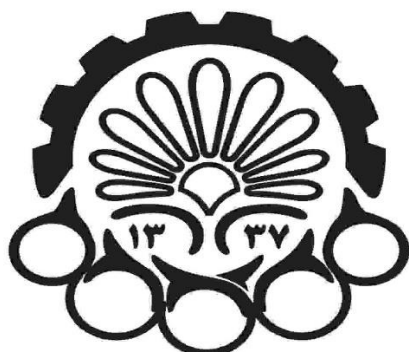


به نام خدا



**دانشگاه صنعتی امیرکبیر**  
( پلی تکنیک تهران )

دانشکده مهندسی کامپیوتر

آزمایشگاه سیستم های عامل

آزمایش چهارم : ارتباط بین پردازنده ها

اعضای گروه :

محمد امین فرح بخش - (40131029)

حسین تاتار - (40133014)

فروردین 1404

## تمرین امتیازی اسلاید 9 پاورپوینت )

برنامه ای بنویسید که : حافظه مشترکی ایجاد کند.

دسترسی آن را به خواندن فقط برای کاربر تغییر دهد.

بررسی کند که آیا نوشتن در حافظه پس از تغییر دسترسی ها امکانپذیر است یا خیر.

در این تمرین ما ابتدا یک فایل Q2.c می سازیم و کد اصلی را در آن قرار می دهیم.

```
C Q2.c  x
home > amin > C Q2.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/shm.h>
4  #include <sys/stat.h>
5  #include <unistd.h>
6  #include <string.h>
7  #include <errno.h>
8  int main() {
9      key_t key = IPC_PRIVATE;
10     int shm_id = shmget(key, 1024, IPC_CREAT | 0666);
11     if (shm_id == -1) {
12         perror("Failed to create shared memory");
13         exit(1);
14     }
15     printf("Shared memory created successfully\n");
16     char *shared_memory = (char *)shmat(shm_id, NULL, 0);
17     if (shared_memory == (char *)-1) {
18         perror("Failed to attach to shared memory");
19         exit(1);
20     }
21     printf("Attached to shared memory\n");
22     strcpy(shared_memory, "Initial message");
23     printf("Initial write to shared memory: %s\n", shared_memory);
24
25     struct shmid_ds shm_info;
26     if (shmctl(shm_id, IPC_STAT, &shm_info) == -1) {
27         perror("Failed to get shared memory info");
28         exit(1);
29     }
30     printf("Original permissions: %o\n", shm_info.shm_perm.mode);
31
32     shm_info.shm_perm.mode = 0444; // Read-only for user, group, others
33     if (shmctl(shm_id, IPC_SET, &shm_info) == -1) {
34         perror("Failed to change permissions");
35         exit(1);
36     }
37
38     if (shmctl(shm_id, IPC_STAT, &shm_info) == -1) {
39         perror("Failed to get shared memory info");
40         exit(1);
41     }
```

```

C Q2.c  x
home > amin > C Q2.c
8  int main() {
42  printf("New permissions: %o\n", shm_info.shm_perm.mode);
43  printf("Attempting to write to shared memory after changing permissions...\n");
44  int write_result = 0;
45
46  if (shmdt(shared_memory) == -1) {
47      perror("Failed to detach from shared memory");
48      exit(1);
49  }
50
51  shared_memory = (char *)shmat(shm_id, NULL, SHM_RDONLY);
52  if (shared_memory == (char *)-1) {
53      perror("Failed to reattach to shared memory");
54      exit(1);
55  }
56
57  errno = 0;
58  strcpy(shared_memory, "New message");
59
60  if (errno != 0) {
61      printf("Write failed: %s\n", strerror(errno));
62      write_result = -1;
63  } else {
64      printf("Write succeeded: %s\n", shared_memory);
65      write_result = 0;
66  }
67
68  if (shmdt(shared_memory) == -1) {
69      perror("Failed to detach from shared memory");
70      exit(1);
71  }
72
73  if (shmctl(shm_id, IPC_RMID, NULL) == -1) {
74      perror("Failed to delete shared memory");
75      exit(1);
76  }
77
78  printf("Test result: Writing to memory after changing permissions to read-only i
79  write_result == 0 ? "possible" : "not possible");
80

