Universidade Federal de Itajubá – UNIFEI
Instituto de Engenharia de Sistemas e Tecnologias da Informação – IESTI
CCO 004 – Sistemas Operacionais – Prof. Edmilson Marmo Moreira

Capítulo 6 – *Deadlocks*

"Não encontre defeitos, encontre soluções. Qualquer um sabe queixar-se."
Henry Ford

1 Introdução

Sistemas computacionais estão repletos de recursos que somente podem ser utilizados por um processo de cada vez. Exemplos comuns incluem impressoras, tape drives etc.

Para muitas aplicações, um processo necessita de acesso exclusivo para diversos recursos. Um processo copiando um arquivo maior que o disco de um *tape* magnético para um impressora necessita de acesso exclusivo para os dois recursos.

Em um sistema com somente um processo, o processo pode simplesmente adquirir acesso para todos os recursos que ele precisa e realizar o seu trabalho.

Entretanto, em um sistema multiprogramado, sérios problemas podem acontecer.

Supondo, por exemplo, que dois processos desejem imprimir um grande arquivo armazenado em um *tape.* O processo *A* requisita permissão para utilizar a impressora e recebe autorização para fazê-lo. O processo *B,* em seguida, requisita permissão para utilizar o *tape drive,* e também consegue a permissão. Agora *A* pede para utilizar o *tape drive,* mas a requisição é negada até que *B* libere o recurso. Infelizmente, ao invés de liberar o recurso, *B* pede para utilizar a impressora. Neste ponto ambos os processos estão bloqueados e irão permanecer assim "para sempre". Esta situação é conhecida como ***Deadlock***.

2 Recursos

Deadlocks podem ocorrer quando processos recebem acesso exclusivo a dispositivos, arquivos, e assim por diante.

Para tornar a discussão sobre deadlock o mais geral possível, os objetos que são obtidos pelos processos serão denominados recursos.

Um recurso pode ser um dispositivo de hardware ou uma informação.

Os sistemas computacionais possuem diversos tipos de recursos para serem solicitados.

Para alguns tipos de recursos, diversas instâncias são permitidas. Quando diversas cópias de um recurso estão disponíveis, qualquer uma delas pode ser utilizada para satisfazer alguma requisição.

Resumindo, um recurso é qualquer coisa que somente pode ser utilizado por um único processo em um determinado instante de tempo.

Recursos podem ser de dois tipos: preemptivos e não-preemptivos.

Recursos preemptivos são aqueles que podem ser retirados dos processos sem causar nenhum prejuízo.

Memória é um exemplo de um recurso preemptivo. Considerando, por exemplo, um sistema com 512k de memória disponível para processos usuários, uma impressora, e dois processos de 512k que desejem imprimir ao mesmo tempo. O processo *A* requisita e obtém a impressora, então inicia o seu processamento. Antes de terminar a computação, ele excede o seu *quantum* de tempo e interrompido (*swapped out*).

Agora, o processo *B* executa e tenta, sem sucesso, adquirir a posse da impressora. Potencialmente, existiria uma situação de *deadlock,* porque *A* tem a impressora e *B* possui a memória.

Felizmente, é possível retirar o recurso de *B* (memória) através de um procedimento de *swapping,* passando o seu espaço de memória para que o processo *A* continue a sua execução. Como *A* pode executar, nenhum *deadlock* ocorre.

Um recurso não-preemptível, ao contrário, não pode ser retirado do processo sem causar falhas no processamento.

Em geral, *deadlocks* envolvem recursos não-preemptíveis. Potencialmente, *deadlocks* que envolvem recursos preemptíveis podem ser resolvidos através da realocação dos recursos para outros processos.

3 Deadlocks

Um conjunto de processos estão em deadlock se cada processo do conjunto está aguardando por um evento que somente um outro processo do mesmo conjunto pode causar.

Condições para que ocorra *Deadlock*

1. Condição de exclusão mútua: Cada recurso só pode estar associado a um único processo em um determinado instante.

2. Condição de posse e espera: Processo de posse de recursos podem requisitar por novos recursos.

3. Condição de Não-preempção: Recursos previamente garantidos não podem ser retirados de um processo.

4. Condição de Espera Circular: Deve existir uma cadeia circular de 2 ou mais processos, cada um aguardando por recursos que estão alocados para membros da mesma cadeia.

Todas essas 4 condições devem estar presentes para que um *deadlock* ocorra. Se uma dessas quatros condições estiver ausente o *deadlock* é impossível.

