Computer Science 367

(Individual) Program 1 (50 points)

Due Wednesday, February 4th, 2015 at 10:00 PM

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

Overview

Our first (real) network programming assignment takes the skills you learned (or sharpened) in Program 0 and applies them a turn-taking two-player networked game. Namely, you will be implementing the server and client software to support three variants of the Connect 4 game (see rules later in this document). Fig. 1 illustrates a game-in-progress. The basic idea is that the two players alternate dropping tokens into one of the slots on the top, where they come to rest in the lowest available row. In the standard game, the first one to connect four of their tokens (horizontally, vertically or diagonally) wins.

(a) Real life game of Connect 4 (image credit: Popperipopp, Wikimedia Commons).

$$\begin{bmatrix} 2 & 1 & 1 & 0 & 0 & 2 & 2 \\ 1 & 1 & 2 & 0 & 2 & 1 & 2 \\ 2 & 2 & 2 & 1 & 2 & 1 & 1 \\ 1 & 2 & 1 & 2 & 1 & 1 & 2 \\ 2 & 1 & 2 & 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 & 1 & 1 & 2 \end{bmatrix}$$

(b) Our simplified display of the same game state, with red=1, yellow=2, and empty=0.

Figure 1: Connect 4 example.

The game proceeds until one party wins, or all of the slots are filled with no winner, resulting in a tie. All interactions between the two players (clients) will be relayed through the server, who is responsible for maintaining the game state and enforcing rules. Your server needs to be able to support an arbitrarily large number of concurrent games taking place (until system resources are exhausted).

You are responsible for implementing (in the C programming language) both the client and the server for this game, using sockets. It must be developed in your Subversion version control repository.

Program 1 Specifications

Your program 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

All files should be contained in a `prog1` directory in your repository. Spacing, spelling and capitalization matter.

- The writeup and all of your source code should be found directly in `yourWorkingCopy/prog1`, where `yourWorkingCopy` is replaced with the full path of your working copy.
- Your client source code should be contained in a file named `prog1_client.c`. If you create a corresponding header file you must name it `prog1_client.h`.
- Your server source code should be contained in a file named `prog1_server.c`. If you create a corresponding header file you must name it `prog1_server.h`.
- Your writeup must be a plain-text file named `writeup.txt`.

Command-Line Specification

The server should take exactly two command-line arguments:

1. The port on which the server will run, a 16-bit unsigned integer
2. The game type specifier; one of the following three options
   - `standard`
   - `popout`
   - `antistack`

   (these games are described later)

An example command to start the server is:

```
./prog1_server 36799 standard
```

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:

```
./prog1_client 127.0.0.1 36799
```

Protocol Specification

The protocol is summarized by the following rules:

- Immediately after any client connects to the server, the server first sends a character to the client to indicate the game type: `S` (standard), `P` (popout) or `K` (antistack). The client should display a message to the user indicating the game type.
- When the first client ("Player 1") contacts the server, after sending the game type the server sends the character `'2'` to indicate that it is waiting for the second player.

- Once a second player ("Player 2") has connected (and received the game type from the server) the game begins. A parent process should go back to accepting new connections to start another game.
- The server initializes the "game state." This is a 42 byte character array, each element representing one slot in the 7 column by 6 row board. The character `'0'` is used to denote an empty spot; the character `'1'` marks a spot containing a Player 1 token; the character `'2'` marks a spot containing a Player 2 token. The game state is initialized to all `'0'`.
- The players take turns. In each round one player is "active" (his or her turn) and the other is "inactive" (not his or her turn). Player 1 always goes first (i.e. is active first).
- Communication in each round consists of the following:
  1. The server sends the character `'Y'` ("your turn") to the active player. The server then sends the current game state to the active player. The client should display the game state and prompt the active player to make a move.
  2. The server sends the character `'H'` ("hold please") to the inactive player. The server then sends the current game state to the inactive player. The client should display the game state and prompt the user to wait.
  3. The active player sends a two byte sequence.
     1) A move type indicator: `'A'` to add a token, or `'P'` to popout a token.
     2) The column index of the move (a *character* between `'0'` to `'6'`)

