به نام ایزد یکتا

گزارش چهارم آزمایشگاه سیستم‌عامل

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

استاد: مهندس قاسمی

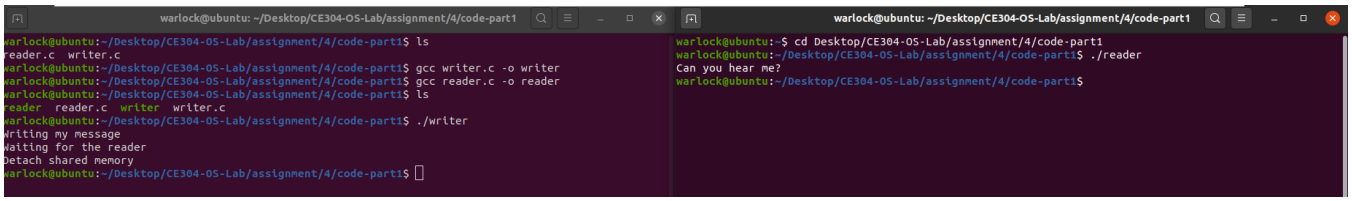
تهیه کننده: بردیا اردکانیان

۹۸۳۱۰۷۲

آزمایش اول:

برای این بخش یک *reader* و یک *writer* داریم. هر دو با استفاده از دستورات دستورکار یک حافظه مشترک می‌سازند. ابتدا فایل *writer* را اجرا می‌کنیم؛ *writer* پیامی در حاظفه مشترک می‌نویسد و منتظر می‌ماند تا *reader* پیام را بخواند. ما *reader* را در ترمینالی جدا اجرا می‌کنیم تا پیام مورد نظر را از حافظه مشترک بخواند.

خروجی:



عکس 1-1

کد *reader*:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define SHMSIZE 50

int main()
{
    int shmid;
    key_t key;
    key = 3232;

    if ((shmid = shmget(key, SHMSIZE, 0666)) < 0) {
        perror("shmget");
        exit(1);
    }

    char *shm;
    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
    }

    printf("%s\n", shm);
```

```
    *shm = '~';

    exit(0);
}
```

کد *writer*:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define SHMSIZE 50

int main()
{
    int shmid;
    key_t key;
    key = 3232;

    if ((shmid = shmget(key, SHMSIZE, IPC_CREAT | 0666)) < 0) {
        perror("shmget");
        exit(1);
    }

    char *shm;
    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
    }

    printf("Writing my message\n");
    sprintf(shm, "Can you hear me?");


    printf("Waiting for the reader\n");
    while (*shm != '~')
        sleep(1);

    printf("Detach shared memory\n");
    if (shmdt(shm) == -1) {
        perror("shmdt");
        exit(1);
    }

    if(-1 == (shmctl(shmid, IPC_RMID, NULL)))
    {
        perror("shmctl");
```

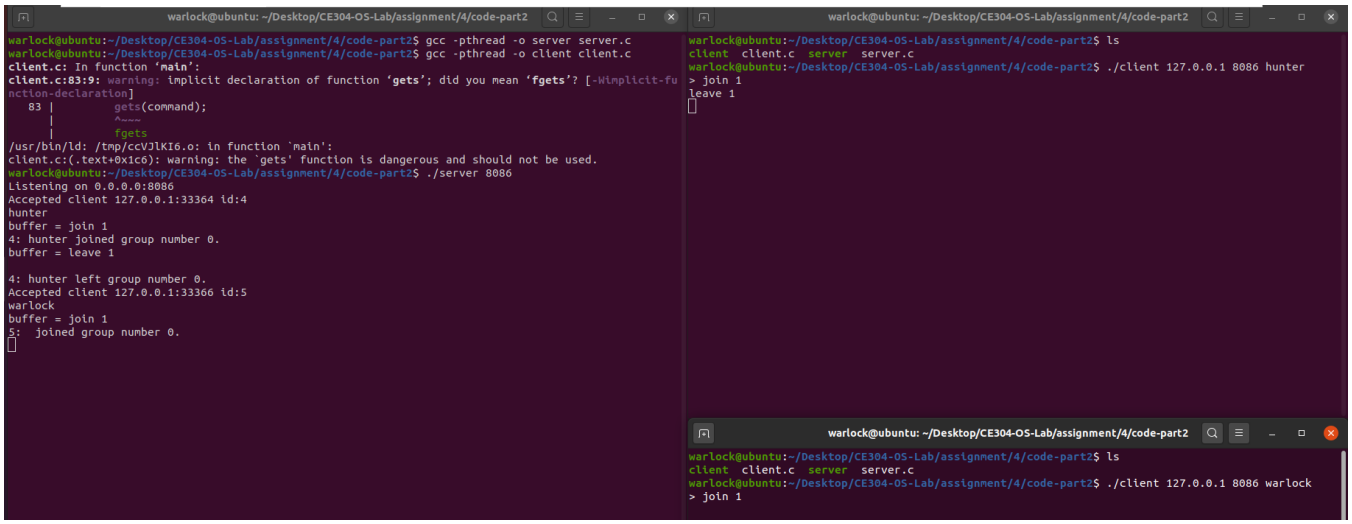
```
        exit(1);
    }

    exit(0);
}
```

