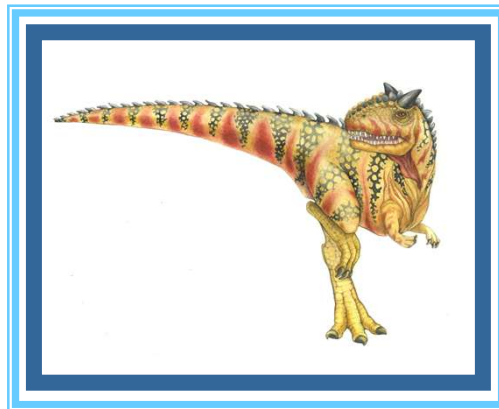


Capítulo 7: Deadlocks



Sobre a apresentação (About the slides)



Os slides e figuras dessa apresentação foram criados por Silberschatz, Galvin e Gagne em 2009. Essa apresentação foi modificada por Cristiano Costa (cac@unisinós.br). Basicamente, os slides originais foram traduzidos para o Português do Brasil.

É possível acessar os slides originais em <http://www.os-book.com>

Essa versão pode ser obtida em <http://www.inf.unisinós.br/~cac>

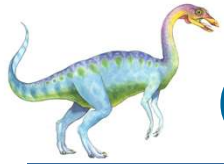


The slides and figures in this presentation are copyright Silberschatz, Galvin and Gagne, 2009. This presentation has been modified by Cristiano Costa (cac@unisinós.br). Basically it was translated to Brazilian Portuguese.

You can access the original slides at <http://www.os-book.com>

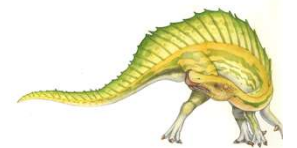
This version could be downloaded at <http://www.inf.unisinós.br/~cac>





Capítulo 7: Deadlock (Deadlocks)

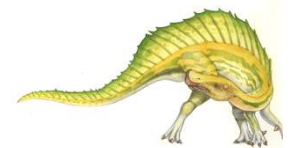
- O Problema do Deadlock
- Modelo de Sistema
- Caracterização de Deadlock
- Métodos para Manipular Impasses
- Prevenção de Deadlock
- Evitar Ocorrência de Deadlock
- Detecção de Deadlock
- Recuperação de Situação de Deadlock





Objetivos do Capítulo

- Desenvolver a descrição de um Deadlock, o qual impede um conjunto de processos concorrentes de completar suas tarefas
- Apresentar alguns métodos diferentes para prevenir ou evitar impasses em um sistema computacional.

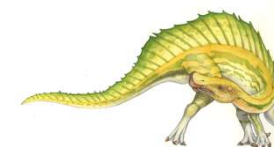


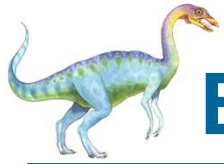


O Problema do Deadlock

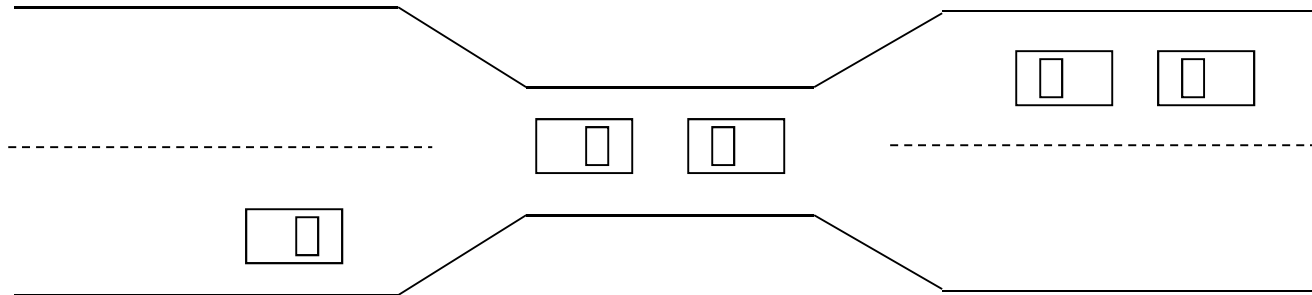
- Um conjunto de processos bloqueados cada um detendo um recurso e esperando para adquirir um recurso detido por outro processo no conjunto.
- Exemplo
 - Sistema tem 2 unidades de disco.
 - P_1 e P_2 detêm uma unidade de disco e necessitam de outra.
- Exemplo
 - semáforos A e B , iniciados em 1

P_0	P_1
wait (A);	wait(B)
wait (B);	wait(A)

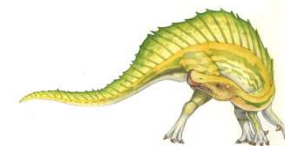


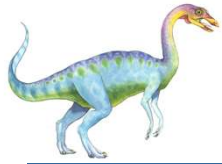


Exemplo do Cruzamento da Ponte



- Tráfego somente em uma direção.
- Cada seção da ponte pode ser vista como um recurso.
- Se ocorre um Deadlock, ele pode ser resolvido se um carro retorna de ré (preempta recurso e desfaz operação - *rollback*).
- Vários carros podem ter que retornar (dar ré) na ocorrência de *Deadlock*.
- *Starvation* (abandono) é possível.
- Nota – maioria dos SOs não previne ou trata Deadlocks



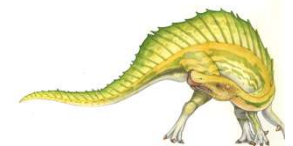


Modelo de Sistemas

- Tipos de Recursos R_1, R_2, \dots, R_m
ciclos de CPU, espaços de memória, dispositivos de E/S

- Cada tipo de recurso R_i tem W_i instâncias.

