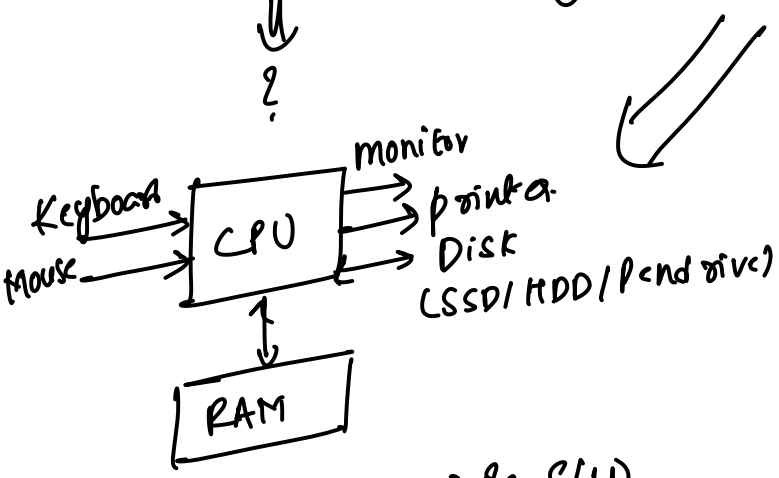
Operating Systems

P. Sailesh Kumar

Operating Systems

1) It is a Resource manager.



Resources can be H/W & S/W

Makes the life easy for S/W engineer

(like how printf works internally
the S/W engineer doesn't need to know)

* OS can be broadly considered as a resource manager.

Eg Browser runs on our S/W.

Eg Search a file, OS provides this utility, basically it talks to hardware.

* OS is a piece of software (Most OS's are implemented in C/C++)

A program can be broken into multiple processes

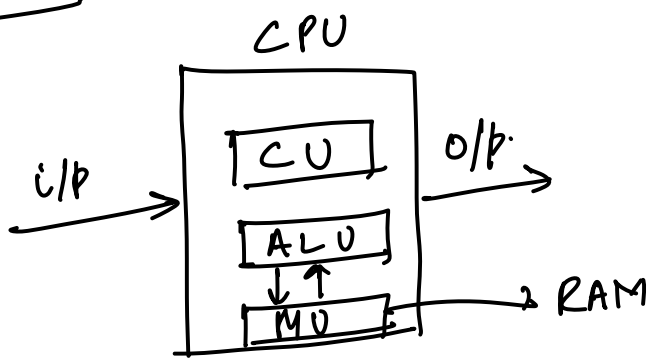
each process uses CPU's resources.

We can see list of processes in our task manager (Windows)

Important topics

1) Process management 2) Memory management
3) File Systems & Device management.
4) Protection & Security Mechanisms

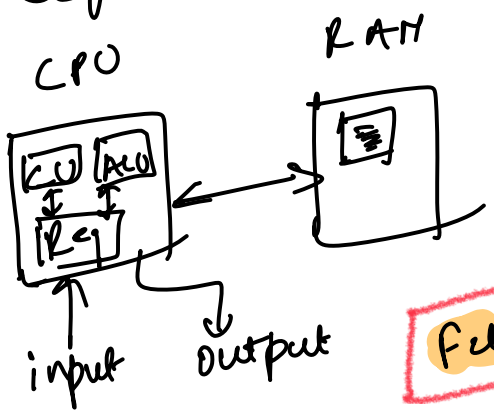
Let's understand Architecture first



Any program we want to execute, it is stored in RAM.

↓

Sequence of instructions.



Equity PC

PC → Program → Address of current instruction.

MAR → Memory address register

MDR → Memory data register

CIR → current instruction register

Instruction cycle

Fetch

1) Address in PC is copied to MAR
2) Increment PC
3) Instr. found at MAR is copied to MDR
4) Instr in MDR is copied to CIR

Decode

5) CU takes over now & decodes the CIR
6) CU sends signal to ALU

Execute

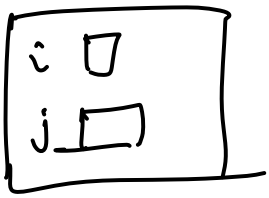
RAM

102 : ADD (120)(121)
(14) (13)

Take val stored at 120 & 121
add them & store back at 120

Interrupts:

```
main()
{
  int i;
  int j = 3+2;
  j = j*2;
  scanf("%d", &i);   // keyboard i/p
  i = i/j;
  printf("%d", i);   → Print to monitor
}
```



