Simple and expanded multi-chat server and client via socket programming.

Simeon, J. s2966176
joseph.simeon@griffithuni.edu.au

I. Credit

I would to credit Brian "Beej Jorgensen" Hall as this project utilises his select server and client as a template.

II. Introduction

The correspondence across the internet is another method to add to world communication, this correspondence would exist in the interaction between persons accessing clients connected to servers, with this specific instance via socket programming. Clients access the server to send and receive messages with the server being the centralised hub for message exchange, this is the simpleness of the client-server relationship with complexity of the relationship increasing as more functionality is needed for the client-server to perform rather than the simplicity of the basic client-server relationship.

III. Communication Protocol

A. Connection of Server to Client

The communication protocol of the Client-server interaction is based on network socket programming, the server creates an open socket that allows for a client to connect, the connection is created via the server's host's available socket. From the client end, a connection is sent and waited for the server to accept and when the accepted connection has been created this allows the correspondence between the server and client through the use of send and receive functions, however for a multi-chat server in which multiple clients are connected, utilising the select function will allow the server to search for clients sending messages via a socket descriptor array using 'FD' functions coupled with the select function. The client utilises parallel programming in the form of a thread used to receive messages while the main process sends messages until the user has cancelled interaction between the client and server.

B. Functionality of the Simple Client-server Multi-chat

The functionality of the simple client-server is that the simple server uses a select function to sort through accepted sockets that stored in a socket descriptor array / 'FD_SET' to check whether any socket has incoming messages which the server will send out to all connections that is not the sending connection. Within the server is a username check and storage which will accept a username six to ten characters long and will check against the stored list of usernames to either fully accept a connection and store the username or deny the connection. Storing the username that is linked with the user's socket connection allows for the server to add the sending user's name to the sent message as a marker to let other users know who sent the message. A logfile is used within the server to track the connections and happenings within the server itself.

C. Functionality of the Expanded Client-server Multi-chat

The functionality of the expanded client-server includes the same functionality of the simple client-server however changes include logfile being saved in terms of current date and time-stamping of messages being sent from clients and time-stamping of logged events, users are placed into public chat and can access between private and public chat, users can see current online users with a difference between public online and private online, and users can private message each other known as whispers. The commands of the client to the server are sent via the send function, when received on the server side the commands are compared with the 'strncmp()' function, upon entering the if statement of compare the server will start complying with the command the user has sent, such as sending the private message known as a whisper by searching the for the corresponding username's index and using that index to send to the socket that is used by the index, as well as requesting a the private server access via password gives a user a different priority that allows the user to be separate from the public chat and vice versa with the private chat. The same priority method is used when the user prompts to see people online, if the user is in public chat than the online person(s) that the requested can only see people within public leaving person(s) in private chat hidden, this goes for the same if the requested person happens to be in private chat.

IV. Statement of Completeness

All functionality discussed within the previous section of this report is working in full, all source code for both client and server can be viewed below with screen dump of the functionality working between the Simple and Expanded Client-Server programs.

V. Simpler Client-server Multi-Chat

A. Simple Server Source

```
/*
--- DIRECTIONS ---
selectserver.c
cc -o selectserver.out selectserver.c
./selectserver.out

--- DOCUMENTATION ---
Joseph Simeon
6305ENG: Advanced Computer Systems
Nathan Campus
Griffith University

--- CREDIT ---
I would like to give credit to Beej's Guide to Network Programming as the selectserver.c "a cheezy mul
tiperson chat server" as the base template for this program.
ADRSS: https://beej.us/guide/bgnet/html/multi/advanced.html#select

--- TO DO LIST ---

*/

// the usual suspects
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
// networking & others
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
// defines
#define PORT "9034" // port to listen to
#define MAXDATASIZE 100 // user allowance
#define MAXSIZE 115 // username + ' >> ' + user allowance + '\0'
#define USERS 20 // max users in chat room
#define MAXUSERSIZE 10
// username only 10 characters // get sockaddr, IPv4 or IPv6:
void * get_in_addr(struct sockaddr * sa) {
    if (sa - > sa_family == AF_INET) {
        return &(((struct sockaddr_in * ) sa) - > sin_addr);
    }
    return &(((struct sockaddr_in6 * ) sa) - > sin6_addr);
}
int main(void) {
    fd_set master; // master file descriptor list
    fd_set read_fds; // temp file descriptor list for select()
    int fdmax; // maximum file descriptor number
    int listener; // listening socket descriptor
    int newfd; // newly accept()ed socket descriptor
    struct sockaddr_storage remoteaddr; // client address
    socklen_t addrlen;
    char buf[MAXDATASIZE]; // buffer for client data
    char message[MAXSIZE]; // message to be sent
    char username[USERS][MAXUSERSIZE]; // user database
    char usertemp[MAXUSERSIZE];
```

