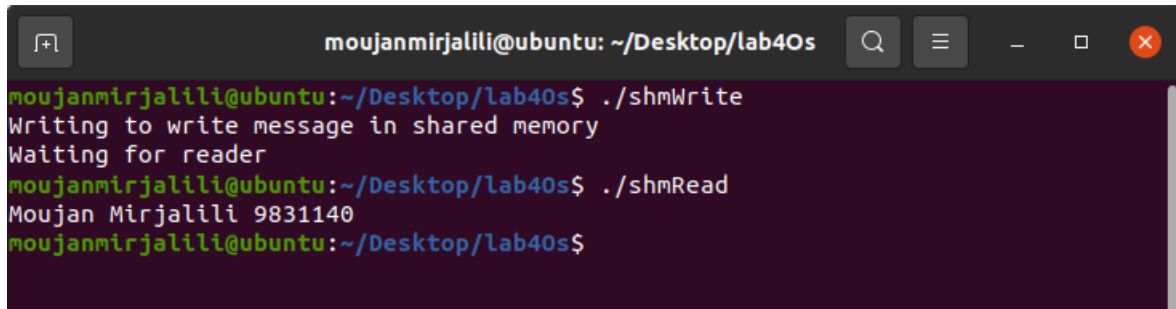


بخش اول

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



```
moujanmirjalili@ubuntu: ~/Desktop/lab4Os
moujanmirjalili@ubuntu:~/Desktop/lab4Os$ ./shmWrite
Writing to write message in shared memory
Waiting for reader
moujanmirjalili@ubuntu:~/Desktop/lab4Os$ ./shmRead
Moujan Mirjalili 9831140
moujanmirjalili@ubuntu:~/Desktop/lab4Os$
```

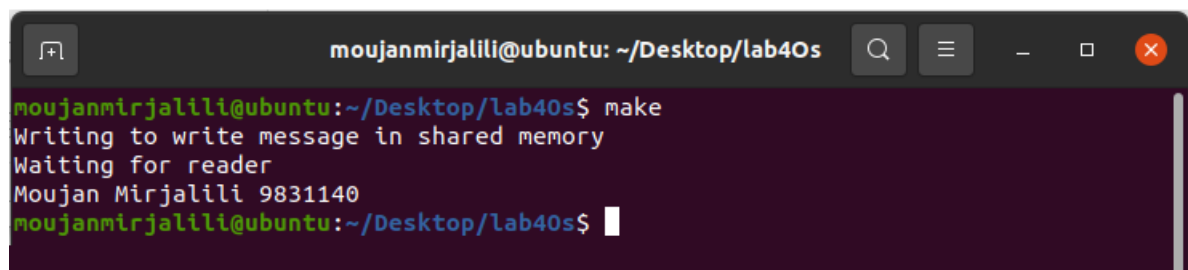
کد بخش نوشتن:

```
1 #include <sys/types.h>
2 #include <sys/ipc.h>
3 #include <sys/shm.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <unistd.h>
7
8 #define SHMSIZE sizeof(int) * 25
9
10 int main()
11 {
12     int shmid;
13     key_t key;
14     char *shm;
15     char *message = "Moujan Mirjalili 9831140";
16
17     key = 1111;
18     if ((shmid = shmget(key, SHMSIZE, IPC_CREAT | 0666)) < 0) {
19         perror("shmget");
20         exit(1);
21     }
22     if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
23         perror("shmat");
24         exit(1);
25     }
26     printf("Writing to write message in shared memory\n");
27     sprintf(shm, "%s", message);
28     // Waiting for reader...
29     printf("Waiting for reader\n");
30     if (shmdt(shm) == -1) {
31         perror("shmdt");
32         exit(1);
33     }
34
35     exit(0);
36 }
```

کد بخش خواندن:

```
1 #include <sys/types.h>
2 #include <sys/ipc.h>
3 #include <sys/shm.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <unistd.h>
7
8 #define SHMSIZE sizeof(int) * 25
9
10 int main()
11 {
12     int shmid;
13     key_t key;
14     char *shm;
15
16     key = 1111;
17
18     if ((shmid = shmget(key, SHMSIZE, 0666)) < 0) {
19         perror("shmget");
20         exit(1);
21     }
22
23     if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
24         perror("shmat");
25         exit(1);
26     }
27
28     printf("%s\n", shm);
29
30     *shm = '~';
31
32     exit(0);
33 }
```

