Departamento de Computação

SISTEMAS OPERACIONAIS 1
21270 A

Departamento de Computação
Prof. Kelen Cristiane Teixeira Vivaldini

Departamento de Computação

Apresentação baseada nos slides
do Prof. Dr. Antônio Carlos Sementille e Prof.
Kalinka C. Branco e nas transparências
fornecidas no site de compra do livro
“Sistemas Operacionais Modernos”

Deadlocks

```
void f() {
        mutex_lock(&lock);
        mutex_lock(&lock);
}
```

Veja o codigo: deadlock.c

Departamento de Computação

Deadlocks

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

Deadlocks

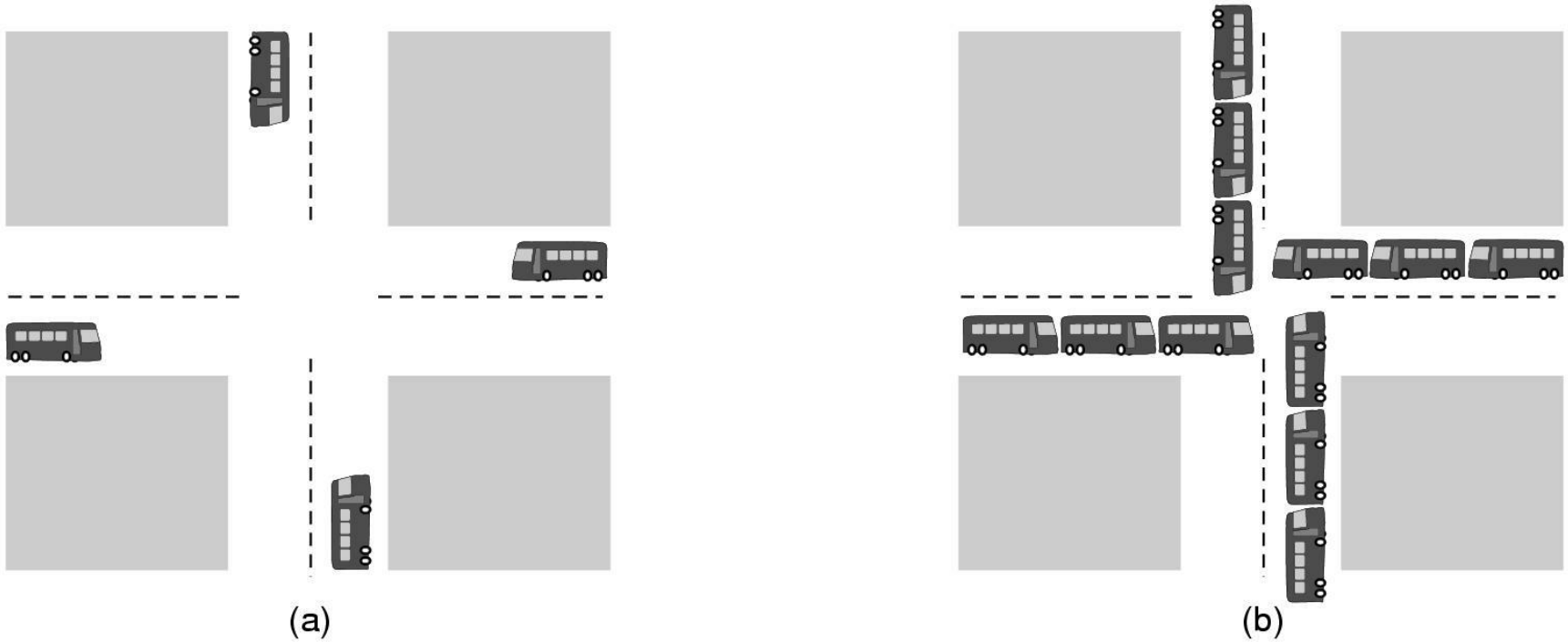
Processo A

Detém recurso **Y**
E espera pelo recurso **X**

Processo B

Detém recurso **X**
E espera pelo recurso **Y**

Deadlocks



Uma situação de *deadlock*

Deadlocks

- Recursos:
  - Preemptivos: podem ser retirados do processo sem prejuízos;
    - Memória;
    - CPU;
  - Não-preemptivos: não podem ser retirados do processo, pois causam prejuízos;
    - CD-ROM;
    - Unidades de fita;
    - *Deadlocks* ocorrem com esse tipo de recurso;

Departamento de Computação

Deadlocks

- Requisição de 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 de Recursos

```
typedef int semaphore;
semaphore resource_1;

void process_A(void) {
        down(&resource_1);
        use_resource_1( );
        up(&resource_1);
}
```

(a)

```
typedef int semaphore;
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);
}
```

(b)

Figura 6.1 O uso de um semáforo para proteger recursos. (a) Um recurso. (b) Dois recursos.

