UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE RUSSAS

Rus0013 - Sistemas Operacionais

Aula 08: Deadlock(Impasses)

Professor Pablo Soares

2022.2

Sumário

- Introdução
- Recursos
  - Recursos Preemptíveis e não Preemptíveis
- Introdução aos Impasses
  - Condições para ocorrência
  - Modelagem
- Algoritmo do Avestruz
- Detecção e Recuperação
- Evitando Impasses
- Prevenção de Impasses

Introdução

- Exemplo : CD + Scanner
  - A e B

Recursos

- Recursos: Hardware ou informação
  - Instâncias
- Preemptivo **x** Não preemptivo
  - Memória, Gravador de CDC

- Uso de Recursos:
  - Pedido (Request ou Open)
  - Uso
  - Liberação

Aquisição de Recurso

- Registro em um sistema de banco de dados
  - Cabe ao processo gerenciar
    - Semáforos

```
typedef int semaphore;
semaphore resource_1;

void process_A(void) {
        down(&resource_1);
        use_resource_1( );
        up(&resource_1);
}
```

(a)

```
typedef int semaphore;
semaphore resource_1;
semaphore resource_2;

void process_A(void) {
        down(&resource_1);
        down(&resource_2);
        use_both_resources( );
        up(&resource_2);
        up(&resource_1);
}
```

(b)

Aquisição de Recurso

- Dois processos A e B
  - Duas Situações

```
typedef int semaphore;
        semaphore resource_1;
        semaphore resource_2;

        void process_A(void) {
                down(&resource_1);
                down(&resource_2);
                use_both_resources( );
                up(&resource_2);
                up(&resource_1);
        }

        void process_B(void) {
                down(&resource_1);
                down(&resource_2);
                use_both_resources( );
                up(&resource_2);
                up(&resource_1);
        }
```

(a)

```
semaphore resource_1;
semaphore resource_2;

void process_A(void) {
        down(&resource_1);
        down(&resource_2);
        use_both_resources( );
        up(&resource_2);
        up(&resource_1);
}

void process_B(void) {
        down(&resource_2);
        down(&resource_1);
        use_both_resources( );
        up(&resource_1);
        up(&resource_2);
}
```

(b)

Introdução aos Impasses

- "Um conjunto de processos está em ***impasse*** se cada processo no conjunto está esperando por um evento que somente um outro processo do conjunto pode causar."
- Para esse modelo
  - Apenas um único thread
  - Não existem interrupções para acordar processos bloqueados
- **Impasse de Recurso**

Deadlocks (Impasses)

- **As seguintes condições podem levar a ocorrência de deadlock**

1. ***Condição de Exclusão mútua***
   - Cada recurso está associado a um processo ou disponível
2. ***Condição de Posse e Espera***
   - Um processo aloca recurso enquanto espera por outros recursos
3. ***Condição de Não preempção***
   - Recursos não sofrem preempção, eles devem ser explicitamente liberados pelo processo

Deadlocks (Impasses)

- As três condições são necessárias mas não suficientes

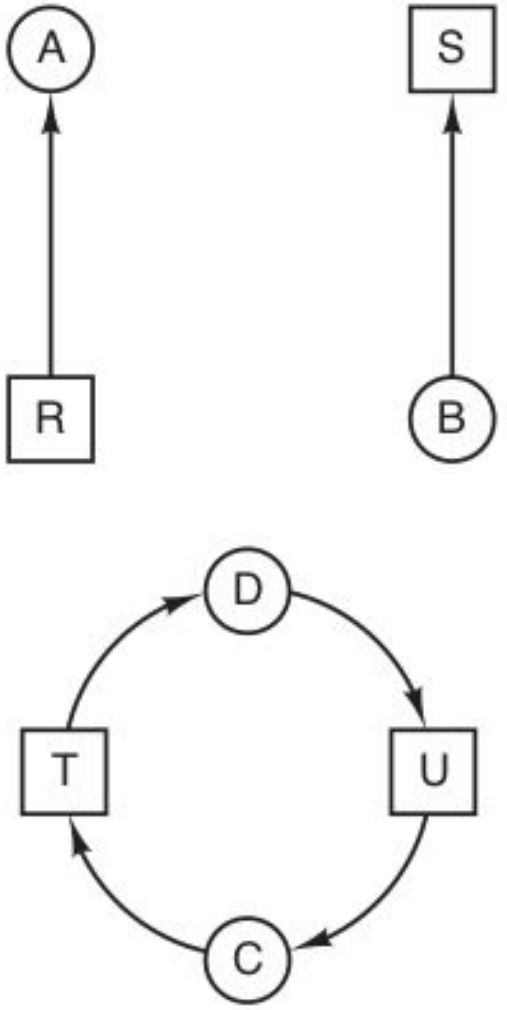
  4. Condição de Esperar Circular

  - Deve existir um encadeamento de dois ou mais processos

- Todas as 4 condições devem estar presentes para que um deadlock ocorra.

