

Sistemas Operacionais

Jantar dos Filósofos

Problemas Clássicos

A área de sistemas operacionais apresenta diversos problemas interessantes de sincronização.

Existem alguns problemas clássicos de sincronização.

- Diversos problemas práticos são variações destes clássicos.
- Suas soluções podem ser utilizadas ou estendidas.

Jantar dos Filósofos

Dijkstra, em 1965, formulou e resolveu (utilizando o seu método - semáforos) um dos clássicos problemas de CIP: **O Jantar dos Filósofos**.

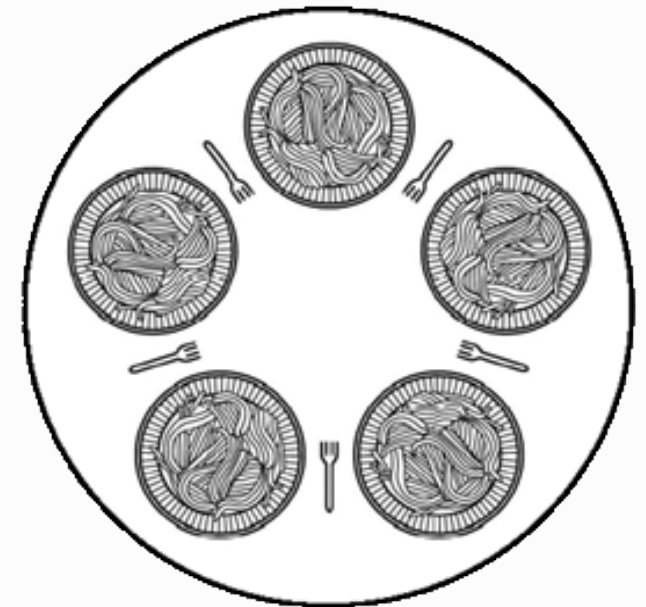
Este problema se tornou tão famoso, que, se você inventar uma nova técnica para CIP, está automaticamente obrigado a demonstrar o quão elegante e eficiente sua técnica resolve o problema do Jantar dos Filósofos.

Serve basicamente para comparação com as outras primitivas já inventadas até o momento (benchmark).

Jantar dos Filósofos

O problema pode ser exposto de uma forma simples:

- Cinco filósofos sentam em uma mesa circular.
 - Cada um com um processamento independente.
- Cada um possui um prato de espaguete.
- O espaguete está escorregadio e cada filósofo precisa de dois garfos para comer
 - Recursos necessários para comer.
- Entre cada par de pratos, existe um garfo.
 - Recursos limitados, dado a quantidade de filósofos.

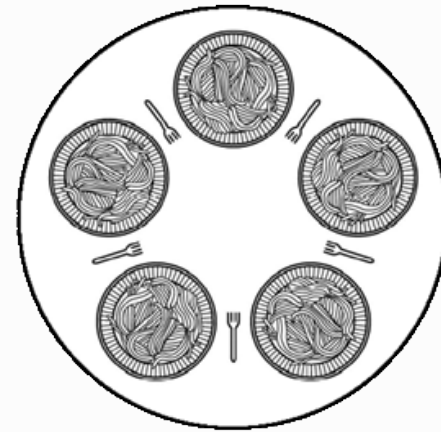
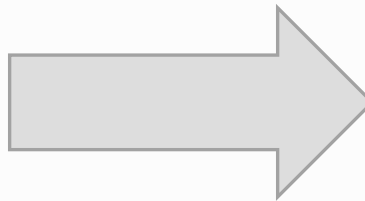


Jantar dos Filósofos

Dinâmica do problema:

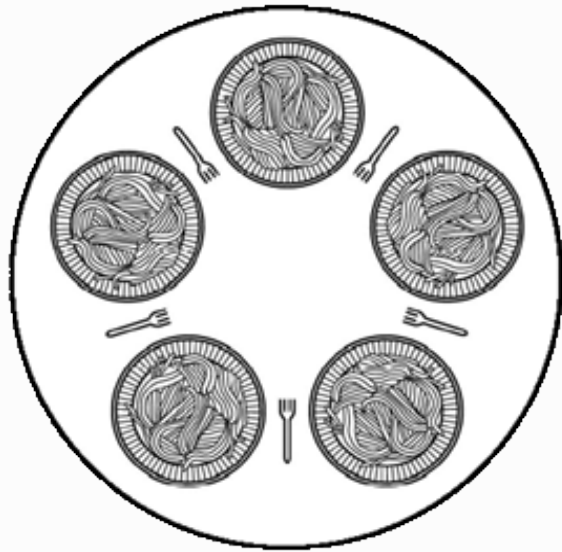
A vida do filósofo se resume a:

- Comer e;
- Pensar.



