CSE310 Spring 2016

Course project: Rock-Paper-Scissors

You are to implement the classic rock-paper-scissors game using Internet domain **stream** sockets. You shall implement a single-player game with the server as the opponent.

Your implementation consists of a client program and a server program. The client program allows a player to throw a shape and sends it to the server. The server starts each game, plays it by throwing a shape randomly, receives the client's move, determines and announces the game outcome.

Read its Wikipedia page if you do not know this game. You are to develop this project individually.

Building a single-player game

A single player plays the game of rock-paper-scissors with a server at a time. The server is started first. It creates a listening socket and binds to a port that is unique to you; use the scheme in Lab 2 Part IV to find the port number that is unique to you. The server then waits for player request and plays with the player on a separate connection as in the sample TCP server code on Slide 2-101.

Log-in - A player runs the client program to play the game. Upon starting the client program, it establishes a TCP connection with the server and allows the player to log in with his/her userid. This id is sent to the server, and printed on the server screen. A game is then immediately started by the server. At each move, the player can choose a shape or choose to log out.

Player statistics - A *player session* is defined as the time period between when a player logs in and when s/he logs out. The server maintains three counters per player session: the total number of games (denoted by G), the total number of wins by the player (denoted by W), and the total number of timed-out games (denoted by O). All three counters are initially zero.

Game start and time-out - When a game is started, the server announces it to the player, and throws one of rock, paper, and scissors at random. It then waits for the player's move. If no move is received within 5 seconds, it announces a "time out" to the player, increments G and O by 1 each, and starts a new game.

Log-out - If the server receives a reply within 5 seconds, it checks to see if the reply is a logout request. If so, it sends the values of G, W, and O to the player, and terminates the player session by closing the connection. It then continues to wait to play with other players.

A tie - If the reply is a valid shape, the server compares it with its own shape, and determines if it is a tie. If so, it randomly throws a shape again, and prompts the player to throw a shape again too.

If it is not a tie, the server determines the outcome of the game, announces the outcome as well as the shapes of both parties to the client, increments G by one, updates W according to game outcome, and starts a new game with the player immediately.

The client program takes two inputs via command line arguments when it is first started: (i) the host name of the machine on which the server program is running, and (ii) the port number that the server is listening at. Once the client program is started, say, by you, the following commands should be supported:

- help: this command takes no argument. It prints a list of supported commands, which are the ones in this list. For each command, the output includes a brief description of its function and the syntax of usage.

- login: this command takes one argument, your name. A player name is a userid or a nickname that uniquely identifies a player. It is entered with this command and sent to the server.
- rock, paper, and scissors: these three commands take no argument. They are used by the player to throw a rock, paper, and scissors respectively, which is sent to the server.
- logout: with no argument. Upon receiving this command, the client sends a logout request to the server, displays G, W, and O values that are returned from the server, terminates the current TCP connection with the server, and the client program exits.

Each command above invokes message exchanges between the client and the server except for the first command. The protocol for the communication between the client and the server, including the types of messages, the syntax and semantics of each type of message, and the actions taken when each type of message is sent and received at each party, needs to be designed and specified by you. You can refer to the HTTP protocol (RFC2616) and use formats similar to the HTTP request and response messages (Slides 2-26 to 2-31) for the types of messages that you define for this project. For example, you can use methods such as "LOGIN", "ROCK", and "PAPER", etc. in the method field of the message that is sent to the server. A command argument can be put in a subsequent field of the same message.

Each message sent to the server should be properly acknowledged by way of response messages. Status codes and status phrases similar to that in the HTTP response messages should be used. Some examples are 200 OK and 400 Bad Request, among others. The protocol specification that you define needs to be clearly stated in the written documentation described below.

For random number generation, see http://www.cs.stonybrook.edu/~ellenyliu/310/Project/rng.html

Bonus tasks

If you have finished the project and still have time, you may complete one or two bonus tasks below.

Bonus task 1. Persistence

You may add persistence of player statistics across multiple player sessions for each player at the server. The statistics of each player, in terms of total number of sessions, games, wins, and time-outs are cumulated and can be displayed upon request. A new client command should be added:

- stats: this command takes no argument. It is issued by the player. It is sent to the server. Upon receiving server reply, it prints the total number of sessions, games, wins, and time-outs that this player had since the game server is first started.

