CSE5232 - Network Programming Project

Nima Aghli and Fitzroy Nembhard

March 18, 2016

Class Project – Milestone 3

1   INTRODUCTION

This project features a UDP and TCP server that process commands associated with projects and tasks. Our solution has been implemented using the Java programming language. The user is expected to send messages in the form of strings to the server over TCP or UDP connections. Once activated, the server listens for connections over a certain port, and clients then send messages to carry out certain transactions. If the messages follow a certain format required by the server (as specified in Section 2.1), data is then saved to a SQLite database and confirmation messages are returned to the client. If a transaction is unsuccessful, a FAIL message is returned to the client.

2   ARCHITECTURE

Figure 1 below outlines the components used in our project and Figure 2 shows the schema of the SQLite database we use to store the data for the projects and tasks.

2.1   Messages

2.1.1   PROJECT_DEFINITION

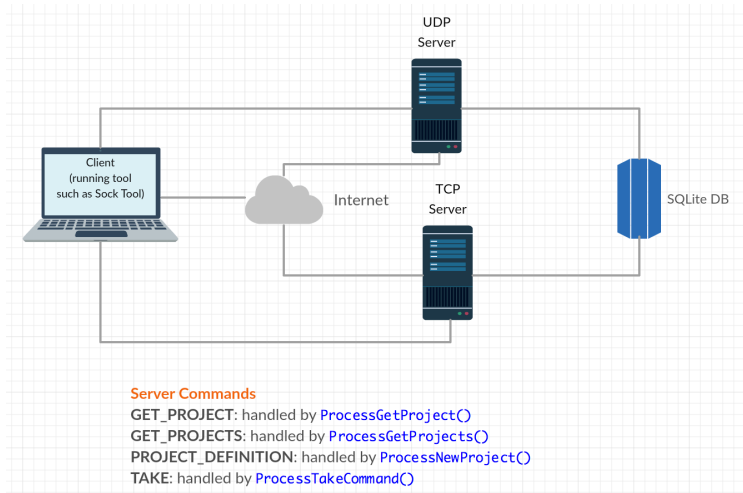User sends a request to define a project along with a list of tasks for that project:
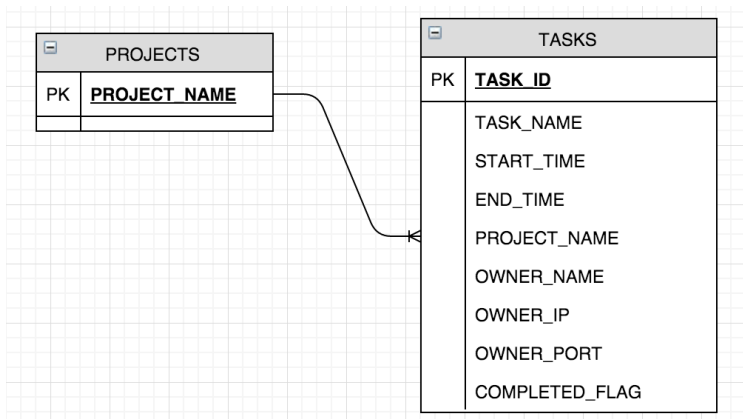
Figure 1: The System Architecture

Figure 2: Entity Relationship Diagram

```
────────────────────── PROJECT_DEFINITION Format ──────────────────────
PROJECT_DEFINITION:[ProjectName];
TASKS:[TaskCount];
[TaskName];
[TaskStartTime];
[TaskEndTime]...
────────────────────────────────────────────────────────────────────────
```

Time Format: YYYY-MM-DD:HH'h'MM'm'SSfffZ

```
———————————————————— PROJECT_DEFINITION Exmple ————————————————————
PROJECT_DEFINITION:Exam;
TASKS:2;
Buy paper;
2016-03-12:18h30m00s001Z;
2016-03-15:18h30m00s001Z;
Write exam;
2016-03-15:18h30m00s001Z;
2016-03-15:18h30m00s001Z;
```

Server Response: OK;PROJECT_DEFINITION:Exam

2.1.2   TAKE

User requests to assign a specific task to a user for a particular project.

```
———————————————————— TAKE Command Format ————————————————————
TAKE;
USER:[UserName];
PROJECT:[ProjectName];
[TaskName]}
```

```
———————————————————— TAKE Command Example ————————————————————
TAKE;
USER:Johny;
PROJECT:Exam;
Buy paper
```

