Syllabus of OS:~

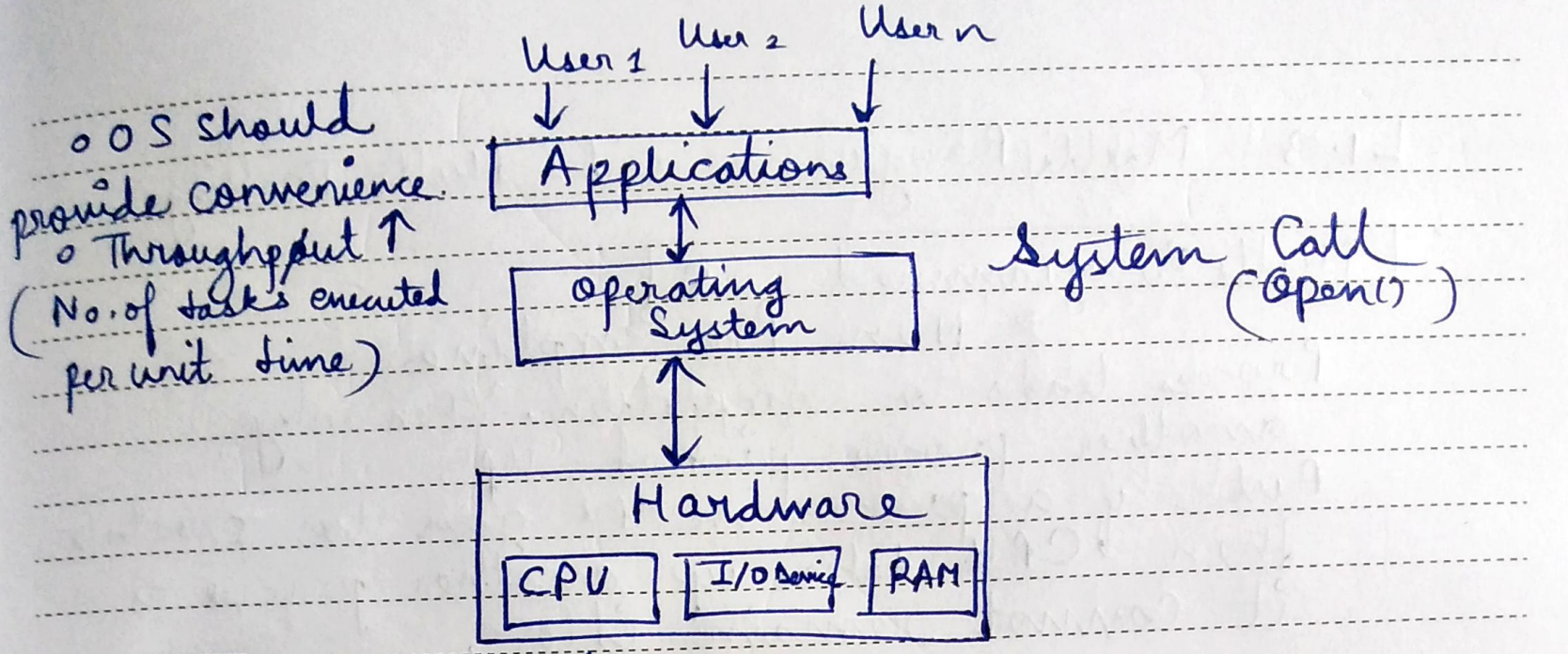
- Basic Introduction → Types of OS
  - Process diagram
  - System Calls
- Process Scheduling → FIFO, SJF, PR, Round Robin
- Process Synchronization → Semaphore
- Deadlocks & Threads → Banker's Algorithm (Theoritical)

- * Memory management → Paging
  - Segmentation
  - Fragmentation
  - Virtual memory
  - Page Replacement Algo
- * Disk scheduling →
  - SCAN algo
  - CSCAN
  - FCFS
- UNIX commands → ls, mkdir, cd, chmod, open
- File Management, Security →
  - Sequential Access
  - Random access
  - linked access

