Sistemas Operacionais

Prof. Robson de Souza

Aulas 25 e 26

Conteúdo: Evitar e Prevenir impasses

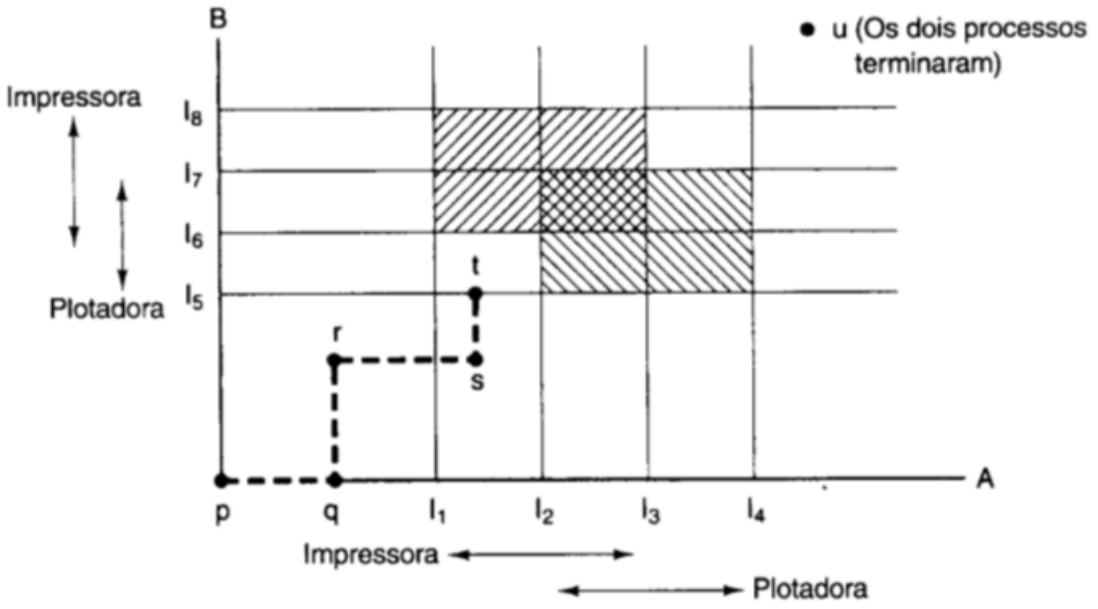
As estratégias para lidar com impasses são variadas, até aqui foram vistos os casos em que o impasse ocorre, é ignorado ou é tratado. Porém, existem algoritmos para tentar evitar ao máximo que um impasse ocorra, ou até mesmo prevenir a ocorrência de um impasse.

Evitando impasses

O sistema deve ser capaz de decidir se liberar um recurso é seguro ou não e somente fazer uma alocação quando ela for segura. Existem algoritmos que evitam os impasses, fazendo sempre a escolha correta, mas algumas informações precisam estar disponíveis.

* Trajetória de recursos

Os principais algoritmos para evitar impasses são baseados no conceito de estados seguros. Para entender isso melhor, observe a figura:



Essa figura mostra a situação de dois processos A e B. Esses dois processos precisam utilizar uma impressora e uma plotadora, os instantes I_1 a I_8 mostram o momento em que os processos requisitam ou liberam o recurso. A linha pontilhada na horizontal significa que a CPU está sendo usada pelo processo A, a linha pontilhada na vertical significa que a CPU está alocada para o processo B.

Os quadrantes:

→ indicam que ambos os processos necessitam da impressora.
→ indicam que ambos os processos necessitam da plotadora.

Com base nisso, o ponto **p** indica que nenhum processo executou ainda, o **q** indica que o processo A executou, o ponto **r** indica que o processo B executou, o **S** indica que o processo A executou e nessa fase ele **já obteve a impressora**.

Ao chegar no instante **t**, o processo B requisita a plotadora, porém, se ele for atendido, no instante I6, o processo vai parar esperando a impressora, que só estará liberada no instante I3 pelo processo A, que até lá já terá requisitado a plotadora, mas a mesma já estará alocada ao processo B, ou seja, qualquer um dos estados demarcados vão levar a uma condição de impasse (I2, I3, I6, I7). Nesse caso, fica claro que a única solução é terminar de executar o processo A até o final, que é o instante I4 e depois executar o processo B, assim, ambos chegarão no estado **u**.

***** Estados seguros e inseguros

A qualquer instante há um estado atual em que os processos estão. Um estado é considerado **seguro** se ele não está em situação de impasse e se existe alguma ordem de escalonamento na qual todo processo possa ser executado até a sua conclusão, mesmo que, repentinamente, todos eles requisitem, de uma só vez, o máximo possível de recursos.

Se a requisição de um processo gera um estado inseguro, ela não pode ser aceita. Vale ressaltar que **um estado inseguro NÃO é uma situação de impasse,** pois o sistema ainda pode funcionar normalmente e algum processo pode liberar algum recurso que permita a continuação. A diferença é que a partir de um estado seguro, o sistema pode GARANTIR que todos os processos terminarão, num estado inseguro não existe essa garantia.

* Algoritmo do banqueiro para um único recurso

Esse algoritmo foi desenvolvido por Djikstra em 1965 e é uma extensão do algoritmo de detecção de impasses com um recurso de cada tipo. Esse algoritmo utiliza a analogia de um banqueiro, onde o mesmo possui créditos e deve liberar esses créditos aos clientes, nesse caso, os clientes são os processos, os créditos são os recursos e o banqueiro é o S.O.

