

Deadlocks

Impasses



Introdução aos *Deadlocks*

- Definição formal:

Um conjunto de processos está em situação de deadlock se todo processo pertencente ao conjunto estiver esperando por um evento que somente um outro processo desse mesmo conjunto poderá fazer acontecer

- Normalmente o evento é a liberação de um recurso atualmente retido
- Nenhum dos processos pode...
 - executar
 - liberar recursos
 - ser acordado

Recursos

- Exemplos de recursos de computador
 - impressoras
 - unidades de fita
 - tabelas
- Processos precisam de acesso aos recursos numa ordem racional
- Suponha que um processo detenha o recurso A e solicite o recurso B
 - ao mesmo tempo um outro processo detém B e solicita A
 - ambos são bloqueados e assim permanecem

Recursos (1)

- *Deadlocks* ocorrem quando ...
 - garante-se aos processos acesso exclusivo aos dispositivos
 - esses dispositivos são normalmente chamados de recursos
- Recursos preemptíveis
 - podem ser retirados de um processo sem quaisquer efeitos prejudiciais
- Recursos não preemptíveis
 - vão induzir o processo a falhar se forem retirados

Recursos (2)

- Sequência de eventos necessários ao uso de um recurso
 1. solicitar o recurso
 2. usar o recurso
 3. liberar o recurso
- Deve esperar se solicitação é negada
 - processo solicitante pode ser bloqueado
 - pode falhar resultando em um código de erro

Quatro Condições para *Deadlock*

1. Condição de exclusão mútua
 - todo recurso está ou associado a um processo ou disponível
2. Condição de posse e espera
 - processos que retêm recursos podem solicitar novos recursos
3. Condição de não preempção
 - recursos concedidos previamente não podem ser forçosamente tomados
4. Condição de espera circular
 - deve ser uma cadeia circular de 2 ou mais processos
 - cada um está à espera de recurso retido pelo membro seguinte dessa cadeia

Modelagem de *Deadlock* (2)

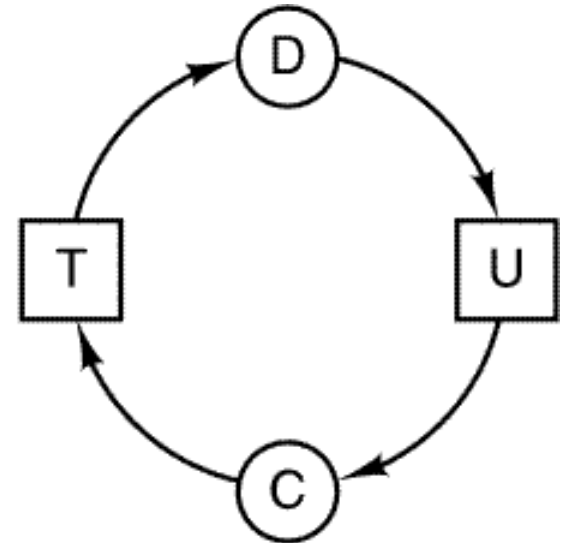
- Modelado com grafos dirigidos



(a)



(b)



(c)

c) processos C e D estão em *deadlock* sobre recursos T e U

Estratégias para Tratar o Deadlock

- As situações de deadlock podem ser tratadas ou não em um sistema, os desenvolvedores devem avaliar o custo/benefício que essas implementações podem trazer. Normalmente, as estratégias usadas para detectar e tratar as situações de deadlocks, geram grande sobrecarga, podendo até causar um dano maior que a própria ocorrência do deadlock, sendo, às vezes, melhor ignorar a situação.

Modelagem de *Deadlock* (3)

Estratégias para tratar *Deadlocks*

1. ignorar por completo o problema
2. detecção e recuperação
3. evitação dinâmica
 - alocação cuidadosa de recursos
4. prevenção
 - negação de uma das quatro condições necessárias

Modelagem de *Deadlock* (4)

A
Requisita R
Requisita S
Libera R
Libera S

(a)

B
Requisita S
Requisita T
Libera S
Libera T

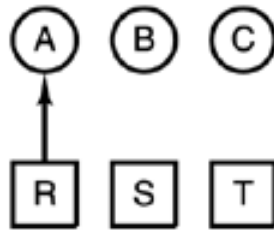
(b)

