Ethan Wieczorek  
CprE 308: Section #F  
Project 02  
Friday midnight

**IPCS**

**Unnamed Pipe**

**1pt) Output of pipe\_test.c**

My child asked "Are you my mummy?"

And then returned 42

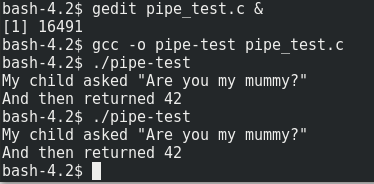
**1pt) What do you notice about the timing of the printing?**

It all prints at the same time.

**2pt) What happens when more than one process tries to write to a pipe at the same time? Be specific: using the number of bytes that each might be trying to write and how that effects what happens.**

**1pt) How does the output of pipe\_test.c change if you move the sleep statement from the child process before the fgets of the parent?**

Mine didn't change at all.



**1pt) What is the maximum size of a pipe in linux since kernel 2.6.11?**

pipe capacity is 65536 bytes since Linux 2.6.11

**Named Pipe (FIFO)**

**1pt) What happens when you run the echo command?**

The cat prints what was echoed through the pipe and then closes the pipe.

**1pt) What happens when you run the echo first and then the cat?**

The pipe stay open until the cat is run and grabs the info out of the pipe, then closes it.

**2pt) Look at the man page fifo(7). Where is the data that is sent through the FIFO stored?**

It is not stored in the file system, it is kept in the sending terminal until it is accessed by the other end of the pipe.

**2pt) Write a short program that uses named FIFO (mkfifo(3)) to print any line entered into the program on one terminal out on the other terminal.**

**Socket**

**2pt) What are the six types of sockets?**

SOCK\_STREAM

SOCK\_DGRAM

SOCK\_SEQPACKET

SOCK\_RAW

SOCK\_RDM

SOCK\_PACKET

**1pt) What are the two domains that can be used for local communications?**

AF\_UNIX, AF\_LOCAL

**Message Queues**

**1pt) What is the output of mq\_test1?**

Received message "I am the Doctor"

**1pt) What is the output of mq\_test2?**

Received message "I am the Master"

**2pt) Change mq\_test2.c to send a second message which reads “I am X” where “X” is your favorite companion. Change mq\_test1.c to wait for and print this second message before exiting.**



**Shared Memory Space**

**1pt) What is the output if you run both at the same time calling shm\_test1 first?**

a\_string = "I am a buffer in the shared memory area"

an\_array[] = {0, 1, 4, 9, 16}

a\_ptr = 140721045262976 = "I am a string allocated on main's stack!"

**1pt) What is the output if you run both at the same time calling shm\_test2 first?**

a\_string = "I am a buffer in the shared memory area"

an\_array[] = {42, 1, 4, 9, 16}

a\_ptr = 140724806487088 = "I am a string allocated on main's stack!"

**1pt) What if you run each by themselves?**

Test 1:

a\_string = "I am a buffer in the shared memory area"

an\_array[] = {0, 1, 4, 9, 16}

a\_ptr = 140731311120976 = "I am a string allocated on main's stack!"

Test 2:

a\_string = "I am a buffer in the shared memory area"

an\_array[] = {42, 1, 4, 9, 16}

Segmentation fault

**2pt) Why is shm\_test2 causing a segfault? How could this be fixed?**

The a\_ptr is not being set so it's trying to access a pointer to a variable that doesn't exist. We can initialize a\_ptr early on with zero and then have it be set later.

**1pt) What happens if the two applications both try to read and set a variable at the same time?**

It might not set properly, or it might read the information from before it was set.

**1pt) How can a shared memory space be deleted from the system?**

shm\_unlink

**2pt) Change the code to share some useful piece of information?**

**Unnamed Semaphores**

**2pt) What is the function call that would be needed to create an unnamed semaphore in a shared memory space called shared\_mem->my\_sem and assign it an intial value of 5?**

sem\_t my\_**sem = (sem\_t\***) shared\_mem;

sem\_init(&my\_sem,1,5);

**Named Semaphores**

**1pt) How long do semaphores last in the kernel?**

Until the system is shut down or the semaphore is unlinked.

