

AULA 12 - Deadlocks

Em alguns casos pode ocorrer a seguinte situação: um processo solicita um determinado recurso e este não está disponível no momento. Quando isso acontece o processo entra para o estado de espera (bloqueado). Em algumas situações é possível que estes processos nunca mais mudem de estado, pois os recursos que eles necessitam podem estar sendo mantidos por outros processos em espera. Essa situação é chamada de **deadlock ou impasses**. Definição:

“Um conjunto de processos está em estado de deadlock quando todos os processos no conjunto estão esperando por um evento que só pode ser causado por outro processo do conjunto.”

A idéia de impasse ou deadlock pode ser mais facilmente entendida se fizermos uma analogia com uma escada de um prédio utilizada para casos de incêndio. Apesar de ter sido construída como uma opção de fuga em caso de incêndio, as pessoas que trabalham no prédio muitas vezes preferem utilizar a escada ao invés dos elevadores. No entanto, há espaço apenas para uma pessoa em cada degrau. Logo, o tráfego pela escada vai bem até que 2 pessoas se cruzam. Por outro lado, existe uma plataforma em cada um dos andares que suporta várias pessoas. Os problemas acontecem no momento em que uma pessoa que está subindo a escada encontra outra que está descendo e ambas se recusam a retroceder até a plataforma. Esta situação permanecerá gerando um impasse ou deadlock.

O texto desta seção foi publicado em <http://msdn.microsoft.com/pt-br/library/ms177433.aspx> e estou inserindo neste material para mostrar a importância do conceito em algumas áreas da computação.

Leiam também as matérias publicadas sobre:

- Detectando e encerrando deadlocks:
 - o <http://msdn.microsoft.com/pt-br/library/ms178104.aspx>
- Manipulando deadlocks:
 - o <http://msdn.microsoft.com/pt-br/library/ms177453.aspx>

Deadlock

Um deadlock acontece quando duas ou mais tarefas bloqueiam uma à outra permanentemente, sendo que cada uma tem o bloqueio de um recurso, que a outra tarefa está tentando bloquear. Por exemplo:

- A transação A adquire um bloqueio compartilhado da linha 1.
- A transação B adquire um bloqueio compartilhado da linha 2.
- A transação A agora solicita um bloqueio exclusivo na linha 2 e é bloqueado até que a transação B termine e libere o bloqueio compartilhado que tem na linha 2.
- A transação B agora solicita um bloqueio exclusivo na linha 1 e é bloqueado até que a transação A termine e libere o bloqueio compartilhado que tem na linha 1.

A transação A não pode terminar até que a transação B termine, mas a transação B está bloqueada pela transação A. Essa condição é também chamada de dependência cíclica: a transação A tem uma dependência da transação B, e a transação B fecha o círculo tendo uma dependência da transação A.

Ambas as transações em um deadlock esperarão indefinidamente, a menos que o deadlock seja quebrado por um processo externo. O monitor de deadlock do Microsoft Mecanismo de Banco de Dados do SQL Server verifica periodicamente as tarefas que estão em um deadlock. Se o monitor detectar uma dependência cíclica, ele escolhe uma das tarefas como vítima e termina sua transação com um erro. Isso permite que a outra tarefa complete sua transação. O aplicativo com a transação que terminou com um erro pode repetir a transação, a qual normalmente é concluída depois que a outra transação em deadlock é encerrada.

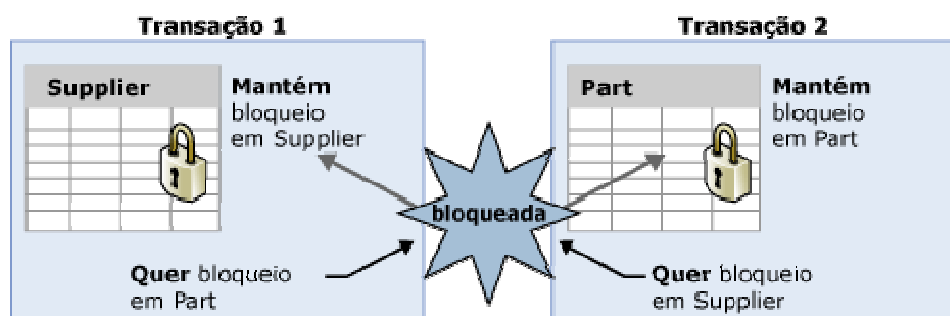
Usando certas convenções de codificação em aplicativos reduz a chance de que os aplicativos causarão deadlocks. Para obter mais informações, consulte [Minimizando deadlocks](#).

