Guilherme Baptista Da Silva
Nathanael Dayan Cella

Sistema de Processamento Paralelo de Requisições a um Banco de Dados

Universidade do Vale do Itajaí – UNIVALI
Faculdade de Ciência de Computação

Brasil
2025

Sumário

0.1 Enunciado do Projeto

Este projeto tem como objetivo o desenvolvimento de um sistema de processamento paralelo de requisições a um banco de dados simulado. O sistema faz uso de comunicação entre processos (IPC) via FIFO (pipe nomeado) e processamento multithread com controle de concorrência por meio de `mutex`. As operações suportadas incluem inserção, remoção, busca e atualização de registros.

0.2 Contextualização e Objetivo

Em sistemas reais de banco de dados, múltiplas requisições são processadas simultaneamente por diversos usuários. Para simular esse comportamento, o sistema proposto implementa um servidor concorrente que gerencia um "banco de dados" leve (vetor de registros em memória, persistido em arquivo), enquanto um cliente envia comandos ao servidor através de um canal de IPC.

A solução desenvolvida visa consolidar os conceitos de concorrência, paralelismo e sincronização em sistemas operacionais, aplicando-os em um cenário prático que simula o funcionamento interno de um SGBD (Sistema Gerenciador de Banco de Dados).

0.3 Implementação do Sistema

0.3.1 Estrutura de Dados

```
typedef struct {
    int id;
    char nome[50];
} Registro;
```

0.3.2 Funcionalidades Implementadas

- **INSERT id=X nome=Y**: Insere um novo registro.
- **DELETE id=X**: Remove um registro existente.
- **SELECT id=X**: Consulta e retorna um registro.
- **UPDATE id=X nome=Y**: Atualiza o nome de um registro.

0.3.3 Componentes

- **Cliente**: Envia requisições ao servidor por meio de pipe.

- **Servidor**: Recebe requisições, processa em threads e manipula o banco com mutex.
- **Banco de Dados**: Arquivo `banco.txt`, persistido após cada modificação.

0.3.4 Controle de Concorrência

O acesso ao vetor de registros (`bancoDados`) é controlado por `std::mutex`, garantindo que apenas uma thread altere os dados simultaneamente.

0.3.5 Threads

Cada requisição recebida é processada por uma nova thread (`std::thread`), utilizando `detach()` para não bloquear o loop principal.

0.4 Execução e Resultados

Comandos de teste utilizados:

```
INSERT id=1 nome=Lucas
INSERT id=2 nome=Mariana
SELECT id=1
UPDATE id=1 nome=João
DELETE id=2
```

Saída esperada no servidor:

```
[INSERT] Registro inserido.
[INSERT] Registro inserido.
[SELECT] id=1 nome=Lucas
[UPDATE] Registro atualizado.
[DELETE] Registro removido.
```

Conteúdo final do banco.txt:

```
1 João
```

0.5 Discussão

A solução proposta atende aos requisitos do enunciado, simulando de forma eficaz um banco de dados leve com múltiplos acessos concorrentes. A criação de threads para

cada requisição permitiu um bom desempenho e paralelismo, e o uso de `mutex` assegurou a integridade dos dados compartilhados.

A separação entre processo cliente e servidor via FIFO oferece um exemplo prático de IPC. A solução pode ser estendida para adicionar retorno direto da resposta ao cliente via um segundo FIFO ou outro mecanismo IPC.

0.6 Repositório do Projeto

<https://github.com/Nathacella/Trabalho_M1/>