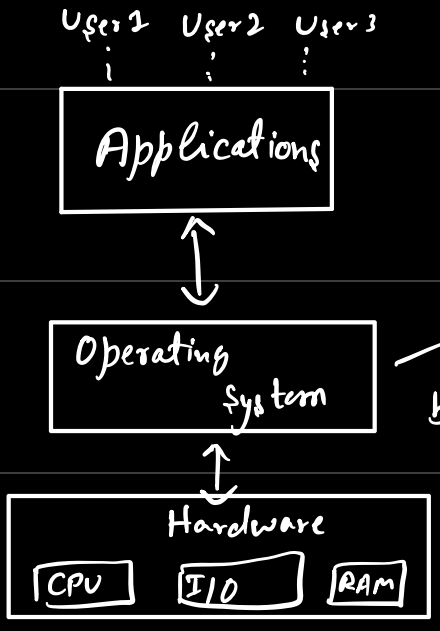
• Introduction to OS & its functionalities



→ Interface b/w user & hardware

→ Primary Goals
- Convenience
- Throughput

- Functionalities

1) Resource management

2) Process management
↳ CPU scheduling Algorithms

3) Storage management
↳ HDD (file system)

4) Memory Mgmt. (RAM)
↓
Limited

5) Security & Privacy

- Types of OS → ① Multiprogrammed OS
↳ Non-preemptive
⇓
Ek process poora execute karega CPU unless I/O na

② Multitasking → • Preemtive / Time sharing
↳ Ek process agar given time me execute ho gya then its fine otherwise it will get rescheduled
↙
Responsiveness (each process will be given response)

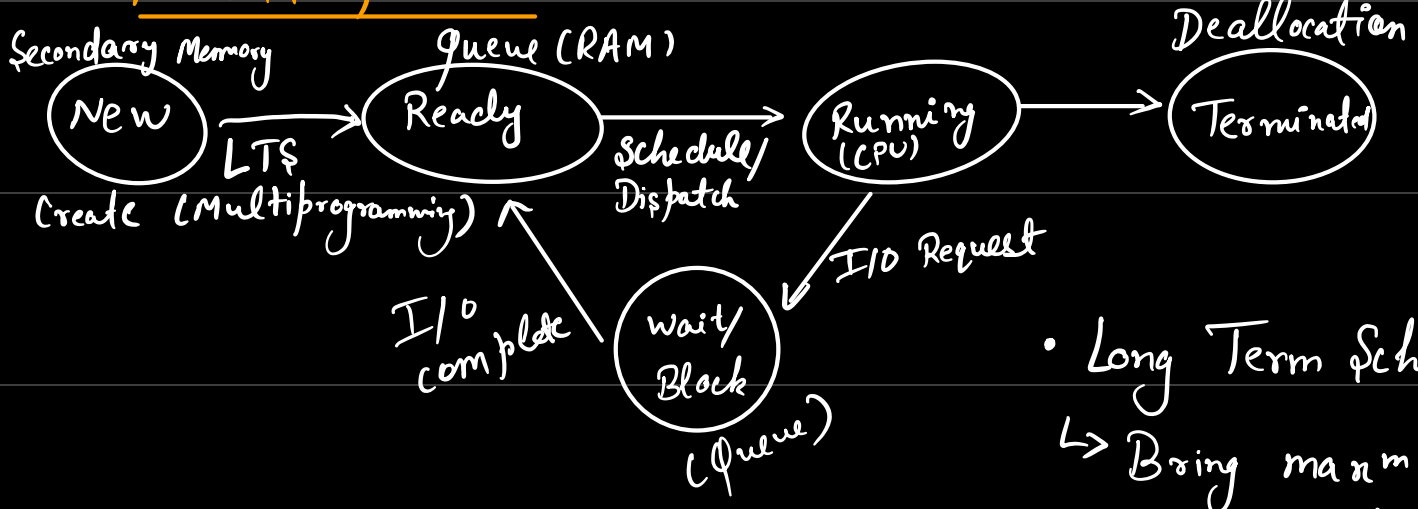
③ Real time OS → Hard → No delay should be there
→ Soft → Not really strict

