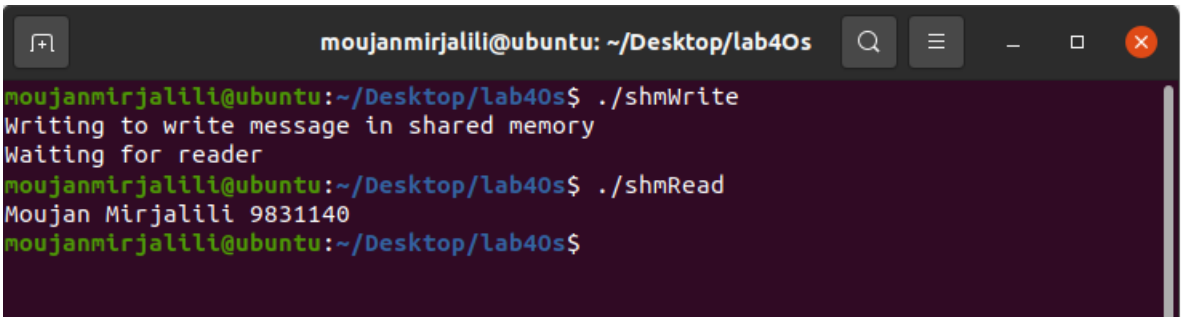
بخش اول

در این بخش باید دو فرآیند ایجاد شد که یکی از آن‌ها می‌نویسد و دیگری می‌خواند.



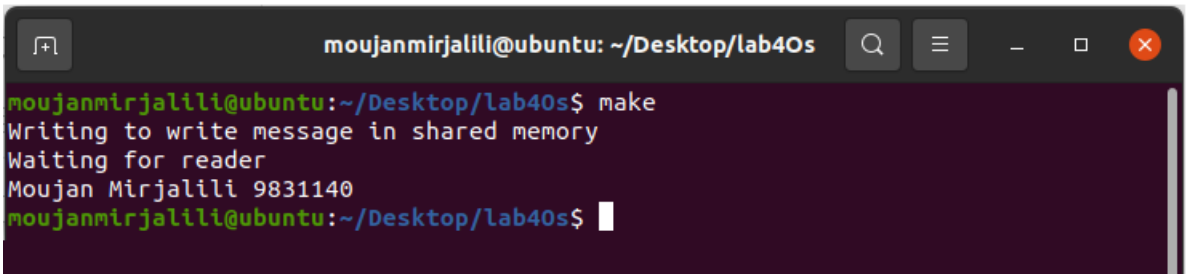
کد بخش نوشتن:

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

#define SHMSIZE sizeof(int) * 25

int main()
{
    int shmid;
    key_t key;
    char *shm;
    char *message = "Moujan Mirjalili 9831140";

    key = 1111;
    if ((shmid = shmget(key, SHMSIZE, IPC_CREAT | 0666)) < 0) {
        perror("shmget");
        exit(1);
    }
    if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
    }
    printf("Writing to write message in shared memory\n");
    sprintf(shm, "%s", message);
    // Waiting for reader...
    printf("Waiting for reader\n");
    if (shmdt(shm) == -1) {
        perror("shmdt");
        exit(1);
    }

    exit(0);
}
```

کد بخش خواندن:

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

#define SHMSIZE sizeof(int) * 25

int main()
{
    int shmid;
    key_t key;
    char *shm;

    key = 1111;

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

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

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

    *shm = '~';

    exit(0);
}
```

با Makefile هم هر دو فرآیند را یک جا نیز می‌توان اجرا کرد.



کد:

```
all: run_writer run_reader

run_writer: writer
        @./shmWrite

run_reader: reader
        @./shmRead

writer:
        @rm -f shmWrite
        @gcc shmWrite.c -o shmWrite

reader:
        @rm -f shmRead
        @gcc shmRead.c -o shmRead

clean:
        @rm *.out
```

