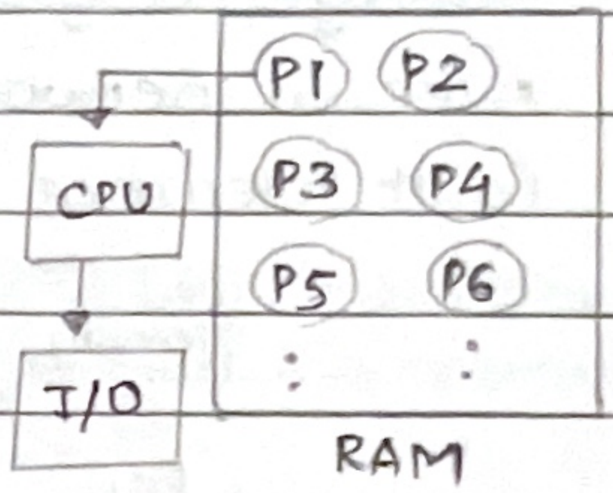
Page No.
Date

L1·3 Multiprogramming OS & Multitasking OS | Types of OS

- **Multiprogramming OS**

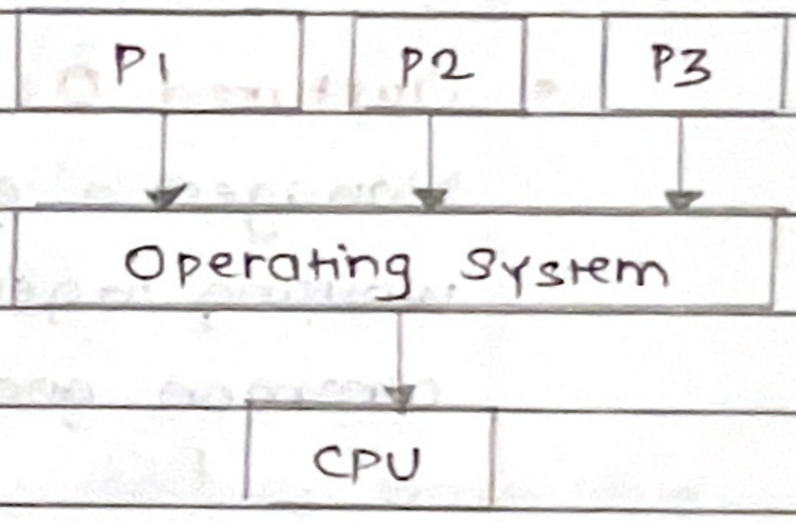
An OS that allows multiple programs to reside in memory simultaneously. It is non-preemptive scheduling. (runs until voluntarily gives up CPU)

When one program is waiting for I/O operations, the CPU switches to another program, keeping CPU busy.



- **Multitasking OS**

An OS that allows multiple tasks (processes) to run concurrently by rapidly switching them. Uses preemptive scheduling (forcibly switching between process)



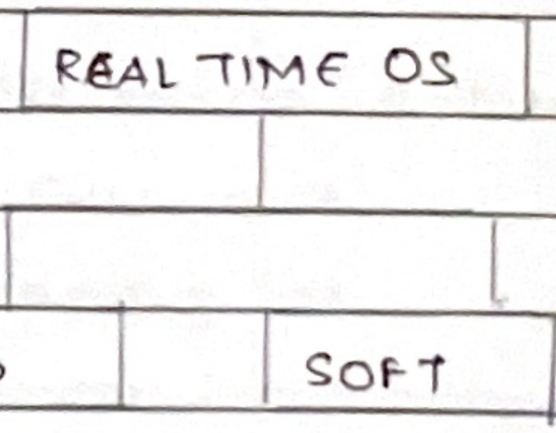
L1·4 Real Time, Distributed, clustered, embedded OS | Types of OS

- **Real Time OS**

An OS designed to process data and execute within strict time constraints.

HARD : Task must be completed within an guranteed deadline (eg airbag dev.)

SOFT : Task are prioritized, but missing a deadline is acceptable. (eg. video streaming)