**1pt) What causes them to be destroyed?**

The system shutting down or the sem\_unlink command.

**2pt) What is the basic process for creating and using named semaphores? (List the functions that would need to be called, and their order).**

sem\_open

sem\_init

sem\_post

sem\_wait

sem\_destroy

**Signals**

**1pt) What happens when you try to use CTRL+C to break out of the infinite loop?**

The signal handler catches the SIGKILL signal and calls my\_quit\_handler.

**1pt) What is the signal number that CTRL+C sends?**

2

**1pt) When a process forks, does the child still use the same signal handler?**

If you register the signal handler before the fork call, then the same signal handler will be used in both processes.

**1pt) How about during a exec call?**

execve() does not return on success, and the text, data, bss, and stack of the calling process are overwritten by that of the program loaded. The program invoked inherits the calling process's PID, and any open file descriptors that are not set to close on exec. Signals pending on the calling process are cleared. Any signals set to be caught by the calling process are reset to their default behaviour. The SIGCHLD signal (when set to SIG\_IGN) may or may not be reset to SIG\_DFL.

**5pt) Write two programs. One which will send a signal of number 42 to the other process. The other program should catch that signal and print out the message “I got the signal!”**

Code here

**Dynamically / Statically Linked Libraries**

**1pt) First output of lib\_test:**

./lib\_test: error while loading shared libraries: libhello.so: cannot open shared object file: No such file or directory

