Capítulo 3

Deadlocks - Impasses

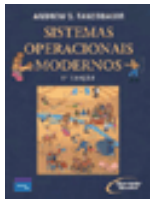
3.1. Recurso
3.2. Introdução aos *deadlocks*
3.3. Algoritmo do avestruz
3.4. Detecção e recuperação de *deadlocks*
3.5. Evitando *deadlocks*
3.6. Prevenção de *deadlocks*

Recursos

- Exemplos de recursos do computador
  - Impressoras
  - Unidades de fita
  - Tabelas
- Processos precisam de acesso aos recursos em ordem racional
- Suponha que um processo detenda 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

Recursos



- *Deadlocks* ocorrem quando ...
  - Garante-se aos processos acesso exclusivo aos dispositivos
  - Esses dispositivos são normalmente chamados de recursos
- Recursos Preemptíveis
  - Podem ser retirdos de um processo sem quaisquer efeitos prejudiciais
- Recursos Não Preemptíveis
  - Vão induzir o processo a falhar se forem retirados

Recursos

- 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 resultado em um código de erro

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

O Jantar dos Filósofos

O Jantar dos Filósofos

Considere 5 filósofos que passam a vida a pensar e a comer.

Partilham uma mesa redonda rodeada por 5 cadeiras sendo que cada uma das cadeiras pertence a um filósofo. No centro da mesa encontra-se uma taça de arroz e estão 5 garfos na mesa, um para cada filósofo.

Quando um filósofo pensa não interage com os seus colegas. De tempos em tempos, cada filósofo fica com fome e tenta apanhar os dois garfos que estão mais próximos (os garfos que estão ou à esquerda ou à direita).

O filósofo apenas pode apanhar um garfo de cada vez e como o leitor compreende, não pode apanhar um garfo se este estiver na mão do vizinho. Quando um filósofo esfomeado tiver 2 garfos ao mesmo tempo ele come sem largar os garfos. Apenas quando acaba de comer, o filósofo pousa os garfos, libertando-os e começa a pensar de novo.

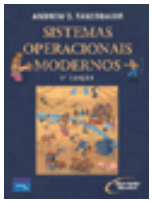
O Jantar dos Filósofos

- Solução 1:
  - O filósofo pode pegar um garfo qualquer, caso não consiga, fique aguardando. Em seguida faz o mesmo para o segundo garfo. Uma vez com os garfos nas mãos começa a comer. Ao final devolve um garfo e depois o outro. O grande problema nesse algoritmo é que existem momentos em que se todos os filósofos pegarem um garfo, todos irão ficar parados para sempre aguardando o segundo garfo ficar disponível; gerando assim um Deadlock ou Impasse! Ou seja, não é uma solução completa.
- Solução 2:
  - Toda vez que um filósofo pegar o primeiro garfo e em seguida não conseguir pegar o segundo garfo, ele devolva o primeiro à mesa e aguarde um tempo fixo para tentar tudo novamente. Nessa nova solução, o filósofo poderá morrer de fome. Ele não ficará travado, mas poderá ficar tentando fazer a mesma operação de pegar e depois devolver o garfo pra sempre. A consequência será nunca conseguir comer.

O Jantar dos Filósofos

- Solução 3:
  - A terceira solução é uma pequena variação da segunda, implementando um fator aleatório para dificultar a possibilidade de que um filósofo morra de fome (starvation). Com tempos aleatórios de espera para tentar pegar novamente um garfo a probabilidade de starvation é mínima, no entanto, não é impossível, logo, ainda não temos uma solução completa.
- Solução 4:
  - A quarta solução é baseada em semáforos. O semáforo irá garantir que haja exclusão mútua, onde apenas um filósofo estará com um determinado garfo por vez. As soluções anteriores também garantem a exclusão mútua, mas o semáforo irá garantir a completude do programa.
  - Não haverá starvation nem deadlock. No problema do Jantar dos filósofos a seção crítica da execução de um filósofo é a operação de comer, que necessariamente, precisa utilizar os dois garfos. Nessa nova implementação podemos representar cada garfo como um mutex. Logo após o filósofo terminar de comer os dois garfos que estavam com ele serão liberados. O sistema operacional garantirá que outras threads interessadas no recurso liberado sejam escalonadas para executar e tentar pegar os garfos livres.

