Apresentação de Sistemas Operacionais

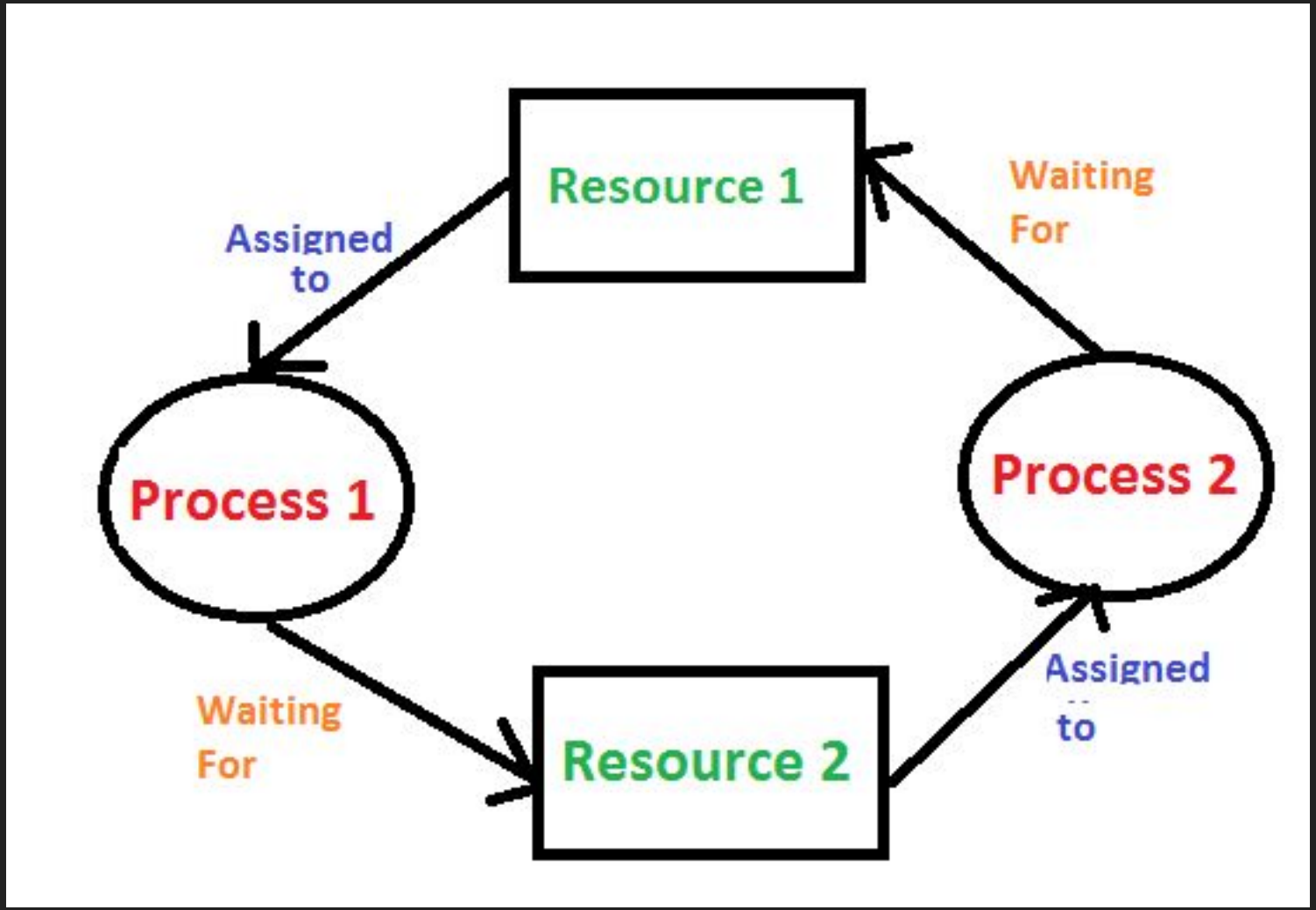
Impasses

Daniel Sant' Anna Andrade
Luan Gadioli Mendonça Berto

Deadlock

Dispositivos e recursos são compartilhados a todo momento: impressora, disco, arquivos, etc…;

