BLG312E - Computer Operating Systems - HW2

Enis Teper - 504221511

1 Introduction

In this homework, it is required from us to implement simultaneously working online shopping process for customers. In order to achieve it, multiprocessing and multithreading methods in operating systems will be used. Also, mutexes will be used to prevent updating product stocks at the same time (race condition) which may result in errors.

2 Multiprocessing

Multiprocessing is a parallel computing technique that enables to run different or same part of codes to be executed in different sub-tasks simultaneously. In C language fork() operation is used to implement this approach. When fork() function is called it generates a copy of a parent process including its memory space and program counter. Pseudo-code of how the code is implemented is given below.

Algorithm 1 Multiprocessing

```
1: define customers, products and mutex
2: function PROCESS_CUSTOMER(customer)
3:     for all order in customer_orders do
4:         lock(mutex)
5:         condition1 = ordered product quantity <= product quantity in stock
6:         condition2 = total product price <= customer balance
7:         if condition1 and condition2 is satisfied then
8:             update product quantity in stock and customer balance
9:             add order to purchased items of customer
10:            print initial and updated product information
11:        else
12:            print the reason of the failure
13:        end if
14:        unlock(mutex)
15:    end for
16: end function
17: function MAIN
18:    initialize customers, products and mutex
19:    create multiple sub-processes representing customers
20:    for all sub-processes do
21:        execute process_customer
22:    end for
23:    wait for all sub-processes to finish shopping process
24:    for all sub-processes do
25:        print customer initial and updated information
26:    end for
27: end function
```

The code is simplified as much as possible to let it human-readable. Products information is initalized as shared, because of the purchase that is done from the child process should affect other processes. Similarly shared counter is defined to make all processes to wait until all of them finish shopping processes. It is done to print customer information at last.

3 Multithreading

Multithreading is a parallel computing technique that allows different or the same parts of code to be executed concurrently within separate threads in a single process. Threads share the same memory space and resources of the parent process, making it more lightweight compared to multiprocessing. In C language, the pthread library is used to implement this approach. When a thread is created using pthread_create() function, it starts executing a specific function with its own stack and program counter. Pseudo-code of how the code is implemented is given below.

Algorithm 2 Multithreading

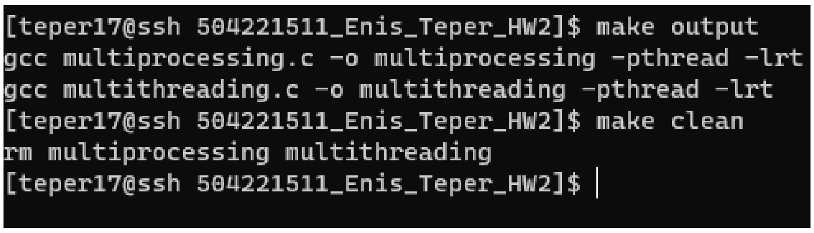
```
define customers, products and mutex
function PROCESS_CUSTOMER(customer)
    for all order in customer_orders do
        lock(mutex)
        condition1 = ordered product quantity <= product quantity in stock
        condition2 = total product price <= customer balance
        if condition1 and condition2 is satisfied then
            update product quantity in stock and customer balance
            add order to purchased items of customer
            print initial and updated product information
        else
            print the reason of the failure
        end if
        unlock(mutex)
    end for
end function
function MAIN
    initialize customers, products and mutex
    create multiple threads representing customers
    for all threads do
        execute process_customer
    end for
    for all customer in customers do
        print customer initial and updated information
    end for
end function
```

Since threads use same memory space it was not required to define products as shared memory. Moreover, since do not work like sub-processes there was no need to define shared counter to make all shopping processes finish.

