Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Programação Concorrente

Relatório de Entrega de Atividades

Aluno(s): Amanda Oliveira Alves e Fillype Alves do Nascimento

Matrícula: 15/0116276 e 16/0070431

Atividade: Aula Prática 02 - Threads

1.1.1

```c
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: hello-thread.c
// atividade: 1.1.1

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

void *rotina(){
    printf("Olá, sou uma thread\n");
    sleep(2);
}

int main(){
    pthread_t t;
    pthread_create(&t, NULL, rotina, NULL);
    pthread_create(&t, NULL, rotina, NULL);
    printf("Olá, sou a main.\n");
return 0;
}
```

1.1.2 - Execute o programa implementado no tópico anterior diversas vezes e elenque os motivos os quais as saídas entre execuções são diferentes entre si.

R: Isso se deve a diversos fatores relacionados ao escalonamento das threads que estão prontas para a execução. Se você estiver executando em uma máquina com mais de um núcleo, lembre-se que há o mapeamento da linha de execução que você criou com uma thread no modo núcleo (modelo 1x1) e estas, individualmente, também concorrem entre si pelo recurso de E/S (no caso, saída padrão) para exibir o seu resultado, quando ganham o processador. Por essas diversas influências e ainda outros motivos que serão explorados nesse curso, não se pode fazer assunções temporais sobre a execução de threads ou processos.

2.1.1

```c
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: contador.c
// atividade: 2.1.1

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

void *quadratica(void *x) {
   double *result = malloc(sizeof(double));
   *result = 10 * *(double *)x * *(double *)x;
   pthread_exit(result);
}

void *cubica(void *x) {
   double *result = malloc(sizeof(double));
   *result = 42 * *(double *)x * *(double *)x * *(double *)x;
   pthread_exit(result);
}

int main() {
```

```
   pthread_t t1, t2;
   double x;
   printf("Digite o valor de x: ");
   scanf("%lf", &x);
   void *r1, *r2;
   pthread_create(&t1, NULL, quadratica, (int *)&x);
   pthread_create(&t2, NULL, cubica, (int *)&x);
   pthread_join(t1, &r1);
   pthread_join(t2, &r2);
   double result = *(double *)r1 + *(double *)r2;
   printf("f(%.4f)= %.4f\n", x, result);
   free(r1);
   free(r2);
   return 0;
}
```

3.1.1

```
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: contador.c
// atividade: 3.1.1

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

void *contador(void *id){
 long int tid = (long int )id;
 int n = 1, i;
 while(n <= 5){
```

```
    printf("Sou a thread %ld e estou no número %d.\n", tid, n);
    n++;
  }
}

int main(){
  const int NUMTHREADS = 10;
  pthread_t threads[NUMTHREADS];
  for (long int i = 0; i < NUMTHREADS; i++) {
    pthread_create(&threads[i], NULL, contador, (void *) i);
  }
  for (long int i = 0; i < NUMTHREADS; i++) {
    pthread_join(threads[i], NULL);
  }
return 0;
}
```

3.2.2

```
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: escalonador.c
// atividade: 3.1.2

#define _GNU_SOURCE
#include <stdio.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <math.h>
#include <sched.h>

void *contador(void *id){
  long int tid = (long int )id;
```

```
 int n = 1, i;
 while(n <= 5){
   printf("Sou a thread %ld e estou no número %d.\n", tid, n);
   sched_yield();
   n++;
 }
}

int main(){
 const int NUMTHREADS = 10;
 pthread_t threads[NUMTHREADS];
 for (long int i = 0; i < NUMTHREADS; i++) {
   pthread_create(&threads[i], NULL, contador, (void *) i);
 }
 for (long int i = 0; i < NUMTHREADS; i++) {
   pthread_join(threads[i], NULL);
 }
 cpu_set_t mascaranucleos;
 CPU_ZERO(&mascaranucleos);
 CPU_SET(0, &mascaranucleos);
 sched_setaffinity(0, sizeof(cpu_set_t), &mascaranucleos);
 return 0;
}
```

4.1.1 - De maneira análoga a demonstrada entre threads, variáveis globais podem compartilhar informações em tempo de execução entre dois processos diferentes, onde um dos processos foi criado pela chamada fork do primeiro? Por quê?

R: Threads compartilham a mesma área de memória, portanto no caso de uma variável global, ela será compartilhada identicamente entre as threads. Já no caso de um fork os processos criam uma cópia independente da variável sem o compartilhamento de memória, então se a variável global for modificada no processo pai, ela não será modificada automaticamente no processo filho.

4.1.2

```c
// autor: Amanda Oliveira Alves e Fillype Alves do Nascimento
// arquivo: escalonador.c
// atividade: 4.1.2

#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include<time.h>

int largada = 3;

void *juizdelargada() {
   while(largada > 0){
        printf("%d\n", largada);
        largada--;
        srand(time(NULL));
        int ra = rand() % 10;
        sleep(ra);
   }
   printf("GO!\n");
   pthread_exit(NULL);
}

void *carro(void *ID) {
   if(largada == 0) {
        printf("Sou o carro %d e terminei a corrida.\n", ID);
        pthread_exit(NULL);
   }
}

int main() {
   pthread_t t;
   pthread_create(&t, NULL, juizdelargada, NULL);
```

```
    pthread_join(t, NULL);

    const int NUMTHREADS = 10;
    pthread_t threads[NUMTHREADS];
    for (long int i = 0; i < NUMTHREADS; i++) {
        pthread_create(&threads[i], NULL, carro, (void *) i);
    }
    for (long int i = 0; i < NUMTHREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```