O que é um DeadLock?

Introdução

Nos sistemas computacionais, existem diversos recursos que devem ser acessados por apenas um processo de cada vez, como impressoras, unidades de fita e entradas em tabelas internas do sistema. Quando dois ou mais processos tentam utilizar simultaneamente um desses recursos, como uma impressora, ocorre um impasse. Portanto, é essencial que os sistemas operacionais garantam o acesso exclusivo de um processo a determinados recursos, mesmo que temporariamente.

Em muitas situações, um processo precisa de acesso exclusivo não apenas a um, mas a múltiplos recursos simultaneamente. Por exemplo, imagine dois processos tentando gravar um documento escaneado em um CD. O processo A está utilizando o scanner, enquanto o processo B está gravando o CD. Quando o processo A tenta usar o gravador de CD, sua solicitação é negada até que o processo B libere o recurso. Entretanto, ao invés de liberar o gravador, o processo B solicita o uso do scanner. Nesse momento, ambos os processos ficam bloqueados, criando uma situação que os impede de continuar suas execuções, conhecida como deadlock.

Deadlock (ou interbloqueio, blocagem, impasse) é um conceito central em sistemas operacionais e bancos de dados, ocorrendo quando dois ou mais processos ficam permanentemente bloqueados, aguardando por recursos que estão alocados a outros processos, resultando em uma paralisação mútua. Esse fenômeno é inerente à natureza desses sistemas e é amplamente estudado, devido às suas implicações no desempenho e na estabilidade.

O deadlock acontece em cenários onde há um conjunto de processos e recursos não-preemptíveis, isto é, recursos que não podem ser tomados à força de um processo. Quando um ou mais processos desse conjunto aguardam a liberação de recursos que estão alocados a outros processos, a cadeia de dependências cria uma situação sem saída.

Embora a definição textual de deadlock possa parecer abstrata, sua compreensão pode ser facilitada por meio de representações gráficas, que serão abordadas adiante. É importante observar que o deadlock pode ocorrer mesmo em sistemas com um único processo, caso ele utilize múltiplos threads, e estes, por sua vez, requeiram os mesmos recursos.

Quais os problemas podem acontecer em uma aplicação quando isso acontecer.

O deadlock independe da quantidade de recursos disponíveis no sistema;

O deadlock independe da quantidade de recursos disponíveis no sistema. Ele costuma ocorrer com recursos como dispositivos, arquivos e memória, entre outros. Embora a CPU também seja um recurso para o sistema operacional, normalmente é preemptível, ou seja, pode ser compartilhada entre diferentes processos com a ajuda de escalonadores, especialmente em ambientes multitarefa.

Erros de deadlock também podem acontecer em bancos de dados. Imagine uma empresa com diversos vendedores e caixas. O vendedor A vende um martelo e uma furadeira. O sistema, então, bloqueia o registro da tabela "ESTOQUE", que contém a quantidade de martelos, e em seguida o registro de furadeiras. Após obter acesso exclusivo a esses registros, o vendedor A lê a quantidade de martelos, subtrai um e atualiza o valor; o mesmo acontece com a furadeira. No entanto, como há vários caixas operando simultaneamente, se outro caixa estiver vendendo uma furadeira nesse mesmo momento, ele precisará esperar até que o registro de furadeiras seja liberado.

Vale ressaltar que os registros só podem ser alterados depois que o sistema garantir exclusividade para todos os recursos necessários. Agora, suponha que outro caixa tenha vendido uma furadeira e um martelo e esteja tentando acessar o registro de martelos, que está bloqueado pelo primeiro caixa. Nenhum dos caixas consegue liberar os recursos que possuem, nem acessar os que estão bloqueados pelo outro. Isso caracteriza um deadlock.

Definição

De acordo com Tanenbaum, deadlock pode ser definido como: "Um conjunto de processos estará em deadlock se todo processo pertencente ao conjunto estiver esperando por um evento que somente outro processo desse mesmo conjunto pode fazer acontecer."

Em termos mais simples, deadlock é um problema que ocorre quando um grupo de processos compete por recursos entre si. A ocorrência de deadlock depende das características dos processos envolvidos e do momento em que são executados. Embora seja possível que esses processos operem repetidamente sem causar deadlock, basta um único processo em condições complicadas para desencadear a situação.

Mesmo fora da computação, podemos exemplificar deadlock com dois carros vindo em direções opostas em uma via estreita, onde apenas um veículo pode passar. Ambos ficam impedidos de continuar seu trajeto, representando um cenário de deadlock.

Existem quatro condições necessárias para que o deadlock ocorra:

Exclusão mútua: Em um determinado instante, cada recurso ou está associado a um único processo ou está disponível.