Deadlocks

```
typedef int semaphore;
        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_2);
                up(&resource_1);
        }
```

(a)

```
        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);
        }
```

(b)

Figura 6.2 (a) Código sem impasse. (b) Código com possibilidade de impasse.

Deadlocks

Um *deadlock* pode ser definido formalmente como:

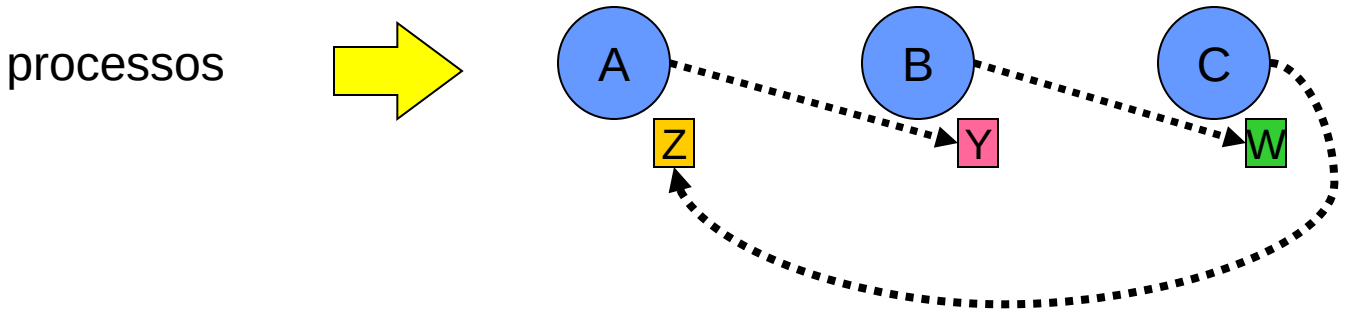
Um conjunto de processos estará em situação de impasse se todo processo pertencente ao conjunto estiver esperando por um evento que somente outro processo desse mesmo conjunto poderá fazer acontecer.

Deadlocks

- Quatro condições devem ocorrer para que um *deadlock* exista:
  - Exclusão mútua: um recurso só pode estar alocado para um processo em um determinado momento;
  - Uso e espera (*hold and wait*): processos que já possuem algum recurso podem requisitar 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;

