

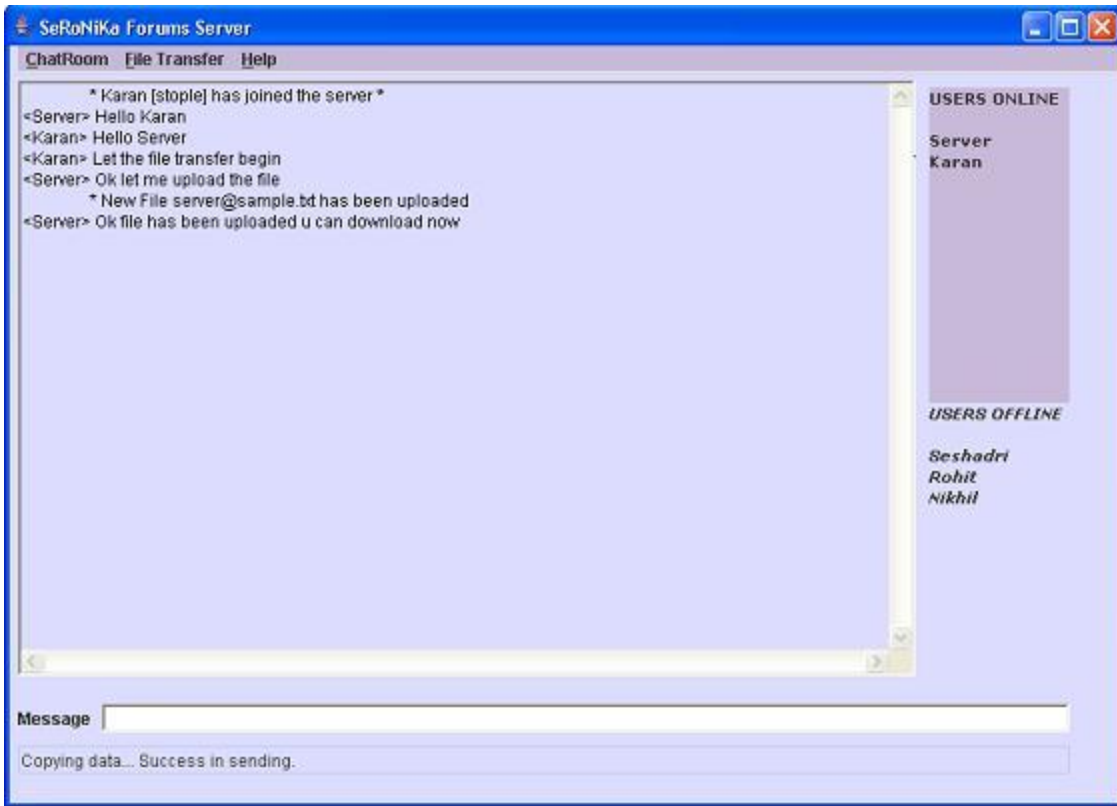
SeRoNiKa Forums

Project by: 1) Karan Kamdar (D15-18)
2) Seshadri N (D15-33)
3) Rohit K Singh (D15-30)
4) Nikhil G (D15-18)

