Project 2

This project is to be completed by the whole class. It is up to the class to decide how to divide the work; however, the entire system must be working by the end of the term.

The finished product will be an online game that allows any number of people to join. The game is called Rock, Paper, Scissors, Lizard, Spock. Each round, each player gives a throw, indicating whether they are choosing Rock, Paper, Scissors, Lizard, or Spock. The pairwise combinations of throws have the outcomes listed in the table on the right.

For each round, each player is credited with the number of pairwise wins. For example, if Alice, Bill and Charlie throw Rock, Lizard and Spock, respectively, then they each get one point. If they throw Scissors, Lizard and Paper, respectively, then Alice gets two points, Bill gets one point, and Charlie gets no points.

The server should keep track of the lifetime points awarded to each player. Each player has a username and usernumber. Each client should remember the clients who have used that client. For known users, the client sends the usernumber to the server. For unknown users, the client sends the username to the server, and the server responds with usernumber, which may be an existing usernumber if that name is known, or may be a new usernumber.
Request Protocol
Each client request will consist of a class or struct with the following fields:

Kind of request:
  - Throw
  - Set User
  - New User
  - Query Wins
  - Query Standing
  - Query Names

Data:
  - Rock
  - Paper
  - Scissors
  - Lizard
  - Spock
  - Usernumber
  - Username

Result Protocol
Each server response will consist of a class or struct with the following fields:

Type
  - Move Registered
  - Move Not Registered
  - User Registered
    - Usernumber
  - Win Query Response
    - Number of Wins
  - Rank Query Response
    - Rank of Player
  - Names Query Result
    - Number of Players
    - Username
      - username
      - usernumber
      } for each current player

Game Protocol
Asynchronously, the server will send the following messages:

Round Started
  - Timestamp
Round Results
  - Number of Players
  - username
  - usernumber
  } for each current player
Once a round has started, the server will wait up to two minutes for all players to send a throw. Each throw receives a response. After two minutes, or all players who were registered when the game began have thrown, the server sends the results of the round to the players who registered a throw. The server follows this with a new round message. When the new round begins, it includes any players who registered once the previous round had started. Also, any players who did not send a throw for the previous round will be eliminated from the list of current players.

The server must remember all players, their names, numbers and number of wins. When a player begins using a client, the client will query the player’s win history from the server, and display a running total of wins as the rounds progress. When the player begins, the client should indicate the player’s current standing, and when the player wants to quit, the client should again indicate the player’s standing. The client does not have to indicate standing after each round, but should have a way for the player to ask, which in turn gets translated to a query to the server.

After each round, the client should indicate the other players, their throws, and wins, losses and ties against current player.

**Deliverables**
The code will be hosted on ix in a subversion repository at /home/users/jallen/svn/cis330f12. (See [http://systems.cs.uoregon.edu](http://systems.cs.uoregon.edu) for help with ix, including the subversion help under Server Tools). The class will meet in Deschutes 100 for the Friday, November 30 lab (4PM). At that time, the class will demo to project for the instructor. The project summaries are due at that time. Over the weekend, the instructor will check out the subversion project and look at the code. Each person in the class should turn in a project summary of the work they accomplished on the project. The general format is that each line is either a coding line or a group progress line. Each line details the particular programmer’s contribution to the project. A sample bit of summary would look like:

11/4 server class: stubbed out

11/5 server.handle_client_message(): fixed connect message bug about ipv6 addresses

11/6 protocol meeting: discussed problems with c/s protocol handling asynchronous server messages

Be sure to keep log of work you do on the project as you go. Reconstructing your work would be error-prone. Your project summary should be neat, but need not be typed.