Machine Problem 1 (MP1)
Due 9PM Friday, February 7th

In this machine problem you are going to implement the game Mastermind game based on a client/server architecture on a Unix platform using C/C++. For more information regarding this game please refer to the following address:
http://en.wikipedia.org/wiki/Mastermind_(board_game)

Goals:

The goals of this MP are to get familiar with:

- Unix-based environments
- Makefile
- sockets API syntax
- Simple socket programming concepts
- Fork / pthread
- And how to correctly submit your MPs with our specified format!

Introduction:

Mastermind is a 2-player game in which one player secretly chooses a colored peg (with 6 choices of color) for each of 4 slots, and the other tries to guess the colors over 8 rounds of guessing. During each turn, the guesser specifies their guess of the color of each slot, and the other player returns some partial information about the guess. This information includes the number of correctly guessed slots, and the number of slots whose colors would have been correct in some other (not already correctly guessed) slot.

For example, if the secret assignment of “colors” was 2531, and the guess was 1552, the response would be “1 correct color+slot, 2 correct colors”: there's a 1 in the answer, but not in the 1st slot as guessed; the 2nd 5 is correct; and there's a 2 in the answer, but not in the 4th slot. Note that the 5 in the 3rd slot gets no response., because although there is a 5 in the answer, this guess already has 5 in the right slot.

Finally, multiple “right color, wrong slot” slots won't result in multiple responses if there's only one right slot of that color in the answer. For example, if the answer is 1222, and the guess is 0111, the response would be “1 correct color”, not 3.

The guesser gets 8 of these rounds to guess the exact assignment.

In this MP, the server acts as the color picker, and the client acts as the guesser.

Server program:

The server will be the side that picks the secret colors and responds to guesses. Before you get to any network communication, your server needs to know what colors it's using. The server will get this by reading stdin. The input should be either “random” or four digits 0-5, e.g. “3045”. In the first case, every new game session will come up with a random assignment. In the second case, every game session will use the specified assignment.

Your server must be able to carry on multiple game sessions with separate clients simultaneously, rather than delaying new connections until the current session is finished. To accomplish this, you'll want one thread of execution that listen()s for and accept()s new connections, and then splits off a new thread or process (your choice) to handle each new session. The server should listen() on the port given on the command line, as described in Notes.

Within a game session, the server should keep track of how many guesses the client has made, and reply to all guesses with the information described in Introduction. On an incorrect 8th guess, the server can close the connection after informing the client that they lost. The format of your programs' messages is up to you; we'll test your client and server only with each other.

Client program:

The client should connect to the hostname+TCP port provided on the command line, and then begin the first round of play. Each round, the client reads a guess on stdin (e.g. 1234), sends that guess to the server, waits for a reply, and then prints a response (you win, you lose, or the information described in the Introduction). After a win or loss, the client can close the connection and exit.

Notes:

- **IMPORTANT**: We will not accept submissions that can only run in the context of an IDE. Running your project shouldn't involve more than “make” and invoking the executables, as described below.
- **You must work alone**.
- You must use C or C++.
- You can and probably should build on MP0 code, and/or Beej’s examples, or any other desired guides available on the Internet. However, you MUST know exactly how your program works.
- For this project you are supposed to use a Makefile. An example is provided.
- Running “./server port” should run a server listening on “port”, e.g. “./server 1234”.
- Running “./client host port” should start a client that tries to connect to “host”, which could be a hostname or directly entered IP address, on port “port”. Examples:
  ./client 192.168.0.1 1234

  ./client remlnx.ews.illinois.edu 5678

  ./client localhost 1324
- Portability generally shouldn't be an issue, but if it does come up, your code only needs to work on the EWS Linux machines.
- Submit the assignment by committing your source files and any READMEs/other miscellaneous things to a directory called MP1 in your svn directory. **REMEMBER, IT'S NOT ON THE SERVER IF YOU DIDN'T RUN “svn ci” AFTER DOING “svn add” OR MAKING ANY CHANGES!**
- And the last, but not the least: Academic honesty plays a key role in our efforts to maintain a high standard of academic excellence and integrity. You are advised that ALL acts of intellectual dishonesty will be handled in accordance with the UIUC Academic Honesty and Student Conduct Policies.