Project_2 Report

Group 18 E24076849 翁麒庭
E24076239 彭恩宇
E24072073 王俊傑
XX1092023 梁師睿

Create a Chat Room in Linux system (Multi-user chatroom)

Part A: Code section

Server code: (*server.c*)

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <pthread.h>

int sockfd;                                                     // Server socket
int fds[20];                                                    // Socketfd of clients,at most 10 (fds[0]~fds[9])
int size =20;                                                   // Max client number set to 10 (allow at most 10 users)
char* IP = "127.0.0.1";                                         // Connection IP number (return to this computer)
short PORT = 10222;                                             // Connection port
typedef struct sockaddr SA;                                     // Structure typd define, easy for coding

void init()                                                     //Initialization code section of the server
{
        sockfd = socket(PF_INET, SOCK_STREAM, 0);               // Create the socket (use stream type; TCP type; no special protocol)
        if (sockfd == -1)                                       // Socket create failed
        {
                perror("Socket creation failed");               // Print out error message
                exit(-1);                                       // Exit the code and return nonzero number
        }

        struct sockaddr_in addr;                                // Construct a socket address struct
        addr.sin_family = PF_INET;                              // We use TCP, af variable = PF_INET
        addr.sin_port = htons(PORT);                            // Function for changing host to network type
        addr.sin_addr.s_addr = inet_addr(IP);                   // Transfer decimal address to binary address
        if (bind(sockfd, (SA*)&addr, sizeof(addr)) == -1)       // Binding function, check if binding success
        {                                                       // Variable: socket name; socket address; length of address
                perror("Bnding failed");                        // Print out error message
                exit(-1);                                       // Exit the code and return nonzero number
        }
        if (listen(sockfd, 20) == -1)                           // Set the port to listening
        {
                perror("Listen deamon setting failed");         // Failed, print out error message
                exit(-1);                                       // Exit the code and return nonzero number
        }
}
```

Figure 1. Server code and comment (initialization)

```c
void SendMsgToAll(char* msg)                                    // Send message to all users in the chat room log
{
        int i;
        for (i = 0; i < size; i++)                              // To all users
        {
                if (fds[i] != 0)                                // ENable fds number (users available number)
                {
                        send(fds[i], msg, strlen(msg), 0);      // Send message in buffer to all users
                        printf("Message sent to: %d\n",fds[i]); // Print log
                }
        }
}

void detect()                                                   // Erads the terminal input (for shut down)
{
        pthread_t id;                                           // New thread for servicing
        void* service(void*);                                   // New service job
        pthread_create(&id, 0, service, 0);                     // Add the recieve job to the thread
        while (1)
        {
                char buf[100] = {};
                scanf("%s",buf);
                if (strcmp(buf, "exit") == 0)                   // User type in leaving keyword (exit)
                {
                        printf("Server shut down\n");           // Print out leaving message
                        break;                                  // Stop service
                }
        }
        close(sockfd);
}
```

Figure 2. Server code and comment (message send and detect)

```c
void* service_thread(void* p)                                          // Create new thread for new client
{
        int fd = *(int*)p;                                             // Create a new thread id number
        printf("(Pthread id = %d)\n",fd);                              // Print out log

        while(1)                                                       // Thread running
        {
                char buf[100] = {};                                    // Chat message buffer
                if (recv(fd, buf, sizeof(buf), 0) <= 0)                // Recieve leaving message from client number fd
                {
                        int i;
                        for (i = 0; i < size; i++)                     // Set the fds number to 0 (return the resource)
                        {
                                if (fd == fds[i])                      // Find the fd number in vector fds
                                {
                                        fds[i] = 0;                    // Return the thread resuorce
                                        break;                         // Break the for loop
                                }
                        }
                        printf("User : fd = %d leaves the chat room\n",fd);   // Broadcast the leaving message to all users
                        pthread_exit((void*)i);                        // Exit the thread
                }
                SendMsgToAll(buf);                                     // Send the message to all the users in the chat room and write log
        }
}
```

Figure 3. Server code and comment (service thread)

```c
void* service(void* p)                                                 // Service section, main section
{
        printf("Server on\n");                                         // Print out status when successfully connect the port
        while(1)
        {
                struct sockaddr_in fromaddr;                           // Socket address datatype of port address
                socklen_t len = sizeof(fromaddr);                      // Length of the address
                int fd = accept(sockfd, (SA*)&fromaddr, &len);         // Accept new connection from client
                if (fd == -1)
                {
                        printf("Can't connect to client..\n");         // Error from connecting to client
                        continue;                                      // Try again
                }

