



Sistemas Operacionais

Deadlock

Prof. Fernando Parente Garcia



Deadlock Conceitos básicos

- Situação na qual um processo bloqueia outro e vice-versa e assim permanecerão para sempre;
- Pode ocorrer deadlock quando dois ou mais processos tentam utilizar um mesmo **recurso** ao mesmo tempo;
- **Recurso**
 - Algo que pode ser usado por somente um único processo em um dado instante de tempo. Um recurso pode ser um dispositivo de hardware ou uma informação;
 - Ex.: Espaço de memória, unidade de CD-ROM, impressora, registro de BD (operação de escrita) .



Deadlock

Conceitos básicos

- Tipos de recursos:
 - **Preemptível**
 - Recurso que pode ser retirado do processo proprietário e entregue a outro processo sem nenhum prejuízo.
 - Ex.: Memória.
 - **Não preemptível**
 - Recurso que não pode ser retirado do processo proprietário sem que a computação apresente falha.
 - Ex.: Gravador de CD.

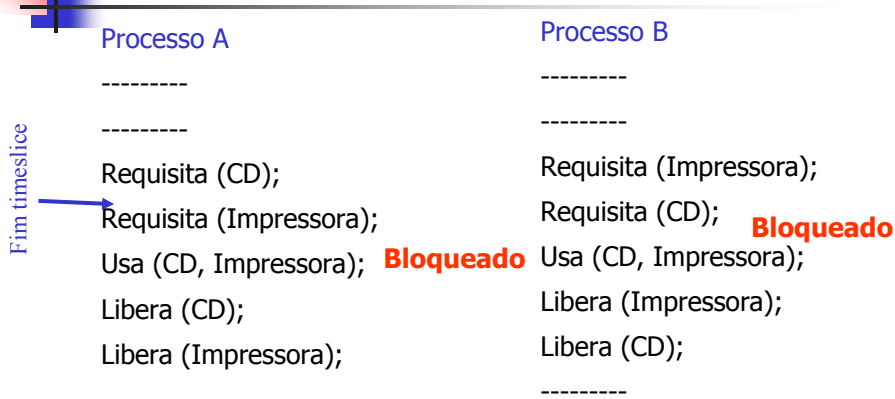


Deadlock

Conceitos básicos

- Sequência de eventos necessários ao uso de um determinado recurso:
 1. **Requisitar** o recurso.
 2. **Usar** o recurso.
 3. **Liberar** o recurso.
- Se o recurso não estiver disponível quando requisitado, o processo solicitante será forçado a esperar (bloqueado).

Deadlock Exemplo



Um conjunto de processos está em deadlock se cada processo estiver esperando por um evento que somente outro processo pertencente ao conjunto poderá fazer acontecer.

Deadlock Condições para ocorrência

1. **Condição de exclusão mútua:** em um determinado instante, o recurso ou está associado a um único processo ou está disponível.
2. **Condição de posse e espera:** processos que já retêm recursos podem requisitar novos recursos.
3. **Condição de não preempção:** recursos concedidos previamente a um processo não podem ser forçosamente tomados.
4. **Condição de espera circular:** deve existir um encadeamento circular de 2 ou mais processos. Cada um deles encontra-se à espera de um recurso que está sendo usado pelo membro seguinte dessa cadeia.

Somente ocorrerá deadlock se estas quatro condições estiverem presentes.



Deadlock Detecção

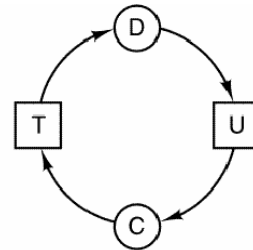
Modelagem através de grafos



Processo A
está de posse
do recurso R



Processo B está
bloqueado
aguardando o
recurso S



Os processos
C e D estão
em deadlock



Deadlock Detecção

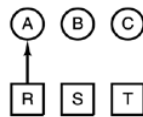
A
Requisita R
Requisita S
Libera R
Libera S
(a)

B
Requisita S
Requisita T
Libera S
Libera T
(b)