- Cada processo utiliza um recurso da seguinte forma:
 - **requisição**
 - **uso**
 - **liberação**

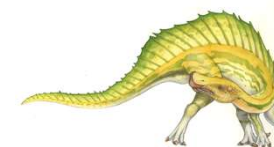




Caracterização de Deadlock

Deadlock pode ocorrer se quatro condições são satisfeitas simultaneamente.

- **Exclusão Mútua:** somente um processo de cada vez pode usar um recurso.
- **Posse e Espera:** um processo que está usando pelo menos um recurso e esperando que outros recursos, que estão nesse instante sendo usados por outros processos, sejam alocados para seu uso.
- **Inexistência de preempção:** um recurso só pode ser liberado voluntariamente pelo processo ao qual está alocado depois que o processo terminar de usá-lo.
- **Espera circular:** deve existir um conjunto $\{P_0, P_1, \dots, P_n\}$ de processos em espera, tal que P_0 esteja esperando por um recurso alocado a P_1 , P_1 que esteja esperando por um recurso alocado a P_2 , ..., P_{n-1} que esteja esperando por um recurso alocado a P_n , e P_n esteja esperando por um recurso alocado a P_0 .





Grafo de Alocação de Recursos

Um conjunto de vértices V e de arcos (edges) E .

- V está particionado em dois tipos:
 - $P = \{P_1, P_2, \dots, P_n\}$, conjunto que consiste de todos os processos no sistema.
 - $R = \{R_1, R_2, \dots, R_m\}$, conjunto que consiste de todos os tipos de recursos do sistema.
- **Arco de requisição** – arco dirigido $P_i \rightarrow R_j$
- **Arco de alocação** – arco dirigido $R_j \rightarrow P_i$





Grafo de Alocação de Recursos (Cont.)

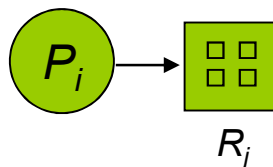
- Processos



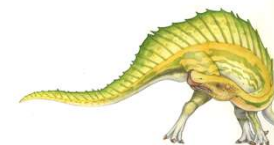
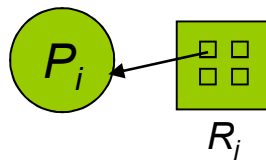
- Tipo de Recurso com 4 instâncias



- P_i solicita uma instância de R_j

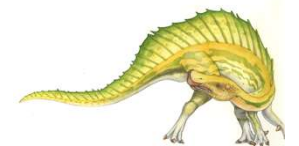
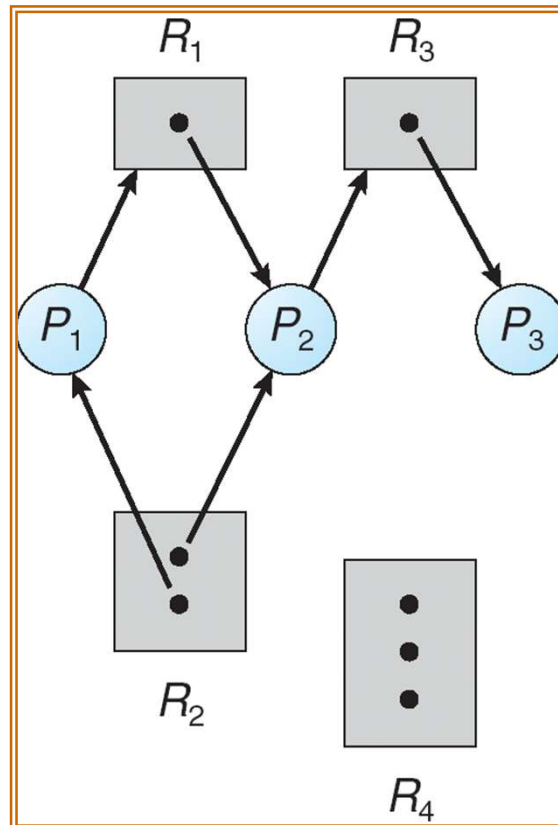


- P_i tem alocado uma instância de R_j



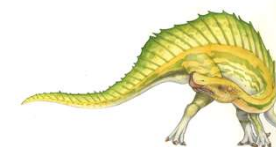
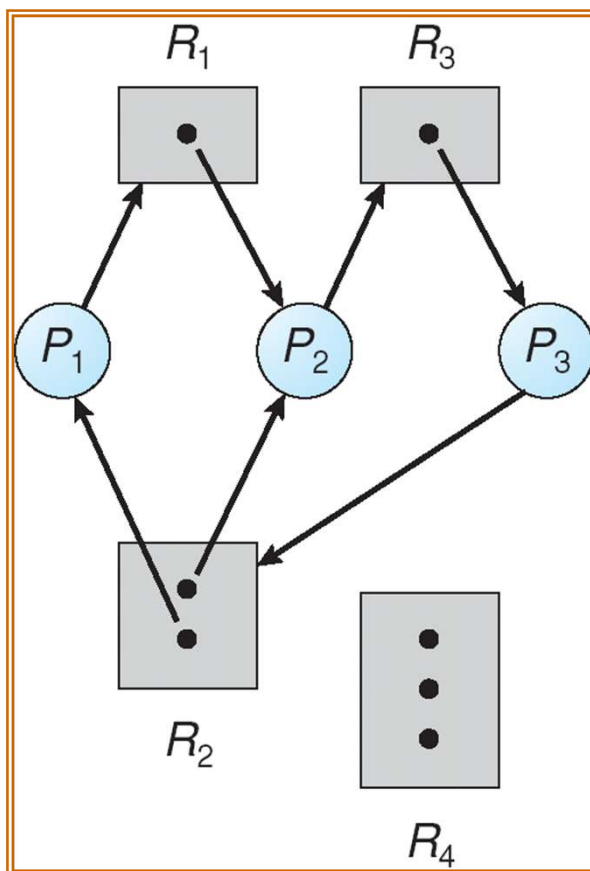


