Relatório do Laboratório 3 de INF 1036 - Sistemas Operacionais

Laboratório 3 - Exercício com Threads

Data: 02/10/2023

Alunos:
Leo Lomardo - matrícula 2020201
Lucas Lucena - matrícula 2010796

Exercício 1: Paralelismo

Objetivo:

O objetivo do programa é calcular a soma total de um grande vetor após cada elemento do vetor ser multiplicado por 2. Ele faz isso de duas maneiras: uma usando threads para paralelizar o processamento e outra executando a operação sequencialmente. O programa visa comparar o desempenho entre as duas abordagens e medir o tempo de execução em ambas.

Estrutura do programa:

Inicialização de um vetor grande vetorGeral com valores iniciais iguais a 5. O tamanho do vetor é definido pela constante TAM_MAX.
Criação de várias threads para processamento paralelo. O número de threads é definido pela constante NUM_THREADS.
Cada thread opera em uma parte do vetor e retorna a soma parcial.
O programa aguarda todas as threads terminarem e calcula a soma total das somas parciais.
Medição do tempo de execução da versão paralela.
Realização da mesma operação de multiplicação e soma, realizada sequencialmente, para fins de comparação.
Medição do tempo de execução da versão sequencial.
Impressão dos resultados, incluindo a soma total e os tempos de execução para ambas as versões.

Solução:

O programa utiliza a biblioteca pthreads para criar threads e realizar o processamento paralelo. Cada thread opera em uma parte do vetor e, em seguida, as somas parciais são

somadas para obter a soma total. A versão sequencial realiza a mesma operação em um único loop.

O código do programa:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

#define TAM_MAX 10000
#define NUM_THREADS 10
#define TAM_PARTES TAM_MAX / NUM_THREADS

int vetorGeral[TAM_MAX];
int vetorSeqGeral[TAM_MAX];

void *operaVet(void *threadID){
      int thread_id = *((int *)threadID);
      int comeco = thread_id * (TAM_PARTES);
int fim = comeco + (TAM_PARTES);
int somaTemp = 0;

for(int i = comeco; i < fim; i++){
vetorGeral[i] *= 2;
somaTemp += vetorGeral[i];
}
return (void *)somaTemp;
}

int main(void){

for(int i = 0; i < TAM_MAX; i++){
      vetorGeral[i] = 5;
}
pthread_t threads[NUM_THREADS];
int thread_id[NUM_THREADS];
int somaTotal,somaSeqTotal = 0;
int i;
clock_t start_time = clock();
```

```
for(i = 0; i < NUM_THREADS; i++){
      thread_id[i] = i;
      pthread_create(&threads[i], NULL, operaVet, &thread_id[i]);
}

for(i = 0; i < NUM_THREADS; i++){
      int somaAux = 0;
      pthread_join(threads[i], (void**)&somaAux);
      somaTotal += somaAux;
}

clock_t end_time = clock();
double execution_time = (double)(end_time - start_time) / CLOCKS_PER_SEC;

printf("Paralelo\nTamanho Vetor:%d\nNúmero Threads:%d\nSoma Total: %d -
Tempo de Execução: %f segundos\n",TAM_MAX, NUM_THREADS, somaTotal,
execution_time);

start_time = clock();

for(i = 0; i < TAM_MAX; i++){
vetorSeqGeral[i] = 5;
vetorSeqGeral[i] *= 2;
somaSeqTotal += vetorSeqGeral[i];
}

end_time = clock();
execution_time = (double)(end_time - start_time) / CLOCKS_PER_SEC;

printf("Sequencial\nTamanhoVetor:%d\nSoma Total: %d - Tempo de Execução:
%f segundos\n",TAM_MAX, somaSeqTotal, execution_time);

return 0;

}
```

Alguns dos nossos resultados:

```
Paralelo
Tamanho Vetor:10000
Número Threads:10
Soma Total: 100000 - Tempo de Execução: 0.000749 segundos
Sequencial
TamanhoVetor:10000
Soma Total: 100000 - Tempo de Execução: 0.000045 segundos
```

```
Paralelo
Tamanho Vetor:10000
Número Threads:100
Soma Total: 100000 - Tempo de Execução: 0.002945 segundos
Sequencial
TamanhoVetor:10000
Soma Total: 100000 - Tempo de Execução: 0.000035 segundos
```

