Computer Science 367

(Pair) Program 2 (50 points)

Checkpoint due Friday, February 2nd, 2018 at 10:00 PM
Program due Tuesday, February 20th, 2018 at 10:00 PM

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

Overview

For this network programming assignment you and your assigned partner will implement a client and a server for turn-taking, two-player word game. Your server should be able to support an arbitrary number of pairs of clients, each playing independent games simultaneously. Beyond the networking skills you developed in Program 1, this program has several new challenges:

- Handling games involving pairs of clients
- Using time limits (setting socket timeout values)
- More complicated network session / game state
- Richer game logic

You and your partner are responsible for implementing (in the C programming language) both the client and the server for this game, using sockets. It will be developed in either your or your partner's Github version control repository. You will work together using *pair programming*, described in the next section.

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. Note that the navigator need only lead the development (e.g. "now we need an if-statement to check if x is greater than zero...") and **does not** need to "speak" code (e.g. "open curly brace enter tab i assigned 0 semicolon enter..."). 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).

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

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 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 2 Specifications

Your client and server must be compliant with all of the following specifications in order to be considered correct. Non-compliance will result in penalties.

File and Directory Naming Requirements

Pick one of the two partners to host `prog2` in their repository. The other partner should *not* add any `prog2` directory. The following instructions apply to the partner hosting `prog2`. Spacing, spelling and capitalization matter.

- The writeup and all of your source code should be found directly in `yourWorkingCopy/prog2`, where `yourWorkingCopy` is replaced with the full path of your working copy.
- Your client source code should be contained in a file named `prog2_client.c`. If you create a corresponding header file you must name it `prog2_client.h`.
- Your server source code should be contained in a file named `prog2_server.c`. If you create a corresponding header file you must name it `prog2_server.h`.
- If you wish to have a shared header used by both the client and server, name it `prog2.h`.
- Your writeup must be a plain-text file named `writeup.txt`.
- Your plan must be a plain-text file named `plan.txt`.

Command-Line Specification

The server should take exactly four command-line arguments:

1. The port on which the server will run, a 16-bit unsigned integer
2. The "board size" (a one byte unsigned integer)
3. The "seconds per round" (a one byte unsigned integer)
4. The path to the word dictionary (one will be provided to you)

An example command to start the server is:

```
./prog2_server 36799 8 30 twl06.txt
```

The client should take exactly two command-line arguments:

1. The address of the server (e.g. 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 client is:

```
./prog2_client 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 prog2_server prog2_server.c trie.c
gcc -o prog2_client prog2_client.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

The protocol for this program is somewhat involved, please read the following description carefully. A game consists of three to five rounds (first player to win three rounds wins the game); each round consists of an arbitrary number of turns (turns continue until one player makes a mistake). Sample output for two clients is included later in this document; you should model your client's output formatting after these samples.

Connection Setup (S)

S When a client connects, the server immediately sends three messages to the client

S.1 A character indicating whether it is player 1 ('1') or player 2 ('2')

S.2 A `uint8_t` indicating the number of letters on the "board"

S.3 A `uint8_t` indicating the number of seconds you have per turn

The client should display all three pieces of information for the user.

Game (G)

G.1 The game begins as soon as the second player joins

G.2 The server should be able to handle an arbitrary number of games simultaneously

G.3 The game consists of a series of rounds, and continues until one player wins three rounds

G.4 If at any point either client unexpectedly disconnects from the server, the game ends prematurely; the server should then immediately close the connection with both clients without notice and end the process playing that game. If a client is ever unexpectedly disconnected from the server, this means the game has ended prematurely.

Round (R). Each round is as follows:

R.1 The server sends both players' scores to each player. Each score is a `uint8_t`. Player 1's score is sent first, followed by Player 2's. *If either player has three points, the game is over, and the client should print whether the user won (or lost), close the socket and exit. If neither player has three points, each client should print the score: your score, dash, opponent score.*

R.2 The server sends the round number as a `uint8_t` to both players. *Each client should print the round number for the user to see.*

R.3 The server randomly generates K characters (these characters are referred to as the "board"). If none of the first $(K-1)$ characters are in the set {a,e,i,o,u} (i.e. vowels) then the server should continue to randomly draw the last character is a vowel

R.4 The server sends the board to both players (only the board characters: no padding or null terminator). *Each client should print the board.*

R.5 The round then consists of a series of turns.

R.6 The round ends when a player makes a mistake on his or her turn.

Turn (T). Each turn is as follows:

T.1 The server then sends the character 'Y' to the player whose turn it is (the "active player"), and 'N' to the other player.

- **T.1.1** Odd numbered rounds begin with player 1's turn; even numbered rounds begin with player 2's turn.
- **T.1.2** Subsequent turns within a round alternate between the two players.

The active player client should print a message to the user indicating that it is his or her turn, while the inactive player client should print a message asking the user to wait.

T.2 The active player has N seconds to send a word to the server. The format of the message is a `uint8_t` indicating the word length, followed by the word as a character sequence (do not send any padding or null terminator).

T.3 For the server to count it as valid, the word must

- **T.3.1** Be a valid word in the server's word dictionary, and
- **T.3.2** Have not been used already in this round, and
- **T.3.3** Use only letters on the board (and no character may appear more times in the word than it appears on the board).

T.4 If the word received is valid, then

- **T.4.1** The server will send the `uint8_t` 1 to the active player

T.4.2 The server will convey the word to the inactive player (first by sending a `uint8_t` indicating the word length, and then sending the word).

T.4.3 Go to step T.1.

T.5 If the word received is invalid OR no word is received within the time limit, then

T.5.1 The server send `uint8_t` 0 to both players, and

T.5.2 The round is over, and

T.5.3 The inactive player's score is incremented by 1, and

T.5.4 The round number is incremented by 1, and

T.5.5 Go to step R.1

The Repository

- All code for this program will be developed under a Github repository. This repository has already been created for you at the following location:

  `https://github.com/hutchteaching/201810_csci367_username`

  where `username` is replaced by your *WWU* username.
- As noted above, your work will be done in a `prog2` sub-directory of your working copy. Therefore, exactly once at the beginning of working on Program 2, either you or your partner will need to create this directory. That person will also need to add your writeup and every source code file to your repository. **If you do not add, commit and push your files, I cannot see them, and thus cannot give you any points for them.** You are responsible for checking the contents of your github repository at the URL listed above (you can browse it on the web) to confirm that all of your files are there and contain your latest version.
- You must actively use git during the development of your program. If you do not have at least **6** commits for Program 2, points will be deducted.
- See the Git Guide on Canvas for some details on git, and feel free to drop in to office hours with your questions.
- *Note: you must never edit files in your repository directly. All interaction will be indirect, via your working copy.*

Plan

Prior to the checkpoint, you and your partner will need to work together to construct a plan for the program, documented in a plaintext file named `plan.txt`. Using proper spelling[2] and grammar, the plan should include the following numbered sections:

1. A one paragraph summary of the program in your own words. What is being asked of you? What will you implement in this assignment?
2. Your thoughts (a few sentences) on what you anticipate being the most challenging aspect of the assignment.
3. A list of at least three resources you plan to draw from if you get stuck on something.

[2]Hint: aspell -c plan.txt

4. Which open-source trie implementation, if any, you plan to use.
5. Your plan for meeting to finish this program (e.g. "every Monday, Wednesday and Friday from 3pm-5pm until it's done.).

Checkpoint

Shortly after the assignment is posted (deadline listed at the beginning of this document), you will have a checkpoint to make sure you are on track. To satisfy the checkpoint, you need to do the following:

- Identify the partner who will host the repository
- Create your `prog2` directory
- Create, add, commit and push the following files:
  - `prog2_client.c` (can be empty for now)
  - `prog2_server.c` (can be empty for now)
  - `trie.c` and `trie.h` (can be empty for now)
  - If you plan to use them, `prog2_server.c`, `prog2_server.h` and/or `prog2.h`
  - `writeup.txt` (can be empty for now)
  - `plan.txt` (the plan in the previous section)

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. I will grade your or your partner's files, whoever has a `prog2` directory. **(Do not forget to add, commit and push the work you want submitted before the due date!)** The repository should have in it, at the least:

- `prog2_client.c` and `prog2_server.c`
- `trie.c`
- Your plan: `plan.txt` (from the checkpoint)
- Your write-up: `writeup.txt`
- Optionally, the header files `prog2_client.h`, `prog2_server.h` and/or `prog2.h`

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

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:
  - Issues with the checkpoint (e.g. no plan, misnamed files or directories)
  - Not including a writeup or providing an overly brief writeup
  - Minor code style issues (inconsistent formatting, etc.)

  - Files and/or directories that are misnamed
  - Insufficient versioning in Github

- Lose a moderate to large amount of points:
  - Not including one of the required source code (`.c`) files
  - Server cannot handle multiple clients concurrently
  - 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
  - Major code style issues (incomprehensible naming schemes, poor organization, etc.)

Write-Up

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

1. Your name and your partner's name
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?

Assignment Details

Trie Implementation

You need a set or dictionary data structure, both to store the set of valid dictionary words, and to keep track of the words already used in the round. A *trie* is a natural data structure for this. For this particular program, I am giving you permission to use any publicly available trie implementation, on the condition that you clearly cite the source in a comment at the top of the file. Your trie implementation should be named `trie.c` (with an optional `trie.h` header file). The code you use must not implement any functionality beyond the basic operations on tries (add, delete, make empty trie, clear trie, check if key is in trie). You must make sure that your trie implementation frees up memory appropriately when it is no longer needed, so there are not any *memory leaks* in the code.

Sample Output

Player 1's output:

```
You are Player 1... the game will begin when Player 2 joins...
Board size: 8
Seconds per turn: 30
```

```
Round 1...
Score is 0-0
Board: e x w k i p c u
Your turn, enter word: wick
Valid word!
Please wait for opponent to enter word...
Opponent entered "pick"
Your turn, enter word: puck
Valid word!
Please wait for opponent to enter word...
Opponent lost the round!

Round 2...
Score is 1-0
Board: y u v b i u j g
Please wait for opponent to enter word...
Opponent entered "big"
Your turn, enter word: jug
Valid word!
Please wait for opponent to enter word...
Opponent lost the round!

Round 3...
Score is 2-0
Board: t e w b n c x w
Your turn, enter word: web
Valid word!
Please wait for opponent to enter word...
Opponent lost the round!
You won!
```

Player 2's output:

```
You are Player 2...
Board size: 8
Seconds per turn: 30

Round 1...
Score is 0-0
Board: e x w k i p c u
Please wait for opponent to enter word...
Opponent entered "wick"
Your turn, enter word: pick
Valid word!
Please wait for opponent to enter word...
Opponent entered "puck"
```

```
Your turn, enter word: kwic
Invalid word!

Round 2...
Score is 0-1
Board: y u v b i u j g
Your turn, enter word: big
Valid word!
Please wait for opponent to enter word...
Opponent entered "jug"
Your turn, enter word: biv
Invalid word!

Round 3...
Score is 0-2
Board: t e w b n c x w
Please wait for opponent to enter word...
Opponent entered "web"
Your turn, enter word: abcdefghij
Invalid word!
You lost!
```

Academic Honesty

To remind you: you must not share code with anyone other than your assigned partner and your professor: 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 any other groups, 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.