CSE 489/589

Programming Assignment 1 Report

Text Chat Application

1 - Group and Contributions

- Name of member 1: Saurabh Tambolkar
  - UBITName: 50412968
  - Contributions – Wrote the files assignment1.c & global.h. Prepared the server side of the report.
- Name of member 2: Shantanu Kumar
  - UBITName: 50418500
  - Contributions – Wrote the file MainProc.c. Prepared the client side of the report.

2 - SHELL Functionality

[5.0] Application Startup

The application is started by executing the command given below –
/local/Fall_2021/<Insert-UB-IT-Name>/cse489589_assignment1/<Insert-UB-IT-Name>/assignment1 <'c'-or-'s'> <Port-Number>
The character 'c' signifies that the application has been started as a client whereas the character 's' signifies that the application has been started as a server. This is followed by the port number on which the application is running.
Once the application has been started, it is constantly waiting for input from the user, which it reads from the standard input and processes sequentially.

3 - Command for Server and Client

[0.0] AUTHOR

```
[underground {/local/Fall_2021/skumar39/cse489589_assignment1/skumar39} > ./assignment1 s 5001
[AUTHOR
[AUTHOR:SUCCESS]
I, skumar39, have read and understood the course academic integrity policy.
[AUTHOR:END]
```

The command AUTHOR states that we have understood and abide by the course academic integrity policy.

[5.0] IP

```
[highgate {/local/Fall_2021/skumar39/cse489589_assignment1/skumar39} > ./assign]
 ment1 s 5001
[IP                                                                          ]
 [IP:SUCCESS]
 IP:128.205.36.33
 [IP:END]
```

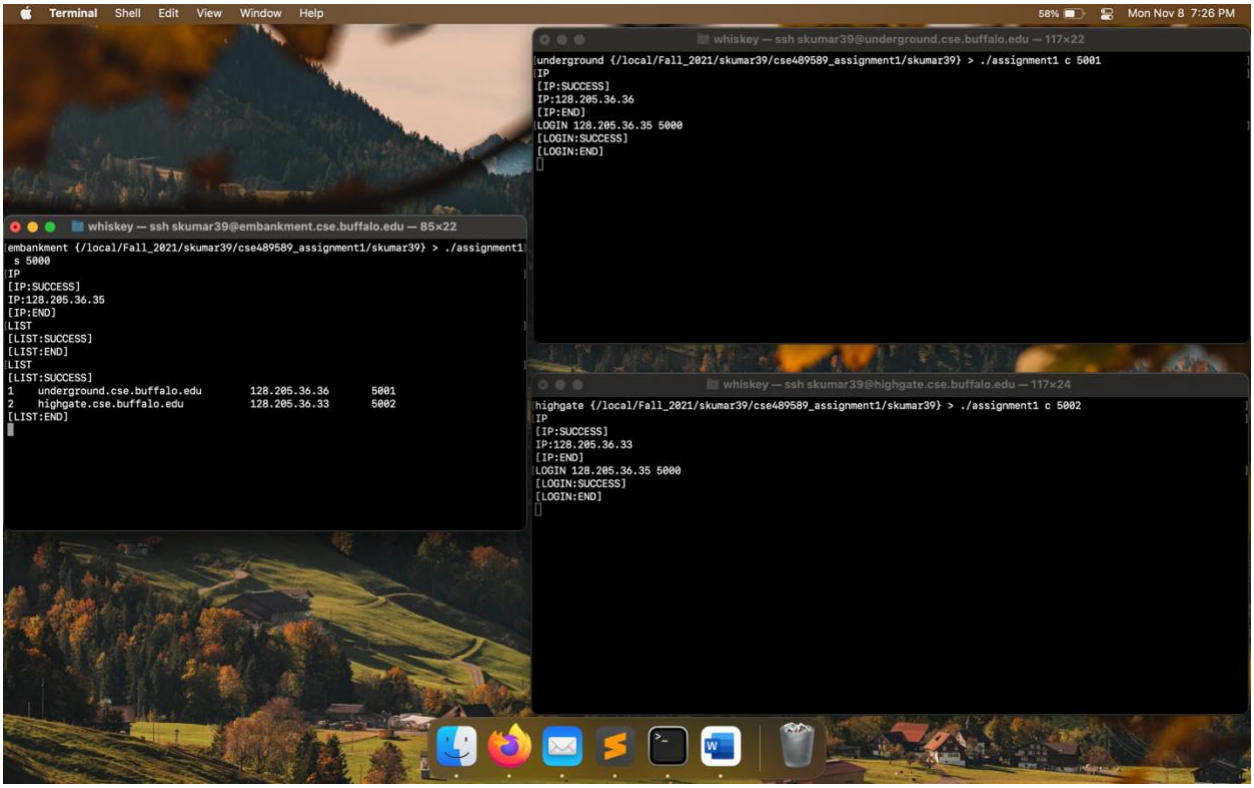
This command prints the IP address of the current process and is handled by the function get_localIP().

[2.5] PORT

```
highgate {/local/Fall_2021/skumar39/cse489589_assignment1/skumar39} > ./assign
ment1 s 5001
IP
[IP:SUCCESS]
IP:128.205.36.33
[IP:END]
PORT
[PORT:SUCCESS]
PORT:5001
[PORT:END]
```

Fetches the port number on which the application is running.
This is handled by capturing the port number the user mentions when starting the application and then outputting it when this command is executed.

