

Sistemas Operacionais

Interação entre tarefas - impasses

Prof. Carlos Maziero

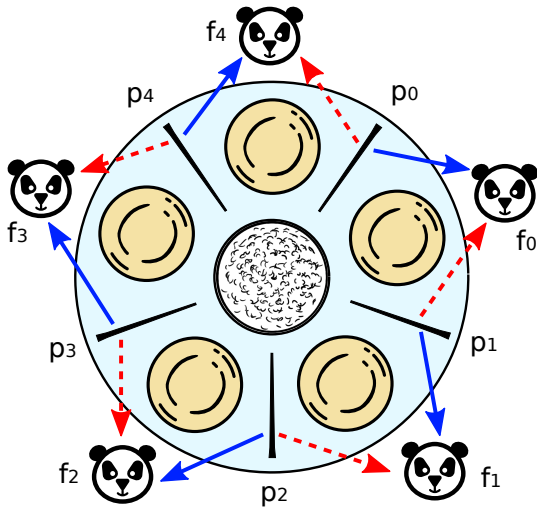
DInf UFPR, Curitiba PR

Fevereiro de 2019

Conteúdo

- 1 O conceito de impasse
- 2 Condições para um impasse
- 3 Grafos de alocação de recursos
- 4 Estratégias de tratamento
 - Prevenção
 - Impedimento
 - Detecção e resolução

O impasse dos filósofos



Impasses

Impasse

Grupo de tarefas bloqueadas aguardando **umas pelas outras**.

Coordenar tarefas implica em bloquear tarefas conflitantes

- Suspende algumas tarefas enquanto outras executam.
- Cada recurso é associado a um semáforo.
- As tarefas aguardam os semáforos para acessar os recursos.
- Essas restrições podem levar a **impasses**.

Exemplo de impasse

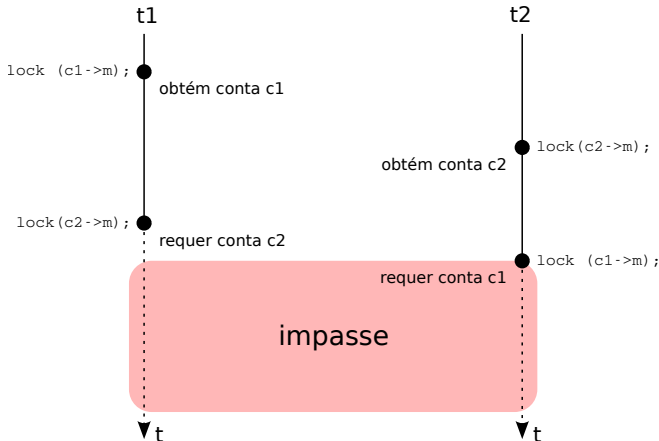
Operação de transferência bancária:

```

1 void transferir (conta_t* contaDeb, conta_t* contaCred, int valor)
2 {
3     sem_down (contaDeb->lock) ; // obtém acesso a contaDeb
4     sem_down (contaCred->lock) ; // obtém acesso a contaCred
5
6     if (contaDeb->saldo >= valor)
7     {
8         contaDeb->saldo -= valor ; // debita valor de contaDeb
9         contaCred->saldo += valor ; // credita valor em contaCred
10    }
11    sem_up (contaDeb->lock) ; // libera acesso a contaDeb
12    sem_up (contaCred->lock) ; // libera acesso a contaCred
13 }
  
```

Exemplo de impasse

Dois clientes (tarefas t_1 e t_2) fazem transferências simultâneas entre suas contas ($c_1 \rightarrow c_2$ e $c_2 \rightarrow c_1$):



Um impasse real (São Paulo SP, 2017)



Condições para um impasse

Exclusão mútua : recursos acessados com exclusão mútua, gerida por semáforos ou similares.

Posse e espera : a tarefa tem um recurso e quer acessar outro.

Não-preempção : a tarefa só libera os recursos quando quiser.

