$ sudo su ⟶ In a root mode

Password :

(*) Vi P1.c :-

```
#include <stdio.h>
main()
{
  printf("Pid = %d\n", getpid());
  while(1);
}
```

cc P1.c

o/a.out &

Then PS -l ↵

PRT OF a.out = 80

NI is range from 60 to 99

Change the NI Value :-

(1) $ man nice :-

nice - run a program with modified scheduling priority.

Synopsis :-

nice [OPTION] [COMMAND [ARG[V]]----]

→ niceness Range from -20 to 19 (least favorable).

# nice -5 ./a.out &

# nice --5 ./a.out &

→ renice is a command to change a priority of the existing process (running process).

renice -5 1670

O/P :- 1670 : old priority = 5 new priority = -5

Process manager divided in two part :-

- CPU bounded.
- I/O bounded.

① CPU bounded :-

a process in it life cycle if it is pending more time for execute those process are called CPU bounded process.

e.g while (1).

② I/O bounded :-

a process in it life cycle it is pending more time to complete external, event to finish those processes are called I/O bounded processes.

e.g Scanf ().

⇒ I/O bounded process is smarter than CPU bounded Process.

---***Signal***---

$ mkdir signal

⇒ Signal is disturbance to normal program execution.

⇒ Signal are also called as software interrupts.

⇒ when a process execution normally because of same reason if execution of process get disturb then we have to say process got a signal.

```
# include <stdio.h>
main()
{
    Printf ("Hello World Pid = %d\n", getpid());
    while(1);
}
```

Output :- Hello world --- Pid = 1711
^C
disturb the process.

* $ man

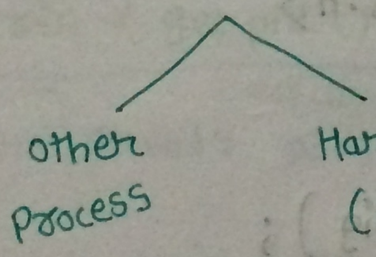
* $ Kill -l :-

→ You will get all the Signals numbers and Names.

| Signal Nuber | Signal Name :- | |
|---|---|---|
| 1 | SIGHUP | |
| 2 | SIGINT | (ctrl + c) |
| 3 | SIGQUIT | (ctrl + \) |
| 8 | SIGFPE | (floating point exception) |
| 9 | SIGKILL | (forcefully Stop) |
| 10 | SIGUSR1 | |
| 11 | SIGSEGV | (Segmentation fault) |
| 12 | SIGUSR2 | |
| 13 | SIGPIPE | |
| 14 | SIGALRM | (Signal Alram) |
| 17 | SIGCHLD | (Signal child) |
| 18 | SIGCONT | (Signal continue) |
| 19 | SIGSTOP | (Signal STOP) |

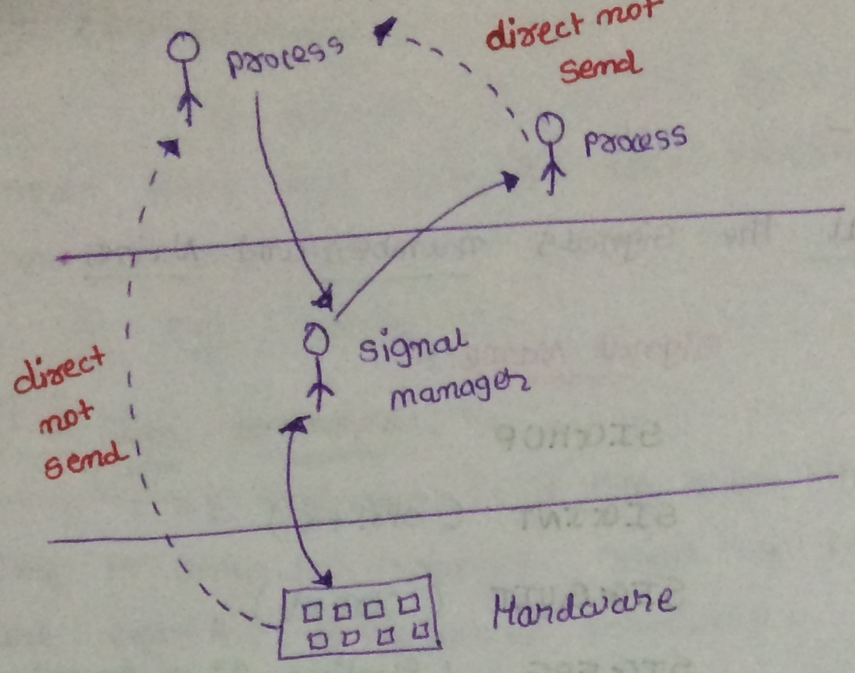
→ Process is receive the Signals.

