

تمرین ها:

-1

در این قسمت می خواهیم با استفاده از روش shared memory رشته ای توسط یک فرایند بنویسیم سپس توسط فرایند دیگر دسترسی به آن دهیم. سپس توسط فرایند دیگر که دسترسی به shared memory به آن می دهیم، رشته را می خوانیم.

در کد از shmget برای ساختن یک shared memory استفاده کرده ایم. این تابع پس از ایجاد آن یک id رمی گرداند. از shmat برای وصل کردن فرایند به shared memory استفاده می کنیم. این تابع آدرس شروع shared memory را به عنوان خروجی بر میگرداند که آنرا در یک اشاره گر به کاراکتر ذخیره می کنیم. برای ورودی shmget نیاز هست که یک key که یکتا می باشد را به عنوان ورودی به آن بدهیم که بین فرستنده و گیرنده یکسان است تا هر دو به shared memory دسترسی داشته باشند.

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <stdio.h>

int main()
{
    // ftok to generate unique key
    key_t key = ftok("shmfile",65);
    // shmget returns an identifier in shmid
    int segment_id = shmget(key,1024,0666|IPC_CREAT);
    // shmat to attach to shared memory
    char *str = (char*) shmat(segment_id,NULL,0);
    printf("Write Data : ");
    gets(str);
    printf("Data written in memory: %s\n",str);
    //detach from shared memory
    shmdt(str);

    return 0;
}
```

در این کد هم مثل قسمت قبلا از توابع `shmget` و `shmat` استفاده کرده ایم. و به کمک پوینتری که خروجی `shmat` برمیگرداند، رشه نوشته شده در حافظه ی مشترک را می خوانیم و آن را در خروجی چاپ می کنیم. در آخر هم از `shared memory` پردازه را جدا می کنیم و `shared memory` را از بین می بریم.

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <stdio.h>

int main()
{
    // ftok to generate unique key
    key_t key = ftok("shmfile",65);
    // shmget returns an identifier in shmid
    int segment_id = shmget(key,1024,S_IRUSR|S_IWUSR);
    // shmat to attach to shared memory
    char *str = (char*) shmat(segment_id,NULL,0);
    printf("Data read from memory: %s\n",str);
    //detach from shared memory
    shmdt(str);
    // destroy the shared memory
    shmctl(segment_id,IPC_CREAT,NULL);
    return 0;
}
```

دو فایل خود را کامپایل کرده سپس writer را اجرا میکنیم که یک رشته را می گیرد و در shared memory می نویسد سپس فایل reader را اجرا می کنیم و از shared memory رشته ی نوشته شده را می خوانیم و چاپ می کنیم.

```
helia@helia-virtual-machine: ~/Desktop/OSlab/lab04/1
helia@helia-virtual-machine:~$ cd Desktop/OSlab/lab04
helia@helia-virtual-machine:~/Desktop/OSlab/lab04$ cd 1
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/1$ ./reader
Segmentation fault (core dumped)
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/1$ gcc -o writer writer.c
writer.c: In function 'main':
writer.c:18:2: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
    gets(str);
    ^~~~~
fgets
/usr/bin/ld: /tmp/ccpjYsL5.o: in function 'main':
writer.c:(.text+0x69): warning: the 'gets' function is dangerous and should not be used.
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/1$ gcc -o reader reader.c
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/1$ ./writer
./writer: command not found
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/1$ ./writer
Write Data : he
Data written in memory: he
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/1$ ./reader
Data read from memory: he
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/1$
```

