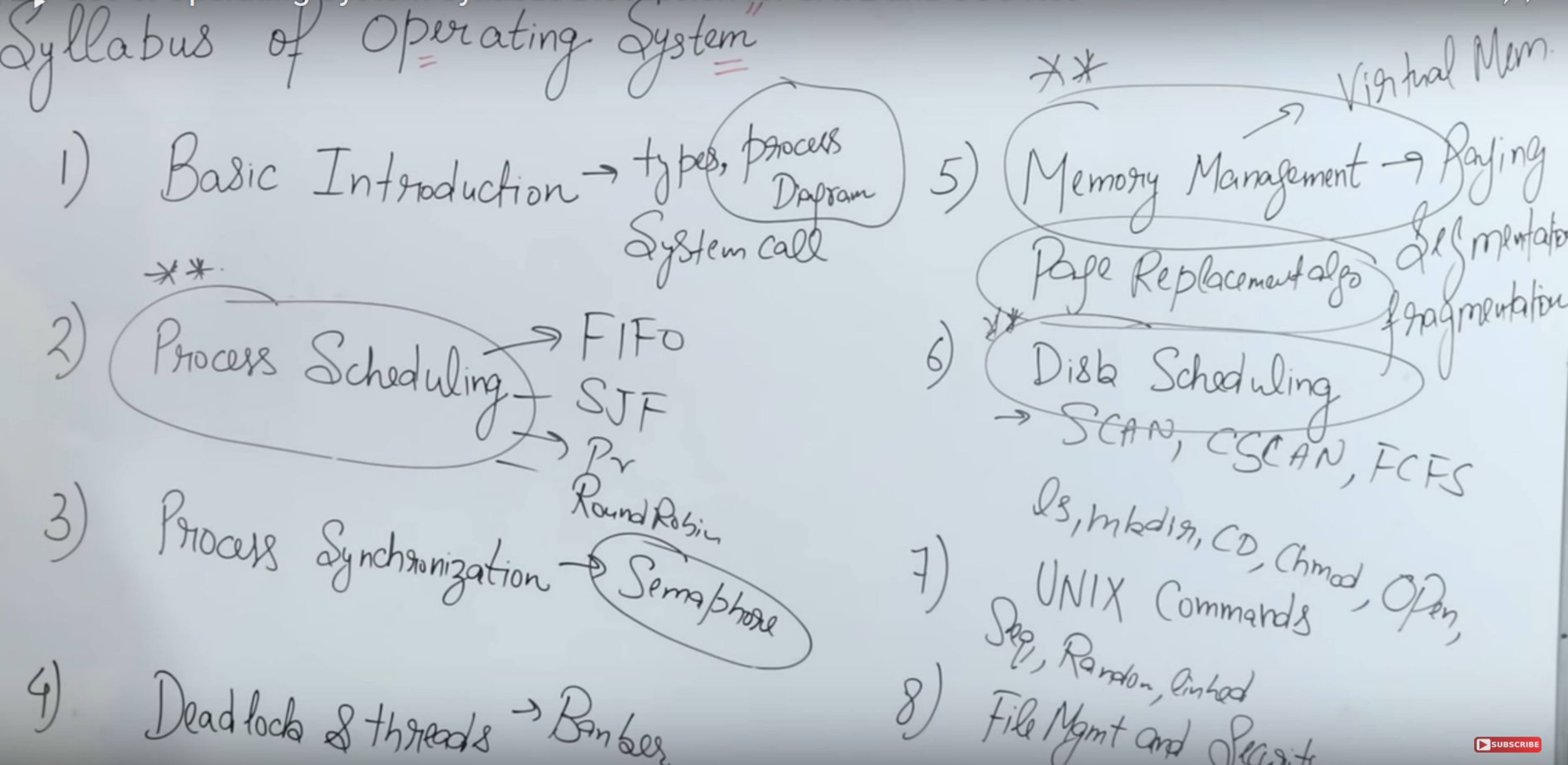


Syllabus of Operating System

1) Basic Introduction → types, Process Diagram
   System call

2) ** Process Scheduling → FIFO
   SJF
   Pr
   Round Robin

3) Process Synchronization → Semaphore

4) Deadlock & threads → Banker

5) ** Memory Management → Paging
   Virtual Mem.
   Segmentation
   fragmentation
   Page Replacement algo

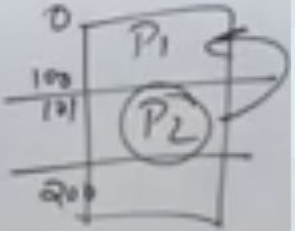
6) ** Disk Scheduling
   → SCAN, CSCAN, FCFS