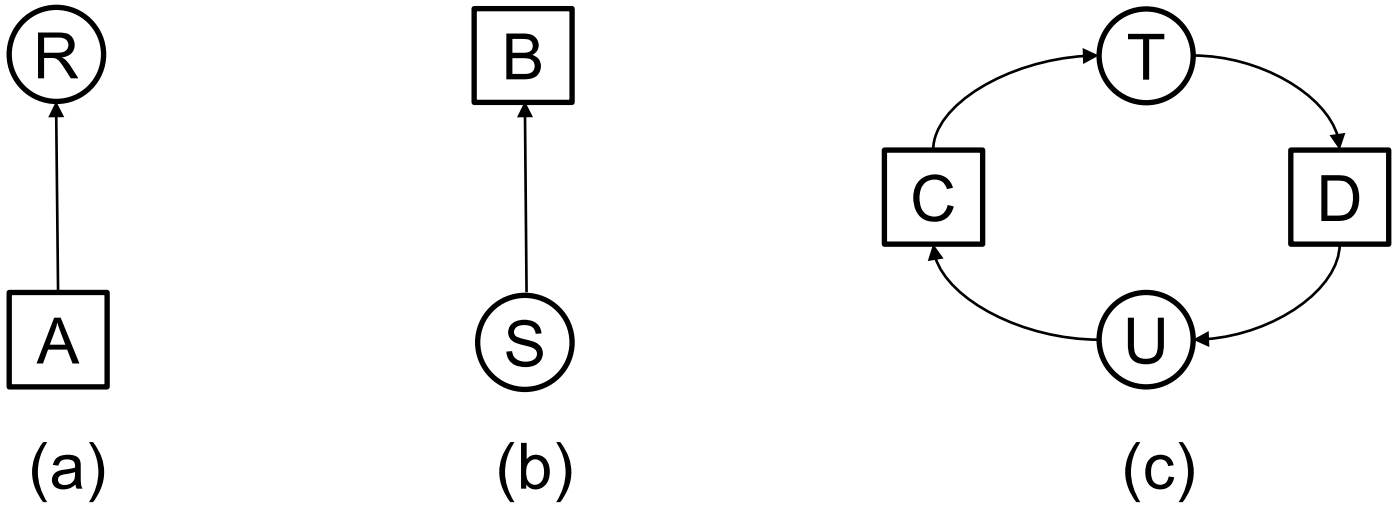
Quatro Condições para *Deadlock*



1. Condição de exclusão mútua
   - Todo recurso está ou associado a um processo ou disponível
2. Condição de posse e espera
   - Processos que retém 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*

- Modelado com grafos dirigidos



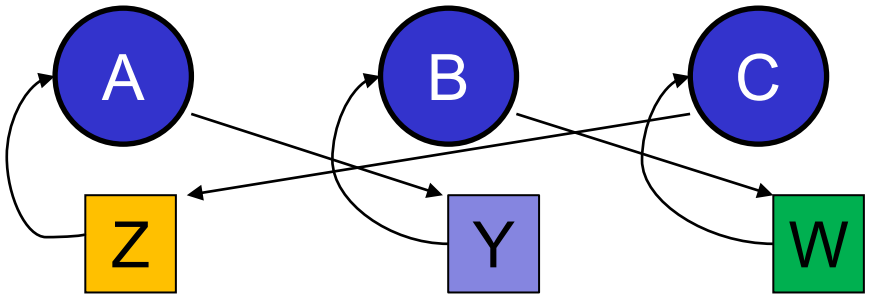
  - Processo A está solicitando/esperando pelo recurso R
  - Recurso S alocado ao processo B
  - Processos C e D estão em deadlock sobre recursos T e U

Modelagem de *Deadlock*

- Estratégias para tratar *Deadlocks*
  1. Ignorar por completo o problema
  2. Detecção e recuperação
  3. Prevenção dinâmica
     - Alocação cuidadosa de recursos
  4. Prevenção estrutural
     - Negação de uma das quatro condições necessárias

Modelagem de *Deadlock*

- Exemplo:
  - O Processo “A” espera pelo processo “B”, que espera pelo processo “C”, que espera pelo processo “A”.



