REPORT OF S BANK BATCH PROCESSING SIMULATION DEMONSTRATION

Tianru Peng (2092575)

Hui Lin (2072293)

Yuhuang Li (2096356)

Contents

ABSTRACT

This project (S Bank Batch Processing Simulation Demonstration) aims to establish a simulation of the nocturnal batch processing operation across multiple branches of a virtual bank. It will leverage a script developed in a Linux environment to emulate the real-world processes involved in nightly batch handling. The routine line-up includes data assembly, data verification and auditing, batch program interest processing, and report creation. The script will exemplify simultaneous batch processing execution across various branch offices within the same time frame. Our team intends to validate and practically illustrate the knowledge points in operating systems. The project will cover concurrent processes, transaction concurrency, file resource sharing, and lock mechanisms. The project explores the efficacy of incorporating operating system principles in real-world problem-solving scenarios. By using a hands-on approach, the project will effectively underscore the importance and real-world applicability of principles of resource allocation, process synchronization, and deadlock handling, thereby enhancing our understanding of operating systems.

Keywords— operating system (OS), batch scheduling, concurrency, file lock.

I. INTRODUCTION

1. Project Objectives:

The project S Bank Batch Processing Simulation Demonstration is designed to validate and demonstrate practical examples of the principles taught in CSCI6638 course. The project aims to strengthen our learning outcomes by simulating bank batch processing scenarios to validate knowledge points in the operating system about multi-process, concurrency control, file locking, etc. The corresponding textbooks [1] chapters are as follows:

- 2.1.7 Modeling Multiprogramming
- 2.3.6 Mutexes
- 2.4.2 Scheduling in Batch Systems
- 4.3.4 Shared Files

2. Project Conception:

The scope of this project is to simulate and demonstrate the batch processing flow of a bank. First, we build business scenarios to clarify requirements. In order to control the complexity of project implementation, we simplify the envisioned business scenarios. Then we build a virtual environment and build a batch processing platform based on the Linux operating system.

By executing scripts, we simulate the real process involved in bank batch processing, including data assembly, data verification and auditing, interest calculation processing, and report generation. Finally, by observing and analyzing the script execution log and output files, we verify the advantages of multi-process processing in the operating system.

3. Prerequisite Dependency Description:

It is worth noting that the project has the following dependencies:

- The implementation of the batch processing system depends on certain hardware and software facilities, as listed in Table 1.
- This project only considers bank batch processing business, not daily online transactions.
- The test data used in this project are all fictitious and do not involve any real customer information.
- For the sake of simplicity, all data are stored in text files, and this project does not involve databases.

Item	Resource Description
Server Hardware	MacBook Pro
Virtual Software	Parallels Desktop 18 for Mac version 18.1.1

Server OS	Ubuntu 22.04.4 LTS
Script Runtime Environment	GNU bash, version 5.1.16
Compiler	gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Development Tool	Shell scripts, C, VI

TABLE 1. PROJECT RESOURCE ENVIRONMENT

II. REQUIREMENT DEFINITIONS

1. The simulated business scenario:

Given that a bank named "S Bank" has three branches in Vancouver sharing an unified set of account data. During the day, each branch handles daily business and generates its own transaction data. At night, the bank will focus on batch processing.

The batch processing system (short for BPS) orchestrates the execution of multiple tasks including data collection, data verification and auditing, updating account information and generating reports/statements in the nighttime batch operations.

- *Data Collection*: Each branch is tasked with the job of collecting the transaction data over business hours. Branch staff uploads the transaction data to the file server. The BPS collects these data and then processes them.
- *Data Verification and Auditing*: Post data import, verification and auditing are essential to catch any inconsistencies or errors that might have cropped up during the day. Data verification is required from the perspective of business logic, and data auditing is required from the perspective of banking regulation.
- *Update Account Information*: The BPS should also be adept at updating customer information based on the transactions carried out during the day. Interest calculation based on the daily closing balance and updating the same in the customer's account are also a part of the system's responsibilities.
- *Report Generation*: The conclusion of the batch processing should see the system generating reports post all the transactions and updates.

2. Use Case Diagram

According to the above project scope, this project only focuses on simulating the batch processing process of the S Bank. To this end, it is centered around the batch processing system and involves the following actors:

- *Branch staff*: responsible for uploading transaction data and viewing various reports.

- *System administrator*: responsible for maintaining the scripts of the BPS and managing related configuration settings (such as execution time, number of concurrent connections, etc.).
- *Maintainer*: responsible for confirming the execution of the batch processing script, and if there is a problem, intervene to manually adjust and re-execute the batch processing job.

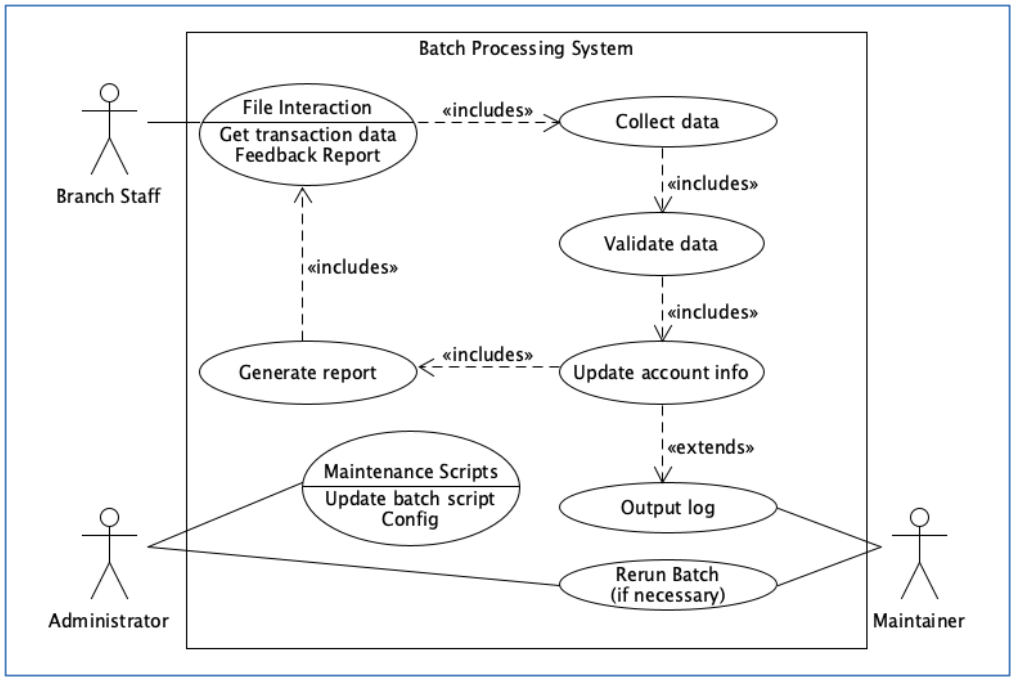


Fig. 1. Use Case Diagram for BPS

3. Activity Diagrams

According to the above requirements, the batch processing of S Bank involves the following specific activities:

Initially, the batch job can be triggered automatically by the BPS at a scheduled time daily, or manually by the administrator or maintainer, typically in cases of handling anomalies.