**1pt) Second output of lib\_test after exporting the library:**

Hello

World

World

World

i=42

**Project 2**

**If you worked with someone else – who was it?**

No one

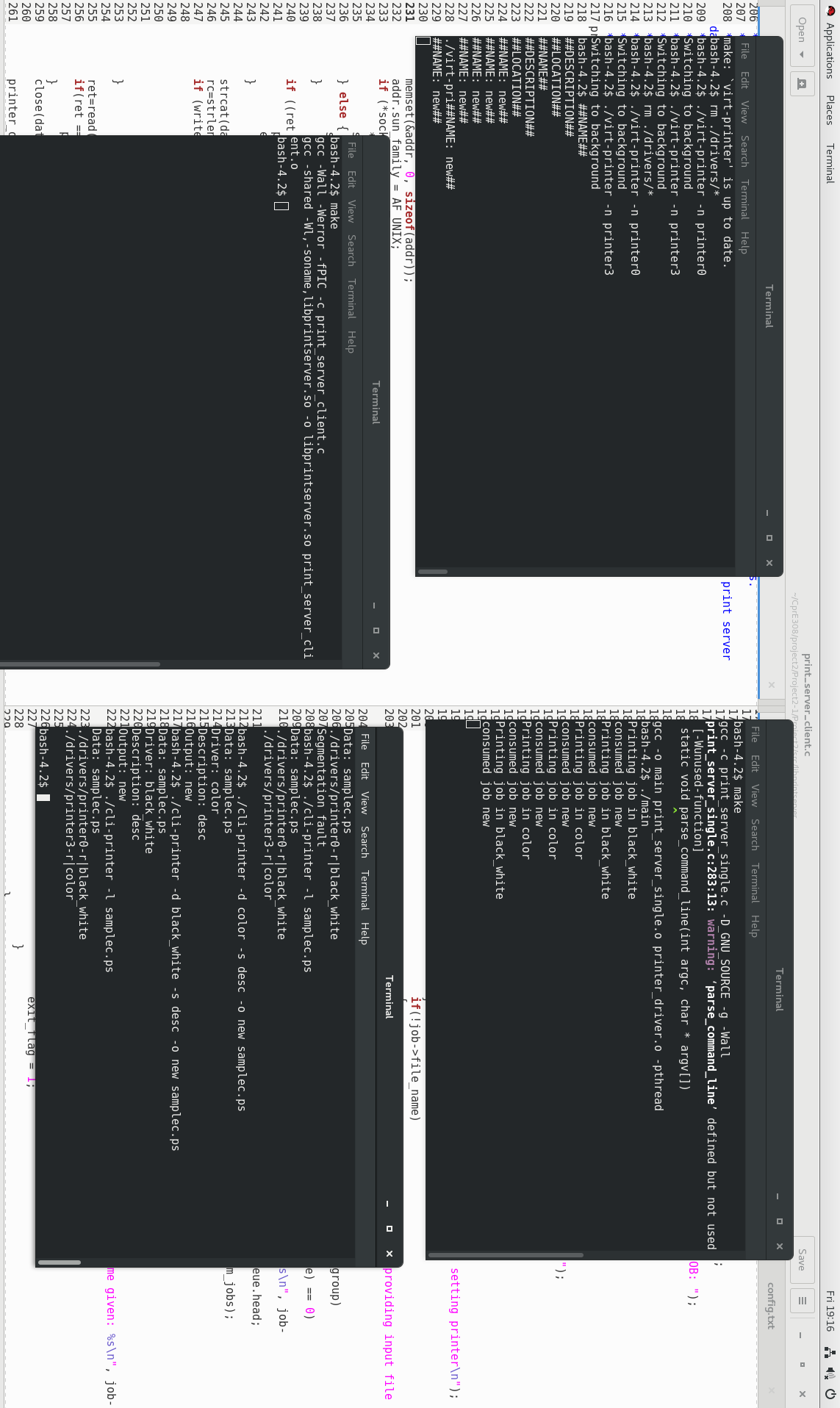
**5pt) Summary**

On the next page is a screenshot of everything working correctly. In this project we mainly focused on the use of sockets. The first task I accomplished was using an AF\_UNIX socket instead of a local host ip socket (which would have made this program much easier to accomplish). Using the provided materials and the unix(7) man page I was eventually able to create a program that could send my data across a socket file using a character buffer, and have the server read that information off of the socket file and parse through it correctly. I ran into issues here where the socket would stop reading if something wasn't written to the file fast enough (which using a localhost ip socket wouldn't have had a problem with) so I ended up programatically building the character buffer of everything I wanted to send and sending it in one write command. The server deals with requests by sitting waiting for a socket connection and the reading it when one comes in. It then check to see if the data in the connection started with “LIST\_DRIVERS” and if it did, it returns the list of drivers over the socket, if “LIST\_DRIVERS” wasn't present it continues onto the provided code from lab 4.

First of all, strtok has been deprecated for a number of years now in favor of strsep, which works much better. I replaced all of the provided code containing the strtok commands and replaced it with my own code that actually worked because the provided code was memory leaking and seg faulting. I was also able to fix the issue everyone had with lab 4 where the printer groups weren't being recognized by removing the newline character from the end of the printer groups that was kept on by strtok, and replacing it with the string null terminator character. The provided function “parse\_command\_line” was throwing errors so I replaced it with my own 4 lines of code to determined whether or not -d was in the the arguments and ran the code in daemon mode. There's also no provided information on how to parse the data for list printer drivers so I made a few assumptions in that function.

**If you did extra credit – tell us what the functionality and how to use it here:**

Fixed the provided code.



**How to run Project 2**

**Terminal 1: ./src/printer-server/printer**

rm /drivers/\*

make clean

make

./virt-printer –n printer0

./virt-printer –n printer3

**Terminal 2: ./src/printer-server**

make clean

make

./main -d

**Terminal 3: ./src/libprintserver**

make clean

make

**Terminal 4: ./src/cli-printer**

make clean

make

export LD\_LIBRARY\_PATH="../libprintserver/"

LIST FUNCTUION:

./cli-printer –l file\_name

PRINT FUNCTION

./cli-printer –d driver –s description –o output\_name file\_name

**Notes:**

Any other details needed to run the program

the socket file must be in the src folder.