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

A terminal window titled 'moujanmirjalili@ubuntu: ~/Desktop/lab40s' with standard window controls. The terminal shows the command 'make' being executed, followed by the output: 'Writing to write message in shared memory', 'Waiting for reader', and 'Moujan Mirjalili 9831140'. The prompt returns to 'moujanmirjalili@ubuntu:~/Desktop/lab40s\$' with a cursor.

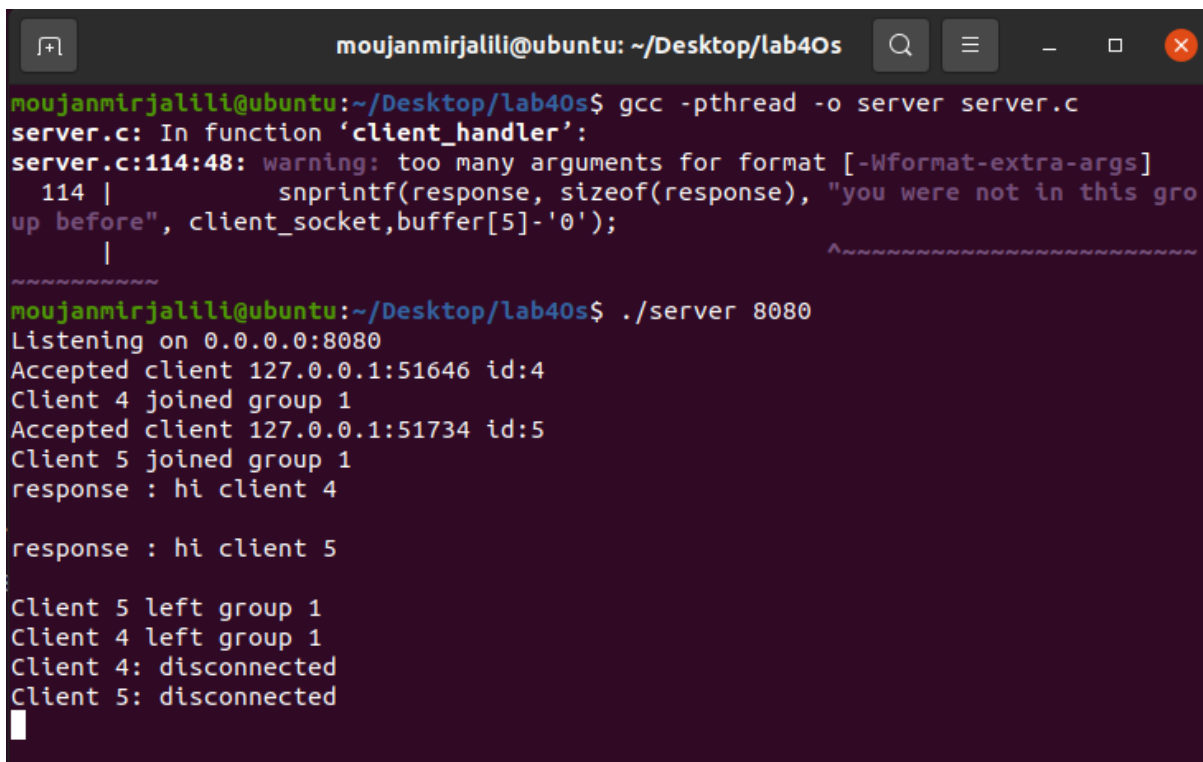
```
moujanmirjalili@ubuntu: ~/Desktop/lab40s
moujanmirjalili@ubuntu:~/Desktop/lab40s$ make
Writing to write message in shared memory
Waiting for reader
Moujan Mirjalili 9831140
moujanmirjalili@ubuntu:~/Desktop/lab40s$
```