7) ls, mkdir, CD, Chmod, Open,
   UNIX Commands

8) Seq, Random, linked
   File Mgmt and Security

'Operating System and its Functions'

→ Windows (82%)



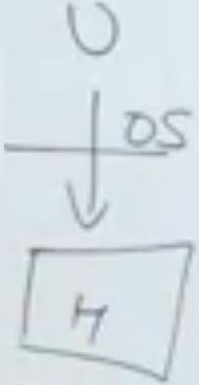
→ Primary goal

→ Convenience

(95%)

→ throughput (Linux)

5) Security

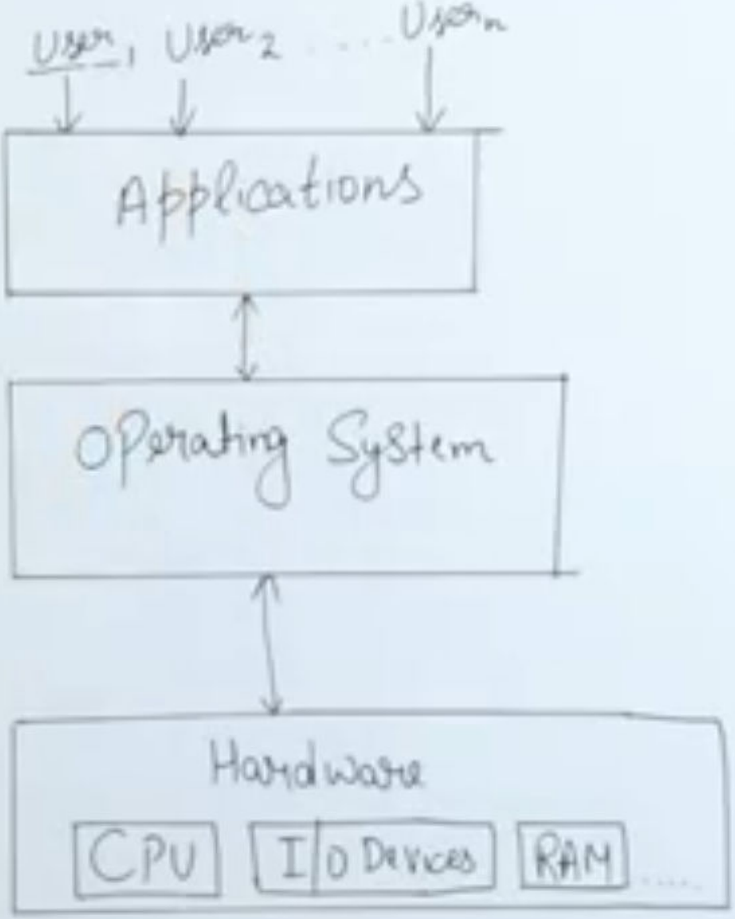


1) Resource Mgmt.

2) Process Mgmt. (CPU Scheduling)

3) Storage mgmt (HD) → file System

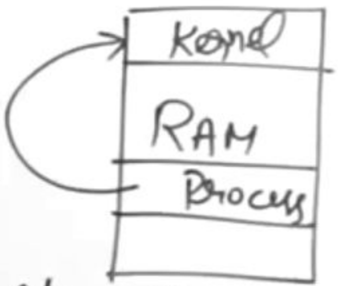
4) Memory Mgmt (RAM)



"System Call"

- → File Related ⇒ Open(), Read(), Write(), Close(), Create file etc.
- → Device Related ⇒ Read, Write, Reposition, ioctl, fcntl
- → Information ⇒ get Pid, attributes, get System time and data
- → Process Control ⇒ Load, Execute, abort, ** Fork, Wait, Signal, Allocate etc.
- → Communication ⇒ Pipe(), Create/delete Connections, Shmget()

Printf.



