Banco Multithread

Relatório de implementação do trabalho 1
INE5645 - Programação Paralela e Distribuída

Giovane Pimentel de Sousa - 22202685
Guilherme Henriques do Carmo - 22201630
Isabela Vill de Aquino - 22201632

November 3, 2024

Seção 1

Introdução

O trabalho solicitado consiste na implementação de um servidor multithreaded para realizar operações bancárias usando um pool de threads e o modelo de produtor/consumidor.

Decisões de Implementação

A implementação foi realizada em **Golang** 1.23, utilizando apenas módulos da biblioteca padrão, em principal "*sync*" para toda a estrutura não sequencial, portanto, não é necessária a instalação de nenhum pacote adicional. Pensamos em utilizar semáforos para controle de workers livres, porém em Go a forma oficial é com channels, que implementa dentro de si o controle de concorrência, o que era uma limitação estabelecida na definição do trabalho, portanto, decidimos por utilizar mutex e condition em toda estrutura de sincronização.

Foram esclhidas quatro parametrizações principais: **número de threads trabalhadoras**, **número de clientes**, **número máximo de requisições** que cada cliente efetuará e por fim o **tempo de serviço** (tempo de espera de cada operação).

Os clientes geram requisições em intervalos aleatórios e arbitrários de 0 a 5 segundos.

A variável *numClients* foi definida para ser tanto relacionada à quantidade de clientes quanto de contas existentes no banco.

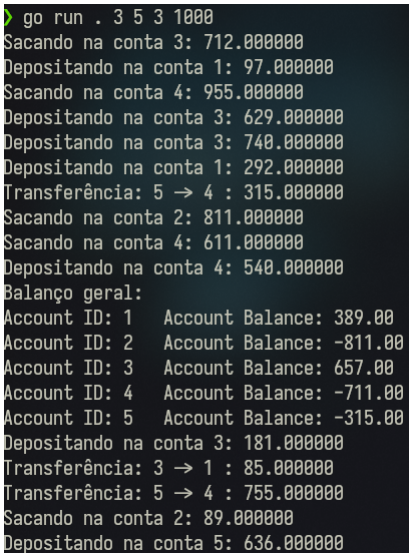
Foi escolhido pelo grupo um ponto de parada: **X** clientes atingirem **N** (maxRequests) requisições definida por parâmetro. Portanto, quando o número de requisições atendidas pelos workers chegarem a X * N, o programa finaliza.

Como executar

A sintaxe básica para execução é:

```
$ go run . [numWorkers] [numClients] [maxRequests] [serviceTimeInMiliseconds]
# Então por exemplo:
$ go run . 1 3 1 1000
$ go run . 2 5 2 300
```

Exemplos de execução:



Conclusões

Ao variar os parâmetros do sistema, observou-se que o número de threads no pool influencia diretamente a capacidade de processamento simultâneo do servidor. Um pool maior tende a reduzir o tempo de resposta das requisições sob cargas mais altas, mas aumenta o uso de recursos e a complexidade do gerenciamento de sincronização. Esses experimentos permitiram entender a importância de uma configuração equilibrada para otimizar o desempenho do servidor multithreaded conforme os requisitos de cada cenário.