

# Sistemas Operacionais

Unidade 2: Gerência de Processos  
– *Deadlock*

# Introdução

- Os sistemas de computadores têm inúmeros recursos adequados ao uso de somente um processo a cada vez.
- Em ambiente de multiprogramação diversos processos podem competir por um número finito de recursos.
- Um processo em espera (bloqueado) não poderá mudar de estado enquanto estiver aguardando algum recurso alocado por outro processo também em espera (bloqueado).

# Introdução

- Tipos de Recursos:

- *Preemptíveis*: podem ser retirados do processo por uma entidade externa(SO, SGBD, etc). Ex: memória, processador
- *Não-preemptíveis*: não podem ser retirados do processo. O processo que os possui, libera-os de livre e espontânea vontade.

- Os *deadlocks* ocorrem normalmente com recursos não preemptíveis.

# Exemplo de *Deadlock*

- Banco de Dados:
  - Um processo A bloqueia o registro R1
  - Um processo B bloqueia o registro R2
  - O processo A entra em espera pois precisa utilizar o registro R2 e
  - O processo B entra em espera pois precisa utilizar o registro R1
- Nesse ponto, esses dois processos vão ficar bloqueados, e assim permanecerão para sempre.
- Essa situação é chamada de **Deadlock**.

# Definição de *Deadlock*

- Um conjunto de processos estará em situação de *deadlock* se todos os processos pertencentes ao conjunto estiver esperando por um evento que somente um outro processo desse mesmo conjunto poderá fazer acontecer.
- O número de processos, bem como, o número e tipo dos recursos não são importantes.
- Isso é válido para qualquer tipo de recurso, tanto para hardware como para software.

# Condições para *Deadlock*

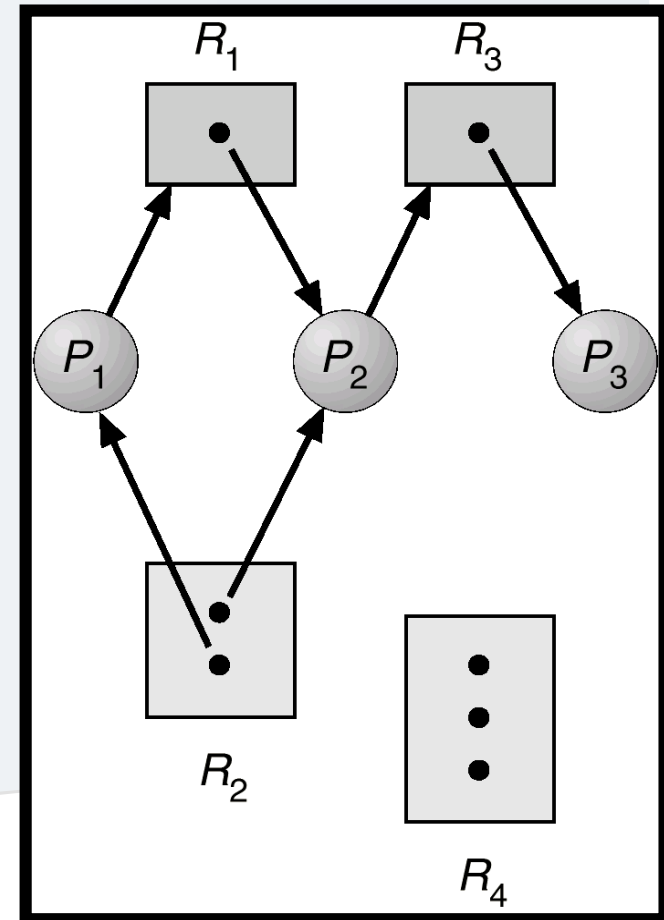
- Há quatro condições para que ocorra um *deadlock*:
  - Exclusão mútua: apenas um processo de cada vez pode utilizar o recurso.
  - Prende e espera: um processo bloqueia os recursos que precisa, e aguarda pelos que estão sendo utilizados por outros processos.
  - Não preempção: um recurso pode ser liberado apenas voluntariamente pelo processo, após o mesmo ter completado sua tarefa.
  - Espera circular: cada um dos processos espera um recurso que está sendo usado por outro processo em uma fila.

# Modelagem de *Deadlock*

- Uma situação de *deadlock* pode ser facilmente modelada por meio de um grafo de alocação de recursos.
- Consiste de um conjunto de vértices V e de arestas A:
  - Vértices – dois tipos
    - Nodos processo (simbolizados por círculos)
    - Nodos recursos (simbolizados por retângulos)
  - Arestas
    - Uma aresta de um recurso para um processo indica que o recurso está atualmente sendo usado pelo referido processo.


