OS E-Commerce Project

Subhajeet Lahiri (IMT2021022)

An E-Commerce Platform for phones with both the front-end and the back-end written in C. The interface on the front-end is text-based. In the backend, a file-based datastore is used, along with flocks and semaphores for concurrency control.

🔗 Github

HELIOSPOOK (https://github.com/Heliospook/OS-E-Commerce-Project.git)

If the badge doesn't work, use :
https://github.com/Heliospook/OS-E-Commerce-Project.git

Demo

Getting the executables ready

The project has been supplied with a makefile.
Run the makefile in the root directory as follows to create the executables :

```
make clean
make main
```

Running the server

Get the server running using :

```
./server
```

Connecting with clients

To connect a client to the running server, open a new terminal and run the client executable.

```
./client
```

Features

Symmetric admin/ user interface

Both of the above interfaces can be accessed by running the `client` executable. The server only interacts with the user through the client.
Whether a user is an admin or not is decided by the isAdmin property of the user object.
While logging in, the user is expected to enter the username and password. Some pre-created logins are :
- username : me, password : pass (admin)
- username : you, password : pass (customer)
- username : another, password : pass (customer)

These can be used to test the application.

Concurrency control

Several different types of concurrency control mechanisms have been used throughout the application

Login

The list of all users has been stored in the users.dat file. To prevent issues arising out of simultaneous access by the same user, **the corresponding record for the user is placed under a write lock(flock) while the user is logged in.**

Checkout

Semaphores have been used to address concurrency issues arising out of multiple people trying to purchase the same product. In my implementation,
Semaphore value = No. of products left in stock
Each product has a corresponding semaphore attached to it. A user is allowed to **lock** a product by subtracting the reqd. quantity, if the available **unlocked** quantity is adequate, as indicated by the semaphore value.

Admin updation

When the admin updates a product, it is placed under a total lock by decreasing the semaphore value to 0. When the admin is done, the value is updated to the new quantity.

File I/O

To ensure max. concurrency, changes are synced to files as soon as possible, perhaps at the cost of efficiency. However, this is necessary to have consistent data across multiple sessions by various customers and admins.

Client-Server Model

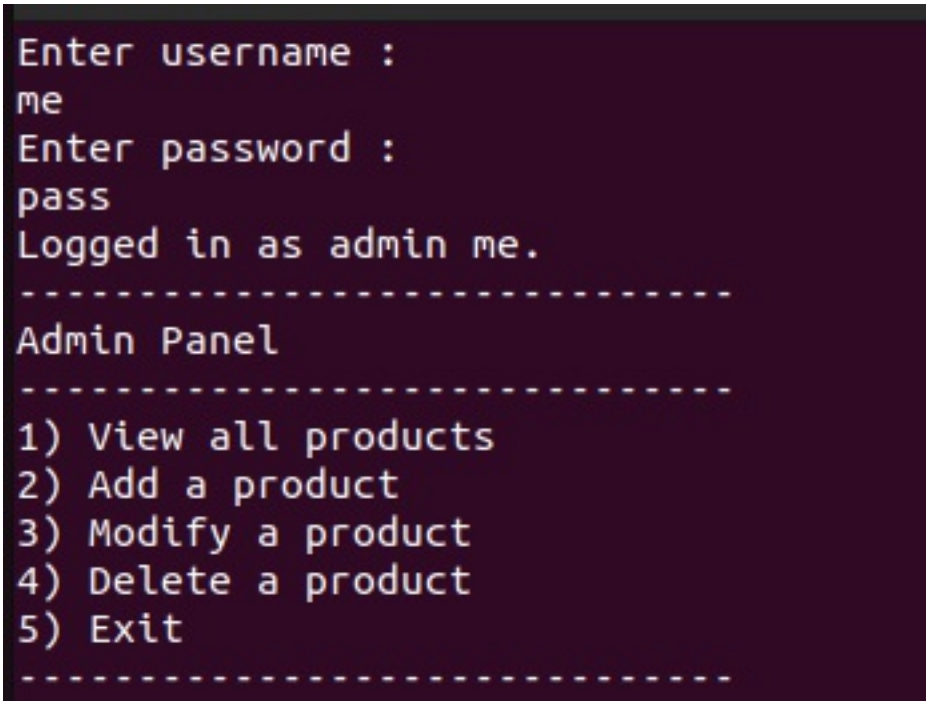
The concurrent server is set-up by spawning **child processes** when a new client connects. The client-server model is respected by ensuring that none of the computation/data-handling is done in the client. The client uses the ux.h header file to render the interface and relays everything back to the server for processing. The server calls the functions in the utilities.h file to perform the required operations. To ensure that the client and the server communicate on the same terms, the headers.h file is used by both of them.

