INF1316 - SISTEMAS OPERACIONAIS - 2022.2 - 3WA
Tarefa 1 - memória compartilhada, tempo sequencial x tempo paralelo

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

1000 -> bloco de 125

erros com fork = 0, tendo durado 0.001028 segundos
erros sem fork = 0, tendo durado 0.000003 segundos

10000 -> bloco de 1250

erros com fork = 0, tendo durado 0.001034 segundos
erros sem fork = 0, tendo durado 0.000007 segundos

100000 -> bloco de 12500

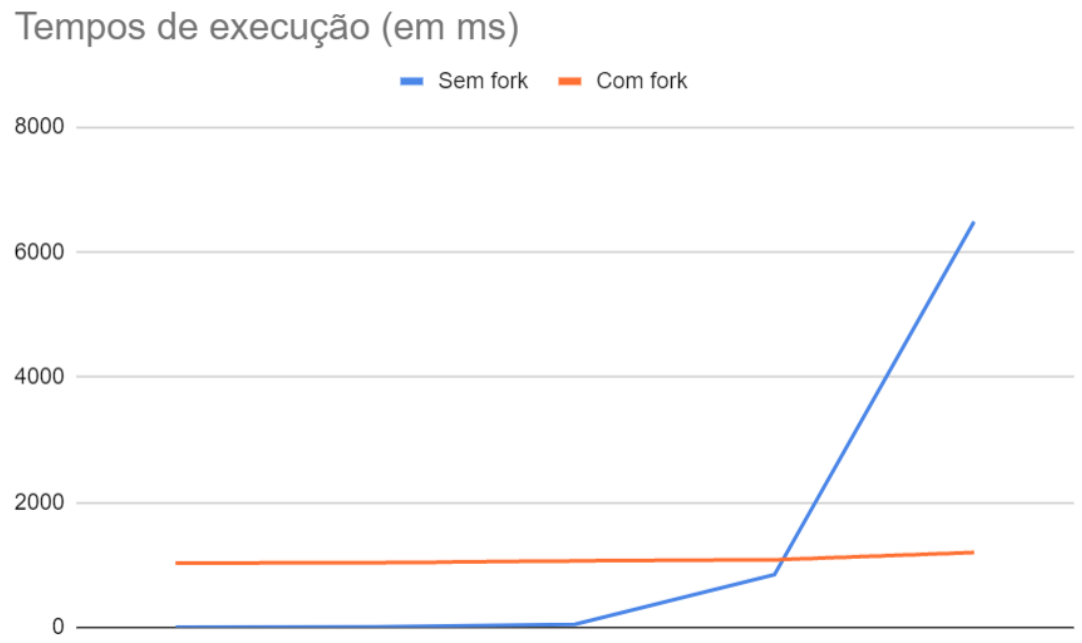
erros com fork = 0, tendo durado 0.001064 segundos
erros sem fork = 0, tendo durado 0.000051 segundos

1000000 -> bloco de 125000

erros com fork = 0, tendo durado 0.001081 segundos
erros sem fork = 0, tendo durado 0.000845 segundos

10000000 -> bloco de 1250000

erros com fork = 0, tendo durado 0.001198 segundos
erros sem fork = 0, tendo durado 0.006492 segundos



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 sem fork executa a tarefa mais rapidamente, com uma vantagem considerável. Contudo, ao aumentar o tamanho da array, o tempo da versão com fork aumenta de forma mais devagar do que o tempo da versão sem fork, tornando a versão com múltiplos processos mais eficiente quando utilizados arrays de tamanhos grandes. Isso ocorre, pois o tempo para se criar um processo é relativamente grande, demorando cerca de 0.000906 para serem criados 8 processos. Ao analisar o gráfico podemos perceber que ao se utilizar uma divisão de processos o tempo que se leva para somar 2 vetores aumenta de forma consideravelmente mais demorada.

Após adicionar um “fake fork” no código podemos ver como os novos tempos se comportam:

1000 -> bloco de 125

erros com fork = 0, tendo durado 0.000800 segundos
erros sem fork = 0, tendo durado 0.000855 segundos

10000 -> bloco de 1250

erros com fork = 0, tendo durado 0.000823 segundos
erros sem fork = 0, tendo durado 0.000878 segundos

