Universidade Federal de Uberlândia
Faculdade de Computação
Sistemas Operacionais

Impasse
(Deadlock)

Prof. Dr. Marcelo Zanchetta do Nascimento

Roteiro

- Conceito de Impasse (Deadlock);
- Recursos;
- Condições de ocorrência;
- Estratégias para tratar o Deadlock;
- Prevenção de Deadlock;
- Leituras Sugeridas
- Exercícios

Deadlock

- Na ausência de sincronização entre processos concorrentes pode ocorrer um **deadlock;**

- É o **congestionamento de requisições de recursos** no âmbito de todo o sistema, que começa quando 2 ou mais processos **são colocados em espera** até que o recurso se torne disponível;

- Pode requerer intervenção externa:

  – Força a ter atitudes drásticas, como por exemplo, provocar manualmente o término do processo.

Deadlock

Existem duas formas de uso de recursos computacionais:

- **Preemptivo:** recurso pode ser retirado do processo sem prejuízo ao sistema;
- **Exemplo:** Memória Principal
  1) Quando 2 processos que solicitam a impressão (sistema time-sharing);
  2) Processo A obtém a impressora;
  3) O escalonador retira da CPU o processo A;
  4) Processo B tenta obter a impressora;
  5) Situação de deadlock;
  - **Solução:** envie o processo B para disco e carrega o processo A na memória ("elimina o deadlock");
  - Nenhum deadlock ocorre nessa situação.

Deadlock

- **Não-preemptivo:** o recurso não pode ser retirado do processo. Isso irá causar prejuizo ao sistema;
- Unidade de Blu-ray:
  - Processo A começou a gravar em um Blu-ray;
  - Retira repentinamente o recurso do processo A;
  - O gravador de Blu-ray é alocado a um outro processo;
  - Resultado: um disco de Blu-ray bagunçado;
  - Um deadlock ocorre com esse tipo de recurso.

Deadlock

- Como é a sequência de eventos para utilização de um recurso compartilhado?
  - Requisição do recurso;
  - Utilização do recurso;
  - Liberação do recurso;
- Se não estiver disponível, o que pode ocorrer?
  - Processo que requisitou o recurso **fica bloqueado** até que o recurso seja liberado;
  - Processo que requisitou o recurso falha e depois de um certo tempo tenta novamente requisitar o recurso;

Deadlock - Na aquisição de recursos

```
typedef int semaphore;
semaphore recurso_1;
semaphore recurso_2;

void processoA(void){
   down(&recurso_1);
   down(&recurso_2);
   Usar_ambos_itens( );
   up(&recurso_2);
   up(&recurso_1);
  }
void processoB(void){
   down(&recurso_1);
   down(&recurso_2);
   Usar_ambos_itens( );
   up(&recurso_2);
   up(&recurso_1);
  }
```

Possibilidade de Impasse

```
typedef int semaphore;
semaphore recurso_1;
semaphore recurso_2;

void processoA(void){
   down(&recurso_1);
   down(&recurso_2);
   Usar_ambos_itens( );
   up(&recurso_2);
   up(&recurso_1);
  }
void processoB(void){
   down(&recurso_2);
   down(&recurso_1);
   Usar_ambos_itens( );
   up(&recurso_1);
   up(&recurso_2);
  }
```

Deadlock

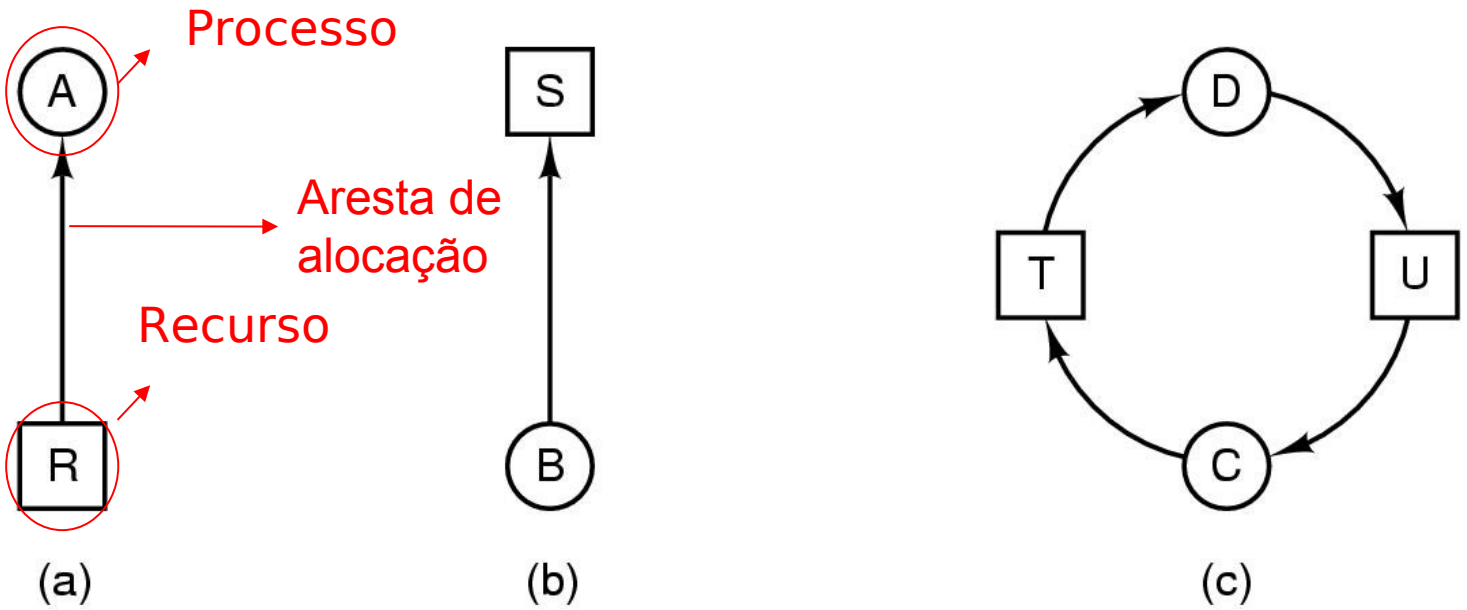
- Situações em um SO, que pode ocorrer um deadlock:
  - Exclusão mútua: um recurso está sendo utilizado por algum processo ou está disponível;
  - Uso e espera (*hold and wait*): os processos que já possuem algum recurso podem requer outros recursos para finalizar;