# Grafo de Alocação de Recursos


- Representação
  - Processos (círculos)
  - Recursos (retângulos)
  - Instâncias dos recursos (pontos)
- Uma aresta orientada é denotada
  - $P_1 \rightarrow R_1$
  - E é denominada **arco**

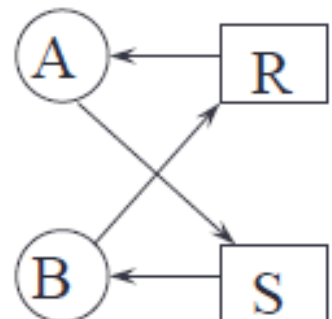




# Grafo de Alocação de Recursos

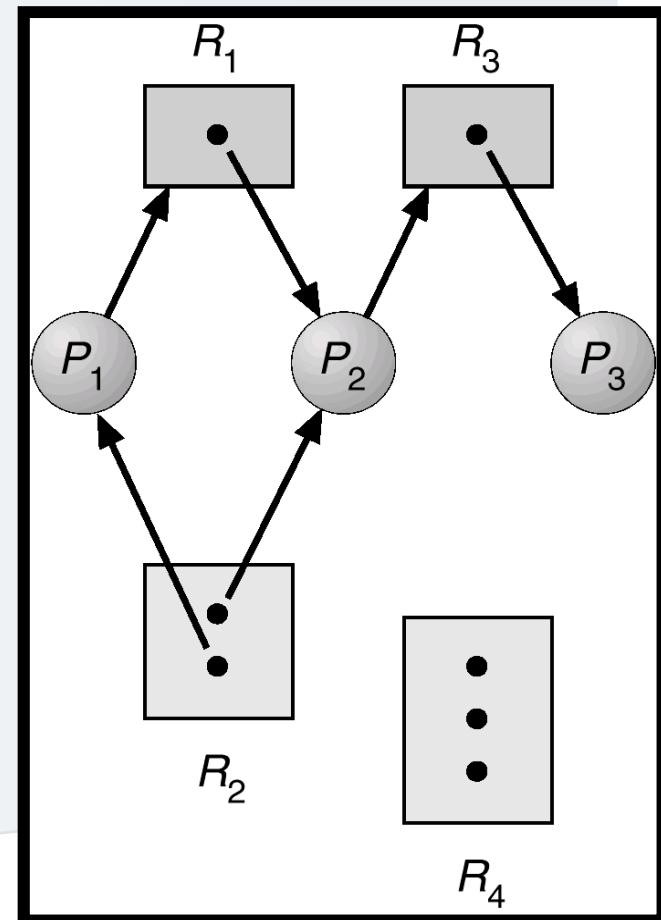
 : o recurso R está alocado ao processo A

 : o processo A deseja obter o recurso R e está bloqueado esperando a sua liberação

 :deadlock envolvendo dois processos

# Grafo de Alocação de Recursos

- P1 está bloqueando R2 e aguardando R1
- P2 está bloqueando R1 e R2 e aguardando R3
- P3 está bloqueando R3



