⊛ PS Command :-

→ PS command will give us list of process in a current terminal.

PID → process ID.

→ The process manager does not recognize the process not with the name of the process but with the id (number) i.e. called PID. (Process ID).

⊛ fg command :-

→ fg is a command to bring the background process to the foreground.

/V15be3/linux/Pm $ fg

./a.out

Now /V15be3/linux/Pm/ $ PS

PID	TTY	TIME	CMD
1422	pts/0	00:00:00	Bash
1637	pts/0	00:00:00	PS

} a.out is terminated

1) cc filename.c
2) ./a.out &
3) PS
4) fg
5) ./a.out is terminate.

→ Pid-t → default upper limit → 32767 (32-bit) but it can be reset via value of Pid_max in Linux specific

/proc/sys/kernel/pid_max.

64-bit → 2^{22} (4 million)

⊛ Synopsis :-

```
# include <sys/types.h>
# include <unistd.h>

# define Pid-t unsigned short int

Pid-t getpid (void);
Pid-t getppid (void);
```

→ returns the PID of calling function

→ returns the PID of parent of the calling function

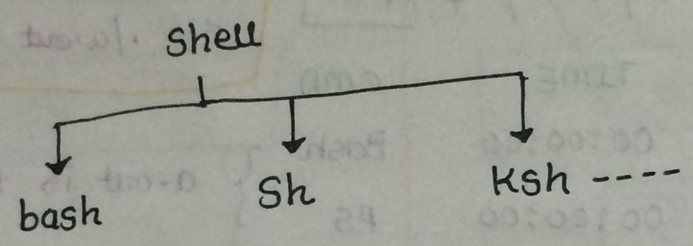
```
main()
{
    printf(" Hello World  pid = %d  ppid = %d \n",
                                getpid(), getppid());
}
```

Output :- If you load the same program into the RAM for the execution again then also same id is not given the process manager gives different PID.

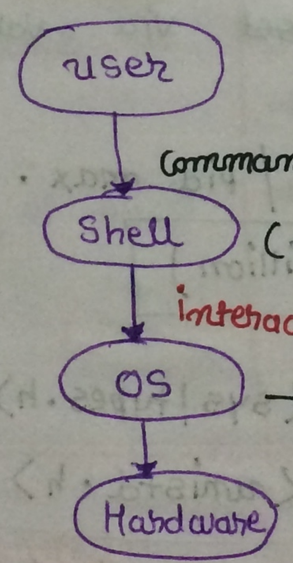
→ Terminal starts then bash shell starts executing by algorithm.

⊛ What is bash ?

Bash is nothing but shell.



→ Shell is a command interpreter.
(mediator betn user and OS).



→ in RAM will create a child process bash is parent of it.

→ Shell is an application not a service.

PID <= 32767 (max value)

to find if

vi | proc | sys | kernel | pid_max

= 32768

b'coz $\boxed{0 \text{ to } 32767}$

⊛ PID

→ The Linux Kernel limits process ID to being less than or equal to 32767.

→ After Reaching 32767 it Reset to 300. not zero.

→ Many lower numbered process ID in permenent use by System process and daemones.

→ The process id of init is 1 (one)

The process id of sedular is 0 (zero)

