Capítulo 7: Deadlocks

Capítulo 7: Deadlocks

- O problema do deadlock
- Modelo do sistema
- Caracterização do deadlock
- Métodos para tratamento de deadlocks
- Prevenção de deadlock
- Evitando deadlock
- Detecção de deadlock
- Recuperação de deadlock

Objetivos do capítulo

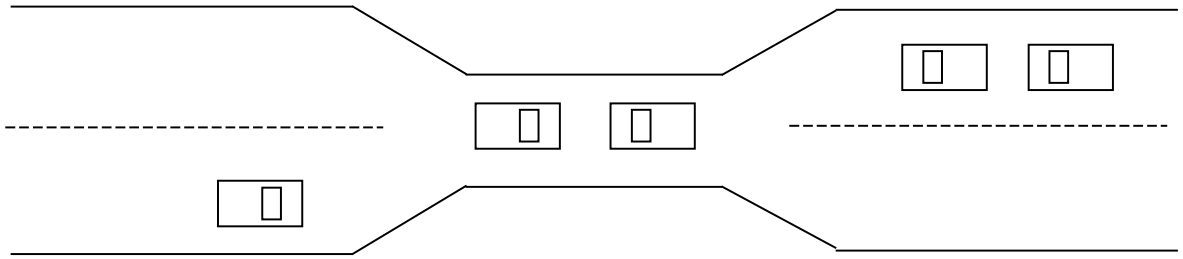
- Desenvolver uma descrição de deadlocks, que impedem que grupos de processos simultâneos completem suas tarefas.
- Apresentar diversos métodos para impedir ou evitar deadlocks em um sistema computadorizado.

O problema do deadlock

- Um conjunto de processos bloqueados, cada um mantendo um recurso e esperando para adquirir um recurso mantido por outro processo no conjunto.
- Exemplo
  - Sistema tem 2 unidades de disco.
  - P_1 e P_2 mantêm uma unidade de disco e cada um precisa da outra.
- Exemplo
  - semáforos *A* e *B*, inicializados com 1

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

Exemplo de cruzamento de ponte



- Tráfego apenas em uma direção.
- Cada seção de uma ponte pode ser vista como um recurso.
- Se houver deadlock, ele pode ser resolvido se um carro parar (apropriar recursos e reverter).
- Vários carros podem ter que parar se houver um deadlock.
- É possível haver starvation.

Modelo do sistema

- Tipos de recurso R_1, R_2, . . ., R_m

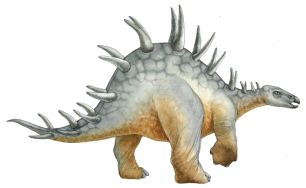
  Ciclos de CPU, espaço de memória, dispositivos de E/S

- Cada tipo de recurso R_i possui instâncias W_i.
- Cada processo utiliza um recurso da seguinte forma:
  - requisição
  - uso
  - liberação

Caracterização do deadlock

Deadlock pode surgir se 4 condições forem mantidas simultaneamente.

- **Exclusão mútua:** apenas um processo de cada vez pode usar um recurso.
- **Manter e esperar:** um processo mantendo pelo menos um recurso está esperando para adquirir outros recursos mantidos por outros processos.
- **Não preempção:** um recurso só pode ser liberado voluntariamente pelo processo que o mantém, depois que esse processo tiver terminado sua tarefa.
- **Espera circular:** existe um conjunto $\{P_0, P_1, \ldots, P_0\}$ de processos esperando tal que P_0 está esperando por um recurso que é mantido por P_1, P_1 está esperando por um recurso que é mantido por P_2, …, P_{n-1} está esperando por um recurso que é mantido por P_n, e P_0 está esperando por um recurso que é mantido por P_0.



grafo de alocação de recursos

Um conjunto de vértices V e um conjunto de arestas E.

- V é particionado em dois tipos:
  - $P = \{P_1, P_2, \ldots, P_n\}$, o conjunto consistindo em todos os processos no sistema.
  - $R = \{R_1, R_2, \ldots, R_m\}$, o conjunto consistindo em todos os tipos de recurso no sistema.

