CS 408 - Computer Networks - Fall 2014

Project – Social Event Scheduler

This project is made of three steps; each has different deadlines specified below;

Project Step 1 Deadline: November 13, 2014, Thursday, 22:00
Project Step 1 Demo: to be announced

Project Step 2 Deadline: December 10, 2014, Wednesday, 22:00
Project Step 2 Demo: to be announced

Project Step 3 Deadline: December 24, 2014, Wednesday, 22:00
Project Step 3 Demo: December 26, 2014, Friday

Introduction

In this project, you will implement a client-server application for a social event scheduler. In the application that you will implement, there is (i) a Server module which manages message transfers, scheduled events and friendship relations among the users, and (ii) a Client module which adds friends, accepts/rejects friendship requests, sends messages, schedules and joins events, views event and participation details.

The server listens on a predefined port and accepts incoming client connections. There may be one or more clients connected to the server at the same time. Each client knows the IP address and the listening port of the server (to be entered through the Graphical User Interface (GUI)). Clients connect to server on corresponding port and identify themselves with their names. Server needs to keep the names of currently connected clients in order to avoid the same name to be connected more than once at a given time to the server.

The abovementioned introduction is for the entire project, which is to be completed in three steps. Each step is built on the previous step and each has specific deadlines and demos.

Project Step 1 (Deadline: November 13, 2014, Thursday, 22:00):

After the server starts listening, clients start to connect to the server. A connected client can broadcast textual messages to all other connected clients via the server. In other words, server behaves like a bridge among all clients. If a user sends a message, server forwards this message to other users (except the sender). Therefore, server needs to keep connected clients' list up-to-date.

Users perform all of the operations through a GUI; such as connecting to the server, entering their name, sending message, etc. Additionally, all received messages with the sender's name must be shown on the client GUI.

This step is the basis of the project. Although the project has some friendship management features, you do not have to implement those in this step of the project. Moreover, in this step, you do not need to implement any event scheduling tasks. You will implement them in coming steps.

For programming rules and submission specifications, please read the corresponding sections at the end of this document.

Server Specifications:

- The port number on which the server listens is not to be hardcoded; it should be taken from the Server GUI.
- The server will start listening on the specified port. It has to handle multiple clients simultaneously. To do so, whenever a client is connected to the listening sockets, it should immediately be switched to another socket.
- The server forwards the incoming messages to all clients other than the sender, but the server should include the name of the sender to the forwarded messages.
- All activities of the server should be reported using a rich text box on the Server GUI including the names of connected clients as well as all the message transfer details. We cannot grade your project if we cannot follow what is going on; so the details contained in this text box is very important.
- Server must handle multiple connections. At the same time, one or more clients can send and receive messages to/from the server.
- Connected clients’ names must be unique; therefore, the server must keep track of connected clients’ names. If a new client comes with an existing name, server must not accept this new client.
- When the server application is closed (even abruptly), nothing should crash!

Client specifications:

- The server’s IP address and the port number must not be hardcoded and must be entered via the client GUI.
- There could be any number of clients in the system.
- If the server or the client application closes, the other party should understand disconnection and act accordingly. Your program must not crash!
- All activities of the client should be reported using a rich text box on the client GUI including sent and received message information. We cannot grade your project if we cannot follow what is going on, so the details contained in this text box is very important.
- Each client must have a unique name. This name must be entered using the client GUI. This name is also sent to the server (preferably during initial connection). The server identifies the clients using their names.
- Each client can send and receive textual messages at any time. If a client sends a message, this message is forwarded to all other online clients by the server.
- Each client can disconnect from the system at any time. Disconnection can be done by pressing a button on client GUI or by just closing the client window.

Project Step 2 (Deadline: December 10, 2014, Wednesday, 22:00):

Second step of the project is built on the first step. In this step, in addition to the message transfer feature of step 1, event management feature is added to the application.

