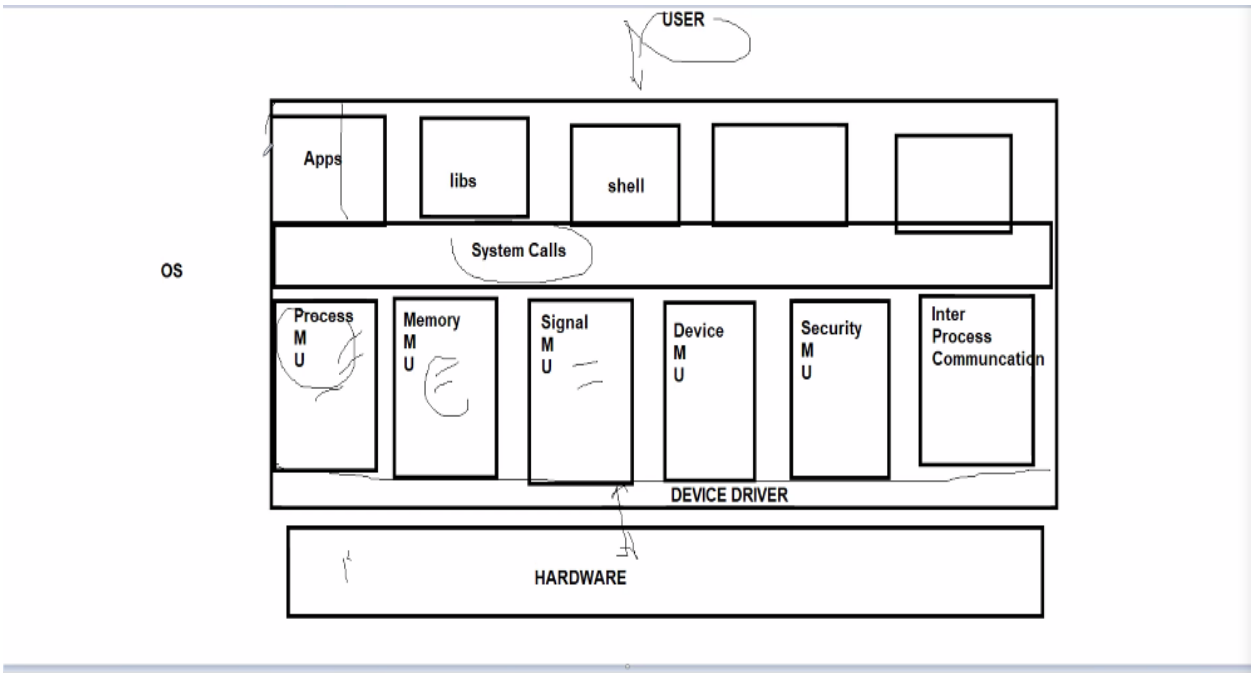
LINUX

An os is an interface bw user and hardware

OS manages the hardware

ARCHITECTURE OF OS



Lower most level of OS is device driver

It interacts directly with hardware .

device drivers has 3Cs :

Control

Configure

Coordinate

a) Device Driver

b)Process MU

c)Signal MU

d)Security MU

e)Inter Process Communication

f)System calls are fns and already a part of linux kernel

No library required for system call

eg:create/kill process ...we need a system call

SYSTEM CALLS

mkdir carl_zeiss

cat > file1

cat file1

cp file1 file2

(cp is done with the help of many system calls)

open f1 read only mode

open f2 create and write only mode

read f1

write f2

close f1

close f2

strace cp file1 file2:

Everything happens with the help of system calls

PROCESS DESCRIPTORS IN LINUX

0-standard inp(stdin)

1-standard op(stdout)

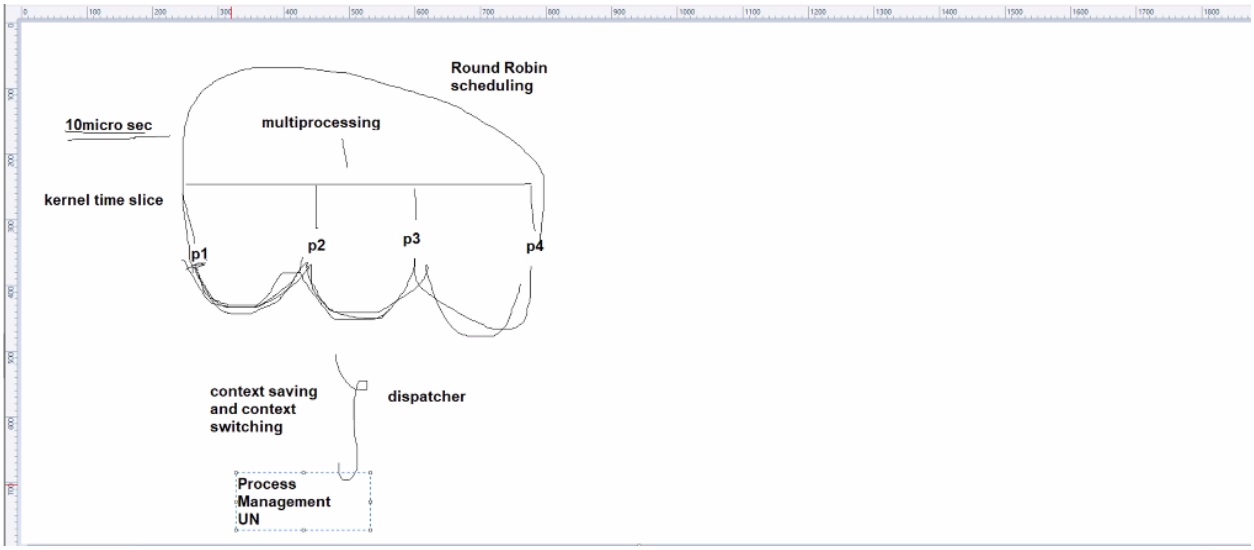
2-standard error(stderr)

