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

**Assignment No. 2-c**

**Subtitle:-** Write a application or program that communicates between to process opened in two terminal using kill() and signal().(I)

**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:**

In Unix and Unix-like operating systems,kill is a command used to send a signal to a process.By default ,the message sent is the termination signal,which requests that the process exit.But kill is something misnomer;the signal sent may have nothing to do with process killing.The kill command is a wrapper around the kill() system call,which sends signals to processes or process group on the system ,referenced by their numeric process IDs(PIDs) or process IDs(PGIDs).

int kill(int pid, int signal) - a system call that send a signal to a process, pid. If pid is greater than zero, the signal is sent to the process whose process ID is equal to pid. If pid is 0, the signal is sent to all processes, except system processes. kill() returns 0 for a successful call, -1 otherwise and sets errno accordingly.

**-KILL -**The kernel will let go of the process without informing the process of it.An unclean kill like this could result in data loss.This is the “hardest”, “roughest” and most unsafe kill signal available and should only be used to stop something that seems unstoppable.

**Program:**

#include<stdio.h>

#include<sys/types.h>

#include<signal.h>

#include<sys/ipc.h>

#include<sys/shm.h>

void SIGINT\_handler(int);

void SIGQUIT\_handler(int);

int ShmID;

pid\_t \*ShmPTR;

void main(void)

{

int i;

pid\_t pid=getpid();

key\_t MyKey;

if(signal(SIGINT,SIGINT\_handler)==SIG\_ERR){

printf("SIGQUIT install error\n");

exit(2);

}

MyKey =ftok(".",'s');

ShmID=shmget(MyKey,sizeof(pid\_t),IPC\_CREAT|0666);

ShmPTR=(pid\_t \*)shmat(ShmID,NULL,0);

\*ShmPTR=pid;

for(i=0;;i++)

{

printf("From Process %d:%d\n",pid,i);

sleep(1);

}

}

void SIGINT\_handler(int sig)

{

signal(sig,SIG\_IGN);

printf("From SIGINT: just got a %d(SIGINT^C) signal\n",sig);

signal(sig,SIGINT\_handler);

}

void SIGQUIT\_handler(int sig)

{

signal(sig,SIG\_IGN);

printf("From SIGQUIT: just got a %d (SIGQUIT^[\\)signal""and](../../../../../../..//)signal) is about to quit\n",sig);

shmdt(ShmPTR);

shmctl(ShmID,IPC\_RMID,NULL);

exit(3);

}

**Flowchart:**

main(void)

signal(SIGINT,SIGINT\_handler)==SIGERR

Parent Process

signal(SIGQUIT,SIGQUIT\_handler)==SIGERR

**true false**

exit(2)

i=0

**false**

**true**

i++;;

sleep(1)

SIGINT\_handler(int sig)

exit(3)

shmdt(ShmID,IPC\_RMID,NULL)

shmdt(ShmPTR)

signal(sig,SIGIGN)

SIGQUIT\_handler(int sig)

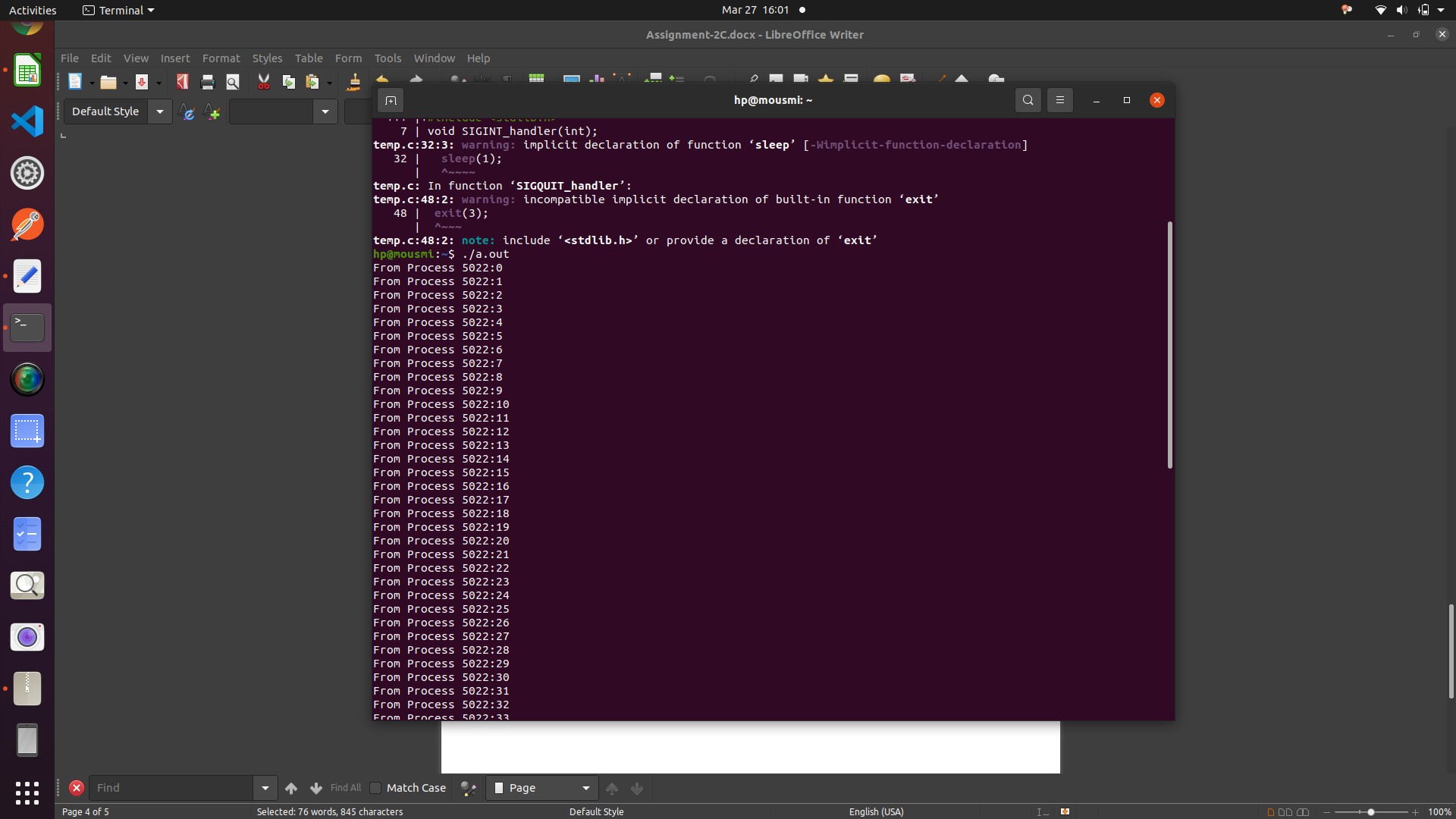
signal(sig,SIGINT\_handler)

signal(sig,SIG\_IGN)

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. Number** | **Variable/Function** | **Datatype** | **Use** |
| 1 | SIGINT\_handler | void | Used for handling interrupt signal. |
| 2 | SIGQUIT\_handler | void | Used for dumped core handling. |
| 3 | pid | pid\_t | Process ID. |
| 4 | MyKey | key\_t | Shared memory key |
| 5 | ShmID | int | ID of shared memory |
| 6 | ShmPTR | pid\_t\* | Pointer |
| 7 | i | int | Iterating for loop |

**Data Variables**:

**Output**:



**Conclusion:**

Kill and signal IPC demonstrated.

**References:**

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

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