Under normal circumstances, batch jobs are executed sequentially. Firstly, transaction data is collected from those three branches. This data is then checked and audited. Upon verification of data accuracy, the system updates the account information and eventually generates reports.

In scenarios where a branch fails to provide complete transaction data before the batch, the administrator or maintainer will initiate offline contact with the branch staff to ensure accurate data provision before continuing with the batch processing.

Prior to the account information update, the system executes a data backup. This precautionary step allows data recovery and rerun of the batch operation in cases of anomalies during the update.

After the system completes the batch processing process, the generated report will be provided to the branch staff for review, and the system operation log will be provided to the administrator or maintainer for review.

Therefore, the activity diagram is designed as shown below Fig 2.

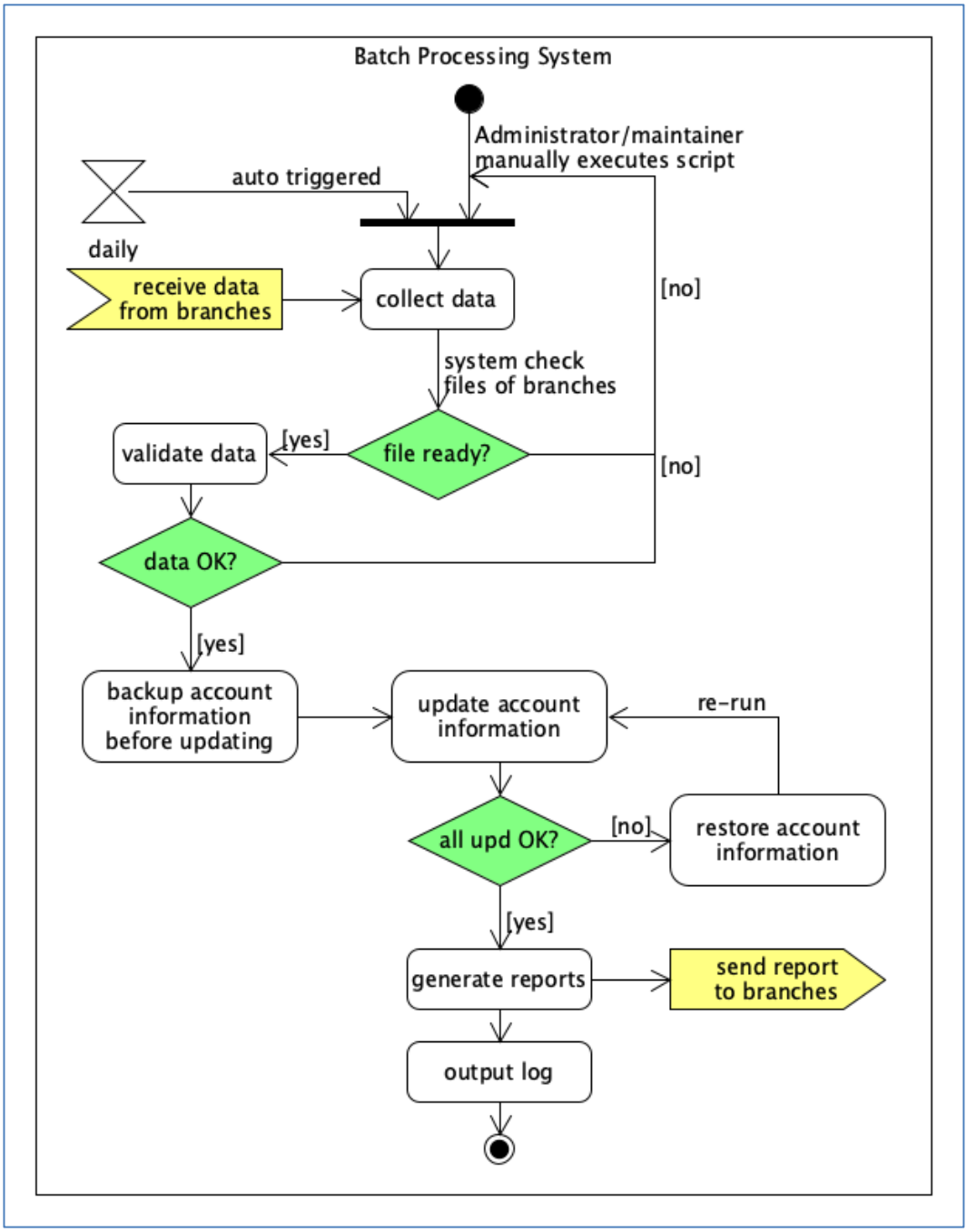


Fig. 2. Activity Diagram for BPS

III. IMPLEMENTATION PROCESS

The implementation of this project is divided into two stages. The first design stage includes data table design, batch process design, and batch program function design. The second implementation stage converts the above designs into data files and script codes.

1. Project Design:

According to the above requirements, the project simulation demonstration process mainly involves data verification, updating and statistical processing of the bank system's account information and transaction information. So, the data table design includes two key data tables:

a. *Account table*, which stores the account data of bank customers. Note: In order to simplify the verification model, customer information is incorporated into the account table.

b. *Transaction table*, which stores the transaction details of branches every day. In order to simplify the verification model, transaction types only include update information, deposit, withdrawal, transfer in, and transfer out.

Table Name	Account	
Field Name	**Constrain**	**Remark**
accountNo	Uniquely identifies each Account of S Bank	Primary Key
clientName	registration information: client's name	Not Null
clientPhone	registration information	Not Null
clientEmail	registration information	Not Null
clientAddress	registration information	Combined by: street, city, postcode
accountStatus	statues of account	Value Scope: normal loss closed
accountType	type of account	Value Scope: SAV CHQ
currency	currency code	'CAD' as default
amount	balance	999999.99

TABLE 2. DESIGN FOR ACCOUNT TABLE

Table Name	Transaction	
Field Name	**Constrain**	**Remark**
txID	Uniquely identifies each transaction	Primary Key YYMMDD +nnnn
accountNo	Account No	Foreign Key ref to Account

		Table
txDate	the date of tx	YYYYMMDD
txTime	the time of tx	HHMMSS
txType	the type of tx	Value Scope: update deposit withdraw transin transout
txCurrency	currency code	'CAD' as default
txAmount	amount of tx	999999.99
updPhone	update registration information	Null
updEmail	update registration information	Null
updAddress	update registration information	Null

TABLE 3. DESIGN FOR TRANSACTION TABLE

2. Implementation:

The project deliverables include batch scheduling scripts, batch processing program source code and related test data. Please see the compressed file in the file link for details.