Deadlock

- Condições para ocorrer um deadlock:
  - Não-preempção: recursos já alocados não podem ser retirados do processo que o alocou; somente o processo que alocou-os pode liberá-los;
  - Espera Circular: Deve existir um encadeamento circular de dois ou mais processos; cada um deles encontra-se à espera de um recursos que está sendo usado pelo membro seguinte dessa cadeia (monopoliza o recurso).

Modelagem de Deadlock

- Holt (1972) demonstrou que as condições de deadlock podem ser visualizadas por meio de **grafos direcionados;**

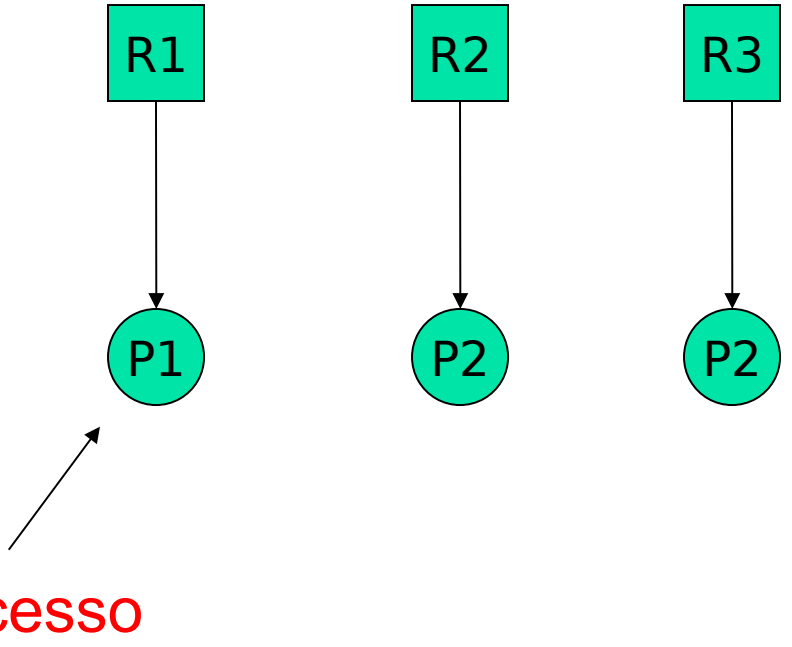


a) **Recurso R** (quadrado) alocado ao **Processo A** (círculo)
b) **Processo B** requisita **Recurso S**
c) **Deadlock – ciclo "C"- T -"D"- U - "C"**

Modelagem de Deadlocks

- Cenário 1 – Grafo de Recursos

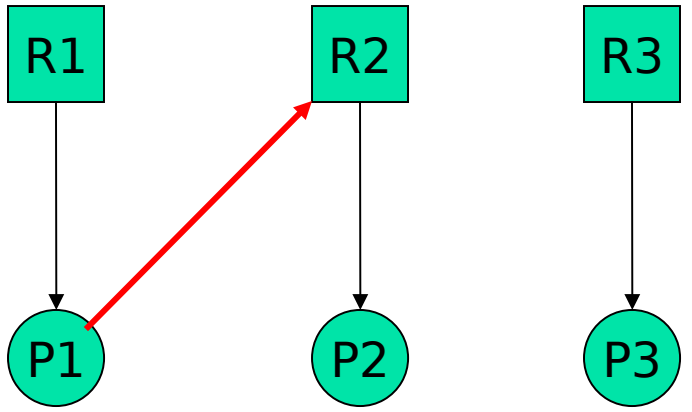
Tempo	Ação
1	P1 requisita e obtém R1
2	P1 libera R1
3	P2 requisita e obtém R2
4	P2 libera R2
5	P3 requisita e obtém R3
6	P3 libera R3



Modelagem de Deadlocks

- Cenário 2: Processos fazem operações de E/S e SO utiliza o algoritmo de escalonamento RR.

