

بخش ۲:

در سمت server برنامه به درخواست های فرستاده شده توسط client جواب می دهد و به ازای هر client پذیرفته شده، یک thread باز می کند و پاسخ مناسب می دهد.

به بررسی کد client می پردازیم.

```
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>

#define PORT 8080
#define MAX_BUFFER 4096

int check(int val , char* message);
void receiver(int sock);
void sender(int sock , pid_t pid);

int main(int argc, char const *argv[]) {
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    check(sock = socket(AF_INET, SOCK_STREAM, 0), "\n Socket creation error \n");
    memset(&serv_addr, '0', sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    if(inet_pton(AF_INET, argv[2], &serv_addr.sin_addr) <= 0)
    {
        printf("\nInvalid address/ Address not supported \n");
        exit(EXIT_FAILURE);
    }
    check(connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)), "\nConnection Failed \n");
    check(send(sock, argv[3], strlen(argv[3]), 0), "send");
    pid_t pid = fork();
    if(pid > 0)
        while (1)
            sender(sock, pid);
    else
        while (1)
            receiver(sock);

    return 0;
}
```

توضیح توابع بخش function prototype:

وظیفه تابع `check` بررسی صحت عملکرد `send` و `read` می‌باشد و در صورت بروز خطا، با تابع `perror` خطای مناسب را باز می‌گرداند.

```
int check(int val , char* message){
    if(val < 0){
        perror(message);
        exit(EXIT_FAILURE);
    }
    return val;
}
```

تابع `receiver` با دریافت شماره سوکت و تعریف آرایه‌ای از کاراکترها به عنوان بافر و پاس دادن آن‌ها به تابع `read`، پیام ارسال شده را دریافت می‌کند.

```
void receiver(int sock){
    char buffer[MAX_BUFFER] = {0};
    while(check(read(sock, buffer, MAX_BUFFER), "send")>0){
        printf("%s\n",buffer);
        memset(buffer , 0 , MAX_BUFFER);
    }
}
```

تابع `sender` با دریافت شماره سوکت و شناسه‌ی پردازنده و همچنین تعریف فضایی برای بافر، پیامی که از کاربر دریافت می‌کند را به شماره سوکت مذکور ارسال می‌کند و در نهایت با دریافت دستور `kill` پردازنده را از بین می‌برد.

```
void sender(int sock , pid_t pid){
    char buffer[MAX_BUFFER] = {0};
    fgets(buffer , sizeof(buffer) , stdin);
    check(send(sock, buffer, strlen(buffer), 0), "send");
    if(strcmp(buffer , "quit\n")==0){
        kill(pid , SIGKILL);
        exit(EXIT_SUCCESS);
    }
}
```

توضیح `main`:

ابتدا سوکتی را به صورت `TCP` به وجود می‌آوریم و در صورت ناموفق بودن ایجاد آن، خطای مناسب را باز می‌گردانیم. سپس با کمک تابع `memset` فضایی را برای آدرس سرور در حافظه جدا می‌کنیم. سپس

درستی آدرس سرور را بررسی می‌کنیم و در صورت معتبر نبودن آن، خطای مناسب را برمی‌گردانیم. به سرور متصل می‌شویم و در صورت عدم اتصال، خطای مناسب را باز می‌گردانیم.

```
int main(int argc, char const *argv[]) {
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    check(sock = socket(AF_INET, SOCK_STREAM, 0), "\n Socket creation error \n");
    memset(&serv_addr, '0', sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    if(inet_pton(AF_INET, argv[2], &serv_addr.sin_addr) <= 0)
    {
        printf("\nInvalid address/ Address not supported \n");
        exit(EXIT_FAILURE);
    }
    check(connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)), "\nConnection Failed \n");
    check(send(sock, argv[3], strlen(argv[3]), 0), "send");
}
```

برای ارسال پیام، با `fork()` پردازهی فرزندی را به وجود می‌آوریم و فرآیند پدر را مامور دریافت پیام قرار می‌دهیم.

```
pid_t pid = fork();
if(pid > 0)
    while (1)
        sender(sock, pid);
else
    while (1)
        receiver(sock);
return 0;
}
```

