**AC22005 ATM and Data Race assignment**

**report**

By Aleksejs Loginovs

In this assignment we were required to create a multi-threaded application that would simulate and demonstrate the data-race condition that occurs when two different threads are trying to access and update the same resource at the same time.

The way I approached this problem is to break it out into steps. I solved the whole problem using the following steps:

1. Create a bank and bank account account system/or adapt the one provided as a starting point
2. Create a basic ATM GUI
3. Create interaction with the ATM menus
4. Connect the ATM and the bank account system
5. Apply multi-threading to the program
6. Simulate the problem
7. Apply the semaphore solution

One of the biggest problems that I faced was finding a way of simulating the problem, as this problem occurs in extremely rare occasions and under very unlikely circumstances. However if it arises, it creates big vulnerabilities in the system. It is physically impossible to click two buttons at the same time using one mouse, so the problem is impossible to reproduce in the available conditions.

My solution was the introduction of an artificial 3 second delay during every cash withdrawal that would make ATM machine treat two transactions from two different machines that happened within the three second of each other as simultaneous.

The other problem that I encountered was applying the solution. It was tricky to find a place where the fix could be properly applied. As I didn’t use the initial bank account structure, my money withdrawal method worked in a different way. The bank account itself would receive a request from the bank with the amount to be withdrawn. Then the account itself would do the validations and deduct money from its balance if possible. With this system the only place to apply semaphore to would be the bank account withdrawMoney method.

However I didn’t want the BankAccount class to know anything about what is happening in the program, so I had to add an additional method in BankAccount named setBalance() and make use of it in the Bank class instead. Implementing the semaphore inside the Bank would make more sense to me, as the Bank usually is the system that manages all the transactions, not the accounts themselves.

The last big problem that I had was passing logs from every cash machine back to the ATM manager after they appear. The problem is that I couldn’t directly call non-static methods of ATM manager from the ATM class and I also couldn’t properly access the form elements on a different thread.

The solution that I found was to make use of delegates. I created a global delegate method UpdateLog that would accept the string as the log entry. Then from within the ATM manager I create an instance of this delegate and initialised to be pointing to updateLog method in the ATM manager. Then I pass this delegate to the ATM instances and that lets me call the method from within the ATM.