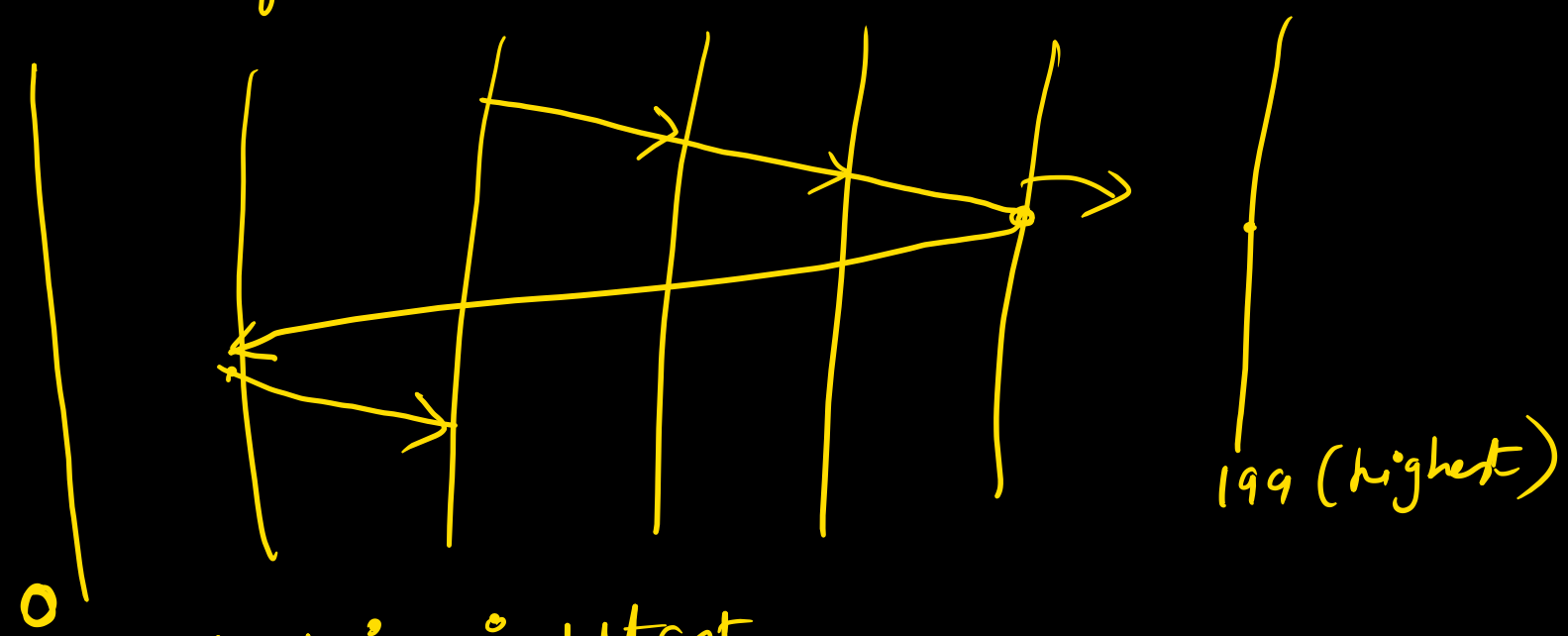
C look: 98, 183, 37, 122, 14, 124, 65, 67 (head is at 53, moving towards higher numbers)

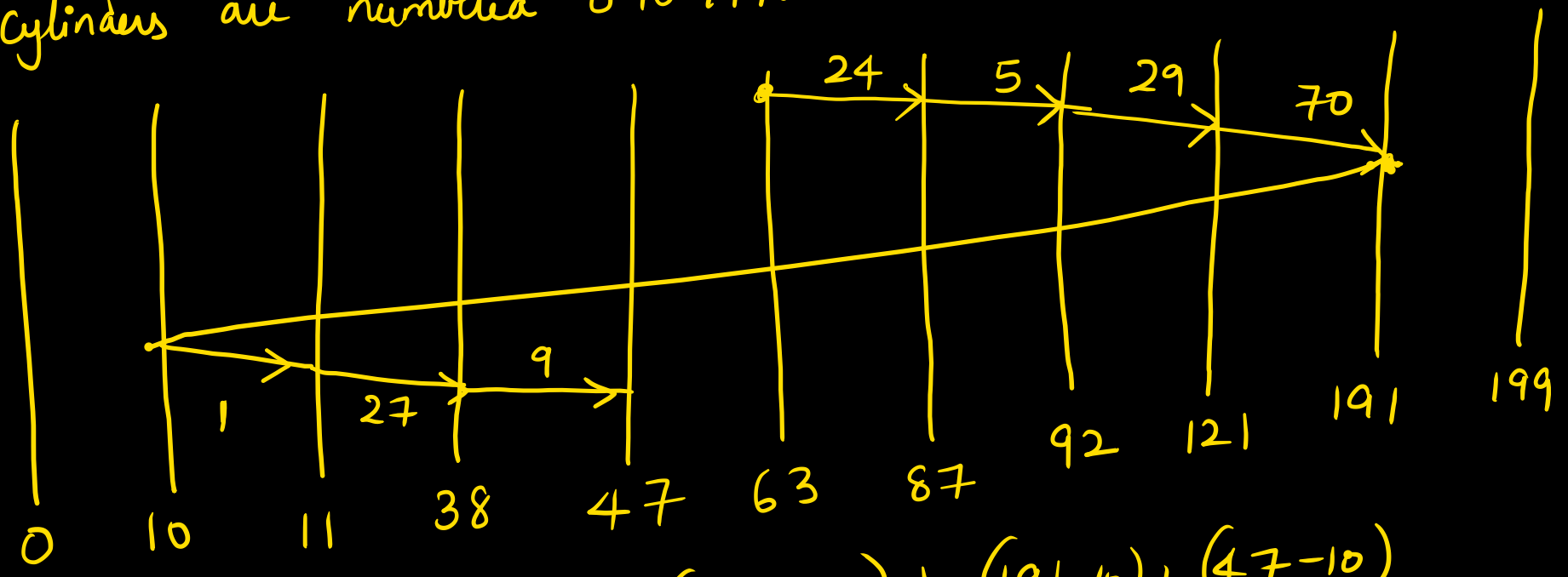
12 + 2 + 31 + 24 + 2 + 59 + 169 + 23 = 322 cylinders

