Lab 1 - TOCTOU

Group 21 | Olof Lindberg oloflind@student.chalmers.se &
Rikard Roos rikardro@student.chalmers.se

Part 0

To compile and run ShoppingCart.java follow these steps:

1. Open the directory lab1/part1/
2. Run “make all”
3. Run “make resetFiles”
4. Run “java ShoppingCart.java”

Part 1

What is the shared resource? Who is sharing it?

> The shared resources are the two files wallet.txt and pocket.txt which can be accessed and edited through the methods getBalance, setBalance, addProduct and getProducts. They are shared between threads and processes running the program.

What is the root of the problem?

> The root of the problem is that multiple threads or processes can access these resources simultaneously which can disturb the intended control flow of the program. Specifically, one thread/process can make changes to the files without considering the other thread’s changes.

Explain in detail how you can attack this system.

> The system can be attacked by starting two threads/processes that make purchases simultaneously. A purchase is made through these steps:
>
> 1. Choose product
> 2. Call getBalance
> 3. Check if product price <= balance
> 4. Update wallet balance to old balance - product price
> 5. Add product to pocket
>
> We attack the system by opening two terminals t1 and t2 where t1 makes a call to purchase a car and t2 makes a call to purchase a pen. The concurrent processing can lead to a situation where t1 purchases the car and pays for it but

t2 successfully purchases a pen and overwrites the balance with 29960. We ensure this situation by adding delays after calls to getbalance.

Provide the program output and result, explaining the interleaving to achieve them.

```
rikar@rikard MINGW64 ~/Documents/LP4-2025/tda602/lab1/part0 (rikard)
$ java ShoppingCart.java
Your current balance is: 30000 credits.
car     30000
book    100
pen     40
candies 1

Your current pocket is:

What do you want to buy? (type quit to stop) car
Your current balance is: 0 credits.
car     30000
book    100
pen     40
candies 1

Your current pocket is:
car
```

Figure 1. Terminal 1 makes a purchase of a car.

We open two terminals t1, t2 where t1 purchases a car and t2 purchases a pen (see Figure 1 and Figure 2 for each terminal). We get the following interleaving:

1. t1 chooses car
2. t1 calls getBalance -> t1 balance = 30000
3. t2 chooses pen
4. t2 calls getBalance -> t2 balance = 30000
5. t1 checks if car price <= t1 balance -> true
6. t1 calls setBalance -> t1 balance - 30000 = 0
7. t1 adds car to pocket
8. t2 checks if pen price <= t2 balance -> true
9. t2 calls setBalance -> t2 balance - 40 = 29960
10. t2 adds pen to pocket

After the execution, we have a pen and a car in the pocket and the wallet will have 29960 credits, i.e. we successfully purchased a car but paid for a pen. Note that the wallet.txt is updated twice, first by t1 which sets it to 0 and then by t2 which sets it to 29960.

