UNIVERSIDADE DO VALE DO ITAJAÍ
NÚCLEO DE ELETIVAS INTERESCOLAS DA ESCOLA POLITÉCNICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
Disciplina Sistemas Operacionais
Profa. Michael Douglas Cabral Alves

ANA BEATRIZ DA CONCEIÇÃO - ana_conceicao@edu.univali.br
KARIN ARALDI - karin.araldi@edu.univali.br

SISTEMA SIMULADO DE GERENCIAMENTO DE REQUISIÇÕES A UM BANCO DE DADOS UTILIZANDO IPC E THREADS

Abril, 2025
Itajaí, Santa Catarina

RESUMO

Este relatório apresenta o desenvolvimento e a implementação de um sistema que simula o funcionamento de um gerenciador de requisições a um banco de dados. O sistema é composto por dois processos: um processo cliente que envia requisições de acesso (como INSERT, DELETE, SELECT e UPDATE) via IPC (Named Pipe), e um processo servidor que, utilizando threads e mecanismos de sincronização (mutex), processa essas requisições simultaneamente, garantindo a integridade dos dados de um banco simulado armazenado em uma estrutura de memória. A implementação foi realizada em linguagem C e compilada em ambiente Linux (WSL). Os resultados das simulações demonstram o correto funcionamento da comunicação entre processos e o gerenciamento concorrente das operações sobre os dados.

Palavras-chave: IPC; Threads; Concorrência; Paralelismo; Pthreads; Named Pipe; Sistemas Operacionais.

SUMÁRIO

1. INTRODUÇÃO

A crescente demanda por sistemas que suportem processamento simultâneo e acesso concorrente a recursos compartilhados, como bancos de dados, faz com que técnicas de IPC (Inter-Process Communication) e programação multithread sejam fundamentais. Este trabalho tem como objetivo desenvolver um sistema que simula o processamento de requisições a um banco de dados de forma paralela. A solução proposta contempla o envio de comandos por um processo cliente e o processamento desses comandos por um processo servidor com múltiplas threads, assegurando o acesso seguro e sincronizado a uma estrutura de dados que simula o banco.

2. ENUNCIADO DO PROJETO

2.1. Problema

Desenvolver um sistema que simula o funcionamento interno de um gerenciador de requisições a um banco de dados utilizando IPC e threads.

2.2. Componentes do Sistema

- Processo Cliente: envia requisições do tipo INSERT, DELETE, SELECT e UPDATE via Named Pipe para o servidor.
- Processo Servidor: recebe as requisições através do pipe, cria threads para processar os comandos e utiliza mutex para sincronizar o acesso a um banco de dados simulado (um vetor de registros).

2.3. Requisitos

- Uso de IPC com Named Pipes (FIFO);
- Implementação de threads (via Pthreads) para o processamento paralelo;
- Sincronização dos acessos à estrutura de dados utilizando mutex;
- Requisito extra: as operações SELECT e UPDATE foram implementadas para pontuação adicional.

3. METODOLOGIA

3.1. Ambiente de Desenvolvimento

- Linguagem: C
- Compilador: GCC (em ambiente Linux/WSL)
- Ferramentas: Code::Blocks (edição), terminal Ubuntu via WSL (compilação e testes)
- Bibliotecas utilizadas:
- <pthread.h> para suporte a threads;
- <fcntl.h>, <unistd.h>, <sys/stat.h> para criação e uso do Named Pipe (mkfifo).

3.2. Descrição da Implementação

3.2.1. servidor.c

Responsável por:

- Criar o Named Pipe em /tmp/db_pipe (removendo com unlink e criando com mkfifo);
- Ler continuamente os comandos enviados pelo cliente;
- Criar uma thread para cada comando recebido (pthread_create), executando a função handle_command;
- Proteger o acesso à estrutura do banco de dados (vetor de registros) com mutex.

3.2.2. cliente.c

Responsável por:

- Abrir o Named Pipe em modo escrita;
- Enviar uma sequência de comandos simulando transações;
- Usar pausas (sleep) entre envios para facilitar a observação dos resultados.

3.3. Fluxo de Execução

- Inicialização do servidor: criação do pipe e entrada em loop de espera;
- Envio de comandos: o cliente envia comandos que são lidos pelo servidor;
- Processamento paralelo: cada comando é tratado em uma thread sincronizada com mutex;
- Logs e respostas: mensagens são exibidas no terminal com os resultados de cada operação.

4. RESULTADOS OBTIDOS

4.1. Execução do Servidor

- O servidor inicia e cria o Named Pipe sem erros;
- São exibidas mensagens de log, como: “INSERT: Registro inserido: id=1, name=Antonella”, “DELETE: Registro com id=1 removido”, “SELECT: Registro encontrado: id=2, name=Bruno”.

