



UNIVERSIDADE DO VALE DO ITAJAÍ
CIÊNCIA DA COMPUTAÇÃO
SISTEMAS OPERACIONAIS

Avaliação M1 - IPC, Threads e Paralelismo

Alunos:

Lucas Rodrigues Pinheiro
Gustavo Cadore da Silva
Pedro Henrique Souza dos Santos

Professor:

Michael Douglas Cabral Alves

Itajaí, 12 de Abril de 2025

Enunciado do Projeto.....	3
Explicação e contexto da aplicação.....	3
Resultados com as simulações.....	3
Códigos importantes.....	7
Análise e discussão.....	8
Referências.....	10

Enunciado do Projeto

O enunciado do projeto proposto tem como objetivo desenvolver um sistema de processamento paralelo de requisições a um banco de dados, simulando o funcionamento interno de um gerenciador de banco utilizando conceitos fundamentais de IPC, threads, concorrência e paralelismo. A implementação deve ser feita em até tríos, com o uso de uma linguagem de programação que suporte esses recursos, como C, C++, Java, Python, etc.

O projeto precisa apresentar um processo cliente, responsável pelo envio de requisições de consulta e/ou inserção via IPC (pipe ou memória compartilhada), um processo servidor, responsável por receber as requisições e usar threads para processá-las em paralelo e as threads precisam utilizar mutex ou semáforo para garantir acesso seguro a uma estrutura compartilhada que simula um banco de dados.

Explicação e contexto da aplicação

Nos tempos atuais, banco de dados são essenciais para sistemas reais, sendo acessados por múltiplos usuários ou por outros sistemas.

Para garantir um funcionamento correto, um banco de dados precisa lidar com vários usuários e processos tentando acessar o mesmo, ou seja, precisa lidar com concorrência. Precisa lidar com diversos processos tentando realizar alterações simultâneas em um determinado dado podendo levar a inconsistências, ou seja, sincronização de acesso e ele também precisa considerar que processos podem precisar comunicar-se entre si para trocar informações, ou seja, realizar comunicações entre Processos (IPC).

Resultados com as simulações

INSERT -

Imagem 1 - Inserção de um novo registro no banco de dados.

```
C:\Users\pepeh\Desktop\Trabalho SO 2>cliente.exe
Servidor não está rodando. Tentando iniciar...
Servidor iniciado com sucesso. Aguardando conexão...
Servidor aguardando conexão... teste123
Cliente conectado!

Digite uma requisição:
INSERT nome          -> Adiciona novo registro
DELETE id            -> Remove um registro pelo ID
SELECT *              -> Mostra todos os registros
SELECT id             -> Mostra o registro com o ID especificado
UPDATE id novo_nome  -> Atualiza o nome do registro
SAIR                  -> Encerra o cliente
>> INSERT Lucas

Resposta do servidor:
[Thread ID: 9412] Registro inserido com ID: 10
```

Imagem 2 - Banco.txt após a inserção (imagem 1).

```
banco.txt
1 1;MARIA
2 2;JOAO
3 4;BOB
4 5;paula
5 6;teste
6 7;cadore
7 8;teste
8 9;teste
9 10;Lucas
0
```

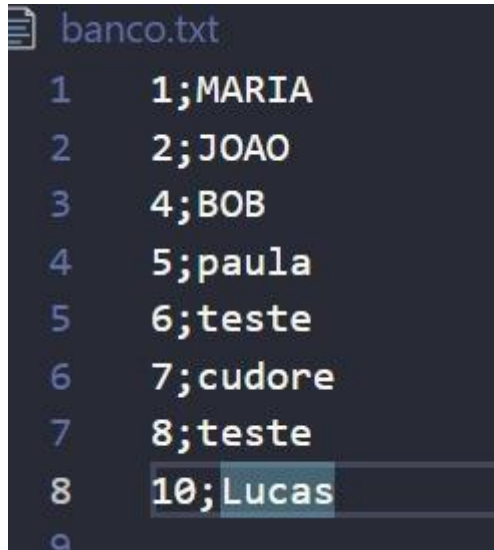
DELETE -

Imagem 3 - Delete de um registro no banco de dados.

```
Digite uma requisição:
INSERT nome          -> Adiciona novo registro
DELETE id            -> Remove um registro pelo ID
SELECT *             -> Mostra todos os registros
SELECT id            -> Mostra o registro com o ID especificado
UPDATE id novo_nome  -> Atualiza o nome do registro
SAIR                 -> Encerra o cliente
>> DELETE 9

Resposta do servidor:
[Thread ID: 9412] Registro com ID 9 removido.
```