In C look, the arm (R/w head) goes only as far as final request in each direction.
then reverses direction immediately without going all the way to the end of the disk.
when head reaches the other end, it immediately return to the other end extreme request without servicing any requests on the trip.



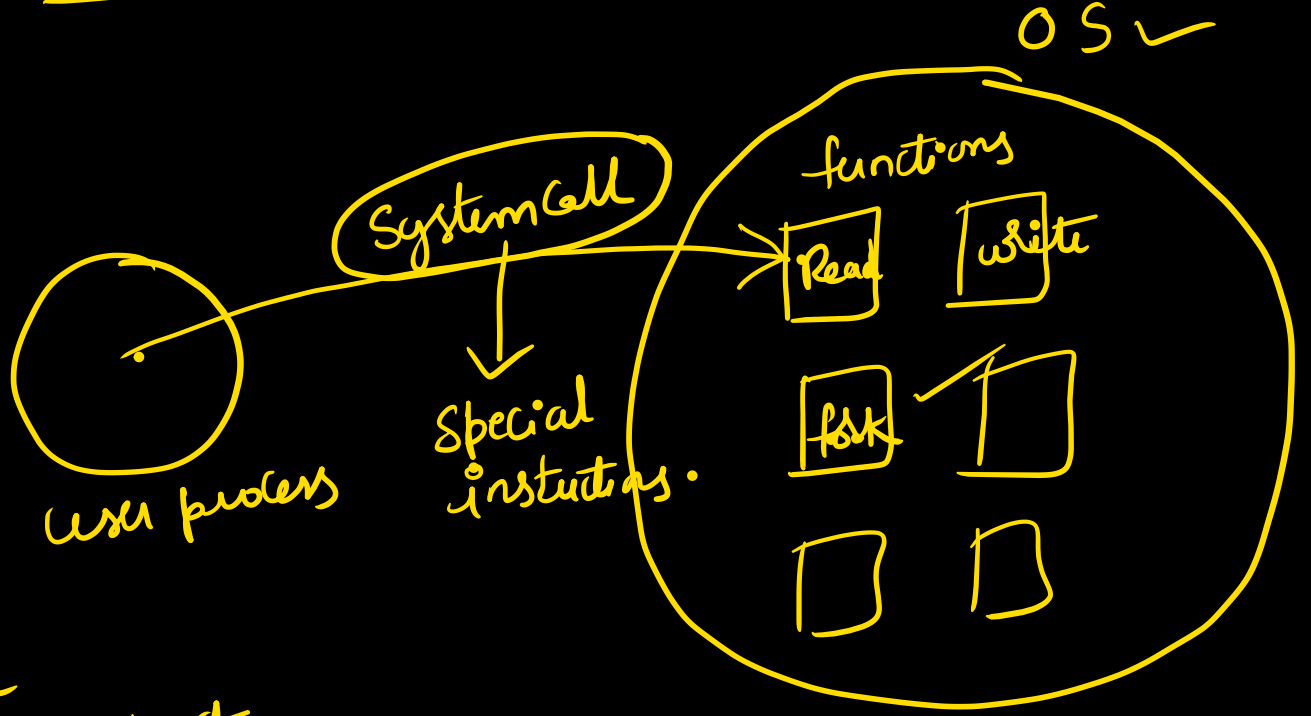
Direction R/w head is important

Gate16) Consider a disk queue with requests for I/O to blocks on cylinders 47, 38, 121, 191, 87, 11, 92, 10. The C look scheduling is used. The head is initially at cylinder 63, moving to larger on its service pass. Cylinders are numbered 0 to 199. The head movement (# cylinders)



$= 346.$ $(191-63) + (191-10) + (47-10)$

Threads and system calls: → In syllabus → recently added



what are different kinds of system calls?