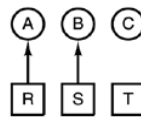
C
Requisita T
Requisita R
Libera T
Libera R
(c)

1. A requisita R
 2. B requisita S
 3. C requisita T
 4. A requisita S
 5. B requisita T
 6. C requisita R
- deadlock

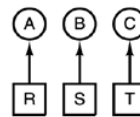
(d)



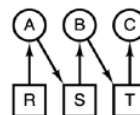
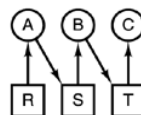
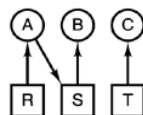
(e)



(f)



(g)

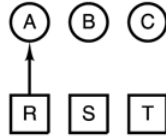




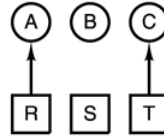
Deadlock Detecção

1. A requisita R
 2. C requisita T
 3. A requisita S
 4. C requisita R
 5. A libera R
 6. A libera S
- nenhum deadlock

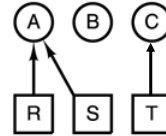
(k)



(l)

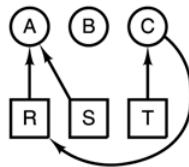


(m)

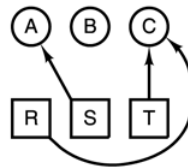


(n)

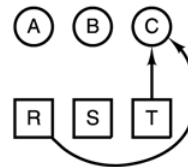
Não ocorre
deadlock



(o)



(p)



(q)



Estratégias usadas para tratar deadlocks

- Ignorar por completo o problema.
 - Algoritmo da avestruz;
- Detecção e recuperação.
 - Deixar os deadlocks ocorrerem, detectá-los e agir.
- Prevenção.
 - Negando estruturalmente uma das 4 condições necessárias para gerar um deadlock.



Deadlock

Algoritmo do Avestruz

Enterre sua cabeça na areia e finja que nada está acontecendo.


- Finge que o problema não existe;
- Razoável se
 - deadlocks ocorrem muito raramente
 - custo da prevenção é alto
 - Na maioria das vezes implica restrições não convencionais aos processos.
- UNIX e Windows seguem esta abordagem;
 - Ignoram o deadlock supondo que a maior parte dos usuários preferiria um deadlock ocasional a uma regra que restrinja cada usuário a somente um processo, um arquivo aberto e um de cada recurso;
- É uma ponderação entre **conveniência** e **correção**.



Deadlock


Deteção e Recuperação

- Essa técnica não tenta prevenir a ocorrência de deadlocks;
- Ela deixará que eles ocorram e tentará detectá-los a medida que isso acontecer;
- Agirá então, de alguma maneira para se recuperar após o fato.



Detecção de deadlocks com um recurso de cada tipo

- Existe somente um recurso de cada tipo;
 - Esse sistema pode ter um scanner, uma unidade de CD, um plotter, uma unidade de fita, mas não mais do que um recurso de cada tipo;
- Pode-se construir um grafo dos recursos;
 - Um deadlock existe se esse grafo contiver um ou mais ciclos;
 - Qualquer processo que faça parte de um ciclo está em situação de deadlock;
 - Se não houver nenhum ciclo, o sistema não estará em deadlock.



Detecção de deadlocks com um recurso de cada tipo

- Exemplo:
 - Imagine um sistema com sete processos (A, B, C, D, E, F e G) e com seis recursos (R, S, T, U, V e W):
 - O processo A possui o recurso R e requisita o recurso S.
 - O processo B possui o recurso S e requisita o recurso T.
 - O processo C nada possui e requisita o recurso S.
 - O processo D possui o recurso U e requisita o recurso S.
 - O processo E possui o recurso T e requisita o recurso V.
 - O processo F possui o recurso W e requisita o recurso S.
 - O processo G possui o recurso V e requisita o recurso U.

Quais processos estão em deadlock?

Quais processos estão impedidos de rodar?