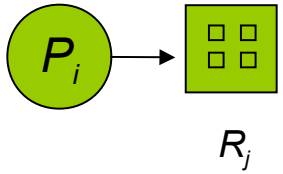
- arco de requisição – arco $P_1 \rightarrow R_j$
- arco de atribuição – arco $R_j \rightarrow P_i$

grafo de alocação de recursos (cont.)

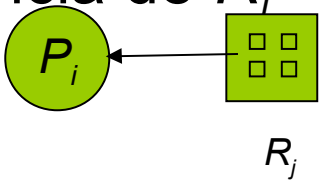
- Processo

- Tipo de recurso com 4 instâncias

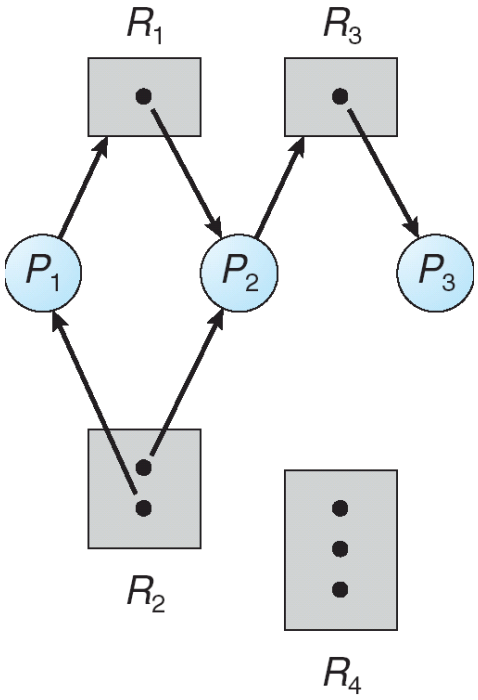
- P_i solicita instância de R_j



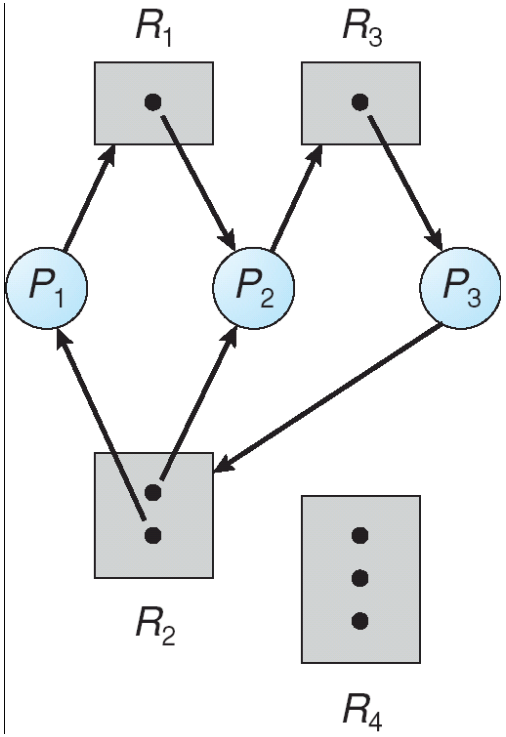
- P_i está mantendo uma instância de R_j



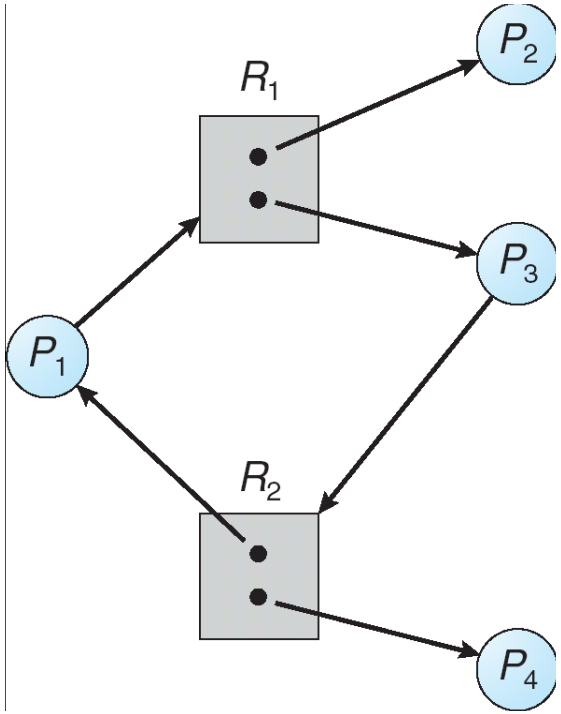
Exemplo de um grafo de alocação de recurso



