Deadlocks - Impasses

Sobre a apresentação (About the slides)

Os slides e figuras dessa apresentação foram criados por Silberschatz, Galvin e Gagne em 2009. Esse apresentação foi modificada por Cristiano Costa (cac@unisinos.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.unisinos.br/~cac

Objetivos do Capítulo

- Desenvolver a descrição de um Impasse, conhecido tecnicamente como 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.

Recursos

- Todo SO deve ser capaz de garantir acesso exclusivo de um processo a certos recursos
- Impasses ou Deadlocks podem ocorrer quando vários processos recebem acesso exclusivo a recursos
- Exemplos de recursos:
  - Arquivos
  - Dispositivos de hardware
  - Variável compartilhada na memória
  - etc
- Cada tipo de recurso R_i pode ter W_i instâncias.

Recursos

- Recursos preemptíveis: podem ser retirado do processo proprietário sem nenhum prejuízo
- Recursos não-preemptíveis: Recurso que NÃO pode ser retirado do processo sem que a computação apresente falha
- Deadlocks ou Impasses envolvem recursos não-preemptíveis.

Recursos

- Cabe aos processos do usuário, ou seja, programas que desenvolvemos, fazer uso apropriado dos recursos computacionais usando as ferramentas que o SO disponibilizar.
- Possível solução:
  - Requisitar o recurso
    - Se não estiver disponível pode gerar uma espera ou um código de erro
  - Usar o recurso
  - Liberar o recurso

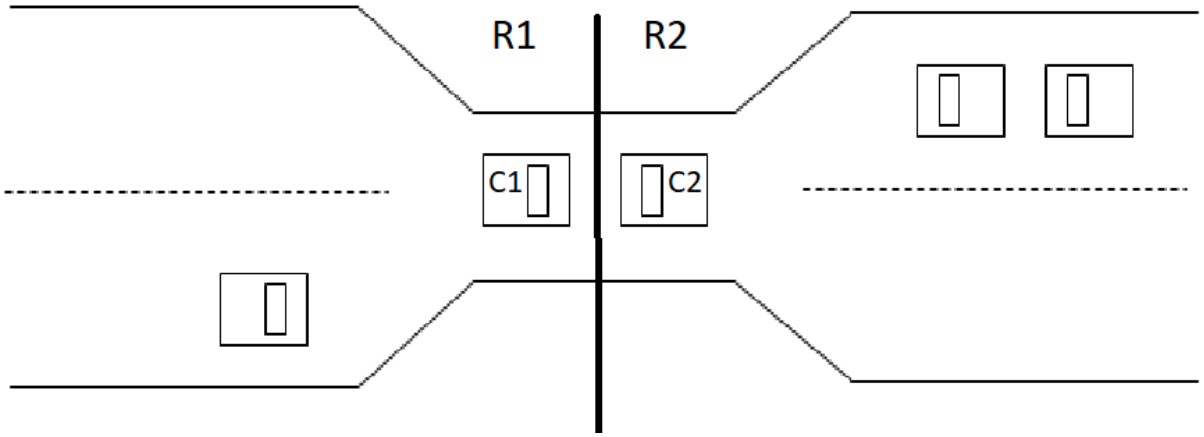
O Problema do Deadlock

- Exemplo - Sistema tem 2 unidades de disco.
  - P_1 e P_2 detêm uma unidade de disco cada um e necessitam de outra.
- Exemplo
  - mutex *A* e *B*, iniciados em 1

P_1	P_2
wait (A);	wait(B);
wait (B);	wait(A);

  - Um conjunto de processos bloqueados cada um detendo um recurso e esperando para adquirir um recurso detido por outro processo no conjunto.

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 – a maioria dos SOs não previne ou trata Deadlocks

Caracterização de Deadlock

Deadlock pode ocorrer se quatro condições

são satisfeitas simultaneamente.

- **Exclusão Mútua:** somente um processo de cada vez pode usar um recurso.
  - Todos os SOs possuem mecanismos para garantir Exclusão Mútua (ex: mutex)

- **Posse e Espera de um recurso:** 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.

Caracterização de Deadlock

Deadlock pode ocorrer se quatro condições são satisfeitas simultaneamente.