C
Requisita T
Requisita R
Libera T
Libera R

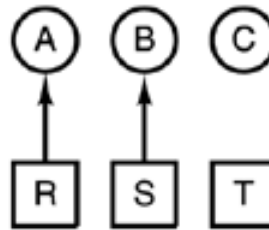
(c)

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

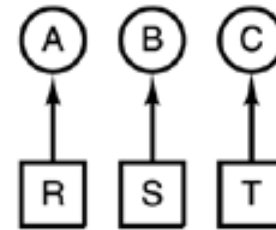
(d)



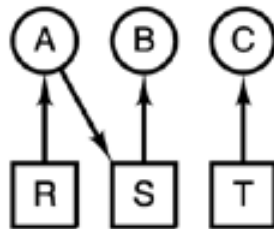
(e)



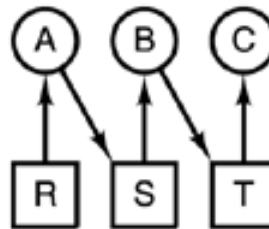
(f)



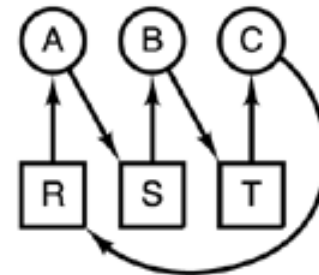
(g)



(h)



(i)



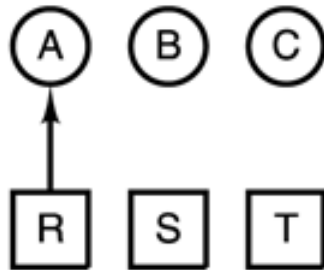
(j)

Como ocorre um *deadlock*

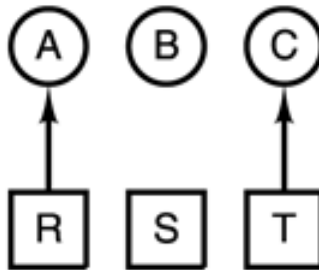
Modelagem de *Deadlock* (5)

1. A requisita R
 2. C requisita T
 3. A requisita S
 4. C requisita R
 5. A libera R
 6. A libera S
- nenhum deadlock

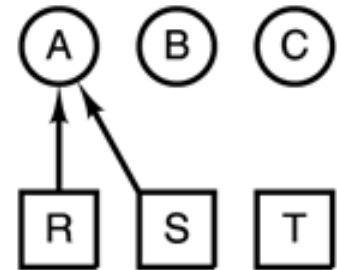
(k)



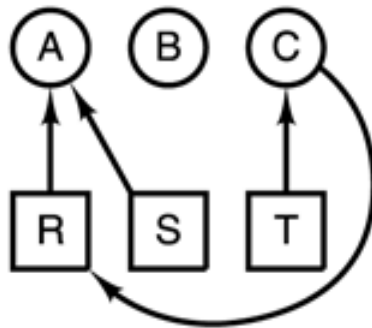
(l)



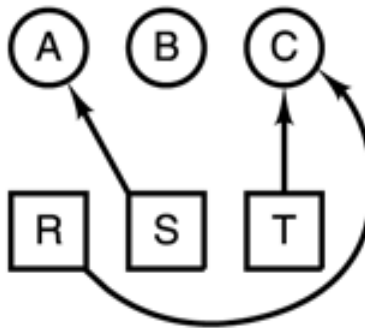
(m)



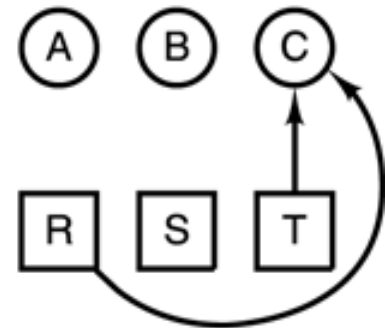
(n)



(o)



(p)



(q)

Como pode ser evitado um *deadlock*

Algoritmo do Avestruz

- Finge que o problema não existe
- Razoável se
 - deadlocks ocorrem muito raramente
 - custo da prevenção é alto
- UNIX e Windows seguem esta abordagem
- É uma ponderação entre
 - conveniência
 - correção

