

Operating Systems

Assignment 2

Aarushi Agarwal 2016216

Surabhi S Nath 2016271

Multi Chat Server

→ Description of your code and how you implemented the function – the logical and implementation details

We implemented multi chat server system using socket inter-process communication. There are two files: ourserver.c and ourclient.c

Multiple clients connect to the socket through different pthreads. The server acts as a mediator in communication across clients.

The server creates a socket, binds to it and listens for client requests. We maintain an array of clients and each time a new client connects to the socket and the server side accepts the incoming connection, the client is added to the array, client count is increased by 1 and a new thread is spawned corresponding to that client. The manage_clients() function handles and controls communication between clients.

We handle three types on inter client communication:

- **PUBLIC** message: a client sends message to all other clients
- **PRIVATE** message: a client sends message to a particular client
- **GROUP** message: a client sends message to a selected other clients

Notifications of a new client joining, on an existing client leaving is sent to all clients.

We implemented 3 functions:

- **send_all()**: send message to all clients
- **send_all_exceptme()**: send message to all clients except sender
- **send_private()**: send message to a particular client

When a client attempts to send a message, either of these 3 formats can be used according to type of communication:

- For public message, **ALL (message)** or directly **(message)** would send message across to all clients except sender.

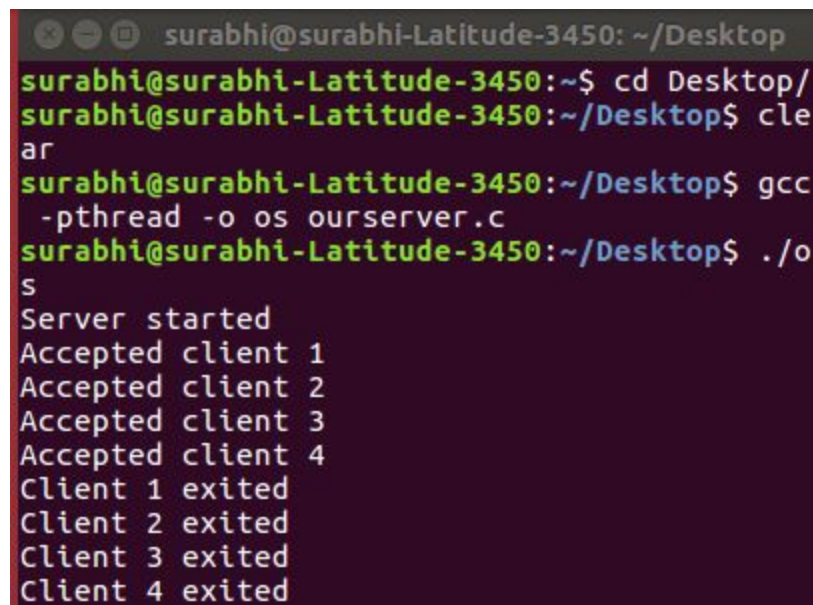
- For private message, **PRIVATE (id to send) (message)** would send message across to client with mentioned id.
- For group message, **GROUP (message) ([ids to send message to])** would send the message to clients with mentioned ids.

The message and ids are extracted using `strtok()` and according to format, the functions are called and communication between clients is conducted.

→ Description of how to compile and test the program

For compiling our code, write `make` command which executes the `makefile` and compiles the c code files and generates .o executables. Multiple clients are represented through different terminals.

For testing, messages can be sent corresponding to each type: PUBLIC, PRIVATE, GROUP. This can be seen in the following image:



```

surabhi@surabhi-Latitude-3450: ~/Desktop
surabhi@surabhi-Latitude-3450:~$ cd Desktop/
surabhi@surabhi-Latitude-3450:~/Desktop$ cle
ar
surabhi@surabhi-Latitude-3450:~/Desktop$ gcc
-pthread -o os ourserver.c
surabhi@surabhi-Latitude-3450:~/Desktop$ ./o
s
Server started
Accepted client 1
Accepted client 2
Accepted client 3
Accepted client 4
Client 1 exited
Client 2 exited
Client 3 exited
Client 4 exited

```

Server Side

<pre> surabhi@surabhi-Latitude-3450: ~/Desktop surabhi@surabhi-Latitude-3450:~/Desktop\$./cl3 Client 3 joined Client 4 joined Client 1 sent public message: HELLO Client 1 sent public message: HELLO Client 2 sent group message: HEYY PRIVATE 4 SUP HIHIHI Client 4 sent public message: BYE Client 4 sent public message: bye Client 4 sent public message: exit Client 2 sent public message: exit Client 1 left Client 2 left ^C surabhi@surabhi-Latitude-3450:~/Desktop\$ </pre>	<pre> surabhi@surabhi-Latitude-3450: ~/Desktop surabhi@surabhi-Latitude-3450:~/Desktop\$./cl1 Client 1 joined Client 2 joined Client 3 joined Client 4 joined HELLO ALL HELLO PRIVATE 2 YESS Client 4 sent public message: BYE Client 4 sent public message: bye Client 4 sent public message: exit Client 2 sent public message: exit ^C surabhi@surabhi-Latitude-3450:~/Desktop\$ </pre>
<pre> surabhi@surabhi-Latitude-3450: ~/Desktop surabhi@surabhi-Latitude-3450:~/Desktop\$./cl4 Client 4 joined Client 1 sent public message: HELLO Client 1 sent public message: HELLO Client 2 sent group message: HEYY Client 3 sent private message: SUP HIHIHI BYE bye exit Client 2 sent public message: exit Client 1 left Client 2 left Client 3 left ^C surabhi@surabhi-Latitude-3450:~/Desktop\$ </pre>	<pre> surabhi@surabhi-Latitude-3450: ~/Desktop surabhi@surabhi-Latitude-3450:~/Desktop\$./cl2 Client 2 joined Client 3 joined Client 4 joined Client 1 sent public message: HELLO Client 1 sent public message: HELLO Client 1 sent private message: YESS GROUP HEYY [3 4] Client 4 sent public message: BYE Client 4 sent public message: bye Client 4 sent public message: exit exit Client 1 left ^C surabhi@surabhi-Latitude-3450:~/Desktop\$ </pre>

Client Side

→ The inputs the user should give

The messages to be send according to type in correct format is the input the client or user should provide. For PRIVATE and GROUP message, the user should also enter the PIDs to which the message needs to be sent.

→ Expected output (and how to interpret it)

Statements are printed for every activity like when the server starts, when the server accepts a client, when a clients exists, and several error statements indicating failure at any step.

→ Error values and how to interpret them

Apart from handling socket errors:

- Error in binding
- Error in listening
- Error in select
- Error in socket

- Invalid address
- Error in connect

We have handled the following errors:

- Input errors:
 - If after PRIVATE, an integer is not inputted, then “**Error: id should be a positive integer**” is printed.
 - If the id entered is not a valid PID, “**Error: This client does not exist**” is printed.
- Too many users: **EUSERS, returns 87**. This error is displayed when the number of users exceeds the MAX_CLIENTS.
- Message too long: **EMSGSIZE, returns 90**. This error is displayed when the size of message exceeds the permissible set size of array.