**2.5 Write a program to send signal by five different signal sending system calls and identify the difference in working with example.**

**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:**

INTERRUPTS AND SIGNALS:

Signals and interrupts behave in pretty similar ways. The difference is that signals happen to a

process (which lives in a virtual environment), while exceptions are system-wide. Certain faults are flagged by the CPU as an exception, and then mapped to a signal that is delivered to the process by the kernel.

TYPES OF SIGNALS:

SIGHUP:-Hang up detected on controlling terminal or death of controlling process

SIGINT:-Issued if the user sends an interrupt signal (Ctrl + C)

SIGQUIT:-Issued if the user sends a quit signal (Ctrl + D)

SIGFPE:-Issued if an illegal mathematical operation is attempted

SIGKILL:-If a process gets this signal it must quit immediately and will not perform any clean-up operations

SIGALRM:-Alarm clock signal (used for timers)

Types of System Call:

1 .Alarm System Call:

alarm - set an alarm clock for delivery of a signal.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 due to be

delivered, or zero if there was no previously scheduled alarm.

2 .kill System Call:

The kill() system call can be used to send any signal to any process group or processIf pid

is positive, then signal sig is sent to the process with the ID specified by pid.If pid equals 0, then sig is sent to every process in the process group of the calling process.If pid equals -1, then sig is sentto every process for which the calling process has permission to send

signals, except for process 1 (init).If pid is less than -1, then sig is sent to every process in the process group whose ID is -pid.

3. Raise System call

raise - send a signal to the caller The raise() function sends a signal to the calling process or threaasingle- threaded program it is equivalent to kill(getpid(), sig); In a multithreaded program it is equivalent to pthread\_kill (pthread\_self(), sig); If the signal

causes a handler to be called,raise() will return only after the signal handler has returned.

4. Killpg System Call:

send signal to a process group

5. Sigaction System Call:

examine and change a signal action

**Data Dictionary:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr | Variable/Function | Datatype | Use |
| Number |  |  |  |
|  |  |  |  |
| 1 | pid | int | Process ID. |
|  |  |  |  |
| 2 | sighup | void | Used for handling signal hang up. |
|  |  |  |  |
| 3 | sigint | void | Used for handling interrupt signal. |
|  |  |  |  |
| 4 | sigquit | void | Used for dumped core handling. |
|  |  |  |  |
| 5 | sighold | void | Function that add sig to the calling process' signal |
|  |  |  | mask. |
|  |  |  |  |
| 6 | sigrelse | void | Function that remove sig from the calling process' |
|  |  |  | signal mask. |
|  |  |  |  |
| 7 | sigpause | void | Function that restore the process' signal mask to its |
|  |  |  | original state before returning. |
|  |  |  |  |
| 8 | sigignore | void | Function that set the disposition of sig to SIG\_IGN. |
|  |  |  |  |

**Program:**

#include<stdio.h>

#include<signal.h>

#include<stdlib.h>

#include<sys/types.h>

#include<unistd.h>

void sighup();

void sigint();

void sigquit();

void sig\_handler(int);

int main()

{

int pid;

if ((pid = fork()) < 0)

{

perror("fork");

exit(1);

}

if (pid == 0)

{

signal(SIGHUP,sighup);

signal(SIGINT,sigint);

signal(SIGQUIT, sigquit);

signal(SIGUSR1, sig\_handler);

signal(SIGSTOP, sig\_handler) ;

for(;;);

}

else

{

printf("\nPARENT: sending SIGHUP\n\n");

kill(pid,SIGHUP);

sleep(3);

printf("\nPARENT: sending SIGINT\n\n");

kill(pid,SIGINT);

sleep(3);

printf("\nPARENT: sending SIGQUIT\n\n");

kill(pid,SIGQUIT);

sleep(3);

printf("\nPARENT: sending SIGUSR1\n\n");

kill(pid,SIGUSR1);

sleep(3);

printf("\nPARENT: sending SIGSTOP\n\n");

kill(pid,SIGSTOP);

sleep(3);

}

}

void sighup()

{

signal(SIGHUP,sighup);

printf("CHILD: I have received a SIGHUP\n");

}

void sigint()

{

signal(SIGINT,sigint);

printf("CHILD: I have received a SIGINT\n");

}

void sigquit()

{

printf("My DADDY has Killed me!!!\n");

exit(0);

}

void sig\_handler(int signo)