Server Response: OK;TAKE;USER:Johny;PROJECT:Exam;Buy paper

2.1.3   GET_PROJECTS

Server returns a list of all projects in the database to the client.

```
———————————————————— GET_PROJECTS Format ————————————————————
GET_PROJECTS
```

```
———————————————————— GET_PROJECTS Example ————————————————————
GET_PROJECTS
```

Server Response: OK;PROJECTS:2;Exam;Enigma

2.1.4 GET_PROJECT

User requests a list of all tasks assigned to a specific project.

```
──────────────────── GET_PROJECT Format ────────────────────
GET_PROJECT;[ProjectName]
```

```
──────────────────── GET_PROJECT Example ────────────────────
GET_PROJECT;Exam
```

Server Response: OK;PROJECT_DEFINITION:Exam;TASKS:2;
Buy paper;2016-03-12:18h30m00s001Z;2016-03-15:18h30m00s001Z;
Johny;10.0.0.1;2501;Done;
Write exam;2016-03-15:18h30m00s001Z;2016-03-15:18h30m00s001Z;
2016-03-18:18h30m00s001Z;2016-03-15:20h30m00s001Z;
Mary;10.0.0.2;2505;Waiting

2.2 Classes

There are 9 classes in the project as outlined below:

2.2.1 DBManager

This class is used to facilitate connections to a SQLite database.

Method Detail
public Connection connectToDB()
This method attempts to connect to the SQLite database selected.
Returns:
a SQL connection to the database

2.2.2 ProcessGetProject

This class handles GET_PROJECT commands, which are used to retrieve information about a particular project.

Method Detail
```
public static String getProject(String message, String dbPath)
```
This method returns the list of tasks associated with a project.
Parameters:
```
message - the GET_PROJECT command string
dbPath - the path to the database
```
Returns:
```
a string specifying success (OK) if command is valid or otherwise (FAIL)
```


2.2.3   ProcessGetProjects

This class handles GET_PROJECTS commands, which are used to fetch the number of projects along with their names/titles to the client.

Method Detail
```
public static String getProjects(String message, String dbPath)
```
This method returns the total number of projects along with a list of names of all the projects in the database.
Parameters:
```
message - the GET_PROJECTS command string
dbPath - the path to the database
```
Returns:
```
a string specifying success (OK) if command is valid or otherwise (FAIL)
```


2.2.4   ProcessNewProject

This class processes new project definitions that are sent to the client using the PROJECT_DEFINITION command.

Method Detail
```
public static String addNewProject(String message, String dbPath)
```
This method adds a new project and a list of tasks to the database.
Parameters:
```
message - the PROJECT_DEFINITION command string
dbPath - the path to the database
```
Returns:
```
a string specifying success (OK) if command is valid and successful or otherwise
(FAIL)
```


2.2.5   ProcessTakeCommand

Handles TAKE commands sent by the client in order to assign a person to a project and define their specific task for the specified project, which is then marked as completed by the server. The following parameters are updated in the database upon successful completion of the TAKE command:
OWNER_NAME

OWNER_IP
OWNER_PORT
COMPLETED_FLAG

Method Detail
`public static String take(String message, String dbPath, String IP, int port)`
This method assigns a task to the selected user and updates the status of the task in the database.
Parameters:
`message - the TAKE command string`
`dbPath - the path to the database`
`IP - the IP address of the user`
`port - the port on which the user connected`
Returns:
an OK message if action was successful or a FAIL message if action failed.

2.2.6   TCPThread

Provides functions for running a thread that listens and processes messages over TCP.

Method Detail
`@Override public void run()`
This is the overridden method that runs when the thread starts. It processes all the commands that are associated with projects and tasks and passed to the server over TCP connections.

2.2.7   UDPThread

Provides functions for running a thread that listens and processes messages over UDP.

Method Detail
`@Override public void run()`
This is the overridden method that runs when the thread starts. It processes all the commands that are associated with projects and tasks and passed to the server over UDP connections.

2.2.8   Util

This class facilitates auxiliary functions such as date parsing and validation.

