

Aula 11: Deadlocks

Professor(a): Virgínia Fernandes Mota
<http://www.dcc.ufmg.br/~virginiaferm>

OCS (TEORIA) - SETOR DE INFORMÁTICA



- Dispositivos e recursos são compartilhados a todo momento: impressora, disco, arquivos, etc.;
- Deadlock: processos ficam parados sem possibilidade de poderem continuar seu processamento;

Deadlocks



- Recursos: objetos acessados, os quais podem ser tanto hardware quanto uma informação
- Preemptivos: podem ser retirados do processo sem prejuízos; (Memória, CPU)
- Não-preemptivos: não podem ser retirados do processo, pois causam prejuízos; (unidades externas)
- Deadlocks ocorrem com recursos não-preemptivos;

- Operações sobre recursos/dispositivos:
 - Requisição do recurso;
 - Utilização do recurso;
 - 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, ou;
 - Processo que requisitou o recurso falha, e depois de um certo tempo tenta novamente requisitar o recurso;

- Aquisição do recurso: Para alguns tipos de recursos, os processos dos usuários gerenciam o uso dos recursos, através, por exemplo, de semáforos.
 - Exemplo: acesso a registros em um sistema de banco de dados
- Se vários processos tentam acessar os mesmos recursos, podem ocorrer situações onde a ordem de solicitação dos recursos pode conduzir ao um deadlock ou não

- Definição formal: "Um conjunto de processos estará em situação de deadlock se todo processo pertencente ao conjunto estiver esperando por um evento que somente um outro processo desse mesmo conjunto poderá fazer acontecer."

- Quatro condições para que ocorra um deadlock:
 - **Exclusão mútua**: cada recurso pode estar somente em uma de duas situações: ou associado a um único processo ou disponível;
 - **Posse e espera (hold and wait)**: processos que já possuem algum recurso podem requer outros recursos;
 - **Não-preempção**: recursos já alocados não podem ser retirados do processo que os alocou; somente o processo que alocou os recursos pode liberá-los;
 - **Espera Circular**: um processo pode esperar por recursos alocados a outro processo, que por sua vez espera por recursos alocados a outro processo, e assim por diante, até que o último espera por recursos alocados ao primeiro;
- Todas as condições devem ocorrer para que ocorra um deadlock!

Deadlocks

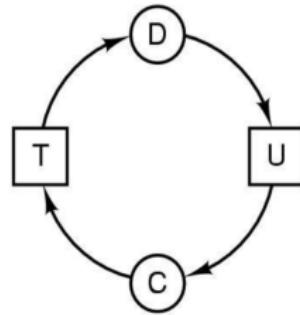
- Geralmente, deadlocks são representados por grafos a fim de facilitar sua detecção, prevenção e recuperação → A ocorrência de ciclos pode levar a um deadlock;
- Grafos de alocação de recursos



(a)



(b)



(c)

- Recurso R alocado ao Processo A**
- Processo B requisita Recurso S**
- Deadlock**

Como ocorre um deadlock

A

Requisita R
Requisita S
Libera R
Libera S

(a)

B

Requisita S
Requisita T
Libera S
Libera T

(b)

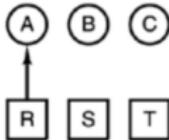
C

Requisita T
Requisita R
Libera T
Libera R

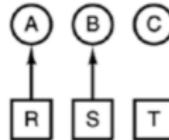
(c)

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

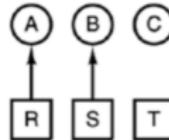
(d)



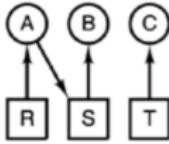
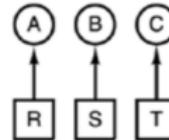
(e)



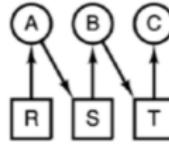
(f)



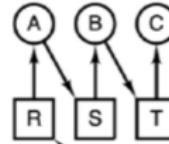
(g)



(h)



(i)

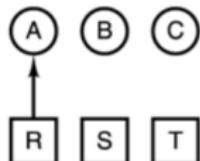


(j)

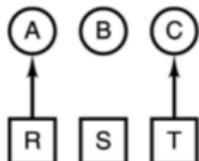
Evitando um deadlock

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

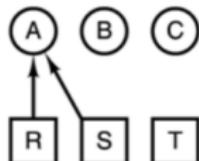
(k)



