Sistemas Operacionais

Deadlocks

Norton Trevisan Roman
Marcelo Morandini
Jó Ueyama

Apostila baseada nos trabalhos de Kalinka Castelo Branco, Antônio Carlos Sementille, Paula Prata e nas transparências fornecidas no site de compra do livro "Sistemas Operacionais Modernos"

Deadlocks

- Dispositivos e recursos são compartilhados a todo momento:
  - Impressora, disco, arquivos, posições nas tabelas internas do sistema etc.
  - S.Os. devem ter a habilidade de garantir a um processo acesso exclusivo a certos recursos
    - Ex: entrada em tabela do sistema de arquivos
- Deadlock:
  - Processos ficam parados sem possibilidade de poderem continuar seu processamento;
    - Ocorrem tanto em recursos de hardware quanto software

Deadlocks

Deadlocks

- Recursos:
  - Deadlocks ocorrem quando processos obtêm acesso exclusivo a recursos.
    - Tanto de hardware (ex: HD) quanto software (entrada em base de dados)
  - Podem ser:
    - Preemptivos: podem ser retirados do processo sem prejuízos;
      - Memória;
      - CPU (como no escalonamento);

Deadlocks

- Recursos:
  - Podem ser:
    - Não-preemptivos: não podem ser retirados do processo, pois causam prejuízos;
      - Criar CD-ROM;
      - Unidades de fita;
      - impressora
    - Deadlocks ocorrem em geral com recursos não-preemptivos
      - Normalmente, com preemptivos, são resolvidos pela realocação de recursos de um processo a outro

Deadlocks

- Eventos necessários para uso de recursos:
  - Requisição do recurso;
  - Utilização do recurso;
  - Liberação do recurso;

  } Pode ser feito via mutexes
- Se o recurso requerido não está disponível, duas situações podem ocorrer:
  - Processo que requisitou o recurso fica bloqueado até que o recurso seja liberado, ou;
  - Processo que requisitou o recurso falha, com uma mensagem de erro, cabendo a ele esperar um pouco e tentar novamente

Deadlock

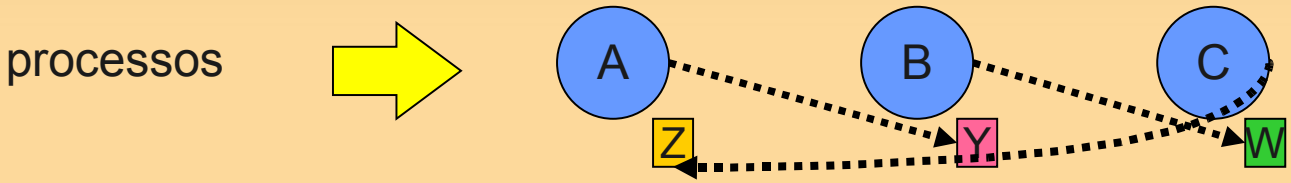
- Definição formal:
  - Um conjunto de processos está em deadlock se cada processo no conjunto estiver esperando por um evento que somente outro processo no conjunto pode causar.
- Muitas vezes, um processo espera por um recurso que outro processo nesse conjunto detém.
  - Deadlock de recurso

Deadlocks

- Quatro condições devem ocorrer para que um deadlock exista:
  - Exclusão mútua: um recurso só pode estar alocado para um processo em um determinado momento;
  - Uso e espera (hold and wait): processos que já possuem algum recurso podem requerer outros recursos;
  - Não-preempção: recursos já alocados não podem ser retirados do processo que os alocou; somente o processo que alocou os recursos pode liberá-los;
  - Espera Circular: um processo espera por recursos alocados a outro processo, em uma cadeia circular

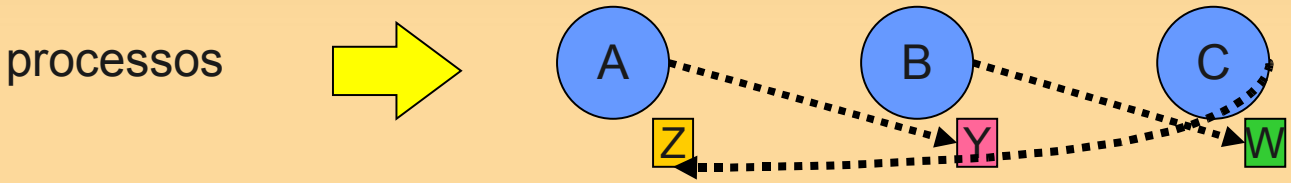
Deadlocks

- Espera circular por recursos.
  - Exemplo:
    - O processo “A” espera pelo processo “B”, que espera pelo processo “C”, que espera pelo processo “A”.