④ Distributed

⑤ Clustered

⑥ Embedded

- Process States →

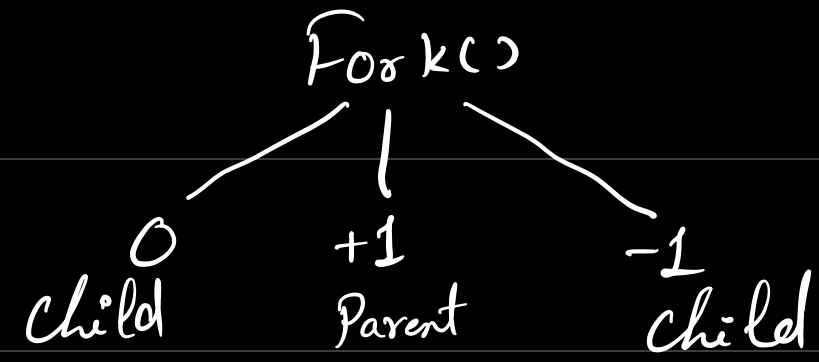


- Long Term Scheduler
  ↳ Bring many programs in Queue

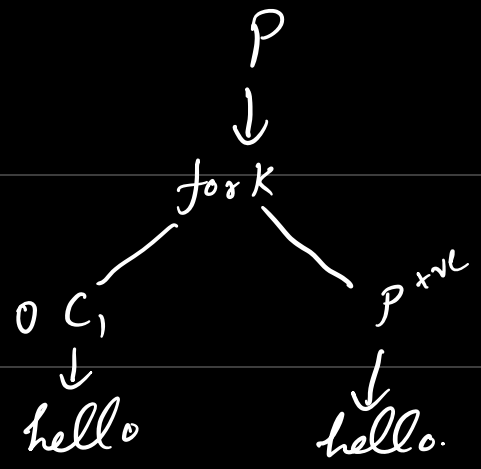
- System calls → Invokes kernel to do certain tasks.
  - File related ⇒ Open(), Read(), Write(), close(), Create file etc.
  - Device related ⇒ Read, write, Reposition, ioctl fcntl.
  - Information → get Pid (→ process_id), attributes, get system time & data

- Process Control → Execute, abort, Fork, wait allocate etc.
- Communication → pipe(), Create/Delete,

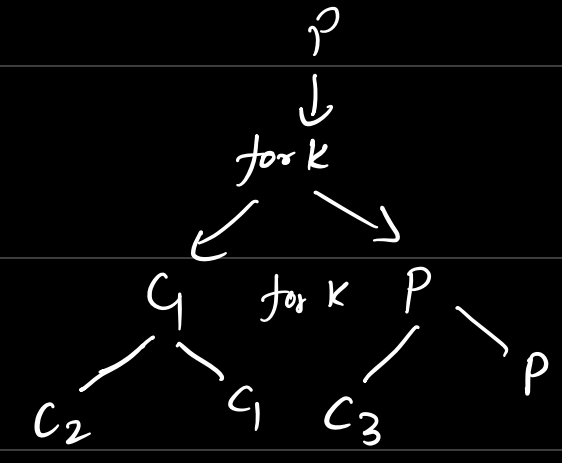
- Fork() → To create a child process.
  It will create a child process having the copy of parent process with a different pid.