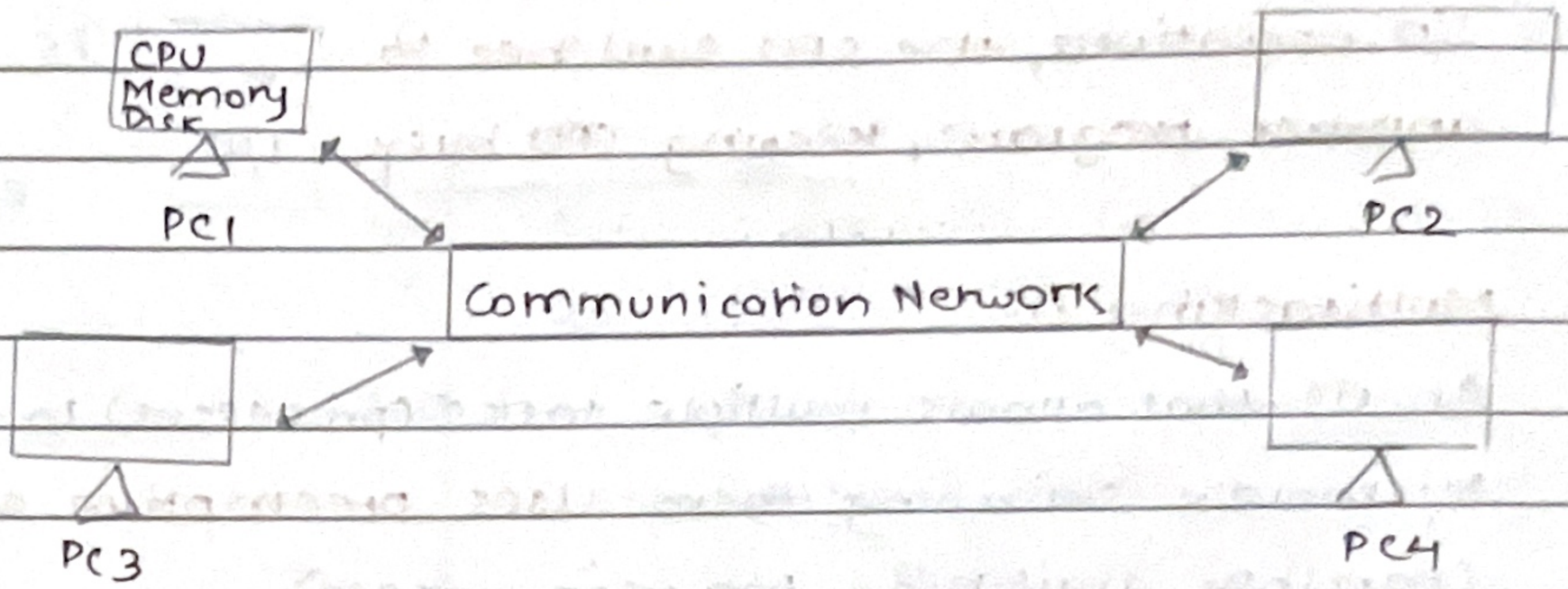


- **Embedded OS**

  A specialized OS designed to run on embedded system lightweight, efficient and tailered for specific hardware