کتابخانه های مورد نیاز و مقادیر global را تعریف کرده ایم

سرور

```

1 #include <unistd.h>
2 #include <stdio.h>
3 #include <sys/socket.h>
4 #include <stdlib.h>
5 #include <netinet/in.h>
6 #include <arpa/inet.h>
7 #include <string.h>
8 #include <pthread.h>
9 #define GroupNumber 5
10 #define NOClient 20
11 #define messageSize 80
12
13
14 int clients[NOClient];
15 int n=0;
16
17 struct group{
18     int id;
19     int clients[NOClient];
20     int memeber_number;
21 };
22 struct group groups[GroupNumber];
23 //build groups
24 void build_groups(){
25     for(int i=0;i<GroupNumber;i++){
26         groups[i].id=i;
27         groups[i].memeber_number=0;
28     }
29     printf("building group ...\n");
30 }
31
32 //send message to members
33 void sendToAll(int current,int group_id, char *message){
34     int i;
35     struct group* gp=&groups[group_id];
36     for(i = 0; i < gp->memeber_number; i++) {
37         if(gp->clients[i]!=current){
38             if(send(gp->clients[i],message,strlen(message),0) < 0) {
39                 printf("sending failed! \n");
40                 continue;
41             }
42             else{
43                 printf("message sent to client %d\n",gp-
44 >clients[i]);
45             }
46         }
47     }
48 }

```

Struct را تعریف کرده ایم که شامل شماره ی گروه و اعضای گروه و تعداد اعضای گروه می باشد.

فرستادن پیام برای تمامی اعضای گروه

برای اینکه عضو جدیدی به گروه اضافه بشود این تابع صدا زده می شود و اگر شماره ی گروه معتبر باشد شخص اضافه شده به همه ی گروه اطلاع می دهد.

```
46 }
47 }
48
49 //join
50 int join_group(int group_id,int client,char* name){
51     if(group_id<0 || group_id>=GroupNumber){
52         printf("invalid group id! try again \n");
53         return -1;
54     }
55     struct group* gp=&groups[group_id];
56     gp->clients[gp->memeber_number]=client;
57     gp->memeber_number=(gp->memeber_number)+1;
58     char message[messageSize];
59     sprintf(message,"%s joined group %d\n",name,group_id);
60     sendToAll(client,group_id,message);
61     return 0;
62 }
63
64 //leave
65 int leave_group(int group_id,int client,char* name){
66     if(group_id<0 || group_id>=GroupNumber){
67         printf("invalid group id! \n");
68         return -1;
69     }
70     int flag= 0;
71     struct group* gp=&groups[group_id];
72     for(int i=0;i<gp->memeber_number;i++){
73         if(client==gp->clients[i]){
74             flag=1;
75             for(int j=i;j<gp->memeber_number;j++){
76                 gp->clients[j]=gp->clients[j+1];
77             }
78             gp->memeber_number=gp->memeber_number-1;
79             gp->clients[gp->memeber_number]=0;
80             char message[messageSize];
81             sprintf(message,"%s left the group
82 %d\n%c",name,group_id,'\0');
83             sendToAll(client,group_id,message);
84             break;
85         }
86     }
87     if(!flag){
88         char message[messageSize];
89         sprintf(message,"error!, you are not in group
90 %d\n",group_id);
91         send(client,message,strlen(message),0);
92         return -1;
93     }
```

برای ترک کردن گروه از تابع بالا استفاده می شود . در آن پیمایش صورت می گیرد و شخص مورد نظر را پیدا کرده و تمام اطلاعات او را پاک می کند. یک شدن flag به معنی پیدا شدن شخص مورد نظر است سپس یک پیام مناسب چاپ می شود

```

91 }
92 return 0;
93 }
94 }
95 //communication
96

```

این تابع درخواست کاربر مدیریت میکند در اصل با گرفتن سوکت مورد نظر و پیام کاربر کار مورد نظر را انجام میدهد.

```

97 int handle(char *message, int sock){
98     printf("msg: %s\n",message);
99     char* cmd;
100    char* client_name;
101    char* gp_id;
102    int group_id=0;
103    client_name=strtok(message,":");
104    printf("name: %s\n",client_name);
105    cmd = strtok (NULL," ");
106    printf("cmd: %s\n",cmd);
107

```

```

108     if(!strcmp(cmd,"join")){
109         gp_id=strtok(NULL,"\0 ");
110         group_id=(int)*gp_id-48;
111         join_group(group_id,sock,client_name);
112     }
113

```

```

114     else if(!strcmp(cmd,"send")){
115         int flag= 0;

