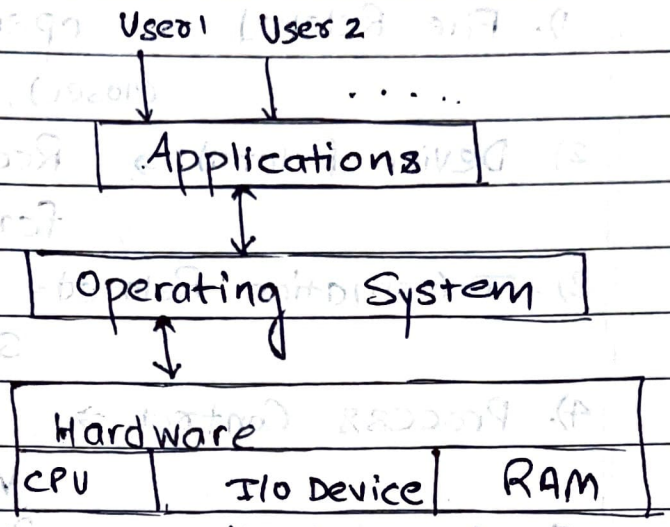
Introduction to Operating System and its functions.



- Convenience
- Throughput

Functions :-

1). Resource Management
2). Process Management (CPU Scheduling)
3). Storage " (HD)
4). Memory " (RAM)
5). Security

Multiprogrammed OS (Non preemptive)

↳ IDLENESS

↳ Execute first process, then only switch to second completely

Multitasking OS / Time Sharing OS :-

↳ Responsiveness

Execute task on the basis of slots.

Date
Page No

"System Call"

1). File Related ⇒ open(), Read(), Write(), Close(), createfile, etc.

2). Device Related ⇒ Read, Write, Reposition, ioctl, fcntl.

3). Information Related ⇒ get Pid, attributes, get System time & data

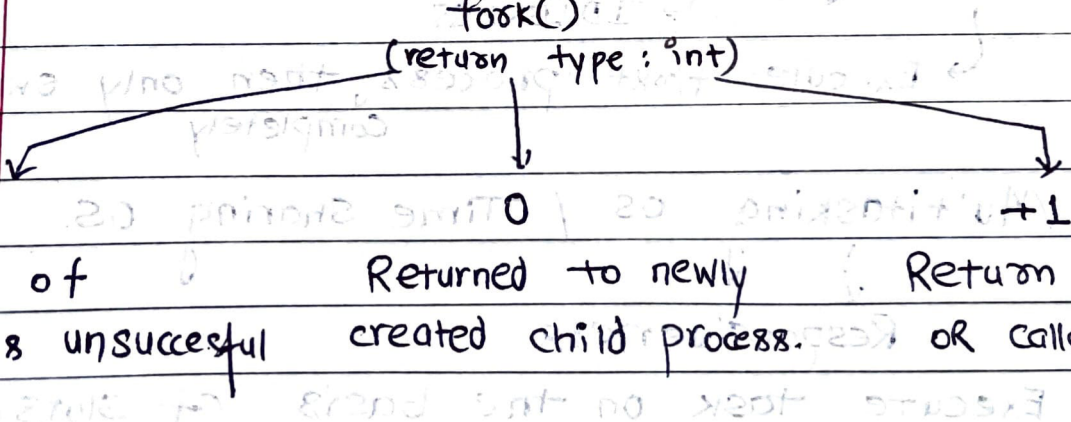
4). Process Control ⇒ Load, Execute, Abort, For Wait, Signal, Allocate, etc

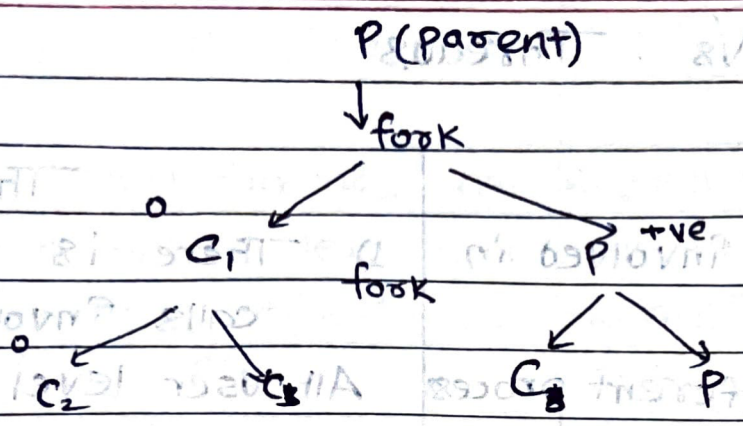
5). Communication ⇒ Pipe(), Create/delete Conn Shmget()

"Fork"

Fork System Call is used for creating new process, which is called child process, which runs concurrently with the process that m the fork() call.