Each user can schedule any number of events. Organizing user enters the details of the event (such as time, place and description of the event) as free text via the GUI. The event details are sent to the server. After that, server forwards the event to all other connected clients by sending the event's time, place, description and organizing user's name. In other words, all other clients are invited automatically to the event by the organizer. Invitees may answer as “attending” or “not attending” using the GUI, or they may not answer. Invitee's participation status remains as “not answered”, until he/she answers the invitation. Any invitee can answer or change his/her answer at any time using the GUI, but once it gives an answer, it cannot change its status back to “not answered”. Answers and participation status changes are sent to the organizer as notifications by the server as soon as a change occurs. These notifications must be shown in the organizer’s GUI. On the other hand, event participation details are not automatically fed to other users; other users (invitees) should be able to explicitly request a particular event's up-to-date invitee list and their participation status from the server; once such a request is received, the server sends the requested information to the requesting user only (not to everyone).

A particular user can organize and attend any number of events. Moreover, the event management operations can be performed at any time while the user is connected to the server.

As in the step 1, all of the operations must be clearly shown on the client and server GUIs.

For programming rules and submission specifications, please read the corresponding sections at the end of this document.

Project Step 3 (Deadline: December 24, 2014, Wednesday, 22:00):

Third step of the project is built on the first and second steps. In this step of the project, friendship relations are to be implemented in the application.

As in step 2, any user can schedule an event, but now only the friends of the organizing user are invited to the events. Similarly, users can send/receive textual messages only to/from their friends. For this reason, friendship relations should be established among the users. Therefore, users send friendship requests to other online users at any time. In order to add a user as friend, its name must be selected from the online clients’ list sent by the server. Users who receive friendship requests can either accept or reject. This decision is sent to the requesting user as a notification by the server. You do not need to have any other functionality (such as changing friendship status, banning, etc.) for friendship management.

In this final step of the project, when a user sends a message, it must be forwarded to all of his/her current friends. Similarly, when a user creates an event, all of his/her current friends must be invited. Other event management details are performed as in step 2.

As in the step 1 and 2, all of the operations must be clearly shown on the client and server GUIs.

For programming rules and submission specifications, please read the corresponding sections at the end of this document.

Group Work

- You can work in groups of **two** people for step 1; **three** people for steps 2 and 3. You may change groups between steps 1 and 2; however, no group changes are allowed between steps 2 and 3.
- Equal distribution of the work among the group members is essential. All members of the group should submit all the codes for both client and server. All members should be present during the demos.

Programming Rules

- Preferred languages are C# and Java, but C# is recommended.
- Your application should have a graphical user interface (GUI). **It is not a console application!**
- You have to use pure sockets as mentioned in the socket lab sessions. You are not allowed to use any other socket classes.
- In your application when a window is closed, **all threads related to this window should be terminated.**
- Client and server programs should be working when they are run on at least two separate computers. So please test your programs accordingly.
- Your code should be clearly commented. This affects up to 10% of your grade.
- Your program should be portable. It should not require any dependencies specific to your computer. We will download, compile and run it. If it does not run, it means that your program is not running. So do test your program before submission.
- Your program should compile and run. Do not submit something incomplete.

Submission

- Submit your work to SUCourse. Each step will be submitted and graded separately.
- Provide a README file explaining your implementation, detailing any issues and/or other implementation details. If your program is lacking in some requested functionalities, clearly explain them. This file should also serve as a user's manual.
- Delete the content of debug folders in your project directory before submission.
- Create a folder named **Server** and put your server related codes here.
- Create a folder named **Client** and put your client related codes here.
- Create a folder named **XXXX_Surname_Name_StepY**, where XXXX is your SUNet ID (e.g. dakdogan_Akdogan_Dilara) and Y is the project step (1, 2 or 3). Put your Server and Client folders into this folder.
- Compress your XXXX_Surname_Name folder **using ZIP**. Please **do not use** other experimental compression utilities like ICE, bz2. etc.
  - You can download winzip here: www.winzip.com Trial version is free!
- You will be invited for a demonstration of your work. Date and other details about the demo will be announced later.
- 24 hour late submission is possible with 10% penalty.

We encourage the use of SUCourse Discussion module for the general questions related to the project. For personal questions and support, you can send mail to dakdogan@sabanciuniv.edu, alperenp@sabanciuniv.edu, and leskandarian@sabanciuniv.edu .

Good luck!

Albert LEVİ
Dilara AKDOĞAN
Alperen PULUR
Laleh ESKANDARIAN