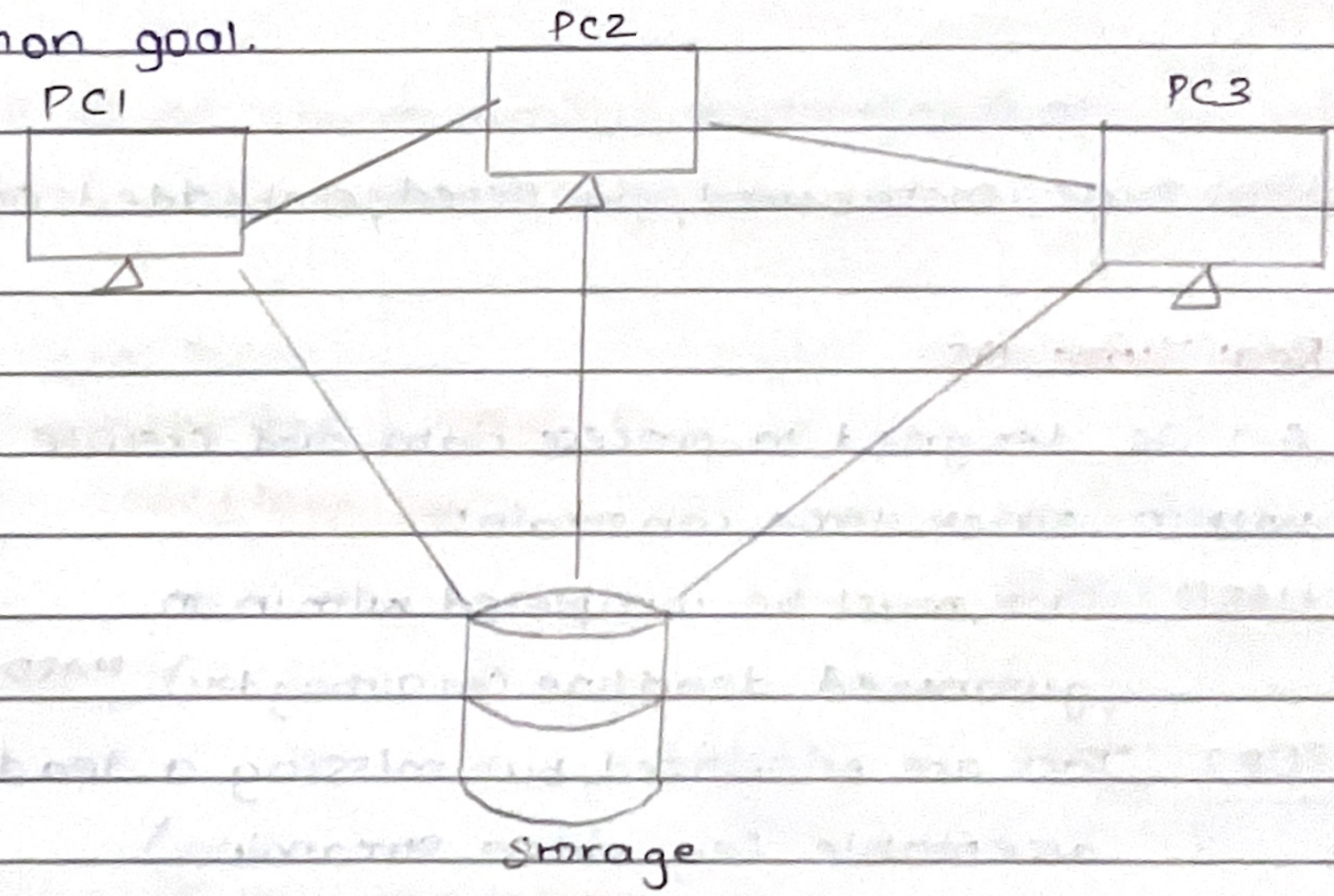
- **Distributed OS**

  Manages a group of independent computers connected over a network making them appear as single system. Fault tolerance - If one system fails other can take over.

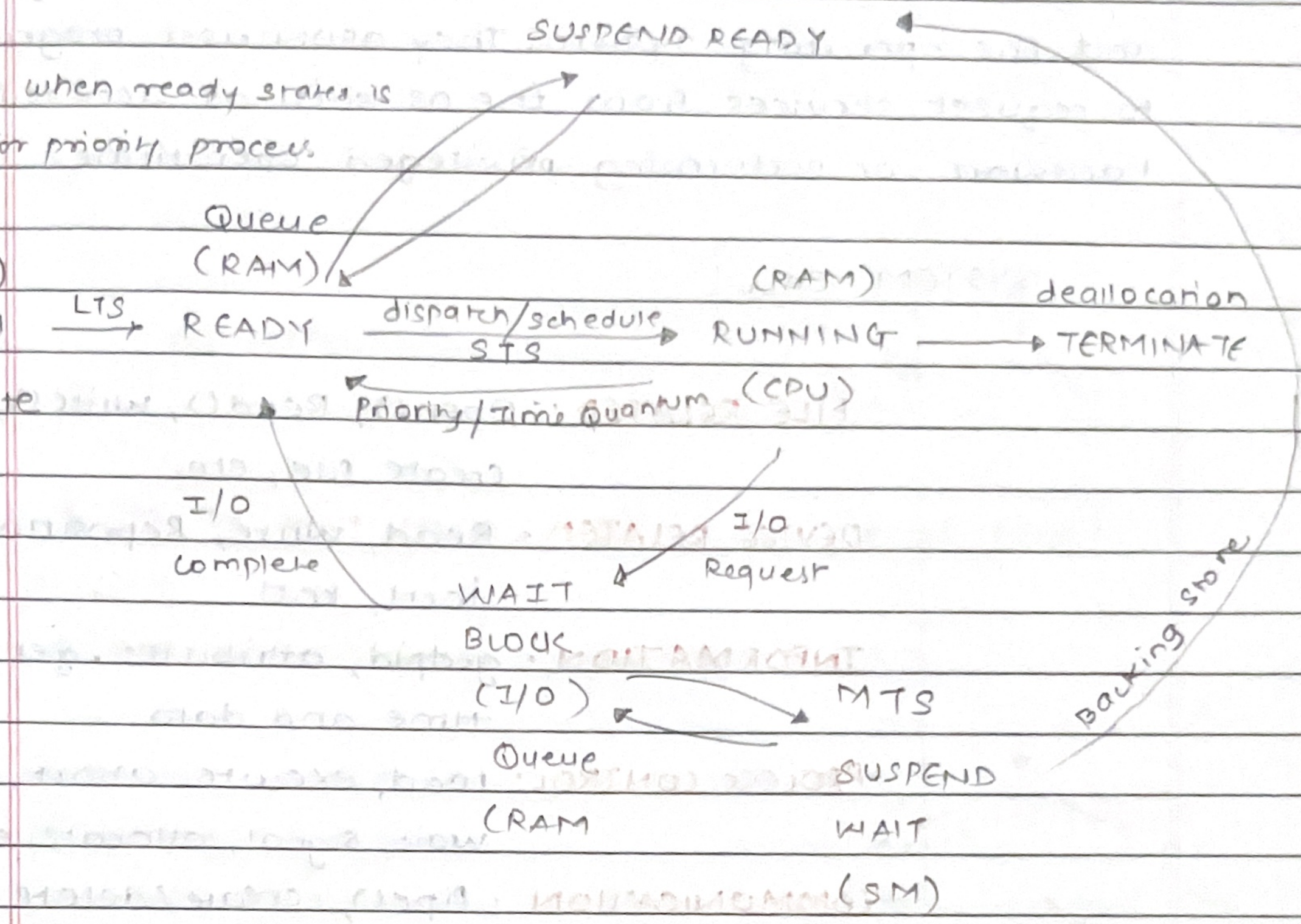


- **Clustured OS**

  Manages a group of interconnected computer (nodes) working together as single unit system to achieve a common goal.