printf will internally call 1 stdout

Everything is in terms of system calls

---

a) -> f) Linux Kernel



***1 processor can execute only 1 process at a single time

processor speed :in nanoseconds

On shifting from 1 process to another :(ROUND ROBIN)

save the current context address (CONTEXT SAVING & CONTEXT SWITCHING)

CONTEXT SAVING & CONTEXT SWITCHING DONE by Dispatcher

All of this is a part of Process MU

pid can never be 0..always +ve

When linux boots , the first process that starts is init /systemd process with pid=1

2 terminals =>2 bashes

1st terminal :ps

(ps will become child of bash1) 2nd terminal :ls

ppid of ps will be equal to pid of bash1

```
prodeep-200905384@prodeep-200905384:~/Documents/LinuxPractice$ ps -o pid,ppid,cmd
    PID    PPID CMD
  21960   21952 bash
  23041   21960 vi file1
  65329   21960 ps -o pid,ppid,cmd
prodeep-200905384@prodeep-200905384:~/Documents/LinuxPractice$
```

Note:ps will only show process in current terminal

**ps -e : shows all processes in system

pstree : shows process in the form of a tree

pstree -p:shows process with pid in ()

Note:gnome-terminal is the parent of all 2 bashes

Note:Process name can be same but pid always different

Note: vi file1.c (vi editor)

Note: sleep requires unistd.h

Note:ctrl c to terminate infinite loop

---

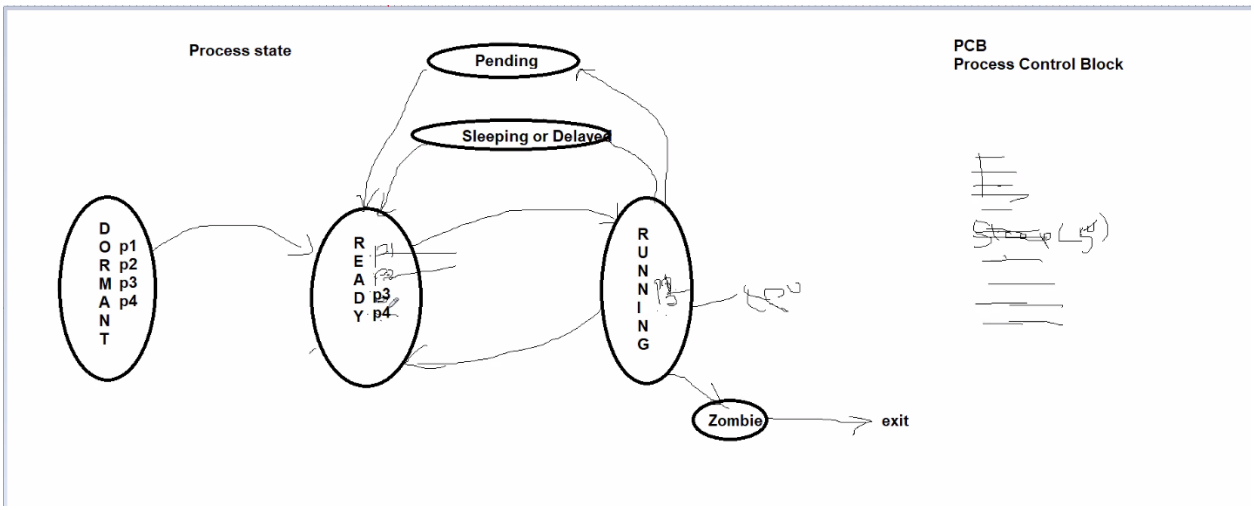
int x,y;

x=getpid();

y=getppid();

//2 system calls

Process States



1)Dormant :Processes yet not created

2)Ready:./p1 ./p2 ./p3 ./p4

Process are in queue in Main Memory

3)Running:Dispatcher picks one of 4 processes

eg:p1 goes to running

executes for kernel time slice ..10ms

p1 goes back to ready state

Same for p2,p3,p4

4)Sleep:

