JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & I.T.

2018-2019

विद्या तत्व ज्योतिसम:

OPERATING SYSTEMS PROJECT REPORT

Chat Bash

A Terminal Chatting Client

SUBMITTED BY:

Aman Verma	17104006	B11
Siddhant N. Trivedi	17104025	B11
Mayank Singh	17104029	B11
Aditya Raushan	17104017	B11

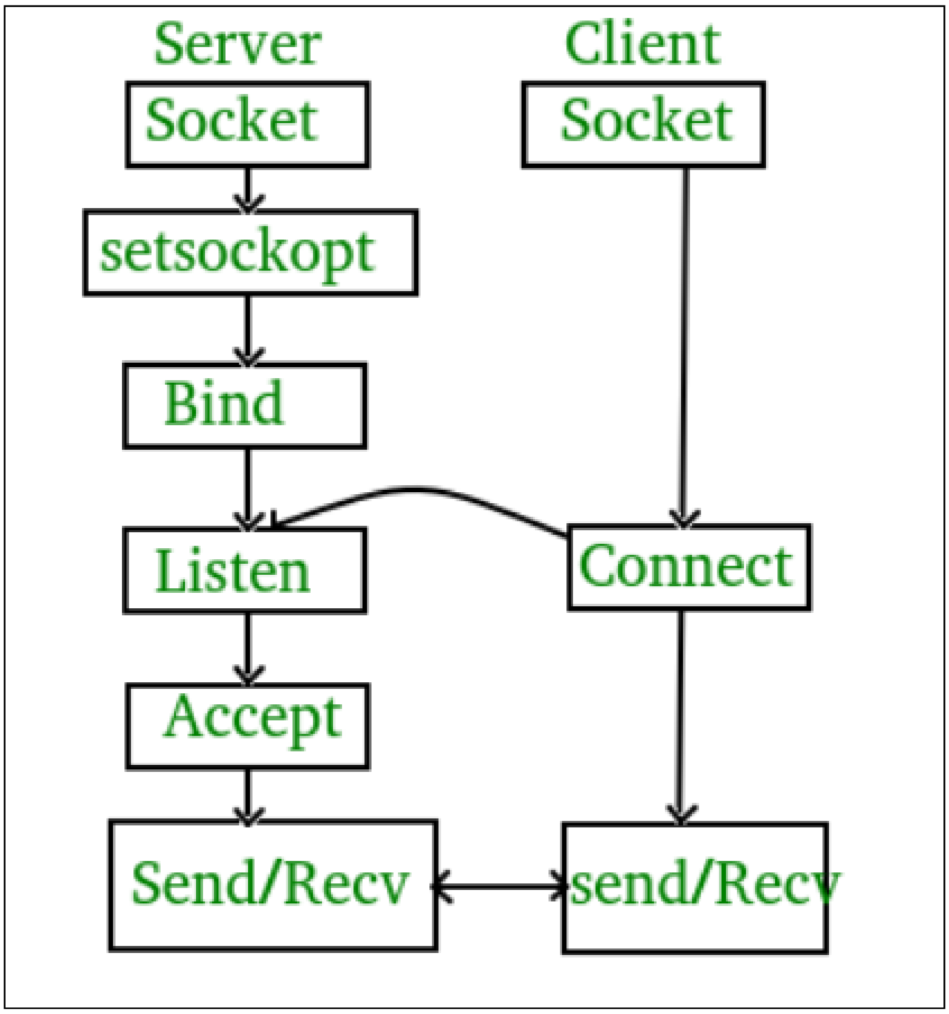
INTRODUCTION

Chat Bash

Communication, It is one of the foremost important thing which existed from the prehistoric time of human evolution, without that It is impossible to imagine the kind of life we lead today, it started with scribbling on the walls, to engraved manuscripts, to using paper extracted from wood, to printed books and finally in the digital era to communication over the network, people have experimented with the kind of instruments they have been using to communicate with others, they have been using telephones, mobile phones, the beloved internet and so on, they have used emails, chat platforms which are hosted within the social media & networking websites, which has led to even faster communication, we have attempted to create something similar by implementing our own chatting room based on linux based distributions & socket programming using concepts learned from Operating Systems & System Programming course.

ABOUT SOCKET :

It is an internal endpoint for two way communication link between two programs running on the network or simple sending or receiving data within nodes on a computer network, normally a server runs on a specific computer and has a socket that is bound to specific port number, the server just waits, listening to the socket for a client to make a connection request, on the client side - the client knows the hostname of the machine on which the server is running and the port number on which the server is listening, to make a connection request the client tries to rendezvous with the server on the sever's machine and port,working of socket is explained through this visual.



<u>FEATURES OF OUR APPLICATION</u> :

We have implemented a commands based approach in our chatting client **Chat Bash**, which helps us to make our application more interactive, we have also used concepts from operating systems course to make our app support multi client connections, also the user feeds in commands through standard input, the commands currently supported are: **/q or /quit** | **/h or /help** | **/list or /l** | **/m** , all these commands have specific purpose which are explained in the following table.