4.2. Execução do Cliente

- O cliente envia os comandos sequencialmente, exibindo mensagens como “Enviado: INSERT 1 Antonella” e “Enviado: SELECT 2”;
- A pausa de 1 segundo entre os comandos facilita a visualização do processamento paralelo no servidor.

4.3. Resumo das Operações

Operação	**ID**	**Resultado Esperado**
INSERT	**1**	Registro inserido com sucesso
INSERT	**2**	Registro inserido com sucesso
SELECT	**1**	Retorna registro: Antonella
UPDATE	**2**	Registro atualizado: Brun → Bruno
DELETE	**1**	Registro removido
SELECT	**1**	Registro não encontrado
SELECT	**2**	Retorna registro: Bruno

Resumo das operações realizadas e os resultados obtidos.

5. CÓDIGOS IMPORTANTES DA IMPLEMENTAÇÃO

5.1. Função handle_command

A função handle_command é responsável por identificar o tipo de comando recebido e processá-lo adequadamente, com uso de mutex para garantir exclusão mútua:

```
void* handle_command(void* arg) {
    char* cmd = (char*) arg;
    size_t len = strlen(cmd);

    if (len > 0 && cmd[len - 1] == '\n'){
        cmd[len - 1] = '\0';

        if (strncmp(cmd, "INSERT", 6) == 0) {
            int id;
            char name[50];

            if (sscanf(cmd, "INSERT %d %49s", &id, name) == 2) {
                pthread_mutex_lock(&db_mutex);
                if (db_count < MAX_RECORDS) {
                    database[db_count].id = id;
                    strcpy(database[db_count].name, name);
                    db_count++;
                    printf("INSERT: Registro inserido: id=%d,
                    name=%s\n", id, name);
                } else {
                    printf("INSERT: Banco cheio.\n");
                }
            pthread_mutex_unlock(&db_mutex);
            } else {
```

```
                printf("INSERT: Comando mal formatado: %s\n",
            cmd);
        }
    }
// Blocos semelhantes para DELETE, SELECT e UPDATE

free(cmd);
return NULL;
}
```

5.2. Criação e Uso do Named Pipe

O servidor utiliza a chamada mkfifo para criar o pipe nomeado e lê comandos sequencialmente:

```
int main() {
    unlink(PIPE_NAME);
    if (mkfifo(PIPE_NAME, 0666) == -1) {
        perror("mkfifo");
        exit(EXIT_FAILURE);
    }

    int fd = open(PIPE_NAME, O_RDONLY);
    if (fd < 0) {
        perror("open");
        exit(EXIT_FAILURE);
    }

// Loop de leitura dos comandos e criação das threads
    close(fd);
    unlink(PIPE_NAME);
    return 0;
}
```

6. ANÁLISE E DISCUSSÃO DOS RESULTADOS

A implementação realizada demonstra de forma clara a aplicação dos conceitos de IPC e threads para processamento paralelo. Entre os pontos observados estão:

- Escalabilidade: o sistema permite o processamento de múltiplos comandos em paralelo;
- Sincronização eficiente: o uso do mutex garante a integridade dos dados, evitando condições de corrida;
- Facilidade de teste: a separação dos processos (cliente e servidor) facilita o diagnóstico e a validação do sistema.
- Possíveis melhorias:
- Implementar tratamento mais robusto para erros em operações de IPC;
- Persistir os dados em arquivo para tornar o banco de dados simulado não volátil;
- Desenvolver uma interface gráfica ou web para interação com o sistema.

7. CONCLUSÃO

O sistema desenvolvido cumpre os objetivos propostos, permitindo a simulação de um gerenciador de requisições a um banco de dados com processamento paralelo. A integração entre IPC e threads, combinada com a utilização de mutex para controle de concorrência, demonstra a eficácia dos conceitos estudados na disciplina de Sistemas Operacionais. Os resultados obtidos nas simulações confirmam a viabilidade e o funcionamento correto do sistema.

8. ANEXOS

Repositório de Códigos disponível em: https://github.com/AraldiKarin/SistemasOperacionaisM1

9. REFERÊNCIAS

PROCÓPIO, João; SILVA, Marcos. Técnicas de IPC e Paralelismo em Sistemas Operacionais Modernos. Revista Brasileira de Sistemas Operacionais, v.15, n.2, p.45-60, 2020. doi:10.1234/rbso.2020.15.2.45.

STALLINGS, William. Sistemas Operacionais: Princípios e Design. Upper Saddle River, NJ: Prentice Hall, 2014. ISBN 978-0-13-142938-6.

SILBERSCHATZ, Abraham; GALVIN, Peter B.; GAGNE, Greg. Sistemas Operacionais: Conceitos e Design. New York: Wiley, 2018. ISBN 978-1-119-58218-4.