
MAC5753 - Sistemas Operacionais

Daniel Macêdo Batista

IME - USP, 29 de Setembro de 2020

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos

Problemas clássicos – Produtores e consumidores

Problemas clássicos – Filósofos famintos

Problemas
clássicos –
Produtores e
▷ consumidores

Problemas clássicos
– Filósofos famintos

Problemas clássicos – Produtores e consumidores

Produtores e consumidores

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos

- Produtores enviam mensagens a serem consumidas pelos consumidores
- Há um buffer compartilhado manipulado por duas operações: armazena e busca
- Produtores rodam armazena e consumidores rodam busca

Para garantir que mensagens não são sobrescritas e recebidas uma única vez, armazena e busca devem alternar a execução

armazena deve executar primeiro

Produtores e consumidores

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos

- Usar mais uma vez semáforos para sinalizar os eventos
- Esses semáforos podem ser implementados de modo a sinalizar:
 - quando processos alcançarem pontos críticos de execução (Início e finalização das operações armazena e recebe); ou
 - mudanças de variáveis compartilhadas (buffer cheio e buffer vazio)
- A solução apresentada considera o estado do buffer (melhor do que as entradas nas operações quando há múltiplos produtores e consumidores)

Produtores e consumidores

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos

- `empty` e `full` são dois semáforos
 - Inicialmente o buffer está vazio, então `empty = 1`
(Significa que o evento “esvazie o buffer” aconteceu)
 - `full` começa valendo 0

Produtores e consumidores

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos

- Se Produtor quer executar armazena, espera pelo buffer ficar vazio
- Após o Produtor chamar armazena, o buffer fica cheio

- Se Consumidor quer executar recebe, ele espera pelo buffer ficar cheio
- Após o Consumidor chamar recebe, o buffer fica vazio

- Lembrando: com semáforos, um processo espera por um evento rodando P e sinaliza um evento rodando V**

Produtores e consumidores

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos

```
typeT buf;  
sem empty = 1, full = 0;  
  
Thread Producer [i = 1 to M] {  
    while (true) {  
        ...  
        /* produz dados e armazena no buffer */  
        P(empty);  
        buf = data;  
        V(full);  
    }  
}
```

Produtores e consumidores

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos

```
Thread Consumer [j = 1 to N] {  
    while (true) {  
        /* le o buffer e consome o resultado */  
        P(full);  
        result = buf;  
        V(empty);  
        ...  
    }  
}
```

Problemas clássicos
– Produtores e
consumidores

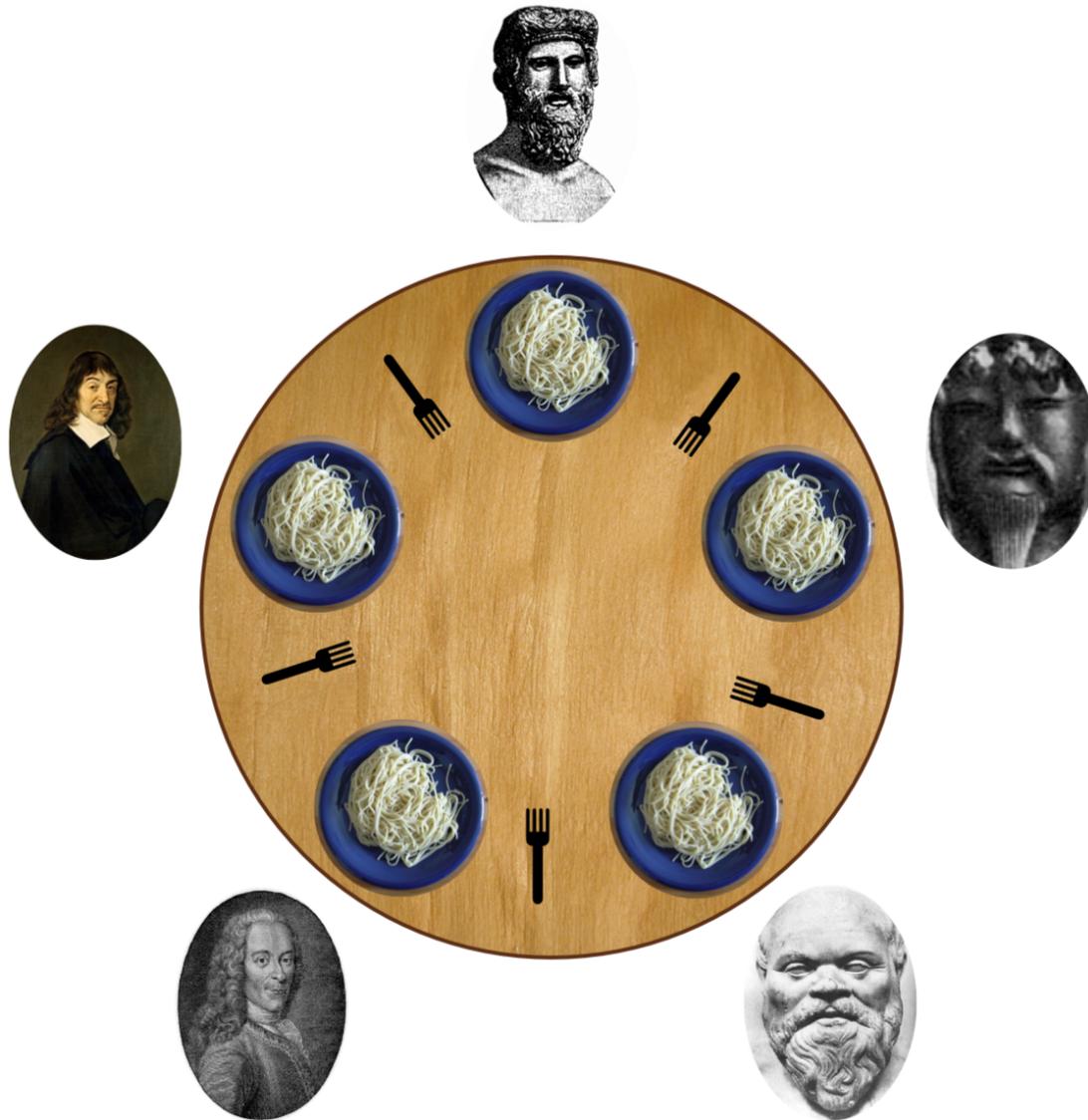
Problemas
clássicos –
Filósofos
▷ famintos

