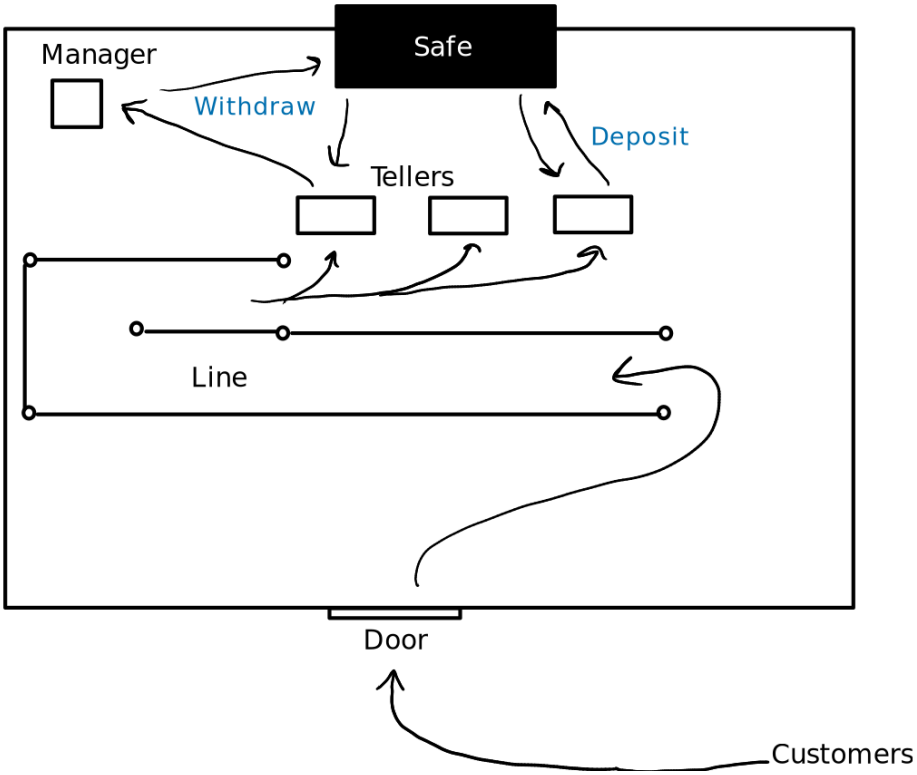
Project 2

Due: Friday, April 11th 11:59pm

Description

You will write a simulation of a certain bank. There are three tellers, and the bank opens when all three are ready. No customers can enter the bank before it is open. Throughout the day, customers will visit the bank to either make a withdraw or make a deposit. If there is a free teller, a customer entering the bank can go to that teller to be served. Otherwise, the customer must wait in line to be called. The customer will tell the teller what transition to make. The teller must then go into the safe, for which only two tellers are allowed in side at any one time. Additionally, if the customer wants to make a withdraw the teller must get permission from the bank manager. Only one teller at a time can interact with the manager. Once the transaction is complete the customer leaves the bank, and the teller calls the next in line. Once all 50 customers have been served, and have left the bank, the bank closes.



Details

You should program a single application in c,c++, java, or python. The program will simulate the bank using threads for the tellers and customers. If you program in c or c++, use pthreads and

POSIX semaphores. Alternatively, in c++ you may use std::thread and the provided Semaphore class. If you program in java use the Thread and Semaphore classes. If you use python, use the threading module, and threading.Semaphore for synchronization. The program will launch three threads for the tellers and then 50 threads for the customers. Details about the threads are below.

Identify the shared resources the threads will be using. For example, the safe and the manager are fairly obvious ones. These will need to be protected by semaphores. Other semaphores will also be needed to synchronize the behavior of the threads. For instance, the customer should not leave until the Teller has finished the transaction. You do not need to ensure that your semaphores are strong (also called fair).

The Teller

When a teller thread is created it should be given an unique id to differentiate it from other tellers. There are three (3) tellers. The teller will follow the following sequence of actions until there are no more customers to serve.

1. Teller will let everyone know it is ready to serve
2. Wait for a customer to approach
3. When signaled by the customer, the teller asks for the transaction
4. Wait until customer gives the transaction
5. If the transaction is a Withdraw, go to the manager for permission.
   - The manager always gives permission, but will take some time interacting with the teller
   - To represent this interaction, the teller thread should block (sleep) for a random duration from 5 to 30 ms.
6. Go to the safe, waiting if it is occupied by two tellers
7. In the safe, the teller will physically perform the transaction
   - represent this by blocking (sleeping) for a random duration of between 10 and 50 ms.
8. Go back and inform the customer the transaction is done
9. Wait for customer to leave teller

The Customer

When the customer thread is created it should be given an unique id to differentiate it from other customers. There are 50 customers. The customer will follow the following sequence actions.

1. The customer will decide (at random) what transaction to perform: Deposit or Withdrawal.
2. The customer will wait between 0 – 100ms
3. The customer will enter the bank (The door only allows two customers to enter at a time).
4. The customer will get in line.
   - If there is a teller ready to serve, the customer should immediately go to a ready teller.
   - otherwise, the customer should wait until called and then go to a ready teller.
5. The customer will introduce itself (give its id) to the teller.
6. The customer will wait for the teller to ask for the transaction.
7. The customer will tell the teller the transaction.
8. The customer will wait for the teller to complete the transaction.
9. The customer will leave the bank through the door (and the simulation).

The Output

The teller and customer threads should print out a line for each (simulated) action they are performing, and should use the following format.

`THREAD_TYPE ID [THREAD_TYPE ID]: MSG`

The `THREAD_TYPE` can be either "Customer" or "Teller". The `ID` is the threads assigned id. The `MSG` is a short description of the action taken. Here is an example.

```
Customer 10 [Teller 0]: selects teller
```

Every time a thread must block for an amount of time, there should be two lines. One line will be document he action taken, and be before the wait, and the other will document the actions completion, and be after the wait.

Similarly, accessing the shared resources (manager and safe), there should be three lines. One indicating the teller is going to the resource, one indicating the thread is using the resource, and another indicating the thread is done using the resource.

For a more detailed example, see the sample run attached to the assignment on e-learning.

Some Suggestions

Here are some suggestions that might make your life easy.

- Do not focus on just the teller or just the customer at the start. The program is about the interactions between the two, so start with a short interaction. First code the program so that all that happens is the customers introduce themselves to the teller.
- Start with a small number of threads. Your final submission should work with 50 customer threads, but nothing is stopping you from starting with 3 threads and then moving to 5 then 10, etc. Smaller number of threads means less output, and makes it easier to visualize what is happening.
- You will need shared variables for the threads to communicate with each other. Consider what information must be passed between two threads, and what variables can be used to manage that. Consider the need to mutual exclusion as well.
- Do not forget that semaphores are used for synchronization. Consider their behavior as part of the simulation. Some things can be done with a semaphore or two without the need of other variables.