Posse e espera: Um processo pode reter recursos já alocados e solicitar novos recursos.

Não-preempção: Recursos alocados a um processo não podem ser retirados à força; devem ser liberados explicitamente pelo processo que os possui.

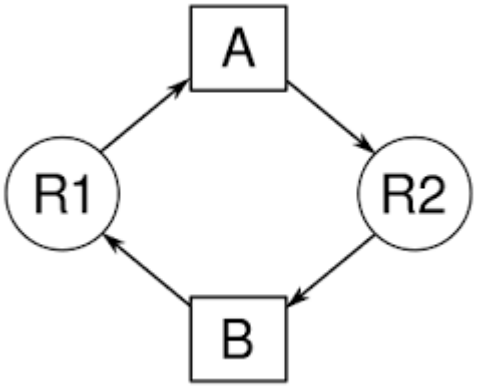
Espera circular: Deve existir um ciclo de espera entre dois ou mais processos, onde cada um aguarda por um recurso que está sendo utilizado pelo próximo processo da cadeia.

Representação de deadlock em grafos

O deadlock também pode ser representado graficamente, utilizando grafos dirigidos. Nesse caso, os processos são representados por quadrados e os recursos por círculos. Quando um processo solicita um recurso, uma seta é traçada do quadrado para o círculo. Quando um recurso é alocado ao processo, a seta vai do círculo para o quadrado.

Em um exemplo clássico, dois processos A e B têm alocados, respectivamente, os recursos R1 e R2. Cada processo precisa do recurso que está com o outro, criando uma situação de espera circular que caracteriza o deadlock.

Na figura do exemplo, podem-se ver dois processos diferentes (A e B), cada um com um recurso diferente alocado (R1 e R2). Nesse exemplo clássico de deadlock, é facilmente visível a condição de espera circular em que os processos se encontram, onde cada um solicita o recurso que está alocado ao outro processo.



Vamos ver agora as 4 estratégias para se tratar um deadlock:

Algoritmo da Avestruz (ignorar o problema): A estratégia mais simples consiste em fazer como a avestruz diante de uma situação de perigo: "esconder a cabeça" e fingir que o problema não existe. Essa solução é amplamente utilizada, pois a probabilidade de ocorrer deadlock é baixa e o custo de implementação também. O sistema UNIX, por exemplo, adota esse método.

Detecção e Recuperação: O sistema operacional (SO) monitora as requisições e liberações de recursos, mantendo um grafo de recursos que é constantemente atualizado. Se um ciclo for detectado no grafo, indicando a ocorrência de deadlock, um dos processos envolvidos deve ser finalizado. Caso o ciclo persista, outros processos devem ser encerrados até que o ciclo seja quebrado. Essa técnica é comumente usada em sistemas de grande porte, geralmente em operações em batch, onde é possível matar um processo e reiniciá-lo posteriormente. Deve-se ter cuidado para que quaisquer arquivos modificados pelo processo encerrado sejam restaurados ao estado original antes de o processo ser reiniciado.

Prevenção de Deadlock (negando uma das quatro condições necessárias): A ideia aqui é impor restrições aos processos para que o deadlock se torne impossível. Existem algumas formas de eliminar as condições que causam o problema:

Exclusão Mútua: Para alguns recursos, a exclusão mútua é necessária, pois não é possível que dois processos os acessem simultaneamente. Além disso, nem sempre a técnica de spooling pode ser aplicada. Portanto, essa condição não pode ser eliminada.

Posse e Espera: Uma maneira de negar essa condição é obrigar cada processo a requisitar todos os recursos de que precisará antes de iniciar a execução. Se todos os recursos estiverem disponíveis, o processo é executado; caso contrário, ele deve aguardar. No entanto, muitos processos não conseguem prever quais recursos precisarão até o início da execução, além de essa abordagem não otimizar o uso dos recursos. Outra forma de quebrar essa condição é fazer com que, ao requisitar um novo recurso, o processo libere temporariamente todos os que já possui. Ele só poderá recuperá-los caso consiga o novo recurso.

Não-preempção: Essa condição não pode ser quebrada, pois forçar a retirada de um recurso de um processo para entregá-lo a outro causaria desordem.

Espera circular: Para evitar ciclos no grafo de recursos, é possível impor que cada processo possa possuir apenas um recurso por vez. Se desejar outro, deverá liberar o recurso atual (isso, porém, inviabilizaria operações simples como copiar dados de uma fita para outra). Outra alternativa é atribuir uma numeração global a todos os recursos, de modo que os processos possam requisitá-los apenas em ordem numérica crescente. No entanto, encontrar uma numeração que atenda à maioria das possíveis condições de acesso pode ser complicado.