Tempo	Ação
1	P1 requisita e obtém R1
2	P2 requisita e obtém R2
3	P3 requisita e obtém R3
4	P1 requisita R2
5	P2 requisita R3
6	P3 requisita R1

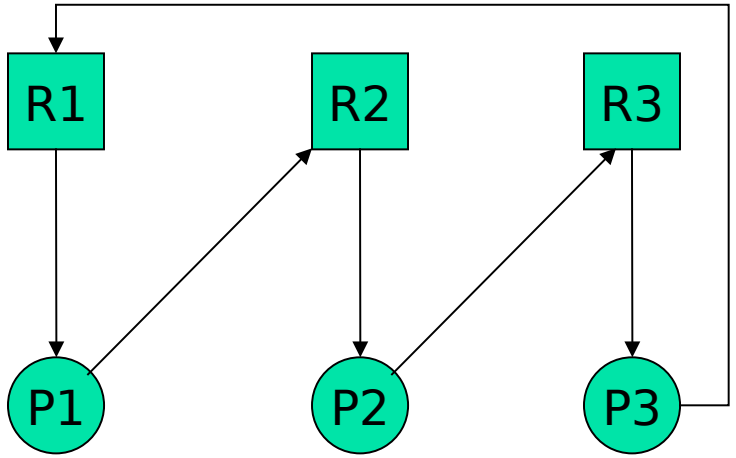


Bloqueado

Modelagem de Deadlocks

- Cenário 2: Algoritmo de escalonamento RR

Tempo	Ação
1	P1 requisita e obtém R1
2	P2 requisita e obtém R2
3	P3 requisita e obtém R3
4	P1 requisita R2
5	P2 requisita R3
6	P3 requisita R1



Impasse

- Como o SO pode resolver esse problema?
- A ordem de execução baseada no algoritmo de escalonamento pode ser a solução?
- Se P2 não alocar recurso?

Tratamento de Deadlock

- Quatro estratégias para tratar deadlock*:*
  - Ignorar o problema;
  - Detectar e recuperar o problema;
  - Evitar dinamicamente o problema:
    - alocação cuidadosa de recursos;
  - Prevenir o problema por meio da não satisfação de uma das quatro condições citadas anteriormente;

Tratamento de Deadlocks

- **Ignorar o problema (Algoritmo do Avestruz):**
  - "Enterre sua cabeça na areia e finja que nada está acontecendo";
  - Maioria dos SOs (Windows e Unix) ignoram o problema;
  - Supondo que a maior parte dos usuários preferem um deadlock ocasional;
  - A uma regra que restrinja cada usuário somente a um processo.
    - **Problema:** Custo é alto e implica em restrições não convencionais de processos.

Tratamento de Deadlock

- Quatro estratégias para tratar deadlock*:*
  - Ignorar o problema;
  - Detectar e recuperar o problema;
  - Evitar dinamicamente o problema:
    - alocação cuidadosa de recursos;
  - Prevenir o problema por meio da não satisfação de uma das quatro condições citadas anteriormente;

Tratamento de Deadlock

Detectar e Recuperar o Problema:

- Técnica que permite que o deadlock ocorra e tenta detectar as causas e solucionar a situação;
- Algoritmos:
  - Detecção com um recurso de cada tipo;
  - Detecção com vários recursos de cada tipo;
  - Recuperação por meio de preempção;
  - Recuperação por meio de *rollback* (volta ao passado);
  - Recuperação por meio de eliminação de processos.

Tratamento de Deadlock

- Detecção com um recurso de cada tipo:
  - Tem um recurso de cada tipo: scanner, ploter e impressora;
  - Se houver ciclo existe possibilidade de *deadlock*;

Situação: com 7 processos e 6 recursos

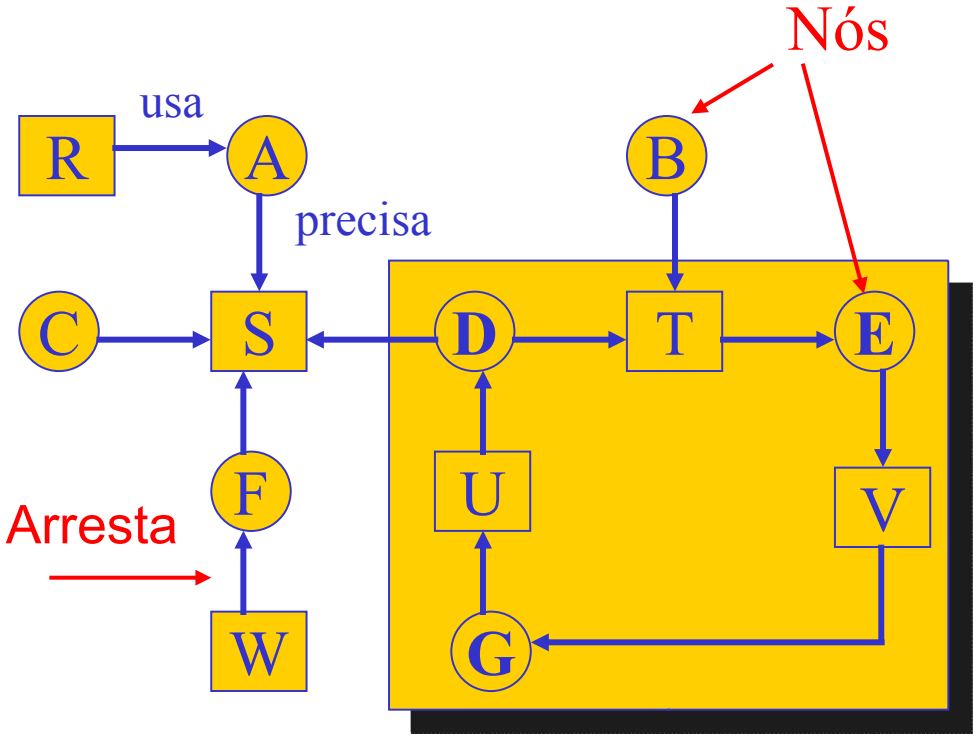
PA usa R e precisa de S;
PB precisa de T;
PC precisa de S;
PD usa U e precisa de S e T;
PE usa T e precisa de V;
PF usa W e precisa de S;
PG usa V e precisa de U;