Assume p3 is running

and inside p3 , there is sleep(50)

p3 goes to sleep state

only p1,p2,p4 in ready state

after 50s , p3 goes from sleep to ready state

5)Pending :

a)In semaphores

b)When parent has to wait for child to finish for indefinite time

Process here goes to pending state

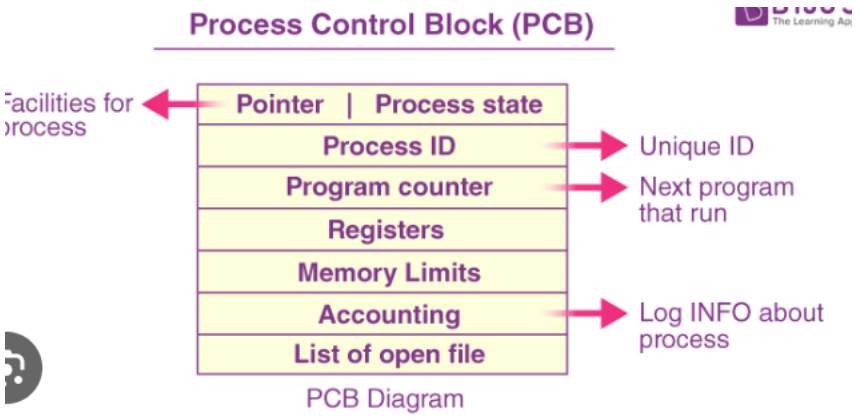
Note: When time is known ->sleep state

When time is unknown/indefinite ->pending state

6)Zombie:

Ctrlc -> kill a process

PCB



1 process has 1 process control block (structure)

PCB contains process related info

FORK SYSTEM CALL

Need for child process ?

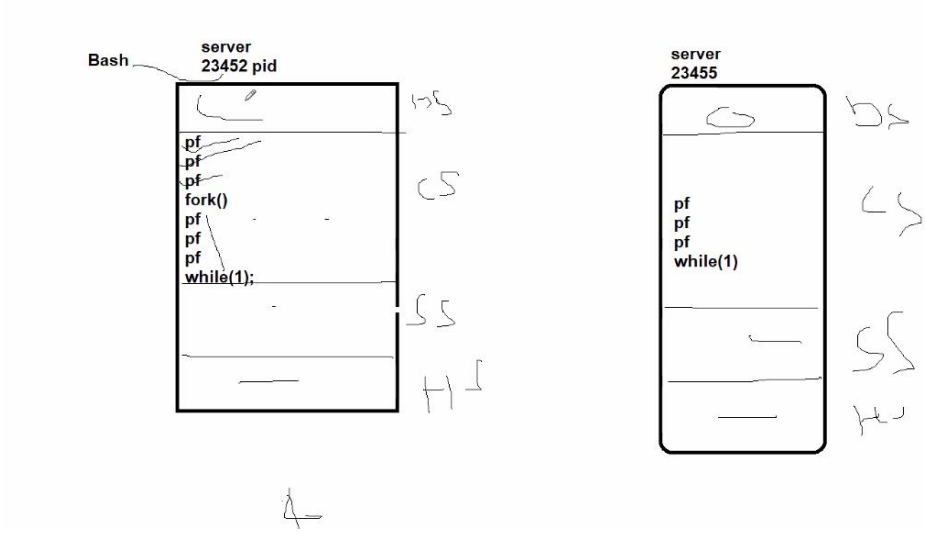
Paytm server

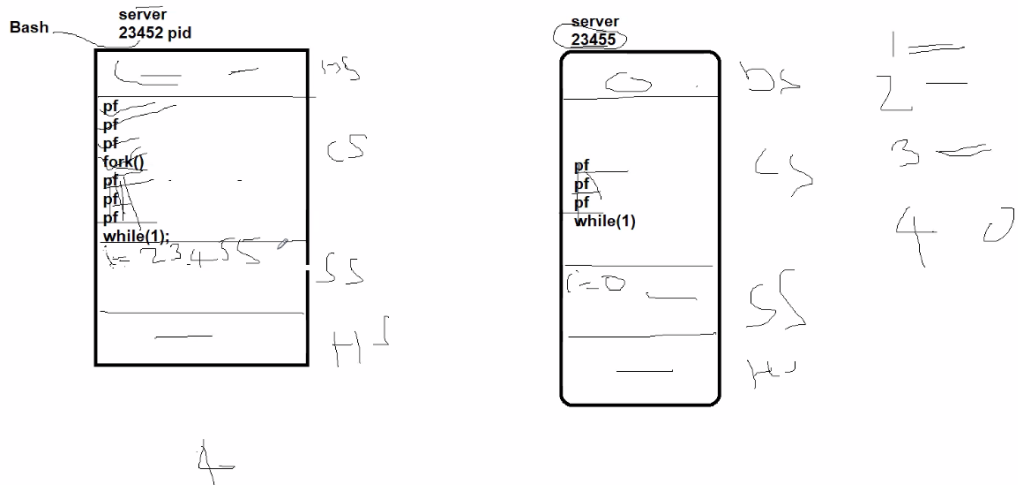
Every time client needs service

The parent process creates a new child , and that child is specific to client1

And so on ….

fork()