[10.0] LIST

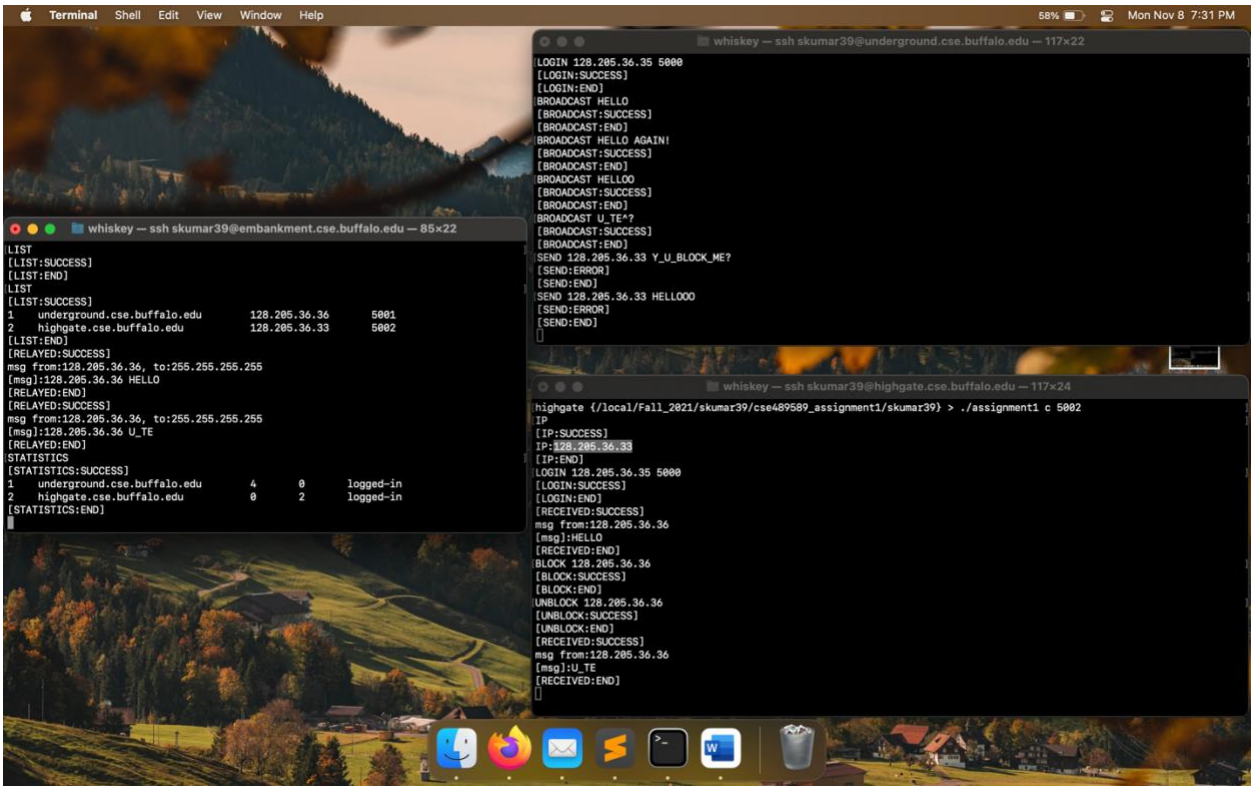


Displays the list of all the currently logged in clients along with their hostname, IP address and listening port number. The struct connection that is defined as a global structure is used to keep tab of all the current connections. When the command ‘LIST’ is executed, we iterate through all the elements of the structure array, find the ones whose current status is shown as logged in and print them along with their IP address and port number.

4 - Command/Event for Server

[5.0] STATISTICS

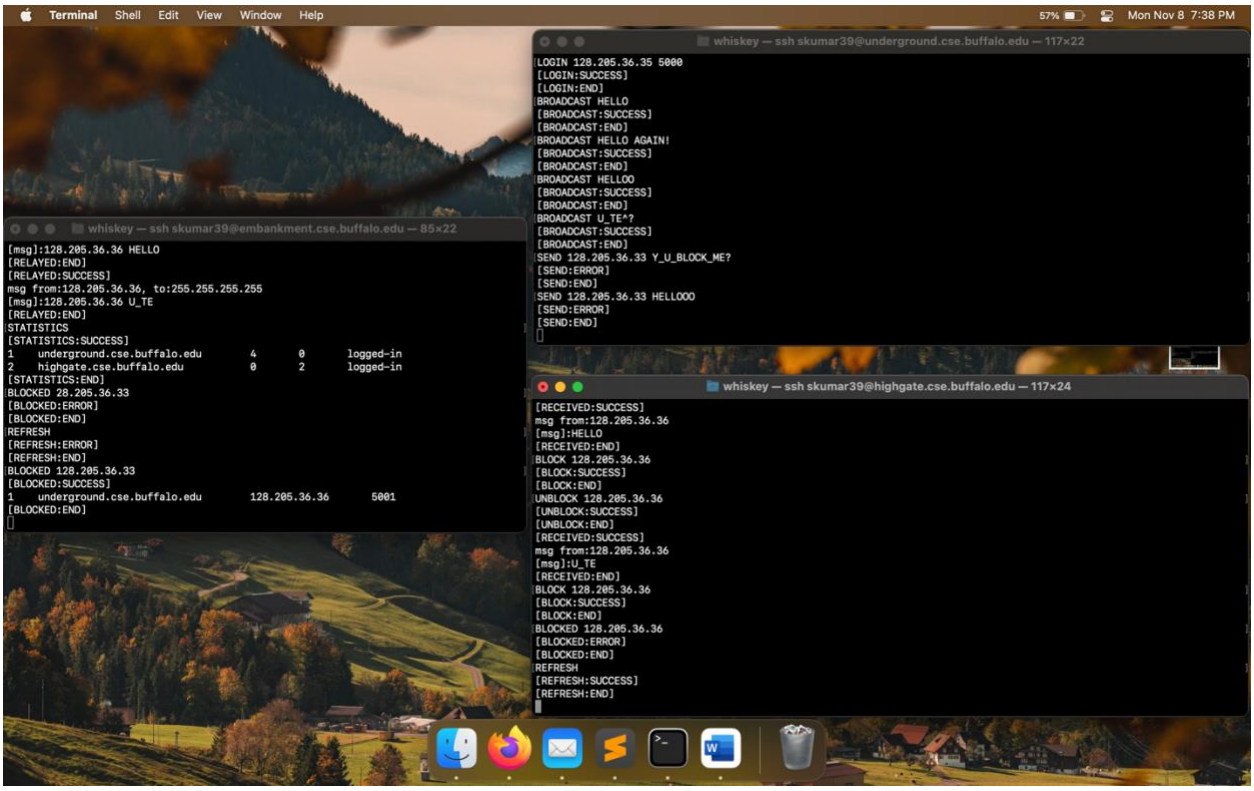
Displays a list of all the clients that have logged in at the server at any point of time along with the number of messages sent by them, the number of messages received them and their current status. This is again handled by the struct connections which maintains a list of all the connections that have been established at any point of time. The variable connections[i].msg_sent & connections[i].msg_received keep a tab of the number of messages sent and received by a hostname respectively. If connections[i].status is equal to the value stored in loged_in, that particular host is logged in, otherwise not.