Evitação Dinâmica de Deadlock: Nessa abordagem, procura-se evitar o deadlock sem impor restrições rígidas aos processos. Isso é possível se o sistema operacional tiver informações antecipadas sobre a utilização de recursos pelos processos. Existem algumas maneiras de fazer isso:

Algoritmo do Banqueiro para um Recurso Único: Este algoritmo simula o comportamento de um banqueiro ao conceder empréstimos, levando em consideração certas condições. O banqueiro reserva uma quantidade fixa de recursos para atender a demanda dos clientes. Por exemplo, suponha que quatro clientes (A, B, C e D) especifiquem o número máximo de unidades de crédito que precisarão. No entanto, eles não necessitam de todo o crédito de uma só vez, de modo que o banqueiro reserva 10 unidades para atender às solicitações de todos, mesmo que o total solicitado seja 32 unidades.

Cliente	***Usado***	***Máximo***
A	0	6
B	0	5
C	0	4
D	0	7

Disponível:10

Cada cliente pode em cada momento realizar um pedido desde que não ultrapasse seu limite de crédito e o banqueiro pode escolher o momento de atendimento dos pedidos realizados. Após o recebimento de todo o crédito que especificou inicialmente o cliente se compromete a devolver tudo. O banqueiro deve analisar para que não entre numa situação difícil.

Chamamos de estado, a situação dos empréstimos e dos recursos ainda disponíveis em um dado momento.

Cliente	**Usado**	**Máximo**
A	1	6
B	1	5
C	2	4
D	4	7

Disponível:2

O exemplo ao lado demonstra um estado seguro, pois existe uma sequência de transições que garante que todos os clientes recebam seus empréstimos até atingirem seus limites de crédito. Nesse cenário, o banqueiro pode emprestar 2 unidades para o cliente C, receber de volta essas 4 unidades, emprestar para o cliente B, receber as 5 unidades de volta e então emprestar para o cliente A. Após receber as 6 unidades de A, o banqueiro pode emprestar 3 dessas unidades para o cliente D, recuperando depois as 7 unidades totais e, assim, garantindo o atendimento de todos os clientes.

Por outro lado, se o cliente B solicitasse mais uma unidade (totalizando 2), isso resultaria em um estado inseguro, pois não haveria como garantir que o banqueiro atendesse a todos os pedidos, o que poderia eventualmente causar um *deadlock*. No entanto, o *deadlock* ainda não ocorreria nesse momento, uma vez que os valores máximos não precisam necessariamente ser atingidos. Por exemplo, o cliente D pode decidir que não precisa mais de novos recursos e devolver os 4 que já utilizou, retornando o sistema a uma situação segura. Porém, o banqueiro não pode contar com essa eventualidade. Portanto, a cada nova solicitação de recursos, o banqueiro deve verificar se concedê-la levaria a um estado seguro. Isso é feito analisando se os recursos disponíveis são suficientes para atender o cliente mais próximo do seu limite máximo de crédito. Se for possível, o sistema "simula" que esse cliente devolveu todos os recursos, marca-o como finalizado e verifica se é possível atender o próximo cliente com necessidades máximas. Se o processo for concluído com sucesso, atendendo todos os clientes, então o estado é considerado seguro.

Algoritmo do Banqueiro para Vários Recursos

Quando trabalhamos com vários tipos de recursos, utilizamos duas matrizes: uma representando os recursos já alocados e outra representando os recursos ainda necessários. Cada linha dessas matrizes corresponde a um processo, e cada coluna representa um tipo de recurso.

Para que os processos funcionem corretamente, eles precisam informar antecipadamente suas necessidades de recursos. Para verificar se o estado é seguro, o seguinte procedimento é adotado:

1. Procura-se uma linha **L** onde os recursos ainda necessários sejam todos menores ou iguais aos valores correspondentes no vetor de recursos disponíveis **D**. Se nenhuma linha atender a essa condição, o sistema entrará em *deadlock* e o estado será considerado inseguro.

2. Assume-se que o processo da linha **L** requisitou todos os recursos necessários e finalizou sua execução, liberando os recursos que estavam em uso.

3. Repete-se os dois passos anteriores até que todos os processos sejam marcados como finalizados.

Se todos os processos forem finalizados com sucesso, o estado do sistema é considerado seguro. O problema com esse algoritmo é que, na prática, os processos raramente sabem com precisão a quantidade máxima de recursos que precisarão, além de o número de processos no sistema variar dinamicamente, o que invalida muitas das análises realizadas pelo sistema.

Referencias:

https://www.devmedia.com.br/introducao-ao-deadlock/24794

https://alexcoletta.eng.br/artigos/deadlock-em-sistemas-operacionais/

https://pt.wikipedia.org/wiki/Deadlock