                int i = 0;
                for (i = 0; i < size; i++)
                {
                        if (fds[i] == 0)
                        {
                                fds[i] = fd;                           // Record the socket number of the client
                                printf("fd = %d, ",fd);                // Print the fd number of client
                                pthread_t tid;                         // Create new thread to service the client
                                pthread_create(&tid, 0, service_thread, &fd);
                                break;
                        }
                        if (size == i)                                 // The chat room is now full
                        {
                                char* str = "The room is full!";       // Full message
                                send(fd,str,strlen(str),0);            // Send message to the new client
                                close(fd);                             // Stop servicing the new client
                        }
                }
        }
}

int main()                                                             // Main server.c code section
{
        init();                                                        // Initialize server
        detect();                                                      // Provide service after initialize successfully
        return(0);
}
```

Figure 4. Server code and comment (service and main section)

Client code：(*client.c*)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>

int sockfd;                                                     // Client soclet number
char* IP = "127.0.0.1";                                         // Server IP number
short PORT = 10222;                                             // Server port number
typedef struct sockaddr SA;
char name[30];                                                  // String of the client name

void init()                                                     // Initialization sevtion of client code
{
        sockfd = socket(PF_INET, SOCK_STREAM, 0);               // Create the socket
        struct sockaddr_in addr;                                // Construct a socket address struct
        addr.sin_family = PF_INET;                              // We use TCP, af variable = PF_INET
        addr.sin_port = htons(PORT);                            // Function for changing host to network type
        addr.sin_addr.s_addr = inet_addr(IP);                   // Transfer decimal address to binary address
        if (connect(sockfd, (SA*)&addr, sizeof(addr)) == -1)    // Try connection
        {
                perror("Can't conect to the server\n");         // Connection fail
                exit(-1);                                       // Exit the code and return nonzero number
        }
        printf("Connect success!\n");                           // Connection success
}

void* recv_thread(void* p)                                      // Message reciving method
{
        while(1)                                                // Keep reciving
        {
                char buf[100] = {};                             // Message buffer
                if (recv(sockfd, buf, sizeof(buf), 0) <= 0)     // If didn't recieve any message (buffer is empty)
                {
                        return (0);                             // Skip
                }
                printf("%s\n", buf);                            // Else, print the message
        }
}
```

Figure 5. Client code and comment (initialization and receive message thread)

```c
void start()                                                    // Service of client, main part
{
        pthread_t id;                                           // New thread for recieving message
        void* recv_thread(void*);                               // New recieve job
        pthread_create(&id, 0, recv_thread, 0);                 // Add the recieve job to the thread
        char buf2[100] = {};                                    // Maeeage buffer
        sprintf(buf2, "User: %s enters the chat room", name);   // Show enter message
        send(sockfd, buf2, strlen(buf2), 0);                    // Send the same message to every other user in the room

        while(1)
        {
                char buf[100] = {};                             // Message buffer
                scanf("%s", buf);                               // User type in message
                char msg[133] = {};                             // Send message buffer
                sprintf(msg, "%s: %s\n", name, buf);            // Message that will show on everyone screen
                send(sockfd, msg,strlen(msg), 0);               // Send to server
                if (strcmp(buf, "bye") == 0)                    // User type in leaving keyword (bye)
                {
                        memset(buf2, 0, sizeof(buf2));          // Set message to leave message
                        sprintf(buf2, "User: %s leaves the room\n", name);  // Print out leaving message
                        send(sockfd, buf2, strlen(buf2), 0);    // Send to all other user in the room
                        break;                                  // Stop service
                }
        }
        close(sockfd);                                          // Close the socket
}

int main()                                                      // Main client.c code section
{
        init();                                                 // Initialize client
        printf("Please enter your name： ");                    // Ask the name of user
        scanf("%s", name);                                      // Read the type in of the user
        start();                                                // Start the service of client
        return 0;                                               // Stop servlce when user leaves
}
```

Figure 6. Client code and comment (main section)

