Jantar dos Filósofos

Axell Brendow Batista Moreira

[1] Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica de Minas Gerais (PUC-MG)
Caixa Postal 1.686 – 30.535-901 – Belo Horizonte – MG – Brazil

Considere 5 filósofos que passam a vida a pensar e a comer. Partilham uma mesa redonda rodeada por 5 cadeiras sendo que cada uma das cadeiras pertence a um filósofo. No centro da mesa encontra-se uma taça de arroz e estão 5 garfos na mesa, um para cada filósofo.

Quando um filósofo pensa não interage com os seus colegas. De tempos em tempos, cada filósofo fica com fome e tenta apanhar os dois garfos que estão mais próximos (os garfos que estão ou à esquerda ou à direita). O filósofo apenas pode apanhar um garfo de cada vez e como o leitor compreende, não pode apanhar um garfo se este estiver na mão do vizinho. Quando um filósofo esfomeado tiver 2 garfos ao mesmo tempo ele come sem largar os garfos. Apenas quando acaba de comer, o filósofo pousa os garfos, libertando-os e começa a pensar de novo. O nosso objetivo é ter uma representação/implementação que nos permita simular este jantar sem que haja problemas de deadlock ou starvation.

Para isso, o jantar será modelado usando uma thread para representar cada filósofo e usaremos semáforos para representar cada garfo. Quando um filósofo tenta agarrar um garfo executa uma operação wait no semáforo, quando o filósofo larga o garfo executa uma operação signal nesse mesmo semáforo. Cada filósofo (thread) vai seguir o algoritmo, ou seja, todos fazem as mesmas ações. Como deve estar já o leitor a pensar, o facto de seguirem o mesmo algoritmo pode dar azo à situação de deadlock, dai a utilização das primitivas de sincronização wait e signal. Uma outra possibilidade de deadlock seria o facto de mais do que um filósofo ficar com fome ao mesmo tempo, os filósofos famintos tentariam agarrar os garfos ao mesmo tempo. Isto é outro ponto que uma solução satisfatória terá que ter em atenção, devendo ser salvaguardado o facto de um filósofo não morrer à fome. Devemos recordar o leitor que uma solução livre de deadlock não elimina necessariamente a possibilidade de um filósofo morrer esfomeado.

Por todos estes motivos, o jantar dos filósofos é um algoritmo que deve ser implementado com algum cuidado por parte do programador.

Jantar dos Filósofos – Implementação em Java

```
class Mesa
{
  final static int PENSANDO = 1;
  final static int COMENDO = 2;
  final static int FOME = 3;
  final static int NR_FILOSOFOS = 5;
  final static int PRIMEIRO_FILOSOFO = 0;
```

```
final static int ULTIMO_FILOSOFO = NR_FILOSOFOS - 1;
boolean[] garfos = new boolean[NR_FILOSOFOS];
int[] filosofos = new int[NR_FILOSOFOS];
int[] tentativas = new int[NR_FILOSOFOS];

public Mesa()
{
  for (int i = 0; i < 5; i++)
  {
    garfos[i] = true;
    filosofos[i] = PENSANDO;
    tentativas[i] = 0;
  }
}

public synchronized void pegarGarfos (int filosofo)
{
  filosofos[filosofo] = FOME;
  while (filosofos[aEsquerda(filosofo)] == COMENDO ||
     filosofos[aDireita(filosofo)] == COMENDO)
  {
    try
    {
      tentativas[filosofo]++;
      wait();
    }
    catch (InterruptedException e)
    {
    }
  }
  //System.out.println("O Filsofo morreu devido a
     starvation");
  tentativas[filosofo] = 0;
  garfos[garfoEsquerdo(filosofo)] = false;
  garfos[garfoDireito(filosofo)] = false;
  filosofos[filosofo] = COMENDO;
  imprimeEstadosFilosofos();
  imprimeGarfos();
  imprimeTentativas();
}

public synchronized void returningGarfos (int filosofo)
{
  garfos[garfoEsquerdo(filosofo)] = true;
  garfos[garfoDireito(filosofo)] = true;
  if (filosofos[aEsquerda(filosofo)] == FOME ||
     filosofos[aDireita(filosofo)] == FOME)
  {
    notifyAll();
```

```
    }
    filosofos[filosofo] = PENSANDO;
    imprimeEstadosFilosofos();
    imprimeGarfos();
    imprimeTentativas();
  }

  public int aDireita (int filosofo)
  {
    int direito;
    if (filosofo == ULTIMO_FILOSOFO)
    {
      direito = PRIMEIRO_FILOSOFO;
    }
    else
    {
      direito = filosofo + 1;
    }
    return direito;
  }

  public int aEsquerda (int filosofo)
  {
    int esquerdo;
    if (filosofo == PRIMEIRO_FILOSOFO)
    {
      esquerdo = ULTIMO_FILOSOFO;
    }
    else
    {
      esquerdo = filosofo - 1;
    }
    return esquerdo;
  }

  public int garfoEsquerdo (int filosofo)
  {
    int garfoEsquerdo = filosofo;
    return garfoEsquerdo;
  }

  public int garfoDireito (int filosofo)
  {
    int garfoDireito;
    if (filosofo == ULTIMO_FILOSOFO)
    {
      garfoDireito = 0;
    }
    else
```

```
    {
      garfoDireito = filosofo + 1;
    }
    return garfoDireito;
  }

  public void imprimeEstadosFilosofos ()
  {
    String texto = "*";
    System.out.print("Filsofos = [ ");
    for (int i = 0; i < NR_FILOSOFOS; i++)
    {
      switch (filosofos[i])
      {
        case PENSANDO :
          texto = "PENSANDO";
          break;
        case FOME :
          texto = "FOME";
          break;
        case COMENDO :
          texto = "COMENDO";
          break;
      }
      System.out.print(texto + " ");
    }
    System.out.println("]");
  }

  public void imprimeGarfos ()
  {
    String garfo = "*";
    System.out.print("Garfos = [ ");
    for (int i = 0; i < NR_FILOSOFOS; i++)
    {
      if (garfos[i])
      {
        garfo = "LIVRE";
      }
      else
      {
        garfo = "OCUPADO";
      }
      System.out.print(garfo + " ");
    }
    System.out.println("]");
  }

  public void imprimeTentativas ()
```

```
  {
    System.out.print("Tentou comer = [ ");
    for (int i = 0; i < NR_FILOSOFOS; i++)
    {
      System.out.print(filosofos[i] + " ");
    }
    System.out.println("]");
  }
}

class Filosofo extends Thread
{
  final static int TEMPO_MAXIMO = 100;
  Mesa mesa;
  int filosofo;

  public Filosofo (String nome, Mesa mesadejantar, int fil)
  {
    super(nome);
    mesa = mesadejantar;
    filosofo = fil;
  }

  public void run ()
  {
    int tempo = 0;
    while (true)
    {
      tempo = (int) (Math.random() * TEMPO_MAXIMO);
      pensar(tempo);
      getGarfos();
      tempo = (int) (Math.random() * TEMPO_MAXIMO);
      comer(tempo);
      returnGarfos();
    }
  }

  public void pensar (int tempo)
  {
    try
    {
      sleep(tempo);
    }
    catch (InterruptedException e)
    {
      System.out.println("O Filfoso pensou em demasia");
    }
  }
```

```
  public void comer (int tempo)
  {
    try
    {
      sleep(tempo);
    }
    catch (InterruptedException e)
    {
      System.out.println("O Filsofo comeu em demasia");
    }
  }

  public void getGarfos()
  {
    mesa.pegarGarfos(filosofo);
  }

  public void returnGarfos()
  {
    mesa.returningGarfos(filosofo);
  }
}

public class Jantar
{
  public static void main (String[] args)
  {
    Mesa mesa = new Mesa ();
    for (int filosofo = 0; filosofo < 5; filosofo++)
    {
      new Filosofo("Filosofo_" + filosofo, mesa,
          filosofo).start();
    }
  }
}
```

