Relatório Final: Exemplo 4 - Deadlock

De: Gabriel Henrique da Silva

RA:22020864


```
Essa parte do projeto foi feita em markdown
```

1. Descrição do Problema:

O objetivo dessa implementação foi simular um **deadlock** entre duas threads utilizando **mutexes**. Deadlock ocorre quando duas ou mais threads ficam esperando indefinidamente por recursos que estão sendo bloqueados pelas próprias threads, criando um ciclo de espera.

2. Explicação do Código:

O código apresentado é composto por duas threads que tentam adquirir dois **mutexes** (`lock1` e `lock2`) de maneira que, se adquirirem os locks em uma ordem diferente, um **deadlock** pode ocorrer.

Estrutura do código:

1. **Cabeçalhos Importados**:
    - `pthread.h` : Usado para criar e gerenciar threads.
    - `stdio.h` : Para operações de entrada e saída (impressão de mensagens no terminal).
    - `stdlib.h` : Para alocação de memória e manipulação de erros.
    - `unistd.h` : Para usar a função `sleep` (que simula o tempo de espera entre as tentativas de adquirir os bloqueios).
2. **Definição de Mutexes**:
    - `pthread_mutex_t lock1, lock2;` são as variáveis que representam os

mutexes que as threads irão tentar bloquear.

3. **Função `thread1_func`**:
    - A primeira thread tenta bloquear `lock1` e, após um atraso de 1 segundo (`sleep(1)`), tenta bloquear `lock2`. Após conseguir bloquear ambos, ela imprime uma mensagem e desbloqueia ambos os mutexes.
4. **Função `thread2_func`**:
    - A segunda thread tenta bloquear `lock2` primeiro, e depois, após um atraso de 1 segundo, tenta bloquear `lock1`. Se ela conseguir bloquear ambos, imprime uma mensagem e desbloqueia os mutexes.
5. **Função Principal (`main`)**:
    - Inicializa os mutexes `lock1` e `lock2`.
    - Cria as duas threads (`pthread_create`).
    - Espera as threads terminarem (`pthread_join`).
    - Destrói os mutexes após a execução.

3. Explicação do Deadlock:

Neste código, ocorre um **deadlock** porque:

- **Thread 1** adquire `lock1` e, em seguida, tenta adquirir `lock2`.
- **Thread 2** adquire `lock2` e tenta adquirir `lock1`.
- Ambas as threads estão bloqueadas uma pela outra, esperando que o mutex que a outra detém seja liberado, mas isso nunca ocorre, pois nenhuma delas pode prosseguir.

Esse bloqueio mútuo cria um ciclo de dependência entre as threads, causando o **deadlock**.

4. Output Esperado:

O comportamento esperado é que o programa imprima mensagens indicando que as threads adquiriram os locks. No entanto, como ocorre um deadlock, o programa nunca chega a liberar os mutexes e não termina corretamente.