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

**Assignment No. 2-b**

**Subtitle:-** Write a application or program that communicates between child and parent processes using kill() and signal().

Objectives:

1.To learn about Processing Environment.

2. To know the difference between fork/vfork and various execs variations.

3. Use of system call to write effective programs.

**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)kill:**

**Name:**

Kill-send signal to a process

**Synopsis:**

#include<sys/types.h>

#include<signal.h>

int kill(pid\_t pid,int sig);

**Description:**

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

If pid is positive ,the signal sig is sent to pid.

If pid equals 0,then sig is sent to every process in the process group of the current process.

If pid equals -1,then sig is sent to 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 –pid.

If sig is 0,then no signal is sent,but error checking is still performed.

For a process to have permission to send a signal is must either be privileged (under Linux:have the CAP\_KILL capability),or the real or effective user ID of the sending process must equal the real or saved set\_user\_ID of the target process.In the case of SIGCONT it suffices when the sending and receiving processes belong to the same session.

**Return Value:**

On success (at least one signal was sent),zero is returned. On error,-1 is returned , and errno is set appropriately.

**2)Signal:**

**Name:**

Signal –ANSI C signal handling

**Synopsis:**

#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)instead. See Portability below. signal() sets the deposition of signal signum to handler,which is either SIG\_IGN,SIG\_DFL, or address of 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\_IGN ,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 probability below ),and then handler is called with argument signum.If invocation of the handler caused the signal to be locked ,then thesignal is unblocked upon return from the handler 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 is set to indicate the cause.

**Flowchart:**

main()

**Not 0 0**

If pid == fork()

signal(SIGHUP,sighup);

signal(SIGINT,sigint);

signal()SIGQUIT,sigquit;

Parent Process

kill(pid,SIGHUP);

kill(pid,SIGINT)

kill(pid,SIGQUIT)

**Data variables:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.Number** | **Variable/Function** | **Datatype** | **Use** |
| 1 | inthandle | void | Used for handling interrupt signal. |
| 2 | quithandle | void | Used for dumped core handling. |
| 3 | huphandle | void | Used for handling signal hang up. |
| 4 | pid | pid\_t | Used for child ID |
| 5 | ppid | pid\_t | Used for parent ID |

**Program:**

**#include<stdio.h>**

**#include<stdlib.h>**

#include<unistd.h>

#include<signal.h>

void inthandle(int sig)

{

signal(SIGINT,inthandle);

printf("SIGINT invoked by daughter\n");

}

void quithandle(int sig)

{

signal(SIGQUIT,quithandle);

printf("SIGQUIT invoked buy son\nHe killed me\n");

exit(1);

}

void huphandle(int sig)

{

signal(SIGHUP,huphandle);

printf("SIGHUP invoked by child\n");

}

int main()

{

pid\_t ppid,pid;

ppid=getpid();

if((pid=vfork())<0)

{

printf("Fork Failed!!!!\n");

exit(1);

}

else if(pid==0)

{

printf("In Child!!!!\n");

signal(SIGINT,inthandle);

signal(SIGHUP,huphandle);

signal(SIGQUIT,quithandle);

printf("Looping\n");

for(;;);

}

else

{

printf("In Parent !!!!!\n");

printf("kill SIGHUP\n");

kill(pid,SIGHUP);

sleep(2);

printf("kill SIGINT\n");

kill(pid,SIGINT);

sleep(2);

printf("kill SIGQUIT\n");

kill(pid,SIGQUIT);

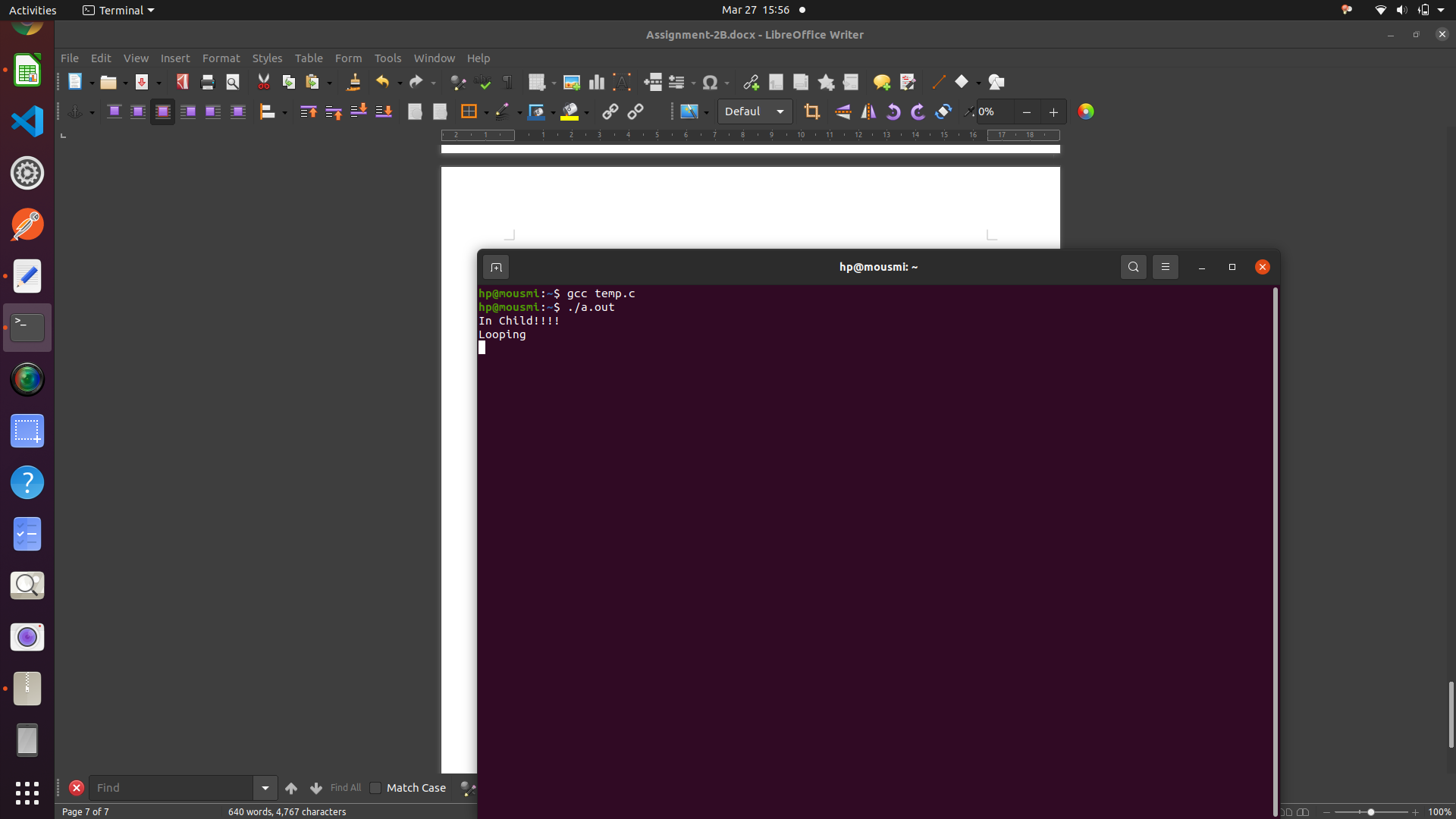
sleep(2);

exit(0);

}

}

**Output:**



**Conclusion:**

Learned about kill() and signal() system calls and how to use them effective in program in various cenarios.

**References:**

<https://unix.stackexchange.com/questions/317492/list-of-kill-signals/>

<https://www.tutorialspoint.com/unix/unix-signals-traps.htm/>