A child process uses same PC (program counter same CPU registers,



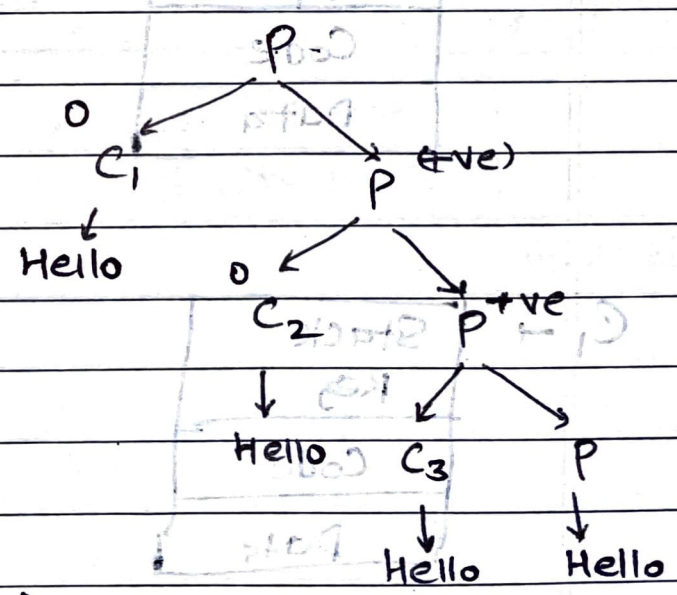


Question On Fork

Q.1.

```
int main()
{
    if (fork() && fork())
        fork();
    print("Hello\n");
    return 0;
}
```

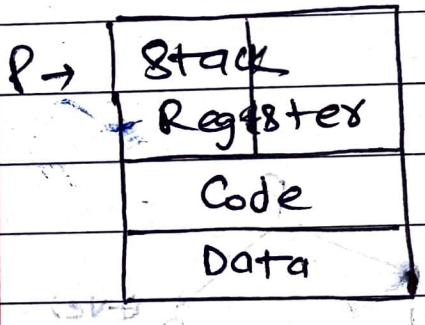
(total will be 0)

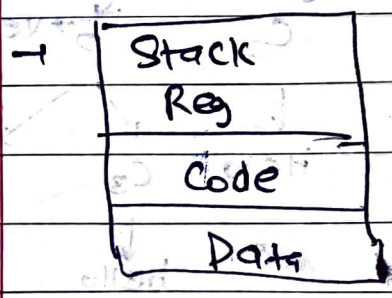


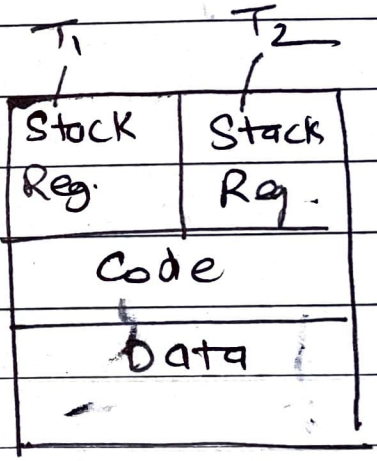
Soln → Hello (4 times)

Process Vs Threads

Process	User level Threads
System Calls involved in the process	1). There is no system calls involved
OS treat different process differently	All user level threads treated as single task for OS
Different process have diff. copies of data, file, codes.	Threads share same copy of data and code
Context switching is slower	Context switching is faster
Blocking a process will not block another	Blocking a thread will block entire process
~~Interdep~~ Independent	~~Interdependent~~