Essa situação deve ser modelada: Grafo de recursos

Tratamento de Deadlock

- **Detecção com um recurso de cada tipo:**
  - Resposta através da construção de um grafo;
  - Se houver ciclo existe possibilidade de *deadlock*;

Processos: A-G
Recursos: R-W



Sistema: 7 processos

Situação:
PA usa R e precisa de S;
PB precisa de T;
PC precisa de S;
PD usa U e precisa de S e T;
PE usa T e precisa de V;
PF usa W e precisa de S;
PG usa V e precisa de U;

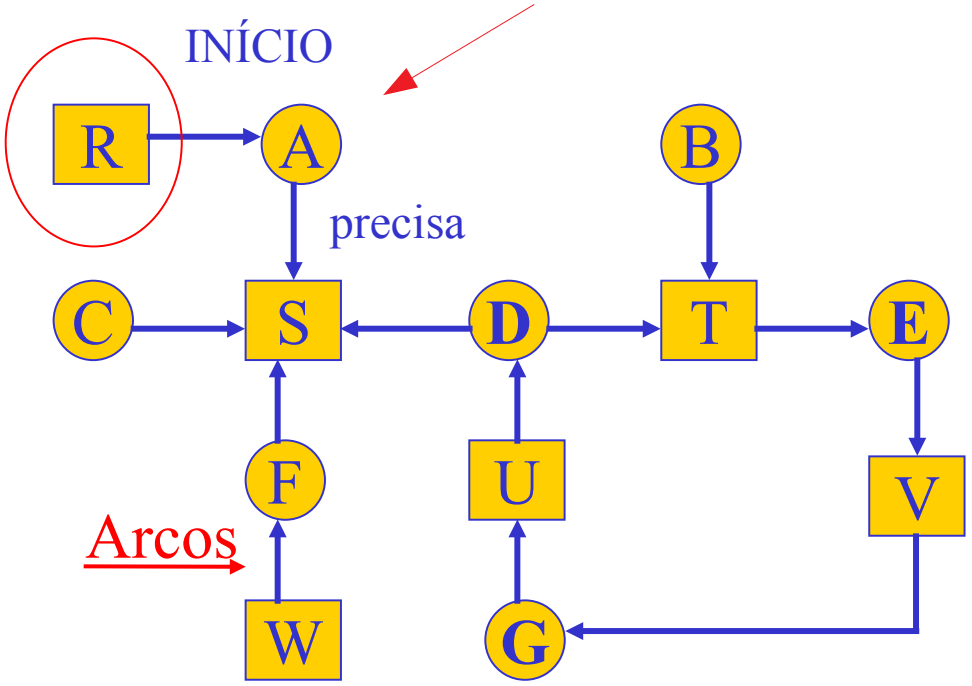
Pergunta: Há possibilidade de *deadlock*?

Tratamento de Deadlock

- **Algoritmo (aplicação):** começa utilizando uma lista L
  - Execução a partir do recurso R:
  - R→A, B, C, S, D, T, E, F (se houver ciclo irá parar);

- Início R : L = [R, A], L = [R, A, S] => S não tem arco de saída (retorna);

- Início A: L=[A, S] S não tem arco de saída (retorna);