```

## توضیح کد :

این برنامه یک بخش حافظه مشترک با مجوزهای کامل ایجاد می کند (0666) یک پیام اولیه برای نشان دادن دسترسی به نوشتن می نویسد مجوزها را برای کاربر (و گروه/دیگران) به فقط خواندنی (0444) تغییر می دهد. برای اطمینان از اعمال تغییرات مجوز، آن را جدا می کند و دوباره به حافظه وصل می کند تلاش برای نوشتن در حافظه پس از تغییر مجوزها گزارش می دهد که آیا نوشتن هنوز پس از تغییر مجوز امکان پذیر است یا خیر این برنامه از تابع shmctl() با دستور IPC\_STAT استفاده می کند تا ابتدا مجوزهای فعلی را بازیابی کند، سپس از IPC\_SET برای تغییر آنها استفاده می کند. پس از تغییر مجوزها، نوشتن را در

بخش حافظه مشترک آزمایش می کند و نتیجه را گزارش می کند. هنگام اجرا، این برنامه نشان می دهد که نوشتن در حافظه مشترک پس از تغییر مجوزها به فقط خواندنی امکان پذیر نیست، که رفتار مورد انتظار در سیستم های یونیکس/لینوکس است. این برنامه شامل رسیدگی مناسب به خطا برای شناسایی مشکلاتی است که ممکن است در طول اجرا رخ دهد. هنگام اجرا، برنامه نشان می دهد که پس از تغییر مجوزها به فقط خواندنی، تلاش برای نوشتن در حافظه با یک خطای حفاظتی شکست می خورد و تأیید می کند که سیستم مجوز همانطور که انتظار می رود کار می کند. این یک سناریوی دنیای واقعی را شبیه سازی می کند که در آن یک فرآیند ممکن است حافظه مشترک را با مجوزهای محدود راه اندازی کند تا از یکپارچگی داده ها اطمینان حاصل کند، در حالی که به سایر فرآیندها اجازه می دهد داده های مشترک را بخوانند، اما تغییر ندهند.

تصویر خروجی :

```
amin@Frb:~$ gcc q2.c -o a.out
ccl: fatal error: q2.c: No such file or directory
compilation terminated.
amin@Frb:~$ gcc Q2.c -o a.out
amin@Frb:~$ ./a.out
Shared memory created successfully
Attached to shared memory
Initial write to shared memory: Initial message
Original permissions: 666
New permissions: 444
Attempting to write to shared memory after changing permissions...
Segmentation fault (core dumped)
amin@Frb:~$
```

**تمرین 1** شرایطی را تصور کنید که در آن شما دو فرآیند دارید، یک فرآیند تولید کننده و یک فرآیند مصرف کننده، که از طریق حافظه مشترک با هم ارتباط برقرار میکنند. فرآیند تولید کننده اعداد تصادفی را تولید میکند و آنها را در حافظه مشترک ذخیره میکند، در حالی که فرآیند مصرف کننده این اعداد را میخواند و مجموع آنها را محاسبه میکند.

