CS349/449 Fall 2004
Project 1
Instant Messaging Client and Server

Sam Small Andreas Terzis

September 25, 2004

Document History

- **Version: 0.9.1 9/25/04:** Included description of OK and ERROR commands
- **Version: 0.9 9/24/04:** Initial version

1 Introduction

For your first project you will have to write a client and a server for an application that resembles IRC (Internet Relay Chat) and IM (Instant Messaging).

Clients connect to the server and supply a username and password. The server verifies that the username exists and the password matches the password stored in a password file read by the server during the initialization phase. If the username and password matches then the server allows the client to connect. Once this is done, the client can send messages that are delivered by the server to all other people currently logged in the chat room. Finally the client can send a bye message to disconnect from the chat room.

Students taking CS449 and 349 students that want to take 10% credit will also have to implement a direct talk connection between two clients of the system.

2 Protocol Description

The protocol that you are going to implement works on top of TCP. It is an ASCII based protocol (case sensitive) and the general format of the commands sent is as follows:

```
COMMAND\n
TagName:  Value\n
TagName:  Value\n
\n
```

Each line is terminated by the newline character (ASCII character 13). The first line contains the command line. This line contains one of the command strings shown in section 2.1. The remaining lines contain "tags" that provide additional information. The format of a tag is a string identifying the tag name followed by a colon ":", one or more spaces then the tag value. The list of tags is described in section 2.1.1. The end of the command is specified by two consecutive newline characters.

2.1 Protocol Commands

The following commands are part of the IM protocol.

1. `CONNECT`. This command is sent by the client to the server and supplies the user's username and password. This command MUST have the following three tags (in order):

- `UID: string\n`
- `PASSWD: string\n`
- `SEQ: number\n`

The string in the `UID` tag is the user's username and the password is the user's password. The server will compare the user's username and password to the locally stored username and password. If they agree the server replies with an `OK` otherwise it replies with an `ERROR` command MUST contain a `SEQ` tag and MAY have a `ERRMSG` tag that contains a string containing an error message. Errors generated in response to malformed commands should have a `SEQ` value of -1. If a received `ERROR` command is malformed, silently discard it.

2. `OK` The `OK` command is interpreted as positive acknowledgement of a previously sent valid message. It MUST always be accompanied by a valid `SEQ` tag.

3. `ERROR` The `ERROR` command is a positive acknowledgement of a previously sent invalid message. The `ERROR` command MUST always have a `SEQ`. If the `ERROR` is being sent in response to a malformed command (in terms of syntax, not content), then the sequence number MUST be -1. `ERROR` messages should not be generated in response to other `ERROR` messages.

4. `SENDALL`. This command is sent by the client when it wants to send a message to all the users that are currently logged in the chat room. The `SENDALL` command MUST contain a `MSG` tag containing a string with the message that the client wants to send to the chat room.

5. `PUBLISH`. When a client sends a `SENDALL` message to the server, the server publishes the message to the other members of the chat room by sending a `PUBLISH` message to each connected client (other than the client that sent the original `SENDALL` message). The `PUBLISH` message MUST contain a `MSG` tag with the same string as in the corresponding `SENDALL` message prepended by the username of its originator (e.g., `<Sam> is anyone there?`).

6. `WHO`. When a client that wants to find who the other users connected in the chat room are, a `WHO` command is sent to the server.

7. `ULIST`. When the server receives a `WHO` request from a client, it replies with a `ULIST` command. The `ULIST` command MUST contain a list of `UID` tags, one for each user that is currently logged in the chatroom (including the user that requested the list).

8. `BYE`. When a client wants to exit the chatroom it sends a `BYE` command to the server. The client MUST wait for the `OK` response from the server before closing the TCP connection to the server.

   Note: The following commands must be implemented only by students taking CS449 and 349 students for additional credit.

9. `TALKTO`. When a client wants to create a direct connection to another client in the system it sends a `TALKTO` command to the server. The `TALKTO` command MUST contain a `UID` tag with the username of the peer the the user wants to connect to. For this assignment, a client may only have one direct connection at a time.

10. `TALKREQ`. When the server receives a `TALKTO` request from "usera" to "userb", the server must forward the request to "userb" in a `TALKREQ` message. The `TALKREQ` message must contain a `UID` tag with the username of the user that initiated the request ("usera" in this example). The client that receives the `TALKREQ` message MUST reply either with an `OK` message containing a `PORT` tag or a MSG tag explaining why the invitation is being denied, or an `ERROR` message if the `TALKREQ` message is malformed. If "userb" has accepted, she should should start listening on her indicated port for incoming connections. The server then sends a `TALKRESP` message to the originator of the `TALKTO` message along with the `PORT` tag and an `IP` tag with the IP address of the responder (i.e., "userb"), which MUST immediately follow the `PORT` tag. The originator then should try to establish a TCP connection to the peer using the supplied port. After the connection is successfully established the client that requested the connection MUST send a `CONNECT` command supplying its username. If the receiver of the `TALKREQ` message did not

accept the invitation, the server MUST send a TALKRESP to "usera" along with the message supplied by "userb". For this assignment, assume a client will only have one pending TALKREQ at a time.

