**Group 12**

Michael Harley - [190006115@dundee.ac.uk](mailto:190006115@dundee.ac.uk)

Viktor Bratov - [2561367@dundee.ac.uk](mailto:2561367@dundee.ac.uk)

Arvin Shafiei - [2503389@dundee.ac.uk](mailto:2503389@dundee.ac.uk)

Word count: 606

**Introduction**

Race conditions occur when multiple threads or processes attempt to modify shared data at the same time, leading to incorrect outcomes. As a group, we were tasked with creating a C# Windows Forms application that displayed a bank system with multiple ATM windows for a single account. The application should implement a semaphore structure that prevents the balance from being incorrectly altered by two simultaneous transactions. Our attempt at solving this was to lock the account being withdrawn from, ensuring that the attempts to modify the balance simultaneously don’t result in incorrect outcomes to the account’s balance.

**Approach**

Starting the project we were provided with the file “atmConsole.cs”, which contained classes for an account and an atm. This was an implementation of an ATM system that ran in a console window and had no multithreading or semaphore implementation for withdrawing cash. The initial ATM program was replaced with three new C# files that carried out the functionality of the original file. These were “Accounts.cs”, “ATM.cs” and “BankSystem.cs”. We also created a Windows Forms application that visually simulated multiple ATMs interacting with a bank system. This admin system could control the number of ATMs being dispatched. Initially, the application was designed to display issues associated with race conditions. It did this by allowing simultaneous transactions to be executed without any synchronization mechanism that used an artificial delay to allow for multiple transactions to easily be requested, leading to the possibility of a negative account balance. To resolve this issue, we decided to use thread locks which ensured that only one thread was able to alter the account balance at a time. We have also made implementations, that for when a new ATM is created the user can choose between locked and unlocked methods for the retrieval of the account balance. The unlocked withdrawing method was implemented for data race demonstration purposes.

**How the race condition was recreated in the program**

When the program is run a main window opens. The program was initialised with an account already added, with account number 333333, pin 3333, and a balance £500. New accounts are added by selecting “open Admin window” and completing the fields. To recreate the race condition “new ATM window” was selected opening a new ATM window. This is done twice to simulate the race condition properly. An option appears to run the ATM unsafe. When it is run unsafe, and you withdraw £500 with two ATMs at the same time from the same account that only has a balance of £500, the balance will go into negative figures, which is not intended behaviour. When running two locked ATM windows, the locked method prevents the balance from going into negative figures. It should be noted that the second withdrawal must be done within five seconds of the first otherwise the race condition will not be simulated properly, it happens because of an artificial delay placed in the withdrawal function.

**Results and Discussion**

The locking mechanism resolved the race condition issue as was expected. Before implementing the locking system, the dispatched ATMs frequently displayed inconsistent account balances after concurrent transactions. After applying thread locks, the system always shows the correct account balance, regardless of the transaction concurrency level.

**Conclusion**

Implementing thread locks in a C# ATM Simulator solved our race condition issues, allowing for consistent and correct updating of account balances across concurrent transactions. This exercise not only reinforced the importance of synchronization in concurrent programming but also demonstrated the practical application of thread locks in preserving data integrity in a multi-threaded environment. Furthermore, it allowed us to improve our skills of programming in C# using Windows Forms.