Execution

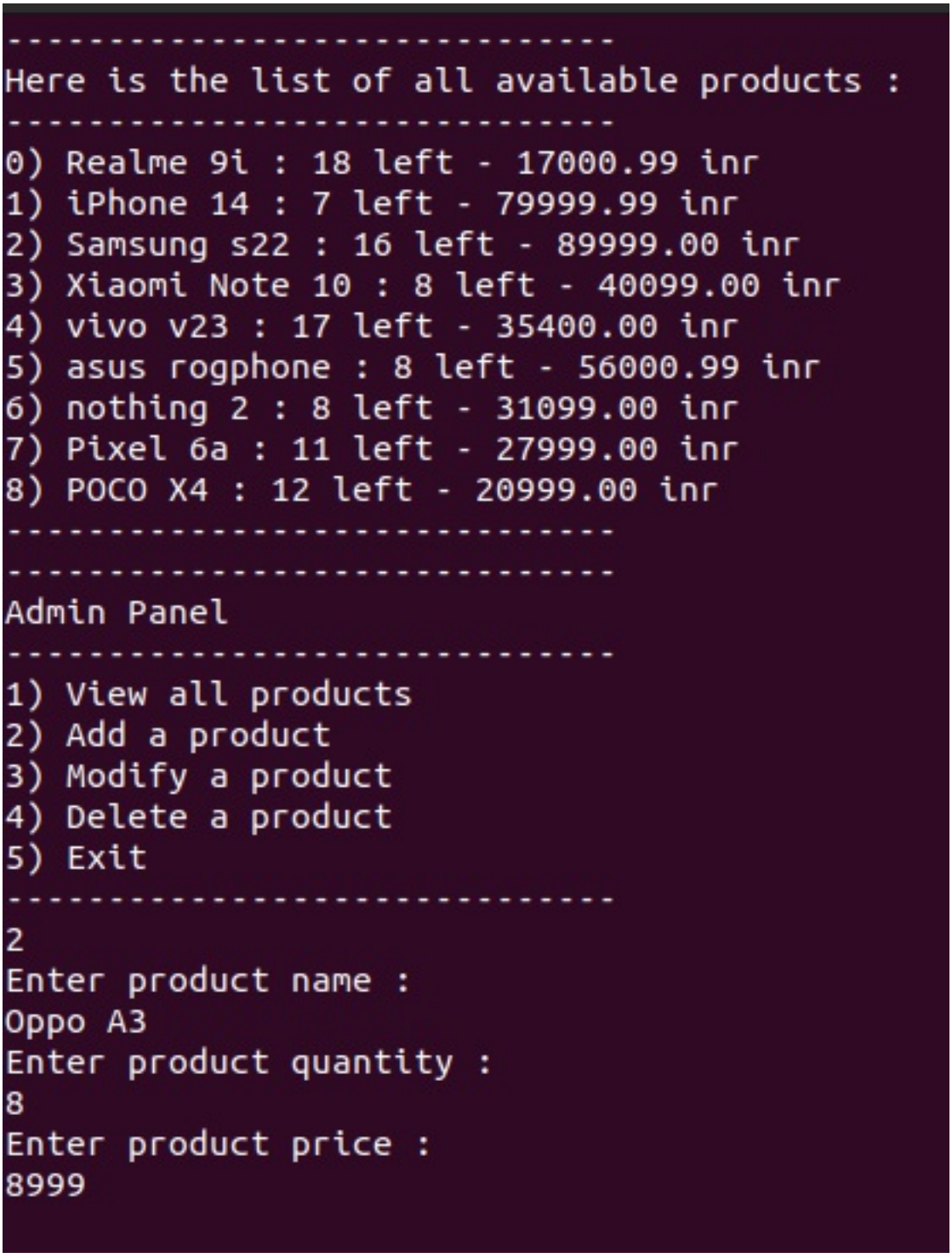
Here are some screenshots of the execution of the client.