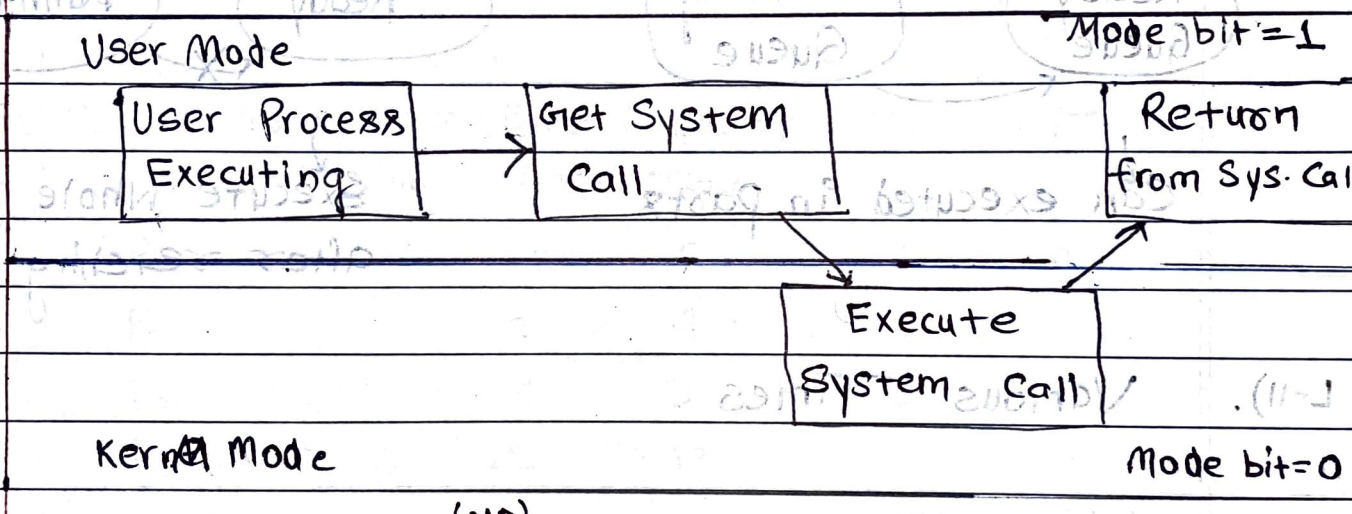






User level Thread.	Kernel level Thread
1). User level threads are managed by User level library	• Kernel level threads are managed by OS.
2). User level threads are typically fast.	• Kernel level threads are slower than User level.
3). Context Switching is fast.	• Context Switching is slow
4). If one user level thread perform blocking operation then entire process gets blocked.	• If one Kernel level thread blocked, no affect on others

User Mode Vs Kernel Mode



(HD)

Scheduling Algorithms

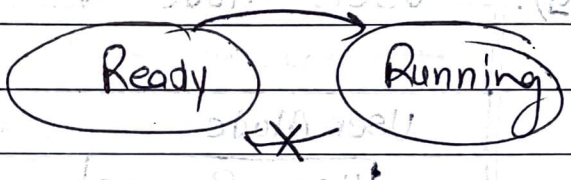
Pre-Emptive

- SRTF (Shortest Remaining time first)
- LRTF (Longest Remaining time first)
- Round Robin
- Priority based

Non Pre-Emptive

- FCFS (First Come First Serve)
- SJF (Shortest Job First)
- LJF (Long Job First)
- HRRN (Highest Response Ratio Next)
- Multilevel Queue
- Priority based

Can executed in parts



Execute whole process after reaching Running

Various Times

ARRIVAL Time : The time at which process enter the ready Queue.

BURST Time : Time Required by a process to get executed on CPU.

Completion Time : The Time at which process complete its execution.

Date:
Page No.

Turn Around Time :
{ Completion Time - Arrival Time }

Waiting Time :
{ Turn Around Time - Burst Time }

Response Time :
{ Time at which a process get CPU first time — Arrival time }

CPU Scheduling Algorithm -->

- First Come First Serve (FCFS)

Criteria : "Arrival Time"

Mode : "Non-Preemptive"

Process No	Arrival time	Burst Time	Completion Time	TAT	WT	RT
P_1	0	2	2	2	0	0
P_2	1	2	4	3	1	1
P_3	5	3	8	3	0	0
P_4	6	4	12	6	2	2

Gran++ Chart

Avg. Wating Time = $\frac{3}{4}$

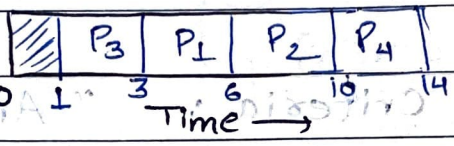
- Shortest Job First (SJF) Algorithm

Criteria : "Burst Time"

Mode : Non-Preemptive

Process No	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
P_1	1	3	6	5	2	2
P_2	2	4	10	8	4	4
P_3	1	2	3	2	0	0
P_4	4	4	14	10	6	6

Grantt Chart



P_1, P_3 — both arrived at 1.

