

CSE 489/589

Programming Assignment 1 Report

Text Chat Application

Notes:

- One of your group members select <File> - <Make a copy> to make a copy of this report for your group and share that Google Doc copy with your teammates so that they can also edit it.
- Report your work in each section. Describe the method you used, the obstacles you met, how you solved them, and the results. You can take screenshots at key points. There are NO hard requirements for your description.
- For a certain command/event, if you successfully implemented it, take a screenshot of the result. You will get full points if it can pass the corresponding test case of the automated grader.
- For a certain command/event, if you tried but failed to implement it, properly describe your work. We will partially grade it based on the work you did.
- **Do NOT claim anything you didn't implement.** If you didn't try on a certain command or event, leave that section blank. We will randomly check your code, and if it does not match the work you claimed, you and your group won't get any partial grade score for this WHOLE assignment.
- There will be 10 bonus points for this report. Grading will be based on the organization, presentation, and layout of your report.
- After you finish, export this report as a PDF file and submit it on the UBLearns. For each group, only one member needs to make the submission.

1 - Group and Contributions

- Name of member 1:
 - UBITName: sanjaykr
 - Contributions – Client and Server implementation
- Name of member 2:
 - UBITName:vruppaga
 - Contributions – Client and Server implementation
- Name of member 3:
 - UBITName:oviyabaa
 - Contributions – Debugging, Error checking and documentation

2 - SHELL Functionality

[5.0] Application Startup

The application is started using two arguments namely, s, c where s denotes server and c denotes client. The second argument is the port on which the application must bind. Once the bind is successfully executed, a socket is created, and its respective file socket descriptor is stored. Using select function call, blocking status is initiated and multiple clients are connected simultaneously with the server. The application for client, user sends a shell command which is received in a buffer and processed accordingly as described in the commands for server and client. The server receives few commands directly from the user and particular number of commands from the client and processes it along with its parameters (if mentioned). The shell commands exception is also handled in both client and server side.

3 - Command for Server and Client

[0.0] AUTHOR

The give format is executed along with the mentioned set of statements in both client and server side. The author command is tested using the autograder and it returns TRUE.

```
AUTHOR
[AUTHOR:SUCCESS]
I, Oviyaa Balanurugan, have read and understood the course academic integrity policy.
[AUTHOR:END]
```

Fig 1. Author

[5.0] IP

A socket is created and connected with the Google's public DNS server and its respective port. This is used to perform a reverse DNS lookup which returns a consistent name/IP pair. It is done so by creating a UDP socket and calling the function getsockname(). Once the return value is received, its exceptions are handled and the IP which is present in the buffer is converted into network address format using inet_ntop and is printed.

The ip command is tested using the autograder and it returns 5.0.

```
IP
[IP:SUCCESS]
IP:128.205.36.46
[IP:END]
```

Fig 2.1 Server IP

```
IP
[IP:SUCCESS]
IP:128.205.36.34
[IP:END]
```

Fig 2.2 Client 1 IP

```
[IP:SUCCESS]
IP:128.205.36.35
[IP:END]
```

Fig 2.3 Client 2 IP

[2.5] PORT

The port value is extracted from the current file descriptor socket which is calling the “printPort()”, which is present in the sin_port and is converted to network to host short format (ntos) to be printed.

The port command is tested using the autograder and it returns 2.5.

```
PORT
[PORT:SUCCESS]
PORT:6000
[PORT:END]
```

Fig 3.1 Port of Server

```
PORT
[PORT:SUCCESS]
PORT:7000
[PORT:END]
```

Fig 3.2 Port of Client 1

```
PORT
[PORT:SUCCESS]
PORT:8000
[PORT:END]
```

Fig 3.3 Port of Client 2

[10.0] LIST

The list command executed over client and server retrieves the client information which are currently logged in. This is done so by retrieving the data from the listInfo structure which stores the data of the clients, and listPointer to point the data. The data of currently logged in clients are checked by using the list_id and if the status is logged-in, the list command prints list_id, hostname, listIP and port.

The list command is tested using the autograder and it returns 10.0.

```
[COMMAND:LIST]
LIST
[LIST:SUCCESS]
1 euston.cse.buffalo.edu 128.205.36.34 7000
2 embankment.cse.buffalo.edu 128.205.36.35 8000
[LIST:END]
```

Fig 4 List with currently logged in clients

4 - Command/Event for Server

[5.0] STATISTICS

Similar to the list command, the statistics shell command retrieves the information about the client using the listPointer which points to the listInfo structure that stores the data of all the clients. The statistics command prints list_id, hostname, number of messages sent, received and the status of client.