Estratégias para Tratar o Deadlock

- **Deteção e Recuperação**
- O sistema permite que ocorra o deadlock e depois executa o procedimento de recuperação, que se resume na detecção da ocorrência e na recuperação posterior do sistema.
- Para detecção do deadlock, deve-se implementar no sistema uma estrutura de dados que armazene as informações sobre os processos e os recursos alocados a eles e essas reflitam a situação de cada processo/recurso no sistema. Porém, é importante ressaltar que o simples procedimento de atualização dessas estruturas gera sobrecarga no sistema, pois toda vez que o processo aloca, libera ou requisita um recurso, elas precisam ser atualizadas.

Estratégias para Tratar o Deadlock

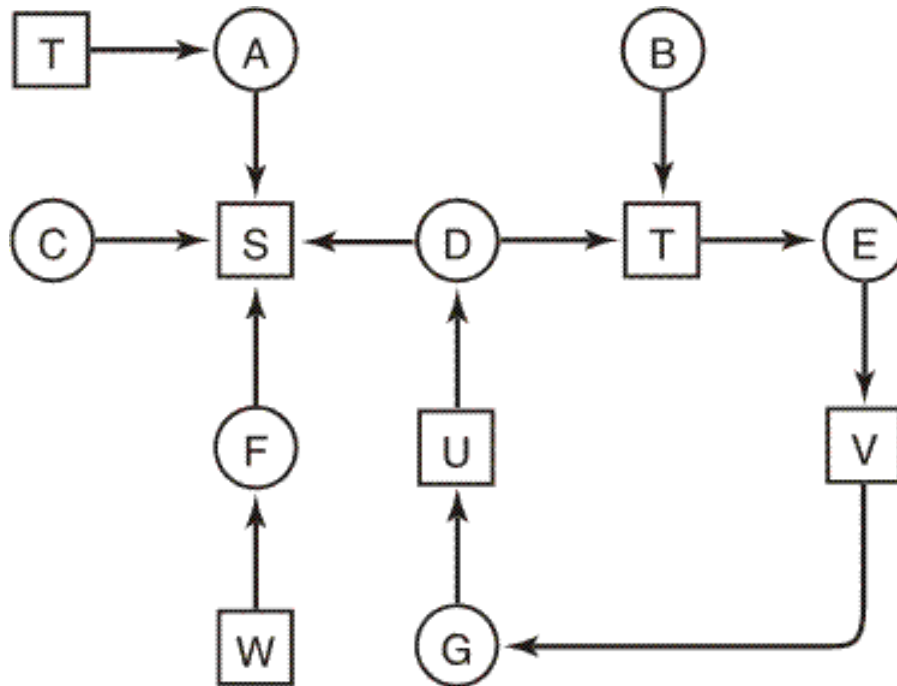
Algumas das técnicas de detecção utilizadas são:

- **Detecção de deadlocks com um recurso de cada tipo** – Se tem apenas um recurso de cada tipo (somente uma impressora, ou um CD,...). Existe um algoritmo para detectar se existem ciclos no grafo dos processos e recursos;

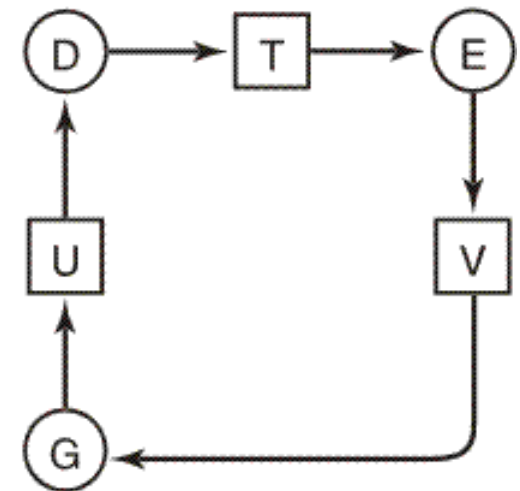
Estratégias para Tratar o Deadlock

- **Detecção de deadlocks com múltiplos recursos de cada tipo** – O algoritmo baseia-se em um ambiente que possui vários recursos do mesmo tipo e os processos solicitam apenas pelo tipo de recursos, não especificando qual recurso desejam utilizar.
- Uma vez que o algoritmo de detecção de deadlocks é bem sucedido, o que se fará em seguida é a recuperação do sistema da situação de deadlock e colocá-lo novamente em condição de funcionamento normal.

