COP4600.001C11: Operating Systems
Project Four: Interprocess Communication

July 3, 2011

1 Description

In this project you will learn about interprocess communication using pipes using a simple model. You will write a program that reads its input from a named pipe, and also communicates bidirectionally with two other processes using pipes.

2 Specifications

I will provide you with a native `ELF` executable, called `commander`, which takes a single argument, the name of a *named* pipe (aka a *FIFO*). `commander` will attempt to create the named pipe and, if successful, will communicate with your `proj4` executable via this FIFO. It issues commands, one per line, with syntax

`command n: <command>`

, where `n` and `command` are variable, of course. Your compiled program, `proj4`, should do the following:

1. Accept two arguments: (1) The name of the FIFO to which `commander` writes, because `proj4` will need to read from it, and (2) The name of another file to use for this execution (I'll refer to this file as `tempfile` for the rest of the specs, but it will get its name from the second argument). Make sure you check the arguments as thoroughly as possible before passing them to functions in your code!

2. If the correct arguments are given, your program will print `begin` to stdout, and spawn two child processes: One for reading to `tempfile`, and one for writing to `tempfile`. The parent process must communicate with the child processes using *unnamed* pipes.

3. At this point, `commander` will generate a series of commands, each one either `read`, `write`, or `quit`. Your program will process commands given from `commander` until a `quit` command is encountered (which it will process, too, but as the final command).

4. If the command is a `write`, the writer child will append to `tempfile` on a newline a random integer between 1 and 100, inclusive.

5. If the command is a `read`, the reader child will read the next unread line (consisting of a single integer) from `tempfile`, pass it back to the parent, who then will print `read n` to stdout, where `n` was the integer read. If no new integers can be read, the reader child will pass -1 back to the parent process.

6. If the command is a `quit`, the parent will communicate to both children to quit. At this point the reader child must send to the parent all remaining unread integers from `tempfile` (if there are any), the parent will print them, then both children should terminate. The parent should then delete `tempfile`, print the total number of commands serviced to stdout, and then print `quit` to stdout before terminating.

Note that the parent process is the only process that ever prints to stdout. The `commander` program will delete the FIFO as its final act before termination, so your program should not attempt to remove it. Your program should test many things (number of arguments, existence of the FIFO to read, success of the `fork` calls, etc.), any one of which could fail. The usual project policies (regarding late submission, tokens, academic dishonesty, etc.) apply. You may not work with anyone else on this project.

3 Files

Download the associated `commander` executable from the assignment link in Blackboard. It will run on the c4 lab pcs, and probably on any major Linux

distribution. You should submit the following files, zipped into an archive named `lastname-firstname-proj3.zip`:

- `makefile`
- `proj4.c`

4 Testing

Your submission will be evaluated using a series of commands similar to the following:

```
$ make all

$ ../commander example
  /* process is executing, but blocked on FIFO */

$ ./proj4
  incorrect arguments

$ ./proj4 example
  incorrect arguments

$ ./proj4 example t t2
  incorrect arguments

$ ./proj4 example tempfile
  begin
  read 13
  read 42
  read -1
  read -1
  read 89
  read -1
  13 commands
  quit
```

5 Extra credit

You have two opportunities for extra credit on this project, and you may attempt either or both. I will give partial credit where appropriate. The weight designation corresponds to points on your final course grade.

1. **A report (1pt.)**: Create a report similar to the report for the last project, including all the sections it had. Answer the following questions in the report:

   (a) What communication models exist for interprocess communication? What communication models exist for interthread communication? Discuss as many differences between the multiprocess and multithreaded program models as you can. Address topics like communication, protection, and efficiency, among other considerations.

   (b) How does Linux implement named pipes and unnamed pipes? Specifically how do their implementations differ?

   (c) Explain in the context of IPC, processes, and file I/O what `bash` (or whatever Unix shell you're running) does with the following command

   ```
   $ cat proj4.c | grep include | tee out
   ```

2. **Interprocess synchronization (1pt.)**: The writer child can always proceed, but the reader child may receive a `read` command when there is nothing new to read. In that case the reader child would return -1 according to the original spec. Modify your code to synchronize the child processes so that the reader child will block on a `read` command until the writer child has written something new. If your solution requires IPC between the two child processes, you must use unnamed pipes.