COMMANDS	USE CASE
/quit or /q:	Exit the program.
/help or /h:	Displays help information.
/list or /l:	Displays list of users in chat room.
/m <username> <message> :	Send a private message to given username

DIVISION OF WORK AMONG GROUP MEMBERS

MEMBER NAME	CONTRIBUTIONS
Aman Verma	Server Side and Documentation
Siddhant N. Trivedi	Client Side and Documentation
Mayank Singh	Client Side and Documentation
Aditya Raushan	Server Side and Documentation

METHODS IMPLEMENTED IN OUR APPLICATION :

Server.c

trim_newline() : Changes the last character of the message from '\n' to '\0' to erase the extra line character attached when the user entered the message.

clear_stdin_buffer() : Clears the extra buffer entered when the user type the message.

socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) : Create an endpoint for communication and return a file descriptor that refers to that endpoint. The parameters are -

- **AF_INET** IPv4 Internet protocols
- **SOCK_STREAM** Provides sequenced, reliable, two-way, connection-based byte streams.
- **IPPROTO_TCP** Protocol for TCP connection.

inet_addr(char *address) : Convert Internet host address from numbers-and-dots notation into binary data in network byte order.

bind() : Gets the unique name of the socket.

initialize_server() : Initialises the server by starting socket using socket(), inet_addr() and bind() function.

send() : Send the message through the socket.

send_public_message() : Sends the message to all the clients connected to the server through sockets.

send_private_message() : Sends the message to a specific user with the help of sender id.

send_connect_message() : Sets the status of message to SUCCESS if the message has been sent successfully otherwise prints “Send failed”.

send_disconnect_message() : Sets the status of message to DISCONNECT and prints “Send failed”.

send_user_list() : Displays the list of all the client connected to the server.

send_to_full_message() : prints “Send failed” if the message length exceeds the limit.

stop_server() : Stops the server by closing the socket.

handle_client_message() : Handles the message send by the client and forward it to all clients or specific client after verifying the flag used.

select() : select() allows a program to monitor multiple file descriptors, waiting until one or more of the file descriptors become ready for input/ouput. A file descriptor is considered ready if it is possible to perform the corresponding I/O operation without blocking.

FD_ZERO() : Initializes the file descriptor set *fdset* to have zero bits for all file descriptors.

FD_SET() : Sets the bit for the file descriptor *fd* in the file descriptor set fdset.

FD_ISSET() : Returns a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set pointed to by *fdset*, and 0 otherwise.

construct_fd_set() : Uses FD_SET(), FD_ZERO() to initialize to file descriptor fd.

handle_new_connection() : Checks if the limit exceeds the number of users to be connected and initialize the socket to each connection .

handle_user_input() : Handles the user input with help of trim_newline() function and closes the socket for that specific client if the client type ‘q’.

Client.c

get_username() : function to get the username from the user and checks if it is less than 20 characters.

set_username() : sets the entered username to the username attribute of the message structure initialised in chatroom_utils.h

send() : Used to transmit a message to another socket.Declared in sys/socket.h

stop_client() : Used to close the connection to another socket.

connect_to_server() : Takes the address and port of the server as parameters and makes a connection to the server using the socket() function included in sys/socket.h library.

socket(AF_INET, SOCK_STREAM, IPPROTO_TCP) : function to create an endpoint for communication and return a file descriptor that refers to that endpoint. The parameters are -

- **AF_INET** IPv4 Internet protocols
- **SOCK_STREAM** Provides sequenced, reliable, two-way, connection-based byte streams.
- **IPPROTO_TCP** Protocol for TCP connection.

inet_addr(char *address) : Convert Internet host address from numbers-and-dots notation into binary data in network byte order.

htons(port) : The htons() function converts the unsigned short integer hostshort from host byte order to network byte order.

connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen) : The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr.The addrlen argument specifies the size of addrlen.

recv(int sockfd, void *buf, size_t len, int flags) : Used to receive message from the socket using the file descriptor fd, the *buf character array contains the message received, len contains the size of the message received. The flags contain the nature of request and the response needed from the receiving end.

handle_user_input() : Used to handle the input given by the user. Maps the input to a certain task.

handle_server_message() : Takes the connection info as a parameter and handles receiving of message from the server.

select() : select() allows a program to monitor multiple file descriptors, waiting until one or more of the file descriptors become ready for input/output. A file descriptor is considered ready if it is possible to perform the corresponding I/O operation without blocking.

FD_ZERO() : Initializes the file descriptor set *fdset* to have zero bits for all file descriptors.

FD_SET() : Sets the bit for the file descriptor *fd* in the file descriptor set fdset.

FD_ISSET() : Returns a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set pointed to by *fdset*, and 0 otherwise.