Deadlocks

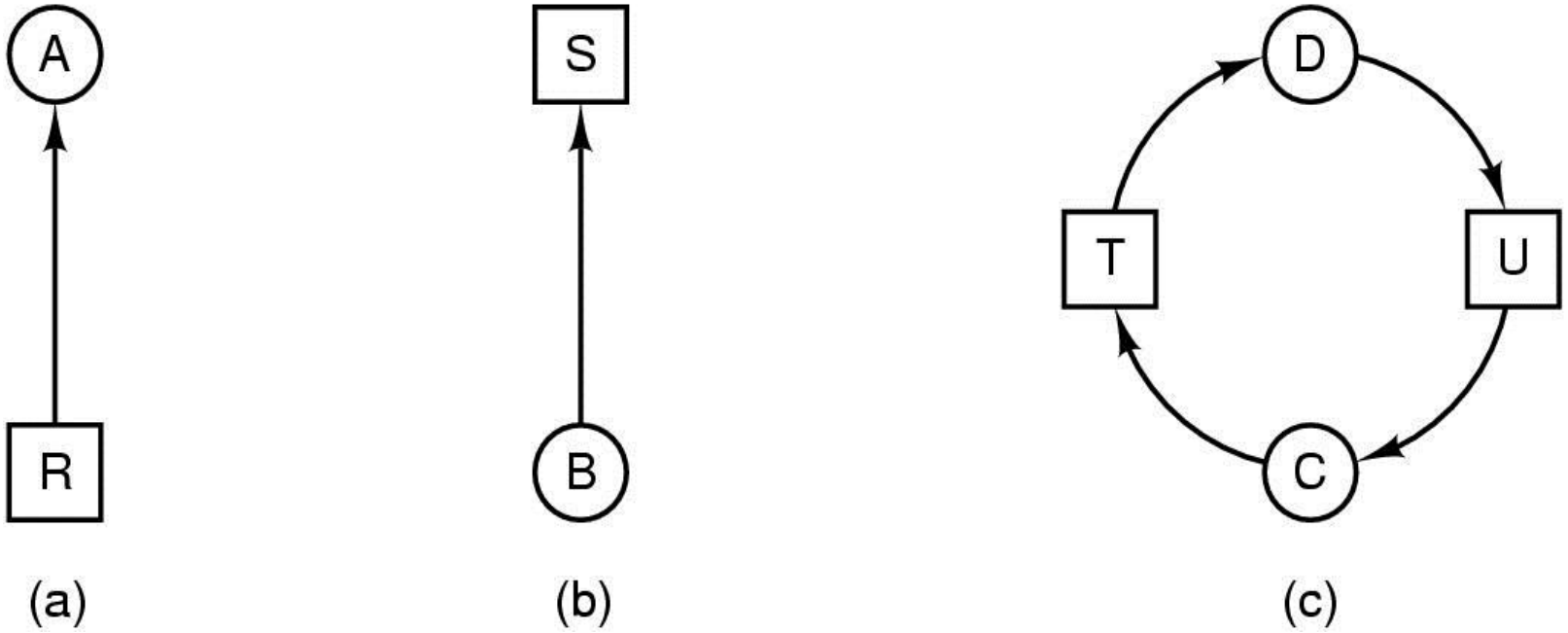
- Espera circular por recursos.
- Exemplo:
  - O processo “A” espera pelo processo “B”, que espera pelo processo “C”, que espera pelo processo “A”.



Deadlocks

- Geralmente, *deadlocks* são representados por grafos a fim de facilitar sua detecção, prevenção e recuperação
  - Ocorrência de ciclos pode levar a um *deadlock*;