O deadlock é freqüentemente confundido com bloqueio normal. Quando uma transação solicita um bloqueio em um recurso bloqueado por outra transação, a transação solicitante espera até que o bloqueio seja liberado. Por padrão, as transações SQL Server não têm tempo limite, a menos que LOCK_TIMEOUT seja configurado. A transação solicitante é

bloqueada, não em deadlock, por que ela não fez nada para bloquear a transação que deve o bloqueio. Finalmente, a transação proprietária vai terminar e liberar o bloqueio e a transação solicitante terá o bloqueio atribuído e processado.

Os deadlocks às vezes são chamados de abraço mortal.

Deadlock é uma condição que pode ocorrer em qualquer sistema com vários threads, não só em sistemas de gerenciamento de banco de dados relacional, e pode ocorrer para outros recursos, além de bloqueios de objetos em bancos de dados. Por exemplo, um thread em um sistema operacional de vários threads pode adquirir um ou mais recursos, como bloqueios de memória. Se o recurso sendo adquirido é atualmente propriedade de outro thread, o primeiro thread pode ter que esperar o thread proprietário liberar o recurso alvo. O thread em espera tem uma dependência do thread proprietário para aquele recurso em particular. Em uma instância do Mecanismo de Banco de Dados, sessões podem fazer um deadlock ao adquirir recursos que não são de banco de dados, como memória ou threads.



Na ilustração, a transação T1 tem uma dependência da transação T2 para o recurso de bloqueio de tabela **Part**. Da mesma forma, a transação T2 tem uma dependência da transação T1 para o recurso de bloqueio de tabela **Supplier**. Devido a essas dependências formarem um ciclo, há um deadlock entre as transações T1 e T2.

Os deadlocks também podem ocorrer quando uma tabela é particionada e a configuração LOCK_ESCALATION do ALTER TABLE é configurada para AUTO. Quando a LOCK_ESCALATION é configurada para AUTO, a simultaneidade aumenta ao permitir que o Mecanismo de Banco de Dados bloqueie partições de tabela no nível de HoBT em vez de no nível de TABLE. Entretanto, quando transações separadas mantêm bloqueios de partição em uma tabela e querem um bloqueio em algum lugar de outra partição de transações, isso

causa um deadlock. Esse tipo de deadlock pode ser evitado configurando `LOCK_ESCALATION` para `TABLE`; embora essa configuração irá reduzir a simultaneidade forçando as atualizações extensas em uma partição a esperarem por um bloqueio de tabela.

Modelo de Sistema

Em nosso sistema computacional existe um número finito de recursos (espaço de memória, ciclos de CPU, arquivos, dispositivos de I/O, etc) que serão distribuídos entre os processos existentes que concorrem pelo uso destes. Assim, sempre que um processo deseja utilizar algum dos recursos do sistema ele deve primeiro solicitá-lo através de uma requisição ao sistema operacional. Consequentemente, após utilizar o recurso o processo deve liberá-lo para outros processos. Em um modo de operação normal, a seguinte seqüência deve ser estabelecida pelo processo:

1. **pedido:** se o processo não tiver seu pedido atendido ele é bloqueado e ficará esperando até ganhar a posse do recurso.
2. **uso:** tendo posse do recurso ele poderá utilizá-lo.
3. **liberação:** desalocar o recurso adquirido pelo processo o liberando para outro processo.

Os pedidos e a liberação de recursos são feitos através de chamadas ao sistema operacional, por exemplo: *open e close file, allocate e free memory*, etc. No entanto, alguns pedidos são gerenciados pelo próprio usuário, como acontece ao utilizar semáforos (UP e DOWN).