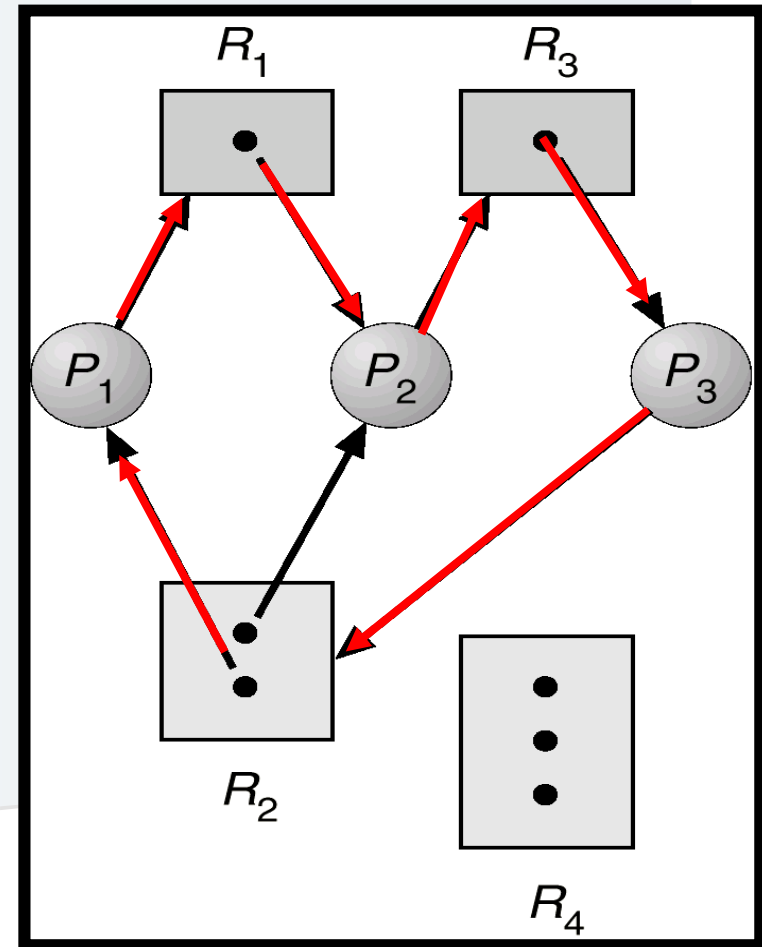
# Grafo de Alocação de Recursos

- O grafo de alocação de recursos pode ser utilizado para mostrar que, se
  - O grafo
    - Não contém ciclos, nenhum processo está em *deadlock*
    - Contém ciclos, há ocorrência de *deadlock*
  - Cada tipo de recurso possui
    - Apenas uma instância, um ciclo implica em ocorrência de *deadlock*
    - Várias instâncias, um ciclo não implica necessariamente em *deadlock*

# Grafo de Alocação de Recursos

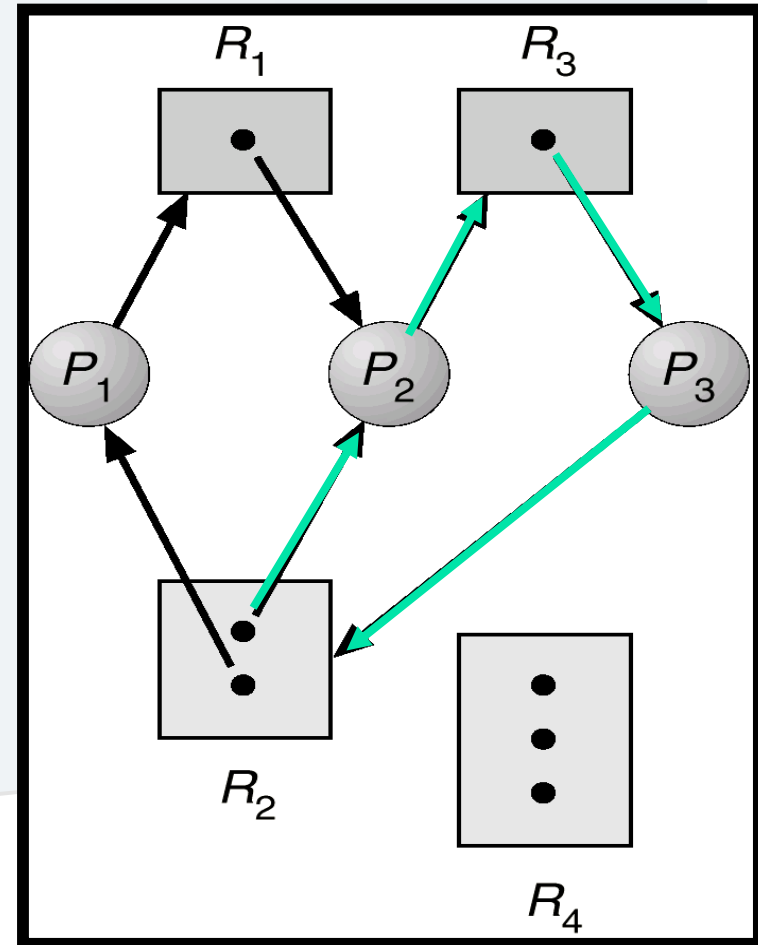
- Exemplo:

- Se o P3 solicitar R2, podem ocorrer pelo menos dois ciclos:
- $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
- $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$