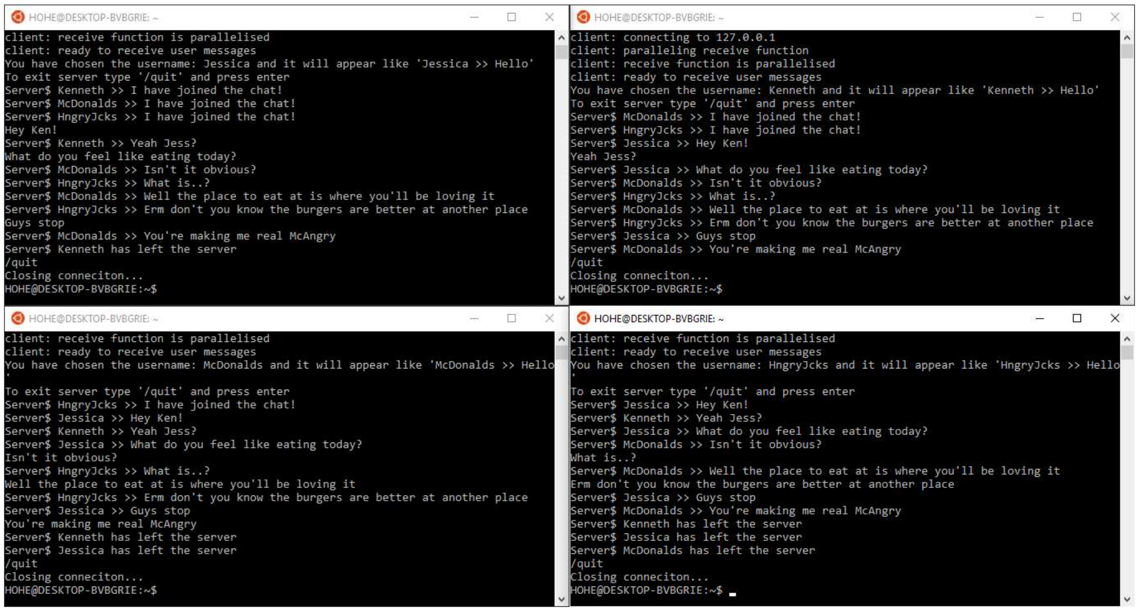
```c
    int nbytes, size;
    char remoteIP[INET6_ADDRSTRLEN];
    int yes = 1; // for setsockopt() SO_REUSEADDR, below
    int i, j, k, m, rv, usercheck = 0;
    FILE * fp;
    char fname[50] = "logfile.txt";
    struct addrinfo hints, * ai, * p; // check to see if file exists as well as open and append
    fp = fopen(fname, "a");
    fprintf(fp, "selectserver: logfile\n");
    fclose(fp); // clear the master, temp and buffers;
    FD_ZERO( & master);
    FD_ZERO( & read_fds);
    memset(buf, 0, MAXDATASIZE * sizeof(char));
    memset(message, 0, MAXSIZE * sizeof(char));
    for (k = 0; k < USERS; k++) {
        username[k][0] = '\0';
    } // get us a socket and bind it
    memset( & hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;
    if ((rv = getaddrinfo(NULL, PORT, & hints, & ai)) != 0) {
        fprintf(stderr, "selectserver: %s\n", gai_strerror(rv));
        exit(1);
    }
    for (p = ai; p != NULL; p = p - > ai_next) {
        listener = socket(p - > ai_family, p - > ai_socktype, p - > ai_protocol);
        if (listener < 0) {
            continue;
        } // lose the pesky "address already in use" error message
        setsockopt(listener, SOL_SOCKET, SO_REUSEADDR, & yes, sizeof(int));
        if (bind(listener, p - > ai_addr, p - > ai_addrlen) < 0) {
            close(listener);
            continue;
        }
        break;
    } // if we got here, it means we didn't get bound
    if (p == NULL) {
        fprintf(stderr, "selectserver: failed to bind\n");
        exit(2);
    }
    freeaddrinfo(ai); // all done with this // listen
    if (listen(listener, 10) == -1) {
        perror("listen");
        exit(3);
    } // add the listener to the master set
    FD_SET(listener, & master); // keep track of the biggest file descriptor
    fdmax = listener; // so far, it's this one // main loop
    for (;;) {
        read_fds = master; // copy it
        if (select(fdmax + 1, & read_fds, NULL, NULL, NULL) == -1) {
            perror("select");
            exit(4);
        } // run through the existing connections looking for data to read
        for (i = 0; i <= fdmax; i++) {
            if (FD_ISSET(i, & read_fds)) { // we got one!!
                if (i == listener) { // handle new connections
                    addrlen = sizeof remoteaddr;
                    newfd = accept(listener, (struct sockaddr * ) & remoteaddr, & addrlen);
                    if (newfd == -1) {
                        perror("accept");
                    } else {
                        FD_SET(newfd, & master); // add to master set
                        if (newfd > fdmax) { // keep track of the max
                            fdmax = newfd;
                        }
                        printf("selectserver: new connection from %s on "
                            "socket %d\n", inet_ntop(remoteaddr.ss_family, get_in_addr((struct sockaddr * ) & remoteaddr), remoteIP, INET6_ADDRSTRLEN), newfd);
                        fp = fopen(fname, "a");
                        fprintf(fp, "selectserver: new connection from %s on "
                            "socket %d", inet_ntop(remoteaddr.ss_family, get_in_addr((struct sockaddr * ) & remoteaddr), remoteIP, INET6_ADDRSTRLEN), newfd);
                        fclose(fp);
                    }
                } else { // handle data from a client
                    if ((nbytes = recv(i, buf, sizeof buf, 0)) <= 0) { // got error or connection closed by client
                        if (nbytes == 0) { // connection closed
                            printf("selectserver: socket %d hung up\n", i); // user has left the server message
                            fp = fopen(fname, "a");
                            fprintf(fp, "selectserver: socket %d hung up", i);
                            fclose(fp);
                            memset(message, 0, MAXSIZE * sizeof(char));
                            strcpy(message, username[i]);
                            strcat(message, " has left the server\n");
                            size = strlen(message);
                            fp = fopen(fname, "a");
                            fprintf(fp, "\nselectserver: %s", message);
                            fclose(fp); // Send message to everyone
                            for (j = 0; j <= fdmax; j++) { // send to everyone!
                                if (FD_ISSET(j, & master)) { // except the listener and ourselves
                                    if (j != listener && j != i) {
                                        if (send(j, message, strlen(message), 0) == -1) {
                                            perror("send");
                                        }
                                        fflush(stdout);
                                    }
                                }
                            }
                            username[i][0] = '\0'; // username removed
                        } else {
                            perror("recv");
                        }
                        close(i); // bye
                        FD_CLR(i, & master); // remove from master set
                    } else { // check username
                        if ((strlen(buf) >= 9) && (strlen(buf) <= 13)) {
                            if ((buf[0] == 'U') && (buf[1] == 'S') && (buf[2] == 'R')) {
                                m = 0;
                                for (k = 3; k <= strlen(buf); k++) {
                                    usertemp[m++] = buf[k];
                                }
                            } else {
                                usercheck = 1;
                            }
                            if (usercheck == 0) {
                                for (k = 0; k <= fdmax; k++) {
                                    if (strcmp(usertemp, username[k]) == 0) {
                                        send(i, "Username error", 14, 0);
                                        fp = fopen(fname, "a");
                                        fprintf(fp, "\nselectserver: username error on socket %d", i);

                                        fclose(fp);
                                    }
                                }
                                strcpy(username[i], usertemp);
                                strcpy(buf, "I have joined the chat!\n");
                                fp = fopen(fname, "a");
                                fprintf(fp, "\nselectserver: %s on socket %d has joined the server\n", username[i], i);
                                fclose(fp);
                                memset(usertemp, 0, MAXUSERSIZE * sizeof(char));
                            }
                            usercheck = 0;
                        }
                        memset(message, 0, MAXSIZE * sizeof(char));
                        strcat(message, username[i]);
                        strcat(message, " >> ");
                        buf[nbytes] = '\0';
                        strcat(message, buf);
                        fp = fopen(fname, "a");
                        fprintf(fp, "client: %s", message);
                        fclose(fp); // we got some data from a client
                        for (j = 0; j <= fdmax; j++) { // send to everyone!
                            if (FD_ISSET(j, & master)) { // except the listener and ourselves
                                if (j != listener && j != i) {
                                    if (send(j, message, strlen(message), 0) == -1) {
                                        perror("send");
                                    }
                                    fflush(stdout);
                                }
                            }
                        }
                    }
                } // END handle data from client
            } // END got new incoming connection
        } // END looping through file descriptors
    } // END for(;;)--and you thought it would never end!
    return 0;
}
```