Modelagem de Deadlock

- Modelo – Grafo Dirigido
  - Processos são **círculos** e recursos são **quadrados**.
  - O recurso R está alocado ao processo A:
    - O arco de R → A
  - O processo B solicitou o Recurso S
    - O arco de B → S
  - Impasse

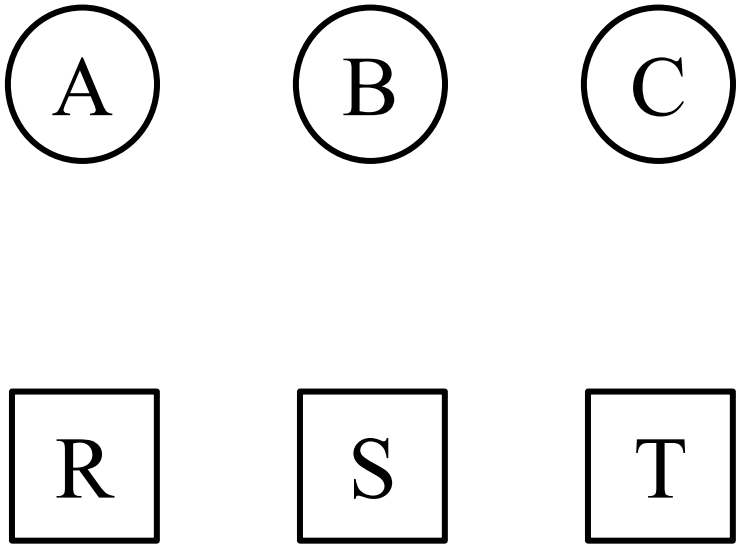


Modelagem de Deadlock

- Uma cadeia circular – não é um exemplo simples

1. **A requisita R**
2. **B requisita S**
3. **C requisita T**
4. **A requisita S**
5. **B requisita T**
6. **C requisita R**

Impasse

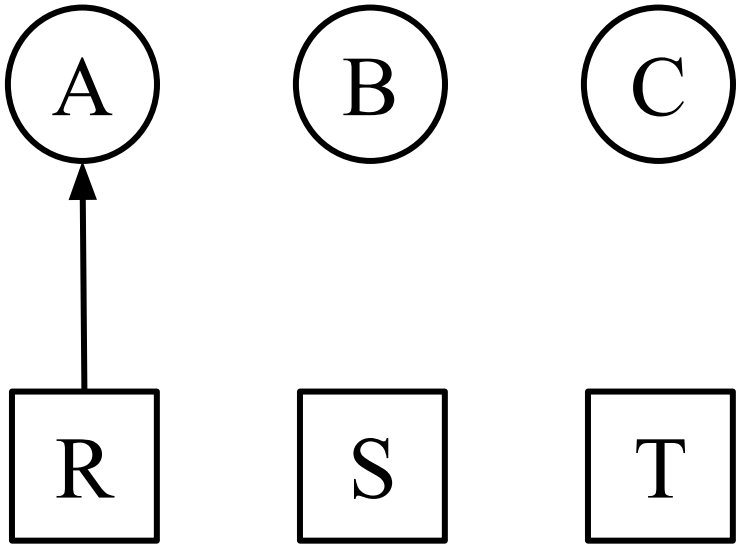


Modelagem de Deadlock

- Uma cadeia circular – não é um exemplo simples

1. **A requisita R**
2. **B requisita S**
3. **C requisita T**
4. **A requisita S**
5. **B requisita T**
6. **C requisita R**

Impasse



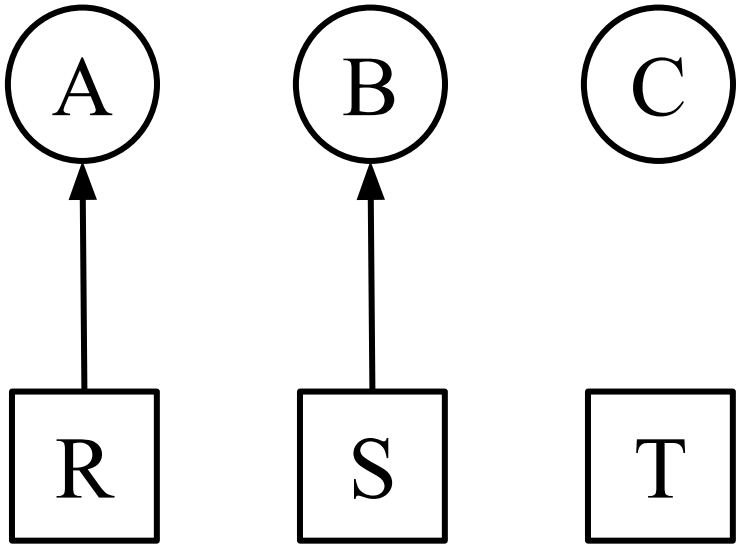
1. A requisita R

Modelagem de Deadlock

- Uma cadeia circular – não é um exemplo simples

1. **A requisita R**
2. **B requisita S**
3. **C requisita T**
4. **A requisita S**
5. **B requisita T**
6. **C requisita R**

