Lab exercise 6

Concurrency

Due date

Mar 28, 2023

Academic Integrity

The following actions are strictly prohibited and violate the honor code. **The minimum penalty for plagiarism is a grade of zero and a report to the Aggie honor system office.**

- Uploading assignments to external websites or tutoring websites such as Chegg, Coursehero, Stackoverflow, Bartleby, etc.
- Copying code from external websites or tutoring websites such as Chegg, Coursehero, Stackoverflow, Bartleby, etc.
- Copying code from a classmate or unauthorized sources and submitting it as your work.

Keywords

Threads.

Introduction

The objective of this exercise is to show you a real-world purpose of concurrency in modern systems.

Your program will execute a set of transactions from a bank account in 3 different ways. You are given a csv file 'transactions.csv' that contains one column, the amount of money to withdraw or deposit from the account. You will carry out each transaction on the account and print the total time taken and the final balance.

Expected Functionality

Your program's input will be the -i flag for the name of the file. Assume the bank account starts with $0. The main function will have three sections that always run, each starting with a new instance of the BankAccount class and iterating over all of the rows in the csv.

The first section will execute every transaction completely synchronously, where we call each transaction in order in one thread. You can use the BankAccount.transaction() function to perform each transaction.

The second approach will use a vector of threads to carry out each transaction. You will use a while loop to start a new, detached (not dependent on the parent), thread for each row of the csv. This will also use the BankAccount.transaction() function.

The third approach will build on the second, but use a new BankAccount.transaction_threadsafe() function. This function will implement mutex variables in order to ensure that each call to the account will not affect a separate call.

Starter code

You are given the BankAccount class that already contains the transaction() function completed. You will need to implement the **transaction_threadsafe()** function using mutexes to ensure proper concurrency.

You are given the file Teller.cpp which contains the main function of your program. This file has comments for each section and uses getopt to read in the filename of the csv. The first section will implement a simple loop that calls BankAccount.transaction() directly for each row of the csv.

For the second and third scenario, you will need to use the thread() class to create threads that use the **teller_thread()** and **teller_threadsafe()** functions inside of Teller.cpp. These functions will simply call the BankAccount transaction functions. Since we are creating threads, you cannot call the BankAccount.transaction functions directly.

Once each section has completed all of the transactions, print the total time that section took to complete as well as the final balance. You can use the BankAccount.print() function to print the account balance.

Rubric

There are three tests that check the elements of the program which you have been asked to fix and grades will be assigned as the score you see on GitHub classrooms (under the Actions tab):

1. Compilation (10 pts)
2. Correct output for section 1 and 3 (33 pts)
3. Use of threads (24 pts)
4. Use of mutex (33 pts) [Graded manually]

Getting started

1. Go to the assignment's GitHub classroom: https://classroom.github.com/a/6fLJHPnq.
2. Create a repository for your project.
3. Watch the Getting Started video: https://www.youtube.com/watch?v=nvdGWF7rQwY