B. Simple Client Source

```c
/*
--- DIRECTIONS ---
client.c
cc -pthread -o client.out client.c
./client.out hostname username

--- DOCUMENTATION ---
Joseph Simeon
6305ENG: Advanced Computer Systems
Nathan Campus
Griffith University

--- CREDIT ---
I would like to give credit to Beej's Guide to Network Programming as   the client.c "a stream socket client demo" as the base template for this program.
ADRSS: https://beej.us/guide/bgnet/html/multi/clientserver.html#simpleclient

--- TO DO LIST ---

*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <pthread.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#define PORT "9034" // the port client will be connecting to
#define MAXDATASIZE 100 // max number of bytes we can get at once
#define MAXUSERSIZE 10
```

```c
#define MAXSIZE 115 // thread data structure

typedef struct {
    char * prompt;
    int socket;
} thread_data; // thread receive function

void * receiveMessage(void * threadData) {
        int sockfd, rv;
        char message[MAXSIZE], * prompt;
        thread_data * localData = (thread_data * ) threadData;
        sockfd = localData - > socket;
        prompt = localData - > prompt;
        memset(message, 0, MAXDATASIZE * sizeof(message[0])); // print the received message

        while (1) {
            rv = recvfrom(sockfd, message, MAXSIZE, 0, NULL, NULL);
            if (rv == -1) {
                perror("receive");
                break;
            } else if (rv == 0) {
                printf("Connection error: user disconnected\n");
                break;
            } else {
                if (strcmp(message, "Username error") == 0) {
                    printf("Username has already been taken\n");
                    close(sockfd);
                    exit(0);
                }
                printf("Server%code%nbsp;%s", message); //printf("%s >> ", prompt); //fflush(stdout);
                memset(message, 0, MAXDATASIZE * sizeof(message[0]));
            }
        }
    } // get sockaddr, IPv4 or IPv6:
void * get_in_addr(struct sockaddr * sa) {
    if (sa - > sa_family == AF_INET) {
        return &(((struct sockaddr_in * ) sa) - > sin_addr);
    }
    return &(((struct sockaddr_in6 * ) sa) - > sin6_addr);
}
int main(int argc, char * argv[]) {
    int sockfd, numbytes;
    char message[MAXDATASIZE], * user;
    struct addrinfo hints, * servinfo, * p;
    int rv, userlen, DATASIZE;
    char s[INET6_ADDRSTRLEN];
    pthread_t thread;
    thread_data data;
    if (argc != 3) {
        fprintf(stderr, "usage: ./client.out hostname username\n");
        exit(1);
    } // check username length
    userlen = strlen(argv[2]);
    if (userlen > 10 || userlen < 6) {
        fprintf(stderr, "usage: ./client.out hostname username\n");
        fprintf(stderr, "Username error: must be between 6-10 characters\n");
        exit(1);
    } else {
        user = malloc(userlen * sizeof(char));
        memset(user, 0, userlen * sizeof(user[0]));
        strcpy(user, argv[2]);
    }
    memset( & hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    if ((rv = getaddrinfo(argv[1], PORT, & hints, & servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
                return 1;
            } // loop through all the results and connect to the first we can
            for (p = servinfo; p != NULL; p = p - > ai_next) {
                if ((sockfd = socket(p - > ai_family, p - > ai_socktype, p - > ai_protocol)) == -1) {
                    perror("client: socket");
                    continue;
                }
                if (connect(sockfd, p - > ai_addr, p - > ai_addrlen) == -1) {
                    perror("client: connect");
                    close(sockfd);
                    continue;
                }
                break;
            }
            if (p == NULL) {
                fprintf(stderr, "client: failed to connect\n");
                return 2;
            }
            inet_ntop(p - > ai_family, get_in_addr((struct sockaddr * ) p - > ai_addr), s, sizeof s);
            printf("client: connecting to %s\n", s);
            freeaddrinfo(servinfo); // all done with this structure // receive thread
            printf("client: paralleling receive function\n");
            data.prompt = user;
            data.socket = sockfd;
            pthread_create( & thread, NULL, receiveMessage, (void * ) & data);
            printf("client: receive function is parallelised\n"); // send loop
            memset(message, 0, MAXDATASIZE * sizeof(char));
            strcpy(message, "USR");
            strcat(message, user); //strcat(message, "I am ready to chat!\n");
            if ((send(sockfd, message, sizeof(message), 0)) == -1) {
                perror("send");
                close(sockfd);
                exit(0);
            } //printf("\n%s", message);
            printf("client: ready to receive user messages\n"); // entry message
            printf("You have chosen the username: %s and it will appear like '%s >> Hello'\n", user, user);
            printf("To exit server type '/quit' and press enter\n");
            while (fgets(message, MAXDATASIZE, stdin) != NULL) {
                if (strncmp(message, "/quit", 5) == 0) {
                    printf("Closing conneciton...\n");
                    exit(0);
                } //printf("client%code%nbsp;%s\n", message);
                send(sockfd, message, strlen(message), 0);
                memset(message, 0, MAXDATASIZE * sizeof(message[0]));
            }
            close(sockfd);
            pthread_exit(NULL);
            free(user);
            return 0;
        }
```