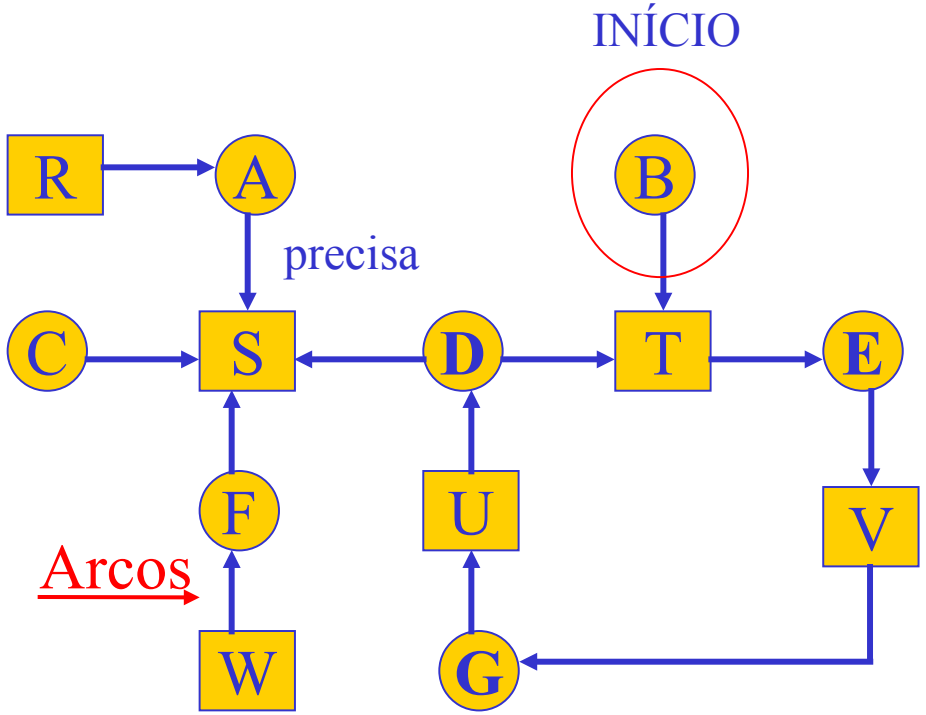


Tratamento de Deadlock

- **Algoritmo (aplicação):** começa utilizando uma lista L
  - Execução a partir de B:

- Início B: L = [B, T, E, V, G, U, D] => escolher S vamos para um nó sem saída e retornará em D;

- Caso contrário (**gera ciclo**): L=[B,T,E,V,G,U,D,T].



Tratamento de Deadlock

Detectar e Recuperar o Problema:

- Técnica permite que os deadlocks ocorram e tenta detectar as causas e solucionar a situação;
- Algoritmos:
  - Detecção com um recurso de cada tipo;
  - Detecção com vários recursos de cada tipo;
  - Recuperação por meio de preempção;
  - Recuperação por meio de *rollback* (volta ao passado);
  - Recuperação por meio de eliminação de processos.

Tratamento de Deadlock

- Detecção com vários recursos (baseado em matrizes)
  - Classes diferentes de recursos:
    - Vetor de recursos existentes (E):
    - Classe1 = unidade de fita -> E_1=2 - duas unidades de fita;
    - Vetor de recursos disponíveis (A):
    - Se as unidades de fita estiverem alocadas, A_1= 0;
  - Duas matrizes:
    - C: matriz de alocação atual;
      - C_{ij}: número de instâncias do recurso j entregues ao processo i;
    - R: matriz de requisições;
      - R_{ij}: número de instâncias do recurso j que o processo i precisa;

Tratamento de Deadlock

4 unidades de fita;
2 *plotter*;
3 impressoras;
1 unidade de Blu-ray

Recursos existentes
E = (4 2 3 1)
UF P I UBR

Recursos disponíveis
A = (**2 1 0 0**)
UF P I UBR

Matriz de requisição

UF P I UBR

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Matriz de alocação

UF P I UBR

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Recursos

<u>Três processos</u>:

P_1 usa 1 impressora;

P_2 usa 2 unidades de fita e 1 de Blue-Ray;

P_3 usa 1 *plotter* e 2 impressoras;

Cada processo precisa de outros recursos conforme matrix R;

Tratamento de Deadlock

4 unidades de fita;
2 *plotter*;
3 impressoras;
1 unidade de Blu-ray

Requisições (satisfazer a condição):

P_1 requisita 2 unidades de fita e 1 Blu-Ray (não pode atender);

P_2 requisita 1 unidade de fita e 1 impressora (não pode atender);

P_3 requisita 2 unidades de fita e 1 *plotter*;

P_3 pode rodar ?
Após rodar P3 => A = (2 2 2 0)

Recursos existentes

UF P I UBR

E = (4 2 3 1)

Recursos disponíveis

UF P I UBR

A = (2 1 0 0)

Matriz de alocação

UF P I UBR

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{0} \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Matriz de requisição

UF P I UBR

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Tratamento de Deadlock

4 unidades de fita;
2 *plotter*;
3 impressoras;
1 unidade de Blu-Ray

Requisições:
P_1 requisita duas unidades de fita e um Blu-ray;
P_2 requisita uma unidade de fita e uma impressora;

Recursos existentes

UF P I UBR

E = (4 2 3 1)

Recursos disponíveis

UF P I UBR

A = (2 1 0 0) ->(resultado após P3)

A = (2 2 2 0) -> P_2 pode rodar

A = (4 2 2 1) -> resultado após P2

Matriz de alocação

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ \mathbf{3} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Matriz de requisição

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow \mathbf{P_2} \\ \leftarrow P_3 \end{matrix}$$

Tratamento de Deadlock

4 unidades de fita;
2 *plotter*;
3 impressoras;
1 unidade de Blu-Ray

Requisição:

P_1 requisita 2 unidades de fita e 1 Blu-Ray;

Recursos existentes

UF P I UBR

E = (4 2 3 1)

Recursos disponíveis

A = (4 2 2 1) → resultado após P2

P_1 pode rodar ?

Matriz de alocação

$$C = \begin{bmatrix} \mathbf{2} & \mathbf{0} & \mathbf{1} & \mathbf{1} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Matriz de requisição

$$R = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow \mathbf{P_1} \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Tratamento de Deadlock

Ao final da execução, temos:

4 unidades de fita;
2 *plotters*;
3 impressoras;
1 unidade de Blu-Ray

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (4 2 3 1)

Matriz de alocação

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Matriz de requisições

$$R = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Tratamento de Deadlock

4 unidades de fita;
2 *plotters*;
3 impressoras;
1 unidade de Blu-Ray

Requisições:

TEMPO DE CPU

Deadlock: P_3 requisita duas unidade de fita, uma impressora e uma unidade de blu-ray;

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
UF P I UBR
A = (2 1 0 0)

Matriz de alocação

UF P I UBR

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Matriz de requisição

UF P I UBR

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Tratamento de Deadlock

Detectar e Recuperar o Problema:

- Técnica permite que o deadlock ocorra e tenta detectar as causas e solucionar a situação;
- Algoritmos:
  - Detecção com um recurso de cada tipo;
  - Detecção com vários recursos de cada tipo;
  - Recuperação por meio de preempção;
  - Recuperação por meio de *rollback* (volta ao passado);
  - Recuperação por meio de eliminação de processos.