Detecção com um Recurso de Cada Tipo (1)



(a)




(b)

- Observe a posse e solicitações de recursos
- Um ciclo pode ser encontrado dentro do grafo, denotando deadlock

Detecção com um Recurso de Cada Tipo (2)

Recursos existentes
($E_1, E_2, E_3, \dots, E_m$)

Matriz de alocação atual




C_{11}	C_{12}	C_{13}	\dots	C_{1m}
C_{21}	C_{22}	C_{23}	\dots	C_{2m}
\vdots	\vdots	\vdots		\vdots
C_{n1}	C_{n2}	C_{n3}	\dots	C_{nm}

Linha n é a alocação
atual para o processo n

Recursos disponíveis
($A_1, A_2, A_3, \dots, A_m$)

Matriz de requisições



R_{11}	R_{12}	R_{13}	\dots	R_{1m}
R_{21}	R_{22}	R_{23}	\dots	R_{2m}
\vdots	\vdots	\vdots		\vdots
R_{n1}	R_{n2}	R_{n3}	\dots	R_{nm}

Linha 2 informa qual é a
necessidade do processo 2

Estruturas de dados necessárias ao algoritmo de detecção de deadlock

Detecção com um Recurso de Cada Tipo (3)

$$E = \begin{pmatrix} 4 & 2 & 3 & 1 \end{pmatrix}$$

Unidades de fita
Plotters
Scanners
Unidades de CD-ROM

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix}$$

Unidades de fita
Plotters
Scanners
Unidades de CD-ROM

Matriz alocação atual

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

Matriz de requisições

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

Um exemplo para o algoritmo de detecção de deadlocks

Recuperação de Deadlock

- Recuperação através de preempção
 - retirar um recurso de algum outro processo
 - depende da natureza do recurso

A idéia é retirar o recurso a um processo e dá-lo a outro sem que o processo proprietário do recurso se aperceba.

Recuperação de Deadlock

- Recuperação através de reversão de estado

O sistema operacional, ao longo da execução dos processos vai guardando imagens dos estados do processo, para que fique como que um posto de fiscalização do processo. Não guarda apenas os estados dos processos, como guarda também os recursos associados ao processo no momento em que é criado o posto de fiscalização.

O sistema não sobrepõe um posto de fiscalização novo a um já guardado anteriormente. Ele cria sempre uma imagem nova. Depois que o algoritmo de detecção de deadlocks detecta um deadlock os processos incluídos no deadlock voltam a estados anteriores.

- verifica um processo periodicamente
- usa este estado salvo
- reinicia o processo se este é encontrado em estado de deadlock

Recuperação de Deadlock

- Recuperação através da eliminação de processos

A maneira mais simples de recuperar o sistema é matar um ou mais processos envolvidos no deadlock. Com um pouco de sorte os outros processos poderão ser capazes de prosseguir.