```

```

116         gp_id=strtok(NULL," ");
117         char* msg ;
118         msg=strtok(NULL,"\n");
119         group_id=(int)*gp_id-48;
120         struct group* gp=&groups[group_id];
121         for(int i=0;i<gp->memeber_number;i++){
122             if(sock==gp->clients[i]){
123                 flag=1;
124                 break;
125             }
126         }
127         char message[100];
128         if(!flag){
129             sprintf(message,"you are not in group %d\n",group_id);
130             send(sock,message,strlen(message),0);
131             return -1;
132         }else{
133             sprintf(message,"%s : %s\n",client_name,msg);
134             sendToAll(sock,group_id,message);
135         }
136     }
137 }
138

```

با استفاده از strtok پیام را تکه تکه کرده و اطلاعات مورد نظر خود را از آن استخراج می کنیم این اطلاعات وابسته به نوع پیام می تواند cmd یا همان دستور کاربر و اسم کاربر و شماره گروه باشد.

اگر درخواست کاربر اضافه شدن به گروه باشد با استخراج شماره ی گروه مورد نظر تابع join_group را صدا می زنیم. اگر درخواست کاربر فرستادن پیام باشد شماره گروه را از پیام به دست آورده و سپس پیمایش در آن گروه با استفاده از متغیر flag چک می کنیم که آیا شخص در گروه هست یا نه اگر بود تابع sendToAll صدا زده می شود در غیر این صورت پیام مناسب چاپ می شود

کاربر قصد خروج به طور کلی را دارد در سرور و کلاینت پیام متنی مناسب نوشته شده و ارتباط ما یعنی ترد مربوطه با سرور قطع شده و دیگر سرور کامندهای کلاینت را نمی تواند بخواند و کلاینت ارتباط خود را با سرور از دست می دهد.

```
139 else if(!strcmp(cmd, "quit")){
140     char message[100];
141     sprintf(message, "disconnecting... %d\n", group_id);
142     send(sock, message, strlen(message), 0);
143     pthread_t pt=pthread_self();
144     pthread_cancel(pt);
145     return -1;
146 }
```

```
147
148 else if(!strcmp(cmd, "leave")){
149     gp_id=strtok(NULL, " ");
150     group_id=(int)*gp_id-48;
151     leave_group(group_id, sock, client_name);
152 }
153
154
155
156
157
158
159
160 }
```

```
161
162 //receives message and calls handler to manage the command
163 void *receive_msg(void *client_sock){
164     int sock = *((int *)client_sock);
165     char message[500];
166     int len;
167     // server thread always ready to receive message and handle it
168     while((len = recv(sock, message, 500, 0)) > 0) {
169         message[len] = '\0';
170         handle(message, sock);
171     }
172 }
173
174
175
176 int main(int argc, char const *argv[]) {
177     char buffer[1024] = {0};
178     char *hello = "Hello from server";
179     pthread_t recvt;
180     // creates socket file descriptor
181     int server_fd;
182     server_fd = socket(AF_INET, SOCK_STREAM, 0);
183     if (server_fd == 0) {
184         perror("socket failed");
185         exit(EXIT_FAILURE);
186     }else{
```

اگر در خواست اشتباه اشد
پیغام مناسب نشان داده شود

اگر کاربر قصد
ترک کردن گروه
را داشته باشد با
استخراج شماره
گروه مد نظر
تابع
leave_group
صدا می زنیم.

این تابع برای دریافت
پیام از کلاینت هست. این
ورودی سوکت به ما
کمک می کند تا راه
ارتباطی مورد نظر را
بدانیم سپس با چک
کردن طول مسیج تابع
handle را برای
برطرف کردن خواسته ی
کاربر صدا میزند.