Deadlocks

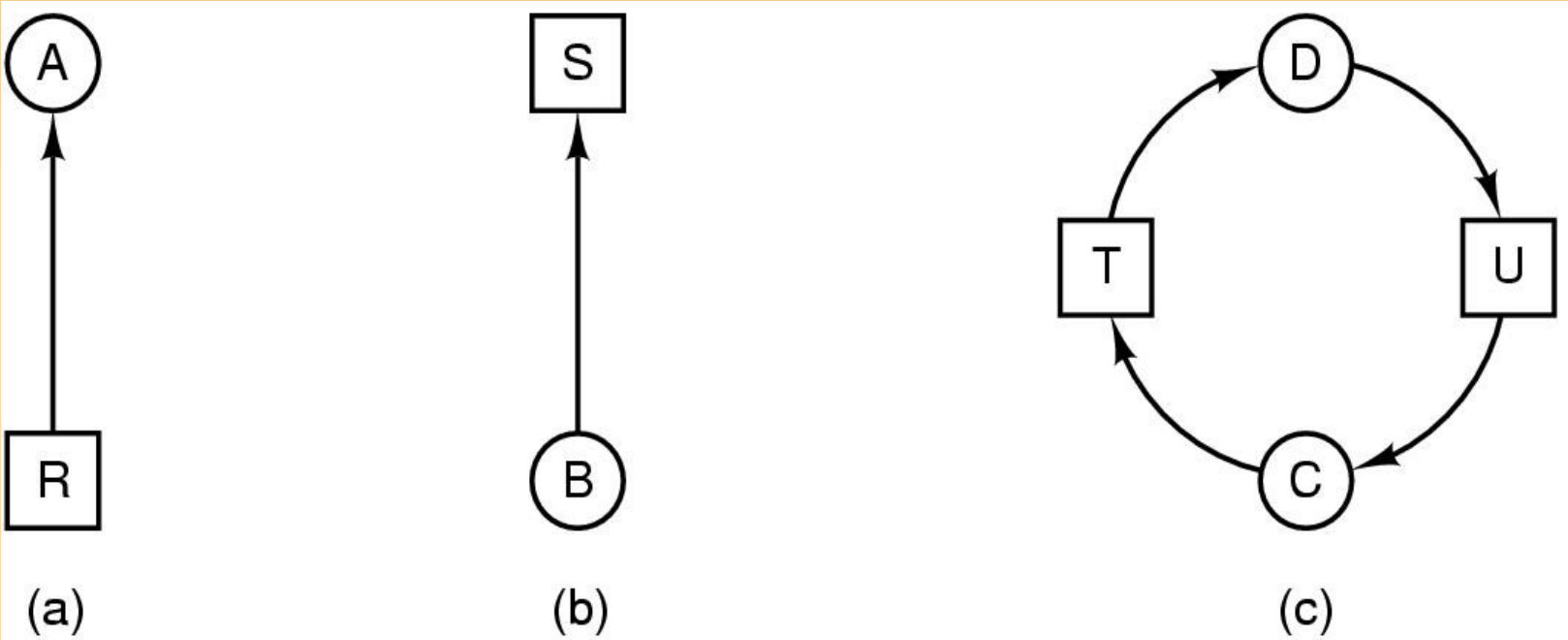
- Espera circular por recursos.
  - Exemplo:
    - O processo “A” espera pelo processo “B”, que espera pelo processo “C”, que espera pelo processo “A”.



- Geralmente, deadlocks são representados por grafos dirigidos, a fim de facilitar sua detecção, prevenção e recuperação
  - Ocorrência de ciclos pode levar a um deadlock;

Deadlocks

- Grafos de alocação de recursos:
  - 2 tipos de nós: processos e recursos



(a) **Recurso R** está alocado ao **Processo A**
(b) **Processo B** requisita **Recurso S** (está bloqueado, esperando pelo recurso)
(c) ***Deadlock***

Deadlocks

- Quatro estratégias para tratar deadlocks:
  - 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;

