BLG312E Homework 2 Report

Ahmet Secaettin Alıcı - 150190097

1 Introduction

There is scarce quantity of products and there are 5 kind of products. There are 3 customers who want to buy them. They may not have enough money, and there may not enough products left. Our program simulates this situation in digital world. To simulate this interesting event, we need power of concurrency, threads and processes, so may the God help us!

A customer's balance is not between 0-200 dollars like it has been asked in homework's pdf, instead it is 100-1000 dollars, and still they are poor:D but like this we can see more variation in resulting events.

2 About the Code

2.1 Thread

Most of the explaining is done in the comments in the source codes, but essential things is worth to repeat. We have customer and product struct, we have list named customers and products. We have one function void * order(void * cst) taking a pointer to void as an argument, after it converts this argument to type customer * via type casting. order function can reach customer's ordered item list, products that customer wants to but, via its argument. Threads use order() to change data structures that shared among threads, so order() should use locks to protect program from race condition in critical section. Each customer has been associated with a thread. When we open the program first we should enter the seed so events may unfold. Below you can see a scenario.

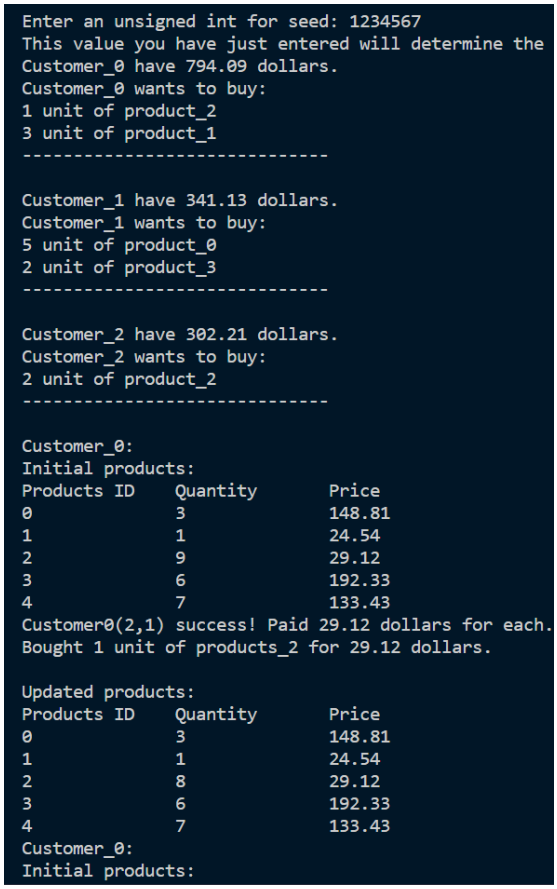

Figure 1: thread 1

```
Customer_0:
Initial products:
Products ID    Quantity       Price
0              3              148.81
1              1              24.54
2              8              29.12
3              6              192.33
4              7              133.43
Customer0(1,3) fail! Only 1 left in stock.

Updated products:
Products ID    Quantity       Price
0              3              148.81
1              1              24.54
2              8              29.12
3              6              192.33
4              7              133.43
------------------------------
Customer0
initial balance: 794.094193
updated balance: 764.979032
Ordered products:
id        quantity
2         1
1         3
Purchased products:
id        quantity
2         1
Customer_1:
Initial products:
Products ID    Quantity       Price
0              3              148.81
1              1              24.54
2              8              29.12
3              6              192.33
4              7              133.43
Customer1(0,5) fail! You are too poor! Ha ha:D

Updated products:
```