دستور `binding` و `listening on server socket` را مثل دستور کار پیاده سازی می کنیم.

```
188     printf("server started\n");
189 }
190
191 int port = atoi(argv[1]);
192 struct sockaddr_in address;
193 address.sin_family = AF_INET;
194 address.sin_addr.s_addr = INADDR_ANY;
195 address.sin_port = htons(port); // host to network -- converts
the ending of the given integer
196 const int addrlen = sizeof(address);
197
198 // binding
199 if (bind(server_fd, (struct sockaddr *)&address,
sizeof(address)) < 0) {
200     perror("bind failed");
201     exit(EXIT_FAILURE);
202 }
203
204 // listening on server socket with backlog size 20.
205 if (listen(server_fd, 20) < 0) {
206     perror("listen");
207     exit(EXIT_FAILURE);
208 }
209
```

```
209
210 build_groups();
211
212 //listening
213 printf("Listen on %s:%d\n", inet_ntoa(address.sin_addr),
ntohs(address.sin_port));
214 int Client_sock;
215 while(1){
216     if( (Client_sock = accept(server_fd, (struct sockaddr *)
&address, (socklen_t*)&addrlen)) < 0 )
217         printf("accept failed \n");
218     else
219         printf("accepted %d\n", Client_sock);
220
221     clients[n]= Client_sock;
222     n++;
223     // creating a thread for each client
224     send(Client_sock,hello,sizeof(hello),0);
225     pthread_create(&recvt,NULL,(void
*)receive_msg,&Client_sock);
226
227 }
228 return 0;
229 }
```

آدرس و پورتهی که سرور در حال پوش دادن آن است را نمایش می دهیم. سپس در یک حلقه ی بی نهایت سعی بر قبول کردن درخواست های ارتباطی کلاینت ها داریم. (مطابق با دستور کار پیاده سازی شده است) وصل شدن یا نشدن کلاینت به سرور را نشان می دهیم و وقتی که صبول شود پیام `hello` ارسال می شود سپس با ایجاد یک ریسمان برای کلاینت آماده انجام کامندهای این کاربر می شویم و در حلقه خود و خارج از این ترد در حال گوش دادن برای ارتباط بعدی می باشیم.

کلاینت

بخش ابتدایی تابع main و receive_msg مثل سرور است.

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

char message[500];
void *receive_msg(void *my_sock)
{
    int sock = *((int *)my_sock);
    int len;
    // client thread always ready to receive message
    while((len = recv(sock,message,500,0)) > 0) {
        message[len] = '\0';
        printf("%s\n",message);
    }
}

int main(int argc, char const *argv[]) {
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    char *hello = "Hello from client";
    char buffer[1024] = {0};
    char send_msg[500],client_name[100];
    pthread_t recvt;
    int port=atoi(argv[1]);
    strcpy(client_name, argv[2]);
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        return -1;
    }
    // sets all memory cells to zero
    memset(&serv_addr, '0', sizeof(serv_addr));
    // sets port and address family
    serv_addr.sin_family = AF_INET; //match the socket() call
    serv_addr.sin_port = htons(port); //specify port to listen on
    serv_addr.sin_addr.s_addr=INADDR_ANY; //bind to any local
    address

    //Connect to server
    if (connect(sock, (struct sockaddr *)&serv_addr,
    sizeof(serv_addr)) < 0)
    {
        printf("\nConnection Failed \n");
    }
}
```

این بخش مربوط به وصل شدن سرور است که پیام مناسب را چاپ می کند.

```
        return -1;
    }else{
        printf("connected...");
    }

    printf("Hello message sent\n");
    valread = read(sock, buffer, 1024);
    if (valread < 0) {
        perror("could not read");
        return -1;
    }

    int len;
    //creat thread
    pthread_create(&recvt, NULL, (void *)receive_msg, &sock);
    while(fgets(message, 500, stdin) > 0) {
        strcpy(send_msg, client_name);
        strcat(send_msg, ":");
        strcat(send_msg, message);
        len = write(sock, send_msg, strlen(send_msg));
        if(len < 0)
            printf("\n message not sent \n");
    }
    //join to thread
    pthread_join(recvt, NULL);
    //close socket
    close(sock);
    printf("%s\n", buffer);
    return 0;
}
```

ترد اصلی و اولیه پردازش را برای نوشتن به روی سرور به واسطه ی سوکت انتخاب کرده و سپس ترد جدیدی می سازیم تا به کمک آن از سرور پیام را دریافت کنیم بنابراین به این ترتیب در هر لحظه می توانیم هم پیامی به سرور فرستاده و هم از آن دریافت بکنیم. در آخر هم سوکت را بسته و بافر را چاپ می کنیم.

- سرور و کلاینت را اجرا و به هم وصل کرده و برای کلاینت اسم فرد و پورت را می دهیم.

