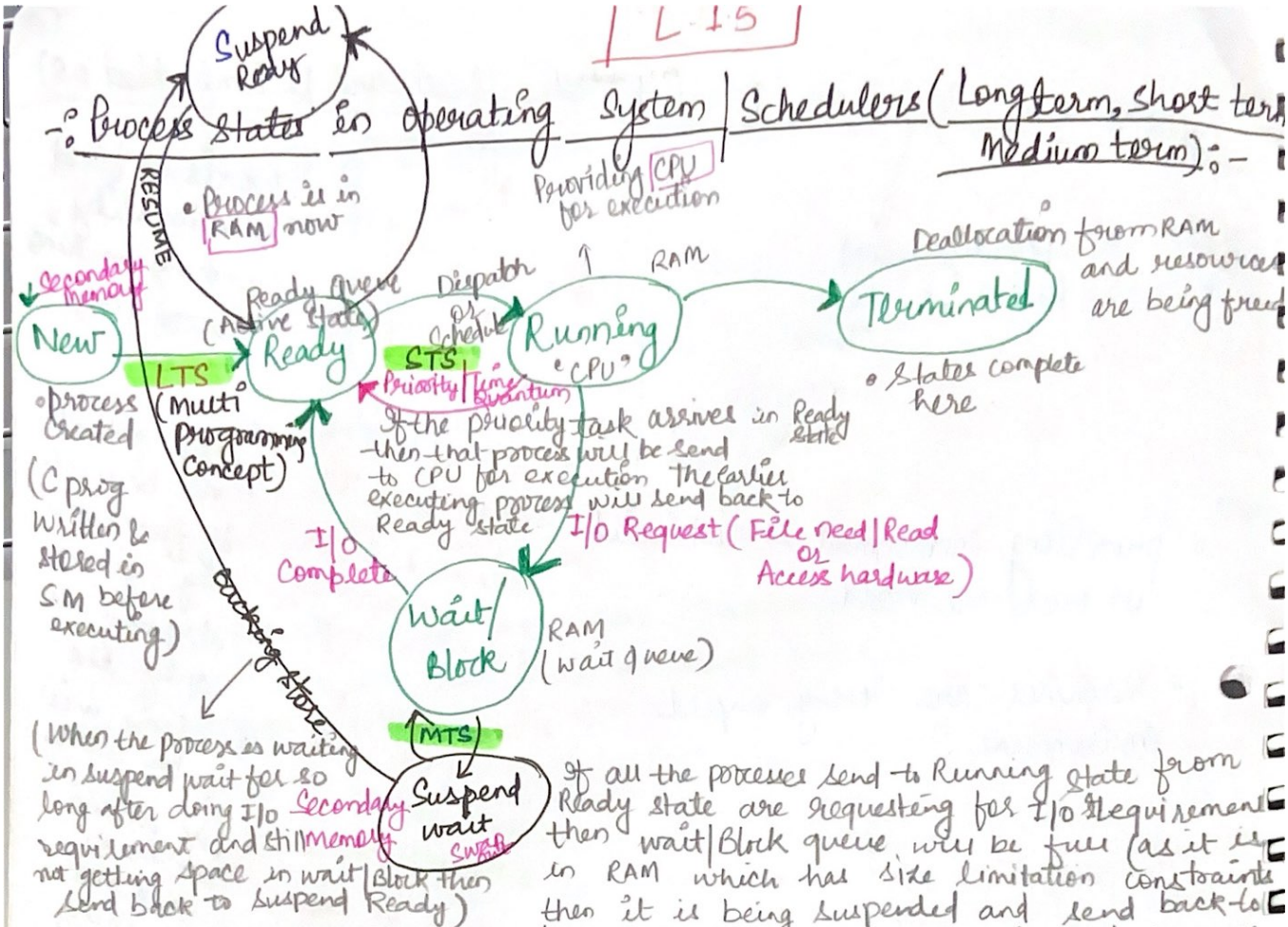
L-15

Process States in Operating System | Schedulers (Long term, short term, Medium term) :-



New → Ready

LTS (Long term Scheduler) :- To carry and place the no. of processes from New state to Ready state.