OS Project Product_TAP team_0702.zip

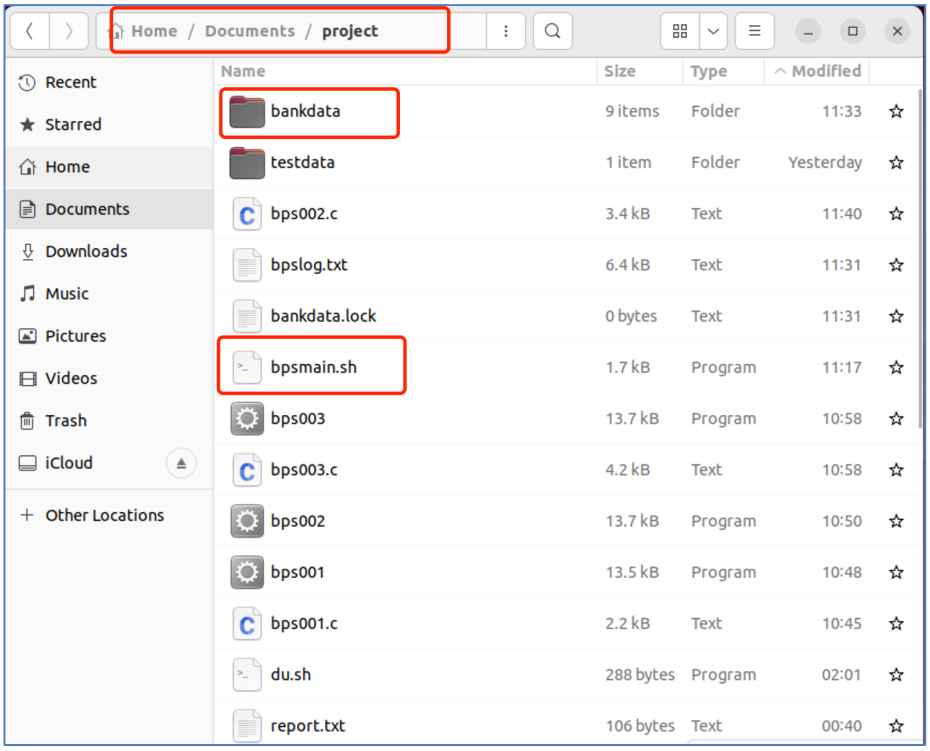
Please see Table 4 below for a detailed list of resources.

Type	Item	Resource Description
Code	bpsmain.sh	Batch processing main script
Code	compile.sh	Program compilation script
Code	bps001.c	Data Validation Program
Code	bps002.c	Account Information Update Program
Code	bps003.c	Report Generation Program
Data	bankdata.lock	file lock
Data	transaction	Daily transaction details of each branch
Data	report	Daily transaction reports for each branch
Data	account & transaction	S Bank's basic account Simulating data
Data	bpslog.txt	Batch execution log file

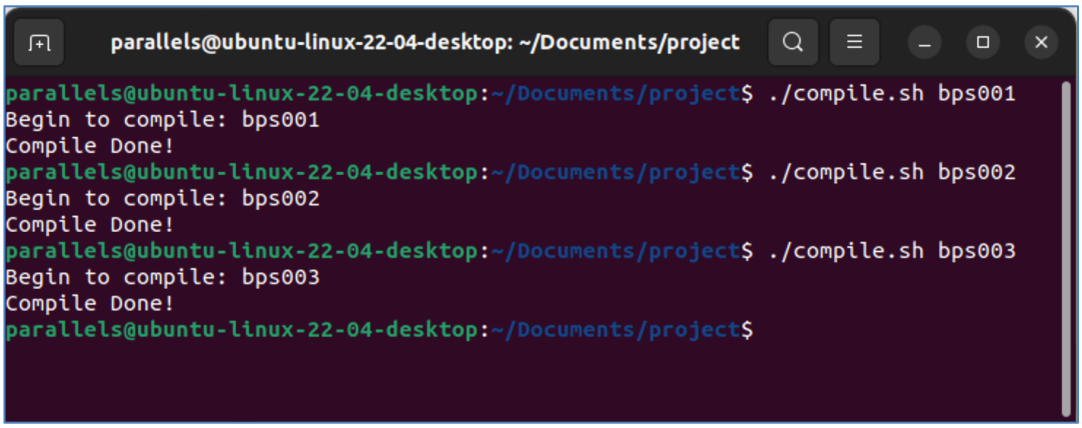
TABLE 4. PROJECT DELIVERY LIST

After creating and configuring the above project resources, the specific simulation verification process steps of this project are as follows:

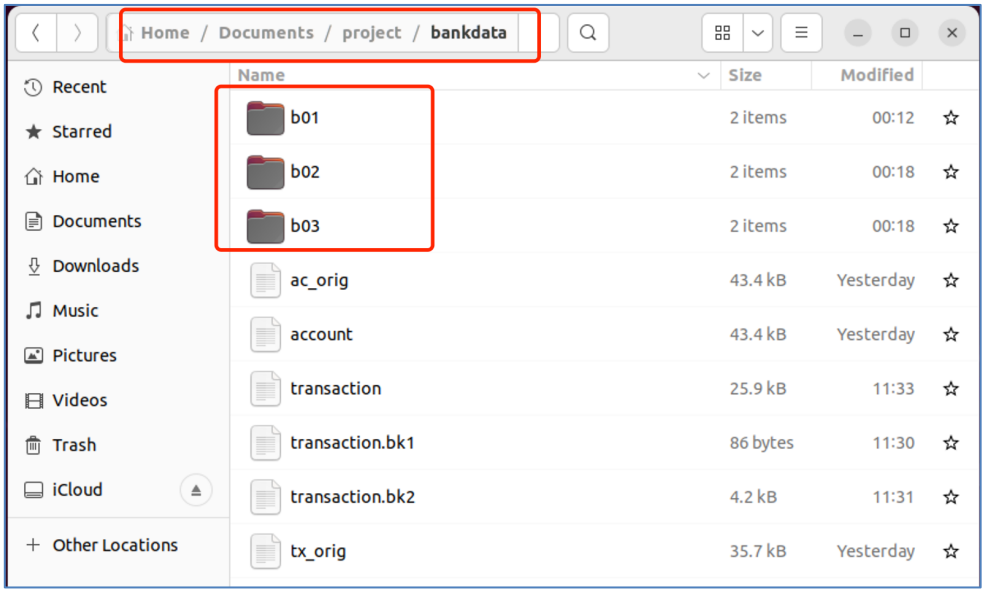
a. Enter the working directory and check and confirm the storage path of batch scripts and batch programs.



