FLORIDA INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCES

CSE 4232/5232: Project Report - Milestone 04

Harshit Bhatt
Josemar Faustino da Cruz

May 6, 2015

1 INTRODUCTION

Enclosed is the software for milestone 4.

It contains all the functions that were implemented on milestones 2 and 3 and the requirements for milestone 4.

As per the listed requirements, the system was added the functionality to listen to UDP commands **REGISTER;GROUP** and **LEAVE;GROUP**. After receiving such commands, the server queries the database for events related to that group and writes then to the client in UDP messages during one hour, or until the client sends the **LEAVE;GROUP** command.

2 ARCHITECTURE

Both client and server are written in Java and was tested with JDK version 1.7 and higher. The code is not supported in earlier JDK versions.

2.1 CLIENT

The client is organized as follows:

1. Getopt parameters parsing (port number and database path);
2. Encodes the data into ASN1 with implicit TAG values;

3. Sends the data via Socket, either TCP or UDP to the Server;
4. Gets the answer from the Server;
5. Decodes the ASN1 and displays;

In this milestone, the client was added the functionality of sending REGISTER;GROUP or LEAVE;GROUP over UDP. This is done in a separated thread.

After starting receiving the events that it registered for, if the client wants to stops the messages, a new client with the command LEAVE should be started.

We treated it separately, because since the client keeps on printing events, it is difficult to type the leave command and send to the server.

In the new client, if the LEAVE;GROUP command has a different group, the server will not stop sending the data.

2.2 SERVER

The server is organized as follows:

1. Getopt parameters parsing (port number and database path);
2. Database creation if it doesn't exists;
3. Call the TCP constructor and start the TCP thread which calls tcp_connect();
4. Call the UDP constructor and run the udp_connect();
5. Funtion tcp_connect() handles TCP connections and client input which is the parameter to call function display(), and calls expireCheck() to update the event in case it is expired as well as decoding the ASN1 from the client;
6. Function udp_connect() handles UDP connections and client ASN1 input over UDP and also calls function display() and function expireCheck(). On udp_connect(), the Thread to handle REGISTER commands is started;
7. Function display() receives an array of STRINGS which is the organized input from the client, process them and return ASN1 encoded data either to tcp_connect() or udp_connet().

For

The server architecture is better showed in Figure 2.1, and the client architecture is showed in Figure 2

Instructions on how to run the server are given in Section 3

For the server, the initial parameters are defined in the GetOpt object are -t which is the TCP port number, -u which is the UDP port number, and -d which is the database number. The file run.sh call the files created in the compilation as well as set the proper classpath with the sqlite JDBC driver jar file. As required in the project instructions, the server listens to both TCP and UDP in the same port.

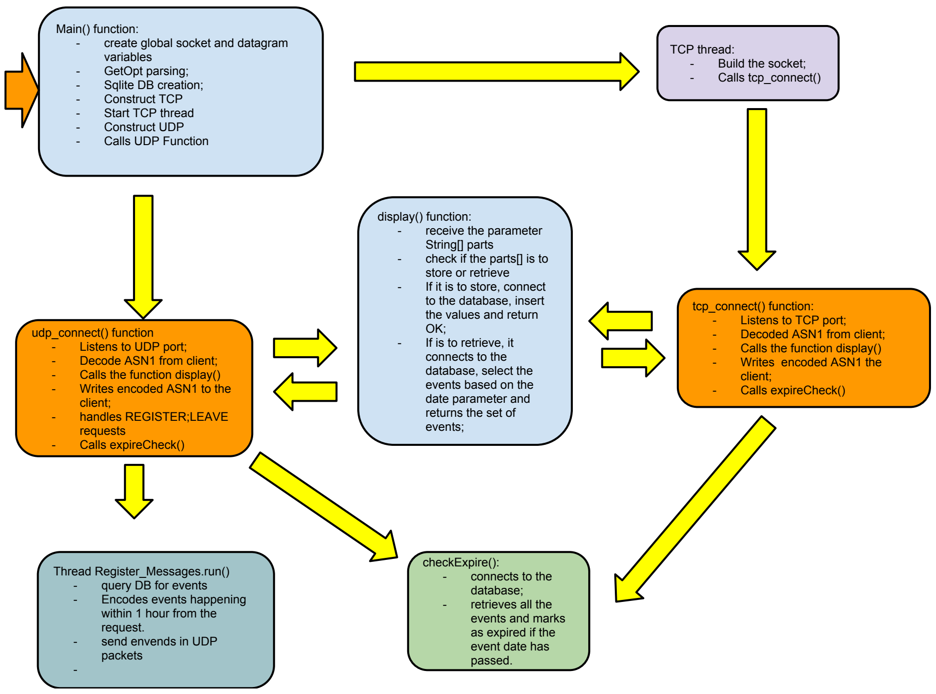
Main() function:
- create global socket and datagram variables
- GetOpt parsing;
- Sqlite DB creation;
- Construct TCP
- Start TCP thread
- Construct UDP
- Calls UDP Function
TCP thread:
- Build the socket;
- Calls tcp_connect()
display() function:
- receive the parameter String[] parts
- check if the parts[] is to store or retrieve
- If it is to store, connect to the database, insert the values and return OK;
- If is to retrieve, it connects to the database, select the events based on the date parameter and returns the set of events;
udp_connect() function
- Listens to UDP port;
- Decode ASN1 from client;
- Calls the function display()
- Writes encoded ASN1 to the client;
- handles REGISTER;LEAVE requests
- Calls expireCheck()
tcp_connect() function:
- Listens to TCP port;
- Decoded ASN1 from client;
- Calls the function display()
- Writes encoded ASN1 the client;
- Calls expireCheck()
Thread Register_Messages.run()
- query DB for events
- Encodes events happening within 1 hour from the request.
- send envends in UDP packets
-
checkExpire():
- connects to the database;
- retrieves all the events and marks as expired if the event date has passed.