Problemas clássicos – Filósofos famintos

Problema dos filósofos famintos

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos



Descrição

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos

- 5 filósofos
- Comem e pensam
- Cada filósofo tem que usar 2 garfos para comer
- Há apenas 5 garfos
- Cada filósofo só pode usar os garfos imediatamente na sua esquerda e na sua direita

Objetivo

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos

- ❑ O objetivo é implementar um programa que simule o comportamento dos filósofos
- ❑ **O programa deve evitar a situação em que todos os filósofos estejam com fome mas não consigam adquirir ambos os garfos – Por exemplo, cada um segura um garfo e se recusa a liberá-lo**
- ❑ Implementar exclusão mútua entre processos que competem por conjuntos de variáveis compartilhadas que se sobrepõem
- ❑ Útil quando um processo requer acesso simultâneo a mais de um recurso

Mais detalhes do problema

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos

- Algumas características do problema:
 - Dois filósofos vizinhos não podem comer ao mesmo tempo
 - No máximo 2 filósofos estarão comendo ao mesmo tempo
- Algumas considerações do problema:
 - Os períodos pensando e comendo podem variar (aleatório)

Algoritmo dos filósofos

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos

```
Thread Philosopher [i = 0 to 4] {  
    while (true) {  
        pensa;  
        pega os garfos;  
        come;  
        libera os garfos;  
    }  
}
```

Algoritmo dos filósofos

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos

- Cada garfo age como uma trava de seção crítica
 - Só pode estar com um filósofo por vez
 - Os garfos serão um vetor de semáforos inicializados com 1 (inicialmente ninguém segura nenhum garfo)
- Com semáforos
 - Pegar um garfo = executar P
 - Soltar um garfo = executar V

Primeira solução

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos

- Tentar ações idênticas
- Por exemplo, cada filósofo pega (tenta) primeiro o garfo da esquerda e depois o da direita

- Resolve o problema?**

Segunda solução

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos

- Evitar o deadlock que a solução anterior permitia
- Evitar a espera circular (o primeiro espera o segundo, que espera o terceiro, ... que espera o primeiro)

- Ideias?**

Segunda solução

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos

- ❑ Quebrar a espera circular
- ❑ Fazer algum filósofo tentar pegar o garfo da direita primeiro

```
sem fork[5] = {1, 1, 1, 1, 1};
```

```
Thread Philosopher [i = 0 to 3] {  
    while (true) {  
        P(fork[i]);P(fork[i+1]);  
        come;  
        V(fork[i]);V(fork[i+1]);  
        pensa;  
    }  
}
```

Segunda solução

Problemas clássicos
– Produtores e
consumidores

Problemas clássicos
– Filósofos famintos

```
Thread Philosopher [4] {  
    while (true) {  
        P(fork[0]);P(fork[4]);  
        come;  
        V(fork[0]);V(fork[4]);  
        pensa;  
    }  
}
```