- **Inexistência de preempção:** um recurso só pode ser liberado voluntariamente pelo processo ao qual está alocado depois que o processo terminar de usá-lo.
  - SO não retira recurso de processo.

- **Espera circular:** deve existir um conjunto {P_0, P_1, …, P_0} 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 alocada a P_n, e P_0 esteja esperando por um recurso alocado a P_0.

Essas condições ocorrem nos SOs atuais (WIN/UNIX)

Grafo de Alocação de Recursos

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

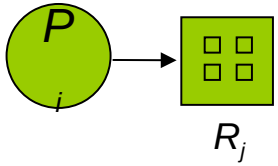
- V está particionado em dois tipos:
  - $P = \{P_1, P_2, \ldots, P_n\}$, conjunto que consiste de todos os processos no sistema.
  - $R = \{R_1, R_2, \ldots, R_m\}$, conjunto que consiste de todos os tipos de recursos do sistema.
- **Arco de requisição** – arco dirigido $P_1 \rightarrow R_j$
- **Arco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

Grafo de Alocação de Recursos (Cont.)

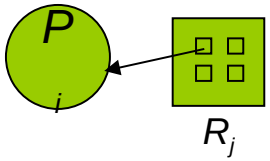
- Processos

- Tipo de Recurso com 4 instâncias

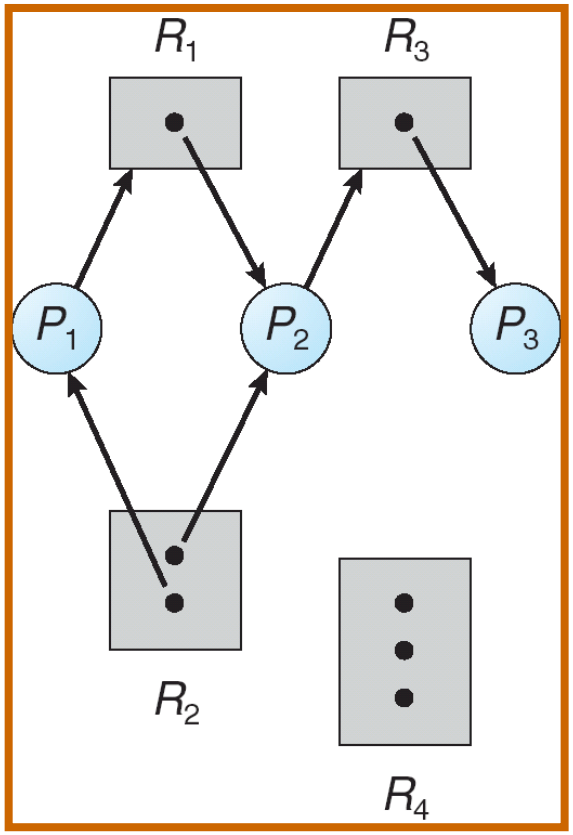
- P_i solicita uma instância de R_j



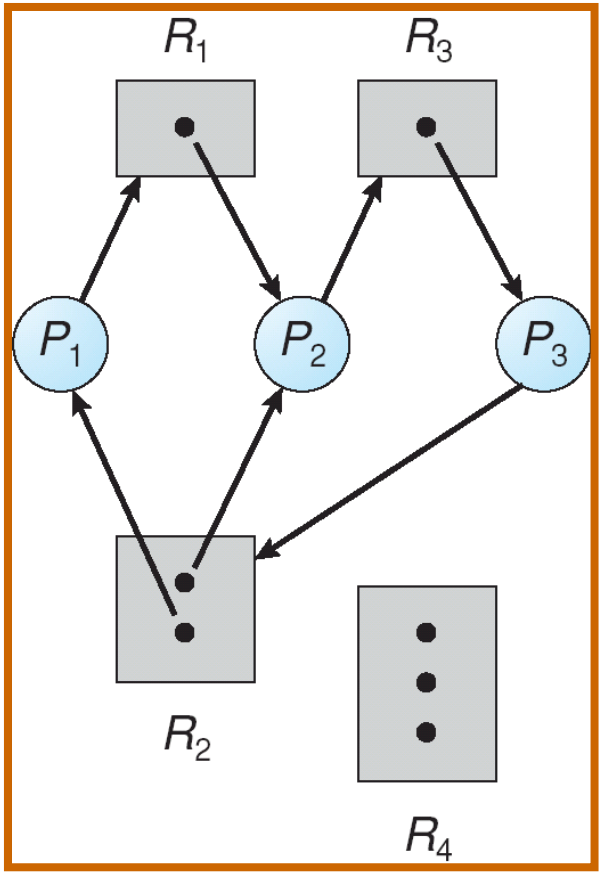
- P_i tem alocado uma instância de R_j



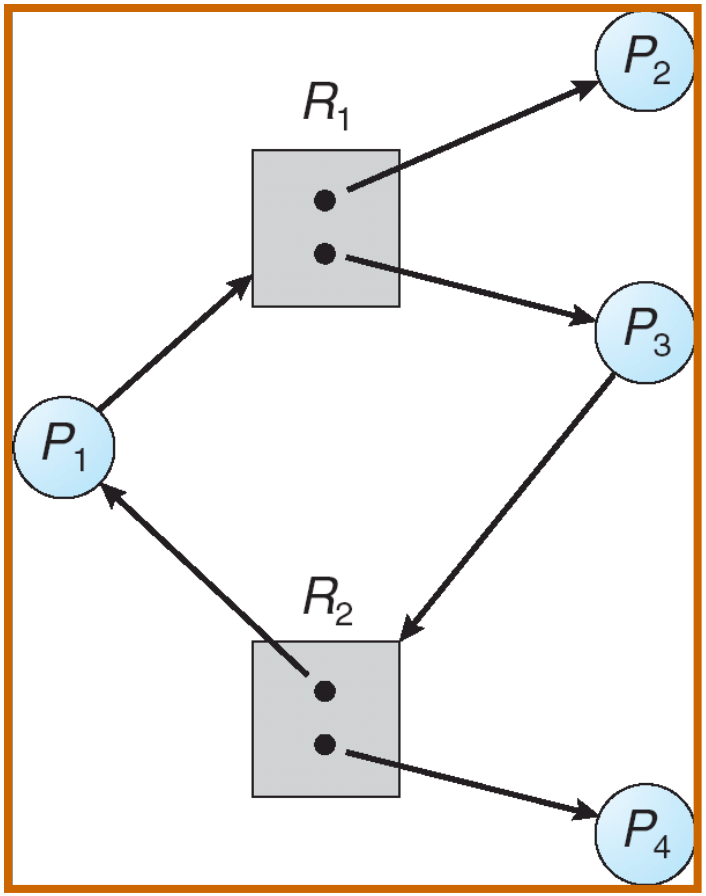
Exemplo de Grafo de Alocação de Recursos



