Lab 9: Sockets

CSE/IT 107

NMT Computer Science

"Today, most software exists not to solve a problem, but to interface with other software."

— Ian O. Angell

"Computer Science is no more about computers than astronomy is about telescopes."

— Edsger W. Dijkstra

"If people never did silly things, nothing intelligent would ever get done."

— Ludwig Wittgenstein

---

1 Introduction

2 Sockets

3 Exercises

Boilerplate

Remember that this lab *must* use the boilerplate syntax introduced in Lab 5.

rps.py Write a program that connects to the server running at 104.131.56.87 port 50001 and plays a game of rock, paper, scissors. The server will send the following messages:

name Next message sent will be your client's display name.

taken The name sent is already in use. Repeat sending a name.

wait Game has not yet been found (waiting for another player). No response required.

opponent <name> A game has been found. The opponent's name will be inserted for "<name>". No response required.

play The next message sent should consist solely of "r", "p", or "s", depending on whether you wish to play rock, paper, or scissors.

tie Your opponent played the same as you, causing a tie. No response required.

win Your play beat your opponent's, so you won. The next message should consist solely of "y" or "n", indicating your desire to play again.

lose Your opponent's play beat yours, so you lost. The next message should consist solely of "y" or "n", indicating your desire to play again.

disconnect Your opponent disconnected at an unexpected time. The next message should consist solely of "y" or "n", indicating your desire to find a new opponent.

again The next message should consist solely of "y" or "n", indicating your desire to play again. This message will only occur if invalid input is given for "win", "lose", or "disconnect", so if you can be sure that will not occur you do not necessarily need this case.

Every message sent will be terminated with a "\n". Every message you send should be terminated with a "\n". It is possible when you read from a socket that you receive multiple commands at once, or less than a full command. Check for "\n" to know where each command ends. If you receive part of a command, you will have to store that partial command until you get the rest of it.

For each of these server responses, you need to display an appropriate message with a prompt for input if appropriate. For example, a "win" message might output "Congratulations, you beat <otherplayer>! Do you want to play again?"

4 Submitting

Files to submit:

- rps.py (Section 3)

You may submit your code as either a tarball (instructions below) or as a .zip file. Either one should contain all files used in the exercises for this lab. The submitted file should be named either `cse107_firstname_lastname_lab9.zip` or `cse107_firstname_lastname_lab9.tar.gz` depending on which method you used.

For Windows, use a tool you like to create a `.zip` file. The TCC computers should have `7z` installed. For Linux, look at lab 1 for instructions on how to create a tarball or use the "Archive Manager" graphical tool.

Upload your tarball or .zip file to Canvas.