```
Paralelo
Tamanho Vetor:50000
Número Threads:100
Soma Total: 500000 - Tempo de Execução: 0.002982 segundos
Sequencial
TamanhoVetor:50000
Soma Total: 500000 - Tempo de Execução: 0.000159 segundos
```

Observações e conclusões:

Os testes realizados provaram o que já esperávamos, e funcionam perfeitamente. A versão paralela é mais rápida do que a versão sequencial quando o vetor é grande e o número de threads é significativo, pois distribui a carga de trabalho entre vários núcleos de CPU. Além disso, o desempenho da versão paralela pode variar dependendo do hardware, do sistema operacional e da implementação das threads. Por fim, a medição do tempo de execução ajuda a avaliar o impacto do paralelismo no desempenho do programa.

Exercício 2: Concorrência

Objetivo:

O objetivo do programa é inicializar um vetor de 10000 posições com o valor 5, criar 2 trabalhadores para processar o vetor de forma paralela, onde cada thread multiplica cada elemento do vetor por 2 e, em seguida, adiciona 2 a cada elemento. O programa visa verificar se todas as posições do vetor têm valores iguais após o processamento.

Estrutura do programa:

Inicialização de um vetor de 10000 posições, chamado vetorGeral, com valores iniciais iguais a 5.
Criação de dois trabalhadores para processar o vetor em paralelo.
Cada trabalhador opera em uma metade do vetor: o primeiro trabalhador processa as posições de índice 0 a TAM_MAX/2 - 1, e o segundo trabalhador processa as posições de índice TAM_MAX/2 a TAM_MAX - 1.
Cada trabalhador multiplica cada elemento do vetor por 2 e, em seguida, adiciona 2 a cada elemento.
O programa imprime se pelo menos uma posição tem valor diferente das demais.

Solução:

A solução envolve a criação de threads que processam o vetor em paralelo e, em seguida, verifica se as posições do vetor têm valores iguais. Se todas as posições tiverem valores iguais após o processamento, o programa imprimirá que "Todos os valores do vetor são iguais." Caso contrário, imprimirá que "Pelo menos um valor do vetor é diferente."
A estratégia de dividir o vetor em duas partes para processamento paralelo é apropriada para este problema, pois minimiza a possibilidade de condições de corrida e permite uma verificação eficiente de igualdade.

O código do programa:

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

#define TAM_MAX 10000
#define NUM_THREADS 2
```

```c
#define TAM_PARTES TAM_MAX / NUM_THREADS

int vetorGeral[TAM_MAX];

void *operaVet(void *threadID){
int thread_id = *((int *)threadID);
int comeco = thread_id * TAM_PARTES;
int fim = comeco + TAM_PARTES;

for(int i = comeco; i < fim; i++){
vetorGeral[i] = 2*vetorGeral[i] + 2;
}
return NULL;
}

int main(void){
for(int i = 0; i < TAM_MAX; i++){
vetorGeral[i] = 5;
}
pthread_t threads[NUM_THREADS];
int thread_id[NUM_THREADS];
int i;
for(i = 0; i < NUM_THREADS; i++){
thread_id[i] = i;
pthread_create(&threads[i], NULL, operaVet, &thread_id[i]);
}

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

int igual = 1; // Assumimos que todos os valores são iguais

for (i = 0; i < TAM_PARTES; i++) {
if (vetorGeral[i] != vetorGeral[i + TAM_PARTES]) {
igual = 0; // Se encontrarmos um valor diferente, definimos igual como 0
break; // Saímos do loop
}
}
if (igual) {
```

```
printf("Todos os valores do vetor são iguais.\n");
} else {
printf("Pelo menos um valor do vetor é diferente.\n");
}
return 0;
}
```

Uma das soluções (a que está no código) é comparar automaticamente os valores:

```
[lucas@fedora output]$ ./"lab3_ex2"
Todos os valores do vetor são iguais.
[lucas@fedora output]$
```

Só para termos certeza, imprimimos os valores manualmente também:

```
Posicao 9975 valor 12
Posicao 9976 valor 12
Posicao 9977 valor 12
Posicao 9978 valor 12
Posicao 9979 valor 12
Posicao 9980 valor 12
Posicao 9981 valor 12
Posicao 9982 valor 12
Posicao 9983 valor 12
Posicao 9984 valor 12
Posicao 9985 valor 12
Posicao 9986 valor 12
Posicao 9987 valor 12
Posicao 9988 valor 12
Posicao 9989 valor 12
Posicao 9990 valor 12
Posicao 9991 valor 12
Posicao 9992 valor 12
Posicao 9993 valor 12
Posicao 9994 valor 12
Posicao 9995 valor 12
Posicao 9996 valor 12
Posicao 9997 valor 12
Posicao 9998 valor 12
Posicao 9999 valor 12
Todos os valores do vetor são iguais.
```

Observações e conclusões:

O código funcionou perfeitamente em todos os nossos testes.

A conclusão principal do programa é que, após o processamento realizado pelas duas threads, todas as posições do vetor têm valores iguais. Isso significa que as operações de multiplicação por 2 e adição de 2 a cada elemento do vetor foram aplicadas corretamente e de forma consistente a todas as posições, sem resultar em valores diferentes entre as posições.