INTRODUÇÃO AOS SISTEMAS OPERACIONAIS outubro/2013

SEMANAS 11 e 12

Estudo das deadlocks.
Caracterização e condições necessárias para a ocorrência.
Soluções, prevenção, impedimento, detecção e recuperação.

1 - Introdução

Em sistema multiprogramado, deadlock é um bloqueio perpétuo ou impasse. Dizemos que um processo está em deadlock quando espera por um evento particular que jamais irá acontecer. Igualmente dizemos que um sistema está em deadlock quando um ou mais processos estão nesta situação.

Segundo Tanenbaum: "Um conjunto de processos está em deadlock quando cada processo do conjunto está esperando por um evento que apenas outro processo do conjunto pode causar".

2 - Caracterização das deadlocks

Uma deadlock pode ocorrer de diferentes maneiras:

- quando um processo é colocado em espera por algo, e o sistema operacional não inclui qualquer provisão para o atendimento desta espera. É o deadlock de um único processo [one-process deadlock];

- quando se forma uma cadeia sucessiva de solicitações de recursos que culminam num arranjo circular, onde um processo P1 segura [holding] um recurso R2 e requisita um recurso R1 alocado para um processo P2, que por sua vez está solicitando o recurso R2, em uso por P1. Como nenhum

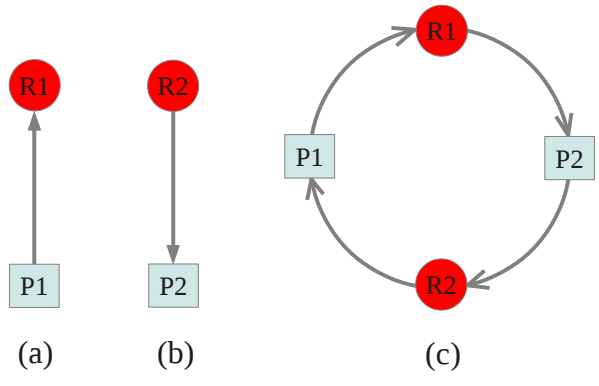

Fig 1 – (a): processo P1 está requisitando o recurso R1;
(b): processo P2 está segurando (holding) o recurso R2;
(c): deadlock.

desses processos se dispõem a liberar o recurso que detêm, isso configura uma situação de deadlock.

Independentemente do tipo, os deadlocks causam sérios prejuízos ao sistema, pois mesmo num one-process deadlock, recursos ficam alocados desnecessariamente, o que significa que restarão menos recursos para os demais processos. Nos deadlocks circulares, além da alocação desnecessária de recursos, podem ser formadas filas de espera pelos recursos envolvidos, deteriorando o tempo de resposta do sistema, podendo até causar situações de instabilidade ou crash do sistema operacional.

Quando um processo é bloqueado indefinidamente, ficando em espera por um recurso, dizemos que está ocorrendo um adiamento indefinido ou bloqueio indefinido (respectivamente indefinite postponement, indefinite blocking). Como um processo nessa situação não pode prosseguir com a sua execução devido a ausência de recursos, também dizemos que ele está em starvation (estagnado).

A maioria dos problemas que culminam com os deadlocks estão relacionados com recursos dedicados, isto é, com recursos que devem ser utilizados serialmente, ou seja, por um processo de cada vez.

O problema de deadlock existe em qualquer sistema operacional multiprogramável, mas as soluções implementadas devem considerar o tipo do sistema e o impacto no seu desempenho. O interesse pelo estudo de deadlock inclui:

- prevenção de deadlock;
- impedimento de deadlock;
- detecção de deadlock;
- recuperação de deadlock.