Here in the terminal on the left we have the server whereas the two terminals on the right represent two clients. Both the clients then connect to the server and exchange messages as shown above. We then execute the command STATISTICS on the server which then displays the clients and the number of messages sent and received by each.

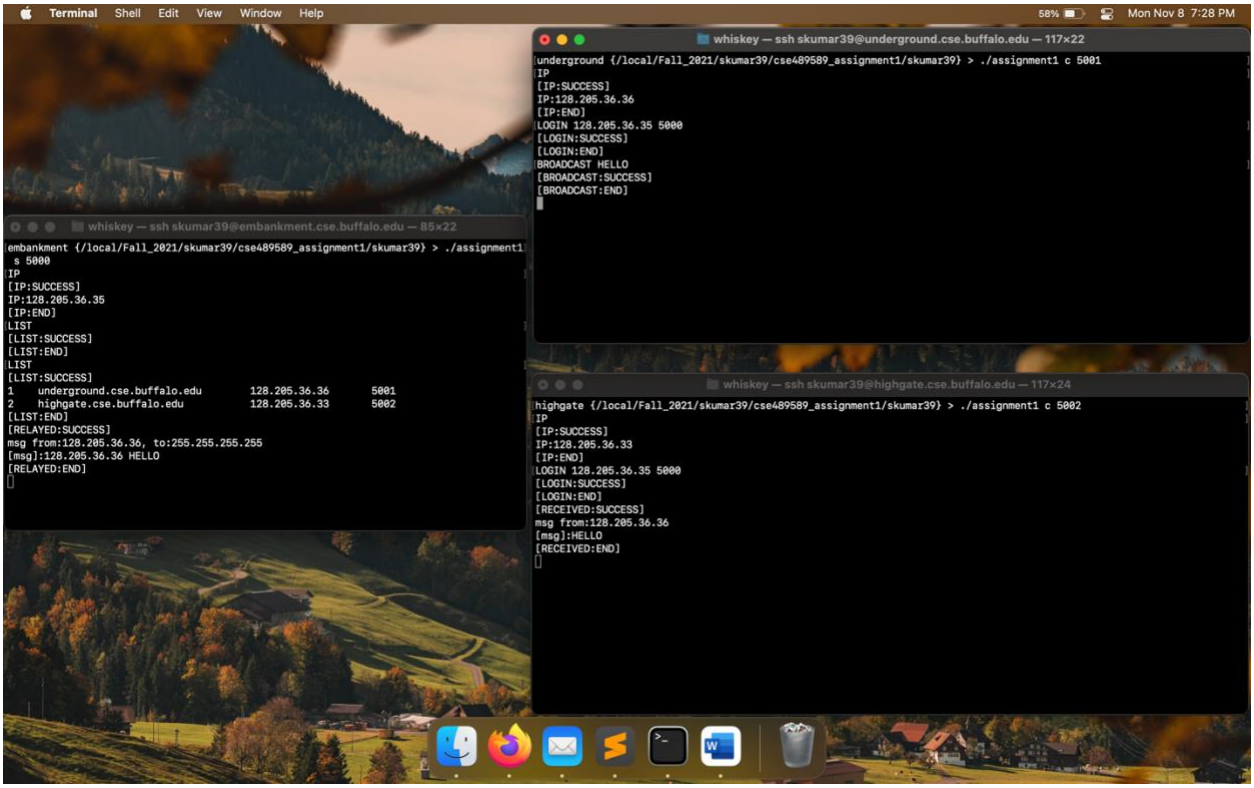
[7.0] BLOCKED <client-ip> + Exception Handling

Displays a list of all the clients blocked by the client with a particular IP address. The struct connections contains an array blockedIPs that maintains a list of all the IP address that have been blocked by that client. We simply iterate through this list and print all the elements in it.



Again, the server is on the left and the clients on the right. The client on the bottom right blocks the client on the top left which is then evident when the server on the left runs the BLOCKED command.