     (Note: the player should type the move into the client in this format; e.g., `A3`)
  4. If the move was valid and caused one of the two players to win, then
     1) The server sends `'W'` to the winning player and closes that socket.
     2) The server sends `'L'` to the losing player and closes that socket.
  5. If the move was valid and caused the game to tie (full board with no winner), then
     1) The server sends `'T'` to both players and closes both sockets.
  6. If the move was invalid,[1] then
     1) The server sends `'I'` ("invalid") to the active player, which should then prompt the user for another move, and we go to step 3 and continue the round.

The game state encodes the board in *row major* order. That is, the indices of the array map to locations in the game board as follows:

$$\begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 & 32 & 33 & 34 \\ 35 & 36 & 37 & 38 & 39 & 40 & 41 \end{bmatrix}$$

[1] There are a large number of things that could cause the move to be invalid, including the column being full for an add move, an invalid move type, and invalid column, popout move for a game that does not support popout, etc. Essentially, a move is invalid if there is no valid way to update the game state to accommodate it.

This ordering makes it easy to display the board (at the expense of being somewhat more complicated to add and pop out tokens). To display the board on the screen, a space should be added between each element in a row, and a newline should be inserted after every seven elements. Below the board the column indices should be printed. See the following for an example of the formatting:

```
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 2 0 0 0 0 0
1 1 0 0 0 0 0
2 2 0 1 0 0 0
1 1 2 1 2 0 0
-------------
0 1 2 3 4 5 6
```

The Repository

- All code for this program will be developed under a Subversion repository. If you did not create your repository before Program 0, you will need to create one now. You can use the `subversion_startup.sh` script to automate this process.
- As noted above, your work will be done in a `prog1` sub-directory of your working copy. Therefore, exactly once at the beginning of working on Program 1, you will need to create this directory and `svn add` it to your repository. You will also need to `svn add` your writeup and every source code file to your repository. **If you do not add and commit your files, I cannot see them, and thus cannot give you any points for them.** You can use the `subversion_check.sh` script to help determine if all of your files have been successfully added and committed.
- You must actively use Subversion during the development of your program. If you do not have at least 5-8 commits for Program 1, points will be deducted.
- See the Subversion Guide on Canvas for all sorts of details on Subversion.
- *Note: you must never edit files in your repository directly. All interaction will be indirect, via your working copy.*

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:

- `prog1_client.c` and `prog1_server.c`
- Your write-up: `writeup.txt`
- Optionally, the header files `prog1_client.h` and `prog1_server.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 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
  - 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

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
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.)

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

Connect 4 Rules

Below are the rules for the three Connect 4 variants your server must support:

Standard

In the standard version of Connect 4, you can only add tokens (no pop out). Tokens can be added to any column that still has space, and will fall to the lowest available free spot. You win by placing your tokens so that four are arranged horizontally, vertically, or diagonally. If the game board has been filled without any winner, the result is a tie.

Pop Out

In the pop out variant of Connect 4, during your turn you can choose one of two moves:

1. You can choose to add a token as described in the standard version, or
2. You can instead choose to remove ("pop out") one of *your* tokens from the bottom of the board. You can only remove a token if it is in the very bottom row. Upon popping out a token, all tokens above it in the column will shift down by one. Again, the first player to have four of their tokens arranged in horizontally, vertically or diagonally wins.

For simplicity of our implementation, we will again call it a tie when the game board that has been filled without any winner.

Anti Stack

This variant is similar to the standard version, except that the first person to get *three* tokens in a row *loses*. That is, your goal is to avoid getting three of your own tokens in a row (horizontally, vertically, or diagonally). Like the standard version, players can only add tokens (no pop out). If the board fills with no winner, the result of the game is a tie.

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 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.