به بررسی کد `server` می‌پردازیم. برای ذخیره `client` ها، یک استراکت به نام `node` تعریف می‌کنیم که داخل آن `file descriptor` مربوط به سوکت آن کلاینت ذخیره میشود.

```

#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 <pthread.h>

#define PORT 8080
#define BUFFER_SIZE 4096
#define MAX_GROUPS 100

struct node{
    int client;
    struct node* next;
};

char** split(char* str,int split_count);
struct node* create_node(int client_id);
struct node* remove_node(struct node* head ,int client);
struct node* add_node(struct node* head , int client);
void send_message_to_group(struct node* head , int sender ,char* message );
void* handle_client(void* pclient_socket);
int create_server(struct sockaddr_in* paddress);
void check(int val , char* message);
struct sockaddr_in * get_address(char* ip_address);
void init(char * ip_address,struct sockaddr_in* address);
void create_service(int socket);
void join_to_group(char** message , int client_socket , char*name);
void send_to_group(char** message , int client_socket , char* name);
void leave_from_group(char** message , int client_socket , char * name);
void quit_from_server(int client_socket);
void handle_bad_request(int client_socket);

struct node** groups;

pthread_t threads[100000];

```

توضیح توابع بخش function prototype:

به کمک این تابع رشته ی ارسالی از طرف کاربر را با اسپیس جداسازی می کنیم و دستور و پیام مورد نظر کاربر را استخراج می کنیم.

```

//split string
char** split(char* str ,int split_count){
    int length = strlen(str);

```

```

char ** splited;
splited = (char**) malloc(sizeof(char*)*3);
for(int i = 0 ; i < split_count;i++){
    splited[i] = (char *)malloc(sizeof(char)*length*2);
}
int k = 0;
int p = 0;
for (int i = 0; i < length; i++) {
    if ((str[i] == ' ' || i == 0)&& k < split_count-1) {
        int f= 0;
        int j= i > 0 ? i+1: 0;
        for(;str[j] != ' ';&& str[j]!='\n';j++,f++)
            splited[k][f] = str[j];
        i=j-1;
        k++;
    }else if (k == split_count-1){
        splited[split_count-1][p] = str[i];
        p++;
    }
}
return splited;
}

```

اعضای هر گروه در یک لینک لیست ذخیره می‌شوند و تاجایی که می‌توانیم کاربر نگه داریم.

```

/*
using array of linked list for groups
users of a group are saved into a linked list
*/
struct node* create_node(int client_id){
    struct node* n_node = (struct node*)malloc(sizeof(struct node));
    n_node->client = client_id;
    return n_node;
}

```

به کمک تابع زیر، به لینک لیست گروه مربوطه، شماره سوکت کلاینت را اضافه می‌کنیم.

```

struct node* add_node(struct node* head , int client){
    struct node* current = head;
    while (current){
        if(current->client == client){
            return head;
        }
        current = current->next;
    }
}

```

```

    struct node* nod = create_node(client);
    nod->next = head;
    head = nod;
    return head;
}

```

به کمک این تابع ما کلاینت مورد نظر را از لیست اعضای یک گروه حذف می‌کنیم.

```

struct node* remove_node(struct node* head ,int client){
    struct node* current = head;
    struct node* prev = NULL;
    if (current != NULL && current->client == client){
        head = current->next;
        free(current);
        return head;
    }
    while (current)
    {
        if(current->client == client){
            prev->next = current->next;
            free(current);
            return head;
        }
        prev = current;
        current = current->next;
    }
    return head;
}

```

به کمک این تابع با پیمایش روی لینک لیست یک گروه، یک پیام را به اعضای گروه ارسال می‌کنیم.

```

void send_message_to_group(struct node* head , int sender ,char* message ){
    struct node* tmp = head;
    int is_member = 0;
    while (tmp){
        if(tmp->client == sender)
            is_member = 1;
        tmp = tmp->next;
    }
}

```

```

if(!is_member){
    char * to_client = "you are not a member of this group";
    check(send(sender, to_client, strlen(to_client), 0),"send");
    return;
}
tmp = head;
while (tmp){
    if (!(tmp->client == sender))
        check(send(tmp->client, message, strlen(message), 0),"send");
    tmp = tmp->next;
}
}

```

این تابع وظیفه ی اجرای دستور ارسالی از طرف کاربر را دارد، بدین صورت که اگر دستور کاربر join بود، تابع join_to_group را فراخوانی می کنیم. و به همین ترتیب توابع مورد نیاز را فراخوانی می کنیم. دقت داریم که از آنجا که اعمالی که کاربران انجام می دهند، منابع مشترک بین ترد ها را تغییر می دهد، برای جلوگیری از وقوع خطا، از pthread_mutex_lock و pthread_mutex_unlock در قبل و بعد دستورات استفاده می کنیم.

```

//client service
void* handle_client(void* pclient_socket){
    int client_socket = *((int*) pclient_socket);
    char name[50] = {0};
    check(read(client_socket, name, 50),"read");
    printf("client name : %s\n" , name);
    while (1)
    {
        char buffer[BUFFER_SIZE] = {0};
        check(read(client_socket, buffer, BUFFER_SIZE),"read");
        char ** message = split(buffer,3);
        printf("(s = %d) %s\n", client_socket, buffer);
        pthread_mutex_lock(&lock);
        if(strcmp(message[0],"join")==0){
            join_to_group(message , client_socket, name);
        }else if (strcmp(message[0],"send")==0){
            send_to_group(message , client_socket , name);
        }else if (strcmp(message[0],"leave")==0){
            leave_from_group(message , client_socket , name);
        }else if (strcmp(message[0],"quit")==0)
        {

