TÓPICO 4

Problemas Clássicos de Programação Concorrente

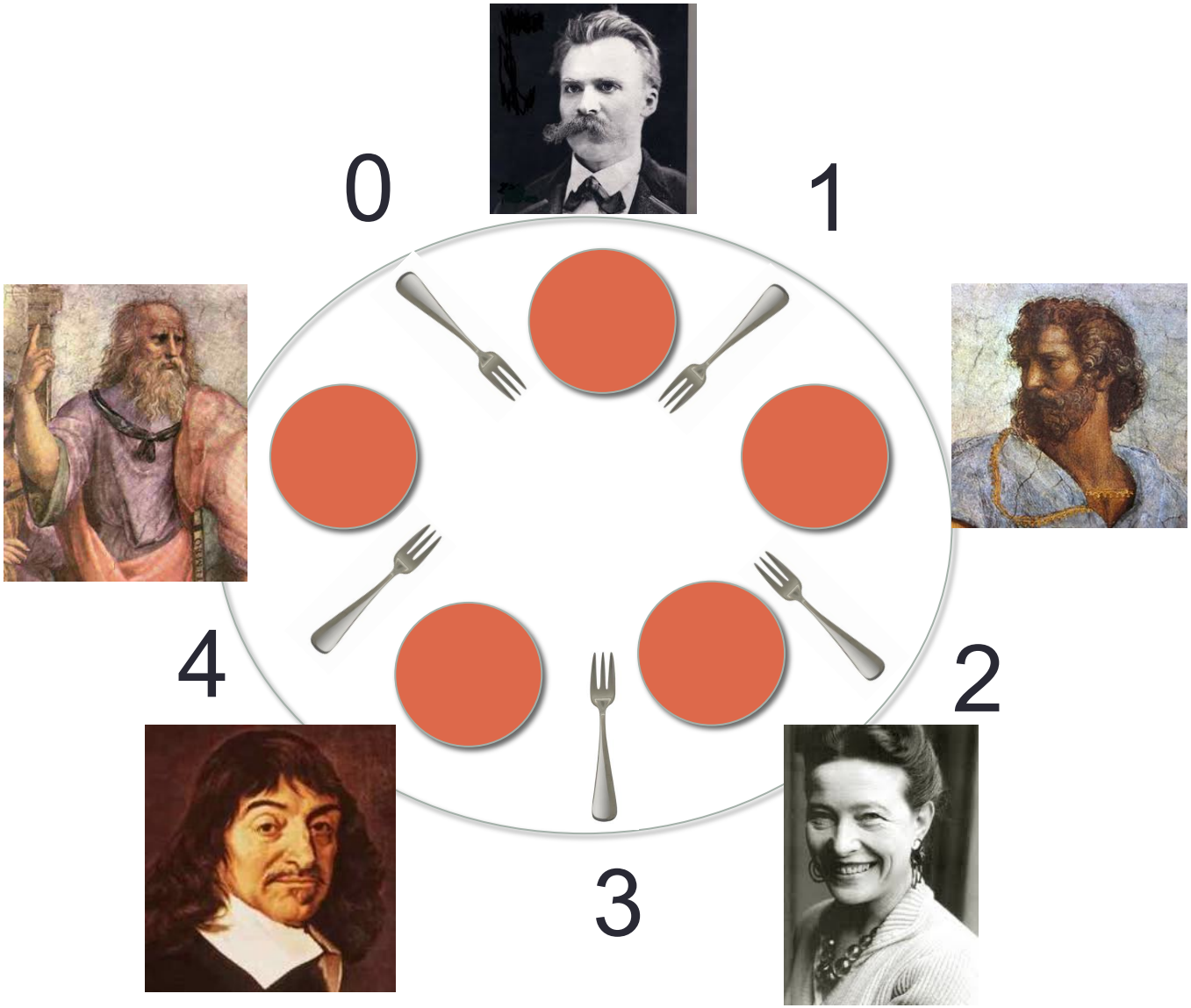
Introdução

- Existem vários problemas que foram propostos na área de comunicação entre processos e servem para ilustrar o que pode ocorrer se a concorrência não for corretamente tratada. Dentre estes problemas, vamos analisar:
  - Os filósofos glutões
  - O problema dos leitores/escritores
  - O barbeiro dorminhoco

Os Filósofos Glutões (Dining Philosophers)

- Proposto por Dijkstra em 1965.
- Formulação:
  - Existem 5 filósofos sentados em torno de uma mesa redonda.
  - Existe um prato na frente de cada filósofo e um garfo entre cada 2 pratos.
  - Cada filósofo necessita de 2 garfos para comer.
  - Os filósofos só tomam 2 ações: pensar e comer.
  - A decisão de comer implica em pegar 1 garfo, pegar o outro garfo e comer.

Os Filósofos Glutões (Dining Philosophers)



Os Filósofos Glutões (Dining Philosophers)

- Questão: Resolver algoritmicamente este problema de maneira que não haja deadlock nem starvation e que seja permitida a máxima concorrência possível.

