Subject:- Unix Operating System

System Lab Class :- TYIT

Name **PRN**

Aditi Sudhir Ghate **2019BTEIT00044**

Assignment No - 2a

Title-Write application or program to use alarm and signal system calls such 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-

alarm()

Syntax-

```
#include <unistd.h>
unsigned int alarm(unsigned int seconds);
```

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

signal()

Syntax-

```
#include <signal.h>
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

The signal() system call installs a new signal handler for the signal with number signum. The signal handler is set to sighandler which may be a user specified function, or either SIG_IGN or SIG_DFL.

Upon arrival of a signal with number signum the following happens. If the corresponding handler is set to SIG_IGN, then the signal is ignored. If the handler is set to SIG_DFL, then the default action associated with the signal (see signal(7)) occurs. Finally, if the handler is set to a function sighandler then first either the handler is reset to SIG_DFL or an implementation-dependent blocking of the signal is performed and next sighandler is called with argument signum.

Using a signal handler function for a signal is called "catching the signal". The signals SIGKILL and SIGSTOP cannot be caught or ignored.

The signal() function returns the previous value of the signal handler, or SIG_ERR on error. The original Unix signal() would reset the handler to SIG_DFL, and System V (and the Linux kernel and libc4,5) does the same. On the other hand, BSD does not reset the handler, but blocks new instances of this signal from occurring during a call of the handler. The glibc2 library follows the BSD behaviour.

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-

```
aditi@aditi-Lenovo-ideapad-330S-14IKB-U:~/ADnOR/Assignments$ gcc 2A.c
aditi@aditi-Lenovo-ideapad-330S-14IKB-U:~/ADnOR/Assignments$ ./a.out
Input now in 10 seconds
44
You entered 44
aditi@aditi-Lenovo-ideapad-330S-14IKB-U:~/ADnOR/Assignments$ gcc 2A.c
aditi@aditi-Lenovo-ideapad-330S-14IKB-U:~/ADnOR/Assignments$ ./a.out
Input now in 10 seconds
8
You entered 8
```

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/