# O PROBLEMA DOS LEITORES E ESCRITORES

ALUNO:MARCELO FERREIRA DE ALMEIDA

O problema dos Leitores e Escritores se refere ao acesso simultâneo de várias threads a um recurso compartilhado, como um banco de dados. Podemos ter dois tipos de processos/threads: **leitores**, que só consultam os dados, e **escritores**, que podem modificar os dados.

**Leitores**: Processos, os quais não são requeridos excluir uns aos outros (entre eles).

**Escritores**: Processos, os quais são requeridos excluir todos os outros processos excluir todos os outros processos, leitores leitores e escritores.

1. Vários leitores podem acessar o recurso ao mesmo tempo, desde que nenhum escritor esteja acessando este recurso.
2. Um escritor deve ter acesso exclusivo ao recurso, ou seja, nenhum outro escritor ou leitor pode acessar o recurso ao mesmo tempo.

**Componentes do Código**

**Variáveis Globais e Semáforos**

1. **sem_t recurso**:
    - Semáforo usado para controlar o acesso ao recurso compartilhado.
    - Quando **escritores** precisam escrever, eles bloqueiam o recurso para que outros leitores e escritores não possam acessá-lo.
2. **sem_t mutexCount**:
    - Semáforo usado para proteger a variável **contador_leitores**, garantindo que ela seja atualizada de maneira segura.
    - Evita **condições de corrida** ao incrementar ou decrementar o contador de leitores.
3. **int contador_leitores = 0**:
    - Mantém o número de leitores que estão atualmente lendo o recurso.
    - É utilizado para saber quando o primeiro leitor entra (bloqueando o recurso para escritores) e quando o último leitor sai (liberando o recurso).

## Função leitor:

```
void* leitor(void* arg) {
    int id = *(int*)arg;
    // Bloqueia para incrementar a contagem de leitores
    sem_wait(&mutexCount);
    contador_leitores++;
    if (contador_leitores == 1) {  // Primeiro leitor bloqueia o recurso para escritores
        sem_wait(&recurso);
    }
    sem_post(&mutexCount);              // Libera o controle da contagem de leitores
    // Leitura do recurso
    printf("Leitor %d está lendo.\n", id);
    // Bloqueia para decrementar a contagem de leitores
    sem_wait(&mutexCount);
    contador_leitores--;
    if (contador_leitores == 0) {  // Último leitor libera o recurso
        sem_post(&recurso);
    }
    sem_post(&mutexCount);              // Libera o controle da contagem de leitores
    return NULL;
}
```

Na função leitor vemos que ao ser adicionado o primeiro leitor, o acesso ao recurso será bloqueado, impedindo o acesso de outros escritores, e quando o último leitor finalizar a tarefa (contador_leitores == 0) o recurso será liberado para o acesso.

## Função escritor:

```
void* escritor(void* arg) {
    int id = *(int*)arg;
    // Escritor bloqueia o recurso
    sem_wait(&recurso);
    // Escrita no recurso
    printf("Escritor %d está escrevendo.\n", id);
    sem_post(&recurso);            // Escritor libera o recurso
    return NULL;
}
```

Ao iniciar um processo de escrita, o escritor em questão bloqueia o acesso ao recurso e quando finaliza o processo, o recurso é liberado.