Modelos para Computação Concorrente ou Sistemas Operacionais

Memória Compartilhada – Semáforos – Filósofos

(com slides de Ben-Ari)

Fernando Luís Dotti



PUCRS – Escola Politécnica – Fernando Luís Dotti

Bibliografia Base

[disponível na biblioteca]

M. Ben-Ari

Principles of Concurrent and Distributed Programming

Second Edition

Addison-Wesley, 2006

O Jantar dos Filósofos





O Jantar dos Filósofos

- Metáfora para processos (filósofos) que compartilham recursos (hashis ou garfos) e têm períodos de processamento local (pensar) e períodos em que usam os recursos compartilhados (comer)
- A mesa circular tem N (e.g. 5) filósofos.
 Entre cada filósofo um hashi/garfo.

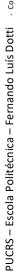
```
    No jantar, cada filósofo:
(loop)
pensa ->
```

fica com fome ->
pega o hashi da direita ->
pega o hashi da esquerda ->

come ->

solta os hashis ->

volta a pensar (end loop)



O Jantar dos Filósofos com Semáforos

- cada garfo é um semáforo
 - semáforo em 1 = garfo livre
 - semáforo em 0 = garfo ocupado
- em uma mesa com 5 filósofos,
 cada filósofo i (0..4) é uma thread (com pseudo-código abaixo), e
 acessa os garfos i e (i+1) MOD 5.
 o filosofo i=4 acessa garfos 4 e 0, fechando a volta na mesa.
- avalie o comportamento do programa
 - houve bloqueio?
 - em que situação ?

Algorithm 6.10: Dining philosophers (first attempt)

semaphore array [0..4] fork $\leftarrow [1,1,1,1,1]$

loop forever

p1: think

p2: wait(fork[i])

p3: wait(fork[i+1])

p4: eat

p5: signal(fork[i])

p6: signal(fork[i+1])

PUCRS – Escola Politécnica – Fernando Luís Dotti · copyright – direitos reservados.

Filósofos – Exemplo em Java

```
/*
Implementação do Jantar dos Filosofos com Semaforo
PUCRS - Escola Politecnica
Prof: Fernando Dotti
import java.util.concurrent.Semaphore;
// ====== Filosofo ========
class Filosofo extends Thread {
    private int i;
    private Semaphore q1, q2;
    private String espaco;
    public Filosofo(int _i, Semaphore _g1,
                            Semaphore q2){
        i = _i; g1 = _g1; g2 = _g2;
       espaco = " ";
        for (int k=0; k<i; k++){</pre>
            espaco = espaco + " ";
    }
    public void run() {
       while (true) {
         // pensa
         System out println(espaco+ i + ": Pensa ");
         // pega um garfo
         try{q1.acquire();
         }catch(InterruptedException ie){}
         System.out.println(espaco+ i + ": Pegou um ");
         // pega outro garfo
         try{q2.acquire();
        }catch(InterruptedException ie){}
         // come, solta garfos
         System.out.println(espaco+ i + ": come ");
         g1.release();
         q2.release();
```

- Exclusao mutua ok
- Starvation ok
- Deadlock ?

- Exclusao mutua ok
- Starvation ok
- Deadlock 🕾
 - O que fazer ?
 - O que é necessário para um deadlock se formar ?

- Deadlock ⊗
 - O que fazer ?
 - O que é necessário para um deadlock se formar ?
 - Hold and wait
 - Não preempção
 - Exclusão mútua
 - Formação de ciclo de espera

- Deadlock 🕾
 - O que fazer ?
 - O que é necessário para um deadlock se formar ?
 - Hold and wait
 - Não preempção
 - Exclusão mútua
 - Formação de ciclo de espera

[Condições de Coffmann]

Combater 1 destas condições é suficiente!

- Deadlock 🕾
 - O que fazer ?
 - O que é necessário para um deadlock se formar ?
 - Hold and wait
 - Não preempção
 - Exclusão mútua
 - Formação de ciclo de espera

Propostas para o problema dos filósofos ?

• Deadlock - 🕾

- O problema dos filósofos:
 - Hold and wait filósofo tenta pegar segundo garfo, se não dá larga o primeiro ?
 - Não preempção um filósofo retira o garfo de outro, se aquele nao estiver comendo e segurando apenas um ?
 - Formação de ciclo de espera não deixar que todos filósofos tenham um garfo na mao e estejam esperando pelo segundo?

Outras hipóteses para não deixar formar deadlock?

Destros e canhotos

- observe a simetria do sistema
- todos filosofos tentam primeiro o da direita
- e se o sistema for assimétrico ?
- Pegar todos os garfos necessários, ou nenhum
 - nao fica em hold-and-wait
 - tenta pegar segundo, se nao der, solta

A mesa só pode ter N-1 filósofos ao simultaneamente

não forma o ciclo

```
Algorithm 6.11: Dining philosophers (second attempt)
                   semaphore array [0..4] fork \leftarrow [1,1,1,1,1]
                   semaphore room \leftarrow 4
     loop forever
       think
p1:
      wait(room)
p2:
      wait(fork[i])
p3:
      wait(fork[i+1])
p4:
       eat
p5:
       signal(fork[i])
p6:
       signal(fork[i+1])
p7:
       signal(room)
:8q
```