Name:
Section:
University ID:

Lab 4 Report

Summary (10pts):

In this lab we went into the different signals that are available. We saw how there is a couple that are CTL-C and CTL-\. We saw that you can get a list of the different signals and their corresponding signal number. A couple of other signals we looked at was the divide by zero and the alarm signals. Next, we looked at the pipe function and how you can direct a stream from one process to another. Lastly, we worked worked with a shared memory. This allows multiple streams to access the same piece of data to share data. We saw that you must free it at the end of a process or otherwise it will stay after the process is done completing. Overall, this lab was a great introduction to signals and great experience with various system calls.

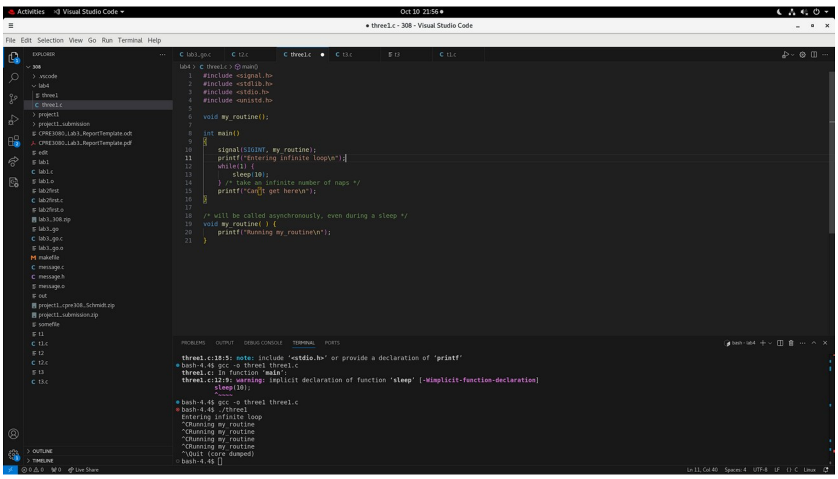
Lab Questions:

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?

We overwrite the signal function. The default option is for it to terminate the program but we made it print out running_my_routine.

3

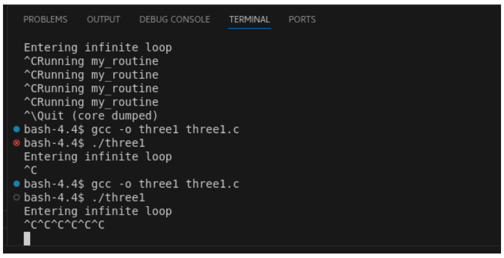


pts Omit the signal(...) statement in main. Recompile and run the program. Type CTRL-C. Why did the program terminate?

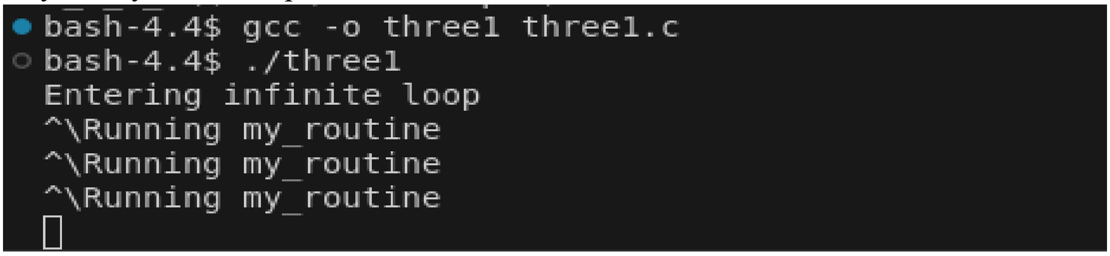
Because you're not binding the signal command to the my_routine function which makes it default to the normal function which is to terminate the process.

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?

It's not doing anything now. We are switching it to SIG_IGN which the manual says does this:
Ign Default action is to ignore the signal. This is the exact behavior we are seeing.



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?

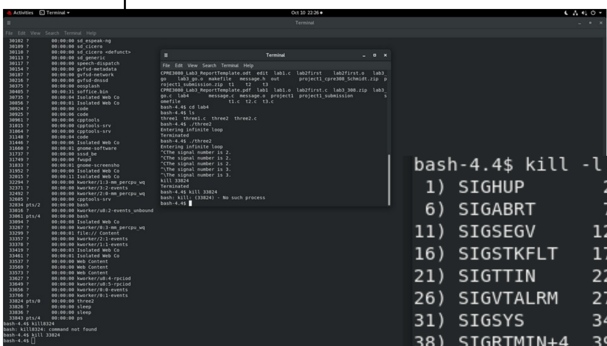


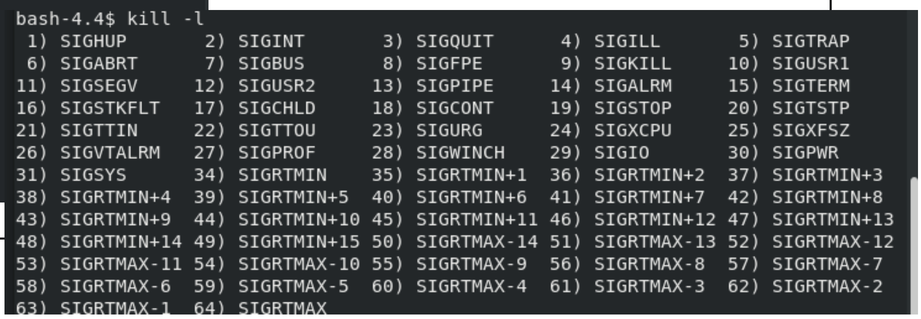
It print out Running my_routine now. This is because we are assigning the SIGQUIT signal with the my_routine() function instead of it's default terminate function. This is just like the first exercise we did except with the SIGQUIT signal instead.

