This projects goal was to concurrently run six processes, each doing 500 transactions, to perform one of the five possible operations on one of the four bank accounts. To assure that none of the balances of the accounts are corrupted, the use of mutual exclusion was necessary. This was done with the use of semaphores. Our original design to was to declare the semaphore with size of 4, one semaphore for each bank account. We thought this would be the most efficient solution because operations would only be blocked if that specific bank account was being operated on. The problem that occurred during testing our solution was deadlock. Rarely would the program run without becoming unresponsive at certain points. We isolated the problem and discovered that this mainly occurred when the transfer of two random bank accounts was attempted. The problem that was occurring in within that operation was circular waiting. A process would hold one of the bank account’s resource and then request for the other random bank account and become held up at that request. By it getting held up in between request (semaphore P operation), the process will never release the resource that it is using causing deadlock between all the processes.

To avoid deadlock we decided to implement a different solution to the problem. The result of that was to modify the size of the semaphore to 1 instead of 4 to represent the entire bank instead of each specific account. This implementation will cause each transaction to block the use of all four bank accounts when one it enters the critical section. Although this solution isn’t as efficient in theory to our previous solution, it prevents deadlock and circular waiting. After 50 executions of the program, it successfully runs with a 100% success rate. We also collected data on the amount of times each of the five operations was performed. Below is a pie chart to display the results of that.