**ADVANCED COMPUTER NETWORKING SECURITY**

**CONFERENCE APPLICATION**

**HOMEWORK 2**

**KISHORE VALLAPUNENI (700639379)**

**SANTHOSH KUMAR VIDALA (700639887)**

**SAIPRASANNAREDDY BOGGULA (700641020)**

**Aim:**

Instant Messaging with Client-Server Architecture.

**Objective:**

The main objective is to Broadcasting messages from the clients to all the rest of the clients connected to the server.

**Code Implementation:**

The classes that we used are:

1) confServer

2) confUser

**Functioning of classes:**

**confServer:**

It stores all the client connections / disconnections and we can provide user defined port number using command prompt.

We used the hash mapping for storing the Ip address and port number of each clients in the server.

**confUser:**

Once connected , messages can be sent/received to /from all users via the server.

**Code for Server:**

**(This chat server can accept up to maxClientsCount clients' connections)**

private static final int maxClientsCount = 10;

private static final clientThread[] threads = new clientThread[maxClientsCount];

**(Open a server socket on the portNumber (default 2222). Note that we can not choose a port less than 1023)**

try {

serverSocket = new ServerSocket(portNumber);

} catch (IOException e) {

System.out.println(e);

}

**(Create input and output streams for the client)**

inputStream = new DataInputStream(cSocket.getInputStream());

outStream = new PrintStream(cSocket.getOutputStream());

**(Adding a new the client)**

outStream.println("Welcome " + name

+ " to our chat room.\nTo leave enter \’/quit\’ in a new line.");

synchronized (this) {

for (int i = 0; i < maxClientsCount; i++) {

if (threads[i] != null && threads[i] == this) {

clientName = "Whisper" + name;

break;

}

for (int i = 0; i < maxClientsCount; i++) {

if (threads[i] != null && threads[i] != this) {

threads[i].outStream.println("New user " + name

+ " entered");

**(Close the output stream, close the input stream, close the socket)**

inputStream.close();

outStream.close();

cSocket.close();

**Code for Client:**

**(Open a socket on a given host and port. Open input and output streams)**

try {

clientSocket = new Socket(host, portNumber);

inputLine = new BufferedReader(new InputStreamReader(System.in));

os = new PrintStream(clientSocket.getOutputStream());

is = new DataInputStream(clientSocket.getInputStream());

} catch (UnknownHostException e) {

System.err.println("Don't know about host " + host);

} catch (IOException e) {

System.err.println("Couldn't get I/O for the connection to the host "

+ host);

**(Create a thread to read from the server)**

new Thread(new ConfUser()).start();

while (!closed) {

os.println(inputLine.readLine().trim());

**Challenging part:**

To design peer-to-peer Architecture is challenging task for us and we tried but we are out of time.