Deadlocks
Grafos de alocação de recursos



a) Recurso R alocado ao Processo A
b) Processo B requisita Recurso S
c) Deadlock

Deadlocks
Grafos de alocação de recursos

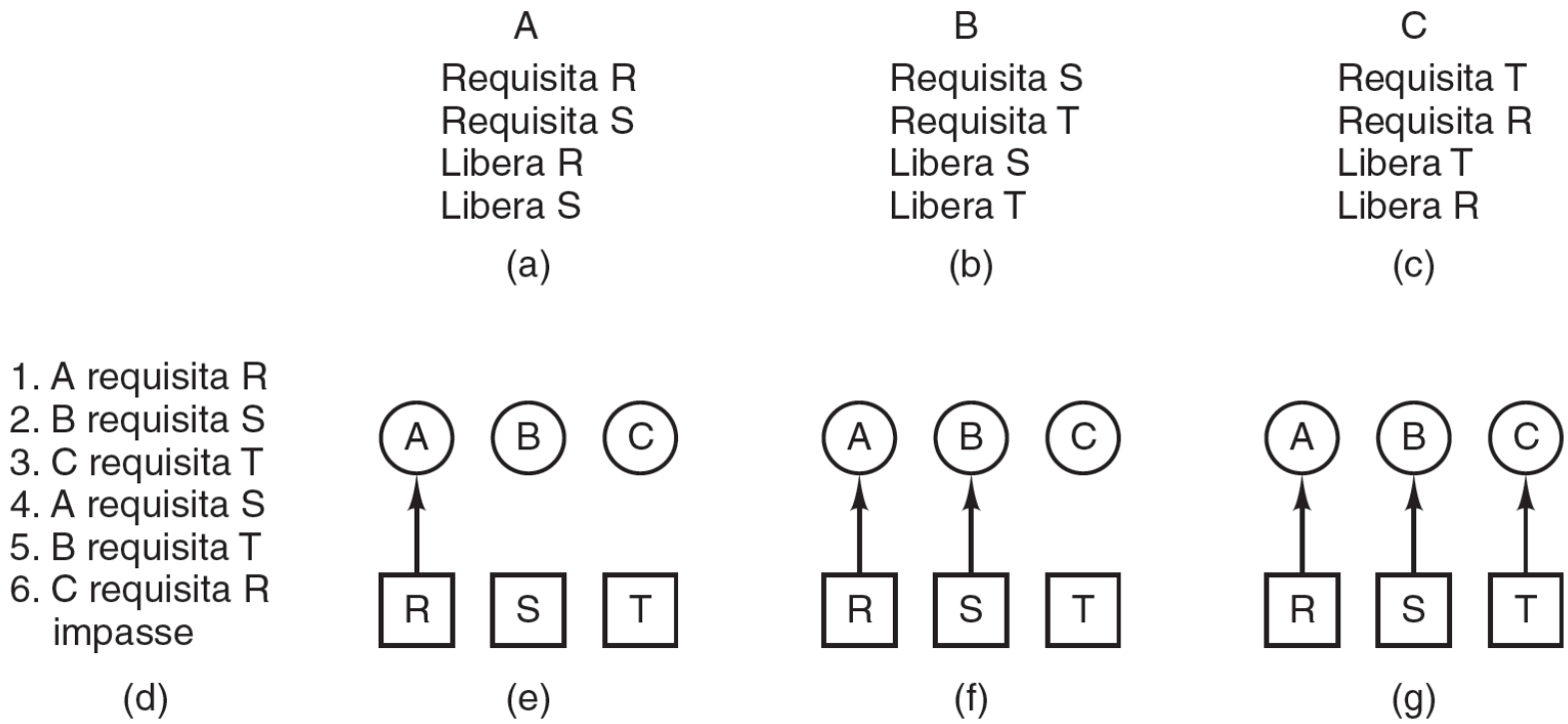


Figura 6.4 Exemplo de como um impasse ocorre e como pode ser evitado.

