Lab 4

3.1 Introduction to Signals

You press the CTRL+C it sends a kill signal to the program from the terminal The program has a function signal(SIGINT, my_routine); what this line does is it catches the signal which is SIGINT. It then runs the function that is specified which is my_routine, and my_routine then runs the printf putting Running my_routine on the terminal and never ends the program.

OUTPUT:

Entering infinite loop
^CRunning my_routine
^CRunning my_routine
^\Quit (core dumped)

It is known as establishing a handler when a signal handler function is used.

The name of the signal handler function for this code is my_routine.

It states in the man pages that if the disposition is set to SIG_DFL that it will default to the normal action. The program terminated because it went to the default function of the signal that was sent and that was to exit the program.

OUTPUT:

Entering infinite loop
^C

Now the program is ignoring SIGINT. This because by specifying SIG_IGN we are telling the program to ignore the SIGINT coming in. So the program ignores it and continues on with its infinite loop.

OUTPUT:

Entering infinite loop
^C^C
^C

With this part when we replace the SIGINT with SIGQUIT it changes what signal it is looking for. That means if you press CTRL+C it will end the program however if you push CTRL+\ now it will go to the function specified. In this case we once again specify my_routine.

OUTPUT:

Entering infinite loop
^\Running my_routine
^\Running my_routine
^C

3.2 Signal Handlers

For this one it takes both the SIGINT(CTRL+C) and SIGQUIT(CTRL+\) and runs the my_routine. For my routine it prints The signal number is and what the signal number is. The

signal number is stored in signo and only has the signal number that called the function. The number of SIGINT is 2 and the number of SIGQUIT is 3.

OUTPUT:

```
Entering infinite loop
^CThe signal number is 2.
^CThe signal number is 2.
^\The signal number is 3.
^\The signal number is 3.
```

3.3 Signals for Exceptions

OUTPUT:

```
Entering infinite loop
Caught a SIGFPE
```

PROGRAM:

```
#include <signal.h>

void zero_Error();

int main() {

        signal(SIGFPE, zero_Error);
        printf("Entering infinite loop\n");

        int a = 4;
        a = a/0;
}

void zero_Error() {
        printf("Caught a SIGFPE\n");
        exit(1);
}
```

The line that should come first is the signal() statement. If you were to do the division by zero first then the signal would come before the handler would be ready to receive it. This would cause the program to fail as it would run the default disposition of the signal and give a floating point exception error.

3.4 Signals using Alarm

The parameters of this program are the command to run so ./program_name, a message to print when the alarm goes off, and finally a time that the alarm will go off.

OUTPUT:

./alarm Hola 10
Entering infinite loop
Hola

The alarm takes a parameter of time. The time needs to be in seconds. Once the specified time passes the SIGALRM signal is delivered to the calling process. Since we specified signal(SIGALRM, my_alarm) once the signal is received the function my_alarm is run.

3.5 Pipes

In this program there are two processes running the parent which puts the msg into the pipe and the child which reads from the pipe and prints it.

This program uses a pipe with integer array p as the parameter. It then forks the program with an if statement to differentiate between parent and child. The parent writes to the pipe using write (p[1], msg, MSGSIZE) where MSGSIZE is defined as 16. The child processes sleeps for 1 to give the parent time to write. It then does read(p[0], inbuff, MSGSIZE); which is where it receives the message and stores it in inbuff. It then prints inbuff to the terminal.

The sleep statement was used to give the parent time to write to the file before the child tries to read from it. To make this statement better you could have had the child write to the pipe and the parent use waitpid to wait for the child to finish before trying to read from the pipe.

3.6 Signals and Fork

OUTPUT:

^CReturn value from fork = 2761
Return value from fork = 0
^CReturn value from fork = 0
Return value from fork = 2761
^CReturn value from fork = 2761
Return value from fork = 0

In this program two process are running the parent and the child.

The process with fork = 0 is the child process and the process where fork = 2761 is the parent process.

Both of the process received the signal. I believe this is because they are child and parent.

3.7 Shared Memory Example

CLIENT OUTPUT:

Message read: abcdefghijklmnopqrstuvwxyz
Client done reading memory

SERVER OUTPUT:
Server detected client read

They use a key that the memory space has and is unique to it to find the same memory space.

The flaw in the program is that server could end before the client could write to the terminal window. This could happen if the shared memory already contained information that started with *.

Without Server OUTPUT:
Message read: *bcdefghijklmnopqrstuvwxyz
Client done reading memory

Running the client without the server prints the above. This is printed because the message that was saved in shared memory was never cleared so it was still there. It will remain there until something else overwrites it for example running ./server again.

WITHOUT SERVER 2ND TIME OUTPUT:
shmget failed: No such file or directory

The two lines added cleared the shared memory space so that there was nothing there.

3.8 Message Queues and Semaphores

The maximum size for a message text is 8192 bytes.

msgrcv fails and is never automatically restarted and returns an errno.

You can use ipcrm to remove a message queue, semaphore set or shared memory id. You can only delete these objects if you are a superuser, creator, or owner of the object.

The semaphores on linux are general as they can be higher than a value of 1.

You can use a named or unnamed semaphore. To use a named semaphore you can pass the name to sem_open() and then you can use it. To use a unnamed semaphore you must create a shared memory space and then use sem_init() to initialize the semaphore.