O gerenciamento desses recursos é feito pelo sistema operacional através de uma tabela, que registra se cada recurso está livre ou ocupado. Neste último caso, também é especificado o processo que tem a posse do recurso. Da mesma forma que acontecia no uso de semáforos, qdo um processo P solicita um recurso X e ele está ocupado o P pode ser inserido em um fila associado ao recurso X.

Existem alguns casos de deadlocks que podemos citar:

1. requisições de arquivos: se for permitido aos programas requisitar e bloquear durante sua execução.

P1 requisita e obtém arquivo F2

P2 requisita e obtém arquivo F1

P1 requisita F1 mas está bloqueado

P2 requisita F2 mas está bloqueado.

2. alocação de dispositivos dedicados: utilização de um grupo de dispositivos dedicados pode gerar deadlocks. Exemplo: um programa que precise de 2 fitas magnéticas para copiar dados de uma para outra.
3. alocação de múltiplos dispositivos: vários processos requisitam e bloqueiam diferentes dispositivos dedicados.

Condições Necessárias

Para que um situação de deadlock seja criada, as seguintes condições devem acontecer simultaneamente:

1. *exclusão mútua*: apenas um processo por vez pode usar o recurso.
2. *monopolização de recursos*: acontece no exemplo da escada quando duas pessoas se cruzam em um lance e elas se recusam a retroceder.
3. *não-preempção*: os recursos só podem ser liberados voluntariamente pelo processo que o mantém.
4. *espera circular*: dado um conjunto de processos $\{P_1, P_2, \dots, P_n\}$ em espera, P_1 está esperando por um recurso mantido por P_2 ; P_2 está esperando um recurso mantido por P_3 e assim sucessivamente, até que P_n esteja esperando por um recurso alocado por P_1 .

Grafo de Alocação de Recursos

Podemos utilizar a notação de grafos para descrever os deadlocks. O grafo de alocação de recursos do sistema consistem em um conjunto de vértices V e um conjunto de arestas R . O conjunto de vértices V é dividido em dois diferentes tipos de nós: $P = \{P_1, P_2, \dots, P_n\}$ descrevem todos os processos ativos no sistema e $R = \{R_1, R_2, \dots, R_m\}$ que designam os tipos de recursos do sistema.

Portanto, uma aresta direcionada do processo P_i para o recurso R_j , denotada por $P_i \rightarrow R_j$ (*aresta de pedido*), indica que o processo P_i solicitou uma instância do recurso R_j e, no momento, está esperando por aquele recurso. Por outro lado, um aresta direcionada de um recurso R_j para um processo P_i , denotada por $R_j \rightarrow P_i$ (*aresta de atribuição*), significa que uma instância do recurso R_j foi alocada para o processo P_i .

Graficamente, utilizamos círculos para representar processos e quadrados para os recursos. Além disso, como cada recurso pode ter mais de uma instância, representamos cada instância por um ponto (bolinha) dentro do quadrado.

Dinâmica: quando um processo P_i solicita uma instância de um determinado recurso, uma aresta de pedido é inserida no grafo. Ao ser atendido, a aresta de pedido é transformada em uma aresta de atribuição. Portanto, quando o processo liberar o recurso, a aresta de atribuição é eliminada.

O que pode ser um indício de deadlock no grafo de alocação de recursos?

Resposta: A existência de ciclos!

Exemplo 1:

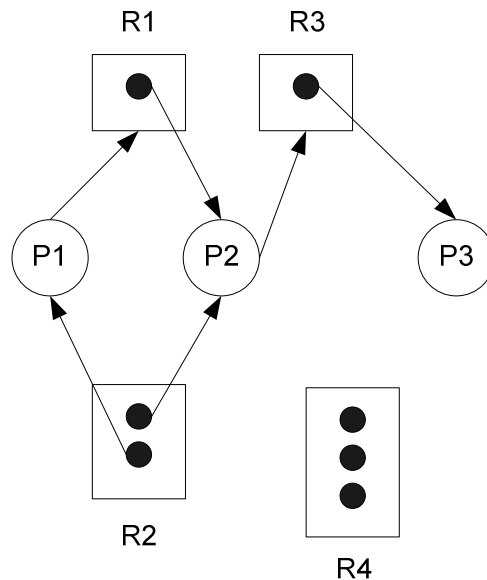


Figura 1. Grafo de Alocação de Recursos.