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


int main()
{
        int i;
        printf("Process Demo\n");
        printf("Process Management Unit\n");
        printf("Before fork\n");
        i=fork();
        printf("after fork\n");
        printf("end\n");
        printf("bye\n");
        while(1);
}
~
~
~
~
~
```

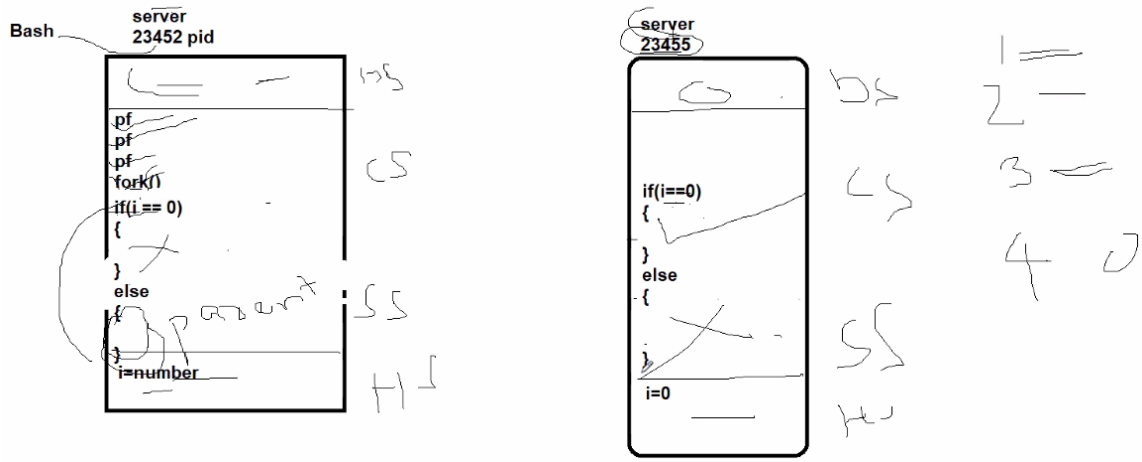
```c
#include<unistd.h>
#include<stdio.h>


int main()
{
        int i;
        printf("Process Demo\n");
        printf("Process Management Unit\n");
        printf("Before fork\n");
        i=fork();
        printf("after fork\n");
        printf("end\n");
        printf("bye\n");
        printf("i = %d\n",i);
        while(1);
}
~
~
~
~
~
~
~
```

```
prashanth@prashanth-VirtualBox:~/carl_zeiss$ ./server
Process Demo
Process Management Unit
Before fork
after fork
end
bye
i = 3107
after fork
end
bye
i = 0
```

RETURN VALUE

On success, the PID of the child process is returned in the parent, and 0 is returned in the child. On failure, -1 is returned in the parent, no child process is created, and errno is set appropriately.




```
int main()
{
        int i;
        printf("Process Demo\n");
        printf("Process Management Unit\n");
        printf("Before fork\n");
        i=fork();
        if(i==0)
        {
                printf("Child process\n");
                printf("In Child Process, Child Process ID = %d\n",getpid());
                printf("In Child Process, Parent Process ID = %d\n",getppid());
        }
        else
        {
                printf("In Parent Process\n");
                printf("In Parent process, Child Process Id = %d\n",i);
                printf("In Parent process, Parent Process Id = %d\n",getpid());


        }


        while(1);
}
```

```
 65598 ?        00:00:00 kworker/7:2-events
 65599 ?        00:00:00 kworker/u16:1
 65635 pts/0    00:00:31 f2
 65636 pts/0    00:00:31 f2
 65638 pts/1    00:00:00 ps
```

```
prodeep-200905384@prodeep-200905384:~/Documents/LinuxPractice$ gcc fork2.c -o f2
prodeep-200905384@prodeep-200905384:~/Documents/LinuxPractice$ ./f2
In parent process
In parent process,Parent Process pid:65635
In parent process, Child Process pid:65636
In child process
In Child Process , Child Process pid:65636
In Child Process,Parent Process pid:65635
```

ORPHAN

SYNCHRONIZATION

ZOMBIE

1)Creates a new child process with same name but different pid

2)Copies all data /stack/heap segment of parent to the child

**But for code segment :

Only instructions after fork is copied from parent to child

3)Return a value (child pid)to the parent process

4)Return a value(0) to child process

Note:ps -ef //shows all processes along with pid and ppid

```c
#include<stdio.h>


int main()
{
        int i;
        printf("Process Demo\n");
        printf("Process Management Unit\n");
        printf("Before fork\n");
        i=fork();
        if(i==0)
        {
                sleep(20);
                printf("Child process\n");
                printf("In Child Process, Child Process ID = %d\n",getpid());
                printf("In Child Process, Parent Process ID = %d\n",getppid());
        }
        else
        {
                printf("In Parent Process\n");
                printf("In Parent process, Child Process Id = %d\n",i);
                printf("In Parent process, Parent Process Id = %d\n",getpid());

