Name: Kent Mark
Section: 5
University ID: 583972417

Lab 4 Report

Summary (10pts):

This lab gave us a better look at inter-process communication and Linux signals. We also learned about semaphores, pipes, and memory sharing.

Lab Questions (80 pts):

3.1 - Introduction to Signals

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

The signal is allowed to be sent into the function and the routine is allowed to keep running in the infinite loop.

3pts Remove the signal(...) statement in main. Recompile and rerun the program. Type CTRL-C. Why did the program terminate? Hint: find out how a program usually reacts to receiving what CTRL-C sends (man 7 signal).

The program terminated because there is no longer a signal processor that receives the termination signal

3pts Replace the signal( ) statement with *signal(SIGINT, SIG_IGN)*. Run the program and press CTRL-C. Look up SIG_IGN in the man pages. What's happening now? (HINT: Look up SIG_IGN in man signal)

As per the man pages, SIG_IGN is a function that sets a signal to be ignored. As a result only the main is now running.

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

The process can't be killed at this point because the routine is running and the process that we wanted to kill has already been stopped due to SIGQUIT.

3.2 – Signal Handlers

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

The integers that are printed are 2 and 3. Each signal is sent because SIGINT and SIGQUIT both interrupt process that is currently running and run my_routine instead.

3.3 – Signals for Exceptions

10pts Include your source code

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

void my_method();

int main() {

        signal(SIGFPE, my_method);

        int a = 4;

        a = a/0;

        return 0;

}

void my_method() {

        printf("Caught a SIGFPE!/n");
}
```

5pts Which of the following statements should come first to trigger your signal handler? The signal() statement, or the division-by-zero statements. Explain why.

The signal() statement comes first because it interrupts and overrides the running method.

3.4 – Signals using alarm()

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

The input parameters are: msg and time.

Time will provide the time delay for the signal processing and msg will send a signal that prints the msg.

6pts What does the statement *alarm(time)* do? Mention how signals are involved.

The statement reads an int value and uses that value as the amount of time it will wait. Signals are used such that after the alarm(time) statement's wait period has ended a signal will be sent to either stop, interrupt, or switch a particular process, while allowing another process to run, and ultimately alarm(time) will print the message and terminate the program.

3.5 – Signals and fork

2pts How many processes are running?

2 process are running

2pts Identify which process prints out which message.

Fork = 0 is from the child process, and fork = 29300 is from the parent process

2pts How many processes received signals?

2

3.6 - Pipes

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

2 process are running. The if is the child process and the else is the parent process

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.

The child process: The if statement executes and pipes it's output to the parent which is the else statement.
The parent process: The else statement takes the output and reads it after execution is completed.

2pts Why is there a sleep statement? What would be a better statement to use instead of sleep for this small example? Hint: you might have to switch the role of parent and child…

The sleep statement allows the child to finish execution first so that the parent can get the output, versus the other way around. For this particular example a wait statement might have been better to use than the sleep statement.

3.7 – Shared Memory

2pts How do the separate processes locate the same memory space?

The separate processes locate the same memory space using the shmget and shmid.

3pts There is a major flaw in these programs. What is it? Hint: think about race conditions.

The flaw here is that more than processes want to gain access to the same resource.

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

There was a race condition.

3pts Recompile the server. First run the server and client together. Then run the client without the server. What do you observe? What did the two added lines do? (HINT: look at the man pages of shmdt and shmctl; look for the explanation of IPC_RMID in the shmctl man pages)

If the server is run first the client will print the output, at this point server will then print it's message and the data will be overwritten.

If the client is run first: We get a race condition

3.8 – Message Queues and Semaphores

2pts Read the man pages of *msgsnd*. 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?

8192 bytes

2pts Read the man pages of *msgrcv*. What will happen to a process that tries to read from a message queue that has no messages of the requested type? Hint: there are multiple different cases.

It will return -1 and an error such as efault, einval, or enornsg

2pts 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.

The structures mentioned can be destroyed when shmctl is called and data is available for the client to use.

2pts Are the semaphores described by the above man pages counting semaphores (i.e. general purpose semaphores), or binary semaphores? Describe in brief how to acquire and initialize them.

The semaphores described are counting semaphores. They can be acquired by looking at increments or decrements of ID semaphore. They can be initialized by sem_init();