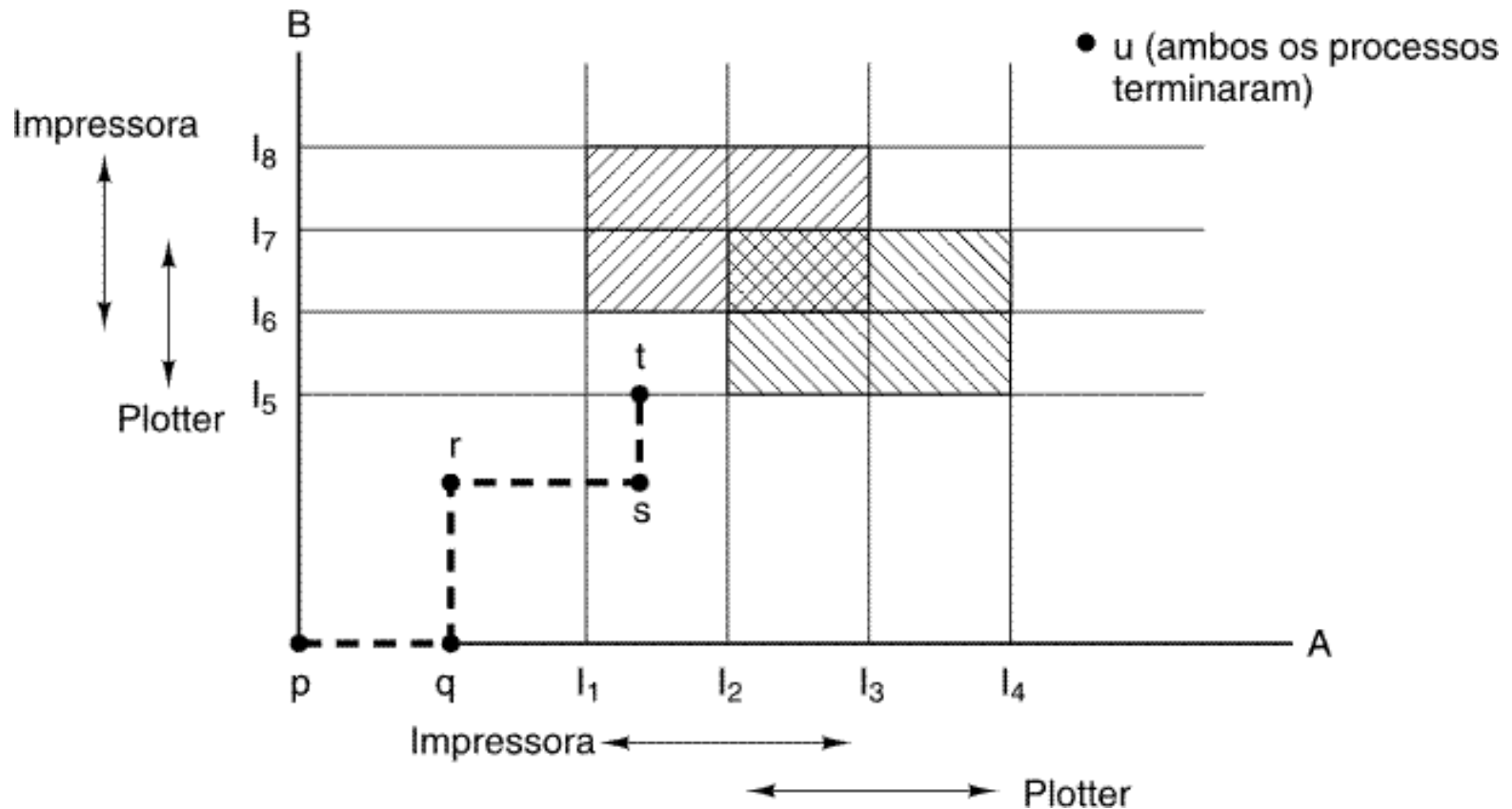
- forma mais grosseira mas também mais simples de quebrar um deadlock
- elimina um dos processos no ciclo de deadlock
- os outros processos conseguem seus recursos
- escolhe processo que pode ser reexecutado desde seu início

Evitar Deadlock

- O deadlock pode ser evitado, mas só quando certas informações estiverem disponíveis.
- O Sistema Operacional que adota esta estratégia, procura evitar a ocorrência de deadlocks por meio de alocação cuidadosa de recursos. O sistema deve ser capaz de saber e decidir se liberar um recurso é seguro ou não.
- Vejamos a seguir alguns processos que visam evitar deadlocks:

Evitando Deadlocks

Trajeto rias de Recursos



Trajeto rias de recursos de dois processos

Evitar Deadlock

- **Estados seguros e não seguros** – É considerado um estado seguro se ele não estiver em situação de deadlock e se existir alguma ordem de escalonamento na qual todo o processo possa ser executado até sua conclusão, mesmo se de repente, todos eles requisitarem, de uma só vez, o máximo possível de recursos.
- Um estado não seguro não é uma situação de deadlock. É um estado em que a partir do mesmo, o sistema não pode dar a garantia de que o processo vai acabar

Estados Seguros e Inseguros (1)

