Sistemas Operacionais

Prof. Robson de Souza

Aulas 23 e 24

Conteúdo: Detecção e recuperação de impasses

Na técnica de detecção e recuperação de impasses, não ocorre prevenção, o sistema deixa o impasse ocorrer e então trata o impasse tentando detectar e agir de alguma forma para se recuperar.

Detecção de impasses com um recurso de cada tipo

Nesse caso, só existe um recurso de cada tipo (uma única impressora, uma única unidade de CD, etc.) Para esse sistema, podemos construir um grafo de recursos. Um impasse vai ocorrer se o grafo tiver um ou mais ciclos. Qualquer processo que fizer parte de um ciclo estará em impasse.

Um algoritmo simples para detecção de impasses seria inspecionar um grafo e terminar quando encontrar um ciclo ou quando perceber que não existe nenhum.

Detecção de impasses com múltiplos recursos de cada tipo

Quando existem várias cópias de algum recurso, é necessário um método diferente para detectar impasses. Um algoritmo que pode funcionar nesse caso seria um algoritmo de matrizes para a detecção de impasses entre n processos, de P1 a Pn. Seja m o número de classes de recursos, com E1 recursos de classe 1, E2 recursos de classe 2, Ei recursos de classe i (1 <= i <= m).

E → Vetor de recursos existentes, fornece o número total de instâncias de cada recurso existente.

Se um recurso estiver alocado a um processo, esse recurso não estará disponível. Seja A o vetor de recursos disponíveis, Ai fornece o número de instâncias do recurso i disponíveis nesse instante, por exemplo, se todos os recursos de classe 1 estiverem alocados, A1=0.

As informações referentes à alocação dos recursos e a requisição dos mesmos serão armazenadas em duas matrizes, uma matriz C e uma matriz R.

C → Matriz de alocação atual.
R → Matriz de requisição.

A i-ésima linha de C informa quantas instâncias de cada classe de recurso o processo Pi possui no momento.
Cij → Número de instâncias do recurso j que estão alocadas ao processo i.
Rij → Número de instâncias do recurso j que Pi quer.

Cada recurso está alocado a um processo ou se encontra disponível. Com isso, se somarmos todas as instâncias do recurso j já alocadas e adicionarmos as instâncias ainda disponíveis, teremos o número de instâncias existentes daquela classe de recursos.

(E1, E2, E3, … , Em) → Recursos existentes.
(A1, A2, A3, … , Am) → Recursos disponíveis.

```
|C11 C12 … C1m|                              |R11 R12 … R1m|
|C21 C22 … C2m|                              |R21 R22 … R2m|
|C31 C32 … C3m|  → Matriz de alocação atual  |R31 R32 … R3m|  → Matriz de Requisições
|…   …   …   |                              |…   …   …   |
|Cn1 Cn2 … Cnm|                              |Rn1 Rn2 … Rnm|
```

Para exemplificar, nas matrizes acima, a linha 2 da matriz de requisições informa qual a necessidade do

processo 2.

Com essas informações, é possível saber se um impasse ocorreu, por exemplo, supondo que existam 2 classes de recursos (1 e 2) e que existem 2 recursos da classe 1 e 3 recursos da classe 2. Se existirem 5 processos, todos eles necessitarem dos dois recursos para executar e os processos receberem os recursos do seguinte modo:

Processo 1 → recurso 1.
Processo 2 → recurso 2.
processo 3 → recurso 2.
processo 4 → recurso 1.
Processo 5 → recurso 2.

Ao transferir esses dados para as matrizes, isso ficará assim:

E → (2, 3) A → (0, 0)

C → | 1 0 |
| 0 1 |
| 0 1 |
| 1 0 |
| 0 1 |

R → | 1 1 |
| 1 1 |
| 1 1 |
| 1 1 |
| 1 1 |

Nesse exemplo, fica fácil perceber que todos os processos precisam de um recurso que já está alocado e não possui mais instâncias disponíveis, logo, temos um impasse.

Recuperação de impasses

Uma vez que um impasse é detectado, é necessário recuperar e colocar o sistema de volta em condições normais de execução, para isso, existem algumas técnicas de recuperação de impasses.

*Recuperação por meio de preempção

Às vezes pode ser necessária uma intervenção manual para retomar um recurso provisoriamente do proprietário atual para dá-lo a outro processo, mas nem sempre é necessária a intervenção manual.
A habilidade para retirar um recurso de um processo, entregá-lo a outro processo e depois devolvê-lo ao primeiro, sem que o processo perceba, é altamente dependente da natureza do recurso, essa operação pode se tornar até impossível.

Escolher o processo que será suspenso depende amplamente de quais processos têm recursos passíveis de serem facilmente retomados.

*Recuperação por meio de retrocesso

Pode-se fazer com que o processo periodicamente gere pontos de gravação (checkpoints). Verificar um processo nesse caso, significa ter seu estado guardado em um arquivo, de modo que possa ser reinicializado posteriormente. Esse arquivo contém a imagem da memória e o estado dos recursos (quais recursos estão em um determinado instante alocados ao processo).

Cada checkpoint deve ser escrito em um novo arquivo para maior eficiência, não escrito sobre o arquivo anterior, de modo que ao ser executado, um processo acumule uma sequencia completa de checkpoints.

Quando um impasse é detectado, é fácil ver quais recursos são necessários. Para fazer a recuperação, um processo que tem um dos recursos necessários é revertido para um estado em um instante anterior ao momento em que adquiriu algum outro recurso. Isso é feito reinicializando o processamento a partir de um de seus checkpoints anteriores.

Nesse caso, todo o trabalho, recursos alocados, etc, depois do checkpoint são perdidos. A execução deverá ser refeita a partir do checkpoint.

*Recuperação por meio da eliminação de processos

“Matar” um ou mais processos é a forma mais grosseira, porém, a mais simples para recuperação de impasses. Pode-se matar um processo no ciclo e torcer para que os outros continuem. Se não for suficiente pode-se ir matando outros processos até o ciclo ser quebrado.

A ideia é matar processos que tenham recursos que podem ser destinados a processos que estão em impasse, logo, processos que não estão no ciclo podem ser vítimas.

Sempre que possível, é melhor matar um processo que pode ser reexecutado desde seu início.

É óbvio que esse método de recuperação precisa ser usado com muita cautela, pois ao sair matando processos pode ser que um processo importante seja vítima, ou pode ser que vários processos sejam mortos e ainda assim o impasse continue. O ideal nesse caso é escolher bem os processos que se quer matar para não ocorrer esse tipo de problema.

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.