Elvin Rivera

Csci 144 MWF

Project Report

I chose to keep this project as simplified as I could. I chose to include basic libraries such as iostream, iomanip (input manipulation for decimals), fstream, semaphore, sstream, cstdlib, and pthreads.h. I had to download the libraries and .dll. To ensure that I was able to implement the unistd and pthreads, I chose to run the program using g++ with Cygwin. Continuing with the process, I ran the program with -ipthrrad elvin144.cc then ./a. I generated random numbers for selling, buying, cost per share. I set a tracker to see where to be finished and where to continue. The most I could use was 10,000 transactions which was the max that I set it to. I set a bools to oversee when no stocks were available or if they were, when the buy thread finished and when the sell thread finished. I also created unique identifiers for each (buy and sell). Semaphore was used to implement mutex. I had to keep track of stock names, stock costs, and stock shares by making them a MAX. I also had to create various lock conditions for each lock. For mutex semaphore I had to use 0 or 1 for false or true respectively. I created a function for buy, sell, and for share, For stock names, I used an array to hold each element. For cost per share I randomized it to 1000%+1 to have only 1000. For sale price per share, I only did 2000. To identify how many stocks were available I set it only 100.

To start I had to create some stock names, share price etc. into a txt file. Then after achieving this, I would use them to determine whether it should buy or sell. However, I implemented a while condition to check whether we had insufficient funds to purchase the stock then we needed to sell. Also, there was a condition where if you couldn’t afford to buy the shares, you had to look one down to find a stock that you could buy the shares for. Also need to check If it is possible to purchase the stock and shares. A print line would display in the screen if there was insufficient funds. The functions were defined as buy and sell to keep the simplicity of the program intact. I also had to have an idea who to release the lock, therefore I created a pthread mutual exclusion to do so. When there were no available stocks to sell, I set it to exit the program. When the sell thread was done, I would then unlock the sell thread and exit the sell thread so it could refresh and another stock could use it (deadlock prevention). I used simple algebra to calculate the percent yield. I had to determine if the price of stock per share that I bought, was either higher or lower than a certain threshold we would diced to sell. This portion is essential in understanding and calculating an overall percent yield. What I did was divide the cost by the sale. If stock’s shares were higher than the original amount that we purchased the stock for, then I called a pthread sell and sell the stock. This would make a profit gain, thus increasing the overall balance remaining. The unistd.h library was implemented in my program so that I could display a certain amount of time while using sleep which can be seen towards the end of the program. This made it easier to display the message, or any message for the amount needed so the program could act as a real time stock market.

This is just the first iteration of the program. If I were to continue working on it, some changes I might make to it include a real-time interface so that you can constantly review the stocks without having to wait for the “sleep” method. I would also implement a more accurate file of the stocks in the real world and their information so that the buyer is well informed to make an educated decision on whether to buy or sell. Minor additions to the overall display would make it more user friendly and easier to read, but for the most part of this first iteration of the program, I feel it is adequate for simulating a simplified version of an automated stock market program.