Laboratório 5

Hanna Epelboim Assunção 2310289
Erica Oliveira Regnier 2211893

Ex1: Paralelismo Criar um vetor de 10.000 posições inicializado com valor 5. Criar 10 trabalhadores (tarefas) que utilizam áreas diferentes do vetor para multiplicar a sua parcela do vetor por 2 e somar as posições do vetor retornando o resultado para um processo coordenador que irá apresentar a soma de todas as parcelas recebidas dos trabalhadores.
Obs: O 1º trabalhador irá atuar nas primeiras 1.000 posições, o 2º trabalhador nas 1.000 posições seguintes e assim sucessivamente.
Repita o código do Ex1 usando agora com 100 trabalhadores e com um vetor de 100.000 posições. Indique o que ocorreu

Codigo fonte:

```c
//Érica Oliveira Regnier
//Hanna Epelboim Assunção

#include <stdio.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h>

#define TAM_VETOR 10000 // Tamanho total do vetor
#define N_THREADS 10 // Número
#define SEGMENT_SIZE (TAM_VETOR / N_THREADS) // Tamanho do segmento
para cada tarefa

int p[TAM_VETOR];
int soma[N_THREADS] = {0};
pthread_t tid_filho[N_THREADS];

void* trabalhador(void* seg){
   int*ptemp = (int*) seg;
   int segmento = *ptemp;
   for (int j = 0; j < SEGMENT_SIZE; j++) {
```

```
            int index = segmento * SEGMENT_SIZE + j;
            p[index] *= 2;
            soma[segmento] += p[index];
    }

    pthread_exit(NULL);
}

int main(){
    pthread_t tid_filho[N_THREADS];

    for (int i = 0; i < TAM_VETOR; i++) {
        p[i] = 5; // Definindo todos os elementos como 5
    }

    int indices[N_THREADS];
    for (int i = 0; i < N_THREADS; i++) {
        indices[i] = i;
        if (pthread_create(&tid_filho[i], NULL, trabalhador,
&indices[i]) != 0) {
            perror("Erro na criação do novo thread\n");
            exit(2);
        } else {
            printf("Criou o trabalhador %d\n", i);
        }
    }


    int somaTotal = 0;
    for (int i = 0; i < N_THREADS; i++) {
        pthread_join(tid_filho[i], NULL);
        printf("Thread: %d Soma: %d\n", i, soma[i]);
        somaTotal += soma[i];
    }
        printf("Soma de todas as threads: %d\n",somaTotal);

    return 0;
}
```

Comando:

```
[c2310289@rainha l5]$ gcc -Wall -o ex1 ex1.c
[c2310289@rainha l5]$ ./ex1
```

Saída com 10 trabalhadores:

```
[c2310289@rainha l5]$ ./ex1
Criou o trabalhador 0
Criou o trabalhador 1
Criou o trabalhador 2
Criou o trabalhador 3
Criou o trabalhador 4
Criou o trabalhador 5
Criou o trabalhador 6
Criou o trabalhador 7
Criou o trabalhador 8
Criou o trabalhador 9
Thread: 0 Soma: 10000
Thread: 1 Soma: 10000
Thread: 2 Soma: 10000
Thread: 3 Soma: 10000
Thread: 4 Soma: 10000
Thread: 5 Soma: 10000
Thread: 6 Soma: 10000
Thread: 7 Soma: 10000
Thread: 8 Soma: 10000
Thread: 9 Soma: 10000
Soma de todas as threads: 100000
```

Saída com 100 trabalhadores e 100.000 posições:

```
Soma de todas as threads: 1000000
```

Explicação:

Esse código implementa uma operação paralela usando threads em C, onde um vetor de tamanho 10.000 é modificado e sua soma é calculada em segmentos. Cada trabalhador é responsável por processar uma parte do vetor. A função trabalhador é executada por cada thread. Ela realiza as seguintes operações:

1. segmento: Recebe o índice do segmento (parte do vetor) que essa thread vai processar.
2. for loop: A thread percorre o segmento designado do vetor p, multiplica cada elemento por 2 e soma esse valor na variável soma[segmento].

