ID:0421312032

Client-server Programming

1. In client side A socket is created using the IP address of localhost and TCP port number as parameter of `Socket()`.

```
Socket s = new Socket(ip, ServerPort);
```

2. Server bind a port to the socket.

```
ServerSocket ss = new ServerSocket(1234);
```

   Server runs infinite loop for getting client request. accepts the incoming request from client using `accept()`.

```
s = ss.accept();
```

3. When new client request received, server creates a new handler for handling this client.

```
ClientHandler mtch = new ClientHandler(s, "client-" + i, i, dis,
dos);
```

   A new thread is also created with this object. Multithreaded system is used for handling multiple clients.

```
Thread t = new Thread(mtch);
```

   By adding this client to the list of active client lists, the thread is started.

```
t.start();
```

4. In server side, the method `sendClientRoll(int id, DataOutputStream dos)` is used for sending roll and key block number to the client. DataInputStream and DataOutputStream are used for handling input and output.

5. For finding the time of server's response to the first client, Date() is used.

```
startDateTime = new Date();
```

6. In server side, ClientHandler class implements the Runnable interface.

7. In client side, getMd5(String input) method is used as hashing algorithm. This method converts the roll no into 32 characters (hash value). Static getInstance() method is called with hashing MD5.

```
MessageDigest md = MessageDigest.getInstance("MD5");
```

`digest()` method is called to calculate message digest of an input `digest()` return array of byte.

```
byte[] messageDigest = md.digest(input.getBytes());
```

The byte array of message digest is converted into signum representation.

```
     BigInteger no = new BigInteger(1, messageDigest);
```

And finally the message digest is converted into hex value

```
String hashtext = no.toString(16);
    while (hashtext.length() < 32) {
        hashtext = "0" + hashtext;
    }
    return hashtext;
```

8. After completing communication, connection is closed using `close()`.

```
this.dis.close();
this.dos.close();
```

Output and the time required to finish execution

- The time required to finish execution are given as follows:

```
Time = server receives response for all 1024 blocks – time of first
request from the first client
```

- when Number of client = 1

  Time = 6152765.893 milliseconds

  Output:
  https://drive.google.com/file/d/1rP7M5CdfzwQr8foT3ADta3K4zT5ofRES/view?usp=sharing

- when Number of client = 2

  Time = 3222709.173 milliseconds

  Output:

  https://drive.google.com/file/d/1SSIcMJ9RciTOf0OW7lwVneXE4wYXZO58/view?usp=sharing

- when Number of client = 4

  Time = 1819372.952 milliseconds

  Output:

  https://drive.google.com/file/d/1kW3w3PmNm1egBZyQilBrCf8o1ZAWBGXt/view?usp=sharing

- when Number of client = 8

  Time = 1539514.624 milliseconds

  Output:

  https://drive.google.com/file/d/19RZlT46qeOYEtesmOIfq86bjA5uMx1n5/view?usp=sharing