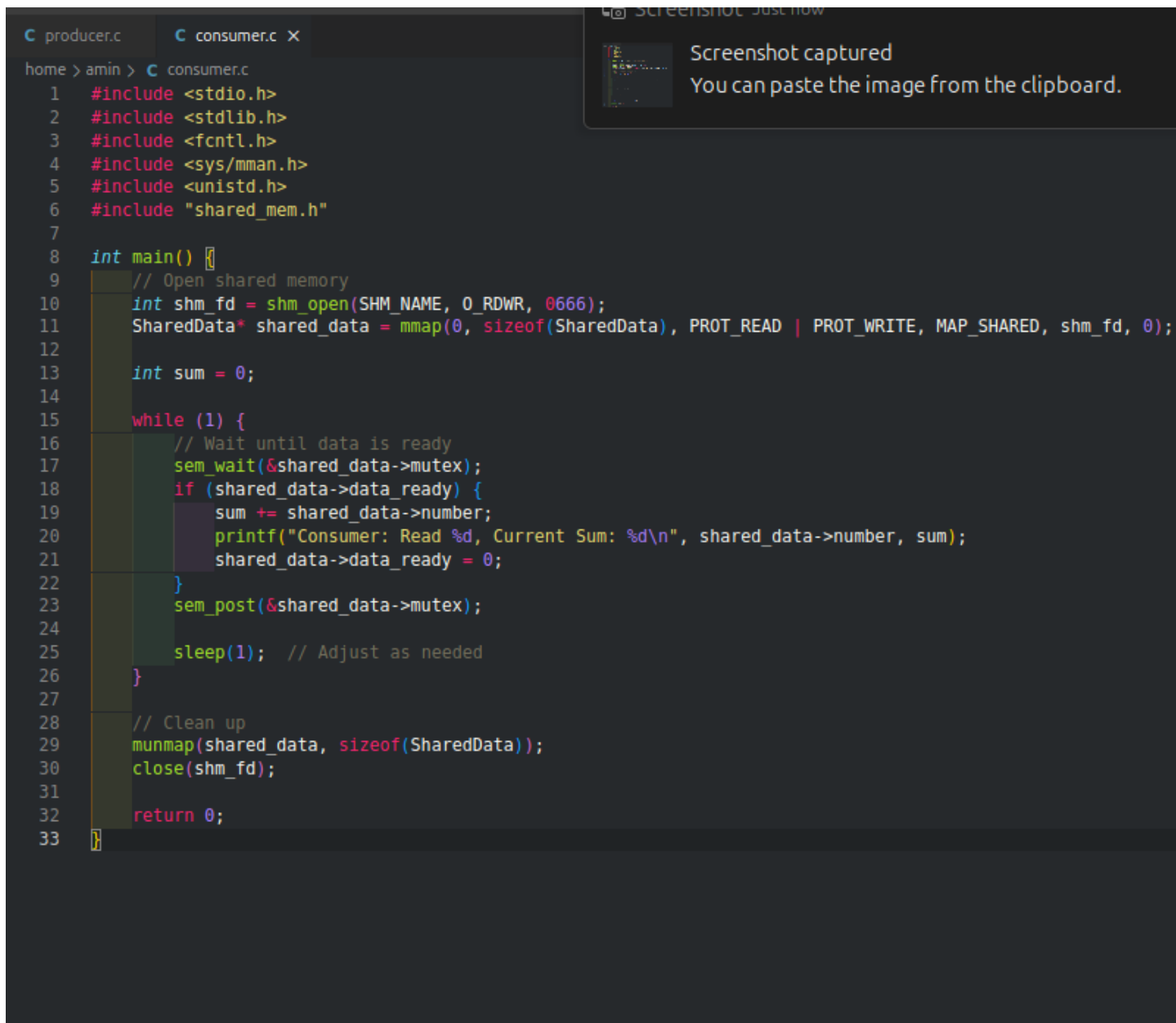
در ابتدا فایل `h.mem_shared` را تعریف میکنیم که شامل یک فایل هدر به زبان C است که شامل تعاریفی برای ارتباط حافظه مشترک بین فرآیندهای تولیدکننده و مصرف کننده است.

```
C shared_mem.h x
home > amin > C shared_mem.h
1  #ifndef SHARED_MEM_H
2  #define SHARED_MEM_H
3
4  #include <semaphore.h>
5
6  #define SHM_NAME "/shm_example" // Shared memory object name
7
8  typedef struct {
9      int number; // Random number produced
10     int data_ready; // Flag to signal data availability
11     sem_t mutex; // Semaphore for synchronization
12 } SharedData;
13
14 #endif
```

