Sistemas Operativos

Relatório do Trabalho Teórico-Prático

Junho de 2022

Fernando Fuzeiro Nº22111
Vasco Araújo Nº23055

ipt
Instituto Politécnico de Tomar

Índice

03 Objetivos

04 Fluxograma - Main

05 Fluxograma - Produtora

06 Fluxograma - Consumidora

07 Código Comentado

10 Descrição resumida do código

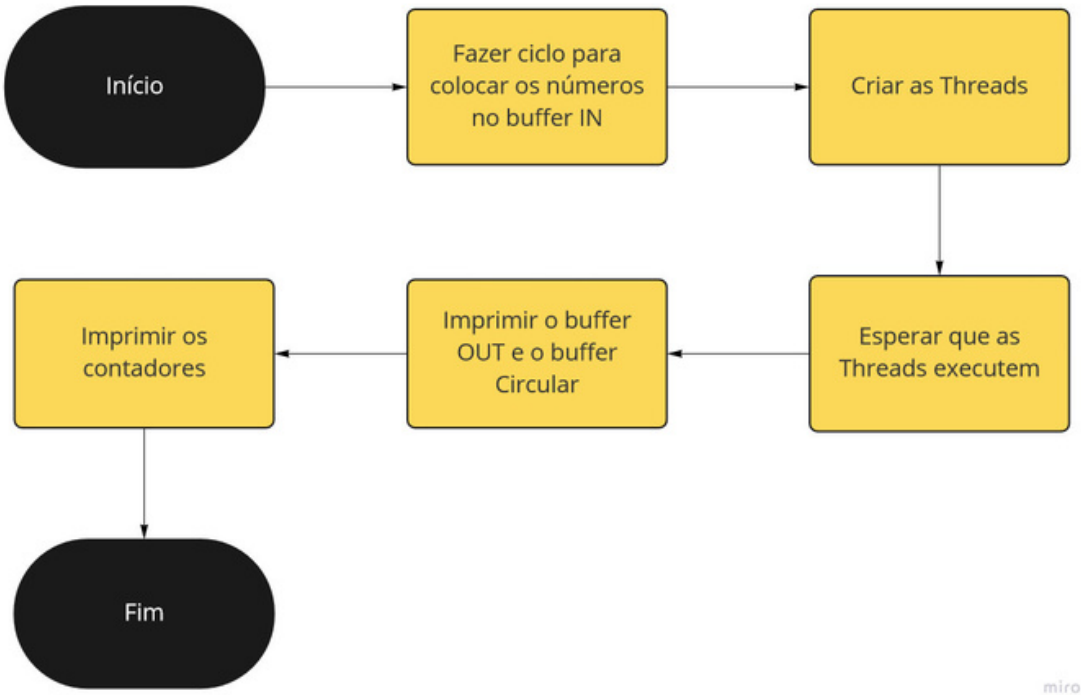
13 Conclusão

Objetivos

- Familiarização com o ambiente de desenvolvimento utilizado nas aulas (Unix).
- Programação em Unix (linguagem C) utilizando chamadas ao sistema (system calls).
- Comunicação e Sincronização de Threads.