آزمایش دوم:

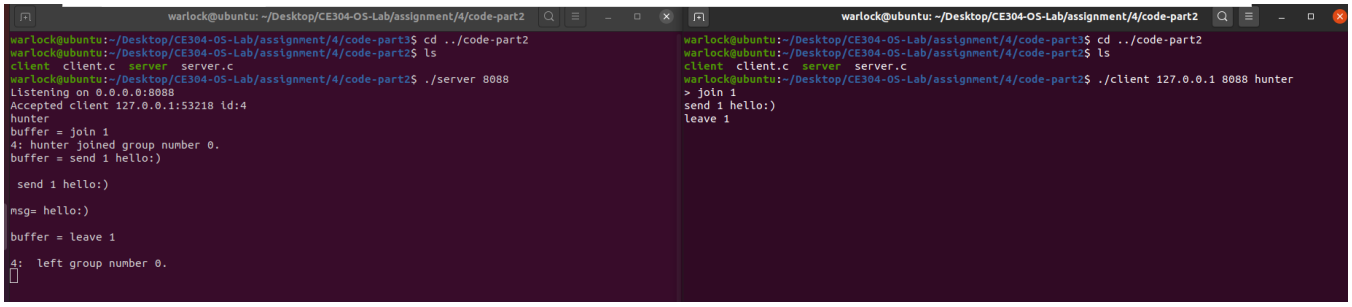
در این بخش، یک کلاینت و یک سرور داریم. کلاینت به صورت مداوم دستوراتی که در دستور کار گفته شده را می‌تواند بفرستد و در سرور پس از اینکه کلاینت را قبول کرد یک ترد جدا برای آن می‌سازد و تابع مختص آن را اجرا می‌کند. به این صورت که هر بار چک می‌کند آن کلاینت چه دستوری را وارد کرده و عملیات مربوط به آن را اجرا می‌کند.

از آنجایی که می‌توانیم تعداد زیادی کلاینت داشته باشیم، یک لیتکد لیست از کاربران را در سرور نگه داری می‌کنیم و به همین منظور یک سری توابع برای جستوجو کاربر و اضافه و کم کردن کاربران از لیست داریم. هر کاربر نیز ساختارش به صورت یک استراکت است که در آن شماره سوکت و نام کاربر نگهداری می‌شود.

همچنین یک لیست از گروه‌های مختلف نیز داریم که می‌توانیم روی آن پیمایش کنیم. حال به عنوان مثال برای اینکه یک پیام را به همه کاربران داخل یک گروه بفرستیم کافیست و تمامی شماره سوکت‌های ان کاربران آن پیام را ارسال کنیم. برای خروج از چت‌روم هم از دستور ***quit*** می‌توان استفاده کرد و اگر می‌خواهیم از گروه خاصی خارج شویم از دستور ***leave*** استفاده می‌کنیم.