در فایل producer.c یک تولید کننده را تعریف میکنیم که بخش حافظه مشترک را ایجاد میکند و سپس اعداد تصادفی تولید میکند. این اعداد تصادفی را در حافظه مشترک ذخیره میکند. از semaphore که در لحظه فقط یک فرآیند به این داده مشترک دسترسی دارد.

```
C producer.c X C consumer.c
home > amin > C producer.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <fcntl.h>
4  #include <sys/mman.h>
5  #include <time.h>
6  #include <unistd.h>
7  #include "shared_mem.h"
8
9  int main() {
10     srand(time(NULL)); // Seed for random number generation
11
12     // Create shared memory
13     int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
14     ftruncate(shm_fd, sizeof(SharedData));
15     SharedData* shared_data = mmap(0, sizeof(SharedData), PROT_WRITE, MAP_SHARED, shm_fd, 0);
16
17     // Initialize semaphore and ready flag
18     sem_init(&shared_data->mutex, 1, 1);
19     shared_data->data_ready = 0;
20
21     while (1) {
22         // Generate random number
23         int random_number = rand() % 100; // Random number between 0 and 99
24
25         // Update for shared data to be ready
26         sem_wait(&shared_data->mutex);
27         shared_data->number = random_number;
28         shared_data->data_ready = 1;
29         printf("Producer: generated %d\n", random_number);
30         sem_post(&shared_data->mutex);
31
32         sleep(1); // Adjust as needed
33     }
34
35     // Cleanup
36     munmap(shared_data, sizeof(SharedData));
37     close(shm_fd);
38     shm_unlink(SHM_NAME);
39
40     return 0;
41 }
```

در فایل c.consumer یک مصرف کننده را تعریف کردیم. در ابتدا آن حافظه مشترک ایجاد شده توسط تولید کننده را باز میکند و چک میکند اگر عدد جدیدی نوشته شده بود آن را میخواند و با اعداد قبلی جمع زده و جمع آن را چاپ میکند.



```
C producer.c  C consumer.c X
home > amin > C consumer.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <fcntl.h>
4  #include <sys/mman.h>
5  #include <unistd.h>
6  #include "shared_mem.h"
7
8  int main() {
9      // Open shared memory
10     int shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
11     SharedData* shared_data = mmap(0, sizeof(SharedData), PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
12
13     int sum = 0;
14
15     while (1) {
16         // Wait until data is ready
17         sem_wait(&shared_data->mutex);
18         if (shared_data->data_ready) {
19             sum += shared_data->number;
20             printf("Consumer: Read %d, Current Sum: %d\n", shared_data->number, sum);
21             shared_data->data_ready = 0;
22         }
23         sem_post(&shared_data->mutex);
24
25         sleep(1); // Adjust as needed
26     }
27
28     // Clean up
29     munmap(shared_data, sizeof(SharedData));
30     close(shm_fd);
31
32     return 0;
33 }
```

Screenshot captured  
You can paste the image from the clipboard.