⊛ Disadvantage of Shell :-

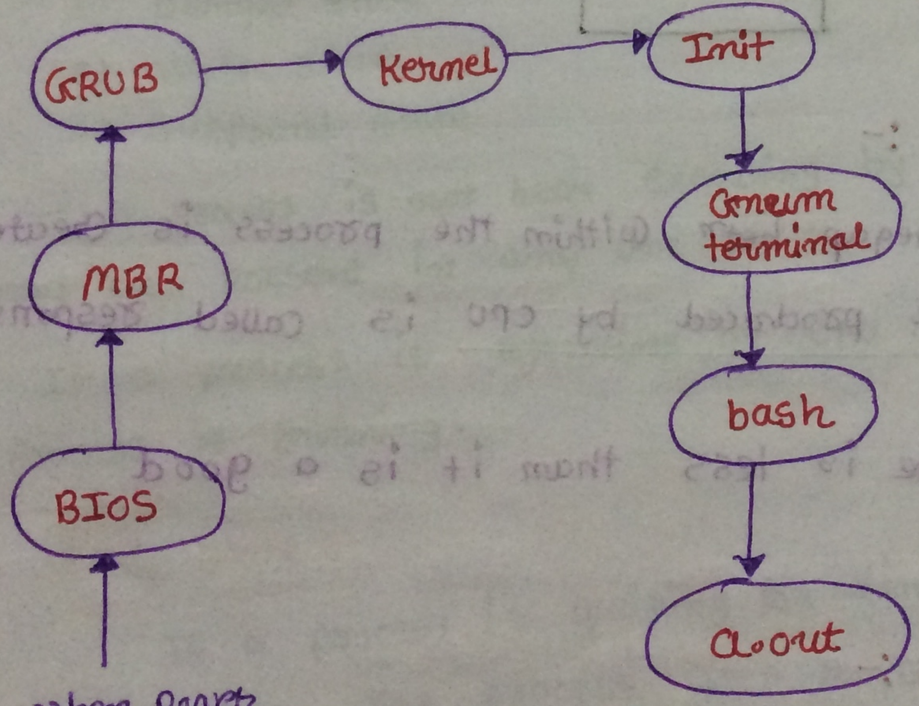
→ up, down arrow, tab not allowed hence not user friendly.

ps -l → long list in current terminal.

ps -e → list of process throughout the system.

ps -el → long list of process throughout the system in current terminal.

exit → from sh to bash.



./a.out &

[1] 1910

Job Id

process id consigned by process manager.

⇒ any command is an executable file, we also create but command e.g. ./my-copy.

(*) When shell will assign Job ID?

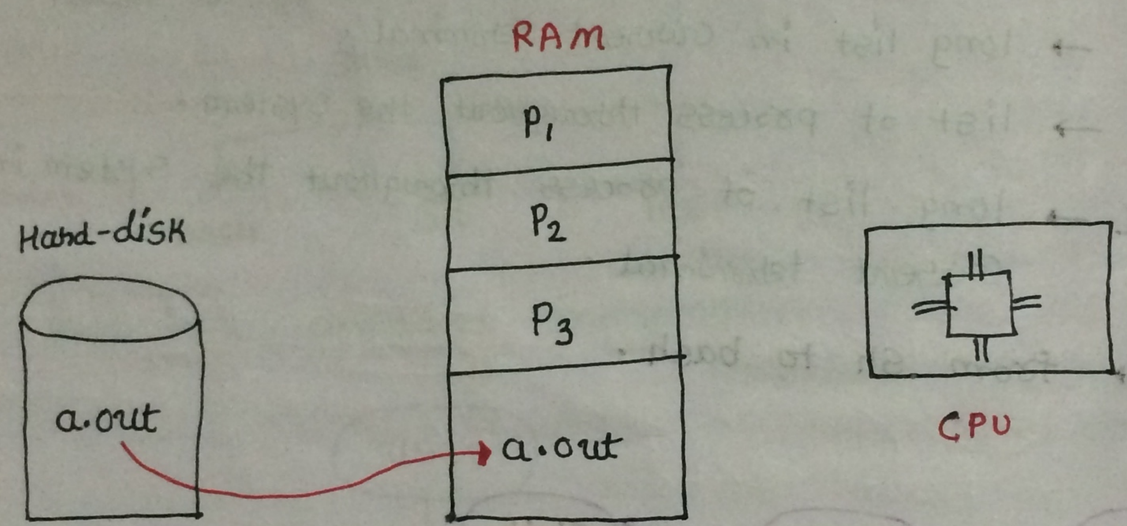
→ If we run the process in background, then shell will assign Job ID to process.

fg → bring out the LID program to the foreground from background first.

LIFO

fg (Job id) :-

Then the particular process with that particular job will be moved to the foreground from the background.



① Response time :-

The timegap betn within the process is created and first response produced by CPU is called response time.

→ If response time is less than it is a good environment.

② Starvation time :-

A process in its life cycle how much of time it is starvated without executing by CPU is called starvation time.

