CSC209: Programming Assignment #4 (PA4)
Due: Thursday, August 9, 2018 @11:30PM

Sockets: For this problem you will implement a client-server version of the rock-paper-scissors-lizard-Spock game, which we will refer to as `rpsls`.

The game server, which you will implement in `rpsls_server.c`, will moderate one or more games of `rpsls` between two clients, which you will implement in `rpsls_client.c`. Three processes will be running when a game is being played (the server and two instances of the client).

rpsls_server.c: The server moderates zero or more games between two players (client processes). The server will create two stream sockets with two different port numbers (60000 and 60001, for example) and waits for requests to play a game from two clients. Once a connection has been made with two clients, the server sends a message to both players informing each of the two players names and asking for their hand gestures (one of rock, paper, scissors, lizard or Spock) for a game of `rpsls`; it then waits for each to send their gesture. Once both players send their gesture, the server decides who wins and informs both players who won that round and asks for their gestures for the next round. The game is repeated until one (or both) players wishes to stop playing. When no more games are to be played the server sends, to both players, the game statistics and then the server closes both sockets.

rpsls_client.c: A client creates a socket and tries to connect to the server. If it is unable to connect to the first port (perhaps the other client is already connected to that port) it tries with the second port number of the server. When a connection is made with the server, the client should send the player's name to the server (the client should ask the user for their at some point when it first runs). When asked for a gesture for a game, the client queries the user (to input a gesture from stdin) and then sends the gesture to the server. This is repeated until either the player wishes to stop playing (or the other player wishes to stop playing). When the server send the game statistics, the client displays the information to the user (stdout), closes its socket and terminates.

Here are some details for the implementation:

- server: the server will optionally accept a command line argument that will act as a shift for the port numbers to use. If no command line argument is given, it will try to use ports 60000 and 60001. If a command line argument is given, it will be an integer (call it n) and the sever will try to use ports 60000+n and 60001+n.
- client: the client will use one or two command line arguments. The first argument (not including the program name) will be the IP address of the server. The second, optional, argument will be the same number that was used when running the server (if one was provided) to specify which ports to try.
- client: after initially sending the player's name to the server, the client will only ever send one of 6 messages (for each game) to the server: ***`r`*** for rock, ***`p`*** for paper, ***`s`*** for scissors, ***`l`*** for lizard, ***`S`*** for Spock, or ***`e`*** for end.
- server: when the server receives ***`e`*** as a gesture from one of the players, it sends the game statistics to the players. That is, it sends the total number of games and how many each player has won.

- client: if the player enters bad data, it will ask the player to try again and enter something proper. It will not send bad data to the server.
- server: the server will have no output (to stdout or stderr) during its execution.
- client: when the client displays who won the previous game (using the information sent from the server), it should display a user-friendly message to **stdout** (for the player to read) and it should also a terse message to **stderr**. The message for stderr should simply be the word "win" or "lose", followed by a newlines. Each client should send the appropriate message to stderr (depending if their player won or lost).

Create an appropriate Makefile for your rpsls_server.c and rpsls_client.c programs. Your output to stdout during game play should be minimal

Processes and Signals: Implement a guessing game. When your program begins, it should fork into two processes: parent and child. The child process will interact with the user via stdin and stdout. It first prompts the user for a secret number between 0 and 1023 (inclusive). The parent process will try to guess the user's number by randomly picking a number between 0 and 1023 and displaying it to stdout. The user will then enter either "hi" or "lo", depending if the parent's guess is either too high or too low, respectively, compared to the secret that the child process knows (which it input from the user). The child will then send either SIGUSR1 or SIGUSR2 to the parent process to let it know if its guess is too high or too low, respectively. The parent then guesses a new number and everything repeats until the parent guesses the secret number. At this point, the user will enter "correct" and the child will send a SIGINT signal to the parent to indicate it has guessed correctly. The parent then displays the number of guesses it needed to guess the correct number and then sends a SIGINT signal to the child. When the child receives the SIGINT, it kills the parent process and then terminates itself.

Implement your program in `guess.c`. Each time a process sends a signal, it should also print a string to stderr with the format

```
senderPID # signal-name # receiverPID
```

where `senderPID` is the `pid` of the sending process, `receiverPID` is the `pid` of the receiving process, and signal-name is the name of the signal being sent.

Check-in the following files to your repository:

- rpsls_server.c
- rpsls_client.c
- guess.c
- Makefile