Lab6 - Programação Concorrente

Aluno: Tales Moreira
DRE: 119047549

Atividade 1: Exclusão Mútua com `main_lock.c`

1. **O TAD lista implementado poderia ser compartilhado por threads? Com qual finalidade e de que forma?**
    - Sim, o TAD lista pode ser compartilhado por várias threads em um programa concorrente. Cada thread pode realizar operações de inserção, remoção e consulta. Contudo, como as operações de escrita (inserção e remoção) modificam a lista, é necessário usar exclusão mútua (com mutex) para garantir a integridade dos dados.
2. **O que poderia acontecer se o programa não implementasse exclusão mútua no acesso às operações da lista encadeada?**
    - Sem exclusão mútua, threads podem tentar acessar ou modificar a lista ao mesmo tempo, o que causaria corrupção de dados. Por exemplo, uma thread pode ler um valor enquanto outra o exclui, levando a acessos inválidos de memória ou resultados inconsistentes.
3. **O que acontece com o tempo de execução quando aumentamos o número de threads? Por que isso ocorre?**
    - O tempo de execução tende a diminuir com mais threads, mas em certo ponto, o benefício diminui devido ao aumento do overhead de sincronização (locks). Muitas threads competindo por um único recurso (a lista) podem causar contenção, o que limita o ganho de desempenho.

Atividade 2: Uso de rwlock com `main_rwlock.c`

1. **Em quais cenários o uso do rwlock pode ser mais vantajoso do que o uso do lock de exclusão mútua?**
    - O uso de `rwlock` é vantajoso quando há muitas operações de leitura, pois várias threads podem ler simultaneamente sem bloquear umas às outras. Se as leituras forem muito mais frequentes que as escritas, isso melhora significativamente o desempenho, pois as threads não precisam esperar umas pelas outras.

Atividade 3: Implementação de rwlock com prioridade para escrita com `main_rwlock_priority_write_no_delete.c`

Esta atividade sofreu várias mudanças para implementar corretamente a prioridade de escrita. Aqui está o que foi ajustado:

1. **Implementação da Função de Remoção (`Delete`):**
    - A função de remoção foi adicionada ao arquivo `list_int.c` para permitir a remoção de itens da lista encadeada. Esta função percorre a lista para

encontrar o valor e o remove, garantindo que a estrutura de dados permaneça consistente.

2. **Modificações no Código:**
    - O código foi modificado para garantir que a remoção de elementos seja feita com exclusão mútua (usando `rwlock`). As operações de leitura, inserção e remoção agora ocorrem com uma distribuição configurada: 98% leituras, 1% remoções, 1% inserções.
    - A variável `write_requests` foi mantida para garantir que, quando uma operação de escrita for solicitada (inserção ou remoção), novas operações de leitura sejam bloqueadas até a conclusão da escrita.
    - A variável `out` foi corrigida para rastrear corretamente o número de remoções realizadas.
3. **Demonstração da Prioridade de Escrita:**
    - O código foi testado para garantir que, quando uma escrita (inserção ou remoção) é solicitada, as operações de leitura são bloqueadas até a conclusão da escrita. As mensagens de log indicam o comportamento esperado, onde as operações de escrita têm prioridade.