عکس 2-1



عکس 2-2

کد server:

```
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
// Thread library
#include <pthread.h>
#define MAX_USERS 100
#define MAX_GROUPS 10
#define MAXDATALEN 256 // max size of messages to be sent
#define MAXGROUP 10    // max number of groups

/* Client structure */
typedef struct{
      int port;
      char username[32];
} User;

void insert_list(int port, char *username, User *list, int *tail); /*inserting new
client */
int search_list(int port, User *list, int tail);
void delete_list(int port, User *list, int *tail);
void delete_all(User *list, int *tail);
void display_list(const User *list, int tail); /*list all clients connected*/
int next_space(char *str);

char username[10];
User users[MAX_USERS] = {0};
int user_tail = 0;
User groups[MAX_GROUPS][MAX_USERS] = {0};
int group_tail[MAX_USERS] = {0};

void *client_handler(void *arguments)
{
    char buffer[MAXDATALEN], uname[10]; /* buffer for string the server sends */
    User *args = arguments;
    int my_port = args->port; /*socket variable passed as arg*/
    char *strp;
    char *msg = (char *)malloc(MAXDATALEN);
    int msglen;
    int x;
    strcpy(uname, args->username);
    //printf("username is: ");
```

```
    //printf("%s\n", uname);
    int valread;
    while (1)
    {

        bzero(buffer, 256);
        valread = read(my_port, buffer, 256);
        printf("buffer = %s\n",buffer);

        /* Client quits */
        if (strstr(buffer, "quit"))
        {
            printf("** %d: %s left chat. Deleting from lists. **\n\n", my_port,
uname);

            delete_list(my_port, users, &user_tail);
            for (int i = 0; i < MAXGROUP; i++)
            {
                delete_list(my_port, groups[i], &group_tail[i]);
            }

            display_list(users, user_tail);

            close(my_port);
            free(msg);
        }
        else if (strstr(buffer, "join"))
        {
            //printf("wants to join!");
            char *group_id_str = malloc(sizeof(MAXDATALEN));
            strcpy(group_id_str, buffer + 6);
            int group_id = atoi(group_id_str);
            printf("%d: %s joined group number %d.\n", my_port, uname, group_id);

            insert_list(my_port, uname, groups[group_id], &group_tail[group_id]);
        }
        else if (strstr(buffer, "leave"))
        {
            char *group_id_str = malloc(sizeof(MAXDATALEN));
            strcpy(group_id_str, buffer + 7);
            int group_id = atoi(group_id_str);
            printf("%d: %s left group number %d.\n", my_port, uname, group_id);

            delete_list(my_port, groups[group_id], &group_tail[group_id]);
        }
        else if (strstr(buffer, "send"))
        {
            int space_pos = next_space(buffer + 6);
            char *group_id_str = malloc(sizeof(MAXDATALEN));
            strncpy(group_id_str, buffer + 6, space_pos);
```

```
            int group_id = atoi(group_id_str);

            if (search_list(my_port, groups[group_id], group_tail[group_id]) == -1)
            {
                continue;
            }

            printf("%s %s\n", uname, buffer);
            strcpy(msg, uname);
            x = strlen(msg);
            strp = msg;
            strp += x;
            strcat(strp, buffer + 7 + space_pos);
            msglen = strlen(msg);

            printf("msg= %s\n",msg);
            for (int i = 0; i < group_tail[group_id]; i++)
            {
                if (groups[group_id][i].port != my_port)