3 - Condições necessárias para a ocorrência de deadlocks

Existem quatro condições para a ocorrência de deadlocks:

i. condição de exclusão mútua [mutual exclusion]: os processos exigem controle exclusivo dos recursos que solicitam. Um recurso só pode estar alocado para um processo de cada vez;

ii. condição de espera por recurso [hold and wait]: os processos mantêm alocados recursos enquanto solicitam novos recursos. O processo que alocou um recurso "exclusivo" pode segurar este recurso enquanto espera obter novos recursos;

iii. condição de ausência de preemptividade [não preemptividade, no preemption]: os recursos não podem ser retirados dos processos que os mantêm alocados enquanto estes processos não finalizam seu uso. Uma vez que um processo obtenha um recurso, o sistema não pode removê-lo do controle do processo até que este tenha finalizado seu uso;

iv. condição de espera circular [circular wait condition]: forma-se uma cadeia circular de processos, onde cada processo solcita um recurso alocado pelo próximo processo na cadeia. Dois ou mais processos estão alocados em uma cadeia circular na qual cada processo está esperando por um ou mais recursos que o próximo processo da cadeia está mantendo alocado.

4 - Soluções para as deadlocks

Segundo Tanenbaum, existem basicamente quatro alternativas para o tratamento dos bloqueios perpétuos, ou seja, quatro estratégias básicas para resolvermos os problemas das deadlocks:

i. ignorar o problema (algoritmo da avestruz): apesar de não parecer uma solução para o problema, devem ser consideradas a possibilidade de ocorrência dos deadlocks e os custos computacionais associados ao seu tratamento. A alternativa mais simples realmente é ignorar o problema e conviver com a possibilidade de sua ocorrência. Os sistemas UNIX utilizam esta aproximação favorecendo aspectos de performance em situações mais comuns;

ii. detecção e recuperação das deadlocks: outra alternativa é permitir que os bloqueios ocorram, mas procurar detectá-los e recuperá-los. Para isso, é usado algum algoritmo que produza um diagrama de alocação de recursos que analise em busca de deadlocks e utilize alguma técnica de recuperação. Estes algoritmos especiais podem evitar maiores transtornos ao sistema, apesar da sobrecarga que provocam. Isso possibilita que a ocorrência de deadlocks seja descoberta e eliminada;

iii. pevenção dinâmica: as deadlocks podem ser prevenidos através de procedimentos cuidadosos de alocação que analisam constantemente a possibilidade de formação de cadeias circulares. Tais algoritmos também são complexos e acabam por onerar o sistema computacional;

iv. prevenção estrutural: as deadlocks podem ser estruturalmente eliminados de um sistema através da negação de uma ou mais das quatro condições de ocorrência. Para isto devem ser tratadas as condições de Coffman, que é resumidamente apresentado na tabela abaixo.

Condição	Aproximação
exclusão mútua	colocar todos os recursos do sistema em *spool*[1]
retenção e espera	exigir a alocação inicial de todos os recursos necessários
ausência de preemptividade	retirada de recursos dos processos
espera circular	ordenação numérica dos pedidos de recursos

1 Spool: Simultaneous Peripheral Operation On Line.

5 - Prevenção das deadlocks

A prevenção de deadlocks é a estratégia preferencialmente adotada pelos projetistas de sistemas, adotando uma política que assume o custo da prevenção como alternativa aos prejuízos possíveis da ocorrência das deadlocks e de sua eliminação. Para prevenir a ocorrência das deadlocks podem ser adotadas uma ou mais das seguintes estratégias, tal como proposto por Havender (1968):

i. um processo só pode solicitar um recurso se liberar o recurso que detêm;

ii. um processo que tem negado o pedido de recurso adicional deve liberar o recurso que atualmente detêm;

iii. se a solicitação de recursos ocorrer em ordem linear ascendente, a espera circular não consegue se formar.

Mas para esta estratégia, antes devemos diferenciar estados seguros e inseguros.

A maioria dos algoritmos conhecidos para prevenção de deadlocks se baseia no conceito de estado seguro. Um estado seguro é aquele em que existe garantia que todos os processos poderão ser finalizados considerando (1) suas necessidades em termos de recursos e (2) os recursos efetivamente disponíveis no sistema. Desta forma os recursos cedidos aos processos serão devolvidos ao sistema para serem alocados para outros processos, numa sequência de estados seguros.