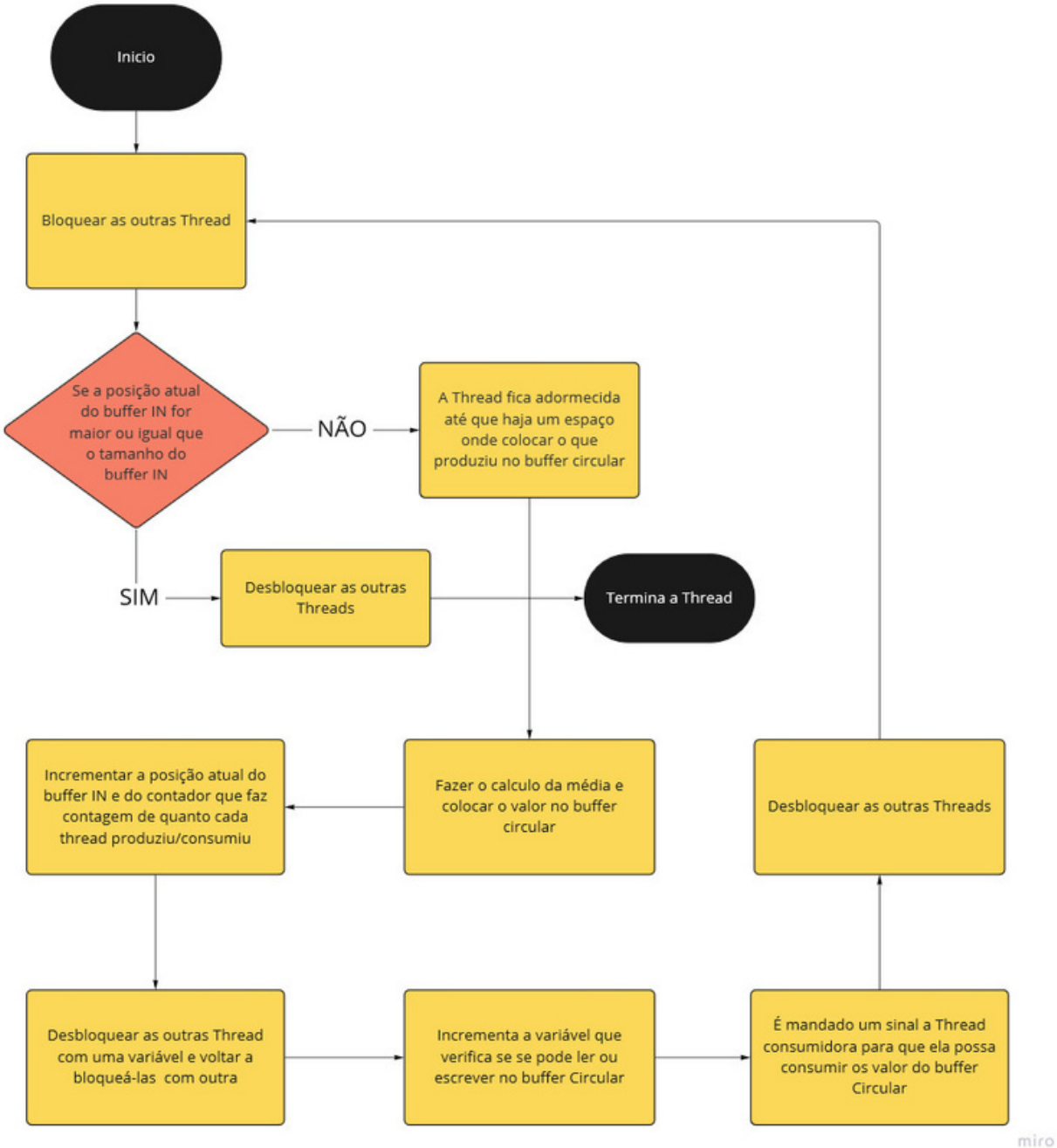
Fluxograma

Main



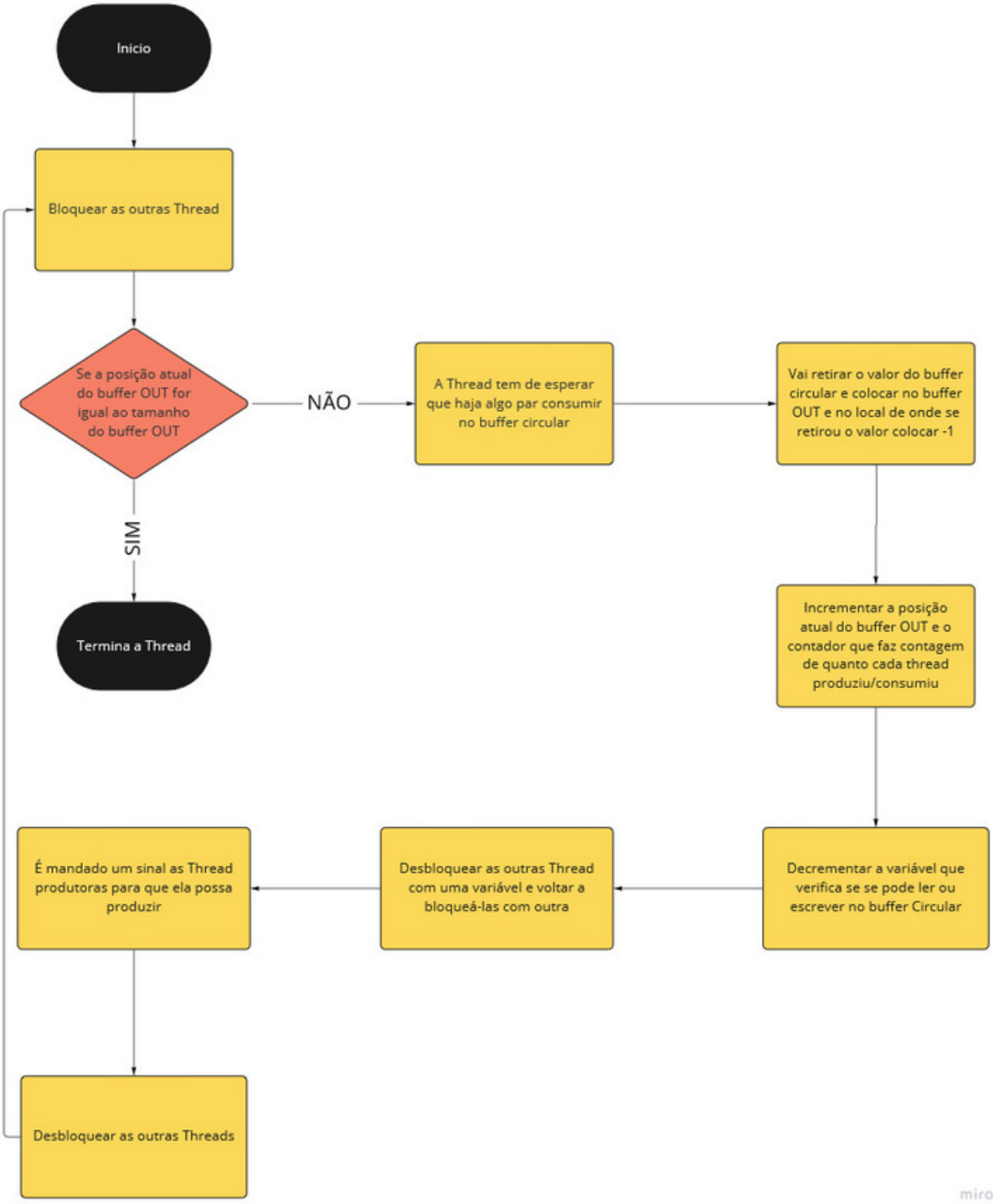
Fluxograma

Produtora



Fluxograma

Consumidora



Código Comentado

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

//definir o tamanho dos vários buffers
#define tamIN 200
#define tamCirc 50
#define tamOUT 198

//declaração dos buffers
int bfIN[tamIN];
int bfCirc[tamCirc];
int bfOUT[tamOUT];

//declaração de mutex para bloquear as Threads
pthread_mutex_t mutexProd = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutexCons = PTHREAD_MUTEX_INITIALIZER;

//condição utilizada para verificar se as produtoras podem continuar a produzir
pthread_cond_t empty = PTHREAD_COND_INITIALIZER;
//condição utilizada para verificar se a consumidora pode continuar a produzir
pthread_cond_t full = PTHREAD_COND_INITIALIZER;

//posição do buffer IN partir do qual se realiza o cálculo da media
int posbfIN;
//posição do buffer OUT onde se vai inserir o próximo elemento do buffer circular
int posbfOut;

//indica se o buffer circular esta pronto para que uma thread produtora possa colocar valores
//e indica se o buffer circular esta pronto para que a thread consumidora possa consumir os seus valores
int ready;

void* producer(void*), * consumer(void*);


int main() {

    //ciclo para colocar os valores no buffer IN
    for (int i = 0, j = 1; i < tamIN; i++, j++) {
        bfIN[i] = j;
    }