Admin

Logging in as admin



View all products



Creating a product

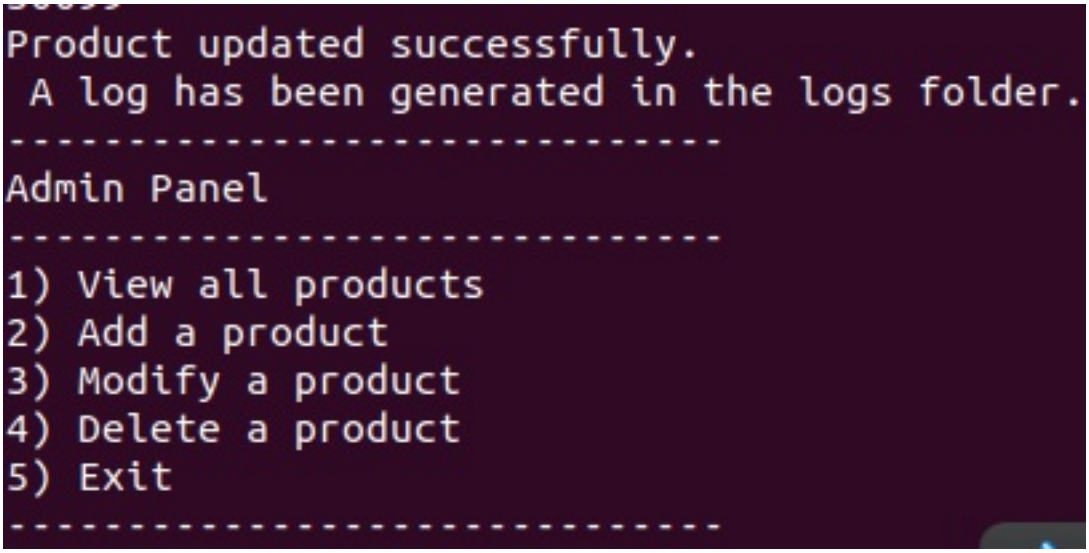
```
2
Enter product name :
Oppo A3
Enter product quantity :
8
Enter product price :
8999
------------------------------
Pdt. created successfully with id = 9
 A log has been generated in the logs folder.
------------------------------
Admin Panel
------------------------------
1) View all products
2) Add a product
3) Modify a product
4) Delete a product
5) Exit
------------------------------
```

Deleting a product

```
Enter the id of the product to be deleted :
9
Product deleted successfully.
 A log has been generated in the logs folder.
------------------------------
Admin Panel
------------------------------
1) View all products
2) Add a product
3) Modify a product
4) Delete a product
5) Exit
------------------------------
```

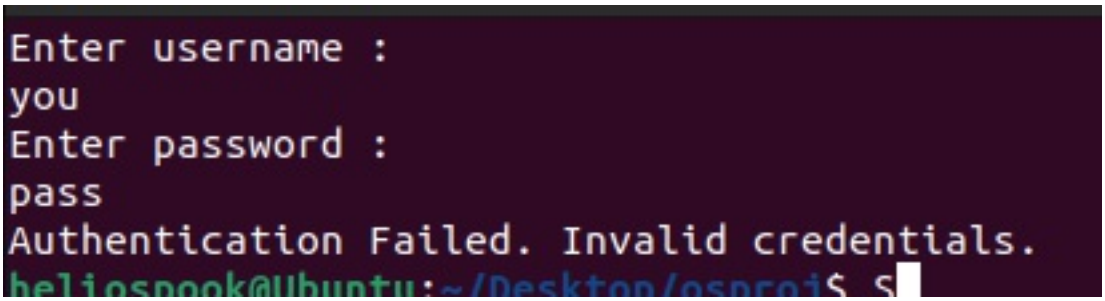
Updating a product

```
------------------------------
Here is the list of all available products :
------------------------------
0) Realme 9i : 18 left - 17000.99 inr
1) iPhone 14 : 7 left - 79999.99 inr
2) Samsung s22 : 16 left - 89999.00 inr
3) Xiaomi Note 10 : 8 left - 40099.00 inr
4) vivo v23 : 17 left - 35400.00 inr
5) asus rogphone : 8 left - 56000.99 inr
6) nothing 2 : 8 left - 31099.00 inr
7) Pixel 6a : 11 left - 27999.00 inr
8) POCO X4 : 12 left - 20999.00 inr
9) Oppo A3 : 8 left - 8999.00 inr
------------------------------
------------------------------
Admin Panel
------------------------------
1) View all products
2) Add a product
3) Modify a product
4) Delete a product
5) Exit
------------------------------
3
Enter the id of the product to be updated :
6
Enter product quantity :
12
Enter product price :
30099
```



Customer

If a customer is already logged in, and we try from another terminal using the same credentials, then we get an error.