Por outro lado, um estado considerado como inseguro é aquele em que não existe a garantia de devolução dos recursos devidos pois o processo não recebe todos os recursos de que necessita e não devolve ao sistema aqueles eventualmente já alocados. A maior consequência de um estado inseguro é que existem grandes chances de ocorrer um deadlock a partir desta situação, sendo por isso necessário evitá-lo.

A ocorrência de um estado inseguro, bem como de um estado seguro, depende estritamente da ordem como os recursos disponíveis são alocados e liberados pelos processos envolvidos. O sistema operacional, portanto, deve analisar se pode ou não atender plenamente as necessidades de um processo antes de ceder recursos que não poderão ser devolvidos.

Deadlocks podem ser prevenidas [ou pelo menos tenta-se] com os seguintes ações:

- remover a condição de exclusão mútua significa que nenhum processo possa ter acesso exclusivo ao recurso. Isso se mostra impossível de ser atingido para recursos que não podem ser spooled, e mesmo para recursos spooled ainda podem ocorrer deadlocks. Algoritmos que evitam a exclusão mútua são chamados de algoritmos de sincronização não bloqueante [non-blocking synchronization];

- a condição de espera por recurso [hold and wait] pode ser removida impondo aos processos que requisitem todos os recursos dos quais vão precisar antes do início, ou então antes de algumas operações em particular. Porém, essa condição é difícil de satisfazer e em muitas ocasiões produz baixa eficiência no uso de recursos. Um outro caminho é impor aos processos que liberem todos os recursos antes de requisitar os recursos de que vão necessitar

naquele momento. Porém, funcionalemente isso se mostra impraticável;

- a condição de ausência de preemptividade pode ser difícil ou até impossível de obter pois o processo precisa ter o recurso por algum tempo, senão a saída do processamento pode se tornar inconsistente ou pode ocorrer thrashing[2]. No entanto, a não habilidade para forçar a preempção pode interferir no desempenho do sistema, pois um recurso no estado "locked out" geralmente vai implicar em reversão ou voltar atrás [rollback[3]] no processamento. O roolback deve ser evitado, pois causa sobrecarga no processamento. Algoritmos que permitem preempção incluem não travamento [lock-free] e sem espera [wait-free] e controle simultâneo otimizado [optimistic concurrency control];

- Algoritmos que evitam a condição de espera circular incluem desabilitar interrupções durante regiões críticas [critical sections] e usar hierarquia para determinar uma ordenação parcial dos recursos. Porém, não existe hierarquia óbvia, e até endereços de memória de recursos são usados para determinar essa ordenação.

Regiões críticas ocorrem para recursos que só podem ser utilizados por um processo de cada vez. Uma região crítica pode ser uma rotina de software especial ou um dispositivo de hardware ou uma rotina de acesso para um certo dispositivo do hardware. Uma região crítica é, no fundo, uma forma de administrar a concessão e devolução de um recurso comum.

A situação que se deseja é a exclusão mútua, ou seja, quando um processo qualquer utiliza a região crítica, todos os demais são impossibilitados de utilizá-la. Apenas quando a região crítica for liberada é que outro processo poderá ter acesso à mesma, também de forma exclusiva. Os processos que desejam utilizar uma região crítica são processos paralelos concorrentes. Quando acidentalmente acontece o acesso de um ou mais processos enquanto a região crítica está ocupada ou quando dois ou mais processos entram na região crítica simultaneamente, dizemos estar ocorrendo um acesso simultâneo. Tal situação geralmente conduz à perda de dados para um ou mais dos processos participantes e, às vezes, o comprometimento da estabilidade do sistema como um todo.

Para prevenir que mais de um processo faça uso de uma região crítica não é razoável que tal controle seja realizado pelos próprios processos, pois erros de programação ou ações mal intencionadas poderiam provocar prejuízos aos usuários ou comprometer a estabilidade do sistema.