```
helia@helia-virtual-machine: ~/Desktop/OSlab/lab04/2
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ gcc server.c -o server -lpthread
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ ./server 8000
server started
building group ...
Listen on 0.0.0.0:8000
```

```
helia@helia-virtual-machine: ~/Desktop/OSlab/lab04/2
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ gcc client.c -o client -lpthread
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ ./client 8000 nikta
connected...Hello message sent
```

```
helia@helia-virtual-machine: ~/Desktop/OSlab/lab04/2
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ gcc client.c -o client -lpthread
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ ./client 8000 helia
connected...Hello message sent
```

- با استفاده از دستور join افرادی که به سرور وصل شده اند را می توان به گروه مورد نظرشان وصل کرد و هر یک از افرادی که اضافه می شود به افراد آن گروه یک پیغام داده می شود.

```
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ gcc server.c -o server -lpthread
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ ./server 8000
server started
building group ...
Listen on 0.0.0.0:8000
accepted 4
msg: helia:join 1

name: helia
cmd: join
accepted 5
msg: nikta:join 2

name: nikta
cmd: join

msg: helia:join 2

name: helia
cmd: join
message sent to client 5
```

```
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ gcc client.c -o client -lpthread
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ ./client 8000 nikta
connected...Hello message sent
join 2
helia joined group 2
```

```
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ gcc client.c -o client -lpthread
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ ./client 8000 helia
connected...Hello message sent
join 1
join 2
```


- می‌توانیم به گروه مورد نظر پیام بفرستیم و در سرور و دیگر اعضای گروه می‌توانند آن پیام نمایش داده می‌شود.

```
helia@helia-virtual-machine: ~/Desktop/OSlab/lab04/2
building group ...
Listen on 0.0.0.0:8000
accepted 4
msg: helia:join 1

name: helia
cmd: join
accepted 5
msg: nikta:join 2

name: nikta
cmd: join

msg: helia:join 2

name: helia
cmd: join
message sent to client 5
msg: nikta:send 2 hi helia

name: nikta
cmd: send
message sent to client 4
msg: helia:send 2 hello nikta

name: helia
cmd: send
message sent to client 5
```

```
helia@helia-virtual-machine: ~/Desktop/OSlab/lab04/2
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ gcc client.c -o client -lpthread
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ ./client 8000 nikta
connected...Hello message sent
join 2
helia joined group 2

send 2 hi helia
helia : hello nikta
```

```
helia@helia-virtual-machine: ~/Desktop/OSlab/lab04/2
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ gcc client.c -o client -lpthread
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ ./client 8000 helia
connected...Hello message sent
join 1
join 2
nikta : hi helia

send 2 hello nikta
```

- با استفاده از `leave` گروه مورد نظر را ترک کرده و به دیگر اعضا و سرور نمایش داده می شود و دیگر نمی توان در آن گروه مسیج بفرستیم.

```

helia@helia-virtual-machine: ~/Desktop/OSlab/lab04/2
name: helia
cmd: join
message sent to client 5
msg: nikta:send 2 hi helia

name: nikta
cmd: send
message sent to client 4
msg: helia:send 2 hello nikta

name: helia
cmd: send
message sent to client 5
msg: nikta:leave 2

name: nikta
cmd: leave
message sent to client 4
msg: nikta:send 2 bye!

name: nikta
cmd: send
wrong command
msg: nikta:send 2 bye

name: nikta
cmd: send

```

```

helia@helia-virtual-machine: ~/Desktop/OSlab/lab04/2
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ gcc client.c -o client -lpthread
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ ./client 8000 helia
connected...Hello message sent
join 1
join 2
nikta : hi helia

send 2 hello nikta
nikta left the group 2

```

```

helia@helia-virtual-machine: ~/Desktop/OSlab/lab04/2
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ gcc client.c -o client -lpthread
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/2$ ./client 8000 nikta
connected...Hello message sent
join 2
helia joined group 2

send 2 hi helia
helia : hello nikta

leave 2
send 2 bye!
wrong command
send 2 bye
you are not in group 2

```

- با دستور quit کلاینت به طور کل حذف می شود و دیگر سرور به او پاسخ نمی دهد.