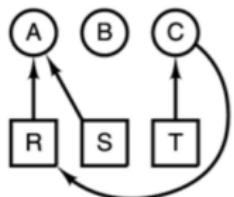
(l)



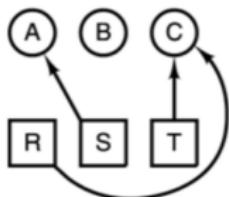
(m)



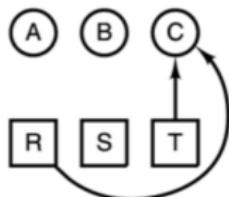
(n)



(o)



(p)



(q)

- Quatro estratégias para tratar deadlocks:
 - ① Ignorar o problema;
 - ② Detectar e recuperar o problema;
 - ③ Evitar dinamicamente o problema - alocação cuidadosa de recursos;
 - ④ Prevenir o problema por meio da não satisfação de uma das quatro condições citadas anteriormente;

Deadlocks - (1) Ignorar o problema

- Frequência do problema;
- Alto custo - estabelecimento de condições para o uso de recursos;
- UNIX e WINDOWS;
- Algoritmo do AVESTRUZ;

- Processos estão com todos os recursos alocados;
- Procedimento: Permite que os deadlocks ocorram, tenta detectar as causas e solucionar a situação;
- 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:
 - Construção de um grafo;
 - Se houver ciclos, existem potenciais deadlocks;

Situação:

PA usa R e precisa de S;
PB precisa de T;
PC precisa de S;
PD usa U e precisa de S e T;
PE usa T e precisa de V;
PF usa W e precisa de S;
PG usa V e precisa de U;

- Há possibilidade de deadlock?

- Detecção com vários recursos de cada tipo:
 - Classes diferentes de recursos - 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 → C_{ij} : número de instâncias do recurso j entregues ao processo i;
 - R: matriz de requisições → R_{ij} : número de instâncias do recurso j que o processo i precisa;

Deadlocks - (2) Detectar e Recuperar o problema

- Detecção com vários recursos de cada tipo:

4 unidades de fita;

2 *plotter*;

3 *scanner*;

1 unidade de CD-ROM

Três processos:

P₁ usa um *scanner*;

P₂ usa duas unidades de fita e uma de CD-ROM;

P₃ usa um *plotter* e dois *scanners*;

Cada processo precisa de outros recursos (R);

Recursos existentes

$$E = (4 \ 2 \ 3 \ 1)$$

UF P S UCD

Recursos disponíveis

$$A = (2 \ 1 \ 0 \ 0)$$

UF P S UCD

Matriz de alocação

$$C = \begin{array}{c|cccc} & \text{UF} & \text{P} & \text{S} & \text{UCD} \\ \hline \text{P}_1 & 0 & 0 & 1 & 0 \\ \text{P}_2 & 2 & 0 & 0 & 1 \\ \text{P}_3 & 0 & 1 & 2 & 0 \end{array}$$

Recursos

Matriz de requisições

$$R = \begin{array}{c|cccc} & \text{UF} & \text{P} & \text{S} & \text{UCD} \\ \hline \text{P}_1 & 2 & 0 & 0 & 1 \\ \text{P}_2 & 1 & 0 & 1 & 0 \\ \text{P}_3 & 2 & 1 & 0 & 0 \end{array}$$

Deadlocks - (2) Detectar e Recuperar o problema

- Detecção com vários recursos de cada tipo:

4 unidades de fita;

2 *plotter*;

3 *scanners*;

1 unidade de CD-ROM

Requisições:

P_1 requisita duas unidades de fita e um CD-ROM;

P_2 requisita uma unidade de fita e um *scanner*;

P_3 requisita duas unidades de fita e um *plotter*;

Recursos existentes

$$E = (4 \ 2 \ 3 \ 1)$$

Recursos disponíveis

$$A = (2 \ 1 \ 0 \ 0) \quad P_3 \text{ pode executar}$$

$$A = (0 \ 0 \ 0 \ 0)$$

Matriz de alocação

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ \mathbf{2} & \mathbf{2} & \mathbf{2} & \mathbf{0} \end{bmatrix} \quad \begin{array}{l} \xleftarrow{} P_1 \\ \xleftarrow{} P_2 \\ \xleftarrow{} P_3 \end{array}$$

Matriz de requisições

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{array}{l} \xleftarrow{} P_1 \\ \xleftarrow{} P_2 \\ \xleftarrow{} P_3 \end{array}$$

Deadlocks - (2) Detectar e Recuperar o problema

- Detecção com vários recursos de cada tipo:

4 unidades de fita;

