**2.1** Write application or program to use alarm and signal system callssuch that, it will read input from user within mentioned time (say 10 seconds) ,otherwise terminate by printing message.

**Objectives:**

1. To learn about IPC through signal.
2. To know the process management of Unix/Linux OS
3. Use of system call to write effective application programs.

**Theory:**

1) Alarm:

Name:

alarm - set an alarm clock for delivery of a signal

Syntax:

#include <unistd.h>

unsigned int alarm(unsigned int *seconds*);

Description:

alarm() arranges for a SIGALRM signal to be delivered to the process in *seconds* seconds.

If *seconds* is zero, no new alarm() is scheduled.

In any event any previously set alarm() is cancelled.

Return Value:

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

Notes:

alarm() and setitimer() share the same timer; calls to one will interfere with use of the other.

sleep() may be implemented using SIGALRM; mixing calls to alarm() and sleep() is a bad idea.

Scheduling delays can, as ever, cause the execution of the process to be delayed by an arbitrary amount of time.

2) Signal:

Name:

signal - ANSI C signal handling

Syntax:

#include <signal.h>

typedef void (\*sighandler\_t)(int);

sighandler\_t signal(int *signum*, sighandler\_t *handler*);

Description:

The behavior of signal() varies across UNIX versions, and has also varied historically across different versions of Linux. Avoid its use: use [sigaction(2)](http://man7.org/linux/man-pages/man2/sigaction.2.html) instead. See *Portability* below.

signal() sets the disposition of the signal *signum* to *handler*, which is either SIG\_IGN, SIG\_DFL, or the address of a programmer-defined function (a "signal handler"). If the signal *signum* is delivered to the process, then one of the following happens:

If the disposition is set to SIG\_IGN, then the signal is ignored.

If the disposition is set to SIG\_DFL, then the default action associated with the signal occurs.

If the disposition is set to a function, then first either the disposition is reset to SIG\_DFL, or the signal is blocked (see *Portability* below), and then *handler* is called with argument *signum*. If invocation of the handler caused the signal to be blocked, then the signal is unblocked upon return from the handler. The signals SIGKILL and SIGSTOP cannot be caught or ignored.

Return Value:

signal() returns the previous value of the signal handler, or SIG\_ERR on error. In the event of an error, [*errno*](http://man7.org/linux/man-pages/man3/errno.3.html) is set to indicate the cause.

**Data Dictionary:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr Number | Variable/Function | Datatype | Use |
| 1 | alarmhandle | void | Used to handle alarm. |
|  |  |  |  |
| 2 | A | int | Input. |
|  |  |  |  |

**Program:**

#include<signal.h>

#include<stdio.h>

#include<unistd.h>

#include<stdbool.h>

#include<stdlib.h>

bool flag=false;

void alarmhandle(int sig)

{

printf("Input time expired\n");

exit(1);

}

int main()

{

int a=0;

printf("Input now in 10 seconds\n");

sleep(1);

alarm(10);

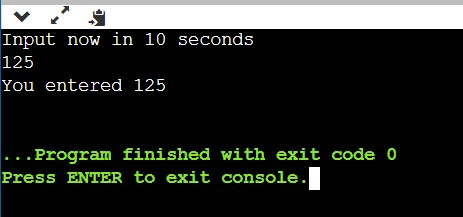
signal(SIGALRM,alarmhandle);

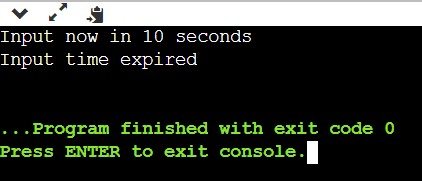
scanf("%d",&a);

printf("You entered %d\n",a);

}

**Output:**





**Conclusion:**

alarm() signal can be used to raise alarm after particular time period. Signal() system call is evoked by alarm() which is further processed by signal handler.

**References:**

*www.tutorialspoint.com/unix\_system\_calls*