Abordagens para o tratamento do *Deadlock*

Existem várias abordagens para o tratamento do *deadlocks*. Cada abordagem apresenta vantagens e desvantagens, sendo o emprego dependente da característica do sistema.

As principais abordagens são:

- Agoritmo do Avestruz.
- Prevenção de Deadlock.
- Impedimento do Deadlock.
- Detecção do Deadlock.
- Recuperação do Deadlock.

O Algoritmo do Avestruz

Essa é uma abordagem simples:

Enfie sua cabeça na terra e imagine que nenhum problema está ocorrendo.

A principal idéia por detrás desta abordagem é que a probabilidade de ocorrer *deadlock* (ou a média de ocorrência) em um período relativamente longo for pequena, perdas causadas pelo problema, quando ele ocorrer, serão compensadas pela economia em não ser preciso detectar, impedir ou recuperar.

Prevenção do *Deadock*

Para previnir que um deadlock oconteça, basta evitar a ocorrência de qualquer uma das quatro condições necessárias para a existência de *deadlock.*

Entretanto, evitar cada uma das condições impõe um custo muito alto ao sistema.

1. Remoção da Exclusão mútua: Sem dúvida a inexistência de exclusão mútua evita o deadlock, mas a sua ausência pode causar problemas maiores com a condição de disputa.

2. Remoção da Não-preempção: A remoção da não-preempção podde causar problemas de escalonamento e introduzir inconsistências nas atividades diárias. Além disso, pode fazer com que os processos percam dados se não for tomado cuidados extras. Essa solução impõem um alto custo ao sistema.

3. Remoção da Posse e Espera: Existem duas formas de se evitar a posse e espera:

    i) Forçar o processo a desistir de todos os recursos sempre que ele solicitar mais. Essa situação contradiz a condição de Não-preempção.

    ii) Proibir que um processo faça solicitações em instantes diferentes, ou seja, fazer com que ele solicite todos os recursos que irá precisar de um só vez. Neste caso, o processo acabará controlando desnecessariamente alguns

recursos por longos períodos de tempo sem realmente usá-los, diminuindo a disponibilidade dos recursos. Outro problema é que um processo pode ser adiado indefinidamente se ele precisar do acesso a muitos recursos (*starvation*).

4. Remoção da Espera Circular: Existem três formas de se evitar a espera circular:

   i) Rejeitar que processos peçam recursos em instantes diferentes. Essa situação é a mesma comentada na item anterior.

   ii) Permitir que processos tenham acesso a apenas um recurso de cada vez. Essa solução é impossível de ser alcançada em sistemas de uso geral, pois depende da forma como os processos foram implementados.

   iii) Forçar os processos a solicitarem recursos em uma determinada ordem. Essa solução impõem um alto custo para o sistema e pode levar os processos a situações de *starvation.*

Impedimento do *Deadlock*

A principal diferença do impedimento e a prevenção é que o impedimento não elimina totalmente o *deadlock,* pois teoricamente é possível ocorrer uma seqüência infeliz de solicitações.

Se o Sistema Operacional perceber que a alocação de um recurso cria riscos de ocorrer *deadlock* ele nega o acesso evitando assim o problema. A idéia é rejeitar solicitações que criam um potencial para a paralisação.

Algoritmo do Banqueiro para um único recurso (Dijkstra)

Este algoritmo faz distinção entre estados seguros e estados inseguros.

Um estado é seguro quando o sistema operacional pode garantir recursos suficientes para que, mesmo que todos os processos solicitem todos os recursos de que precisam, eles irão terminar sem que ocorra *deadlock.* Em um estado seguro, o sistema pode sempre evitar a paralisação rejeitando solicitações seletivamente.

Um estado é inseguro quando, independentemente da reposta do sistema, os processos podem fazer solicitações que causam *deadlock.*

Um estado inseguro não implica em uma paralisação iminente. Ele indica que o SO não pode garantir que não ocorrerá *deadlock*.

Algoritmo

```
var Seguro : Boolean;
    Disponivel : integer;
begin
  Seguro := true;

  {Define L como sendo uma lista temporária de todos
   os processos ativos }

  Disponivel := {n. recursos disponíveis};

  repeat
    if {existe um processo em L tal que
        Disponivel ≥ n. recursos não requisitados pelo
        processo}
    Then begin
              {Remove processo da lista L}
              Disponivel := Disponivel +
                    {n. recursos alocados ao
                     processo removido}
         end
    else Seguro := False;
  until (L = ∅) or (Not Seguro);
end.
```

Algoritmo do Banqueiro para Recursos Múltiplos

Este algoritmo é uma generalização do algoritmo anterior.