Ex.
```
main() {
    fork();
    pf("hello");
}
```

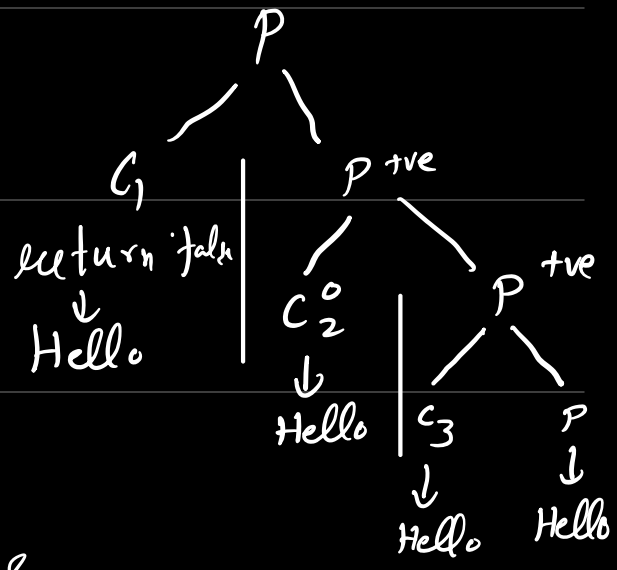


Ex.
```
main() {
    fork();
    fork();
    pf("hello");
}
```



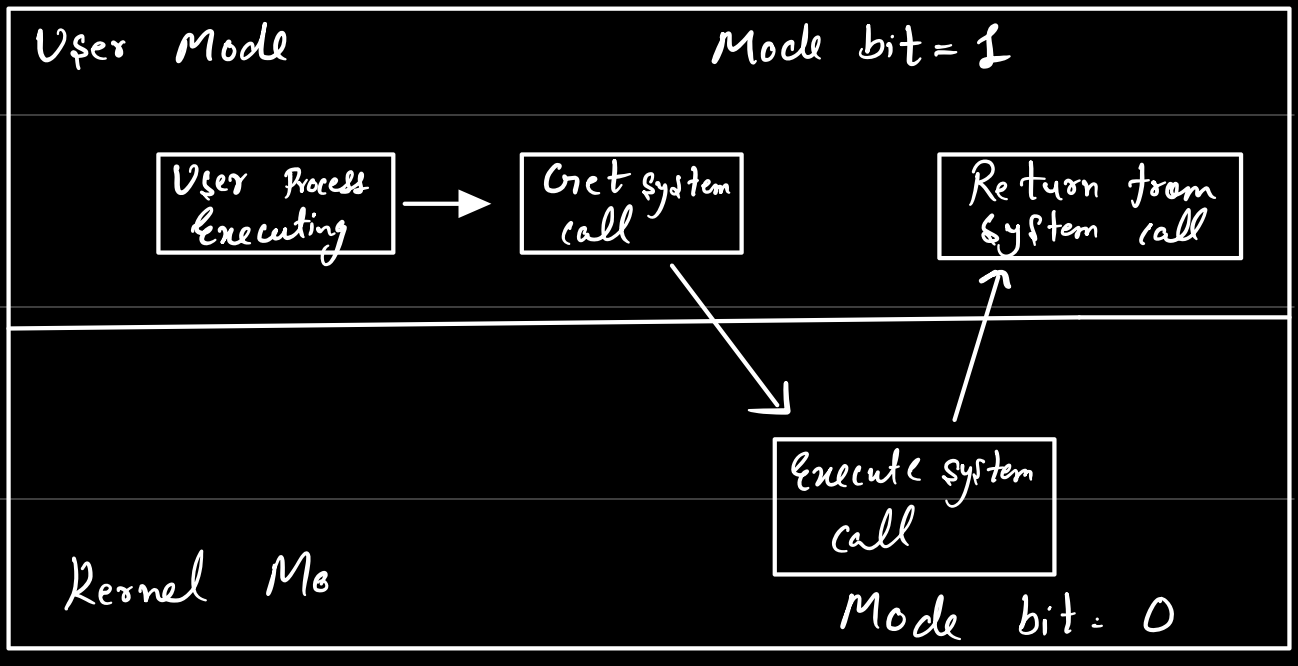
Total child = $2^n - 1$

```
main() {
    if ( fork() && fork() )
        fork();
    printf("Hello");
    return 0;
}
```



Q Use || instead of &&.

- "User Mode" Vs "Kernel Mode"



• Process

1) System calls involved in process.

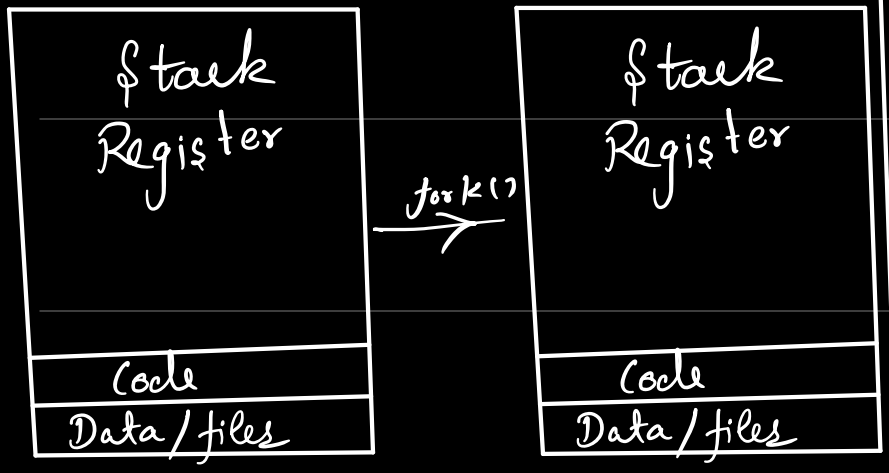
2) OS treats diff. processes differently

3) Diff. process have diff copies of Data, files, Code

4) Content switching is slower

5) Blocking a process will not block another

6) Independent



• Threads

1) There is no system call involved (User Lvl thread)

2) All user level threads treated as single task for OS.

3) Threads share same copy of code & data

4) Context switching is faster

5) Blocking a thread will block entire process

6) Interdependent.