Chat application report

Abdun Nihaal

August 17, 2019

1 Introduction

The objective set by this assignment is to create a chat application using java that enables users to send messages and transfer files to other users one on one or in a group.

2 Design

2.1 One thread per client architecture

In this chat server architecture, whenever a client connects to the server, the server creates a separate thread to handle the client. The problem with this architecture is that it may create more threads than what the server CPU cores can handle. Context switching between threads can be expensive and take a lot of memory. In this architecture, the threads perform blocking reads and writes with the client.

2.2 Selector based non-blocking server architecture

Java Nio's selector class can manage multiple channels from a single thread. With the use of Java Nio's selector class, this server does non blocking IO when reading or writing to the client.

The Chat Server Architecture is shown in fig. 1. The server consists of a selector thread and a number of worker threads. The selector thread is responsible for performing network IO and reacting to events. The worker threads process the messages that are received by the selector thread. This architecture enables the worker threads to share the workload.

The communication between the selector and worker threads is achieved by using two data structures. UnprocessedTask queue is a queue of messages that were received by the selector thread. When a worker thread is free, it retrieves a message from the UnprocessedTask queue and processes it. When the worker wants to send a message to a client, it adds the message to the PendingWrite queue. The selector thread writes the data from PendingWrite queue to the corresponding socket channel.

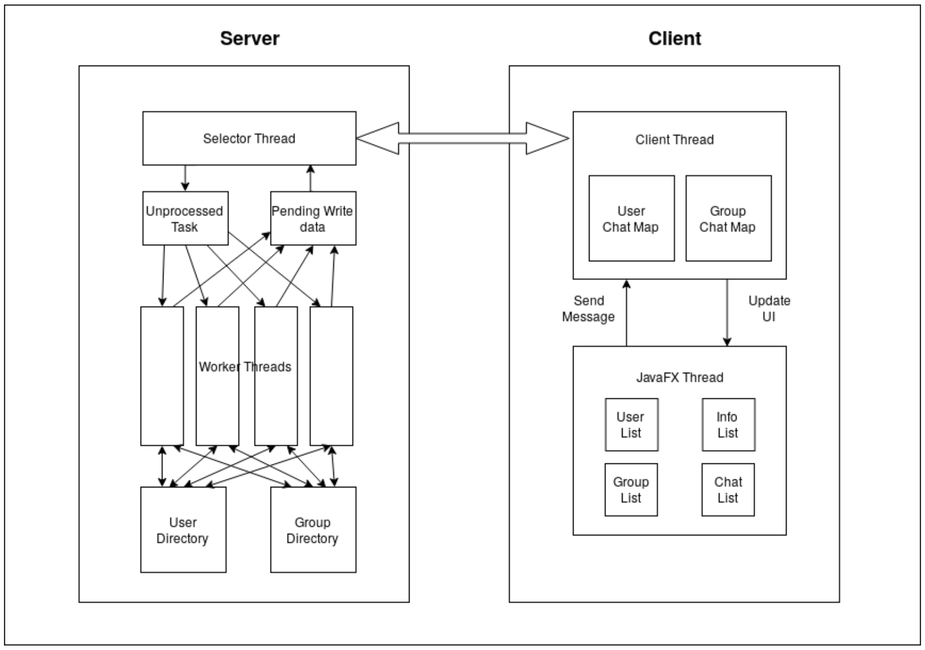

Figure 1: Chat server architecture

The server maintains a user directory and a group directory. The user directory keeps track of all the online users. It stores the username and their corresponding SelectionKey which can be used to send and receive messages from the user's client.

The group directory contains a list of groups created and the users who are present in that group. Whenever a group message is received, the server retrieves the list of users in that group and sends the message to all of them.

2.3 File transfer architecture

The file transfer is done through the use of a different thread on the server that listens for requests on a different port. When a client wants to send a file to another client. The client first sends the file to the fileserver. The fileserver on receipt of the file will generate a random KeyString that identifies the file. The client then sends the KeyString to the receipient of the file. The receipient then requests for the file by sending the KeyString to the file server thread.

In the current design, the fileserver thread blocks on file send and receive. But it can be extended to also use the Selector pattern to improve throughput. The fileserver thread and the client that sends and receives files uses Java NIO's zero copy functionality that allows faster transfer of files without creating a user space buffer. Instead it uses kernel supported functionality to send files to sockets directly from the kernel buffer.