Deadlocks

- Ignorar o problema.
  - Comparar a freqüência de ocorrência de deadlocks com a freqüência de outras falhas do sistema.
    - Falhas de hardware, erros de compiladores, erros do Sistema Operacional, etc.
  - Se o esforço em solucionar o problema for muito grande em relação a freqüência com que o deadlock ocorre, ele pode ser ignorado.
  - Algoritmo do AVESTRUZ:
    - Freqüência do problema (e quão sério ele é);
    - Alto custo – estabelecimento de condições para o uso de recursos;

Deadlocks

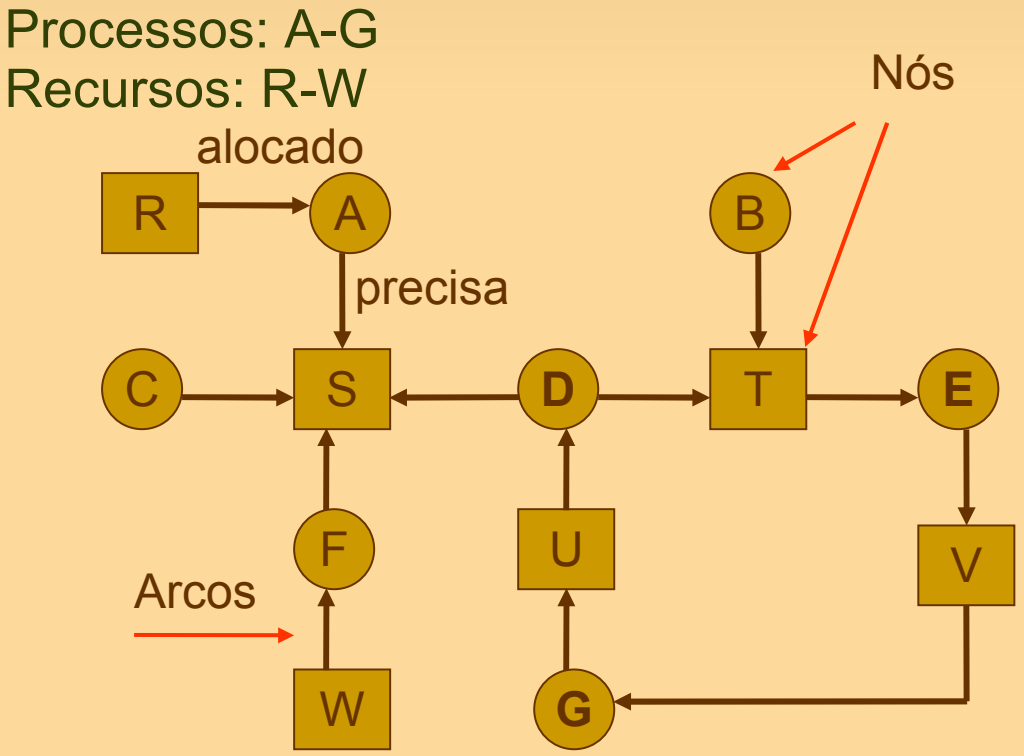
- Detectar e Recuperar o problema:
  - Processos estão com todos os recursos alocados;
  - Procedimento: Permite que os deadlocks ocorram, tenta detectar as causas e solucionar a situação;
  - Algoritmos:
    - Detecção com um recurso de cada tipo;
    - Detecção com vários recursos de cada tipo;
    - Recuperação por meio de preempção;
    - Recuperação por meio de rollback (volta ao passado);
    - Recuperação por meio de eliminação de processos;

Deadlocks

- Detecção com um recurso de cada tipo:
  - Construa o grafo de alocação de recursos;

Situação:
A detém R e quer S;
B não detém nada e quer T;
C não detém nada quer S;
D detém U e quer S e T;
E detém T e quer V;
F detém W e quer S;
G detém V e quer U;