The statistics command is tested using the autograder and it returns 5.0

```
STATISTICS
[STATISTICS:SUCCESS]
1  euston.cse.buffalo.edu      1    1    logged-in
2  embankment.cse.buffalo.edu  1    1    logged-in
[STATISTICS:END]
```

Fig 5 Statistics

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

The blocked command displays the list of all the clients that are blocked by the IP address given in the block command. The IP compares the block status via block ID (bid) of all the client IP addresses in the blockList structure which is being pointed by clientPointer. Once it finds the bid to be not 0, It displays the hostname, IP, and the port.

Exception handling: Invalid IP is checked using validateIP().

[EVENT]: Message Relayed

This event is used to print the sender client's IP and its respective message in the server to understand who the sender and receiver is and what the message is.

```
[RELAYED:SUCCESS]
msg from:128.205.36.34, to:128.205.36.35
[msg]:HI
Done!
[RELAYED:END]
```

Fig 6 Message relayed information


5 - Command/Event for Client

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

The login command extracts the server IP, server port and creates an address structure to store the respective port and IP in it. This address is connected using the connect() system call and its respective file descriptor is returned and stored in fdsocket. This fdsocket is added to the client master list and the clientLoginFlag is set to 1, to notify the server that atleast one client is connected.

Exception handling: Validation of IP is done using validateIP(). This function checks the address structure of the IP by comparing it with sockaddr_in.sin_addr's IP structure.

Exception handling: Validation of port is done by verifying if the port values are digits. Finally, we check if the port lies in the range of 1-65535 by converting the port string to integer type by predefined atoi().



```
LOGIN 128.205.36.46 6000
Logged in
[LOGIN:SUCCESS]
[LOGIN:END]
```

Fig 7 LOGIN


[5.0] REFRESH

The refresh command checks if any of the clients are logged in using the clientLoginFlag and updates the currently logged in clients.

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

The send shell command extracts the command, IP, message from the string parsed as input in the client's shell SEND call. After extracting, the command, IP and the message is copied to a new structure called clientMessageStruct. This structure is sent to the server where it checks the receiver IP and the command to relay it to the corresponding IP mentioned in the structure. The receiver client extracts the message from the structure and displays it.

Exception handling: Invalid IP is checked using validateIP().



```
SEND 128.205.36.35 HI
[SEND:SUCCESS]
[SEND:SUCCESS]
[SEND:END]
```

Fig 8.1 SEND message from client

```
[RELAYED:SUCCESS]
msg from:128.205.36.34, to:128.205.36.35
[msg]:HI
Done!
[RELAYED:END]
```

Fig 8.2 Message information displayed in server

[10.0] BROADCAST <msg>

The broadcast shell command extracts the command, message from the string parsed as input in the client's shell BROADCAST call. After extracting, the command and the message is copied to the structure called clientMessageStruct (as mentioned in SEND shell command). The structure is sent to the server where the command is checked and once BROADCAST command is checked, the message is sent to all the logged-in clients using looping structure.

```
BROADCAST HELLO ALL HI
[BROADCAST:SUCCESS]
[BROADCAST:END]
```

Fig 9.1 BROADCAST message from client

```
[RELAYED:SUCCESS]
MESSAGE from:128.205.36.34, to:255.255.255.255
[MSG:]:HELLO ALL HI5
[RELAYED:END]
```

Fig 9.2 Broadcast message information displayed in server

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

The block command extracts the IP and the command and sends it to the server. The server receives it and checks for the corresponding IP and other IP's block id (bid). If the block id is not zero, then it checks if the block IP corresponding to it is null for all logged in clients (max four). If the comparison is success, i.e., the IP is null, then the clientPointer's respective block IP is set to sender's IP.

Exception handling: Invalid IP is checked using validateIP().

```
BLOCK 128.205.36.35
Entered ip is:- 128.205.36.35
[BLOCK:SUCCESS]
[BLOCK:END]
```

Fig 10.1 Blocking a client IP

```
BLOCK SENDER IP:-128.205.36.34[RELAYED:SUCCESS]
```

Fig 10.2 Block information displayed in server

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

The unblock command extracts the IP and the command and sends it to the server. The server receives it and checks for the corresponding IP and other IP's block id (bid). If the block id is not zero, then it checks the block IP corresponding to the sender's IP for all logged in clients (max four). If the comparison is success, i.e., the IP is equal to the sender's IP, then the clientPointer's respective block IP is set to null.