L-1.1 Introduction to OS

- OS is system software
- Interface between user & hardware.
- RAM → main memory
- Secondary memory → hard disk

₹



- OS should provide convenience
- Throughput ↑ (No. of tasks executed per unit time)

System Call (Open())

OS:- functions:-

- Resource Manager
- Storage Management (HD) (through file system)
- Process Management (CPU scheduling)
- Memory Management (RAM)
- Security

L1.2 Types of OS :~

- Batch

→ Punch Cards, Paper tape, Magnetic Tape
Jobs were offline loaded into this.
Jobs given to operator
→ Similar jobs grouped into batches

B_1 B_2 B_3
2 3 4

CPU ⇄ I/O

When jobs goes to I/O CPU is idle
(non pre-emption)

IBM Fortran IBSys 1960

L1.3 Multi Program and Multi Tasking

Multi Programmed OS

→ Non Pre emptive

Process leads to completion. Then only another process picked up.

But, if a process in CPU goes to I/O state, then CPU takes up another process as it cannot remain idle.

→ less idleness

Multi Tasking OS → Pre emptive / Time Sharing

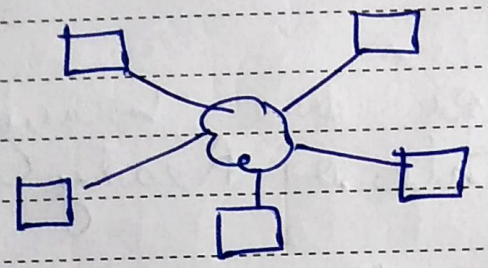
Adv :- Responsiveness

→ Used mostly in our laptops

L.1.4 Real Time OS, Distributed, Clustered, Embedded

- Real Time → No delay
  - → Hard → Strictly No delay allowed (missile)
  - → Soft → Yt live stream, gaming

- Distributed → loosely coupled environment

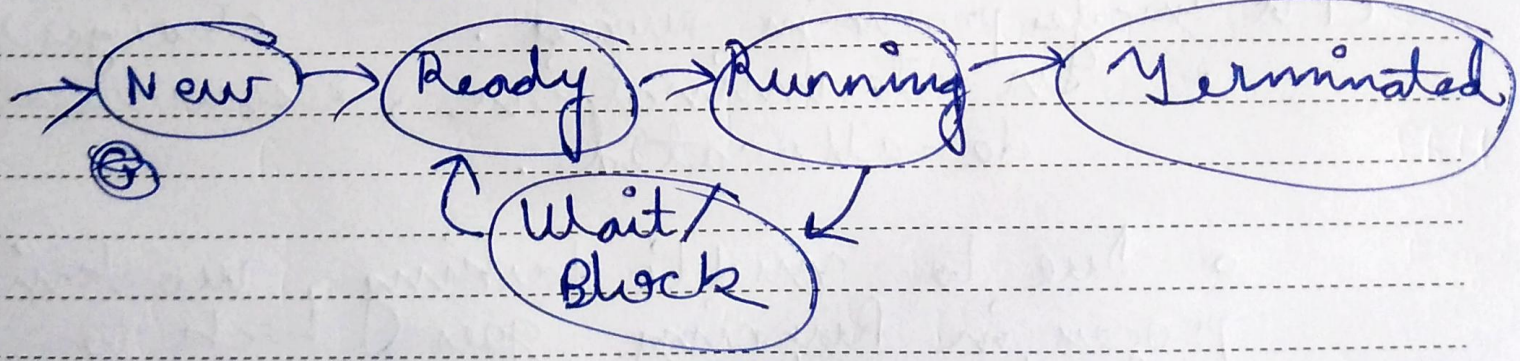


- Clustered → ◦ LAN
  ◦ Multiple computers used to inc computation power
  ◦ Load balancing done
  ◦ Scalability done

- Embedded → ◦ AC, Microwave
  ◦ Same functionality for a particular OS.