Exemplo de Grafo de Alocação de Recursos



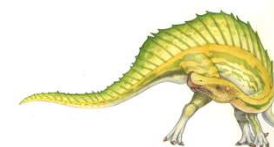
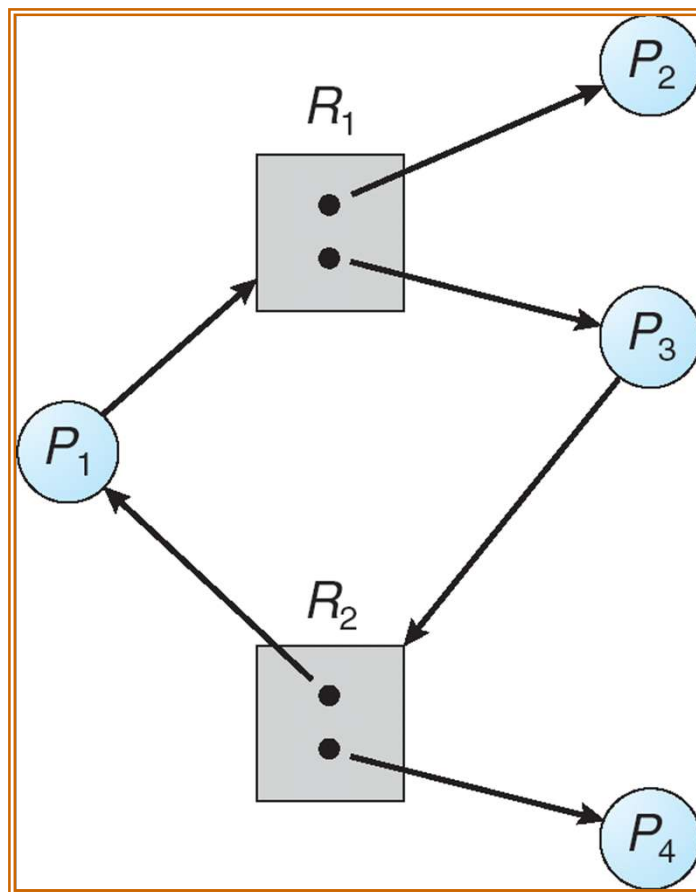


Grafo de Alocação de Recursos com Deadlock





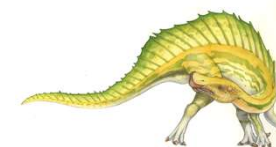
Grafo de Alocação de Recursos com um Ciclo, Mas sem Deadlock

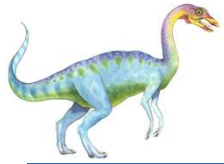




Fatos Básicos

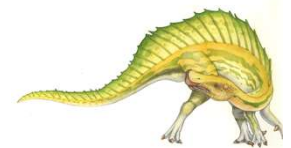
- Se um grafo não contém ciclos \Rightarrow não há *Deadlock*.
- Se um grafo contém um ciclo \Rightarrow
 - Se existe somente uma instância por tipo de recurso, então há *Deadlock*.
 - Se existem várias instâncias por tipo de recurso, ocorre a possibilidade de *Deadlock*.





Métodos de Tratar Impasses

- Garantir que o sistema **nunca** entrará em estado de Deadlock.
- Permitir ao sistema entrar em estado de Deadlock e então recuperar.
- Ignorar o problema e fingir que Deadlock nunca ocorrem no sistema; usado pela maior parte dos sistemas operacionais, incluindo o UNIX.



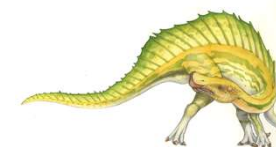


Prevenção de Deadlock

Restringir as formas em que as requisições podem ser feitas.

- **Exclusão mútua** – não é necessária para recursos compartilháveis; deve ocorrer para recursos não compartilháveis.

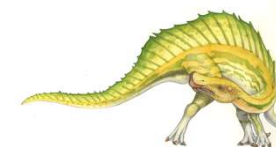
- **Posse e Espera** – deve garantir que sempre que um processo requerer um recurso, ele não esteja de posse de nenhum outro recurso.
 - Exigir que um processo faça a requisição de recursos e tenha todos os recursos alocados antes do início da execução, ou permitir que processos façam requisição de recursos somente quando não possuírem nenhum.
 - Baixa utilização de recursos; possibilidade de abandono (*starvation*).