2 *plotter*;

3 *scanners*;

1 unidade de CD-ROM

Requisições:

P_1 requisita duas unidades de fita e um CD-ROM;

P_2 requisita uma unidade de fita e um *scanner*;

P_3 requisita duas unidades de fita e um *plotter*;

Recursos existentes

$$E = (4 \ 2 \ 3 \ 1)$$

Recursos disponíveis

$$A = (2 \ 1 \ 0 \ 0) \quad P_3 \text{ pode executar}$$

$$A = (2 \ 2 \ 2 \ 0)$$

Matriz de alocação

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{array}{l} \xleftarrow{} P_1 \\ \xleftarrow{} P_2 \\ \xleftarrow{} P_3 \end{array}$$

Matriz de requisições

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{array}{l} \xleftarrow{} P_1 \\ \xleftarrow{} P_2 \\ \xleftarrow{} P_3 \end{array}$$

Deadlocks - (2) Detectar e Recuperar o problema

- Detecção com vários recursos de cada tipo:

4 unidades de fita;

2 *plotter*;

3 *scanners*;

1 unidade de CD-ROM

Requisições:

P_1 requisita duas unidades de fita e um CD-ROM;

P_2 requisita uma unidade de fita e um *scanner*;

P_3 requisita duas unidades de fita e um *plotter*;

Recursos existentes

$$E = (4 \ 2 \ 3 \ 1)$$

Recursos disponíveis

$$A = (2 \ 1 \ 0 \ 0)$$

$A = (2 \ 2 \ 2 \ 0)$ **P_2 pode executar**

$$A = (1 \ 2 \ 1 \ 0)$$

Matriz de alocação

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 3 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \leftarrow P_1$$
$$\leftarrow P_2$$
$$\leftarrow P_3$$

Matriz de requisições

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \leftarrow P_1$$
$$\leftarrow P_2$$
$$\leftarrow P_3$$

Deadlocks - (2) Detectar e Recuperar o problema

- Detecção com vários recursos de cada tipo:

4 unidades de fita;

2 *plotter*;

3 *scanners*;

1 unidade de CD-ROM

Requisições:

P_1 requisita duas unidades de fita e um CD-ROM;

P_2 requisita uma unidade de fita e um *scanner*;

P_3 requisita duas unidades de fita e um *plotter*;

Recursos existentes

$$E = (4 \ 2 \ 3 \ 1)$$

Recursos disponíveis

$$A = (2 \ 1 \ 0 \ 0)$$

$A = (2 \ 2 \ 2 \ 0)$ **P_2 pode executar**

$$A = (4 \ 2 \ 2 \ 1)$$

Matriz de alocação

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{array}{l} \xleftarrow{} P_1 \\ \xleftarrow{} P_2 \\ \xleftarrow{} P_3 \end{array}$$

Matriz de requisições

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{array}{l} \xleftarrow{} P_1 \\ \xleftarrow{} P_2 \\ \xleftarrow{} P_3 \end{array}$$

Deadlocks - (2) Detectar e Recuperar o problema

- Detecção com vários recursos de cada tipo:

4 unidades de fita;

2 *plotter*;

3 *scanners*;

1 unidade de CD-ROM

Requisições:

P_1 requisita duas unidades de fita e um CD-ROM;

P_2 requisita uma unidade de fita e um *scanner*;

P_3 requisita duas unidades de fita e um *plotter*;

Recursos existentes

$$E = (4 \ 2 \ 3 \ 1)$$

Recursos disponíveis

$$A = (2 \ 1 \ 0 \ 0)$$

$$A = (2 \ 2 \ 2 \ 0)$$

$A = (2 \ 2 \ 2 \ 0)$ **P_1 pode executar**

Matriz de alocação

$$C = \begin{bmatrix} 2 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xleftarrow{\quad} P_1$$
$$\xleftarrow{\quad} P_2$$
$$\xleftarrow{\quad} P_3$$

Matriz de requisições

$$R = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xleftarrow{\quad} \mathbf{P}_1$$
$$\xleftarrow{\quad} P_2$$
$$\xleftarrow{\quad} P_3$$

- Detecção com vários recursos de cada tipo:

Ao final da execução, temos:

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

Recursos existentes
 $E = (4 \ 2 \ 3 \ 1)$

Recursos disponíveis
 $A = (4 \ 2 \ 3 \ 1)$

Matriz de alocação

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{array}{l} \xleftarrow{} P_1 \\ \xleftarrow{} P_2 \\ \xleftarrow{} P_3 \end{array}$$