3. pthread_exit(NULL): Após processar o segmento, a thread termina sua execução.

Na função main ocorrem os seguintes eventos:

1. Inicialização do vetor p: O vetor de 10.000 posições é inicializado com o valor 5 em todas as posições.
2. Criação das threads:
    - Um array índices é criado para armazenar o índice do segmento para cada thread.
    - A função pthread_create é chamada para criar cada thread, passando o índice do segmento que ela deve processar. Se a criação da thread for bem-sucedida, uma mensagem de confirmação é exibida.
3. Esperar pelas threads:
    - O programa aguarda que todas as threads terminem sua execução, utilizando pthread_join.
    - A soma parcial de cada thread é exibida, e ao final, a soma total de todas as threads é calculada e exibida.

Resumindo, cada thread processa um segmento diferente do vetor. Para cada posição do vetor, o valor é multiplicado por 2, e o resultado é somado na variável correspondente àquela thread. O vetor é dividido em segmentos, cada thread trabalha em um segmento específico. Dessa forma, o código usa threads para paralelizar o trabalho, aumentando a eficiência quando executado em sistemas com múltiplos núcleos.

Ex2: concorrência Considere o vetor de 10.000 posições inicializado com o valor 5. Crie 2 trabalhadores, ambos multiplicam por 2 e somam 2 em todas as posições do vetor. Verifique se todas as somas têm valores iguais e explique o que ocorreu.

Repita o código do Ex2 usando agora 10 ou 100 trabalhadores e um vetor de 100.000 posições. Indique o que ocorreu.

Código fonte:

```c
//Érica Oliveira Regnier
//Hanna Epelboim Assunção

#include <stdio.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>
#include <pthread.h>

#define TAM_VETOR 10000 // Tamanho total do vetor
#define N_THREADS 2 // Número de tarefa

int p[TAM_VETOR];
int soma[N_THREADS] = {0};
pthread_t tid_filho[N_THREADS];

void* trabalhador(void* seg){
   // int*ptemp = (int*) seg;
   // int segmento = *ptemp;
   for (int j = 0; j < TAM_VETOR; j++) {
       p[j] *= 2;
       p[j] += 2;
   }
   pthread_exit(NULL);
}

int main(){
   pthread_t tid_filho[N_THREADS];

   for (int i = 0; i < TAM_VETOR; i++) {
       p[i] = 5; // Definindo todos os elementos como 5
   }
```

```
   int indices[N_THREADS];
   for (int i = 0; i < N_THREADS; i++) {
       indices[i] = i;
       if (pthread_create(&tid_filho[i], NULL, trabalhador,
&indices[i]) != 0) {
           perror("Erro na criação do novo thread\n");
           exit(2);
       } else {
           printf("Criou o trabalhador %d\n", i);
       }
   }

   for (int i = 0; i < N_THREADS; i++) {
       pthread_join(tid_filho[i], NULL);
   }
   int valor = p[0];
   int somaTotal = 1;
   // printf("Vetor: ");
   // for (int i = 0; i < TAM_VETOR; i++) {
   //     printf("%d\n", p[i]);
   // }

   for (int j = 0; j < TAM_VETOR; j++) {
       // printf("%d\n",p[j]);
       if(p[j]!=valor){
          printf("Valores diferentes!!!\n Vetor[0] = %d\n Vetor[%d] =
%d\n", valor, j, p[j]);
          somaTotal = 0;
       }
   }

  if(somaTotal){
      printf("Todos os valores iguais à %d!!!\n", valor);
  }

   return 0;
}
```

Comando:

```
[c2310289@rainha l5]$ gcc -Wall -o ex2 ex2.c
[c2310289@rainha l5]$ ./ex2
```

Saída com 10000 posições e 2 trabalhadores:

```
Criou o trabalhador 0
Criou o trabalhador 1
Todos os valores iguais à 26!!!
```