Impasse



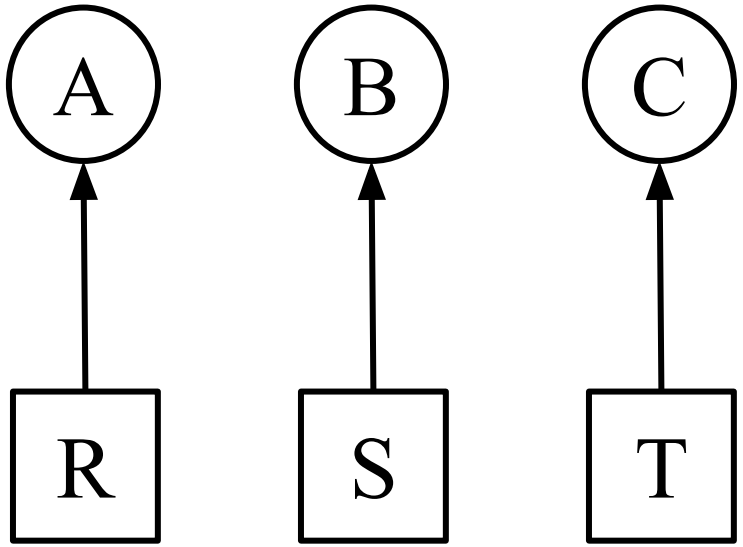
2. B requisita S

Modelagem de Deadlock

- Uma cadeia circular – não é um exemplo simples

1. **A requisita R**
2. **B requisita S**
3. **C requisita T**
4. **A requisita S**
5. **B requisita T**
6. **C requisita R**

Impasse



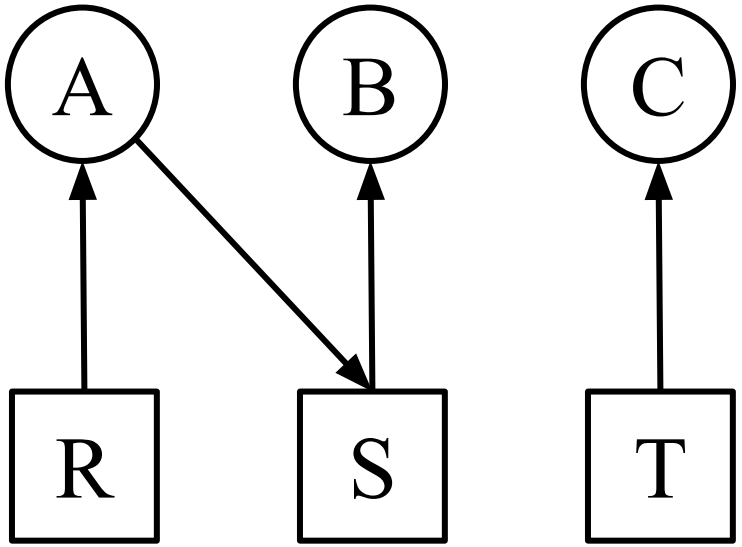
3. C requisita T

Modelagem de Deadlock

- Uma cadeia circular – não é um exemplo simples

1. **A requisita R**
2. **B requisita S**
3. **C requisita T**
4. **A requisita S**
5. **B requisita T**
6. **C requisita R**

Impasse



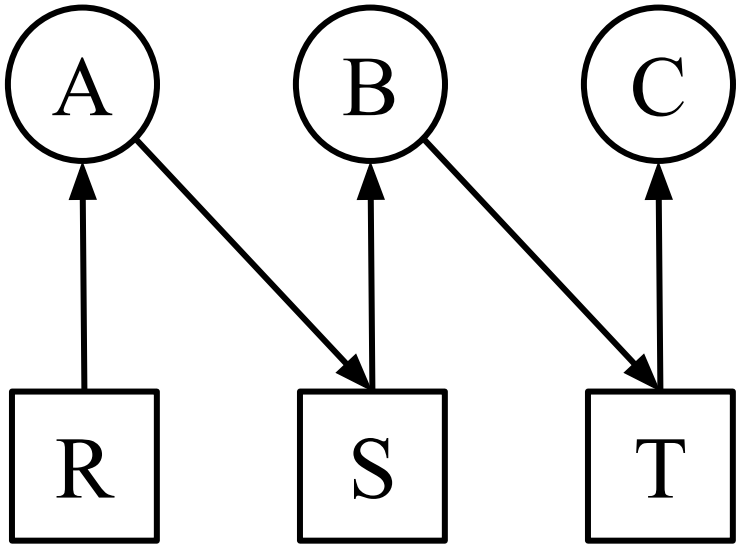
4. A requisita S

Modelagem de Deadlock

- Uma cadeia circular – não é um exemplo simples

1. **A requisita R**
2. **B requisita S**
3. **C requisita T**
4. **A requisita S**
5. **B requisita T**
6. **C requisita R**