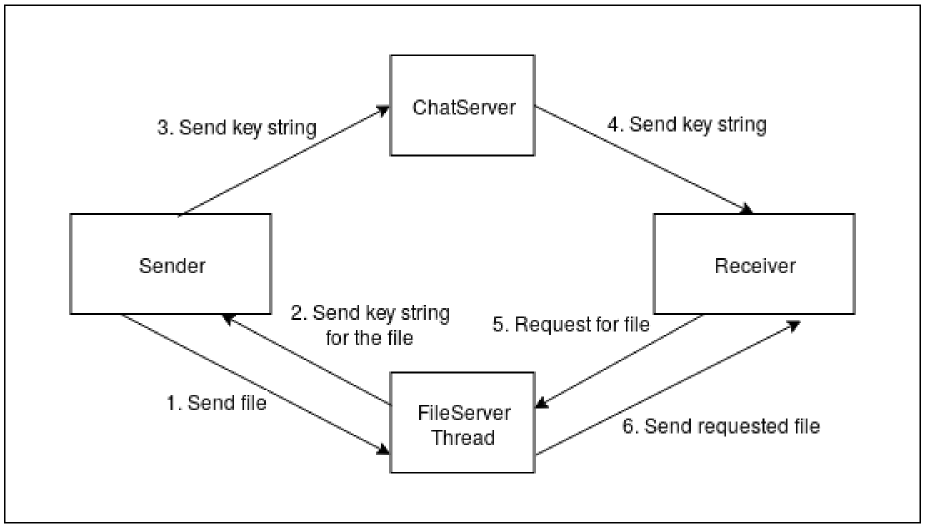

Figure 2: File server Architecture

2.4 Message processing

Messages are processed using string patterns. Messages exchanged between the server and the client are of the form
COMMAND$arg1$arg2$...##
Some of the message types used are

- LOGIN$username##

  When this message is sent to the server, the server tries to log the user in with the given username

- MSG$ReciverName$Message##

  This is sent by the client if it wants to send the message to receiver denoted by ReceiverName

- GMSG$GroupName$Message##

  This is sent by the client if it wants to send a group message to the group denoted by GroupName

- GCREATE$GroupName##

  This creates a group with the group name given and adds the user as a member of the group

- GADD$GroupName$list,of,users##

  This adds the users present in the list of users to the group given by the group name

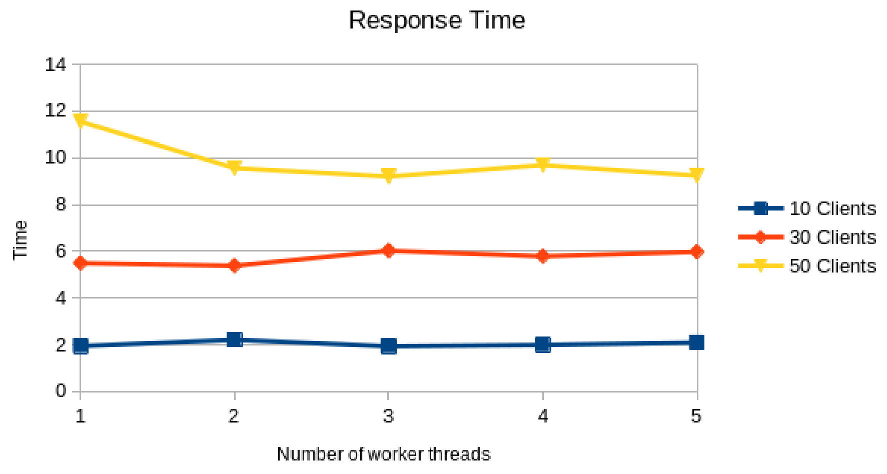

Figure 3: Response Time

- GDELETE$GroupName##

  This deletes the group given by the GroupName

- FILE$Reciever$Filename$NumberOfBytes$KeyString##

  This command is used to send a file to the receiver by sharing the keystring with which the receiver can download the file from the file server.

- GFILE$GroupName$Filename$NumberOfBytes$KeyString##

  This command is used to send a file identified by the KeyString to every member of the group identified by GroupName.

- LOGOUT##

  This logs the user out. The server deletes the user from the user directory

3 Analysis

The response time of the server was measured using a java program that creates a given number of client threads. Each client thread simultaneously sends 10000 echo message i.e. it sends a random string as a message to itself and measures the average time between sending and receiving the message.

In fig. 3, the different lines show the response times for different number of client threads. While performing analysis the client and the server threads were run on the same computer so the improvement when using more threads is countered by the switching of threads. We can see a slight decrease in response time when using multiple worker threads when the number of clients are more.

4 Conclusion

Thus the Chat server for P2P and group chat is implemented using Java Nio and the client application is implemented using JavaFx and Java Nio.

5 References

References

[1] Java NIO tutorial http://rox-xmlrpc.sourceforge.net/niotut/

[2] Getting started with JavaFX https://openjfx.io/openjfx-docs/

[3] Java NIO server client example https://crunchify.com/java-nio-non-blocking-io-with-server-client-example-java-nio-bytebuffer-and-channels-selector-java-nio-vs-io/

[4] Java logging basics https://www.loggly.com/ultimate-guide/java-logging-basics/