- Ocorre com um conjunto de processos e recursos não-preemptíveis;
- Pode ocorrer mesmo que haja somente um processo no SO, caso esse processo utilize múltiplos threads e elas requisitam recursos de outros threads no mesmo processo;
- Independe da quantidade de recursos disponíveis no sistema;


Resource 1
Assigned to
Waiting For
Process 1
Process 2
Waiting For
Resource 2
Assigned to

Recursos

Preemptivos: podem ser retirados do processo sem maiores prejuízos, SO pode tomar de volta sem risco de problemas ;

- exemplo: memória, arquivo aberto em modo leitura;

Não-preemptivos: não podem ser retirados do processo, pois, Se forem tomados de volta, há risco de problemas ;

- exemplo: impressora, CD-ROM, arquivo aberto em modo escrita;

Recursos

Requisição de recursos

1. Requisição do recurso
2. Utilização do recurso
3. Liberação do recurso;

Se o recurso requerido não está disponível, duas situações podem ocorrer:

- Processo que requisitou o recurso fica bloqueado até que o recurso seja liberado
- Processo que requisitou o recurso falha, e depois de um certo tempo tenta novamente requisitar o recurso

Condições para Deadlock

- Não-preempção: recursos já alocados a processos não podem ser tomados à força. Eles precisam ser liberados pelo processo que possui a sua posse;
- Exclusividade mútua: cada recurso ou está alocado a um processo ou está disponível;
- Posse-e-espera: cada processo pode solicitar um recurso, ter esse recurso alocado para si e ficar bloqueado, esperando por um outro recurso;
- Espera circular: deve existir uma cadeia circular de dois ou mais processos, cada um dos quais esperando por um recurso que está com o próximo integrante da cadeia.

Como tratar deadlock

- Ignorar o problema;
- Detectar e recuperar o problema;
- Evitar dinamicamente o problema
- Fazer alocação cuidadosa de recursos;
- Prevenir o problema por meio da não satisfação de uma das quatro condições citadas anteriormente;

Ignorar o problema

Algoritmo do Avestruz - Enfie a cabeça na areia e finja que não há problemas. O algoritmo do avestruz é uma estratégia para tratamento de deadlock, que consiste em simplesmente ignorar o problema. Com base no fato de que eles possam ser raros e não precisaria sobrecarregar a cpu com codigos extras de tratamento, onde é tolerável reiniciar o sistema com uma ação corretiva. Unix e windows seguem essa abordagem.

- Alto custo
- Acontecer raramente
- algoritmo do avestruz

Detecção e recuperação de impasses

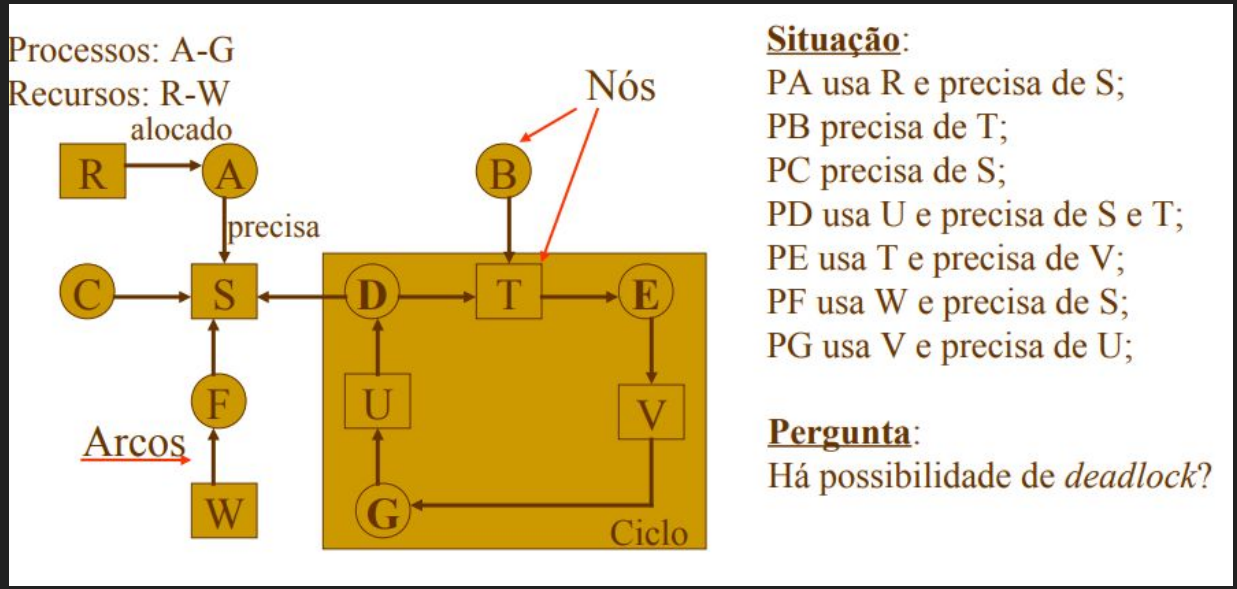
O sistema não tenta evitar a ocorrência dos impasses. Em vez disso, ele os deixa ocorrer, tenta detectá-los quando acontecem e então toma alguma medida para recuperar-se após o fato.

