```
main()
{
    Printf("Hello World ---- %d\n", getpid());
    Signal (2, my_ish);
    Pause();
    Printf("Bye ----\n");
}
```

→ Pause is (usefull) used-full for the Blocking the process to sleep until the process receive the Signal.

$ man alarm :-

alarm → set an alarm Clock for delivering of a Signal.

Synopsis

```
# include <unistd.h>

unsigned int alarm (unsigned int Second);
```

→ alarm() arranges for a SIGALRM Signal to be delivered to the Calling process in Seconds-seconds.

→ If Seconds is 0 then no new alarm Schedules.

→ If any previous event is alarm() then old alarm() is cancelled.

(*) program

```
# include <Stdio.h>
main()
{
  Printf("Hello ---\n");
  alarm (10);
  Printf("Hai ---\n");
  while(1);
}
```

Output

Hello ---

Hai ---

Alarm Clock

terminate process.

① Alram function will not block the process.

② Alarm function will not generate the alarm. It set the alarm.

③ defaruit action is terminate the process.

④ for a perticular it is not possible to set multiple alarms at a time.
(last alarm is display).

```
# include <stdio.h>
main()
{
    Printf("Hello---\n");
    alarm(10);
    alarm(2);
    Printf("Hai---\n");
    while(1);
}
```

Output :-

Hello---
Hai----
(after 2 second)
terminate the process.

⊛ Program :-

```
# include <signal.h>
# include <stdio.h>
Void my_isr(int n)
{
    Printf("In isr----%d\n",n);
}
main()
{
    Signal(SIGALRM, my_isr);
    Printf("Hello----\n");
    alarm(10);
    Printf("Hai---\n");
    while(1);
}
```

O/P :-

Hello----
Hai----
In isr----14
^C terminate the process.

Program :-

```
# include <stdio.h>
# include <signal.h>
Void my-ish (int n)
{
    Printf (" In ish ----%d\n", n);
    alarm (2);
}
main ()
{
    Signal (SIGALRM, my-ish);
    Printf (" Hello ----\n");
    alarm (10);
    Printf (" Hai ---\n");
    While (1);
}
```

O/P :-

Hello ---
Hai ----
(after 10 second)
in ish ----14
(every 2 second)
in ish ----14
in ish ----14
in ish ----14
⋮ ⋮

Program :-

```
#include <stdio.h>
# include <signal.h>
Void my-ish (int n)
{
    Printf (" In ish ----%d\n", n);
    alarm (2);
    signal (14, SIG_DFL);
}
main()
{
    signal (SIGALRM, my-ish);
    Printf (" Hello ---\n");
    alarm (10);
    Printf (" Hai ----\n");
    while (1);
}
```

Output :-

Hello ---
Hai ---
(after 10 second)
In ish ----14
(after 2 second)
terminate process

⊛ RETURN VALUE OF ALARM :-

⇒ alarm() return the number of seconds remaining until any previously scheduled alarm was due to be delivered, or zero if there was no previously scheduled alarm.

⊛ program :-

```
#include <stdio.h>
# include <signal.h>
main()
{
    int ret;
    Printf ("hello---\n");
    ret = alarm (10);
    Printf (" 1) ret = %d\n", ret);
    ret = alarm (3);
    Printf (" a) ret = %d\n", ret);
    while (1);
}
```

Output :-

hello ---
hai ---
1) ret = 0
a) ret = 10
(After 3 second)
AlARM Clock
terminate process.

⊛ Program :-

```
# include <stdio.h>
# include <signal.h>
main()
{
    int ret;
    Printf ("hello---\n");
    ret = alarm (10);
    Printf (" 1) ret = %d\n", ret);
    Sleep (3);
    ret = alarm (3);
```

Output :-

hello---
1) ret = 0
(after 3 Second)
a) ret = 7
Alarm Clock
(terminate process).

```
printf (" a) ret = %d\n", ret);
while (1);
}
```

Program 1 Disable (ctrl+c) and (ctrl+\) in a process for 10 second.