Exception handling: Invalid IP is checked using validateIP().

```
UNBLOCK 128.205.36.35
[UNBLOCK:SUCCESS]
[UNBLOCK:END]
[RECEIVED:SUCCESS]
```

Fig 11 Unblocking a client

[2.5] LOGOUT

The logout command retrieves the file socket descriptor value of the current client port that is requesting to logout, and it binds with the port mentioned. Once the bind is successful, it closes the connection with the file socket descriptor.

The logout command is tested using the autograder and it returns 0.5.

```
LOGOUT
[LOGOUT:SUCCESS]

LOGOUT BIND SUCCESS
[LOGOUT:END]
```

Fig 12 Logging out

[2.5] EXIT

The exit shell command closes the socket and terminates its connection with the server, and it prints the acknowledgement for it.

The exit command is tested using the autograder and it returns 2.5.

```
EXIT
[EXIT:SUCCESS]
[EXIT:END]
enbankment {/local/Fall_2021/ovlyaaba/cse489589_assignment1/ovlyaaba} > 
```

Fig 13 Exit

[EVENT]: Message Received

This event is used to print the sender client's IP and its respective message in the receiver's client to understand who sent the message and what the message is.

```
[RECEIVED:SUCCESS]
MESSAGE from:128.205.36.35
[message]:HELLO HI
[RECEIVED:END]
```

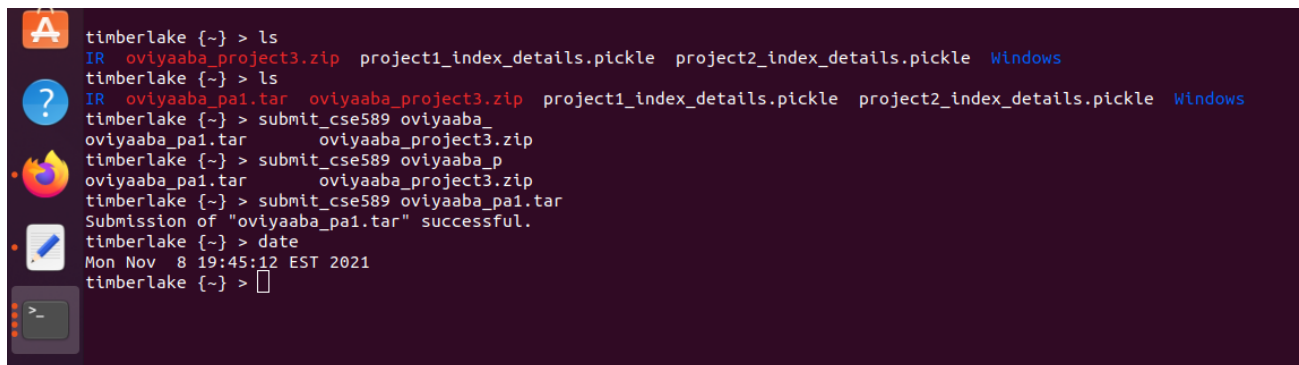
Fig 14 Message received information

6 - BONUS: Peer-to-peer (P2P) file transfer

[20.0] SENDFILE <client-ip> <file>

(Describe your work here...)

Submission Proof:- Date: 11/08/2021 – 19:45:12

A terminal window with a dark background and light-colored text. On the left side, there are four circular icons: an orange 'A' icon, a blue question mark icon, a Firefox logo, and a blue notepad icon. The terminal text shows a series of commands and their outputs. The user runs 'ls' and lists several files including 'oviyaaba_project3.zip', 'project1_index_details.pickle', and 'project2_index_details.pickle'. Then, they run 'submit_cse589' multiple times with different file names, and finally 'date' which shows 'Mon Nov 8 19:45:12 EST 2021'. The last command is a prompt for more input.

```
timberlake {~} > ls
IR oviyaaba_project3.zip project1_index_details.pickle project2_index_details.pickle Windows
timberlake {~} > ls
IR oviyaaba_pa1.tar oviyaaba_project3.zip project1_index_details.pickle project2_index_details.pickle Windows
timberlake {~} > submit_cse589 oviyaaba_
oviyaaba_pa1.tar oviyaaba_project3.zip
timberlake {~} > submit_cse589 oviyaaba_p
oviyaaba_pa1.tar oviyaaba_project3.zip
timberlake {~} > submit_cse589 oviyaaba_pa1.tar
Submission of "oviyaaba_pa1.tar" successful.
timberlake {~} > date
Mon Nov 8 19:45:12 EST 2021
timberlake {~} >
```