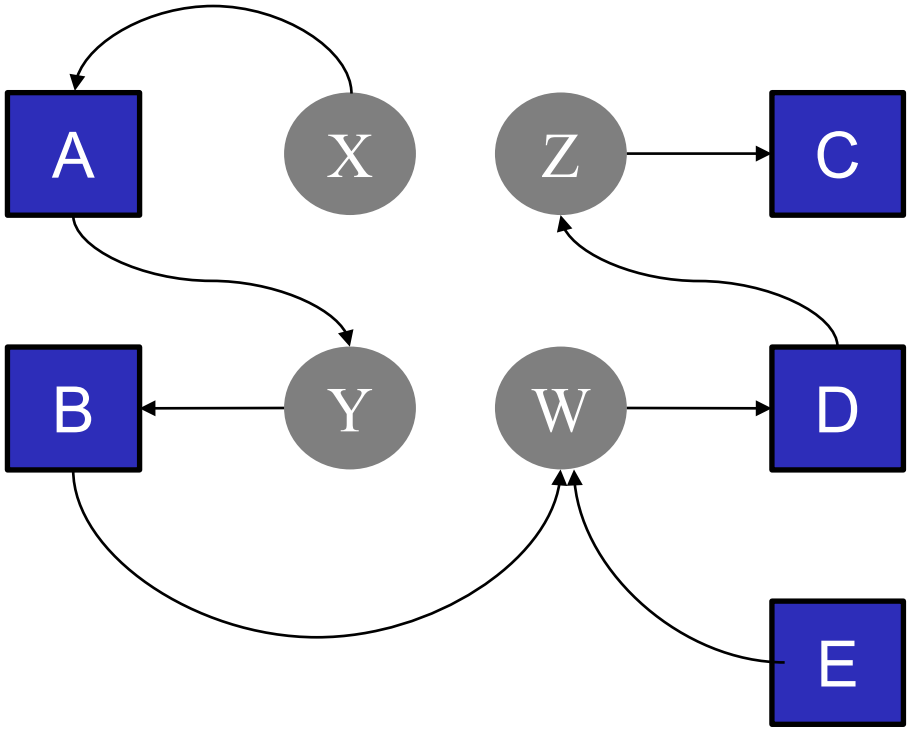
Algoritmo do Avestruz

- Finge que o problema não existe
- Razoável se
  - Deadlocks ocorrem muito raramente
  - Custo da prevenção é muito alto
- UNIX e Windows seguem essa abordagem
- É uma ponderação entre
  - Conveniência
  - Correção

Detecção com um Recurso de Cada Tipo

- Exemplo:

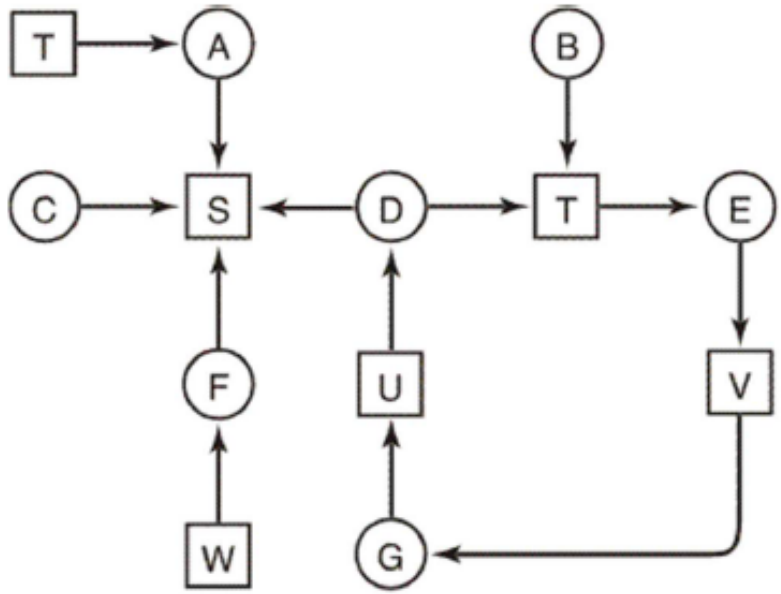
Recurso	Processo Alocado	Processos Requisitantes
X	A	
Y	B	A
W	C	B, E
Z	D	C



