

## CSE 531: Distributed and Multiprocessor Operating Systems

---

# Diego Coss Written Report

### Problem Statement

Running a bank is hard in the 21<sup>st</sup> century. One bank may have multiple locations in a region. Customers can be expected to visit a few locations. This comes with some issues. Keeping an accurate record of their balance between locations is a must. If the branches are not synchronized then customers can double-spend or spend money that they don't have. Also, if they deposit money in one location then they should be able to take it out at another location. Thus an efficient way of communicating between the branches to provide quality service and accurate balances. Also, some locations may use different hardware and operating systems. To eliminate the hassle of communication we need a standardized way of talking between the branches.

### Goal

Use gRPC stubs to communicate between the branches of the bank. We should have a few branches of the bank open and a customer to interact with each branch. A customer should be able to query, deposit, and withdraw from all branches. If the deposit or withdrawal is requested then the branch should update its balance as well as all other branch balances. Our main method of communication will be the `MsgDelivery` method to execute the three main actions. `MsgDelivery` will also facilitate `propagate_deposit` and `propagate_withdraw` to update the other branches. The goal includes the completion of the `customer.py`, `branch.py`, and `week3.proto` file.

### Setup

If using python then make sure python is up to date. Python version should be at least 3.6 but 3.8 was used. First update python: `python -m pip install --upgrade pip`. Then update pip: `python -m pip install --upgrade pip`. Then install grpc: `python -m pip install grpcio`. Finally, install tools: `python -m pip install grpcio-tools`. The quick start guide is fairly useful for getting up to speed on how gRPC works and how to implement it. Open

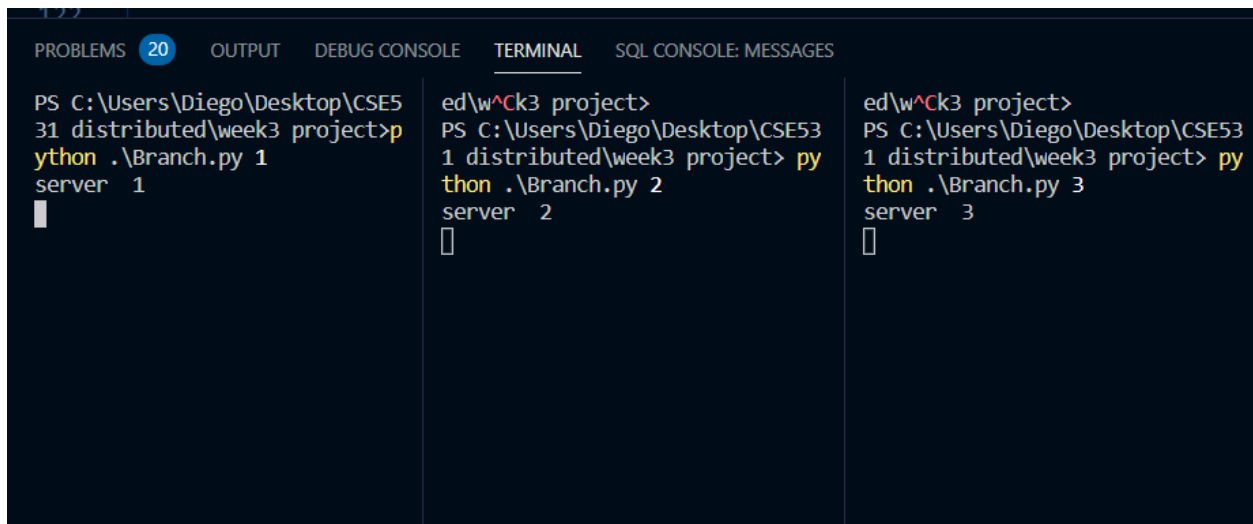
<https://grpc.io/docs/languages/python/quickstart/>. Once python is setup you should make a folder for your `client.py`, `server.py`, and `proto` file. The following command is useful for compiling the gRPC stub code: `python -m grpc_tools.protoc -I . --python_out=. --grpc_python_out=. week3.proto`.

## Implementation Processes

Week3.proto should be easy to implement. Following the guide, we state syntax is proto3, service, and messages. The service is simple we have one interface to implement and that is MsgDelivery. MsgDelivery takes a request\_service message and returns a success message. The request\_service has attributes string request and int64 money. The request will tell the branch which of the 5 requests to process, eg deposit, etc. The success message has attributes bool success and int 32 money. So, if the customer sends request\_service with request equal to "query" and money equal to 0 then the branch should return success and the account balance. Compile the proto file and import those files in the following python files. Next, we need to implement the Customer.py. You can use python processes but I opted to use terminal commands and if statements. So python customer.py 1 would send customer 1 requests to branch 1. Each customer needs to open a localhost channel to their respective branch ports : `channel = grpc.insecure_channel('localhost:50051')`. Following that we need to create a stub to be able to communicate with the branch: `stub = week3_pb2_grpc.RPCServicerStub(channel)`. Finally, the query request can be made like so: `response = stub.MsgDelivery(week3_pb2.request_service(request='query', money=0))`. You may also choose to display your response if needed. The last file to implement is branch.py. If python branch.py 2 is executed then it should set up the second server at port 50052. First setup the servicer : `server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))`. After that, add the branch object to the server : `week3_pb2_grpc.add_RPCServicerServicer_to_server(Branch(1, start_balance, end_id - start_id + 1), server)`. Next, add the port to the server, note the last number should be the server id: `server.add_insecure_port(':::50052')`. Next, the server should be started and wait for termination. The last part of this is implementing the MsgDelivery and connect the branches. Connecting the branches to all other branches is similar to how the customer was implemented. Lastly, the interface needs to parse the 5 commands query, deposit, withdraw, propagate deposit, and propagate withdraw. A bank should check that a customer does not withdraw more than they have, propagate withdraws to the other banks, and return if it succeeded. Once every step is completed then you can run the 3 branches and 3 customers. Each customer has predefined actions per the input file: customer 1 queries, customer 2 deposits 170 then queries, and customer 3 withdraws 70 and queries. Branches should all start with 400 as the balance.