L 1.5 Process States in Schedulers
Schedulers (Long term, Short term, Medium)

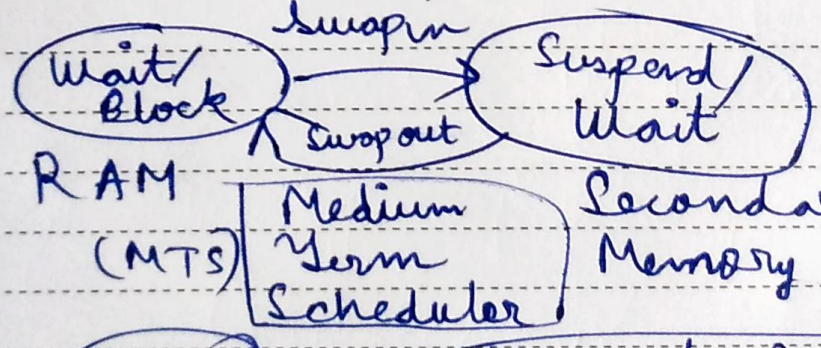
- A model to make user understand this.



- Many processes are in secondary memory
- Some processes are brought to Ready state or Ready Queue (RAM)
- Who brings them?
- LTS or long Term Scheduler
- Bring as much process in Ready (~~long Term~~) Multi Programming.
- Some processes dispatched (Short Term Scheduler) from Ready Queue.
means we take it to Running state and want to create them
No. of processes in Running state depends on CPU.
Its still in RAM. (address in RAM changes)
CPU executes proces in running.
- In termination, resources are de-allocated.
- Due to multi tasking, working process in Running goes back to Ready and the priority process

goes from Ready to Running.
This might also happen due to ~~of~~ time quantum (Pre emptive)

- But if proces goes directly from Running to Termination, it is non premptive (without any stopping in between)
- (Wait/Block) State (in RAM again) happen due to I/O request.
- After I/O completion, process gets to Ready.
- Now, if all process do I/O, i.e go to wait/block state, then process is suspended or sent to secondary memory.



- Also, Ready → (MTS) Suspend Ready occurs if too many process from secondary memory brought to Ready State (RAM).

Notes :

- From Suspend —Backing Store→ Suspend Ready

might happen if even suspend is full.

- PS command to get process information.

1.6 Imp Linux Commands

Qs) Which command is used to assign only Read permission to all 3 categories file node?

chmod is used → change mode i.e.
(read, write) → types of modes

r w x r w x r w x
u g o
(user)

Why 3? (each)
A → 3 permissions → r, w, x (x, y, z that x)
each

user, group, other
u g o
(read, write or execute (x))

Lets say, we don't want to give execute to others we will leave it as blank

r w x r w x r w -

Options :-

a) Chmod a-rw
b) Ch mod go+r note
c) chmod ugo = r note
d) chmod u+r, g+r, o-r note

Qs) 'Chmod ugo+rw note' command can be represented as in octal notation as

Ans:

u	r	4	→ fixed
g	w	2	
o	x	1	

rwx → 7, ug → 6, rx → 5

Options:-

a) Chmod 555 note
b) Chmod 666 note
c) Chmod 333 note
d) Chmod 444 note

User given read write → 6
group " " " → 6
Others " " " → 6

Qs) Suppose you have file "f1" whose contents are:- 1234567890abcdefghij
here 'lseek' is used 2 times sequentially
lseek(n, 10, SEEK_CUR);
lseek(n, 5, SEEK_SET); n is file descriptor
After applying lseek two times, what will be current position of R/W head?
(Index starts from 0)

Ans 2 lseek's System call.
to lseek, read, write, fork →
These System calls are important.

To randomly access data, we need to move Read / Write HEAD.
Read / Write head is by default at 1st position (index 0).
To move head, we use lseek command.

→ 10, SEEK_CUR → move head 10 positions ahead.