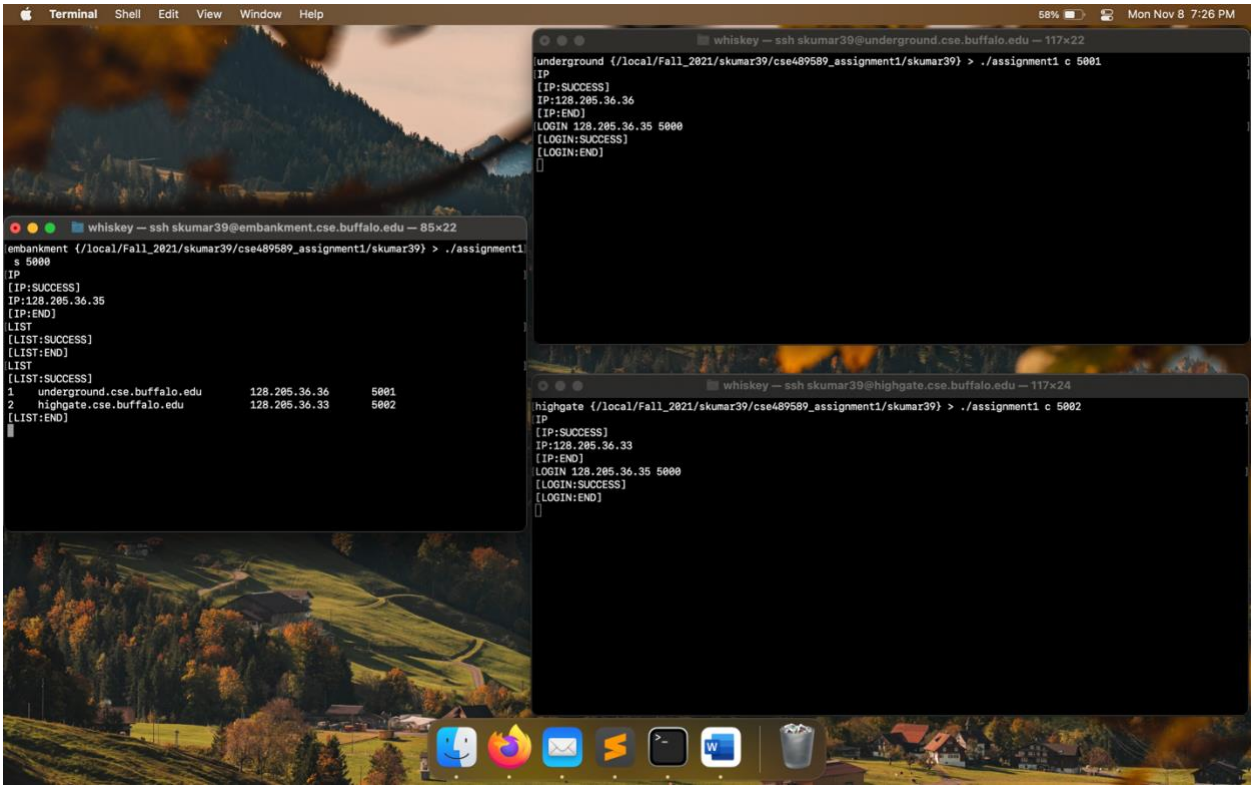
[EVENT]: Message Relayed



Message relayed.

5 - Command/Event for Client

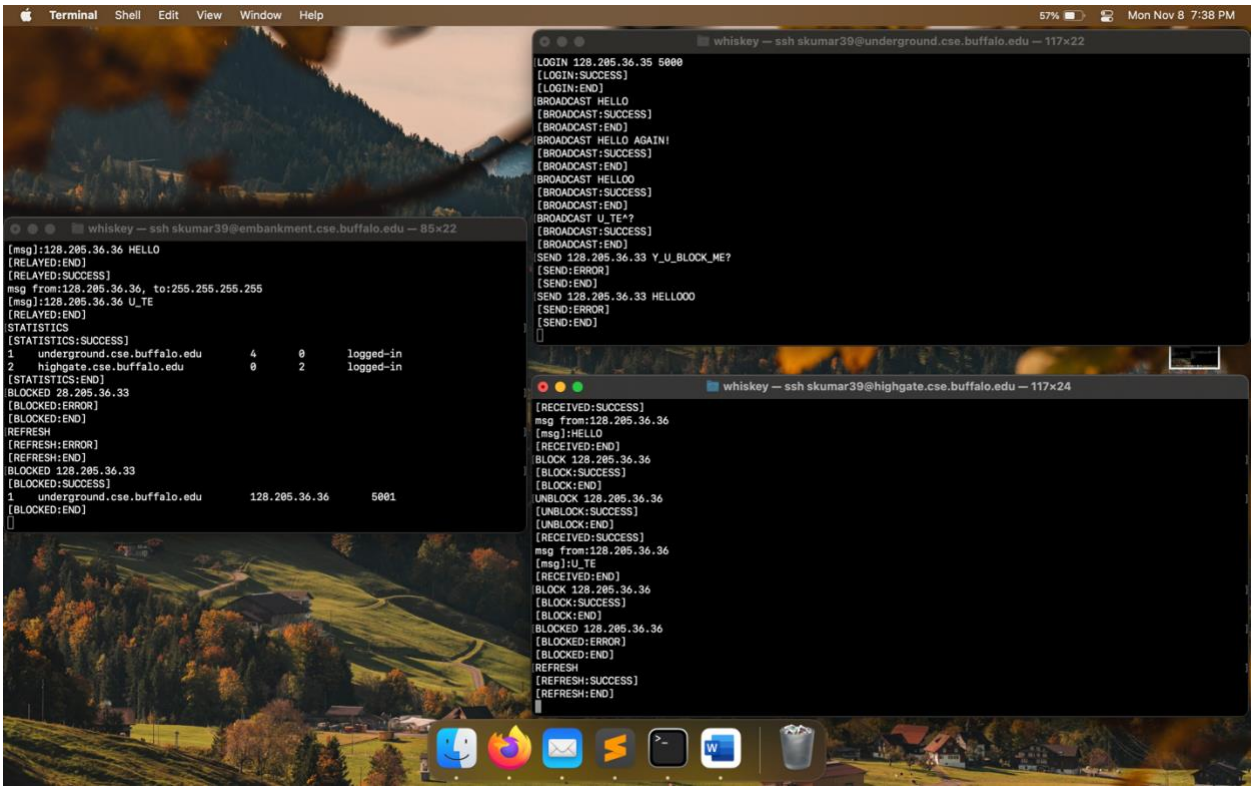
[17.0] LOGIN <server-ip> <server-port> + Exception Handling



This command is used by a client to login to the server located a particular IP address.