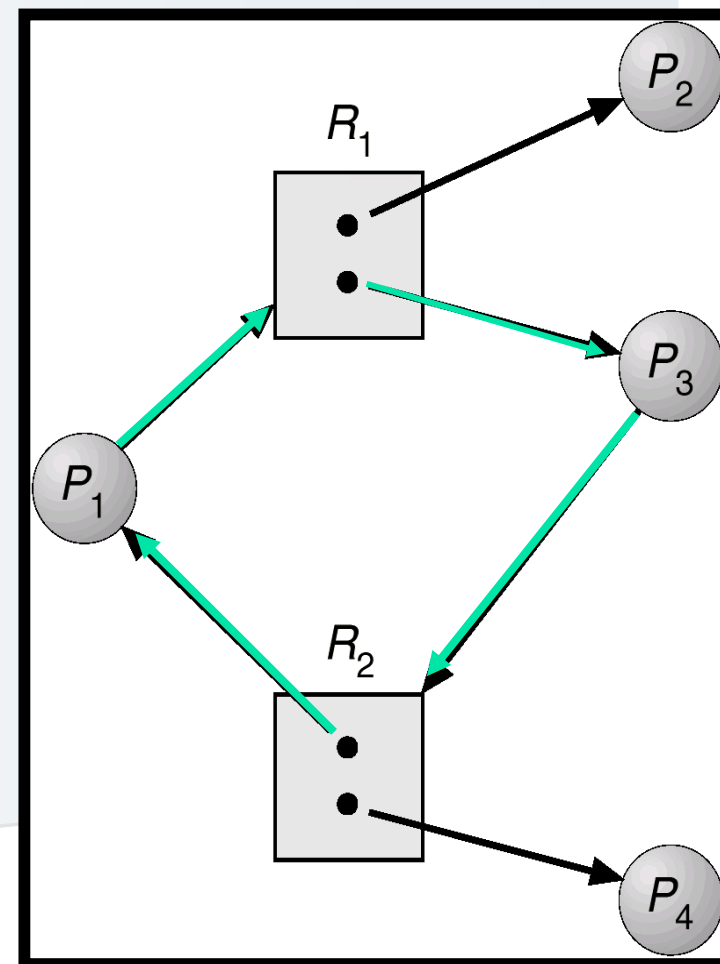
# Grafo de Alocação de Recursos

- Exemplo:
  - Se o  $P_3$  solicitar  $R_2$ , podem ocorrer pelo menos dois ciclos:
    - $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
    - $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$



# Grafo de Alocação de Recursos

- Nesse grafo de alocação é possível observar a ocorrência de um ciclo, mas não há *deadlock*.
- $P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$



# Métodos para lidar com *Deadlock*

- Usar um protocolo para garantir que um sistema nunca entrará em estado de *deadlock* (prevenção).
- Permitir que o sistema entre em *deadlock*, detectá-lo e recuperá-lo.
- Ignorar o problema, fingindo que o *deadlock* nunca acontecerá no sistema.

# Algoritmo Avestruz

- O método mais simples é o do algoritmo do avestruz: enterre a cabeça na areia e finja que nada está acontecendo.
- A maioria dos sistemas operacionais, incluindo Unix e Windows, adotam essa técnica.
  - Supondo que a maior parte dos usuários preferiria um deadlock ocasional a uma regra que restrinja cada usuário a somente um processo, um arquivo aberto e um de cada recurso.
- Se fosse possível eliminar *deadlock* sem custo, não haveria nenhuma discussão.



# Prevenção de *Deadlock*

- Garantir que pelo menos uma das quatro condições não possa ocorrer, pode-se prevenir a ocorrência de um *deadlock*.
- Exclusão Mútua
  - Se nunca acontecer de um recurso ser alocado exclusivamente a um único processo, nunca haverá *deadlock*.
  - Recursos compartilháveis, como um arquivo aberto para leitura, não exigem acesso por exclusão mútua e, portanto, não podem estar envolvidos em um *deadlock*.
  - No entanto, em geral, não é possível prevenir *deadlock* negando a condição de exclusão mútua, pois alguns recursos necessitam dessa garantia.

# Prevenção de *Deadlock*

## ■ Manter e Esperar

- Se pudermos impedir que processos que já mantêm a posse de recursos esperem por mais recursos, somos capazes de eliminar *deadlocks*.
- Uma solução seria usar um protocolo que exige que cada processo requisiite e receba a alocação de todos os seus recursos antes de iniciar sua execução.
- Outra solução é um protocolo que só permite que um processo requisiite um recurso apenas quando não tiver nenhum outro.
  - É possível haver **starvation** (espera eterna), pois um processo que precise de vários recursos populares pode ter de esperar indefinidamente, pois pelo menos um dos recursos de que precisa sempre estará alocado a algum outro processo.