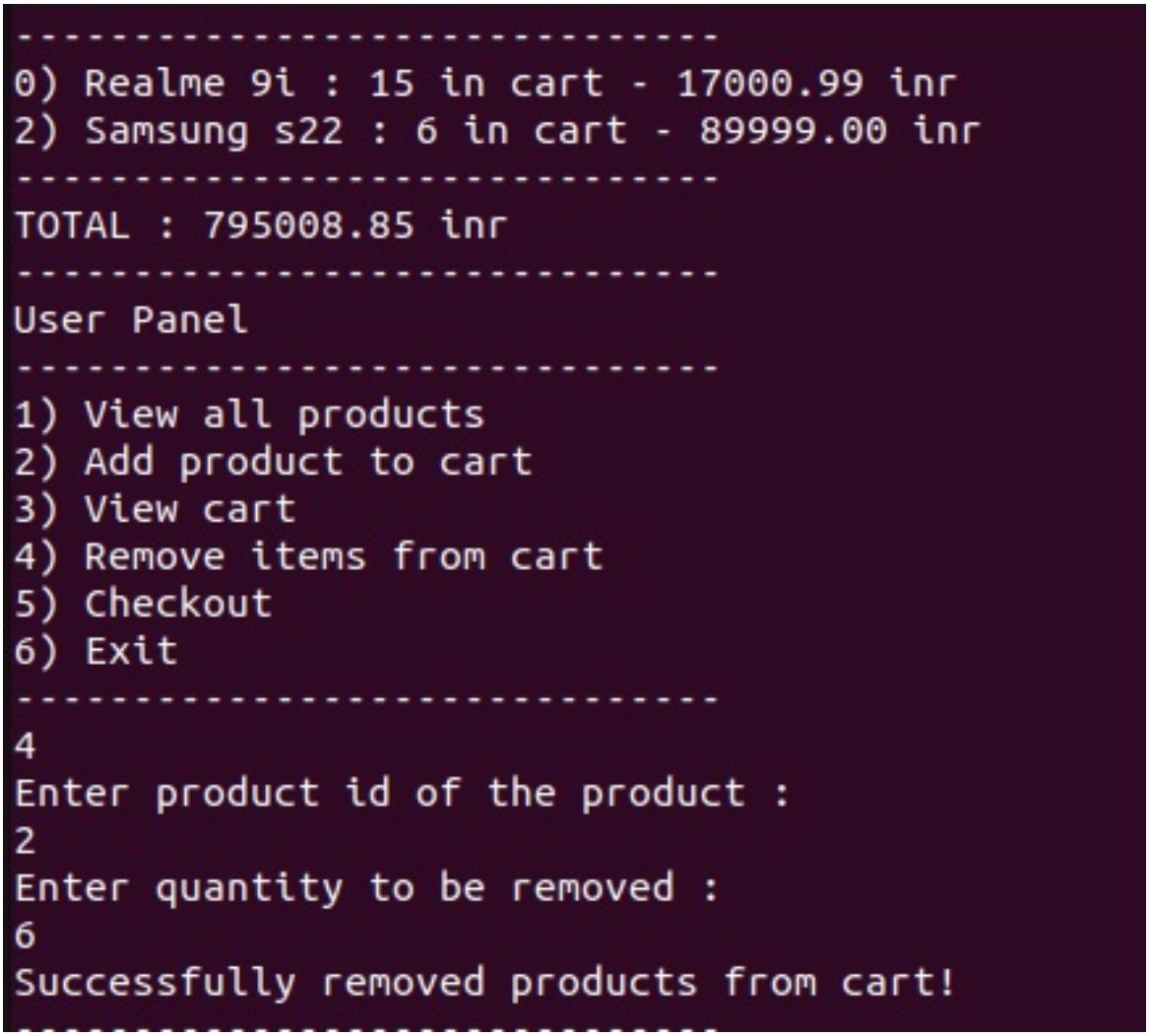
Adding products to cart

```
Enter product id to be added to cart :
0
Enter product quantity to be added to cart :
1
Successfully added products to cart!
------------------------------
User Panel
------------------------------
1) View all products
2) Add product to cart
3) View cart
4) Remove items from cart
5) Checkout
6) Exit
------------------------------
```

