Sistemas Operacionais II

Gerência de Processos

Comunicação Interprocessos
Problemas Clássicos

Material produzido por Prof. Humberto Brandão
humberto@bcc.unifal-mg.edu.br

Prof. Douglas Castilho
douglas.castilho@ifsuldeminas.edu.br

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.

Problemas Clássicos na CIP

1º: Jantar dos Filósofos

Jantar dos Filósofos

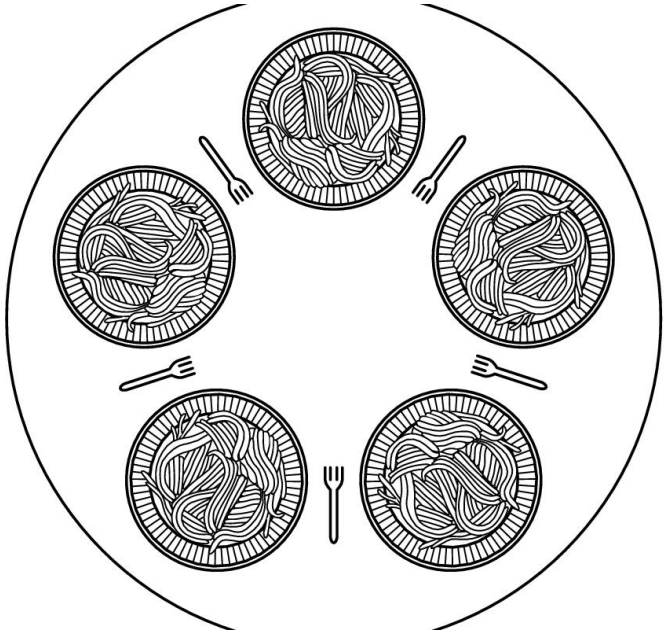
- Dijkstra, em 1965, formulou e resolveu (utilizando o seu método - semáforos) um dos clássicos problemas de CIP:
  - O Jantar dos Filósofos;
- Este problema se tornou tão famoso, que, se você inventar uma nova técnica para CIP, está automaticamente obrigado a demonstrar o *quão elegante e eficiente* 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

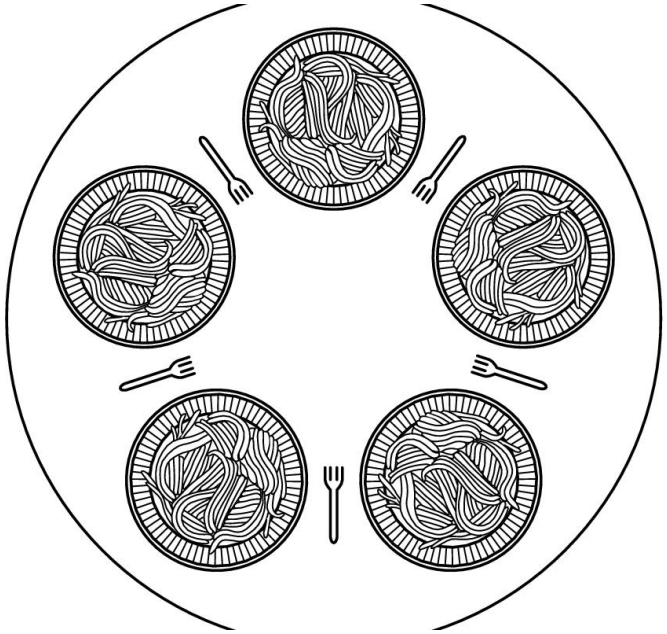
- O problema pode ser exposto de uma forma simples:
  - Cinco filósofos sentam em uma mesa circular;
    - Cada um com um processamento independente;
  - Cada um possui um prato de espaguete;
  - O espaguete está escorregadio e cada filósofo precisa de dois garfos para comer
    - Recursos necessários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

Jantar dos Filósofos - Formulação

- *Dinâmica do problema*:
  - A vida do filósofo se resume a:
    - Comer;
    - Pensar;
  - Nunca comem e pensam ao mesmo tempo...
    - Cada linha independente, pensa e depois come...