C. Simple Client-server Screen-dump



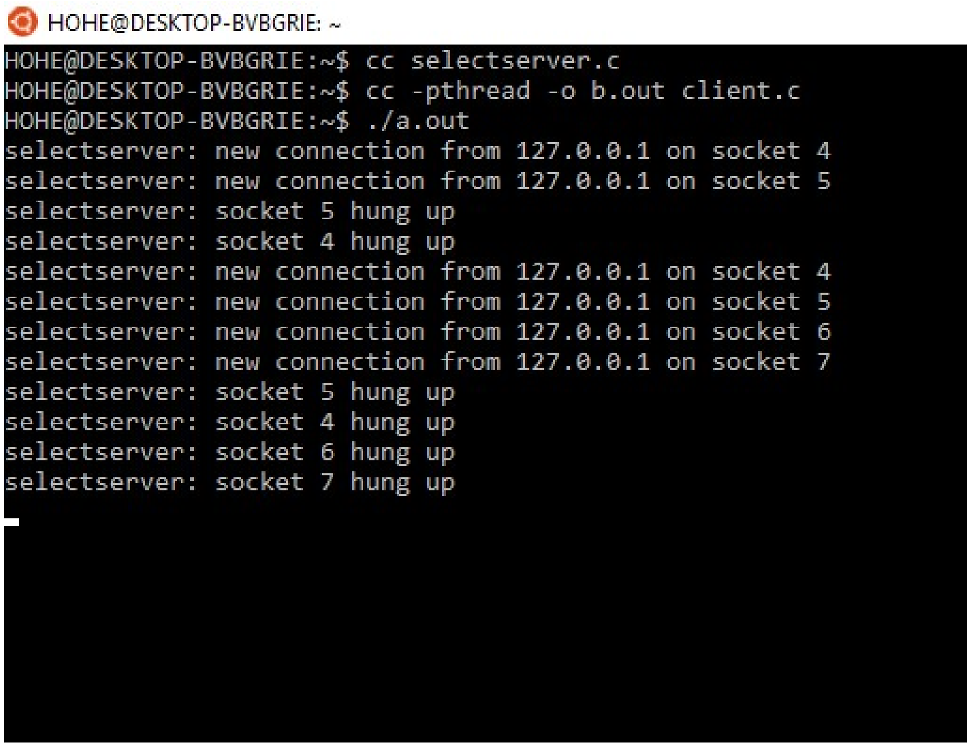
```
HOHE@DESKTOP-BVBGRIE: ~
client: receive function is parallelised
client: ready to receive user messages
You have chosen the username: Jessica and it will appear like 'Jessica >> Hello'
To exit server type '/quit' and press enter
Server$ Kenneth >> I have joined the chat!
Server$ McDonalds >> I have joined the chat!
Server$ HngryJcks >> I have joined the chat!
Hey Ken!
Server$ Kenneth >> Yeah Jess?
What do you feel like eating today?
Server$ McDonalds >> Isn't it obvious?
Server$ HngryJcks >> What is..?
Server$ McDonalds >> Well the place to eat at is where you'll be loving it
Server$ HngryJcks >> Erm don't you know the burgers are better at another place
Guys stop
Server$ McDonalds >> You're making me real McAngry
Server$ Kenneth has left the server
/quit
Closing conneciton...
HOHE@DESKTOP-BVBGRIE:~$
```

```
HOHE@DESKTOP-BVBGRIE: ~
client: connecting to 127.0.0.1
client: paralleling receive function
client: receive function is parallelised
client: ready to receive user messages
You have chosen the username: Kenneth and it will appear like 'Kenneth >> Hello'
To exit server type '/quit' and press enter
Server$ McDonalds >> I have joined the chat!
Server$ HngryJcks >> I have joined the chat!
Server$ Jessica >> Hey Ken!
Yeah Jess?
Server$ Jessica >> What do you feel like eating today?
Server$ McDonalds >> Isn't it obvious?
Server$ HngryJcks >> What is..?
Server$ McDonalds >> Well the place to eat at is where you'll be loving it
Server$ HngryJcks >> Erm don't you know the burgers are better at another place
Server$ Jessica >> Guys stop
Server$ McDonalds >> You're making me real McAngry
/quit
Closing conneciton...
HOHE@DESKTOP-BVBGRIE:~$
```

```
HOHE@DESKTOP-BVBGRIE: ~
client: receive function is parallelised
client: ready to receive user messages
You have chosen the username: McDonalds and it will appear like 'McDonalds >> Hello
'
To exit server type '/quit' and press enter
Server$ HngryJcks >> I have joined the chat!
Server$ Jessica >> Hey Ken!
Server$ Kenneth >> Yeah Jess?
Server$ Jessica >> What do you feel like eating today?
Isn't it obvious?
Server$ HngryJcks >> What is..?
Well the place to eat at is where you'll be loving it
Server$ HngryJcks >> Erm don't you know the burgers are better at another place
Server$ Jessica >> Guys stop
You're making me real McAngry
Server$ Kenneth has left the server
Server$ Jessica has left the server
/quit
Closing conneciton...
HOHE@DESKTOP-BVBGRIE:~$
```

```
HOHE@DESKTOP-BVBGRIE: ~
client: receive function is parallelised
client: ready to receive user messages
You have chosen the username: HngryJcks and it will appear like 'HngryJcks >> Hello
'
To exit server type '/quit' and press enter
Server$ Jessica >> Hey Ken!
Server$ Kenneth >> Yeah Jess?
Server$ Jessica >> What do you feel like eating today?
Server$ McDonalds >> Isn't it obvious?
What is..?
Server$ McDonalds >> Well the place to eat at is where you'll be loving it
Erm don't you know the burgers are better at another place
Server$ Jessica >> Guys stop
Server$ McDonalds >> You're making me real McAngry
Server$ Kenneth has left the server
Server$ Jessica has left the server
Server$ McDonalds has left the server
/quit
Closing conneciton...
HOHE@DESKTOP-BVBGRIE:~$
```

Figure 1. Screen dump of communication of multiple clients



```
HOHE@DESKTOP-BVBGRIE: ~
HOHE@DESKTOP-BVBGRIE:~$ cc selectserver.c
HOHE@DESKTOP-BVBGRIE:~$ cc -pthread -o b.out client.c
HOHE@DESKTOP-BVBGRIE:~$ ./a.out
selectserver: new connection from 127.0.0.1 on socket 4
selectserver: new connection from 127.0.0.1 on socket 5
selectserver: socket 5 hung up
selectserver: socket 4 hung up
selectserver: new connection from 127.0.0.1 on socket 4
selectserver: new connection from 127.0.0.1 on socket 5
selectserver: new connection from 127.0.0.1 on socket 6
selectserver: new connection from 127.0.0.1 on socket 7
selectserver: socket 5 hung up
selectserver: socket 4 hung up
selectserver: socket 6 hung up
selectserver: socket 7 hung up
```

Figure 2. Screen dump of server events

D. Simple Client-server Logfile

```
selectserver: new connection from 127.0.0.1 on socket 4
selectserver: Jessica on socket 4 has joined the server
client: Jessica >> I have joined the chat!
selectserver: new connection from 127.0.0.1 on socket 5
selectserver: Kenneth on socket 5 has joined the server
client: Kenneth >> I have joined the chat!
selectserver: new connection from 127.0.0.1 on socket 6
selectserver: McDonalds on socket 6 has joined the server
```

```
client: McDonalds >> I have joined the chat!
selectserver: new connection from 127.0.0.1 on socket 7
selectserver: HngryJcks on socket 7 has joined the server
client: HngryJcks >> I have joined the chat!
client: Jessica >> Hey Ken!
client: Kenneth >> Yeah Jess ?
client: Jessica >> What do you feel like eating today ?
client: McDonalds >> Isn 't it obvious?
client: HngryJcks >> What is.. ?
client: McDonalds >> Well the place to eat at is where you 'll be loving it
client: HngryJcks >> Erm don 't you know the burgers are better at another place
client: Jessica >> Guys stop
client: McDonalds >> You 're making me real McAngry
selectserver: socket 5 hung up
selectserver: Kenneth has left the server
selectserver: socket 4 hung up
selectserver: Jessica has left the server
selectserver: socket 6 hung up
selectserver: McDonalds has left the server
selectserver: socket 7 hung up
selectserver: HngryJcks has left the server
```

VI. Expanded Client-server

A. Expanded Server Source

```
/*
--- DIRECTIONS ---
selectserver.c
cc -o selectserver.out selectserver.c
./selectserver.out

--- DOCUMENTATION ---
Joseph Simeon
6305ENG: Advanced Computer Systems
Nathan Campus
Griffith University

--- CREDIT ---
I would like to give credit to Beej's Guide to Network Programming as the selectserver.c "a cheezy mul
tiperson chat server" as the base template for this program.
ADRSS: https://beej.us/guide/bgnet/html/multi/advanced.html#select

--- TO DO LIST ---

--- COMMANDS ---
/quit
/message username <your message goes here>
/private password
/public
/online

*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#define PORT "9034" // port to listen to
```