Figure 2: thread 2

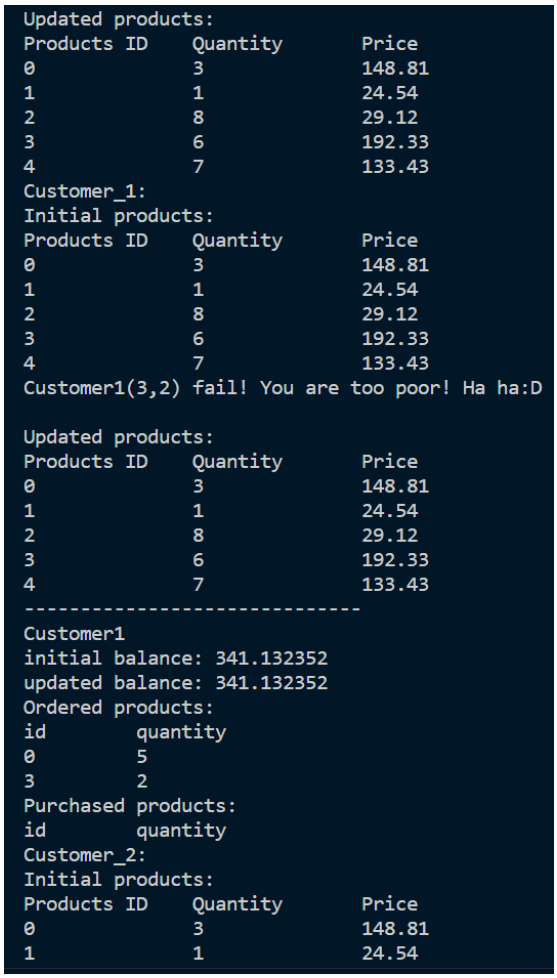

Figure 3: thread 3

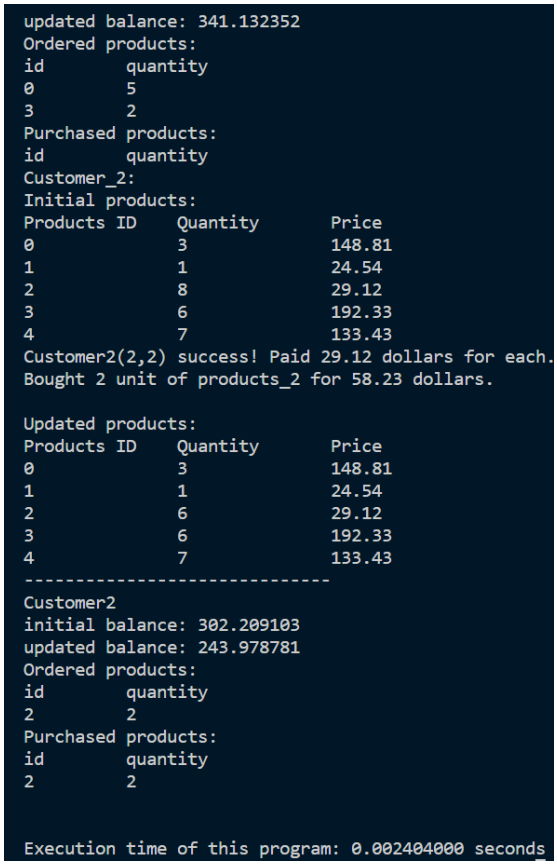

Figure 4: thread 4

2.2 Process

Most of the code here is same with the thread one. Differently, at first, we have new struct named globals, this globals struct contains customers', products' and semaphore (for locki binary semaphore) information. After we allocated a memory for globals and share it between processes we can continue similar to thread case. In main() we allocate a memory for it and we copy this memory to shared memory we have created by mmap function. We can reach this shared memory via a pointer named shmem in processes. We initialize for the data structures in the shared memory. For ease of understanding we print which customer intends to buy how many and which products. Each child process is associated with one customer. So customer have an attribute pid. In this process case, order function should take 2 arguments, one for customer and other for shared memory. order() takes shared memory as an arguments so it can reach there. Below is an example execution.

```
Enter an unsigned int for seed: 1234
This value you have just entered will determine the fate.
Customer_0 have 238.53 dollars.
Customer_0 wants to buy:
5 unit of product_3
2 unit of product_4
5 unit of product_0
-----------------------------

Customer_1 have 334.98 dollars.
Customer_1 wants to buy:
5 unit of product_0
-----------------------------

Customer_2 have 270.77 dollars.
Customer_2 wants to buy:
2 unit of product_1
1 unit of product_2
5 unit of product_0
-----------------------------

Customer_0:
Initial products:
Products ID    Quantity       Price
0              10             45.40
1              8              90.06
2              2              114.30
3              9              95.51
4              4              190.85
Customer0(3,5) fail! You are too poor! Ha ha:D

Updated products:
Products ID    Quantity       Price
0              10             45.40
1              8              90.06
2              2              114.30
3              9              95.51
4              4              190.85
Customer_0:
```