Matriz de requisições

$$R = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{array}{l} \xleftarrow{} P_1 \\ \xleftarrow{} P_2 \\ \xleftarrow{} P_3 \end{array}$$

Deadlocks - (2) Detectar e Recuperar o problema

- Detecção com vários recursos de cada tipo:

4 unidades de fita;

2 *plotters*;

3 *scanners*;

1 unidade de CD-ROM

Requisições:

P_1 requisita duas unidades de fita e um CD-ROM;

P_2 requisita uma unidade de fita e um *scanner*;

P_3 requisita uma unidade de fita e um *plotter*;

Recursos existentes

$$E = (4 \ 2 \ 3 \ 1)$$

Recursos disponíveis

$$A = (0 \ 1 \ 2 \ 0)$$

Matriz de alocação

$$C = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \quad \begin{array}{l} \xleftarrow{} P_1 \\ \xleftarrow{} P_2 \\ \xleftarrow{} P_3 \end{array}$$

Matriz de requisições

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 2 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \begin{array}{l} \xleftarrow{} P_1 \\ \xleftarrow{} P_2 \\ \xleftarrow{} P_3 \end{array}$$

Nessa situação, nenhum processo pode ser atendido!

DEADLOCK

- Detecção com vários recursos de cada tipo:
 - Para esse algoritmo, o sistema, geralmente, procura periodicamente por deadlocks;
 - CUIDADO: Evitar ociosidade da CPU → quando se tem muitos processos em situação de deadlock, poucos processos estão em execução;

- Recuperação de Deadlocks:

- Por meio de preempção: possibilidade de retirar temporariamente um recurso de seu atual dono (processo) e entregá-lo a outro processo;
- Por meio de retrocesso (rollback): recursos alocados a um processo são armazenados em arquivos de verificação; quando ocorre um deadlock, os processos voltam ao estado no qual estavam antes do deadlock → solução cara;

- Recuperação de Deadlocks:

- 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;
 - Ex1.:compilação - sem problemas;
 - Ex2.: atualização de um base de dados - problemas;

- Alocação individual de recursos → à medida que o processo necessita;
- Soluções também utilizam matrizes;
- Escalonamento cuidadoso → alto custo;
 - Conhecimento prévio dos recursos que serão utilizados;
- Algoritmos:
 - Banqueiro para um único tipo de recurso;
 - Banqueiro para vários tipos de recursos;
- Definição de Estados Seguros e Inseguros;

Deadlocks - (3) Evitar dinamicamente o problema

- Estados seguros: não provocam deadlocks e há uma maneira de atender a todas as requisições pendentes finalizando normalmente todos os processos;
 - A partir de um estado seguro, existe a garantia de que os processos terminarão;
- Estados inseguros: podem provocar deadlocks, mas não necessariamente provocam;
 - A partir de um estudo inseguro, não é possível garantir que os processos terminarão corretamente;

Estados seguros e inseguros

Possui máx.		
A	3	9
B	2	4
C	2	7

Disponível: 3
(a)

Possui máx.		
A	3	9
B	4	4
C	2	7

Disponível: 1
(b)

Possui máx.		
A	3	9
B	0	-
C	2	7

Disponível: 5
(c)

Possui máx.		
A	3	9
B	0	-
C	7	7

Disponível: 0
(d)

Possui máx.		
A	3	9
B	0	-
C	0	-

Disponível: 7
(e)

Demonstração de que o estado em (a) é seguro

Estados seguros e inseguros

Possui máx.
A 3 9
B 2 4
C 2 7

Disponível: 3
(a)

Possui máx.
A 4 9
B 2 4
C 2 7

Disponível: 2
(b)

Possui máx.
A 4 9
B 4 4
C 2 7

Disponível: 0
(c)

Possui máx.
A 4 9
B - -
C 2 7

Disponível: 4
(d)

Demonstração de que o estado em (b) é inseguro

- Algoritmos do Banqueiro:
 - Idealizado por Dijkstra (1965);
 - Considera cada requisição no momento em que ela ocorre, verificando se essa requisição leva a um estado seguro; Se sim, a requisição é atendida, se não o atendimento é adiado para um outro momento;
 - Premissas adotadas por um banqueiro (SO) para garantir ou não crédito (recursos) para seus clientes (processos);
 - Nem todos os clientes (processos) precisam de toda a linha de crédito (recursos) disponível para eles;

Deadlocks - (3) Evitar dinamicamente o problema

- Algoritmo do Banqueiro para um único tipo de recurso:

Possui

Máximo de linha de crédito = 22

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

Livre: 10

Seguro

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

Livre: 2

Seguro

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

Livre: 1

Inseguro

- Solicitações de crédito são realizadas de tempo em tempo;
- * C é atendido e libera 4 créditos, que podem ser usados por B ou D;

Deadlocks - (3) Evitar dinamicamente o problema

- Algoritmo do Banqueiro para um único tipo de recurso:

Possui

Máximo de linha de crédito = 22

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

Livre: 10

Seguro

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

Livre: 2

Inseguro

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

Livre: 1

- Solicitações de crédito são realizadas de tempo em tempo;
- * B é atendido. Em seguida os outros fazem solicitação, ninguém poderia ser atendido;

Deadlocks - (3) Evitar dinamicamente o problema

- Algoritmo do Banqueiro para vários tipos de recursos: Mesma idéia, mas duas matrizes são utilizadas;

	Processos	Unidade de Fita	Plotters	Impressoras	Unidade 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	

$C = \text{Recursos Alocados}$

Recursos $\rightarrow E = (6 \ 3 \ 4 \ 2)$;
Alocados $\rightarrow P = (5 \ 3 \ 2 \ 2)$;
Disponíveis $\rightarrow A = (1 \ 0 \ 2 \ 0)$;

	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	

$R = \text{Recursos ainda necessários}$

Deadlocks - (3) Evitar dinamicamente o problema

- Algoritmo do Banqueiro para vários tipos de recursos:

	Processos	Unidade de Fita	Plotters	Impressoras	Unidade de CD-ROM
A	3	0	1	1	
B	0	1	1	0	
C	1	1	1	0	
D	1	1	0	1	
E	0	0	0	0	

C = Recursos Alocados

- Podem ser atendidos: D, A ou E, C;

Alocados → P = (5 3 3 2);
Disponíveis → A = (1 0 1 0);

	1	1	0	0
B	0	1	0	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

R = Recursos ainda
necessários

Deadlocks - (3) Evitar dinamicamente o problema

- Algoritmo do Banqueiro para vários tipos de recursos:

	Processos	Unidade de Fita	Plotters	Impressoras	Unidade de CD-ROM
A	3	0	1	1	
B	0	1	1	0	
C	1	1	1	0	
D	1	1	0	1	
E	0	0	1	0	

C = Recursos Alocados

Alocados → P = (5 3 **4** 2);
Disponíveis → A = (1 0 **0** 0);

	1	1	0	0
A	1	1	0	2
B	0	1	0	0
C	3	1	0	0
D	0	0	1	0
E	2	1	0	0

R = Recursos ainda necessários

- *Deadlock → atender o processo E; Solução:* Adiar a requisição de E por alguns instantes;

- 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;
 - Vantagem
 - Na teoria o algoritmo é ótimo;

- Atacar uma das quatro condições:

Condição	Abordagem
Exclusão Mútua	Alocar todos os recursos usando um <i>spool</i>
Posse e Espera	Requisitar todos os recursos inicialmente para execução – difícil saber; sobrecarga do sistema
Não-preempção	Retirar recursos dos processos – pode ser ruim dependendo do tipo de recurso; praticamente não implementável
Espera Circular	Ordenar numericamente os recursos, e realizar solicitações em ordem numérica Permitir que o processo utilize apenas um recurso por vez

- Deadlocks podem ocorrer sem o envolvimento de recursos, por exemplo, se semáforos forem implementados erroneamente;

```
..          ..
down (&empty); down (&mutex);
down (&mutex); down (&empty);
..          ..
```

- Inanição(Starvation)

- Todos os processos devem conseguir utilizar os recursos que precisam, sem ter que esperar indefinidamente;
- Alocação usando FIFO;

Deadlocks

Potencial deadlock

```
semaphore resource_1;
semaphore resource_2;

void Process_A (void){
    down(&resource_1);
    down(&resource_2);
    use_both_resources();
    up(&resource_2);
    up(&resource_1);
}

void Process_B (void){
    down(&resource_2);
    down(&resource_1);
    use_both_resources();
    up(&resource_1);
    up(&resource_2);}
```

Livre de deadlock

```
semaphore resource_1;
semaphore resource_2;

void Process_A (void){
    down(&resource_1);
    down(&resource_2);
    use_both_resources();
    up(&resource_2);
    up(&resource_1);
}

void Process_B (void){
    down(&resource_1);
    down(&resource_2);
    use_both_resources();
    up(&resource_1);
    up(&resource_2);}
```

Exercícios e Prova