Sistemas Operacionais

Detectar, recuperar e evitar deadlocks

Profª Drª Thaína Aparecida Azevedo Tosta

tosta.thaina@unifesp.br

UNIFESP
Universidade Federal de São Paulo

São José dos Campos
2024/01

Aula passada

- Conceitos iniciais
- Recursos
- Introdução aos impasses
- Algoritmo do avestruz

Sumário

- Detecção de impasses
- Recuperação de um impasse
- Evitando impasses

Objetivo: quais as limitações do algoritmo do banqueiro?

Detecção de impasses

Além do algoritmo do avestruz, podemos deixar o impasse acontecer, detectá-lo e tomar medidas para se recuperar dele, como:

- Com um recurso de cada tipo;
- Com múltiplos recursos de cada tipo.

Detecção de impasses

Detecção de impasses com um recurso de cada tipo

Não mais do que um de cada classe de recurso.

1. O processo A possui R e solicita S.
2. O processo B não possui nada, mas solicita T.
3. O processo C não possui nada, mas solicita S.
4. O processo D possui U e solicita S e T.
5. O processo E possui T e solicita V.
6. O processo F possui W e solicita S.
7. O processo G possui V e solicita U.

Esse sistema está em impasse? E se estiver, quais processos estão envolvidos?

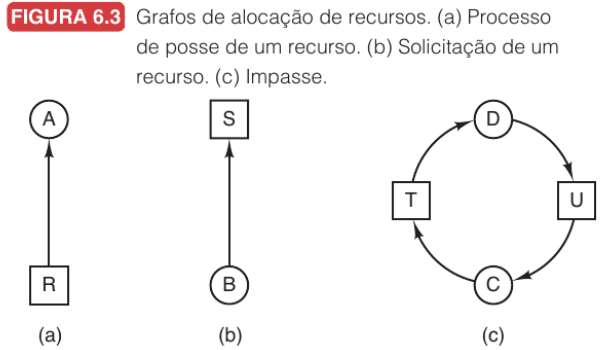
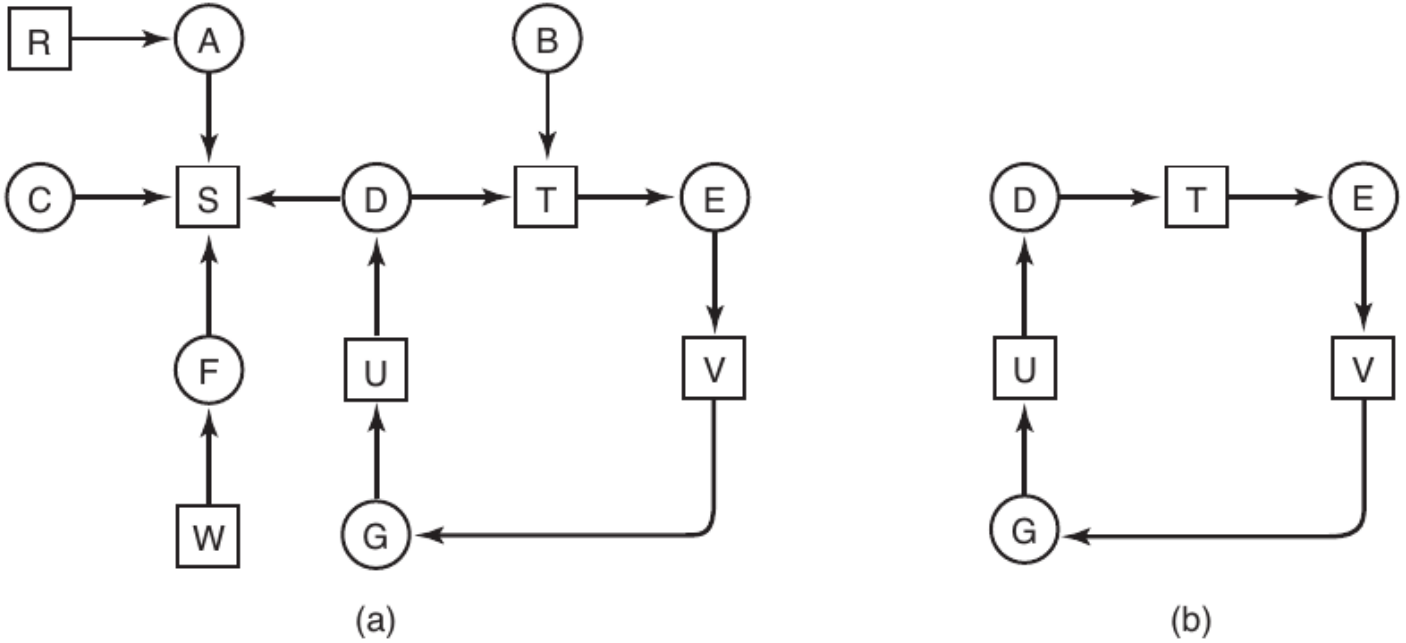