{

if (signo == SIGUSR1)

{

signal(SIGUSR1, sig\_handler);

printf("received SIGUSR1\n");

}

else if (signo == SIGSTOP)

{

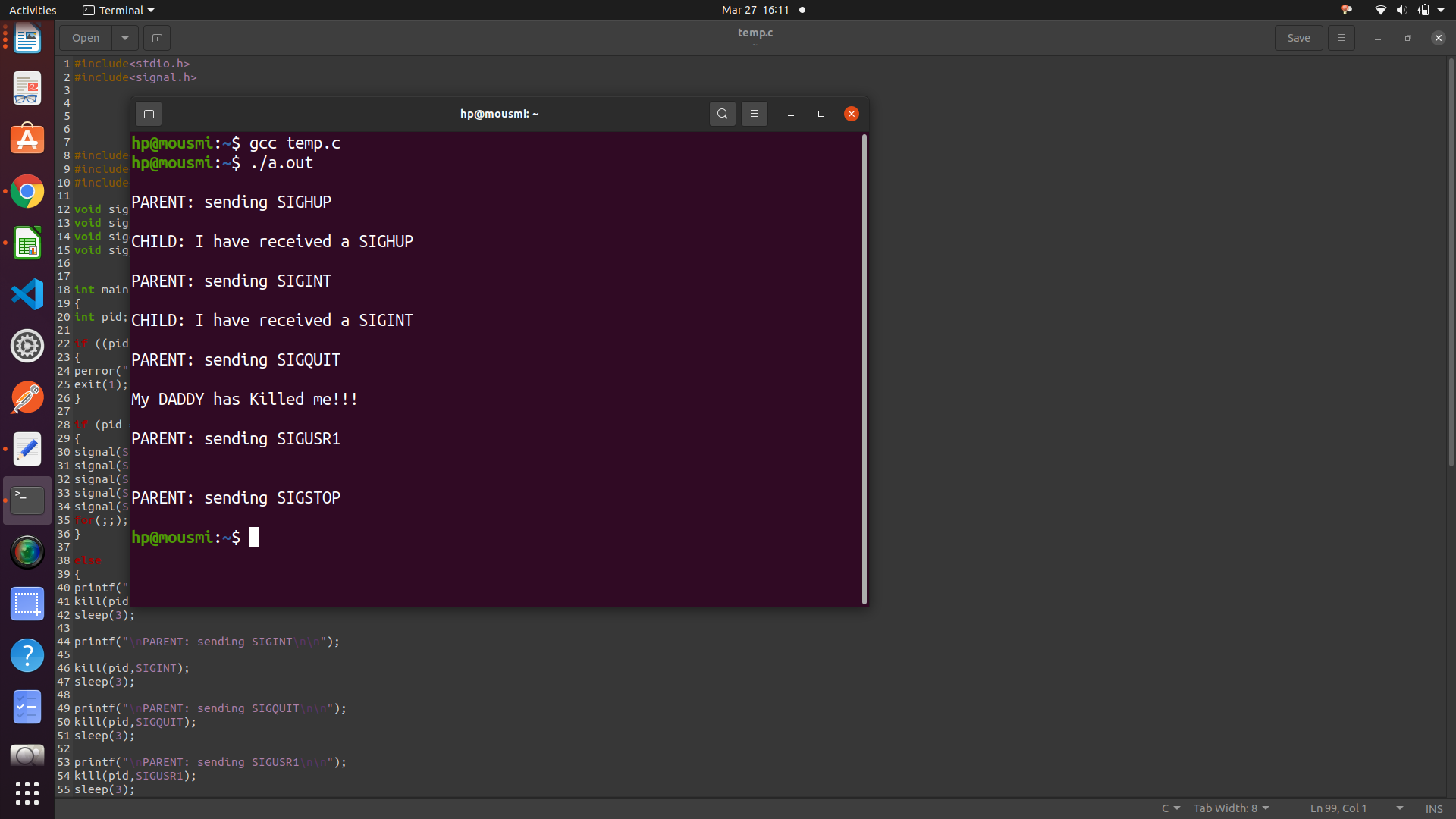
signal(SIGSTOP, sig\_handler);

printf("received SIGSTOP\n");

}

}

**Output:**

****

**Conclusion:**

The following assignment deals with providing the guideline about the various types of system call and their usage in IPC and management about process in windows and linux operating system and written the application using the signal and alarm system call.

**References:**

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