کد:

```
1 all: run_writer run_reader
2
3 run_writer: writer
4     @./shmWrite
5
6 run_reader: reader
7     @./shmRead
8
9 writer:
10     @rm -f shmWrite
11     @gcc shmWrite.c -o shmWrite
12
13 reader:
14     @rm -f shmRead
15     @gcc shmRead.c -o shmRead
16
17 clean:
18     @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/lab40s
moujanmirjalili@ubuntu:~/Desktop/lab40s$ 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/lab40s$ ./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/lab40s
moujanmirjalili@ubuntu:~/Desktop/lab40s$ gcc -pthread -o client client.c
client.c: In function 'main':
client.c:68:5: warning: implicit declaration of function 'pthread_create' [-Wimplicit-function-declaration]
   68 |     pthread_create(&thread1, NULL, thread_handle, (void *) &sock);
      |     ^
moujanmirjalili@ubuntu:~/Desktop/lab40s$ ./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/lab40s$
```

```
moujanmirjalili@ubuntu: ~/Desktop/lab40s
moujanmirjalili@ubuntu:~/Desktop/lab40s$ gcc -pthread -o client client.c
client.c: In function 'main':
client.c:68:5: warning: implicit declaration of function 'pthread_create' [-Wimplicit-function-declaration]
   68 |     pthread_create(&thread1, NULL, thread_handle, (void *) &sock);
      |     ^
moujanmirjalili@ubuntu:~/Desktop/lab40s$ ./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/lab40s$
```

```

server.c
client.c
1 // Socket libraries
2 #include <unistd.h>
3 #include <stdio.h>
4 #include <sys/socket.h>
5 #include <stdlib.h>
6 #include <netinet/in.h>
7 #include <arpa/inet.h>
8 //
9 #include <string.h>
10 #include <ctype.h>
11 #include <time.h>
12 // Thread library
13 #include <pthread.h>
14
15
16 typedef struct
17 {
18     int port;
19     int groups[10];
20 } User;
21 User users[10] = {0};
22
23
24 // client handler
25 void *client_handler(void *vargp)
26 {
27     int *temp = (int *)vargp;
28     int client_socket = *temp;
29
30     int valread;
31     char buffer[1024] = {0};
32     char response[1024] = {0};
33
34     int index = -1;
35     for (int i = 0; i<10; i++){
36         if (users[i].port == client_socket){
37             index = i;
38             break;
39         }
40     }
41     if (index == -1){
42         for (int i = 0; i<10; i++){
43             if (users[i].port == 0){
44                 users[i].port = client_socket;
45                 index = i;
46                 break;
47             }
48         }
49     }
50     while (1)
51     {
52         valread = read(client_socket, buffer, sizeof(buffer));
53         if (valread < 0)
54         {
55             perror("Nothing to read");
56             exit(EXIT_FAILURE);
57         }
58
59         buffer[valread] = '\0';
60         // quit
61         if (strcmp(buffer, "quit", 4) == 0)
62         {
63             printf("Client %d: disconnected\n", client_socket);
64             users[index].port = 0;
65             for (int i=0; i<10;i++){
66                 users[index].groups[i] = 0;
67             }
68             break;
69         }
70
71         // join
72         else if (strcmp(buffer,"join",4) == 0){
73             int gp_num = buffer[5]-'0';
74             if (users[index].groups[gp_num] == 0) {
75                 users[index].groups[gp_num] = 1;
76                 printf("Client %d joined group %d \n",
77                     client_socket,buffer[5]-'0');
78                 sprintf(response, sizeof(response), "Client %d
79                     joined group %d \n", client_socket,buffer[5]-'0');
80                 send(users[index].port, response,
81                     sizeof(response), 0);
82             }
83         }
84     }
85 }
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

80
81         } else {
82             response[0] = '\0';
83             snprintf(response, sizeof(response), "Client %d
was joined group %d \n", client_socket,buffer[5]-'0');
84             send(users[index].port, response,
sizeof(response), 0);
85
86
87         }
88
89
90
91     } // send message
92     else if (strncmp(buffer,"send",4) == 0){
93         int gr = buffer[5]-'0';
94         response[0] = '\0';
95         memcpy( response, &buffer[7],sizeof (buffer)-7);
96         printf("response : %s \n",response);
97         for (int i=0; i<10;i++){
98             if (users[i].groups[gr] == 1){
99
100                 send(users[i].port,response, sizeof(response),
0);
101             }
102         }
103     }
104
105     // leave
106     else if (strncmp(buffer,"leave",5) == 0){
107         if ( users[index].groups[buffer[6]-'0'] == 1){
108             users[index].groups[buffer[6]-'0'] = 0;
109             printf("Client %d left group %d \n",
client_socket,buffer[6]-'0');
110             snprintf(response, sizeof(response), "Client %d
left group %d \n", client_socket,buffer[6]-'0');
111             send(users[index].port, response, sizeof(response),
0);
112         } else {
113             response[0] = '\0';
114             snprintf(response, sizeof(response), "you were
not in this group before", client_socket,buffer[5]-'0');
115             send(users[index].port, response,
sizeof(response), 0);
116         }
117
118
119
120     } else send(users[index].port,"enter another", 13, 0);
121
122
123     fflush(stdout);
124     buffer[0] = '\0';
125     response[0] = '\0';
126 }
127 }
128
129 int main(int argc, char const *argv[])
130 {
131     // descripto
132     int server_fd;
133     server_fd = socket(AF_INET, SOCK_STREAM, 0); // TODO: What is
this do
134     if (server_fd == 0)
135     {
136         perror("Socket failed");
137         exit(EXIT_FAILURE);
138     }
139
140     // configuration
141     struct sockaddr_in address;
142     address.sin_family = AF_INET;
143     address.sin_addr.s_addr = INADDR_ANY;
144     address.sin_port = htons(atoi(argv[1]));
145     const int addrlen = sizeof(address);
146
147     // bind
148     if (bind(server_fd, (struct sockaddr *)&address, addrlen) < 0)
149     {
150         perror("Bind failed");
151         exit(EXIT_FAILURE);
152     }
153     ...