Impasse



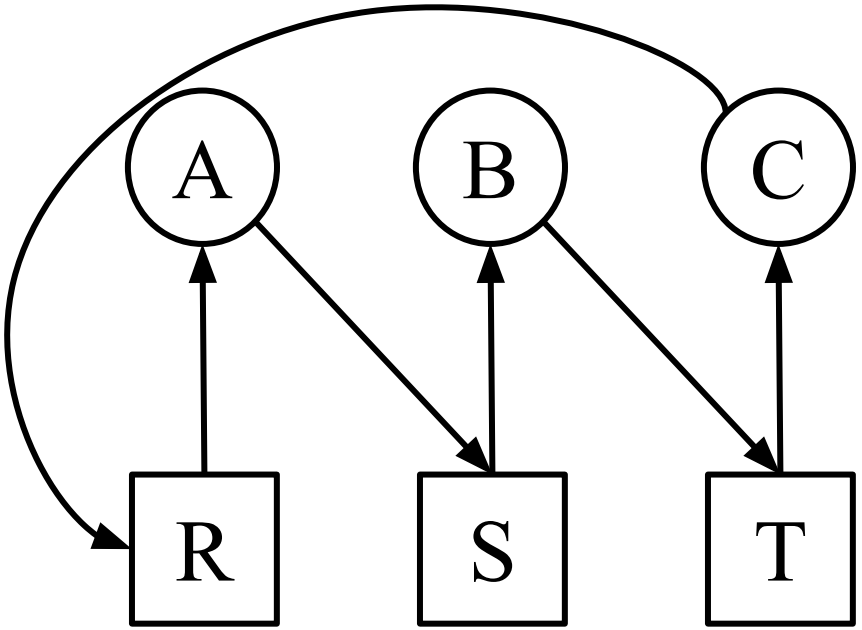
5. B requisita T

Modelagem de Deadlock

- Uma cadeia circular – não é um exemplo simples

1. **A requisita R**
2. **B requisita S**
3. **C requisita T**
4. **A requisita S**
5. **B requisita T**
6. **C requisita R**

Impasse



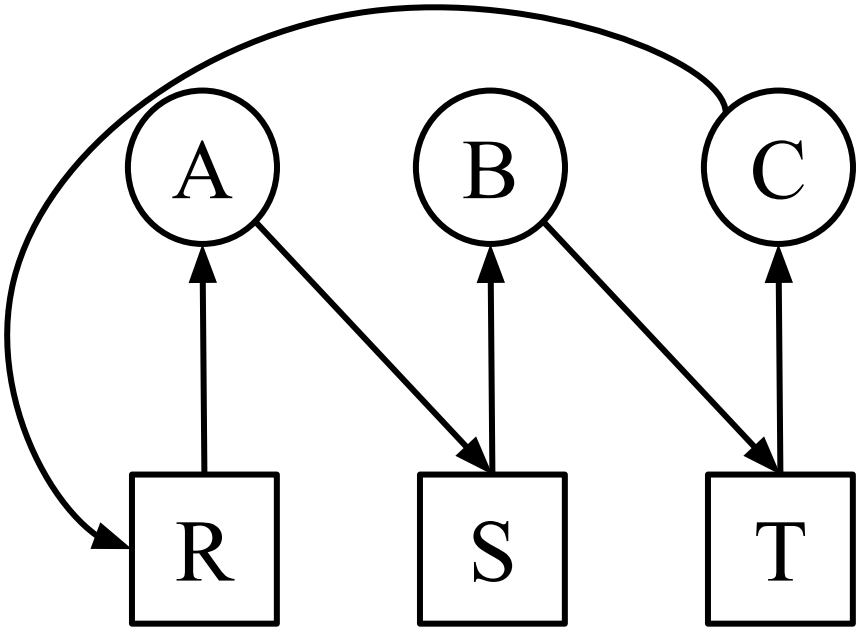
6. C requisita R

Modelagem de Deadlock

- Uma cadeia circular – não é um exemplo simples

1. **A requisita R**
2. **B requisita S**
3. **C requisita T**
4. **A requisita S**
5. **B requisita T**
6. **C requisita R**

Impasse



Impasse

A →S →B→T→C→R→A

- Considere mesmo exemplo sem 2 e 5

Deadlock

- **Estratégias para tratar de deadlocks**
  1. **Ignorar por completo o problema**
     - Se você ignorar, talvez ele ignore você
  2. **Detecção e Recuperação**
     - Deixar o deadlock acontecer, detectar e agir
  3. **Anulação dinâmica por meio de alocação cuidadosa de recursos**
  4. **Prevenção**
     - Negando estruturalmente uma das 4 condições necessárias para ocorrer deadlock

Deadlock

1. Algoritmo do avestruz

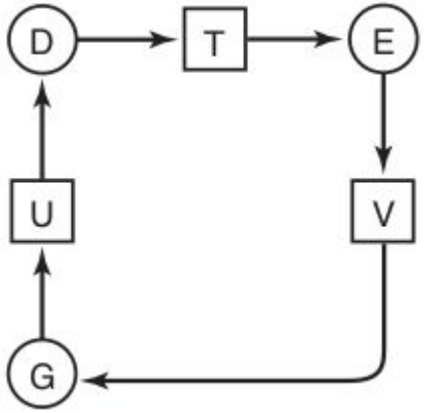
- Enterre a cabeça na areia e finja que nada está acontecendo.

