CS 2204 Lab 13

your name here (please print):

your student ID number here:

Create a subdirectory called `lab13` under your home directory. Perform any necessary work for this lab assignment in that directory. Ensure that this assignment is performed from within a `bash` shell.

Our study of UNIX system programming thus far has concentrated on parent and child processes, and exploring all the intricacies of using the `fork()` system call. In this lab, we will round up our understanding of processes by learning one more concept called 'named pipes.' Recall that a pipe is a way to channel the output of one command as the input to another command. Think of a pipe as just that: a pipe where one program chucks stuff into one end and another program retrieves the stuff from another end. A named pipe is a pipe with a given name. We can create a named pipe, call it `mypipe`, and thereafter tell programs to specifically use `mypipe` to either dump stuff onto or read from. Unlike `fork` which creates processes, named pipes are a way to allow existing processes to talk to each other.

1. The `mkfifo` command creates a named pipe. Open a terminal and type the following commands in it:

   ```
   mkfifo mypipe
   ls -l > mypipe
   ```

   The first creates the named pipe which can henceforth be treated as if it is a regular file (but, of course, it is not actually a file). You will notice that the last command appears to 'hang,' or perhaps is waiting for something. Do not worry about this and instead open up a second terminal. In the second terminal, type:

   ```
   cat < mypipe
   ```

   As you can see, the named pipe has established a connection between the process in the first terminal and the one in the second terminal.

2. (2 points) Perform the commands in reverse, i.e., type the `cat` first and then go and do the `ls -l` in the other terminal. Explain what you observe.

3. (3 points) Let us write two new shell script programs, called `consumer` and `producer`. The `producer` is a shell script that repeatedly (i.e., infinitely) asks the user to type in a command and then passes on the command to a named pipe. The `consumer` is a different shell script that (infinitely) reads from the named pipe and executes the command just read. For instance, they work as follows:

   ```
   ./producer
   Please enter a command:
   ls -l
   Please enter a command:
   date
   ```

On another terminal window, we have (example):

```
./consumer
Received command: ls -l
total 28
-rwxrw-r--  1 ramakris ramakris 116 Nov 29 19:22 consumer
-rwxrw-r--  1 ramakris ramakris 109 Nov 29 19:21 producer
...
Received command: date
Tue Nov 29 19:24:48 EST 2005
```

4. (2 points) Open yet another terminal window and invoke a second `producer` process that communicates using the same pipe. Note what happens at the `consumer` terminal window as both `producer`s begin to bombard it with commands. What does this tell you about how named pipes work?

5. (2 points) Now create an additional `consumer` process in yet another terminal window. Go back to a `producer` window and type commands. What do you learn?

6. (1 point) What do you think the letters 'fifo' mean in `mkfifo`?