Ready → Running (Short Term Scheduler) Dispatch the Jobs :- how many processes are being forwarded to Running state from Ready state is dependent on the no. of CPUs you have.

Consider this
- uniprocessor system → 1 CPU
- Multi " " → MultiCPU (grid computing or Advance system)

↓ we can do parallel Processing

- process is still in RAM but now the state is being changed to execution

Imp • If CPU is executing the task (whole task) till the termination then it is known as multiprogramming. (Non-preemptive)

Imp • If one priority task comes and forces to be executed first then the running process/task in running states send the running task to ready state and execute that priority task, known as multitasking here. (preemptive)

Running → Wait/Block :- If one process at the time of execution in Running state request for some I/O resource then that process transferred to wait/Block state, does that I/O work and after the completion of that I/O work, goes back to Ready state. In this whole process, CPU does not sit Ideal for that particular process to execute its requirement.

Wait/Block → Suspend wait. :- whenever the RAM is fulled due to many processes then MTS does the work of swapping out the processes to Suspend wait

Medium term Scheduler

- Suspend task is only being done when your processes are full in RAM or in the state where RAM memory is used. or to free the space for VVIP process
- MTS based on least frequently, most frequently, priority assign the process to Suspend wait or Suspend Ready.

L.1.7

System Calls in operating system and its types :-

↓

is a programmatical way to shift from user mode to Kernel mode

Ex. I want to add 2 nos and want to print it on monitor or printer then add is the process work but to print the o/p to monitor or printer you want to access the hardware so here Kernel mode concept comes.

User mode	Kernel mode
• If user is using any app^n or API or writing a prog: then we ~~write~~ do this in user mode	• If we want to access any functionality of operating system then we have to go to Kernel mode

H/D access

e System call ?

→ File Related ⇒ open(), Read(), write(), close(), Createfile etc

→ Device Related ⇒ Read, write, Reposition, ioctl, fcntl (is a header file but generally used here)
ioctl → i/o control
fcntl → file control

If we want to access hardware then user can not directly access the hardware so system call gives user the privilage to access that requested hardware.

→ Information → get Pid, attributes, get System time and data

If we want to have the information ~~related~~ of attribute related to device or processes

Want to know

Process id, get ppid → parent process id (generally metadata i.e. audio file app^n → properties info)

→ Process Control ⇒ load, Execute, abort, Fork, wait, Signal, Allocate etc

If we want to load/execute etc to main memory

• Create multiprocessing environment

If we want to up the values of semaphore

memory allocation

• Parent process create child process and they both doing their work independently

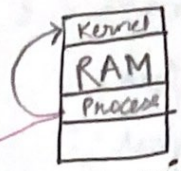
→ Communication ⇒ Pipe(), create/delete connections, Shmget()
↳ to get the shared memory value

use (Inter Process Comm^n) (two process are communicating)

- In some system we don't write system calls directly instead use APIs, local libraries (like printf, scanf() etc)
- whatever we are using either API or lib. etc they all are accessing system call (invoke Kernel to perform the task)

printf (is a lib. or function but what it is actualy doing is accessing system call, means we can say printf is same as write in file related work)

Ex Kernel & process both are in main memory



When we write 'C' program

When we execute it comes to RAM now created as process (in running mode)

If we want to access something in process and file privilage access is not in process then that previlage is given by Kernel, then process will invoke system call and will access particular file.

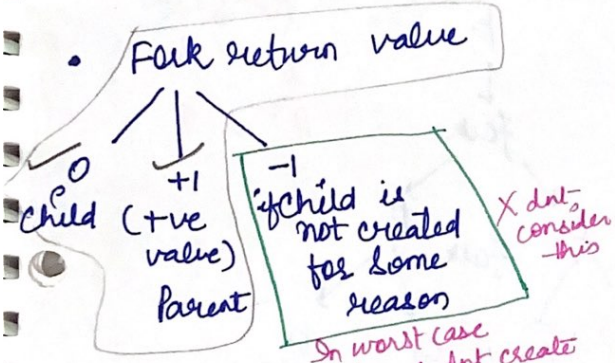
L.1.8

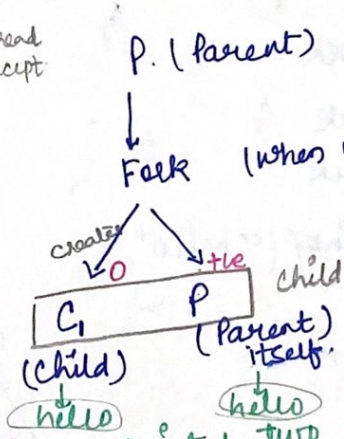
Fork System call with Example | Fork() System call questions:-