→ Signal generate using two method




→ Otherprocess or Hardware not Send a directly

→ Send Via Signal Manager.




⇒ life of the signal start when somebody generate the signal.

⇒ life of the signal end, signal manager deliver the signal to the particular process.

⇒ process is terminate PCB is also terminate.

True :- How one process can send a signal to another process ?

⇒ using Kill Command....

## $ man Kill :-

Kill → send signal to a process.

Synopsis :-

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

int kill (Pid_t Pid, int sig);
```

## (*) Vi my_Kill.c :-

```
#include <stdio.h>
main (int argc, char **argV)
{
```

```
if (argc != 3)
{
    Printf (" Usage : ./my_kill  Pid  signal\n");
    return ;
}
Kill ( atoi (argV[1] ), atoi (argV[2]) ) ;
perror (" Kill") ;
}
```

Output :- ./my_Kill 1822 11

O/P :- Segmentation Fault.

./my_Kill 1876 8

O/P :- Floating point exception.

## $ man raise :-

raise :- send a signal to the caller.

Synopsis :- # include <Signal.h>

```
int raise (int sig) ;
```

⇒ The raise() function sends a signal to the calling process or theard.

```
Kill (getpid(), sig) ;
```

⇒ multithreads program it is act as

```
Pthread_Kill ( Pthread_self(), sig ) ;
```

⇒ raise return 0 = success

return non zero = failure

## $ man Pause :-

Pause → wait for signal

Synopsis :- #include <unistd.h>
int Pause (Void);

⇒ Pause() causes the calling process (or thread) to sleep until a signal is delivered that either terminates the process or causes the invocation of a signal catching function.

```
# include <stdio.h>
main()
{
    Printf ("Hello World ---- %d\n", getpid());

    Pause();
    Printf ("Bye ---\n");
}
```

O/P :- Hello World ---- 1845

when we Kill -9 1845 Then print Killed

Bye is not printed.

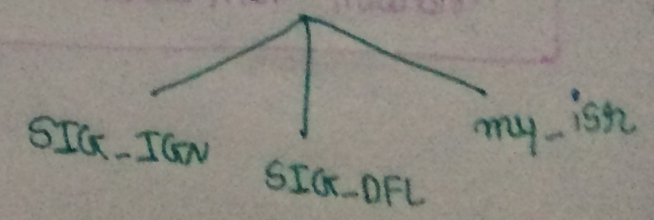
## $ man signal :-

Signal :- ANSI C signal handling

Synopsis :- #include <signal.h>

typedef Void (*sighandler_t)(int);

Sighandler_t signal (int signum, Sighandler_t handler);



* The behaviour of signal() varies across unix versions and has also varied historically across different version of Linux.

```
# include <stdio.h>
Void my_isr (int n)
{
    Printf (" In isr ----\n") ;
}
main()
{
    Printf (" Hello world ----%d\n", getpid());
    signal (2, my_isr);
    Printf (" Bye ----\n");
    while(1);
}
```

Signal table

| Signum | Action |
|---|---|
| 1 | SIG_DFL |
| 2 | my_isr |
| 3 | SIG_DFL |
| 4 | SIG_DFL |
| 5 | SIG_DFL |
| 6 | |
| 7 | |

(*)

```
#include <stdio.h>
# include <signal.h>
Void my_isr (int n)
{
    Printf (" In isr ----%d\n", n) ;
    Signal (" 2, SIG_DFL) ;
}
main()
{
```

```
    Printf (" Hello World ---- %d\n", getpid());
    Signal (2, my_isr);
    Printf (" Bye ---\n");
    while (1);
}
```

Output :-

```
Hello World ---- 1848
Bye ----
^C In isr ---- 2
^C
terminate process
```

⇒ Signal function not generate signal

* Kill command is send a signal to other and also itself
* Raise command is send a signal only itself.

⇒ Signal function is not block the signal.

⇒ signal function is used for the modified the signal table.

Signal ( _____ , _____ )

signal number — Signal table data

(*) Program :-

```
# include <stdio.h>
# include <signal.h>
Void my_isr (int n)
{
    Printf (" In isr --- %d\n", n);
}
```

Output

```
Hello World ---- 1818
^C In ISR ----
Bye ----
```