```

```

154     // listen
155     if (listen(server_fd, 3) < 0)
156     {
157         perror("Listen failed");
158         exit(EXIT_FAILURE);
159     }
160
161     printf("Listening on %s:%d\n", inet_ntoa(address.sin_addr),
162           ntohs(address.sin_port));
163
164     // Accepting client
165     while (1)
166     {
167         int client_socket;
168         if ((client_socket = accept(server_fd, (struct sockaddr
169 *)&address, (socklen_t *)&addrlen)) < 0)
170         {
171             perror("Accept failed");
172             exit(EXIT_FAILURE);
173         }
174         printf("Accepted client %s:%d id:%d\n",
175               inet_ntoa(address.sin_addr), ntohs(address.sin_port),
176               client_socket);
177
178         // MultiThreading
179         pthread_t thread_id;
180         pthread_create(&thread_id, NULL, client_handler, (void
181 *)&client_socket);
182     }
183
184     return 0;
185 }

```

server.c	client.c
1	// socket libraries
2	#include <stdio.h>
3	#include <sys/socket.h>
4	#include <stdlib.h>
5	#include <netinet/in.h>
6	#include <arpa/inet.h>
7	//
8	#include <string.h>
9	#include <unistd.h>
10	
11	
12	// thread handler
13	void* thread_handle(void *socket){
14	int valread;
15	int sock = * (int *) socket;
16	while (1){
17	char buffer[1024] = {0};
18	valread = read(sock, buffer, sizeof(buffer));
19	if (valread < 0){
20	perror("can not read");
21	}
22	printf("response : %s\n", buffer);
23	
24	}
25	
26	
27	
28	}
29	int main(int argc, char const *argv[])
30	{
31	int sock = 0;
32	struct sockaddr_in serv_addr;
33	
34	// create socket
35	if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
36	{
37	perror("Socket creation error");
38	exit(EXIT_FAILURE);
39	}
40	
41	// memory set
42	memset(&serv_addr, '0', sizeof(serv_addr));
43	
44	// address family
45	serv_addr.sin_family = AF_INET;
46	serv_addr.sin_port = htons(atoi(argv[2]));
47	
48	// convert IPv4 and IPv6 to binary
49	if (inet_pton(AF_INET, argv[1], &serv_addr.sin_addr) <= 0)
50	{
51	perror("Invalid address");
52	exit(EXIT_FAILURE);
53	}
54	
55	// Connecting to the server
56	if (connect(sock, (struct sockaddr *)&serv_addr,
57	sizeof(serv_addr)) < 0)
58	{
59	perror("Connection failed");
60	exit(EXIT_FAILURE);
61	}
62	// User commands
63	int valread;
64	char buffer[1024] = {0};
65	char command[50];
66	
67	pthread_t thread1;
68	pthread_create(&thread1, NULL, thread_handle, (void *) &sock);
69	
70	while (1)
71	{
72	printf("> ");
73	fgets(command, 50, stdin);
74	
75	send(sock, command, strlen(command), 0);
76	
77	if (strcmp(command, "quit", 4) == 0)
78	{
79	printf("Disconnected\n");
80	break;
81	}
82	
83	}
84	
85	return 0;
86	}