بخش دوم

برای این بخش دو فایل ایجاد کرده یکی برای سرور و یکی نیز برای کلاینت. آرایه‌ای ۱۰ تایی از کاربرها ایجاد کرده و داخل هر عضو کاربر آرایه‌ای وجود دارد به جهت این که گروه‌هایی که کاربر در آن عضو است را ذخیره کند. در اینجا امکان حضور ۱۰ کاربر و ۱۰ گروه وجود دارد که با تغییر اندازه آرایه‌های مربوطه می‌توان آن را افزایش داد. ابتدا کد را پیاده سازی کرده و بعد به ترتیب یک سرور و دو کلاینت را اجرا می‌کنیم.

```
moujanmirjalili@ubuntu: ~/Desktop/lab4Os
moujanmirjalili@ubuntu:~/Desktop/lab4Os$ gcc -pthread -o server server.c
server.c: In function ‘client_handler’:
server.c:114:48: warning: too many arguments for format [-Wformat-extra-args]
  114 |            snprintf(response, sizeof(response), "you were not in this gro
up before", client_socket,buffer[5]-'0');
      |                                                ^~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~
moujanmirjalili@ubuntu:~/Desktop/lab4Os$ ./server 8080
Listening on 0.0.0.0:8080
Accepted client 127.0.0.1:51646 id:4
Client 4 joined group 1
Accepted client 127.0.0.1:51734 id:5
Client 5 joined group 1
response : hi client 4

response : hi client 5

Client 5 left group 1
Client 4 left group 1
Client 4: disconnected
Client 5: disconnected
```

```
moujanmirjalili@ubuntu: ~/Desktop/lab4Os
moujanmirjalili@ubuntu:~/Desktop/lab4Os$ gcc -pthread -o client client.c
client.c: In function ‘main’:
client.c:68:5: warning: implicit declaration of function ‘pthread_create’ [-Wimp
licit-function-declaration]
   68 |     pthread_create(&thread1, NULL, thread_handle, (void *) &sock);
      |     ^~~~~~~~~~~~~~
moujanmirjalili@ubuntu:~/Desktop/lab4Os$ ./client 127.0.0.1 8080
join 1
> response : Client 4 joined group 1

send 1 hi client 4
> response : hi client 4

response : hi client 5

leave 1
> response : Client 4 left group 1

quit
Disconnected
moujanmirjalili@ubuntu:~/Desktop/lab4Os$
```

```
moujanmirjalili@ubuntu: ~/Desktop/lab4Os
moujanmirjalili@ubuntu:~/Desktop/lab4Os$ gcc -pthread -o client client.c
client.c: In function ‘main’:
client.c:68:5: warning: implicit declaration of function ‘pthread_create’ [-Wimp
licit-function-declaration]
   68 |     pthread_create(&thread1, NULL, thread_handle, (void *) &sock);
      |     ^~~~~~~~~~~~~~
moujanmirjalili@ubuntu:~/Desktop/lab4Os$ ./client 127.0.0.1 8080
> join 1
> response : Client 5 joined group 1

response : hi client 4

send 1 hi client 5
> response : hi client 5

leave 1
> response : Client 5 left group 1

quit
Disconnected
moujanmirjalili@ubuntu:~/Desktop/lab4Os$
```

server.c × client.c ×

```c
// 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>
// Thread library
#include <pthread.h>


typedef struct
{
    int port;
    int groups[10];
} User;
User users[10] = {0};


// client handler
void *client_handler(void *vargp)
{
    int *temp = (int *)vargp;
    int client_socket = *temp;

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

    int index = -1;
    for (int i = 0; i<10; i++){
        if (users[i].port == client_socket){
            index = i;
            break;
        }
    }
    if (index == -1){
        for (int i = 0; i<10; i++){
            if (users[i].port == 0){
                users[i].port = client_socket;
                index = i;
                break;
            }
        }
    }
    while (1)
    {
        valread = read(client_socket, buffer, sizeof(buffer));
        if (valread < 0)
        {
            perror("Nothing to read");
            exit(EXIT_FAILURE);
        }

        buffer[valread] = '\0';
        // quit
        if (strncmp(buffer, "quit", 4) == 0)
        {
            printf("Client %d: disconnected\n", client_socket);
            users[index].port = 0;
            for (int i=0; i<10;i++){
                users[index].groups[i] = 0;
            }
            break;
        }

        // join
        else if (strncmp(buffer,"join",4) == 0){
            int gp_num = buffer[5]-'0';
            if (users[index].groups[gp_num] == 0) {
                users[index].groups[gp_num] = 1;
                    printf("Client %d joined group %d \n",
        client_socket,buffer[5]-'0');
                    snprintf(response, sizeof(response), "Client %d
joined group %d \n", client_socket,buffer[5]-'0');
                    send(users[index].port, response,
sizeof(response), 0);

```

```
            } else {
                response[0] ='\0';
                snprintf(response, sizeof(response), "Client %d was joined group %d \n", client_socket,buffer[5]-'0');
                send(users[index].port, response, sizeof(response), 0);


            }



        } // send message
        else if (strncmp(buffer,"send",4) == 0){
            int gr = buffer[5]-'0';
            response[0] = '\0';
            memcpy( response, &buffer[7],sizeof (buffer)-7);
            printf("response : %s \n",response);
            for (int i=0; i<10;i++){
                if (users[i].groups[gr] == 1){

                    send(users[i].port,response, sizeof(response), 0);
                }
            }
        }

        // leave
        else if (strncmp(buffer,"leave",5) == 0){
            if ( users[index].groups[buffer[6]-'0'] == 1){
                users[index].groups[buffer[6]-'0'] = 0;
                printf("Client %d left group %d \n", client_socket,buffer[6]-'0');
                snprintf(response, sizeof(response), "Client %d left group %d \n", client_socket,buffer[6]-'0');
                send(users[index].port, response, sizeof(response), 0);
            } else {
                response[0] ='\0';
                    snprintf(response, sizeof(response), "you were not in this group before", client_socket,buffer[5]-'0');
                    send(users[index].port, response, sizeof(response), 0);

            }


        } else send(users[index].port,"enter another", 13, 0);


        fflush(stdout);
        buffer[0] = '\0';
        response[0] = '\0';
    }
}

int main(int argc, char const *argv[])
{
    // descripto
    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);
    }

    // configuration
    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);

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