1) Process control system calls: → These are used to control processes

- → End, abort
- → load, execute
- → Create a process, terminate a process
- → Get process attributes, set process attributes (Pid, owner...)
- → wait for time (sleep)
- → wait event, signal event (semaphores)
- → allocate and free memory (malloc & free)

"mug up"

2) File manipulation system calls:

Create File, Delete File

Open, close

read, write, reposition

get file attributes, set file attributes

} mug up.

3) Device manipulation system calls:

Request device, Release device

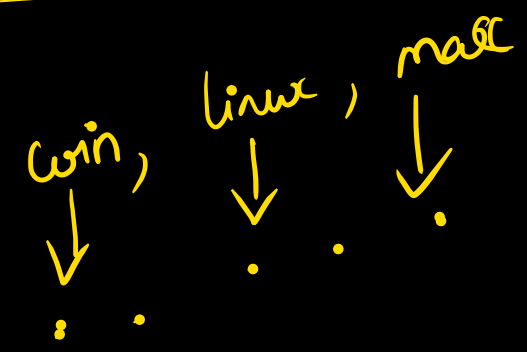
Read, write, reposition

get device attributes, set device attributes

logically attach & detach devices (device drivers)

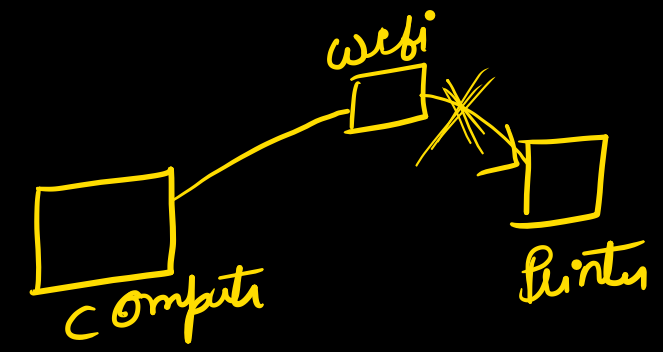
4) Information maintenance system Calls:

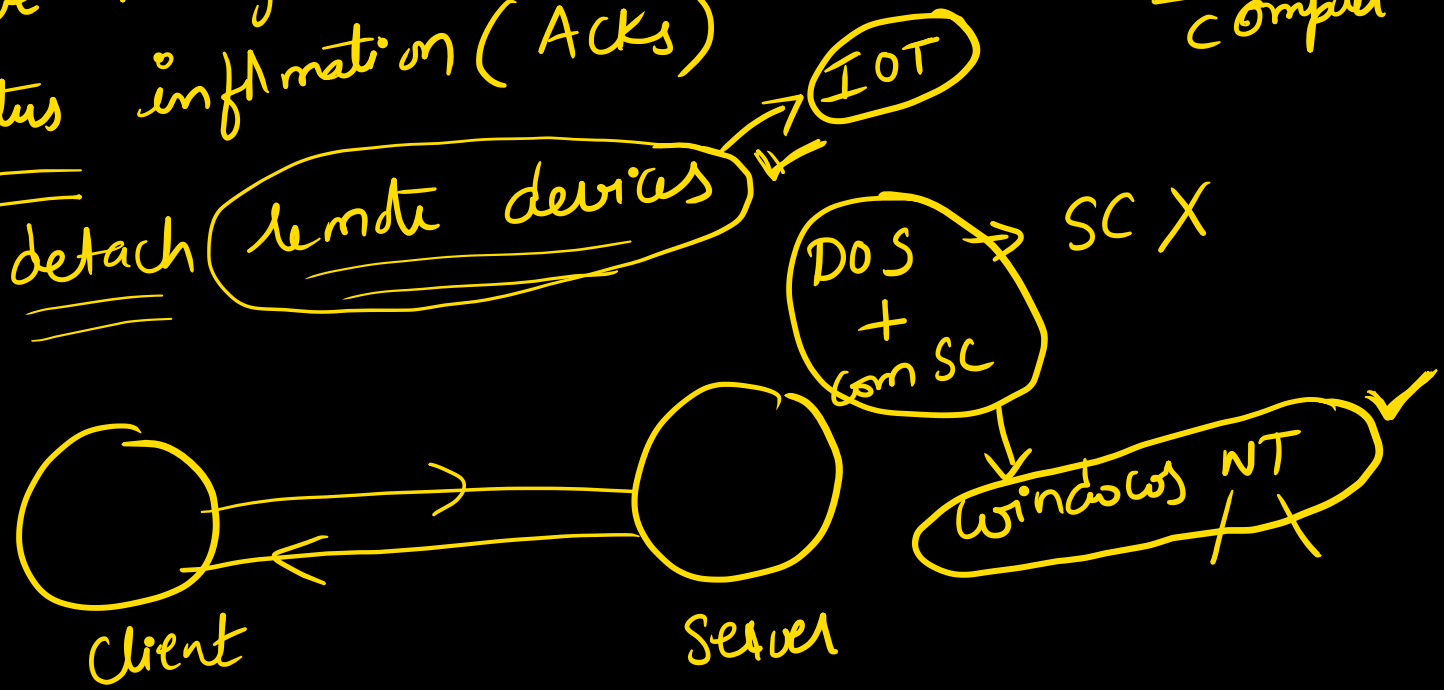
- → Get time & date, set time & date
- → Get system data, set system data (RAM, HD, CPU)
- → Get process, file & device data
- → Set process, file & device data



