Universidade do Vale do Itajaí
Disciplina: Sistemas Operacionais
Avaliação M1 – IPC, Threads e Paralelismo

Instruções

1. Esta avaliação deve ser feita em dupla ou em trio.

2. Data de entrega: 15/04/2025 até as 19:00. Não pode entregar em atraso.

3. Esta avaliação tem por objetivo consolidar o aprendizado sobre conceitos de IPC, threads, concorrência e paralelismo.

4. A implementação deverá ser desenvolvida utilizando a linguagem de programação de sua preferência (C, C++, Python, Java, C#, Javascript/Node.js, Rust, etc). Porém, a utilização e suporte a IPC e threads pela linguagem escolhida é de responsabilidade do(s) aluno(s), seja usado corretamente o conceito de threads e IPC. As bibliotecas usadas devem ser equivalentes a Pthreads. Bibliotecas que também implementem e que permitam usar conceitos de paralelismo também podem ser usadas, mas o aluno também é responsável pelo seu uso e apresentação

5. O sistema deve ser entregue funcionando corretamente. Sistemas não compilando e executando não serão aceitos. É de responsabilidade do(s) aluno(s) apresentar a execução funcionando corretamente.

6. Deve ser disponibilizado os códigos da implementação em repositório do(s) aluno(s) (github, bitbucket, etc...), deve ser fornecido o link e o repositório dever ser público.

7. O relatório deve seguir o formato de artigo científico ou seguindo as normas da ABNT e as orientações para produção de trabalhos acadêmicos da Univali contendo:

    - Identificação do autor e do trabalho.
    - Enunciado dos projetos.
    - Explicação e contexto da aplicação para compreensão do problema tratado pela solução.
    - Resultados obtidos com as simulações.
    - Códigos importantes da implementação.
    - Resultados obtidos com a implementação (tabelas, gráficos e etc).
    - Análise e discussão sobre os resultados finais.

8. Deve ser disponibilizado os códigos da implementação juntamente com o relatório (salvo o caso da disponibilidade em repositório aberto do aluno, que deve ser fornecido o link). O repositório deve estar aberto do momento da entrega em diante, sendo que o professor não responsabiliza caso o projeto não esteja disponível para consulta no momento da correção, sendo do(s) aluno(s) essa responsabilidade de manter disponível. Cópias entre alunos implicará em nota zero para todos os envolvidos.

9. O trabalho deverá ser apresentado em data definida pelo professor. É de responsabilidade do(s) aluno(s) explicar os conceitos, comandos, bibliotecas usadas. É de responsabilidade do(s) aluno(s) fazer a solução funcionar e ela deverá ser baixada do local de entrega no momento da apresentação. Trabalhos não apresentados terão como nota máxima 5,0, além dos descontos aplicados no restante da avaliação da implementação. Todos os alunos poderão apresentar ou o professor poderá escolher um representante para apresentar o trabalho em nome do grupo.

Descrição do projeto a ser desenvolvido

Problemática: Sistema de Processamento Paralelo de Requisições a um Banco de Dados

Descrição Geral

Você irá desenvolver um sistema que simula o funcionamento interno de um **gerenciador de requisições a um banco de dados**, com múltiplos processos e threads:

- Um **processo cliente** envia requisições de consulta ou inserção via IPC (pipe ou memória compartilhada);
- Um **processo servidor** recebe as requisições e usa várias **threads** para processá-las em paralelo;
- As threads utilizam **mutex ou semáforo** para garantir acesso seguro a uma estrutura compartilhada que simula um banco de dados (como um vetor ou um arquivo).

Componentes do Sistema

1. Processo Cliente

- Envia "requisições de banco" para o servidor:
  - Exemplo de requisição: SELECT nome WHERE id=5 ou INSERT id=7 nome='João'
- A comunicação é feita via **pipe** ou **memória compartilhada**.

2. Processo Servidor (Gerenciador de Banco)

- Lê as requisições do pipe ou memória compartilhada.
- Cria um pool de **threads** para processar cada requisição.
- As threads acessam um "banco de dados simulado" (pode ser um vetor ou arquivo .txt/.json/.sqlite) e usam **mutex/semáforo** para garantir a integridade dos dados.
- As respostas podem ser salvas em um segundo canal IPC, ou em um arquivo de log.

Requisições Suportadas

- INSERT: Adiciona um novo registro (linha) no "banco".
- DELETE: Remove um registro.

Estrutura interna do "banco":

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

Uso de Threads e Mutex/Semáforo

- Threads compartilham a mesma "tabela de dados".
- Acesso à tabela (inserção, delete) é protegido por:
  - pthread_mutex_t (para exclusão mútua), ou
  - sem_t (para controle mais geral, como múltiplas leituras simultâneas).
- Isso evita condições de corrida e garante a consistência dos dados.
- Adapte para a linguagem que você escolheu.

Fluxo do Sistema

[Cliente envia requisição SQL]
↓
[Pipe ou Memória Compartilhada]
↓
[Servidor (com threads)]
↓
[Tabela de Dados com Mutex/Semáforo]
↓
[Respostas/Logs ou Arquivo com Resultados] (Opcional)

Estrutura de Arquivos Sugerida (pode ser adaptado para a linguagem que vocês escolherem)

```
bd_simulado/
├── cliente.c          # Envia requisições para o servidor
├── servidor.c         # Recebe requisições e usa threads
├── banco.h            # Structs e defines
└── banco.txt          # Simulação da base de dados
```

Conceitos Praticados

- IPC com pipe ou memória compartilhada.
- Criação e sincronização de processos.
- Threads com controle de concorrência (mutex/semaphore).
- Simulação de SGBD leve (com INSERT/DELETE).
- Leitura e escrita concorrente em arquivo ou estrutura em memória.

Pontuação Extra na M1

Vocês irão receber de 0,5 à 1,5 pontos na prova da M1 caso implemente também as operações:

- SELECT: Busca e retorna dados com base em critérios simples (por exemplo, por ID).
- UPDATE: Modifica um campo de um registro.

A nota extra fica a critério do professor em avaliar o contexto de aplicação, bem como a qualidade do código e aplicabilidade.