Escola de Artes, Ciências e Humanidades
da Universidade de São Paulo

EDGAR HENRIQUE DE OLIVEIRA LIRA (12717266)
PEDRO AUGUSTO DE MELO DELAMURA SOARES (12542800)

SISTEMAS OPERACIONAIS

EP3

SÃO PAULO

2022

EDGAR HENRIQUE DE OLIVEIRA LIRA (12717266)
PEDRO AUGUSTO DE MELO DELAMURA SOARES (12542800)

SISTEMAS OPERACIONAIS
EP3

Trabalho apresentado à Escola de Artes, Ciências e Humanidades da Universidade de São Paulo como requisito da disciplina de Sistemas Operacionais.

Prof. Dr.ª Gisele da Silva Craveiro.

SÃO PAULO
2022

SUMÁRIO

1 CÓDIGO

1.1 Código

```
#include <stdio.h> // Printf
#include <pthread.h> // Threads
#include <semaphore.h> // Semáforo
#include <unistd.h>  // Sleep
#include <stdlib.h> // Função Random

#define true 1
#define false 0
#define MAX 5

typedef int bool;

sem_t s;
sem_t in, ex;
sem_t turno;

typedef struct{
   int A[MAX];
   int inicio;
   int nroElem;
}FILA;

FILA f;

void inicializarFila(FILA *f){
   f->inicio = 0;
   f->nroElem = 0;
}

void inserir(FILA *f, int id){
   sem_wait(&in);
   if(f->nroElem==MAX){
   sem_post(&in);
   return ;
   }

   f->A[(f->inicio+f->nroElem)%MAX] = id;
   f->nroElem++;

   sem_post(&in);
```

```
}

void excluir(FILA *f){
   sem_wait(&ex);
   if(f->nroElem == 0){
       sem_post(&ex);
       return ;
   }
   f->inicio = (f->inicio +1) % MAX;
   f->nroElem--;
   sem_post(&ex);

}

int vez(FILA *f){
   return f->A[f->inicio];
}

void* Thread(void * arg) {
int id = *(int *) arg;

int v;
while(1){

   inserir(&f, id);
   while(true){
       sem_wait(&turno);
       v = vez(&f);
       sem_post(&turno);
       if (id == vez(&f))
           break;
       sleep(0.2);
   }
   sem_wait(&s);
   printf("Thread %d Passou pelo semaforo e está na seção crítica\n",
id);
   sleep(1);
   printf("Thread %d esta voltando para fila...\n", id);
   excluir(&f);
   sem_post(&s);
   sleep(0.3);
}
}
```

```
int main() {

inicializarFila(&f);
pthread_t threads[3];
int idThread[3];
sem_init(&s, 0, 1);
sem_init(&ex, 0, 1);
sem_init(&in, 0, 1);
sem_init(&turno, 0, 1);

printf("------Problema da seção crítica com semáforo------\n");
printf("\n   ------Iniciando Threads e semáforo------\n\n");
for (int i = 0; i<3; i++){
  idThread[i] = i;
  pthread_create(&threads[i],NULL,Thread, (void*)&idThread[i]);
}
for (int i = 0; i<3; i++){
pthread_join(threads[i],NULL);
}

 return 0;
}
```

1.2 Print da saída, como o código foi compilado e executado.

```
[edgarlira@Arch EP3]$ gcc semaforo.c -o semaforo -lpthread
[edgarlira@Arch EP3]$ ./semaforo
------Problema da seção crítica com semáforo------

   ------Iniciando Threads e semáforo------

Thread 0 Passou pelo semaforo e está na seção crítica
Thread 0 esta voltando para fila...
Thread 1 Passou pelo semaforo e está na seção crítica
Thread 1 esta voltando para fila...
Thread 2 Passou pelo semaforo e está na seção crítica
Thread 2 esta voltando para fila...
Thread 0 Passou pelo semaforo e está na seção crítica
Thread 0 esta voltando para fila...
Thread 1 Passou pelo semaforo e está na seção crítica
Thread 1 esta voltando para fila...
Thread 2 Passou pelo semaforo e está na seção crítica
Thread 2 esta voltando para fila...
Thread 0 Passou pelo semaforo e está na seção crítica
Thread 0 esta voltando para fila...
Thread 1 Passou pelo semaforo e está na seção crítica
Thread 1 esta voltando para fila...
Thread 2 Passou pelo semaforo e está na seção crítica
Thread 2 esta voltando para fila...
Thread 0 Passou pelo semaforo e está na seção crítica
Thread 0 esta voltando para fila...
Thread 1 Passou pelo semaforo e está na seção crítica
Thread 1 esta voltando para fila...
Thread 2 Passou pelo semaforo e está na seção crítica
Thread 2 esta voltando para fila...
Thread 0 Passou pelo semaforo e está na seção crítica
Thread 0 esta voltando para fila...
Thread 1 Passou pelo semaforo e está na seção crítica
Thread 1 esta voltando para fila...
Thread 2 Passou pelo semaforo e está na seção crítica
Thread 2 esta voltando para fila...
Thread 0 Passou pelo semaforo e está na seção crítica
Thread 0 esta voltando para fila...
```