L1:5 Process states in OS



- Long Term Scheduller (LTS): selects processes from the job pool and loads them into memory for execution. Decides which programs are brought into ready queue
- Short Term Scheduller (STS): selects a process from q Deque and allocates CPU to it.
- Medium Term Scheduller (MTS): Handles swapping of processes between memory and disk.
- If a process runs without interruption it is non-preemptive if the OS can forcibly switch it's preemptive.

L1:6 Linux commands question.

L1:7 System call in OS

System calls are interfaces between users programs and the operating system. They allow user program to request services from the OS such as accessing hardware or performing privileged operations.

SYSTEM CALL

- FILE RELATED: Open(), Read(), write(), close(), create file, etc.
- DEVICE RELATED: Read, write, Reposition, ioctl, fntl
- INFORMATION: getpid, attributes, ger system time and data.
- PROCESS CONTROL: Load, execute, abort, fork, wait, signal, allocate, etc.
- COMMUNICATION: Pipe(), create/delete connection

L1:8 fork system call

Fork(): Creates a new process (child process) that is an exact copy of the calling process (parent process

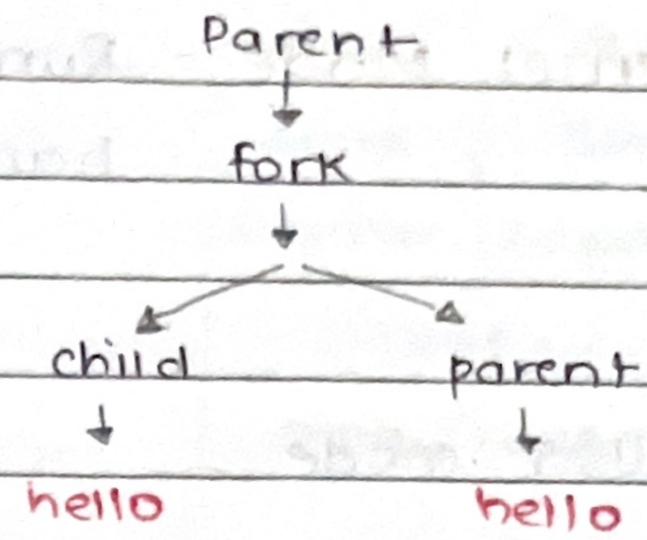
- Return values:

In the parent process: Returns the PID (Process ID) of the child process

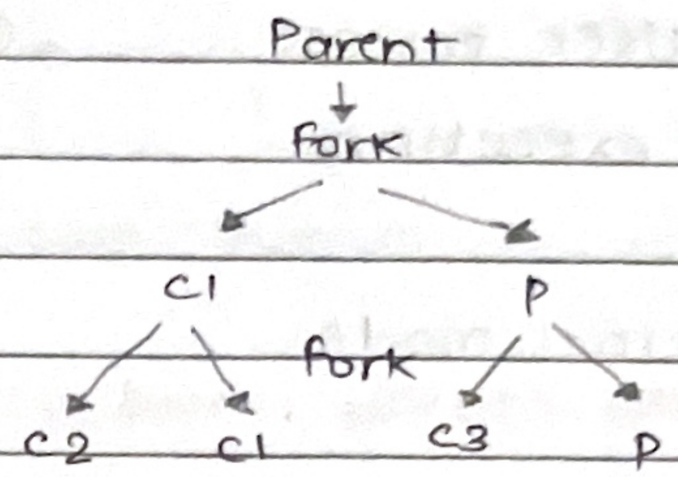
In the child process : Returns 0

On failure : Returns -1

```
main() {
    fork();
    printf("hello");
}
```




```
main() {
    fork();
    fork();
    printf("hello");
}
```