Saída com 100000 posições e 100 trabalhadores:

```
[c2310289@rainha l5]$ gcc -Wall -o ex2pt2 ex2pt2.c
[c2310289@rainha l5]$ ./ex2pt2
Criou o trabalhador 0
Criou o trabalhador 1
Criou o trabalhador 2
Criou o trabalhador 3
Criou o trabalhador 4
Criou o trabalhador 5
Criou o trabalhador 6
Criou o trabalhador 7
Criou o trabalhador 8
Criou o trabalhador 9
Valores diferentes!!! Vetor[0] = 7166    Vetor[51122] = 3582
Valores diferentes!!! Vetor[0] = 7166    Vetor[51123] = 3582
Valores diferentes!!! Vetor[0] = 7166    Vetor[51124] = 3582
Valores diferentes!!! Vetor[0] = 7166    Vetor[51125] = 3582
Valores diferentes!!! Vetor[0] = 7166    Vetor[51126] = 3582
Valores diferentes!!! Vetor[0] = 7166    Vetor[51127] = 3582
Valores diferentes!!! Vetor[0] = 7166    Vetor[51128] = 3582
Valores diferentes!!! Vetor[0] = 7166    Vetor[51129] = 3582
Valores diferentes!!! Vetor[0] = 7166    Vetor[51130] = 3582
Valores diferentes!!! Vetor[0] = 7166    Vetor[51131] = 3582
Valores diferentes!!! Vetor[0] = 7166    Vetor[51132] = 3582
Valores diferentes!!! Vetor[0] = 7166    Vetor[51133] = 3582
```

Explicação:

O objetivo do código é inicializar um vetor com 10.000 (ou 100.000) elementos, aplicar uma operação de transformação sobre cada um dos elementos do vetor (multiplicar por

2 e adicionar 2), utilizando duas (10 ou 100) threads, e depois verificar se todos os valores no vetor são idênticos após a operação.

Função trabalhador:

- Essa função será executada pelas threads. Ela realiza operações em cada elemento do vetor p: Multiplica cada elemento por 2 e depois adiciona 2.
- No final, a função usa pthread_exit(NULL) para finalizar a execução da thread.

Função main:

O vetor p é preenchido com o valor 5 em todas as suas 10.000 posições. Um loop é utilizado para criar 2 threads (de acordo com a constante N_THREADS). A função pthread_create é chamada para cada thread, passando como argumento o identificador da thread e a função trabalhador que a thread deve executar. Se a criação da thread falhar, uma mensagem de erro é exibida. Se a thread for criada com sucesso, o programa imprime uma mensagem informando que a thread foi criada.

Depois de criar as threads, a função pthread_join é chamada para garantir que a thread principal (o processo principal do programa) espere até que todas as threads terminem seu trabalho. O programa então verifica se todos os elementos do vetor têm o mesmo valor após o processamento das threads. O valor do primeiro elemento (p[0]) é salvo na variável valor.

O vetor é percorrido, e cada valor é comparado com o valor da primeira posição. Se algum valor for diferente, o programa imprime uma mensagem indicando que existem valores diferentes no vetor. Se todos os valores forem iguais, o programa imprime que todos os valores são iguais.

Esse programa verifica a concorrência entre as tarefas, dessa forma, quando temos um vetor de 10000 posições e duas threads, não houve valores diferentes, pois como era um vetor pequeno e poucos trabalhadores, porém aumentando o número de posições e de threads, verificamos que houve concorrência. A função abaixo nos ajudou a calcular o valor esperado em cada posição(verificamos que a posição zero está com o valor esperado, entretanto demais posições, por conta de problemas de concorrência, apresentaram outros valores).

```
#include <stdio.h>

int main(void){
        long p=5;
        for(int i = 0; i<10; i++){
        p= (p*2) + 2;
        }
        printf("%ld", p);
}
```

Dificuldade:
Tivemos dificuldade para utilizar o pthread_create, pois não estávamos conseguindo mandar o segmento do vetor para o trabalhador, mas conseguimos entender a lógica - conforme o código do exercício 1 mostra.