[5.0] REFRESH

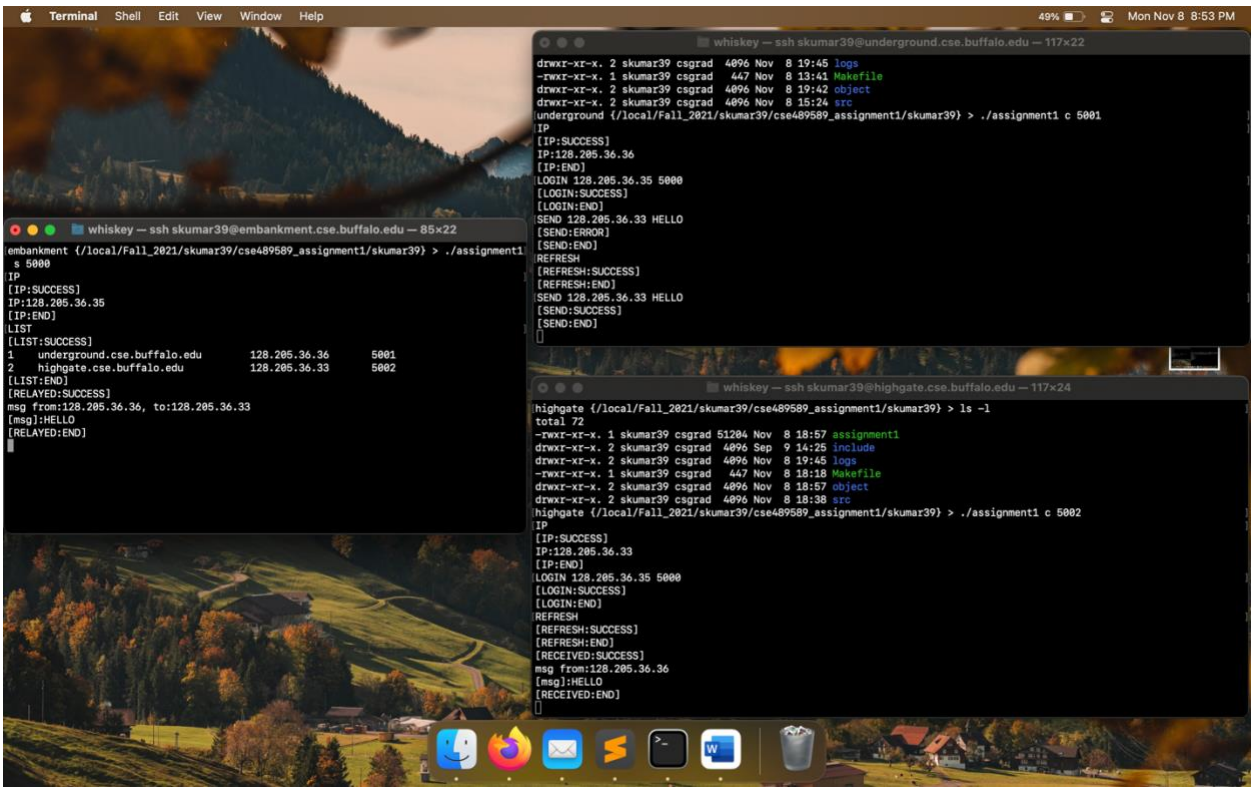
Fetches the list of all the currently logged in clients from the server. We iterate through the struct connections, check the status of all the clients, those whose status is currently equal to loged_in are printed.



The server displays the correct output for the command BLOCKED after a REFRESH is executed at one of the clients post blocking the other client.

[17.0] SEND <client-ip> <msg> + Exception Handling

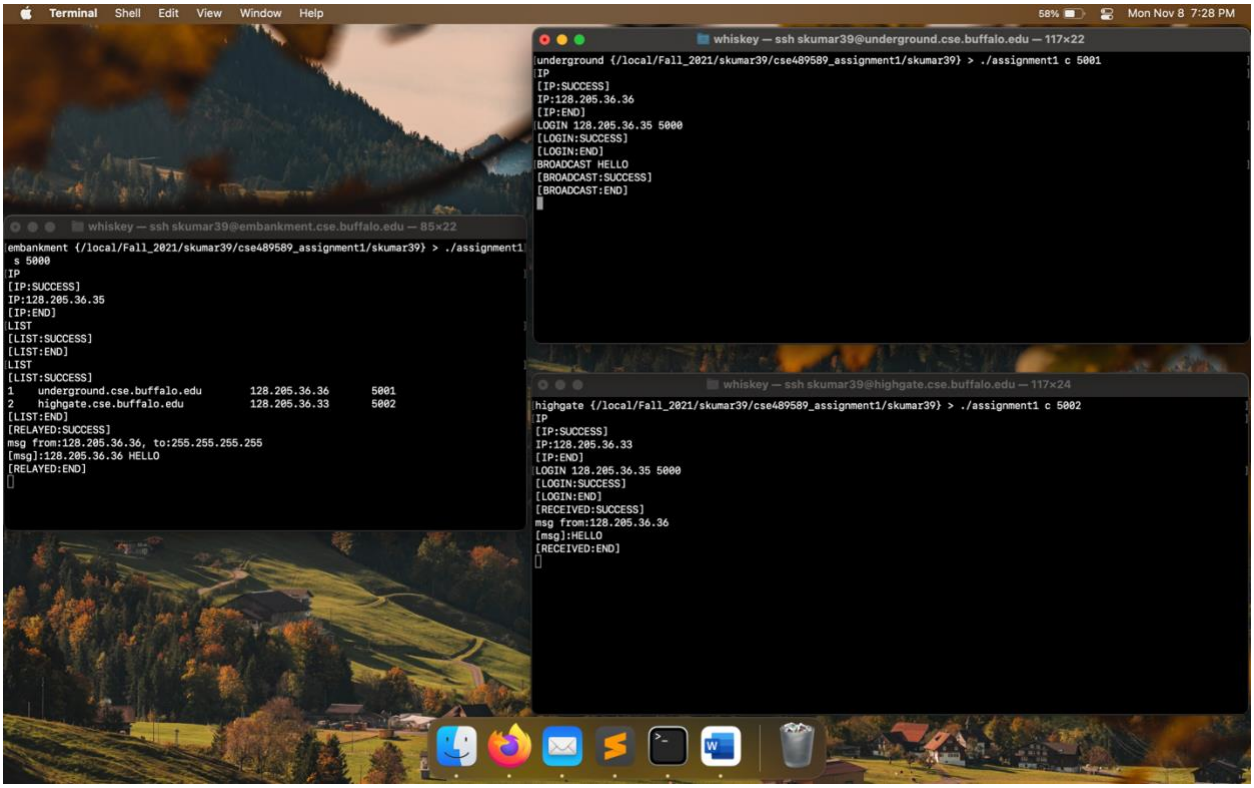
Sends a message to the provided IP address. For the message to be successfully transmitted, the sender should not be in the list of IPs blocked by the receiver and the receiver should not be in the blocked list of the sender. Also, the user to which the message is being sent should be valid and currently logged in, otherwise the message will not be sent, and an error will be generated. This command also increments the count of messages sent by the sender the count of messages received by the receiver.



Message received by the client on the bottom right when sent by the client on the top right.