Ele utiliza 3 vetores:

E = recursos existentes;
P = recursos de posse dos processos;
D = recursos disponíveis.

E duas matrizes:

A = Matriz de recursos X processos (Alocados);
R = Matriz de recursos X processos (Requisitados);

Algoritmo

Passos do Algoritmo:

1. Escolher uma linha em R, cujo n. De recursos necessários são todos ≤ ao vetor D. Se não existir o sistema poderá entrar em *deadlock,* ou seja, o estado é inseguro.

2. Assumir que o processo desta linha jã completou a utilização dos recursos:

   - Marcá-lo com terminado;
   - Adicionar todos os recursos alocados ao processo ao vetor D;

3. Repetir os passos 1 e 2 até que todos os processos estejam marcados. Neste caso, o estado seguro foi atingido, caso contrário o estado é inseguro.

Exemplo:

Supondo a existência de 3 processos e quatro classes de recursos (*tape drives*, *plotters*, impressoras e CD-Rom).

O processo 1 está de possa de uma impressora. O processo 2 possui dois *tape drives* e um CD-Rom. O processo 3 possui uma *plotter* e duas impressoras.

Cada processo necessita recursos adicionais, como mostrado pela matriz *R.*

Recursos Existentes

Tape Drives
Plotters
Impressoras
CD - Roms

E = (4, 2, 3, 1)

Recursos Disponíveis

Tape Drives
Plotters
Impressoras
CD - Roms

D = (2, 1, 0, 0)

Matriz de recursos alocados

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

Matriz de recursos Requisitados

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

Para executar o algoritmo de detecção de *deadlock*, primeiro é escolhido o processo que pode ter suas solicitações atendidas.

O primeiro processo não pode ser satisfeito porque não existe CD-Rom disponível. O segundo também não pode ser atendido, porque não existe impressora livre.

Felizmente, o terceiro pode ser atendido. Então o processo 3 executa e eventualmente devolve todos os seus recursos. O vetor D fica,

$$D = (2, 2, 2, 0)$$

Neste ponto o processo 2 pode executar e devolver os seus recursos em seguida. O resultado do vetor D é

$$D = (4, 2, 2, 1)$$

Agora o processo 1 pode executar. Como todos os processos está marcados não existe *deadlock* no sistema.

Realizando uma pequena variação da situação apresentada. Supondo que o processo 2 precisa de um CD Rom e também de 2 *tape drives* e a *plotter.*

Nenhuma das requisições podem ser atendidas, então o sistema pode entrar em *deadlock* e o estado é inseguro.

Detecção do *Deadlock*

Supondo que se tenha um sistema no qual o *deadlock* pode ocorrer. Ou seja, não há esquemas para preveni-la ou evitá-la.

Em suma, sempre que um processo solicitasse um recurso, o *deadlock* poderia ocorrer.

É preciso responder a duas perguntas:

- Como o sistema pode detectar uma situação de *deadlock*?
- Se o *deadlock* ocorrer o que o sistema fará?

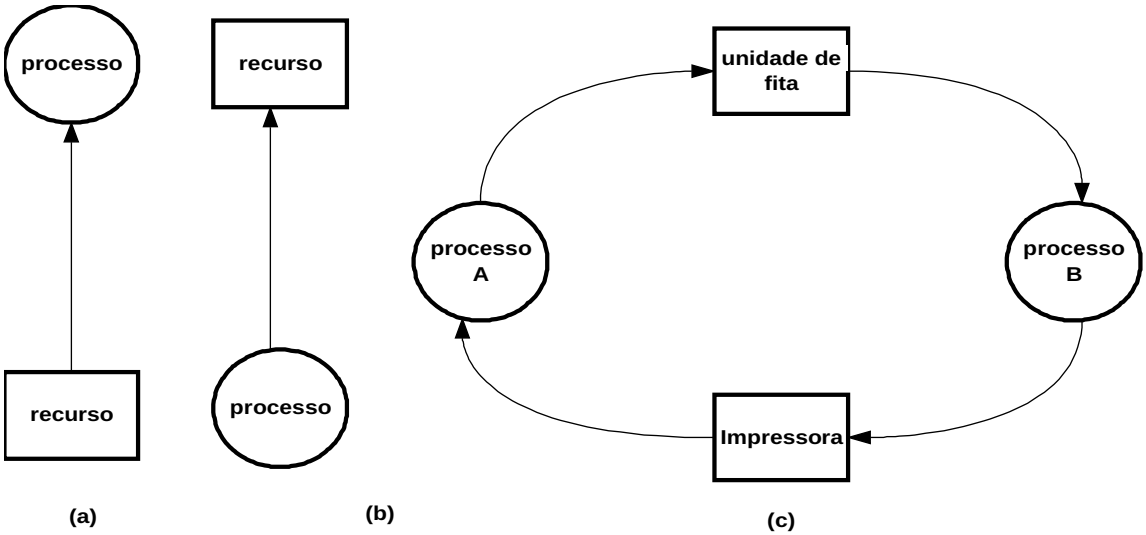
Uma maneira de detectar o *deadlock* é por meio de um dígrafo de alocação de recursos.