در دو ترمینال به صورت موازی هر دو را اجرا میکنیم و خروجی به صورت زیر خواهد بود:

```
amin@Frb: ~  
amin@Frb:~$ gcc consumer.c -o cons  
amin@Frb:~$ ./cons  
Consumer: Read 70, Current Sum: 70  
Consumer: Read 6, Current Sum: 76  
Consumer: Read 52, Current Sum: 128  
Consumer: Read 45, Current Sum: 173  
Consumer: Read 13, Current Sum: 186  
Consumer: Read 46, Current Sum: 232  
Consumer: Read 18, Current Sum: 250  
Consumer: Read 31, Current Sum: 281  
Consumer: Read 43, Current Sum: 324  
Consumer: Read 75, Current Sum: 399  
Consumer: Read 84, Current Sum: 483  
Consumer: Read 38, Current Sum: 521  
Consumer: Read 50, Current Sum: 571  
Consumer: Read 7, Current Sum: 578  
Consumer: Read 61, Current Sum: 639  
Consumer: Read 65, Current Sum: 704  
Consumer: Read 57, Current Sum: 761  
Consumer: Read 48, Current Sum: 809  
Consumer: Read 29, Current Sum: 838  
Consumer: Read 37, Current Sum: 875  
Consumer: Read 5, Current Sum: 880  
Consumer: Read 54, Current Sum: 934  
Consumer: Read 30, Current Sum: 964  
Consumer: Read 22, Current Sum: 986  
Consumer: Read 81, Current Sum: 1067  
Consumer: Read 10, Current Sum: 1077  
Consumer: Read 94, Current Sum: 1171  
Consumer: Read 26, Current Sum: 1197  
Consumer: Read 97, Current Sum: 1294  
Consumer: Read 6, Current Sum: 1300  
Consumer: Read 75, Current Sum: 1375  
Consumer: Read 66, Current Sum: 1441  
Consumer: Read 12, Current Sum: 1453  
^C  
amin@Frb:~$  
  
amin@Frb: ~  
amin@Frb:~$ gcc producer.c -o prod  
amin@Frb:~$ ./prod  
Producer: generated 6  
Producer: generated 52  
Producer: generated 45  
Producer: generated 13  
Producer: generated 46  
Producer: generated 18  
Producer: generated 31  
Producer: generated 43  
Producer: generated 75  
Producer: generated 84  
Producer: generated 38  
Producer: generated 50  
Producer: generated 7  
Producer: generated 61  
Producer: generated 65  
Producer: generated 57  
Producer: generated 48  
Producer: generated 29  
Producer: generated 37  
Producer: generated 5  
Producer: generated 54  
Producer: generated 30  
Producer: generated 22  
Producer: generated 81  
Producer: generated 10  
Producer: generated 94  
Producer: generated 26  
Producer: generated 97  
Producer: generated 6  
Producer: generated 75  
Producer: generated 66  
Producer: generated 12  
^C  
amin@Frb:~$
```

**تمرین 2** در حالت پایه برنامه، پیاده سازی به صورت یک معماری کلاینت-سرور انجام شده است. ابتدا سرور با استفاده از توابعی مانند socket و bind یک سوکت روی پورت مشخصی ایجاد میکند و سپس در انتظار اتصال کلاینت میماند. در این حالت، تنها یک کلاینت در هر لحظه میتواند به سرور متصل شود و دستورات را ارسال کند. سرور درخواست های کلاینت را پردازش کرده و اطلاعات کالاها را در یک لیست ذخیره میکند؛ این لیست شامل نام کالا و مقدار موجودی هر کالا است که در یک ساختار داده مانند آرایه ای از ساختارها ذخیره شده است. عملیات های مختلف مانند افزودن، کاهش، و حذف کالاها نیز در این ساختار پیاده سازی شده است. کلاینت پس از اتصال به سرور، دستورات مختلف را از کاربر دریافت کرده و به سرور ارسال میکند. سرور با پردازش هر دستور، نتیجه را به کلاینت برمیگرداند. به علاوه، در این حالت برای هر دستور خطاهای احتمالی مانند درخواست موجودی منفی یا درخواست برای کالاهایی که وجود ندارند، شناسایی شده و پیغام مناسب به کلاینت ارسال میشود.

کدهای مربوط به حالت پایه در server1.c و client1.c ذخیره شده اند.