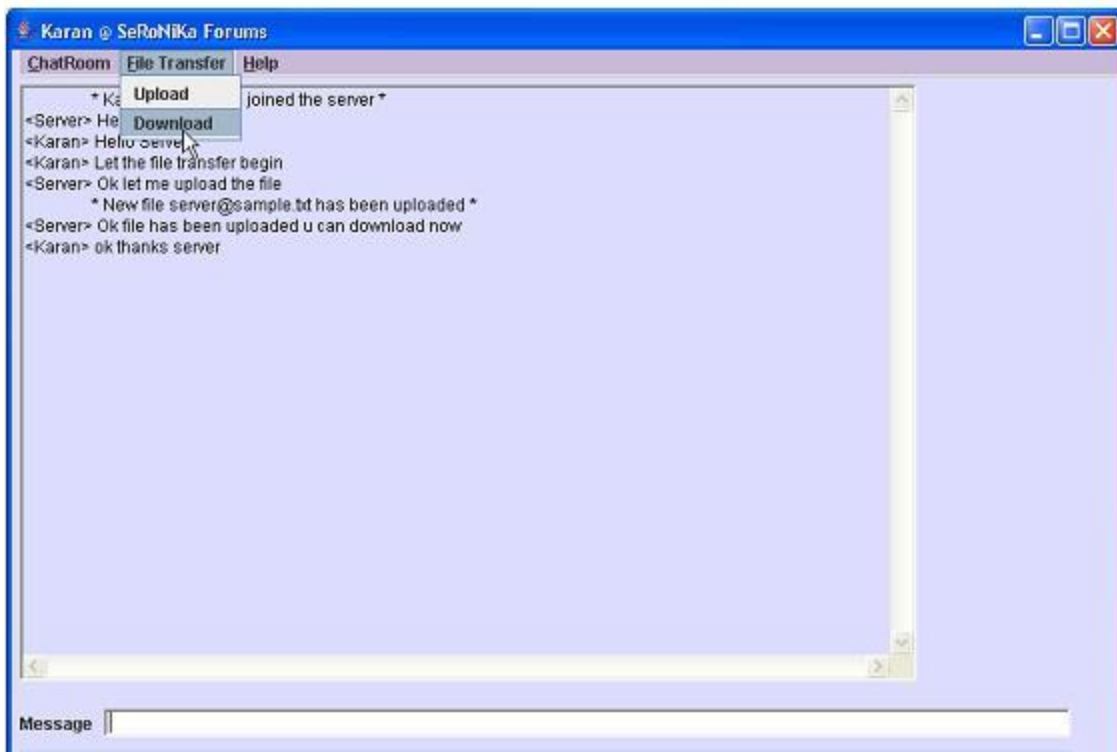
Introduction

This project aims at developing a one to many chat applet in order to build a Discussion Group online. The applet allows communication between one central server and multiple number of clients and also enables file transfer between the server and any of the clients which may supplement the discussion.

The communication will take place depending upon the IP addresses of the client. Multiple computers can login and be a part of the conference at the same time. The members will undergo an authentication process before joining the conference. For this the passwords will be stored in a separate file and then compared with the typed password for verification. If the password matches then the person is allowed to join the conference. Important files and data can be submitted or downloaded with the help of file transfer facility. Communication on the LAN is also possible.



The Server



The Client

Features

The chat applet is divided into 3 parts. The first part consists of a text box in which the conversation with the central server can be seen. Text and files can be exchanged with the help of certain tools. There is another small text box in which the text to be transferred during conversation is typed. The menubar consists of the following

1)Chat Room : There is an option **“View Users”**. By selecting this option, to the right of the chat applet a column appears which displays the members who have logged in and also the members who are offline. The members that are online are highlighted.

The option **“Clear messages”** clears the contents of the display window.

2)File: This tool is used for file transfer. There is a file upload option with the help of which we can upload files onto the server. To upload firstly the **“upload file”** option is selected. Then the required file is selected. The file transfer is indicated by the **“Sending file”** statement which appears on the status bar of the applet. After the file is transferred onto the server the statement **“File sent”** appears.

There is another option named **“Download file”** by which the users can download the files which have been uploaded onto the server. When this option is selected a separate Download manager window appears which lists the files that have been uploaded. The required file is selected and the file can be downloaded by the user.

3)Help: There is an help option as well which gives a brief description of the applet and its developers.

Design and Implementation

Client-server communication can be defined as a process which provides services to other processes. The client and server can be run on the same machine or on different machines on opposite sides of the world. A non-programming example of a client-server situation is the telephone system. You are the client (or customer) and the central office is the server. By having a telephone connected to the system (and your bill current!) you are subscribing to the services that the central office provides. Requests are made of the server (central office) to place and receive calls. The server also does accounting on each call made or received and handles emergency (911) requests. In the program, we

- Provide a server which can accept multiple simultaneous connections.
- Provide a client that can join with server.

- The server can be either a single process concurrent or multiple process server.

Why Java?

Besides getting extra credit for doing a graphical user interface, we chose Java in order to simplify programming of the client and server. Java is used specifically for the following reasons:

- **Portability.** The project application can be run on any operating system with the same efficiency.
- **Functionality.** Not just for web page animations, Java is a very useful programming language. Java is object-oriented and very similar to C and C++. A simple user interface is relatively easy to write.
- **Threads.** Java allows multiple threads (like a background process) of execution. A thread can be launched that will listen for incoming messages. When a message comes in, it is automatically sent to output.