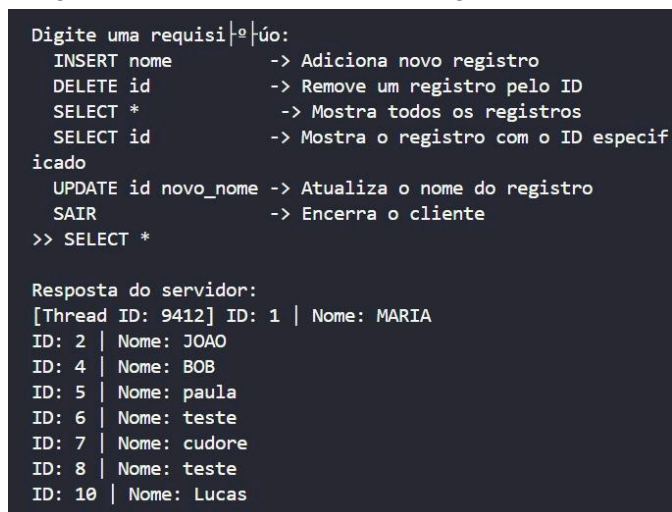
Imagem 4 - Banco.txt após o delete (Imagem 3).



```
banco.txt
1    1; MARIA
2    2; JOAO
3    3; BOB
4    4; paula
5    5; teste
6    6; cudore
7    7; teste
8    10; Lucas
9
```

SELECT -

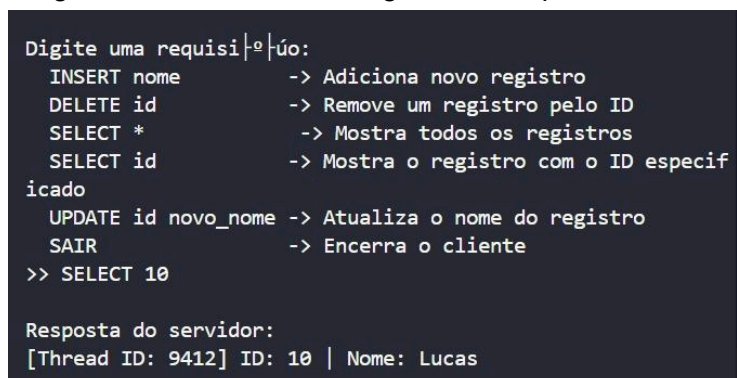
Imagem 5 - Selecionar todos os registros no banco de dados.



```
Digite uma requisição:
INSERT nome      -> Adiciona novo registro
DELETE id        -> Remove um registro pelo ID
SELECT *         -> Mostra todos os registros
SELECT id        -> Mostra o registro com o ID especificado
UPDATE id novo_nome -> Atualiza o nome do registro
SAIR             -> Encerra o cliente
>> SELECT *

Resposta do servidor:
[Thread ID: 9412] ID: 1 | Nome: MARIA
ID: 2 | Nome: JOAO
ID: 4 | Nome: BOB
ID: 5 | Nome: paula
ID: 6 | Nome: teste
ID: 7 | Nome: cudore
ID: 8 | Nome: teste
ID: 10 | Nome: Lucas
```

Imagem 6 - Selecionar um registro em específico no banco de dados.



```
Digite uma requisição:
INSERT nome      -> Adiciona novo registro
DELETE id        -> Remove um registro pelo ID
SELECT *         -> Mostra todos os registros
SELECT id        -> Mostra o registro com o ID especificado
UPDATE id novo_nome -> Atualiza o nome do registro
SAIR             -> Encerra o cliente
>> SELECT 10

Resposta do servidor:
[Thread ID: 9412] ID: 10 | Nome: Lucas
```

UPDATE -

Imagem 7 - Banco.txt antes do Update.

```
1;LUCAS
2;LARISSA
3;AJUSTAR AQUI
```

Imagem 8 - Update de um registro no banco de dados.

```
Digite uma requisição:
INSERT nome          -> Adiciona novo registro
DELETE id            -> Remove um registro pelo ID
SELECT *             -> Mostra todos os registros
SELECT id            -> Mostra o registro com o ID especificado
UPDATE id novo_nome  -> Atualiza o nome do registro
SAIR                 -> Encerra o cliente
>> UPDATE 3 JOANA
```

Imagem 9 - Banco.txt após o Update.

```
Banco.txt
1;LUCAS
2;LARISSA
3;JOANA
```

Códigos importantes

As próximas 3 (três) imagens abordam a criação dos processos, atendendo os requisitos “Um processo cliente enviar requisições” e “Um processo servidor recebe as requisições”.

