



Universidade Federal de Santa Catarina

Centro de Engenharias da Mobilidade
Curso de Engenharia Mecatrônica

Exercício de Threads

Disciplina: Sistemas Operacionais

Aluno: Maykon Hopka

Joinville – 2025

1 Exercícios

Exercício 1

Pesquise sobre a função `pthread_join` e explique o que ela faz.

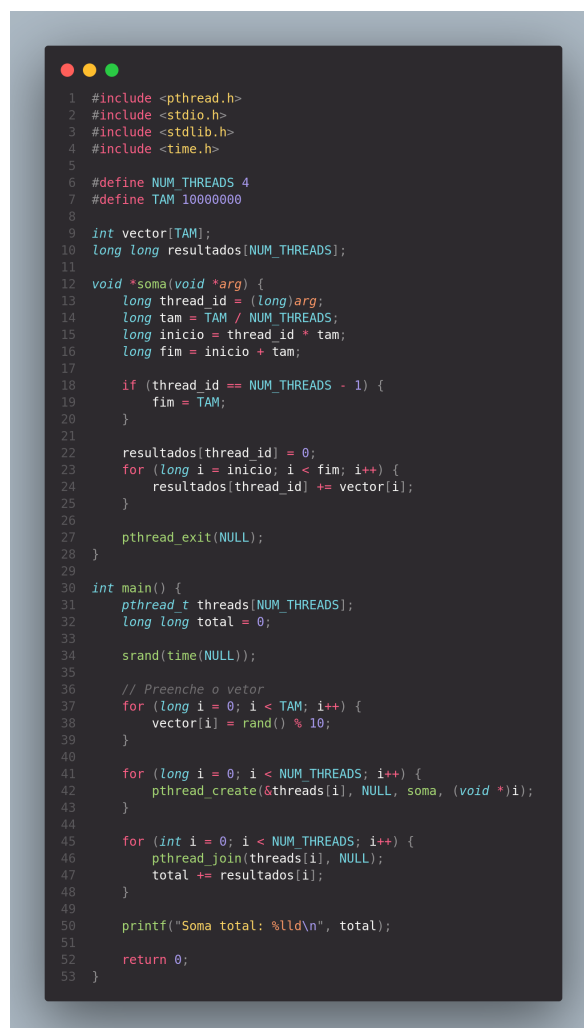
Solução 1.1: Exercício 1

A função `pthread_join` serve para ****sincronizar threads****, ou seja, faz a thread chamadora esperar até que a thread especificada termine.

Exercício 2

Implemente um programa que calcule a soma dos valores contidos em um vetor de tamanho N utilizando *threads*.

1. Divida a tarefa de cálculo proporcionalmente entre o número de *threads* criadas, garantindo que cada *thread* acesse apenas a sua parte do vetor, sem sobreposição.
2. Meça o tempo de execução do programa utilizando o comando `time` no terminal.
3. Analise os resultados obtidos e discuta se houve ganho de desempenho ao utilizar múltiplas *threads* em comparação à execução com uma única *thread*.



```
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <time.h>
5
6 #define NUM_THREADS 4
7 #define TAM 10000000
8
9 int vector[TAM];
10 long long resultados[NUM_THREADS];
11
12 void *soma(void *arg) {
13     long thread_id = (long)arg;
14     long tam = TAM / NUM_THREADS;
15     long inicio = thread_id * tam;
16     long fim = inicio + tam;
17
18     if (thread_id == NUM_THREADS - 1) {
19         fim = TAM;
20     }
21
22     resultados[thread_id] = 0;
23     for (long i = inicio; i < fim; i++) {
24         resultados[thread_id] += vector[i];
25     }
26
27     pthread_exit(NULL);
28 }
29
30 int main() {
31     pthread_t threads[NUM_THREADS];
32     long long total = 0;
33
34     srand(time(NULL));
35
36     // Preenche o vetor
37     for (long i = 0; i < TAM; i++) {
38         vector[i] = rand() % 10;
39     }
40
41     for (long i = 0; i < NUM_THREADS; i++) {
42         pthread_create(&threads[i], NULL, soma, (void *)i);
43     }
44
45     for (int i = 0; i < NUM_THREADS; i++) {
46         pthread_join(threads[i], NULL);
47         total += resultados[i];
48     }
49
50     printf("Soma total: %lld\n", total);
51
52     return 0;
53 }
```

Figura 1

1. Medição do Tempo de Execução: O comando `time` foi utilizado para medir a execução da versão sequencial (`./soma`) e da versão paralela (`./thread`).

Execução Sequencial (1 thread):

```
$ time ./soma
Soma total: 2250037019
./soma 6,84s user 0,43s system 99% cpu 7,270 total
```

Execução Paralela (4 threads):

```
$ time ./thread
Soma total: 2250000642
./thread 9,20s user 0,42s system 133% cpu 7,213 total
```

2. Análise:

- Tempo Sequencial (T_s): 7,270 segundos.
- Tempo Paralelo (T_p): 7,213 segundos.

$$\text{Ganho \%} = \left(\frac{T_s - T_p}{T_s} \right) \times 100 = \left(\frac{7,270 - 7,213}{7,270} \right) \times 100 \approx 0,78\%$$

A implementação paralela foi aproximadamente **0,78% mais rápida**.

Análise do Uso de CPU: Apesar de não se mostrar muito mais rápido temos a certeza que está utilizando mais de um núcleo pois:

- **99% cpu (Sequencial):** Um processo de thread única está limitado a um único núcleo. O valor de 99% confirma que o programa utilizou a capacidade máxima de **um núcleo** de processador.
- **133% cpu (Paralelo):** Um valor superior a 100% é a prova de que o sistema operacional alocou as threads do processo para execução em **mais de um núcleo simultaneamente**.