→ 1 2 3 4 5 6 7 8 9 0 a b c d e f g h i j

→ 5, SEEK_SET → move to 5th position

1 2 3 4 5 6 7 8 9 0 a b c d e f g h i j

Option :-

a) 0 b) 5 c) 10 d) 15

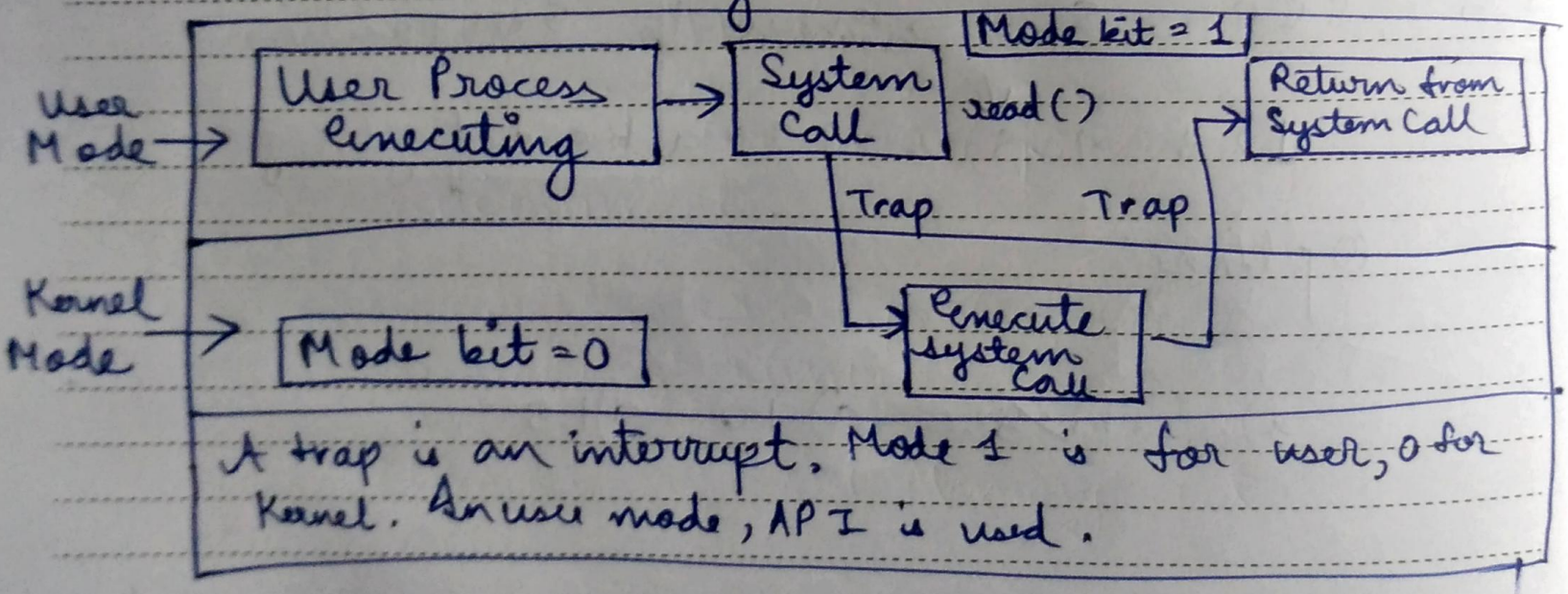
1.7 → System calls in OS

But before this L.1.10 to know about user, Kernel mode

L 1.10

Processor keeps switching between user and Kernel mode.

When we use ~~you~~ an application, we use User mode, but when we need to print something ~~on it~~ on screen, we switch to Kernel via ~~int~~ interrupt as screen ~~is~~ or monitor is hardware, which can be accessed only via Kernel or OS.



A trap is an interrupt. Mode 1 is for user, 0 for Kernel. In user mode, API is used.

To use OS functionality, we go to Kernel mode.

Back to 1.7

- To access file in hard disk, switch to Kernel mode.
- In linux, when we open text editor, we can directly use system calls in it.
- Either directly use system call or through an API of function like printf to access Kernel.
- Around 700 system calls in windows.

System Call

*) → File Related ⇒ open(), read(), write(), close(), create file

~~In Running~~

Lets say we have a C file, we compile and run it. In running mode, a process is inside RAM. The process was created when we started running the program. Kernel is also inside RAM. RAM is main memory.

