Computer Science 367

Program 3 (50 points)
Due Wednesday, March 11th, 2015 at 10:00 PM

Read all of the instructions. Late work will not be accepted.

Overview

In our final network programming assignment, you will implement the server and two distinct types of client for a networked trivia game. Between February 18th and February 25th, your class collectively designed the protocol for this assignment, including the roles of the server and each kind of client. The goal for the protocol design period was to create a protocol so clearly-defined that all students in the class could program to it and have interoperable programs. If at any time during development you come across something that you feel is sufficiently ambiguous that it might break interoperability, please post your concerns immediately on the class discussion thread so that you and your classmates can refine the protocol as needed.

Unlike past assignments, this assignment can be done individually or with a pair. Based on the preferences provided on an earlier survey, I have paired some students and left others individual. Look at Canvas' People page to check if you are part of a Program 3 group.

You (along with your assignment partner, if you have one) are responsible for implementing (in the C programming language) the server and both clients for this program, using sockets. It must be developed in your Subversion version control repository. If working in a pair, version the files in one of the two partners' Subversion repositories.

If working in a pair, please review the pair programming guidelines.

Pair Programming[1]

Pair programming is a software development technique where two programmers work together in front of one keyboard. One partner types code while the other is suggesting and/or reviewing every line of code as it is being typed. The person typing is called the driver. The person reviewing and/or suggesting code is called the navigator. The two programmers should switch roles frequently (e.g. every 20 minutes). For this to be a successful technique the team needs to start with a good program design so they are on the same page when it is finally time to start typing on the computer. **No designing or programming is to be done without both partners present!** Pair programming has been shown to increase productivity in industry and may well increase yours, but there are additional reasons it is being used in this class. First, it is a means to increase collaboration, which is something department graduates now working in industry report that they wish they had more experience with. Second, working in pairs is a good teaching tool. Inevitably, in each pairing the partners will have different styles and abilities (for instance, one person may be better at seeing the big picture while the other is better at finding detailed bugs or one person might like to code on paper first while the other likes to type it in and try it out). Because of that you will have to learn to adjust to another person's style and ideally you will meet each other half way when there are

[1]These guidelines are based on a previous version developed by Perry Fizzano.

differences in approach. It's important that each person completely understands the program and so both parties need to be assertive. Be sure to explain your ideas carefully and ask questions when you are confused. Also it is crucial that you be patient! There is plenty of time allotted to complete this assignment as long as you proceed at a steady pace. Ask for help from me or the department tutors if you need it.

You will be assigned a partner by me via Canvas groups. Because there is no lab, you will need coordinate with your partner to find times when you can both be present. You will need to contact me ASAP if there is any reason you will not be able to collaborate. **Let me stress again that no designing or programming is to be done without both partners present! If I determine this happened I will fail you and your partner for this assignment.**

Program 3 Specifications

The protocol itself was decided by the class and has already been provided to you. Below are some additional constraints.

File and Directory Naming Requirements

All files should be contained in a `prog3` directory in your repository. Spacing, spelling and capitalization matter.

- The writeup and all of your source code should be found directly in `yourWorkingCopy/prog3`, where `yourWorkingCopy` is replaced with the full path of your working copy.
- Your participant client source code should be contained in a file named `prog3_participant.c`. If you create a corresponding header file you should name it `prog3_participant.h`.
- Your observer client source code should be contained in a file named `prog3_observer.c`. If you create a corresponding header file you should name it `prog3_observer.h`.
- Your server source code should be contained in a file named `prog3_server.c`. If you create a corresponding header file you should name it `prog3_server.h`.
- Your writeup must be a plain-text file named `writeup.txt`.

The server should take exactly two command-line arguments:

1. The port on which your server will listen for participants.
2. The port on which your server will listen for observers.

An example command to start the server is:

```
./prog3_server 36798 36799
```

Either client should take exactly two command-line arguments:

1. The name or address of the server (e.g. `linux.cs.wwu.edu` or a 32 bit integer in dotted-decimal notation)
2. The port on which the server is running, a 16-bit unsigned integer

An example command to run the participant client is:

```
./prog3_participant 127.0.0.1 36798
```

An example command to run the observer client is:

```
./prog3_observer 127.0.0.1 36799
```

Compilation

Your code should be able to compile cleanly with the following commands on CF 416 lab machines:

```
gcc -o prog3_server prog3_server.c
gcc -o prog3_observer prog3_observer.c
gcc -o prog3_participant prog3_participant.c
```

If you wish to use a different C standard (e.g. one of `c99`, `c11`, `gnu99`, `gnu11`), then you must create and add to your repository a file named `standard` that includes only one of the four above standard names.

Protocol Specification

For the protocol specification, see the official document posted on Canvas.

Grading

Submitting your work

When the clock strikes 10 PM on the due date, a script will automatically check out the latest committed version of your assignment. **(Do not forget to add and commit the work you want submitted before the due date!)** The repository should have in it, at the least:

- `prog3_observer.c`, `prog3_participant.c` and `prog3_server.c`
- Your write-up: `writeup.txt`
- Optionally, your header files

Your repository need not and **should not contain your compiled binaries / object files.** Upon checking out your files, I will compile all three of your programs, run them in a series of test cases, analyze your code, and read your documentation.

Points

By default, you will begin with 50 points. Issues with your code or submission will cause you to lose points, including (but not limited to) the following issues:

- Lose fewer points:
  - Not including a writeup or providing a overly brief writeup
  - Poor code style (inconsistent formatting, incomprehensible naming schemes, etc.)
  - Files and/or directories that are misnamed
  - Insufficient versioning in Subversion

- Lose a moderate to large amount of points:
  - Not including one of the required source code (`.c`) files
  - A server that polls (i.e. continuously spins in a loop of non-blocking calls)
  - Code that does not compile
  - Code that generates runtime errors
  - A client or server that does not take the correct arguments
  - A client or server that does not follow the specified protocol

Write-Up

You need to create, add and commit a *plaintext*[2] document named `writeup.txt`. In it, you should include the following (numbered) sections.

1. Your name, and if working in a pair, the name of your partner
2. Declare/discuss any aspects of your client or server code that are not working. What are your intuitions about why things are not working? What issues you already tried and ruled out? Given more time, what would you try next? Detailed answers here are critical to getting partial credit for malfunctioning programs.
3. In a few sentences, describe how you tested that your code was working.
4. What was the most challenging aspect of this assignment, and why?
5. What variant/extension of this assignment would you like to try (e.g. a variant that is more powerful, more interesting, etc.)

Academic Honesty

To remind you: you must not share code with anyone other than your professor (and if working in a pair, your assigned partner): you must not look at any one else's code or show anyone else your code. You cannot take, in part or in whole, any code from any outside source, including the Internet, nor can you post your code to it. If you need help from anyone, all involved parties *must* step away from the computer and *discuss* strategies and approaches - never code specifics. I am available for help during office hours. I am also available via email (do not wait until the last minute to email). If you participate in academic dishonesty, you will fail the course.

[2]E.g. created with vim, kate or gedit.