Figure 5: process 1

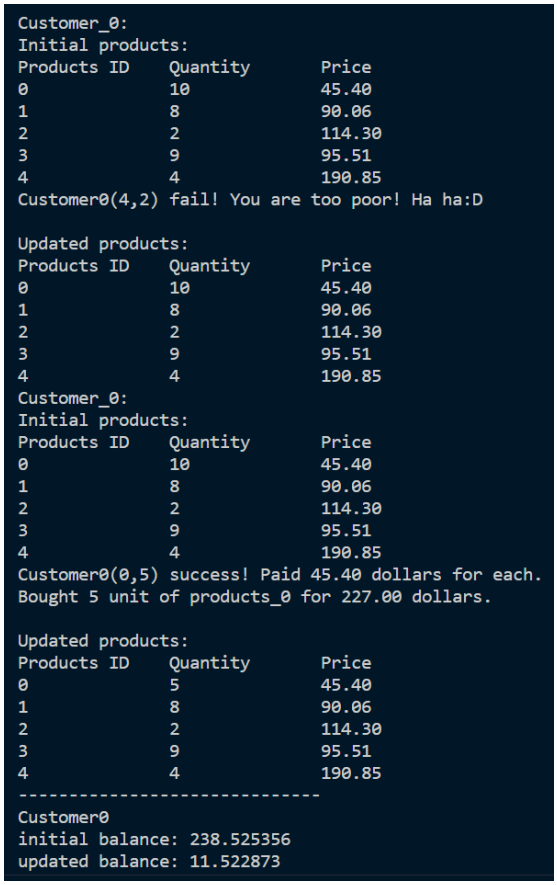

Figure 6: process 2

```
Customer0
initial balance: 238.525356
updated balance: 11.522873
Ordered products:
id        quantity
3         5
4         2
0         5
Purchased products:
id        quantity
0         5
Customer_2:
Initial products:
Products ID    Quantity       Price
0              5              45.40
1              8              90.06
2              2              114.30
3              9              95.51
4              4              190.85
Customer2(1,2) success! Paid 90.06 dollars for each.
Bought 2 unit of products_1 for 180.13 dollars.

Updated products:
Products ID    Quantity       Price
0              5              45.40
1              6              90.06
2              2              114.30
3              9              95.51
4              4              190.85
Customer_2:
Initial products:
Products ID    Quantity       Price
0              5              45.40
1              6              90.06
2              2              114.30
3              9              95.51
4              4              190.85
Customer2(2,1) fail! You are too poor! Ha ha:D
```

Figure 7: process 3

```
Updated products:
Products ID    Quantity       Price
0              5              45.40
1              6              90.06
2              2              114.30
3              9              95.51
4              4              190.85
Customer_2:
Initial products:
Products ID    Quantity       Price
0              5              45.40
1              6              90.06
2              2              114.30
3              9              95.51
4              4              190.85
Customer2(0,5) fail! You are too poor! Ha ha:D

Updated products:
Products ID    Quantity       Price
0              5              45.40
1              6              90.06
2              2              114.30
3              9              95.51
4              4              190.85
----------------------------
Customer2
initial balance: 270.766595
updated balance: 90.638016
Ordered products:
id        quantity
1         2
2         1
0         5
Purchased products:
id        quantity
1         2
Customer_1:
Initial products:
Products ID    Quantity       Price
```

Figure 8: process 4

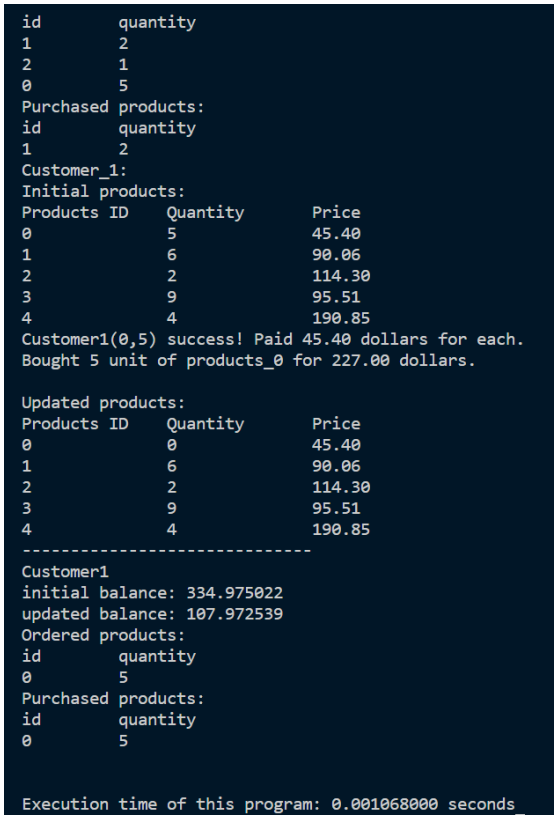

Figure 9: process 5

3 Discussion

- According to our printed cases, we can see multiprocessing is faster than multithreading. Multiprocessing finished the execution in 0.001068000 seconds, multithreading finished the execution in 0.002404000 seconds. I do not understand exactly why but clearly, if there is a small number of active customers multiprocessing is faster.

- As active customer number grows, context switch cost become more visible. Cost of context switch in processes is much more than threads because threads share address space. For example in the case of 10000 active customer, multithreading will be better than multiprocessing case because of the cost of context switch.