INF1316 - SISTEMAS OPERACIONAIS - 2022.2 - 3WA
Tarefa 2 - Threads, tempo sequencial e tempo paralelo, comparação com processos

Nome: Eric Leão Matrícula: 2110694
Nome: Pedro Machado Peçanha Matrícula: 2110535

1000 -> bloco de 125

erros com fork = 0, tendo durado 1.285000 ms
erros com thread = 0, tendo durado 0.853000 ms
erros sequencial = 0, tendo durado 0.886000 ms

10000 -> bloco de 1250

erros com fork = 0, tendo durado 1.256000 ms
erros com thread = 0, tendo durado 0.860000 ms
erros sequencial = 0, tendo durado 1.003000 ms

100000 -> bloco de 12500

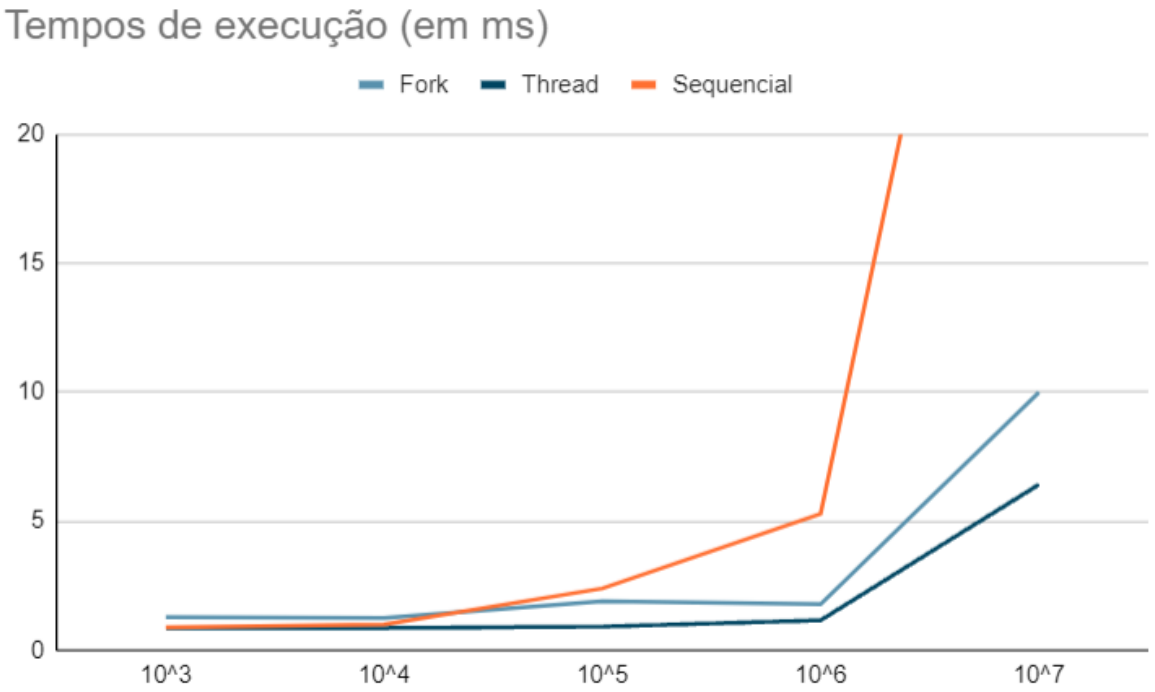
erros com fork = 0, tendo durado 1.895000 ms
erros com thread = 0, tendo durado 0.915000 ms
erros sequencial = 0, tendo durado 2.398000 ms

1000000 -> bloco de 125000

erros com fork = 0, tendo durado 1.786000 ms
erros com thread = 0, tendo durado 1.160000 ms
erros sequencial = 0, tendo durado 5.290000 ms

10000000 -> bloco de 1250000

erros com fork = 0, tendo durado 10.000000 ms
erros com thread = 0, tendo durado 6.422000 ms
erros sequencial = 0, tendo durado 45.367001 ms



(limitamos em 20ms para melhorar a visualização das informações)

Conseguimos fazer o código e entendemos que ele está funcionando por completo. Ao analisar os resultados utilizando os tamanhos solicitados pelo enunciado, percebemos que a versão com thread executa a tarefa mais rapidamente em todos os casos de teste. No entanto, para valores baixos, como 1000 e 10000, a diferença de tempo pode variar de tal forma a fazer com que o tempo com thread seja equivalente ou até mais lento que o tempo sequencial. Ao aumentar o tamanho da array, o tempo da versão com thread aumenta de forma mais devagar do que o tempo da versão sequencial, tornando a versão com múltiplas threads mais eficiente quando utilizados arrays de tamanhos grandes. Isso ocorre, pois o tempo para se criar uma thread é relativamente grande, demorando cerca de 0.212000 para serem criadas 8 threads. Ao analisar o gráfico podemos perceber que ao se utilizar uma divisão de threads o tempo que se leva para somar 2 vetores aumenta de forma consideravelmente mais demorada.

Além disso, ao compararmos os tempos utilizando fork e utilizando threads, podemos perceber que ao se utilizar threads, temos um resultado sempre superior. Isso ocorre, pois o tempo para se criar threads é menor que o tempo para se criar forks, tendo em vista que threads precisam carregar menos coisas na memória. O tempo para se criar 8 threads é cerca de 4 vezes mais rápido do que o de criar 8 forks.