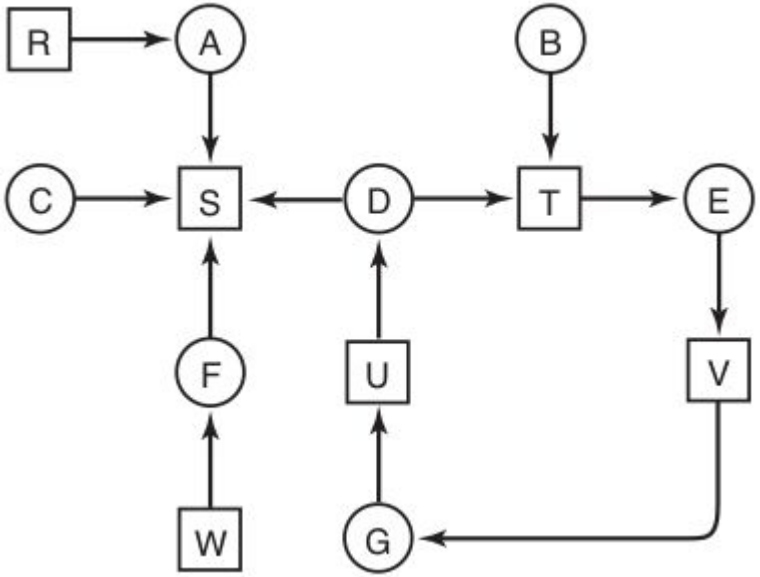
- Deadlocks
  - Em média, uma vez a cada 5 anos
- Falha no sistema
  - Ocorram a cada semana
- Preferível deixar ocorrer do que perder desempenho.

Deadlock

2. Detecção e Recuperação (Um recurso de cada tipo)

1. O processo *A* possui *R* e solicita *S*.
2. O processo *B* não possui nada, mas solicita *T*.
3. O processo *C* não possui nada, mas solicita *S*.
4. O processo *D* possui *U* e solicita *S* e *T*.
5. O processo *E* possui *T* e solicita *V*.
6. O processo *F* possui *W* e solicita *S*.
7. O processo *G* possui *V* e solicita *U*.





Detecção

Para cada nó do grafo execute os seguintes passos:

1) Seja L uma lista vazia inicialmente e designe todos os arcos como não marcados

2) Acrescente o nó atual ao final de L e verifique se o nó parece em L duas vezes. Se sim, o grafo contém um ciclo e o algoritmo termina.

3) Do dado nó, veja se existem quaisquer arcos de saída não marcados

   Se sim, vá para o passo 4;

   Senão vá para o passo 5;

Detecção

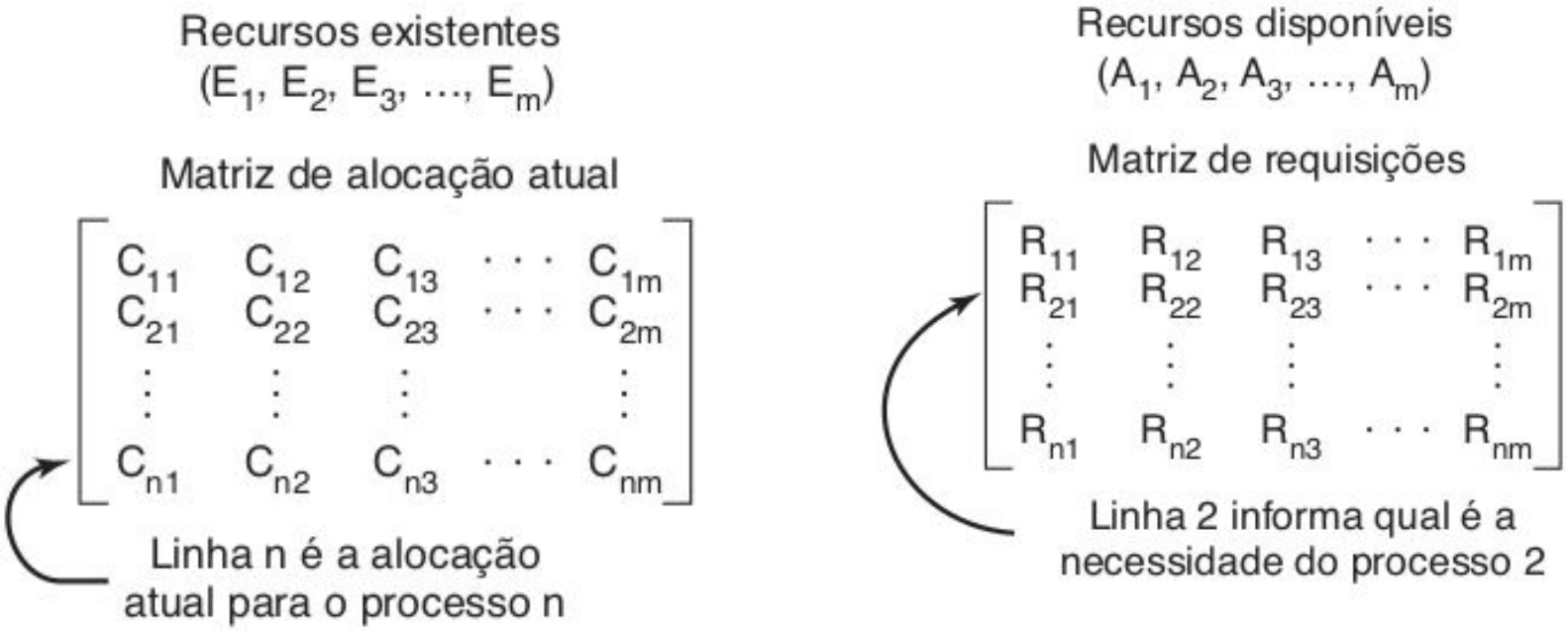
4) Pegue qualquer arco de saída não marcado aleatoriamente e marque-o.

