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

Relatório de Entrega de Atividades

Aluno(s): Henrique Mendes de Freitas Mariano; Leonardo Rodrigues de Souza.
Matrícula: 170012280; 170060543
Atividade: Aula Prática 02 - Threads

1.1.1 - Hello Threads

```c
// autores: Henrique Mendes de Freitas Mariano e Leonardo Rodrigues de Souza
// arquivo: 1-1-1-hello-threads.c
// atividade: 1.1.1

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

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

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 - Devido a função printf não ser reentrante, as chamadas a mesma são acumuladas, causando um comportamento anômalo em diferentes chamadas.

2.1.1 - Divisão de Tarefas

```c
// autores: Henrique Mendes de Freitas Mariano e Leonardo Rodrigues de Souza
// arquivo: 2-1-1-divisao-de-tarefas.c
// atividade: 2.1.1

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

void* solve(void* args){
    double* x = (double*)args;
    double* ans = malloc(sizeof(double));
    *ans = (pow(*x, *(x + 2)) * *(x + 1));
    pthread_exit(ans);
}

int main() {
    pthread_t t1, t2;
    double x;
    void *ans1, *ans2;
    scanf("%lf", &x);
    double* args1 = malloc(sizeof(double) * 3);
    double* args2 = malloc(sizeof(double) * 3);
    *(args1) = x;
    *(args1 + 1) = 10;
    *(args1 + 2) = 2;
    *(args2) = x;
    *(args2 + 1) = 42;
    *(args2 + 2) = 3;
    pthread_create(&t1, NULL, solve, args1);
    pthread_create(&t2, NULL, solve, args2);
    pthread_join(t1, &ans1);
    pthread_join(t2, &ans2);
    printf("A resposta é: %.4lf\n", *(double*)ans1 + *(double*)ans2);
```

```
    free(args1);
    free(args2);
    free(ans1);
    free(ans2);
    return 0;
}
```

3.1.1 - Iterações entre Threads

```
// autores: Henrique Mendes de Freitas Mariano e Leonardo Rodrigues de Souza
// arquivo: 3-1-1-n-threads.c
// atividade: 3.1.1

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

void *contador(void *id){
    long int tid = (long int )id;
    for(int i = 0; i < 5; i++) printf("Sou a thread TID %ld e estou no
número %d\n", tid, i);
}
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;
}
```

3.1.2 - Escalonador

```c
// autores: Henrique Mendes de Freitas Mariano e Leonardo Rodrigues de Souza
// arquivo: 3-1-2-escalonador.c
// atividade: 3.1.2

#define _GNU_SOURCE
#include <stdio.h>
#include <pthread.h>
#include <sched.h>

void *contador(void *id){
    long int tid = (long int )id;
    for(int i = 0; i < 5; i++) {
        printf("Sou a thread TID %ld e estou no número %d\n", tid, i);
        sched_yield();
    }
}

int main(){
    cpu_set_t mascaranucleos;
    CPU_ZERO(&mascaranucleos);
    CPU_SET(0, &mascaranucleos);
    sched_setaffinity(0, sizeof(cpu_set_t), &mascaranucleos);
    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;
}
```

4.1.1 - O comportamento de compartilhamento de variáveis globais entre processos não ocorre, em comparação as threads. Ao criar um novo processo, utilizando fork, o escopo do processo pai é também clonado, criando um novo escopo global para o processo filho. Portanto, **alterações deste escopo em diferentes processos não se refletem nos demais.** Assim, apesar do fork clonar o escopo do processo pai, os dois não **compartilham o mesmo espaço de endereçamento**.

4.1.2 - Corrida de Threads

```c
// autores: Henrique Mendes de Freitas Mariano e Leonardo Rodrigues de Souza
// arquivo: 4-1-2-corrida.c
// atividade: 4.1.2

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

int largada = 3;

void* corrida(void* i) {
    int ID = *((int*)i);
    while (largada != 0)
        continue;
    printf("Sou carro ID %d e terminei a corrida\n", ID);
    pthread_exit(NULL);
}

void* juizdelargada(){
    while(largada) {
        largada--;
        if (!largada) break;
        sleep(rand() % 10);
    }

    printf("GO!\n");
    pthread_exit(NULL);
}

int main(){
    const int NUMTHREADS = 10;
    pthread_t threads[NUMTHREADS + 1];

    for (int i = 0; i < NUMTHREADS; i++)
        pthread_create(&threads[i], NULL, corrida, (void*)&i);
```

```
    pthread_create(&threads[NUMTHREADS], NULL, juizdelargada, NULL);

    for(int i = 0; i <= NUMTHREADS; i++)
        pthread_join(threads[i], NULL);
    return 0;
}
```