Seção críticas e Técnica de Exclusão mútua:

Seção Crítica (ou Critical Section), se refere a um trecho de código em um programa, que por necessidade deste, acessa recursos compartilhados, como variáveis globais, arquivos ou dispositivos I/O. Esses recursos compartilhados, por sua vez, precisam ser protegidos contra o acesso simultâneo por mais de uma thread ou processo, a fim de impedir que ocorram comportamentos anômalos ou resultados ambíguos e inconsistentes, para isso é feito uso da Técnica de Exclusão Mútua.

A **Técnica de Exclusão Mútua** (ou Mutual Exclusion Techniques) é um conjunto de técnicas usadas para garantir que apenas uma thread ou processo acesse a Seção Crítica de cada vez, evitando assim condições de corrida/interferência de concorrência. Essas condições ocorrem quando duas ou mais threads ou processos tentam acessar ou modificar o mesmo recurso simultaneamente, resultando em comportamentos imprevisíveis ou resultados inconsistentes.

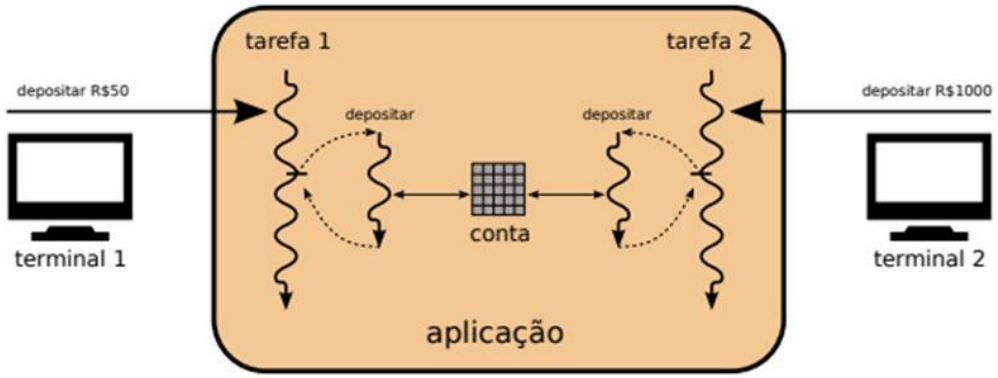

Figura 10.1: Acessos concorrentes a variáveis compartilhadas.

Duas da soluções mais utilizadas em códigos para contornar os problemas e comportamentos anômalos dos programas consistem em:

Semáforos, são uma técnica popular para garantir exclusão mútua em Seções Críticas. Eles usam um contador de semáforo que é inicializado em 1. Quando um processo ou thread deseja entrar na Seção Crítica, ele tenta decrementar o contador de semáforo. Se o contador for 0, isso significa que outra thread já está na Seção Crítica, então a thread atual é bloqueada até que o semáforo seja incrementado novamente.

Quando a thread atual sai da Seção Crítica, ela incrementa o contador de semáforo, permitindo que outra thread entre. O uso de semáforos permite que várias threads acessem a Seção Crítica, mas apenas uma por vez, evitando condições de corrida e inconsistências de dados.

Locks, Locks, ou bloqueios, são outra técnica comum para garantir exclusão mútua em Seções Críticas. Eles são semelhantes aos semáforos, mas em vez de um contador, um objeto Lock é usado. Quando um processo ou thread deseja entrar na Seção Crítica, ele tenta obter o Lock. Se o Lock já estiver sendo usado por outra thread, a thread atual é bloqueada até que o Lock seja liberado.

Assim como os semáforos, o uso de Locks garante que apenas uma thread possa acessar a Seção Crítica por vez, evitando condições de corrida e inconsistências de dados.

Exemplo de Código em C usando **Semáforos:**

```c
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

#define NUM_THREADS 5

int contador = 0; // recurso compartilhado
sem_t semaforo; // semáforo binário

void *thread_func(void *arg) {
  int id = *(int *) arg;

  // acessa a Seção Crítica
  sem_wait(&semaforo);
  contador++;
  printf("Thread %d incrementou o contador para %d\n", id, contador);
  sem_post(&semaforo);

  pthread_exit(NULL);
}

int main() {
  pthread_t threads[NUM_THREADS];
  int ids[NUM_THREADS];
  int i;

  // inicial semáforo com valor 1 (seção livre)

sem_init(&semaforo, 0, 1);

// cria as threads
for (i = 0; i < NUM_THREADS; i++) {
ids[i] = i;
pthread_create(&threads[i], NULL, thread_func, &ids[i]);
}

// aguarda as threads terminarem
for (i = 0; i < NUM_THREADS; i++) {
pthread_join(threads[i], NULL);
}

// destroi o semáforo
sem_destroy(&semaforo);

return 0;
}
```

Exemplo de Código, também em C, usando **Locks:**

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int valor = 0;
pthread_mutex_t lock; // Declaração do Lock

void* incrementar(void* arg) {
    pthread_mutex_lock(&lock); // Adquire o Lock
    valor++;
    printf("Valor incrementado para %d\n", valor);
    pthread_mutex_unlock(&lock); // Libera o Lock
    pthread_exit(NULL);
}

int main() {
    pthread_t threads[10];
    pthread_mutex_init(&lock, NULL); // Inicializa o Lock
    int i;

    for (i = 0; i < 10; i++) {
        pthread_create(&threads[i], NULL, incrementar, NULL);
    }

    for (i = 0; i < 10; i++) {
        pthread_join(threads[i], NULL);
    }

    pthread_mutex_destroy(&lock); // Libera o Lock
    return 0;
}
```