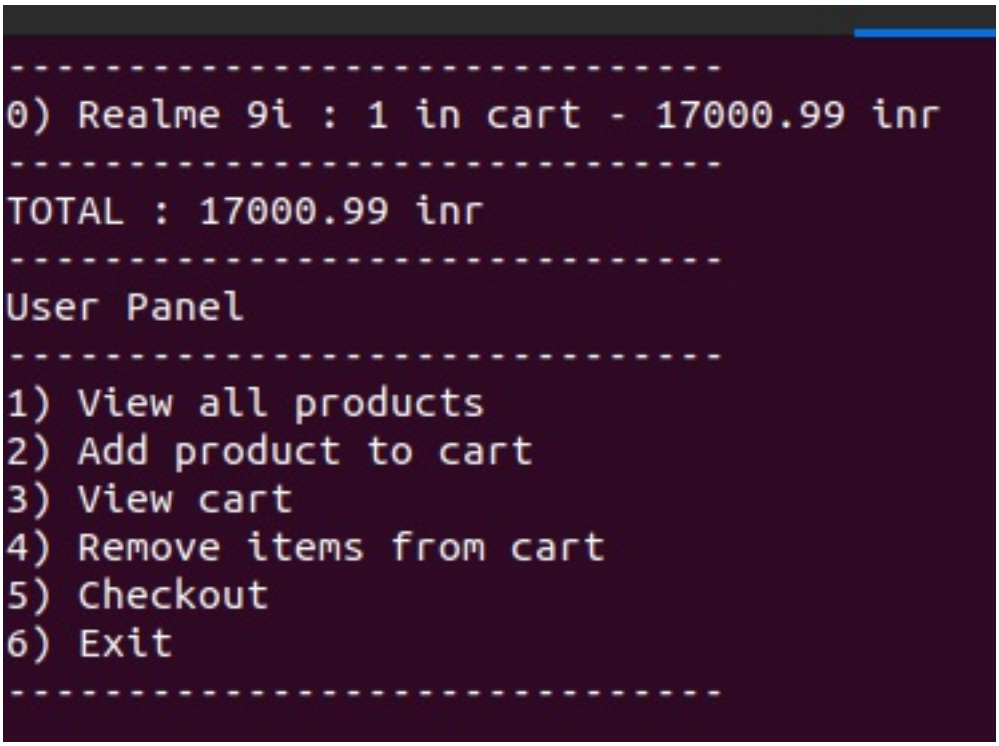
Viewing the cart

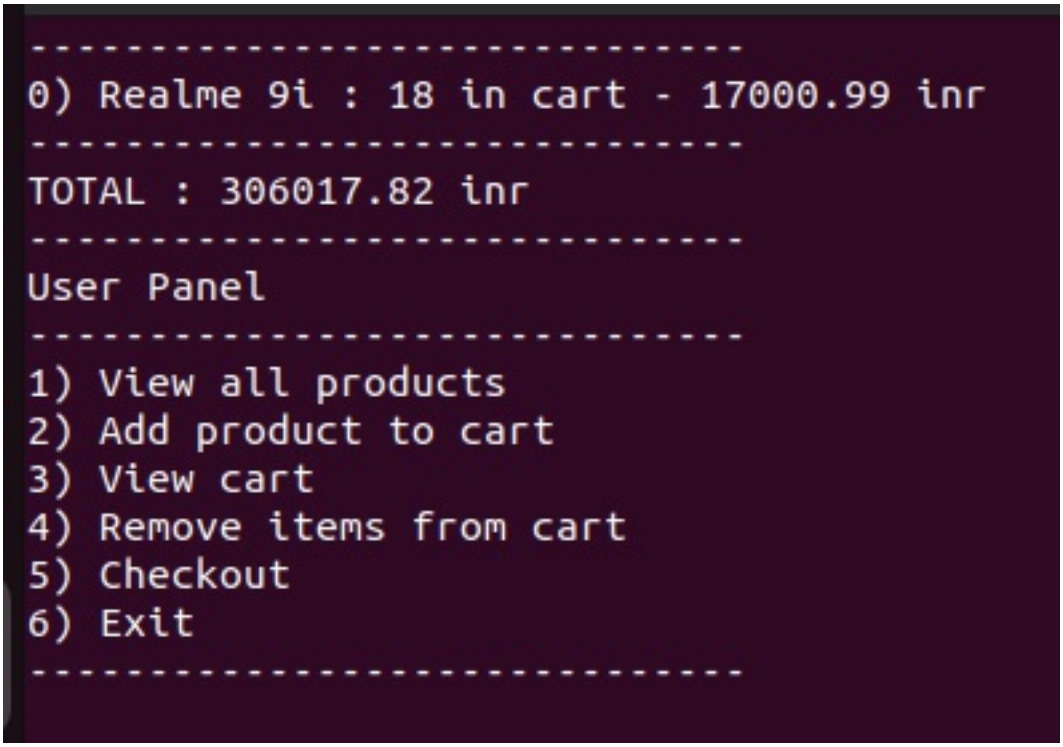
```
------------------------------
3
------------------------------
0) Realme 9i : 1 in cart - 17000.99 inr
------------------------------
TOTAL : 17000.99 inr
------------------------------
```