Algoritmos:

- Detecção com um recurso de cada tipo
- Detecção com vários recursos de cada tipo
- Recuperação por meio de preempção
- Recuperação por meio de rollback (volta ao passado);
- Recuperação por meio de eliminação de processos

Detecção com um recurso de cada tipo

O primeiro passo é a construção de um grafo e se houver ciclos, existem potenciais deadlocks;



Detecção com vários recursos de cada tipo

Para esse algoritmo, o sistema, geralmente, procura periodicamente por deadlocks.

vetor de recursos existentes (E): Se classe1=unidade de fita e E1=2, então existem duas unidades de fita.

Vetor de recursos disponíveis (A): Se ambas as unidades de fita estiverem alocadas, A1=0.

Duas matrizes:

C: matriz de alocação corrente;

- Cij: número de instâncias do recurso j entregues ao processo i;

R: matriz de requisições;

- Rij: número de instâncias do recurso j que o processo i precisa;

Deadlocks

4 unidades de fita;
2 *plotter*;
3 impressoras;
1 unidade de CD-ROM

Recursos existentes
E = (4 2 3 1)
UF P I UCD

Matriz de alocação

	UF	P	I	UCD	
C =	0	0	1	0	P_1
	2	0	0	1	P_2
	0	1	2	0	P_3

Recursos

Três processos:
P_1 usa uma impressora;
P_2 usa duas unidades de fita e uma de CD-ROM;
P_3 usa um *plotter* e duas impressoras;

Cada processo precisa de outros recursos (R);

Recursos disponíveis
A = (2 1 0 0)
UF P I UCD

Matriz de requisições

	UF	P	I	UCD	
R =	2	0	0	1	P_1
	1	0	1	0	P_2
	2	1	0	0	P_3

17

4 unidades de fita;
2 *plotters*;
3 impressoras;
1 unidade de CD-ROM

Requisições:
P_2 requisita duas unidade de fita, uma impressora e uma unidade de CD-ROM;

Recursos existentes
E = (4 2 3 1)

Recursos disponíveis
A = (2 1 0 0) P_3 pode rodar
A = (2 2 2 0)

Matriz de alocação

C =	0	0	1	0	P_1
	2	0	0	1	P_2
	0	0	0	0	P_3

Matriz de requisições

R =	2	0	0	1	P_1
	2	**0**	**1**	**1**	$\mathbf{P_2}$
	0	0	0	0	P_3

Nessa situação, nenhum processo pode ser atendido!