Deadlocks
Grafos de alocação de recursos

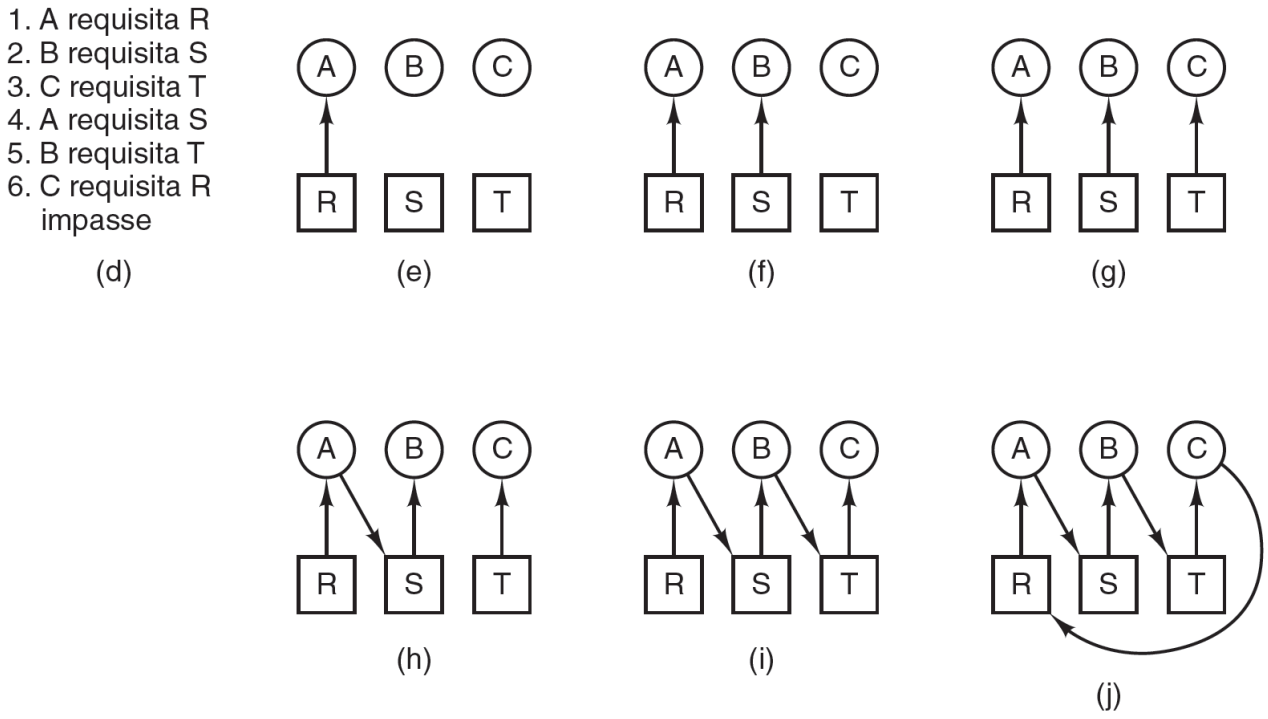


Figura 6.4 Exemplo de como um impasse ocorre e como pode ser evitado.

Deadlocks
Grafos de alocação de recursos

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 impasse

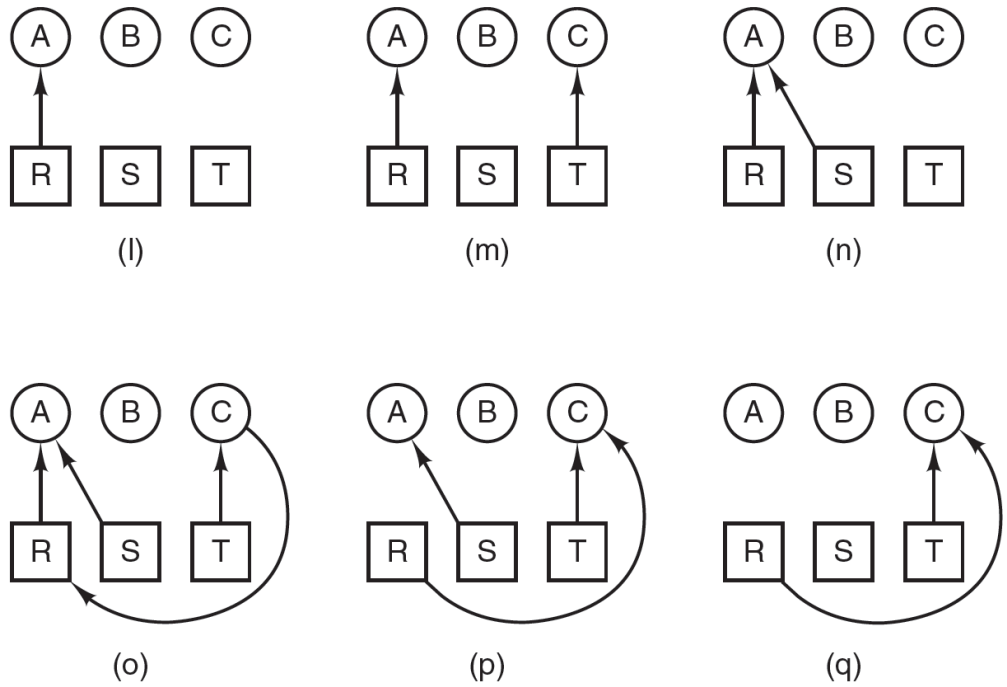


Figura 6.4 Exemplo de como um impasse ocorre e como pode ser evitado.

Deadlocks

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

Tratamento de Deadlocks

- Ignorar o problema.
  - Comparar a freqüência de ocorrência de *deadlocks* com a freqüência de outras falhas do sistema.
    - Falhas de *hardware*, erros de compiladores, erros do Sistema Operacional, etc.
  - Se o esforço em solucionar o problema for muito grande em relação a freqüência com que o *deadlock* ocorre, ele pode ser ignorado.

Deadlocks

- Ignorar o problema:
  - Freqüência do problema;
  - Alto custo – estabelecimento de condições para o uso de recursos;
  - Algoritmo do AVESTRUZ;