                    send(groups[group_id][i].port, msg, msglen, 0);
            }


            bzero(msg, MAXDATALEN);
        }
        //display_list(users, user_tail);
    }
}

int main(int argc, char const *argv[])
{
    pthread_t thr;
    int server_fd;
    server_fd = socket(AF_INET, SOCK_STREAM, 0); // TODO: What is this do
    if (server_fd == 0)
    {
        perror("Socket faild");
        exit(EXIT_FAILURE);
    }

    struct sockaddr_in address;
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(atoi(argv[1]));
    const int addrlen = sizeof(address);

    if (bind(server_fd, (struct sockaddr *)&address, addrlen) < 0)
    {
        perror("Bind failed");
        exit(EXIT_FAILURE);
```

```
    }

    if (listen(server_fd, 3) < 0)
    {
        perror("Listen faild");
        exit(EXIT_FAILURE);
    }

    printf("Listening on %s:%d\n", inet_ntoa(address.sin_addr),
ntohs(address.sin_port));


    // Accepting client
    int valread;
    char buffer[1024] = {0};
    while (1)
    {
        int client_socket;
        if ((client_socket = accept(server_fd, (struct sockaddr *)&address,
(socklen_t *)&addrlen)) < 0)
        {
            perror("Accept faild");
            exit(EXIT_FAILURE);
        }

        printf("Accepted client %s:%d id:%d\n", inet_ntoa(address.sin_addr),
ntohs(address.sin_port), client_socket);

        valread = read(client_socket, buffer, sizeof(buffer));
        if (valread < 0)
        {
            perror("Empty read");
            exit(EXIT_FAILURE);
        }
         /* getting username */

        strcpy(username, buffer);
        printf("%s\n",username);

        insert_list(client_socket, username, users, &user_tail);

        // Using thread to handle the client
        pthread_t thread_id;
        pthread_create(&thread_id, NULL, client_handler, (void *)&client_socket);

        User args;
        args.port = client_socket;
        strcpy(args.username, username);

        //printf("%d\n", args.port);
```

```
        //printf("%s\n", args.username);
        pthread_create(&thr, NULL, client_handler, (void *)&args);
        pthread_detach(thr);
    }

    return 0;
}

void insert_list(int port, char *username, User *list, int *tail)
{
    if (search_list(port, list, *tail) != -1)
    {
        return;
    }
    User *temp;
    temp = malloc(sizeof(User));
    if (temp == NULL)
        printf("Out of space!");
    temp->port = port;
    strcpy(temp->username, username);
    list[(*tail)++] = *temp;
}

int search_list(int port, User *list, int tail)
{
    for (int i = 0; i < tail; i++)
    {
        if (list[i].port == port)
            return i;
    }
    return -1;
}

void delete_list(int port, User *list, int *tail)
{
    int ptr = search_list(port, list, *tail);
    if (ptr == -1)
    {
        return;
    }

    for (int i = ptr; i < *tail - 1; i++)
    {
        list[i] = list[i + 1];
    }
    (*tail)--;
}

void display_list(const User *list, int tail)
{
```

```
    printf("Current online users:\n");
    if (tail == 0)
    {
        printf("No one is online\n");
        return;
    }

    for (int i = 0; i < tail; i++)
    {
        printf("%d: %s\t", list[i].port, list[i].username);
    }
    printf("\n\n");
}

void delete_all(User *list, int *tail)
{
    *tail = 0;
}

int next_space(char *str)
{
    int i = 0;
    while (str[i] != '\0')
    {
        if (str[i] == ' ')
        {
            return i;
        }
        i++;
    }
    return -1;
}
```