Detecção com um Recurso de Cada Tipo

- Nesse exemplo, embora existe disputa por recursos (Processos B e E requisitam o uso do recurso W), não existe nenhum caminho fechado, ou seja, não existe qualquer deadlock:
    1. D finaliza o uso de Z
    2. Com Z livre, C aloca Z e ao finalizar libera Z e W
    3. Com W livre, B e E poderá aloca-lo. Se B for favorecido nesta disputa, alocando W, embora E permaneça em espera, W e Y serão liberados ao término de B
    4. Com Y livre, A aloca Y e, ao finalizar, libera X e Y
    5. Com W livre, E aloca W e, ao finalizar, libera W

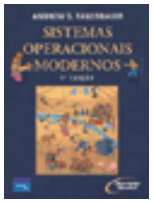
Detecção com um Recurso de Cada Tipo



É possível ocorrer um Deadlock ?

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

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

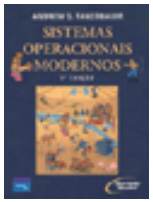
- Em um computador temos disponíveis:
  - 4 Unidades de fita
  - 2 Plotters
  - 3 Impressoras
  - 1 Unidade de CDROM
- Três processos são inicializados com as alocações:
  - P1 – 1 Impressora
  - P2 – 2 Unidades de Fita e 1 Unidade de CDROM
  - P3 – 1 Plotter e 2 Impressoras
- Ao longo da execução, os processos solicitam mais recursos para finalizar, sendo:
  - P1 – 2 Unidades de Fita e 1 Unidade de CDROM
  - P2 – 1 Unidades de Fita e 1 Impressora
  - P3 – 2 Unidades de Fita e 1 Plotter

Detecção – Algoritmo do Banqueiro

Algoritmo do Banqueiro para Múltiplos Recursos:

- Deve-se montar, portanto, matrizes (Unidimensionais e Bidimensionais) para que seja possível aplicar o algoritmo.
  - A Matriz Unidimensional E representa os totais de recursos existentes no computador
  - A Matriz Bidimensional Matriz de Alocação representa os recursos previamente alocados para cada processo
  - A Matriz Unidimensional P representa as somas de todos os recursos que já estão alocados, inicialmente, em todos os processos
  - A Matriz Unidimensional A representa os recursos que ainda estão disponíveis após a alocação inicial de todos os processos
  - A Matriz Bidimensional Matriz de Requisições representa todos os recursos que serão solicitados ao longo das execuções de cada processo

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	2	1	0	0

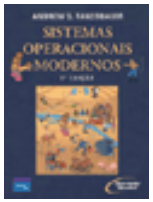
Matriz de Alocação

	UF	P	I	CD
P1	0	0	1	0
P2	2	0	0	1
P3	0	1	2	0

Matriz de Requisições

	UF	P	I	CD
P1	2	0	0	1
P2	1	0	1	0
P3	2	1	0	0

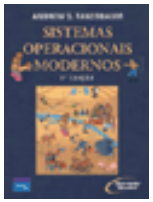
Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

- A dinâmica do algoritmo se dá da seguinte maneira, 1 processo por vez:
  1. Comparando, elemento a elemento do vetor A com cada linha da matriz de requisições para verificar, para quais processos, ainda existem recursos para que continuem rodando;
  2. No caso de 1 ou mais ocorrências, pega-se o primeiro processo possível, subtrai os valores da linha de requisição daquele processos no vetor A e soma os mesmos valores na linha correspondente ao processo na matriz de alocação;
  3. Considerando o processo encerrado, se faz a "devolução" dos valores da linha do processo no vetor A, somando-se com os valores que nesse vetor sobraram;
  4. Reinicia a comparação linha a linha proposta no passo 1 para verificar novas possibilidades de execução de processo.