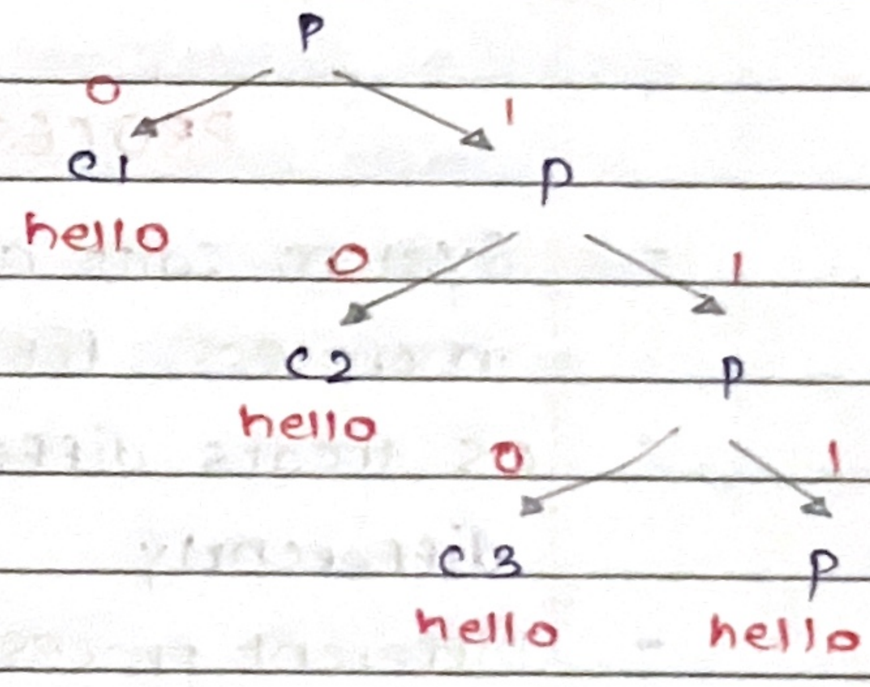


3 child
1 parent

No. of child process $2^n - 1$ (n - no. of Fork

L1:9 Questions on fork() system call

```
#include <stdio.h>
#include <unistd.h>
int main() {
    if (fork() && fork()) {
        fork();
    }
    printf("hello");
}
```



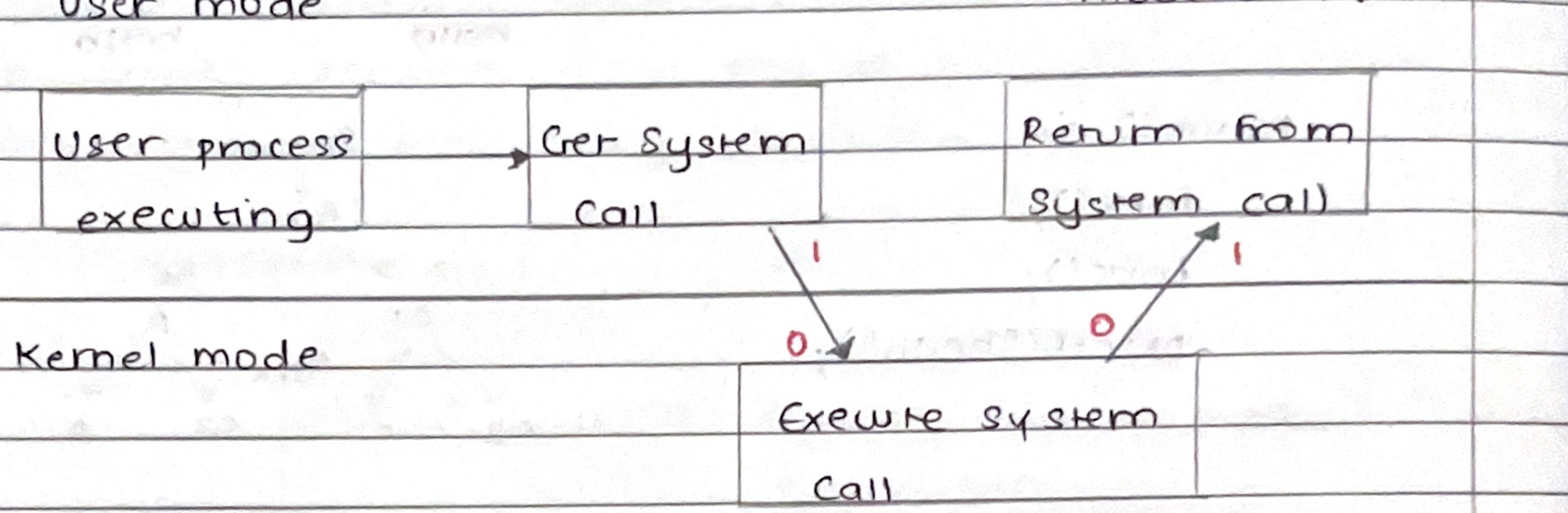
4 times hello printed.

