Virginia Tech
Bradley Department of Electrical and Computer Engineering
ECE 3574: Applied Software Engineering ◆ Spring 2013

Homework 6

Submission Details

The solutions for this homework must be submitted as an electronic submission. Your submission will be a tarred and gzipped file containing the source files for the programming problem. These files must be in a directory entitled HW6_*LAST_FIRST* where LAST is your last or family name and FIRST is your first or given name. Your program will be run to evaluate its correctness, and the source code will be reviewed for adherence to Qt programming style. Your program must run on the latest version of Ubuntu and be compiled/built using the GNU C/C++ compiler and qmake. The following information must be included in your source files: your full name, your student ID number, your email address, class (ECE 3574), title of the assignment, and date of submission. You must submit the solutions using the "Submission" button on HW 6 assignment page on the Scholar class web site. The file must be given a name of the following form: HW6_*LAST_FIRST*.tar.gz where LAST and FIRST are as specified above. You can make multiple submissions. Paper and email submissions will not be accepted. All work must be submitted by the announced due date/time.

Honor Code

As stated in the syllabus, in working on homework and projects, discussion and cooperative learning are allowed. However, copying or otherwise using another person's detailed solutions to assigned problems is an honor code violation. See syllabus for details.

Homework 6:

For this homework you should write two programs. One is a "server" program that displays a single QTextEdit window. The second is a "client" program that can start up with multiple "chat" windows. Each of these chat windows is managed by a separate Qt thread (derived from QThread). The communication between the client program threads and the server program thread will be through POSIX message passing.

The Server Program:

The server program displays a single QTextEdit window that includes all the input and records all the state changes of the client chat windows. The server window displays four types of messages in its window. The name of the server program is "chatServer".

- When a client first contacts the server, the server displays a message of the form "client X: Has joined the chat session." Where the X is the name of the client.
- When a client exits the chat session, the server window displays a message of the form "client X: Has exited the chat session." Where the X is the name of the client.
- When a client sends a general chat message to the server, the server displays a message of the form "client X: Y." Where X is the client name, and Y is a string which is the message sent by the client.
- A client can also send a private chat. When client X sends a private chat to client Y, the server display should say "client X sent private chat to Y: Z" where Z is the private chat message.

The Client Program:

The name of the client program should be "chatClient". The client program is started up with the command:

```
./chatClient Z
```

where Z is an integer that is the number of client windows that should be run (again each one has a separate thread).

Each client window should have the following properties:

- On startup, each client thread should display an initial window with a button labeled "Connect as" and a QTextLine sub-window where one can input a name. The "Connect as" button should be disabled (grayed out) until the user starts to type some text in the QTextLine box. Once a name is entered, the user can press the "Connect as" button and the client will try to connect to the server. One of three things can then happen. First, if the server is not started up, a QDialog error window pops up informing the user that they must first start the server. Second, if another client has already connected to the server with the same name, a QDialog error window should pop up informing the user to choose a new name as that name is already connected to the server. Third, after a successful connection to the server, the initial window should disappear, and a new chat window entitled "X's chat window" should appear. This chat window is specified below.
- The client window should have a single buttons labeled "Exit". Clicking on the "Exit" button should exit from the server and close the chat window. None of the other chat windows should be affected.
- The client window should have two text windows. The main text window is a "QTextEdit" window that displays everything that the user should see from the server (described below). The window should be "QLineEdit" window. The "QLineEdit" window should have a QComboBox GUI associated with it.
- The combo box should display the names of all the active clients currently connected to the server (except for the name of the user on the particular client). In addition, it should display the phrase "General Chat".
- The user can select one of these names and type something into the QLineEdit window. When the user types something into the general chat "QLineEdit" window and hits return, the text is sent to the server. The server then displays the text as discussed above in the server specification. If the combo box is set to "General Chat", then server communicates the text to all the active clients, and they display this text in their "QTextEdit" windows in the same format as the server window. If the name associated with one of the clients is selected in the combo box, then the server only communicates this message to client associated with this name and the sending client. Again, the format on the clients main window and the server window should be as specified above.
- The client program should wait for all the client threads to exit, and then exit in a graceful manner (e.g., no segmentation faults!).

The thread and POSIX requirements:

There should be a thread associated with each window that appears in your application for both the server and the client). The QThread class should be used as a base class for derived class used to create and manage all threads.

The client and the server threads should communicate via POSIX message passing. See the examples from class for any information on how POSIX message passing works. The POSIX message passing should be blocking.

What to turn in:

The server and client programs should be in separate directories under the main directory that you turn in. The directory names should be "chatServer" and "chatClient".

Grading Policy

- QT style programming and indentation – 20%.
- The code complies and runs – 0%.
- The code satisfies the specifications – 80%.