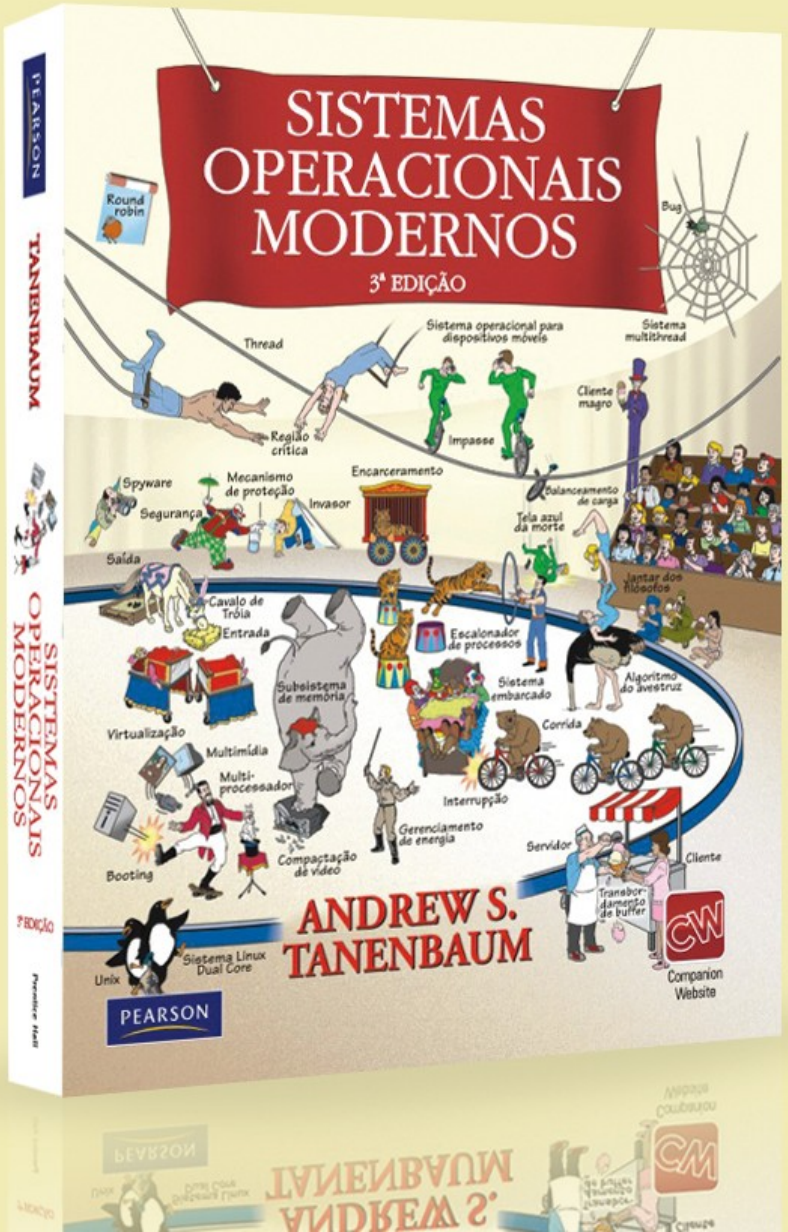
Sistemas operacionais modernos

Terceira edição

ANDREW S. TANENBAUM

Capítulo 6 Impasses

ou Capítulo 7 (Silberschatz)



Recursos preemptíveis e não preemptíveis

A sequência abstrata de eventos necessária ao uso de um determinado recurso é apresentada abaixo:

1. Requisitar o recurso.
2. Usar o recurso.
3. Liberar o recurso.

Exemplos de recursos:

Recursos de hardware:

- Dispositivos de I/O (ex. Scanner e impressora)

Recursos de software:

- Registros de banco de dados

Aquisição de recursos

Exemplos de uso de semáforos para obter acesso exclusivo a recursos compartilhados

```
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.

Exemplo