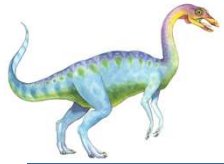




Prevenção de Deadlock (Cont.)

- **Inexistência de Preempção** –
 - Se um processo que está detendo algum recurso requerer outro recurso que não pode ser imediatamente alocado a ele, então todos os recursos correntemente alocados a este processo são liberados.
 - Recursos preemptados são adicionados à lista de recursos pelos quais o processo está esperando.
 - Processo irá ser reiniciado apenas quando ele puder usar todos os recursos de que precisa, tanto os que estavam alocados a esse processo anteriormente quanto os novos que ele esteja requisitando.
- **Espera Circular** – impor um ordenação total de todas as classes de recursos e requerer que a requisição de recursos por cada processo sempre ocorra em uma ordem crescente.

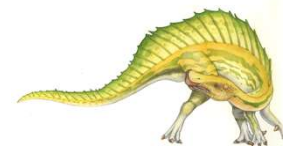




Evitar Ocorrência de Deadlock

Requer que o sistema tenha alguma informação adicional disponível antecipadamente.

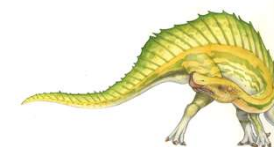
- O modelo mais simples e mais útil necessita que cada processo declare o *número máximo* de recursos de cada classe que podem ser necessários.
- O algoritmo para evitar Deadlock examina dinamicamente o estado da alocação de recursos para garantir que nunca haverá uma condição de espera circular.
- O *estado* da alocação de recursos é definido pelo número de recursos disponíveis e alocados, e a demanda máxima de recursos por processos.

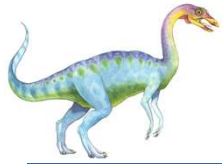




Estado Seguro

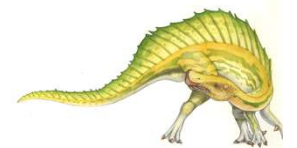
- Quando um processo solicita um recurso disponível para alocação, o sistema deve decidir se a alocação imediata leva o sistema para um estado seguro.
- Um sistema está em **estado seguro** se existe uma sequência $\langle P_1, P_2, \dots, P_n \rangle$ para TODOS os processos no sistema de forma que para cada processo P_i , os recursos que P_i pode ainda vir a solicitar serão satisfeitos pelos recursos atualmente disponíveis + recursos alocados por todos os processos P_j , sendo $j < i$.
- Isto é:
 - Se os recursos que P_i necessitar não estiverem imediatamente disponíveis, P_i pode esperar até que todos os processos P_j tenham terminado.
 - Quando P_j estiver terminado, P_i pode obter os recursos necessários, executar, liberar recursos alocados e terminar.
 - Quando P_i termina, P_{i+1} pode obter os recursos que necessita e assim por diante.

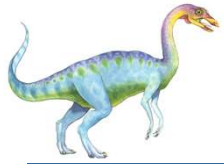




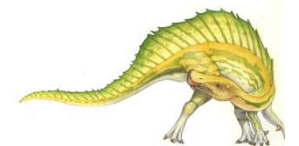
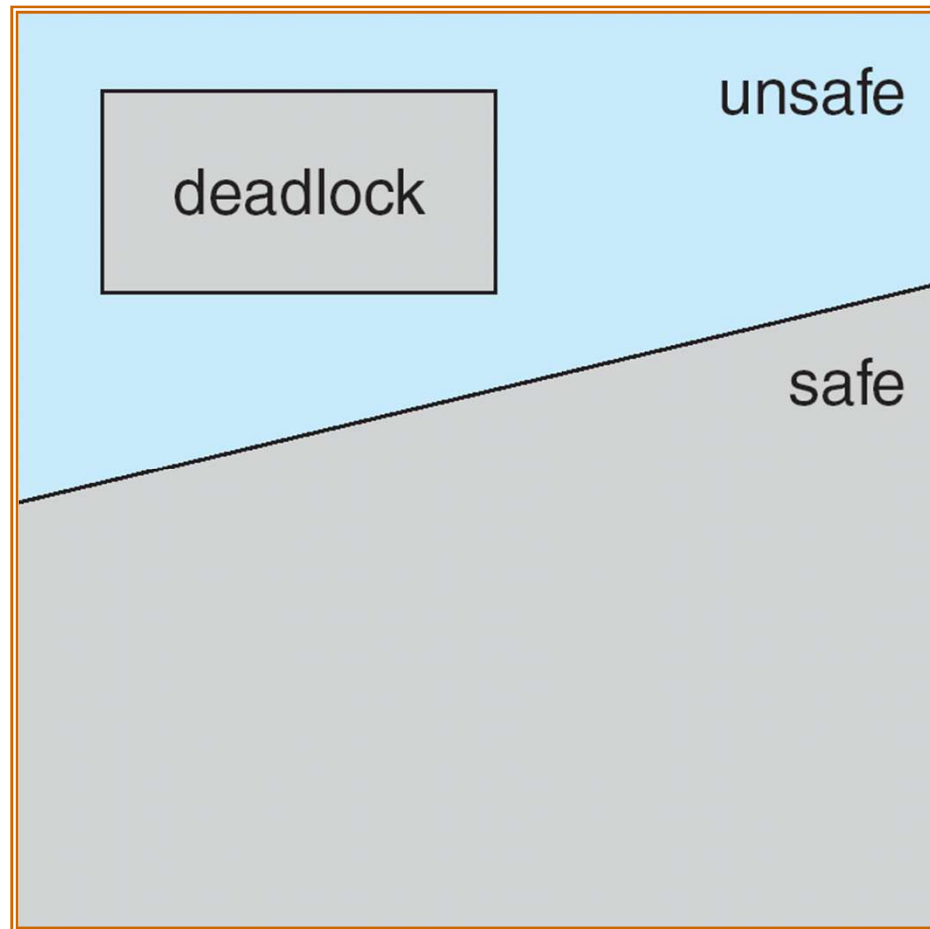
Fatos Básicos