Removing products from cart



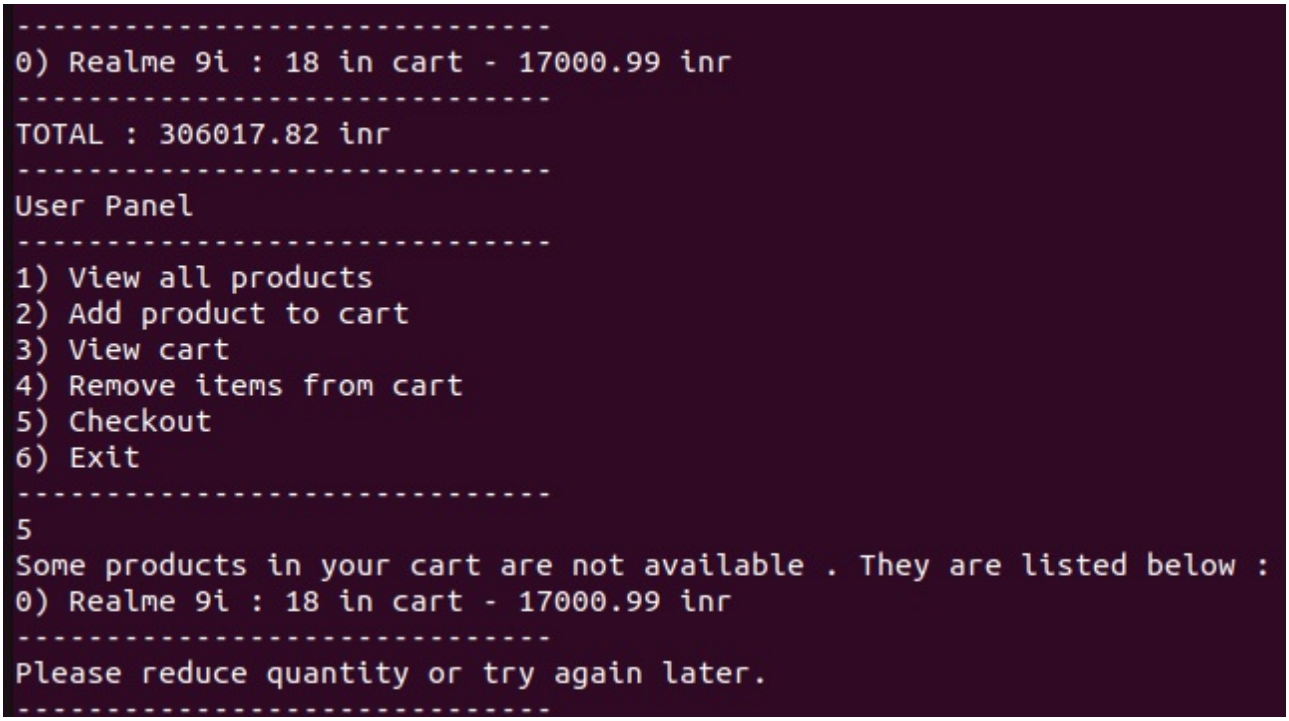
Suppose two users have such quantities of the same product in their carts that both the orders can't be satisfied :





Then the user who checks out first will be allowed to do so :

While the other who couldn't would have to reduce the quantity.



Successful payment