CIS 415 Operating Systems

Project 3 Report Collection
Submitted to:

Prof. Allen Malony

Author:
Amelia Bates
abates
951915806

Introduction

The Duck Bank project is about creating a system that simulates a real-life bank. This system can handle customer requests like deposits, withdrawals, transfers, and checking their balance. The biggest challenge in this project is to process these requests quickly and accurately, even when multiple threads are working at the same time.

The project is divided into four parts, starting from a basic single-threaded program and ending with advanced features like thread coordination and communication between two separate processes. It helped me understand how threads work, how to prevent errors when multiple threads access the same data, and how to use shared memory between processes. I ran into a lot of memory leaks and errors, so I also gained more hands-on experience using gdb and valgrind to help solve my problems.

Background

This project uses threads to process transactions faster by dividing the work among multiple threads. Threads run in parallel, so we used tools like mutexes locks to protect shared data and condition variables to help threads communicate with each other.

In Part 2, we added locks to ensure that there were not multiple threads updating the same account at the same time. In Part 3, we used barriers to make sure all threads paused and waited for the bank thread to update balances after processing a certain number of requests. In Part 4, we connected two separate banking processes (The Duck Bank and The Puddles Bank) using shared memory. This allowed them to exchange account data efficiently.

Implementation

Part 1

In Part 1, I created a basic program that processed all transactions one by one. This part helped ensure the logic for deposits, withdrawals, transfers, and balance checks was correct. Everything worked correctly and output as expected.

Part 2

In Part 2, the program was improved by using 10 worker threads to process transactions. Each thread handled a portion of the transaction list. I used mutexes to prevent errors when threads updated account balances at the same time. A separate bank thread was added to apply reward rates after all threads completed their tasks. Everything worked correctly and output as expected.

Part 3

In Part 3, the threads were designed to pause after processing 5000 transactions. This allowed the bank thread to update account balances with reward rates before the worker threads continued. I used barriers to synchronize all threads and condition variables to let threads notify each other. Each account's balance was logged in a file after every update. I spent many hours trying to figure out what went wrong, I could not understand why I was not getting the expected output. The number of updates that it produces is incorrect, along with the balance. However, the final balance is still correct.

Part 4

In Part 4, a second process, Puddles Bank, was introduced. Duck Bank shared account data with Puddles Bank using shared memory. The Puddles Bank accounts started with 20% of the balance that they had in Duck Bank. Both banks applied their reward rates, and balances were updated separately. For this project I also struggled with getting the correct output. For the Ledger, there is only supposed to be 200 lines, I had almost 300. I am unsure where I went wrong.

Performance and Results

Both Part1 and Part2 worked as expected. However, I really struggled with parts 3 and 4. I had to decide to just turn in what I have as I need to focus my time on studying for the final in this class and in my other computer science class.

Conclusion

All in all, this project taught me a lot about how to manage threads and prevent errors when multiple threads access shared data. I also learned how to coordinate threads to work together effectively and how to use shared memory to connect different processes. One of the biggest challenges was debugging issues like deadlocks, where threads got stuck waiting for each other. Using tools like valgrind helped me identify memory leaks and fix them. This project showed me how important it is to carefully design and test programs when dealing with threads. If I had extended time, I would troubleshoot the issues that I had in Part 3 and Part 4.