100000 -> bloco de 12500

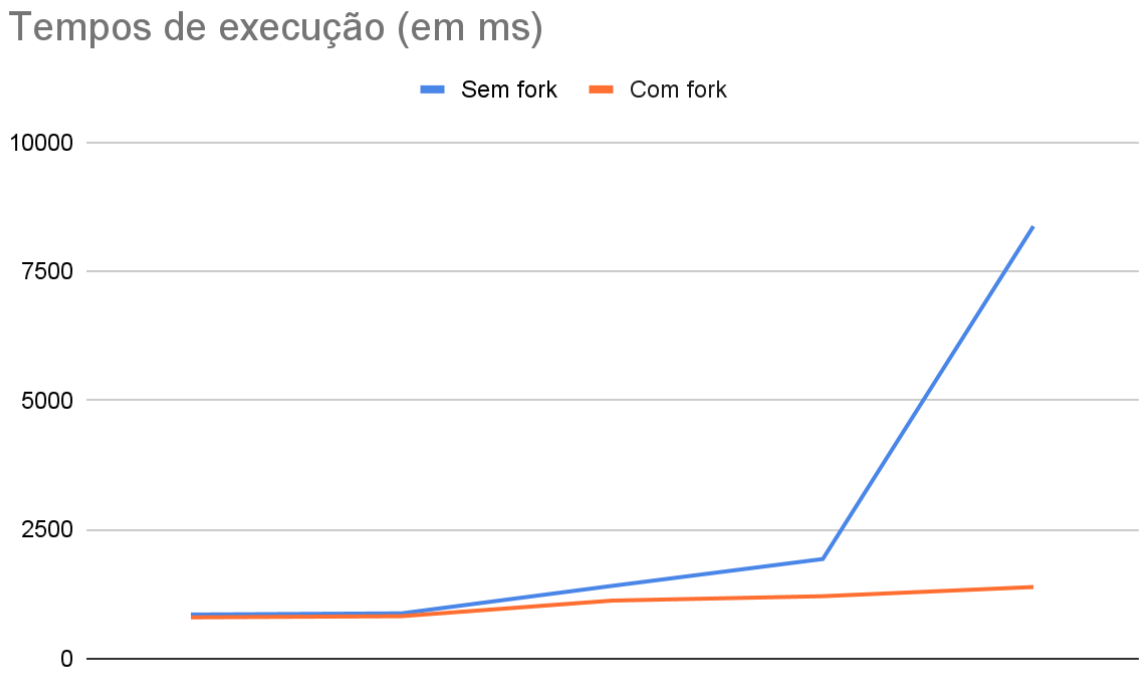
erros com fork = 0, tendo durado 0.001125 segundos
erros sem fork = 0, tendo durado 0.001410 segundos

1000000 -> bloco de 125000

erros com fork = 0, tendo durado 0.001210 segundos
erros sem fork = 0, tendo durado 0.001930 segundos

10000000 -> bloco de 1250000

erros com fork = 0, tendo durado 0.001338 segundos
erros sem fork = 0, tendo durado 0.008374 segundos



Código:

```
/*INF1316 - SISTEMAS OPERACIONAIS - 2022.2 - 3WA
Tarefa 1 - memória compartilhada, tempo sequencial x tempo paralelo
Nome: Eric Leão     Matrícula: 2110694
Nome: Pedro Machado Peçanha    Matrícula: 2110535*/

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

#define TAM 10000000
#define BLOCO 1250000
#define VAR1 1
#define VAR2 2
#define CLOCKS_PER_SEC ((__clock_t) 1000000)

int main(void) {
  int segmento1, segmento2, segmento3, *arr1, *arr2, *arr3, status, id;
  clock_t start, stop;
  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);
  int a;
  for (a = 0; a < TAM; a++) {
    arr1[a] = VAR1;
    arr2[a] = VAR2;
    arr3[a] = 0;
  }
  start = clock();
  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);
  stop = clock();
  total = (double)(stop - start) / CLOCKS_PER_SEC;
  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 segundos\n", maxError,
```

```
total);
  for (a = 0; a < TAM; a++) {
    arr3[a] = 0;
  }
  start = clock();
  // fake fork, tire o comentario para usa-lo
/*
  for (int a = 0; a < TAM / BLOCO; a++) {
    if ((id = fork()) < 0) {
      puts("Erro na criação do novo processo");
      exit(-2);
    } else if (!id) {
      exit(0);
    }
  }
  for (int i = 0; i < TAM / BLOCO; i++)
    wait(&status);
  */
  for (a = 0; a < TAM; a++)
    arr3[a] = arr2[a] + arr1[a];
  stop = clock();
  total = (double)(stop - start) / CLOCKS_PER_SEC;
  for (a = 0; a < TAM; a++)
    maxError =
        (maxError > (diffError = fabs((double)(arr3[a] - (VAR1 +
VAR2))))
            ? maxError
            : diffError;
  printf("erros sem fork = %d, tendo durado %f segundos\n", maxError,
total);
  return 0;
}
```