Possui máx.	Possui máx.	Possui máx.	Possui máx.	Possui máx.																																													
<table><tr><td>A</td><td>3</td><td>9</td></tr><tr><td>B</td><td>2</td><td>4</td></tr><tr><td>C</td><td>2</td><td>7</td></tr></table>	A	3	9	B	2	4	C	2	7	<table><tr><td>A</td><td>3</td><td>9</td></tr><tr><td>B</td><td>4</td><td>4</td></tr><tr><td>C</td><td>2</td><td>7</td></tr></table>	A	3	9	B	4	4	C	2	7	<table><tr><td>A</td><td>3</td><td>9</td></tr><tr><td>B</td><td>0</td><td>–</td></tr><tr><td>C</td><td>2</td><td>7</td></tr></table>	A	3	9	B	0	–	C	2	7	<table><tr><td>A</td><td>3</td><td>9</td></tr><tr><td>B</td><td>0</td><td>–</td></tr><tr><td>C</td><td>7</td><td>7</td></tr></table>	A	3	9	B	0	–	C	7	7	<table><tr><td>A</td><td>3</td><td>9</td></tr><tr><td>B</td><td>0</td><td>–</td></tr><tr><td>C</td><td>0</td><td>–</td></tr></table>	A	3	9	B	0	–	C	0	–
A	3	9																																															
B	2	4																																															
C	2	7																																															
A	3	9																																															
B	4	4																																															
C	2	7																																															
A	3	9																																															
B	0	–																																															
C	2	7																																															
A	3	9																																															
B	0	–																																															
C	7	7																																															
A	3	9																																															
B	0	–																																															
C	0	–																																															
Disponível: 3	Disponível: 1	Disponível: 5	Disponível: 0	Disponível: 7																																													
(a)	(b)	(c)	(d)	(e)																																													

Demonstração de que o estado em (a) é seguro

Estados Seguros e Inseguros (2)

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)

Demonstração de que o estado em (b) é inseguro

Evitar Deadlock

Algoritmo de banqueiro – Usado para determinar se um processo pode executar de maneira segura ou não. Todos os processos declaram o máximo de recursos que vão usar durante a execução. A execução é permitida se a soma dos recursos requisitados é menor que os recursos disponíveis no sistema. O algoritmo verifica se a libertação de uma requisição pode levar a um estado não seguro.

Em caso positivo, a requisição é negada. Se a libertação de uma requisição levar a um estado seguro, então ela é atendida.

O Algoritmo do Banqueiro para um Único Recurso

Possui máx.

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

Disponível: 10

(a)

Possui máx.

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

Disponível: 2

(b)

Possui máx.

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

Disponível: 1

(c)

Três estados de alocação de recursos

- a) seguro
- b) seguro
- c) inseguro

O Algoritmo do Banqueiro para Múltiplos Recursos

	Processo	Unidades de fita	Plotters	Scanners	Unidades 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	

Recursos alocados

	Processo	Unidades de fita	Plotters	Scanners	Unidades de CD-ROM
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	

Recursos ainda necessários

E = (6342)

P = (5322)

A = (1020)

Exemplo do algoritmo do banqueiro com múltiplos recursos

Prevenção de Deadlock

- Evitar deadlock é praticamente impossível. Por isso, a prevenção de deadlock tenta garantir que pelo menos uma das condições para ocorrência de deadlock, não aconteça.
- Sabendo que são quatro as condições para que possa ocorrer uma situação de deadlock simultaneamente, a prevenção procura eliminar pelo menos uma delas utilizando as seguintes técnicas:
- **Condição de exclusão mútua , Condição de posse e espera , Condição de não preempção e a Condição de espera circular .**

Prevenção de Deadlock

Atacando a Condição de Exclusão Mútua

O processo solicita o recurso para uso de forma mutuamente exclusiva. Essa condição é eliminada se o processo solicita todos os recursos que necessita em uma única vez.

- Alguns dispositivos (como uma impressora) podem fazer uso de *spool*
 - o daemon de impressão é o único que usa o recurso impressora
 - desta forma deadlock envolvendo a impressora é eliminado
- Nem todos os dispositivos podem fazer uso de *spool*
- Princípio:
 - evitar alocar um recurso quando ele não for absolutamente necessário
 - tentar assegurar que o menor número possível de processos possa de fato requisitar o recurso

Prevenção de Deadlock

Atacando a Condição de Posse e Espera

Os processos devem pedir os recursos antes de iniciarem a sua execução.