Detecção de deadlocks com múltiplos recursos de cada tipo

- Quando existem várias instâncias de algum recurso, é necessário um algoritmo baseado em matrizes para detectar deadlocks.
- **n** = quantidade de processos (P1 a Pn);
- **m** = quantidade de classes (tipos) diferentes de recursos;
- **E** = vetor de recursos existentes;
 - **E_j** = quantidade de recursos de classe j ($1 \leq j \leq m$);
- **A** = vetor de recursos disponíveis;
 - **A_j** = quantidade de recursos de classe j disponíveis ($1 \leq j \leq m$);
- **C** = matriz de alocação corrente
 - **C_{ij}** = quantidade de instâncias de recurso j alocada ao processo i;
- **R** = matriz de requisições
 - **R_{ij}** = quantidade instâncias recurso j requisitada pelo processo i;



Detecção de deadlocks com múltiplos recursos de cada tipo

- Estruturas de dados necessárias ao algoritmo de detecção de deadlock

Recursos existentes
(E₁, E₂, E₃, ..., E_m)

Recursos disponíveis
(A₁, A₂, A₃, ..., A_m)

Matriz de alocação atual

Matriz de requisições

C ₁₁	C ₁₂	C ₁₃	...	C _{1m}
C ₂₁	C ₂₂	C ₂₃	...	C _{2m}
⋮	⋮	⋮		⋮
C _{n1}	C _{n2}	C _{n3}	...	C _{nm}

R ₁₁	R ₁₂	R ₁₃	...	R _{1m}
R ₂₁	R ₂₂	R ₂₃	...	R _{2m}
⋮	⋮	⋮		⋮
R _{n1}	R _{n2}	R _{n3}	...	R _{nm}

Linha n é a alocação atual para o processo n

Linha 2 informa qual é a necessidade do processo 2



Detecção de deadlocks com múltiplos recursos de cada tipo

1. Monte o vetor de recursos existentes **E**;
2. Monte a matriz de alocação corrente **C**;
3. Monte a matriz de requisições **R**;
4. Encontre o vetor de recursos disponíveis **A**;
5. Percorra as linhas da matriz **R** verificando se algum dos processos poderá ser executado com os recursos disponíveis **A**;
 - **Se sim**, então
 - Simule a entrega dos recursos para o processo;
 - Atualize **C**, **R** e **A**;
 - Simule a conclusão do processo e a liberação dos seus recursos alocados;
 - Atualize o vetor **A**;
 - Volte para o passo 5;
 - **Se não**, então existe deadlock entre os processos que não podem ser executados.



Detecção de deadlocks com múltiplos recursos de cada tipo

Exemplo:

- **Recursos existentes:**
 - 4 unidades de fita;
 - 2 plotters;
 - 3 impressoras;
 - 1 unidade de CD;
- **Situação atual:**
 - P1 possui 1 impressora;
 - P2 possui 2 fitas e 1 CD;
 - P3 possui 1 plotter e 2 impressoras;
 - P1 necessita de 2 fitas e 1CD;
 - P2 necessita de 1 fita e 1 impressora;
 - P3 necessita de 2 fitas e 1 plotter;
- **Existe deadlock? Se existir, entre quais processos?**



Detecção de deadlocks com múltiplos recursos de cada tipo

Exercício: Considere um sistema com 4 tipos de recursos (A,B,C,D), com (1,5,2,0) recursos disponíveis, e 5 processos em execução. Dada a tabela de alocação de recursos abaixo, verifique se existe deadlock e se existir diga quais processos estão em deadlock.

Processo	Alocados				Máximo			
	A	B	C	D	A	B	C	D
0	0	0	1	2	0	0	1	2
1	1	0	0	0	1	7	5	0
2	1	3	5	4	2	3	5	6
3	0	6	3	2	2	6	5	2
4	0	1	1	4	0	6	5	6



Recuperação de situações de deadlock

- **O que fazer caso tenhamos detectado um deadlock?**
- **Recuperação por meio de preempção:**
 - Consiste em tomar provisoriamente um recurso de seu proprietário atual para dá-lo a outro processo.
 - Pode ser utilizado somente para recursos preemptíveis.
- **Recuperação por meio de reversão de estados:**
 - Os processos são verificados periodicamente (checkpointed) e seu estado guardado em um arquivo de log (checkpoint file).
 - Quando um deadlock é detectado, torna-se fácil ver quais recursos são necessários.
 - Um dos processos será revertido ao instante anterior que alocou o recurso e cederá ao outro. Este continuará sua execução e outro ficará bloqueado até que o primeiro libere o recurso.