(Note :- within a 10 second ctrl+c and ctrl+\ are not work, after 10 second its working).

```
#include <stdio.h>
#include <signal.h>
Void my_isr (int);
main()
{
    Signal ( SIGINT, SIG_IGN);
    Signal ( SIGQUIT, SIG_IGN);
    Signal ( SIGALRM, my_isr);
    alarm(10);
    while (1);
}
Void my_isr (int n)
{
    Printf (" in isr----\n");
    Signal ( SIGINT, SIG_DFL);
    Signal ( SIGQUIT, SIG_DFL);
}
```

Output :-

(Start the process after 10 second)

^C } not work
^\

(after 10 second)

in isr----

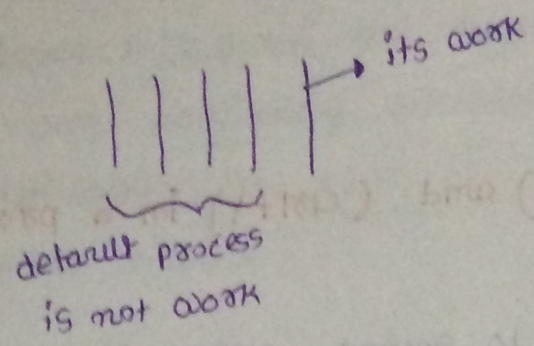
^C

terminate process.

Program 2 Disable SIG_INT and SIG_QUIT in your process restore SIG_INT to SIG_DFL when it occurs 4th time and restore SIG_QUIT to Default it occurs again.

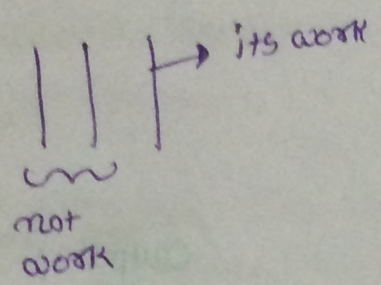
Note :-

SIG_INT Process



restore 4th time
means
1) restore in 4th time
2) but execute 5th time

SIG_QUIT Process



restore occurs again
means restore at 2nd
time and execute
at 3rd time.

```
# include <stdio.h>
# include <signal.h>
Void my_ish (int) ;
main()
{
    Signal (SIGINT, ~~SIG_IGN~~ my_ish)
    signal (SIGQUIT, ~~SIG_IGN~~ my_ish)
    while (1) ;
}
Void my_ish (int n) ;
{
    Static C=0 , C1=0 ;
    if (n==2)
        C++ ;
    else if (n==3)
        C1++ ;
    if (C==4)
        Signal (2, SIG_DFL) ;
    if (C==2)
        Signal (3, SIG_DFL) ;
}
```

th time

th time

5th time

ccurs again
re at 2nd
execute
time.

program 3 Observe that when terminal terminate than the process executing under it receive SIGHUP signal.

Note :- SIGINT signal generate when press ctrl+c
SIGQUIT signal generate when press ctrl+\

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

output :-
other term. = ./a.out
main term.

ps -e
pid is present but some ?

✱ daemon Process :-

1) Switch on the system than its runs.
2) Its ignore some user signal.

→ It is long-lived. often a daemon is created at system startup and runs until the system is shutdown.

→ It runs in the background and no controlling terminal the lack of a controlling terminal ensures that kernal never automatically generate any Job control.

Cron :- executes command at a scheduler time.

ssnd :- which permits logins from remort hosts using a source communication protocol.

httpd :- The HTTP Server daemon, which serve web pages.

Program 4 — observe the child becomes a zombie, Parent signal with a SIGCHLD (17).

```
#include <stdio.h>
#include <signal.h>
Void my_isr(int n)
{
    Printf(" in isr ---\n");
}
main()
{
    if (fork()==0)
    {
        Sleep(5);
    }
    else
    {
        Signal (17, my_isr);
        while (1);
    }
}
```

Output :-

(Sleep after 5 sec)

in isr ----

Child is in a Zombie State.

2nd method :-

```
#include <stdio.h>
#include <signal.h>
Void my_isr (int n)
{
    Printf(" in isr ---\n");
    Wait(0);
}
main()
{
    if (fork()==0)
        Sleep(5);
    else
    {
        Signal (17, my_isr);
        while(1);
    }
}
```

Output

Child is not in a Zombie State