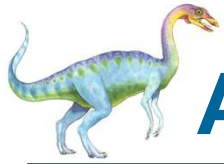
- Se o sistema está em um estado seguro \Rightarrow sem Deadlock.
- Se o sistema está em um estado inseguro \Rightarrow possibilidade de Deadlock.
- Evitar Deadlock \Rightarrow garantir que o sistema nunca vai entrar em um estado inseguro.





Estados Seguros, Não-Seguros e de Deadlock

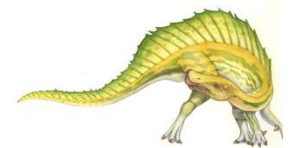




Algoritmos para Evitar Ocorrência

- Uma única instância de cada tipo de recurso.
 - Usa um grafo de alocação de recursos

- Múltiplas instâncias de um tipo de recurso.
 - Usa o algoritmo do banqueiro





Esquema do Grafo de Alocação de Recursos

- Um **arco de reivindicação** $P_i \rightarrow R_j$ indica que o processo P_i pode requisitar o recurso R_j ; e é representado por uma linha tracejada.
- Arcos de reivindicação são convertidos para *arcos de requisição* quando um processo efetivamente solicita um recurso.
- Arcos de requisição são convertidos para arcos de alocação quando o recurso é alocado ao processo.
- Quando um recurso é liberado por um processo, *arcos de alocação* são re-convertidos para arcos de reivindicação.
- Recursos devem ser reivindicados *a priori* em um sistema.

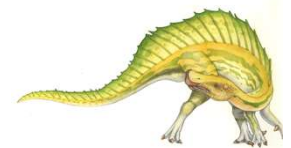
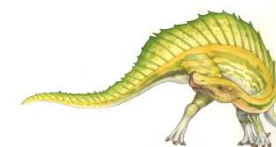
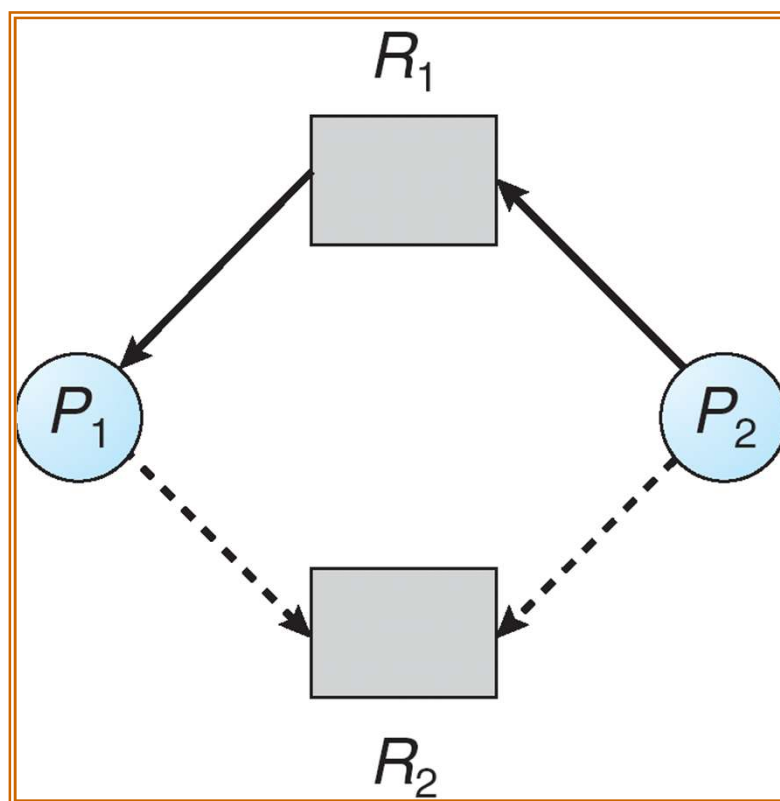