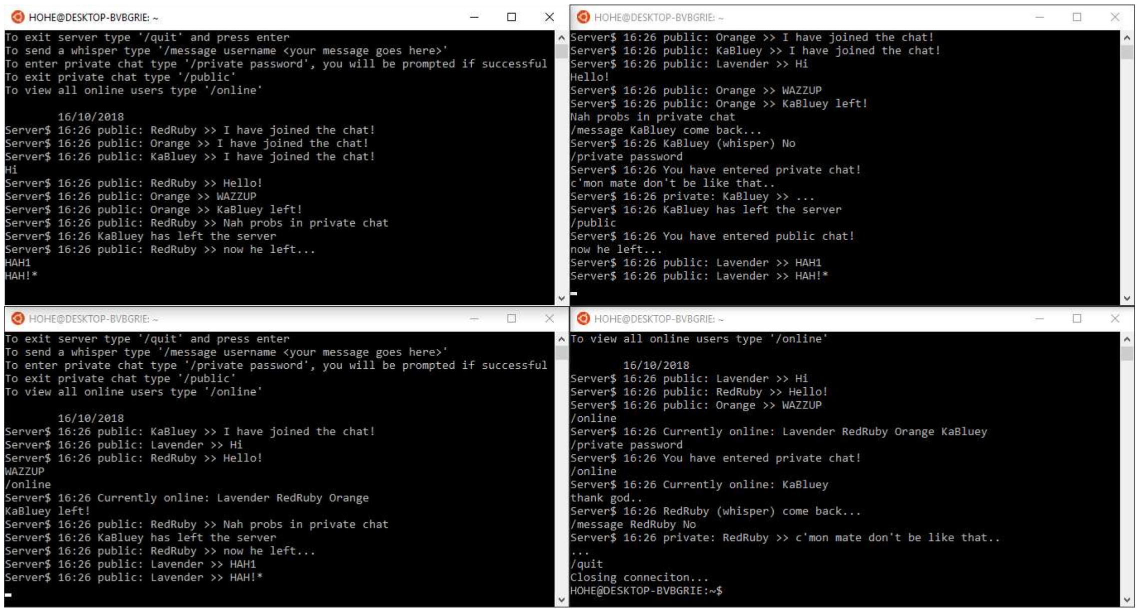
```c
#define MAXDATASIZE 100 // user allowance
#define MAXSIZE 150 // username + ' >> ' + user allowance + '\0'
#define USERS 20 // max users in chat room
#define MAXUSERSIZE 10

// username only 10 characters // get sockaddr, IPv4 or IPv6:
void * get_in_addr(struct sockaddr * sa) {
    if (sa - > sa_family == AF_INET) {
        return &(((struct sockaddr_in * ) sa) - > sin_addr);
    }
    return &(((struct sockaddr_in6 * ) sa) - > sin6_addr);
}

int main(void) {
    fd_set master; // master file descriptor list
    fd_set read_fds; // temp file descriptor list for select()
    int fdmax; // maximum file descriptor number
    int listener; // listening socket descriptor
    int newfd; // newly accept()ed socket descriptor
    struct sockaddr_storage remoteaddr; // client address
    socklen_t addrlen;
    char buf[MAXDATASIZE]; // buffer for client data
    char message[MAXSIZE]; // message to be sent
    char username[USERS][MAXUSERSIZE]; // user database
    char usertemp[MAXUSERSIZE];
    int nbytes, size;
    char remoteIP[INET6_ADDRSTRLEN];
    time_t t = time(NULL);
    struct tm tm = * localtime( & t);
    int yes = 1; // for setsockopt() SO_REUSEADDR, below
    int i, j, k, m, rv, usercheck = 0;
    FILE * fp;
    char fname[50];
    char empty[10];
    char password[10];
    char privmsg_user[MAXUSERSIZE];
    char privmsg_msg[MAXDATASIZE];
    int privmsg_ucheck = 0;
    int privmsg_fdi;
    int stop = 0;
    int userprivate[USERS] = {
        0
    };
    int userconnect[USERS] = {
        0
    };
    struct addrinfo hints, * ai, * p; // check to see if file exists as well as open and append
    sprintf(fname, "%d-%d-%d-Logfile.txt", tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday);
    fp = fopen(fname, "a");
    fprintf(fp, "%d:%d selectserver: logfile\n", tm.tm_hour, tm.tm_min);
    fclose(fp); // clear the master, temp and buffers;
    FD_ZERO( & master);
    FD_ZERO( & read_fds);
    memset(buf, 0, MAXDATASIZE * sizeof(char));
    memset(message, 0, MAXSIZE * sizeof(char));
    for (k = 0; k < USERS; k++) {
        username[k][0] = '\0';
    } // get us a socket and bind it
    memset( & hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;
    if ((rv = getaddrinfo(NULL, PORT, & hints, & ai)) != 0) {
        fprintf(stderr, "selectserver: %s\n", gai_strerror(rv));
        exit(1);
    }
    for (p = ai; p != NULL; p = p - > ai_next) {
        listener = socket(p - > ai_family, p - > ai_socktype, p - > ai_protocol);
        if (listener < 0) {
            continue;
        } // lose the pesky "address already in use" error message
        setsockopt(listener, SOL_SOCKET, SO_REUSEADDR, & yes, sizeof(int));
        if (bind(listener, p - > ai_addr, p - > ai_addrlen) < 0) {
            close(listener);
            continue;
        }
        break;
    } // if we got here, it means we didn't get bound
    if (p == NULL) {
        fprintf(stderr, "selectserver: failed to bind\n");
        exit(2);
    }
    freeaddrinfo(ai); // all done with this // listen
    if (listen(listener, 10) == -1) {
        perror("listen");
        exit(3);
    } // add the listener to the master set
    FD_SET(listener, & master); // keep track of the biggest file descriptor
    fdmax = listener; // so far, it's this one // main loop
    for (;;) {
        read_fds = master; // copy it
        if (select(fdmax + 1, & read_fds, NULL, NULL, NULL) == -1) {
            perror("select");
            exit(4);
        } // run through the existing connections looking for data to read
        for (i = 0; i <= fdmax; i++) {
            if (FD_ISSET(i, & read_fds)) { // we got one!!
                if (i == listener) { // handle new connections
                    addrlen = sizeof remoteaddr;
                    newfd = accept(listener, (struct sockaddr * ) & remoteaddr, & addrlen);
                    if (newfd == -1) {
                        perror("accept");
                    } else {
                        FD_SET(newfd, & master); // add to master set
                        if (newfd > fdmax) { // keep track of the max
                            fdmax = newfd;
                        }
                        printf("%d:%d selectserver: new connection from %s on "
                            "socket %d\n", tm.tm_hour, tm.tm_min, inet_ntop(remoteaddr.ss_family, get_in_addr((struct sockaddr * ) & remoteaddr), remoteIP, INET6_ADDRSTRLEN), newfd);
                        fp = fopen(fname, "a");
                        fprintf(fp, "%d:%d selectserver: new connection from %s on "
                            "socket %d", tm.tm_hour, tm.tm_min, inet_ntop(remoteaddr.ss_family, get_in_addr((struct sockaddr * ) & remoteaddr), remoteIP, INET6_ADDRSTRLEN), newfd);
                        fclose(fp);
                    }
                } else { // handle data from a client
                    if ((nbytes = recv(i, buf, sizeof buf, 0)) <= 0) { // got error or connection closed by client
                        if (nbytes == 0) { // connection closed
                            printf("%d:%d selectserver: socket %d hung up\n", tm.tm_hour, tm.tm_min, i); // user has left the server message
                            fp = fopen(fname, "a");
                            fprintf(fp, "%d:%d selectserver: socket %d hung up", tm.tm_hour, tm.tm_min, i);
                            fclose(fp);
                            memset(message, 0, MAXSIZE * sizeof(char));
                            sprintf(message, "%d:%d ", tm.tm_hour, tm.tm_min);;
                            strcat(message, username[i]);
                            strcat(message, " has left the server\n");
                            size = strlen(message);
                            fp = fopen(fname, "a");
                            fprintf(fp, "\n%d:%d selectserver: %s", tm.tm_hour, tm.tm_min, message);
                            fclose(fp);
                            userprivate[i] = 0;
                            userconnect[i] = 0; // Send message to everyone
                            for (j = 0; j <= fdmax; j++) { // send to everyone!
                                if (FD_ISSET(j, & master)) { // except the listener and ourselves
                                    if (j != listener && j != i) {
                                        if (send(j, message, strlen(message), 0) == -1) {
                                            perror("send");
                                        }
                                        fflush(stdout);
                                    }
                                }
                            }
                            username[i][0] = '\0'; // username removed
                        } else {
                            perror("recv");
                        }
                        close(i); // bye
                        FD_CLR(i, & master); // remove from master set
                    } else { // check username
                        if (strncmp(buf, "USR", 3) == 0) {
                            m = 0;
                            for (k = 3; k <= strlen(buf); k++) {
                                usertemp[m++] = buf[k];
                            }
                            for (k = 0; k <= fdmax; k++) {
                                if (strcmp(usertemp, username[k]) == 0) {