4 Results

4.1 Makefile

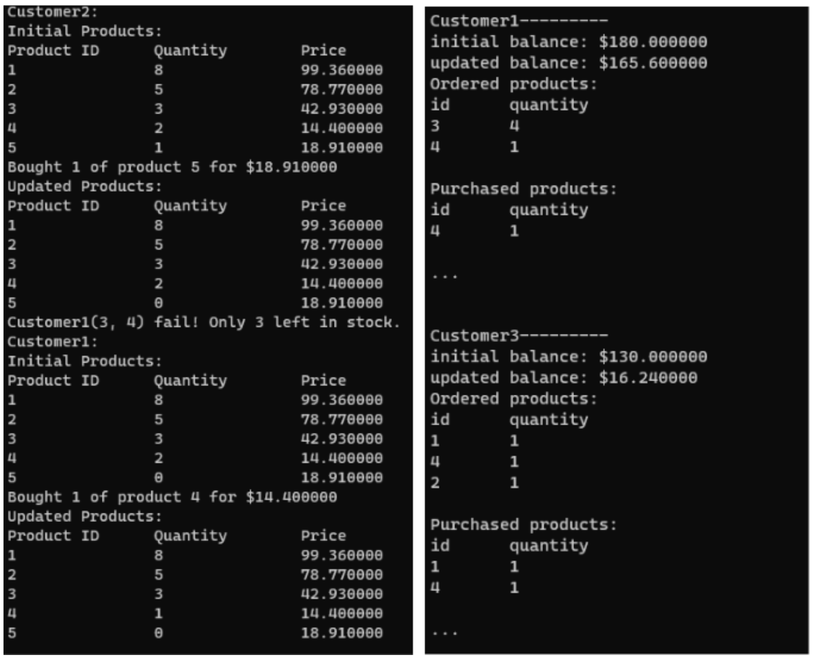
Makefile has two main commands, **make output** is used to compile the code and **make clean** to remove executable files.



```
[teper17@ssh 504221511_Enis_Teper_HW2]$ make output
gcc multiprocessing.c -o multiprocessing -pthread -lrt
gcc multithreading.c -o multithreading -pthread -lrt
[teper17@ssh 504221511_Enis_Teper_HW2]$ make clean
rm multiprocessing multithreading
[teper17@ssh 504221511_Enis_Teper_HW2]$ 
```

Figure 1: Makefile Commands

4.2 Multiprocessing

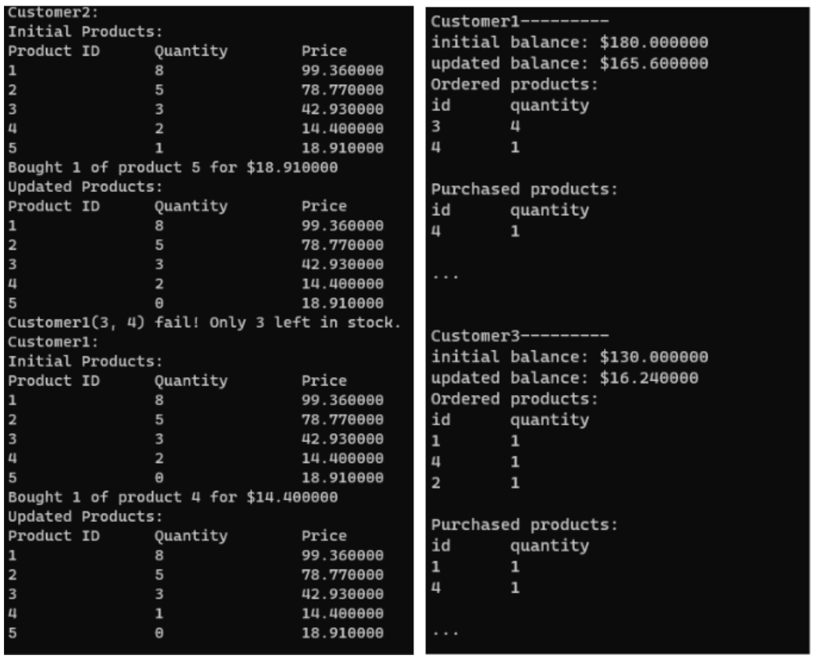


```
Customer2:
Initial Products:
Product ID      Quantity        Price
1               8               99.360000
2               5               78.770000
3               3               42.930000
4               2               14.400000
5               1               18.910000
Bought 1 of product 5 for $18.910000
Updated Products:
Product ID      Quantity        Price
1               8               99.360000
2               5               78.770000
3               3               42.930000
4               2               14.400000
5               0               18.910000
Customer1(3, 4) fail! Only 3 left in stock.
Customer1:
Initial Products:
Product ID      Quantity        Price
1               8               99.360000
2               5               78.770000
3               3               42.930000
4               2               14.400000
5               0               18.910000
Bought 1 of product 4 for $14.400000
Updated Products:
Product ID      Quantity        Price
1               8               99.360000
2               5               78.770000
3               3               42.930000
4               1               14.400000
5               0               18.910000
```

```
Customer1---------
initial balance: $180.000000
updated balance: $165.600000
Ordered products:
id      quantity
3       4
4       1

Purchased products:
id      quantity
4       1

...


Customer3---------
initial balance: $130.000000
updated balance: $16.240000
Ordered products:
id      quantity
1       1
4       1
2       1

Purchased products:
id      quantity
1       1
4       1

...
```

