This project will be implementing the distributed consensus problem as outlined in the project descriptions. We will be using C for this project, and are currently not planning on using any external libraries. The actors are all different running instances/nodes/processes of the program. For the purposes of this project, we will be implementing a version that supports five total instances of the program, forked from the main program. As this is using the Two Phase Commit protocol, one of these instances will randomly be given the role of coordinator. This coordinator will communicate with the other processes using sockets, and each process will be assigned a port number. This communication line will be used to get votes for a transaction from all the other instances, and then to commit this transaction if all instances were in consensus. To avoid race conditions, we will use mutexes to access shared memory that will contain information about the bank account. This information includes the information about the current balance, credits, and debits from the one bank account this program is simulating.

This program will support three actions. The first is *credit* which will credit money to the simulated bank account using the consensus protocol. The second is *debit* which will take out money from the simulated bank account using the consensus protocol. The final action is *query* which returns the local value of the bank account when called. The only fault case handled by this project is for when an instance of the program terminates. When revived, this instance will use the consensus protocol to get the current balance of the bank account from other processes. If the coordinator is the one to crash, a different process will be assigned the role of coordinator.

To test this project, we will have different test cases that are used to demonstrate that our program will behave as we expect it to, as outlined above. We will first test action cases, such as using *credit*, *debit*, and *query*. This will require having different numbers of processes running, and using these actions in distinct orders to ensure that they work as expected. We will additionally test the fault case by terminating a program and ensuring that it will get the correct balance when we revive it.

There were a few assumptions made in this design. One of those is that the *credit* action will run the consensus protocol, but there is no reason for any process to vote against the credit transaction. Therefore, all processes would vote for a credit always. Another assumption is that we are only implementing one bank account. Realistically, this doesn’t make a lot of sense, but for the purposes of this assignment and simplicity, only one bank account is simulated. Another assumption is that the initial balance of this bank account is zero dollars.

Finally, to implement this, we will be breakdown all of the work. Together, Aneesha and Veena will create the basic structure of the program. Aneesha will work for the proper function of the command line, and will also read through the article that is also a part of this project. Veena will work to create the sockets and ports connecting different processes. Both team members will implement the actions of *credit*, *debit*, and *query* together to ensure that both understand the two phase commit protocol. Realistically, this breakdown will change based on what is needed when, but overall, the work will be distributed evenly amongst both members of the team.

The resources used will be the manual pages offered, as well as the provided information about the consensus protocol, two-phase commit at <http://the-paper-trail.org/blog/consensus-protocols-two-phase-commit/>. We will also use internet resources and class code to help us understand more about sockets and how to implement them, as well as about other aspects of coding this project.