Sistemas Operativos

LEI - 2019/2020

:: Problemas clássicos de sincronização ::

Escola Superior de Tecnologia de Setúbal - IPS

Conteúdos

- Problema do produtor-consumidor
- Problema dos leitores-escritores
- Problema do jantar dos filósofos

Problema do Produtor-Consumidor

- Existe um array de tamanho fixo
- O produtor coloca valores no array
- O consumidor remove os valores do array
- O produtor não pode colocar valores se o array estiver cheio
- O consumidor não pode remover valores se o array estiver vazio

```
n = int
mutex = semaphore(1)
empty = semaphore(n)
full = semaphore(0)

Producer:                          Consumer:
    repeat                             repeat
        wait(empty)                        wait(full)
        wait(mutex)                        wait(mutex)
            write(num)                         read(num)
        post(mutex)                        post(mutex)
        post(full)                         post(empty)
    forever                            forever
```

Problema dos Leitores-Escritores

Uma Base de Dados é partilhada entre vários processos concorrentes:

- Leitores: apenas lêem dados
- Escritores: podem ler e escrever dados

Dois processos leitores podem ler ao mesmo tempo, mas um processo escritor deve ter acesso exclusivo à Base de Dados.

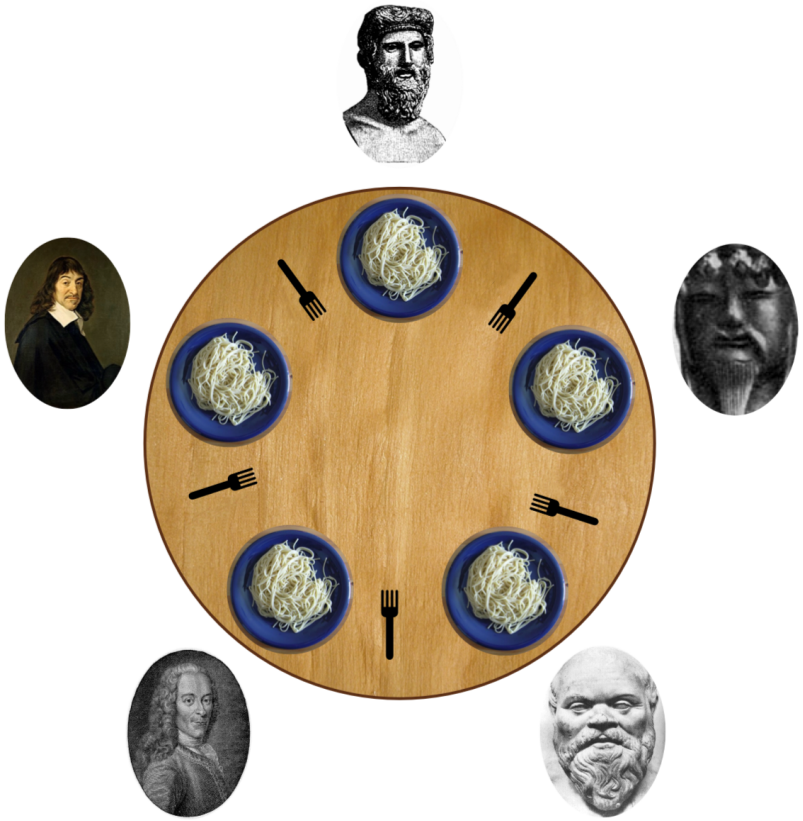
- Os leitores só devem esperar se um escritor já tiver obtido permissão para escrever..

```
mutex = semaphore(1)
rw_mutex = semaphore(1)
read_count = 0

Writer:
    repeat
        wait(rw_mutex)
        ...
        write()
        ...
        post(rw_mutex)
    forever
```

```
Reader:
    repeat
        wait(mutex)
        read_count++
        if (read_count==1)
            wait(rw_mutex)
        post(mutex)
        ...
        read()
        ...
        wait(mutex)
        read_count--
        if (read_count==0)
            post(rw_mutex)
        post(mutex)
    forever
```

Problema do jantar dos filósofos



Problema do jantar dos filósofos (cont.)

- Quando um filósofo tem fome, deve usar os dois garfos
- Quando terminar de comer, deve pousar os dois garfos
- Não pode usar um garfo que já esteja a ser usado

Uma solução...

```
chopstick[5] = semaphore(1)

Philosopher:
    repeat
        wait(chopstick[i])
        wait(chopstick[(i+1) mod 5])
        ...
        eat("n" seconds)
        ...
        post(chopstick[i])
        post(chopstick[(i+1) mod 5])
    forever
```

⚠️ O que acontece se todos os processos obtiverem o garfo *i* ao mesmo tempo?

Outras soluções

- Permitir que apenas 4 filósofos se sentem à mesa
- Permitir que um filósofo pegue nos garfos apenas se os dois estiverem disponíveis
- Os filósofos pares pegam primeiro no garfo esquerdo e só depois no direito, e os ímpares fazem ao contrário

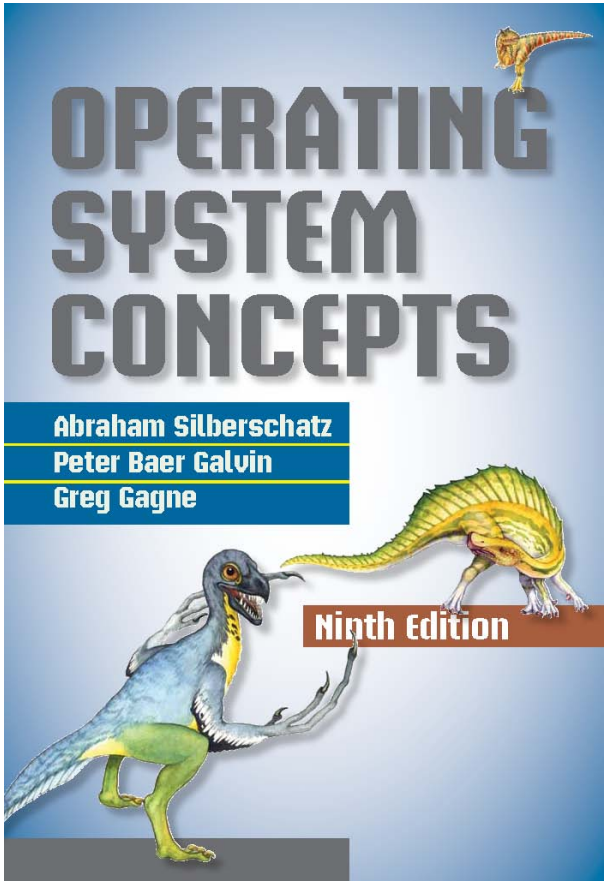
Ex: Disponibilidade dos dois garfos

```
chopstick[5] = semaphore(1)
mutex = semaphore(1)

Philosopher:
    repeat
        wait(mutex)
            wait(chopstick[i])
            wait(chopstick[(i+1) mod 5])
        post(mutex)
        ...
        eat("n" seconds)
        ...
        post(chopstick[i])
        post(chopstick[(i+1) mod 5])
    forever
```

Sumário

- Exemplos clássicos de problemas concorrenciais
- São usados para testar novos mecanismos de sincronização



Ler capítulo 5 (secção 5.7)...