Part B: Result

Figure 7. Server running screenshot

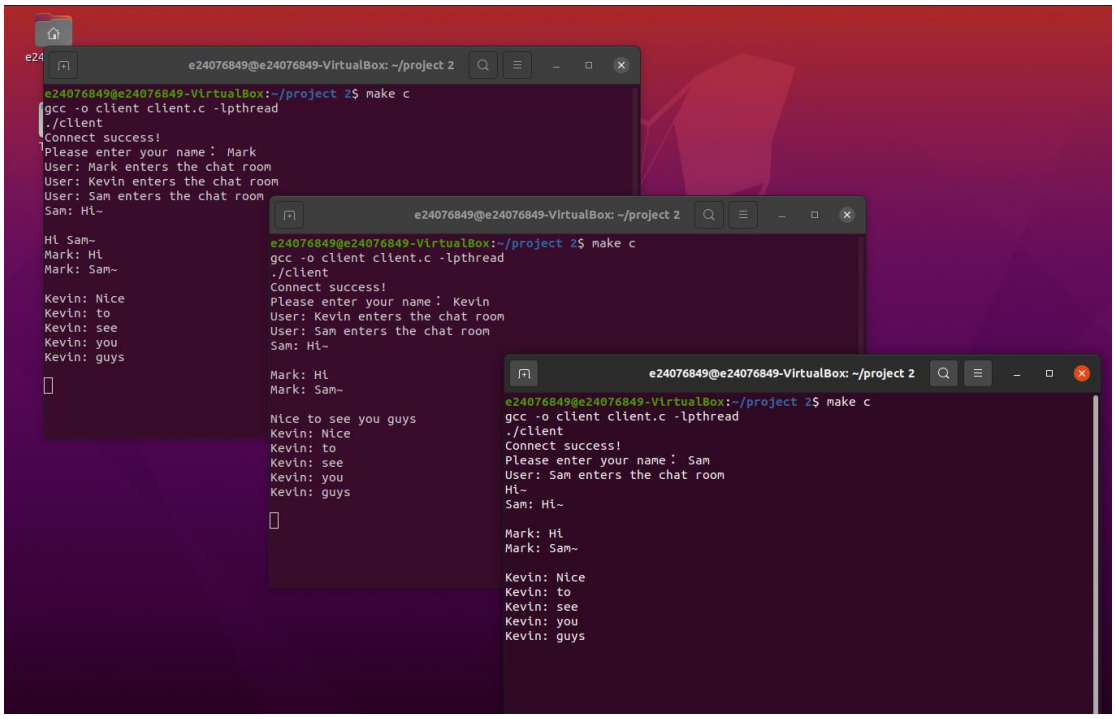

Figure 8. Three users chatting screenshot

Part C: Explanation of code

Connection:

We create sockets to link server and clients. IP number is set to 127.0.0.1 to connect back to our computer, port number is set to 10222 a random number larger than 1024.

Send and receive:

In both server and client, we create threads to send and receive messages.

On server side, server will receive message from every client, and broadcast (send) to all the clients (include the message sender).

On client side, when user types message, client will send a message to the server. When the client receives the broadcast from server, it will show up on the screen.

Flow chart:

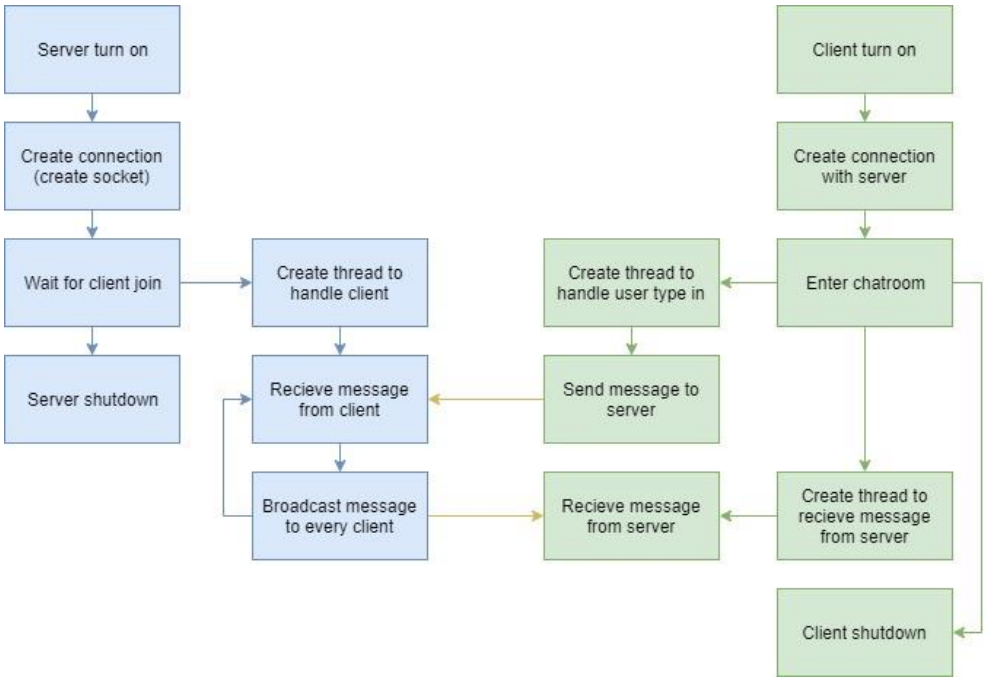

Figure 9. Flow chart of client and server

Structure chart:

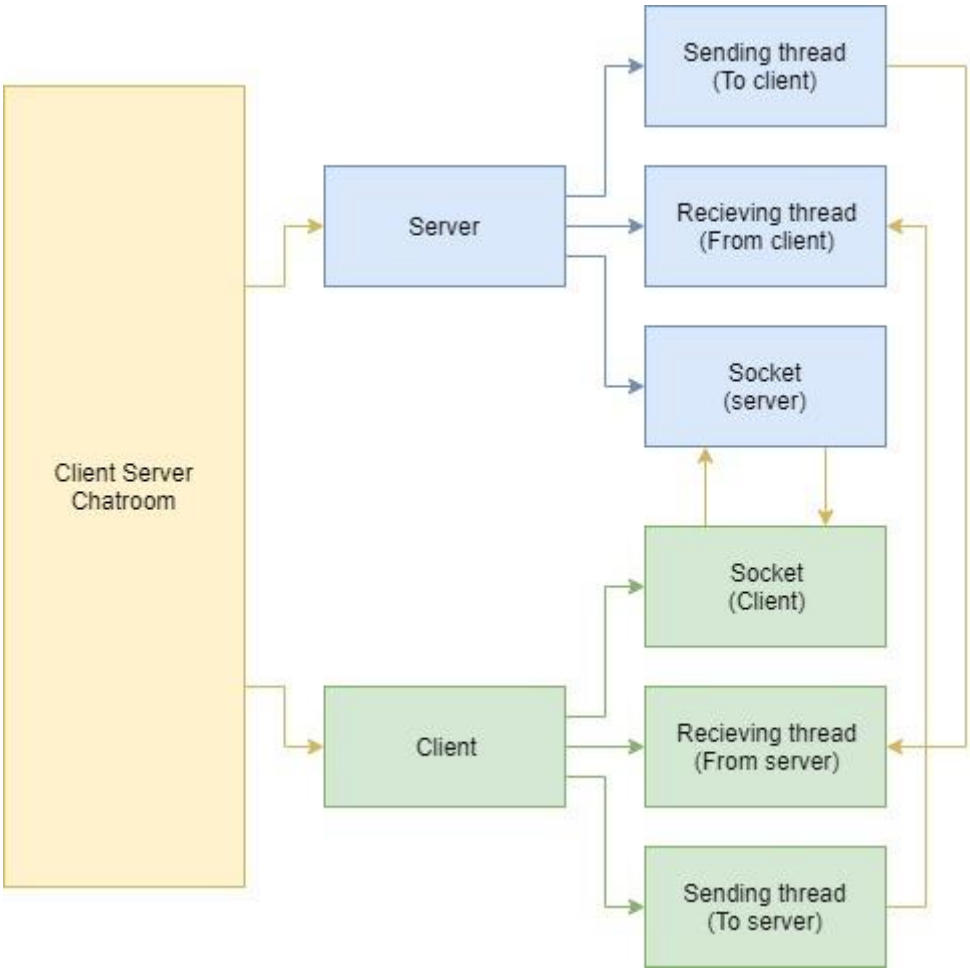

Figure 10. Structure chart of chatroom

Since we found that send function already have synchronization (if one thread wants to send to a socket and there exists another thread sending to the same socket, it will wait until the previous sending ends), thus no two clients can send message to the same socket (server).

As the send function setting, if one thread ends its message sending, other thread can send its message, thus the whole thing is in progress.

We only allow at most 20 clients to connect to the server at the same time, thus the time every thread occupies the critical section will not be long. Besides, the threads will send only when user finish typing message in, which is much longer than sending the message through socket, thus the synchronization is bounded waiting.

Part D: READ ME file

We provide a makefile to easily execute the code correctly

Command: *make s* → compile server code and run.

make c → compile client code and run.

When the server is on, it will keep writing log on the terminal, include new client connected (created a new thread to handle the client message sending) and message send to clients.

Type in *exit* can shut down the server.

When a client is created, it will automatically try to connect to the server if there exists one. If connected successfully, system will ask user to type in a name to recognize in chatting.

After enter the chat room, the user can type in every message freely. But the system distinguishes messages by the space, thus on sentence will be separated to words and show on the terminal.

If the user wishes to leave the chatroom, just type in command *bye.*

Part E. Division of work

E24076849 翁麒庭:

Researching, Discussion, Coding (Server), Debugging

E24076239 彭恩宇:

Researching, Discussion, Coding (Client), Debugging

E24072073 王俊傑:

Researching, Discussion, Coding (Server), Report writing

XX1092023 梁師睿:

Researching, Discussion, Coding (Client), Report writing

Part F. References

https://www.cnblogs.com/Newdawn/p/5509687.html

https://blog.csdn.net/stpeace/article/details/18279593

https://blog.csdn.net/weixin_41413441/article/details/80156696?tdsourcetag=s_pcqq_aiomsg

https://www3.physnet.uni-hamburg.de/physnet/Tru64-Unix/HTML/APS33DTE/DOCU_010.HTM

https://www.systutorials.com/docs/linux/man/3-sem_open/

https://www.kshuang.xyz/doku.php/programming:c:pthread_and_semaphore

https://gist.github.com/junfenglx/7412986

https://www.itread01.com/content/1543651384.html