grafo de alocação de recurso com um deadlock



grafo com um ciclo, mas sem deadlock



Fatos básicos

- Se o grafo não contém ciclos então não há deadlock.
- Se o grafo contém um ciclo então
  - se há apenas uma instância por tipo de recurso, então há deadlock
  - se várias instâncias por tipo de recurso, então há possibilidade de deadlock.

Métodos para tratamento de deadlocks

- Garantir que o sistema *nunca* entrará em um estado de deadlock.
- Permitir que o sistema entre em um estado de deadlock e depois se recupere.
- Ignorar o problema e fingir que os deadlocks nunca ocorrem no sistema; usado pela maioria dos sistemas operacionais, incluindo UNIX.

Prevenção de deadlock

Restringir as formas de como a requisição pode ser feita.

- **Exclusão mútua** – não exigido para recursos compartilháveis; deve manter para recursos não compartilháveis.
- **Manter e esperar** – deve garantir que sempre que um processo solicita um recurso, ele não mantém quaisquer outros recursos.
  - Exige que o processo solicite e tenha todos os seus recursos alocados antes de iniciar a execução, ou permite que o processo solicite recursos somente quando o processo não tiver recursos.
  - Starvation possível.

Prevenção de deadlock (cont.)

- **Preempção**
  - Se um processo que está mantendo alguns recursos solicitar outro recurso que não pode ser alocado imediatamente a ele, então todos os recursos atualmente sendo mantidos são liberados.
  - Recursos preemptados são acrescentados à lista de recursos pelos quais o processo está esperando.
  - O processo só será reiniciado quando puder reobter seus antigos recursos, além dos novos que está solicitando.
- **Espera circular** – impõe uma ordenação total de todos os tipos de recurso, e exige que cada processo solicite recursos em ordem crescente (considerando a ordem pré-fixada dos recursos).

Evitando deadlock

Exige que o sistema tenha alguma informação adicional *a priori*.

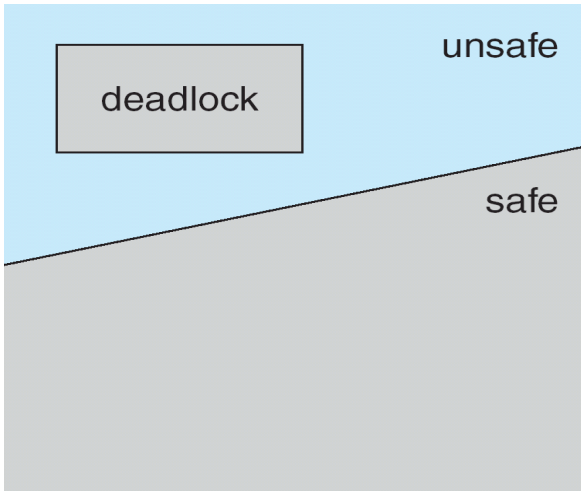
- Modelo mais simples e mais útil exige que cada processo declare o *número máximo* de recursos de cada tipo que ele pode precisar.
- O algoritmo para evitar impasse examina dinamicamente o estado de alocação de recurso para garantir que nunca poderá haver uma condição de espera circular.
- O *estado* de alocação de recurso é definido pelo número de recursos disponíveis e alocados, e as demandas máximas dos processos.

Estado seguro

- Quando um processo solicita um recurso disponível, o sistema deve decidir se a alocação imediata deixa o sistema em um estado seguro.
- O sistema está em estado seguro se houver uma seqüência $<P_1, P_2, \ldots, P_n>$ de todos os processos, tal que, para cada P_i, os recursos que P_i ainda pode solicitar possam ser satisfeitos pelos recursos atualmente disponíveis + recursos mantidos por todo P_j, com $j < i$.
- Ou seja:
  - se as necessidades de recurso de P_i não estiverem disponíveis imediatamente, então P_i pode esperar até que todo P_j tenha terminado.
  - Quando P_j tiver terminado, P_i pode obter os recursos necessários, executar, retornar recursos alocados e terminar.
  - Quando P_i termina, P_{i+1} pode obter seus recursos necessários, e assim por diante.