L1:10 User Mode and Kernel Mode

The CPU operates in two modes to ensure security & Scalability in system

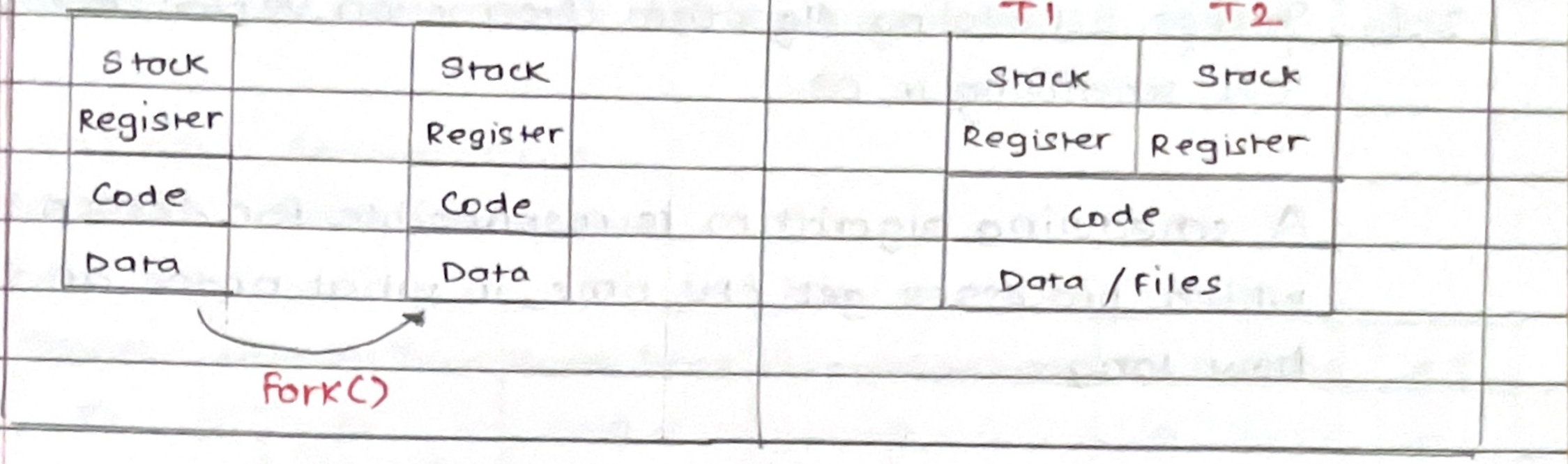
- **User Mode** - Runs application program with limited access to system resources.

- Kernel Mode - Runs the OS core function, manages hardware and executes priviileged instruction



L1:11 Process Vs Threads

	PROCESS	THREADS
-	System calls are involved in process for eg: fork(),	There is no system calls involved in threads.
-	OS treats different process differently,	All user level treated as single task for OS
-	Different process has different copies of data/files, code	Threads share copy of code and data/files
*-	context switching is slower	context switching is faster
*-	Blocking a process will not block another process	Blocking a thread will block entire process.
-	Independent	Interdependent



L1:12 User level Thread Vs Kernel Level Thread

	USER LEVEL THREAD	KERNEL LEVEL THREAD
-	User level threads are managed by user level library	Kernel level threads are managed by OS (system call).
-	User level threads are typically fast	Kernel level threads are typically slower than user.
-	Context switching is faster	Context switching is slower.
-	If one user level thread perform blocking operation the entire process is blocked.	If one user level thread r perform blocking operation it doesn't affect others.

L2:1 Process Scheduling Algorithm (Preemption VS Non Preemption)

CPU scheduling in OS

A scheduling algorithm is responsible for determining which processes get CPU time, in what order and for how long.

SCHEDULING ALGORITHM

PREEMPTIVE	NON-PREEMPTIVE
• SRTF (Shortest Remaining Time first)	• FCFS (First come first serve)
• LRTF (Longest Remaining Time first)	• SJF (Shortest Job first)
• Round Robin	• LJF (Longest Job first)
• Priority based	• HRRN (Highest Responsive Ratio Next)
	• Multilevel Queue
	• Priority based

L2:2 What is Actual, Burst, Completion, Turnaround, Waiting and Response Time in CPU.

- ARRIVAL - The time at which process enters the ready queue of state.
- BURST TIME - The time required by a process to get execute on CPU. (duration).
- COMPLETION - The time at which process completes its execution.
- TURNAROUND - { completion time - Actual Time }
- WAITING - { Turnaround time - Burst Time }
- RESPONSE TIME - { The time at which a process gets CPU first time - Arrival time }

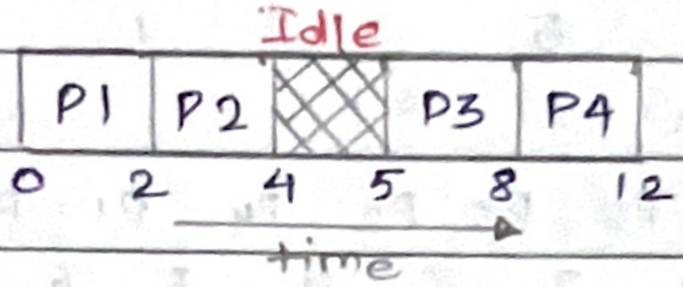
L2:3 First come first serve (FCFS) CPU scheduling algorithm

Criteria - Arrival Time

Mode - Non-preemptive

Process	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
P1	0	2	2	2	0	0
P2	1	2	4	3	1	1
P3	5	3	8	3	0	0
P4	6	4	12	6	2	2