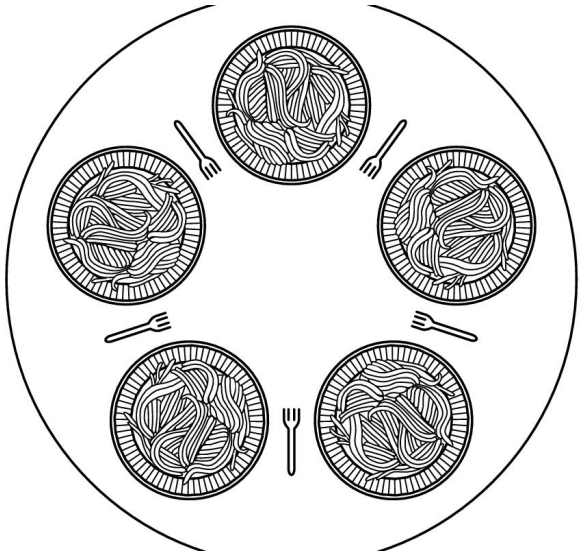
Jantar dos Filósofos – Geral do algoritmo base



- 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 - Formulação

- 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

"Solução" óbvia

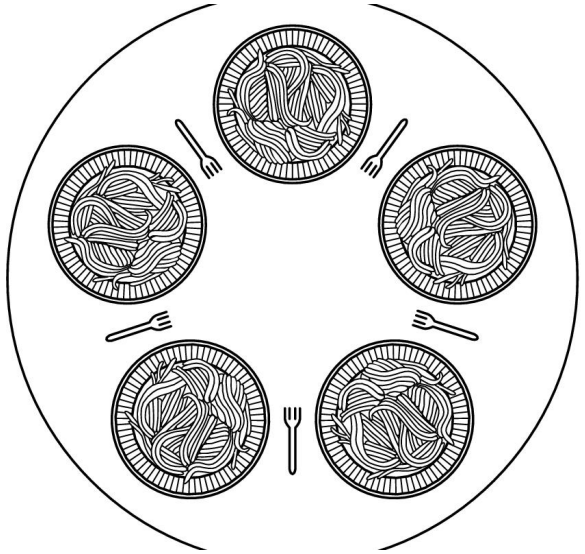
Jantar dos Filósofos

- 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! Espaguete! */
            put_fork(i);                        /* devolve o garfo esquerdo à mesa */
            put_fork((i+1) % N);                /* devolve o garfo direito à mesa */
      }
}
```

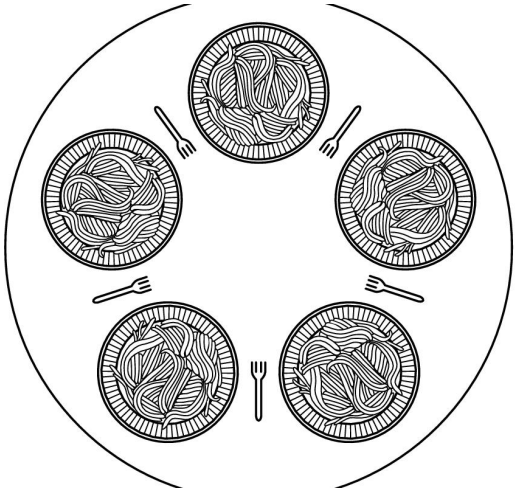
Jantar dos Filósofos

- *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! Espaguete! */
            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! Espaguete! */
            put_fork(i);                        /* devolve o garfo esquerdo à mesa */
            put_fork((i+1) % N);                /* devolve o garfo direito à mesa */
      }
}
```

Algoritmo trivial... Mas contem erro

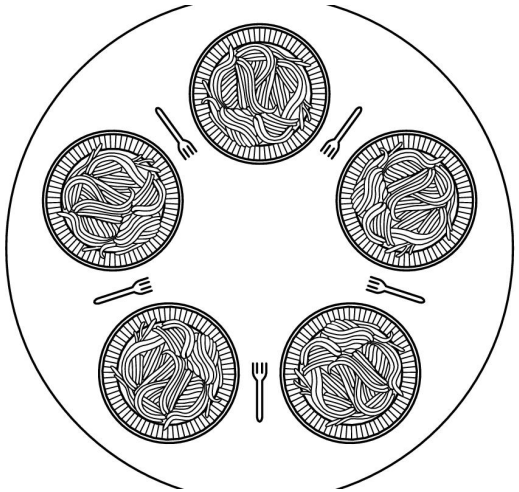
```
#define N 5                                     /* número de filósofos */

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! Espaguete! */
            put_fork(i);                        /* devolve o garfo esquerdo à mesa */
            put_fork((i+1) % N);                /* devolve o garfo direito à mesa */
      }
}
```

- 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*)

Outro algoritmo trivial...