Jantar dos Filósofos – Implementação em C

```
#include<stdio.h>
#include<stdlib.h>
#include<semaphore.h>
#include<pthread.h>
#define N 5
#define PENSAR 0
#define FOME 1
#define COMER 2
#define ESQUERDA (nfilosofo+4)%N //agarrar garfo
                          //da esquerda
```

```c
#define DIREITA (nfilosofo+1)%N //agarrar garfo
                          //da direita
void *filosofo(void *num);
void agarraGarfo(int);
void deixaGarfo(int);
void testar(int);

sem_t mutex;
sem_t S[N]; //inicializacao do semforo
int estado[N];
int nfilosofo[N]={0,1,2,3,4};

void *filosofo(void *num)
{
  while(1)
  {
    int *i = num;
    sleep(1);
    agarraGarfo(*i);
    sleep(1);
    deixaGarfo(*i);
  }
}

void agarraGarfo(int nfilosofo)
{
  sem_wait(&mutex);
  estado[nfilosofo] = FOME;
  printf("Filosofo %d tem fome.\n", nfilosofo+1);
  //+1 para imprimir filosofo 1 e nao filosofo 0
  testar(nfilosofo);
  sem_post(&mutex);
  sem_wait(&S[nfilosofo]);
  sleep(1);
}

void deixaGarfo(int nfilosofo)
{
  sem_wait(&mutex);
  estado[nfilosofo]=PENSAR;
  printf("Filosofo %d deixou os garfos %d e %d.\n",
     nfilosofo+1, ESQUERDA+1, nfilosofo+1);
  printf("Filosofo %d esta a pensar.\n", nfilosofo+1);
  testar(ESQUERDA);
  testar(DIREITA);
  sem_post(&mutex);
}

void testar(int nfilosofo)
```

```
{
  if(estado[nfilosofo]==FOME && estado[ESQUERDA]
 !=COMER && estado[DIREITA]!=COMER)
  {
    estado[nfilosofo]=COMER;
    sleep(2);
    printf("Filosofo %d agarrou os garfos %d e %d.\n",
       nfilosofo+1, ESQUERDA+1, nfilosofo+1);
    printf("Filosofo %d esta a comer.\n", nfilosofo+1);
    sem_post(&S[nfilosofo]);
  }
}

int main() {
  int i;
  //identificadores das threads
  pthread_t thread_id[N];
  sem_init(&mutex,0,1);

  for(i=0;i<N;i++)
  {
    sem_init(&S[i],0,0);
  }

  for(i=0;i<N;i++)
  {
    pthread_create(&thread_id[i], NULL, filosofo,
       &nfilosofo[i]);
    //criar as threads
    printf("Filosofo %d esta a pensar.\n",i+1);
  }

  for(i=0;i<N;i++)
  {
    //para fazer a juncao das threads
    pthread_join(thread_id[i],NULL);
  }

  return(0);
}
```