Espera circular : ciclo de esperas: a tarefa t_1 quer um recurso retido por t_2 , que quer um recurso retido por t_3 , que quer um recurso retido por t_1 .

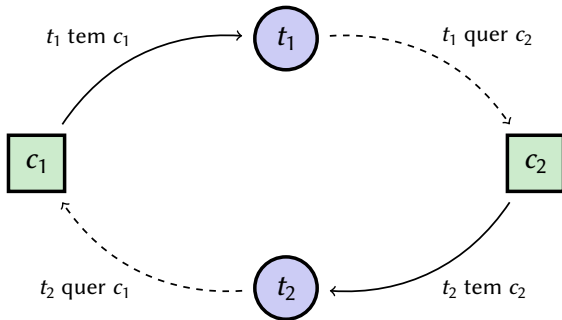
Estas quatro condições são **necessárias** (mas não suficientes) para a ocorrência de impasses.

Grafo de alocação de recursos

- Permite visualizar a alocação de recursos
- Permite detectar impasses

○	tarefa
□	recurso
□ → ○	posse de um recurso por uma tarefa (o recurso “pertence” à tarefa)
○ --> □	requisição de um recurso por uma tarefa (a tarefa “quer” o recurso)
□ ● ●	dois recursos do mesmo tipo

Impasse entre as contas bancárias



Impasse entre as contas bancárias

Percebe-se que as quatro condições estão presentes:

Exclusão mútua : as contas são protegidas por semáforos

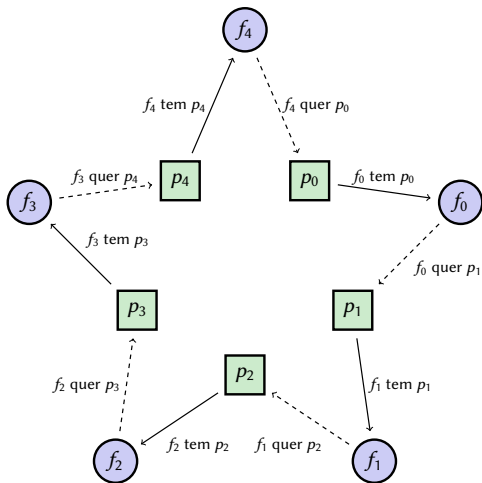
Posse e espera : requer conta c_i , requer conta c_j

Não-preempção : conta só é liberada com `sem_up(c)`

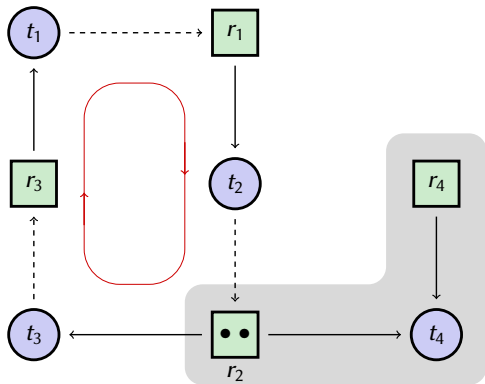
Espera circular : $c_1 \rightarrow t_1 \dashrightarrow c_2 \rightarrow t_2 \dashrightarrow c_1$

Quatro condições + um recurso de cada tipo \rightarrow **impasse!**

Impasse entre os filósofos



Múltiplas instâncias de recursos



Sem impasse, mesmo com $t_1 \dashrightarrow r_1 \rightarrow t_2 \dashrightarrow r_2 \rightarrow t_3 \dashrightarrow r_3 \rightarrow t_1$

Impasses

Estratégias para tratar de impasses:

Ignorar : impasse é um problema do programador!

Prevenir : regras de programação para evitar impasses

Impedir : monitorar o uso de recursos e impedir impasses

Detectar : detectar a ocorrência de impasses e desfazê-los

A mais usada em SOs: ignorar... 