FIGURA 6.3 Grafos de alocação de recursos. (a) Processo de posse de um recurso. (b) Solicitação de um recurso. (c) Impasse.

Detecção de impasses

Detecção de impasses com um recurso de cada tipo

Quais processos estão em impasse? E o recurso S?



Detecção de impasses

Detecção de impasses com um recurso de cada tipo

Algoritmo simples que inspeciona um grafo e termina quando encontra um ciclo ou não, com lista de nós (L) e lista de arcos.

1. Para cada nó, N, no grafo, execute os cinco passos a seguir com N como o nó de partida.
2. Inicialize L como uma lista vazia e designe todos os arcos como desmarcados.
3. Adicione o nó atual ao final de L e confira para ver se o nó aparece agora em L duas vezes. Se ele aparecer, o grafo contém um ciclo (listado em L) e o algoritmo termina.
4. A partir do referido nó, verifique se há algum arco de saída desmarcado. Se afirmativo, vá para o passo 5; se não, vá para o passo 6.
5. Escolha aleatoriamente um arco de saída desmarcado e marque-o. Então siga-o para gerar o novo nó atual e vá para o passo 3.
6. Se esse nó é o inicial, o grafo não contém ciclo algum e o algoritmo termina. De outra maneira, chegamos agora a um beco sem saída. Remova-o e volte ao nó anterior, isto é, aquele que era atual imediatamente antes desse, faça dele o nó atual e vá para o passo 3.

Detecção de impasses

Detecção de impasses com um recurso de cada tipo

Algoritmo simples que inspeciona um grafo e termina quando encontra um ciclo ou não, com lista de nós (*L*) e lista de arcos.

Começar da esquerda para a direita, de cima para baixo (RABCSDTEF...).

L = []

L = [R]

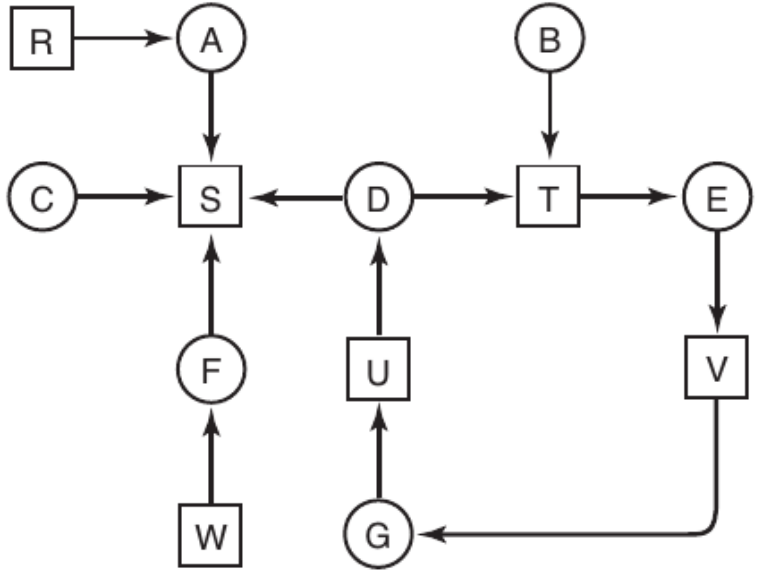
L = [R, A]

L = [R, A, S]

S não tem arcos de saída, forçando-nos a recuar para A

A não tem mais arcos de saída, então recuamos para R

R não tem mais arcos de saída, então finalizamos por R.



