Universidade Federal de Santa Catarina

Centro de Engenharias da Mobilidade
Curso de Engenharia Mecatrônica

Exercício IPC

Disciplina: Sistemas Operacionais

Aluno: Maykon Hopka

Joinville – 2025

1 Exercícios

Exercício 1

Analisar e apontar quais são as regiões críticas do código sujeitas a condição de corrida.

Solução 1.1: Exercício 1

A região crítica ocorre dentro da função `ThreadAdd`, especificamente nas instruções que leem, incrementam e escrevem o valor da variável global `count`:

- `tmp = count;`
- `tmp = tmp + 1;`
- `count = tmp;`

Exercício 2

Mostrar que o resultado da execução é sujeito a condição de corrida.

Solução 1.2: Exercício 2

Executando o programa original (sem uso de mutex), é possível observar que o valor final de `count` nem sempre atinge o esperado `2 * NITER`. Um exemplo de saída:

```
$ ./sem_mutex
BOOM! count is [1724351217], should be 2000000000
```

Isso demonstra que houve **perda de incremento** devido à interferência entre as duas threads que acessam `count` simultaneamente.

Exercício 3

Implementar a técnica de sincronização apropriada para resolver a condição de corrida.

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

#define NITER 1000000000

long count = 0;
pthread_mutex_t mutex;  // Declaração do mutex

void * ThreadAdd(void * a)
{
    int i;
    long tmp;
    for(i = 0; i < NITER; i++)
    {
        pthread_mutex_lock(&mutex);  // Bloqueia o mutex

        tmp = count;      /* copy the global count locally */
        tmp = tmp+1;      /* increment the local copy */
        count = tmp;      /* store the local value into the global count */

        pthread_mutex_unlock(&mutex);  // Libera o mutex
    }
}

int main(int argc, char * argv[])
{
    pthread_t tid1, tid2;

    // Inicializa o mutex
    if (pthread_mutex_init(&mutex, NULL)) {
        printf("\n ERROR initializing mutex");
        exit(1);
    }

    if(pthread_create(&tid1, NULL, ThreadAdd, NULL))
    {
      printf("\n ERROR creating thread 1");
      exit(1);
    }

    if(pthread_create(&tid2, NULL, ThreadAdd, NULL))
    {
      printf("\n ERROR creating thread 2");
      exit(1);
    }

    if(pthread_join(tid1, NULL))  /* wait for the thread 1 to finish */
    {
      printf("\n ERROR joining thread");
      exit(1);
    }

    if(pthread_join(tid2, NULL))        /* wait for the thread 2 to finish */
    {
      printf("\n ERROR joining thread");
      exit(1);
    }

    // Destroi o mutex
    pthread_mutex_destroy(&mutex);

    if (count < 2 * NITER)
        printf("\n BOOM! count is [%ld], should be %ld\n", count, (long)2*NITER);
    else
        printf("\n OK! count is [%ld]\n", count);

    pthread_exit(NULL);
}
```

Figura 1: código com mutex para evitar condição de corrida