```

```

        quit_from_server(client_socket);
    }else{
        handle_bad_request(client_socket);
    }
    pthread_mutex_unlock(&lock);
}
return NULL;
}

```

به کمک این تابع ما آدرسی را برای سرور اختصاص می‌دهیم و سرور را در حالت `listen` قرار می‌دهیم.

```

//create file descriptor , bind socket and listen
int create_server(struct sockaddr_in* paddress){
    struct sockaddr_in address = *paddress;
    int server_fd=0;
    if ((server_fd =socket(AF_INET, SOCK_STREAM, 0)),"socket failed" );
    int addrlen = sizeof(address);
    check(bind(server_fd, (struct sockaddr *)&address, sizeof(address)) , "bind")
;
    check(listen(server_fd, 1000) ,"listen");
    return server_fd;
}

```

این تابع نیز مانند تابع چک ارور در کلاینت عمل می‌کند و خطای مناسب را برمی‌گرداند.

```

//convention for errors
void check(int val , char* message){
    if(val < 0){
        perror(message);
        exit(EXIT_FAILURE);
    }
}

```

در این بخش، پراپرتی‌های آدرس، مقدار دهی می‌شوند.

```

//address format
struct sockaddr_in * get_address(char* ip_address){
    struct sockaddr_in address;
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = inet_addr(ip_address);
    address.sin_port = htons(PORT);
    struct sockaddr_in * paddress = &address;
    return paddress;
}

```


در این تابع مقدار دهی های اولیه مربوط به `pthread_mutex_lock` انجام میشود و آدرس آی پی و `host name` چاپ می شوند و پیامی برای اینکه آدرس و پورت سرور را متوجه شویم، چاپ می شود.

```
//first logs
void init(char* ip_address , struct sockaddr_in* address){
    if (pthread_mutex_init(&lock, NULL) != 0) {
        printf("\n mutex init has failed\n");
        exit(EXIT_FAILURE);
    }

    printf("%s\n" , ip_address);
    char hostname[1024];
    hostname[1023] = '\0';
    gethostname(hostname, 1023);

    printf("host name : %s\n" , hostname);

    printf("Listen on %s:%d\n", inet_ntoa(address->sin_addr), ntohs(address->sin_port));
}
```

در این تابع با توجه به سوکت داده شده، تردی به کلاینت اختصاص می دهیم و آنرا در لیست ترد ها قرار می دهیم.

```
//create thread for client
void create_service(int socket){
    int * pclient_socket = (int*)malloc(sizeof(int));
    *pclient_socket = socket;
    pthread_t thread;
    pthread_create(&thread , NULL , handle_client , pclient_socket);
    threads[socket] = thread;
}
```

تابع زیر وظیفه اضافه کردن کلاینت به گروه را دارد.

```
//joind two group
void join_to_group(char** message , int client_socket,char * name){
    int group_id;
    char* to_client = (char *)malloc(sizeof(char)*BUFFER_SIZE);
    sscanf(message[1] , "%d" , &group_id);
    if(group_id >MAX_GROUPS){
        to_client = "not a valid group id";
        check(send(client_socket, to_client, strlen(to_client), 0), "send");
    }else
    {
        groups[group_id] = add_node(groups[group_id] , client_socket);
        printf("group %d members: \n" , group_id);
        struct node * curr = groups[group_id];
        while (curr)
        {
            printf("member %d\n",curr->client);
            curr= curr->next;
        }

        sprintf(to_client , "you have added to %d" , group_id);
        check(send(client_socket, to_client, strlen(to_client), 0),"send");
        char to_client[BUFFER_SIZE];
        sprintf(to_client , "%s joined the group %d" , name , group_id);
        send_message_to_group(groups[group_id] , client_socket , to_client);
    }
}
```

این تابع برای ارسال پیام کاربر به گروه مورد نظر می باشد.

```
//send message to group
void send_to_group(char** message , int client_socket,char * name){
    int group_id;
    char* to_client;
    sscanf(message[1] , "%d" , &group_id);
    if(group_id >MAX_GROUPS){
        to_client = "not a valid group id";
        check(send(client_socket, to_client, strlen(to_client), 0),"send");
    }else{
        printf("***%s\n" , message[2]);
        char to_client[BUFFER_SIZE];
        sprintf(to_client , "%s : %s" , name , message[2]);
        send_message_to_group(groups[group_id] , client_socket , to_client);
    }
}
```

```
}
```