Na Fig. 1 os processos P1, P2 e P3 estão em deadlock? (Não!)

Exemplo 2:

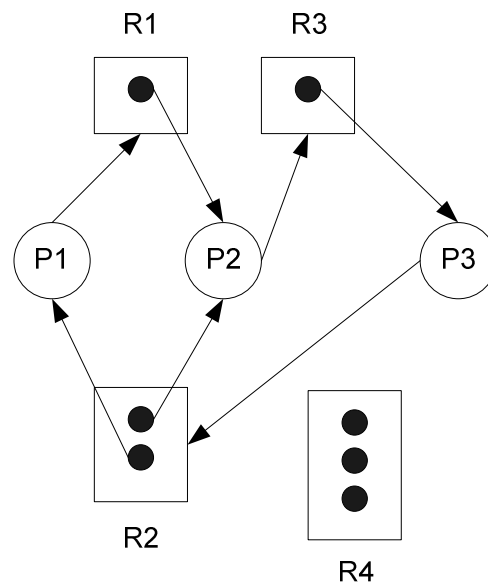


Figura 2. Grafo de Alocação de Recursos.

Na Fig. 2 os processos P1, P2 e P3 estão em deadlock? (Sim!)

Método para tratar Deadlocks

Em geral, podemos utilizar 3 métodos para tratar o problema de deadlocks:

1. podemos usar um protocolo para garantir que o sistema nunca entre em estado de deadlock: **PREVENÇÃO** (o Sistema operacional deve eliminar uma das 4 condições para que o deadlock ocorra)
2. podemos permitir que o sistema entre em deadlock e se recupere: **DETECÇÃO e RECUPERAÇÃO**.
3. podemos ignorar o problema (Algoritmo do avestruz). Esta opção é utilizada pela maioria dos sistemas operacionais, incluindo o UNIX.

Para garantir que um sistema nunca entre em deadlock é possível usar um esquema de prevenção de deadlocks ou de impedimento de deadlocks. Esta prevenção é realizada através de um conjunto de métodos que garantem que pelo menos uma das situações necessárias para que um deadlock aconteça seja falha. Basicamente, isso é feito limitando a forma como as solicitações de recursos acontecem.

Caso o sistema operacional não utilize um esquema de prevenção, será necessário um mecanismo de detecção e recuperação de deadlocks. Pelo fato do deadlock não ser evitado, até que ele seja descoberto é provável que o desempenho do sistema tenha sofrido uma considerável queda.

Exercícios

1. O que é deadlock, quais as condições para o acontecimento desses impasses e quais as soluções possíveis?
2. Um conjunto de estratégias foram propostas por Havender com o intuito de prevenir a ocorrência de deadlocks, analise cada uma delas e diga se as estratégias conseguem ou não atingir o seu propósito:
 - a. Estratégia 1: Cada processo deve requisitar todos os recursos de que precisa de uma vez só e não pode continuar até que todos tenham sido concedidos.
 - b. Estratégia 2: Se for negada mais uma requisição a um processo que retém certos recursos, ele deve liberar seus recursos originais e, se necessário, requisitá-los novamente junto com os recursos adicionais.
 - c. Estratégia 3: Deve ser imposta uma ordenação linear de recursos a todos os processos; se um processo recebeu certos recursos, ele somente poderá requisitá-los novamente mais tarde, conforme a ordem.
3. Pesquise e descreva o mecanismo (idéia) de funcionamento do algoritmo do banqueiro de Dijkstra proposto para evitar deadlocks. Além disso, cite as deficiências presentes nesse algoritmo.
4. Por que a prevenção de deadlock não é uma preocupação primária para muitos sistemas operacionais?
5. Dê um exemplo de deadlock simples de recurso envolvendo 4 processos e 4 recursos. Desenhe o grafo de alocação de recursos apropriado.
6. Por que a recuperação de deadlock é um problema tão difícil?

Bibliografia

[1] SILBERSCHATZ, A. et al. **Sistemas Operacionais: conceitos e aplicações**. Campus, 2001. (Cap. 8)