Computer Operating Systems
Assignment 2

Deadline May 6, 2023 @23:59

I. Introduction

The subject of this assignment is to design an online shopping routine using multiprocessing and multithreading concepts in operating systems (OS). In an online shopping routine, more than one customer can order at the same time and this can cause a problem called race condition. When more than one customer wants to purchase the same product at the same time, this flow should be managed with a proper synchronization mechanism avoiding any race condition. Otherwise, the program may have some unexpected/inconsistent outputs. For instance, two customers may order product1 which is n in stock initially. They may reach and modify the same product (shared resource) concurrently and both processes/threads read the stock as n, and the count of the product may be decreased just one time instead of two times. To prevent race condition in OS, mutual exclusion methods are used. In this assignment, you are asked to perform a properly working online shopping routine using *mutexes*.

Sample structures for customer and product are given as follows:

struct Customer

customer_ID	Unique value for identifying each customer (Start from 1 and increment by one)
customer_Balance	Customer's available money in their account (between 0 – 200$).
ordered_Items	The list of products to be purchased.
purchased_Items	The list of all items purchased in an order (*with the quantity of each product).
*Add more members if needed.	

struct Product

product_ID	Unique value for identifying each product.
product_Price	Price of the product (between 1 – 200$).
product_Quantity	Quantity of the products in stock (1-10 for each).
*Add more members if needed.	

NOTE: You can randomly generate the *balance*, *price* and *quantity* values according to given ranges above.

II. Implementation Rules

- A purchase request (ordered_Items) is sent by specifying the products to be purchased and their quantities.
- If the desired product quantity is in stock, it can be purchased if the budget (balance) is sufficient.
- A customer can order at most five items (quantity) of the same product.
- More than one customer can order at the same time for the same product.
- A customer’s balance can never be less than 0. In other words, if the price of the product to be purchased is more than the balance, the purchase will not be accepted.
- A customer can order a <u>*single product*</u> *(Scenario 1)* or <u>*list of products*</u> *(Scenario 2)* at the same time.
    - o Ordering single product at a time: order_product(product_id,quantity)
    - o Ordering a list of products at a time: order_products (find out a proper way to pass arguments)
    - o Be careful that in *Scenario 2*, the customer may buy all/several or none of the products ordered. For example, if customer1 orders 5 of product1 and 3 of product2 whose stocks are 4 and 3, respectively, customer1 can buy only 3 of product2.
- Once the purchases are finalized, the total balance, purchased products list, and inventory are updated (i.e. the purchase price is deducted from the total balance, the purchased quantity is deducted from the stock for each product).
- There must be at least 3 customers and 5 products created in your codes.

III. Output

At the end of the shopping process,

- Print out the **initial balance**, **updated balance**, **ordered_Items** and **purchased_Items** for each customer.
- Print out the **initial products information** (id, quantity, price) and **updated products information** (id, quantity, price) after each purchase.

Sample output:

Note that the following examples are for the customers who order just one product of the given quantity at a time. (e.g. Customer1(3,4) means that *customer1* ordered 4 of product3).

```
Customer 1:
Initial products:
Product ID    Quantity    Price
1             9           99.36
2             5           78.77
3             5           42.93
4             3           14.40
5             1           18.91
Bought 4 of product 3 for $171.72

Updated products:
Product ID    Quantity    Price
1             9           99.36
2             5           78.77
3             1           42.93
4             3           14.40
5             1           18.91
```

```
Customer1(3,4) success!  Paid $42.93 for each.
Customer2(3,2) fail!  Only 1 left in stock.
Customer1(4,1) fail!  Insufficient funds.

Customer1----------
initial balance: $180
updated balance: $8.28
Ordered products:
id      quantity
3       4
4       1

Purchased products:
id      quantity
3       4

...

Customer2----------
initial balance: $150
updated balance: $150
Ordered products:
id      quantity
3       2

Purchased products:
id      quantity
-       -
```

IV. Implementation Details

In this assignment, you can create multiple processes and threads that represent customers (e.g. *customer1*, *customer2*, *customer3*, …). You may create a customer class and produce multiple processes/threads as instances. The customer_ID is a unique integer that starts from 1 and increases by one. The customer_Balance a is double which is assigned in the range of [1, 200]. You need to create at least 3 customer instances for each method.

The products will be created at the beginning of the program so customers can buy any of them. Again, you can follow the class-instances pattern for product creation. The *product_id* is a unique integer that starts from 1 and increases by one. The *product_Price* is a double which will be randomly assigned in the range of [1,200]. The product_Quantity is an integer which is also randomly assigned between 1 to 10. You need to create at least 5 product instances.

V. Submission Rules

You should upload a zip archive file containing the following items:

i. Your report that explains your code and contains a discussion section where you will answer the following questions:

- Which one of the two methods works faster? Explain. (You can prove your answer by adding a code snippet which calculates the execution time of each program)
- Assuming there are simultaneously 10000 active customers in the shopping system you implemented, which methodology (multiprocessing vs multithreading) is more efficient to use? Why?

ii. Source files containing your C programs.

iii. **makefile** for each program (multiprocessing vs multithreading).

- Under the zip folder you can separate multithreading and multiprocessing as two main folders.
- Prepare your report in LaTeX format to get full marks. You can use the following template: https://www.overleaf.com/read/dqshxtrpthgh
- Your report must not exceed 5 pages.
- Any form of **plagiarism** will not be tolerated. You must code by yourself in a neat way and explain the details, so that there are no unclear parts left in your code.
- You must implement your solution in the C programming language and make sure that it works on a linux environment.

For any queries, please contact T.A. ayvaz18@itu.edu.tr