Fatos básicos

- Se um sistema estiver no estado seguro, então não há possibilidade de ocorrer deadlock.
- Se um sistema estiver em estado inseguro, então há **possibilidade** de ocorrer deadlock.



- Uma forma de evitar deadlocks (“cortar o mal pela raiz”) é garantir que o sistema nunca entrará em estado inseguro.

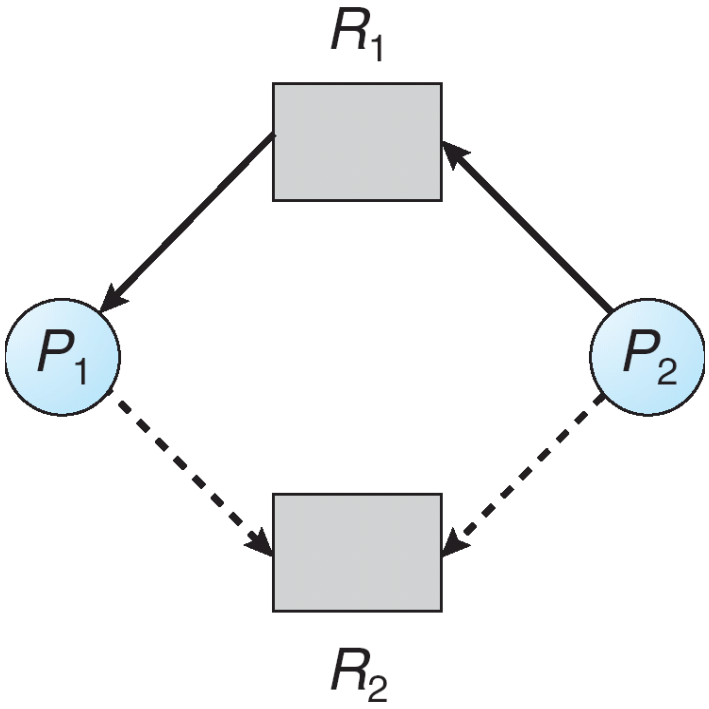
Algoritmos para evitar deadlock

- Única instância de um tipo de recurso. Use um grafo de alocação de recursos.
- Múltiplas instâncias de um tipo de recurso. Use o algoritmo do banqueiro.