Program

Process

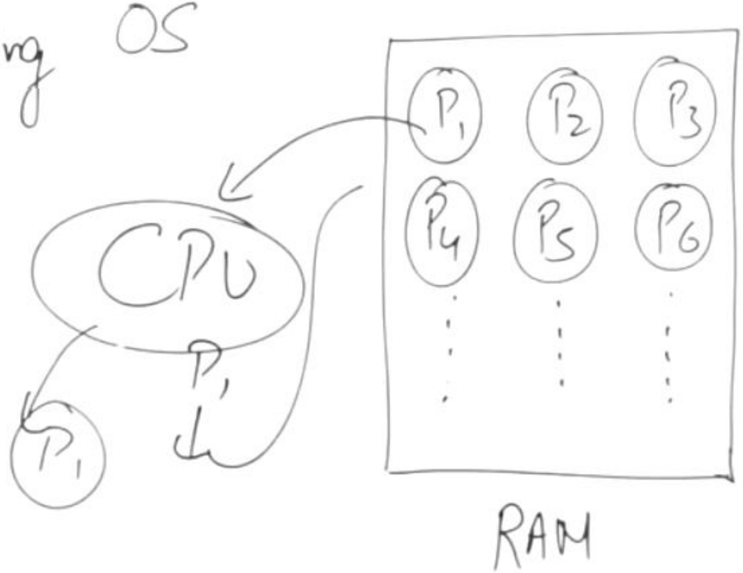
Multiprogrammed OS — Now Preemptive

↳ IDLENESS

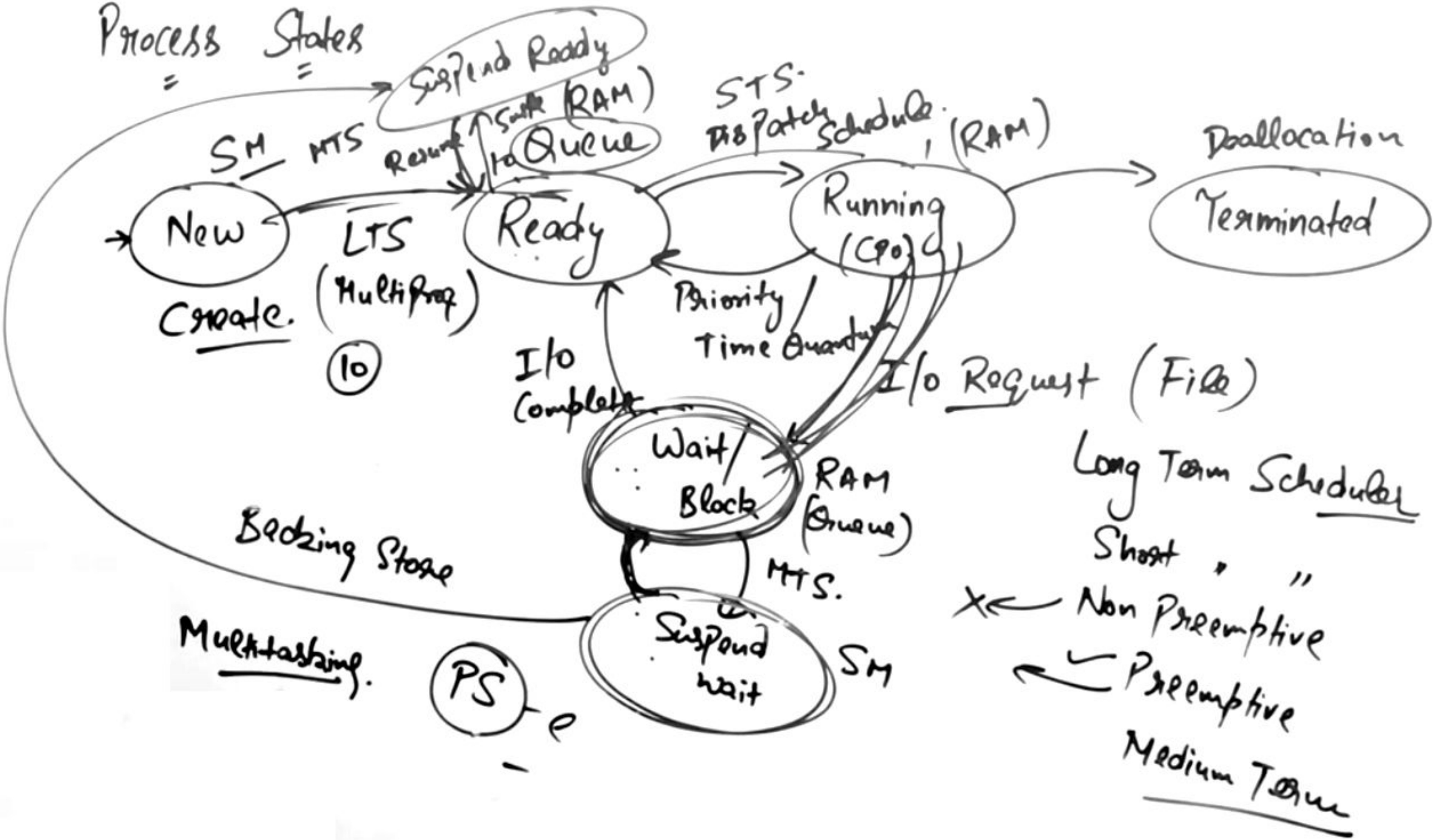
Multitasking / Time Sharing OS

↓

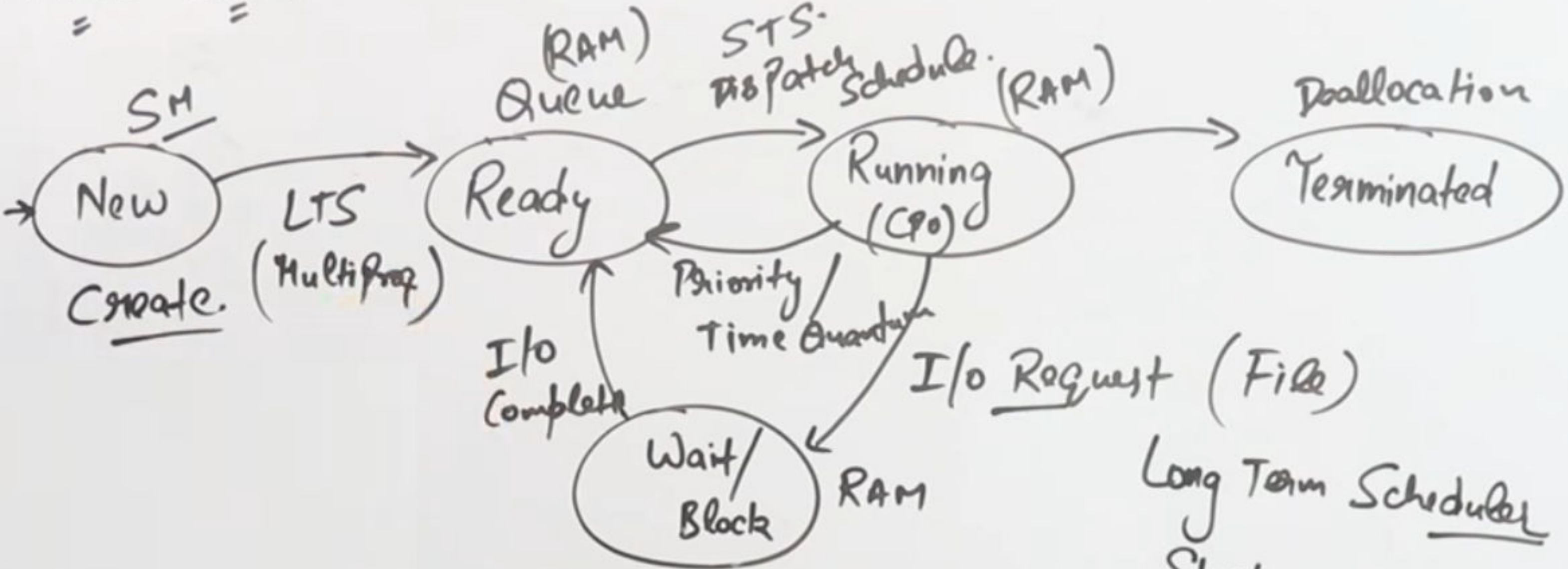
Responsiveness



Process States



Process States



Long Term Scheduler

Short " "

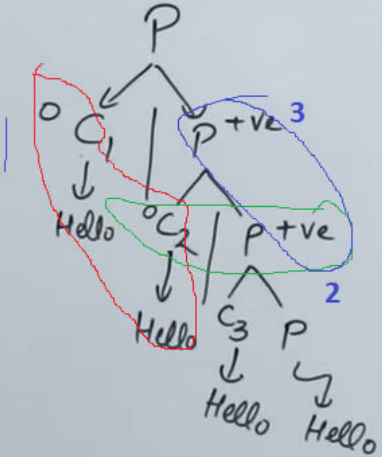
x ← Non Preemptive

← ✓ Preemptive

. c

```
# include < stdio.h >
# include < unistd.h >
int main()
{            0
→ if (fork() && fork())
      +ve   ||     0
   fork();  | |   +ve
                 true
  printf("Hello");
  return 0;
}
```