Gantt Chart



- In non preemptive waiting time & response time are same

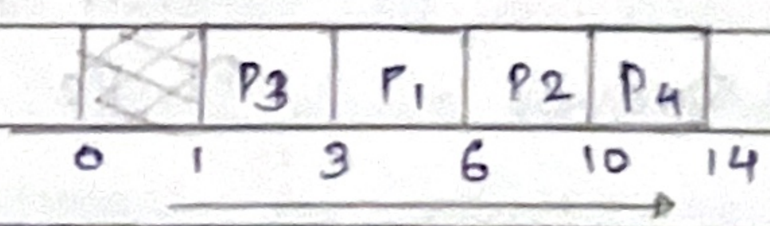
L2:4 Shortest Job First (SJF) scheduling algorithm

Criteria - Burst Time

Mode - Non preemptive

Process	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
P1	1	3	6	5	2	2
P2	2	4	10	8	4	4
P3	1	2	3	12	0	0
P4	4	4	14	10	6	6

Gantt Chart



- If both processes have same burst time the one with earlier arrival time is selected, if arrival time also becomes same the process with lower ID is chosen

Page No.
Date

*L2:5 Shortest Remaining Time First (SRJF with preemption) scheduling algorithm.

Criteria - Burst Time

Mode - preemptive

Process	ArrivalTime	CompletionTime	BurstTime	TAT	WAT	RT
P1	0	9	5	9	4	0
P2	1	4	3	3	0	0
P3	2	13	4	11	7	7
P4	4	5	1	1	0	0

Gantt chart

P1	P2	P2	P2	P4	P1	P1	P1	P1	P3

0 1 2 3 4 5 6 7 8 9 13

*L2:7 Round Robin (RR) CPU scheduling algorithm

Criteria - Time Quantum

Mode - preemptive

Process	Arrival Time	Burst Time	Completion	TAT	WAT	RT
P1	0	5	12	12	7	0
P2	1	4	11	10	6	1
P3	2	2	6	4	2	2
P4	4	1	9	5	4	4

Given TimeQuantum (TQ) = 2

Based on arrival time

Ready Queue

P1	P2	P3	P1	P4	P2	P1

0

Running Queue (Gantt chart) (context switching)

P1	P2	P3	P1	P4	P2	P1

0 2 4 6 8 9 11 12 time

- context switching means saving the running process and bringing new process.

L2:8 Preemptive priority scheduling algorithm

Criteria - Priority

Mode - preemptive

Priority	Process	Arrival	Burst	Completion	TAT	WAT
P0	P1	0	5	12	12	7
20	P2	1	4	8	7	3
30	P3	2	2	4	2	0
40	P4	4	1	5	1	0

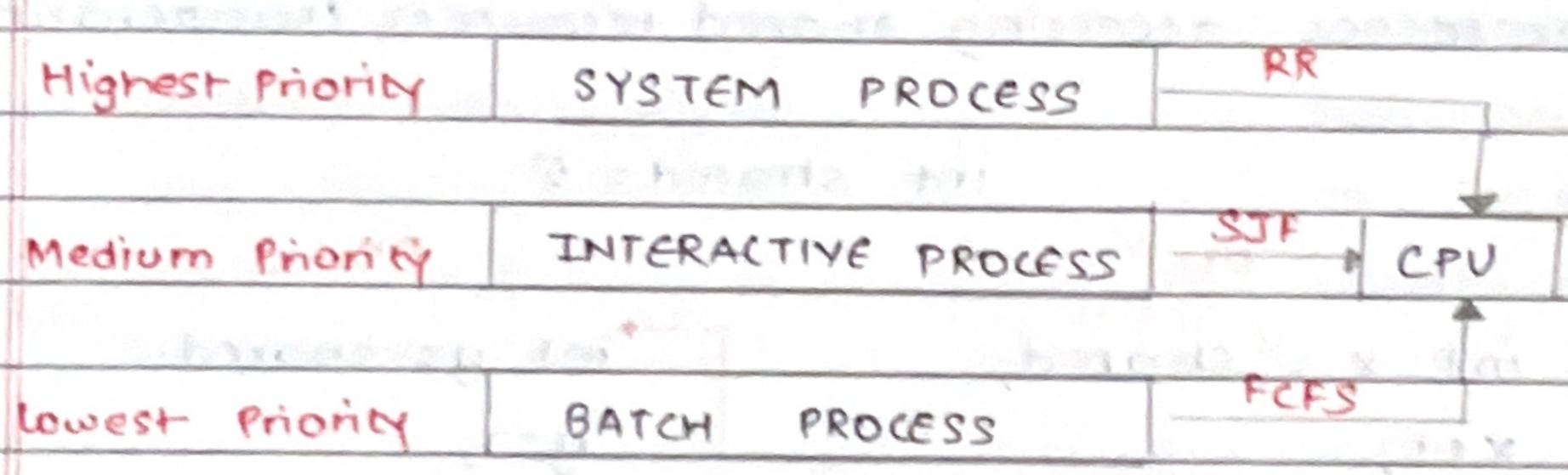
Grantt chart

P1	P2	P3	P3	P4	P2	P1

0 1 2 3 4 5 8 12

L2:10 Multilevel Queue scheduling

Processes are divided into multiple queue based on priority or type.