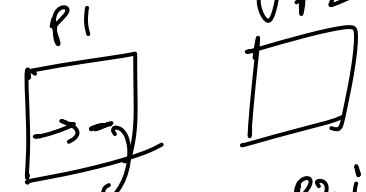
Now CPU is waiting for keyboard & hence it is a interrupt

CPU is in idle state

Uni Vs Multi-programming

Multi → Multiple programs executed simultaneously

Uni → one program executed at a time.
Huge wastage of resources



P1 Interrupted → CPU is idle now

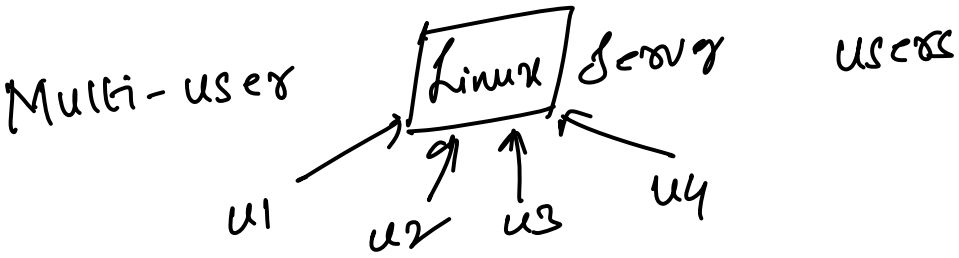
P2 is not executed

We want P2 to load in RAM when P1 is not in execution

Multi-programming Vs Multi-Tasking

(Unix Terminology) (Windows Terminology)

At the root both are same.



Types of multi-programming

1) Preemptive (Forcefully removing)

→ Force a process to give away control of resources in CPU

Eg Win10, Linux, Android.

OS can interrupt the current program from running at any given point of time to let another program use the CPU

2) Non-preemptive

Process has to give away control itself.

↳ completion

↳ I/O event

Eg Win 3.0 & Win 3.1

Non-preemptive OS lets a program run, until it finishes or gives up voluntarily.

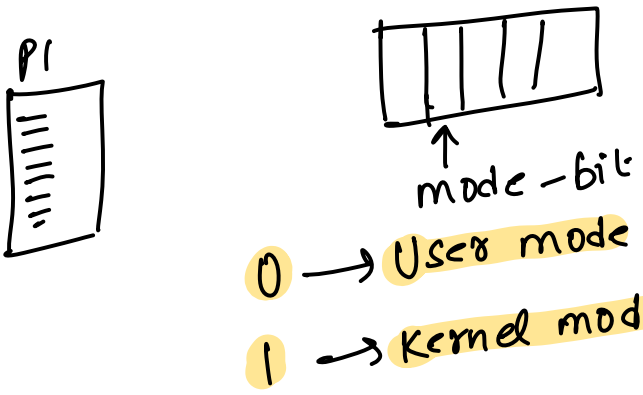
Pre-emptive multi-programming is Today's World

Modes of execution in CPU

1) User mode

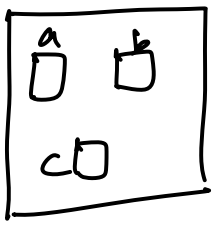
2) Kernel mode
   - core of the operating system
   - controls & manages most of the resources
   - Non-preemptive (No one can alter it!)
   - Atomic execution (everything is executed)

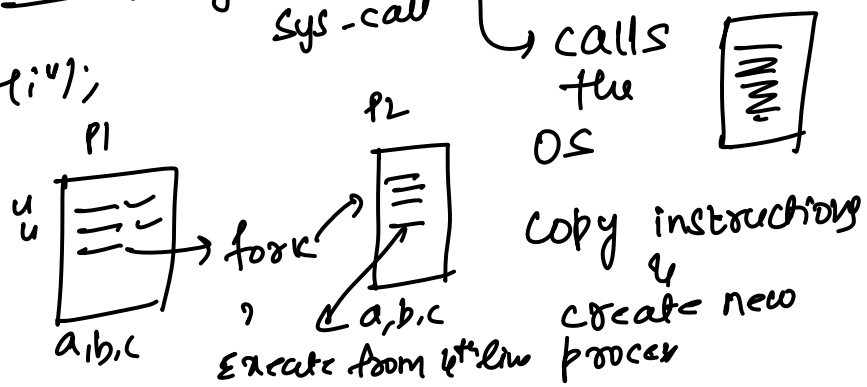
For every program/process there is a data structure that OS maintain called as PSW (Process Status Word)




```
main()
{
  int a,b,c;   // user mode
  c = a+b;
  fork();  ──→ System call
  printf("Hi");
}
```

sys-call → calls the OS





fork → Sys-call → OS (kernel mode) → duplicate

Execute printf in P2 & go back to P1

P1 → Parent process.

