CIS 415 Operating Systems

Project 3: Duck Bank Report Collection

Submitted to:

Prof. Allen Malony

Author:

<Tong Guan>

<UO ID - tongg>

<951871617>

Report

Introduction

In the project3, we build a multithreaded solution for the duck bank to handle tons of thousands of transactions.

This project includes 4 parts, the part1 presents a single thread solution, the part2 presents a basic multithread solution to allocate the transaction tasks to 10 worker thread, the part3 presents two different kinds of thread — worker and bank thread, and how we corporate these two threads to work together, the part4 presents the multithreading with multiprocess for duck bank client to own another account at puddle bank.

Background & Implementation

In this part, I will try to explain my idea about how to approach the solution in each part in detail.

In part1, I reused the function and file “string_parser.c/h”from project1 to tokenize the input file for initializing bank accounts and reading transactions. After reading the commands, we transfer each account to finish the transaction.

In part2, instead of “online” getting and reading the transaction, after initializing the accounts, I store all the tokenized commands and allocate them evenly to 10 worker thread. By transferring the index of thread during the thread creation, I divide the commands to 10 different area. Considering the critical section protection, I lock each transaction part to make sure the thread safe and the correctness of the balance update.

In part3, when thinking of the design about how to corporate the worker and bank thread, I used the classic model of the producer and consumer as an important reference.

Here is the core coding part of my design:

```
worker_thread_deal_transaction(){
while(not end){
    do transaction…
    pthread_mutex_lock(&lock)
    total_request++
   if(total_request == 4999){
      pthread_cond_broadcast(&cond)}
   while(total_request >= 4999){
     pthread_cond_wait(&cond, &lock);}
    pthread_mutex_unlock(&lock)}
```

```
banker_thread_update_balance(){
    while(processed_request < count){
       pthread_mutex_lock(&lock)
       while(total_requests < 4999){
             if(last_call) break
             pthread_cond_wait(&cond, &lock);}
       update balance and request…
       pthread_cond_broadcast(&cond);
       pthread_mutex_unlock(&lock);}
```

In part4, considering of the communication and the shared information of the duck and puddle process, I created the shared memory to store all the account information and shared by the two processes, and use the two signal SIGSTOP and SIGCONT to control the puddle process being updated with the update_balance function in the duck process.

Performance Results and Discussion

I'd like to discussion the performance result by two parts, one is the result of completed transactions, the other is about leaking memory issue.

(1) result of transactions: All of parts of project successfully finished the tasks and generated correct answers.

(2) Leaking memory: Unfortunately, all parts faced the same kinds of leaking memory issue, after debugging for very long time, I'm still unable to fix this leaking. I will try to rethink this issue and solve it after final exam.

Conclusion

This is the last project of the CS class 415. During these 3 projects, my coding ability has improved greatly in two ways, one is the code design, the other is debugging.

Code Design: Back when I was working on my first project, I was so confused as to how to design a project. Many previous homework only needed to fill in some key code in a specified function, and the need to design and grasp the whole project by myself was a great challenge. Now I'm more confident and capable of writing code, in the part3 and part4 of this project, I designed and wrote nearly 500 lines of code independently to satisfy the all the requirement of the projects.

Debugging: The debugging ability includes two parts, finding the bugs and solving the bugs, either one needs patience for adjusting and intelligence for reviewing the concepts and redesign the code. I met a lot of different errors in the part3 and part4, and spent great amount of fixing them, and I'm very happy to solve all of them except the memory leaking due to time limitation and review for finals.

There is a saying, "always get your hands dirty", by writing the code and debugging, I have a better understanding of the operating system and how can we better control the whole system to improve our performance while keep the safety and smooth communication.