Fork(). → To create a child process

like clone or use thread concept

- Fork return value






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

→ Hello will be printed two times as child and parent both will give the output.

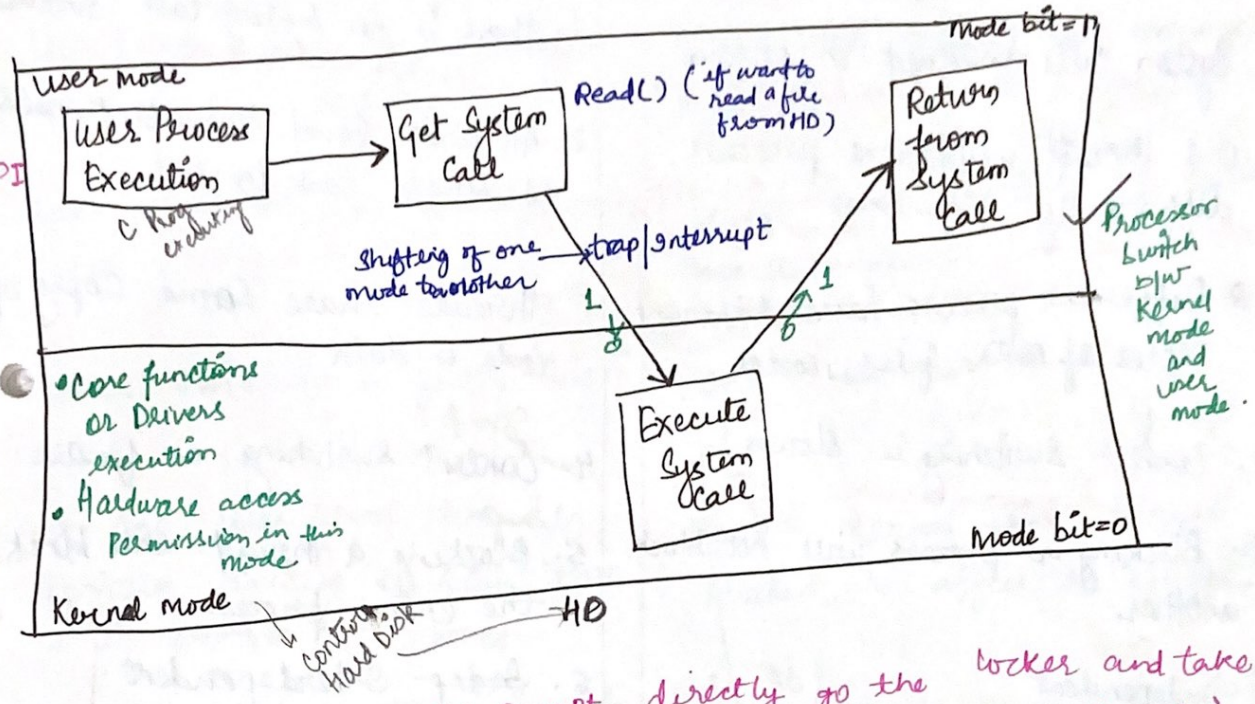
```
main() {
printf("hello");
}
```

← hello will be printed once.

L-1.10

User mode and Kernal mode in Operating System :-

(Dual mode)



Ex if we go to bank, we cannot directly go the locker and take the data / gold / cash out. to do so, we ~~ha~~ (as in user mode) approach clerk / manager (in Kernel mode) to give access to open locker then clerk / manager having all the access can give us the ~~perm~~ permission.

Operating System → Resource Management work here.

Ex whatever you are doing either MS word, Youtube, Paint etc through API is in user mode.