Prevenção de impasses

Programar de forma a evitar as quatro condições:

■ Exclusão mútua:

- Reduzir áreas de exclusão ao mínimo
- Usar técnicas alternativas, como o *spooling*
- Exemplo: acesso a impressoras em um SO

■ Posse e espera:

- Usar um recurso de cada vez
- Obter todos os recursos antes de iniciar
- Exemplo: na transferência, tratar uma conta por vez

Prevenção de impasses

- **Não-preempção:**

- Poder “arrancar” os recursos dos processos
- Difícil de implementar, pode gerar inconsistências

- **Espera circular:**

- Os recursos são ordenados
- As tarefas os solicitam nessa ordem
- Exemplo: acessar as contas segundo os números de conta

Basta quebrar **uma** das condições!

Prevenir impasse com um saleiro!

```

1  #define NUMFILO 5
2  semaphore hashi [NUMFILO] ; // palitos são semáforos (iniciam em 1)
3  semaphore saleiro ;        // um semáforo para o saleiro
4
5  task filosofo (int i)      // filósofo i (entre 0 e 4)
6  {
7      int dir = i ;
8      int esq = (i+1) % NUMFILO ;
9
10     while (1)
11     {
12         meditar () ;
13         down (saleiro) ;    // pega saleiro
14         down (hashi [dir]) ; // pega palito direito
15         down (hashi [esq]) ; // pega palito esquerdo
16         up (saleiro) ;     // devolve saleiro
17         comer () ;
18         up (hashi [dir]) ;  // devolve palito direito
19         up (hashi [esq]) ;  // devolve palito esquerdo
20     }
21 }
  
```

Impedimento de impasses

Estratégia:

Acompanhar a alocação dos recursos às tarefas e negar acessos de recursos que possam levar a situações inseguras.

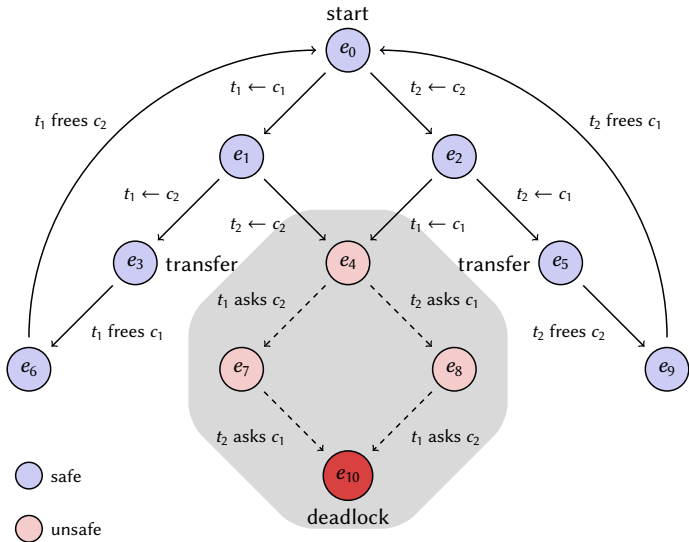
Grafo de estados do sistema:

- **Estado:** uma distribuição de recursos entre as tarefas
- **Aresta:** uma alocação ou liberação de recursos

Estados do sistema:

- **Estados seguros:** permitem evoluir aos demais estados
- **Estados inseguros:** somente levam a impasses

Grafo de estados do sistema



Detecção e correção de impasses

Observar o sistema e, quando ocorrer um impasse, resolvê-lo.

Como **detectar** impasses?

- Manter um grafo de alocação de recursos:
 - Atualizar a cada requisição/alocação/liberação
 - Detectar a formação de ciclos no grafo
 - Pode exigir muito processamento
- Monitorar nível de atividade do sistema:
 - Analisar tarefas suspensas há muito tempo

Detecção e correção de impasses

Como **resolver** impasses?

- **Eliminar tarefas** de modo a romper os ciclos
 - Eliminar a mais nova? A mais antiga? A menos prioritária?
- **Retroceder tarefas**, retornando a um estado seguro
 - Técnica de *rollback* usada em bancos de dados
 - São necessários *checkpoints* periódicos
 - Algumas operações não podem ser desfeitas
 - interações com o usuário
 - envio de pacotes de rede