Grafo de Alocação de Recursos com Deadlock



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

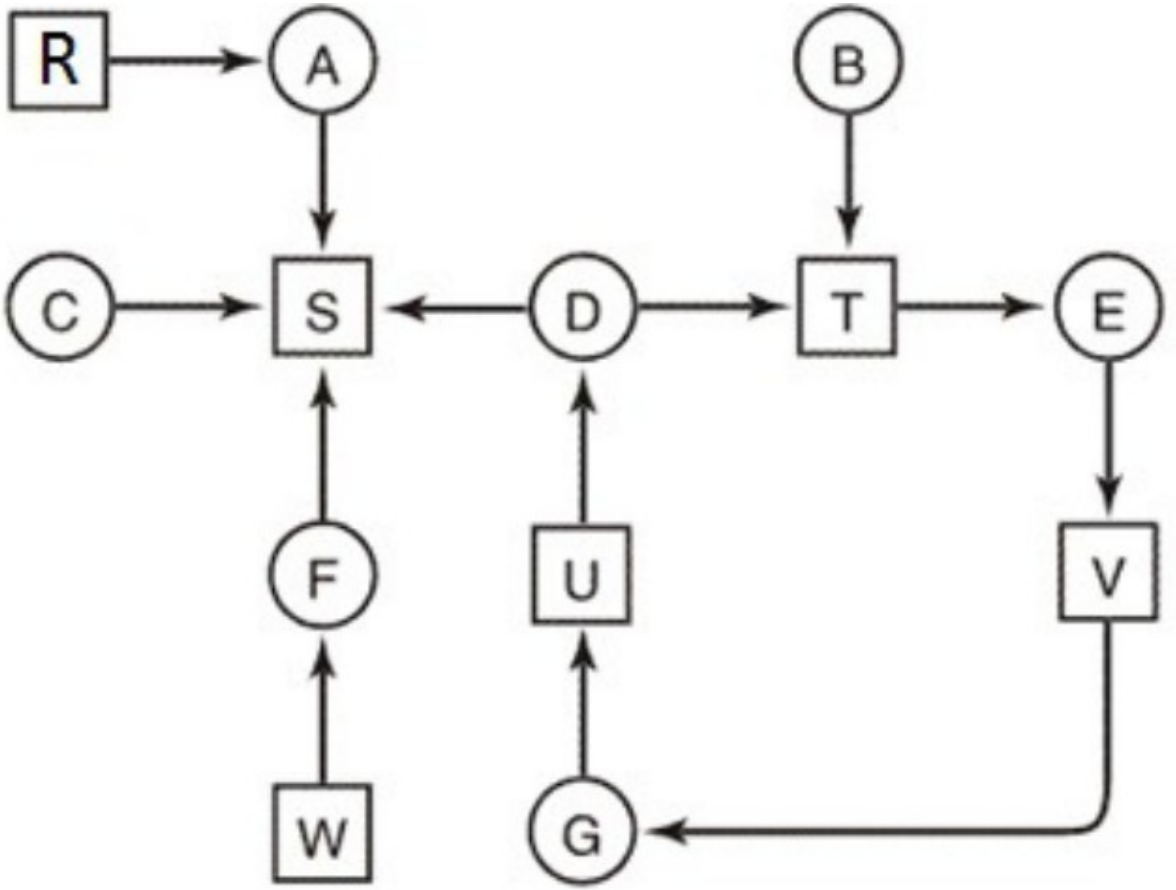


Fatos Básicos

- Se um grafo não contém ciclos ⇒ não há *Deadlock*.
- Se um grafo contém um ciclo ⇒
  - 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*.

Exercício 1

- Encontre o impasse



Exercício 2

- Faça um grafo de Holt para as ações a seguir
- Assuma que todos os recursos estão livres e que requisições de recursos livres resultam em alocação
- A requisita R, A requisita S, B requisita T, C requisita S, D requisita U, D requisita S, D requisita T, E requisita T, E requisita V, F requisita W, F requisita S, G requisita V e G requisita U.
- Nesse caso, ocorre um deadlock ?

Métodos para Tratar Impasses

1. Ignorar o problema e fingir que Deadlocks nunca ocorrem no sistema; usado pela maior parte dos sistemas operacionais, incluindo o Windows e o UNIX.

2. Permitir ao sistema entrar em estado de Deadlock e então recuperar (O SO se recuperaria)

3. Garantir que o sistema nunca entrará em estado de Deadlock.

4. Realizar o Bloqueio em 2 Fases

“Algoritmo do Avestruz”

- Finge que o problema não existe
- Razoável se
  - deadlocks ocorrem muito raramente
  - custo da prevenção é alto
- **UNIX e Windows seguem esta abordagem**
- É uma ponderação entre
  - Conveniência
  - Correção
- Usuário deverá decidir qual processo deverá ser terminado para que o outro prossiga

Detecção e Recuperação de Impasses

- Utiliza-se o Grafo de Holt quando houver um recurso de cada tipo
  - Havendo um ciclo, caracteriza-se um deadlock
- Recuperação de Impasses
  - 1 - Regressão por preempção do recurso
    - Retira recurso do processo
    - Pode bloquear processo e tentar devolver recurso antes de escaloná-lo novamente
    - Não é viável

Recuperação de Impasses

- O que o SO faria?
  - Pausa a execução de 1 dos processos
  - Tira o recurso de um dos processos
  - Atribui o recurso para o outro processo
    - Com isso os outros processos podem executar
  - Depois, restaura o snapshot do processo tirado antes dele obter o recurso e coloca em estado pronto
- Problema: 1) Performance 2) Não desfaz o que foi feito no recurso e pode deixá-lo num estado inconsistente