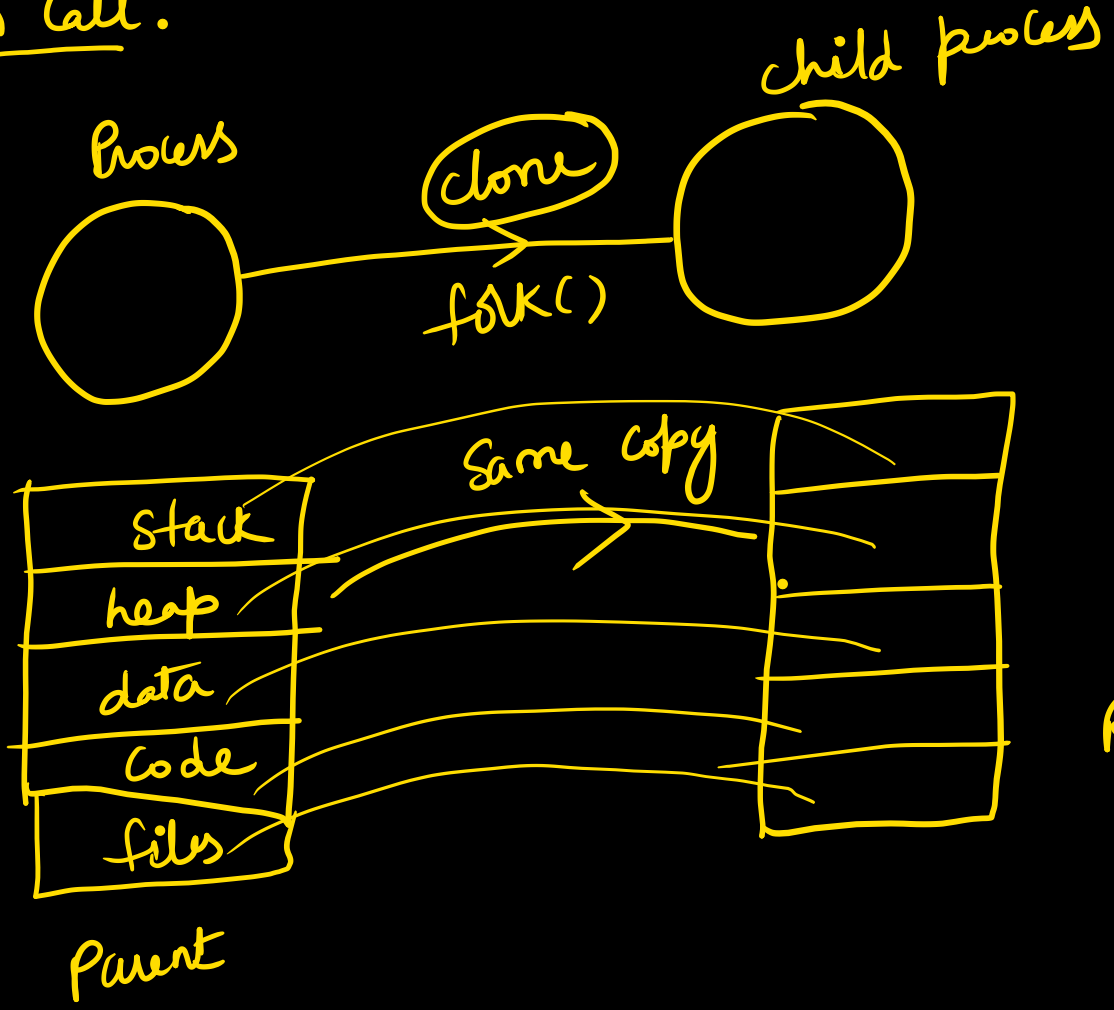
Communications related system calls: (Socket programming

- → Create, delete Communication Connection (
- → Send, receive messages
- → Transfer status information (ACKs)
- → Attach & detach (remote devices)



