Name: Taylor Dugger

Section: B

University ID: 945506562

Lab 4 Report

Summary (10pts):

This lab was a lot of information and usage on signals. I had little knowledge on this topic, so I learned quite a bit actually. I didn't know the signal for when you use CTRL-C to end a process on the command line. So I learned more about what signals do what, and how you can change them to your purpose. I also didn't know anything about alarms, so I learned how to create and signal an alarm. It was nice getting an example in 3.6 using signals with forks. This can be confusing, but it was good to see examples with this. I also knew nothing about shared memory and message queues. While we didn't do a ton into this on this lab, I still got a good base about these topics.

Lab Questions:

3.1:

2pts After reading through the man page on signals and studying the code, what happens in this program when you type CTRL-C?

my_routine is called and Running my routine is printed out

1pt According to the man page, what is it called when a signal handler function is used?

There is an interrupt from the keyboard when CTRL-C is pressed

1pts What is name of the signal handler function for this code?

SIGINT

2pts Omit the signal( ) statement in main and run the program. Type CTRL-C. Look up SIG_DFL in the man pages (man signal). Why did the program terminate? Hint: find out how a program usually reacts to receiving what CTRL-C sends (man 7 signal).

When it is set to sig_dfl the default action is called associated with signal. When CTRL-C is normally pressed it will terminate the program.

2pts In main, replace the signal( ) statement with *signal(SIGINT, SIG_IGN)*. Run the program and type CTRL-C. Look up SIG_IGN in the man pages. What's happening now?

SIG_IGN is ignore the signal. So nothing will happen now.

2pts 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? You should be able to terminate the process with CTRL-C.

Because we put sigquit in signal we are replacing the value of sigquit with my_routine instead of quit. So when we press CTRL-\ we get my_routine instead now.

3.2:

4pts In your report, note what causes each signal to be sent and what the number the signal is. You can kill the process using the kill command without any options (look up the process id with ps in a new terminal window).

SIGINT and SIGQUIT are value 2 and 3, respectively. Each is called from a different command, interrupt or quit. So depending on what is called from the keyboard then either signal will be called and my_routine is executed. Based on the chart in man 7 signal, SIGINT is value 2 and SIGQUIT is 3. So depending on if you enter CTRL-C (SIGINT) or CTRL_\ (SIGQUIT) my_routine will print out either value from signo.

3.3:

5pts Include your source code

```c
#include <signal.h>
#include <stdio.h>
void my_routine();
int main(){
	signal(SIGFPE, my_routine);
	int a = 4;
	a = a/0;
	while(1){
		sleep(10);
	}
}
void my_routine(){
	printf("Caught a SIGFPE.\n");
}
```

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

The division by zero statement. Cause there is no reason for it to trigger from just the signal statement. I commented out the a/0 statement and my_routine was never called, so it only calls when a division by zero is ran.

3.4:

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

The first parameter is the message you want to display and the second parameter is after how many seconds it will appear.

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

Alarm will set an alarm to go off after how many seconds. The signal SIGALARM will look for alarms set and will signal the function in signal(). So you need to set up the signal with sigalarm(so it knows to look out for alarms) and the function call. Then you make an alarm with a time to go off. The signal will then call the function after the alarm goes off.

3.5:

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

2

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? Think about what happens if one process runs first. Think back to lab 2 -- what would be a better statement to use instead of sleep for this small example? You might have to reverse the role of parent and child.

3.6:

1pts Include the output from the program.

```
^CReturn value from fork = 11335
Return value from fork = 0
```

1pt How many processes are running?

2. The parent and its child.

2pts Identify which process sent each message.

The parent returned the value 11335. And the child returned 0.

2pts How many processes received signals?

Both of them did. The fork statement was before the signal(). So both processes received the signal, so both of them printed my_routine.

3.7:

1pts How do the separate processes locate the same memory space?

The shared memory space uses keys that both of these have. So the key will allow them both to use the same space.

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

You don't really know what thread is going to run. Plus since we are waiting for the client to put a character in memory, this could be a problem. You don't know if it could be locked which could lead to issues with memory there. You also don't know what thread is going to go first so the client could put data first and the server won't know what to look for.

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

The first character a is replaced with *. The way that this is set up, is that the client is looking for the server so it can get all the correct data. Without it then the client won't get the correct data and * will be in place of a.

3pts 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?

It was faster and it waited for client to finish.

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?

The max size of a message is 8192.

1pt 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)?

mq_receive will block until a message becomes available or interrupted. If a flag is on, then it will fail.

1pt 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.

It will exist until it is marked with IPC_RMID and the last process using it detaches.

1pt Are the semaphores in Linux general or binary? Describe in brief how to acquire and initialize them.

Semaphores can be both. They are binary when a single shared resource is used. Otherwise it is counting. Since they are just counters, you can either wait on the variable or signal the variable.