```
1  typedef struct conta_t
2  {
3     int saldo ;                      // saldo atual da conta
4     sem_t lock ;                     // semáforo associado à conta
5     ...                              // outras informações da conta
6  } conta_t ;
7
8  void transferir (conta_t* contaDeb, conta_t* contaCred, int valor)
9  {
10    sem_down (contaDeb->lock) ;      // obtém acesso a contaDeb
11    sem_down (contaCred->lock) ;     // obtém acesso a contCred
12
13    if (contaDeb->saldo >= valor)
14    {
15       contaDeb->saldo -= valor ;    // debita valor de contaDeb
16       contaCred->saldo += valor ;   // credita valor em contaCred
17    }
18    sem_up (contaDeb->lock) ;        // libera acesso a contaDeb
19    sem_up (contaCred->lock) ;       // libera acesso a contaCred
20 }
```

Considerando um sistema multithreads, o código acima pode gerar deadlock?

Exemplo

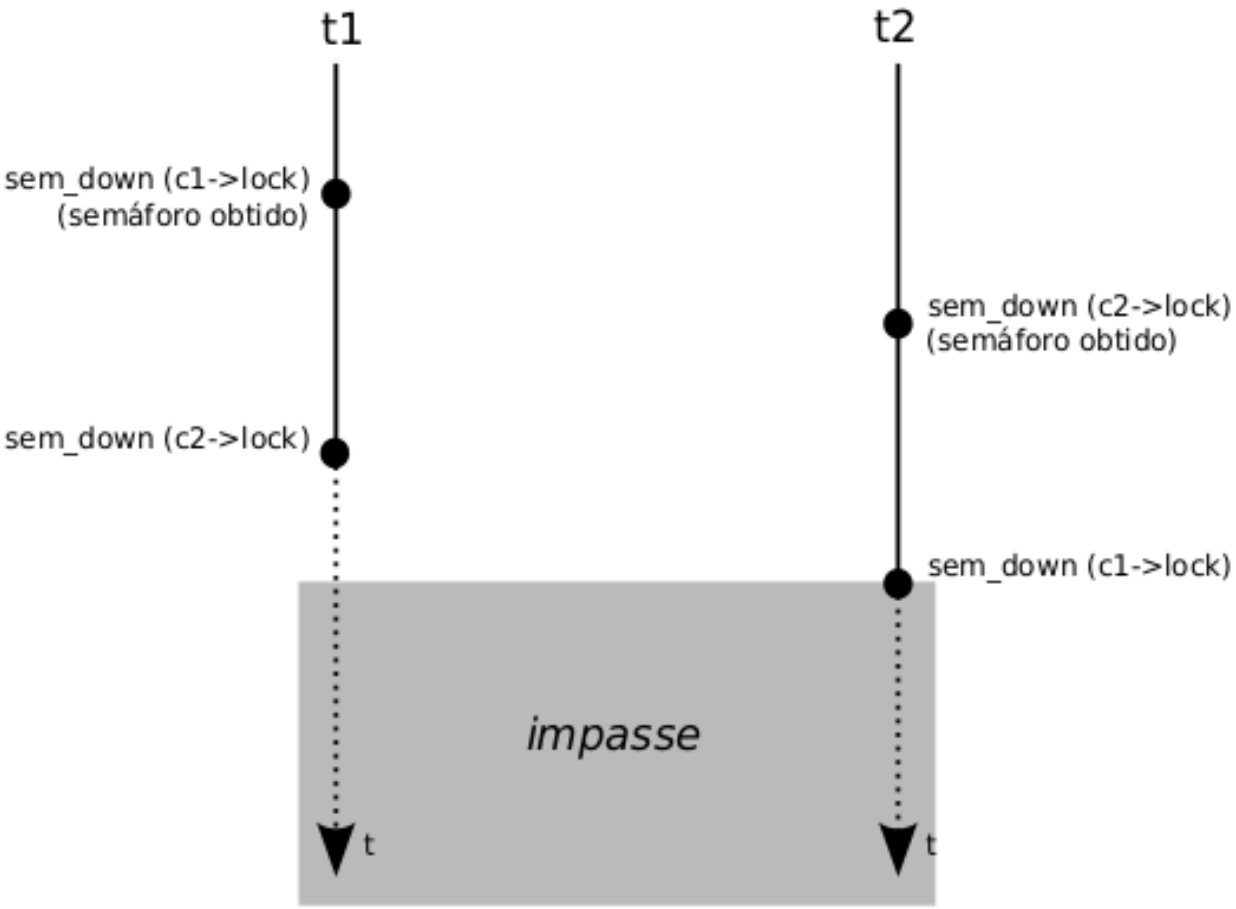
```
1  typedef struct conta_t
2  {
3     int saldo ;                    // saldo atual da conta
4     sem_t lock ;                   // semáforo associado à conta
5     ...                            // outras informações da conta
6  } conta_t ;
7
8  void transferir (conta_t* contaDeb, conta_t* contaCred, int valor)
9  {
10    sem_down (contaDeb->lock) ;    // obtém acesso a contaDeb
11    sem_down (contaCred->lock) ;   // obtém acesso a contCred
12
13    if (contaDeb->saldo >= valor)
14    {
15       contaDeb->saldo -= valor ;  // debita valor de contaDeb
16       contaCred->saldo += valor ; // credita valor em contaCred