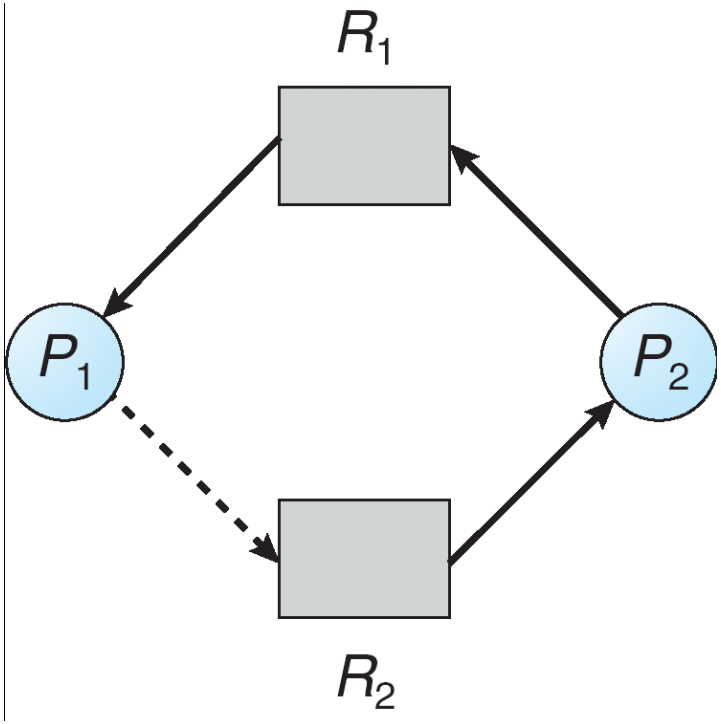
Esquema do grafo de alocação de recurso

- *arco de pretensão* $P_i \rightarrow R_j$ indica que o processo P_j pode solicitar recurso R_j; representado por uma linha tracejada.
- arco de pretensão converte para arco de requisição quando um processo requisita um recurso.
- arco de requisição convertida para um arco de atribuição quando o recurso é alocado ao processo.
- Quando um recurso é liberado por um processo, a arco de atribuição volta para um arco de pretensão.
- Os recursos devem ser reivindicados *a priori* no sistema.

grafo de alocação de recurso



Estado inseguro do grafo de alocação de recurso



Algoritmo do grafo de alocação de recurso

- Suponha que o processo P_i solicite um recurso R_j
- A requisição só pode ser concedida se a conversão da arco de requisição para um arco de atribuição não resultar na formação de um ciclo no grafo de alocação de recurso

Algoritmo do banqueiro

- Instâncias múltiplas.
- Cada processo precisa reivindicar uso máximo *a priori*.
- Quando um processo solicita um recurso, ele pode ter que esperar.
- Quando um processo apanha todos os seus recursos, ele precisa retorná-los em uma quantidade de tempo finita.

Estruturas de dados para o algoritmo do banqueiro

Seja *nP* = número de processos, e nR = número de tipos de recursos.

***Disponível**:* Vetor de tamanho *nR*. Se *Disponível*[*r*] = *k*, existem *k* instâncias do recurso *r* disponíveis.

***Máximo**:* Matriz nP *x nR*. Se *Máximo* [*p,r*] = *k*, então processo *p* pode solicitar no máximo *k* instâncias do recurso *r*.

***Alocação**:* Matriz *nP* x *nR*. Se *Alocação*[*p,r*] = *k* então *o processo p* alocou atualmente *k* instâncias do recurso *r*

***Necessário**:* Matriz *nP* x *nR*. Se *Necessário*[*p,r*] = *k*, então *o processo p* pode precisar de mais *k* instâncias do recurso *r* para completar sua tarefa.

Necessário [*p,r*] = *Máximo*[*p,r*] – *Alocação* [*p,r*].

Algoritmo de segurança (verifica se está em estado seguro)

1. Sejam ***Disp*** e ***Fim*** vetores de tamanho *nR* e *nP*, respectivamente. Inicialize:

   Disp[r] = *Disponível[r], para todo r*

   Fim [*p*] = *false,* para *todo processo p*

2. Encontre um processo *p* tal que:

   (a) *Fim* [*p*] = *false*

   (b) *Necessário[p,r]* <= Disp*[r], para todo recurso r*

   Se não existir *p*, vá para etapa 4.

3. *Disp[r]* = Disp[*r*] + *Alocação[p,r], para todo recurso r*
   Fim[*p*] = *true*
   vá para etapa 2.

4. Se *Fim* [*p*] == *true* para todo processo *p*, então o sistema está em um estado seguro.

Algoritmo de requisição de recurso para um processo *p*

// Se Requisição[p,r] = k então p deseja k instâncias do recurso r

1. Se *Requisição[p,r] <= Necessário[p,r], para todo r, então* vá para etapa 2. Caso contrário, ERRO (o processo ultrapassou sua pretensão máxima).
2. Se *Requisição[p,r] <= Disponível[r]*, para todo *r*, então vá para etapa 3. Caso contrário, *p* deve esperar, pois os recursos não estão disponíveis.
3. Finja ter alocado os recursos requisitados ao processo *p*, modificando o estado da seguinte maneira (para todo *r*):

   Disponível[r] = Disponível[r] – Requisição[p,r];

   Alocação[p,r] = Alocação[p,r] + Requisição[p,r];

   Necessário[p,r] = Necessário[p,r] – Requisição[p,r];

Se for seguro (roda o algoritmo de segurança para verificar) então os recursos são alocados a p.

Caso contrário, então p deve esperar e o antigo estado de alocação de recurso é restaurado.