```
quit  
disconnecting... 0  
  
join 2  
█
```

```
name: nikta  
cmd: quit  
█
```

برای این بخش من دو فرزند ایجاد کردم و برای هر کدام از آن‌ها یک خط لوله قرار دادم که از آن جهت خواندن و نوشتن استفاده کنند. همچنین تیکه کدی را جهت تبدیل حروف کوچک به بزرگ و بالعکس را قرار دادم. (با توجه به حروف اسکی تشخیص می‌دهیم)

```

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 int main() {
19     int fds1[2], fds2[2];
20     // int returnstatus1, returnstatus2;
21     int pid;
22     char pipelwritemessage[20] = "Helia";
23     char pipe2writemessage[20] = "Hashemipour";
24     char readmessage[20];
25     // returnstatus1 = pipe(fds1);
26
27     if (pipe(fds1) == -1) {
28         perror("Unable to create pipe 1 \n");
29         exit(EXIT_FAILURE);
30     }
31     // returnstatus2 = pipe(fds2);
32
33     if (pipe(fds2) == -1) {
34         perror("Unable to create pipe 1 \n");
35         exit(EXIT_FAILURE);
36     }
37     pid = fork();
38
39     if (pid != 0){ // Parent process {
40
41         close(fds1[0]); // Close the unwanted pipel read side
42         close(fds2[1]); // Close the unwanted pipe2 write side
43         gets(pipelwritemessage);
44
45         printf("Parent: Writing to pipe 1 Message : %s\n",
46 pipelwritemessage);
47         write(fds1[1], pipelwritemessage, sizeof(pipelwritemessage));
48         read(fds2[0], readmessage, sizeof(readmessage));
49         printf("Parent: Reading from pipe 2 Message : %s\n",
49 readmessage);
50     } else { //child process

```

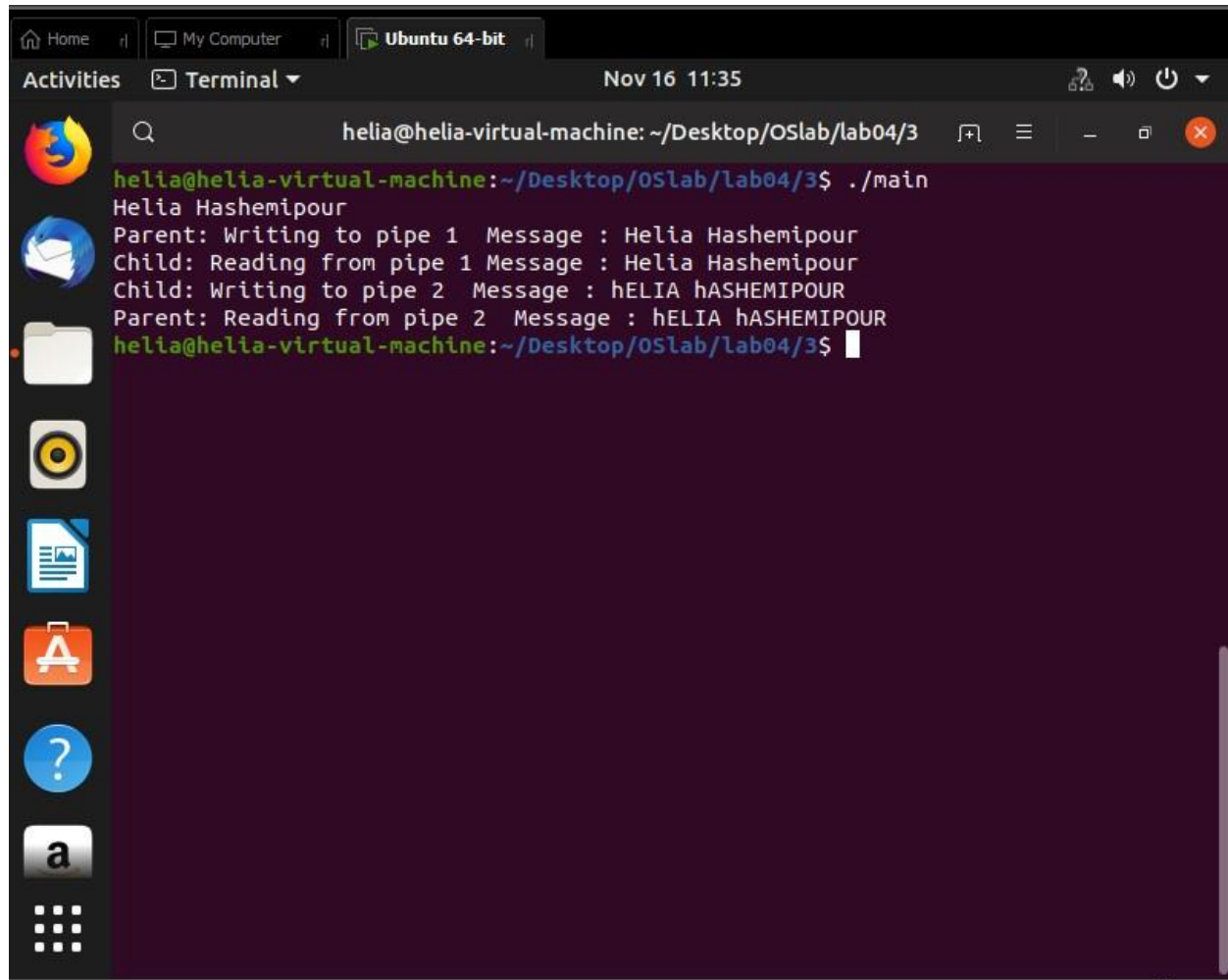
نمی‌توان خط لوله
را ساخت بنابراین
پیغام مناسب چاپ
می‌شود.

فرایند پدر

این قسمت
مربوط به
تبدیل به
حروف بزرگ
و یا کوچک
هست