   Então siga para o próximo nó e vá para o passo 3

5) Volte ao nó anterior, torne-o atual e volte ao passo 4. Se este nó é o inicial, o grafo não contém ciclos e o algoritmo termina.

Deadlock

2. Detecção e Recuperação (Varios recurso de cada tipo)



$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$

Detecção

	Unidades de fita	Plotters	Scanners	Blu-rays
E =	4	2	3	1

	Unidades de fita	Plotters	Scanners	Blu-rays
A =	2	1	0	0

Matriz alocação atual

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

$$A = (2\ 2\ 2\ 0)$$

Matriz de requisições

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

$$A = (4\ 2\ 2\ 1)$$

Recuperação de Situação de Deadlock

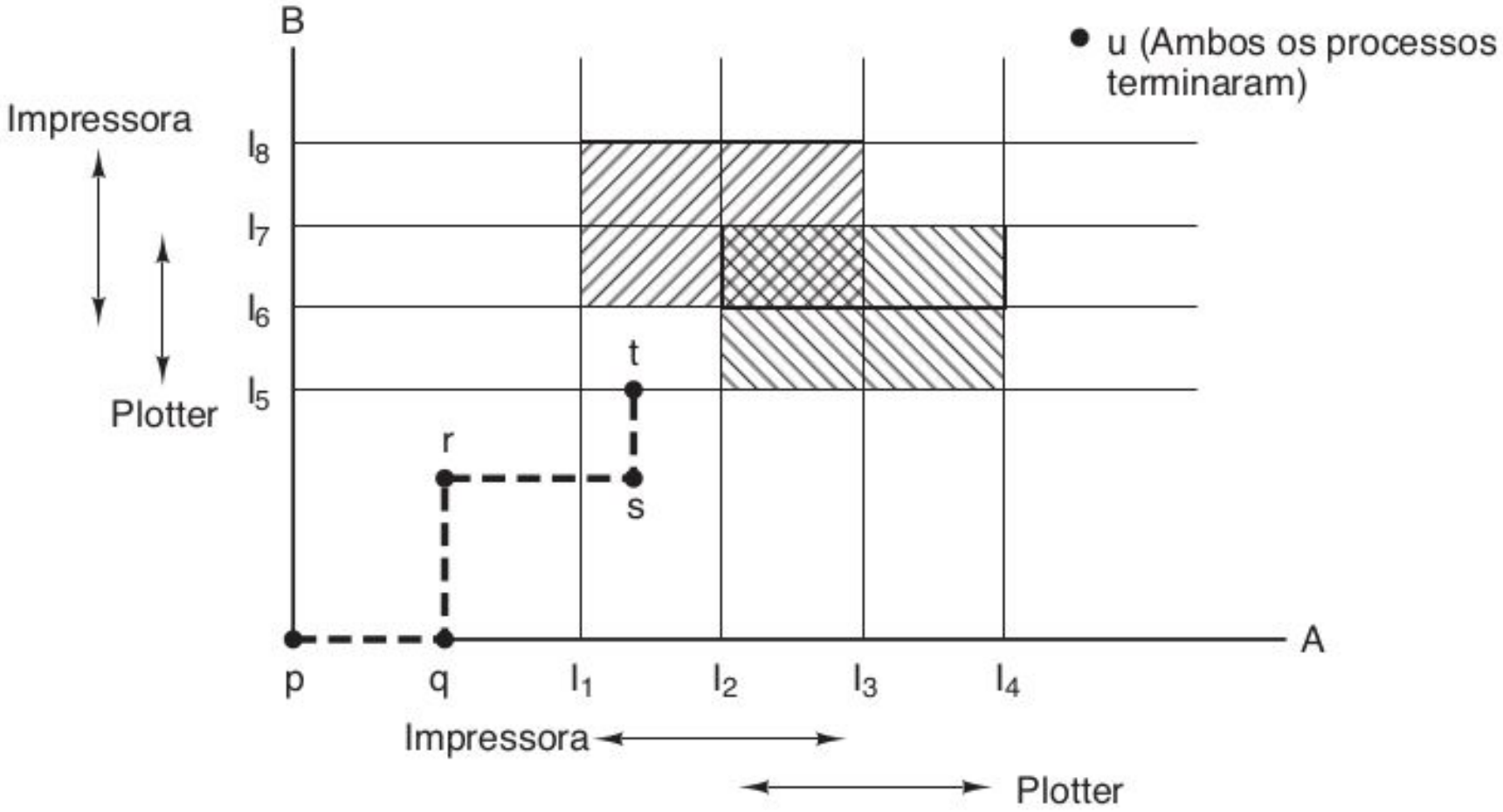
1. **Recuperação por meio de preempção**
   - Retirar o recurso de um processo e dar a outro
   - Não se pode fazer com qualquer recurso
2. **Recuperação por meio de reversão de estado**
   - *Checkpointing*: imagem da memória do processo
   - Guarda o estado em um arquivo-imagem
     - Estados dos recursos
3. **Recuperação por meio de eliminação de processo**
   - Matar um processo
     - Do ciclo
     - Outro que tenha recursos