کد client:

```
/*
Client is a simple user interface to get the data
and send it to the server by socket.
Every client can do the following commands:
    1. Start
    2. Ping
    3. Stop
*/
// socket libraries

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
```

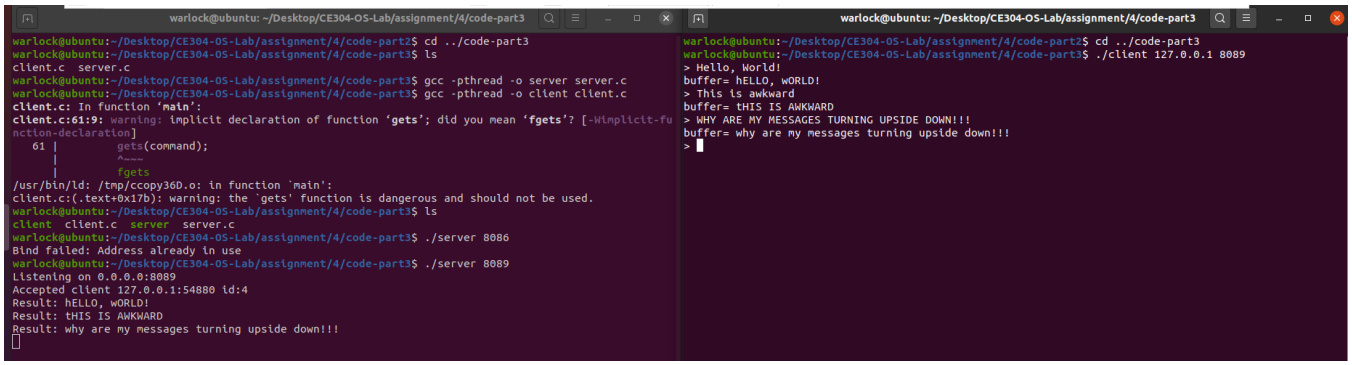
```c
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <signal.h>

#define MAXDATALEN 256

int *quit();
void *chat_write(int);
void *chat_read(int);

int n;                          /*variables for socket*/
struct sockaddr_in serv_addr; /* structure to hold server's address */
char buffer[MAXDATALEN];
char buf[10];

int main(int argc, char const *argv[])
{
    pthread_t thr1, thr2;
    int sock = 0;
    struct sockaddr_in serv_addr;

    // TODO: How does socket create?
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("Socket creation error");
        exit(EXIT_FAILURE);
    }

    // TODO: What is memory cell?
    memset(&serv_addr, '0', sizeof(serv_addr));

    // TODO: What is address family
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(atoi(argv[2]));

    // TODO: Why do we convert IPv4 and IPv6 to binary
    if (inet_pton(AF_INET, argv[1], &serv_addr.sin_addr) <= 0)
    {
        perror("Invalid address");
        exit(EXIT_FAILURE);
    }

    // Connecting to the server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }
```

```
    // User commands
    int valread;
    char buffer[1024] = {0};
    char command[50];
    char name[20];


    strcpy(name, argv[3]);
    send(sock, name, strlen(name), 0);

    //printf("name=%s\n",name);

    while (1)
    {
        printf("> ");
        gets(command);

        send(sock, command, strlen(command), 0);

        if (strcmp(command, "stop") == 0)
        {
            printf("Disconnected\n");
            break;
        }

        pthread_create(&thr2, NULL, (void *)chat_write, (void *) (intptr_t) sock);
//thread for writing
        pthread_create(&thr1, NULL, (void *)chat_read, (void *) (intptr_t)
sock);  //thread for reading

        pthread_join(thr2, NULL);
        pthread_join(thr1, NULL);

        /*valread = read(sock, buffer, sizeof(buffer));

        if (valread < 0)
        {
            perror("Reading failed");
            exit(EXIT_FAILURE);
        }*/

    }

    return 0;
}

void *chat_read(int sockfd)
{
    signal(SIGINT,(void *)quit);
```

```
    while (1)
    {
        n = recv(sockfd, buffer, MAXDATALEN - 1, 0);
        if (n == 0)
        {
            printf("\n==== SERVER HAS BEEN SHUTDOWN ====\n");
            exit(0);
        }

        if (n > 0)
        {
            printf("-> %s", buffer);
            bzero(buffer, MAXDATALEN);
        }
    }
}

void *chat_write(int sockfd)
{
    while (1)
    {
        // printf("%s", buf);
        fgets(buffer, MAXDATALEN - 1, stdin);

        if (strlen(buffer) - 1 > sizeof(buffer))
        {
            printf("buffer size full\t enter within %ld characters\n",
sizeof(buffer));
            bzero(buffer, MAXDATALEN);
            //__fpurge(stdin);
        }

        n = send(sockfd, buffer, strlen(buffer), 0);

        if (strncmp(buffer, "/quit", 5) == 0)
            exit(0);

        bzero(buffer, MAXDATALEN);
    }
}

int *quit()
{
    printf("\nType '/quit' TO EXIT\n");
    return 0;
}
```

آزمایش سوم:

در این بخش مانند بخش قبلی یک کلاینت و سرور داریم. کلاینت پیامی را روی یک پایپ‌لاین می‌فرستد و سرور با گرفتن پیام آن را ابتدا با توجه به خواسته سوال، اصلاح می‌کند و حروف بزرگ را کوچک و کوچک را بزرگ می‌کند. سپس در یک پایپ‌لاین دیگر برای کلاینت می‌فرستد.