Grafo de Alocação de Recursos

Os dígrafos podem representar muitos tipos diferentes de informação. De interesse para o estudo dos *deadlocks,* é o seu uso para representar alocação de recursos.

Um dígrafo desses é chamado de **Dígrafo de Alocação de Recursos**.

Em suma, sempre que um processo solicitasse um recurso, o *deadlock* poderia ocorrer.



(a) Um processo de posse de um recurso
(b) Um processo aguardando por um recurso
(c) Deadlock

Exemplo:

Considerando um sistema com 7 processos, A, B, C, D, E, F e G; e 6 recursos R, S, T, U, V e W; e com os recursos alocados da seguinte maneira:

1 O processo A detém R e deseja S.

2 O processo B não detém recursos, mas deseja T.

3 O processo C não detém recursos, mas deseja S.

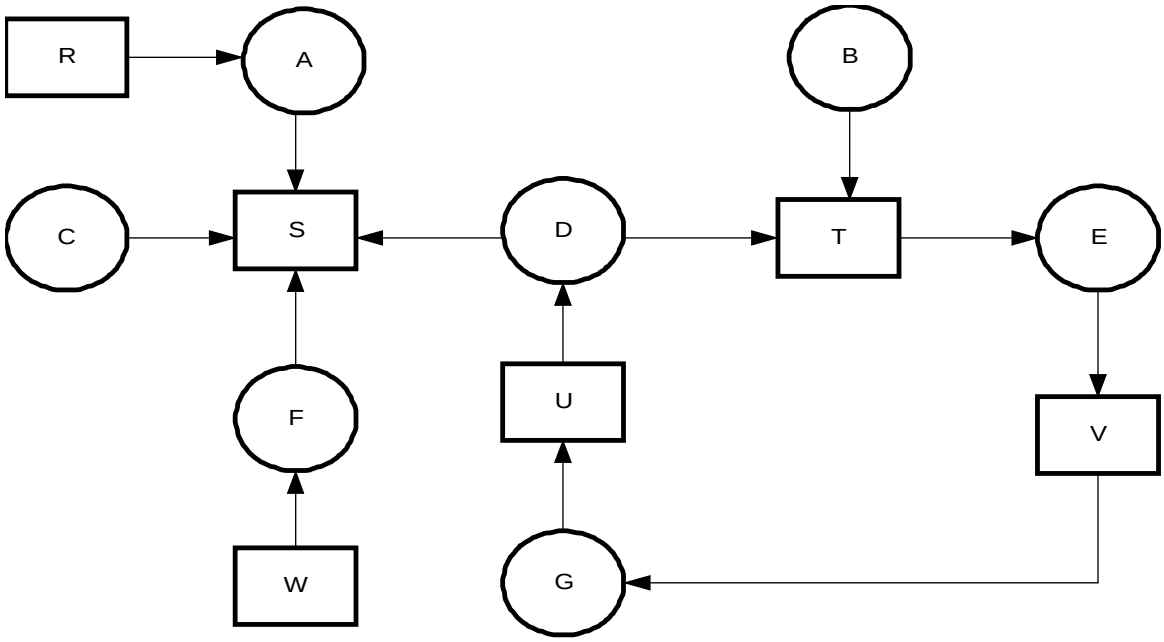
4 O processo D detém U e quer S e T.