[10.0] BROADCAST <msg>

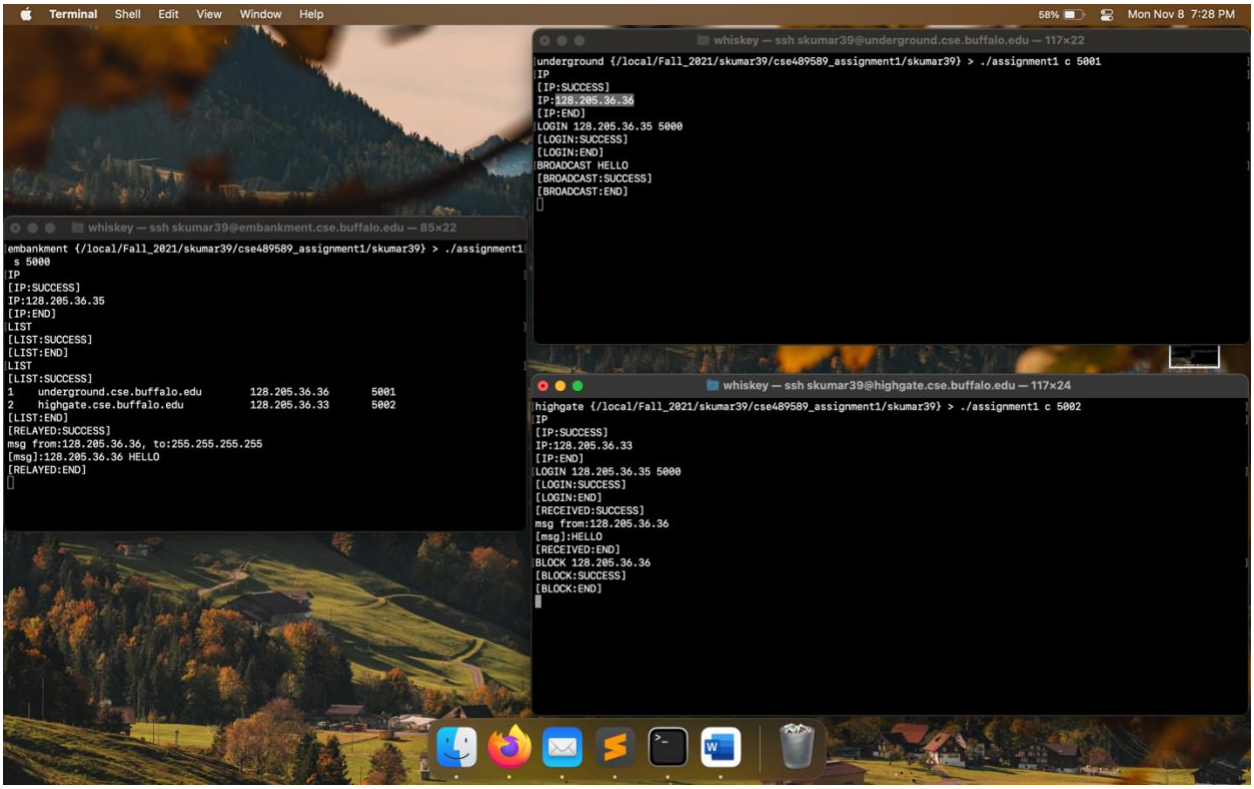
Sends a message to all the currently logged in clients that are not a part of the list of IPs blocked by the sender. A receiver that has blocked the sender's IP address will also not receive the message. Like the SEND command, this also increments the count of messages sent by the client and the count of messages received by the currently logged in recipients.



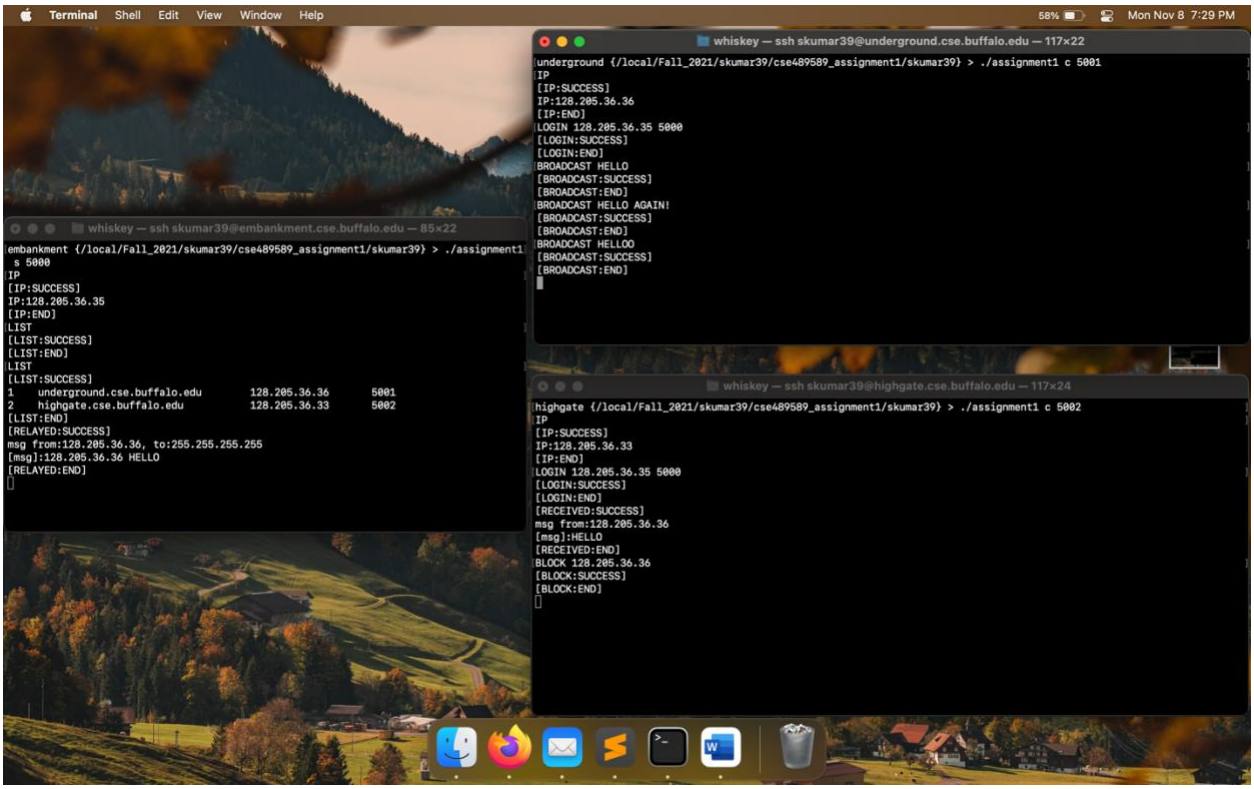
Broadcasted message by the client (top right) received by the other client (bottom right).