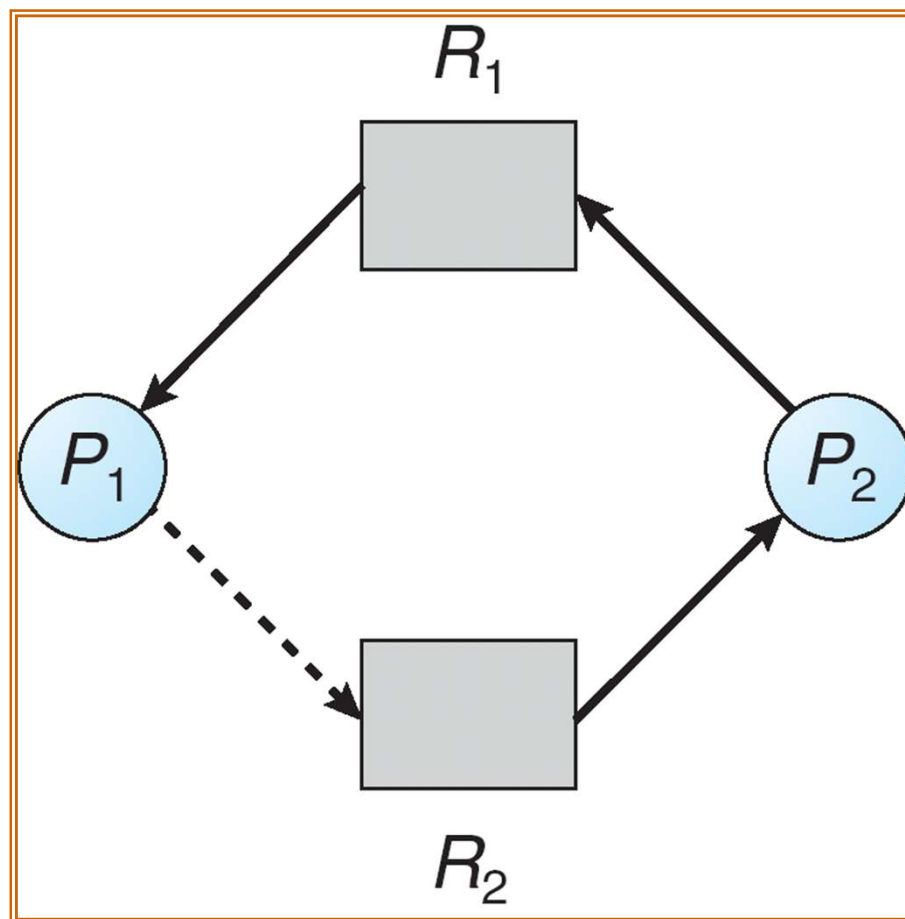


Gráfico de Alocação de Recursos





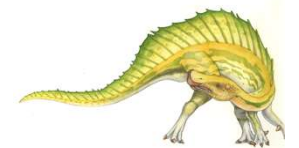
Um estado não-seguro no Grafo





Algoritmo do Grafo de Alocação de Recursos

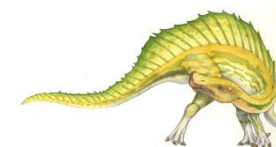
- Suponha que o processo P_i solicita o recurso R_j
- A requisição pode ser concedida somente se ao converter o arco de requisição em arco de alocação não resultar na formação de um ciclo no grafo de alocação de recursos





Algoritmo do Banqueiro

- Classes de recursos com mais de um recurso.
- Cada processo deve inicialmente declarar o número máximo de recursos que irá usar.
- Quando um processo requer um recurso ele pode ter de esperar.
- Quando um processo obtém todos os recursos que necessita, ele deve liberá-los em uma quantidade de tempo finita.





Estrutura de Dados do Algoritmo do Banqueiro

Seja n = número de processos, e m = número de classes de recursos.

- **Disponível (Disp)**: Vetor de tamanho m . Se $Disp [j] = k$, existem k recursos da classe R_j disponíveis.
- **Max**: uma matriz $n \times m$. Se $Max [i,j] = k$, então o processo P_i pode requerer no máximo k instâncias da classe de recursos R_j .
- **Alocação (Aloc)**: uma matriz $n \times m$. Se $Aloc [i,j] = k$ então P_i está no momento alocado k instâncias de R_j .
- **Necessidade (Ne)**: uma matriz $n \times m$. Se $Ne [i,j] = k$, então P_i pode necessitar mais k instâncias de R_j para completar sua tarefa.

$$Ne [i,j] = Max[i,j] - Aloc [i,j].$$





Algoritmo de Segurança

1. Sejam **trab** e **fim** vetores de tamanho m e n , respectivamente. Inicie com:

$trab := Disp$

$fim[i] = false$ para $i = 1, 2, \dots, n$.

2. Encontre i tal que ambos:

(a) $fim[i] = false$

(b) $Ne_i \leq trab$

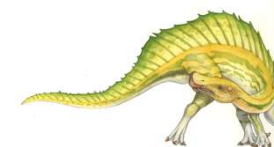
Se não existir tal i , vá para o item 4.

3. $trab := trab + Aloc$;

$fim[i] := true$

vá para o passo 2.

4. Se $fim[i] = true$ para todo i , o sistema está em um estado seguro.





Algoritmo de Requisição de Recursos para o Processo P_i

req_i = vetor de requisição para o processo P_i . Se $req_i[j] = k$ então P_i requer k recursos da classe R_j .

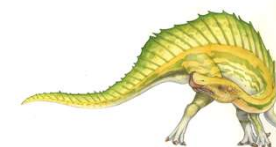
1. Se $Req_i \leq Ne_i$ vá para o passo 2. Se não, indique a ocorrência de um erro, uma vez que o processo excedeu seu número máximo de requisição.
2. Se $Req_i \leq Disp$, vá para o passo 3. Se não P_i deve esperar, uma vez que os recursos não estão disponíveis.
3. Finja alocar recursos requisitados para P_i modificando o estado como segue:

$$Disp := Disp - Req_i;$$

$$Aloc_i := Aloc_i + Req_i;$$

$$Ne_i := Ne_i - Req_i;$$

- Se seguro \Rightarrow os recursos são alocados para P_i .
- Se inseguro $\Rightarrow P_i$ deve esperar e o estado anterior, antes das atribuições feitas no passo 3, é recuperado

