Ira A. Fulton Schools of Engineering
Arizona State University
Engineering | School of Computing, Informatics, and Decision Systems Engineering

CSE 531: Distributed and Multiprocessor Operating Systems

Name: Varad Deshmukh

ASU ID: 1225369184

Project 1: gRPC Written Report

Name: Varad Vijay Deshmukh **ASU ID: 1225369184**

Problem Statement

The project aims to develop a simplified gRPC-based distributed banking system. Each customer is uniquely identified by an integer, and they exclusively interact with their corresponding branch. Only one customer can be active at any given time, submitting deposit, withdrawal, or balance query requests.

In this shared bank account system, the account balance is shared across all branches, necessitating synchronization of balances among them. To achieve this, branch-branch communication is essential to propagate customer-initiated transactions, including deposits and withdrawals.

The project requires defining a communication schema in a .proto file, facilitating RPC communications for customer-branch interactions through the gRPC interface.

The primary objective is to create a distributed banking system where customers interact with their designated branches to perform deposit and withdrawal transactions. Consistency in updating all branch replicas, the absence of concurrent updates, and an emphasis on security and scalability are key priorities.

Goal

The primary goal of this distributed system is to establish a distributed banking infrastructure where multiple customers can conduct deposit and withdrawal transactions across different branches. Every customer is assigned a unique ID and shares a common bank account. Importantly, the system operates under the assumption that concurrent updates do not occur. Each branch is responsible for maintaining consistent replicas of the account balance, ensuring that customer transactions are accurately reflected.

Key Components:

- Customer Processing: This segment includes the completion of functions within Customer processes which handle the task of the creation of customer stubs and establishment of gRPC communication for all customer processes. This is vital for setting up and managing customer interactions.

- Message Processing: Branch processes are responsible for receiving and handling requests from Customer processes. These requests encompass methods such as query, deposit, and withdraw, resulting in adjustments to the branch's balance.

- Inter-Branch Interaction: Branch processes engage with other branches to propagate updates, including the methods Propagate_Withdraw and Propagate_Deposit. These methods enable branches to synchronize their replica account balances and ensure consistency across the system.

Setup And Run

Setup: To setup a proper environment to run the program follow the steps below:

1. Install Ubuntu Linux on Windows:
    - Ubuntu Version: 22.04 LTS

2. Activate Windows Subsystem for Linux 2 (WSL2):
    - Configure WSL2 for use with Ubuntu Linux.

3. Install Python 3.10.6:
    - In the Ubuntu Linux environment, install Python version 3.10.6.

4. Install Python Libraries:

    Inside the Ubuntu environment, install the following Python libraries:

    - grpcio (version: 1.59.0)
    - grpcio-tools (version: 1.59.0)
    - protobuf (version: 4.24.3)

5. Download and Load Zip File:
    - Download the provided zip file from the submission and load it into the Ubuntu Linux system.

Run: To run the program and get output follow the steps below:

1. Run command to get into project folder (**CSE531_Project_1_Varad**)

    - cd CSE531_Project_1_Varad

2. Now you can build the proto file (Optional). **comm.proto** is the proto file for the Project

    - python3 -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. comm.proto

3. Now, first run the server file (**server_branch.py**) with a json file input.

    - python3 server_branch.py input.json

4. Next, run the client file (**customer_client.py**) with a json file input.

    - python3 customer_client.py input.json

5. After execution is complete, an output json file will be generated and stored in the folder **CSE531_Project_1_Varad.**

Implementation Processes

A. Customer Programs:

1. customer.py:
   Customer.py contains customer processes which are help in creating stubs
    a. creatstubs ():
       This function takes input Json file and then processes it to create list of all customer process order by event id. Then returns this list which will used for stub communication in customer_client.py.
2. customer_client.py:
    - The primary purpose of this program is to serve as the orchestrator for executing customer processes. It establishes **gRPC communication** with the **`server_branch.py`**. The program initiates instances of the **customer class** and utilizes the **`createstubs()`** function to build a list containing essential information, including customer ID, event ID, interface, and transaction amount for each event.
    - Subsequently, the program iterates through this list, creating gRPC stubs using the protocol buffer generated code. These stubs invoke the **`Customer_Process`** method, which is defined in the server. The response obtained from this

interaction includes the new account balance for the customer and information on the success or failure of the process.

- Lastly, the program collates all the outputs, and these results are used to generate a output JSON file that encompasses the execution outcomes. This JSON file serves as a comprehensive record of output of all Customer Processes.

B. Branch Process:

Following all are functions stored in Branch class inside branch.py:

1. createstub_branch:
   This function is used to process the input json file to create list of branchs wh=ith their branch_id and balance. This list will futher used to create stubs and server port corresponding to their respective branch ids.
2. msgDelivery:
   The main objective of this function is to intermediary between branch functions and server communication. msgDelivery service information from server_branch.py customer_process which include the requested interface. Then inside msgDelivery based on the interface one out of deposit, withdraw or query functions is called. MsgDelivery returns with updated balance.
3. Deposit:
   The deposit function receives money from msgDelivery and then adds that amount to the existing balance of branch and returns the new balance.
4. Withdraw:
   The withdraw function receives money from msgDelivery and then subtracts that amount from the existing balance of branch and returns new balance.
5. Query:
   The main task of the query function is to return the existing branch balance.
6. Update_Balance:
   Update_Balance is an essential function for branch-to-branch communication. It is called inside BranchInfo which grpc branch to branch communicator and receives interface, money and branch_id. It then depends on interface either call Propogate_Deposit and Propogate_Withdraw.
7. Propagate_Deposit:
   This function receives branch_id and money need to be added to branch balance. It calls deposit function and updates branch balance of the branch.
8. Propagate_Withdraw:
   This function receives branch_id and money need to be subtracted from branch balance. It calls withdraw function and updates branch balance of the branch.