Recuperação de Impasses

- Recuperação de Impasses
  - 2 - Regressão por reversão de estado
    - Salva estado de um processo periodicamente
    - usa este estado salvo
    - reinicia o processo a partir do estado salvo anterior se este é encontrado em estado de deadlock
    - Inviável – por falta de recursos computacionais

Recuperação de Impasses

- P1

```
// snapshot do processo
   e
// da entrada dele na
   tabela
// de processos
requisita(R1);
// le ou escreve no R1
// 20 linhas de codigo
requisita(R2); //
   deadlock
```

- P2

```
// snapshot do processo e
// da entrada dele na
   tabela
// de processos
requisita(R2);
// 5 linhas de codigo
requisita(R1); // deadlock
```

Recuperação de Impasses

- Recuperação de Impasses
  - 3 - Eliminação de processos causadores do deadlock
    - 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
  - Problemas
    - Não desfaz o que foi feito no recurso
    - Pode matar um processo importante

Exercícios

1. Explique como funciona o algoritmo do Avestruz.
2. O algoritmo acima é utilizado atualmente pelos Sistemas Operacionais ?
3. Explique as 3 formas de se recuperar um sistema em que um impasse já ocorreu ?

Prevenção de Deadlock

Restringir as formas em que as requisições podem ser feitas.

- **Atacar a Exclusão mútua** – não é necessária para recursos compartilháveis (preemptíveis); deve ocorrer para recursos não compartilháveis (não-preemptíveis).
- Alguns dispositivos (como uma impressora) podem fazer uso de spool
  - o daemon de impressão é o único que usa o recurso impressora
  - desta forma deadlock envolvendo a impressora é eliminado
- Nem todos os dispositivos podem 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

Restringir as formas em que as requisições podem ser feitas.

- **Atacando a condição de Posse e Espera** –
  - Garantir que sempre que um processo requerer um recurso, ele não esteja de posse de nenhum outro recurso ou
    - Poderia requerer no máximo 1 recurso por vez
  - Exigir que um processo faça a requisição de recursos somente quando não possuírem nenhum.
    - Não é de todo ruim ou inviável.. Apenas limita um pouco o desenvolvimento de software
  - Problemas
    - Pode ser muito limitante para o desenvolvedor de software só poder usar um recurso por vez

Prevenção de Deadlock (Cont.)

- **Atacar a 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.
    - O SO perempta o recurso que o processo possui assim que ele solicitar o segundo
    - Tenho um recurso, pedi o 2º mas ele está alocado então eu libero os recursos que eu aloquei
    - Seu código deve estar preparado para isso e não ter modificado nenhum dos recursos que serão liberados

Prevenção de Deadlock (Cont.)

- **Atacar a Espera Circular** – impor uma ordenação total de todas as classes de recursos e requerer que a requisição de recursos por cada processo sempre ocorra em uma ordem crescente.
  - Tem um potencial para dar certo

Resumo

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

Bloqueio em 2 fases – Não é feito pelo SO

- Fase um
  - processo tenta bloquear todos os recursos de que precisa, um de cada vez
  - Se recurso necessário já estiver bloqueado, reinicia novamente
  - (nenhum trabalho real é feito na fase um)
- Se a fase um for bem sucedida, começa a fase dois,
  - execução de atualizações
  - liberação de bloqueios
- Observe a similaridade com a requisição de todos os recursos de uma só vez
- Algoritmo funciona onde **o programador tiver organizado tudo cuidadosamente para que o programa fazer a retentativa** de alocação dos recursos
  - É mais uma boa prática do que algo que possa ser feito pelo SO

Exercícios

1. Prevenção de Deadlocks. Explique as 4 técnicas abaixo
    1. Atacando condição de exclusão mútua
    2. Atacando condição de Posse-e-Espera
    3. Atacando a condição de não preempção
    4. Atacando a condição de espera circular
2. No que consiste o Bloqueio em 2 fases ? Esta técnica é efetiva no combate a deadlocks ?

Fim