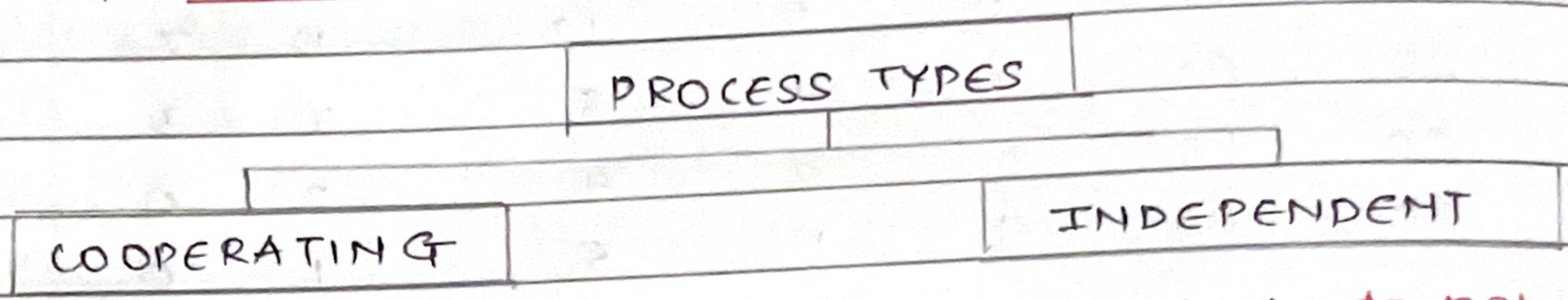
L2:11 Multilevel Feedback Queue scheduling

A more flexible version of multilevel queue scheduling where processes can move between quees based on behaviour (eg: aging to prevent starvation)

- Starvation occurs when a process is indefinetly delayed because higher priority process keep getting scheduled before it. preventing it from accessing resources like CPU time.

L3:1 Process Synchronization | Process Types | Race condition

Process synchronization is mechanism used in operating systems to coordinate multiple processes or threads to ensure correct execution while accessing shared resources. It prevents race conditions and ensure data consistency.



Processes that ~~do not~~ share data or resources with other processes.

Processes that do not share data or resources with other processes

Race Condition

A situation where the outcome of a process depends on the order of execution of multiple threads or processes accessing shared resources (cooperating)

```
                int shared = 5

        P1                              P2

int x = shared;                 int y = shared;
x++;                            y--;
sleep(1);                       sleep(1);
shared = x;                     shared = y
```

If P1 is executed first then shared = 4

If P2 is exewted first then shared = 6

13:2 Producer consumer Problem | Process synchronization problem.

Producer: Produces data and adds it to shared buffer

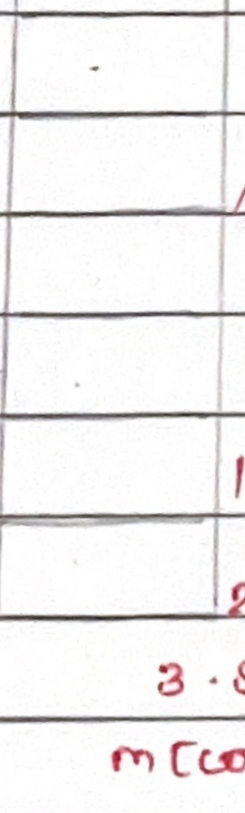
Consumer: Consumes data from buffer.

Issue: Without synchronization:

- Producer may OVERFLOW (add to a full buffer)
- Consumer may UNDERFLOW, (remove from empty buffer)

- BEST CASE - The buffer is neither full nor empty
  - Producer adds item to the buffer at the same rate as the consumer removes them.

n = 8 Buffer[0...n-1]




```
void consumer(void)
{
    int itemc;
    while (true) {
        while(count==n);
        itemc = Buffer[out];
        out = (out+1) mod n;
        count = count-1;
        process_item(itemc);
    }
}
```

//Buffer Empty ← while(count==n);

1. Load register with m[count]
2. Derement register
3. Store m[count] with register.

```
int count=0; //global
void producer(void) {
    int temp;
    while (true) {
        produce_item(item);
        while(count==n);
        Buffer[in]=item;
        in = (in+1) mod n;
        count=count+1;
    }
}
```

//Buffer full ← while(count==n);

1. Load Rp, m[count] ← count=count+1;
2. INCR Rp
3. Store m[count], Rp

count	in	out
0	0	0
1	1	1

→ x1 removed at 0 and moved to 1

↳ x1 added at 0 and moved to 1

Buffer: A fixed size memory space where data is stored temporarily.