UNIVERSIDADE DO VALE DO ITAJAÍ
CIÊNCIA DA COMPUTAÇÃO

GABRIEL BELTRÃO LAUS
RAFAEL DA CUNHA

TRABALHO DE SISTEMAS OPERACIONAIS

Simulação do gerenciamento de um banco de dados

ITAJAÍ
(2025)

RESUMO

Esse trabalho possui a finalidade de estudar e entender melhor sobre a utilização e prática de conceitos como processos, threads, IPC's, paralelismo e outros conceitos importantes quando se é falado sobre sistemas operacionais. Para tanto, será apresentado os resultados de uma simulação de um funcionamento interno do gerenciador de um banco de dados, analisando desde a criação dos arquivos até os resultados finais de sua simulação. Tudo sendo utilizado os conceitos de sistemas operacionais para que sua implementação seja clara e entendida.

Palavras-chave: gerenciamento, banco de dados, requisição, processos, threads, servidor.

SUMÁRIO

1. Introdução.

No mundo da computação em que vivemos hoje, é extremamente necessário continuar estudando e descobrindo novas maneiras de aprimorar a tecnologia na qual usamos diariamente, e o gerenciamento de requisições se encaixa perfeitamente nesse aspecto. Já que a cada dia que se passa, produtos e softwares ficam cada vez mais complexos e que, se não forem bem gerenciados, por diversas vezes podem acarretar no atraso ou falha de projetos, códigos incompletos ou com problemas, entre diversas outras causas. E o gerenciamento de requisições é o que garante que o código e sua implementação não saia com problemas, superando complexidades, interdependências e diversos outros problemas.

Para poder demonstrar de uma melhor forma sobre o assunto, será demonstrado a seguir a criação e os resultados de um sistema que simula o funcionamento interno de um gerenciador de requisições em um banco de dados. Esse sistema foi desenvolvido, testado e rodado na linguagem de programação Python e também criado a partir de diversos conceitos muito importantes que serão brevemente explicados para melhor contextualização do programa, como por exemplo: processos, threads, IPC's, pipes, entre outros.

Será mostrado desde a criação de seus componentes, o fluxo de como a simulação do sistema irá ocorrer, os testes e resultados gerados através deles e algumas conclusões geradas após todos os testes de simulação.

Através desse sistema, será possível ver como todos os componentes se conectam e trabalham para poder gerar os resultados da requisição feita pelo usuário. Para que então possa ser entendido como que o uso de todos os componentes conseguem fazer com que o código fique mais organizado e funcional, sem a presença de erros no código ou de falhas na execução.

2. Desenvolvimento.

2.1. O que é um processo?

Um processo se trata de um programa em execução, ou também um conjunto de informações que são necessárias para a concorrência do programa em um sistema operacional.

2.2. O que é uma thread?

Thread se trata de uma linha de execução que compõem a execução de um processo.

2.3. O que é IPC?

Se trata de um mecanismo com dois ou mais processos realizando trocas entre si, se provando útil já que cada processo possui a própria região de memória onde outros processos não tem a permissão de acessar aquele espaço de memória.

2.4. Explicação e fluxo dos arquivos.

O sistema desenvolvido para a simulação possui 3 componentes montados, sendo eles: banco (onde terá os structs do sistema e os defines), cliente (onde será enviado as requisições de consulta ou inserção através de IPC) e o servidor (onde receberá as requisições e usa várias threads para processá-las em paralelo).

Ao iniciar o programa, o cliente irá enviar a requisição (INSERT, DELETE, UPDATE, SELECT), e que, através da comunicação via pipe/memória compartilhada, chegará ao servidor. O servidor lerá as requisições e irá criar um pool de threads para processar as requisições, threads essas que irão acessar o “banco de dados simulado”, que será um arquivo .txt, e irão usar mutex/semáforo para poder garantir a integridade desses dados. Com isso, o arquivo com o resultado será atualizado e um log será criado indicando qual alteração, inserção ou outro procedimento utilizado.