Recuperação de situações de deadlock

- **Recuperação por meio da eliminação de processos:**
 - A maneira mais grosseira e mais simples de eliminar um deadlock é matando um ou mais processos até o ciclo ser quebrado.
 - O processo a ser morto deverá ser cuidadosamente escolhido.
 - Em que ordem abortar?
 - Prioridade dos processos
 - Quanto tempo de CPU tem um processo
 - Quanto falta para terminar
 - Recursos que o processo usa
 - Recursos que o processo precisa
 - O processo é interativo ou batch?



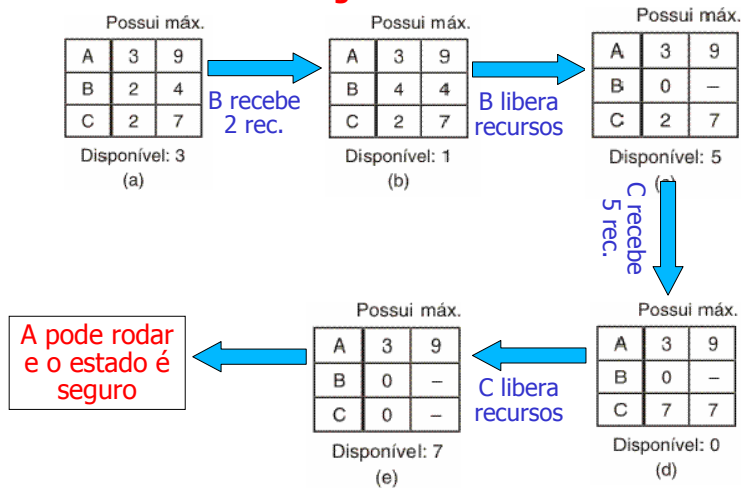
Evitando deadlocks

- Os algoritmos principais para evitar deadlocks são baseados no conceito de **estados seguros**.
- Um estado é considerado seguro se ele não está em situação de deadlock, e assim o sistema pode garantir que todos os processos terminarão.
- A partir de um estado inseguro, o sistema não garante que os processos terminarão.
- É importante enfatizar que um estado inseguro não é uma situação de deadlock, mas indica que pode ocorrer uma situação de deadlock.

Evitando deadlocks

Estado seguro e inseguro

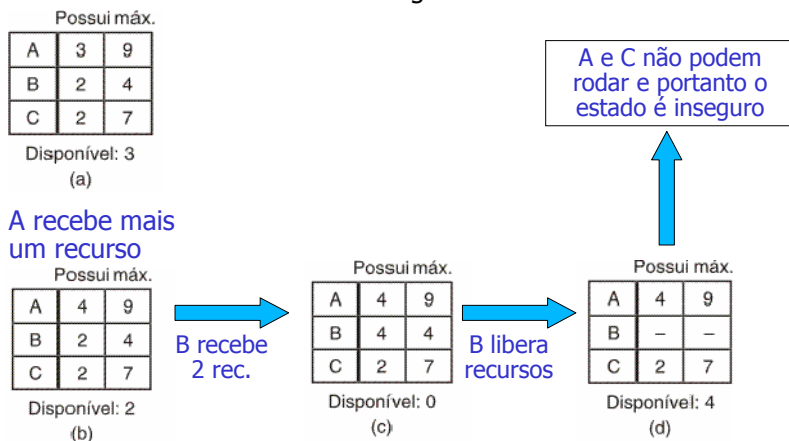
O estado abaixo é seguro?



Evitando deadlocks

Estado seguro e inseguro

O estado abaixo é seguro. Se **A** receber mais um recurso, o sistema continua em um estado seguro?