Detecção de impasses

Detecção de impasses com um recurso de cada tipo

Algoritmo simples que inspeciona um grafo e termina quando encontra um ciclo ou não, com lista de nós (*L*) e lista de arcos.

Começar da esquerda para a direita, de cima para baixo (RABCSDTEF...).

L = []

L = [B]

L = [B, T]

L = [B, T, E]

L = [B, T, E, V]

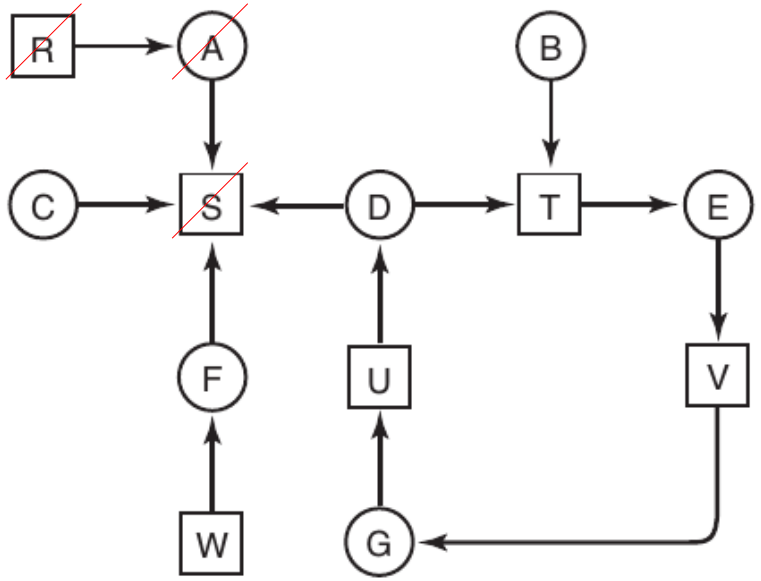
L = [B, T, E, V, G]

L = [B, T, E, V, G, U]

L = [B, T, E, V, G, U, D]

L = [B, T, E, V, G, U, D, T]

É ótimo?



Detecção de impasses

Detecção de impasses com múltiplos recursos de cada tipo

Com múltiplas cópias de alguns dos recursos, é necessária uma abordagem diferente para detectar impasses.

n: processos (P_1 a P_n)

m: nº de classes de recursos

E_1: recursos da classe 1

E_2: recursos da classe 2

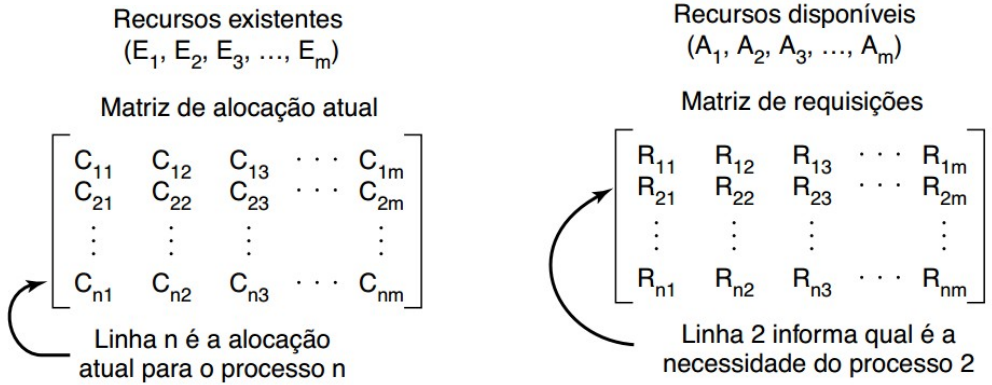
E_i: recursos da classe i

E: vetor de recursos existentes

A: vetor de recursos disponíveis

C: matriz de alocação atual

R: matriz de requisição



$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$

Detecção de impasses

Detecção de impasses com múltiplos recursos de cada tipo

P_1 não consegue executar;
P_2 não consegue executar;
P_3 executa e finaliza com A = (2 2 2 0);
Com isso, P_2 executa e finaliza com A = (4 2 2 1);
Com isso, P_1 executa.

