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 03 - Condições de Corrida

1.1.2 - Devido ao escopo global da variável, bem como a inexistência da exclusão mútua entre as threads. Por algumas vezes mais de uma thread tenta incrementar a mesma unidade da variável ao mesmo tempo, ou seja, mais de uma thread vai incrementar a variável de 90 para 91, porém como elas executam essa operação ao mesmo tempo, para todas elas a variável está em 90 e deve ir para 91. Assim, a variável que deveria ser incrementada 1000 vezes ao total acaba totalizando valores diferentes dependendo da execução dessas threads. Este problema não ocorre em todas a vezes, devido a aleatoriedade da ordem de execução das threads.

1.1.3 - Utilizando algorítmos para exclusão mútua para N threads, como o de Djikstra, seria possível garantir o correto funcionamento do programa, bem como a correta execução de turnos entre as threads.

1.2 -

1. Dois ou mais processos não podem estar acessando simultaneamente a mesma região crítica.
2. Não é possível fazer considerações a respeito da velocidade de execução dos processos, ou a respeito dos processos disponíveis.
3. Nenhum processo pode esperar indefinidamente para entrar na sua região crítica.
4. Nenhum processo fora da região crítica pode bloquear a execução de outro processo.

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

2.1.1 -

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

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

const int NUMTHREADS = 2;
int contabancaria = 0, vez = 0;

bool interessados[2] = {false, false};

void enter_region(int processo){
    int outro;
    outro = 1 - processo;
    interessados[processo] = true;
    vez = processo;
    while(vez == processo && interessados[outro] == true) {}
}

void out_region(int processo){
    interessados[processo] = false;
}

void* deposito(void* arg){
    int i = *((int *) arg);
    while(true){
        enter_region(i);
        contabancaria += 20;
        printf("Contabancaria: %d\n", contabancaria);
        out_region(i);
    }
```

```
}

void* retirada(void* arg){
    int i = *((int *) arg + 1);
    while(true){
        enter_region(i);
        if(contabancaria - 10 >= 0) contabancaria += -10;
        printf("Contabancaria: %d\n", contabancaria);
        out_region(i);
    }
}

int main(){
    pthread_t threads[NUMTHREADS];
    int *arg = malloc(sizeof(int) * 2);

    *arg = 0;
    *(arg + 1) = 1;

    pthread_create(&threads[0], NULL, deposito, (void *) arg);
    pthread_create(&threads[1], NULL, retirada, (void *) arg);

    pthread_join(threads[0], NULL);
    pthread_join(threads[1], NULL);

    free(arg);
    return 0;
}
```

2.1.2 - Apenas a thread responsável pela retirada irá rodar e portanto a conta bancária ficará em zero infinitamente.

```c
// autores: Henrique Mendes de Freitas Mariano e Leonardo Rodrigues de Souza
// arquivo: 2-1-2-conta-bancaria.c
// atividade: 2.1.2

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

const int NUMTHREADS = 2;
int contabancaria = 0, vez = 0;

bool interessados[2] = {false, false};

void enter_region(int processo){
    int outro;
    outro = 1 - processo;
    interessados[processo] = true;
    vez = processo;
    while(vez == processo && interessados[outro] == true) {}
}

void out_region(int processo){
    interessados[processo] = false;
}

void* deposito(void* arg){
    int i = *((int *) arg);
    while(true){
        enter_region(i);
        contabancaria += 20;
        printf("Contabancaria: %d\n", contabancaria);
```

```
        out_region(i);
        pthread_exit(NULL);
    }
}

void* retirada(void* arg){
    int i = *((int *) arg + 1);
    while(true){
        enter_region(i);
        if(contabancaria - 10 >= 0) contabancaria += -10;
        printf("Contabancaria: %d\n", contabancaria);
        out_region(i);
    }
}

int main(){
    pthread_t threads[NUMTHREADS];
    int *arg = malloc(sizeof(int) * 2);

    *arg = 0;
    *(arg + 1) = 1;

    pthread_create(&threads[0], NULL, deposito, (void *) arg);
    pthread_create(&threads[1], NULL, retirada, (void *) arg);

    pthread_join(threads[0], NULL);
    pthread_join(threads[1], NULL);

    free(arg);
    return 0;
}
```

2.1.3 -

```c
// autores: Henrique Mendes de Freitas Mariano e Leonardo Rodrigues de Souza
// arquivo: 2-1-3-conta-bancaria.c
// atividade: 2.1.3

