Project 1: A Client-Server Chat Program

Department of Mathematical Sciences
Computer Science Division
University of Stellenbosch
7600 Stellenbosch

July 2014

1 Introduction

For this project, you will be required to write a chat program based on the client-server model. Understanding the principles used in writing such a program will provide you with insight into practical computer networks and the problems faced when implementing them.

2 General Information

Please note of the following points when implementing your solution:

- You may code in any programming language, but we recommend Java. Take note that the demonstrators might not know your chosen language and would be unable to help you if you run into trouble.
- You may work in groups of no more than two persons. You do not have to work in a group.
- Every project must be accompanied by a report. The report must contain the following:

  1. Name(s) and student number(s).
  2. A description of the project.
  3. Features included in your solution.
  4. Features not included in your solution.
  5. Extra features included.
  6. Algorithms and data structures used to implement your solution.

  All project sources and reports will be checked for plagiarism, including being checked against solutions from previous years. If you are found guilty there will be serious consequences, including receiving a mark of zero for the project.

3 Project Specifications

The model used for this project is the single server - multiple client model. The following general specifications must be implemented:

1. Multiple clients must be able to connect to a single server.
2. No GUI is needed for the server.

3. A simple GUI may be implemented for the client.
4. Clients must be able to "whisper" to each other.
5. Clients must be able to choose a nickname.

3.1 The Server

A single server program should handle all requests from the clients. As such, you will have to implement a multi-threaded service solution for your server. The following must be implemented in your server application:

1. Server operations (such as connect requests and disconnect requests) should be printed out by the server.
2. The server must handle connections / disconnections without disruption of other services.
3. Clients must have unique nicknames, duplicates must be resolved before allowing a client to be connected.
4. All clients must be informed of changes in the list of connected users.

3.2 The Client

The following must be implemented in your client application:

1. A list of online users must be displayed (via GUI or command).
2. Connection / disconnection actions of users must be displayed.
3. Messages from the originating user and other users must be displayed (in other words the messages you send must also be displayed).
4. Must still be able to receive messages / actions while typing a message.
5. Clients must be able to disconnect without disrupting the server.

4 Dates

- **Project starts:** 23 July 2014 (14:00)
- **Project deadline:** 6 August 2014 (13:00)

Your project with the report must be emailed to `16148517@sun.ac.za` before the deadline.

5 Helpful Hints

1. Start early, do not leave this project to the last few days and think you will finish on time.
2. Look at the marking scheme and at what needs to be done to help you gauge your progress.
3. Do not think you are almost done after you have successfully had your client connect to your server.
4. Threading is a big part of this project so make sure you do not wait until the end before you start implementing multiple clients handlers.
5. Implement threading in your client as well. If you only use one thread in your client, your GUI will freeze while waiting for a connection. Implement a GUI thread and another concurrently running client thread.

6. Use packet objects to send and receive information. Do not use the PrintWriter class in Java. It will just make things difficult later on. Packets give you the ability to exchange data as well as house-keeping information between clients and your server. (Java ObjectInput/OutputStream)

7. Make sure to implement mutex locks in your threads. You need to have a data structure containing information on connected clients and you will have multiple clients writing/reading to/from the same structure. Without mutual exclusion the data will be corrupted. Use the "synchronize" statement to implement mutual exclusion. Other thread utilities like "countdown latches", "thread priorities" and the "wait" and "notify" methods might also come in handy.

8. Disconnecting clients can cause a lot of problems, make sure you do not leave the issues for later.

6 Marking Scheme

Client GUI	**3**
Sending / Receiving Messages	**3**
Sending / Receiving Messages (concurrency)	**3**
Whispering	**4**
- Recognizing whisper command	2
- Sending whisper messages	2
List of currently connected users	**4**
Program Stability and Error Reporting	**13**
- Updating user list after user connection / disconnection	3
- Client Stable after Server Termination	3
- Server Stable after Client Termination	3
- Whispering to a Non-Existing Client	2
- Trying to Connect to Non-Existing Server	2
All Opened Sockets and Streams Closed on Client Termination	**1**
Server Service Model (concurrency)	**2**
Server Notification System (notifying clients of activities)	**2**
Report	**15**
- Language, spelling and grammar	3
- Selection of experiments discussed	2
- Description of the experiments	5
- Conclusions drawn from the results	5
TOTAL	**50**