4

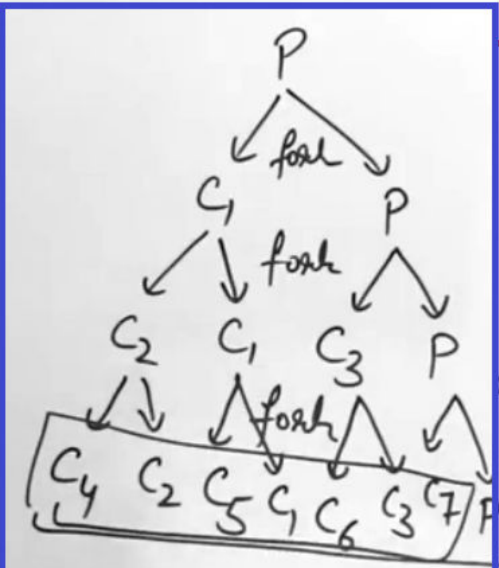


running in parrellel due to fork() call - child & parent

Fork() — 0 Child ✓

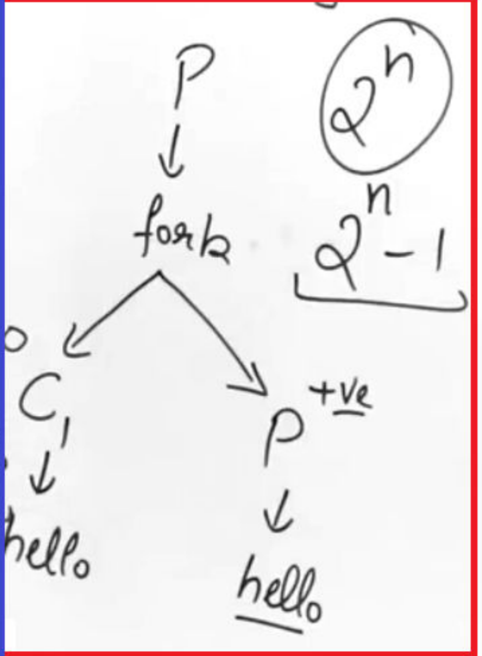
+1 +ve → Parent

−1 → Child x] x



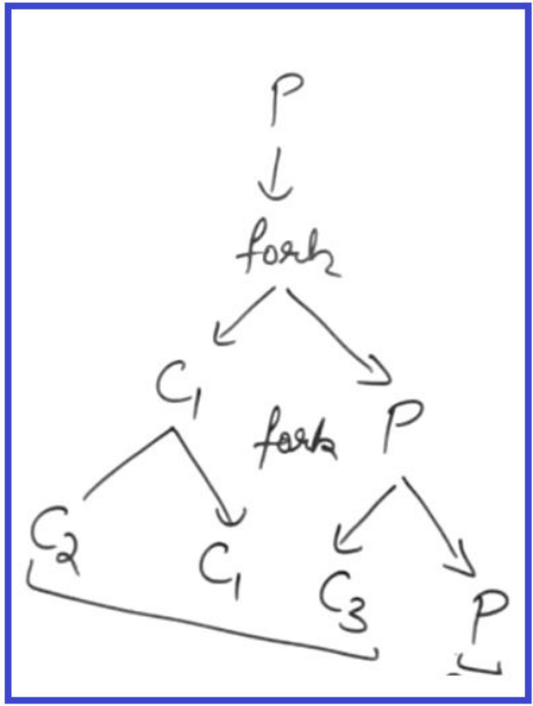
7 - Child

1 - Parent



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

```
fork
fork
fork
Pf("hello")
```



Q.
```
#include <stdio.h>
#include <unistd.h>
int main()
{
  int a;
  for (a=1; a<5; a++)
          fork();
  printf("1");
}
```

How many times it will print “1” in output?

(A) 15 (B) 16

(C) 31 (D) 32

Q.
```
#include <stdio.h>
#include <unistd.h>
int main()
 {
   if (fork() && fork())
        fork();
   printf("Hello");
   return 0;
 }
```

How many times it will print “Hello” in output?

(A) 2 (B) 3

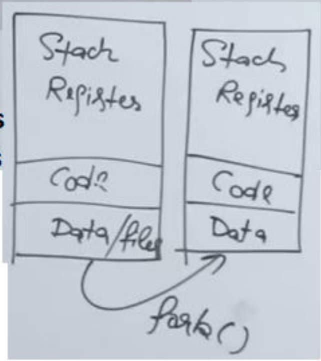
(C) 4 (D) 5

Process

1) System Calls involved in Process
2) OS treats different processes differently
3) Different process have different Copies of Data, files, Code
4) Content switching is slower
5) Blocking a process will not block another
6) Independent

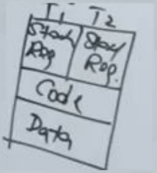
eacb process will have PID in process call. But in Threads there is 1 PID, however is this process is paused then the whole thread will be paused

content swiitching is slower as it has to save in process control block i.e. in RAM - Process



Threads (User level)

1) There is no system Call involved
2) All User level threads treated as single task for OS
3) Threads share same copy of Code and data
4) Content switching is faster
5) Blocking a thread will block entire process
6) Interdependent



blocking - in i/o request, then whole process will be blocked which includes all the Threads