Computer Science 367

Project 2 (70 points)

v1.0

Deadlines

- Checkpoint 1: Oct 21, 2020
- Checkpoint 2: Oct 26, 2020
- Team submission: Nov 2, 2020
- Individual submission (canvas): Nov 3, 2020

All deadlines are 11:59 pm PDT (UTC-7).

1 Overview

Like the previous project, this is also a team assignment. For this network programming assignment you and your 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 Socket Lab and Project 1, this project 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 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 your GitLab version control repository.

2 Project 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.**

2.1 Repository and file naming requirements

Your team repository has the following files. Do not rename them and use them as instructed below. You can add additional files.

1. `client.c`: Your client source code should be contained in this file. If you create a corresponding header file you must name it `client.h`.
2. `server.c`: Your server source code should be contained in this file. If you create a corresponding header file you must name it `server.h`. If you wish to have a shared header used by both the client and server, name it `proj.h`.
3. `trie.c` and `trie.h`: For implementation of trie data structure.
4. `twl06.txt`: Word dictionary. Do not modify this file.
5. `writeup.txt`: Your writeup should be written in this file.
6. `plan.txt`: Your plan should be written in this file.

You must actively use git during the development of your program. If you do not have at least 3-5 commits, you will loss points.

2.2 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 (File `twl06.txt` in your repository)

An example command to start the server is: `./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: `./client 127.0.0.1 36799`

2.3 Compilation

Your code should be able to compile cleanly with the following commands on department linux machines:

```
gcc -o server server.c trie.c
gcc -o client client.c
```

2.4 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 things 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

3 Submission

This project has two checkpoints, a final team code submission, and an individual submission.

Checkpoint 1 (5 points). You and your partner will need to work together to construct a plan for the project and document your plan in the file `plan.txt` in your repository. Your plan should include:

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

To submit this checkpoint, push your changes to the `master` branch of your GitLab repository.

Checkpoint 2 (10 points). For this checkpoint, you should submit the following things:

- A working implementation of trie data structure. See Section 6.1 for details.
- An outline for your server and client code. The outline should describe the different logical pieces in your code (e.g., functions, code blocks). Describe *what* those pieces are supposed to do and *why*; at this stage, do not focus on *how* you'll implement them. Write the outline as comments in `server.c` and `client.c`. There should not be any C code in the outline.

To submit this checkpoint, create a new branch called `checkpoint2` and push the changes for this checkpoint to that branch. After submitting this checkpoint, you can continue working on your implementation in the `master` branch.

The goal for this checkpoint is to make sure you have a correct implement of trie and to give you feedback on your project structure and outline before you start implementing it. So, if finish this checkpoint before its deadline, do let me know and I'll try to give you feedback early.

Final code submission (50 points). To submit your code, simply push your changes to the `master` branch of your GitLab repository. I'll grade the last commit before the deadline.

If you wishes to use a late day, fill out the late day form [form URL]. Remember that to use a late on a team assignment, both team members should use a late day.

Individual submission (5 points). A short online survey on Canvas. This will be posted 2 days before the deadline.

4 Grading

By default, you will begin with full 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 checkpoints (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 memory management
  - Insufficient versioning in GitLab
- Lose a moderate to large amount of points:
  - Server cannot handle multiple clients concurrently
  - 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.)
- Lose all points:
  - Code that does not compile

5 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. 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.
2. Any assumptions you made for things not described in the specifications.
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.)

6 Assignment Details

6.1 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 project, you can use any publicly available trie implementation, but you must clearly cite the source in a comment at the top of the file. Your trie implementation should contained in files `trie.c` and `trie.h`. 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.

6.2 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!
```

7 Academic Honesty

To remind you: you must not share code with anyone other than 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 students, 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 and Slack (do not wait until the last minute to email). If you participate in academic dishonesty, you will fail the course.