        }


}
```

```
prashanth@prashanth-VirtualBox:~/carl_zeiss$ vi server.c
prashanth@prashanth-VirtualBox:~/carl_zeiss$ gcc server.c -o server
prashanth@prashanth-VirtualBox:~/carl_zeiss$ ./server
Process Demo
Process Management Unit
Before fork
In Parent Process
In Parent process, Child Process Id = 3173
In Parent process, Parent Process Id = 3172
prashanth@prashanth-VirtualBox:~/carl_zeiss$
prashanth@prashanth-VirtualBox:~/carl_zeiss$ Child process
In Child Process, Child Process ID = 3173
In Child Process, Parent Process ID = 1879

prashanth@prashanth-VirtualBox:~/carl_zeiss$
```

When any parent dies , all childs will become orphan

//when child wakes up , it sees its parent is dead

All orphan process adopted by init/systemd

Note: In ubuntu , orphan is adopted by local init/systemd

Other versions , the pid1 systemd

```
{
        printf("Child process\n");
        printf("In Child Process, Child Process ID = %d\n",getpid());
        printf("In Child Process, Parent Process ID = %d\n",getppid());
        for(int i=0;i<20;i++)
        {
                printf("child running\n");
                printf("In Child Process, Parent Process ID = %d\n",getppid());
                sleep(1);
        }
        printf("child completes\n");

}
else
{
        printf("In Parent Process\n");
        printf("In Parent process, Child Process Id = %d\n",i);
        printf("In Parent process, Parent Process Id = %d\n",getpid());

        wait(0);
        printf("Parent completes\n");


}
```

For sync and making sure parent doesnt finish exec before child

child->sleep(1);

parent->wait(0);

parent goes from running to pending state till child finishes jobs

CHILD WILL NEVER BECOME ORPHAN

wait:parent waits until child finishes its execution

Note:#include<sys/wait.h> required for wait()

```
int main()
{
        int i;
        printf("Process Demo\n");
        printf("Process Management Unit\n");
        printf("Before fork\n");
        i=fork();
        if(i==0)
        {
                printf("Child process\n");
                printf("In Child Process, Child Process ID = %d\n",getpid());
                printf("In Child Process, Parent Process ID = %d\n",getppid());
                printf("child completes\n");

        }
        else
        {
                sleep(40);
                printf("In Parent Process\n");
                printf("In Parent process, Child Process Id = %d\n",i);
                printf("In Parent process, Parent Process Id = %d\n",getpid());

                printf("Parent completes\n");


        }
```

Child is dead but parent is sleeping

NO ritual for child

child is a zombie process

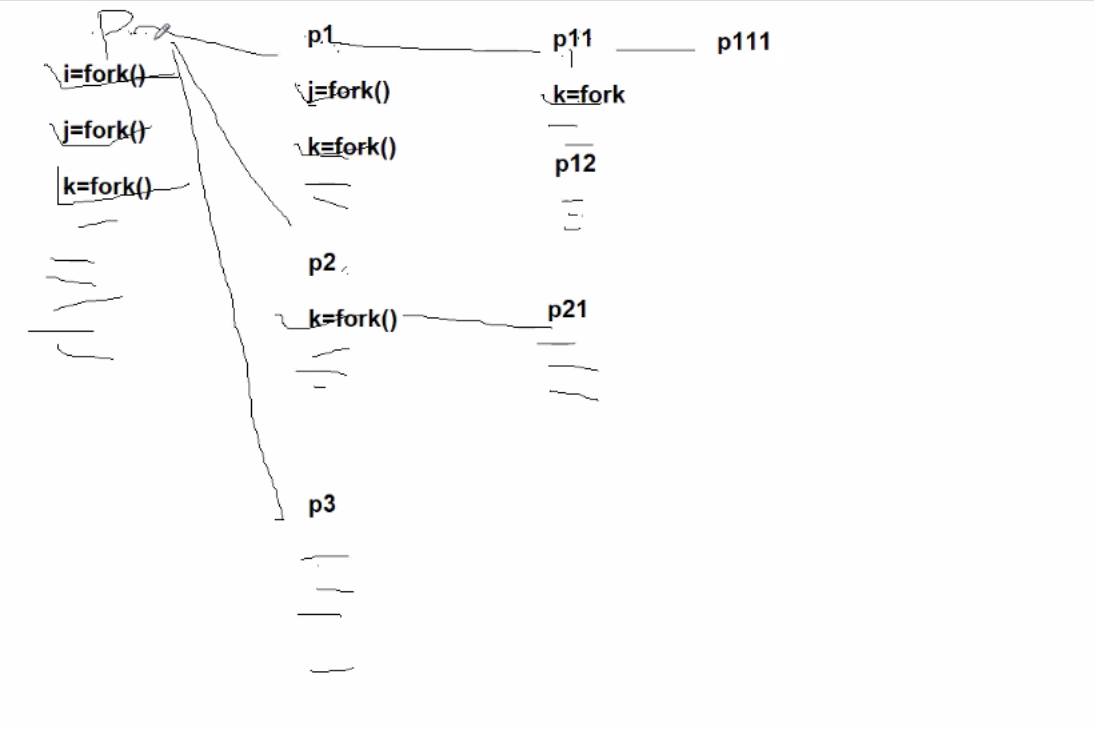
After 40s , status from zombie to exit status (parent frees the childs memory)

<defunct> =>zombie process

Note: No method to avoid zombie process

Like incase of orphan process

N consecutive fork



We will have 2^n -1 total new child processes

Including parent =>2^n processes

---

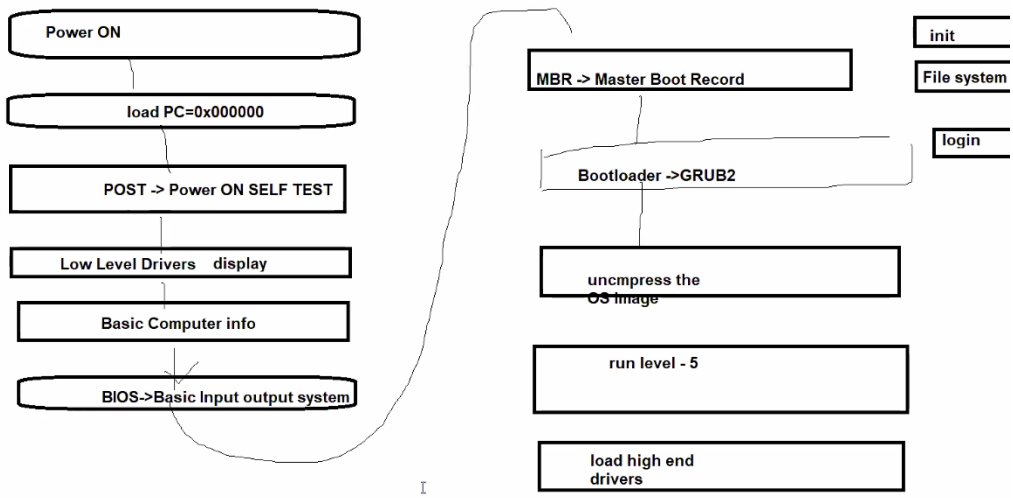
3 CHILD PROCESSES