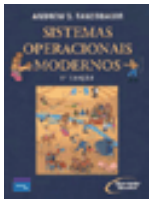
Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

- Caso nenhum processo consiga rodar com os recursos disponíveis demonstrados no vetor A, o conjunto está em um estado inseguro, portanto está em **DEADLOCK**.
- Em caso de ocorrência de DEADLOCK, pode-se retornar a passos anteriores onde havia mais de um processo possível de se executar e, ao invés de executar o primeiro, executar o segundo e verificar se ainda ocorre o DEADLOCK. Essa reversão pode ser feita quantas vezes forem necessárias para encontrar um caminho seguro possível, ou seja sem DEADLOCK.

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	2	1	0	0

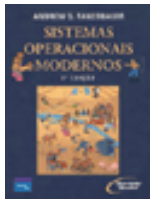
Matriz de Alocação

	UF	P	I	CD
P1	0	0	1	0
P2	2	0	0	1
P3	0	1	2	0

Matriz de Requisições

	UF	P	I	CD	
P1	2	0	0	1	X
P2	1	0	1	0	X
P3	2	1	0	0	V

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	2	1	0	0

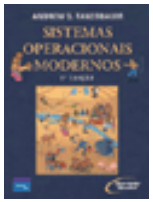
Matriz de Alocação

	UF	P	I	CD
P1	0	0	1	0
P2	2	0	0	1
P3	0	1	2	0

Matriz de Requisições

	UF	P	I	CD	
P1	2	0	0	1	X
P2	1	0	1	0	X
P3	2	1	0	0	V

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	0	0	0	0

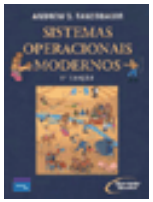
Matriz de Alocação

	UF	P	I	CD
P1	0	0	1	0
P2	2	0	0	1
P3	2	2	2	0

Matriz de Requisições

	UF	P	I	CD	
P1	2	0	0	1	X
P2	1	0	1	0	X
P3	-	-	-	-	V

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	2	2	2	0

Matriz de Alocação

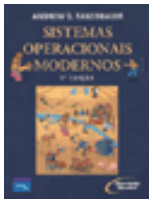
	UF	P	I	CD
P1	0	0	1	0
P2	2	0	0	1
P3	-	-	-	-

Matriz de Requisições

	UF	P	I	CD
P1	2	0	0	1
P2	1	0	1	0
P3	-	-	-	-

1o.

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	2	2	2	0

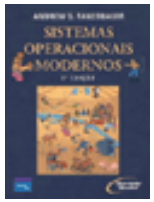
Matriz de Alocação

	UF	P	I	CD
P1	0	0	1	0
P2	2	0	0	1
P3	-	-	-	-

Matriz de Requisições

	UF	P	I	CD	
P1	2	0	0	1	X
P2	1	0	1	0	V
P3	-	-	-	-	1o.

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	1	2	1	0

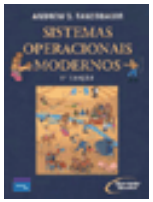
Matriz de Alocação

	UF	P	I	CD
P1	0	0	1	0
P2	3	0	1	1
P3	-	-	-	-

Matriz de Requisições

	UF	P	I	CD	
P1	2	0	0	1	X
P2	-	-	-	-	V
P3	-	-	-	-	1o.

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	4	2	2	1

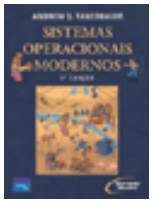
Matriz de Alocação

	UF	P	I	CD
P1	0	0	1	0
P2	3	0	1	1
P3	-	-	-	-

Matriz de Requisições

	UF	P	I	CD	
P1	2	0	0	1	
P2	-	-	-	-	2o.
P3	-	-	-	-	1o.