Os Filósofos Glutões (Dining Philosophers)

- Primeira solução (imediata):

```
#define N 5
void filosofo (int i)
{
    while (1) {
        pensa();
        pega_garfo(i);
        pega_garfo((i+1) % N);
        come();
        libera_garfo(i);
        libera_garfo((i+1) % N); }
}
```

Análise da primeira solução:
Pode levar a deadlock se todos os filósofos pegarem o garfo ao mesmo tempo.

Os Filósofos Glutões (Dining Philosophers)

- Segunda solução (correção imediata da primeira):

```
#define N 5
void filosofo (int i)
{
while (1) {
        pensa();
        pega_garfo(i);
        if (ocupado_garfo((i+1)%N);
            libera_garfo(i);
        else {
            pega_garfo((i+1) % N);
            come();
            libera_garfo(i);
            libera_garfo((i+1) % N); } }
}
```

Análise da segunda solução: Pode levar facilmente a starvation.

Os Filósofos Glutões (Dining Philosophers)

- Terceira solução:
  - Baseia-se na introdução de estados dos filósofos (pensando, com fome, comendo). Um filósofo só pode estar comendo se os seus vizinhos não estiverem comendo.
  - Existem inúmeras soluções para este problema que seguem estas constatações. A solução apresentada a seguir usa monitores e variáveis de condição.

Os Filósofos Glutões (Dining Philosophers)

Terceira solução (1/2)

```
monitor filosofos_glutoes
    var estado: array[0..4] of (pensando, comfome, comendo)
    var meu_estado: array[0..4] of condicao;
    for (i=0;i<5;i++) estado[i] = pensando;
    procedure pega(i) {
       estado[i] = comfome;
       testa(i);
       if (estado[i] != comendo)
          condition_wait(meu_estado[i]); }
procedure larga(i) {
      estado[i] := pensando;
      testa ((i-1) % 5);
      testa ((i+1) % 5); }
procedure testa(i) {
      if ((estado[(i-1) % 5] != comendo) && (estado[i] == comfome) &&
          (estado[(i+1) % 5] != comendo))
      {
          estado[i] = comendo;
          cond_signal(meu_estado[i]);}}
end monitor
```

Os Filósofos Glutões (Dining Philosophers)

- Terceira solução (2/2):

```
filosofos_glutoes.pega(i);
/* come */
filosofos_glutoes.larga(i);
```

Análise da terceira solução: Esta solução é livre de deadlock e garante a exclusão mútua (nunca dois vizinhos vão estar comendo ao mesmo tempo) e a concorrência máxima (até dois filósofos podem comer ao mesmo tempo), porém um filósofo pode ficar esperando indefinidamente (starvation).

Os leitores/escritores

- Proposto por Courtois et al. em 1971.
- Também conhecido como protocolo Multiple Readers/ Single Writer (MR/SW).
- Modela problemas onde vários leitores podem acessar um dado simultaneamente porém o escritor deve ter acesso exclusivo. Ex: bases de dados.

Os leitores/escritores

```
semaphore mutex = 1, db =1;
int num_leitores;
void reader()
{
   while (1) {
      P(&mutex);
      num_leitores = num_leitores + 1;
      if (num_leitores == 1)
          P(&db);
      V(&mutex);
      le_dados();
      P(&mutex);
      num_leitores = num_leitores -1;
      if (num_leitores == 0) V(&db);
      V(&mutex);
      /* fora da secao critica */
  }
}
```

```
void writer()
{
  while (1)
  {
   /* secao não critica */
   P(&db);
   escreve_dados();
   V(&db);
  }
}
```

Análise da solução:
Esta solução favorece os leitores: o escritor não escreve enquanto há leitor lendo.

O Barbeiro dorminhoco (sleeping barber)

- Proposto por Dijkstra.
- Numa barbearia, existem um barbeiro, uma cadeira de barbeiro e várias cadeiras de clientes.
- Se não há nenhum cliente na barbearia, o barbeiro dorme.
- Se chegarem clientes e o barbeiro estiver cortando o cabelo de alguém, os clientes esperam nas cadeiras.
- Se não houver cadeiras, os clientes vão embora.

O Barbeiro dorminhoco (sleeping barber)

```
semaphore clientes = 0;
semaphore barbeiro = 0;
semaphore mutex = 1;
int esperando = 0;
void Barbeiro()
{
  while (1) {
    P(clientes);
    P(mutex);
    esperando = esperando - 1;
    V(barbeiro);
    V(mutex);
    /* corta cabelo */
  }
}
```

```
void cliente()
{
   P(mutex);
   if (esperando < CADEIRAS)
   {
      esperando = esperando +1;
      V(clientes);
      V(mutex);
      P(barbeiro);
      /*tem o cabelo cortado */
  }
  else
     V(mutex);
}
```