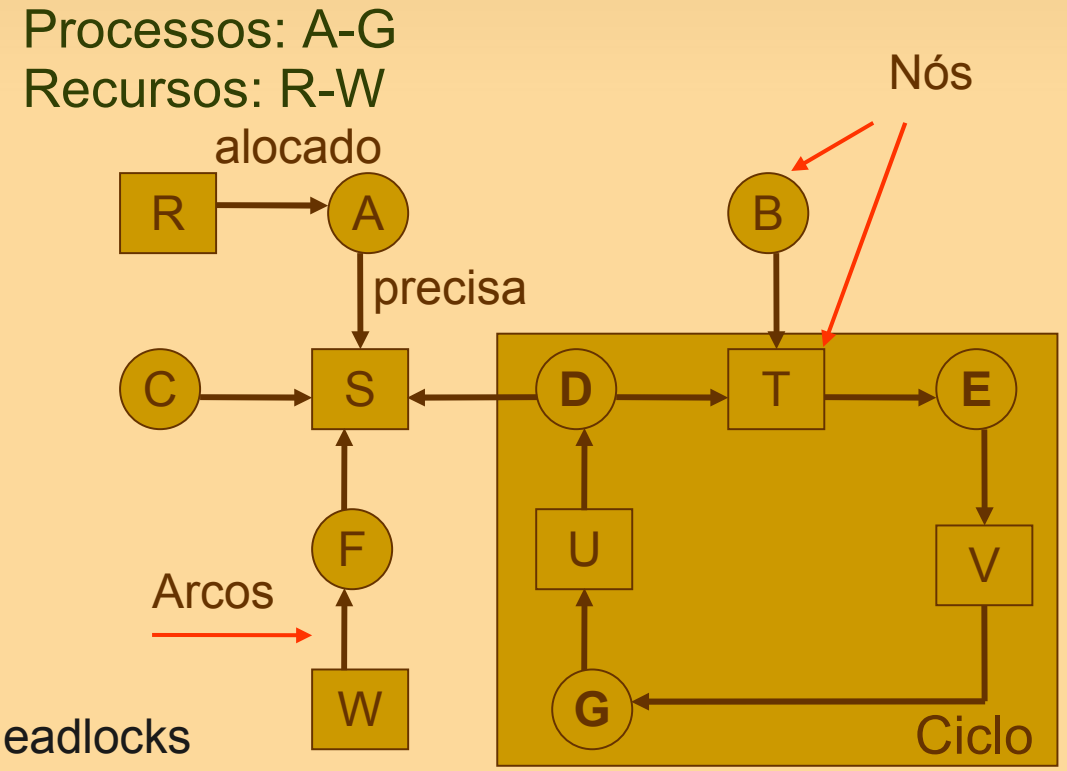
Pergunta:
Há possibilidade de *deadlock*?



Deadlocks

- Detecção com um recurso de cada tipo:
  - Construa o grafo de alocação de recursos;

Situação:
A detém R e quer S;
B não detém nada e quer T;
C não detém nada quer S;
D detém U e quer S e T;
E detém T e quer V;
F detém W e quer S;
G detém V e quer U;



Pergunta:
Há possibilidade de *deadlock*?
Se houver ciclos, existem potenciais deadlocks

Deadlocks

- Detecção com vários recursos de cada tipo:
  - N processos: P_1 a P_n
  - M classes diferentes de recursos:
    - E_i recursos da classe i ($1 \leq i \leq m$)
  - Usamos um vetor de recursos existentes – E
    - Contém o número total de cada recurso existente
    - Se classe 1 for impressora, e E_1=2, então existem duas impressora;

Deadlocks

- Detecção com vários recursos de cada tipo:
  - Vetor de recursos disponíveis (A):
    - $A_i \rightarrow$ quantidade do recurso i disponível nomomento
    - Se ambas as impressoras estiverem alocadas, A_1=0;
  - Duas matrizes:
    - C: matriz de alocação corrente;
      - Cij: número de instâncias do recurso j mantidos pelo processo i;
    - R: matriz de requisições;
      - Rij: número de instâncias do recurso j que o processo i deseja;

Deadlocks

4 unidades de fita;
2 *plotter*;
3 impressoras;
1 unidade de CD-ROM

Três processos:
P_1 usa uma impressora;
P_2 usa duas unidades de fita e uma de CD-ROM;
P_3 usa um *plotter* e duas impressoras;
Cada processo pode precisar de outros recursos (em R);

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

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

Matriz de alocação

	UF	P	I	UCD	
C =	0	0	1	0	P_1
	2	0	0	1	P_2
	0	1	2	0	P_3

Recursos

Matriz de requisições

	UF	P	I	UCD	
R =	0	0	0	0	P_1
	0	0	0	0	P_2
	0	0	0	0	P_3

Deadlocks

4 unidades de fita;
2 *plotter*;
3 impressoras;
1 unidade de CD-ROM

Requisições:
P_1 requisita duas unidades de fita e um CD-ROM;
P_2 requisita uma unidade de fita e uma impressora;
P_3 requisita duas unidades de fita e um *plotter*;

Comparamos cada processo em R com os recursos em A, de modo a encontrar um que possa ser rodado.

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (2 1 0 0) **P_3 pode rodar**

Matriz de alocação

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

Matriz de requisições

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

Deadlocks

Comparamos cada processo em R com os recursos em A, de modo a encontrar um que possa ser rodado.

