CSCI415—Systems Programming

Spring 2017

Programming Assignment 7

A GUI for Distributed Tic-Tac-Toe

You may use the code in `~kearns/public/415/tictactoe` or your own code from P4 as a starting point for a graphical user interface for `ttt`, the tic-tac-toe client. Specifically, modify `ttt.c` so that it spawns a subprocess running `wish` to provide a GUI for the client. Of course, `ttt` and `wish` will communicate by means of a pair of pipes, and `ttt` will communicate with `TTT` through an IPv4 virtual circuit.

When the `ttt` program is invoked, it should operate as before up through prompting the user for a handle. After obtaining the handle, it should interact exclusively through the GUI whose minimal properties are described below:

- A canvas widget should display the tic-tac-toe game board. A standard tic-tac-toe grid will define the nine regions of the board.
- Beneath the board, there should be an area in which the handle of the players and the symbol (X or O) each is using is displayed.
- There should also be an area in which status information is displayed in textual form. The following messages would be appropriate for status information:
  - Awaiting Match
  - Your Move
  - Awaiting Opponent Move
  - You Win
  - You Lose
  - Draw
  - Opponent Resigned
- When it is your move, the system "bell" should sound. See the `bell(n)` manual page. Mousing left on an empty square registers your move into that square. An X or O is displayed in that square as appropriate. If not your move, mousing in a square is a no-op. If you mouse on a square that is already occupied, an audible beep should be produced.
- When an opponent moves, an X or O should appear in the appropriate square to show the move.
- When a game does not end in a draw, a line should be drawn through one sequence of winning moves.

- There should be three buttons associated with your display. The Exit button does the obvious thing: it will shut down the client, but only after the game is over. Mousing on this button while the game in being played is a no-op. This is the mechanism by which the GUI should be taken down under normal circumstances.

  Another button should be labeled `Silent`. If you mouse left on this button, it will disable the required beeps. It will also change its label to `Sound`. If you mouse left on the `Sound` button, it will turn beeping back on (and will change its label to `Silent`). Effectively, this button will allow you to toggle beeps on and off as you choose.

  A button labeled `Resign` should be used by a player to resign from the game before the play is complete. The resigning player loses and the other player wins; the message area of each client should reflect the fact. We expect that the Exit button would then be used to take down the client. Note that resignation is a new operation for the tic-tac-toe system, and modifications in the underlying client/server will be required. This button should only be active when it is a player's move (hence, you can't resign until it is your move).

Tips and Hacks

- Lots of what you want to do in the GUI can be phrased as calls to TCL procedures. TCL procedures can reference global TCL variables (i.e., defined at the outermost level of the script, not within a procedure body). In a given procedure, to access global variable `x`, you must execute the command

  ```
  global x
  ```

  probably early in the procedure, but certainly before a reference to `x`.

- Debugging support is weak. Diagnostic `puts` work for me.

- I emphasize that these are minimal specs for the GUI. Feel free to extend it, but you should curb any inclination to extend until you have the minimal GUI working.

Submission

Create a gzipped tarball of all the C source and TCL files used to build your system. Also include a `Makefile` from which the TA can build your `binaries`. Here's what I would have done to create the tarball:

```
% cd my_working_directory
% tar zcvf submit.tgz Makefile TTT.c ttt.c msg.c common.h child.c child.h ttt.tcl
% gpg -r CSCI415 --sign -e submit.tgz
% mv submit.tgz.gpg ~
% chmod 644 ~/submit.tgz.gpg
```

We will decrypt and unpack your files into an empty directory. We will then make that directory current and invoke the command `make` to build your system. Finally, we will bring your system into execution with an invocation of `./TTT`, and (in separate windows) a couple of invocations of `./ttt`. All activity will take place from the directory in which the TA builds your system.