Recuperação de Deadlock

Se localizado o Impasse. O que deve ser feito

Recuperação de Deadlock

- Por meio de preempção: possibilidade de retirar temporariamente um recurso de seu atual dono (processo) e entregá-lo a outro processo;
- Por meio de revisão de estado: recursos alocados a um processo são armazenados em arquivos de verificação.
  - Quando ocorre um deadlock, os processos voltam ao estado no qual estavam antes do deadlock;

Recuperação de Deadlocks

- Por meio de eliminação de processos: processos que estão no ciclo com *deadlock* são retirados do ciclo.
- Os processos não causam efeitos negativos ao sistema;
  - Ex1.: compilação – sem problemas;

Tratamento de Deadlock

- Quatro estratégias para tratar deadlock*:*
  - Ignorar o problema;
  - Detectar e recuperar o problema;
  - Evitar dinamicamente o problema:
    - alocação cuidadosa de recursos;
  - Prevenir o problema por meio da não satisfação de uma das quatro condições citadas anteriormente;

Evitar Deadlocks

- É possível evitar impasse fazendo uma escolha correta?

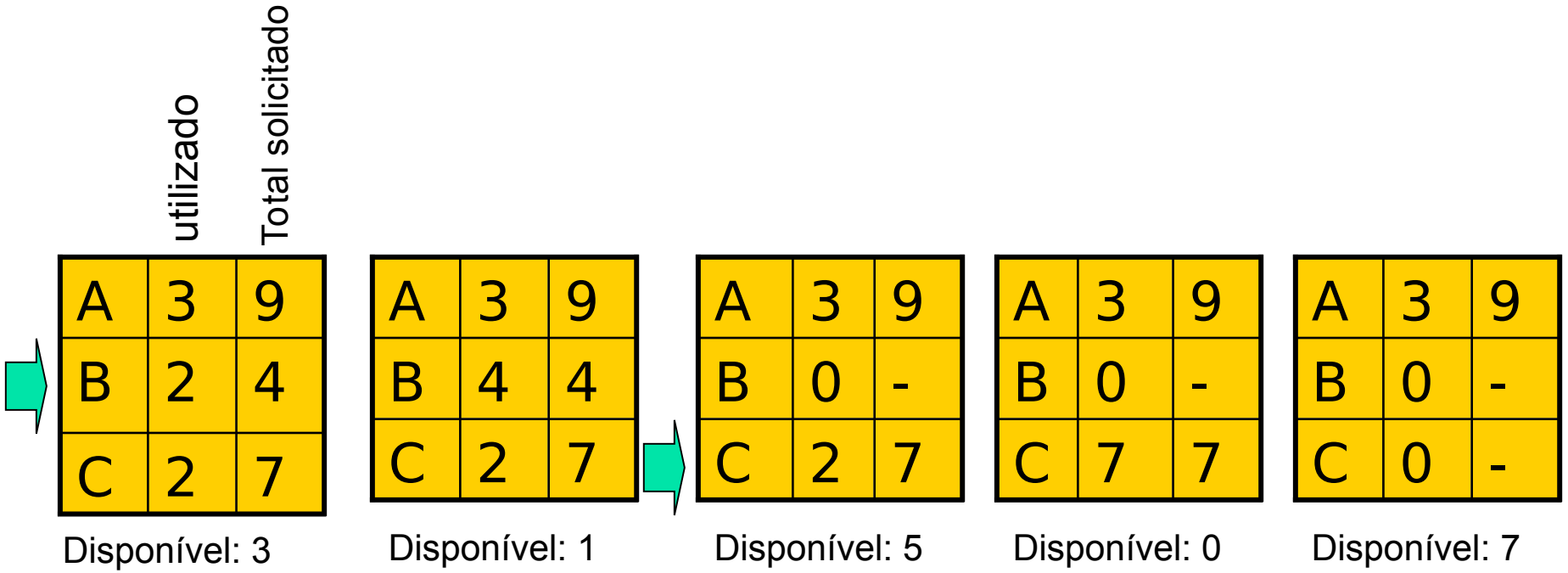
- Evitar dinamicamente o problema:
  - Alocação individual de recursos (Multiprogramação fica comprometida);
  - Utiliza-se de matrizes (alocação de recursos);
  - Escalonamento “cuidadoso”;
  - Trabalhar com Estado: Seguros e Inseguros;

Evitar Deadlocks

- Estados seguros: não provocam *deadlocks* e há uma maneira de atender a todas as requisições pendentes até que seja finalizada normalmente os processos;
  - Deve existir uma ordem de escalonamento na qual todo o processo possa ser executado até a sua conclusão;
- Estados inseguros: podem provocar *deadlock,* mas não necessariamente ocorrerá.

