

Deeksha Juneja

Cpre 308

Lab 6

3/8/2016

Introduction

This lab is about inter process communication through various methods like pipe, sockets, shared memory space and message queues. In this lab we are supposed to extend last week's lab to run as a daemon.

Lab Questions

Unnamed Pipe

1. What is the output of pipe_test.c?

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

The timing out the output was delayed by 2 seconds as specified in the program.

2. What happens when more than once 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?

Writes to a pipe less than the buffer size must be atomic, where as the writes to a pipe more than the buffer size may not be. This may cause the kernel to interleave the data with the data written by other processes.

3. 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?

No change.

4. What is maximum size of a pipe in linux since kernel 2.6.11?

65536 bytes

Named Pipe

1. What happens when you run the echo command?

"hello fifo" gets echoed to the other terminal because it's watching the fifo.

2. What happens if you run the echo first and then the cat?

It waits for cat to be executed.

3. 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. Thus, the FIFO special file has no contents on the filesystem; the filesystem entry merely serves as a reference point so that processes can access the pipe using a name in the filesystem.

Sockets

1. What are the six types of sockets?

- a. SOCK_STREAM
- b. SOCK_DGRAM
- c. SOCK_SEQPACKET
- d. SOCK_RAW
- e. SOCK_RDM

- f. SOCK_PACKET
2. What are the two domains that can be used for local communications?
 - a. AF_UNIX
 - b. AF_LOCAL

Message Queues

1. What is the output from each program?

```
[deeksha@co2046-11 ipc-types]$ gcc mq_test1.c -o mq_test1 -lrt
[deeksha@co2046-11 ipc-types]$ ./mq_test1
Received message "I am Clara"
[deeksha@co2046-11 ipc-types]$ gcc mq_test2.c -o mq_test2 -lrt
[deeksha@co2046-11 ipc-types]$ ./mq_test2
Received message "I am the Doctor"
Received message "I am the Master"
[deeksha@co2046-11 ipc-types]$
```

2. What happens if you start them in opposite order?
Same output
3. 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 sent "I am Rose".

```
[deeksha@co2046-11 ipc-types]$ ./mq_test2
Received message "I am the Doctor"
Received message "I am the Master"
[deeksha@co2046-11 ipc-types]$ gcc mq_test1.c -o mq_test1 -lrt
[deeksha@co2046-11 ipc-types]$ ./mq_test1
Received message "I am Clara"
Received message "I am Rosea"
```

Shared Memory Space

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

```
deeksha@co2046-11 ipc-types]$ ./shm_test1
_string = "I am a buffer in the shared memory area"
an_array[] = {42, 1, 4, 9, 16}
_ptr = 140726806454720 = "I am a string allocated on main's stack!"
deeksha@co2046-11 ipc-types]$
[deeksha@co2046-11 ipc-types]$ gcc shm_test2.c -o shm_test2 -lrt
[deeksha@co2046-11 ipc-types]$ ./shm_test2
a_string = "I am a buffer in the shared memory area"
an_array[] = {42, 1, 4, 9, 16}
Segmentation fault
[deeksha@co2046-11 ipc-types]$
```

2. What is the output if you run both at the same time calling shm_test2 first

```
[deeksha@co2046-11 ipc-types]$ ./shm_test2
a_string = "I am a buffer in the shared memory area"
an_array[] = {0, 1, 4, 9, 16}
Segmentation fault
[deeksha@co2046-11 ipc-types]$
[deeksha@co2046-11 ipc-types]$ ./shm_test1
a_string = "I am a buffer in the shared memory area"
an_array[] = {0, 1, 4, 9, 16}
a_ptr = 140723230344032 = "I am a string allocated on main's stack!"
[deeksha@co2046-11 ipc-types]$
```

3. What if you run each by themselves

```
[deeksha@co2046-11 ipc-types]$ ./shm_test2
a_string = "I am a buffer in the shared memory area"
an_array[] = {42, 1, 4, 9, 16}
Segmentation fault
[deeksha@co2046-11 ipc-types]$
```

```
[deeksha@co2046-11 ipc-types]$ ./shm_test1
a_string = "I am a buffer in the shared memory area"
an_array[] = {0, 1, 4, 9, 16}
a_ptr = 140724543254096 = "I am a string allocated on main's stack!"
```

4. Why is shm_test2 causing a segfault? How could this be fixed?
 Shared_mem->a_ptr is not declared
 This can be fixed by adding the line shared_mem->a_ptr = shared_mem->a_string; before sleep(5);
5. What happens if the two applications both try to read and set a variable at the same time?
 (eg. Shared_mem->count++)
 This would cause an error, we need to just put mutexes to protect from writing at the same time.
6. How can a shared memory space be deleted from the system?
 We can delete shared memory space through an ipcrm call.

