INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Department of Computer Science and Engineering

CS69202 – Design Lab

Assignment - 7

# Multi-Client Chat server

Submitted By: Shailesh Chaudhary(22CS60R37)

## Introduction

- In this assignment we implemented a client-server chat application using TCP sockets. There will be a single server and multiple clients communicating with the server. The server process can handle at most 10 concurrent connections. Each client process will open a new connection with the server and add the client socket id to its fd_set(). Use select to handle multiple client requests.

## Implementation

- We implemented two program using C in this assignment:
    1. Client
    2. Server

➢ **Implementation of Client:**

The program creates a client socket and connects to a server socket at IP address 127.0.0.1 and port number 5001 using the socket() and connect() system calls respectively.

1. The program forks a child process to read messages from the server and a parent process to write messages to the server.

2. The child process reads messages from the server using the read() system call and prints them to the console using printf(). It exits if it receives the string "YY" from the server.

3. The parent process reads input from the user using the fgets() function, writes it to the server using the write() system call, and repeats this process indefinitely.

4. The program uses various standard C libraries like stdio.h, sys/types.h, stdlib.h, unistd.h, string.h, sys/socket.h, netinet/in.h, netdb.h, signal.h to implement the functionalities.

5. The program initializes a sockaddr_in struct with the server IP address, port number, and address family.

6. The program uses the bzero() function to initialize buffers to zero before use.

7. The program uses the pid_t data type to store the process ID of the child process.

8. The program checks the return value of fork() to handle errors during process creation.

9. The program uses the kill() function to terminate the parent process when the child process receives the string "YY" from the server.

10. The program uses the strncmp() function to compare strings with a specified number of characters.

11. The program closes the client socket using the close() system call before exiting.

➢ **Implementation of Server:**

- In the server part we implemented different function using which client can send request to server to perform different actions.
- Uses select() for implementation of server
- The server code contain three structure that is used for store information that is going to help to execute different command

    i. strurct group_data, this is used to store group information the code is given below:

```
// structure to store group data
struct group_data
{
    int gid;
    int mem[5];   // used to store member in grp
    int ind;      // used to count number client in group
    int admin[5]; // used to store admin in a grp
    int gON;
    int adminCnt;
    int reqCnt;
    bool broadcast; // used
};
```

ii. struct client_data, this is used to store client information the code is given below:

```
// structure to store client data
struct client_data
{
    int id; //used to store id of client
    int sid; //used to store socket id of client
    bool ON; //used to check client is active or not
};
```

i. struct req_admin, this is used to store adminreq information the code is given below:

```
// structure to store request to become admin
struct req_admin
{
    int rcnt; // used to store the request of client
    int acnt; // used to keep track approved count
    int dcnt; // used to keep track decline count
    int rcid; // used to store client id
    int rgid; // used to store group id
};
```

- There is list of command function implemented in server side they are listed below:

1. **/active** : To display all the available active clients that are connected to the server.
2. **/send** : To send a message to the client corresponding to
3. **/broadcast** : Message should be broadcasted to all the active clients
4. **/makegroup <client id1> <client id2> ….<client idn>**:A group with unique id will be made including all the mentioned clients along with the admin client.
5. **/makegroupreq <client id1> <client id2> ….<client idn>:**A group having unique id should be made with currently only the admin client. The request message for joining the group should be notified to all the specified clients. Clients can respond to join that group.
6. **/joingroup <group id>** : If this message is sent by a client having the request for joining the group, then he will be added to the group immediately.
7. **/declinegroup <group id>:** If this message is sent by a client having the request for joining the group, then the client will not be added to the group.
8. **/sendgroup <group id> <messege>** : The sender should be in the group to transfer the message to all his peers of that group. The message should be sent to all the peers along with group info.

9. **/makeadmin <group id> <client id>** : The admin should be able to alleviate the privileges of client id to that of admin.

10. **/addtogroup <group id> <client id1> <client id2> ….<client idn>:** The admin should be able to add member(s) to the group.

11. **/removefromgroup <group id> <client id1> <client id2> ….<client idn>**: The admin should be able to remove member(s) from the group

12. **/makegroupbroadcast <group id>** : Any admin of the group should be able to modify the type of group as broadcast-only in which only admins are allowed to message.

13. **/activegroups** : To display all the groups that are currently active on the server and the sender is a part of. Here you have to display at the client side the group ids followed by the group admin's client id, and the ids of all the clients that are part of this group.

14. **/quit** : The client will be removed from the server. This client will be removed from all the active groups

15. **/makeadminreq <group id>** : A member of a group should use this functionality to make a request to become an admin. All the admin should be notified of this request and they can respond appropriately.

16. **/approveadminreq <group id> <client id>** : An admin of a group can approve the /makeadminreq using this functionality.

17. **/declieadminreq <group id> <client id>** : An admin of a group can decline the /makeadminreq using this functionality.