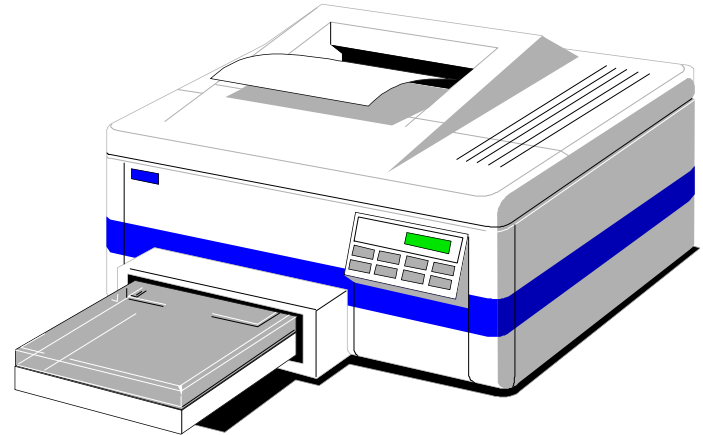
- Exigir que todos os processos requisitem os recursos antes de iniciarem
 - um processo nunca tem que esperar por aquilo que precisa
- Problemas
 - podem não saber quantos e quais recursos vão precisar no início da execução
 - e também retêm recursos que outros processos poderiam estar usando
- Variação:
 - processo deve desistir de todos os recursos
 - para então requisitar todos os que são imediatamente necessários

Prevenção de Deadlock

Atacando a Condição de Não Preempção

Elimina-se essa condição se os recursos forem ordenados e os processos devem requisitar os recursos em uma sequência que respeite esta ordem.

- Esta é uma opção inviável
- Considere um processo de posse de uma impressora
 - no meio da impressão
 - retoma a impressora a força
 - !!??



Prevenção de Deadlock

Atacando a Condição de Espera Circular (1)

Pode ser eliminada se for construído um grafo e se for verificado, para cada requisição, se o atendimento não levará o sistema a um estado não seguro. As requisições que levarem o sistema a um estado não seguro ou de deadlock não deverão ser atendidas.

1. Imagesetter
2. Scanner
3. Plotter
4. Unidade de fita
5. Unidade de CD-ROM



(a)



(b)

- a) Recursos ordenados numericamente
- b) Um grafo de recursos

Prevenção de Deadlock

Atacando a Condição de Espera Circular (2)

Condição	Abordagem contra deadlocks
Exclusão mútua	Usar spool em tudo
Posse-e-espera	Requisitar inicialmente todos os recursos necessários
Não preempção	Retomar os recursos alocados
Espera circular	Ordenar numericamente os recursos

Resumo das abordagens para prevenir deadlock

Outras Questões

Bloqueio em Duas Fases

- Fase um
 - processo tenta bloquear todos os registros de que precisa, um de cada vez
 - Se registro necessário já estiver bloqueado, reinicia novamente
 - (nenhum trabalho real é feito na fase um)
- Se a fase um for bem sucedida, começa a fase dois,
 - execução de atualizações
 - liberação de bloqueios
- Observe a similaridade com a requisição de todos os recursos de uma só vez
- Algoritmo funciona onde o programador tiver organizado tudo cuidadosamente para que
 - o programa possa ser parado, reiniciado

Deadlocks que não Envolvem Recursos

- É possível que dois processos entrem em situação de deadlock
 - cada um espera que o outro faça algo
- Pode ocorrer com semáforos
 - cada processo executa um *down()* sobre dois semáforos (*mutex* e outro qualquer)
 - se executados na ordem errada, resulta em deadlock

Condição de Inanição - *Starvation*

- Algoritmo para alocar um recurso
 - pode ser ceder para o *job mais curto primeiro*
- Funciona bem para múltiplos jobs curtos em um sistema
- Jobs longos podem ser preteridos indefinidamente
 - mesmo não estando bloqueados
- solução:
 - política do *primeiro a chegar, primeiro a ser servido*

Novidades do mercado

<https://youtu.be/TAWIxB5hUI>

<https://aws.amazon.com/pt/>

<https://cloud.google.com/?hl=pt-br>

<https://azure.microsoft.com/pt-br>