[7.0] BLOCK <client-ip> + Exception Handling

Blocks a given IP for a client. Struct connection has an array blockedIPs that contains the list of all the IPs blocked for that client. The BLOCK command simply places the given IP in that array. A client is prohibited from blocking itself and trying to re-block an IP that is already blocked has no effect.



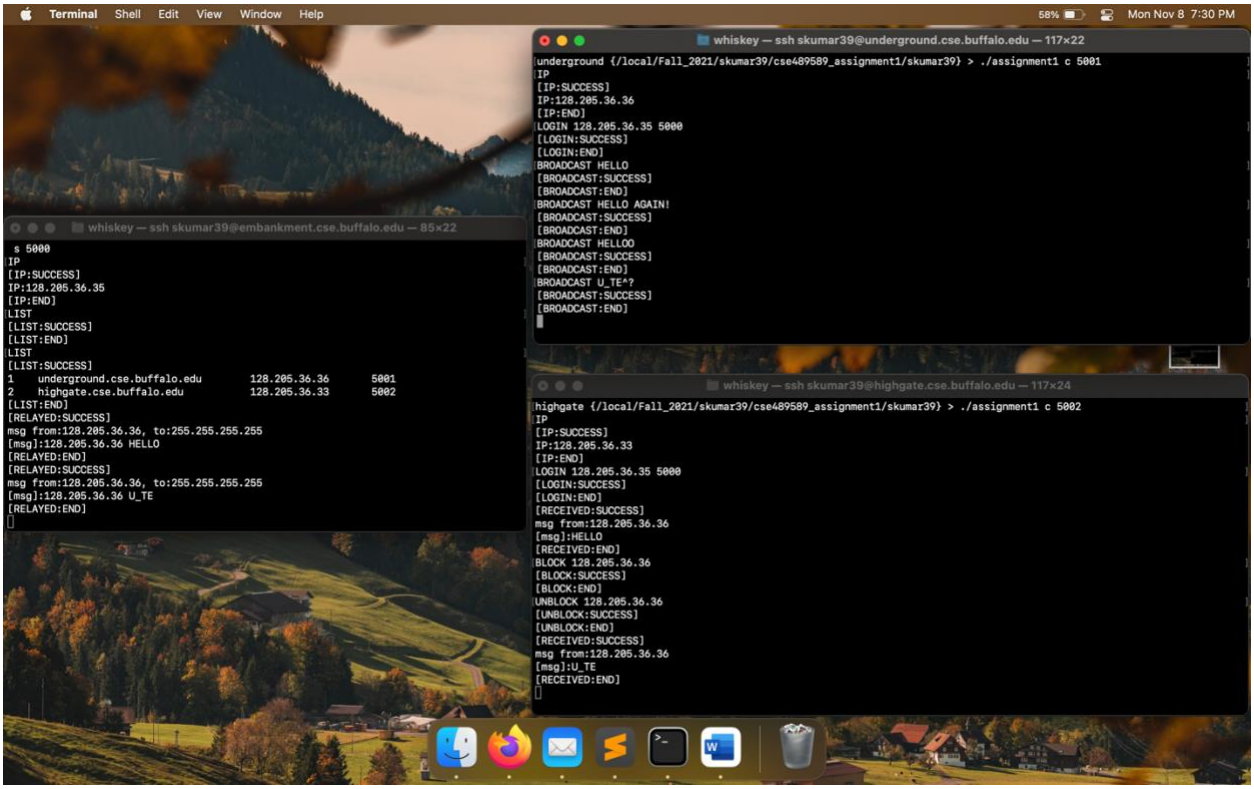
Here the client on the bottom right blocks the client on the top right.



The client on the top right then attempts to message the client on the bottom right but fails since it is currently blocked.

[4.5] UNBLOCK <client-ip> + Exception Handling

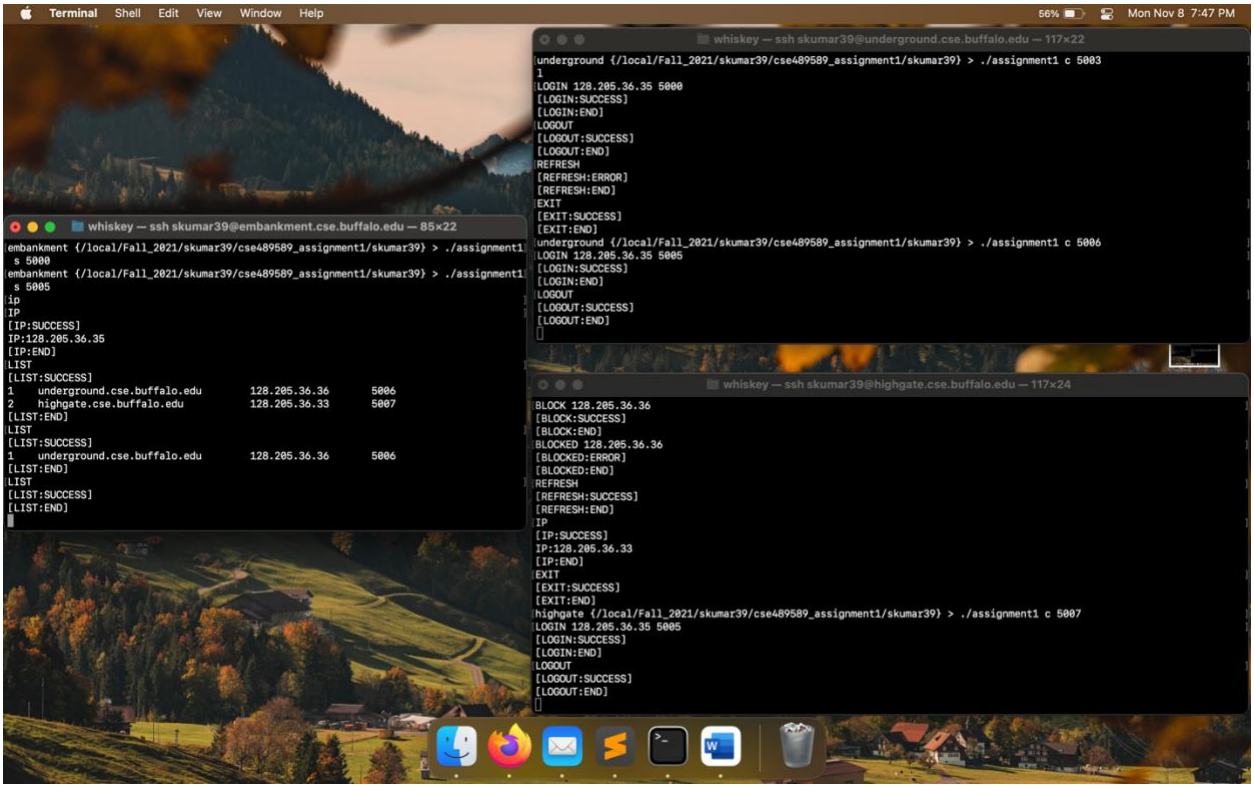
Unblocks an already blocked IP. Removes that IP from the blockedIP array by iterating through all its elements. Only an IP that has been previously blocked can be unblocked using this command. Trying to unblock an IP that has not been blocked will result in an error.