## Results

After the 3 servers were started we get this.



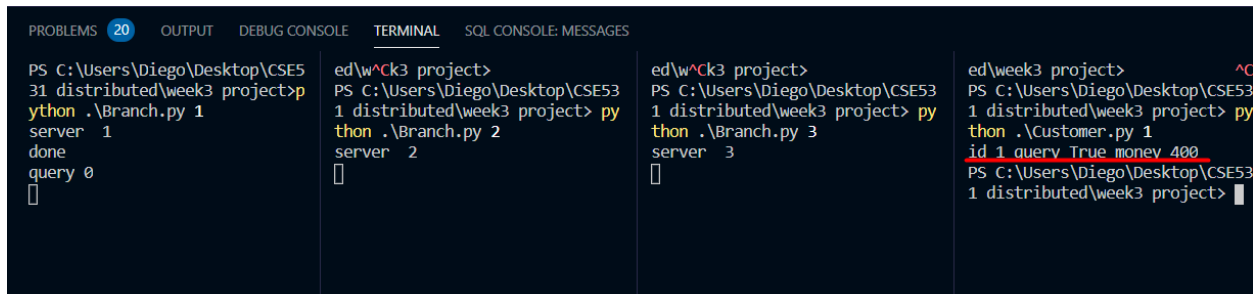
```
PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Branch.py 1
server 1

ed\w^ck3 project> PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Branch.py 2
server 2

ed\w^ck3 project> PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Branch.py 3
server 3
```

Figure 1: All servers started.

Then you can execute the three customers' requests. Customer 1 action is as follows.



```
PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Branch.py 1
server 1
done
query 0

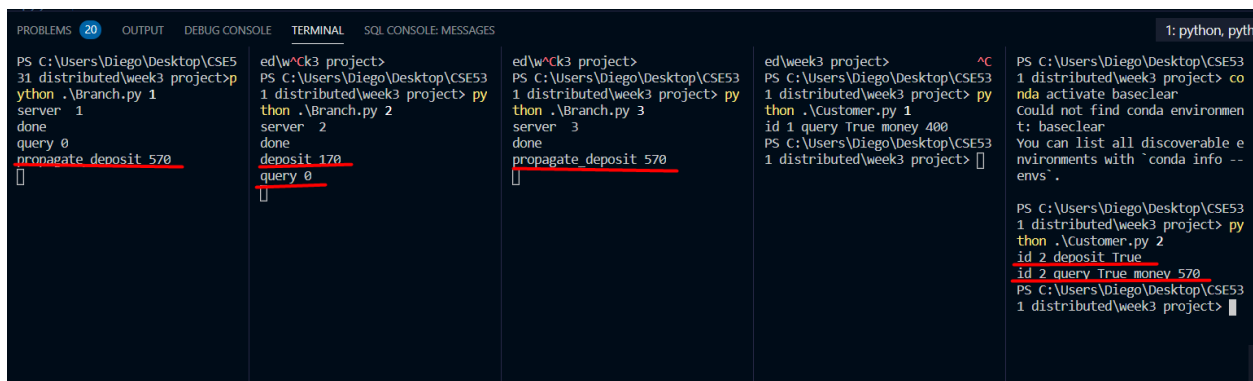
ed\w^ck3 project> PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Branch.py 2
server 2

ed\w^ck3 project> PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Branch.py 3
server 3

ed\week3 project> PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Customer.py 1
id 1 query True money 400
PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project>
```

Figure 2: Customer 1 queries branch 1.

Here we see that the balance amount is correct. Customer 2 actions are as follows.



```
PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Branch.py 1
server 1
done
query 0
propagate deposit 570

ed\w^ck3 project> PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Branch.py 2
server 2
done
deposit 170
query 0

ed\w^ck3 project> PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Branch.py 3
server 3
done
propagate deposit 570

ed\week3 project> PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Customer.py 1
id 1 query True money 400
PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project>

PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Customer.py 2
id 2 deposit True
id 2 query True money 570
PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project>
```

Figure 3: Customer 2 deposits 170 then queries branch 2.

In Figure 3 we see that branch 2 has a deposit request for 170 and is successful. The initial balance was 400. Add 170 dollars to the balance and the query comes out to be 570. So deposit works as intended. Branch 2 then requests the other branches to update their balance with the propagate deposit. This then synchronizes all balances to be the same. Lastly, customer 3 executes its requests.

```

PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Branch.py 1
server 1
done
query 0
propagate deposit 570
propagate withdraw 500
[]

PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Branch.py 2
server 2
done
deposit 170
query 0
propagate withdraw 500
[]

PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Branch.py 3
server 3
done
propagate deposit 570
withdraw 70
query 0
[]

PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Customer.py 1
id 1 query True money 400
PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project>

PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Customer.py 2
id 2 deposit True
id 2 query True money 570
PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project>

PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project> python .\Customer.py 3
id 3 withdraw True
id 3 result True money 500
PS C:\Users\Diego\Desktop\CSE531 distributed\week3 project>
  
```

Figure 4: Customer 3 withdraws 70 and queries branch 3. Branch 3 requests branches 1 and 2 to update their balances to the new sum.

So our goal is reached. Customers can query, deposit, and withdraw from any of the branches and have those changes reflected in the balances. gRPC is an effective way of communicating between clients and servers. The advantages of using gRPC are platform, hardware, and os independence. These make coding complex systems much easier.