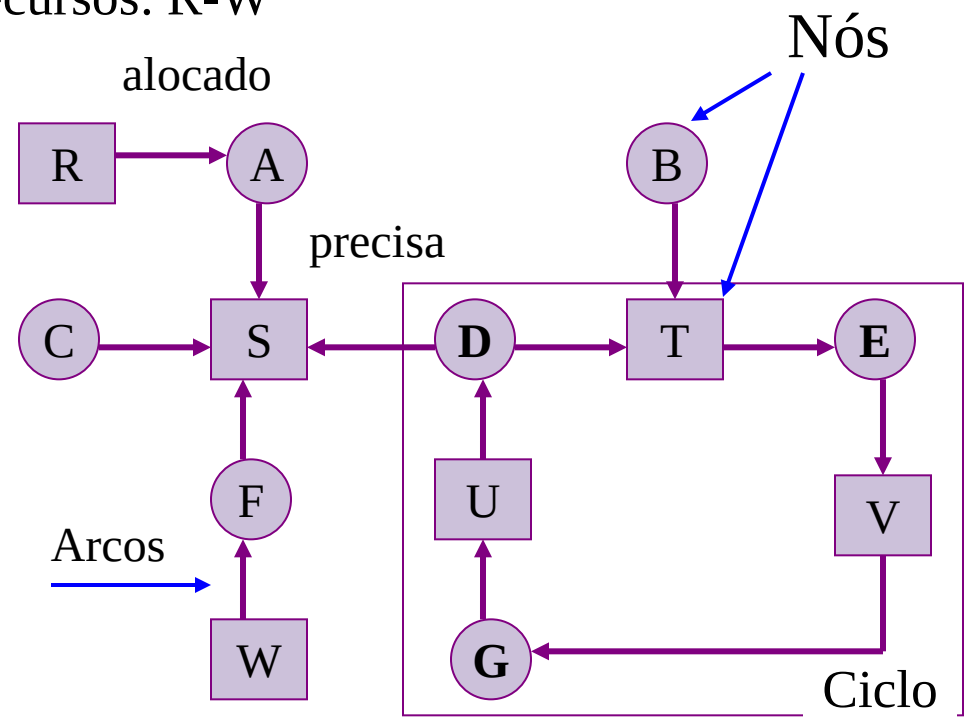
Deadlocks

- **Detectar e Recuperar o problema:**
  - 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;

Deadlocks

- **Detecção com um recurso de cada tipo:**
  - Construção de um grafo;
  - Se houver ciclos, existem potenciais *deadlocks*;

Processos: A-G
Recursos: R-W



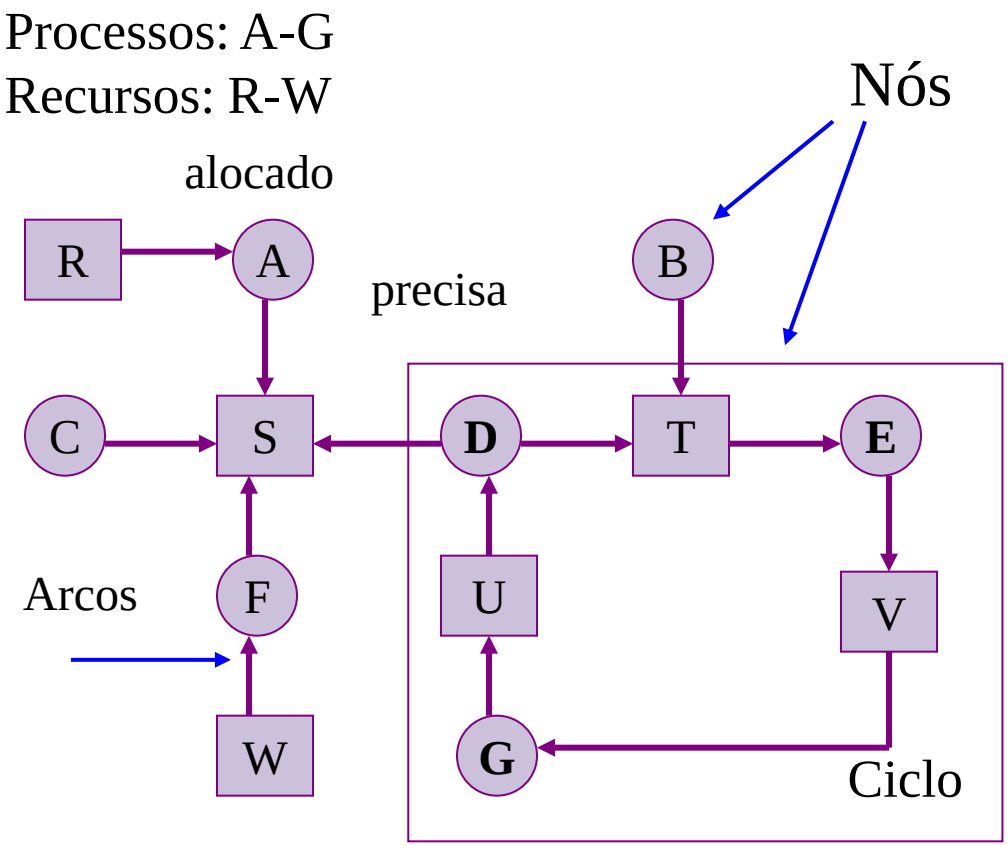
Situação:
PA usa R e requisita S;
PB requisita de T;
PC requisita de S;
PD usa U e requisita de S e T;
PE usa T e requisita de V;
PF usa W e requisita de S;
PG usa V e requisita de U;