Method Detail
`public static boolean isDateValid(String dateStr)`
This method checks if a date string matches a specified date regular expression.
Parameters:
`dateStr - the date string`
Returns:
`true if date matches date regular expression (regex), false otherwise`

```
public static String formatDate(String dateStr)
```
This method formats a date so that it can be parsed by Java SimpleDateFormat function and used for date comparison

Parameters:
```
dateStr - the date-string to parse
```
Returns:
```
a reformatted date that is parsable by SimpleDateFormat
```

```
public static boolean isValidDateRange(String startDateStr, String endDateStr)
```
This method checks if an end-date is >= to start-date

Parameters:
```
startDateStr - the start-date string
endDateStr - the end-date string
```
Returns:
```
true if end-date >= start-date
```

2.2.9   RunServer

This is the main class to run the server. It is responsible for opening the server socket on a port specified by user and connecting to a database, which is also specified as an argument by the user. If the database already exists, the server will attempt to open and use it for storing data, otherwise a database will be created and two tables will be added to store the respective tasks and projects.

Method Detail
```
public static void main(String[] args)
```
The runServer main function expects two parameters in order to run successfully. These parameters are explained in the program execution in Section 3.2.

3   USER MANUAL

3.1   Compilation

This command uses *make* to compile the java files.
```
$ ./compile.sh
```

The following command deletes/cleans all class files and deletes the database associated with the project in order to create a new system.
```
$ ./compile.sh -c
```

3.2   Execution

This command allows the user to specify a port (<p>) and path (<d>) to the SQLite database to be used for the project.

```
$ ./run.sh -p <p> -d <d>
```

The following shortened command executes the program by using a default database that is specified in the bash script. The argument <p> represents a port number on which the server should run.

```
$ ./run.sh <p>
```
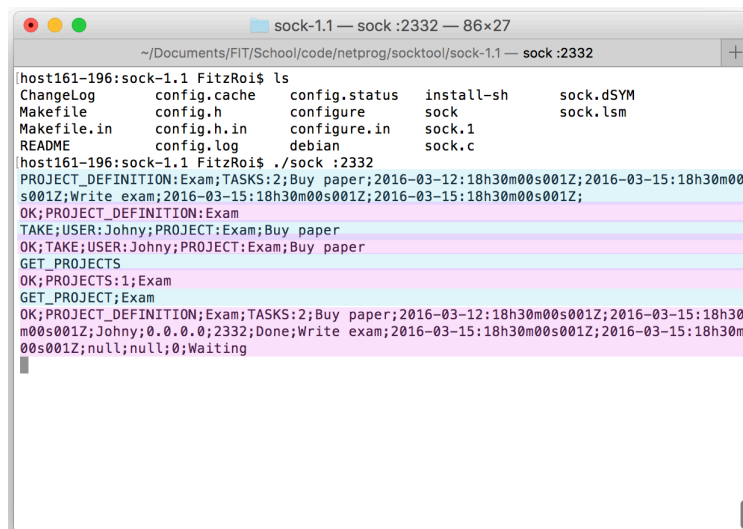
Figure 3 presents a snapshot of a successful compilation and execution of the program.

Figure 3: Screenshot of Successful Compilation and Execution of the Program

Figure 4 shows the results of a successful communication between the *sock* tool (client) and the server. The client messages are highlighted in blue and the server in purple.

Figure 4: Screenshot of Successful Communication between Client (Sock tool) and Server

Figure 5 shows an unsuccessful communication between the client and server. The malformed commands are highlighted in blue and the FAIL messages from the server are highlighted in red. It can be seen that the first PROJECT_DEFINITION command contains an invalid date-range (i.e, the end-date is earlier than the start date), so this results in a failure. Additionally, the other commands are incomplete and are rejected by the server.

8

Figure 5: Screenshot of Un-successful Communication between Client (Sock tool) and Server

The following image shows the result when the user tries to issue a TAKE command on an expired task. The portion highlighted in blue shows the due date of the task and the portion highlighted in red shows the result of attemping to complete an expired task.

Figure 6: Screenshot of Failed TAKE Command Due to Expired Task

4   CONCLUSION

This project is milestone 3 of the Network Programming project for Spring 2016. We have successfully programmed two servers that listen over TCP and UDP and store locally in a SQLite database the set of tasks associated with projects. Users are able to issue PROJECT_DEFINITION, TAKE, GET_PROJECTS, an GET_PROJECT commands over TCP/UDP to the server. If the commands are formulated correctly according to the format specified in this document, the requests will be successful and OK messages will be returned to the user, otherwise the requests will be unsuccessful and FAIL messages will be returned to the user.