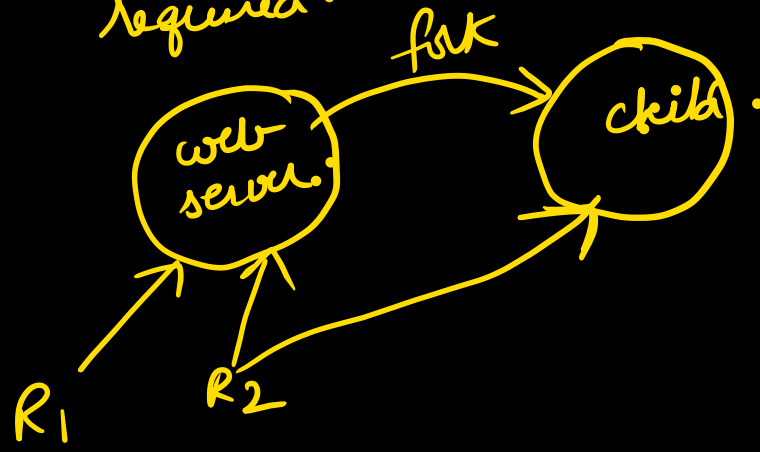


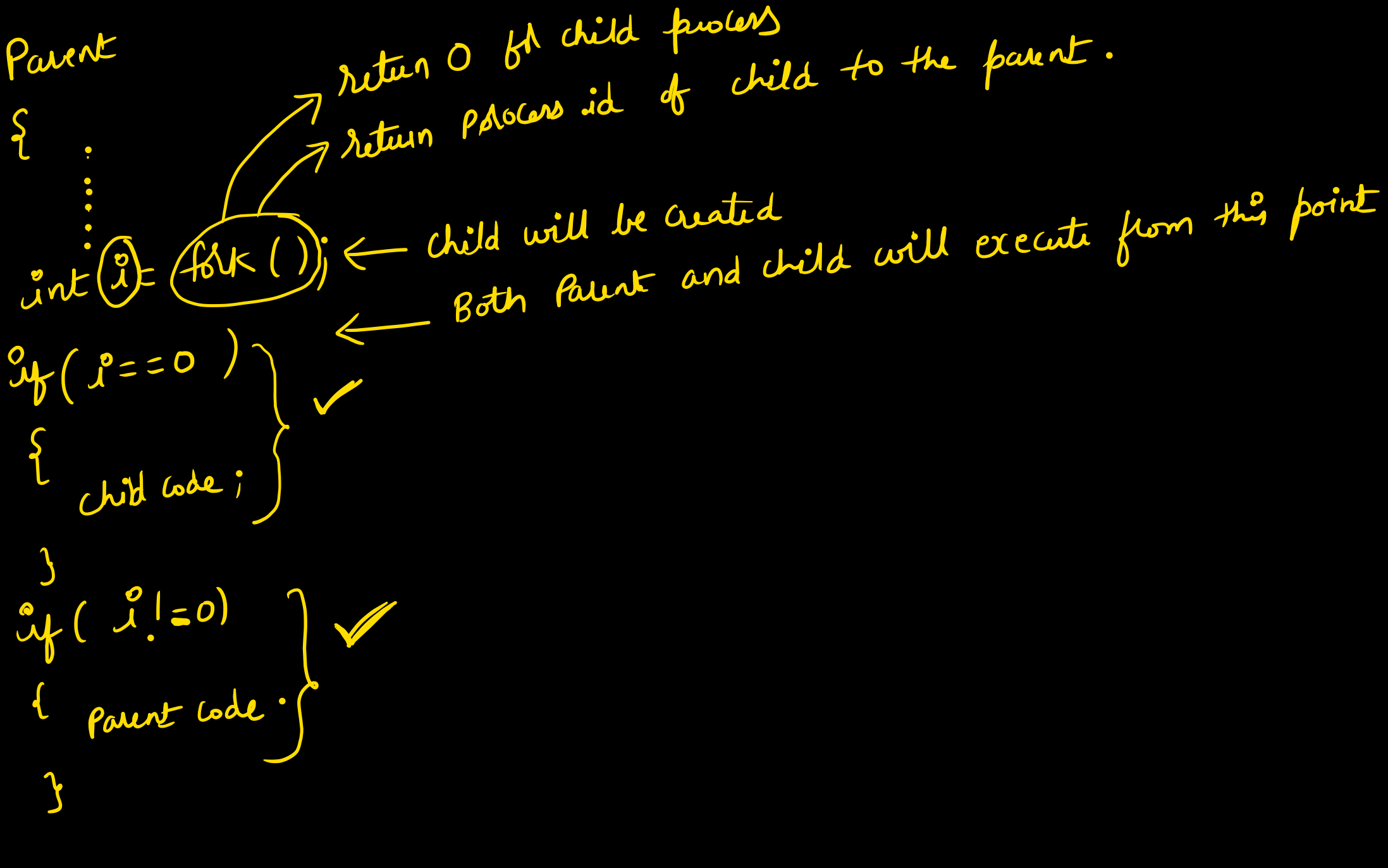
FORK System Call:



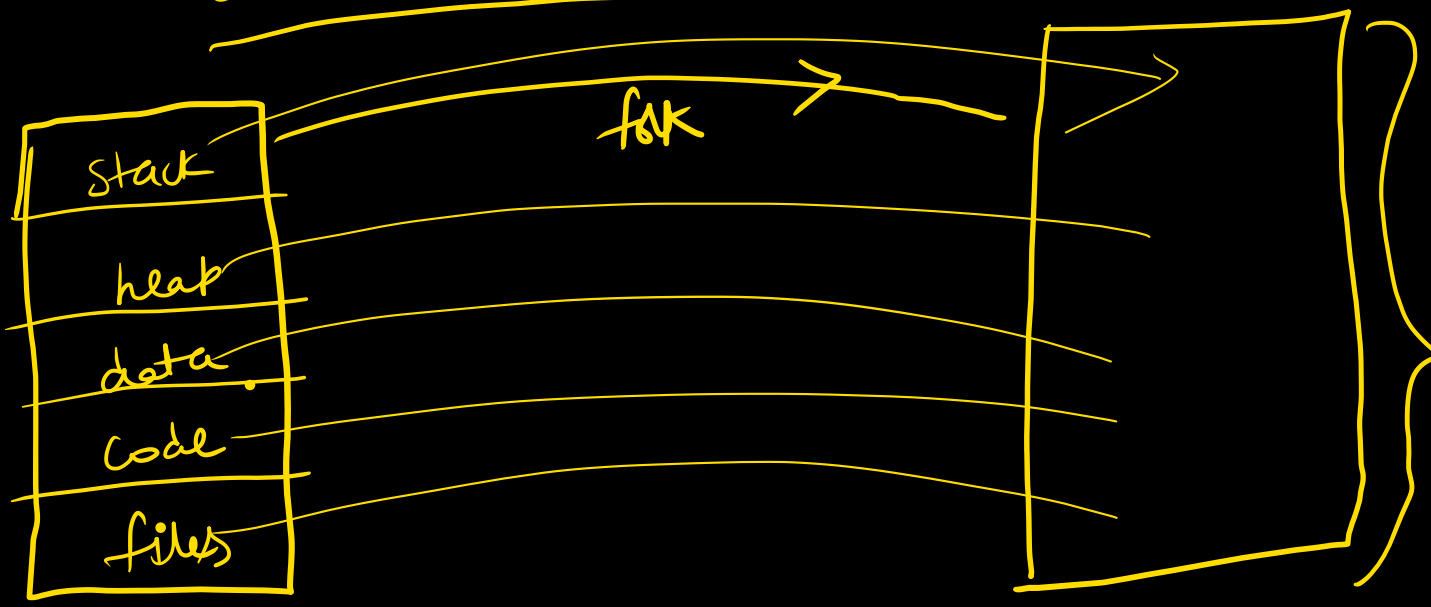
Process id → diff.

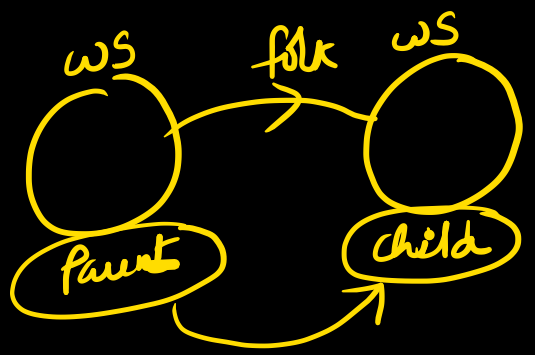
In some situations it is required.




Parent
{
int i = fork();
return 0 for child process
return process id of child to the parent.
Child will be created
Both Parent and child will execute from this point
if(i==0)
{
child code;
}
if(i!=0)
{
parent code;
}

Process VS threads:



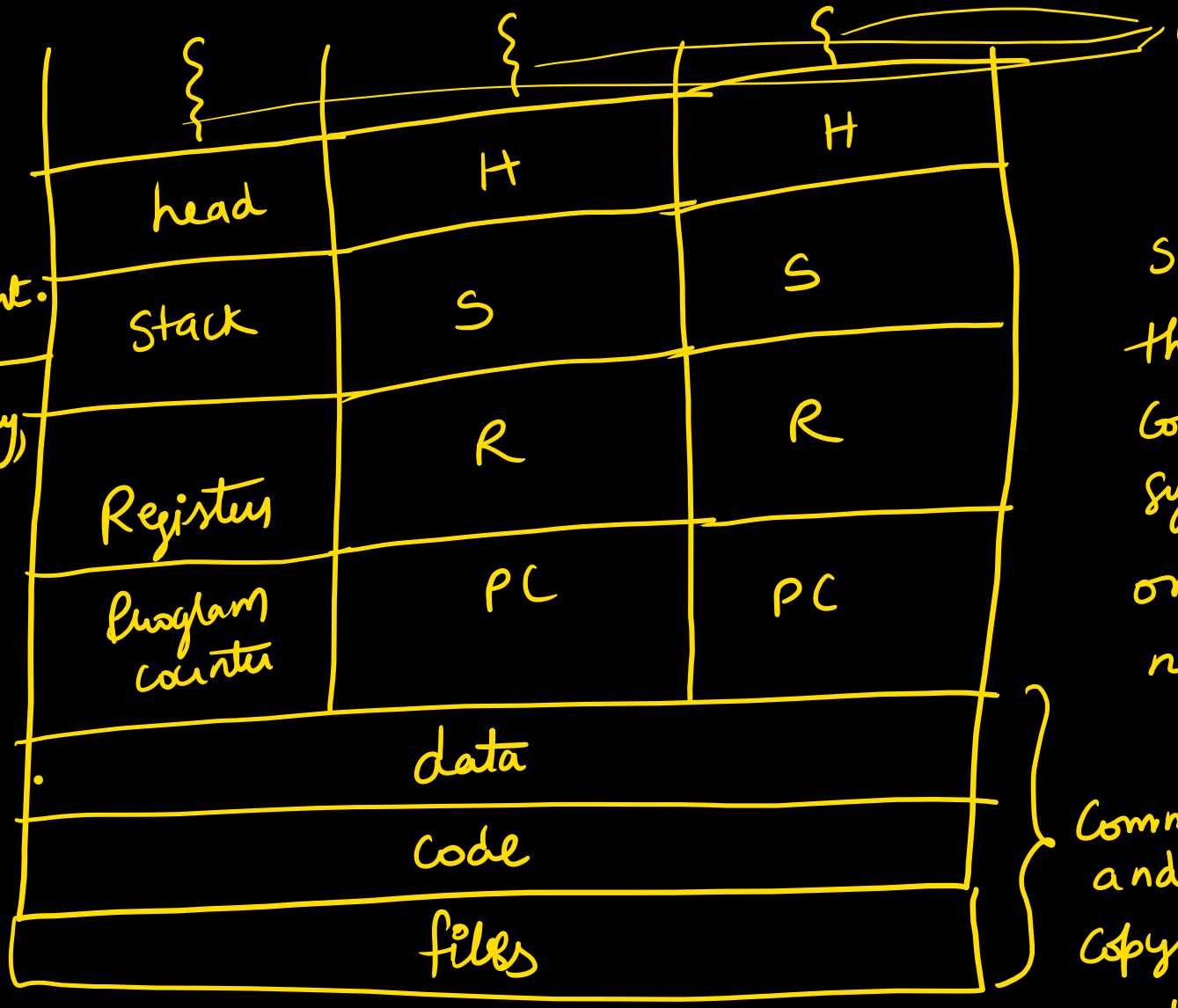


memory consuming
Context switching
→ Time consuming.

only diffrence in Registers and program counter.

Time and memory are wasted in fork.

Solution in threads



It is not time consuming and it is mostly efficient.

Threads: New, Ready, Running, Block, Terminated.

User level threads.

~~Kernel~~ OS doesn't know about threads.

So for switching between threads, no need of context switch & any system calls.

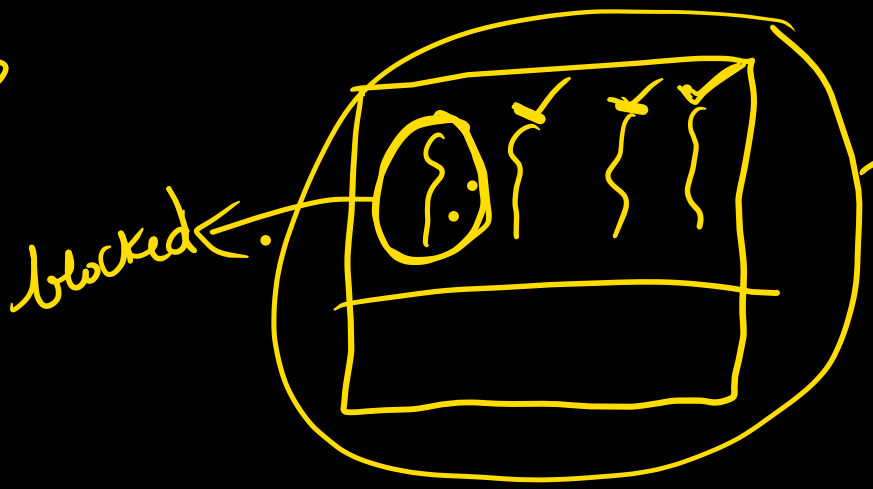
only Register switch is needed.

