GUILHERME CREMASCO GULMINI

nUSP: 11816077

RELATÓRIO – PROBLEMA DO JANTAR DOS FILÓSOFOS

SSC0541 – SISTEMAS OPERACIONAIS I

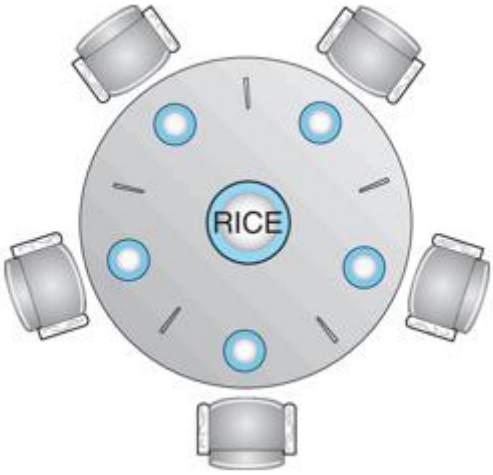
DOCENTE: VANDERLEI BONATO

SÃO CARLOS, SP

2021

Índices

1. Introdução ao Problema



Cinco filósofos estão sentados em uma mesa com cinco tigelas com arroz, uma para cada filósofo. Há também cinco hashis na mesa, sendo que para comer, um filósofo precisa usar dois deles.

Os filósofos ficam alternando entre pensar e comer. No entanto, não há hashis suficientes para que todos comam ao mesmo tempo. Sendo assim, deve-se criar um programa, usando os conceitos de threads e escalonamento, que ajude os filósofos a estabelecerem uma ordem das ações, evitando deadlocks e inanição (starvation).

2. Lógica do Código

A estrutura do monitor usado no código se resume basicamente ao uso de uma struct que possui todas as variáveis de controle. Dentre elas, seis semáforos: um mutex que limita somente um filósofo a executar uma ação, seja ela soltar ou pegar os hashis, e outros cinco semáforos, um para cada filósofo, que impede que outro filósofo pegue os mesmos hashis que um filósofo estiver usando. Além dos semáforos, a struct possui uma variável inteira que representa o que cada filósofo está fazendo e um vetor auxiliar que enumera cada filósofo.

```c
struct monitor {
    enum {PENSATIVO, FAMINTO, COMENDO} estado[5];
    sem_t mutex;
    sem_t semFilosofo[5];
    int nFilosofo[5];
};

#define ESQUERDA ((nFilosofo + 4) % 5)
#define DIREITA ((nFilosofo + 1) % 5)

struct monitor MONITOR;
pthread_t threadFilosofo[5];
```

Uma variável global dessa struct representa o monitor usado no código. O monitor é inicializado na função `inicializarMonitor()`, que cria e inicializa os semáforos da struct e verifica se foram inicializados corretamente. Depois, são inicializadas as threads que representam cada filósofo. Essas threads também são variáveis globais, mas só são inicializadas na função `inicializarThreads()`.

```c
int inicializarMonitor() {
    int check = 1;
    for(int i = 0; i < 5; i++) {
        MONITOR.nFilosofo[i] = i;
    }

    if(sem_init(&MONITOR.mutex, 0, 1) == 0) {
        for(int i = 0; i < 5; i++) {
            sem_init(&MONITOR.semFilosofo[i], 0, 0);
        }
        check = 0;
    } else {
        printf("Erro ao inicializar os semaforos\n");
    }

    return check;
}

void inicializarThreads() {
    for(int i = 0; i < 5; i++) {
        if(pthread_create(&threadFilosofo[i], NULL, filosofo, &MONITOR.nFilosofo[i])) {
            printf("Falha ao criar a thread %d\n", i+1);
        }
        else {
            printf("Filosofo %d esta filosofando\n", i+1);
        }
    }
}
```

Todas as threads iniciam no modo “Pensativo”, ou seja, todos os filósofos ainda não estão comendo, até que uma das threads irá disparar a função `pegaHashi(int nFilosofo)`. Nesse momento, o mutex principal e o semáforo da thread que disparou a função recebe sinal de `wait` e mais nenhuma thread irá chamar o método de pegar os mesmos hashis que a primeira thread pegou. Como são cinco hashis no total, um outro par de hashis ainda estará disponível e será pego por outro filósofo, do mesmo modo.

```c
void pegaHashi(int nFilosofo) {
    sem_wait(&MONITOR.mutex);
    MONITOR.estado[nFilosofo] = FAMINTO;
    printf("Filosofo %d esta faminto\n", nFilosofo+1);
    teste(nFilosofo);
    sem_post(&MONITOR.mutex);
    sem_wait(&MONITOR.semFilosofo[nFilosofo]);
    pausarExecucao();
}
```

