Introduction

This project implements a basic server client event server. Two basic commands are implemented:

Commands
As the requirements specified, our Server and Client support the following commands.

- "***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;***"

- To define the project "**Exam**", composed of two tasks. Each task is given as a sequence of three components (name, start time, end time). The first task is "***Buy paper***" starting on "***2015-03-12:18h30m00s001Z***", i.e., 12/03/2015 at 18h30 and 1 millisecond, GMT, and ending on "***2015-03-15:18h30m00s001Z***", i.e., 15/03/2015 at 18h30 and 1 millisecond, GMT. The second task is "***Write exam***" starting on "***2015-03-15:18h30m00s001Z***", i.e., 15/03/2015 at 18h30 and 1 millisecond, GMT, and ending on "***2015-03-15:20h30m00s001Z***", i.e., 15/03/2015 at 20h30 and 1 millisecond, GMT.
- On success, the server answers with "***OK;PROJECT_DEFINITION:Exam***", i.e. by prepending "OK;" to the first component of the received message. With TCP, the string is terminated either by newline (\n) or by the closing of the stream. Any other answer from the server will be considered a failure.
- On any failure the server returns "***FAIL***;..." where the remaining components after ***FAIL*** list the exact received request being answered.

"***TAKE;USER:Johny;PROJECT:Exam;Buy paper***"

- To declare that the user ***Johny*** commits to accomplish the task "***Buy Paper***" for the Project "***Exam***".
- The server replies on success with: "***OK;TAKE;USER:Johny;PROJECT:Exam;Buy paper***"

"***GET_PROJECTS***"

- To ask the server the list all the projects.
- On success the server answers with: "***OK;PROJECTS:2;Exam;Enigma***", i.e., with "OK" followed by a component specifying the number of projects (2), followed by the list of the project names.

"***GET_PROJECT;Exam***"

- To retrieve the whole set of tasks relevant for the project "***Exam***"

- If the project is found, the server answers with a string "***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***", i.e. similar to the message defining the project, but each task gets 4 extra attributes (name owner, IP owner, port owner, flag completed). The flag for completed task is one of: ***Done,Waiting***. We assume that each task undertaken by somebody was accomplished at the corresponding end time. The name, IP and port of the takers are recorded from TAKE commands.

    - "REGISTER;Exam" and "LEAVE;Exam" in ASN1:
        - Register ::= [13] SEQUENCE {project UTF8String}
        - Leave ::= [14] Register
    - There must be a single program for the client, and it must distinguish between tcp and udp mechanisms based on a -t or -u option.
    - The client must interprete the domain parameter passed with option "-d" (e.g., "olin.fit.edu" in examples), which specify the host of the server
    - After the first among commands, the server should immediately reply by UDP with ***ProjectOK***, and then announce by a UDP message each event that is happening and is relevant to the mentioned *project*, to the socket from which the register message arrived. Such messages must be encoded with ASN1 type ***Project***. Announcements should stop after the **Leave** command, or after 1 hour from the register command.
    - For java programmers, you must have a separate file/class for each ASN1 structure (***Task, Project, ProjectOK***, **Register**, **Leave**, etc)
    - The client should print on standard output the obtained decoded messages.
    - The client should be started into this mode using your script modified to support:
        - ***client.sh olin.fit.edu 2356 "Exam"***
        - In which case it sends **Register** on start and **Leave** when exiting

Architecture

This section will attempt to provide a general overview of the program, followed by a listing of functions by enclosing class.

General Overview:

Main thread parses arguments, getting the port number to listen on and the file path to the SQLite database, after which it spawns two threads: one for TCP and another for UDP, waiting on when those threads finish. The TCP thread listens for incoming connections and spawns a new handler thread for each connection. There is no hard limit on how many concurrent TCP connections can be handled, besides TCP server having a queue of unhandled connection being set to 50 connections, after which it starts dropping new connections. The UDP thread listens for incoming datagrams and handles them as they come. It doesn't spawn any additional threads to handle them. When TCP or UDP handlers receive some data, they try to parse, look which of commands we are asked to execute, after which the storage class is accessed, and specified events are either stored or retrieved.

Class and Function List

- **Server.java**
  - This is the server class where the main server method is. It handles parsing of the command line arguments with the help of JavaCC. It also spawns threads for TCP and UDP as required.
- **Client.java**
  -
- **Database.java**
  - Contains the tables where data from the Projects and Clients registered to the server.
- **ASNEVENT.java**
  - When an asn object is sent to the client or server, they do not know which asn object it is. The ASNEvent contains a constant that tells the server or client how that object should be decoded. This way the client and server will always be decoding an asn event.
- **ASNLeave.java**
  - Contains the group name of the project to start listening for.

  -
    ```
    Leave ::= [2] SEQUENCE {
            Group UTF8String
    }
    ```
- **ASNProject.java**

  -
    ```
    Project ::= [1] SEQUENCE {
            name UTF8String,
    ```

```
        tasks SEQUENCE OF Task
    }
```

- **ASNProjectOK.java**
  - Returns 0 on success and nonzero for error

  -
    ```
    ProjectOK ::= [1] SEQUENCE {
            Code INTEGER
    }
    ```
- **ASNProjects.java**
  - Projects ::= [2] SEQUENCE {}
- **ASNProjectsAnswer.java**

  -
    ```
    ASNProjectsAnswer ::= [1] SEQUENCE {
            projects SEQUENCE OF ASNProject
    }
    ```
- **ASNRegister.java**

  -
    ```
    Register ::= [13] SEQUENCE {
    project UTF8String
    }
    ```
- **ASNTake.java**

  -
    ```
    Take ::= [5] SEQUENCE {
            user UTF8String, project UTF8String, task UTF8String
    }
    ```
- **ASNTask.java**

  -
    ```
    Task ::= [1] SEQUENCE {
            name UTF8String,
            start GeneralizedTime,
            end GeneralizedTime,
            user UTF8String,
            ip UTF8String,
            port INTEGER,
            done BOOLEAN
    }
    ```

User Manual
To compile the code run the make command.

Server Scripts:

- Run.sh <port>
    - Launch the server to listen for clients.

Client Scripts:

- Script1udp.sh
    - creates a project, assigns a user, prints all the projects and the new project.
- Script2udp.sh
    - retrieves a project
- Script3register.sh <domain> <port> <group>
    - Registers the client to listen for any action relating to the specified group.
    - Ctrl-c stops the client and sends the leave command to the server.

Conclusion: