Tegan Lamoureux

Waterfall Chat

Design Document

Table of Contents

1 Introduction

This document will describe the design and operation of the Waterfall Chat server-client system on a component level. We will discuss design considerations, a system overview, the system architecture, and a detailed system design.

1.1 Purpose and Scope

Exploring the design and operation of the program has many facets. When looking at design considerations, for example, we're going to be discussing constraints and dependencies of the program, as well as the methodology used in the design of the program. For constraints, we're looking at what requirements shape the design of the program, and for methodology, we will be looking at what design and programming practices and paradigms we will be using.

1.2 Target Audience

This document is intended for Tegan, the author, to allow her to understand and effectively design the program. It's also intended for the TA, and Professor Xie, to allow them to gain an understanding of my design process, and the overall design of my program.

1.3 Terms and Definitions

- User - A client (user of the program) that logs into the server.
- Server - A program that accept connections from multiple users, and allows them to communicate with each other.
- Chatroom - An online space that allows users to communicate with each other by sending public messages to everyone else in the room.

- Private Messaging - The sending of a message from one user to another user.

2 Design Considerations

When designing this program, many things have to be taken into consideration. Mainly, the server-client architecture is a vital component, and will dictate many of the possible actions and components of the program. We want to allow a large number of clients to connect to the server and still maintain the functionality of the program. This will require making most methods and subsystems generic, for any number of clients.

2.1 Constraints and Dependencies

Formal constraints are mainly dependant on the resources of the system, and the time available to the designer and programmer. Given this, some limitations were imposed. Mainly, that the system will work for up to 100 clients, but will not be in a supported state should that number be exceeded. Also, given ideal networking conditions, the system should operate as normal, and should non-ideal or erroneous conditions arise, the system should let the user know that the system is not in a supported state, and messages may not be received/sent out.

2.2 Methodology

This program will use a server-client architecture, which requires non-blocking communication between multiple clients and a server. We will also use an object oriented approach to the design and implementation of the program, which allows for a contained, reusable, and data-safe paradigm of programming.

3 System Overview

The system will consist of two main components, the server program, which will be instantiated only once, and the client program, which will be instantiated N number of times such that $N \leq 100$.

For the server, it will essentially run in a loop, interrupting on the receival of a network packet. Once data is received, it will be parsed, and the appropriate action will be performed. In this case, there will be multiple types of actions to perform. Some common ones are assigning a client ID to a given connected client, checking whether a username already exists, creating a new account on the server for that username, logging in an account (which ties the client ID to the username), sending a message to a single user, or sending a message to multiple users.

For the client, we're going to also have it running in a loop, interrupting on user actions or messages received from the server. Common user actions will be to connect to the server, create an account, login, and send messages. Common actions to be performed at the request of the server are displaying messages meant to all users, and displaying messages meant for a single user, as well as verifying account status, username status, etc.

4 System Architecture

As described above, the system will follow a client-server architecture. This will involve multiple clients both communicating and receiving data from the server.

4.1 Server Program

The server program will be instantiated only once, and wait in a loop, performing actions only when contacted and prompted to do so. It will not have any user interaction directly, but instead will coordinate communications between all of the connected client programs behind the scenes.

It will have a main operating loop, which waits for the receival of UDP packets on the incoming port. Once received, these packets are parsed for the client ID of the sender (if assigned) and an instruction value, which will tell the server what to do. Depending on the requested action, the server will respond appropriately, whether that's sending a message to all of the other clients (or one specific one), or creating a new account, or validating a username, etc.

4.2 Client Program

The client program will be instantiated as many times as needed for every desired user. Each user will have access to the same types of inputs, which allow for a homogenized standard of communication with the server (and therein, other clients).

The client program, in some regards, needs to be more robust than the server. While the server only takes in commands, parses them and executes them in sequence, the client needs to not only take in commands and messages from the server, but also to simultaneously send out messages to the server (which allows outbound communication with other clients).

5 Detailed System Design

5.1 Server

The server will accept incoming connections on a UDP socket. In this way, clients will connect to the server and communicate with it. When a client first connects, it will be assigned a client ID that will identify that unique client from then on out. Then, once the user logs in, their username will be tied to that client ID, allowing further communication to be properly identified. The server, from this point on, will mainly serve as a relay between clients. When a user wishes to communicate with another user, they will send a message to the server with the destination user in the packet. The server will then relay this to the desired user. When a user wishes to communicate with every other user in the chatroom (logged into the server), they send a message to the server with the appropriate flag set. The server will then send this message to all users.

5.2 Client

Once the client is connected to the server, the client is now free to communicate with the server. Several different options are available here. As the main function of the client is to send and receive messages, it will always be listening to the server for messages sent to that client. Once received, there will be a flag denoting whether it's a private message sent from one user to the client (a private message) or a message sent from one user to all users (public message). For the sending aspect, the client will send a message to the server any time the user wants to send a message out. Likewise, it will have a flag denoting whether the message is public or private. In the case of a private message, the destination user will also be specified.