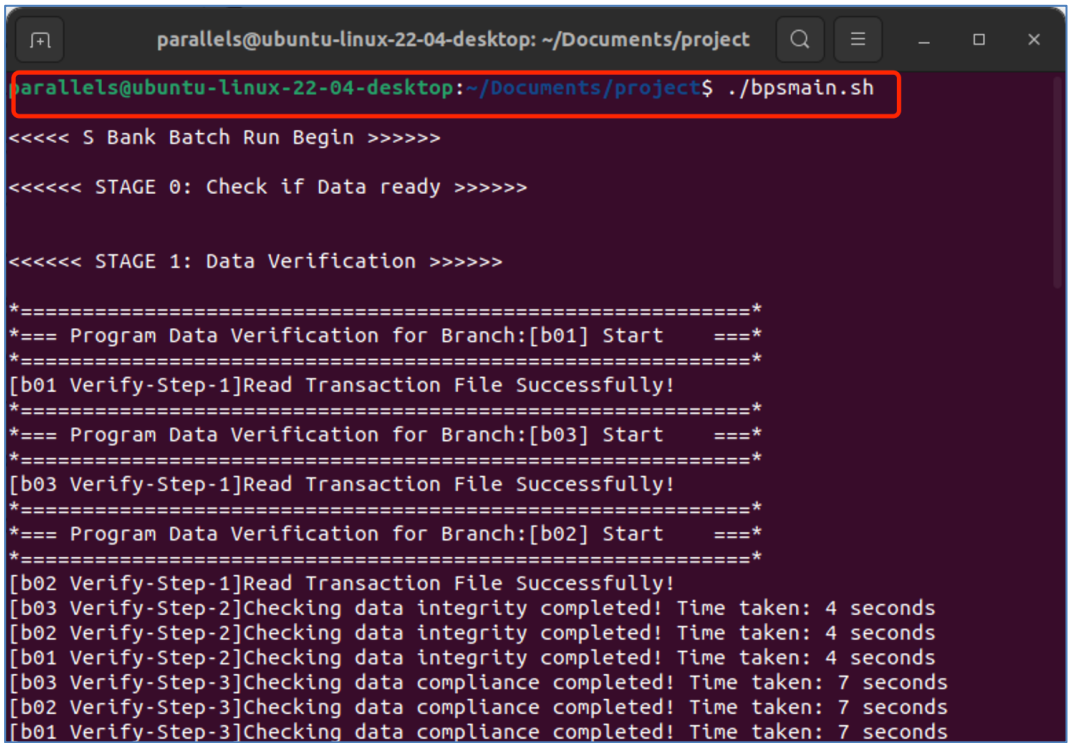
b. Compile the batch program.



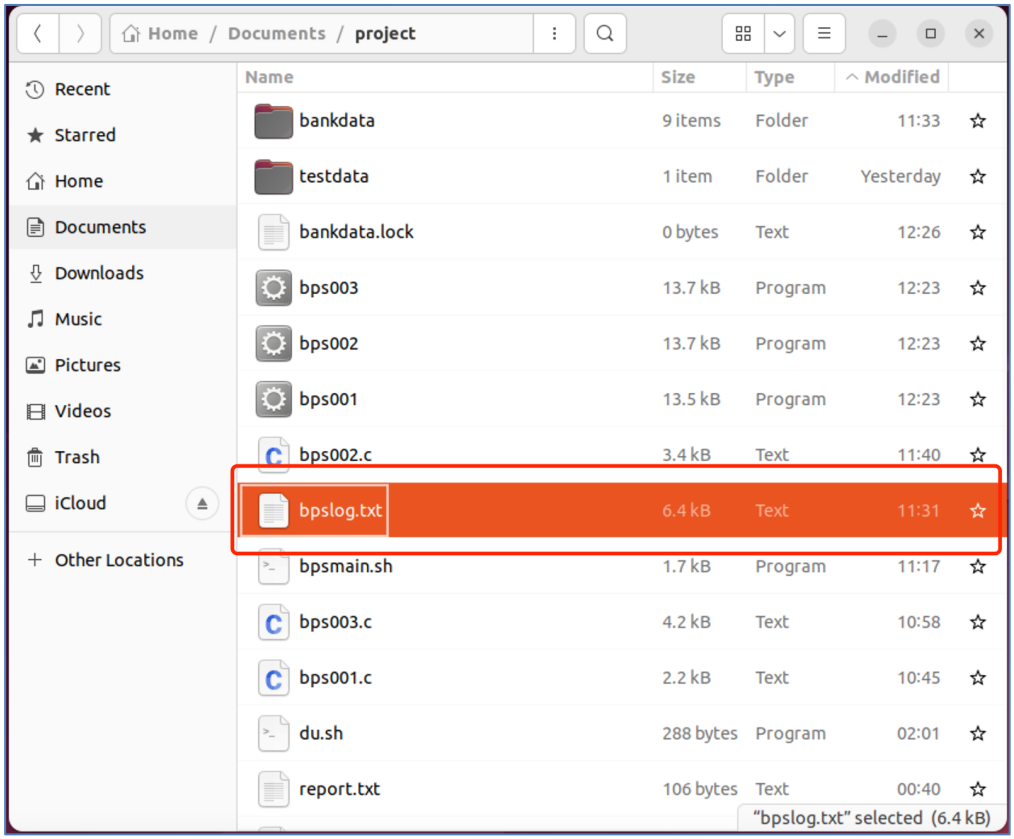
c. Check whether the input data is ready.



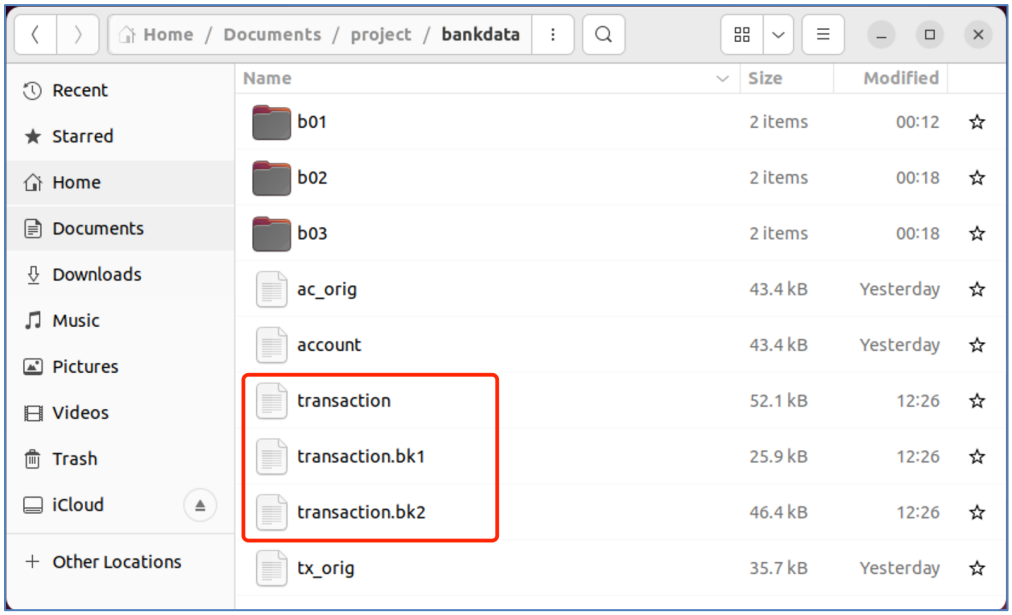
d. Execute the BPS main batch script in the terminal.