Somente P_3 pode

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (2 1 0 0) **P_3 pode rodar**
A = (**0 0 0 0**)

Matriz de alocação

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

Matriz de requisições

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

Deadlocks

Ao terminar, P_3 devolve os recursos usados ao vetor de recursos disponíveis.

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (2 1 0 0)
A = (**2 2 2 0**)

Matriz de alocação

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

Matriz de requisições

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

Deadlocks

Ao terminar, P_3 devolve os recursos usados ao vetor de recursos disponíveis.

P2 pode então rodar

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (2 1 0 0)
A = (2 2 2 0) **P_2 pode rodar**
A = (**1 2 1 0**)

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 requisições

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

Deadlocks

Ao terminar, P_3 devolve os recursos usados ao vetor de recursos disponíveis.

P_2 pode então rodar

Ao terminar, também ele devolve os recursos

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (2 1 0 0)
A = (2 2 2 0)
A = (**4 2 2 1**)

Matriz de alocação

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

Matriz de requisições

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

Deadlocks

Ao terminar, P_3 devolve os recursos usados ao vetor de recursos disponíveis.

P_2 pode então rodar

Ao terminar, também ele devolve os recursos

Finalmente, o último processo pode rodar – não há deadlock

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (2 1 0 0)
A = (2 2 2 0)
A = (2 2 1 0) **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 requisições

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

Deadlocks

Ao final da execução, temos:

4 unidades de fita;
2 *plotters*;
3 impressoras;
1 unidade de CD-ROM

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

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

Deadlocks

4 unidades de fita;
2 *plotters*;
3 impressoras;
1 unidade de CD-ROM

Requisições:
P_2 requisita duas unidade de fita, uma impressora e uma unidade de CD-ROM;

Recursos disponíveis
A = (2 1 0 0)

Recursos existentes
E = (4 2 3 1)

Matriz de alocação

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

Matriz de requisições

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

Deadlocks

4 unidades de fita;
2 *plotters*;
3 impressoras;
1 unidade de CD-ROM

Requisições:
P_2 requisita duas unidade de fita, uma impressora e uma unidade de CD-ROM;

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (2 1 0 0) P_3 pode rodar
A = (0 0 0 0)

Matriz de alocação

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

Matriz de requisições

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

Deadlocks

P_3 termina, devolvendo os recursos

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (2 1 0 0)
A = (2 2 2 0)

Matriz de alocação

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

Matriz de requisições

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

Deadlocks

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (2 1 0 0)
A = (2 2 2 0)

Matriz de alocação

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

Matriz de requisições

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

Nessa situação, nenhum processo pode ser atendido!
DEADLOCK

Deadlock ocorrerá se não pudermos mais distribuir recursos disponíveis aos processos requisitantes (em R)

Deadlocks

- Detecção com vários recursos de cada tipo:
  - Precisamos saber de antemão que recursos serão requisitados
  - Quando procurar por eles?
    - Toda vez que um requerimento é feito
    - A cada k minutos
    - Quando o uso da CPU caiu abaixo de um determinado valor
      - Quando se tem muitos processos em situação de deadlock, poucos processos estão em execução, e o uso da CPU cairá

Deadlocks

- Recuperação de Deadlocks:
  - Após detectar, há que se recuperar dele
  - Por meio de preempção:
    - Possibilidade de retirar temporariamente um recurso de seu atual dono (processo) e entregá-lo a outro processo
    - Depende muito da natureza do recurso
    - Frequentemente difícil ou impossível

Deadlocks

- Recuperação de Deadlocks:
  - Por meio de rollback:
    - O estado de cada processo (e recurso por ele usado) é periodicamente armazenado em um arquivo de verificação (checkpoint file);
    - Novas checagens são armazenadas em novos arquivos, à medida que o processo executa
    - Quando ocorre um deadlock, o processo que detém o recurso é voltado a um ponto antes de adquirir esse recurso (via o checkpoint file apropriado)
      - O trabalho feito após esse ponto é perdido
      - O processo é retornado a um momento em que não possuía o recurso, que será então dado a outro

Deadlocks

- Recuperação de Deadlocks:
  - Por meio de eliminação de processos (kill):
    - Um ou mais processos que estão no ciclo com deadlock são interrompidos
    - Talvez isso evite o deadlock, senão, continue eliminando até quebrar o ciclo
    - Melhor solução para processos que não causam algum efeito negativo ao sistema;
      - Ex1.: compilação – sem problemas;
      - Ex2.: atualização de um base de dados – problemas;

