Relatório Laboratório 2- SIstemas Operacionais

Leo Lomardo - 2020201

Lucas Lucena - 2010796

Exercício 1-

Código:

#include <sys/shm.h>

#include <sys/stat.h>

#include <sys/ipc.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/time.h>

#define NUM\_TRAB 10

#define TAM\_MAX 10000

#define TAM\_PARTES TAM\_MAX / NUM\_TRAB

typedef struct{

int valorInicial[TAM\_MAX];

long somaSegmento[NUM\_TRAB];

}InfoVetor;

int main(void){

InfoVetor \*vetor;

long int somaTotal = 0;

int segmento;

//Estrutura para calcular tempo de execução

struct timeval start\_time, end\_time;

gettimeofday(&start\_time, NULL);

//Memorial Compartilhada

segmento = shmget(IPC\_PRIVATE, sizeof(InfoVetor), IPC\_CREAT | 0666);

if (segmento == -1) {

puts("Erro ao criar a memória compartilhada1");

exit(1);

}

vetor = (InfoVetor\*)shmat(segmento, 0, 0);

if (vetor ==(InfoVetor\*) -1) {

puts("Erro ao criar a memória compartilhada2");

exit(1);

}

for(int i =0; i<TAM\_MAX; i++){

vetor->valorInicial[i] = 5;

}

for(int j = 0; j < NUM\_TRAB; j++){

pid\_t pid = fork();

if(pid == 0){

int comeco = j\*TAM\_PARTES;

int fim = (j+1) \* TAM\_PARTES;

int somaAux = 0;

for(int a = comeco; a < fim; a++){

vetor->valorInicial[a] \*= 2;

somaAux += vetor->valorInicial[a];

}

vetor->somaSegmento[j] = somaAux;

shmdt(vetor);

exit(0);

}else if(pid == -1){

puts("Erro ao criar a memória compartilhada2");

exit(1);

}

}

for (int i = 0; i < NUM\_TRAB; i++) {

wait(NULL);

}

for(int l = 0; l < NUM\_TRAB; l++){

somaTotal+= vetor->somaSegmento[l];

}

gettimeofday(&end\_time, NULL);

double execution\_time = (end\_time.tv\_sec - start\_time.tv\_sec) + (end\_time.tv\_usec - start\_time.tv\_usec) / 1000000.0;

int numTrab = NUM\_TRAB;

int tamVet = TAM\_MAX;

printf("Número de Trabalhadores: %d\n", numTrab);

printf("Tamanho Vetor: %d\n", tamVet);

printf("Tempo de execução: %.8f seg\n", execution\_time);

printf("Soma total: %d\n", somaTotal);

// Libera a memória compartilhada

shmctl(segmento, IPC\_RMID, NULL);

return 0;

}

Medimos o tempo de execução para diferentes configurações de número de trabalhadores e tamanho do vetor. Os resultados estão abaixo:

Número de Trabalhadores: 10

Tamanho Vetor: 10000

Tempo de execução: 0.000555 seg

Soma total: 100000

//

Número de Trabalhadores: 100

Tamanho Vetor: 10000

Tempo de execução: 0.004541 seg

Soma total: 100000

//

Número de Trabalhadores: 5

Tamanho Vetor: 10000

Tempo de execução: 0.000345 seg

Soma total: 100000

//

Número de Trabalhadores: 20

Tamanho Vetor: 10000

Tempo de execução: 0.000891 seg

Soma total: 100000

//

Número de Trabalhadores: 50

Tamanho Vetor: 10000

Tempo de execução: 0.001982 seg

Soma total: 100000

//

Número de Trabalhadores: 50

Tamanho Vetor: 100000

Tempo de execução: 0.002525 seg

Soma total: 1000000

//

Número de Trabalhadores: 10

Tamanho Vetor: 1000000

Tempo de execução: 0.004139 seg

Soma total: 10000000

//

Número de Trabalhadores: 100

Tamanho Vetor: 1000000

Tempo de execução: 0.006567 seg

Soma total: 10000000

Para o código com execução sequencial, mantendo o mesmo tamanho de vetor, obtivemos o tempo médio de execução:

Tempo de execução sequencial: 0.004876 seg

Importante destacar que o aumento no número de trabalhadores, não necessariamente melhorou o tempo de execução do código. Acredito que existe um valor ótimo de trabalhadores, para executar determinada tarefa.

Um ponto que ficamos um pouco perdidos na construção do nosso código, é no caso onde o tamanho do vetor não é divisível pelo número de trabalhadores, como automatizar a divisão das partes do vetor.

Código Execução Sequencial

#include <stdio.h>

#include <stdlib.h>

#include <sys/time.h>

#include <time.h>

#define TAM\_MAX 10000

int main(void){

int vetor[TAM\_MAX];

long int somaVet = 0;

//Estruturas para medir tempo de execução

struct timeval start\_time, end\_time;

gettimeofday(&start\_time, NULL);

for(int i = 0; i< TAM\_MAX ; i++ ){

vetor[i] = 5;

}

for(int k = 0; k <TAM\_MAX; k++){

somaVet+= 2\*vetor[k];

}

gettimeofday(&end\_time, NULL);

double execution\_time = (end\_time.tv\_sec - start\_time.tv\_sec) + (end\_time.tv\_usec - start\_time.tv\_usec) / 1000000.0;

printf("Soma Total: %d\n", somaVet);

printf("Tempo de execução sequencial: %.6f seg\n", execution\_time);

return 0;

}