Figure 2: Multiprocessing Instance

4.3 Multithreading



```
Customer2:
Initial Products:
Product ID      Quantity        Price
1               8               99.360000
2               5               78.770000
3               3               42.930000
4               2               14.400000
5               1               18.910000
Bought 1 of product 5 for $18.910000
Updated Products:
Product ID      Quantity        Price
1               8               99.360000
2               5               78.770000
3               3               42.930000
4               2               14.400000
5               0               18.910000
Customer1(3, 4) fail! Only 3 left in stock.
Customer1:
Initial Products:
Product ID      Quantity        Price
1               8               99.360000
2               5               78.770000
3               3               42.930000
4               2               14.400000
5               0               18.910000
Bought 1 of product 4 for $14.400000
Updated Products:
Product ID      Quantity        Price
1               8               99.360000
2               5               78.770000
3               3               42.930000
4               1               14.400000
5               0               18.910000
```

```
Customer1---------
initial balance: $180.000000
updated balance: $165.600000
Ordered products:
id      quantity
3       4
4       1

Purchased products:
id      quantity
4       1

...


Customer3---------
initial balance: $130.000000
updated balance: $16.240000
Ordered products:
id      quantity
1       1
4       1
2       1

Purchased products:
id      quantity
1       1
4       1

...
```

Figure 3: Multithreading Instance

5 Conclusion

5.1 Which one of the two methods works faster? Explain.

As can be seen at table below, multiprocessing approach is slower than the multithreading approach. The reason might be related to process creation by copying exactly the parent process. Since multithreading approach uses shared memory instead of copying the parent process and usage of mutexes decreases parallelism

Table 1: Time Results of Algorithms

Algorithm	Time Consumed (seconds)
Multiprocessing	0.001747
Multithreading	0.001043

5.2 Assuming there are simultaneously 10000 active customers in the shopping system you implemented, which methodology (multiprocessing vs multithreading) is more efficient to use? Why?

Both methods have their own advantages and disadvantages. In multiprocessing because of copying the memory space will result in overhead, it would not be suitable until the computer has large enough RAM. Since, multithreading approach uses same shared memory space, it uses RAM efficiently. However, since multithreading uses same shared memory instead of independent copies multiprocessing utilizes parallelism more efficiently. Even though the computer has enough RAM and achieves better parallelism with multiprocessing it is not efficient way to generate exact copies of thousands of products and customers. Additionally, because of the race condition problem usage of mutex decreasing parallelism utilization already. In conclusion, because of mutexes parallelism utilization drops significantly and multithreading uses memory much more efficiently, so multithreading based aproach is more efficient.