Se P_3 precisar de R = (2 1 0 1)?

Unidades de fita | Plotters | Scanners | Blu-rays

E = (4 2 3 1)

Vetor de recursos existentes

Unidades de fita | Plotters | Scanners | Blu-rays

A = (2 1 0 0)

Vetor de recursos disponíveis

Matriz alocação atual

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Matriz de requisições

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Detecção de impasses

Detecção de impasses com múltiplos recursos de cada tipo

Entrada: todos os processos desmarcados

Suposição: todos os processos mantêm todos os recursos adquiridos até que terminem

1. Procure por um processo desmarcado, Pi, para o qual a i-ésima linha de R seja menor ou igual a A.

2. Se um processo assim for encontrado, adicione a i-ésima linha de C a A, marque o processo e volte ao passo 1.

3. Se esse processo não existir, o algoritmo termina.

Saída: sabe-se que quaisquer processos desmarcados estão em situação de impasse

Recuperação de um impasse

Suponha que nosso algoritmo de detecção de impasses teve sucesso e detectou um impasse, e agora?

- Recuperação mediante preempção
- Recuperação mediante retrocesso
- Recuperação mediante a eliminação de processos

Recuperação de um impasse

Recuperação mediante preempção

- Em alguns casos pode ser viável tomar temporariamente um recurso do seu proprietário atual e dá-lo a outro processo;
- Exemplo: intervenção de operador para tomar uma impressora do processo-proprietário;
- Dependente da natureza do processo, sendo difícil ou impossível escolher o processo que tem recursos que podem ser facilmente devolvidos.

Recuperação de um impasse

Recuperação mediante retrocesso

- Se a probabilidade da ocorrência de impasses é grande, os processos podem gerar pontos de salvaguarda (*checkpoints*) com escrita em arquivo do seu estado, imagem da memória e quais recursos estão alocados a ele;
- Novos arquivos devem ser escritos para não haver sobrescrita;
- O processo que tem um recurso necessário é retrocedido pelos *checkpoints*.

Recuperação de um impasse

Recuperação mediante a eliminação de processos

- A maneira mais bruta de eliminar um impasse, mas também a mais simples, é matar um ou mais processos;
  - Se isso não ajudar, essa ação pode ser repetida até que o ciclo seja rompido;
- Exemplo: processo 1 (tem impressora e quer um plotter), processo 2 (tem plotter e quer impressora) e ~~processo 3 (tem impressora e plotter)~~;
- Escolha cuidadosa de processo que pode ser reexecutado: atualização de banco de dados.

Evitando impasses

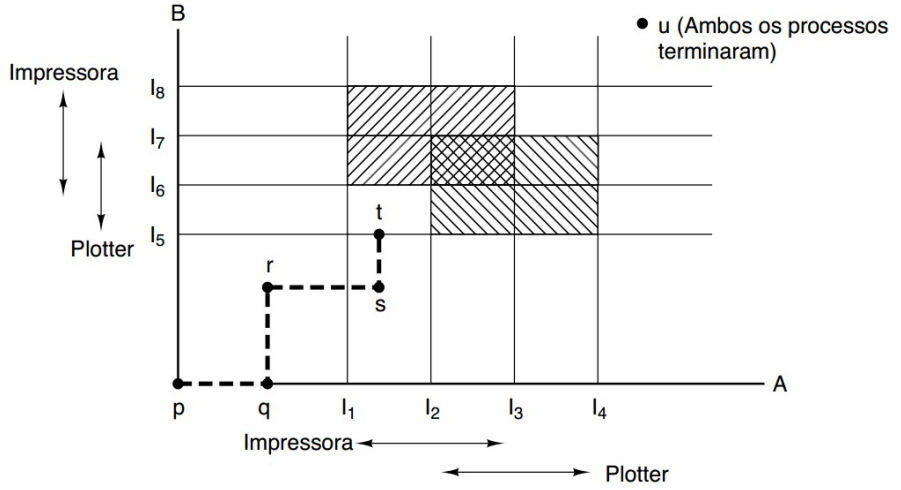
- Na maioria dos sistemas, os recursos são solicitados um de cada vez, diferente do que vimos na detecção de impasses;
- O sistema precisa ser capaz de decidir se conceder um recurso é seguro ou não e fazer a alocação somente quando for;
- Existe um algoritmo pra isso? Sim e não.