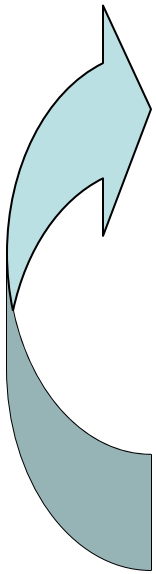
- O algoritmo anterior pode ser adaptado...
  - Se (filósofo estiver com o garfo esquerdo na mão E garfo direito estiver ocupado) então
    - o filósofo devolve o garfo esquerdo e reinicia seu procedimento, ao invés de dormir...

- O algoritmo está correto agora?

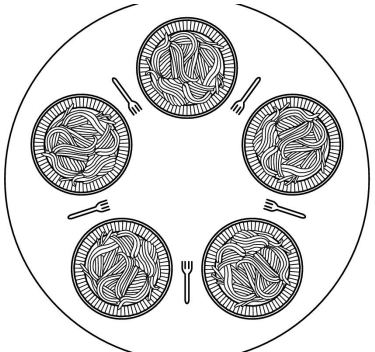


Outro algoritmo trivial...

Exemplo de execução...



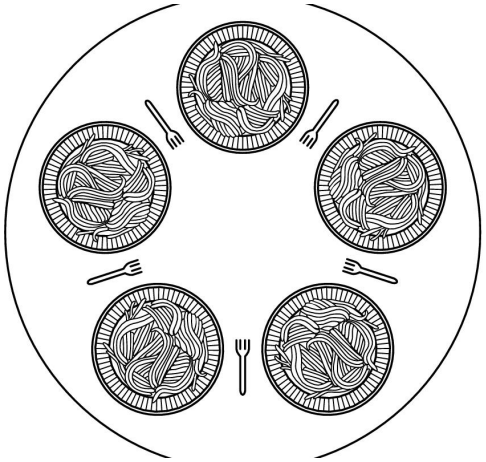
- Todos os filósofos podem em sincronismo pegar os garfos com a mão esquerda...
- Cada um verifica se pode pegar o da direita (e nenhum vai poder)...
- Todos devolvem seus garfos da mão esquerda.
- Reiniciando o processo...



Outro algoritmo trivial... Mas errado...

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

Deadlock != Starvation



Jantar dos Filósofos

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

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 */
    }
}
```

Algoritmo do JF (modelado por *Dijkstra* (1965))

```
void take_forks(int i)                  /* i: o número do filósofo, de 0 a N–1 */
{
    down(&mutex);                       /* entra na região crítica */
    state[i] = HUNGRY;                  /* registra que o filósofo está faminto */
    test(i);                            /* tenta pegar dois garfos */
    up(&mutex);                         /* sai da região crítica */
    down(&s[i]);                        /* bloqueia se os garfos não foram pegos */
}

void put_forks(i)                       /* i: o número do filósofo, de 0 a N–1 */
{
    down(&mutex);                       /* entra na região crítica */
    state[i] = THINKING;                /* o filósofo acabou de comer */
    test(LEFT);                         /* vê se o vizinho da esquerda pode comer agora */
    test(RIGHT);                        /* vê se o vizinho da direita pode comer agora */
    up(&mutex);                         /* sai da região crítica */
}

void test(i)                            /* i: o número do filósofo, de 0 a N–1 */
{
    if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING) {
        state[i] = EATING;
        up(&s[i]);
    }
}
```

Trabalho Prático 2

- Implementar os filósofos jantando com solução do livro;
- Implementar uma solução sua (individual);
  - Utilize sua criatividade;
- Comparar a eficácia e eficiência com o algoritmo do livro;
  - Seu sistema bloqueou algum filósofo por tempo indeterminado?
  - Qual é o tempo médio de espera para comer nos dois algoritmos implementados?

Bibliografia

- TANENBAUM, A. S., Sistemas Operacionais Modernos, 2ª. Edição, São Paulo: Prentice Hall, 2003.

- SILBERSCHATZ, A; GALVIN, P e GAGNE, G.; Sistemas Operacionais. Conceitos e Aplicações. Rio de Janeiro: Editora Campus, 2000.

- TANENBAUM, A. S.; WOODHULL, A.S.; Sistemas operacionais: projeto e implementação. 2a Ed. Porto Alegre: Ed. Bookman, 2000.