```
i=fork();
if(i==0)
{


}
else
{
  j=fork();
  if(j ==0)
  {

 }
else
{
  k = fork()
}
```

BOOTING PROCESS STEPS



POWER ON SELF TEST=>

sends signal to all the devices

and checks if they are working

.That is why many lights glow

Low level drivers:

eg HP/DELL/ASUS

very dull and no Graphical enhancement

Basic comp info displayed

BIOS has basic drivers for keyboard

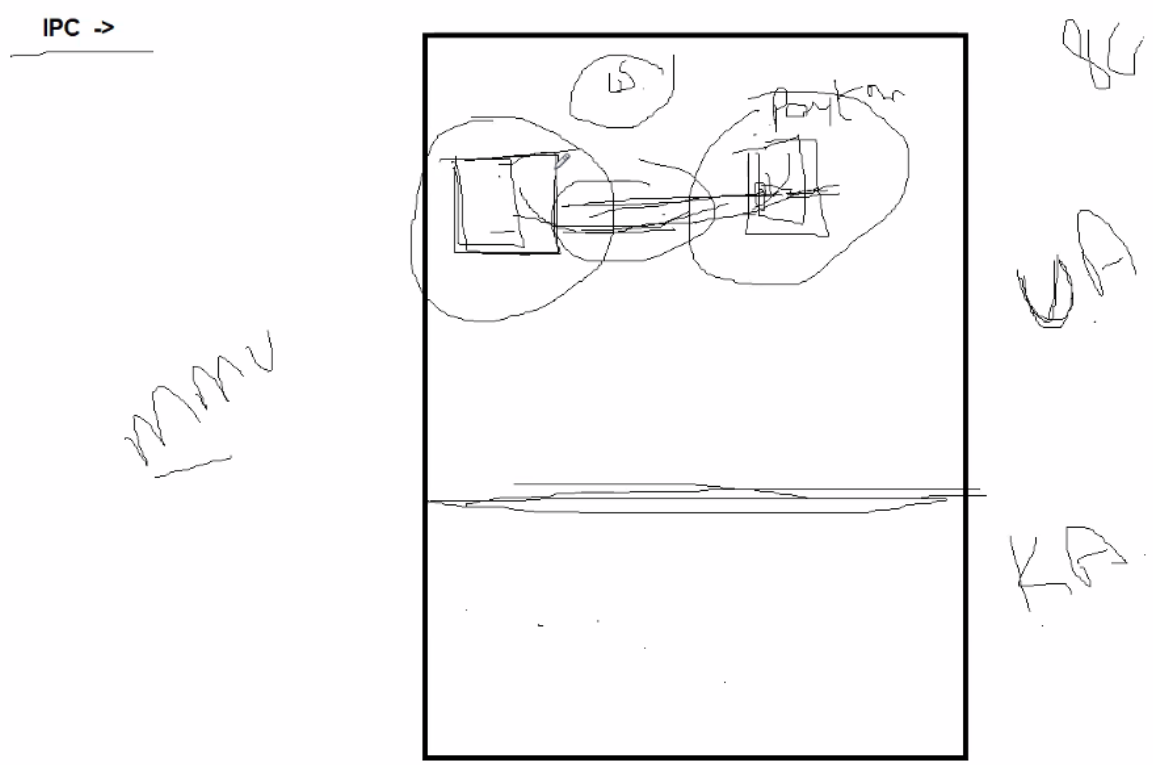
MBR holds the address of Windows OS/Linux Os(Bootloader)

GRUB :Takes into action if multiple os in single device

IPC

RAM:a)Kernel Area b)User Area

MMU DIVIDES RAM into 2 parts



MMU protects user process

It wont allow 1 process to use memory space of another

process.

Eg:Paytm ...enter mobile no

2nd process overwrites mobile no

This shouldnt be allowed

**Each Process has separate memory area

**Through pipes, 2 process can exchange data with each other

eg:copy text from whatsapp to gmail(2 diff apps)

Kernel Area :system calls like fork , create file

User Area :.cpp file,c file , games

Types of IPC

PIPES

FIFO

(above 2 primitive)

Message Queue

Shared Memory

Semaphores

(above 2 system V)

Socket

(BERKLEY SW DEVELOPMENT)

---

1)Pipe ->Only Related processes

(only bw parent & child)

Half duplex

(Only 1 operation at a time)

Unidirectional

For Full duplex:2 pipes

2 Descriptors /Ends

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

int main()
{
        int a,b;
        a=open("file1",O_RDONLY);
        b=open("file2",O_WRONLY);
        printf("a = %d\n",a);
        printf("b = %d\n",b);
        while(1);



}
~
~
~
~
```

```
a = 3
b = 4
```