Evitando impasses

Trajetórias de recursos

- O eixo horizontal representa o número de instruções executadas pelo processo A. O eixo vertical representa o número de instruções executadas pelo processo B;
- Todo ponto no diagrama representa um estado de junção dos dois processos.

A trajetória de recursos de dois processos.



- I_1 - I_3: A solicita e recebe impressora;
- I_2 - I_4: A solicita e recebe plotter;
- I_5 - I_7: B solicita e recebe plotter;
- I_6 - I_8: B solicita e recebe impressora;
- p - q: escalonador escolhe A;
- q - r: escalonador escolhe B.

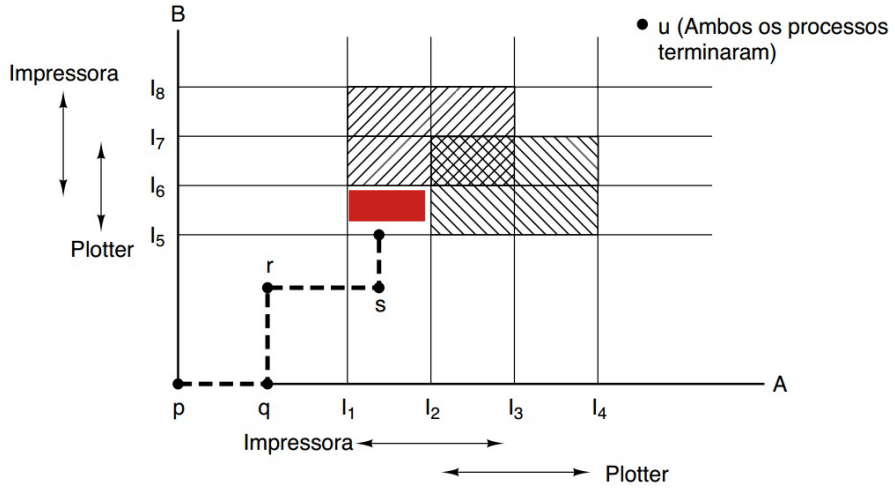
Com um único processador, os caminhos podem ser diagonais? O que são as regiões sombreadas?

Evitando impasses

Trajetórias de recursos

- Na interseção de I_2 e I_6, A está solicitando a plotter e B, a impressora, e ambas já foram alocadas;
- Em t (região em vermelho), o sistema precisa decidir se concede ou não o plotter a B;
- Se sim, o sistema entrará em uma região insegura e eventualmente em uma situação de impasse;
- Suspender B e executar A até I_4 é uma boa solução?

A trajetória de recursos de dois processos.



Evitando impasses

Estados seguros e inseguros

- Os principais algoritmos para evitar impasses são baseados no conceito de estados seguros;
- Temos um estado seguro se existir alguma ordem de escalonamento na qual todos os processos puderem ser executados até sua conclusão mesmo que todos eles subitamente solicitem seu número máximo de recursos imediatamente;
- Esse cenário é seguro porque o escalonador pode executar B - C - A, sem impasses.