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	4	2	2	1

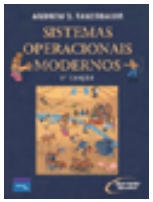
Matriz de Alocação

	UF	P	I	CD
P1	0	0	1	0
P2	-	-	-	-
P3	-	-	-	-

Matriz de Requisições

	UF	P	I	CD	
P1	2	0	0	1	
P2	-	-	-	-	2o.
P3	-	-	-	-	1o.

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	4	2	2	1

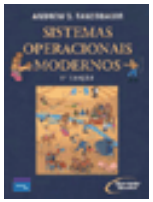
Matriz de Alocação

	UF	P	I	CD
P1	0	0	1	0
P2	-	-	-	-
P3	-	-	-	-

Matriz de Requisições

	UF	P	I	CD	
P1	2	0	0	1	V
P2	-	-	-	-	2o.
P3	-	-	-	-	1o.

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	2	2	2	0

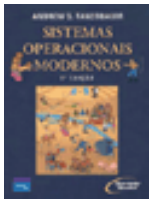
Matriz de Alocação

	UF	P	I	CD
P1	2	0	1	1
P2	-	-	-	-
P3	-	-	-	-

Matriz de Requisições

	UF	P	I	CD	
P1	-	-	-	-	V
P2	-	-	-	-	2o.
P3	-	-	-	-	1o.

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	4	2	3	1

Ao final: A = E, portanto

ESTADO SEGURO

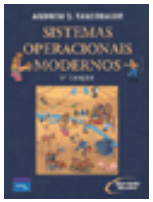
Matriz de Alocação

	UF	P	I	CD
P1	-	-	-	-
P2	-	-	-	-
P3	-	-	-	-

Matriz de Requisições

	UF	P	I	CD	
P1	-	-	-	-	3o.
P2	-	-	-	-	2o.
P3	-	-	-	-	1o.

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Caso 2:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	2	1	0	0

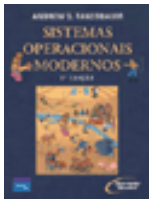
Matriz de Alocação

	UF	P	I	CD
P1	0	0	1	0
P2	2	0	0	1
P3	0	1	2	0

Matriz de Requisições

	UF	P	I	CD
P1	2	0	0	1
P2	1	0	3	0
P3	2	1	0	0

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Caso 2:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	2	1	0	0

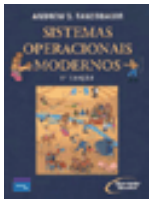
Matriz de Alocação

	UF	P	I	CD
P1	0	0	1	0
P2	2	0	0	1
P3	0	1	2	0

Matriz de Requisições

	UF	P	I	CD	
P1	2	0	0	1	X
P2	1	0	3	0	X
P3	2	1	0	0	V

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Caso 2:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	0	0	0	0

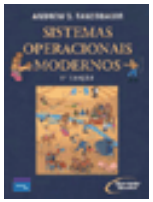
Matriz de Alocação

	UF	P	I	CD
P1	0	0	1	0
P2	2	0	0	1
P3	2	2	2	0

Matriz de Requisições

	UF	P	I	CD	
P1	2	0	0	1	X
P2	1	0	3	0	X
P3	-	-	-	-	V

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Caso 2:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	2	2	2	0

Matriz de Alocação

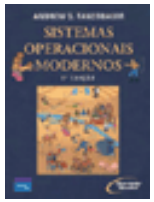
	UF	P	I	CD
P1	0	0	1	0
P2	2	0	0	1
P3	-	-	-	-

Matriz de Requisições

	UF	P	I	CD
P1	2	0	0	1
P2	1	0	3	0
P3	-	-	-	-

1o.

Detecção – Algoritmo do Banqueiro



Algoritmo do Banqueiro para Múltiplos Recursos:

Caso 2:

Recursos Existentes

	UF	P	I	CD
E	4	2	3	1

Recursos Alocados

	UF	P	I	CD
P	2	1	3	1

Recursos Disponíveis

	UF	P	I	CD
A	2	2	2	0

Ao final:

A ≠ E, e não existem reversões possíveis, portanto

ESTADO INSEGURO

Matriz de Alocação

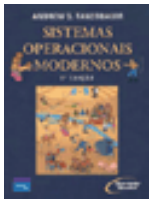
	UF	P	I	CD
P1	0	0	1	0
P2	2	0	0	1
P3	-	-	-	-

Matriz de Requisições

	UF	P	I	CD	
P1	2	0	0	1	X
P2	1	0	3	0	X
P3	-	-	-	-	1o.