Deadlocks

- Evitar dinamicamente o problema:
  - Alocação individual de recursos são normalmente feitos à medida que o processo necessita;
  - Escalonamento cuidadoso → alto custo;
    - Conhecimento prévio dos recursos que serão utilizados;
  - Algoritmos:
    - Banqueiro para um único tipo de recurso;
    - Banqueiro para vários tipos de recursos;
  - Usam a noção de Estados Seguros e Inseguros;

Deadlocks

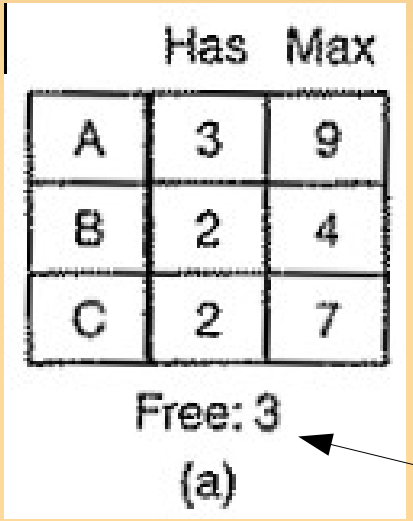
- Evitar dinamicamente o problema:
  - Estados seguros: não provocam deadlocks e há uma maneira de atender a todas as requisições pendentes finalizando normalmente todos os processos;
    - A partir de um estado seguro, existe a garantia de que os processos terminarão;
  - Estados inseguros: podem provocar deadlocks, mas não necessariamente provocam;
    - A partir de um estado inseguro, não é possível garantir que os processos terminarão corretamente;

Deadlocks

- Evitar dinamicamente o problema:
  - Usa as mesmas estruturas da detecção com vários recursos
    - Um estado consiste das estruturas E, A, C e R
  - Estado seguro:
    - Se houver alguma ordem de alocação na qual todo processo irá terminar, mesmo se todos peçam seu número máximo de recursos imediatamente

Deadlocks

- Evitar dinamicamente o problema:
  - Estado seguro:
    - Ex: Tabela para um único recurso



	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

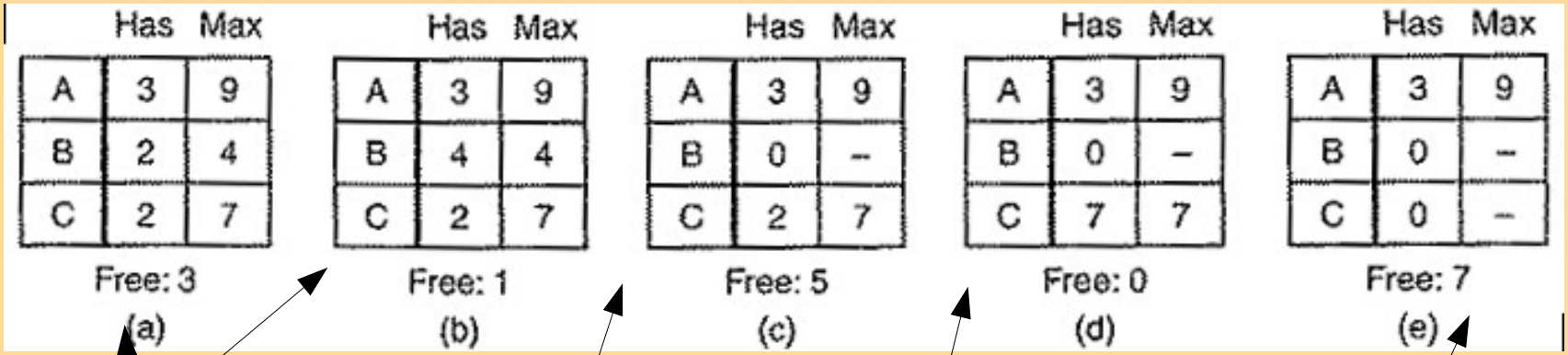
(a)

A tem 3 instâncias de um recurso, mas pode precisar de até 9
B tem 2, e pode precisar de 4
C tem 2 e pode precisar de 7

Havendo um total de 10 instâncias do recurso, diante dessas necessidades, sobrarão 3

Deadlocks

- Evitar dinamicamente o problema:
  - Estado seguro:
    - Há seqüência que permite que todos os processos terminem



	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3 (a)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1 (b)

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5 (c)