11. TALKRESP. A TALKRESP message is sent from the server back to the originator of a TALKTO message. TALKRESP must contain a PORT tag followed by an IP tag if the other principle accepted, or a MSG tag if the client denied the request.

12. SEND. Once the connection between the two peers has been successfully established (that is after the first CONNECT and OK message exchange) each peer can send SEND commands to each other. The SEND command MUST contain a UID tag and a MSG tag. SEND commands MUST be acknowledged by an OK command. Each client can terminate the connection by sending a BYE command.

2.1.1 Tags

The following tags are defined as part of the IM protocol

1. UID. This tag provides a user's username. The TagValue should be a string equal to a user's username.

2. PASSWD. This tag provides the user's password. The TagValue should be a string with the user's password.

3. SEQ. Used to correlate replies to requests. The TagValue should be an integer.

4. ERRMSG. This tag is used to give more information regarding the reason of failure. The TagValue is a free form string (i.e. spaces are allowed) with diagnostic information.

5. MSG. This tag contains the message that the client wants to forward to the chat room. The TagValue is a string. The maximum length of the message string is 64000 characters. This tag may additionally be used in the OK message resulting from a TALKREQ. See the description of the TALKREQ command for more details.

6. PORT. This tag identifies the port that a client is listening to for direct connections. The TagValue is a integer in the range 1025-65535.

7. IP. This tag identifies the IP address that a client is listening for direct connections. The TagName is an IP address in dotted decimal format (e.g. 128.220.220.1).

All commands MUST contain the SEQ tag. This tag is used by the server to keep track of the messages sent by each client. Each client should use a unique sequence number for all requests it sends. OK and ERROR replies should use the sequence number of the command they are acknowledging. The SEQ tag should always be the last tag for any of the protocol commands.

2.2 Client Commands

The following commands are used in the client application by the user to perform a number of tasks:

1. /login <username> <password> This should be the first command issued by a user when using the client and trigger a protocol CONNECT command.

2. /who This command allows the user to discover the names of all users currently on the system and triggers a protocol WHO command.

3. /logout This command should trigger a protocol BYE command and exit the chat client (for 349 extra credit and 449 it should also generate a BYE message to be sent to the other principle if a private connection has been established).

Additionally, 349 students doing the extra credit and all 449 students must support the following client commands:

1. `/knock <username>` This command should tigger a protocol `TALKTO` command to be sent to the server with `<username>` as the `UID`.

2. `/accept` This command is used to accept a `TALKREQ` message.

3. `/deny` This command is used to deny a `TALKREQ` message.

4. `/msg <message>` This command is used after a direct connection has been established to another user via `/knock` and allows the user to send the private message `<message>` to the other principle. This command should trigger a protocol `SENDTO` command to be sent directly to the principle.

5. `/endprivate` This command is used to end a private conversation and will trigger a `BYE` command to be sent directly to the other principle.

3 Transaction Examples

3.1 Client Server Message Exchanges

Figure 1 shows an example of the communication between two clients and the chatroom server. First the clients CONNECT and after the server validates the username and password supplied by the clients, it replies with an `OK` message. **NOTE: In this example the username and password are sent "in the clear" something that should not be done if you are building a real application**.

At this point each client can send a `SENDALL` command with a message that it wants to be delivered to all the chatroom participants. Once the server receives a `SENDALL` message it replies to the sender with an `OK` message and sends a `PUBLISH` message to all the other clients connected to the chatroom. The server is not required to send a `PUBLISH` message back to the sender of the original message. Also the server can send the `OK` message back to the sender of the `SENDALL` command before it delivers the `PUBLISH` messages to all the other clients.

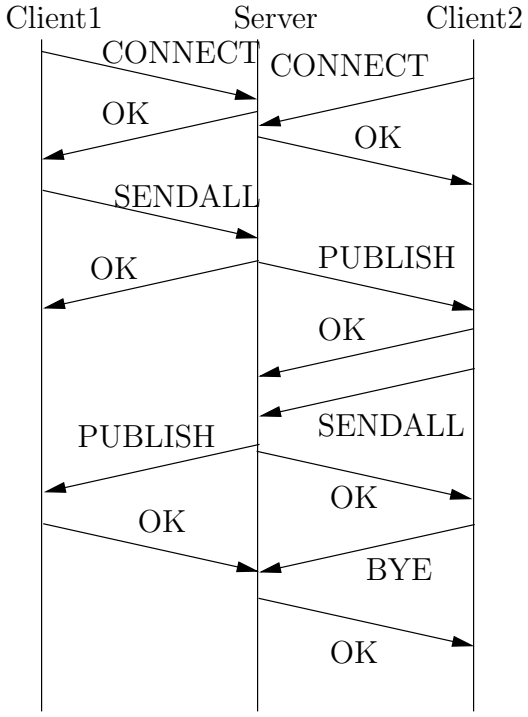

Figure 1: Example of Client Server communications

At some point the client might want to disconnect from the chatroom. The client then sends a `BYE` command to the server, waits for the `OK` reply from the server and then it closes the TCP connection to the server.