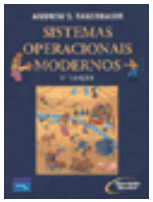
DEADLOCK

Recuperação de *Deadlock*



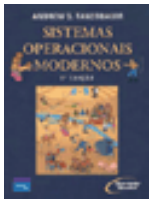
- Recuperação através de preempção
  - Retirar um recurso de algum outro processo
  - Depende da natureza do recurso
- Recuperação através de reversão de estado
  - Verifica um processo periodicamente
  - Usa este estado salvo
  - Reinicia o processo se este é encontrado em estado de *deadlock*
- Recuperação através da eliminação de processos
  - 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

Evitando *Deadlocks*



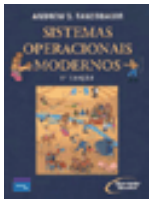
- Evitar dinamicamente o problema:
  - Alocação individual de recursos são, normalmente, feitos à medida que o processo necessita
  - Escalonamento cuidadoso → Alto Custo;
  - Algoritmos:
    - Banqueiro para único recurso
    - Banqueiro para múltiplos recursos
  - Usam a notação de Estados Seguros e Estados Inseguros

Evitando *Deadlocks*



- Prevenção Dinâmica
  - Estados Seguros: Existe uma maneira de atender às solicitações dos processos – Não há DEADLOCK
    - Em um estado seguro, todos os processos serão concluídos
  - Estados Inseguros: Podem Provocar DEADLOCKS
    - Não é possível garantir que os processos serão todos finalizados

Estados Seguros e Inseguros



	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

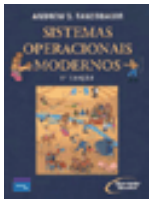
Disponível: 0
(d)

	Possui máx.	
A	3	9
B	0	–
C	0	–

Disponível: 7
(e)

Demonstração de que o estado em (a) é seguro

Estados Seguros e Inseguros



	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)

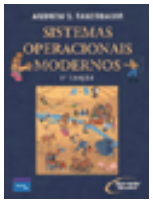
	Possui máx.	
A	4	9
B	–	–
C	2	7

Disponível: 4

(d)

Demonstração de que o estado em (b) é inseguro

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

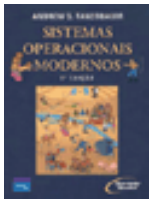
Disponível: 1

(c)

Três estados de alocação de recursos

a) seguro
b) seguro
c) inseguro

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*

Atacando a Condição de Exclusão Mútua

- Alguns dispositivos (como uma impressora) podem fazer uso de *spool*
  - O daemon de impressão é o único recurso da impressora
  - Desta forma deadlock envolvendo a impressora é eliminado
- Nem todos os dispositivos pode 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*

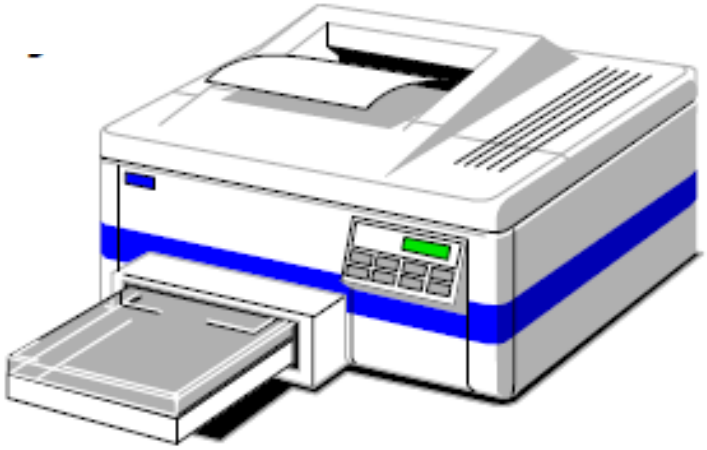
Atacando a Condição de Posse e Espera

- 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
  - Tem podem reter recursos que outros processos poderiam estar usando
- Variação
  - Processo deve desistir de todos os recursos para requisitar apenas os que são imediatamente necessários

Prevenção de *Deadlock*

Atacando a Condição de Não Preempção

- Opção inviável
- Considere um processo de posse de uma impressora
  - No meio da impressão
  - Retoma a impressora a força ?