# Prevenção de *Deadlock*

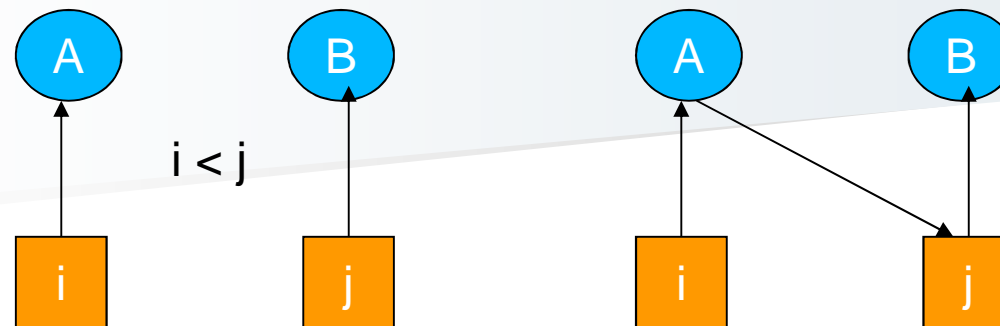
## ■ Preempção

- Se um processo estiver mantendo alguns recursos e requisitar outro recurso que não possa ser alocado imediatamente a ele, então todos os recursos retidos são liberados.
- O processo é reiniciado somente quando puder reaver seus recursos antigos, assim como os novos.
- Esse protocolo só pode ser aplicado a recursos cujo estado pode ser salvo e restaurado mais tarde.

# Prevenção de *Deadlock*

## ■ Espera Circular

- Uma maneira de garantir que essa condição nunca aconteça é impor uma ordenação total de todos os tipos de recursos e exigir que cada processo requisiite recursos em uma ordem crescente (ou decrescente) de enumeração.
- Com essa regra, o grafo de alocação de recursos nunca conterá ciclos.



# Prevenção de *Deadlock*

- Os algoritmos de prevenção de *deadlock* funcionam restringindo o modo como as requisições são feitas.
- As restrições garantem que pelo menos uma das condições necessárias para o *deadlock* não possa ocorrer.
- Contudo, os efeitos colaterais de prevenir *deadlocks* dessa maneira são a baixa utilização de dispositivos e a redução do *throughput* do sistema.
- Um método alternativo para evitar *deadlocks* é exigir informações adicionais sobre como os recursos devem ser requisitados.

# Prevenção de *Deadlock*

- Para prevenir *deadlocks* dessa maneira é necessário que os algoritmos de alocação, para cada requisição, decida se o processo deverá ou não esperar.
- Cada requisição exige que, ao tomar essa decisão, o sistema considere os recursos disponíveis, os recursos alocados a cada processo e as requisições e liberações futuras de cada processo.
- Os diversos algoritmos que usam essa técnica diferem na quantidade e no tipo de informações exigidas.

# Detecção e Recuperação de *Deadlock*

- Nessa técnica, o sistema não tenta prevenir a ocorrência de *deadlock*.
- Em vez disso, ele deixará que ocorram e tentará detectá-los à medida que isso acontecer.
- Agirá, então, de alguma maneira para se recuperar após o fato.
- Nessa técnica, o sistema deverá prover:
  - Um algoritmo que examine o estado do sistema para determinar se ocorreu um *deadlock*;
  - Um algoritmo para se recuperar do *deadlock*.

# Detecção e Recuperação de *Deadlock*

- Para detecção de *deadlock* pode ser usado qualquer algoritmo para descoberta de ciclos em grafos dirigidos.
- Para recuperação de *deadlock* há três opções:
  - Recuperação por meio de preempção:
    - A habilidade para retirar um recurso de um processo, entregá-lo a outro processo e depois devolvê-lo ao primeiro, sem que o processo perceba, é altamente dependente da natureza do recurso.
  - Abortar os processos:
    - Abortar todos os processos em *deadlock*;
    - Abortar um processo de cada vez até que o ciclo de *deadlock* seja eliminado.
  - Recuperação por meio de reversão de estado (*rollback*):
    - Fazer *checkpoint* de um processo, de modo que ele possa ser reiniciado, posteriormente, a partir do momento imediatamente anterior ao pedido de recurso que provocou o *deadlock*.



# Referências Bibliográficas

- [1] Tanenbaum, A. S., Woodhull, A. S. **Sistemas Operacionais: projeto e implementação**. 2ª ed. Porto Alegre: Bookman, 2000.
- [2] Stallings, William. **Operating Systems: internals and Design Principles**. 4ª ed. New Jersey: Prentice Hall. 2001.
- [3] Silberschatz, A., Galvin, P. B., Gagne, G. **Operating System Concepts**. 6ª ed. Editora John Wiley & Sons, Inc. 2002.
- [4] SHAY, William A., Sistemas Operacionais. Makron Books. São Paulo. 1996.
- [5] Deitel, Deitel, Choffnes – Sistemas Operacionais . Prentice Hall – São Paulo 3ª Edição, 2005.