## **IPC: Interrupts and Signals: signal(any fives type of signal ), alarm, kill, raise, killpg, signal , sigaction**

**2.6** Write application of signal handling in linux OS and program anyone.

**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:**

What are signals?

1.Signal is a notification, a message sent by either operating system or some application to your program (or one of its threads).

2.Each signal identified by a number, from 1 to 31. Signals don’t carry any argument and their names are mostly self explanatory. For instance SIGKILL or signal number 9 tells the program that someone tries to kill it.

3.An application program can specify a function called a signal handler to be invoked when a specific signal is received. When a signal handler is invoked on receipt of a signal, it is said to catch the signal. A process can deal with a signal in one of the following ways:

-The process can let the default action happen

-The process can block the signal (some signals cannot be ignored)

-The process can catch the signal with a handler.

4.Signal handlers usually execute on the current stack of the process. This lets the signal handler return to the point that execution was interrupted in the process. This can be changed on a per-signal basis so that a signal handler executes on a special stack. If a process must resume in a different context than the interrupted one, it must restore the previous context itself.

**Data Dictionary:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr Number | Variable/Function | Datatype | Use |
|  |  |  |  |
| 1 | sig\_handler | void | Catch interrupt signal. |
|  |  |  |  |

**Program:**

#include <fcntl.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

int main (int argc, char\* argv[])

{

char\* file = argv[1];

int fd;

struct flock lock;

printf ("opening %s\n", file);

/\* Open a file descriptor to the file. \*/

fd = open (file, O\_WRONLY);

printf ("locking\n");

/\* Initialize the flock structure. \*/

memset (&lock, 0, sizeof(lock));

lock.l\_type = F\_WRLCK;

/\* Place a write lock on the file. \*/

fcntl (fd, F\_SETLKW, &lock);

printf ("locked; hit Enter to unlock... ");

/\* Wait for the user to hit Enter. \*/

getchar ();

printf ("unlocking\n");

/\* Release the lock. \*/

lock.l\_type = F\_UNLCK;

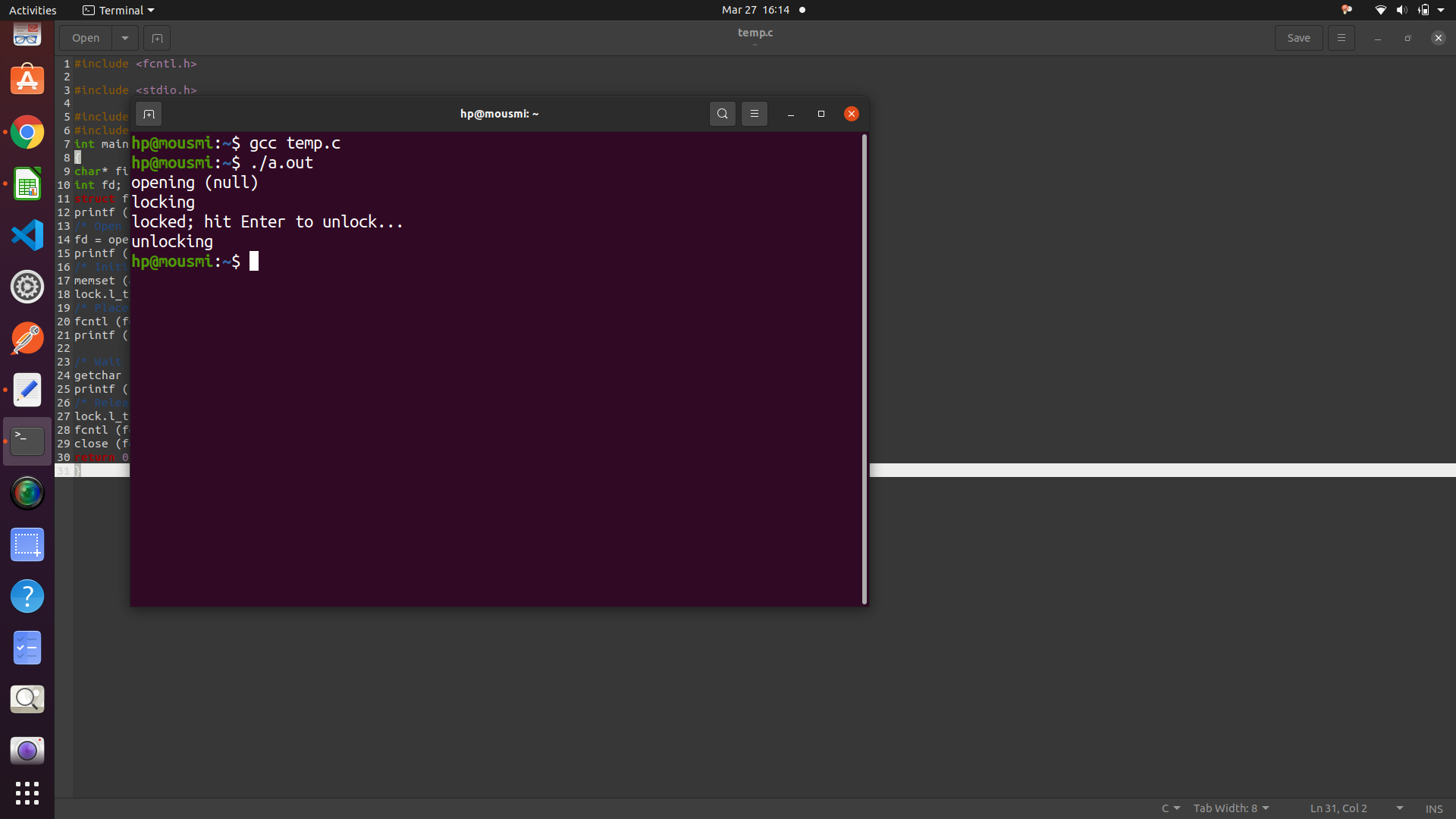
fcntl (fd, F\_SETLKW, &lock);

close (fd);

return 0;

}

**Output:**

****

**Conclusion:**

Signal handling is used for various applications in Linux based OS. File locking is one of the applications discussed above

**References:**

[www.cs.cf.ac.uk/Dave/C/CE.html](http://www.cs.cf.ac.uk/Dave/C/CE.html)