If we want to access file, that privilege is not in process. That priviledge is in Kernel, for which System Call is invoked.

☆ Device Related ⇒ Read, write, Reposition, ioctl, fcntl

ioctl → input output control
fcntl → file control (its a header file)

*) Information ⇒ get Pid, attributes, get System time and data

Pid → Process Id

☆) Process Control ⇒ load, execute, abort, Fork, Wait, Signal, Allocate etc.

Fork used for multi processing environment.

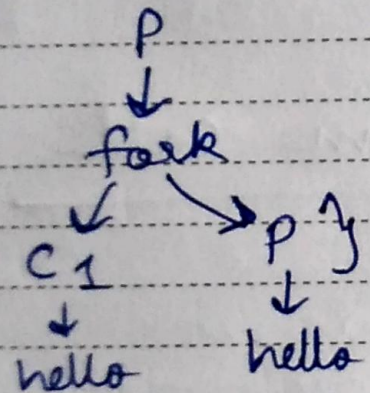
☆) Communication ⇒ Pipe(), Create/delete connections, shmget()

Shmget → Shared Memory get

L-1.8 → Fork()

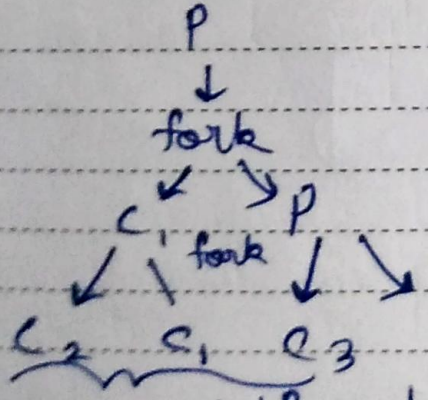
If there is parent process, and we write fork(), then child process is created.
Parent has unique id.
Child has unique id.

returns
- 0 → child
- +1 → +ve → Parent
- -1 → child not created



Parallel processing

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

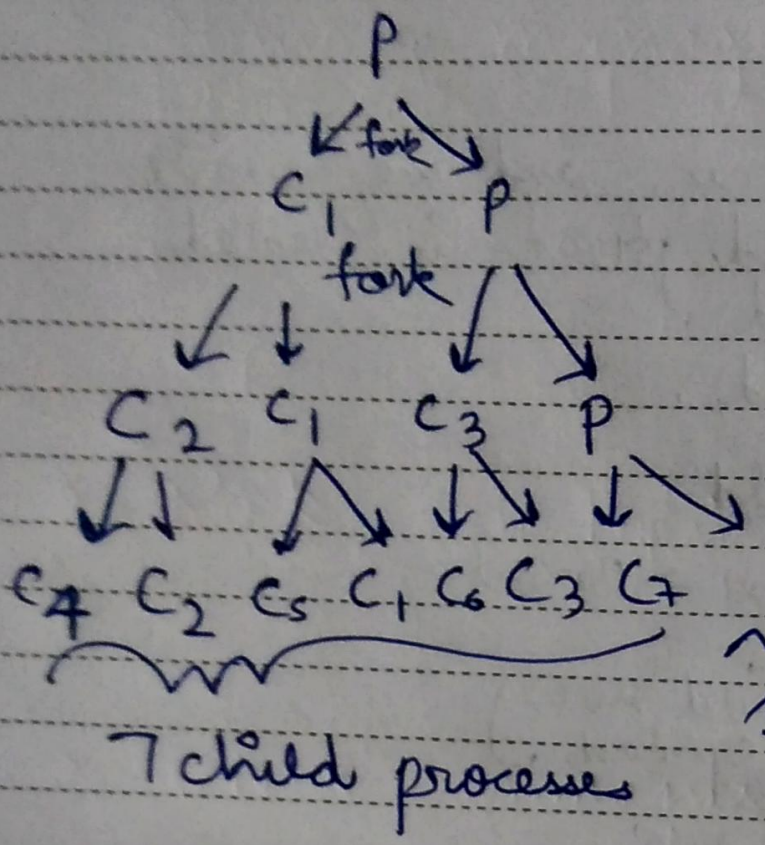



```
main () {
  fork();
  fork();
  printf ("hello");
}
```

1 time hello printed.