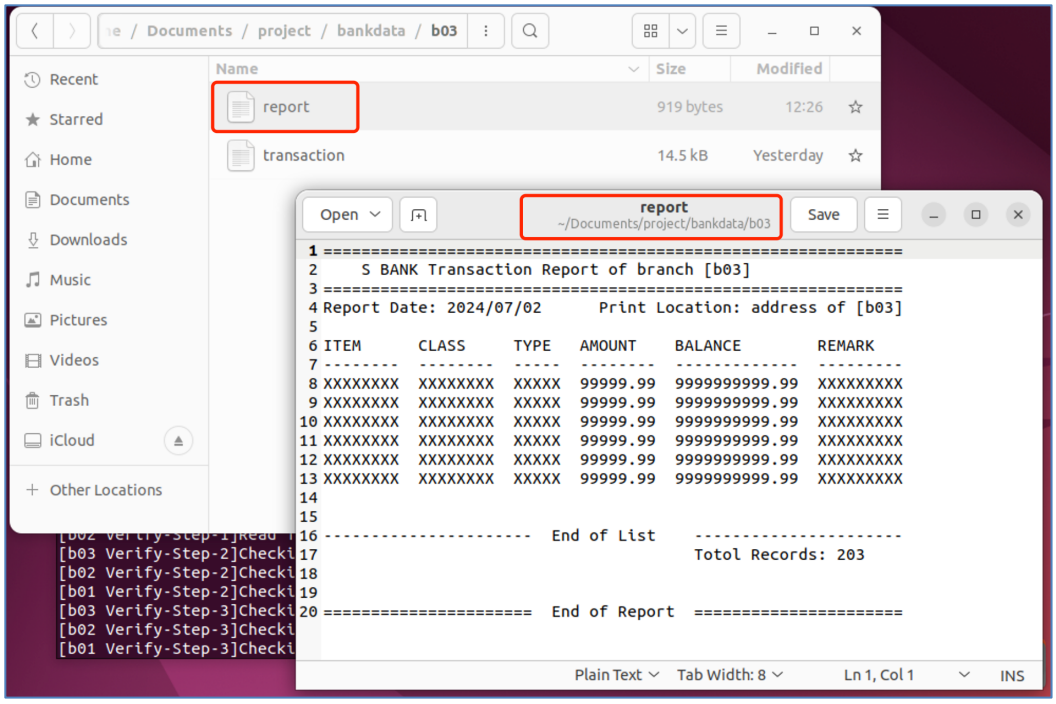


e. Check the running results.



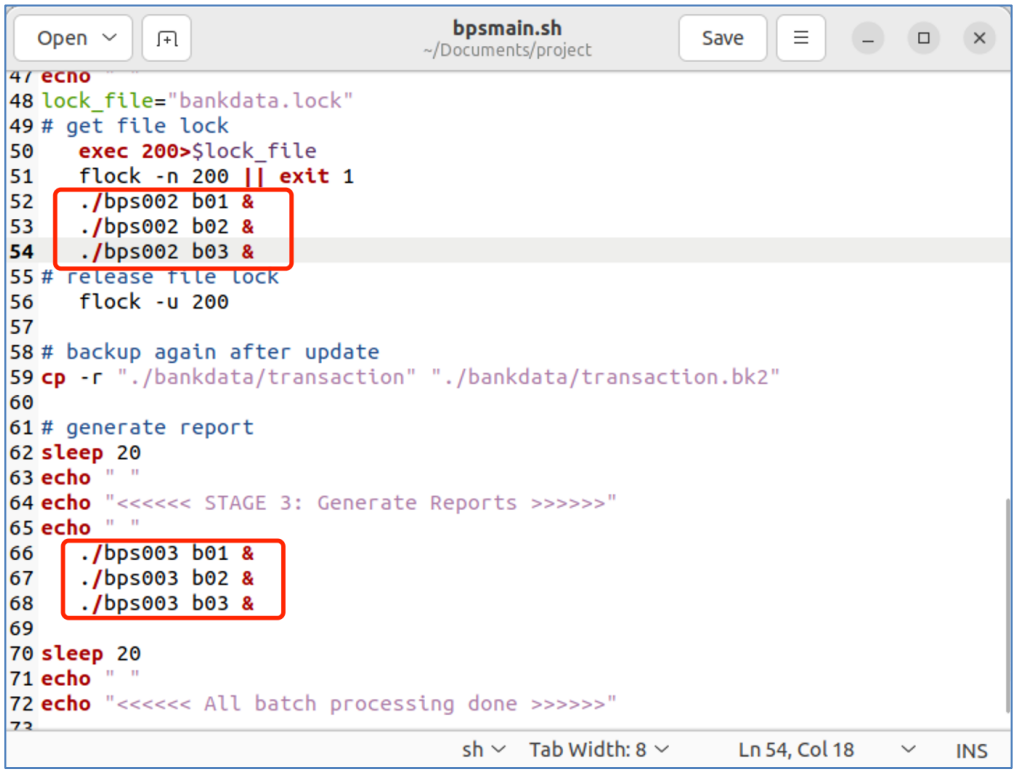
f. Check the file update status.