DEADLOCK

Recuperação de Deadlocks

Recuperação por meio de preempção - possibilidade de retirar temporariamente um recurso de seu atual dono (processo) e entregá-lo a outro processo;

Recuperação por meio de rollback (volta ao passado) - recursos alocados a um processo são armazenados em arquivos de verificação (checkpoint files); quando ocorre um deadlock, os processos voltam ao estado no qual estavam antes do deadlock (solução cara);

Recuperação por meio de eliminação de processos - processos que estão no ciclo com deadlock são retirados do ciclo; Melhor solução para processos que não causam algum efeito negativo ao sistema;

Estados do processo

Seguro - Não provocam deadlocks e há uma maneira de atender a todas as requisições pendentes finalizando normalmente todos os processos. Existe alguma ordem de escalonamento na qual todo o processo possa ser executado até a sua conclusão.

Inseguro - Podem provocar deadlocks, mas não necessariamente provocam. A partir de um estado inseguro, não é possível garantir que os processos terminarão corretamente

Evitar dinamicamente o problema

- Alocação individual de recursos à medida que o processo necessita (Conhecimento prévio dos recursos que serão utilizados)
- Soluções também utilizam matrizes
- Escalonamento cuidadoso demanda alto custo

Algoritmo do Banqueiro

• Quando um processo requisita um recurso deve esperar por este recurso.

• Quando um processo detém todos os recursos, deve devolver dentro de um tempo finito. O Banqueiro pode ser implementado com um tipo de recurso com várias instâncias ou vários recursos com várias instâncias.

• Considera cada requisição no momento em que ela ocorre verificando se essa requisição leva a um estado seguro; Se sim, a requisição é atendida.

• Senão, o atendimento é adiado para um outro momento.

Algoritmo do Banqueiro

Desvantagens

- Pouco utilizado, pois é difícil saber quais recursos serão necessários; Escalonamento cuidadoso é caro para o sistema.
- O número de processos é dinâmico e pode variar constantemente, tornando o algoritmo custoso.
- O número de processos é dinâmico e pode variar constantemente, tornando o algoritmo custoso

Vantagem

- Na teoria o algoritmo é ótimo.

Prevenção de impasses

- **Prevenção:** É atacar uma das condições necessárias para se ter o estado de deadlock.

- **Posse e Espera:** Exigir que todos os processos requisitem os recursos antes de iniciarem (um processo nunca tem que esperar por aquilo que precisa). Porém podem não saber quantos e quais recursos vão precisar no início da execução – e também retêm recursos que outros processos poderiam estar usando. Com isso o processo deve desistir de todos os recursos – para então requisitar todos os que são imediatamente necessários.

- **Não Preempção:** Prevenção de Deadlock Atacando a Condição de Não Preempção. Esta é uma opção inviável. O problema é que alguns recursos devem ser não-preemptivos, para evitar erros de computação. Exemplo: impressora.

- **Espera Circular:** A condição de espera circular pode ser evitada se um processo somente puder usar um recurso em um dado intervalo de tempo. ordenação numérica dos recursos

