CS550 Advanced Operating System
Programming Assignment 1
Shan Huang A20281384
Fan Zhang A20280966

A Simple Napster Style Peer to Peer File Sharing System

Overview

This document describes the overall design and implementation of file consistency mechanisms of the Napster Style Peer to Peer file sharing system. The whole system is implemented in JAVA, and the technologies that involved in the system are Object-Oriented method, TCP socket, and multi-threads programming. The system can work on any operating system, which Java environment is provided, and can be set up on one or multiple machines.

Conceptual Design

1.Overall Structure

Both Index-server and client-server are based on C/S architecture. For Index-server, it needs to provide index information of a client. For client, it needs to communicate with index-server and sends message query to other clients using the index information the index-server provided. . Further, for client, it also needs to act as a server for other clients to download files. Therefore, we can figure it out:

Index-server object includes message, server object, and peer information.

Client object includes message, client object, peer information, and server object.

Figure 1 shows the whole structure of the system.

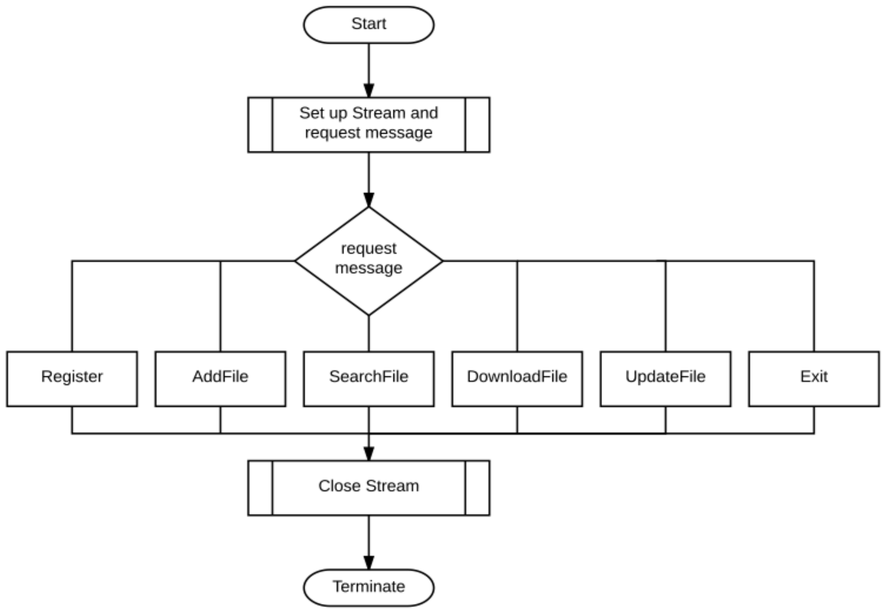

Figure 1

2. Peer information, sockets and messages.

2.1 peer information and messages.

Peer information includes peer name, location, file list, IP Address, and Listening port. This information is encapsulated into peer class. Message's filed includes request function, a single peer, messages, file name, peer list, and byte stream, which are also encapsulated into message class for a client or server's need.

2.2 Sockets connections

A client needs an IP address and the port that it wants to be connected with. A server needs an IP address and the port that the server is listening. Server will use server. accept() method to accept the connections which is initialized by the socket from a client. For each session or request, we provide methods called ConnectionSetUp() and DestroyConnection() to set up the connection and remove the connection.

3. Server Object

An Index-server object called "IndexServer" is designed to handle the connection from clients, connections set up and destroy. How to handle connections is a key part in our design.

Basically a server is always waiting for accepting the request, hence a new thread called HandleConnection() is designed for handling the request, which implements run() method. This thread is totally controlled by the request function in a message. For instance, if the request function in a message is "SearchFile", then it will directly go to the search file part, reading all peer information (the peer information is stored in index-server) into the memory, comparing the filename the client provided with all the file names in all peer information, if the file is found, the thread will enclose all the peers who contains the file, and return the peer list to the client. All peer information is stored in a D source file by the index-server, which is used for handling the connection.

4. Client Object

Client acts as two functions, one is Client, and the other is Server. When user first enters into the system, he/she needs to register this client to the index-server, and then sends request to it. When downloading files, it becomes a server which is used for providing files to another client.

4.1 Client as a client.

Client can input request functions to the server, such as Register, AddFile, SearchFile, Download File, Update File, DeleteFile and Exit. All these methods are enclosed into a class called "ClientRemote".

4.2 Client as a server.

Every client has a corresponding server object, since they are in a single process, and there is more than one client, therefore, obviously using multiple threads is a good choice for this scenario.

The major responsibility of the client as a server is downloading, which also is the key part of the whole system. The sockets send streams and get streams. For sending part, we first need to transfer the file into binary stream. For receiving part, we need to transfer it back. In our design, we make it read several bytes at the same time for sending a file, the message for downloading includes tempbyte[][] and byteread[], which are all encapsulated into message class. By using this method, the file sending function is not limited to the type of ".txt". Also, a file lock is added to the downloading part for the security issue.

Design tradeoffs

This system was implemented in a test-driven development method. Because of the time limitation, we did not focus too much on error detections, and small details for user's convenient. We did not provide GUI either, but all the functions are implemented and tested successfully.

Improvements & Extensions

The possible improvement of the system is to provide more detailed design for user's convenient. For example, in our design, we assume the file we want to download is not a very large one. If user wants to download a very large file, but there is a memory limitation, the way to solve this issue is downloading it partly, and then merges the parts into a whole file.

Contributions of each member

Shan Huang:

Design the framework of the system. Write code for major functions. Test and Debug the program, write documents.

Fan Zhang:

Test and Debug the program, write code for part of functions, write documents.