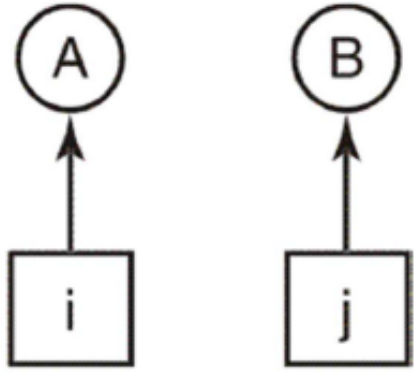


Prevenção de *Deadlock*

Atacando a Condição de Espera Circular

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

(a)

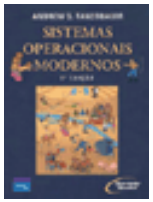


(b)

a) Recursos ordenados numericamente

b) Um grafo de recursos

Prevenção de *Deadlock*



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

Condição de Inanição - *Starvation*

- Não preemptivos
  - Algoritmo para alocar um recurso
    - Ceder para o *job* mais curto primeiro
    - Funciona bem para múltiplo Jobs curtos em um sistema
    - Jobs longos podem ser preteridos indefinidamente
    - Solução
      - Aplicar escalonamento FCFS
- Preemptivos
  - Evitar a mudança de prioridades
  - Deixar apenas para penalizações aplicadas para o S.O. em função de não atendimento da solicitação de mudança para estado Bloqueado
  - Threads de baixa prioridade são visitadas com menor frequência

Exercício

1) Verificar se o estado é seguro ou inseguro
   - Caso seguro: Propor a possível resolução
   - Caso inseguro: Apresentar onde se dará o *deadlock*
   - Sistema A tem 12 dispositivos, apenas 1 está disponível

Total 12 Recursos

	Possui	Max (Necessidade Total)
1	5	6
2	4	7
3	2	4
4	0	2
Disp		1

Exercício

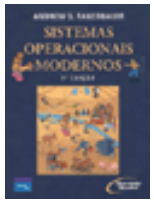
2) Verificar se o estado é seguro ou inseguro

- Caso seguro: Propor a possível resolução
- Caso inseguro: Apresentar onde se dará o *deadlock*
- Sistema B tem 14 dispositivos, apenas 2 estão disponíveis

Total 14 Recursos

	Possui	Max (Necessidade Total)
1	5	7
2	3	9
3	4	7
Disp		2

Exercício



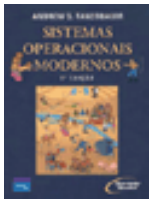
3) Verificar se o estado é seguro ou inseguro

- Caso seguro: Propor a possível resolução
- Caso inseguro: Apresentar onde se dará o *deadlock*
- Sistema C tem 10 dispositivos, apenas 2 estão disponíveis

Total 10 Recursos

	Possui	Max	(Necessidade Total)
1	3	5	
2	3	9	
3	2	8	
	Disp	2	

Exercício



4) Verificar se o estado é seguro ou inseguro

- Caso seguro: Propor a possível resolução
- Caso inseguro: Apresentar onde se dará o *deadlock*

Processo	Unidades de fita	Plotters	Scanners	Unidades de CD-ROM
A	0	1	0	0
B	2	0	1	1
C	0	1	0	2
D	2	0	0	0
E	0	1	0	2

Recursos alocados

Processo	Unidades de fita	Plotters	Scanners	Unidades de CD-ROM
A	1	1	0	2
B	0	2	0	1
C	1	0	2	0
D	0	2	0	2
E	2	0	1	0

Recursos ainda necessários

E(7426)
P(4315)
A(3111)