Burst time of P_3 is smaller So P_3 will be selected.

Now P_1 P_2 will be in the ready Queue at 3.

Burst time (P_1) < Burst time (P_2)

Now P_2 P_4 → Burst time Same

So select on the basis of arrival time

In Non-preemptive Response time == WT

Average Turn Aroud Time

$$= \frac{5+8+2+10}{4}$$

$$= 6.25$$

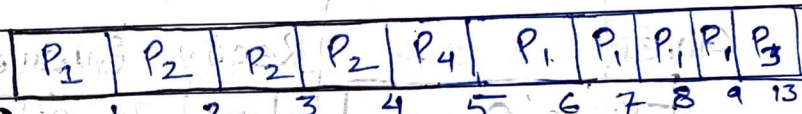
Avg WT = 3

- Shortest Remaining Time First (SRTF)

Criteria : "Burst Time"
Mode : "Preemptive"

Process No	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
P_1	0	5	9	9	4	0
P_2	1	3	4	3	0	0
P_3	2	4	13	11	7	7
P_4	4	1	5	1	0	0

Grantt Chart



At 0 → P_1 (4)
At 1 → P_1 P_2 (4,3)
At 2 → P_1 P_2 P_3 (4, 2, 4)
At 3 → P_1 P_2 P_3 (4, 1, 4)
At 4 → P_1 P_3 P_4 (4, 4, 1)
At 5 → P_1 P_3 (4, 4) → Now Arrival

Avg. Waiting Time $= \frac{24}{4}$

$= 6$

Avg. Response Time $= \frac{7}{4}$

• Round Robin (RR) CPU scheduling Algorithm

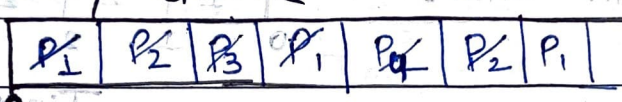
Criteria : "Time Quantum"

Mode : "Preemptive"

Process No	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
P_1	0	5	12	12	7	0
P_2	1	4	11	10	6	1
P_3	2	2	6	4	2	2
P_4	4	1	9	5	4	4

Time Quantum
TQ = 2

Ready Queue



Running Queue (Gantt Chart)

Context Switching
(Sending Running Process back to Ready Queue and loading new Program)

How

Context Switching at (2, 4, 6, 8, 9, 11)

No. of Context Switching = 6

• Preemptive Priority Scheduling Algorithm :

Criteria : "Priority"
Mode: "Preemptive"

Priority	Process No	Arrival Time	Burst Time	Completion Time	TAT	WT	RT
10	P_1	0	5	12	12	7	
20	P_2	1	4	8	7	3	
30	P_3	2	2	4	2	0	
40	P_4	4	1	5	1	0	