Evitando Deadlock

- **Presumimos que os recursos são requisitados todos de uma vez**
  - Nem todos os sistemas são assim
- **O sistema deve ser capaz de decidir**
  - Liberar ou não recursos
- **Existe um algoritmo que sempre possa evitar impasses fazendo a escolha certa?**
  - A resposta é sim com restrições
    - Tem que saber alguns informações
  - Se ele tivesse como poder prever o futuro... Hein...

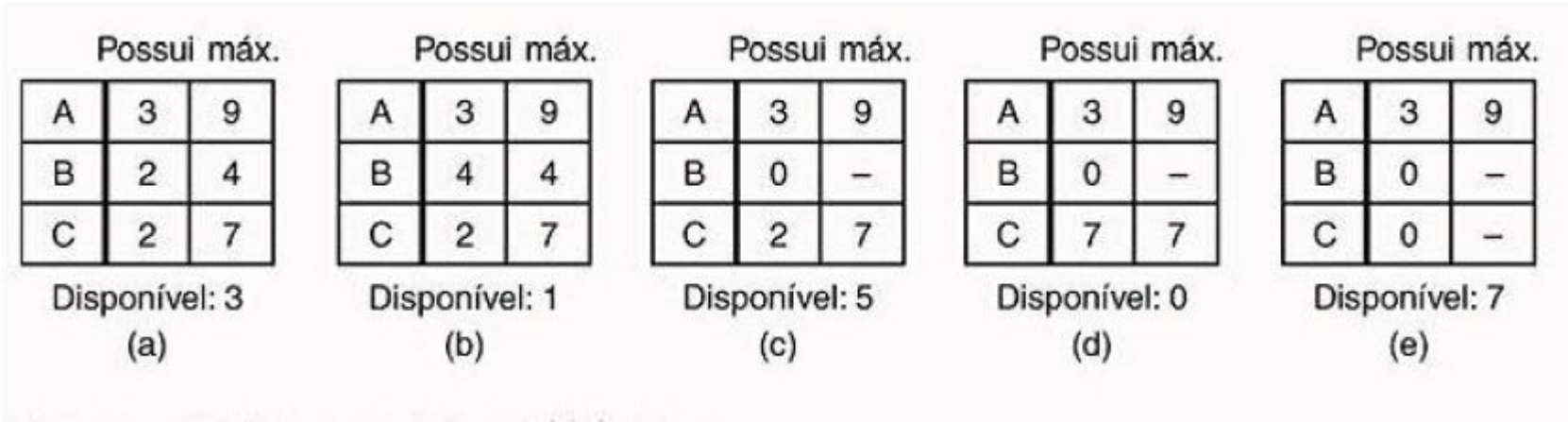
Evitando Deadlock

- Trajetória de recursos



Evitando Deadlock

- Estados **Seguros** e **Inseguros**
  - Usar informações de E, A, C e R
- Estado **Seguro**
  - Se não está em situação de impasse e
  - Se Existe alguma ordem de escalonamento
    - Todo processo possa ser executado até o final
  - (a) é um estado seguro



	Possui	máx.
A	3	9
B	2	4
C	2	7

Disponível: 3
(a)

	Possui	máx.
A	3	9
B	4	4
C	2	7

Disponível: 1
(b)

	Possui	máx.
A	3	9
B	0	–
C	2	7

Disponível: 5
(c)