Why Not C?

Using C for the client and server would require more programming to accomplish the same results. First, some sort of GUI builder like Motif or X-Forms would need to be used. C does not have threads, so all I/O would have to be polled. User input as well as incoming messages would have to be polled and processed accordingly. Without a GUI, some type of command codes would have to be developed for the user to control the client and server. This would probably be very cryptic and difficult to use--not to mention difficult to implement.

A Few Words on Sockets

Sockets work almost the same in Java as their counterparts in C. Since Java is object-oriented, you must create an instance of the socket object. This is done by a simple line of code:

```
public Socket connection;  
connection = new Socket (hostAddress,portNumber);
```

where *connection* is the instance of type *Socket* and *hostAddress,portNumber* are the name of the host and port to connect to. But a socket by itself is not very useful without a data input and data output stream. The code segment below sets up a data input and output stream to talk to the socket:

```
public ObjectInputStream in;  
public ObjectOutputStream out;  
out = new ObjectOutputStream(connection.getOutputStream());  
out.flush();  
in = new ObjectInputStream(connection.getInputStream());
```

The output stream is created as a buffered output stream. Data will not be written across the socket unless either Java feels that there is enough data to write, or we force to write--using a flush by using something like: *out.flush()*; this calls the flush method on the data output stream. On the reading side of the socket, we can simply go into an infinite loop and poll for data from the server, since the listener is running as a separate thread.

The Server

The server is a simple single-process concurrent server. Simply, the server polls each connection, and processes requests in order. An alternative would be to fork a new process for each incoming connection. In this situation the single process server is far simpler (and, after all, the computer can only really do one thing at a time). The basic order of things is:

1. Create the master socket on the well-known port.
2. Bind the socket so that incoming requests are directed to the proper place.
3. Listen for connections.
4. Accept incoming connections.

The "well-known port" is a port which is known to all clients. All clients can't connect to the same port, so the server "hands off" connection requests to a different port. This is done automatically by the TCP/IP layer.

To support multiple clients, the concept of multithreading has been used. The server creates a `ServerSocket` object and opens a port for the maximum number of clients which is 10 in this application. Every time a new client logs in a separate socket is created for him and bound to the `ServerSocket` object. A new thread is also created for each user to allow multiple users working simultaneously.

Server program is also capable of receiving the files from the client.

public void receiveFile(int port)() is responsible for accepting of file.

The data is read by the socket object using the *read()* method of the *ObjectInputStream* object. The transfer of data between the client program and server program takes place character-by-character and the file gets stored in folder named *SeRoNiKa* on the server with the name of the form *<nick>@<filename>*.

The server can also upload files from its system to allow the clients to download it.

The Client

The user can view users who are on the current channel. The middle of the window contains the message area. Here all messages from other users and the server are listed. The bottom field is for entering the message.

For client side, the application has a provision for uploading the file which is explained as follows:

public static boolean send(String host,int port,String nick,String filename,String fn) () is responsible for sending the file to server .

Socket skt = new Socket(host,port); creates a socket which is explained earlier.

FileInputStream fis = new FileInputStream(filename); creates an object of type class *FileInputStream* which is used for performing input operations.

“ *BufferedInputStream in* ” is used to read the file.

“ *BufferedOutputStream out* ” is used to output the file to socket.

Any client can download the files uploaded on server by all other clients by using download option.

The client sends the request for the file by selecting the filename from the Download Manager. The server searches for the file in its *SeRoNiKa* directory and sends it to the client's *SeRoNiKa1* directory.

Application of the Project

This project is designed to support online conferencing. It is a means for technical discussion forum where the Technical Expert is on the server side while he answers queries from others at the client side.

The code can be made available to the expert for inspection by uploading it on the server while any other tutorial or material can be made available to the clients for download.

Suggestions

The following features can be added to the application in the future:

- Provision can be made for new users to register at the server so that more people can be a part of the conference.
- Provision for audio and video conferencing can be made.
- Provision for creating a log of the discussion.