Ben Williams
Lab 6
CprE 308

The purpose of this lab was to learn about and implement IPC (inter-process communication.) There are many different methods that users can take from the system to send messages through pipes, named pipes, named semaphores, shared memory, signals, and sockets to name a few. I have some background experience with sockets and I am excited to learn about other ways of sending messages back and forth between processes.

In your report and record the output of this program along with anything you notice about the timing of when things are printed.

My child asked "Are you my mummy?"
And then returned 42

The was a pause of about 2 seconds before anything happened

• 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.
If the pipe is full, then ***write****(2)* fails, with *errno* set to **EAGAIN**. Otherwise, from 1 to *n* bytes may be written (i.e., a "partial write" may occur; the caller should check the return value from ***write****(2)* to see how many bytes were actually written), and these bytes may be interleaved with writes by other processes.

• 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?
My child asked "Are you my mummy?"
And then returned 42
There was no pause between the two lines, they both waited for 2 seconds and then printed instanteously.

• What is maximum size of a pipe in linux since kernel 2.6.11?
65536 bytes.

• What happens when you run the echo command?
“hello fifo” is displayed in the second terminal.

• What happens if you run the echo first and the the cat?
The first terminal blocks until the second terminal reads in the data sent.

• Look at the man page fifo(7). Where is the data that is sent through the FIFO stored?
The kernel passes all data internally without writing it to the file system. I'm assuming it is either in the stack or heap depending on implementation.

• What are the six types of sockets?
SOCK_STREAM
Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data

transmission mechanism may be supported.

SOCK_DGRAM

Supports datagrams (connectionless, unreliable messages of a fixed maximum length).

SOCK_SEQPACKET

Provides a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer is required to read an entire packet with each input system call.

SOCK_RAW

Provides raw network protocol access.

SOCK_RDM

Provides a reliable datagram layer that does not guarantee ordering.

SOCK_PACKET

Obsolete and should not be used in new programs; see ***packet****(7)*.

• What are the two domains that can be used for local communications?

AF_UNIX, AF_LOCAL

• What is the output from each program

./mq_test1
Received message "I am the Doctor"
./mq_test2
Received message "I am the Master"

• What happens if you start them in the opposite order

./mq_test2
Received message "I am the Doctor"
Received message "I am the Master"
./mq_test1
Received message "I am Clara"

• 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. Include the output of these programs in your report.
Note: if you are unsure what we mean by companion just have it send “I am Rose”.

./mq_test1
Received message "I am Clara"
Received message "I am Nick Cage"
./mq_test2
Received message "I am the Doctor"
Received message "I am the Master"

What is the output if you run both at the same time calling shm_test1 first?

./shm_test1
a_string = "I am a buffer in the shared memory area"
an_array[] = {42, 1, 4, 9, 16}
a_ptr = 140735611795616 = "I am a string allocated on main's stack!"

./shm_test2
a_string = "I am a buffer in the shared memory area"
an_array[] = {42, 1, 4, 9, 16}
Segmentation fault

What is the output if you run both at the same time calling shm_test2 first?
./shm_test2
a_string = "I am a buffer in the shared memory area"
an_array[] = {0, 1, 4, 9, 16}
Segmentation fault
./shm_test1
a_string = "I am a buffer in the shared memory area"
an_array[] = {0, 1, 4, 9, 16}
a_ptr = 140733817182880 = "I am a string allocated on main's stack!"

What if you run each by themselves?
./shm_test1
a_string = "I am a buffer in the shared memory area"
an_array[] = {0, 1, 4, 9, 16}
a_ptr = 140734399254704 = "I am a string allocated on main's stack!"
./shm_test2
a_string = "I am a buffer in the shared memory area"
an_array[] = {42, 1, 4, 9, 16}
Segmentation fault

Why is shm_test2 causing a segfault? How could this be fixed?
It's because the pointer in a_ptr is pointing to space owned by shm_test1, not shm_test2. This can be fixed by using strncpy() instead of simply setting a pointer as well as allocating memory space for this string in the struct in shm_test.h

What happens if the two applications both try to read and set a variable at the same time? (e.g. shared_mem->count++).
If there is no memory locks in place, the final value could be unchanged, +1 instead of +2, or +2

How can a shared memory space be deleted from the system?
shm_unlink()

Convince yourself that you understand what is going on here, and if not, please ask questions. Then change the code to share some useful piece of information. Use your imagination for how this might be used. Include your new code in your write up.

```
//Set the message from a_ptr in shm_test1
char my_string[100];
printf("Enter a word: ");
scanf("%s", my_string);
strcpy(shared_mem->a_ptr, my_string);

//Change the value of an_array[0] in shm_test2
shared_mem->an_array[0] = 600;
```

In your lab report include 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.

```
int sem_init(shared_mem->my_sem, 1, 5);
```

• How long do semaphores last in the kernel?
They stick around until they are either specifically destroyed or the system shuts down.

• What causes them to be destroyed?
sem_unlink()

• 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() to name and create your semaphore/open a semaphore with the existing name
sem_post() to increment your semaphore
sem_wait() to decrement your semaphore
sem_close() to close
sem_unlink() to remove the semaphore from memory

What happens when you try to use CTRL+C to break out of the infinite loop?
It just says that damn phrase from Jurassic Park

What is the signal number that CTRL+C sends?
2

When a process forks, does the child still use the same signal handler?
It does

```
       A child created via fork(2) inherits a copy of its parent's signal
       dispositions.
```

How about during a exec call?
It does not

```
       During an execve(2), the dispositions of handled
       signals are reset to the default; the dispositions of ignored signals
       are left unchanged
```

Signal Catcher/Receiver

./signal_catcher
.....
Signal Recieved!

pgrep signal_catcher
3490
./signal_sender 3490

Now try to run the program ./lib_test and record the output in your lab report. It didn't work because the program didn't know where to find the library.

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

Output of lib_test
./lib_test
Hello
World
World
World
i=42

Output after recompiling Library (return 31)
./lib_test
Hello
World
World
World
i=31

Make sure that you answer ALL of the questions asked in this lab in your report. Also talk about what method you choose for the IPC of the last part and why you made this decision. Talk about any problems you ran into and how you solved those problems.

For the last part of the lab, I chose to use a combination of named pipes (FIFO), shared semaphores, and shared memory. I used shared memory and a shared semaphore for printer_print(), where I essentially had shared variables for every value passed between the processes and a semaphore used as a mutex.

Printer_print() works, but only on my test script inside of the libprintserver directory (server and client.) I created a PRINT_JOB_STRUCT struct and used that as the basis for my shared memory between the processes. I then opened my semaphore named “/driver_mutex”, updated the values in shared memory, and then posted to the semaphore. I then finally closed it and returned 0, assuming all went well.

On the server end of this, I had to first unlink() and then recreate the semaphore when the server first starts running. It then waits on the semaphore (initialized to 0), and when it does run it skims the values in stored memory and prints them out to the console (my lab 5 didn't work very well.) It then loops back to waiting on the semaphore. Since I am using the semaphore, all writes and read from the shared memory are atomic.

I was unable to successfully implement printer_list_drivers(). I managed to open a FIFO between the server and client processes, and notify the number of drivers I wanted back from the server. I was unable to figure out how to sequentially send data back from the server and get it into the client. It simply returned NULL for the data inside each driver returned.