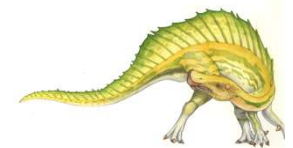




Exemplo do Algoritmo do Banqueiro

- 5 processos P_0 a P_4 ;
- 3 classes de recursos:
A (10 instâncias), B (5 instâncias), e C (7 instâncias).
- Situação no instante T_0 :

	<u>Aloc</u>	<u>Max</u>	<u>Disp</u>
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	



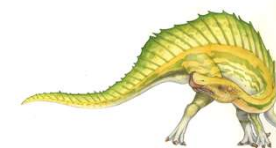


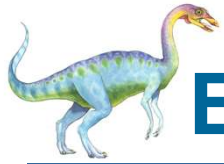
Exemplo (Cont.)

- Conteúdo da matriz Ne . Ne é definido como Max – Alloc.

	<u>Ne</u>		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

- O sistema está nesse instante em um estado seguro. De fato, a seqüência $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ satisfaz o critério de segurança.



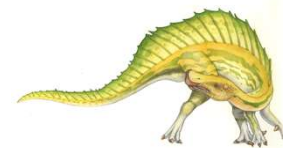


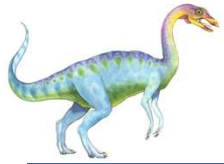
Exemplo (Cont.): P_1 solicita (1,0,2)

- Verifique se $Req \leq Disp$ (isto é, $(1,0,2) \leq (3,3,2) \Rightarrow true$).

	<u>Aloc</u>	<u>Ne</u>	<u>Disp</u>
	A B C	A B C	A B C
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 1	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

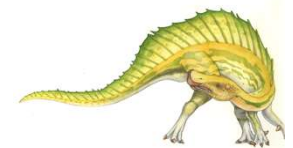
- Executando o algoritmo de segurança mostra que a seqüência $\langle P_1, P_3, P_4, P_0, P_2 \rangle$ satisfaz necessidade de segurança.
- Pode uma requisição de (3,3,0) por P_4 ser atendida?
- Pode uma requisição de (0,2,0) por P_0 ser atendida?





Detecção de Deadlock

- Permite que o sistema entre em estado de Deadlock
- Algoritmo de detecção
- Esquema de recuperação



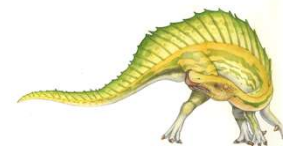


Unica Instância de cada Tipo de Recurso

- Manter um grafo de espera
 - Nodos são processados.
 - $P_i \rightarrow P_j$ se P_i está esperando por P_j .

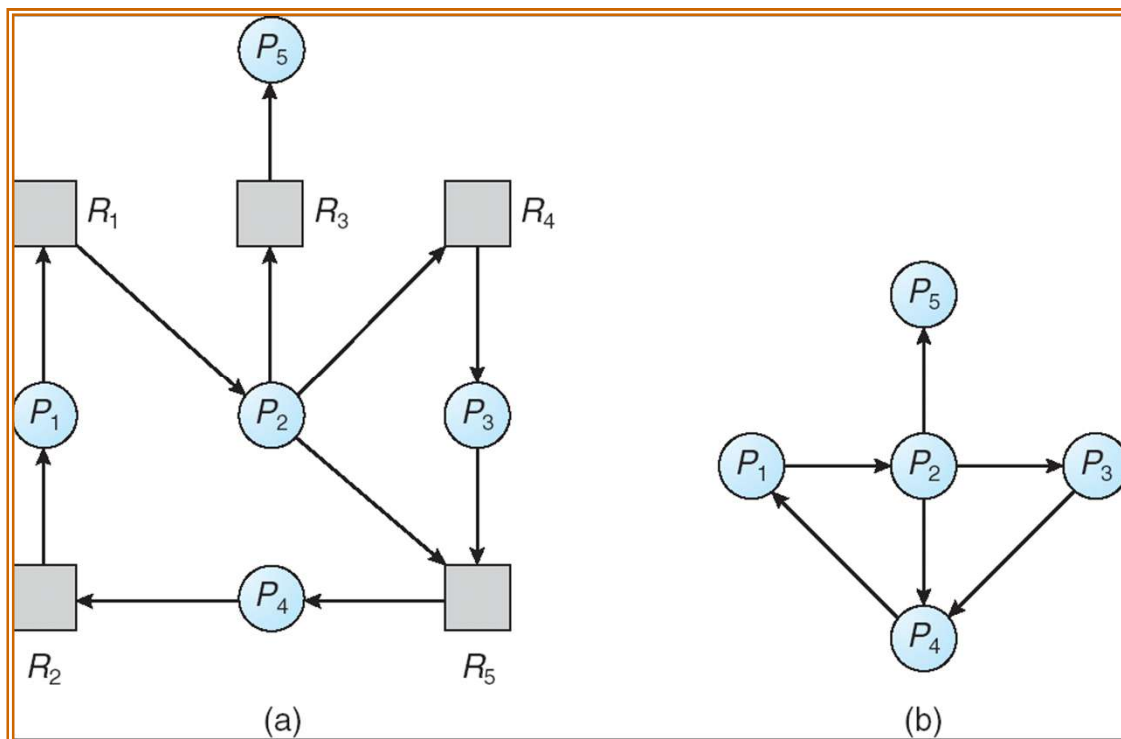
- Periodicamente usar um algoritmo para verificar se há ciclos no grafo. Se existem ciclos, existe Deadlock.