No. of times printed = 2^n (n = no. of times fork written)

No. of child processes = $2^n - 1$

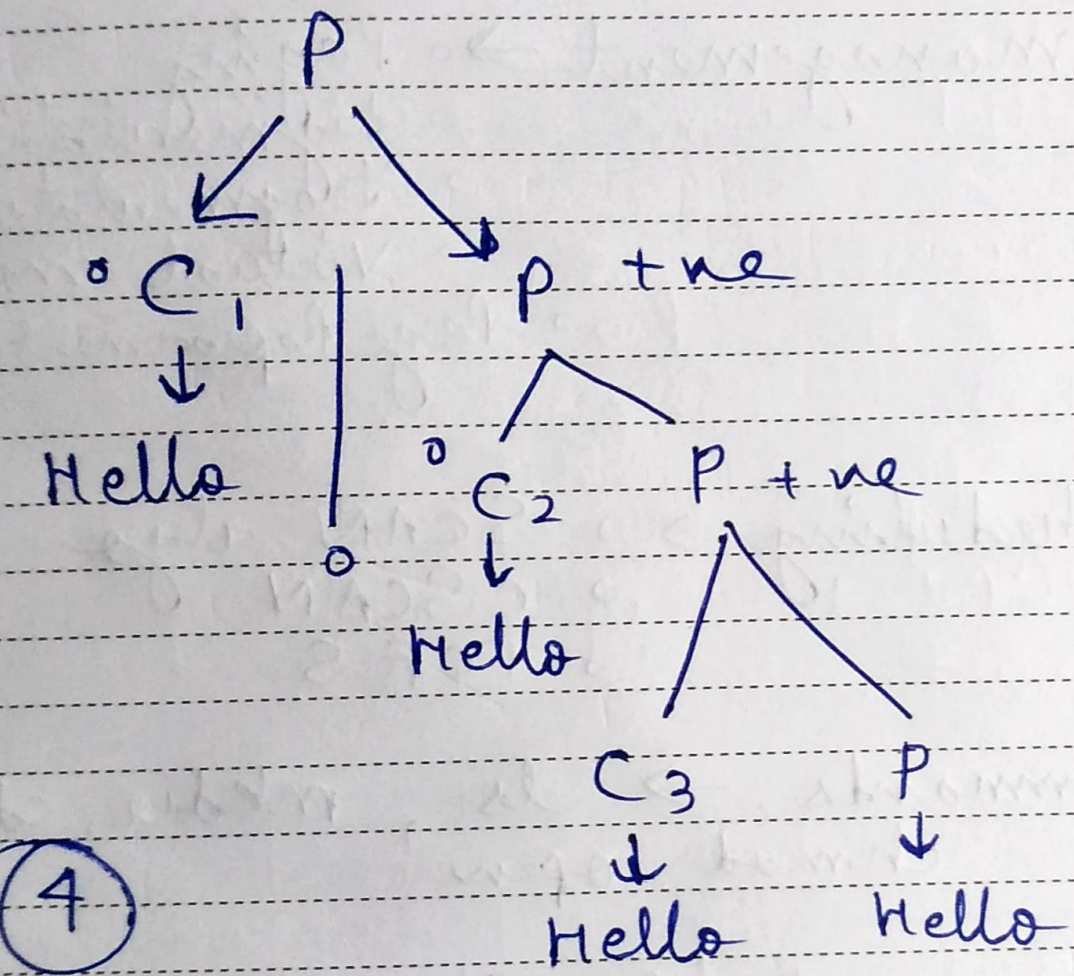


8 times hello printed if fork written, 3 times.

L→ 1.9 Qs on Fork system call

```
#include <stdio.h>
#include <unistd.h>

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


P
2 cases
• Child → 0
• Parent → +1 (+ve no.)
C1
Hello
0
P +ve
C2
Hello
P +ve
C3
Hello
P
Hello
4