Para isso, deverá ser criado os arquivos para que o fluxo ocorra de maneira correta. O primeiro criado foi o “banco.py”.

```
    from enum import Enum
    from dataclasses import dataclass

    PIPE_NAME = "/tmp/banco_pipe"   # Caminho do pipe nomeado usado
para comunicação entre cliente e servidor
```

```
    MAX_NOME = 50  # Tamanho máximo do nome (não é usado diretamente,
mas pode ser útil para validação)

    # Enum que define os tipos de requisição que o cliente pode fazer
    class TipoRequisicao(Enum):
        INSERT = 0
        DELETE = 1
        SELECT = 2
        UPDATE = 3

    # Classe que representa um registro no banco de dados (id + nome)
    @dataclass
    class Registro:
        id: int
        nome: str

    # Classe que representa uma requisição feita pelo cliente
    @dataclass
    class Requisicao:
        tipo: TipoRequisicao
        reg: Registro = None      # Usado no INSERT
        id_busca: int = None      # Usado no DELETE, SELECT e UPDATE
        novo_nome: str = None     # Usado no UPDATE
```

O segundo criado foi o “cliente.py”

```
    import pickle
    import os
    import time
    from banco import Requisicao, Registro, TipoRequisicao, PIPE_NAME

    # Função que envia uma requisição serializada para o servidor
    def enviar_requisicao(req: Requisicao):
         with open(PIPE_NAME, 'wb') as pipe:  # Abre o pipe em modo
escrita binária
                pickle.dump(req, pipe)  # Serializa a requisição e
escreve no pipe

    # Função principal do cliente
    def main():
         print("Tipo (0: INSERT, 1: DELETE, 2: SELECT, 3: UPDATE): ",
end='')
```

```
        tipo = int(input())  # Lê o tipo de requisição
        tipo = TipoRequisicao(tipo)  # Converte para enum

        req = Requisicao(tipo=tipo)  # Cria a requisição

        # Monta a requisição de acordo com o tipo
        if tipo == TipoRequisicao.INSERT:
                        req.reg  =  Registro(id=int(input("ID:  ")),
nome=input("Nome: "))
        elif tipo in [TipoRequisicao.DELETE, TipoRequisicao.SELECT]:
            req.id_busca = int(input("ID: "))
        elif tipo == TipoRequisicao.UPDATE:
            req.id_busca = int(input("ID: "))
            req.novo_nome = input("Novo nome: ")

        # Mede o tempo de envio da requisição
        inicio = time.time()
        enviar_requisicao(req)
        fim = time.time()
                print(f"Tempo  de  envio  (cliente):  {(fim  -
inicio)*1_000_000:.0f} microssegundos")

    if __name__ == "__main__":
        main()
```

O terceiro criado será o “servidor.py”.

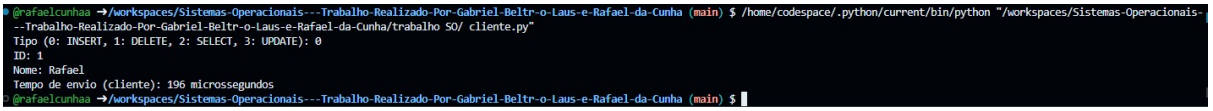
```
    import os
    import threading
    import pickle
    import time
    from banco import Requisicao, Registro, TipoRequisicao, PIPE_NAME
    from threading import Lock

    banco = []  # Lista que simula o banco de dados na memória
    mutex = Lock()  # Mutex para sincronização entre threads

    # Carrega o banco a partir de um arquivo texto
    def carregar_banco():
        if not os.path.exists("banco.txt"):
            return
        with open("banco.txt", "r") as f:
```

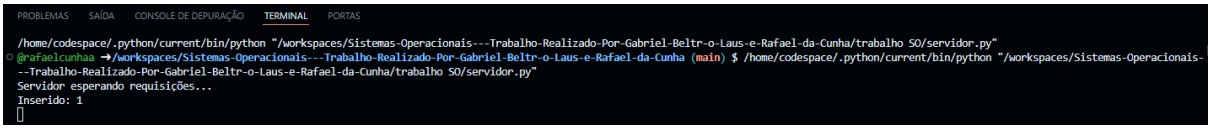
```
            for linha in f:
                id, nome = linha.strip().split()
                banco.append(Registro(int(id), nome))

    # Salva o banco atual no arquivo
    def salvar_banco():
        with open("banco.txt", "w") as f:
            for r in banco:
                f.write(f"{r.id} {r.nome}\n")

    # processa a requisição recebida em uma nova thread
    def processar(req: Requisicao):
        inicio = time.time()
         with mutex:   # garante que só uma thread por vez manipule o
banco
            if req.tipo == TipoRequisicao.INSERT and req.reg:
                banco.append(req.reg)
                print(f"Inserido: {req.reg.id}", flush=True)
            elif req.tipo == TipoRequisicao.DELETE:
                banco[:] = [r for r in banco if r.id != req.id_busca]
                print(f"Deletado ID: {req.id_busca}", flush=True)
            elif req.tipo == TipoRequisicao.SELECT:
                for r in banco:
                    if r.id == req.id_busca:
                        print(f"Encontrado: {r.nome}", flush=True)
            elif req.tipo == TipoRequisicao.UPDATE:
                for r in banco:
                    if r.id == req.id_busca:
                        r.nome = req.novo_nome
                        print(f"Atualizado ID: {r.id}", flush=True)
            salvar_banco()
        fim = time.time()
        tempo = (fim - inicio) * 1_000_000
        with open("desempenho.log", "a") as log:
             log.write(f"Tempo de processamento (thread): {tempo:.0f}
microssegundos\n")

    # loop principal do servidor
    def main():
        if not os.path.exists(PIPE_NAME):
                os.mkfifo(PIPE_NAME)   # cria o pipe nomeado, se não
existir
```