	Possui máximo	
A	3	9
B	2	4
C	2	7

Disponível: 3

(a)

	Possui máximo	
A	3	9
B	4	4
C	2	7

Disponível: 1

(b)

	Possui máximo	
A	3	9
B	0	–
C	2	7

Disponível: 5

(c)

	Possui máximo	
A	3	9
B	0	–
C	7	7

Disponível: 0

(d)

	Possui máximo	
A	3	9
B	0	–
C	0	–

Disponível: 7

(e)

Evitando impasses

Estados seguros e inseguros

- De (a) para (b), saímos de um estado seguro para um inseguro;
- Um estado inseguro não é um estado em situação de impasse, apenas indica que ele pode acontecer;
- A partir de um estado seguro, o sistema pode **garantir** que todos os processos terminarão; a partir de um inseguro, nenhuma garantia nesse sentido pode ser dada.

Possui máximo

A	3	9
B	2	4
C	2	7

Disponível: 3

(a)

Possui máximo

A	4	9
B	2	4
C	2	7

Disponível: 2

(b)

Possui máximo

A	4	9
B	4	4
C	2	7

Disponível: 0

(c)

Possui máximo

A	4	9
B	–	–
C	2	7

Disponível: 4

(d)

Evitando impasses

O algoritmo do banqueiro para um único recurso

Ele é modelado da maneira pela qual um banqueiro de uma cidade pequena poderia lidar com um grupo de clientes para os quais ele concedeu linhas de crédito;

- Considera cada solicitação à medida que ela ocorre, vendo se concedê-la leva a um estado seguro;
- Para ver se um estado é seguro, o banqueiro confere para ver se ele tem recursos suficientes para satisfazer algum cliente.

  (a) seguro (b) seguro (c) inseguro

(a)

	Possui	máximo
A	0	6
B	0	5
C	0	4
D	0	7

Disponível: 10

(b)

	Possui	máximo
A	1	6
B	1	5
C	2	4
D	4	7

Disponível: 2

(c)

	Possui	máximo
A	1	6
B	2	5
C	2	4
D	4	7

Disponível: 1

Evitando impasses

O algoritmo do banqueiro com múltiplos recursos

- E: recursos existentes;
- P: recursos alocados;
- A: recursos disponíveis.

1. Partindo do estado seguro alocado, se B solicita e recebe uma impressora, A = (1010) e D pode terminar solicitando a última impressora;

   Com o fim de D, A = (2121), o que permite o fim de A ou E.

2. Partindo do estado seguro alocado, se B solicita e recebe uma impressora (A = (1010)) e E solicita a última impressora (A = (1000)), essa solicitação deve ser negada.

Processo	Unidades de fita	Plotters	Impressoras	Blu-rays
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Recursos alocados

Processo	Unidades de fita	Plotters	Impressoras	Blu-rays
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Recursos ainda necessários

E = (6342)
P = (5322)
A = (1020)

Evitando impasses

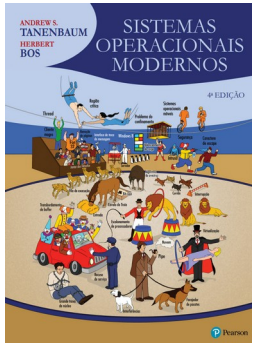
- Algoritmo para verificar se um estado é seguro:

1. Procure por uma linha, R, cujas necessidades de recursos não atendidas sejam todas menores ou iguais a A. Se essa linha não existir, o sistema irá em algum momento chegar a um impasse, dado que nenhum processo pode executar até o fim (presumindo que os processos mantenham todos os recursos até sua saída);
2. Presuma que o processo da linha escolhida solicita todos os recursos que ele precisa (o que é garantido que seja possível) e termina. Marque esse processo como concluído e adicione todos os seus recursos ao vetor A;
3. Repita os passos 1 e 2 até que todos os processos estejam marcados como terminados (caso em que o estado inicial era seguro) ou nenhum processo cujas necessidades de recursos possam ser atendidas seja deixado (caso em que o sistema não era seguro).

- Se vários processos são elegíveis para serem escolhidos no passo 1, não importa qual seja escolhido.

Quais as limitações do algoritmo do banqueiro?

Referências



TANENBAUM, Andrew S.; BOS, Herbert. Sistemas operacionais modernos. 4. edição. São Paulo: Pearson, 2016. xviii, 758 p. ISBN 9788543005676.