- Um algoritmo para detectar ciclos em um grafo requer algo na ordem de n^2 operações, onde n é o número de vértices no grafo.





Grafo de alocação de recursos e Grafo de espera correspondente



Grafo de alocação de recursos

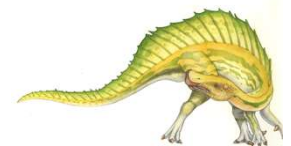
Grafo de espera correspondente





Várias Instâncias de um Tipo de Recurso

- **Disp**: Um vetor de tamanho m indica o número de recursos disponíveis de cada tipo.
- **Aloc**: Uma matriz $n \times m$ define o número de recursos de cada classe alocados a cada processo no instante atual.
- **Req**: Uma matriz $n \times m$ indica a requisição atual de cada processo. Se $Req [ij] = k$, então o processo P_i está requisitando mais k instâncias da classe R_j .





Algoritmo de Detecção

1. Sejam **Trab** e **Fim** vetores de tamanho m e n , respectivamente iniciados com:
 - (a) $Trab := Disp$
 - (b) Para $i = 1, 2, \dots, n$, se $Aloc_i \neq 0$, então $Fim[i] := false$; Se não, $Fim[i] := true$.

2. Ache um índice i tal que ambos:
 - (a) $Fim[i] = false$
 - (b) $Req_i \leq Trab$

Se não existe tal i , vá para o passo 4.

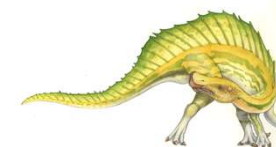




Algoritmo de Detecção (Cont.)

3. $Trab := Trab + Aloc_i$
 $Fim[i] := true$
vá para o passo 2.
4. Se $Fim[i] = false$, para algum i , $1 \leq i \leq n$, então o sistema está em estado de Deadlock. Além disso, se $Fim[i] = false$, então P_i está em Deadlock.

Esse algoritmo requer na ordem de $O(m \times n^2)$ operações para detectar se o sistema está em um estado de Deadlock.



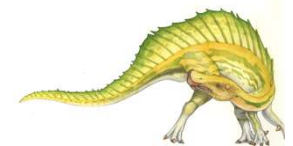


Exemplo de Algoritmo de Detecção

- Cinco processos P_0 a P_4 ; Três classes de recursos A (7 instâncias), B (2 instâncias), e C (6 instâncias).
- Situação no momento T_0 :

	<u>Aloc</u>			<u>Req</u>			<u>Disp</u>		
	<i>A</i>	<i>B</i>	<i>C</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>A</i>	<i>B</i>	<i>C</i>
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

- A seqüência $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ irá resultar em $Fim[i] = \text{true}$ para todo i .





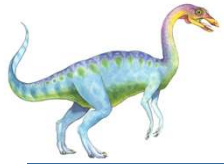
Exemplo (Cont.)

- P_2 solicita uma instância adicional de tipo C.

	<u>Req</u>		
	A	B	C
P_0	0	0	0
P_1	2	0	1
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

- Estado do sistema?
 - É possível liberar os recursos alocados ao processo P_0 , mas os recursos disponíveis não são suficientes para atender às requisições dos outros processos.
 - Existe Deadlock entre os processos P_1 , P_2 , P_3 e P_4 .

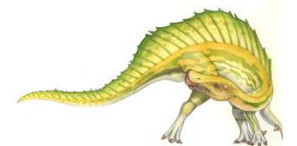




Uso do Algoritmo de Detecção

- Quando, e quão freqüentemente, usar o algoritmo depende de:
 - Com que freqüência ocorre um Deadlock?
 - Quantos processo serão afetados quando ocorre um Deadlock?
 - ▶ Um para cada ciclo disjunto

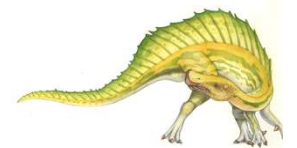
- Se algoritmo de detecção for usado arbitrariamente, podem ocorrer muitos ciclos no grafo de recursos e então não seria possível saber qual processo envolvido “causou” o Deadlock.





Recuperação de *Deadlock*: Terminação de Processo

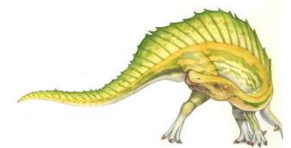
- Abortar todos os processos em *Deadlock*.
- Abortar um processo de cada vez até que o ciclo de *Deadlock* seja eliminado.
- Em que ordem deve-se abortar os processos?
 - Prioridade dos processos.
 - Há quanto tempo a execução do processo foi iniciada e quanto tempo mais será necessário para que o processo termine.
 - Recursos que o processo usou.
 - Recursos necessários para o processo terminar.
 - Términos de execução de processos que serão necessários.
 - Se o processo é interativo ou não.





Recuperação de Deadlock: Preempção de Recurso

- Selecionar uma vítima – minimiza o custo.
- Retornar (*Rollback*) – retornar para um estado seguro, reiniciar o processo deste ponto.
- Abandono (*Starvation*) – o mesmo processo pode ser sempre escolhido como vítima, incluir o número de *rollbacks* no fator de custo.



Fim do Capítulo 7