Nunca comem e pensam ao mesmo tempo.
Cada linha independente, pensa e depois come.

Jantar dos Filósofos



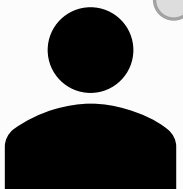
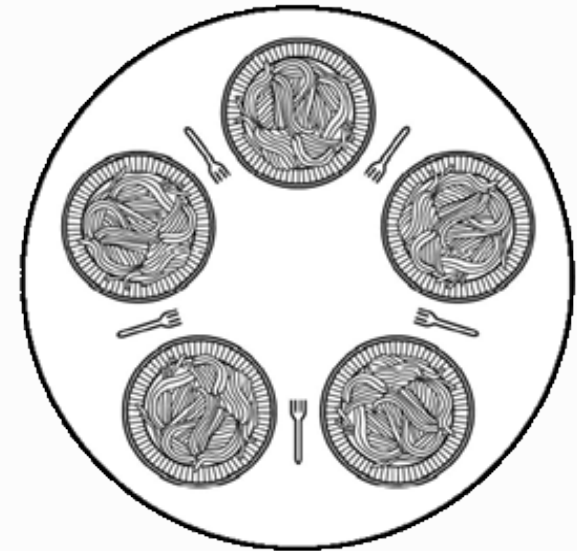
Quando um filósofo fica com fome, ele tenta pegar os garfos da direita e da esquerda (não necessariamente nesta ordem).

- O importante é pegar um de cada vez.
- Se conseguir os dois garfos, ele irá comer.
- Posteriormente coloca os garfos na mesa.
- E volta a pensar.

Jantar dos Filósofos

A questão fundamental é:

Você é capaz de desenvolver um algoritmo para o Jantar dos Filósofos que faça o que deve fazer e nunca “trave”?



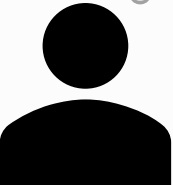
Jantar dos Filósofos

Algoritmo apresentado é a solução óbvia;

- **take_fork** pega o garfo.
 - Esta função bloqueia a thread caso o garfo esteja sendo utilizado.
- **put_fork** devolve o garfo.
 - Esta função acorda threads que se bloquearam pela indisponibilidade de garfo.

```
#define N 5                                /* número de filósofos */  
  
void philosopher(int i)                   /* i: número do filósofo, de 0 a 4 */  
{  
    while (TRUE) {  
        think( );                        /* o filósofo está pensando */  
        take_fork(i);                    /* pega o garfo esquerdo */  
        take_fork((i+1) % N);            /* pega o garfo direito; % é o operador modulo */  
        eat( );                          /* hummm! Espaguetel */  
        put_fork(i);                     /* devolve o garfo esquerdo à mesa */  
        put_fork((i+1) % N);             /* devolve o garfo direito à mesa */  
    }  
}
```


Jantar dos Filósofos

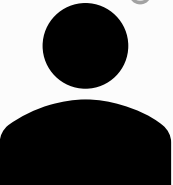


Onde estão
os
semáforos?

```
#define N 5                                     /* número de filósofos */

void philosopher(int i)                         /* i: número do filósofo, de 0 a 4 */
{
    while (TRUE) {
        think( );                             /* o filósofo está pensando */
        take_fork(i);                          /* pega o garfo esquerdo */
        take_fork((i+1) % N);                  /* pega o garfo direito; % é o operador modulo */
        eat( );                                /* hummm! Espagete! */
        put_fork(i);                           /* devolve o garfo esquerdo à mesa */
        put_fork((i+1) % N);                   /* devolve o garfo direito à mesa */
    }
}
```

Jantar dos Filósofos



Onde está
o **erro** do
algoritmo?

```
#define N 5                                     /* número de filósofos */

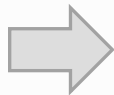
void philosopher(int i)                         /* i: número do filósofo, de 0 a 4 */
{
    while (TRUE) {
        think( );                               /* o filósofo está pensando */
        take__fork(i);                          /* pega o garfo esquerdo */
        take__fork((i+1) % N);                  /* pega o garfo direito; % é o operador modulo */
        eat( );                                 /* hummm! Espaguetel */
        put__fork(i);                           /* devolve o garfo esquerdo à mesa */
        put__fork((i+1) % N);                  /* devolve o garfo direito à mesa */
    }
}
```

