**CS 375**

**Fall, 2021**

**Final Project**

**Due: Wednesday, December 8, 2021**

**The Problem: Rock, Paper, Scissors (50 points)**

We would like two programs (a referee and a player - there will be two players running at the same time) that interact over the Internet to play the "Rock, Paper, Scissors" game. In this game, two players secretly choose one item out of the set of rock, paper, and scissors. They then reveal their choice. A referee decides who wins as follows:

* Paper beats rock (by covering it).
* Rock beats scissors (by smashing it).
* Scissors beats paper (by cutting it).
* Matching choices are a draw.

The winning player gets a point. In a draw, no points are awarded.

The referee program (the "server") should create and bind a network socket with a random port chosen. The port number should be displayed. Then the referee should wait for two players to connect and then send data to the players indicating that the game may begin.

The player program (the "client") should take a server machine name and a port number as command-line arguments. It should create a network socket and connect to the referee process. (If there is no referee process to connect to, the program should exit with an error message.) A player then should prompt the local user to enter a choice. Proper input validation should be done to make sure the player program never sends an invalid choice to the referee. The choice is sent to the referee. After the referee receives two choices, it should decide which player has won the round and send information about the round back to both players so that they may display the results to the local user. The referee may assume that all data it is given is valid.

The system must implement the following network protocol. The player programs must send the string "READY" to the referee when they are ready to make a choice. The referee should then respond to each player with the string "GO". Thereafter, the players alternate sending an item choice and receiving the results of the round. (The format of this information is your choice.) When a user wishes to exit, the player program should send the "STOP" string to the referee. The referee should respond with the string "STOP" to both players followed by summary game information (number of rounds won by each player, etc., again the format of this information is your choice). The players should display this information to the local user and then exit. The referee should return to waiting for the next pair of player connections. (I.e., the referee process is intended to run indefinitely.)

**Assignment**

Write programs named **referee** and **player** that implement the referee and player programs. These programs can be written in any language as long as they use network sockets and can be run on a Linux machine. Here is an example of possible referee and player sessions running on (fictitious) machines named newton, galileo, and einstein. (User input is shown in bold.)

|  |  |  |
| --- | --- | --- |
| On newton | On galileo | On einstein |
| $ **./referee**  Referee is using port 2345  Referee is waiting for players  Player 1 has connected  Player 2 has connected  P1 choice: Rock  P2 choice: Scissors  P1 wins  P1 choice: Rock  P2 choice: Exit  Game has ended Referee is waiting for players | $ **./player newton** 2345  You are player 1  0: Exit  1: Rock  2: Paper  3: Scissors  Enter Choice: **1**  Round 1:  Player 1: Rock  Player 2: Scissors  You win!  0: Exit  1: Rock  2: Paper  3: Scissors  Enter Choice: **1**  Game has ended  Final Score  Player 1: 1  Player 2: 0  $ | $ **./player newton** 2345  You are player 2.  0: Exit  1: Rock  2: Paper  3: Scissors  Enter Choice: **3**  Round 1:  Player 1: Rock  Player 2: Scissors  Sorry, you lose  0: Exit  1: Rock  2: Paper  3: Scissors  Enter Choice: **0**  Game has ended  Final Score  Player 1: 1  Player 2: 0  $ |

Ideally, we would like to test the project using three different machines. In all likelihood, testing will have to be done with all processes on the same machine run in different terminal windows. Use **localhost** as the machine name for this type of testing. Nevertheless, the programs still must be implemented using network sockets (AF\_INET and PF\_INET). Programs using local UNIX sockets (AF\_UNIX and PF\_UNIX) will be accepted for minimal credit.

**Extra Credit** (10 points, only if the first part is complete) We would like a player program with a GUI. The layout and interaction is your design, but at a minimum it should have the following functionality:

* It should allow the user to give the machine name and port number of a referee process and connect to it using a connect button.
* It should allow the user to indicate their item choice via mouse click (i.e., the user should not need to type the item choice), then send it to the referee using a send button.
* It should display the referee result.
* It should have a quit button that causes the player program to display the final results and require another button click to actually quit the program.

Note that the GUI player program must follow the same network protocol as the text-only player program in Part 1. That is, there is only one referee program, and it will referee games between any two player programs whether they are text-only or have a GUI.

**Extra Credit Assignment**

Write a C/C++ program named gui-player that implements the GUI player. This program must use the GTK 3 toolkit. This program must run on Ubuntu 18.04LTS or later.

What to submit

* Provide a makefile named Makefile that will make all programs (referee, player, and maybe gui-player) for this assignment as the default target (typically called all). Each program must be a separate target.
* Provide a README textfile with instructions on how to user your programs.
* Create a tarfile or zipfile containing the program source files, the makefile, and the README file.
* Submit your archive on Blackboard.