Bonus task 2. Multiple single-player games simultaneously

Multiple players/clients can access the server at the same time; each player plays a separate game with the server. The server maintains per-player statistics. Note that in this case, the player statistics database can be accessed by multiple client sessions at the same time. Some are writing to it while some others are reading it. You should handle such concurrent accesses properly. Describe your scheme to deal with this in your documentation.

Bonus task 3. A single dual-player game

Implement a single dual-player game of rock-paper-scissors. The server allows two players to login. It then starts each new game and announces it to both players. Both player throw a shape and sends to the server. The server determines if it is a tie or not, and inform both players accordingly.

If either player or both players time out on their moves, a new game is started after G and O for each player is updated. If either player is logged out, the other player is so informed, which can choose to wait for a subsequent player to login, or exit.

All client commands are identical to the single-player case. Include in your documentation any changes made with respect to the protocol used for communication between the client and server.

Development

You may implement the project using Java or Python. You should implement a command line interface similar to Linux command line for the client program. Your server program should print adequate messages on the standard output, to convince the TA that it has required functionalities. You program should run on the allv Linux machines that we used for in Labs 2 and 3.

Python stream socket programming was covered in Lab 2 as well as the textbook. For Java socket programming, a simple example is provided in the folder. You are also free to read online tutorials.

You are responsible to clean up any processes of yours that might be left around running as you develop, debug, and test your programs. Orphan processes consume extensive resources, slowing down both others' work and yours. Under Linux, to kill a process, type "`ps aux | grep your_id`", where `your_id` is your login id, to find the process id(s) of the orphan processes (in the second column of the `ps` command output), and type "`kill -9 pid`" to kill the process with process id `pid`.

What to submit

Submit the following two materials separately before the due date:

(a) Well-documented source code files, together with a README file.

- Each program should have proper program documentation in it. Program documentation is the comments that appear in the source code to aid in the understanding of the program. They tell what effect the code will have or to elucidate a complex statement. Do not reiterate in English what is obvious from the code such as "variable x is incremented".
- You can **only** submit the source code. No executable or object file is accepted. This means before you submit, you must make a clean submit directory that has only the required files in it. Name all Python program files with .py suffix and all Java program files with .java suffix.
- Submit a single .tar or .zip file instead of multiple files. Include in your README file instructions on how to obtain the source files from the archive or compressed file. This must be able to be performed in the lab environment as well.
- If you implemented bonus tasks, you may choose to submit one code that has both bonus and non-bonus functions, or submit multiple versions, one for the base, one for each bonus added.

(b) Written project documentation in a single file that includes the following sections:

- Overview: indicate how much of the project you have completed. A brief description of the major functionality in each part that you have implemented.
- User documentation: it contains everything that the user of your program (and the grader) needs to know to properly use the program to obtain desired results. This section does not describe how the program works, but only what it does and how to use it. You should

describe the syntax and parameters used to run your programs and the program output (including possible error messages).

- System documentation: system documentation is for people who need to know what is going on inside the program so they can perhaps change it or add new features. It describes the client server communication protocol that you defined for this application; it describes how concurrent operations are handled if applicable, major data structures and algorithms if any, where to find things, error conditions, what causes them and what happens, and how to compile and generate the executables. Try to be complete but concise.
- Testing documentation: include a list of testing scenarios to convince the grader that the program works. Explain what expected output and actual output are.
- The written documentation should be type-set. Only pdf is accepted.

How to submit

Both the source code archive file and the documentation file are to be submitted via Blackboard Assignment. Blackboard Assignment submission instructions are at: http://it.stonybrook.edu/help/kb/creating-and-managing-assignments-in-blackboard. You must read the submission instructions very carefully, and check to make sure that your project has been submitted correctly before the deadline. You can only submit once. Only click on "Submit" after you have checked and are certain that all requirements are followed.

Due date

The project due date is 23:55pm, Thursday, May 5. No late submissions will be accepted.

Tentative marking scheme

- 30% of marks will be allocated to the documentation, within which 20% will be on logically structured, well documented, and easy to understand code, 80% for a clean written documentation.
- 70% of marks will be allocated to the correct implementation of the specified functionality.

Total: 100pts (The three bonus tasks account for 4, 6, and 10pts extra respectively. Since the project is 15% of the course grade, the bonus tasks do not add too much to your course grade. You should place your priority on the final exam instead.)