1. 3 Ambientes utilizados

Ambiente usado para programar: Vscode
Compilação: gcc semaforo.c -o semaforo -lpthread
Execução: ./semaforo
Ambiente usado para compilar e executar: gnome-terminal

2 Seção Crítica

- Exclusão mútua:

A exclusão mútua é atendida, pois somente uma das threads entra na seção crítica por vez. Isso ocorre porque dentro da função *void* Thread* ocorre o fechamento e abertura do semáforo da seção crítica com *sem_wait* e *sem_post*, respectivamente, permitindo assim que somente uma thread execute a seção crítica por vez, e neste código, após a thread sair da seção crítica e liberar o semáforo, ela retorna para o fim da fila, após a liberação do semáforo, outra thread entra na seção, prosseguindo da mesma maneira.

- Progresso:

O progresso é mantido, pois as threads se inserem em uma fila, passando como parâmetro seus id's, após isso o código verifica qual thread é a primeira da fila. A thread que estiver em primeiro na fila sai do loop, podendo verificar se é possível passar pelo semáforo. Caso o semáforo esteja aberto, a thread entra na seção crítica e fecha o semáforo, caso o semáforo esteja fechado, significa que alguma thread já está na seção crítica, então, ele irá esperar a thread terminar a execução da seção crítica, e assim a seção crítica nunca ficará ociosa quando requisitada. Além disso, as threads sempre liberam a seção crítica após a execução dela, garantindo que a próxima thread que está aguardando na fila consiga entrar. Portanto o progresso é mantido.

- Espera Limitada:

Já a propriedade da espera limitada é garantida devido a existência de uma fila e do semáforo binário. Como já foi dito, inicialmente qualquer thread pode executar sua seção crítica, se suprir as condições de estar na sua vez de entrar na seção crítica e se o semáforo liberar a passagem da thread, sendo que, após a execução da sua seção crítica, a thread será colocada em uma fila para esperar novamente pela sua vez de executar sua seção crítica. Como a inserção em uma fila é sempre feita na última posição e a remoção é sempre feita na primeira posição, assim que uma thread termine de executar sua seção crítica, ela irá para o final da fila e irá esperar exatamente uma execução de seção crítica de cada uma das threads que estiverem na sua frente, garantindo assim o cumprimento da propriedade da espera limitada.

Dessa maneira, a **exclusão mútua** é mantida, já que somente uma thread pode executar sua seção crítica por vez. A **espera limitada** está garantida, observando que cada thread não irá esperar mais do que apenas uma execução de cada uma das outras em sua frente na fila. Assim como o **progresso** também é garantido, pois sempre teremos uma thread executando sua seção crítica. Portanto, atendendo-se às três condições previamente citadas, é possível concluir que o algoritmo é uma solução válida para o problema da seção crítica.

3 Suposições

Uma primeira suposição que pode ser feita a respeito dessa implementação é que a implementação dessa fila está limitada para no máximo 5 threads. Para conseguirmos garantir a espera limitada, entretanto, podemos aumentá-la se alterarmos o código.

Uma outra suposição a ser feita é a de que, para escalas maiores e problemas mais complexos, talvez um semáforo de contagem seria uma alternativa melhor do que o semáforo binário, já que o semáforo de contagem permitiria que o poder de processamento da máquina em questão fosse muito mais explorado, isto é, caso tal potencial exista, permitindo assim uma eficiência muito maior.