                                    send(i, "Username error", 14, 0);
                                    fp = fopen(fname, "a");
                                    fprintf(fp, "\n%d:%d selectserver: username error on socket %d", tm.tm_hour, tm.tm_min, i);
                                    fclose(fp);
                                    usercheck = 1;
                                }
                            }
                            if (usercheck == 0) {
                                strcat(username[i], usertemp);
                                strcpy(buf, "I have joined the chat!\n");
                                fp = fopen(fname, "a");
                                fprintf(fp, "\n%d:%d selectserver: %s on socket %d has joined the server\n", tm.tm_hour, tm.tm_min, username[i], i);
                                fclose(fp);
                                memset(usertemp, 0, MAXUSERSIZE * sizeof(char));
                                userconnect[i] = 1;
                            } else {
                                usercheck = 0;
                            }
                        } // private message
                        if (strncmp(buf, "/message", 8) == 0) { // grab username
                            sscanf(buf, "%s %s", empty, privmsg_user);
                            k = strlen(empty) + strlen(privmsg_user) + 2;
                            buf[nbytes] = '\0';
                            strcpy(privmsg_msg, buf + k); //check user
                            privmsg_fdi = -1;
                            for (k = 0; k <= fdmax; k++) {
                                if (strcmp(privmsg_user, username[k]) == 0) {
                                    privmsg_fdi = k;
                                }
                            }
                            if (privmsg_fdi != -1) {
                                memset(message, 0, MAXSIZE * sizeof(char));
                                sprintf(message, "%d:%d ", tm.tm_hour, tm.tm_min);
                                strcat(message, username[i]);
                                strcat(message, " (whisper) ");
                                privmsg_msg[strlen(privmsg_msg)] = '\0';
                                strcat(message, privmsg_msg);
                                if (send(privmsg_fdi, message, strlen(message), 0) == -1) {
                                    perror("send");
                                }
                                fflush(stdout);
                            } else {
                                memset(message, 0, MAXSIZE * sizeof(char));
                                sprintf(message, "%d:%d ", tm.tm_hour, tm.tm_min);
                                strcat(message, "User does not exist, try again\n");
                                if (send(i, message, strlen(message), 0) == -1) {
                                    perror("send");
                                }
                                fflush(stdout);
                            }
                        } else if (strncmp(buf, "/private", 8) == 0) {
                            sscanf(buf, "%s %s", empty, password);
                            if (strcmp(password, "password") == 0) {
                                userprivate[i] = 1;
                                memset(message, 0, MAXSIZE * sizeof(char));
                                sprintf(message, "%d:%d You have entered private chat!\n", tm.tm_hour, tm.tm_min);
                                if (send(i, message, strlen(message), 0) == -1) {
                                    perror("send");
                                }
                                fflush(stdout);
                            } else {
                                memset(message, 0, MAXSIZE * sizeof(char));
                                sprintf(message, "%d:%d Password was incorrect\n", tm.tm_hour, tm.tm_min);
                                if (send(i, message, strlen(message), 0) == -1) {
                                    perror("send");
                                }
                                fflush(stdout);
                            }
                        } else if (strncmp(buf, "/public", 7) == 0) {
                            userprivate[i] = 0;
                            memset(message, 0, MAXSIZE * sizeof(char));
                            sprintf(message, "%d:%d You have entered public chat!\n", tm.tm_hour, tm.tm_min);
                            if (send(i, message, strlen(message), 0) == -1) {
                                perror("send");
                            }
                            fflush(stdout);
                        } else if (strncmp(buf, "/online", 7) == 0) {
                            for (k = 0; k <= fdmax; k++) {
                                if (k == 0) {
                                    memset(message, 0, MAXSIZE * sizeof(char));
                                    sprintf(message, "%d:%d Currently online: ", tm.tm_hour, tm.tm_min);
                                }
                                if (userprivate[i] == userprivate[k] && userconnect[k] == 1) {
                                    strcat(message, username[k]);
                                    strcat(message, " ");
                                }
                            }
                            strcat(message, "\n");
                            if (send(i, message, strlen(message), 0) == -1) {
                                perror("send");
                            }
                            fflush(stdout);
                        } else {
                            memset(message, 0, MAXSIZE * sizeof(char));
                            sprintf(message, "%d:%d ", tm.tm_hour, tm.tm_min);
                            if (userprivate[i] == 1) {
                                strcat(message, "private: ");
                            } else {
                                strcat(message, "public: ");
                            }
                            strcat(message, username[i]);
                            strcat(message, " >> ");
                            buf[nbytes] = '\0';
                            strcat(message, buf);
                            fp = fopen(fname, "a");
                            fprintf(fp, "%d:%d client: %s", tm.tm_hour, tm.tm_min, message);
                            fclose(fp); // we got some data from a client
                            for (j = 0; j <= fdmax; j++) { // send to everyone!
                                if (FD_ISSET(j, & master)) { // except the listener and ourselves
                                    if (j != listener && j != i && userprivate[i] == userprivate[j]) {
                                        if (send(j, message, strlen(message), 0) == -1) {
                                            perror("send");
                                        }
                                        fflush(stdout);
                                    }
                                }
                            }
                        }
                    }
                } // END handle data from client
            } // END got new incoming connection
        } // END looping through file descriptors
    } // END for(;;)--and you thought it would never end!
    return 0;
}
```