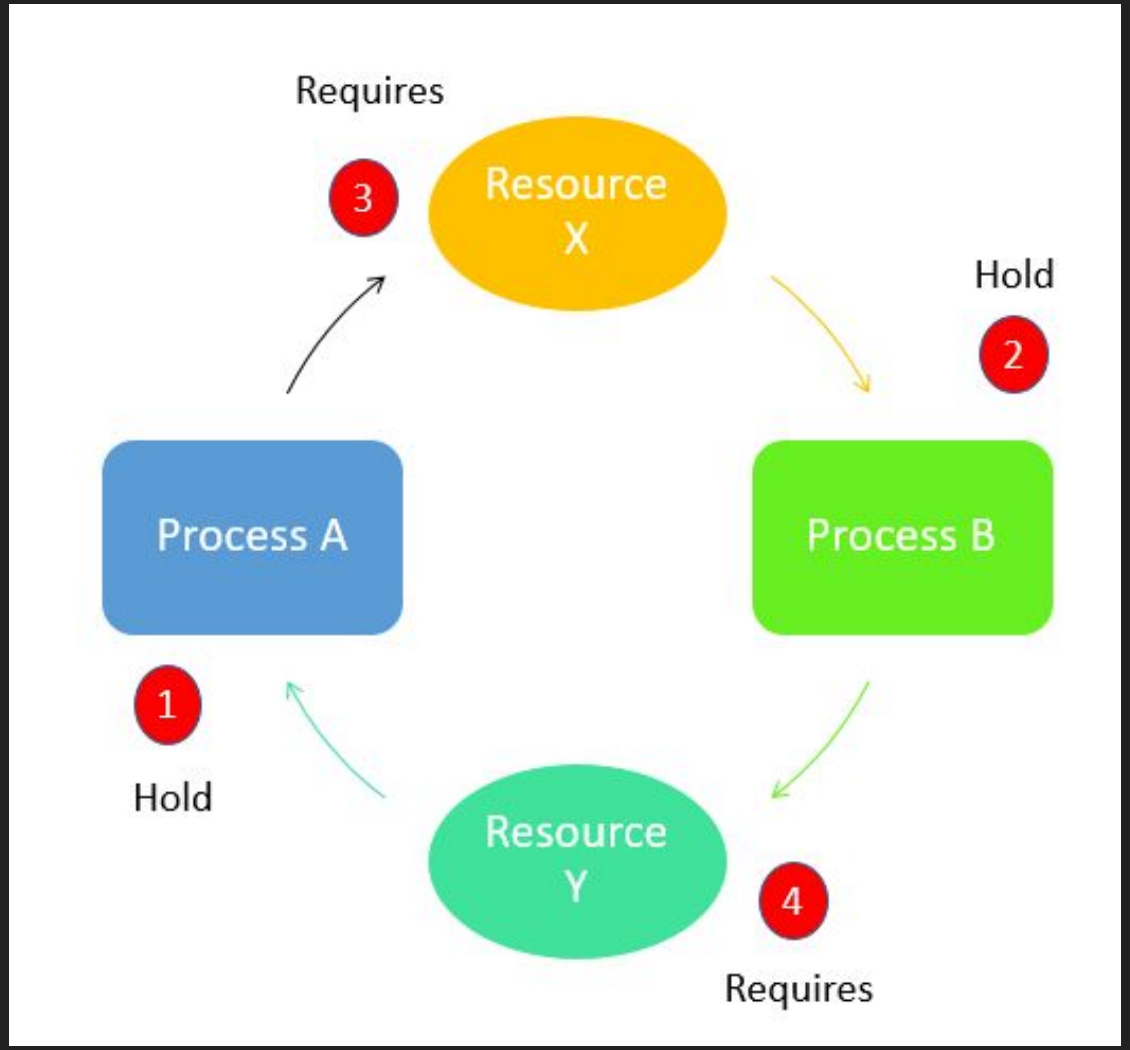
- Evitar alocar um recurso quando ele não for absolutamente necessário - tentar assegurar que o menor número possível de processos possa de fato requisitar o recurso.

Inanição (Starvation)

- Em um sistema dinâmico é necessário tomar uma decisão sobre quem recebe recurso. Porém pode levar a alguns processos nunca serem servidos, mesmo que não gere um impasse.
- A principal forma de se evitar Inanição seria com uma política de FIFO. Assim, qualquer processo receberá o recurso de que necessita.
- Alguns autores não fazem distinção entre a inanição e impasse, porque em ambos os casos não há um progresso.

Livelock

- Livelock é um caso especial de inanição de recursos, onde um processo específico não está progredindo.
- Risco em alguns algoritmos que detectam e se recuperam de um conflito, caso um processo execute uma ação, o algoritmo de detecção de deadlock poderá ser acionado repetidamente.
- Na maioria dos sistemas operacionais, livelocks são apenas ignorados, presumindo que a maioria dos usuários preferiria um livelock ocasional a uma restrição de todos os usuários a um processo


Requires
3
Resource X
Hold
2
Process A
Process B
1
Hold
Resource Y
4
Requires

Referências

TANENBAUM, A. S. , Sistemas Operacionais Modernos. 4ª ed. - São Paulo: Pearson Education do Brasil, 2016.