Pergunta:
Há possibilidade de *deadlock*?

Deadlocks

- **Detecção com um recurso de cada tipo:**
  - Execução a partir de R->A,B,C,S,D,T,E,F (ciclo para);

1. Início R => L=[R, A], L=[R, A, S] =>S não tem arco de saída (retorna);
2. Início A => L=[A,S] S não tem arco de saída (retorna); 3)
3. Início B => L=[B,T,E,V,G,U,D] = escolher S vamos para um nó sem saída e retornamos em D
4. Caso contrário: L=[B,T,E,V,G,U,D,T] =>ciclo



Deadlocks

- **Detecção com vários recursos de cada tipo:**
  - Classes diferentes de recursos – vetor de recursos existentes (E):
    - Se classe1=unidade de fita e E_1=2, então existem duas unidades de fita;
  - Vetor de recursos disponíveis (A):
    - Se ambas as unidades de fita estiverem alocadas, A_1=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

- **Detecção com vários recursos de cada tipo:**

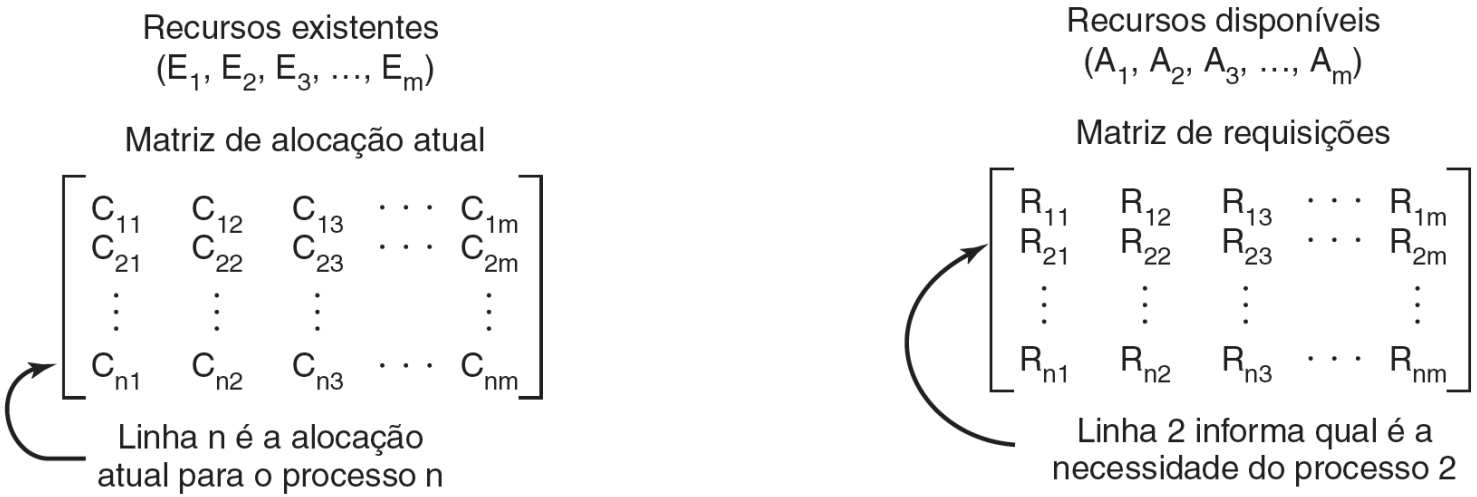


Figura 6.6 As quatro estruturas de dados necessárias ao algoritmo de detecção de impasses.