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0 (d)

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7 (e)

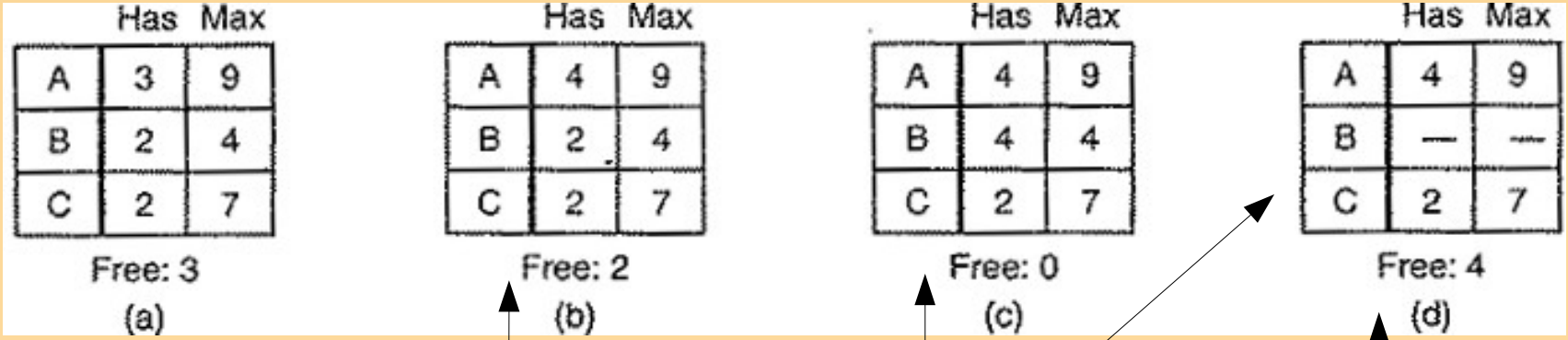
O escalonador roda B até que ele peça mais recursos

Quando B termina roda C até terminar

Agora A pode obter as 6 instâncias de que precisa para terminar

Deadlocks

- Evitar dinamicamente o problema:
  - Estado seguro:
    - Suponha agora a seguinte configuração



	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4

(d)

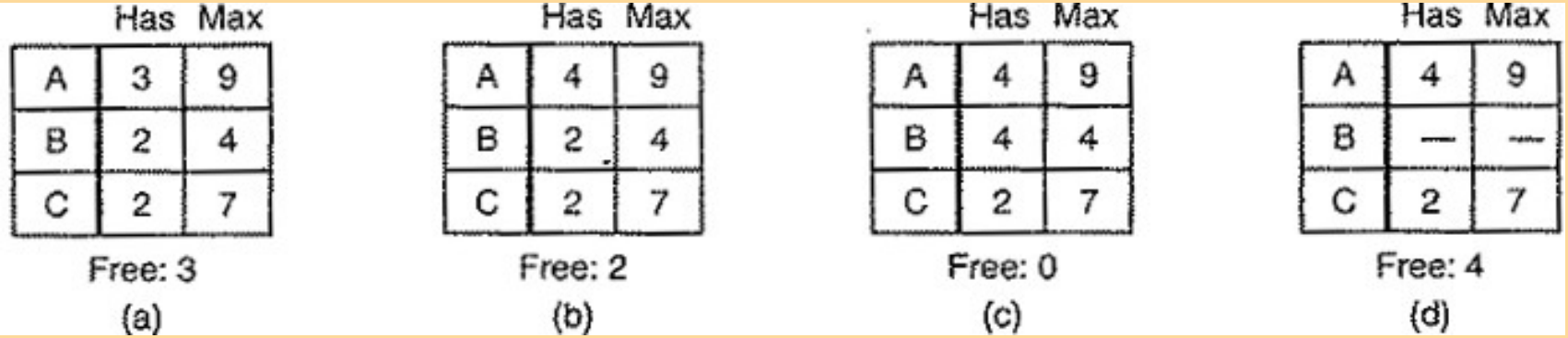
A recebe um novo recurso

B roda até o fim

Ficamos presos: precisamos de 5 para cada um, mas temos só 4 livres

Deadlocks

- Evitar dinamicamente o problema:
  - Estado seguro:
    - Não há seqüência capaz de garantir que os programas terminem
    - A decisão que moveu de (a) para (b) levou de um estado seguro a um inseguro
      - Não deveríamos tê-la tomado



	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4

(d)