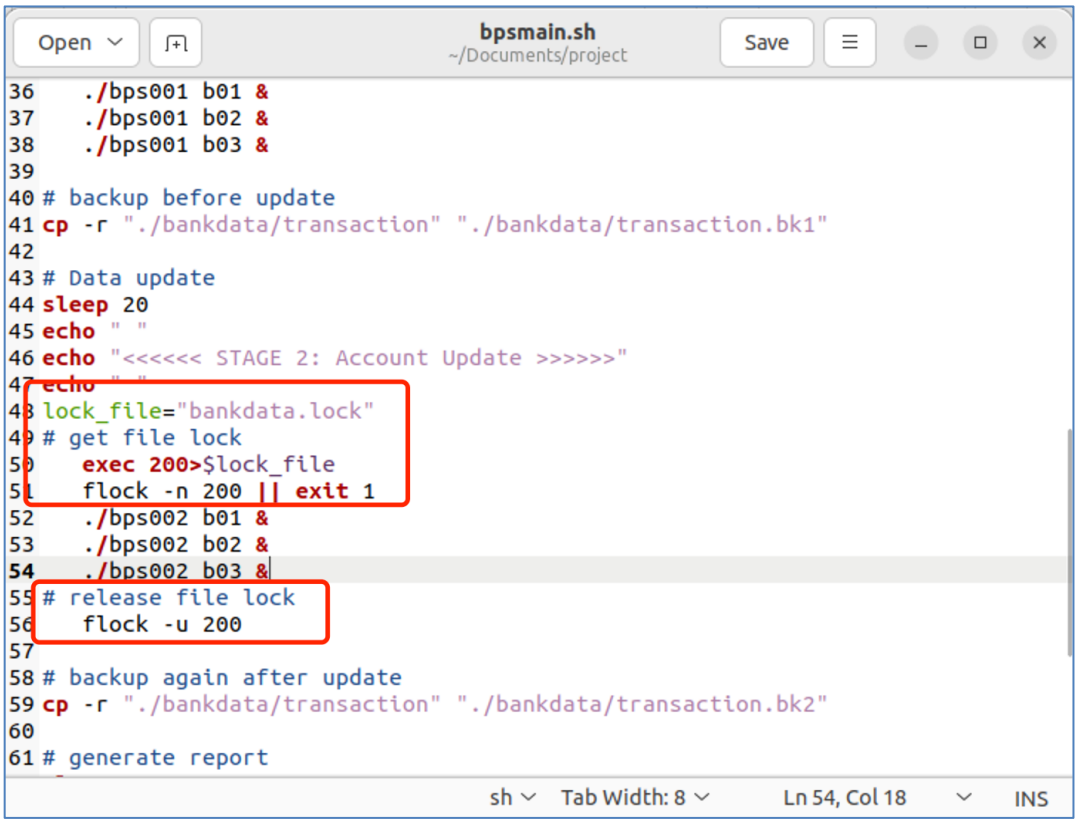


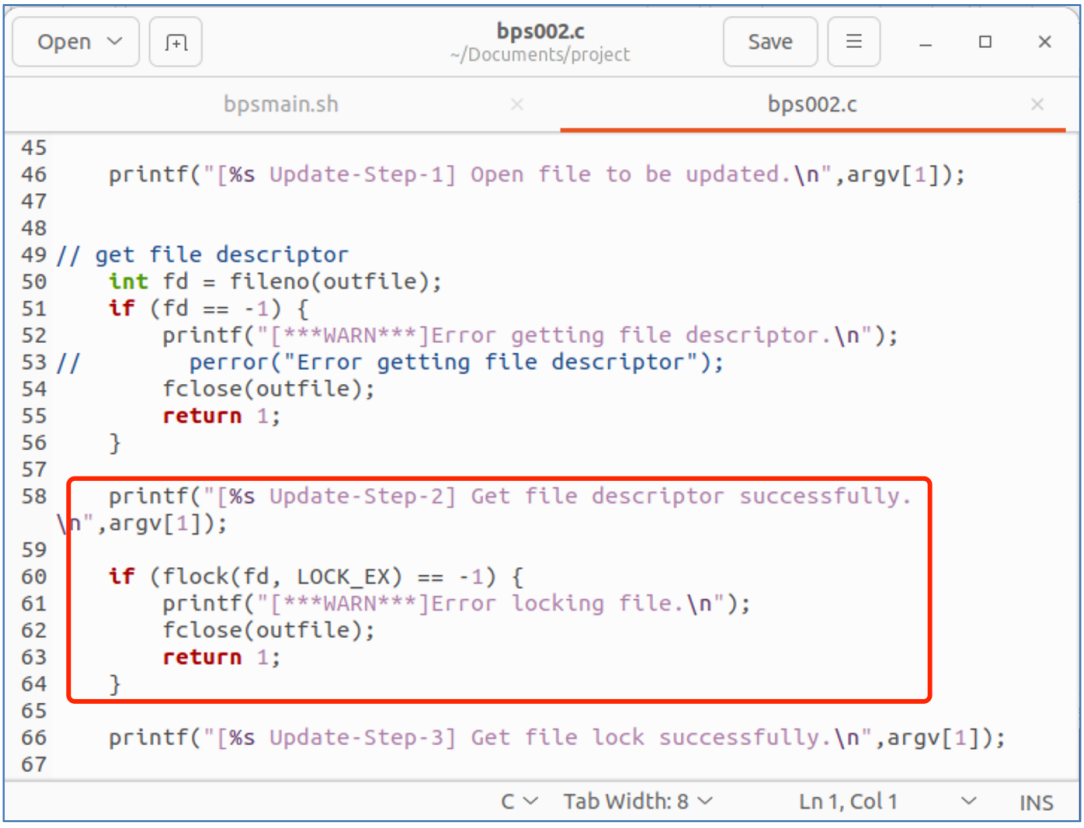
The technical points that need to be explained are as follows:

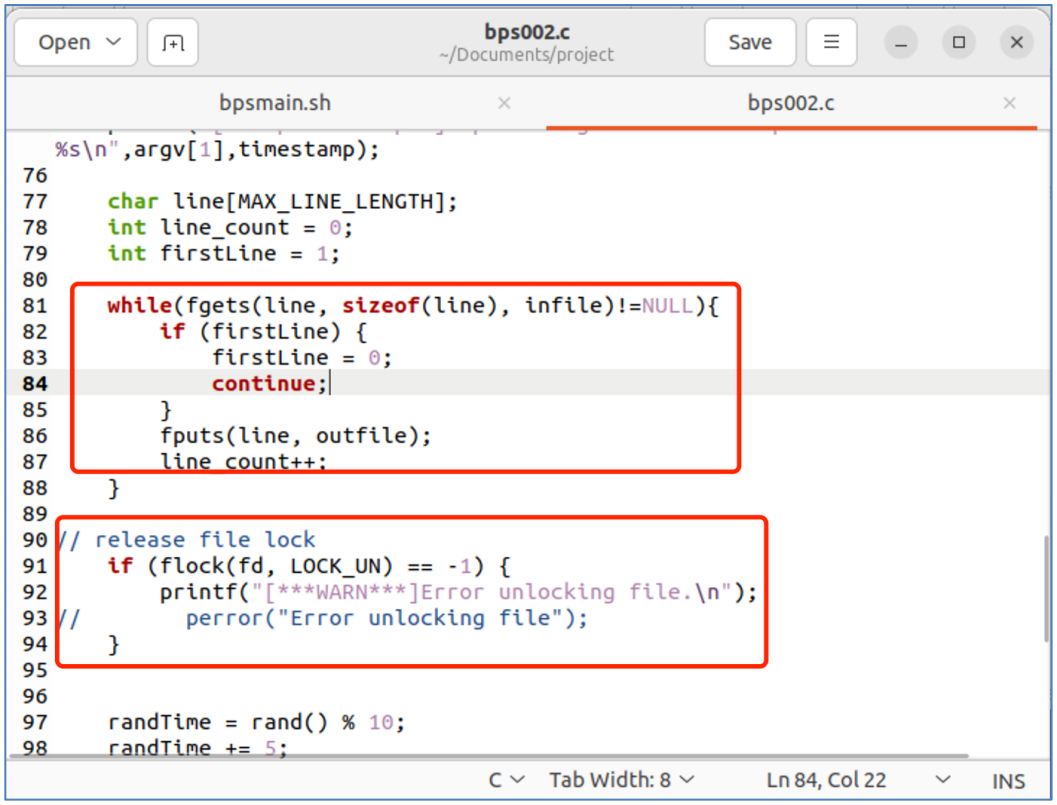
a. By calling batch programs in the background simultaneously in the batch scheduling script, the multi-process processing mechanism of the OS is simulated.



b. Through the file lock processing of the script and the program, the mutually exclusive process of multiple processes processing the same file at the same time is simulated and verified.