B. Expanded Client Source

```c
/*
--- DIRECTIONS ---
client.c
cc -pthread -o client.out client.c
./client.out hostname username

--- DOCUMENTATION ---
Joseph Simeon
6305ENG: Advanced Computer Systems
Nathan Campus
Griffith University

--- CREDIT ---
I would like to give credit to Beej's Guide to Network Programming as the client.c "a stream socket client demo" as the base template for this program.
ADRSS: https://beej.us/guide/bgnet/html/multi/clientserver.html#simpleclient

*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <pthread.h>
#include <time.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#define PORT "9034" // the port client will be connecting to
#define MAXDATASIZE 100 // max number of bytes we can get at once
#define MAXUSERSIZE 10
#define MAXSIZE 150 // thread data structure
```

```c

typedef struct {
    char * prompt;
    int socket;
}
thread_data; // thread receive function
void * receiveMessage(void * threadData) {
        int sockfd, rv;
        char message[MAXSIZE], * prompt;
        thread_data * localData = (thread_data * ) threadData;
        sockfd = localData - > socket;
        prompt = localData - > prompt;
        memset(message, 0, MAXDATASIZE * sizeof(message[0])); // print the received message
        while (1) {
            rv = recvfrom(sockfd, message, MAXSIZE, 0, NULL, NULL);
            if (rv == -1) {
                perror("receive");
                break;
            } else if (rv == 0) {
                printf("Connection error: user disconnected\n");
                break;
            } else {
                if (strcmp(message, "Username error") == 0) {
                    printf("Username has already been taken\n");
                    close(sockfd);
                    exit(0);
                }
                printf("Server%code%nbsp;%s", message); //printf("%s >> ", prompt); //fflush(stdout);

                memset(message, 0, MAXDATASIZE * sizeof(message[0]));
            }
        }
    } // get sockaddr, IPv4 or IPv6:
void * get_in_addr(struct sockaddr * sa) {
    if (sa - > sa_family == AF_INET) {
        return &(((struct sockaddr_in * ) sa) - > sin_addr);
    }
    return &(((struct sockaddr_in6 * ) sa) - > sin6_addr);
}
int main(int argc, char * argv[]) {
    int sockfd, numbytes;
    char message[MAXDATASIZE], * user;
    struct addrinfo hints, * servinfo, * p;
    int rv, userlen, DATASIZE;
    char s[INET6_ADDRSTRLEN];
    pthread_t thread;
    thread_data data;
    time_t t = time(NULL);
    struct tm tm = * localtime( & t);
    if (argc != 3) {
        fprintf(stderr, "usage: ./client.out hostname username\n");
        exit(1);
    } // check username length
    userlen = strlen(argv[2]);
    if (userlen > 10 || userlen < 6) {
        fprintf(stderr, "usage: ./client.out hostname username\n");
        fprintf(stderr, "Username error: must be between 6-10 characters\n");
        exit(1);
    } else {
        user = malloc(userlen * sizeof(char));
        memset(user, 0, userlen * sizeof(user[0]));
        strcpy(user, argv[2]);
    }
    memset( & hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    if ((rv = getaddrinfo(argv[1], PORT, & hints, & servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
        return 1;
    } // loop through all the results and connect to the first we can
    for (p = servinfo; p != NULL; p = p - > ai_next) {
        if ((sockfd = socket(p - > ai_family, p - > ai_socktype, p - > ai_protocol)) == -1) {
            perror("client: socket");
            continue;
        }
        if (connect(sockfd, p - > ai_addr, p - > ai_addrlen) == -1) {
            perror("client: connect");
            close(sockfd);
            continue;
        }
        break;
    }
    if (p == NULL) {
        fprintf(stderr, "client: failed to connect\n");
        return 2;
    }
    inet_ntop(p - > ai_family, get_in_addr((struct sockaddr * ) p - > ai_addr), s, sizeof s);
    printf("client: connecting to %s\n", s);
    freeaddrinfo(servinfo); // all done with this structure // receive thread
    printf("client: paralleling receive function\n");
    data.prompt = user;
    data.socket = sockfd;
    pthread_create( & thread, NULL, receiveMessage, (void * ) & data);
    printf("client: receive function is parallelised\n"); // send loop
    memset(message, 0, MAXDATASIZE * sizeof(char));
    strcpy(message, "USR");
    strcat(message, user); //strcat(message, "I am ready to chat!\n");
    if ((send(sockfd, message, sizeof(message), 0)) == -1) {
        perror("send");
        close(sockfd);
        exit(0);
    } //printf("\n%s", message);
    printf("client: ready to receive user messages\n"); // entry message
    printf("\n\n\nYou have chosen the username: %s and it will appear like '%s >> Hello'\n", user, user);
    printf("To exit server type '/quit' and press enter\n");
    printf("To send a whisper type '/message username <your message goes here>'\n");
    printf("To enter private chat type '/private password', you will be prompted if successful\n");
    printf("To exit private chat type '/public'\n");
    printf("To view all online users type '/online'\n");
    printf("\n\t%d/%d/%d\n", tm.tm_mday, tm.tm_mon + 1, tm.tm_year + 1900);
    while (fgets(message, MAXDATASIZE, stdin) != NULL) {
        if (strncmp(message, "/quit", 5) == 0) {
            printf("Closing conneciton...\n");
            exit(0);
        } //printf("client%code%nbsp;%s\n", message);
        send(sockfd, message, strlen(message), 0);
        memset(message, 0, MAXDATASIZE * sizeof(message[0]));
    }
    close(sockfd);
    pthread_exit(NULL);
    free(user);
    return 0;
}
```