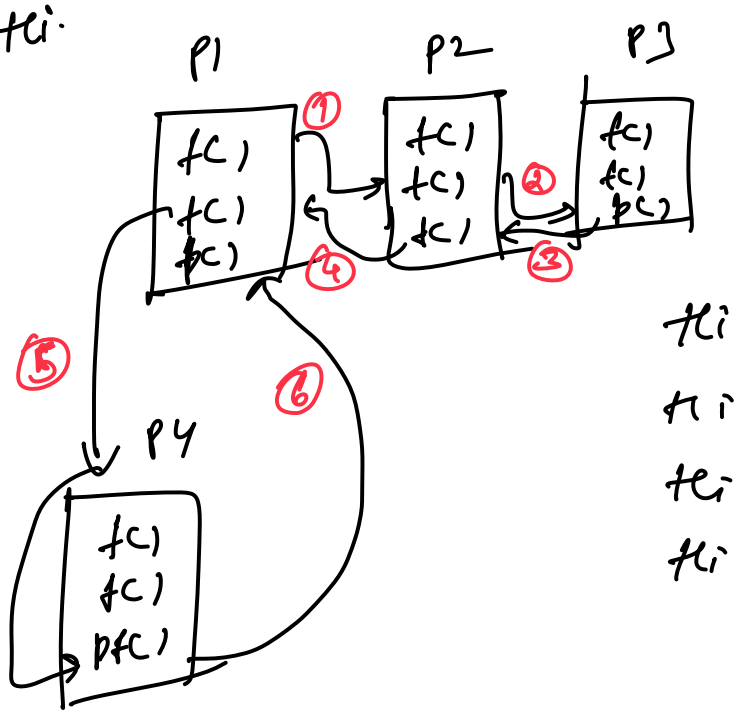
P2 → child process.

* fork copies the content of parent process & duplicates that in child process
* In child process the line printf "Hi" is executed (line after fork)
* It comes back to parent process, points to Hi

So Hi Hi.

Eg 2.

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



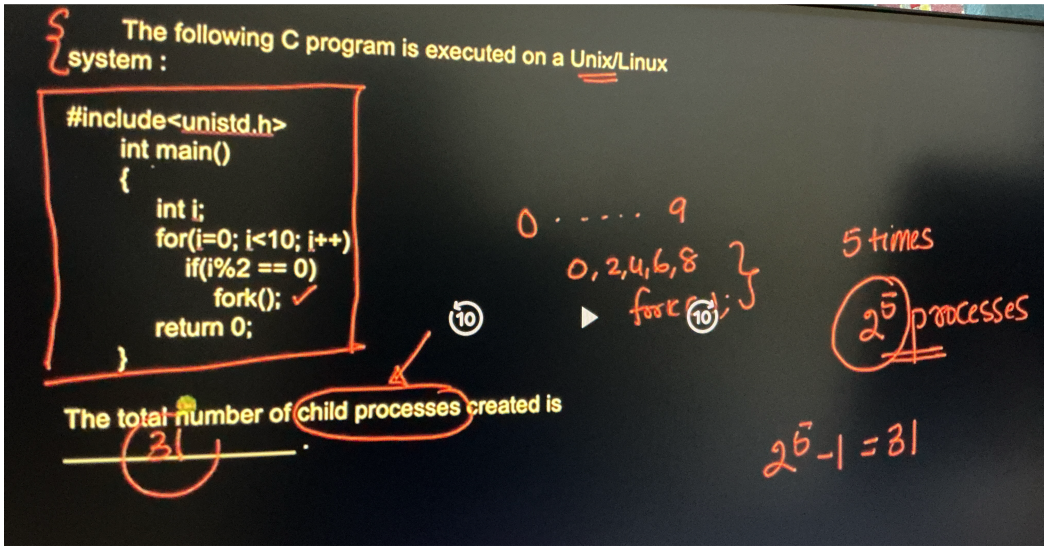
Hi
Hi
Hi
Hi

2 fork → $2^2 \rightarrow 2^1$

n → forks ⇒ 2^n processes

Just iterate it manually.

Eg:


The following C program is executed on a Unix/Linux system :
#include<unistd.h>
int main()
{
int i;
for(i=0; i<10; i++)
if(i%2 == 0)
fork();
return 0;
}
0 9
0, 2, 4, 6, 8
fork();
10
10
5 times
2^5 processes
$2^5 - 1 = 31$
The total number of child processes created is
31
.