Trabalho de Fundamentos de Sistemas Paralelos e Distribuídos

Rita Rezende Borges de Lima

May 2022

1 Introdução

Esse relatório se refere ao segundo trabalho da disciplina de Fundamentos de Sistemas Paralelos e Distribuídos, cursada no primeiro semestre letivo de 2022. Seu intuito era implementar dois serviços, um de banco e um de loja utilizando *gRPC* com a linguagem Python.

2 Implementação do Serviço de Banco

O serviço de banco foi dividido em dois arquivos, um para o servidor e uma para o cliente que acessa o serviço gerado pelo primeiro. O servidor do banco foi modelado com uma classe BankServer que por sua vez herda as especificações dos arquivos gerados pelo arquivo `bank.proto`. O arquivo possui uma função `Serve()`, chamada pela *main* que liga o objeto BankServer a um servidor *gRPC* faz uma conexão TCP, inicia o servidor e espera até que um evento de parada seja chamado.

2.1 A classe Bank Server

A classe `BankServer` possui os seguintes métodos:

- **Um construtor:** que armazena o evento de parada e o caminho para o arquivo das carteiras em atributos da classe e chama uma função que lê este último.
- **read database():** função destinada a ler linha a linha do arquivo de carteiras, cada uma dessas é armazenada em um dicionário, atributo da classe, sendo a chave da carteira a chave e o saldo o valor do dicionário.
- **save changes():** função que escreve as carteiras do atributo dicionário da classe no arquivo que representa o banco de dados.
- **balance():** função declarada no arquivo proto. Verifica se o identificador de carteira recebido como parâmetro é valido e retorna seu valor armazenado no dicionário, caso seja inválido, retorna -1.
- **payment():** função também declarada no arquivo proto. Verifica se o identificador de carteira recebido como parâmetro é valido e se o valor armazenado no dicionário é maior que o valor recebido como parâmetro, neste caso subtrai o valor do dicionário, cria um token pra transação e salva este. Caso a transação seja inválida, retorna erro, -1 ou -2.
- **transfer():** função também declarada no arquivo proto. Verifica se o identificador de carteira e o token de transação recebidos como parâmetros são válidos e se o valor armazenado no dicionário de transação é igual o valor recebido como parâmetro, neste caso incrementa o valor do dicionário de cliente e remove a transação. Caso a transação seja inválida, retorna erro, -1, -2 ou -3.
- **end of work():** função de término do serviço, chama a função *save changes()*, chama o evento de término do servidor e retorna a quantidade de carteiras do sistema.

2.2 Cliente de Banco

O aquivo cliente de banco possui uma função `run()` que abre a conexão com o serviço de banco e lê comandos da entrada padrão até receber o comando de parada. Os comandos possíveis são:

- **S:** Uma operação de saldo que chama a função `get balance()` que por sua vez chama a função do servidor `balance ()` para o identificador de carteira passado como parâmetro global e retorna seu saldo.
- **O:** Uma operação de pagamento que gera uma ordem de pagamento chamando a função `make payment()` que por sua vez chama a função do servidor `payment()` e salva essa ordem de pagamento em um dicionário com a função `save payment()`.
- **X:** Uma operação de transferência que de posse de uma ordem de pagamento chama a função `make transfer()` que por sua vez chama a função do servidor `transfer()` e retorna seu status.
- **F:** Uma operação de término de processamento que chama a função do servidor `end of work()` e retorna a quantidade de contas salvas.

3 Implementação do Serviço de Loja

O serviço de loja também foi dividido em dois arquivos, um para o servidor e um para o cliente. Este último acessa o serviço de loja gerado pelo primeiro arquivo e o serviço de banco descrito na sessão anterior. O servidor da loja foi modelado com uma classe *StoreServer*. Esta herda as especificações dos arquivos gerados pelo arquivo `store.proto`. O arquivo possui uma função `Serve()`, chamada pela *main* que liga o objeto *StoreServer* a um servidor *gRPC* faz uma conexão TCP, inicia o servidor e espera até que um evento de parada seja chamado.

3.1 A classe Store Server

A classe *StoreServer* possui os seguintes métodos:

- **Um construtor:** que armazena atributos da execução corrente como o valor do serviço, a chave da carteira no banco e o saldo da loja, e atributos de serviço como o evento de parada e o *stub* de banco.
- **price():** função declarada no arquivo proto. Apenas retorna o valor do produto ou serviço ofertado pela loja em execução, este é armazenado no atributo de classe `product value`.
- **sales():** função também declarada no arquivo proto. Recebe uma ordem de transação e chama a função de transferência, transfer, do servidor de banco. Caso esta não retorne erro, o saldo da loja é incrementado, caso contrário é retornado erro.
- **end of work():** função de término do serviço, chama o evento de termino do servidor, fecha o canal do servidor de banco e retorna o saldo da loja.

3.2 Cliente de Loja

O aquivo cliente de loja possui uma função `run()` que abre a conexão com o serviço de banco e o serviço de loja lê comandos da entrada padrão até receber o comando de parada. Os comandos possíveis são:

- **P:** Uma operação que retorna o saldo da carteira atual e o preço do produto da loja atual. Para isso, chama a função `get balance()` que por sua vez chama a função do servidor `balance ()` para a carteira passada como parâmetro global e retorna seu saldo.
- **C:** Uma operação de compra que chama a função *make sale()*. Esta, gera uma ordem de pagamento chamando a função do servidor payment() e imprime na tela seu status. Caso esta retorne sem erros, utiliza a ordem de pagamento gerada para chamar a função `transfer()` e por fim também imprime o status desta.
- **T:** Uma operação de término de processamento que chama a função de servidor `end of work()` para bank e store e retorna a quantidade de contas salvas e o saldo da loja.