	Possui	máx.
A	3	9
B	0	–
C	7	7

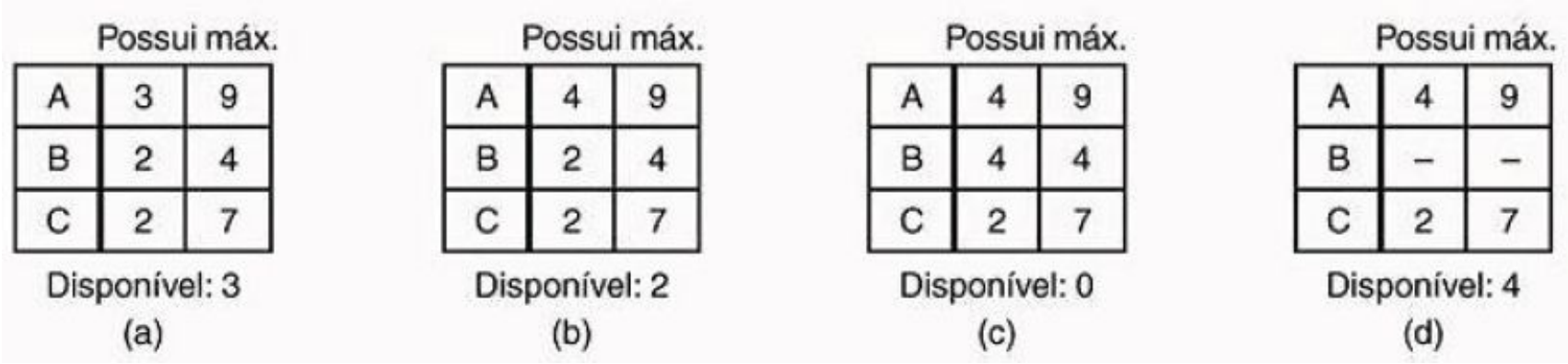
Disponível: 0
(d)

	Possui	máx.
A	3	9
B	0	–
C	0	–

Disponível: 7
(e)

Evitando Deadlock

- Estado **Inseguro**
  - Não é uma situação de impasse
  - O sistema não pode garantir que todos os processos terminarão
  - (b) é inseguro



	Possui	máx.
A	3	9
B	2	4
C	2	7

Disponível: 3
(a)

	Possui	máx.
A	4	9
B	2	4
C	2	7

Disponível: 2
(b)

	Possui	máx.
A	4	9
B	4	4
C	2	7

Disponível: 0
(c)

	Possui	máx.
A	4	9
B	–	–
C	2	7

Disponível: 4
(d)

Prevenção de Deadlock

- Atacando a condição de **Exclusão Mútua**
- Alguns recursos (ex. dispositivo impressora) podem ser tratados com um gerente (spool)
  - Apenas o gerente utiliza o dispositivo
  - Eliminando **deadlock**
- Nem todos os dispositivos podem ser tratados desta forma
- Princípio:
  - Evitar atribuir recurso quando desnecessário
  - Quanto menos processos reclamando o recurso melhor

Prevenção de Deadlock

- Atacando a condição de **Posse e Espera**
  - Impedir que processos esperem por recursos
  - Exigir que todos os processos requisitem todos os recursos
    - Se algum recurso estiver alocado o processo terá que esperar
- Problema
  - O processo não tem conhecimento dos recursos necessários antes de começar
- Variação
  - Processos devem desistir dos recursos
  - Então requisitar os imediatamente necessários

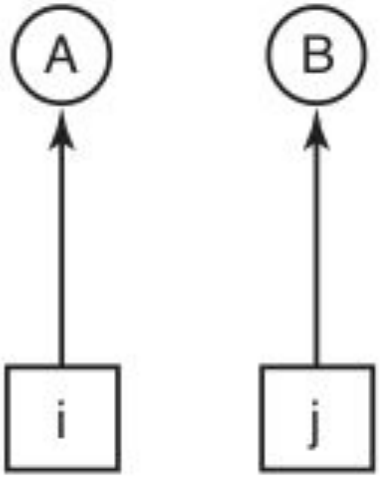
Prevenção de Deadlock

- Atacando a condição de **Não Preempção**
- Se um processo que possui alguns recursos requisita outro recurso que não pode ser imediatamente alocado para ele, então todos recursos mantidos são liberados.
- Solução inviável
- Considere o caso de um processo liberando impressora
  - Trabalho está no meio do caminho
  - Forçosamente retiram a impressora

Prevenção de Deadlock

- Atacando a condição de **Espera Circular**
- Impõe o ordenamento total de todos os tipos de recursos e requer que cada processo requisite recursos em ordem crescente de enumeração.

1. Máquina de fotolito
2. Impressora
3. Plotter
4. Unidade de fita
5. Unidade de Blu-ray



- Se i>j, Então A não tem permissão de requisitar j
- Se i<j, então B não tem permissão de requisitar i

Prevenção de Deadlock

Condição	Abordagem contra impasses
Exclusão mútua	Usar spool em tudo
Posse e espera	Requisitar todos os recursos necessários no início
Não preempção	Retomar os recursos alocados
Espera circular	Ordenar numericamente os recursos

UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE RUSSAS

Rus0013 - Sistemas Operacionais

Aula 08: Deadlock(Impasses)

Professor Pablo Soares

2022.2