Exemplo do algoritmo do banqueiro

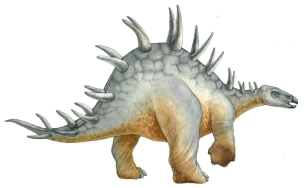
- 5 processos P_0 a P_4;

  3 tipos de recursos:

  A (10 instâncias), *B* (5 instâncias), e *C* (7 instâncias).

- Estado no instante T_0:

	Alocação	*Máximo*	*Disponível*
	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.)

- O conteúdo da matriz *Necessário* é definido como *Máximo – Alocação.*

	Necessário
	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 em estado seguro, pois a seqüência < P_1, P_3, P_4, P_2, P_0> satisfaz os critérios de segurança.

Exemplo: P_1 solicita (1,0,2)

- P_1 solicita (1,0,2)
- Verifique se requisição[P_1, r] <= Disponível[P_1, r], para todo r

	Alocação	*Necessário*	*Disponível*
	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	

- A execução do algoritmo de segurança mostra que a seqüência < P_1, P_3, P_4, P_0, P_2> satisfaz o requisito de segurança.
- A requisição de (3,3,0) por P_4 pode ser concedida?
- A requisição de (0,2,0) por P_0 pode ser concedida?

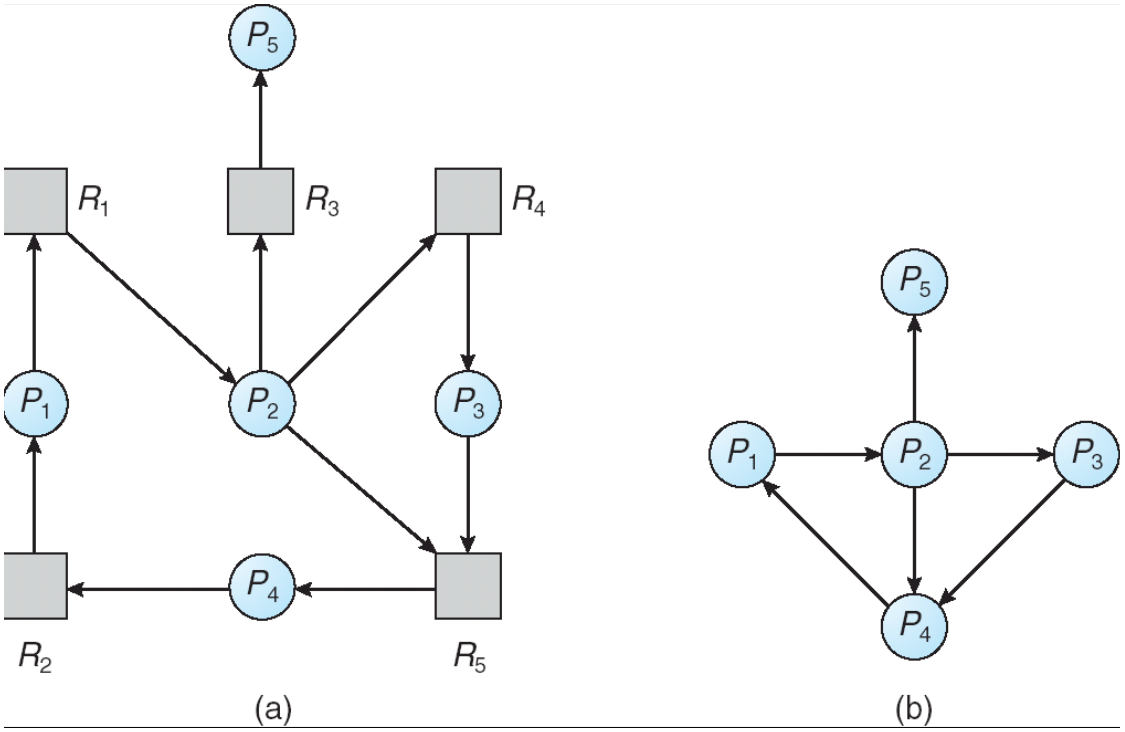
Detecção de deadlock

- Permita que o sistema entre no estado de deadlock
- Algoritmo de detecção
- Esquema de recuperação