Deadlocks

- Algoritmos do Banqueiro:
  - Idealizado por Dijkstra (1965);
  - Algoritmo de escalonamento capaz de evitar deadlock
  - Premissas adotadas por um banqueiro (SO) para garantir ou não crédito (recursos) para seus clientes (processos);
    - Verifica se atender um pedido leva a um estado inseguro.
    - Se levar, o pedido é negado, senão, é executado

Deadlocks

- Algoritmo do Banqueiro para um único tipo de recurso:

4 clientes: A, B, C e D

O banqueiro sabe que não precisarão de todo o crédito disponível, por isso reservou 10 dos 22 para distribuir

Máximo de linha de crédito = 22

Possui

A	0	6
B	0	5
C	0	4
D	0	7

Livre: 10

Seguro

A	1	6
B	1	5
C*	2	4
D	4	7

Livre: 2

Seguro

O estado é seguro porque, com 2 sobrando, o banqueiro pode terminar C, que liberaria 4, permitindo terminar D ou B (que precisam de 3 e 4), e assim por diante

• Solicitações de crédito são realizadas de tempo em tempo; em um determinado instante, a situação é essa

Deadlocks

- Algoritmo do Banqueiro para um único tipo de recurso:

Deadlocks

- Algoritmo do Banqueiro para um único tipo de recurso:
  - Considera cada pedido à medida em que ocorre, e vê se atendê-lo leva a um estado seguro
  - Para ver se o estado é seguro, o banqueiro verifica se tem recursos suficientes para satisfazer algum cliente
    - Se tiver, assume que os empréstimos serão pagos, e o cliente mais próximo do limite é checado
    - Sempre tenta emprestar o máximo a um único por vez

Deadlocks

- Algoritmo do Banqueiro para vários tipos de recursos:
  - Mesma idéia, mas duas matrizes são utilizadas;

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

Processos	Unidade de Fita	Plotters	Impressoras	Unidade de CD-ROM
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

C = Recursos Alocados a cada processo

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 que cada processo ainda necessita para terminar

Cada processo deve dizer de antemão de quantos recursos precisará, antes de executar

Deadlocks

- Algoritmo do Banqueiro para vários tipos de recursos:
  - Busque em R uma linha que possa ser satisfeita pelos valores em A (deve possuir valores todos menores ou iguais aos de A)
    - Se não existir, pode haver deadlock
  - Assuma que o processo na linha escolhida pede todos os recursos de que precisa e termina.
    - Marque este processo como terminado e adicione todos seus recursos a A
  - Repita os passos acima até que todos os processos sejam marcados como terminados (caso em que o estado inicial era seguro) ou reste processo que não possa ser satisfeito (deadlock)

Deadlocks

O que aconteceria se atendêssemos uma requisição de B?

Processos	Unidade de Fita	Plotters	Impressoras	Unidade de CD-ROM
A	3	0	1	1
B	0	1	1	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

C = Recursos Alocados

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

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

R = Recursos ainda necessários

- Podem ser atendidos: D, A ou E, C;

Deadlocks

Ao atender E, teríamos um deadlock

Processos	Unidade de Fita	Plotters	Impressoras	Unidade de CD-ROM
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

Alocados → P = (5 3 **4** 2);
Disponíveis → A = (1 0 **0** 0);

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

- *Solução*: Adiar a requisição de E por alguns instantes;

Deadlocks

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

Deadlocks

- Prevenir Deadlocks:
  - Atacar uma das quatro condições:

Condição	Abordagem
Exclusão Mútua	Alocar todos os recursos usando spooling (só o printer daemon tem acesso direto à impressora)
Uso e Espera	Processos requisitam todos os recursos de que precisam antes da execução – difícil saber; sobrecarga do sistema
Não-preempção	Retirar recursos dos processos – pode ser ruim dependendo do tipo de recurso; praticamente não implementável
Espera Circular	Ordenar numericamente os recursos, e realizar solicitações em ordem numérica - não há ciclos Permitir que o processo utilize apenas um recurso por vez (se quiser um segundo, deve liberar o primeiro)

Deadlocks

- Deadlocks podem ocorrer sem o envolvimento de recursos, por exemplo, se semáforos forem implementados erroneamente;

```
..                      ..
down(&empty);           down(&mutex);
    down(&mutex);       down(&empty);
    ..                  ..
```

- Inanição (Starvation)
  - Todos os processos devem conseguir utilizar os recursos que precisam, sem ter que esperar indefinidamente;
    - Colocar os processos em uma fila