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

const int NUMTHREADS = 2;
int contabancaria = 0, vez = 0;

bool interessados[2] = {false, false};

void enter_region(int processo){
    int outro;
    outro = 1 - processo;
    interessados[processo] = true;
    vez = processo;
    while(vez == processo && interessados[outro] == true) {}
}

void out_region(int processo){
    interessados[processo] = false;
}

void* deposito(void* arg){
    int i = *((int *) arg);
    for(int j = 0; j < 10000; j++) {
        enter_region(i);
        contabancaria += 20;
        printf("Contabancaria: %d\n", contabancaria);
        out_region(i);
    }
```

```
    pthread_exit(NULL);
}

void* retirada(void* arg){
    int i = *((int *) arg + 1);
    while(true){
        enter_region(i);
        if(contabancaria - 10 >= 0) contabancaria += -10;
        printf("Contabancaria: %d\n", contabancaria);
        out_region(i);
    }
}

int main(){
    pthread_t threads[NUMTHREADS];
    int *arg = malloc(sizeof(int) * 2);

    *arg = 0;
    *(arg + 1) = 1;

    pthread_create(&threads[0], NULL, deposito, (void *) arg);
    pthread_create(&threads[1], NULL, retirada, (void *) arg);

    pthread_join(threads[0], NULL);
    pthread_join(threads[1], NULL);

    free(arg);
    return 0;
}
```

2.1.4 - Algoritmo de Djikstra para 3 threads

```c
// autores: Henrique Mendes de Freitas Mariano e Leonardo Rodrigues de Souza
// arquivo: 2-1-4-conta-bancaria.c
// atividade: 2.1.4

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

#define max(a,b) \
   ({ __typeof__ (a) _a = (a); \
       __typeof__ (b) _b = (b); \
     _a > _b ? _a : _b; })

const int NUMTHREADS = 3;
int contabancaria = 0, vez = 0;

int escolhendo[3] = {0, 0, 0};

int numero[3] = {0, 0, 0};

void* deposito(void* arg){
    int i = *((int *) arg);
    int j;
    while(true){
        int val_max = -1;
        escolhendo[i] = 1;

        val_max = max(numero[0], numero[1]);
        val_max = max(val_max, numero[2]);

        numero[i] = 1 + val_max;
        escolhendo[i] = 0;
```

```
        for(j = 0; j < NUMTHREADS; j++){
            while(escolhendo[j] != 0){}
            while(numero[j] != 0 && ((numero[j] < numero[i]) || ((numero[j]
== numero[i]) && j < i))){}
        }
        // sessao critica
        contabancaria += 20;
        printf("Contabancaria: %d i: %d\n", contabancaria, i);
        numero[i] = 0;
        //sessao nao critica
    }
}

void* retirada(void* arg){
    int i = *((int *) arg);
    int j;
    while(true){
        int val_max = -1;
        escolhendo[i] = 1;

        val_max = max(numero[0], numero[1]);
        val_max = max(val_max, numero[2]);

        numero[i] = 1 + val_max;
        escolhendo[i] = 0;

        for(j = 0; j < NUMTHREADS; j++){
            while(escolhendo[j] != 0){}
            while(numero[j] != 0 && numero[j] < numero[i]){}
        }
        // sessao critica
        if(contabancaria - 10 >= 0) contabancaria += -10;
        printf("Contabancaria: %d i: %d\n", contabancaria, i);
        numero[i] = 0;
        //sessao nao critica
    }
}
```

```c
int main(){
    pthread_t threads[NUMTHREADS];
    int *arg_1 = malloc(sizeof(int));
    int *arg_2 = malloc(sizeof(int));
    int *arg_3 = malloc(sizeof(int));

    *arg_1 = 0;
    *arg_2 = 1;
    *arg_3 = 2;

    pthread_create(&threads[0], NULL, deposito, (void *) arg_1);
    pthread_create(&threads[1], NULL, retirada, (void *) arg_2);
    pthread_create(&threads[2], NULL, retirada, (void *) arg_3);

    pthread_join(threads[0], NULL);
    pthread_join(threads[1], NULL);
    pthread_join(threads[2], NULL);

    free(arg_1);
    free(arg_2);
    free(arg_3);

    return 0;
}
```

3.1.1 - Incremento Atômico

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

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

volatile int contador = 0;

void* incrementos(){
    for(int i = 0; i < 100; i++)
        __sync_fetch_and_add(&contador, 1);
}

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

    for (int i = 0; i < NUMTHREADS; i++)
        pthread_create(&threads[i], NULL, incrementos, NULL);

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

    printf("Contador = %d\n", contador);

    return 0;
}
```

3.1.2 - Foi convertida em **lock addl $1, contador(%rip)**.