CmpE 436

Concurrent and Distributed

Programming in Java

Term Project – WorkFlow

Abdullah Furkan Ilisu – 2013400201

December 2017

**1. Abstract**

In this project, the main idea is that the members of a project report their works by editing the related file. There can be many members of the project. It means that there will be multiple clients which can do reading and writing on files. To handle the requests that will make by the multiple clients, the server was implemented as multi-threaded by using Java. The client side is an android application which make connection to the server with TCP sockets. Due to the existing of multiple clients, race conditions can occur on files when someone read the file and the other someone write to the file or different clients do writing at the same time on the same file. To prevent this, I have implemented an algorithm that can solve similar problems. The algorithm uses semaphore approach.

**2. Introduction**

I am inspired by version control systems while building the project idea. Version control systems are a category of software tools that help a software team manage changes to source code over time.

Software developers working in teams are continually writing new source code and changing existing source code. The code for a project, app or software component is typically organized in a folder structure or "file tree". One developer on the team may be working on a new feature while another developer fixes an unrelated bug by changing code, each developer may make their changes in several parts of the file tree. Version control helps teams solve these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting.

In my project, I tried to implement a basic version control system. Although it does not supply merge operation that merges different versions into one version, it lets only the users that has the last version of the file.

**3. Aproach**

This project consists of two parts:

1. The Android Application: for client use
2. The JAVA Server: is implemented as multi-threaded to handle multiple requests from multiple clients

**3.1 Server Side**

Existing of multiple clients necessitates multi-threaded server to handle multiple requests come from multiple clients. The server needs to be able to handle multiple clients’ requests and process the requests and response to clients. Multi-thread server accepts the connection from a client and process each client connection in different threads. A new thread is created for each client request. This approach makes the system more faster and reliable.

A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester. When the listener socket of server catch a connection request from client, it accepts the request and create an active socket which all messages between client and server send and transfer on. Then give this active socket to a new thread. All requests and responses which are carried out on this socket process in the thread.

There are 3 types of requests the client can make to the server. The client sends one byte Action Code that is indicates that the client will make which request type.

Action.LIST indicates a request to list all files

Action.READ indicates a request to read the selected file

Action.UPDATE indicates a request to edit the selected file

*1. List all files on the database*

The server takes the one byte code which is indicate that the client wants to list all files on the database. Then the server read the “init.json” file under the “./db” path. The “init.json” file keeps all files name and its version number as a JsonObject. After the server read the file, send it to the client via DataOutputStream of the socket which is attached to the client socket. While the server send a message, it use “sendMessage(String message)” function I wrote. The function get a message string as parameter and convert the string to bytes, then write to the socket’s DataOutputStream. The client-side will explain next section.

*2. Read the content of the file indicated by id*

The server reads one byte code that means the client want to read a file. The server waits a message from client which contains id number of the file the client wants to read. The server get the message from client via “receiveMessage()” function I wrote. The function reads bytes while available from DataInputStream of the socket which is attached to the client socket. After the server receives the id number, read the indicated file by this id from under the “./db” path by using FileInputStream. When the server read the file, to prevent the possible race conditions that can be occur on the file because of that multiple clients do reading or writing on the file at the same time, I implemented the algorithm named “ReaderWriter”. I will explain it after. The content of the file is sent to the client. The server also sends the last version number of the file. The version number will be explained in its part.

*3. Edit the content of the file indicated by id*

I explained how the server get the file that the client indicates by id. After the server get the id number and read the content of the file, get clientVersion number is meant that last version of the file on the client. Then the server sends the serverVersion number of the file on the server. By this way, the server and the client know version numbers of each other. Both side take an action according to these version numbers. If the client back of the server, the server will tell the client that “please update your file to the last version, then make your changes”. Else, the server updates the content of the file by using FileOutputStream.

As I mentioned before, since all files are shared memories and all clients are using the same database simultaneously, race conditions can be occur. To prevent this problem, The ReaderWriter 2 solution taught in class have been implemented. The algorithm coordinates access to a shared database by multiple readers and writers. This algorithm is implemented using two binary semaphores.

**3.2 Client Side**

The client side is an Android Application developed by using native Android. I used 2 activity that are MainActivity and FileEditorActivity. Launcher activity is MainActivity that contains a ListView to list of all files on the database. A LIST request is made while the app starting. Then, the listview is filled.

Because of the rules of Android, you cannot make an Internet connection whatever you use to do that like socket or HTTP request in main thread. Since the main thread is for GUI, if you want to make an Internet connection or request, you must create a new thread. In my app, communication with the server is performed on the TCP Socket. Each socket connection is created and handled by “Client” class which extends to AsyncTask.

ListView object has setOnItemClickListener() functionality to handle the tapping an item on the list. When an item selected on the list, the new activity named FileEditorActivity is opened.

The FileEditorActivity contains an EditText to show and edit the file content and a Button to start and finish editting. While it is starting, a READ request is made with fileId that indicate that which file is selected on the listview. Then, the content of the file is received via DataInputStream of the client socket which is attached to the server socket. By the way, the client receives the last version of the file from the server. The EditText is on read mode on default. When the button is clicked, mode of the edittext change to write mode. The user can makes changes on the file. When the second clicking on the button, the client sends a UPDATE request to the server. While making the request, the client sends also fileId, clientVersion number of the file and the new content of the file. Also receive serverVersion number of the file from the server. All messages send and receive on the TCP Socket by using DataInputStream and DataOutputStream.