Common and no copying is needed

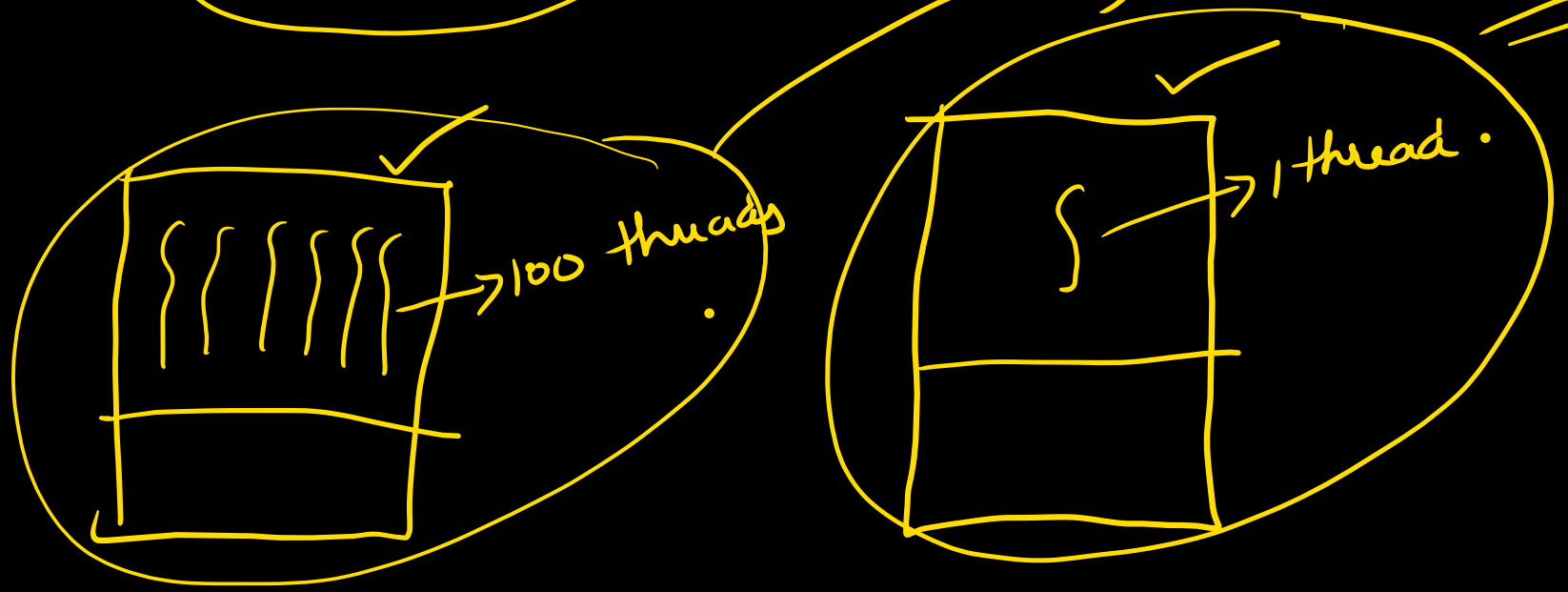
Process (FORK	Threads (user level)
→ System Calls are involved	→ NO system calls
→ Context switching is required	→ Register set switch only
→ expensive (Time & memory)	→ cheaper
→ NO sharing of data and code	→ Sharing of data and code.

Problems with user level threads:

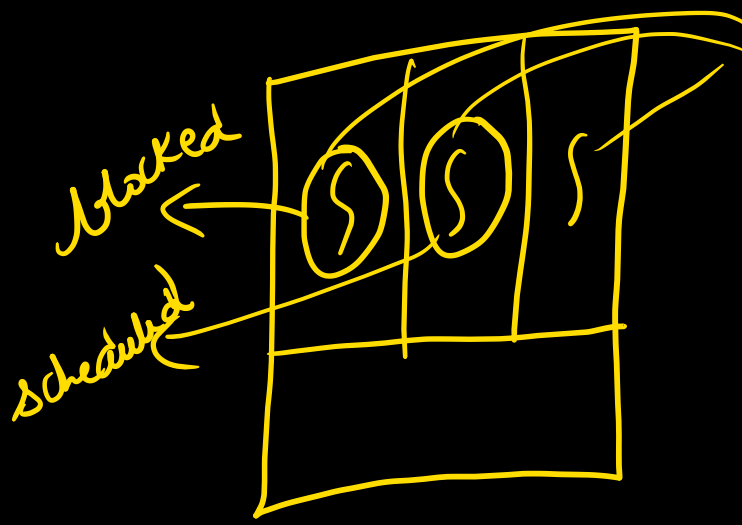
→ If one thread gets blocked, then entire task gets blocked.



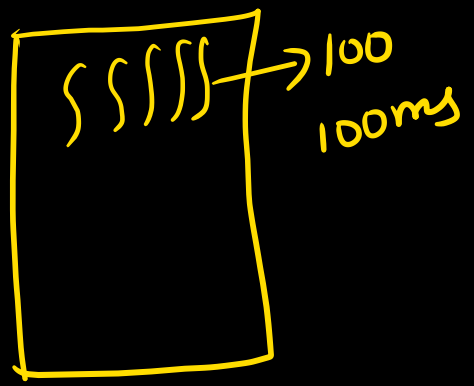
→ OS will give same execution time Ex: 1ms.

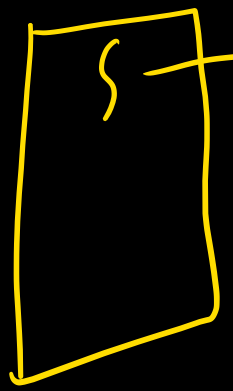


Kernel level threads: These are same as user level threads. But creation and scheduling is done by OS.



System calls for creation
System calls are required for switching (Context switching)





Expenses:

Processes > Kernel level threads > user level threads.

CN, OS → 15th

5th

10 days ahead.