```

```
    // listen
    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
    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);

        // MultiThreading
        pthread_t thread_id;
        pthread_create(&thread_id, NULL, client_handler, (void *)&client_socket);
    }

    return 0;
}
```

server.c | client.c

```c
// 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>


// thread handler
void* thread_handle(void *socket){
        int valread;
        int sock = * (int *) socket;
        while (1){
                char buffer[1024] = {0};
                valread = read(sock, buffer, sizeof(buffer));
                if (valread < 0){
                        perror("can not read");
                }
                printf("response : %s\n", buffer);


        }


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

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

    // memory set
    memset(&serv_addr, '0', sizeof(serv_addr));

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

    //  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];

    pthread_t thread1;
    pthread_create(&thread1, NULL, thread_handle, (void *) &sock);

    while (1)
    {
        printf("> ");
        fgets(command, 50, stdin);

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

        if (strncmp(command, "quit", 4) == 0)
        {
            printf("Disconnected\n");
            break;
        }

    }

    return 0;
}
```

بخش سوم

برای این بخش دو فرآیند والد و فرزند ایجاد کرده و برای هرکدام از آن‌ها یک خط لوله قرار می‌دهیم که از آن جهت خواندن و نوشتن استفاده می‌شود. فرآیند والد پیامی را می‌نویسد و در خط لوله قرار قرار می‌دهد و فرآیند فرزند آن را می‌خواند و تغییرات را روی حروف اعمال می‌کند و روی خط لوله قرار می‌دهد و فرآیند والد آن را از روی خط لوله می‌خواند و کل این پروسه با استفاده از ۲ خط لوله انجام می‌شود.

```
moujanmirjalili@ubuntu: ~/Desktop/lab4Os
moujanmirjalili@ubuntu:~/Desktop/lab4Os$ gcc -o main main.c
main.c: In function ‘main’:
main.c:44:9: warning: implicit declaration of function ‘gets’; did you mean ‘fge
ts’? [-Wimplicit-function-declaration]
   44 |         gets(pipe1writemessage);
      |         ^~~~
      |         fgets
/usr/bin/ld: /tmp/ccYP2boa.o: in function `main':
main.c:(.text+0xd1): warning: the `gets' function is dangerous and should not be
 used.
moujanmirjalili@ubuntu:~/Desktop/lab4Os$ ./main
Moujan Mirjalili
Parent: Writing to pipe 1  Message : Moujan Mirjalili
Child: Reading from pipe 1 Message : Moujan Mirjalili
Child: Writing to pipe 2  Message : mOUJAN mIRJALILI
Parent: Reading from pipe 2  Message : mOUJAN mIRJALILI
moujanmirjalili@ubuntu:~/Desktop/lab4Os$
```

```c
#include<stdio.h>
#include<unistd.h>
// 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>
// Thread library
#include <pthread.h>
//#include <winsock.h>


int main() {
    int fds1[2], fds2[2];
    // int returnstatus1, returnstatus2;
    int pid;
    char pipe1writemessage[20] = "Hi";
    char pipe2writemessage[20] = "Hello";
    char readmessage[20];
    // returnstatus1 = pipe(fds1);

    if (pipe(fds1) == -1) {
        perror("Unable to create pipe 1 \n");
        exit(EXIT_FAILURE);
    }
    // returnstatus2 = pipe(fds2);

    if (pipe(fds2) == -1) {
        perror("Unable to create pipe 1 \n");
        exit(EXIT_FAILURE);
    }
    pid = fork();

    if (pid != 0){ // Parent process {

        close(fds1[0]); // Close the unwanted pipe1 read side
        close(fds2[1]); // Close the unwanted pipe2 write side
        gets(pipe1writemessage);

        printf("Parent: Writing to pipe 1  Message : %s\n", pipe1writemessage);
        write(fds1[1], pipe1writemessage, sizeof(pipe1writemessage));
        read(fds2[0], readmessage, sizeof(readmessage));
        printf("Parent: Reading from pipe 2  Message : %s\n", readmessage);
    } else { //child process

        close(fds1[1]); // Close the unwanted pipe1 write side
        close(fds2[0]); // Close the unwanted pipe2 read side
        read(fds1[0], readmessage, sizeof(readmessage));

        printf("Child: Reading from pipe 1 Message : %s\n", readmessage);
        memcpy(pipe2writemessage, readmessage, sizeof(readmessage));
        for (int i = 0; i<20; i++){
                if ( pipe2writemessage [i] >= 65 && pipe2writemessage [i] <= 90)
                        pipe2writemessage[i] = pipe2writemessage[i] + 32;
                else if (pipe2writemessage [i] >= 65+32 && pipe2writemessage [i] <= 90+32)
            pipe2writemessage[i] = pipe2writemessage[i] - 32;


        }


        printf("Child: Writing to pipe 2  Message : %s\n", pipe2writemessage);
        write(fds2[1], pipe2writemessage, sizeof(pipe2writemessage));
    }
    return 0;
}
```