Quando a segunda thread “pegar os hashis”, haverá somente um hashi disponível e, assim, as outras três threads irão aguardar até que um par seja devolvido à mesa. Essa condição é garantida pela função `teste(int nFilosofo)`, que faz uma verificação e analisa se a thread da “esquerda” `((nFilosofo + 4) % 5)` e a thread da “direita” `((nFilosofo + 1) % 5)` de uma dada thread estão “comendo”, através da variável inteira dos estados.

```c
void teste(int nFilosofo) {
    if(MONITOR.estado[ESQUERDA] != COMENDO && MONITOR.estado[nFilosofo] == FAMINTO && MONITOR.estado[DIREITA] != COMENDO) {
        MONITOR.estado[nFilosofo] = COMENDO;
        pausarExecucao();
        printf("Filosofo %d pegou os hashis %d e %d\n", nFilosofo+1, ESQUERDA+1, nFilosofo+1);
        printf("Filosofo %d esta comendo\n", nFilosofo+1);
        sem_post(&MONITOR.semFilosofo[nFilosofo]);
    }
}
```

Assim que a thread de um dos filósofos terminar de comer, tempo esse que dura um segundo, a thread irá chamar a função `largaHashi(int nFilosofo)`. Novamente, o mutex principal recebe sinal de `wait` e o estado do filósofo será definido como “pensativo”, indicando que ele largou de fato os hashis. Logo após, o programa vai verificar se os filósofos da esquerda e da direita podem pegar os hashis. Se sim, o mutex será desbloqueado e um filósofo que estava esperando poderá pegar os hashis.

```c
void largaHashi(int nFilosofo) {
    sem_wait(&MONITOR.mutex);
    MONITOR.estado[nFilosofo] = PENSATIVO;
    printf("Filosofo %d largou os hashis %d e %d\n", nFilosofo+1, ESQUERDA+1, nFilosofo+1);
    printf("Filosofo %d voltou a filosofar, satisfeito e feliz :)\n", nFilosofo+1);
    teste(ESQUERDA);
    teste(DIREITA);
    sem_post(&MONITOR.mutex);
}
```

Quando todos os filósofos terminarem de comer, as threads e os semáforos serão finalizados, e o programa se encerra, sem nenhum deadlock ou starvation.

```
91  void finalizarMonitor() {
92      sem_destroy(&MONITOR.mutex);
93      for(int i = 0; i < 5; i++) {
94          sem_destroy(&MONITOR.semFilosofo[i]);
95      }
96  }
97
98  void finalizarThreads() {
99      for(int i = 0; i < 5; i++) {
100         pthread_join(threadFilosofo[i], NULL);
101     }
102 }
103
```

3. Resultados

Seguem algumas saídas do programa:

*Sem título - Bloco de Notas

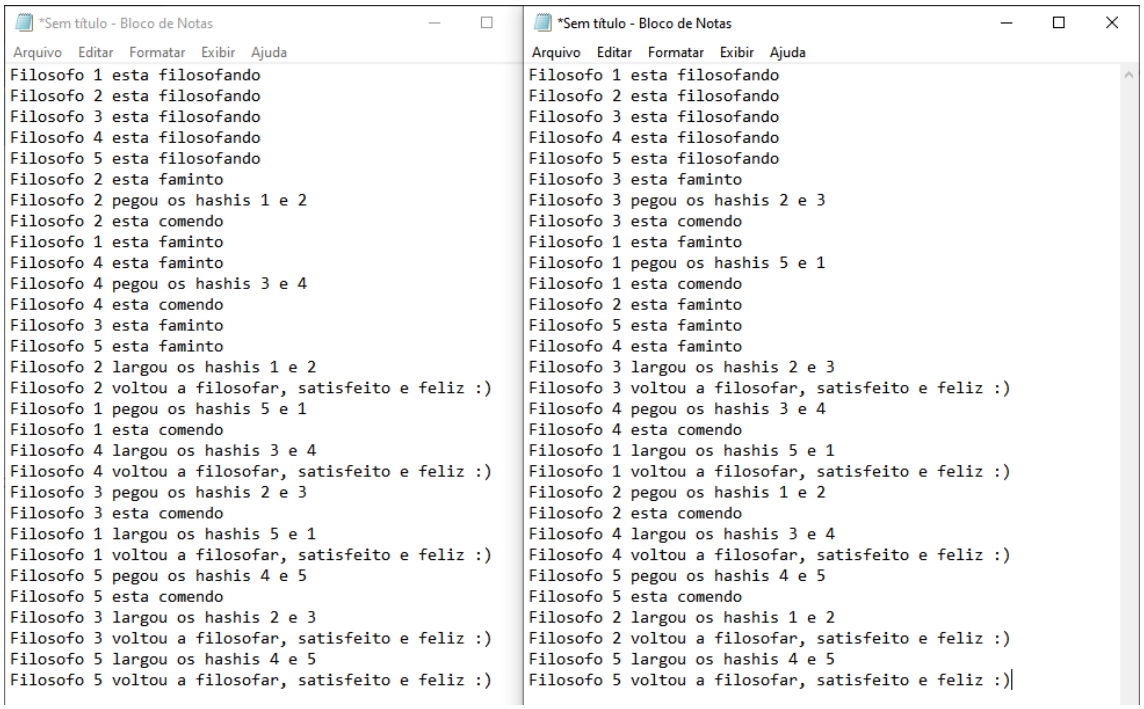
Arquivo Editar Formatar Exibir Ajuda

```
Filosofo 1 esta filosofando
Filosofo 2 esta filosofando
Filosofo 3 esta filosofando
Filosofo 4 esta filosofando
Filosofo 5 esta filosofando
Filosofo 1 esta faminto
Filosofo 1 pegou os hashis 5 e 1
Filosofo 1 esta comendo
Filosofo 2 esta faminto
Filosofo 4 esta faminto
Filosofo 4 pegou os hashis 3 e 4
Filosofo 4 esta comendo
Filosofo 3 esta faminto
Filosofo 5 esta faminto
Filosofo 1 largou os hashis 5 e 1
Filosofo 1 voltou a filosofar, satisfeito e feliz :)
Filosofo 2 pegou os hashis 1 e 2
Filosofo 2 esta comendo
Filosofo 4 largou os hashis 3 e 4
Filosofo 4 voltou a filosofar, satisfeito e feliz :)
Filosofo 5 pegou os hashis 4 e 5
Filosofo 5 esta comendo
Filosofo 2 largou os hashis 1 e 2
Filosofo 2 voltou a filosofar, satisfeito e feliz :)
Filosofo 3 pegou os hashis 2 e 3
Filosofo 3 esta comendo
Filosofo 5 largou os hashis 4 e 5
Filosofo 5 voltou a filosofar, satisfeito e feliz :)
Filosofo 3 largou os hashis 2 e 3
Filosofo 3 voltou a filosofar, satisfeito e feliz :)
```

*Sem título - Bloco de Notas

Arquivo Editar Formatar Exibir Ajuda

```
Filosofo 1 esta filosofando
Filosofo 2 esta filosofando
Filosofo 3 esta filosofando
Filosofo 4 esta filosofando
Filosofo 5 esta filosofando
Filosofo 1 esta faminto
Filosofo 1 pegou os hashis 5 e 1
Filosofo 1 esta comendo
Filosofo 2 esta faminto
Filosofo 3 esta faminto
Filosofo 3 pegou os hashis 2 e 3
Filosofo 3 esta comendo
Filosofo 4 esta faminto
Filosofo 5 esta faminto
Filosofo 1 largou os hashis 5 e 1
Filosofo 1 voltou a filosofar, satisfeito e feliz :)
Filosofo 5 pegou os hashis 4 e 5
Filosofo 5 esta comendo
Filosofo 3 largou os hashis 2 e 3
Filosofo 3 voltou a filosofar, satisfeito e feliz :)
Filosofo 2 pegou os hashis 1 e 2
Filosofo 2 esta comendo
Filosofo 5 largou os hashis 4 e 5
Filosofo 5 voltou a filosofar, satisfeito e feliz :)
Filosofo 4 pegou os hashis 3 e 4
Filosofo 4 esta comendo
Filosofo 2 largou os hashis 1 e 2
Filosofo 2 voltou a filosofar, satisfeito e feliz :)
Filosofo 4 largou os hashis 3 e 4
Filosofo 4 voltou a filosofar, satisfeito e feliz :)
```



4. Problemas Encontrados

O principal problema que tive durante o desenvolvimento do trabalho foi conseguir configurar os semáforos de modo que não houvesse nenhuma race condition. Havia tentado só com o mutex principal e com outras soluções, mas sempre acontecia algum erro. Com o vetor de semáforos para cada thread, o problema foi resolvido.

Outro impasse que teve durante o desenvolvimento foi organizar o monitor e suas variáveis auxiliares, mas esse foi um problema menor.

De resto, correu tudo bem. Com os slides da aula, além da aula de reforço de quinta-feira e alguns materiais auxiliares, creio que consegui atender a todos os requisitos no código-fonte.

5. Orientações Para Execução

- Instalar a biblioteca “pthread.h” instalada para a execução do código.
    - `sudo apt install libpthread-stubs0-dev`
- Compilar o código “JantarDosFilosofosMonitor.c”.
    - `gcc -pthread JantarDosFilosofosMonitor.c`

- Executar o programa.
    - `./a.out`