Unnamed semaphores

1. Include the function call that would be needed to create an unnamed semaphore shared memory space
 Sem_init(sem,1,5)

Named Semaphore

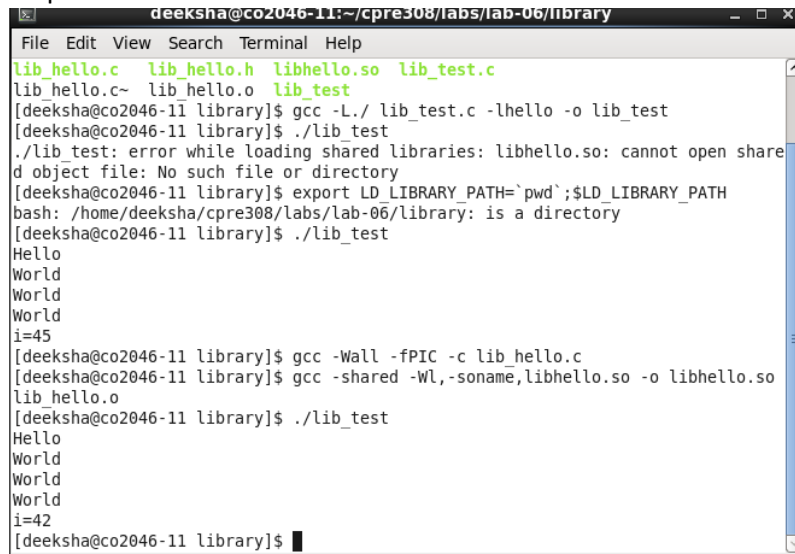
1. How long do semaphores last in the kernel?
 Unless destroyed using sem_unlink(), the semaphore will exist until the system is shut down.
2. What causes them to be destroyed?
 Sem_unlink() or/and sem_destroy()
3. What is the basic process for creating and using named semaphores? (list the functions that would need to be called, and their order).
 Sem_open()
 Sem_post()/sem_wait()
 Sem_close()
 Sem_unlink()

Signals

1. What happens when you try to use CTRL+C to break out of the infinite loop?
 "Ah Ah Ah, you didn't say the magic word"
2. What is the signal number that CTRL+C sends?
 SIGINT
3. When a process forks, does the child still use the same signal handler?
 Yes
4. How about during a exec call?
 No, it doesn't.

Dynamically/Statically Linked Libraries

Output



```
deeksha@co2046-11:~/cpre308/labs/lab-06/library
File Edit View Search Terminal Help
lib_hello.c lib_hello.h libhello.so lib_test.c
lib_hello.c~ lib_hello.o lib_test
[deeksha@co2046-11 library]$ gcc -L./ lib_test.c -lhello -o lib_test
[deeksha@co2046-11 library]$ ./lib_test
./lib_test: error while loading shared libraries: libhello.so: cannot open share
d object file: No such file or directory
[deeksha@co2046-11 library]$ export LD_LIBRARY_PATH=`pwd`;LD_LIBRARY_PATH
bash: /home/deeksha/cpre308/labs/lab-06/library: is a directory
[deeksha@co2046-11 library]$ ./lib_test
Hello
World
World
World
i=45
[deeksha@co2046-11 library]$ gcc -Wall -fPIC -c lib_hello.c
[deeksha@co2046-11 library]$ gcc -shared -Wl,-soname,libhello.so -o libhello.so
lib_hello.o
[deeksha@co2046-11 library]$ ./lib_test
Hello
World
World
World
i=42
[deeksha@co2046-11 library]$
```

Making the IPC

First I decided to go with IPC but then after a discussion with the TA, I realized that message queues were much more appreciate choice. Hence, I made my IPC with message queue. It was actually pretty straight forward to write the code for it. I didn't have too much trouble writing it. The man pages about some of the functions was really helpful. Rather, I found it harder to implement cli-printer. I wasn't exactly sure what was needed in it. I was able to implement all the requirements of cli-printer with some difficulty. The way I am doing my output file, it doesn't necessarily require me to give a name for it. My output file will take any name or extension given to it. If no name is give, it will just keep the name of the input file without any extension. Tags `-i` and `-o` can be used for giving input and output. I have also made a makefile to compile and create `.so` for `print_server_client.c`.

One of the major areas that I faced troubles with was that I actually didn't know that I was supposed to change `print_server_single.c`. It required quite a lot of change in its producer. I changed it so that it basically receives all the information about the print job and prints it. Another one of the biggest issues was testing my code. It took me extremely long to figure out how to compile the code. The compilation was done in various steps. Though, I think that was also the biggest learning point about the lab, that I learnt how to compile and execute big c project with multiple files.

Conclusion

This lab was easier than I expected it to be in terms of coding. Though, there were a lot of unexpected things that happened. My biggest problem came in because I couldn't compile my code immediately after writing it, so it took me longer to debug it. I wish the daemon was explained in more detail because I feel like I don't understand it properly. Though, I must say that the examples before were very help in understanding what was happening and actually helped me write the code. Overall, I think was very successful in completing the lab.