اگر server1.c و client1.c را اجرا کنیم خروجی به این صورت خواهد بود :

```
amin@Frb: ~  
amin@Frb:~$ ./server1 8081  
Server listening on port 8081  
Received command: creat p1 2  
creat p1 2: creat p1 2  
Received command: create p1 2  
create p1 2: create p1 2  
Received command: add p2 3  
add p2 3: add p2 3  
Received command: add p1 3  
add p1 3: add p1 3  
Received command: remove p1 1  
remove p1 1: remove p1 1  
Received command: remove p1 2  
remove p1 2: remove p1 2  
Received command: reduce p1 2  
reduce p1 2: reduce p1 2  
Received command: list  
list: list  
Received command: add p3 4  
add p3 4: add p3 4  
  
amin@Frb:~$ ./client1 127.0.0.1 8081 client1  
Enter command: creat p1 2  
Server response: Error: Invalid command.  
  
Enter command: create p1 2  
Server response: Product created successfully.  
  
Enter command: add p2 3  
Server response: Error: Product not found.  
  
Enter command: add p1 3  
Server response: Product quantity updated successfully.  
  
Enter command: remove p1 1  
Server response: Error: Product must have zero quantity to be removed.  
  
Enter command: remove p1 2  
Server response: Error: Product must have zero quantity to be removed.  
  
Enter command: reduce p1 2  
Server response: Product quantity reduced successfully.  
  
Enter command: list  
Server response: Product: p1, Count: 3  
  
Enter command: add p3 4  
Server response: Error: Product not found.  
  
Enter command:
```

در حالت پیشرفته، برنامه با تغییراتی برای پشتیبانی از چندین کلاینت همزمان بهبود یافته است. سرور در این حالت با استفاده از توابعی مانند fork یا pthread، به ازای هر اتصال جدید، یک فرآیند یا رشته جدید ایجاد میکند تا بتواند همزمان چندین کلاینت را مدیریت کند. این امکان باعث میشود هر کلاینت بتواند به صورت جداگانه دستورات خود را ارسال کند و نتیجه را دریافت کند. علاوه بر این، سرور به جای نگهداری یک لیست مشترک، برای هر کلاینت لیستی جداگانه از کالاها و موجودی های آنها در نظر میگیرد که این لیست اختصاصی به هر کلاینت اجازه میدهد داده های خود را بدون تداخل با دیگران مدیریت کند. دستور جدید send نیز برای انتقال موجودی کالا از یک کلاینت به کلاینت دیگر پیاده سازی شده است؛ به این صورت که مقدار کالا از موجودی کلاینت فرستنده کسر و به موجودی کلاینت گیرنده اضافه میشود و در صورتی که کالا در انبار کلاینت گیرنده وجود نداشته باشد، به عنوان کالای جدید به آن اضافه میشود. برای جلوگیری از تداخل های احتمالی بین کلاینت ها، از قفل های همزمانی استفاده شده است تا دسترسی چند کلاینت به داده های مشترک بدون تداخل باشد و همچنین از ورود دو کلاینت با نام یکسان جلوگیری شود. این ساختار موجب میشود سرور بتواند به طور همزمان به چند کلاینت سرویس دهد و هر کلاینت به صورت مستقل داده های مربوط به خود را مدیریت کند. کدهای مربوط به حالت پیشرفته در client2.c و server2.c ذخیره شده اند.



در نهایت اگر server2.c و client2.c را اجرا کنیم خروجی به این صورت خواهد بود :

```
amin@Frb: ~  
amin@Frb:~$ ./server2 8082  
Server is listening on port 8082...  
  
amin@Frb:~$ ./client2 127.0.0.1 8082 client2  
Connected to server as client2  
Enter command: create p1 3  
Product created successfully.  
  
Enter command: create p2 4  
Product created successfully.  
  
Enter command: send client2 p2 2  
Product transferred successfully.  
  
Enter command: list  
Product: p1, Count: 3  
Product: p2, Count: 4  
  
Enter command: remove p1  
Error: Product must have zero quantity to be removed.  
  
Enter command: create p3  
Product created successfully.  
  
Enter command: remove p3  
Product removed successfully.  
  
Enter command: send client1 p2 -1  
Error: Invalid amount.  
  
Enter command: add p3 5  
Error: Product not found.  
  
Enter command: ^C  
amin@Frb:~$
```