Higher the Value (Priority) - higher the priority

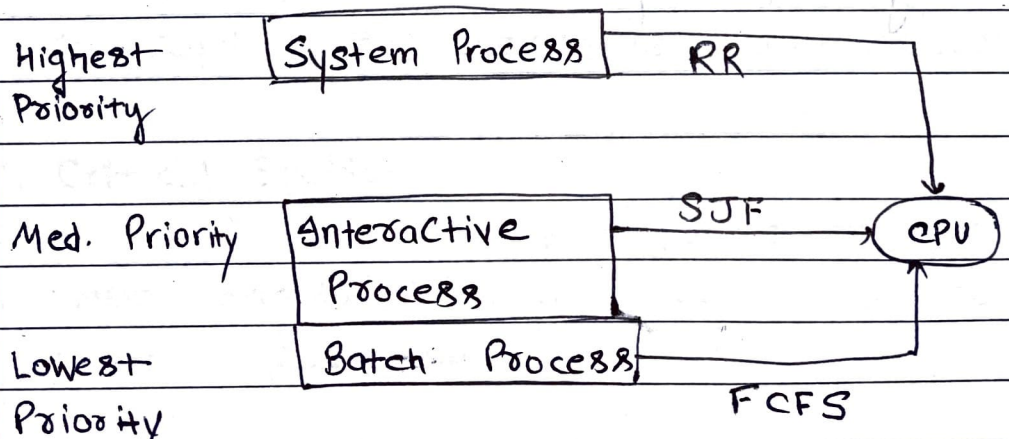
Grantt Chart

P_1	P_2	P_3	P_3	P_4	P_2	P_2	P_2	P_1	P_1

0 1 2 3 4 5 6 7 8 12

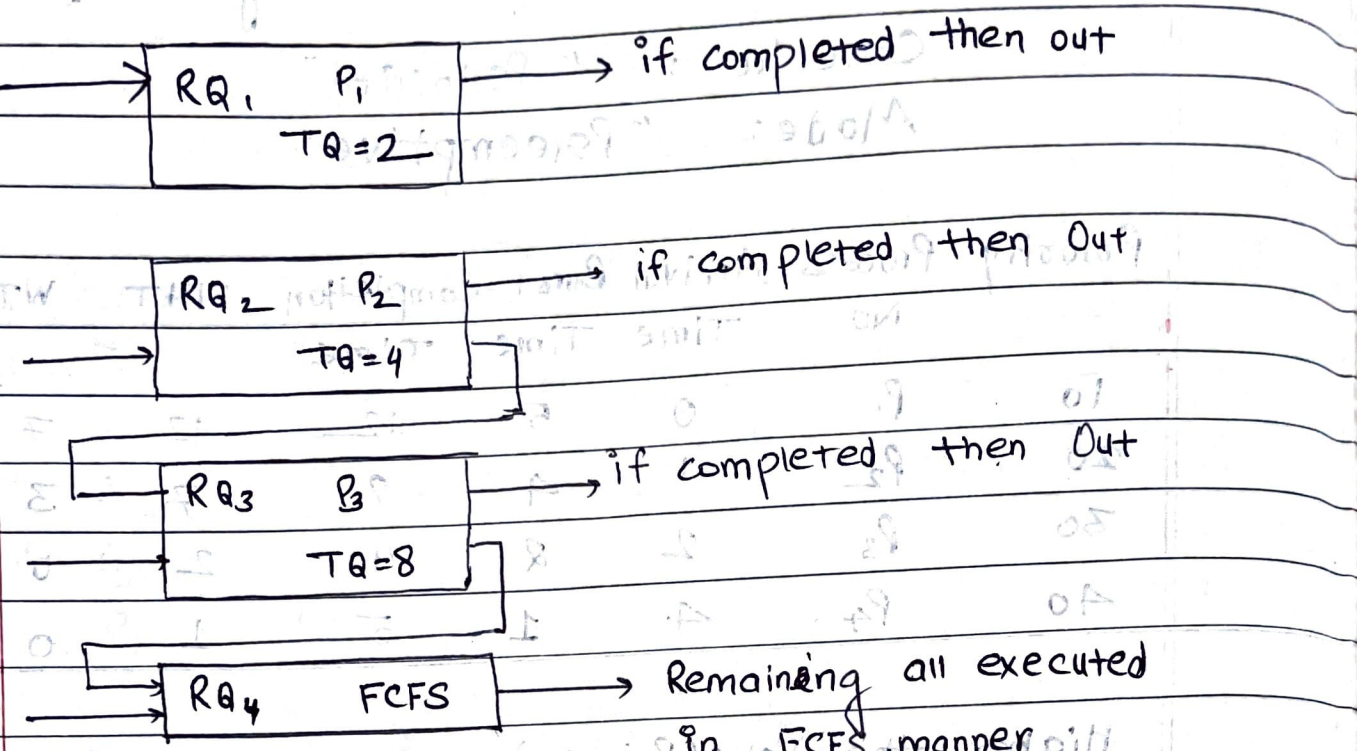
Time	P_1	P_2	P_3	P_4
	5 4	4 3	2 1	1

Multi Level Queue Scheduling



Due to priority — Starvation

Multilevel Feedback Queue Scheduling



Assume P_1 = 19 (takes 19 unit to execute)

RQ_1	~~19~~	17
RQ_2	~~17~~	13
RQ_3	~~13~~	5
RQ_4	~~5~~	0

So the process will get executed for Time-Quantum TQ and if not completed then passed on to higher priority Queue.