3.2 Client to Client Message Exchanges

Figure 2 shows an example of how two clients can start talking directly to each other. In this figure, Client1 initiates the sequence by sending a `TALKTO` command to the server supplying the username of the person it wants to connect to in a `UID` tag. The server will then forward the request to the appropriate client in the form of a `TALKREQ` command and send an `OK` command back to the initiator. The `TALKREQ` command should contain the username of the requester supplied in a `UID` tag. If the user the requester is trying to connect it is not currently logged in the server MUST respond with an `ERROR` message.

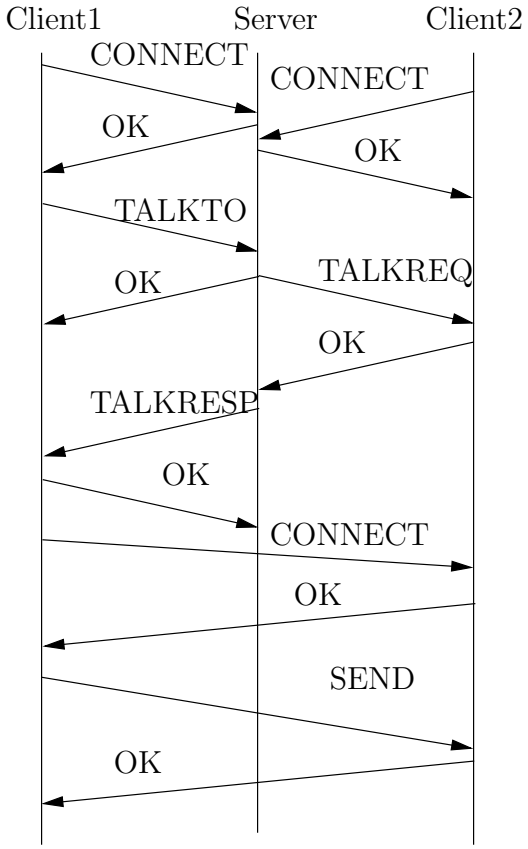

Figure 2: Example of Peer communications

On the other hand, if the client is currently logged in and accepts the connection it MUST reply with an `OK` message containing the port the client will start listening to. This port is supplied in a `PORT` tag. The server then sends a `TALKRESP` command to the original requester using the specified `PORT` and inserts a `IP` tag with the IP address of the second client. If the receiving client doesn't want to establish a connection the server MUST reply with an `MSG` tag in the `TALKRESP` command. For now, your client implementation must support only one direct connection so if the receiving client is already directly connected to another client, any subsequent `TALKREQ` request will fail.

4 Deliverables

You will need to implement both the client and the server implementing the IM protocol described above. The server should be capable of supporting multiple concurrent clients. The client should be capable of supporting one direct connection with another client. You can implement the client and server either in Java or C/C++.

You will need to submit all the source code that you wrote along with appropriate Makefiles. Your code should be well documented. In addition to the source code you must return a short writeup (1-2 pages with an explanation of how your code works). In the writeup you should explain what was the responsibility of each member of the team. The writeup should be in plain text format (i.e., no MSWord or PDF files). One writeup per team is enough.

Tar and gzip all of your materials into one file named **LastnameFirstname.tgz**. Make sure that decompressing your tarball will place all of your materials in the current working directory (i.e., no subdirectories). Send this file to cs349hw@magneto.cs.jhu.edu with the subject "PROJECT1" (without quotes).

4.1 Deadline

The project is due on **Friday 10/15 at 5pm.**

4.2 Server

The server should take two command line arguments. The first one is the port that it should be listening to for client requests. The second command line argument is the name of a file containing the password file. The server uses this password file to verify that the password supplied in CONNECT requests is a valid password. The password file contains a username, password pair per line. A sample password file is shown below:

```
andreas:andreas
sam:sam
```

NOTE: It's not a good practice to store unencrypted passwords on the disk. So again, a "real" application must actually store the passwords in encrypted form.

Your server executable must be named HiNRCServer if done in C++ (created by Makefile) or originate from HiNRCServer.java if done in Java.

4.3 Client

The client should take two command line arguments. The first one is the DNS name or the IP address of the server that the client should connect to. The second argument is the port that the server is listening to.

Your client executable must be named HiNRCClient if done in C++ (created by Makefile) or originate from HiNRCClient.java if done in Java.

5 Important Notes

We are going to provide a demo server that you can use to test your clients. The location of the server is TBA.

For students doing the assignment in Java, we will provide a class file that implements a GUI for your chat client.

6 Policy for Late Submissions

No late submissions accepted.