3.2:

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

2 for ctl+C and 3 for ctl+\. You can see that SIGINT's number is 2 which was printed when ctl+C was pressed and SIGQUIT's number is 3 which was printed when ctl+\ was pressed.





3.3:

10 pts Include your source code

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

void my_routine();

int main()
{

   signal(SIGFPE, my_routine);

   int a = 4;
   a = a/0;

   return 0;

}

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 signal statement. The signal needs to be bound before the division by 0 happens otherwise it won't catch it.

3.4:

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

First arg is executable name, second is the message that will be printed once the alarm is done. The third is the time before the alarm prints out the message given.

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

The function alarm sets the timer for the SIGALRM signal. You give it an int that is the seconds it will wait before triggering the signal.

```
lab4 > C three4.c > ...
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>

char msg[100];
void my_alarm();
int main(int argc, char * argv[]){
    int time;
    if (argc < 3) {
        printf("not enough parameters\n");
        exit(1);
    }
    time = atoi(argv[2]);   // Converts third argument into int time
    strcpy(msg, argv[1]);   // Copying second argument into message, which will get printed by the alarm
    signal(SIGALRM, my_alarm);  // Alarm gets triggered by SIGALRM signal, SIGALARM gets triggered when the time for a specified time runs out
    alarm(time);                // This is specifying the time to wait before raising the signal, which is our third arg into program
    printf("Entering infinite loop\n");
    while (1)
    {
        sleep(10);
    }
    printf("Can't get here\n");
}
void my_alarm() {
    printf("%s\n", msg);
    exit(0);
}
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

bash-4.4$ ./three4 cheeseburger 7
Entering infinite loop
cheeseburger
bash-4.4$
```

3.5:

2pts Include the output from the program.

```
^CReturn value from fork = 0
Return value from fork = 36598
^CReturn value from fork = 0
Return value from fork = 36598
```

```
36584 ?       00:00:00 kworker/1:0-e
36597 pts/2   00:00:00 three5
36598 pts/2   00:00:00 three5
36673 ?       00:00:00 sleep
36679 pts/4   00:00:00 ps
ash-4.4$
```

2pt How many processes are running?

Two processes.

3pts Identify which process sent each message.

The parent process sent the 36598 since fork returns a valid pid to the parent and returns 0 to the child. So the 0 is the child.

3pts How many processes received signals?

Both the parent and the child processes.

3.6:

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

Two processes. The parent is in the write and the child is in the read.

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.

We create the buffer for the message size. We then create a pipe with two file descriptors one for reading and one for writing. We then create the child process with the fork call. The parents enters the if block and since we specify p[1] and write we want to send the message stored in msg to the end of the pipe. The child then specifies read with p[0] meaning it wants to get whatever has been written to the end of the pipe and puts it into inbuff. We then print out the inbuff.

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

So that the child won't try to read before the parent has wrote. We could use the wait() function instead to make sure that the child isn't trying to read before the parent has wrote.

3.7:

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

You use the shmget() function which calls a system call that allocates a chunk memory that can be accessed via a key that has been specified when it was created in this case 5768.

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

They can be in a race condition where they are both trying to access this shared memory at the same time.

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

This is because the client is the last to touch the memory and writes the star into the shared memory location.

```
bash-4.4$ ./client
Message read: abcdefghijklmnopqrstuvwxyz
Client done reading memory
bash-4.4$ ./client
Message read: *bcdefghijklmnopqrstuvwxyz
Client done reading memory
bash-4.4$
```

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?

The * is never outputted. This is because the memory section is destroyed after every run so the client is unable to run by itself adding the asterisk. The shmdt is used to detach from the memory location. The schmctl with the IPC_RMID is used to destroy the shared memory location.

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?

This is determined by the system specifically the MSGMNB constant. You can use the msgctl to find the limit on a particular system. You can change this manually with a function call.

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)?

If the message queue is not initialized then the default behavior of msgsnd() will block until the space becomes available. Otherwise if the IPC_NOWAIT flag is used the msgrcv will return with an error that there are no messages to read.

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.

They can be destroyed using the msgctl function with the IPC_RMID command. This command marks the message queue for deletion. The caller must have privileges or the the user ID must be the creator or owner of the message queue. Otherwise msgctl() will be ignored.

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

In linux they have binary and counting. Binary semaphores act like mutexes allowing only one process to access a resource at a time. There is also a counting semaphore that allows for multiple processes to access a limited number of resources. We create semaphores with semget. We intialize them with the semctl funtion. And we acquire the semaphores with the semop function.