I have implemented a basic version control system. Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. In my project, the basic approach for version control is comparing clientVersion number and serverVersion number and act according to that. If the clientVersion number smaller than the serverVersion number, means that the file at the client is old, a GUI message is showed to the user to ask update the file. Else, the client sends the new content to the server to update file.

Since multiple clients read and write the same database, race conditions can be occur. To prevent this, I implemented ReaderWriter solution from the lecture notes. This solution allows a single writer or only readers and no writers. The solution is built with 2 Binary Semaphores.

* *No read-write conflict:* A reader and a writer do not access the database concurrently
* *No write-write conflict:* Two writers do not access the database concurrently

I used enum type to describe actions that a client can takes. An enum type is a special data type that enables for a variable to be a set of predefined constants. Java programming language enum types are much more powerful than their counterparts in other languages. The enum declaration defines a class (called an enum type). The enum class body can include methods and other fields. In my enum “Action”, each constant represents its name related request type I mentioned above. Actually, all constants are equal to a byte. I also defined a method to return the constant’s value as a byte.

* Action.LIST → *List all files on the database*
* Action.READ → *Read the content of the file indicated by id*
* Action.UPDATE → *Edit the content of the file indicated by id*

I created an interface to define a callback function that will be used for handling response come from socket connection AsyncTask completion. By this way, I can take reaction according to response where I call the client AsyncTask.

**4. Experimental Methodology**

In the beginning, I tought that I should built the server firstly. After complete the base of the server, I can start to the Android App. To test the server in this phase, I wrote basic TCPClient by using Java. It connect to the server via TCP Socket as will be in the Android App.

1. **TCP Connection Test:**

I printed the client IP adress which is accepted by the listener socket of the server

The function I used activeClient.getInetAddress().getHostAddress()

1. **Send and Receive Message Functionality Test:**

I wrote sendMessage(String message) and receiveMessage() functions to send message to the client and receive message from the client orderly and also vice versa. To test this functions, I implemented TCP Echo Server and TCP Echo Client.

1. **Reader Semaphore Test:**

I tested this property with 2 android clients. Any writers cannot come in the critical section while some readers in. No read-write conflict.

1. **Writer Semaphore Test:**

I tested this property with 2 android clients. While a writer in, any other writers cannot come in the critical section . No write-write conflict.

1. **List Request Test:**

I tested this property on both server and client side. I observed some steps to see success:

* List request of the client fall into the server console
* Names of the files on the database are wrote to the server console
* Receiving message is wrote to the client console
* Parsing the message and fill the ListView on the MainActivity

1. **Read Request Test:**

I tested this property on both server and client side. I observed some steps to see success:

* Read request of the client fall into the server console
* Id number of the selected file can be seeing on the server console
* Name of the selected file is equal to name of the tapping to ListView Item
* Readed content of the file at server is equal to the content that is printed the EditText on the FileEditorActivity
* A writer and reader cannot be in the critical section.

1. **Update Request Test:**

I tested this property on both server and client side. I observed some steps to see success:

* Update request of the client fall into the server console
* Updated content is showed on the EditText

**5. Related Work**

I tried to implement a basic version control system like Github. While Github offers merge operation that merges different versions into one version, my version control system server lets only the users that has the last version of the file. To do that before every read and update requests both server and client send their current version number of the selected file to each other. Then act as the comparison of the clientVersion number and the serverVersion number.

**6. Conclusion**

In this project I implemented a native Android App as Client-Side and a multi-threaded

Java server. The purpose of the project was provide basic version control system. I used multi-threaded server because there are multiple clients that will read and write data at the same time, so their requests should not block each other.

There were couple of problems when implementing the multi-threaded

server, since synchronous writes to the files could create a problem. I implemented the ReaderWriter solution to solve the possible problems.

**7. References**

<https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>

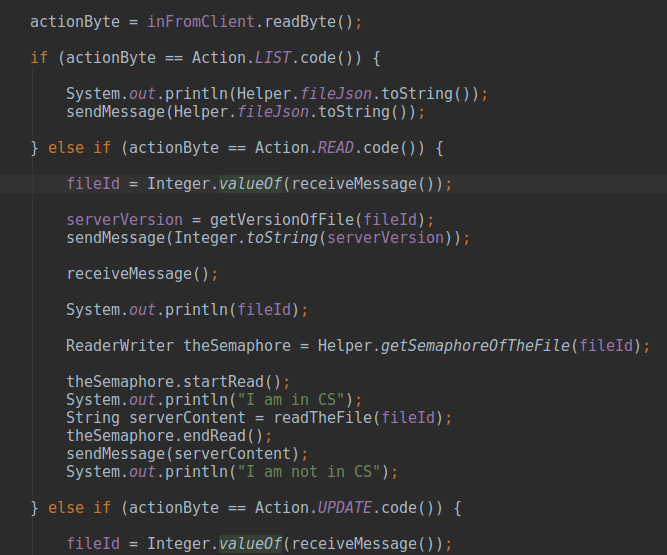
<https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html>

<https://www.atlassian.com/git/tutorials/what-is-version-control>

**8. Appendix**

***Server-Side***

******

******

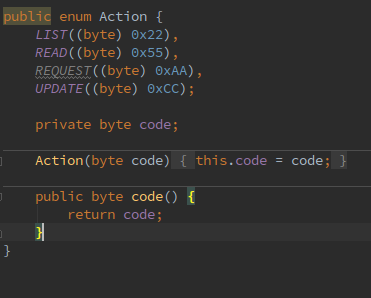
***Client-Side***

******

******

******

******

******