5 O processo E detém T e quer V.

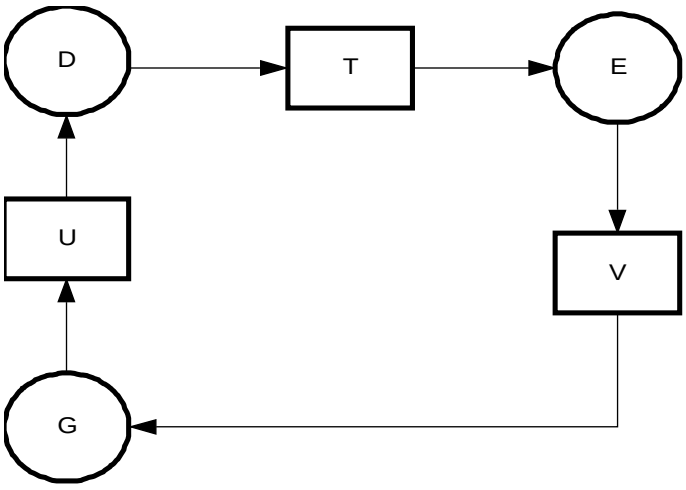
6 O processo F detém W e quer S.

7 O processo G detém V e quer U.

Este sistema está em *deadlock*? Se estiver, quais processos estão envolvidos?



Processos em Deadlock



Detecção de *Deadlock* com múltiplos recursos de cada tipo

Seja o *m* o número de classes de recursos, com e_1 recursos da classe 1, e_2 recursos da classe 2 e, generalizando, e_i recursos da classe i ($1 \leq i \leq m$). *E* é o vetor de recursos existentes.

Seja *D* o vetor de recursos disponíveis, com D_i informando o número de instâncias do recurso *i* atualmente disponíveis.

São necessárias duas matrizes:

C = Matriz de recursos alocados correntemente;

R = Matriz de requisições.

A *i*-ésima linha de *C* representa os recursos alocados para o processo *i.* Assim $C_{i,j}$ é o número de instâncias do recurso *j* que o processo *i* possui.

Da mesma forma $R_{i,j}$ é o número de instâncias do recurso *j* que o processo *i* necessita.

Portanto,

$$\sum_{i=1}^{m} C_{ij} + A_j = E_j$$

Algoritmo

Passos do Algoritmo:

1. Cada processo inicia desmarcado

2. Escolher uma linha em R, cujo n. de recursos necessários são todos ≤ ao vetor D.

   Se encontrou
   Então $C_i \leftarrow C_i + A$; e marcar o processo
   Senão interrompa

3. repetir passo 2 até que todos os processos estejam marcados ou até que seja impossível escolher um processo.

4. Se existir processo desmarcado ele se encontra em *deadlock.*

Recuperação do *Deadlock*

Uma vez detectado um *deadlock* o sistema deverá corrigir a situação.

Uma possibilidade é simplesmente encerrar o processo e remover os recursos alocados a ele. Isso elimina o ciclo, e conseqüentemente o *deadlock.*

Entretanto, e se esse processo já estiver executando por uma hora e estava quase terminando o seu processamento quando o sistema o abortou? O sistema deve ser capaz de responder a questões importantes sobre qual processo o sistema operacional deve abortar. Por exemplo, quem será responsável pelo trabalho perdido?

Outra possibilidade é fazer um *rollback* em um processo, ou seja, remover todos os seus recursos correntemente alocados a ele. O processo perde todas as atualizações feitas por meio do uso desses recursos e sofre uma perda de trabalho, porém, ele não é abortado.

O sistema força o processo a voltar ao estado em que ele se encontrava antes da solicitação e alocação dos recursos removidos.

Isso pode corresponder a um ponto de partida ou a um *checkpoint* na lógica do processo.

Bibliografia:

DEITEL, H. M., DEITEL, P. J. & CHOFFNES, D. R.
Sistemas Operacionais
3a. Edição: Person (2005) – São Paulo / SP

MACHADO, F. B. & MAIA, L. P.
Arquitetura de Sistemas Operacionais
3a. Edição: LCT (2002) – Rio de Janeiro / RJ

SILBERSCHATZ, ABRAHAM & GALVIN, PETER B.
Operating System Concepts
5th Edition: John Wiley (1999) – Massachusetts

STALLINGS, WILLIAM
Operating Systems
2nd Edition: Prentice-Hall (1995)

TANENBAUM, ANDREW S.
Sistemas Operacionais Modernos
2ª. Edição. Person (2003) – São Paulo / SP

TANENBAUM, ANDREW S. & WOODHULL, ALBERT S.
Operating Systems: Design and Implementation
2nd Edition: Prentice-Hall (1997) – Upper Saddle River / NJ

TOSCANI, S. S.; OLIVEIR, R. S. & CARISSIMI, A. S.
Sistemas Operacionais e Programação Concorrente
Editora Sagra-Luzzatto (2003) – Porto Alegre / RS