C. Expanded Client-server Screen-dump



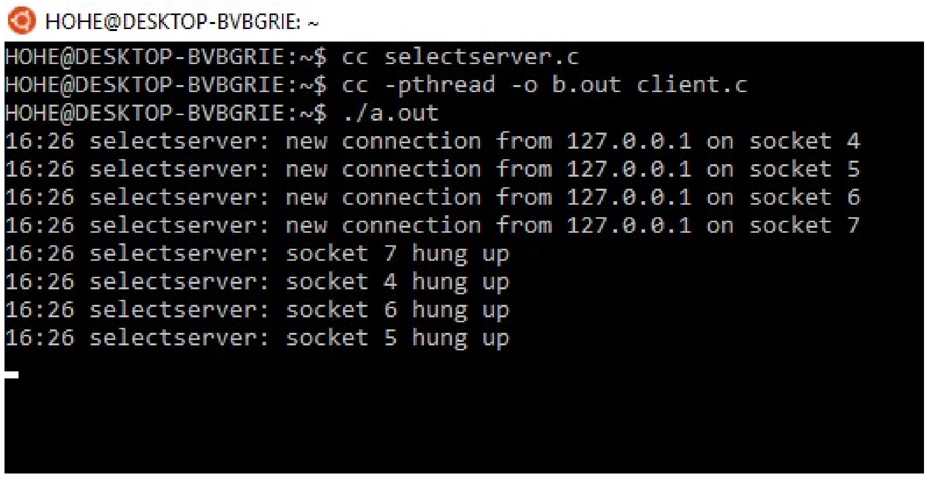
```
HOHE@DESKTOP-BVBGRIE: ~
To exit server type '/quit' and press enter
To send a whisper type '/message username <your message goes here>'
To enter private chat type '/private password', you will be prompted if successful
To exit private chat type '/public'
To view all online users type '/online'

        16/10/2018
Server$ 16:26 public: RedRuby >> I have joined the chat!
Server$ 16:26 public: Orange >> I have joined the chat!
Server$ 16:26 public: KaBluey >> I have joined the chat!
Hi
Server$ 16:26 public: RedRuby >> Hello!
Server$ 16:26 public: Orange >> WAZZUP
Server$ 16:26 public: Orange >> KaBluey left!
Server$ 16:26 public: RedRuby >> Nah probs in private chat
Server$ 16:26 KaBluey has left the server
Server$ 16:26 public: RedRuby >> now he left...
HAH1
HAH!*
```

```
HOHE@DESKTOP-BVBGRIE: ~
Server$ 16:26 public: Orange >> I have joined the chat!
Server$ 16:26 public: KaBluey >> I have joined the chat!
Server$ 16:26 public: Lavender >> Hi
Hello!
Server$ 16:26 public: Orange >> WAZZUP
Server$ 16:26 public: Orange >> KaBluey left!
Nah probs in private chat
/message KaBluey come back...
Server$ 16:26 KaBluey (whisper) No
/private password
Server$ 16:26 You have entered private chat!
c'mon mate don't be like that..
Server$ 16:26 private: KaBluey >> ...
Server$ 16:26 KaBluey has left the server
/public
Server$ 16:26 You have entered public chat!
now he left...
Server$ 16:26 public: Lavender >> HAH1
Server$ 16:26 public: Lavender >> HAH!*
```

```
HOHE@DESKTOP-BVBGRIE: ~
To exit server type '/quit' and press enter
To send a whisper type '/message username <your message goes here>'
To enter private chat type '/private password', you will be prompted if successful
To exit private chat type '/public'
To view all online users type '/online'

        16/10/2018
Server$ 16:26 public: KaBluey >> I have joined the chat!
Server$ 16:26 public: Lavender >> Hi
Server$ 16:26 public: RedRuby >> Hello!
WAZZUP
/online
Server$ 16:26 Currently online: Lavender RedRuby Orange
KaBluey left!
Server$ 16:26 public: RedRuby >> Nah probs in private chat
Server$ 16:26 KaBluey has left the server
Server$ 16:26 public: RedRuby >> now he left...
Server$ 16:26 public: Lavender >> HAH1
Server$ 16:26 public: Lavender >> HAH!*
```

```
HOHE@DESKTOP-BVBGRIE: ~
To view all online users type '/online'

        16/10/2018
Server$ 16:26 public: Lavender >> Hi
Server$ 16:26 public: RedRuby >> Hello!
Server$ 16:26 public: Orange >> WAZZUP
/online
Server$ 16:26 Currently online: Lavender RedRuby Orange KaBluey
/private password
Server$ 16:26 You have entered private chat!
/online
Server$ 16:26 Currently online: KaBluey
thank god..
Server$ 16:26 RedRuby (whisper) come back...
/message RedRuby No
Server$ 16:26 private: RedRuby >> c'mon mate don't be like that..
...
/quit
Closing conneciton...
HOHE@DESKTOP-BVBGRIE:~$
```

Figure 3. Screen dump of client conversation



```
HOHE@DESKTOP-BVBGRIE: ~
HOHE@DESKTOP-BVBGRIE:~$ cc selectserver.c
HOHE@DESKTOP-BVBGRIE:~$ cc -pthread -o b.out client.c
HOHE@DESKTOP-BVBGRIE:~$ ./a.out
16:26 selectserver: new connection from 127.0.0.1 on socket 4
16:26 selectserver: new connection from 127.0.0.1 on socket 5
16:26 selectserver: new connection from 127.0.0.1 on socket 6
16:26 selectserver: new connection from 127.0.0.1 on socket 7
16:26 selectserver: socket 7 hung up
16:26 selectserver: socket 4 hung up
16:26 selectserver: socket 6 hung up
16:26 selectserver: socket 5 hung up
```

Figure 4. Screen dump of server during operation

D. Expanded Client-server Logfile

```
16:26 selectserver: logfile
16:26 selectserver: new connection from 127.0.0.1 on socket 4
16:26 selectserver: Lavender on socket 4 has joined the server
16:26 client: 16:26 public: Lavender >> I have joined the chat!
16:26 selectserver: new connection from 127.0.0.1 on socket 5
16:26 selectserver: RedRuby on socket 5 has joined the server
16:26 client: 16:26 public: RedRuby >> I have joined the chat!
16:26 selectserver: new connection from 127.0.0.1 on socket 6
16:26 selectserver: Orange on socket 6 has joined the server
16:26 client: 16:26 public: Orange >> I have joined the chat!
16:26 selectserver: new connection from 127.0.0.1 on socket 7
16:26 selectserver: KaBluey on socket 7 has joined the server
16:26 client: 16:26 public: KaBluey >> I have joined the chat!
16:26 client: 16:26 public: Lavender >> Hi
16:26 client: 16:26 public: RedRuby >> Hello!
16:26 client: 16:26 public: Orange >> WAZZUP
16:26 client: 16:26 private: KaBluey >> thank god..
16:26 client: 16:26 public: Orange >> KaBluey left!
```

```
16:26 client: 16:26 public: RedRuby >> Nah probs in private chat
16:26 client: 16:26 private: RedRuby >> c 'mon mate don't be like that..
16:26 client: 16:26 private: KaBluey >> ...16: 26 selectserver: socket 7 hung up
16:26 selectserver: 16:26 KaBluey has left the server
16:26 client: 16:26 public: RedRuby >> now he left...
16:26 client: 16:26 public: Lavender >> HAH1
16:26 client: 16:26 public: Lavender >> HAH! *
16:26 selectserver: socket 4 hung up
16:26 selectserver: 16:26 Lavender has left the server
16:26 selectserver: socket 6 hung up
16:26 selectserver: 16:26 Orange has left the server
16:26 selectserver: socket 5 hung up
16:26 selectserver: 16:26 RedRuby has left the server
```

VII. References

[1] Hall B. "Beej's Guide for Network Programming", Accessed: 11/9/2018 [Online]: https://beej.us/guide/bgnet/html/single/bgnet.html