بخش سوم

برای این بخش دو فرآیند والد و فرزند ایجاد کرده و برای هر کدام 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/lab40s
moujanmirjalili@ubuntu:~/Desktop/lab40s$ gcc -o main main.c
main.c: In function 'main':
main.c:44:9: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-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/lab40s$ ./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/lab40s$
```

```

1#include<stdio.h>
2#include<unistd.h>
3// Socket libraries
4#include <unistd.h>
5#include <stdio.h>
6#include <sys/socket.h>
7#include <stdlib.h>
8#include <netinet/in.h>
9#include <arpa/inet.h>
10//
11#include <string.h>
12#include <ctype.h>
13#include <time.h>
14// Thread library
15#include <pthread.h>
16//#include <winsock.h>
17
18
19int main() {
20    int fds1[2], fds2[2];
21    // int returnstatus1, returnstatus2;
22    int pid;
23    char pipe1writemessage[20] = "Hi";
24    char pipe2writemessage[20] = "Hello";
25    char readmessage[20];
26    // returnstatus1 = pipe(fds1);
27
28    if (pipe(fds1) == -1) {
29        perror("Unable to create pipe 1 \n");
30        exit(EXIT_FAILURE);
31    }
32    // returnstatus2 = pipe(fds2);
33
34    if (pipe(fds2) == -1) {
35        perror("Unable to create pipe 1 \n");
36        exit(EXIT_FAILURE);
37    }
38    pid = fork();
39
40    if (pid != 0){ // Parent process {
41
42        close(fds1[0]); // Close the unwanted pipe1 read side
43        close(fds2[1]); // Close the unwanted pipe2 write side
44        gets(pipe1writemessage);
45
46        printf("Parent: Writing to pipe 1 Message : %s\n",
pipe1writemessage);
47        write(fds1[1], pipe1writemessage,
sizeof(pipe1writemessage));
48        read(fds2[0], readmessage, sizeof(readmessage));
49        printf("Parent: Reading from pipe 2 Message : %s\n",
readmessage);
50    } else { //child process
51
52        close(fds1[1]); // Close the unwanted pipe1 write side
53        close(fds2[0]); // Close the unwanted pipe2 read side
54        read(fds1[0], readmessage, sizeof(readmessage));
55
56        printf("Child: Reading from pipe 1 Message : %s\n",
readmessage);
57        memcpy(pipe2writemessage, readmessage, sizeof(readmessage));
58        for (int i = 0; i<20; i++){
59            if ( pipe2writemessage [i] >= 65 &&
pipe2writemessage [i] <= 90)
60                pipe2writemessage[i] = pipe2writemessage[i]
+ 32;
61            else if (pipe2writemessage [i] >= 65+32 &&
pipe2writemessage [i] <= 90+32)
62                pipe2writemessage[i] = pipe2writemessage[i] - 32;
63
64        }
65
66        printf("Child: Writing to pipe 2 Message : %s\n",
pipe2writemessage);
67        write(fds2[1], pipe2writemessage,
sizeof(pipe2writemessage));
68    }
69    return 0;
70 }
71
72 }

```