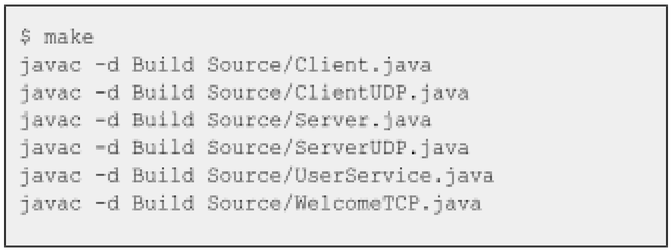
COMP3331 assignment

Project structure

Assignment is written in java. The project is compiled with a Makefile, and can be done so by running "make" in the assign directory.

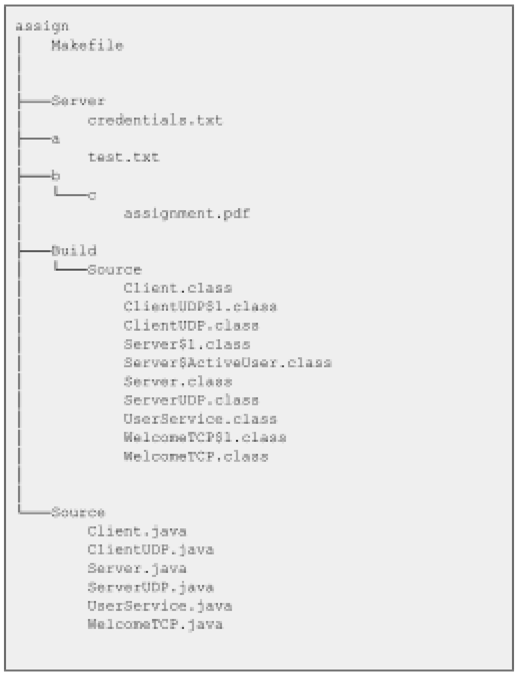


```
$ make
javac -d Build Source/Client.java
javac -d Build Source/ClientUDP.java
javac -d Build Source/Server.java
javac -d Build Source/ServerUDP.java
javac -d Build Source/UserService.java
javac -d Build Source/WelcomeTCP.java
```

The build files will end up in Build/Source/ as *.class files.
To run Server.class, *cd into the starting directory of the server (assuming the starting directory is a direct child folder of assign)* and run:
`$ java -cp ../Build Source.Server <port>`
or
`$ java -cp ../Build Source.Server`
Which defaults the server port to 54321.

To run Client.class, *cd into the starting directory of a client (assuming the starting directory is a direct child folder of assign)* and run:
`$ java -cp ../Build Source.Client <port>`
or
`$ java -cp ../Build Source.Client`
Which defaults the server port to 54321.

Below is a sample of what the directory might look like after compilation and execution of Server.class and Client.class in the JVM (with client files already in place):



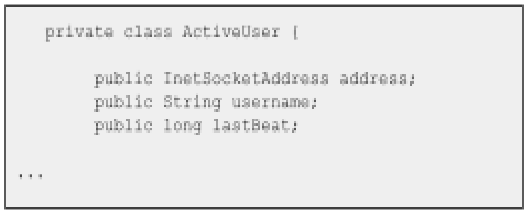
```
assign
│   Makefile
│
│
├───Server
│       credentials.txt
├───a
│       test.txt
├───b
│   └───c
│           assignment.pdf
│
├───Build
│   └───Source
│           Client.class
│           ClientUDP$1.class
│           ClientUDP.class
│           Server$1.class
│           Server$ActiveUser.class
│           Server.class
│           ServerUDP.class
│           UserService.class
│           WelcomeTCP$1.class
│           WelcomeTCP.class
│
│
└───Source
        Client.java
        ClientUDP.java
        Server.java
        ServerUDP.java
        UserService.java
        WelcomeTCP.java
```

Program Design & State Information

<u>Client.java</u> is the entry point of the Client program. It is responsible for reading and validating user input before passing the inputs down to a UDP socket and displaying the response on the Client side. It is also responsible for managing the lifetime of the UDP socket and TCP welcome socket, opening them on startup and closing them on exit. Onan authentication request, the client will pass the port and address of the TCP welcome socket to the server.

<u>ClientUDP.java & ServerUDP.java</u> are wrappers for the UDP socket class provided by java.net. Client.java and Server.java use them respectively to pass and receive messages to each other via UDP. Additionally, ClientUDP manages the lifetime of the heartbeat mechanism that sends pings to the Server every 2 seconds starting upon authentication. The nominal buffer size of UDP packets is 1024 bytes for both sending and receiving.

<u>Server.java</u> is the entry point of the Server program. It is responsible for processing client requests before passing the response down to a UDP socket. It also stores the state of active users via a struct with the fields;



```
   private class ActiveUser {

        public InetSocketAddress address;
        public String username;
        public long lastBeat;

...
```

Which identifies the TCP port of a user, username and the last heartbeat sent upon startup of the client, measured in milliseconds. The server spawns a thread that periodically checks with respect to System.currentTimeMillis() to determine if 3 seconds has passed and a user needs to be removed.
A hashmap<SocketAddress, ActiveUser> is used to map socket addresses of incoming requests to active users. Server.java holds a UserService object that it uses to query the state of both inactive and active user information. The server is also responsible for managing the lifetime of the server UDP socket.

<u>UserService.java</u> is where the state of user credentials and user file (file paths) are stored.
A hashmap<String, String> is used to map usernames to passwords.
A hashmap<String, List<String>> is used to map usernames to a list of user files.

<u>WelcomeTCP.java</u> is the TCP welcome port for clients. It is responsible for listening to download requests from other clients and spawning new threads & connection sockets to accommodate downloads. It also holds the download method used on a get request.

Message format

Messages passed for both UDP and TCP are formatted as strings. Messages via UDP from the client to the server follow the format:

```
Command arg1 arg2…
```

and will be deserialised into a List<String> before being processed. Messages via UDP from server to client follow no format, as the client will automatically display the server response except when responding to a “get” response.

Messages via TCP from sendee to sender take the format:

```
<Filepath>
```

Messages via TCP from sender to sendee take the format:

```
<Filelength> <Rest of file>
```

Known limitations

- The IP addresses of clients are hardcoded as localhost in the Server database.
- Users upon ‘xit’ will not send a message to the Server to be unlisted and instead will be removed after the 3 second timeout on the Server-side check of the heartbeat mechanism.
- The server will not gracefully shutdown.

References

UDPClient.java
UDPServer.java
TCPClient.java
TCPServer.java
https://webcms3.cse.unsw.edu.au/COMP3331/24T3/resources/103456

Were used as references when writing the main portions of socket code. I.e sending and receiving functions for UDP requests and TCP downloads.