Imagem 10 - Criação de um processo.

```
// Tenta criar o processo do servidor
if (!CreateProcessA(
    NULL,
    const_cast<char*>(caminhoServidor.c_str()),
    NULL,
    NULL,
    FALSE,
    0,
    NULL,
    NULL,
    &si,
    &pi)) {
    cerr << "Erro ao iniciar o servidor. Código: " << GetLastError() << endl;
    return false;
}

cout << "Servidor iniciado com sucesso. Aguardando conexão...\n";
```

Imagem 11 - Criação da conexão com o pipe nomeado do servidor.

```
// Tenta se conectar ao pipe
Handle_Pipe = CreateFile(
    TEXT("\\\\.\\pipe\\MeuPipe"),
    GENERIC_READ | GENERIC_WRITE,
    0, NULL, OPEN_EXISTING, 0, NULL);
```

Imagem 12 - Criação do pipe nomeado.

```
HANDLE Handle_Pipe = CreateNamedPipe(
    TEXT("\\\\.\\pipe\\MeuPipe"),
    PIPE_ACCESS_DUPLEX,
    PIPE_TYPE_MESSAGE | PIPE_READMODE_MESSAGE | PIPE_WAIT,
    PIPE_UNLIMITED_INSTANCES,
    512, 512, 0, NULL);
```

A próxima imagem aborda a criação e o uso de Threads atendendo o requisito “Criar um pool de threads para processar cada requisição”.

Imagem 13 - Criação e utilização de Threads no Servidor.

```
thread t(atenderCliente, Handle_Pipe);  
t.detach();  
}
```

A próxima imagem demonstra a utilização de mutex para controle de concorrência, atendendo o requisito “As threads utilizam mutex para garantir acesso seguro a uma estrutura compartilhada”.

Imagem 14 - Controle de concorrência por meio de mutex.

```
lock_guard<mutex> lock(bancoMutex);  
stringstream resposta;
```

As duas (2) próximas imagens abordam a comunicação Cliente-Servidor com IPC (Named Pipes) , atendendo o requisito “A comunicação é feita via pipe ou memória compartilhada”.

Imagem 15 - Comunicação Cliente-Servidor com IPC - Leitura.

```
BOOL sucesso = ReadFile(Handle_Pipe, buffer, sizeof(buffer) - 1, &Bytes_Lidos, NULL);
```

Imagem 16 - Comunicação Cliente-Servidor com IPC - Resposta.

```
WriteFile(Handle_Pipe, respostaComThread.str().c_str(), respostaComThread.str().size(), &Bytes_Escritos, NULL);
```

Análise e discussão

Após toda a implementação do projeto podemos chegar a algumas observações:

- Todos os quatro tipos de instruções, INSERT, SELECT, DELETE e UPDATE, funcionam de maneira perfeita, salvando todas as alterações no arquivo “banco.txt”. Foi utilizado um vetor de registros para representar a tabela de dados.
- Todos os processos estão sendo criados e usados corretamente, sendo eles um processo para o cliente, um processo para o servidor.
- A criação de Threads está sendo realizada e é possível demonstrar que cada thread fica responsável por atender cada conexão estabelecida de forma individual. Foi utilizado a biblioteca std::thread

- O uso de mutex foi necessário para implementar mecanismos de exclusão mútua, protegendo então a região crítica, onde as threads realizam operações de leitura e de escrita. Foi utilizado a biblioteca `std::mutex`
- A comunicação entre cliente e servidor foi implementado com Named Pipes

Link do github: [PedroHss04/Trabalho-SO](https://github.com/PedroHss04/Trabalho-SO)

Referências

TANENBAUM, Andrew Stuart; BOS, Herbert. **Sistemas Operacionais Modernos**. 4. ed. São Paulo: Pearson, 2016. 778 p. Disponível em: <https://archive.org/details/andrew-s.-tanenbaum-herbert-bos-sistemas-operacionais-modernos-pearson/page/n777/mode/2up>. Acesso em: 12 abr. 2025.

LEARN, Microsoft. **Função CreateProcessA (processthreadsapi.h)**. 2024. Disponível em: <https://learn.microsoft.com/pt-br/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa>. Acesso em: 7 abr. 2025.

LEARN, Microsoft. **Função CreateNamedPipeA (winbase.h)**. 2024. Disponível em: <https://learn.microsoft.com/pt-br/windows/win32/api/winbase/nf-winbase-createnamepipea>. Acesso em: 7 abr. 2025.

LEARN, Microsoft. **Criando threads**. 2023. Disponível em: <https://learn.microsoft.com/pt-br/windows/win32/procthread/creating-threads>. Acesso em: 7 abr. 2025.