    //contador para apresentar quantas vezes as threads produziram
    //e consumiram elementos
    int countP1 = 0;
    int countP2 = 0;
    int countC = 0;

    //declaração das threads
    pthread_t PM_T1, PM_T2, CM_T;

    //inicializção da Threads
    pthread_create(&PM_T1, NULL, producer, &countP1);
    pthread_create(&PM_T2, NULL, producer, &countP2);
    pthread_create(&CM_T, NULL, consumer, &countC);
```

```
    //espera que as threads terminem de correr
    pthread_join(PM_T1, NULL);
    pthread_join(PM_T2, NULL);
    pthread_join(CM_T, NULL);

    //ciclo para imprimir na consola o buffer OUT
    printf("Buffer OUT : ");
    for (int i = 0; i < tamOUT; i++) {
        printf("%d |", bfOUT[i]);
    }
    printf("\n\n");

    //ciclo para imprimir na consola o buffer Circular
    printf("Buffer CIRC : ");
    for (int i = 0; i < tamCirc; i++) {
        printf("%d |", bfCirc[i]);
    }
    printf("\n\n");

    //imprimir na consola o que cada threads consumiu/produziu
    printf("\nProdutora 1: %d ", countP1);
    printf("\nProdutora 2:  %d ", countP2);
    printf("\nConsumidora: %d \n", countC);

    return 0;
}
void* producer(void* count) {

    while (1) {
        //faz se um bloqueio na Thread
        pthread_mutex_lock(&mutexProd);
        //verifica se a Thread termina
        if (posbfIN >= tamIN - 2) {
            pthread_mutex_unlock(&mutexProd);
            break;
        }
        //ciclo para adormecer a Thread produtora enquanto a Thread consumidora não consome algo
        while (ready >= tamCirc - 1) {
            //colocar em espera threads produtoras enquanto o bfCirc tiver cheio
            pthread_cond_wait(&empty, &mutexProd);
        }
        //colocar no bufCirc a média de 3 posiçoes consecutivas do bufIN
        bfCirc[posbfIN % tamCirc] = (bfIN[posbfIN] + bfIN[posbfIN + 1] + bfIN[posbfIN + 2]) / 3;
        //incrementar a psoiçao do bufIN
        posbfIN++;
        //incrementa o contador
        *((int*)count) += 1;
        //faz se o desbloqueio da Thread
        pthread_mutex_unlock(&mutexProd);

        //faz se um bloqueio na Thread
        pthread_mutex_lock(&mutexCons);
        //incrementa a variavel ready
        ready++;
        //acorda a Thread Consumidora quando estiver algo para esta consumir no bfCirc
        pthread_cond_signal(&full);
        //faz se o desbloqueio da Thread
        pthread_mutex_unlock(&mutexCons);

    }

}
```

```c
void* consumer(void* count) {

    while (1) {
        //faz se um bloqueio na Thread
        pthread_mutex_lock(&mutexCons);
        //verifica se a Thread termina
        if (posbfOut == tamOUT) {
            pthread_mutex_unlock(&mutexCons);
            break;
        }
        //ciclo para adormecer a Thread consumidora enquanto uma das Thread produtoras não produz algo
        while (ready == 0) {
            pthread_cond_wait(&full, &mutexCons);
        }
        //retirar o valor do bfCirc e colocar no bfOUT
        bfOUT[posbfOut] = bfCirc[posbfOut % tamCirc];
        //colocar o valor -1 na posiçao de onde se retirou o valor do bfCirc
        bfCirc[posbfOut % tamCirc] = -1;
        //incrementar índice do bfOUT
        posbfOut++;
        //incrementa o contador
        *((int*)count) += 1;
        //decrementar a variável ready
        ready--;
        //faz se o desbloqueio da Thread
        pthread_mutex_unlock(&mutexCons);

        //faz se um bloqueio na Thread
        pthread_mutex_lock(&mutexProd);

        //acorda uma Thread Produtora quando o bfCirc tiver pelo menos uma posição já consumida
        pthread_cond_signal(&empty);
        //faz se o desbloqueio da Thread
        pthread_mutex_unlock(&mutexProd);