→ If starvation time is less than it is good environment

③ **Turnaround time (Total time) :-**

→ The time of the creation to the time of the completion of process is called turnaround (total) time.

→ If turnaround time is less, then good environment.

④ **Throughput time :-**

→ No. of processes completed per unit of time is called throughput time.

→ If throughput is more, then good environment.

⇒ Program execution will not only depended on the process manager but also depends upon the hardware support.

Imp

8 bit µp
16 bit µp
32 bit µp
} different executing speed.

State of a process :-

1) Ready state
2) Delayed State
3) Wait State
4) Susppend State

⇒ If a process is not been executing by the CPU, the process may be present in any one of the above state.

⇒ If a process is executing by CPU, then the state of a process is running.

① **~~Ready~~ Wait state :-**

If a process is waiting for (intrupt) external event to finish, then the process goes to wait state.

```
scanf("%d", &i);

main()
{
    int num;
    Printf("pid = %d", getpid());
```

```
printf("Enter number---(n");
scanf("%d", &num);
printf("num = %d(n", num);
while(1);
}
```

(Control switching between places)

Ready
↓
Run
↓
Wait
↓
Ready
↓
Run

in another terminal

```
ps -e | grep pts/0
```

Now if we want to terminate one process running in a terminal through other terminal then use -

```
kill -signum pid
```

signal number

eg:- kill -9 1588

→ forcefully killing

→ kill is useful for sending some signal to the process.

→ If a process goes to intentionally delay, then the process is treated as delayed state process.

→ To move the process to delayed state use sleep() function.

→ provide ...

② Ready state :-

→ If a process is ready for execution but not executed then process goes to ready state.

③ Delayed state :-

→ If a process goes to the intentionally delay then that process is treated as delayed state.

→ To move the process to the delay state use sleep() function.

```
main()
{
    print
    Slee
    Print
    while
}
```

Ready

→ If a
Process

in an
ps

1390
1660

using

1660
to terma
Kill
to Kill
Kill

Stop

```
main()
{
    printf ("pid = %d\n", getpid());
    sleep (10);
    Printf ("after sleep\n");
    while (1);
}
```

$ bg
⇒ recent background process display

$ Jobs
⇒ detail for the process (stop or running)

Ready → Run → delayed → Ready → Run

⇒ If a process is suspended because of some signal that process goes to suspended state.

in another terminal

```
ps -e/grep pts/0

1390  pts/0   00:00:00   bash
1660  pts/0   00:00:00   a.out

using Kill -19 1660
        → suspended

1660  pts/0   00:00:100  a.out
```

to resume
Kill -18 1660

to kill it
Kill -9 1660

$ bg 5
⇒ 5 process is stop then using this command Process 5 is running.

Kill all the process

Killall a.out
a.out related Process is Killed.

⇒ when we do Ctrl+2, then it will go into suspended state and it will go into background.

Random Number Generation :-

Synopsis :-

```
#include <stdlib.h>
int rand(void);
int rand_r (unsigned int *seed);
void srand (unsigned int seed);
```

⇒ rand is useful for generating the random number in between 0 to RAND-MAX.

⇒ Srand function is useful giving a seed to the rand fun so that rand() will generat another sequence of random number.

define RAND-MAX 2147483647
(Same as MAX-INT)

define Exit-SUCCESS 1

define EXIT-FAILURE 0

```
# include <stdio.h>
main()
{
    int a[5], i;
    for (i=0; i<5; i++)
      a[i] = rand();
    for (i=0; i<5; i++)
      printf("%d ", a[i]);
    Printf("\n");
}
```

output :-

10 17 24 7 48

again ./a.out

again ./a.out

every time number is fixe because seed is not changed.

```
main()
{
    int a[5];
    // Srand (time(0));
    // srand (2);
    // Srand (getpid());
}
```

copy above block

op :-

Every time different sequence. but in 1 sec if we press a.out fast it gives same number.

To overcome

// srand (getpid());

Put this to get different number every second.

a[i] = rand

a[i] = rand

a[i] = rand

① wap to ge
+ 50.

⇒ X = (

0

-50

-5

② wap to
-50 to

⇒ X =

print

③ wap to
+0.25

0

25

X =

Printf

New

New process