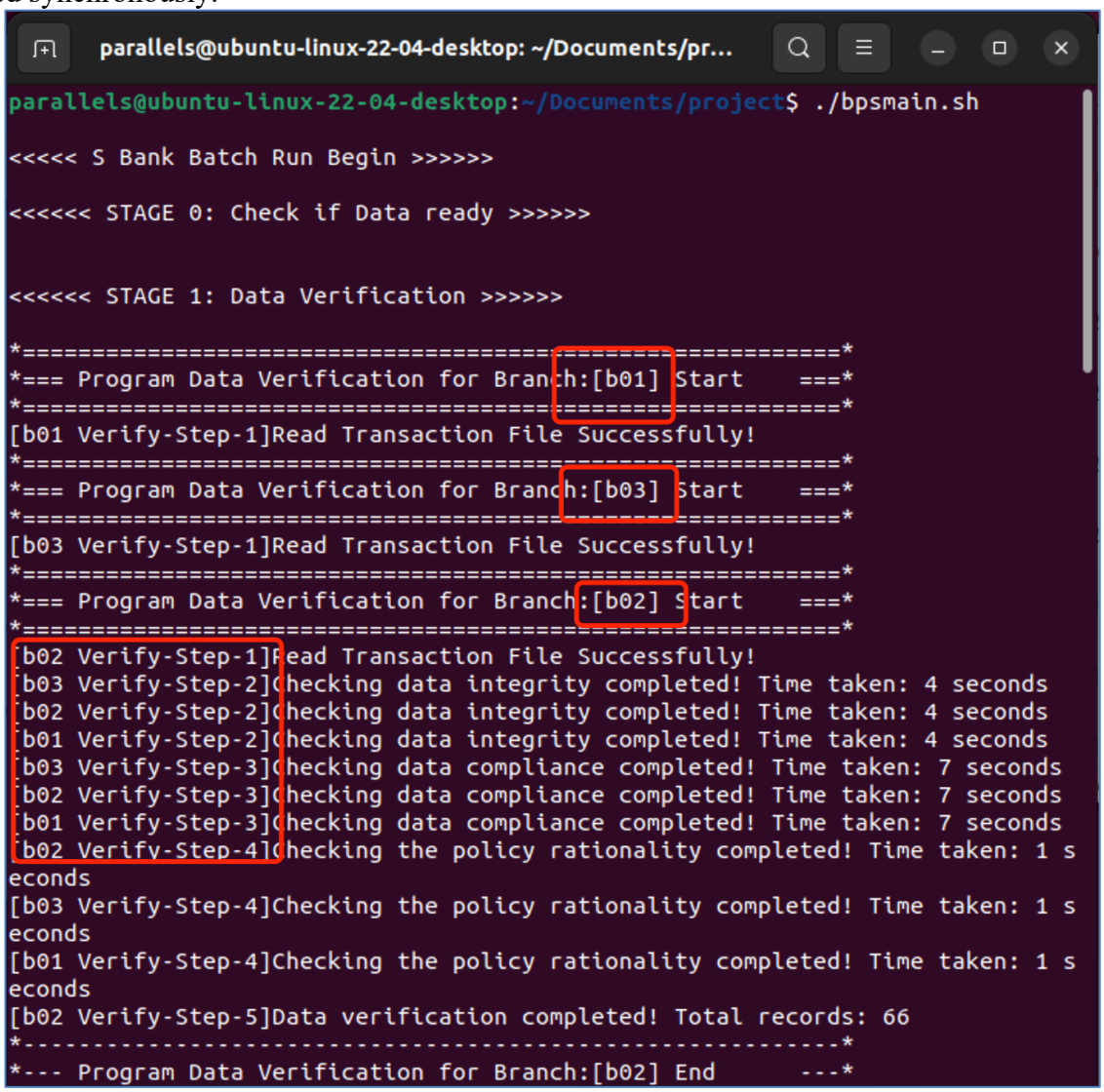



Open
bps002.c
~/Documents/project
Save
bpsmain.sh
bps002.c
%s\n",argv[1],timestamp);
76
77 char line[MAX_LINE_LENGTH];
78 int line_count = 0;
79 int firstLine = 1;
80
81 while(fgets(line, sizeof(line), infile)!=NULL){
82 if (firstLine) {
83 firstLine = 0;
84 continue;
85 }
86 fputs(line, outfile);
87 line_count++;
88 }
89
90 // release file lock
91 if (flock(fd, LOCK_UN) == -1) {
92 printf("[***WARN***]Error unlocking file.\n");
93 // perror("Error unlocking file");
94 }
95
96
97 randTime = rand() % 10;
98 randTime += 5;
C
Tab Width: 8
Ln 84, Col 22
INS

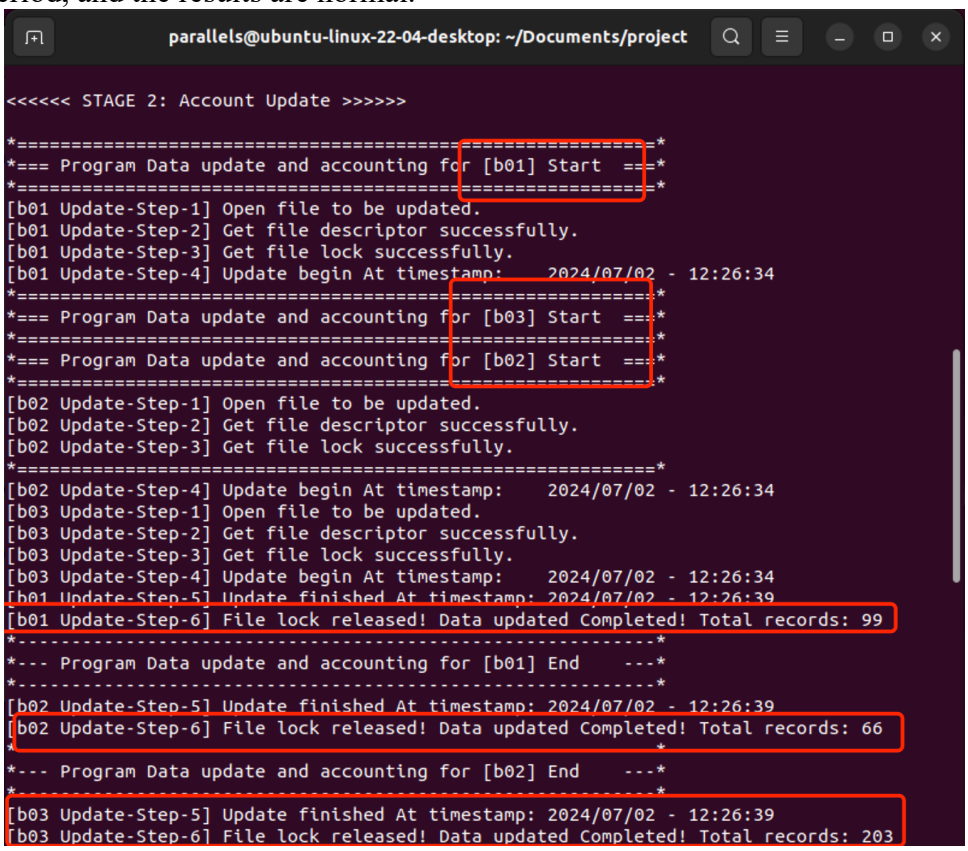
3. Simulation Results:

According to the terminal's standard output, the execution of the batch main script is normal, and the simulation verification goal envisioned by this project has been achieved. The specific information is as follows:

a. From the batch code execution sequence, the batch programs of the three branches are executed synchronously.



b. For the accounting files that have been locked, the three branches take turns to update them in the same period, and the results are normal.



4. Discussions:

During the project, we discovered that concurrent access and modification of the same files by multiple processes led to data corruption. To address this, we implemented file locking mechanisms, which successfully prevented data collisions and ensured the accuracy of our batch processing results.

The initial version of our batch processing script was slow, especially when handling large volumes of transaction data. We tackled this by optimizing our algorithm to reduce redundant operations and leveraging multi-threading to parallelize certain tasks, significantly improving execution speed and overall system efficiency.

In the early stages of development, our system lacked robust mechanisms for handling exceptions and recovering from errors, leading to incomplete or failed batch processes. We addressed this by implementing comprehensive error logging and a rollback mechanism, enhancing the reliability and resilience of our batch processing system.

During the project, we encountered bottlenecks in concurrent processing. By introducing more efficient thread management strategies and optimizing process scheduling, we overcame these bottlenecks and improved the efficiency of concurrent processing.

We recognized the importance of backing up data before updates to allow recovery in case of anomalies. We incorporated data backup and recovery functionalities into the system, ensuring data security and system stability.

IV. CONCLUSION

This project allowed us to apply and reinforce the knowledge gained from our coursework in operating systems. By simulating the batch processing operations of a bank, we were able to practice key concepts such as multi-process concurrency, file resource sharing, and locking mechanisms. This hands-on experience provided us with a deeper understanding of how these principles are implemented and managed in real-world scenarios. It has also enhanced our comprehension of resource allocation, process synchronization, and deadlock handling. Successfully completing this project has solidified our theoretical foundations and prepared us for more advanced studies and applications in operating systems, offering valuable practical insights that will be beneficial in future endeavors.

V. FUTURE WORK

This project is used to simulate the batch processing of banks to verify the knowledge points such as multi-process processing and file locks learned in the OS course. The design principle is to simplify as much as possible. Therefore, there are still the following future work that can be continuously optimized.

1. The bank data storage method is changed from files to databases. Index operations are added to improve the efficiency of batch processing.

2. The script function optimization can include two directions: one is to change the overall backup to step-by-step backup to support a more flexible re-run mechanism. The other is to expand the module program function to support more complex business processes.

REFERENCES

[1] A. S. Tanenbaum, "Modern Operating Systems," 4th ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014.

[2] Y. Paik, C. H. Kim, W. J. Lee and S. W. Kim, "Achieving the Performance of All-Bank In-DRAM PIM With Standard Memory Interface: Memory-Computation Decoupling," in IEEE Access, vol. 10, pp. 93256-93272, 2022

[3] Jin-Qiu Li and Di Liu, "Micro Bank System based on mainframe," 2009 International Conference on Apperceiving Computing and Intelligence Analysis, Chengdu, China, 2009

[4] J. Cui, J. Pang, Z. Shan and X. Liu, "The multi-threaded optimization of dynamic binary translation," 2011 Eighth International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Shanghai, China, 2011

[5] D. Linz, B. Ahmadi, R. Good, E. Musabandesu and F. Loge, "Multi-Threaded Simulation Optimization Platform for Reducing Energy Use in Large-Scale Water Distribution Networks with High Dimensions," 2020 Winter Simulation Conference (WSC), Orlando, FL, USA, 2020, pp. 704-714

[6] N. Xoxa, A. Mehilli, I. Tafa and J. Fejzaj, "Implementations of File Locking Mechanisms, Linux and Windows," *2015 12th International Conference on Information Technology - New Generations*, Las Vegas, NV, USA, 2015, pp. 756-757

[7] W. -k. Liao and A. Choudhary, "Dynamically adapting file domain partitioning methods for collective I/O based on underlying parallel file system locking protocols," *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, Austin, TX, USA, 2008, pp. 1-12

[8] M. Müller, T. Reggelin, I. Kutsenko, H. Zadek and L. S. Reyes-Rubiano, "Towards Deadlock Handling with Machine Learning in a Simulation-Based Learning Environment," *2022 Winter Simulation Conference (WSC)*, Singapore, 2022, pp. 1485-1496

[9] K. Miyagawa, K. Hasegawa, L. Tang and W. Wu, "Batch Scheduling and Robust Batch Scheduling to Minimize Maximum Lateness," *2022 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, Kuala Lumpur, Malaysia, 2022, pp. 0706-0710

[10]H. Cai, Y. Zhang, Y. Ding and L. Tong, "Research on Flexible Job Shop Batching and Scheduling Problem with Setup Times," *2018 10th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, Hangzhou, China, 2018, pp. 226-229

[11]B. Mikolajczak and C. A. Sefranek, "Integrating object oriented design with concurrency using Petri nets-A case study of a banking system and the Syroco-Macao environment," *2001 IEEE International Conference on Systems, Man and Cybernetics. e-Systems and e-Man for Cybernetics in Cyberspace (Cat.No.01CH37236)*, Tucson, AZ, USA, 2001, pp. 1559-1564

[12]B. Montrucchio, M. Rebaudengo and A. David Velasco, "Software-implemented fault injection in operating system kernel mutex data structure," *2014 IEEE 5th Latin American Symposium on Circuits and Systems*, Santiago, Chile, 2014, pp. 1-6

[13]X. -f. Yu and Wang Hui, "Formal verification of mutex of concurrent system," *International Conference on Automatic Control and Artificial Intelligence (ACAI 2012)*, Xiamen, 2012, pp. 1077-1082