این تابع برای خروج کاربر از گروه مورد نظر کاربر می‌باشد.

```
//leave from group
void leave_from_group(char** message , int client_socket , char * name){
    int group_id;
    char* to_client;
    sscanf(message[1] , "%d" , &group_id);
    if(group_id >MAX_GROUPS){
        to_client = "not a valid group id";
        check(send(client_socket, to_client, strlen(to_client), 0),"send");
    }else{
        char to_client[BUFFER_SIZE];
        sprintf(to_client , "%s left the group %d" , name , group_id);
        send_message_to_group(groups[group_id] , client_socket , to_client);
        remove_node(groups[group_id] , client_socket);
    }
}
```

این تابع وظیفه‌ی حذف یک کاربر از لینک لیست کاربران مربوط به یک گروه و ارسال پیام حذف موفق به کلاینت را داراست، همچنین ترد مربوط به کلاینت حذف می‌شود.

```
//handle quit from server
void quit_from_server(int client_socket){
    for(int i=0 ;i < MAX_GROUPS;i++){
        groups[i] = remove_node(groups[i] , client_socket);
    }
    char * message = "successfully removed";
    check(send(client_socket , message , strlen(message) , 0) , "send");
    close(client_socket);
    pthread_cancel(threads[client_socket]);
}
```

با کمک این تابع، درخواست‌های غیر معتبر کاربران با پیام **bad request** پاسخ داده می‌شود.

```
//handling bad requests and irrelevant requests
void handle_bad_request(int client_socket){
    int group_id;
    char* to_client;
    to_client = "bad request";
    check(send(client_socket, to_client, strlen(to_client), 0),"send");
}
```

حال به بررسی تابع main می‌پردازیم.

ابتدا آی پی آدرس مورد نظر که توسط آرگومان به سرور داده می‌شود، به عنوان ip_address ذخیره می‌شود و لینک لیستی برای گروه‌ها ساخته می‌شود و با فراخوانی تابع init مقداردهی اولیه صورت می‌گیرد و با فراخوانی به create_server، سرور با آی پی مورد نظر ساخته می‌شود.

سپس با فراخوانی تابع select، به برنامه این اجازه را می‌دهیم که چندین file descriptor پایش بشوند و این تابع تا زمانی که یک file descriptor در حالت ready قرار بگیرد، منتظر میماند. سپس در قسمت بعد، کاربر مورد نظر را اکسپت می‌کنیم و اتصال را برقرار می‌کنیم و در صورت خطا پیام مورد نظر را نمایش می‌دهیم و برای کاربری که اولین بار متصل می‌شود، پیام خوش آمد ارسال می‌کنیم، در غیر این صورت با فراخوانی تابع create_service، تردی را برای رسیدگی به دستورات ورودی کاربر اختصاص می‌دهیم.

```
int main(int argc, char const *argv[]) {
    char * ip_address = malloc(sizeof(char)*strlen(argv[1]));
    strcpy(ip_address , argv[1]);
    groups = (struct node**) malloc(sizeof(struct node*)*BUFFER_SIZE);
    int server_fd;
    struct sockaddr_in address = *(get_address(ip_address));
    init(ip_address , &address);
    server_fd = create_server(&address);
    int addrlen = sizeof(address);

    //avoiding one thread block other threads
    fd_set current_socket , ready_socket;
    FD_ZERO(&current_socket);
    FD_SET(server_fd , &current_socket);
```

```

while(1){
    ready_socket = current_socket;
    int client_socket;
    check(select(FD_SETSIZE , &ready_socket , NULL , NULL , NULL) , "select\n");
    for(int i = 0; i<FD_SETSIZE;i++){
        if(FD_ISSET(i ,&ready_socket)){
            if ( i == server_fd){
                check((client_socket = accept(server_fd, (struct sockaddr *)&address,&addrlen)) , "not accepted\n");
                FD_SET(client_socket , &current_socket);
                printf("Hello client %s:%d:%d\n", inet_ntoa(address.sin_addr) , ntohs(address.sin_port), client_socket);
            }else{
                create_service(i);
                FD_CLR(i,&current_socket);
            }
        }
    }
}
return 0;
}

```

در آخر هم نمونه های ورودی دستورات و خروجی را می بینیم:

./server <ip_address>

./client <host_name> <ip_address> <client_name>