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

int main()
{
        int a,b;
        int x[2];
        a=open("file1",O_RDONLY);
        b=open("file2",O_WRONLY);
        printf("a = %d\n",a);
        printf("b = %d\n",b);
        pipe(x);
        printf("x[0] = %d\n",x[0]);
        printf("x[1] = %d\n",x[1]);
        while(1);
}
~
~
~
~
```

```
prashanth@prashanth-VirtualBox:~/carl_zeiss$ ./demo
a = 3
b = 4
x[0] = 5
x[1] = 6
```

```
1140  1393  1555  1819  298   592  91   locks
prashanth@prashanth-VirtualBox:/proc$ c

prashanth@prashanth-VirtualBox:/proc$ cd 2058
prashanth@prashanth-VirtualBox:/proc/2058$ ls
arch_status      environ    mountinfo      personality   statm
attr             exe        mounts         projid_map    status
autogroup        fd         mountstats     root          syscall
auxv             fdinfo     net            sched         task
cgroup           gid_map    ns             schedstat     timers
clear_refs       io         numa_maps      sessionid     timerslack_ns
cmdline          limits     oom_adj        setgroups     uid_map
comm             loginuid   oom_score      smaps         wchan
coredump_filter  map_files  oom_score_adj  smaps_rollup
cpuset           maps       pagemap        stack
cwd              mem        patch_state    stat
prashanth@prashanth-VirtualBox:/proc/2058$ cd fd
prashanth@prashanth-VirtualBox:/proc/2058/fd$ ls
0  1  2  3  4  5  6
prashanth@prashanth-VirtualBox:/proc/2058/fd$ ls -l
total 0
lrwx------ 1 prashanth prashanth 64 Jan 22 14:56 0 -> /dev/pts/0
lrwx------ 1 prashanth prashanth 64 Jan 22 14:56 1 -> /dev/pts/0
lrwx------ 1 prashanth prashanth 64 Jan 22 14:56 2 -> /dev/pts/0
lr-x------ 1 prashanth prashanth 64 Jan 22 14:56 3 -> /home/prashanth/carl_zeiss/file1
l-wx------ 1 prashanth prashanth 64 Jan 22 14:56 4 -> /home/prashanth/carl_zeiss/file2
lr-x------ 1 prashanth prashanth 64 Jan 22 14:56 5 -> 'pipe:[35661]'
l-wx------ 1 prashanth prashanth 64 Jan 22 14:56 6 -> 'pipe:[35661]'
prashanth@prashanth-VirtualBox:/proc/2058/fd$
```

FILE DESCRIPTORS

```
int a,b;
a=open("File1",O_RDONLY);
b=open("File2",O_WRONLY);
```

To check file descriptors

cd /proc

cd 1994

cd fd

ls

PIPE DESCRIPTORS

It has 2 ends

so pipe() takes an array of 2 size

int x[2];

pipe(x);

x[0]->read end

x[1]->write end

dis of pipes:cannot communicate bw 2 unrelated processes

Pipe -> Related process
Half duplex
2 pipe

pipe

0	STDIN
1	STDOUT
2	STDERR
3	file1
4	file2

a = open("file1")
3
b = open("file2")
4

descriptor is always the lowest available number

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

int main()
{
        int i;
        int x[2];
        pipe(x);
        i=fork();
        if(i==0)
        {
                printf("chile process\n");
                write(x[1],"HELLO BANGLORE",15);
                printf("child writes\n");

        }
        else
        {
                char buf[20];
                printf("Parent Process\n");
                read(x[0],buf,15);
                printf("read data from pipe = %s\n",buf);
        }

}
```

```
prashanth@prashanth-VirtualBox:~/carl_zeiss$ vi demo.c
prashanth@prashanth-VirtualBox:~/carl_zeiss$ gcc demo.c -o demo
prashanth@prashanth-VirtualBox:~/carl_zeiss$ ./demo
Parent Process
chile process
read data from pipe = HELLO BANGLORE
prashanth@prashanth-VirtualBox:~/carl_zeiss$ child writes

prashanth@prashanth-VirtualBox:~/carl_zeiss$
```

