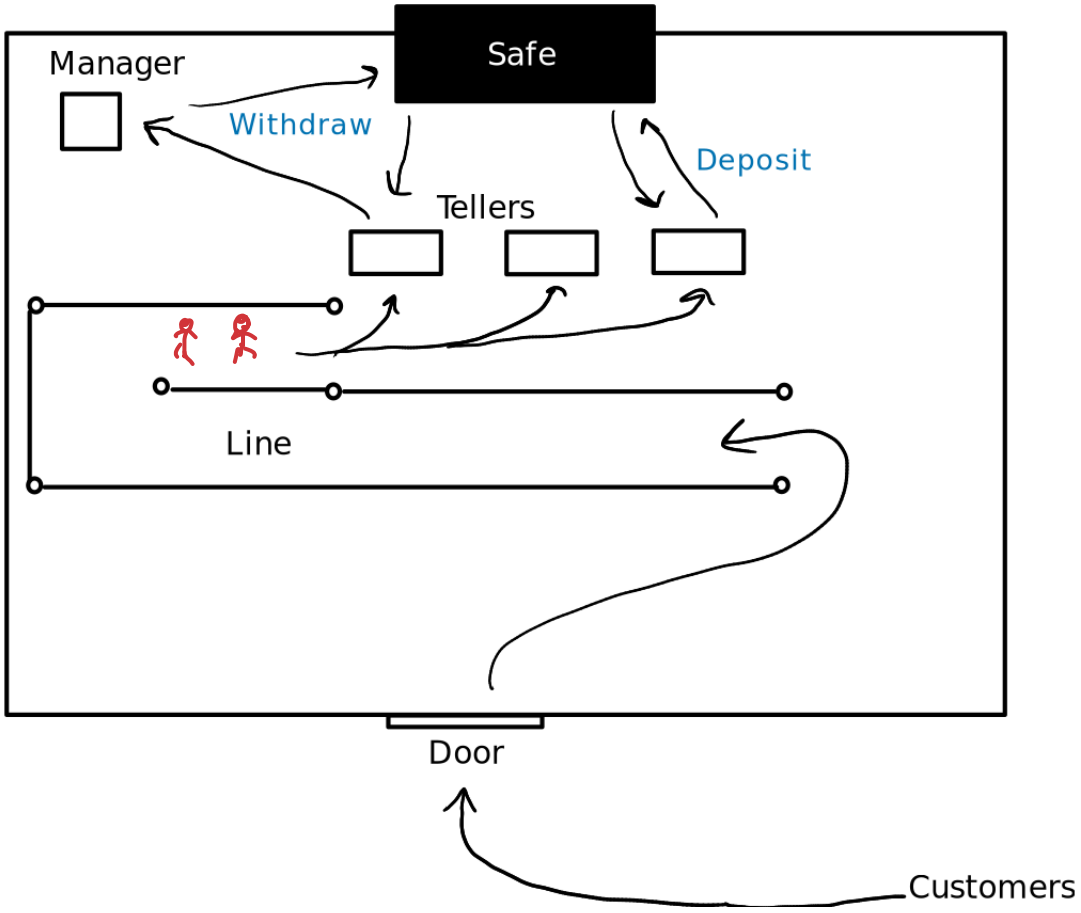
Project 02

Due: Wednesday, November 2nd 11:59pm

1 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.



2 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. 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).

2.1 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.

- Teller will let everyone know it is ready to serve
- Wait for a customer to approach
- When signaled by the customer, the teller asks for the transaction
- Wait until customer gives the transaction
- 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 for a random duration from 5 to 30 ms.
- Go to the safe, waiting for it if it is occupied by two tellers
- In the same, the teller will physically perform the transaction
  - represent this by blocking for a random duration of between 10 and 50 ms.
- Go back and inform the customer the transaction is done

2.2 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.

- The customer will decide (at random) what transaction to perform: Deposit or Withdrawal
- The customer will enter the bank (The door only allows two customers to enter at a time).
- The customer will get in line
  - If there is a teller ready to serve, the customer should immediately go to the first ready teller
  - otherwise, the customer should wait until called and then go to a ready teller
- The customer will introduce itself (give its id) to the teller
- The customer will wait for the teller to ask for the transaction
- The customer will tell the teller the transaction
- The customer will wait for the teller to complete the transaction
- The customer will leave the bank (and the simulation) ← implement it tonight.

3 Output

The teller and customer threads should print out a line for each action they are performing. Each element of the bulleted lists above should correspond to a line of output, at minimum. The output will the only way for you to debug the program, so the more of these lines detailing what is happening the more information you have for debugging. Here is an excerpt from a sample run:

```
Customer 7 is going to the bank.
Customer 7 is getting in line.
Customer 7 is selecting a teller.
Customer 7 goes to Teller 0.
Customer 7 introduces itself to Teller 0.
Teller 0 is serving Customer 7.
Customer 7 asks for a withdrawal transaction.
Teller 0 is handling the withdrawal transaction.
Teller 0 is going to the manager.
Teller 0 is getting the manager's permission.
Teller 0 is got the manager's permission.
Teller 0 is going to the safe.
Teller 0 is in the safe.
Customer 4 is going to the bank.
Customer 4 is getting in line.
Customer 4 is selecting a teller.
Customer 4 goes to Teller 1.
Customer 4 introduces itself to Teller 1.
```

```
Teller 1 is serving Customer 4.
Customer 4 asks for a deposit transaction.
Teller 1 is handling the deposit transaction.
Teller 1 is going to the safe.
```

4 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. Maybe 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.

5 What to Turn in

Upload your submission as a zip archive containing the following:

- Source code (c, c++, java, or python files)
  - Source code should not require a particular IDE to compile and run.
  - Should work on the cs1 and cs2 machines
- Readme (Plain text document)
  - List the files included in the archive and their purpose
  - Explain how to compile and run your project
  - Include any other notes that the TA may need
- Write-up (Microsoft Word or pdf format)
  - How did you approach the project?
  - How did you organize the project? Why?
  - What problems did you encounter?

- How did you fix them?
- What did you learn doing this project?
- If you did not complete some feature of the project, why not?
  * What unsolvable problems did you encounter?
  * How did you try to solve the problems?
  * Where do you think the solution might lay?
    · What would you do to try and solve the problem if you had more time?

6 Grading

The grade for this project will be out of 100, and broken down as follows:

Followed Specifications	50
Use of Semaphores	20
Correct Output	20
Write-up	10

If you were not able to complete some part of the program discussing the problem and potential solutions in the write-up will reduce the points deducted for it. For example, suppose there is a bug in your code that sometimes allows two customers to approach the same worker, and could not figure out the problem before the due date. You can write 2-3 paragraphs in the write-up to discuss this issue. Identify the error and discuss what you have done to try to fix it/find the problem point, and discuss how you would proceed if you had more time. Overall, inform me and the TA that you know the problem exists and you seriously spend time trying to fix the problem. Normally you may lose 5 points (since it is a rare error) but with the write-up you only lose 2. These points can make a large difference if the problem is affecting a larger portion of the program.