17    }
18    sem_up (contaDeb->lock) ;      // libera acesso a contaDeb
19    sem_up (contaCred->lock) ;     // libera acesso a contaCred
20 }
```

Se a operação na conta fosse apenas retirada (saque), seria necessário o uso de um semáforo?

Se a operação na conta fosse apenas retirada (saque), mesmo assim poderia ocorrer deadlock?

Situação de deadlock

Situação de deadlock para o exemplo anterior (transação bancária)

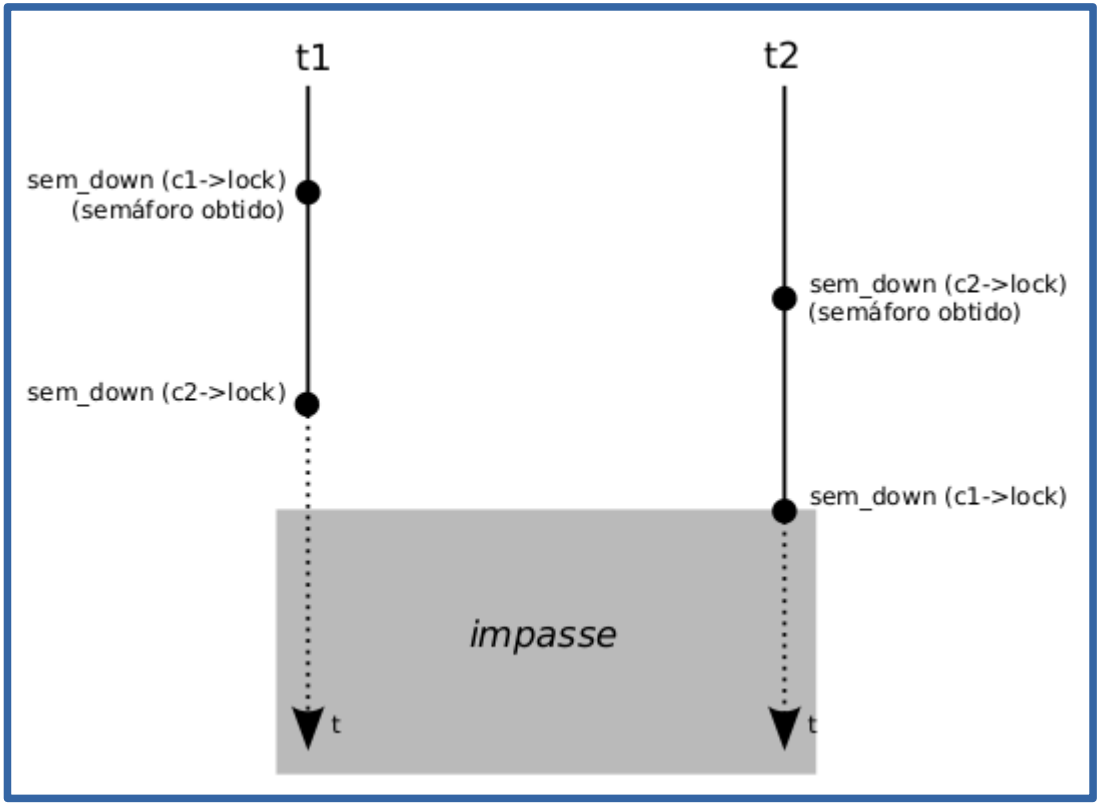


Condições para impasse de recursos

1. Condição de exclusão mútua.
2. Condição de posse e espera.
3. Condição de não preempção.
4. Condição de espera circular.

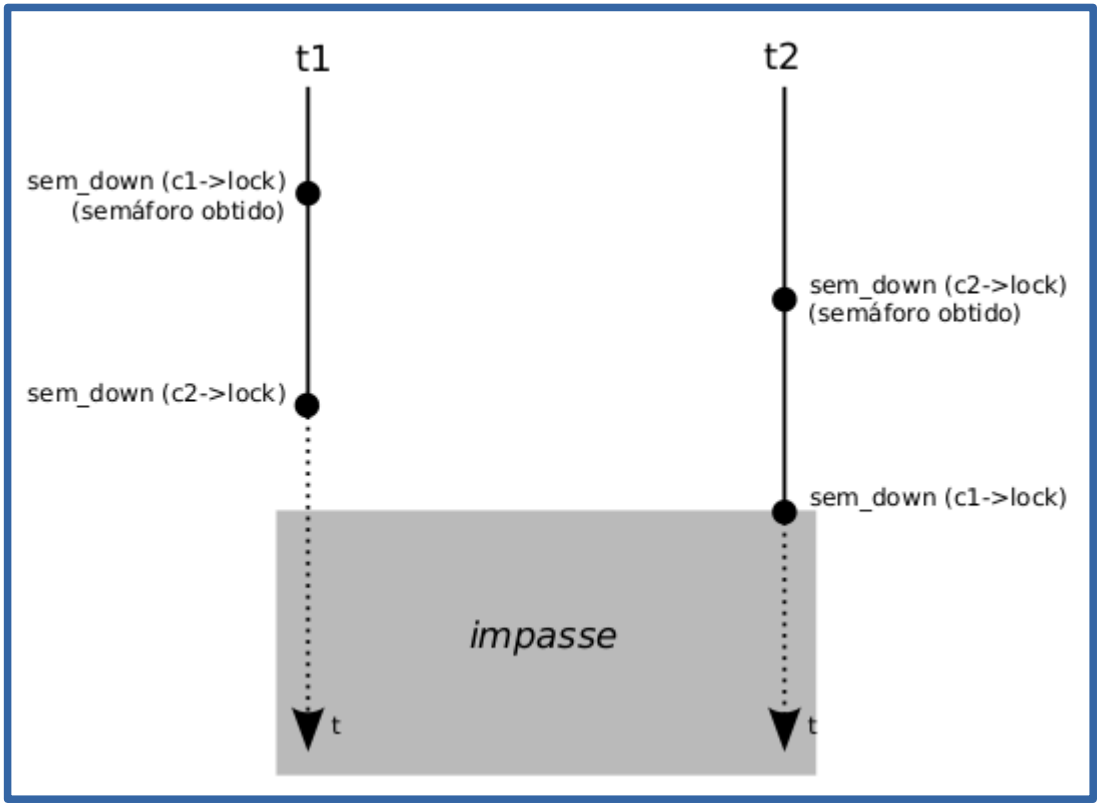
```
sem_down (contaDeb->lock) ;
sem_down (contaCred->lock) ;