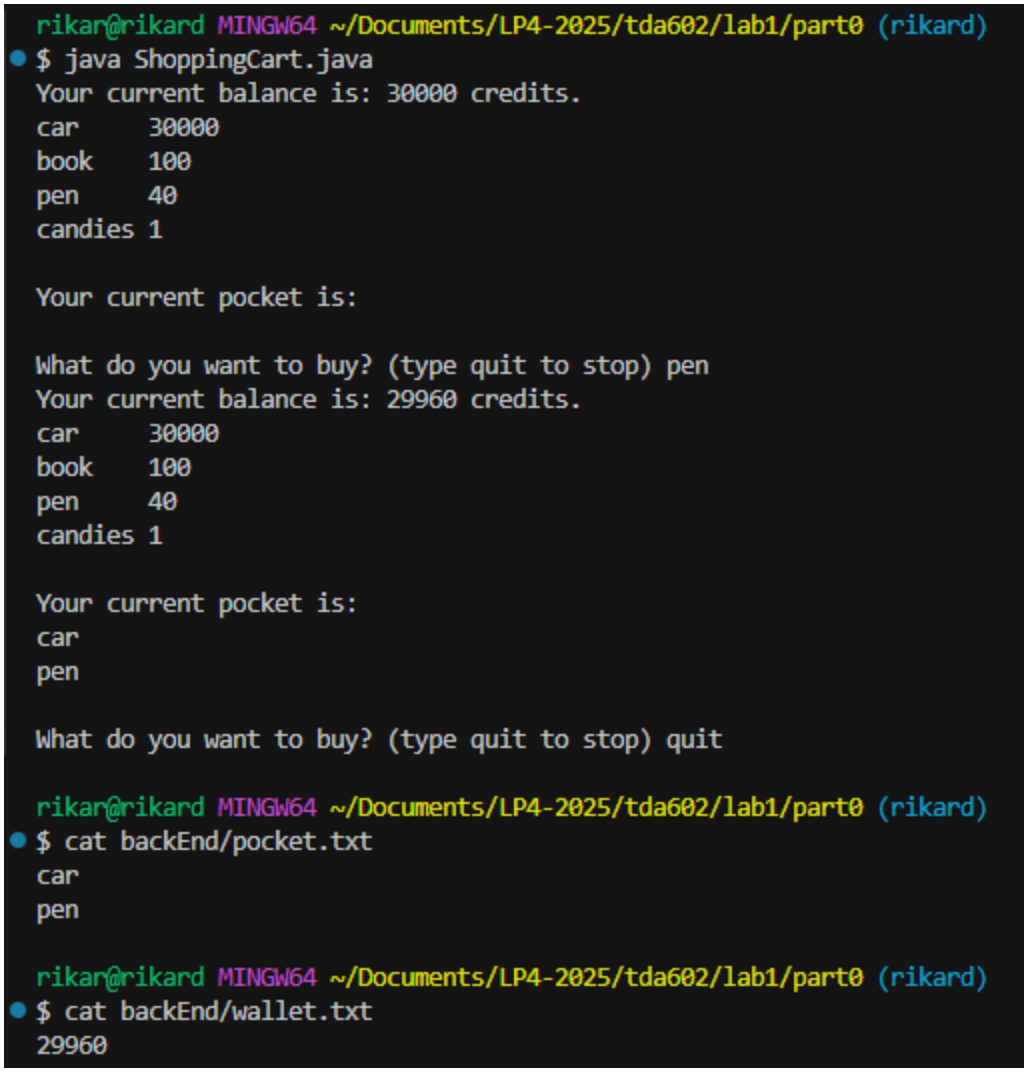

Figure 2. Terminal 2 makes a purchase of a pen.

Part 2

Were there other APIs or resources suffering from possible races? If so, please explain them and update the APIs to eliminate any race problems.

Other concerns we had with the code were the setBalance methods in wallet.java. It was public with no security whatsoever. We made the decision to make setBalance private since we saw no scenario where something outside the wallet class should edit the balance. For example, we could add a safeDeposit function similarly to safeWithdraw. Since setBalance can only be accessed through locked functions inside Wallet, we eliminated race problems through setBalance.

The getBalance method is used in ShoppingCart.java when the balance is displayed to the user. Since it tries to read from the wallet.txt file we wrapped the getter in a try block catching IOExceptions so that no data races occur when reading from the wallet.txt file. Furthermore, we added an overloaded (private) getBalance that is called within safeWithdraw that takes the RandomAccessFile as argument to avoid unintended file lock exceptions.

When eliminating all race problems, it is important to not implement the protections too aggressively, as it could lead to performance hits in the real world. This is something you should consider in the lab. Why are these protections enough and at the same time not too excessive?

We updated the backend so that the frontend could not directly update the balance by making setBalance private, and put locks on the wallet.txt file when it is accessed in a method. By doing so, requests to read or write to wallet.txt must wait for the file to be unlocked. While this could be a performance hit, it is absolutely necessary this resource is secure in order to ensure a correct control flow. After the changes, it is no longer possible to exploit a data race to get a car for less than its price (see Figure 3 and Figure 4).

```
rikar@rikard MINGW64 ~/Documents/LP4-2025/tda602/lab1/part2 (rikard)
$ java ShoppingCart.java
Your current balance is: 30000 credits.
car     30000
book    100
pen     40
candies 1

Your current pocket is:

What do you want to buy? (type quit to stop) car
File is locked, waiting to read...
File is locked, waiting to read...
Your current balance is: 0 credits.
car     30000
book    100
pen     40
candies 1

Your current pocket is:
car

What do you want to buy? (type quit to stop) quit
```

Figure 3. Terminal 1 makes a purchase of a car.

```
rikar@rikard MINGW64 ~/Documents/LP4-2025/tda602/lab1/part2 (rikard)
$ java ShoppingCart.java
Your current balance is: 30000 credits.
car     30000
book    100
pen     40
candies 1

Your current pocket is:

What do you want to buy? (type quit to stop) pen
You can not afford that product.
Your current balance is: 0 credits.
car     30000
book    100
pen     40
candies 1

Your current pocket is:
car

What do you want to buy? (type quit to stop) quit

rikar@rikard MINGW64 ~/Documents/LP4-2025/tda602/lab1/part2 (rikard)
$ cat backEnd/pocket.txt
car

rikar@rikard MINGW64 ~/Documents/LP4-2025/tda602/lab1/part2 (rikard)
$ cat backEnd/wallet.txt
0
```

Figure 4. Terminal 2 trying to purchase a pen.