    }

}
```

Descrição resumida do código

No código começamos por adicionar as bibliotecas necessárias para a realização do trabalho, entre elas temos a biblioteca <pthread.h>, a mais importante para utilizarmos threads e conseguirmos sincroniza-las. De seguida definimos os tamanhos do buffers, tanto do buffer in, como do buffer out e do buffer circular, e declaramos os arrays que irão servir de buffer com os tamanhos previamente indicados. Também declarámos duas variáveis do tipo pthread mutex, que irão servir para fazer pthread lock e unlock, duas variáveis do tipo pthread cond que irão ser utilizadas para fazer Condition Variables, e ainda duas variáveis que irão conter respetivamente o valor do índice do buffer in, que irá ser utilizado para tirar os valor do buffer in para fazer as médias, e o valor da posição do buffer out, que irá ser utilizado para definir o índice onde se coloca o valor da média. Ainda temos uma variável do tipo inteiro que vai conter a quantidade de posições com valores não lidos no buffer circular. Ainda antes do main temos a pré-declaração das funções producer e consumer que serão posteriormente criadas.

No Main começamos por encher o buffer circular com números de 1 até 20000 (tamanho do buffer in), declaramos os contadores utilizados para contar o que cada thread produziu\consumiu, e as threads utilizadas no trabalho. De seguida criamos as threads utilizadas, com o respetivo ponteiro para a variável que representa a thread, com a função a ser executada pela thread e passa-se como argumento para a thread um ponteiro para o respetivo contador. Seguidamente fazemos pthread_join para cada thread, ou seja, vai-se esperar que as threads acabem de correr. Para finalizar o Main, vai ser impresso na consola o buffer out, o buffer circular e os contadores, indicado o que cada thread produziu\consumiu.

Na função producers (função que as threads produtoras vão correr), temos um ciclo infinito que é quebrado quando a posição atual do buffer in é maior ou igual ao tamanho do buffer in menos 2, isto verifica-se utilizando a condição "posbfIN >= tamIN - 2", sendo o posbfIN o valor atual do índice do buffer in que vai ser incrementado conforme se vai percorrendo esse buffer. Dentro deste ciclo a thread produtora fica adormecida até que haja algum espaço no buffer circular onde possa introduzir o valor que irá calcular, quando o buffer circular tem algum índice onde possa escrever, ele irá lá colocar o valor da média de três valores consecutivos do buffer in, e incrementa a variável que contem o valor do índice atual do buffer in e o contador da thread. Seguidamente noutro bloco crítico é incrementada a variável ready, atualizando assim o número de posições ocupadas no buffer circular, e para finalizar a função producer é mandado um signal para a thread consumidora para estas acordarem e poder consumir os valores presentes no buffer circular.

Na função consumer (função que as threads consumidores vão correr), temos um ciclo infinito que é quebrado quando o buffer out fica totalmente preenchido, isto verifica-se utilizando a condição "posbfOut == tamOUT", sendo o posbfOut o valor atual do índice do buffer out que vai ser incrementado conforme se vai percorrendo esse buffer. Dentro deste ciclo a thread consumidora fica adormecida quando o buffer circular está vazio. Seguidamente retira o valor do buffer circular para o buffer out e substitui no buffer circular o valor que retirou por -1 incrementado a variável posbfOut que indicia o índice do buffer out e o contador da thread. Seguidamente é decrementada a variável ready, atualizando assim o número de posições ocupadas no buffer circular, e para finalizar a função consumer é mandado um signal para as threads produtoras para estas acordarem e produzirem mais.

Conclusão

Através dos conhecimentos adquirimos na unidade curricular de Sistemas Operativos foi possível resolver o problema apresentado com o objetivo principal de dominar a sincronização de Threads.

Com a análise minuciosa dos exemplos disponibilizados pelo professor da unidade curricular e com alguma pesquisa feitas, foi possível dominar melhor esta área de estudo.

Inicialmente resolvemos o problema de sincronização com recurso à biblioteca <semaphore.h> mas posteriormente, para cumprir com todos os objetivos deste trabalho prático, fizemos alterações para utilizar apenas a biblioteca <pthread.h> para sincronização de threads através dos mecanismos *Mutexes* e *Condition Variables.*