عکس 3-1

کد server:

```
/*
Using sockets to connect to our server.
Our server which will accept clients and give response to them.
Our main input command are:
    1. Start [name]
    2. Ping
    3. Stop
Server responses:
    1. Init user
    2. Pong [status]
    3. Release client
*/

// Socket libraries
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
#include <pthread.h>

int main(int argc, char const *argv[])
```

```
{
    // TODO: What is a socket file descriptor
    int server_fd;
    server_fd = socket(AF_INET, SOCK_STREAM, 0); // TODO: What is this do
    if (server_fd == 0)
    {
        perror("Socket faild");
        exit(EXIT_FAILURE);
    }

    // TODO: What the fuck are these??
    struct sockaddr_in address;
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(atoi(argv[1]));
    const int addrlen = sizeof(address);

    // TODO: What is binding??
    if (bind(server_fd, (struct sockaddr *)&address, addrlen) < 0)
    {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    // TODO: What is backlog?
    if (listen(server_fd, 3) < 0)
    {
        perror("Listen faild");
        exit(EXIT_FAILURE);
    }

    printf("Listening on %s:%d\n", inet_ntoa(address.sin_addr),
ntohs(address.sin_port));

    // Accepting client

    int client_socket;
    if ((client_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t
*)&addrlen)) < 0)
    {
        perror("Accept faild");
        exit(EXIT_FAILURE);
    }

    printf("Accepted client %s:%d id:%d\n", inet_ntoa(address.sin_addr),
ntohs(address.sin_port), client_socket);


    int valread;
    char buffer[1024] = {0};
```

```
    char response[1024] = {0};


    while (1)
    {
        valread = read(client_socket, buffer, sizeof(buffer));
        if (valread < 0)
        {
            perror("Empty read");
            exit(EXIT_FAILURE);
        }

        buffer[valread] = '\0';


        if (strcmp(buffer, "stop") == 0)
        {
            printf("Client %d: disconnected\n", client_socket);
            break;
        }

        for (int i = 0; buffer[i]!='\0'; i++) {
            if(buffer[i] >= 'a' && buffer[i] <= 'z') {
            buffer[i] = buffer[i] - 32;
            continue;
            }
            if(buffer[i] >= 'A' && buffer[i] <= 'Z') {
            buffer[i] = buffer[i] + 32;
            continue;
            }
        }

        printf("Result: %s\n", buffer);
        for (int i = 0; buffer[i]!='\0'; i++) {
            response[i] = buffer[i];
        }
        send(client_socket, response, sizeof(response), 0);
        fflush(stdout);
        response[0] = '\0';
    }


    return 0;
}
```

کد client:

```
/*
Client is a simple user interface to get the data
and send it to the server by socket.
```

```
Every client can do the following commands:
    1. Start
    2. Ping
    3. Stop
*/
// socket libraries
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <unistd.h>


int main(int argc, char const *argv[])
{
    int sock = 0;
    struct sockaddr_in serv_addr;

    // TODO: How does socket create?
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("Socket creation error");
        exit(EXIT_FAILURE);
    }

    // TODO: What is memory cell?
    memset(&serv_addr, '0', sizeof(serv_addr));

    // TODO: What is address family
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(atoi(argv[2]));

    // TODO: Why do we convert IPv4 and IPv6 to binary
    if (inet_pton(AF_INET, argv[1], &serv_addr.sin_addr) <= 0)
    {
        perror("Invalid address");
        exit(EXIT_FAILURE);
    }

    // Connecting to the server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        perror("Connection failed");
        exit(EXIT_FAILURE);
    }

    // User commands
    int valread;
```

```c
    char buffer[1024] = {0};
    char command[50];

    while (1)
    {
        printf("> ");
        gets(command);

        send(sock, command, strlen(command), 0);

        if (strcmp(command, "stop") == 0)
        {
            printf("Disconnected\n");
            break;
        }

        valread = read(sock, buffer, sizeof(buffer));

        if (valread < 0)
        {
            perror("Reading failed");
            exit(EXIT_FAILURE);
        }

        printf("buffer= %s\n", buffer);
    }

    return 0;
}
```