Evitando deadlocks

Algoritmo do banqueiro

- **Modelagem:** um banqueiro de uma pequena cidade pode negociar com um grupo de clientes para os quais ele libera linhas de crédito.
- O algoritmo do banqueiro verifica se a liberação de uma requisição é capaz de levar a um estado inseguro.
 - Em caso positivo, a requisição será negada.
 - Se a liberação de uma requisição levar a um estado seguro, ela será atendida.



Algoritmo do banqueiro para um único tipo de recurso

Situação atual de um sistema:

Possui máx.

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

Disponível: 2

- O processo **C** solicita mais um recurso ao SO. O SO deve conceder?
- O processo **B** solicita mais um recurso ao SO. O SO deve conceder?



Algoritmo do banqueiro para vários tipos de recurso

	Processo	Unidades de fita	Plotters	Scanners	Unidades de CD-ROM
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	Scanners	Unidades de CD-ROM
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)

- O processo **B** solicita mais uma impressora ao SO. O SO deve conceder?
- Após o SO conceder a impressora ao processo **B**, o processo **E** solicita mais um impressora. O SO deve conceder?



Prevenção de deadlocks

- Atacando a condição de exclusão mútua;
- Atacando a condição de posse e espera;
- Atacando a condição de não-preempção;
- Atacando a condição de espera circular.



Prevenção de deadlock

Atacando a condição de exclusão mútua

- Alguns dispositivos (como uma impressora) podem fazer uso de spool;
 - O daemon de impressão é o único que usa o recurso impressora;
 - Deadlock envolvendo a impressora é eliminado;
- Nem todos os dispositivos podem fazer uso de spool;
- **Princípio:**
 - 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.



Prevenção de deadlock

Atacando a condição de posse e espera


- **Exigir que todos os processos requisitem os recursos antes de iniciarem;**
 - Um processo nunca tem que esperar por aquilo que precisa;
- **Problemas**
 - Os processos podem não saber no início da execução quantos e quais recursos vão precisar;
 - Os processos retêm recursos que outros processos poderiam estar usando;
- **Variação:**
 - O processo deve desistir de todos os recursos que estão na sua posse, para então requisitar todos os que são imediatamente necessários.



Prevenção de deadlock

Atacando a condição de não preempção

- Se um processo que está de posse de alguns recursos pede outro recurso que não pode ser imediatamente disponibilizado, então todos os recursos que ele atualmente possui são liberados;
- Recursos que sofreram preempção são acrescentados à lista de recursos pelos quais o processo está esperando;
- O processo será reiniciado somente quando ele puder obter novamente seus antigos recursos, assim como os novos solicitados;
- **Problema:**
 - Considere um processo de posse de uma impressora, como liberar a impressora no meio de uma impressão?



Prevenção de deadlock

Atacando a condição de espera circular

- Uma regra que determine que o processo tenha permissão de possuir somente um recurso de cada vez, e se ele necessitar de outro recurso, deverá eliminar o primeiro é um meio simples para eliminar deadlocks;
 - Para um processo que necessita copiar um arquivo grande de uma fita para impressora, essa restrição é inaceitável;
- Outra maneira de evitar a condição de espera circular é fornecer uma numeração global de todos os recursos. Processos podem requisitar recursos sempre que necessário, mas todas as solicitações devem ser feitas em ordem numérica.



Prevenção de deadlock

Resumo

Condição	Abordagem contra deadlocks
Exclusão mútua	Usar spool em tudo
Posse-e-espera	Requisitar inicialmente todos os recursos necessários
Não preempção	Retomar os recursos alocados
Espera circular	Ordenar numericamente os recursos



Starvation

- **Starvation** ou **condição de inanição** ocorre quando um processo nunca obtém um determinado recurso mesmo sem haver deadlock, pois toda vez que ele tenta tomar posse do recurso outro processo já tomou posse anteriormente;
- O processo pode esperar durante muito tempo pelo recurso e o usuário acha que ele está “travado”;
- Este problema pode ser resolvido se o SO priorizar o processo que está aguardando o recurso a muito tempo.



Exercícios

- Exercícios do livro Sistemas Operacionais Modernos (Tanenbaum) 2a Edição: 1, 2, 3, 7, 8, 13, 14, 15, 17, 20, 27 e 28.