Jantar dos Filósofos

Um situação de erro ocorre de forma simples.:

- Imagine que todos os filósofos resolvam comer.
- Todos capturam o garfo do lado esquerdo.
- Quando tentarem pegar o garfo do lado direito, nenhum encontrará disponibilidade, então todos os threads serão bloqueados. (**Deadlock!**)

Algoritmo
trivial... Mas
contem erro



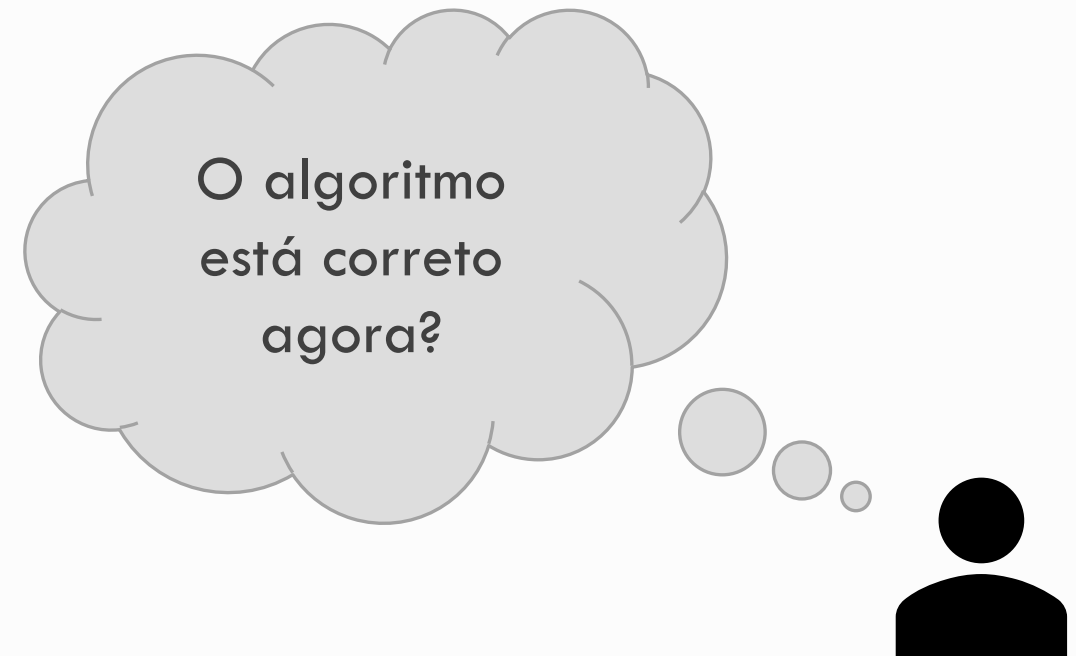
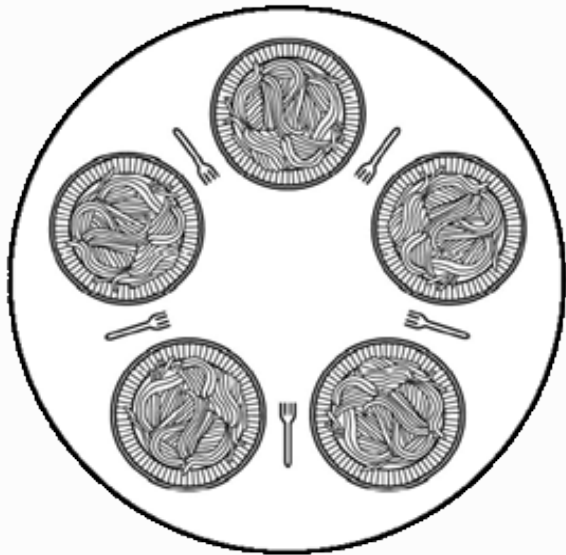
```
#define N 5                                /* número de filósofos */  
  
void philosopher(int i)                    /* i: número do filósofo, de 0 a 4 */  
{  
    while (TRUE) {  
        think( );                          /* o filósofo está pensando */  
        take_fork(i);                       /* pega o garfo esquerdo */  
        take_fork((i+1) % N);               /* pega o garfo direito; % é o operador modulo */  
        eat( );                             /* hummm! Espaguetel */  
        put_fork(i);                        /* devolve o garfo esquerdo à mesa */  
        put_fork((i+1) % N);               /* devolve o garfo direito à mesa */  
    }  
}
```

Jantar dos Filósofos

O algoritmo anterior **pode ser** adaptado.