Evitar Deadlocks

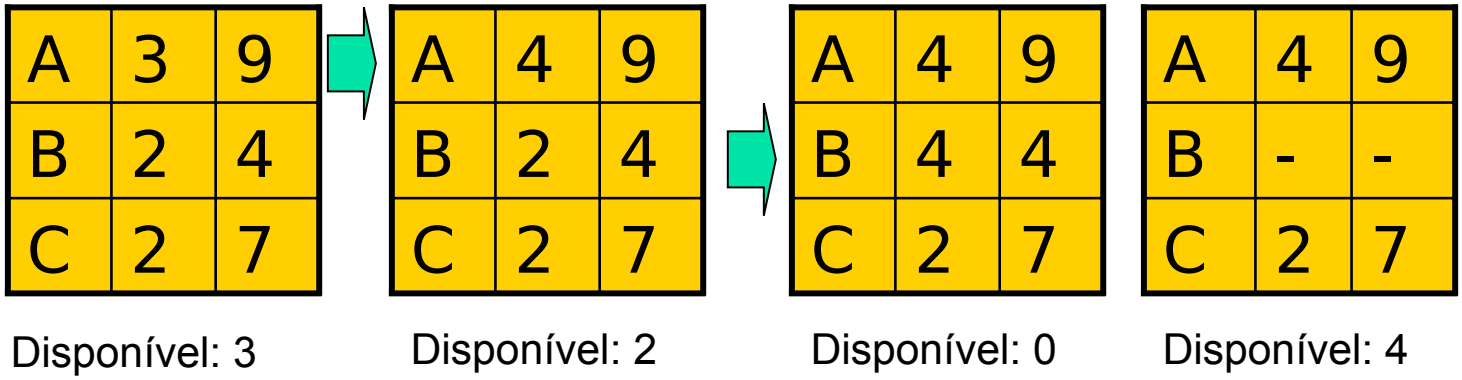
- Seguro: Começa escalonando o processo B



Evitar Deadlocks

- Inseguro (não é deadlock):
  - Solicitará e obterá outro recurso
  - Não há garantia que todos irão terminar

Começa escalonando o processo A – 1 recurso



Evitar Deadlocks

- Algoritmo do Banqueiro:
  - Idealizado por Dijkstra (1965);
  - Segue os seguintes princípios (**analogia**):
    - Nenhum cliente receberá um empréstimo maior do que o capital total do banco.
    - Todos os clientes receberão um limite de crédito ao abrir suas contas.
    - Nenhum cliente poderá ultrapassar esse limite.
    - A soma de todos os empréstimos não poderá ultrapassar o capital total do banco.

Evitar Deadlocks

- Algoritmo do Banqueiro:
  - Considera cada requisição no momento em que ela ocorre verificando se essa requisição leva a um estado seguro;
    - Se sim, a requisição é atendida,
    - Senão, o atendimento é adiado para um outro momento;
  - Premissas por um banqueiro para garantir ou não crédito (recursos) para seus clientes (processos);
  - Nem todos os clientes (processos) precisam de toda a linha de crédito (recursos) disponível.

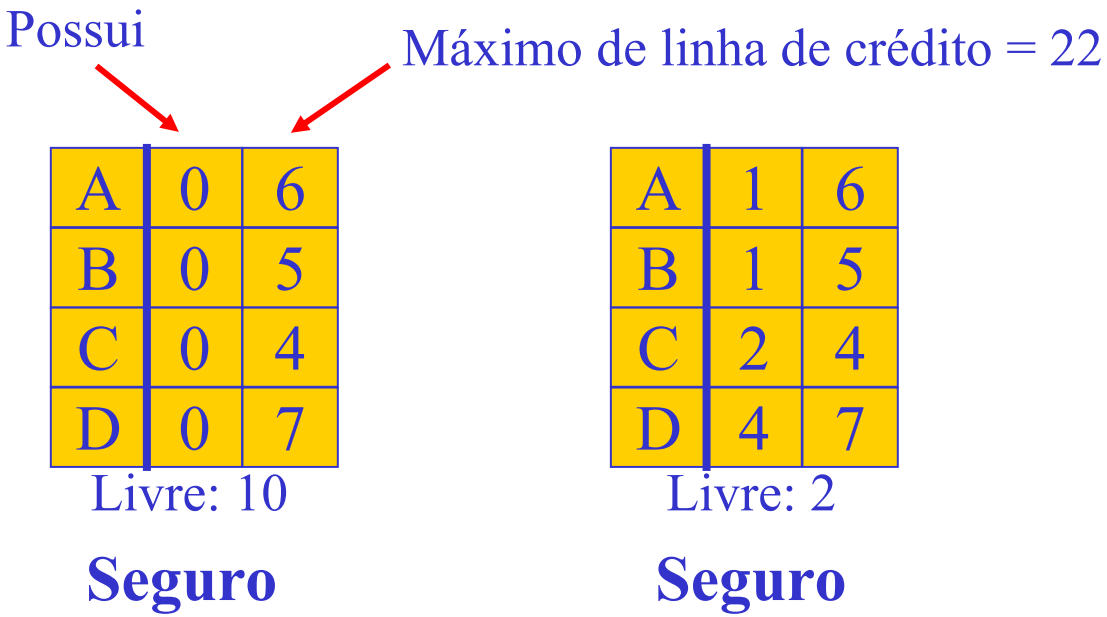
Evitar Deadlocks

- **Algoritmo do Banqueiro para um único tipo de recurso:**

- Solicitações de crédito são realizadas de tempo em tempo;
- *C é atendido e libera 4 créditos, que podem ser usados por B ou D;