2 way through pipes

```c
int main()
{
        int i;
        int x[2],y[2];
        pipe(x);
        pipe(y);
        i=fork();
        if(i==0)
        {
                char buf[20];
                printf("child process\n");
                read(x[0],buf,16);
                printf("read data from parent = %s\n",buf);
                write(y[1],"HELLO MY PARENT",16);

        }
        else
        {
                char data[20];
                printf("Parent Process\n");
                write(x[1],"HELLO MY CHILD",16);
                read(y[0],data,16);
                printf("read data from child = %s\n",data);
                wait(0);
        }
```

FIFO

FIFO pipes->named pipes

Also half duplex

1st terminal :mkfifo f1

cat f1 //read

2nd terminal:

cat >f1 //write

Happens only when both parties are ready to read and write

---

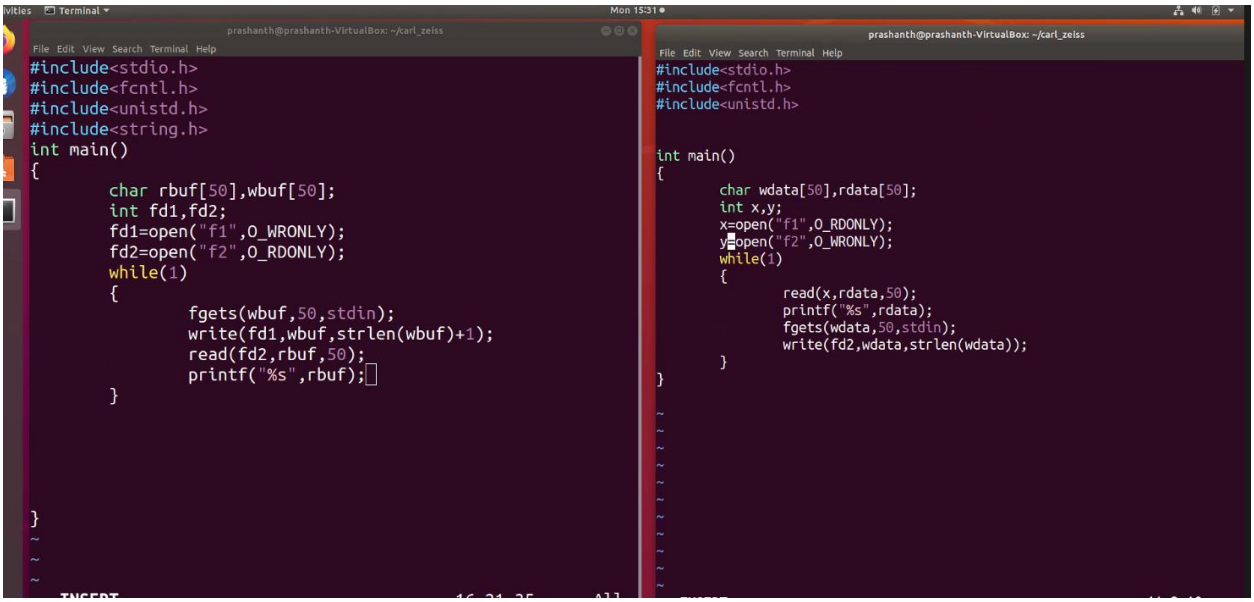
2 Programs for FIFO

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


int main()
{
        char data[20];
        int x;
        x=open("f1",O_RDONLY);
        read(x,data,20);
        printf("read data from fifo = %s\n",data);
}
~
~
~
~
~
~
```

```
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
int main()
{
        int fd1;
        fd1=open("f1",O_WRONLY);
        write(fd1,"HELLO OTHER PROCESS",20);
        printf("process1 writes\n");
}
~
~
~
~
```

2 WAY COMMUNIATION



```c
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include<string.h>
int main()
{
        char rbuf[50],wbuf[50];
        int fd1,fd2;
        fd1=open("f1",O_WRONLY);
        fd2=open("f2",O_RDONLY);
        while(1)
        {
                fgets(wbuf,50,stdin);
                write(fd1,wbuf,strlen(wbuf)+1);
                read(fd2,rbuf,50);
                printf("%s",rbuf);
        }





}
~
~
```

```
prashanth@prashanth-VirtualBox:~/carl_zeiss$ ./p1
Hello
Hi
How are u?
I am gud
```

```
prashanth@prashanth-VirtualBox:~/carl_zeiss$ ./p2
Hello
Hi
How are u?
I am gud
```