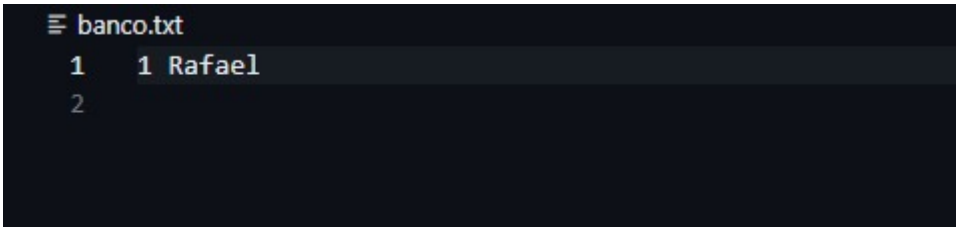
```
          carregar_banco()
          print("Servidor esperando requisições...")

           while True: #cada requisição que chegue seja criado mais uma
thread
                with open(PIPE_NAME, 'rb') as pipe:  # abre o pipe para
leitura
                    req = pickle.load(pipe)  # desserializa a requisição
recebida
                     t = threading.Thread(target=processar, args=(req,))
# cria uma nova thread para processar a requisição
                    t.daemon = True  # define como daemon (encerra junto
com o programa principal)
                  t.start()  # inicia a thread

      if __name__ == "__main__":
          main()
```

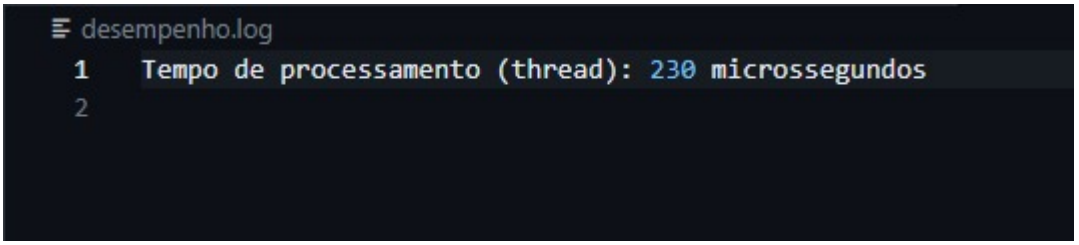
2.4. Testes.

O primeiro teste será o INSERT









O segundo será com DELETE.

```
@rafaelcunhaa ➜/workspaces/Sistemas-Operacionais---Trabalho-Realizado-Por-Gabriel-Beltr-o-Laus-e-Rafael-da-Cunha (main) $ /home/codespace/.python/current/bin/python "/workspaces/Sistemas-Operacionais-
--Trabalho-Realizado-Por-Gabriel-Beltr-o-Laus-e-Rafael-da-Cunha/trabalho SO/ cliente.py"
Tipo (0: INSERT, 1: DELETE, 2: SELECT, 3: UPDATE): 1
ID: 1
Tempo de envio (cliente): 217 microssegundos
@rafaelcunhaa ➜/workspaces/Sistemas-Operacionais---Trabalho-Realizado-Por-Gabriel-Beltr-o-Laus-e-Rafael-da-Cunha (main) $
```

```
/home/codespace/.python/current/bin/python "/workspaces/Sistemas-Operacionais---Trabalho-Realizado-Por-Gabriel-Beltr-o-Laus-e-Rafael-da-Cunha/trabalho SO/servidor.py"
@rafaelcunhaa ➜/workspaces/Sistemas-Operacionais---Trabalho-Realizado-Por-Gabriel-Beltr-o-Laus-e-Rafael-da-Cunha (main) $ /home/codespace/.python/current/bin/python "/workspaces/Sistemas-Operacionais-
--Trabalho-Realizado-Por-Gabriel-Beltr-o-Laus-e-Rafael-da-Cunha/trabalho SO/servidor.py"
Servidor esperando requisições...
Inserido: 1
Deletado ID: 1
```

banco.txt

```
1  Pressione Ctrl+I para pedir para fazer algo. Comece a digitar para ignorar.
```

desempenho.log

```
1  Tempo de processamento (thread): 230 microssegundos
2  Tempo de processamento (thread): 206 microssegundos
3
```

Após ser realizado mais uma vez o INSERT terceiro será com SELECT.