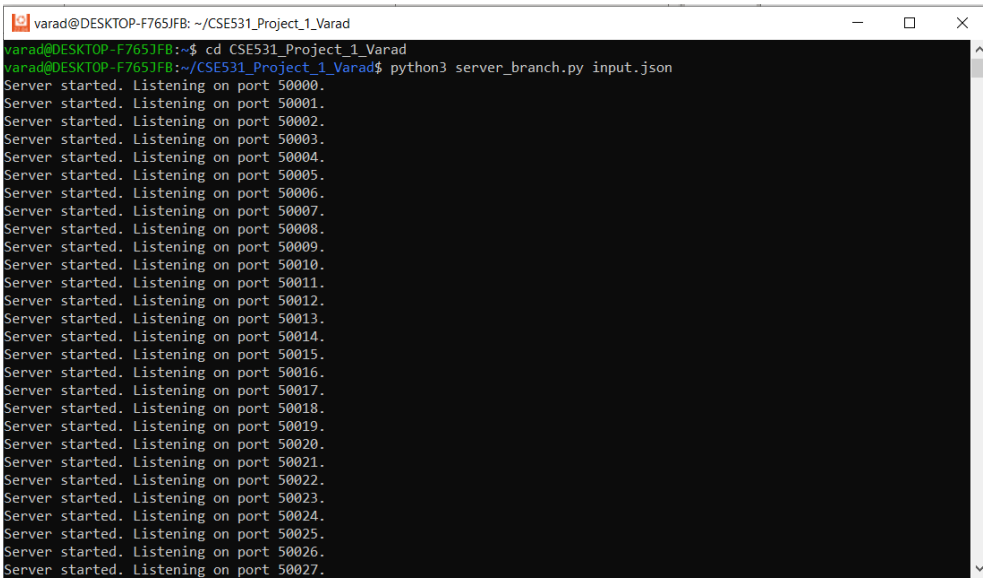
C. Proto file and RPC Functions:

1. Customer_Process
   Customer_Process is a rpc service for Customer-to-Branch communication made in proto file. It is defined in server_branch.py and is called inside customer_client. It receives customer process information and passes it to branch function. The

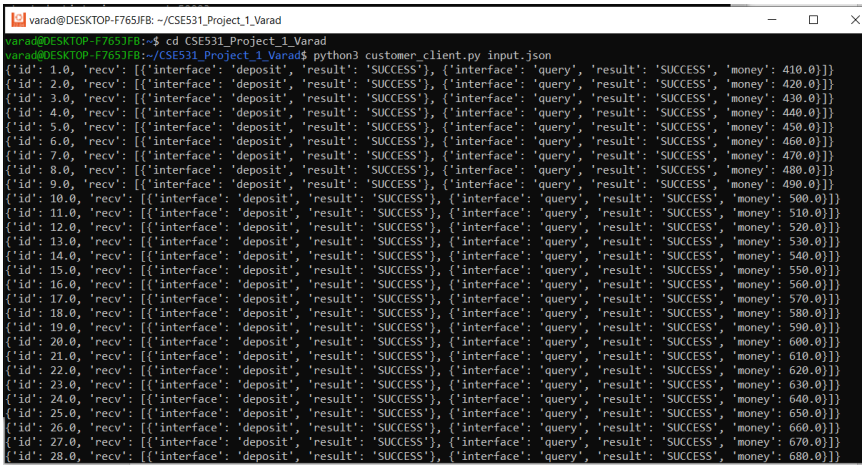
response of Customer_Process includes the new balance for the customer process and status of Success of process.

2. BranchInfo
   BranchInfo is a rpc service for Branch-to-Branch communication made in proto file. It is defined in server_branch.py and is called inside Customer_Process to start communication from one branch to all other branches to update their balances based on interface via Propagate_Deposit or Propagate_Withdraw.

Results



Screenshot of Server_Branch.py running where its starts multiple server ports which listening for rpc communication



Screenshot of Output of customer_client.py where the output is result of customer processes.

output.json

```
[
  {
    "id": 1.0,
    "recv": [
      {
        "interface": "deposit",
        "result": "SUCCESS"
      },
      {
        "interface": "query",
        "result": "SUCCESS",
        "money": 410.0
      }
    ]
  },
  {
    "id": 1.0,
    "recv": [
      {
        "interface": "deposit",
        "result": "SUCCESS"
      },
      {
        "interface": "query",
        "result": "SUCCESS",
        "money": 410.0
      }
    ]
```

Screenshot of output Json which complied list of results of customer processes.