Evitar Deadlocks

- Algoritmo do Banqueiro



- Solicitações de crédito são realizadas de tempo em tempo;
- B* é atendido.
- Em seguida, se os outros solicitarem recursos, ninguém poderá ser atendido para finalização do processo;

Evitar Deadlocks

- **Algoritmo do Banqueiro para vários tipos de recursos:**

Processos	Unidade de Fita	Plotters	Impressoras	Unidade de Blu-Ray
A	3	0	1	1
B	0	1	**0**	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Matriz C
Recursos Alocados

UF P I UBR

Existentes → E = (6 3 4 2);
Alocados → P = (5 3 2 2);
Disponíveis → A = (1 0 2 0);

A	1	1	0	0
B	0	1	**1**	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

R = Recursos ainda necessários

Se houver uma requisição de B para 1 impressora, o sistema é seguro.

Pode atender a solicitação de D.

Evitar Deadlocks

- **Algoritmo do Banqueiro**

Alocados → P = (5 3 3 2);
Disponíveis → A = (1 0 1 0);

Processo E solicita impressora
Disponíveis → A = (1 0 **0** 0);

Processos	Unidade de Fita	Plotters	Impressoras	Unidade de Blu-Ray
A	3	0	1	1
B	0	1	1	0
C	1	1	1	0
D	1	1	0	1
E	0	0	**1**	0

C = Recursos Alocados

A	1	1	0	0
B	0	1	0	2
C	3	1	0	0
D	0	0	1	0
E	2	1	**0**	0

R = Recursos ainda necessários

- Se E solicitar a alocação de uma Impressora?
- Deadlock pode ocorrer
- Solução: Adiar a requisição de E por alguns instantes;

Inseguro: negar

Evitar Deadlocks

- Algoritmo do Banqueiro:
  - Desvantagens
    - Pouco utilizado, pois é difícil saber quais recursos serão necessários;
    - O número de processos é dinâmico e pode variar constantemente tornando o algoritmo custoso;
  - Vantagem
    - Teoricamente: o algoritmo é ótimo;

Tratamento de Deadlock

- Quatro estratégias para tratar deadlock*:*
  - Ignorar o problema;
  - Detectar e recuperar o problema;
  - Evitar dinamicamente o problema:
    - alocação cuidadosa de recursos;

- Prevenir o problema por meio da não satisfação de uma das quatro condições citadas anteriormente;

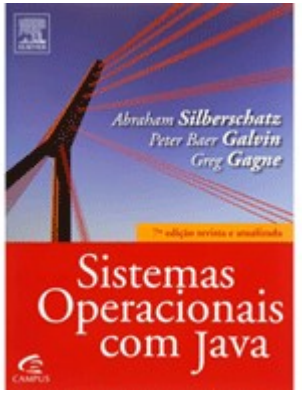
Previnir Deadlock

- Voltar para as condições estabelecidas por Coffman et al. (1971):

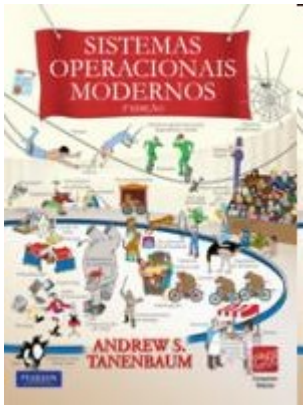
Condição	Abordagem
Exclusão Mútua	Alocar todos os recursos usando a técnica de *spooling* com *daemon* – processo background
Uso e Espera	Requisitar todos os recursos inicialmente antes do processamento.
Não-preempção	Retirar recursos dos processos, por meio de virtualização de alguns recursos
Espera Circular	Processo tem permissão de possuir somente um recurso de cada vez.

Leituras Sugeridas

- Silberschatz, A., Galvin, P. B. Gagne, G. Sistemas Operacionais com Java. 7º edição. Editora Campus, 2008.
  - Capítulo 6



- TANENBAUM, A. Sistemas Operacionais Modernos. Rio de Janeiro: Pearson, 3 ed. 2010
  - Capítulo 6



Exercício

Em um sistema em que os dispositivos são todos do mesmo tipo e utilizando as definições apresentada sobre o algoritmo do Banqueiro responda às seguintes perguntas:

a) Determine as requisições restantes para cada programa no sistema.

b) Determine se cada sistema é seguro ou inseguro.

c) Se o sistema tiver em estado seguro, relacione a sequência de requisições e liberações que possibilitará a execução completa de todos os processos.

d) Se o sistema estiver em estado inseguro, mostre como é possível ocorrer um impasse.

Exercício

i) Sistema A tem 12 dispositivos; apenas 1 está disponível.

Número do programa	Dispositivos alocados	Máximo de requisições	Requisições restantes
1	5	6	
2	4	7	
3	2	6	
4	0	2	

ii) Sistema B tem 14 dispositivos; apenas 2 estão disponíveis.

Número do programa	Dispositivos alocados	Máximo de requisições	Requisições restantes
1	5	8	
2	3	9	
3	4	8	