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  PORTAS

Tempo de envio (cliente): 206 microssegundos
@rafaelcunhaa ➜/workspaces/Sistemas-Operacionais---Trabalho-Realizado-Por-Gabriel-Beltr-o-Laus-e-Rafael-da-Cunha (main) $ /home/codespace/.python/current/bin/python "/workspaces/Sistemas-Operacionais-
--Trabalho-Realizado-Por-Gabriel-Beltr-o-Laus-e-Rafael-da-Cunha/trabalho SO/ cliente.py"
Tipo (0: INSERT, 1: DELETE, 2: SELECT, 3: UPDATE): 2
ID: 1
Tempo de envio (cliente): 239 microssegundos
@rafaelcunhaa ➜/workspaces/Sistemas-Operacionais---Trabalho-Realizado-Por-Gabriel-Beltr-o-Laus-e-Rafael-da-Cunha (main) $
```

Code language not supported or defined.

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  PORTAS

--Trabalho-Realizado-Por-Gabriel-Beltr-o-Laus-e-Rafael-da-Cunha/trabalho SO/servidor.py"
Servidor esperando requisições...
Inserido: 1
Deletado ID: 1
Inserido: 1
Encontrado: Rafael
```

Code language not supported or defined.

O quarto será com UPDATE.

```
@rafaelcunhaa ➜/workspaces/Sistemas-Operacionais---Trabalho-Realizado-Por-Gabriel-Beltr-o-Laus-e-Rafael-da-Cunha (main) $ /home/codespace/.python/current/bin/python "/workspaces/Sistemas-Operacionais-
--Trabalho-Realizado-Por-Gabriel-Beltr-o-Laus-e-Rafael-da-Cunha/trabalho SO/ cliente.py"
Tipo (0: INSERT, 1: DELETE, 2: SELECT, 3: UPDATE): 3
ID: 1
Novo nome: Laus
Tempo de envio (cliente): 857 microssegundos
@rafaelcunhaa ➜/workspaces/Sistemas-Operacionais---Trabalho-Realizado-Por-Gabriel-Beltr-o-Laus-e-Rafael-da-Cunha (main) $
```

```
EOFError: Ran out of input
@rafaelcunhaa ➜/workspaces/Sistemas-Operacionais---Trabalho-Realizado-Por-Gabriel-Beltr-o-Laus-e-Rafael-da-Cunha (main) $ /home/codespace/.python/current/bin/python "/workspaces/Sistemas-Operacionais-
--Trabalho-Realizado-Por-Gabriel-Beltr-o-Laus-e-Rafael-da-Cunha/trabalho SO/servidor.py"
Servidor esperando requisições...
Inserido: 1
Atualizado ID: 1
```

3. Conclusão.

Como foi possível ver nos testes feitos acima, a utilização dos componentes acima gerou, em uma rápida execução, os resultados requeridos pelo usuário. Os comandos do INSERT, DELETE, SELECT e UPDATE foram todos testados e, com um tempo de execução muito abaixo, foi capaz de obter todos os requerimentos necessários.

Com essa simulação, foi possível ver que os requerimentos foram atingidos por meio do uso de processos, threads, e técnicas de IPC. Com tudo isso foi possível simular, de forma eficiente, o funcionamento interno de um gerenciador de banco de dados, destacando os resultados dos testes efetuados. E mostrando que o seu uso ajuda e muito na aplicação rápida e eficiente no gerenciamento de requisições.

4. Referências bibliográficas.

1. IBM. *O que é gerenciamento de requisitos?*. Disponível em: https://www.ibm.com/br-pt/topics/what-is-requirements-management. Acesso em: 15 abr. 2025.

2. WIKIPÉDIA. *Thread (computação)*. Disponível em: https://pt.wikipedia.org/wiki/Thread_(computa%C3%A7%C3%A3o). Acesso em: 15 abr. 2025.

3. NFO94. *O que é um processo em um sistema operacional?*. DEV.to, 2022. Disponível em: https://dev.to/nfo94/o-que-e-um-processo-em-um-sistema-operacional-2769. Acesso em: 15 abr. 2025.

4. EMBARCADOS. *Comunicação entre processos*. Disponível em: https://embarcados.com.br/comunicacao-entre-processos/. Acesso em: 15 abr. 2025.