Deadlocks

Algoritmo de detecção de *deadlock*:

1. Procure um processo desmarcado, *Pi*, para o qual a *i*-ésima linha de *R* seja menor ou igual à correspondente de *A*.
2. Se esse processo for encontrado, adicione a *i*-ésima linha de *C* à correspondente de *A*, marque o processo e volte para o passo 1.
3. Se não existir esse processo, o algoritmo terminará.

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 atual

UF P I UCD

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

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

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

Deadlocks

4 unidades de fita;
2 *plotter*;
3 impressoras;
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 uma impressora;
P_3 requisita duas unidades de fita e um *plotter*;

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

Recursos disponíveis
A = (2 1 0 0) **P_3 pode rodar**
A = (**0 0 0 0**)
UF P I UCD

Matriz de alocação

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

Matriz de requisições

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

Deadlocks

4 unidades de fita;
2 *plotter*;
3 impressoras;
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 uma impressora;
P_3 requisita duas unidades de fita e um *plotter*;

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

Recursos disponíveis
A = (2 1 0 0) **P_3 pode rodar**
A = **(2 2 2 0)**
UF P I UCD

Matriz de alocação

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

Matriz de requisições

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

Deadlocks

4 unidades de fita;
2 *plotter*;
3 impressoras;
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 uma impressora;
P_3 requisita duas unidades de fita e um *plotter*;

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

Recursos disponíveis
A = (2 1 0 0)
A = (2 2 2 0) **P_2 pode rodar**
A = (**1 2 1 0**)

Matriz de alocação

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

Matriz de requisições

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

Deadlocks

4 unidades de fita;
2 *plotter*;
3 impressoras;
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 uma impressora;
P_3 requisita duas unidades de fita e um *plotter*;

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

Recursos disponíveis
A = (2 1 0 0)
A = (2 2 2 0) **P_2 pode rodar**
A = (**4 2 2 1**)

Matriz de alocação

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

Matriz de requisições

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

Deadlocks

4 unidades de fita;
2 *plotter*;
3 impressoras;
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 uma impressora;
P_3 requisita duas unidades de fita e um *plotter*;

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

Recursos disponíveis
A = (2 1 0 0)
A = (2 2 2 0)
A = (2 2 1 0) **P_1 pode rodar**

Matriz de alocação

$$C = \begin{bmatrix} \mathbf{2\ 0\ 1\ 1} \\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0 \end{bmatrix} \begin{matrix} \leftarrow P_1 \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Matriz de requisições

$$R = \begin{bmatrix} 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0 \end{bmatrix} \begin{matrix} \leftarrow \mathbf{P_1} \\ \leftarrow P_2 \\ \leftarrow P_3 \end{matrix}$$

Deadlocks

Ao final da execução, temos:

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

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

Recursos disponíveis
A = (4 2 3 1)

Matriz de alocação

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

Matriz de requisições

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

Deadlocks – Situação 1

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)

UF P I UCD

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

Matriz de alocação

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

Matriz de requisições

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

Nessa situação, nenhum processo pode ser atendido!

DEADLOCK

Deadlocks

- 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;

Deadlocks

- **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 *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**;

Deadlocks

- **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;

Deadlocks

- **Evitar dinamicamente o problema**:
  - 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

- 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 estado inseguro, não é possível garantir que os processos terminarão corretamente;

Deadlocks

- 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

- 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

Seguro

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

Seguro

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

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

- 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

Seguro

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

Seguro

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

Inseguro

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

Deadlocks

- 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 = Recursos Alocados

Recursos → E = (6 3 4 2);
Alocados → P = (5 3 2 2);
Disponíveis → 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 = Recursos ainda necessários

Deadlocks

- 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

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

A	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

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

Deadlocks

- 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);

A	1	1	0	0
B	0	1	0	2
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:*

Deadlocks

- 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;

Deadlocks

- Prevenir *Deadlocks*:
  - Atacar uma das quatro condições:

Condição	Abordagem
Exclusão Mútua	Alocar todos os recursos usando um *spool*
Uso 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

- *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);}
```

Departamento de Computação

- *Capitulo 6 - Tanenbaum*