```
50
51     close(fds1[1]); // Close the unwanted pipe1 write side
52     close(fds2[0]); // Close the unwanted pipe2 read side
53     read(fds1[0], readmessage, sizeof(readmessage));
54
55     printf("Child: Reading from pipe 1 Message : %s\n",
readmessage);
56     memcpy(pipe2writemessage, readmessage, sizeof(readmessage));
57     for (int i = 0; i<20; i++){
58         if ( pipe2writemessage [i] >= 65 &&
pipe2writemessage [i] <= 90)
59             pipe2writemessage[i] = pipe2writemessage[i]
+ 32;
60         else if (pipe2writemessage [i] >= 65+32 &&
pipe2writemessage [i] <= 90+32)
61             pipe2writemessage[i] = pipe2writemessage[i] - 32;
62
63
64     }
65
66
67     printf("Child: Writing to pipe 2 Message : %s\n",
pipe2writemessage);
68     write(fds2[1], pipe2writemessage, sizeof(pipe2writemessage));
69 }
70 return 0;
71 }
```

همانطور که میبینید فرآیند والد پیامی را می نویسد و در خط لوله قرار قرار می دهد و فرآیند فرزند آن را می خواند و تغییرات را روی حروف اعمال می کند و روی خط لوله قرار می دهد و فرآیند والد آن از روی خط لوله می خواند و کل این پروسه با استفاده از 2 خط لوله انجام می شود



```
helia@helia-virtual-machine: ~/Desktop/OSlab/lab04/3
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/3$ ./main
Helia Hashemipour
Parent: Writing to pipe 1 Message : Helia Hashemipour
Child: Reading from pipe 1 Message : Helia Hashemipour
Child: Writing to pipe 2 Message : hELIA hASHEMIPOUR
Parent: Reading from pipe 2 Message : hELIA hASHEMIPOUR
helia@helia-virtual-machine:~/Desktop/OSlab/lab04/3$
```