CIS 415 Operating Systems

Project <3> Report Collection

Submitted to:

Prof. Allen Malony

Author:

*<Andrew Chan>*

*<UO ID - 951908437>*

*<Duck ID - achan10>*

**Report**

**Introduction**

*In this project, we are making a multithreaded banking system called Duck Bank which is supposed to simulate a concurrent processing environment that is used for bank financial transactions. This project is supposed to be able to handle high traffic volumes efficiently while making sure that the correctness of account balances is intact. There is also functionality for transactions such as being able to deposit, withdraw, transfer, and balance checks. These are also being concurrently executed by multiple threads. The project was divided into four parts where each part built upon the previous ones just like project 2. This eventually led to the IPC (inter-process communication) between the two bank systems, Duck Bank and Puddles Bank.*

*This project gave me hands-on experience with core concepts such as multithreading, synchronization, shared memory, and process communication. The following report will show the challenges I face, how I overcame them, the implementation of parts, and the results.*

**Background**

*This project required implementing multithreaded solutions for being able to process financial transactions as well as the inter-process communication between the banks. We used thread management and synchronization, shared memory, and inter-process communication. Worker threads processes transactions concurrently, which are protected by mutexes and condition variables to avoid race conditions and to also ensure the safety of the threads. Shared memory was used to establish communication between the Duck Bank and the secondary Puddles Bank. Puddles Bank maintained a savings account for each user of the Duck Bank, which was initialized with 20% of their Duck Bank balance and a flat 2% reward rate. And we also had pipes that facilitated the communication between the Duck Bank and the Auditor process which was the “IRS”. The Auditor logged specific events, such as balance checks, and interest updates to a ledger.txt file.*

**Implementation**

*When it came to the implementation of this project, the project was split into 4 parts. As mentioned before, the implementation used multithreading, synchronization, inter-process communication, and shared memory to simulate a concurrent banking system.*

*In part 1, I implemented a single-threaded system to basically just process account setup and the transactions sequentially. This program read the account data and the transaction from the input file, processed each transaction in the order they appeared in the file, and also it gave an output file with the final account balances for all of the users. This gave a baseline for not only the correctness, but also formed the foundation for the multithreaded implementation for the next parts.*

*In part 2, we used multithreading with 10 worker threads to handle the transactions concurrently. Each of the threads was assigned to a subset of all of the transactions. And a further bank thread updated the balances once all of the worker threads completed their tasks. I used mutexes which are a form of synchronization mechanisms to protect critical sections, thus ensuring the safety of the threads. One key feature of this part was the even distribution of transactions among worker threads which was achieved by counting all of the transactions and splitting it amongst them. Also I used pthread\_mutex\_t to ensure the protection of the shared resources. This part required me to identify and protect the critical sections, such as updating account balances, and accessing the shared resources to prevent race conditions.*

*In part 3, I implemented coordination among threads using condition variables and barriers. Worker threads paused after processing a threshold number of transactions which didn’t include balance checks to notify the bank thread to update balances. After the bank thread completed the updates, it then signaled the worker threads to resume processing. I used pthread\_cond\_wait and pthread\_cond\_signal to synchronize worker threads with the bank thread. The bank thread also appended balance updates specific to each account to the output file after each update cycle. I also used pthread\_barrier\_t to make sure that all threads started at the same time which therefore improved coordination. I focused on ensuring efficient thread communication and avoiding deadlocks which was achieved by structuring wait and signal mechanisms around shared data.*

*In part 4, I extended this Duck Bank by introducing inter-process communication and shared memory to a Puddles Bank. This is implemented as a separate process which initializes the used fork. Shared memory was used to share account data between Duck Bank and Puddles Bank. lThe Puddle Bank maintained savings accounts for each of its users and initialized with 20% of their Duck Bank balance and a standard flat 2% reward rate. Every time the Duck Bank applied interest to accounts, Puddles Bank performed a similar operation for their savings accounts. A key feature was using nmap to initialize and share account data between Duck Bank and Puddles Bank processes. I also used inter-process communication which ensured the synchronization between both of the banks through shared memory updates.*

**Performance Results and Discussion**

*I believe my code runs well. I was able to achieve the correct final balances throughout all of the parts and even correctly mapped account specific logs. The output files were also consistent across multiple runs and dynamic updates at threshold intervals were achieved successfully without any deadlocks or race conditions. I did however have issues throughout with deadlocking. But this was easily fixed after making sure everything was mutex locked at the right places and also making sure that nothing was waiting cyclically. I also used the runner.sh file that was provided to test it over and over a very high amount of times.*

**Conclusion**

*In conclusion I liked this project and learned a lot. I think it was really cool seeing this project progress throughout the multiple parts and it was very insightful with how multithreading works with real-world constraints. This project was very challenging but also very rewarding. The introduction of shared memory and inter-process communication in Part 4 was really cool as it showed practical methods for extending functionality across multiple systems. I also liked that aspect of protecting things with a mutex lock and I found that really helpful. Overall this project deepened my understanding of OS concepts, specifically with thread management and IPC since this was a complex yet fulfilling experience that was able to enhance my ability to look at and solve problems in the sphere of concurrent programming.*