Assignment 3
C# Threading
Report

Introduction

Our task was to build an ATM constructed out of Windows forms built in Visual C#, but crucially this would demonstrate an interesting and potentially very dangerous computing concept known as a data race, whereby two concurrent programs or processes in a computer try to change the same data value at the same time. In the context of an Automated Teller Machine, i.e. two ATM machines making a cash withdrawal from the same account at the same time, this would result in the balance on the account after the withdrawal to be incorrect, possibly to a very great extent, which would have catastrophic consequences for the bank and economy. Thankfully, software nowadays, including ATMs, is restricted in the sense that critical code can only be accessed by one process, or thread, at the same time. The software we have written demonstrates both scenarios and uses threads to create different concurrent ATMs that both access the same account at the same time.

Design

We built the ATM program around the fact that each instance of an ATM needed a central point of reference such that separate instances of an ATM can both access the same figure at the same time. We decided to store the details in the program class, which required them to be static fields. We also kept the data race field in the program class as a static boolean.

Due to the extreme speed of computers compared to humans, ensuring that two instances of the ATM class could interact with the same field at the same time is practically impossible, so we had to artificially simulate this. After brainstorming, we came to the conclusion that as long as each ATM object is interacting with the same figure when making a withdrawal, the result will be the same for each withdrawal. When implemented, this involves each ATM copying the current balance to its own local field and then pausing its currently running thread before actually amending the central balance. This pause allows for other ATM objects to retrieve their own copy of the central balance before then performing the same calculation as the other instances and writing the same value to the central balance. What this means in a practical sense is, when two ATMs try to take out £5, the central balance's value only decrements by £5 instead of £10.

Problems & Solutions

Despite this relatively straightforward solution, we spent a long time figuring out if it was possible to make all threads execute at the same time so that a proper data race could occur. This resulted in some fiddling around with many thread functions and using timers to sync everything up, along with the .Start(), .Resume() and .WaitOne() functions. Needless to say, this didn't end well and we settled for second place by artificially producing a delay which represents an instance in time where two or more ATMs interact with the same figure. This is far better at demonstrating to the user what happens during a data race and how.

We also struggled with where to store the details of the accounts. We would have liked to have kept them in the Menu class but this made them hard to access due to difficulties with the instance of the currently running menu. As such, we solved this by storing the details in the Program class, making the static in the process. This allows for the Menu and ATM classes to interact with them while also accessing the Account classes and its functions and fields.

Word Count: 584