Única instância de cada tipo de recurso

- Manter grafo de *espera (wait for)*
  - Nós são processos.
  - $P_i \rightarrow P_j$ se P_i estiver esperando por P_j.
- Periodicamente, chame um algoritmo que procure um ciclo no grafo. Se houver um ciclo, existe um deadlock.

grafo de alocação de recurso e grafo de espera



grafo de alocação de recurso

grafo de espera correspondente

Várias instâncias de um tipo de recurso

- ***Disponível**:* Um vetor de tamanho *m* indica o número de recursos de cada tipo.
- ***Alocação**:* Uma matriz *n* x *m* define o número de recursos de cada tipo atualmente alocados a cada processo.
- ***Requisição**:* Uma matriz *n* x *m* indica a requisição atual de cada processo. Se *Requisição*[*i,j*] = *k*, então o processo P_i está requisitando *k* mais instâncias do tipo de recurso R_j.

Algoritmo de detecção

1. Sejam *Trabalho* e *Fim* vetores de tamanhos *m* e *n*, respectivamente. Inicialize:
   (a) *Trabalho[r] = Disponível[r], para todo r*
   (b) Para todo processo *p*
   Se *Alocação[p, r]* == 0 (para todo *r*)
   Então *Fim[p] = true*
   Senão *Fim[p] = false*
2. Encontre um processo *p* tal que:
   (a) *Fim[p] == false*
   (b) *Requisição[p, r] <= Trabalho[p, r], para todo r*
   Se não houver um tal *p*, vá para etapa 4.

Algoritmo de detecção (cont.)

3. *Trabalho[r] = Trabalho[r] + Alocação[p, r],*
   para todo r
   Fim[p] = true
   vá para etapa 2.
4. Se *Fim[p] == false*, para algum processo *p*, então o sistema está em estado de deadlock. Além do mais, se *Fim[p] == false*, então o processo *p* está "travado" no deadlock.

Similar ao algoritmo do banqueiro, porém ao invés de considerar o máximo que os processos podem solicitar de cada recurso, considera as requisições que de fato foram feitas mas que estão pendentes

Exemplo de algoritmo de detecção

- Cinco processos de P_0 a P_4
- Três tipos de recurso A (7 instâncias), *B* (2 instâncias) e *C* (6 instâncias)
- No instante T_0:

	Alocação	*Requisição*	*Disponível*
	A B C	*A B C*	*A B C*
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	

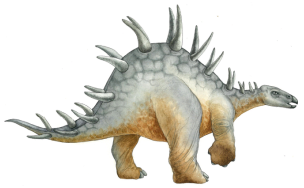
- Seqüência <P_0, P_2, P_3, P_1, P_4> resultará em *Fim*[*i*] = true para todo *i*

Exemplo (cont.)

- P_2 requer uma instância adicional do tipo *C*.

	Requisição
	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?
  - Pode reivindicar recursos mantidos pelo processo P_0, mas recursos insuficientes para atender os demais processos
  - Existe deadlock envolvendo os processos P_1, P_2, P_3, e P_4.



Uso do algoritmo de detecção

- Quando e com que freqüência invocar depende de:
  - Com que freqüência um deadlock provavelmente ocorrerá?
  - Quantos processos terão que ser revertidos?
    - um para cada ciclo
- Se algoritmo de detecção for invocado arbitrariamente, pode haver muitos ciclos no grafo de recursos e, portanto, não poderíamos saber quais dos muitos processos em deadlock "causou" o deadlock.

Recuperação de deadlock: término do processo

- Aborte todos os processos em deadlock.
- Aborte um processo de cada vez até que o ciclo de deadlock seja eliminado.
- Em que ordem devemos decidir abortar?
  - Prioridade do processo.
  - Por quanto tempo o processo foi executado, e quanto falta para terminar.
  - Recursos que o processo utilizou.
  - Recursos que o processo precisa para completar.
  - Quantos processos terão que ser terminados.
  - O processo é interativo ou batch?

Recuperação de deadlock: preempção de recursos

- Seleção de uma vítima – minimizar custo.
- Reversão – retornar a algum estado seguro, reiniciar processo para esse estado.
- Starvation – algum processo pode sempre ser apanhado como vítima; incluir número de rollbacks no fator de custo.

Final do Capítulo 7