Distributed Systems Lab Report 1

1. What did you use in your implementation to ensure that the *Election* service records a vote whenever a client thinks they have cast a vote?

To ensure that the *Election* service records a vote when a client casts a vote, the vote method checks if the voter’s number has been used. If the number is unique, then the hash table of results is checked for the candidate. If there is no candidate, then the method adds the candidate to the hash table. After the client places their vote, the server prints out that the vote has been submitted, and the client prints out the current results.

1. How did you achieve the guarantee that all votes are safely stored even when the server process crashes?

To ensure that the votes are safely stored in the event that the server process crashes, the state of the *Election* is recorded into a text file after each vote method is called. This allows the server to read the text file upon restarting to retrieve the last state of the *Election* before the server crashed.

1. Outline your implementation for ensuring that the records remain consistent when it is concurrently accessed by multiple clients. You should include snippets of your program to help with your explanation.

In order to ensure that records remain consistent when it is concurrently accessed by multiple clients, the vote method has a mutex lock. Each client that invokes the vote method must acquire the key before they can cast their vote. If they cannot access the key, the clients stay in a recursive loop until the key becomes available. This ensures that all votes eventually will be cast, while also ensuring that there are no conflicts with accessing the resources.