Name: Khushveen Kaur Umra
Section: 2
University ID: 947005108

Lab 4 Report

Summary (10pts):

In this lab, I learned a lot of information in-regards to the different signals, and their use cases. Through exercises 3.1-3.5, I learned about the Linux signals. With these exercises, I learned about how different methods of communications take place between UNIX processes.

I learned how the order of signals is important in such cases, as it will be used to determine that the signal will actually catch properly, and will not terminate the program directly. I also learned how to time signal using an alarm, and how it can be complicated when you use fork, as it makes it difficult to manage it.

Lab Questions:

3.1:

6 pts After reading through the man page on signals and studying the code, what happens in this program when you type CTRL-C? Why?

In this program, CTRL-C is considered to be an interrupt signal. Here, when you type CTRL-C, the signal code picks it up, and it runs the “my_routine”, rather than it closing the program. This happens because it is looking for a keyboard interrupt with “SIGINT”.

3 pts Omit the signal(...) statement in main. Recompile and run the program. Type CTRL-C. Why did the program terminate?

In this program, we have removed the signal(…) statement, which means that we are no longer catching and changing its use, which is to terminate the signal. Since CTRL-C is a term type keyboard interrupt signal, it means that its default response is to terminate the signal, and that is what happens when recompile and run the program.

3 pts In main, replace the signal( ) statement with *signal(SIGINT, SIG_IGN)*. Recompile, and run the program then type CTRL-C. What's happening now?

When we replace the signal(..) statement with signal(SIGINT, SIG_IGN), we are technically using "SIG_IGN" to ignore the signal. This also sets CTRL-C's signal to do nothing instead of terminating the process.

3pts The signal sent when CTRL-\ is pressed is SIGQUIT. Replace the *signal()* statement with *signal(SIGQUIT, my_routine)* and run the program. Type CTRL-\. Why can't you kill the process with CTRL-\ now?

When we replace the signal() statement with signal(SIGQUIT, my_routine) and run the program, we are technically doing the same thing when we replaced the signal() statement with signal(SIGINT, SIG_IGN), The reason we can't kill the process with CTRL-\ is because we are changing it's use to running "my_routine", instead of it terminating the process, as it's default action will still be to kill the process.

3.2:

5pts What are the integer values of the two signals? What causes each signal to be sent?

The integer value of "CTRL-C"is 2.
The integer value of "CTRL-\" is 3.

The signal() statement causes each signal to be sent, as it calls both SIGINT and SIGQUIT, and changes their default action from terminating the process to be running "my_routine". This eventually starts "my_routine" and also assigns it to its proper integer values.

3.3:

10 pts Include your source code

```c
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

void divZero();

int main()
{
        signal(SIGFPE, divZero);

        int a = 4;
        a = a / 0;

        return 0;
}

void divZero()
{
        printf("Caught a SIGFPE, \n");

        exit(0);
}
```

5pts Explain which line should come first to trigger your signal handler: the signal() statement or the division-by-zero statement? Explain why.

To trigger the signal handler, the signal() statement should come first because it will wait for a specific signal, and will change what would normally happen. This is because it will act as an action listener.
If division-by-zero came first, it wouldn't allow the program to have enough time to catch the signal before it eventually terminates the code.

This is why, the signal() statement needs to come first, in order for the program or the code to run smoothly with enough time.

3.4:

4pts What are the parameters input to this program, and how do they affect the program?

The two parameters input to this program are “message”and an “integer value”.
The message parameter is what gets printed when you run the program.
The integer value will determine the time in seconds.
The time is given to the alarm, and its value will determine how long the alarm will wait before it signals. After the signal is initiated, “my_routine” will run, and it will print the message. The different values that will affect the printed message, and will determine how long the program will have to wait before the next message.

6pts What does the function “alarm” do?? Mention how signals are involved.

The function “alarm(time)” waits for time in seconds, and then sends a SIGALRM signal.
In this specific program, we changed the function of SIGALRM, by making it run a function, and not allowing its normal function of terminating the process as a term signal.

3.5:

2pts Include the output from the program.

2pt How many processes are running?

There are two processes that are running.

3pts Identify which process sent each message.

The parent process prints: Return value from fork = 4944

The child process prints: Return value from fork = 0

3pts How many processes received signals?

Both the parent and the child processes received signals. (2)

3.6:

2pts How many processes are running? Which is which (refer to the if/else block)?

6pts Trace the steps the message takes before printing to the screen, from the array msg to the array inbuff, and identify which process is doing each step.

2pts Why is there a sleep statement? What would be a better statement to use instead of sleep (Refer to lab 2)?

3.7:

3pts How do the separate processes locate the same memory space?

3pts There is a major flaw in these programs, what is it? (Hint: Think about the concerns we had with threads)

3pts Now run the client without the server. What do you observe? Why?

6pts Now add the following two lines to the server program just before the exit at the end of main:
shmdt(shm)
shmctl(shmid, IPC_RMID, 0)
Recompile the server. Run the server and client together again. Now run the client without the server. What do you observe? What did the two added lines do?

3.8:

2pts Message queues allow for programs to synchronize their operations as well as transfer data. How much data can be sent in a single message using this mechanism?

2 pts What will happen to a process that tries to read from a message queue that has no messages (hint: there is more than one possibility)?

3pt Both Message Queues and Shared Memory create semi-permanent structures that are owned by the operating system (not owned by an individual process). Although not permanent like a file, they can remain on the system after a process exits. Describe how and when these structures can be destroyed.

3pt Are the semaphores in Linux general or binary? Describe in brief how to acquire and initialize them.