6 - Impedimento

As deadlocks podem ser evitadas se certas informações sobre os processos estiverem disponíveis antes da alocação dos recursos. Para cada requisição de recurso, o sistema então avalia se garantir o recurso solicitado significa deixar o sistema num estado inseguro, que poderia resultar

2 Thrashing: é uma situação em que grande quantidade de recursos computacionais são usados para obter um mínimo de resultado. Uma vez iniciado, thrashing é tipicamente auto sustentado até que algo ocorra para remover a situação que levou ao comportamento de thrashing. Usualmente thrashing se refere a dois ou mais processos acessando repetidamente o mesmo recurso compartilhado, produzindo uma séria degradação na performance do sistema pois está sendo gasta uma quantidade de tempo desproporcional no acesso ao recurso.

3 Rollback: é retornar para algum estado seguro, isto é, significa reiniciar o processo para um estado anterior. Para possibilitar esse reinício é necessário algum tipo de arquivo de controle [checkpoint file].

em deadlock. Desse modo, o sistema só vai garantir requisições que levem a estados seguros. Para o sistema ser capaz de descobrir se o próximo estado será seguro ou não, a qualquer momento ele precisa saber de antemão o número e tipo de recursos existentes, disponíveis e requisitados. Um algoritmo conhecido usado para evitar deadlock é o algoritmo do Banqueiro[4] [Banker's], o qual exige que se conheça de antemão o limite no uso do recurso. No entanto, para muitos sistemas é impossível conhecer de antemão o que cada processo irá requisitar. E isso significa que evitar deadlock é muitas vezes impossível.

É importante notar que um processo pode estar num estado inseguro sem resultar em deadlock. A noção de estado seguro e inseguro se refere somente à habilidade do sistema entrar em deadlock ou não. Por exemplo, se um processo requisita o recurso A isso vai resultar num estado inseguro, mas se esse processo libera o recurso B previne a condição de espera circular, então o estado é inseguro mas o sistema não está em deadlock.

7 - Detecção

Frequentemente, não é usado prevenção nem se evita deadlock. Ao invés disso é usado detecção de deadlock e reinicialização de processo, pelo emprego de um algoritmo que persegue a alocação de recursos e estados de processos. Esse algoritmo aplica reversão [rollback] e reinicializa um ou mais processos para remover a deadlock. Detectar uma deadlock depois da ocorrência é fácil pois os recursos que o processo bloqueou [locked] ou requisitou são conhecidos do escalonador de recursos [resource scheduler] do sistema operacional.

A detecção da possibilidade de ocorrência de uma deadlock antes que ela ocorra é muito mais difícil. Como regra geral, não é possível distinguir entre algoritmos que esperam pela ocorrência de um conjunto improvável de circunstâncias e algoritmos que nunca irão finalizar por causa de uma deadlock.

8 - Recuperação de deadlocks

Ainda que os deadlocks ocorram, dentro de certas circunstâncias é possível resolvê-los, isto é, recuperá-los ou eliminá-los, utilizando algumas técnicas:

- recuperação através de preempção: retirando-se algum recurso envolvido no bloqueio perpétuo do processo que o aloca permite a quebra do caminho fechado e consequente solução do deadlock. O problema reside que nem sempre um recurso pode ser retirado de um processo sem efeitos colaterais prejudiciais a este processo;

- recuperação através de operações de rollback: exige a implementação de checkpoints, isto é, um mecanismo de armazenamento de estados seguros do sistema através da cópia dos estados individuais dos processos em arquivos especiais. Isto possibilita que tais estados

4 Algoritmo do Banqueiro: esse nome vem da analogia pela maneira como os banqueiro lidam com restrições na liquidez. Lida com créditos e provisões para evitar levar a estado inseguro.

sejam retomados a partir daquele ponto. Esta solução, além da difícil implementação, exige muitos recursos e tem elevado custo computacional, embora resolva bem o problema;

- recuperação através de eliminação de processos: esta é maneira mais simples, embora também a mais drástica. Um ou mais dos processos identificados como envolvidos no bloqueio perpétuo podem ser sumariamente eliminados, de modo que o bloqueio seja resolvido. Enquanto alguns processos podem ser seguramente reiniciados (por exemplo, uma compilação), procedimentos de atualização em bancos de dados nem sempre podem ser interrompidos e reiniciados. A eliminação de processos que não podem ser simplesmente reiniciados pode provocar prejuízos ao sistema.