Esse algoritmo considera cada solicitação de empréstimo quando ela ocorre, analisando se a sua concessão levará a um estado seguro. Para saber se um estado é seguro, o banqueiro verifica se ele dispõe de recursos suficientes para satisfazer algum dos clientes. Se sim, presume-se que os empréstimos a esse cliente serão pagos (devolvidos); Em seguida, o cliente mais próximo ao limite é considerado e assim por diante. Se todos os empréstimos puderem ser pagos, o estado será seguro e a requisição inicial poderá ser atendida.

* Algoritmo do banqueiro com múltiplos recursos

Esse algoritmo utiliza as matrizes C e R de recursos alocados e recursos necessários. Os processos devem declarar antes da execução todos os recursos de que vão precisar, para que o sistema possa calcular os recursos necessários para cada processo.

Figura 3-13 O algoritmo do banqueiro com múltiplos recursos.

E → Recursos existentes
P → Recursos alocados
A → Recursos disponíveis

Embora esse algoritmo seja bom na teoria, na prática ele é inútil, pois os processos raramente sabem de quantos recursos vão precisar, o número de processos não é fixo e os recursos podem desaparecer repentinamente. São poucos, ou talvez nenhum sistema, que utilizam o algoritmo do banqueiro para evitar impasses.

Prevenção de impasses

Evitar impasses é praticamente impossível, pois requer informações que geralmente não se sabe. Se pudermos garantir que pelo menos uma das quatro condições para ocorrência de impasses seja negada, estruturalmente os impasses serão impossíveis.

* Atacando a condição de exclusão mútua

Se nunca acontecer de um recurso ser alocado exclusivamente a um único processo, nunca teremos impasses, obviamente, que dependendo do recurso, o uso simultâneo seria caótico.

Usando uma técnica de spooling, vários processos poderão gerar suas saídas ao mesmo tempo. Nesse modelo, levando em consideração uma impressora, o único processo que realmente requisita uma impressora física é o daemon de impressão. Como o daemon nunca requisita qualquer outro recurso, não se tem impasses envolvendo a impressora.

Para não ocorrer problemas, os daemons são programados para só imprimir após todo o arquivo de saída estar disponível. A ideia aqui é manter os processos alocando seus dados em um spool de dados antes de imprimir, a impressão só ocorre se o documento estiver completo no spool.

Se ocorrer de dois processos preencherem cada um metade do spool, o espaço acabaria e não poderiam continuar, isso é o que se conhece como **impasse em disco**. Uma atitude para amenizar esse problema seria evitar alocar um recurso quando ele não for absolutamente necessário e tentar assegurar que o menor número possível de processos possa, de fato, requisitar o recurso.

* Atacando a condição de posse e espera

Essa estratégia tem como objetivo impedir que processos que já mantém a posse de recursos esperem por mais recursos. Pode-se exigir que todos os processos requisitem todos os seus recursos antes de iniciar, ou aloca tudo de uma vez ou espera. O problema é que os processos geralmente não sabem, a princípio, exatamente quantos recursos eles precisarão ao longo da execução, senão o algoritmo do banqueiro poderia ser usado. Outro problema é que o uso dos recursos não será otimizado.

Uma possível solução é exigir que um processo que requisita um recurso, primeiro libere temporariamente os que estão em sua posse e depois tente obter os que precisa de uma só vez.

* Atacando a condição de não preempção

Dependendo do caso isso é complicado na melhor das hipóteses e impossível na pior delas, por exemplo, tomar a impressora a força no meio de uma impressão. Pode-se virtualizar os recursos, porém, nem todos os recursos podem ser virtualizados. Por exemplo, registros de banco de dados ou tabelas internas do S.O devem ser travados para uso. Isso é favorável para impasses.

* Atacando a condição de espera circular

Existem várias maneiras de se eliminar a espera circular. Um meio simples é ter uma regra que determine que um processo tenha permissão para possuir somente um recurso de cada vez. Se ele necessitar de um

segundo recurso, deverá liberar o primeiro. Dependendo do caso isso é inaceitável.

Outra maneira é fornecer uma numeração global de todos os recursos, os processos então poderiam requisitar os recursos sempre que necessário, mas todas as solicitações devem ser feitas em ordem numérica. Com essa regra, o grafo de alocação nunca conterá impasses. Por exemplo, suponha 4 recursos numerados da seguinte forma:

1 – Impressora
2 – Scanner
3 – Plotter
4 – Unidade de CD-ROM

A ideia é que os processos solicitarão recursos de ordem mais alta, não poderão solicitar recursos já alocados e a requisição deve seguir uma ordem crescente. Então, se um processo necessitar da impressora e do scanner, ele primeiro deve solicitar a impressora e só depois o scanner. Caso a impressora já esteja alocada, esse processo não alocará o scanner, pois as alocações devem ser feitas na ordem.

O maior problema desse algoritmo é que pode ser impossível encontrar uma ordem que satisfaça todos os processos.

Condição	**Abordagem contra impasses**
Exclusão mútua	Usar Spool
Posse e Espera	Requisitar inicialmente todos os recursos necessários
Não preempção	Retomar os recursos alocados
Espera circular	Ordenar numericamente os recursos

Referências bibliográficas:

TANENBAUM, Andrew. 2ª ed. **Sistemas Operacionais Modernos,** Editora Pearson, 2003.

SILBERSCHATZ, Abraham. **Sistemas Operacionais com JAVA,** 6ª ed. Editora Campus

MACHADO, Francis B. **Arquitetura de Sistemas Operacionais,** 4ª ed, LTC, 2007.