Figure 2.1: Server Architecture

For the client, the options are also parsed in GetOpt, being -s for server (e.g. localhost); -t if using TCP or -u if using UDP, both containing the port number; -i for the input that is being sent;

If the client is receiving a **REGISTER** command, it will ignore the -t (for TCP port) option and sends the data over UDP.

We provide a populated database with data for testing the **REGISTER** and **LEAVE** commands. For **REGISTER** command, the server will reply via UDP to the client, events that are happening within 24 hours from the request, and this UDP messages will last for 1 hour or until the client sends a **LEAVE** command.

The DB name can be altered in the run.sh file.

3 User Manual

3.1 Client

The program is contained in a folder called harshit_bhatt_josemar_dacruz.tgz. After decompression, enter the created folder and run the command *make* to compile the source files.

To start the server, inside the folder run the command: `./run.sh 2356`, where 2356 is the port number. This will start the server, which will listen on both TCP and UDP on port 2356.

The functionalities for milestones 2 and 3 are kept and can be tested using these commands:

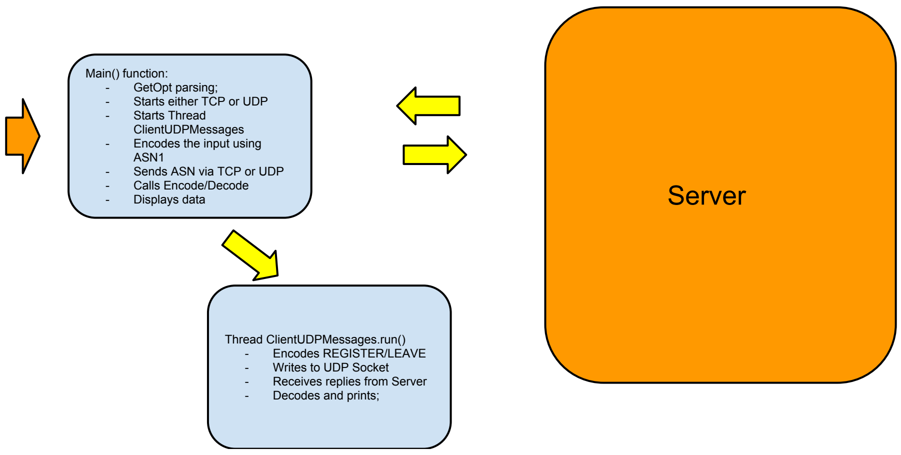
Main() function:
- GetOpt parsing;
- Starts either TCP or UDP
- Starts Thread ClientUDPMessages
- Encodes the input using ASN1
- Sends ASN via TCP or UDP
- Calls Encode/Decode
- Displays data
Thread ClientUDPMessages.run()
- Encodes REGISTER/LEAVE
- Writes to UDP Socket
- Receives replies from Server
- Decodes and prints;
Server

Figure 2.2: Client Architecture

`client.sh localhost 2356 ‘‘EVENT_DEFINITION;2015-03-12:18h30m00s001Z;Meeting with the user.;CSE5232’’`
If the input is successful, the server will answer with the number 0, if not it will return a number 1;

The client can also be tested using the command
`client.sh localhost 2356 ‘‘GET_NEXT_EVENTS;CSE5232;2015-03-12:18h30m00s000Z’’`
In this case the server will reply back a list of EVENTS that occurs after the specified date; Both client and server sends the data encoded in ASN1, and upon receiving, print the input that was sent over the network;

By default, the client will connect to the server over TCP. But it is also capable of connecting via UDP, in order to do that, you need to edit the file `client.sh` and change the option `-p` to `-u`;

For this milestone, the new functions can be tested as follows: The functionalities for milestones 2 and 3 are kept and can be tested using these commands:
`client.sh localhost 2356 ‘‘CSE5232’’`
Right after receiving that command, the server replies with an EventOK ASN1 object which contains code = 0, if the server accepted and started the UDP messages, or code $\neq 0$ if it's not going to reply.

The server only sends messages for one group and for one client at the time. If it's busy, it will reply and EventOK with code $\neq 0$. It code has a meaning, and the client prints what as the reason the server did not replied.

If the server is idle, it will reply with UDP messages all the events in group CSE5332 that are happening within 24 hours. If the server is busy, or if there are no events with such group, the server answers;

If the user wants to stop the messages, it should start a new client and send the proper LEAVE command with the same group it was registered before. Then the server quits sending messages and acknowledges with a EventOK which contains the reason why it stopped, then the client prints it.

4 Fixed Issues from last version

The following issues were present last time and are now removed:

- Implicit TAGs not being set.
- java.NullPointerException when running in command line.