Se (filósofo estiver com o garfo esquerdo na mão **E** garfo direito estiver ocupado) **então**

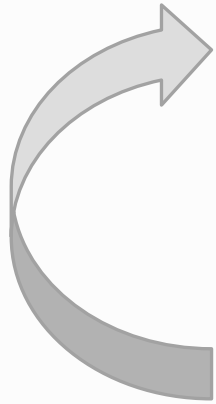
- o filósofo devolve o garfo esquerdo e reinicia seu procedimento, ao invés de dormir.



Jantar dos Filósofos

Outro algoritmo trivial...

Exemplo de execução:



- Todos os filósofos podem em sincronismo pegar os garfos com a mão esquerda...
- Cada um verifica se pode pegar o da direita (e nenhum vai poder)...
- Todos devolvem seus garfos da mão esquerda.
- Reiniciando o processo.

Jantar dos Filósofos

Outro algoritmo trivial... Mas errado...

Esta situação na qual todos os programas continuam executando indefinidamente, mas falham ao tentar progredir, é conhecida como **Starvation**.

Não confundir **Deadlock** com **Starvation**!

Algoritmo do Jantar dos Filósofos (modelado por *Dijkstra* (1965))

```
#define N          5          /* número de filósofos */
#define LEFT      (i+N-1)%N   /* número do vizinho à esquerda de i */
#define RIGHT     (i+1)%N     /* número do vizinho à direita de i */
#define THINKING  0          /* o filósofo está pensando */
#define HUNGRY    1          /* o filósofo está tentando pegar garfos */
#define EATING    2          /* o filósofo está comendo */
typedef int semaphore;        /* semáforos são um tipo especial de int */
int state[N];                /* arranjo para controlar o estado de cada um */
semaphore mutex = 1;         /* exclusão mútua para as regiões críticas */
semaphore s[N];              /* um semáforo por filósofo */

void philosopher(int i)      /* i: o número do filósofo, de 0 a N-1 */
{
    while (TRUE) {           /* repete para sempre */
        think();             /* o filósofo está pensando */
        take_forks(i);        /* pega dois garfos ou bloqueia */
        eat();               /* hummm! Espaguete! */
        put_forks(i);         /* devolve os dois garfos à mesa */
    }
}
```

```
public synchronized void
down(Semaforo s) throws Exception {
    synchronized(this){
        while( s.contador == 0 ){
            this.wait();
        }
        s.contador--;
    }
}
```

Algoritmo do Jantar dos Filósofos (modelado por Dijkstra (1965))

```
public synchronized void
up(Semaforo s) throws Exception {
    synchronized(this){
        num.contador++;
        this.notifyAll();
    }
}
```

```
void take_forks(int i)
{
    down(&mutex);
    state[i] = HUNGRY;
    test(i);
    up(&mutex);
    down(&s[i]);
}
```

```
void put_forks(i)
{
    down(&mutex);
    state[i] = THINKING;
    test(LEFT);
    test(RIGHT);
    up(&mutex);
}
```

```
void test(i)
{
    if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING) {
        state[i] = EATING;
        up(&s[i]);
    }
}
```

/* i: o número do filósofo, de 0 a N-1 */

/* entra na região crítica */

/* registra que o filósofo está faminto */

/* tenta pegar dois garfos */

/* sai da região crítica */

/* bloqueia se os garfos não foram pegos */

/* i: o número do filósofo, de 0 a N-1 */

/* entra na região crítica */

/* o filósofo acabou de comer */

/* vê se o vizinho da esquerda pode comer agora */

/* vê se o vizinho da direita pode comer agora */

/* sai da região crítica */

/* i: o número do filósofo, de 0 a N-1 */

Próxima aula

Leitura:

Sistemas operacionais modernos

Deadlock (impasse)

Referências

Sistemas Operacionais Modernos. Tanenbaum, A. S. 2ª edição. 2003.

Sistemas Operacionais. Conceitos e Aplicações. A. Silberschatz; P. Galvin; G. Gagne. 2000.

Sistemas Operacionais – Projeto e Implementação. Tanenbaum, A. S. 2ª edição. 2000.

Slides Prof. Humberto Brandão