8 threads = 0.212000
8 forks = 0.906000

```
/*

INF1316 - SISTEMAS OPERACIONAIS - 2022.2 - 3WA
Tarefa 2 - Threads, tempo sequencial e tempo paralelo, comparação com
```

```
processos
Nome: Eric Leão Matrícula: 2110694
Nome: Pedro Machado Peçanha Matrícula: 2110535

*/

#include <math.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <unistd.h>

#define TAM 1000
#define BLOCO 125
#define VAR1 1
#define VAR2 2

int arrt1[TAM], arrt2[TAM], arrt3[TAM];

void *somaVetor(void *a) {
  for (long i = (long)a * BLOCO; i < (long)a * BLOCO + BLOCO; i += 1) {
    arrt3[i] = arrt1[i] + arrt2[i];
  }
  pthread_exit(NULL); /*not necessary*/
}

void *fakethread(void *a) { pthread_exit(NULL); /*not necessary*/ }

int main(void) {
  int segmento1, segmento2, segmento3, *arr1, *arr2, *arr3, status, id;
  struct timeval inicio, fim;
  double total;
  segmento1 = shmget(IPC_PRIVATE, TAM * sizeof(int),
                     IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
  segmento2 = shmget(IPC_PRIVATE, TAM * sizeof(int),
                     IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
  segmento3 = shmget(IPC_PRIVATE, TAM * sizeof(int),
                     IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);

  arr1 = (int *)shmat(segmento1, 0, 0);
  arr2 = (int *)shmat(segmento2, 0, 0);
  arr3 = (int *)shmat(segmento3, 0, 0);
```

```
  long a;
  for (a = 0; a < TAM; a++) {
    arr1[a] = VAR1;
    arr2[a] = VAR2;
    arr3[a] = 0;
  }
  gettimeofday(&inicio, NULL);
  for (a = 0; a < TAM / BLOCO; a++) {
    if ((id = fork()) < 0) {
      puts("Erro na criação do novo processo");
      exit(-2);
    } else if (!id) {
      for (int i = a * BLOCO; i < a * BLOCO + BLOCO; i += 1) {
        arr3[i] = arr1[i] + arr2[i];
      }
      shmdt(arr1);
      shmdt(arr2);
      shmdt(arr3);
      exit(0);
    }
  }
  for (int i = 0; i < TAM / BLOCO; i++)
    wait(&status);
  shmctl(segmento1, IPC_RMID, 0);
  shmctl(segmento2, IPC_RMID, 0);
  shmctl(segmento3, IPC_RMID, 0);
  gettimeofday(&fim, NULL);
  total = (fim.tv_sec - inicio.tv_sec) * 1000.0f +
          (fim.tv_usec - inicio.tv_usec) / 1000.0f;
  int maxError = 0;
  int diffError = 0;
  for (a = 0; a < TAM; a++)
    maxError =
        (maxError > (diffError = fabs((double)(arr3[a] - (VAR1 +
VAR2)))))
            ? maxError
            : diffError;
  printf("erros com fork = %d, tendo durado %f ms\n", maxError, total);

  pthread_t threads[TAM / BLOCO];
  for (a = 0; a < TAM; a++) {
    arrt1[a] = VAR1;
    arrt2[a] = VAR2;
    arrt3[a] = 0;
  }
  gettimeofday(&inicio, NULL);
```

```
  for (a = 0; a < TAM / BLOCO; a++) {
    pthread_create(&threads[a], NULL, somaVetor, (void *)a);
  }
  for (long i = 0; i < TAM / BLOCO; i++)
    pthread_join(threads[i], NULL);
  gettimeofday(&fim, NULL);
  total = (fim.tv_sec - inicio.tv_sec) * 1000.0f +
          (fim.tv_usec - inicio.tv_usec) / 1000.0f;
  maxError = 0;
  diffError = 0;
  for (a = 0; a < TAM; a++)
    maxError =
        (maxError > (diffError = fabs((double)(arr3[a] - (VAR1 +
VAR2))))) 
            ? maxError
            : diffError;
  printf("erros com thread = %d, tendo durado %f ms\n", maxError,
total);
  for (a = 0; a < TAM; a++)
    arr3[a] = arr2[a] + arr1[a];
  gettimeofday(&fim, NULL);
  /*
  for (a = 0; a < TAM / BLOCO; a++) {
    pthread_create(&threads[a], NULL, fakethread, NULL);
  }
  for (long i = 0; i < TAM / BLOCO; i++)
    pthread_join(threads[i], NULL);
*/
  total = (fim.tv_sec - inicio.tv_sec) * 1000.0f +
          (fim.tv_usec - inicio.tv_usec) / 1000.0f;
  maxError = 0;
  diffError = 0;
  for (a = 0; a < TAM; a++)
    maxError =
        (maxError > (diffError = fabs((double)(arr3[a] - (VAR1 +
VAR2))))) 
            ? maxError
            : diffError;
  printf("erros sequencial = %d, tendo durado %f ms\n", maxError,
total);
  return 0;
}
```