The client on the bottom right starts receiving messages from client on the top right after it gets unblocked.

[2.5] LOGOUT

This command is used to logout from a server. This command closes the socket associated with the connection and clears its file descriptor using the FD_CLR command. However, this command does not destroy any states associated with the client, those remain intact in the connection structure and can be accessed in the future.



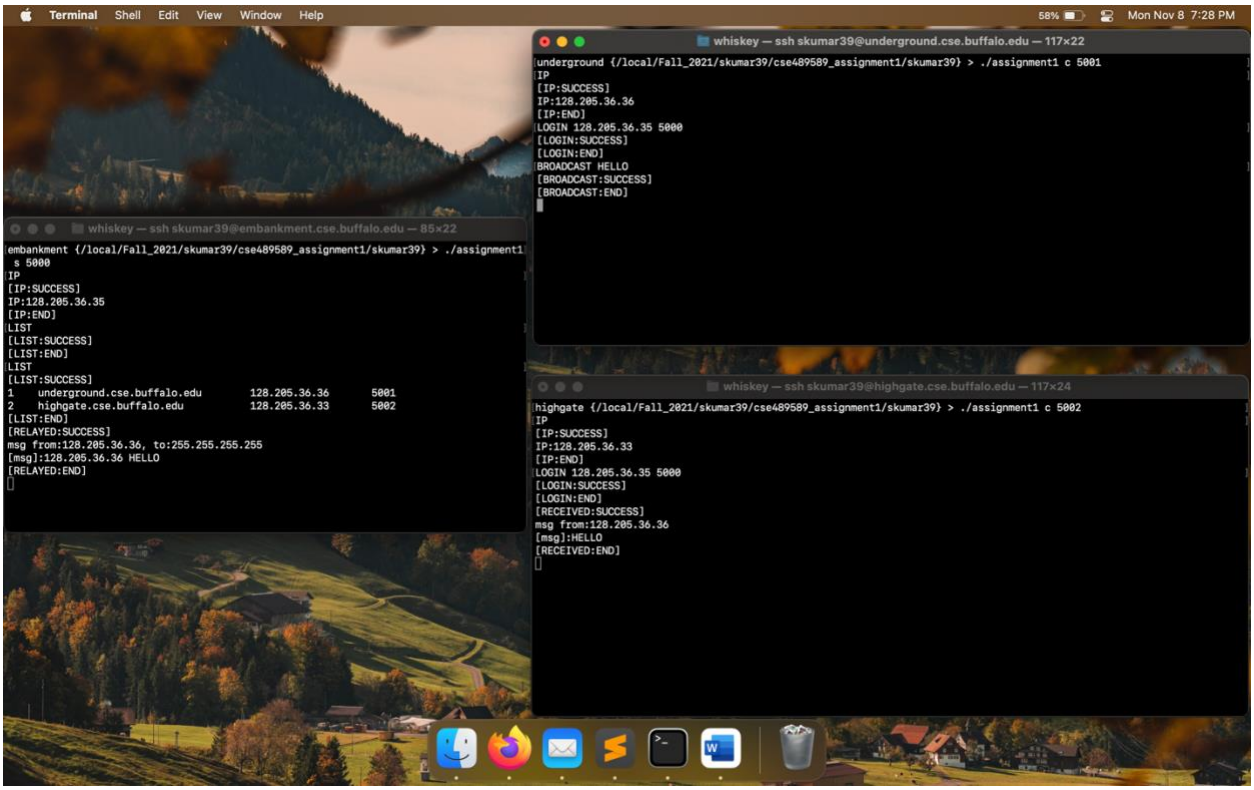
Initially, both the clients are connected to the server because of which we see two entries when the LIST command is executed. The client on the top right then logs out after which the LIST command results in one entry only. The client on the bottom right also logs out, resulting in no entries when the LIST command is executed by the server on the left.

[2.5] EXIT

```
highgate {/local/Fall_2021/skumar39/cse489589_assignment1/skumar39} > ./assign
ment1 c 5001
IP
[IP:SUCCESS]
IP:128.205.36.33
[IP:END]
EXIT
[EXIT:SUCCESS]
[EXIT:END]
highgate {/local/Fall_2021/skumar39/cse489589_assignment1/skumar39} >
```

This command closes the connection and unlike the LOGOUT command, destroys all states associated with the client. Destroying the states implies removing all data associated with that client from the connection structure.

[EVENT]: Message Received



Message relayed from one client (top right) to the other (bottom right) by the server (left).