if (contaDeb->saldo >= valor)
{
   contaDeb->saldo -= valor ;
   contaCred->saldo += valor ;
}
sem_up (contaDeb->lock) ;
sem_up (contaCred->lock) ;
```



Definição de Impasses (deadlocks)

- *<u>DEFINIÇÃO</u>: 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.*



Os programas 1 e 2 apresentados abaixo podem apresentar impasses ?

Programa 1

```
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)

Programa 2

```
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)

Modelagem de impasses

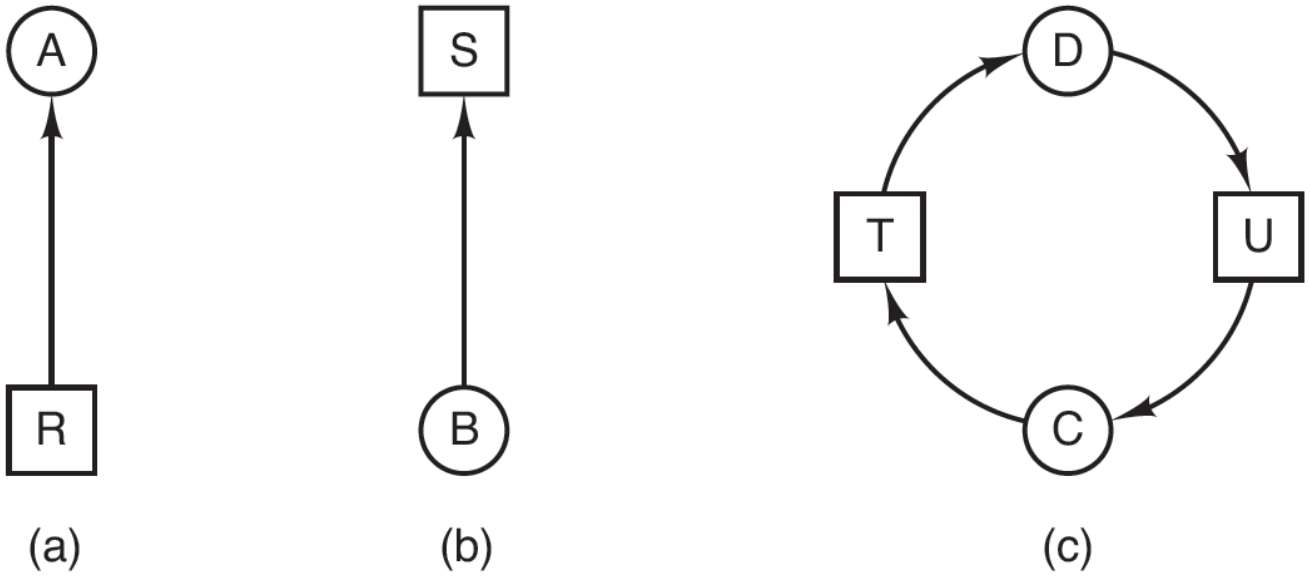


Figura 6.3 Grafos de alocação de recursos. (a) Processo de posse de um recurso. (b) Processo requisitando um recurso. (c) Impasse.

Modelagem de impasses

Considere que 3 processos A, B e C podem utilizar 3 recursos: R, S e T.

A	B	C
Requisita R	Requisita S	Requisita T
Requisita S	Requisita T	Requisita R
Libera R	Libera S	Libera T
Libera S	Libera T	Libera R
(a)	(b)	(c)

Se o algoritmo de escalonamento por revezamento (RR) estiver sendo usado, poderá ocorrer deadlock no sistema? E se for o FCFS?

Modelagem de impasses

Deadlock pode ocorrer dependendo da ordem que os recursos são solicitados.

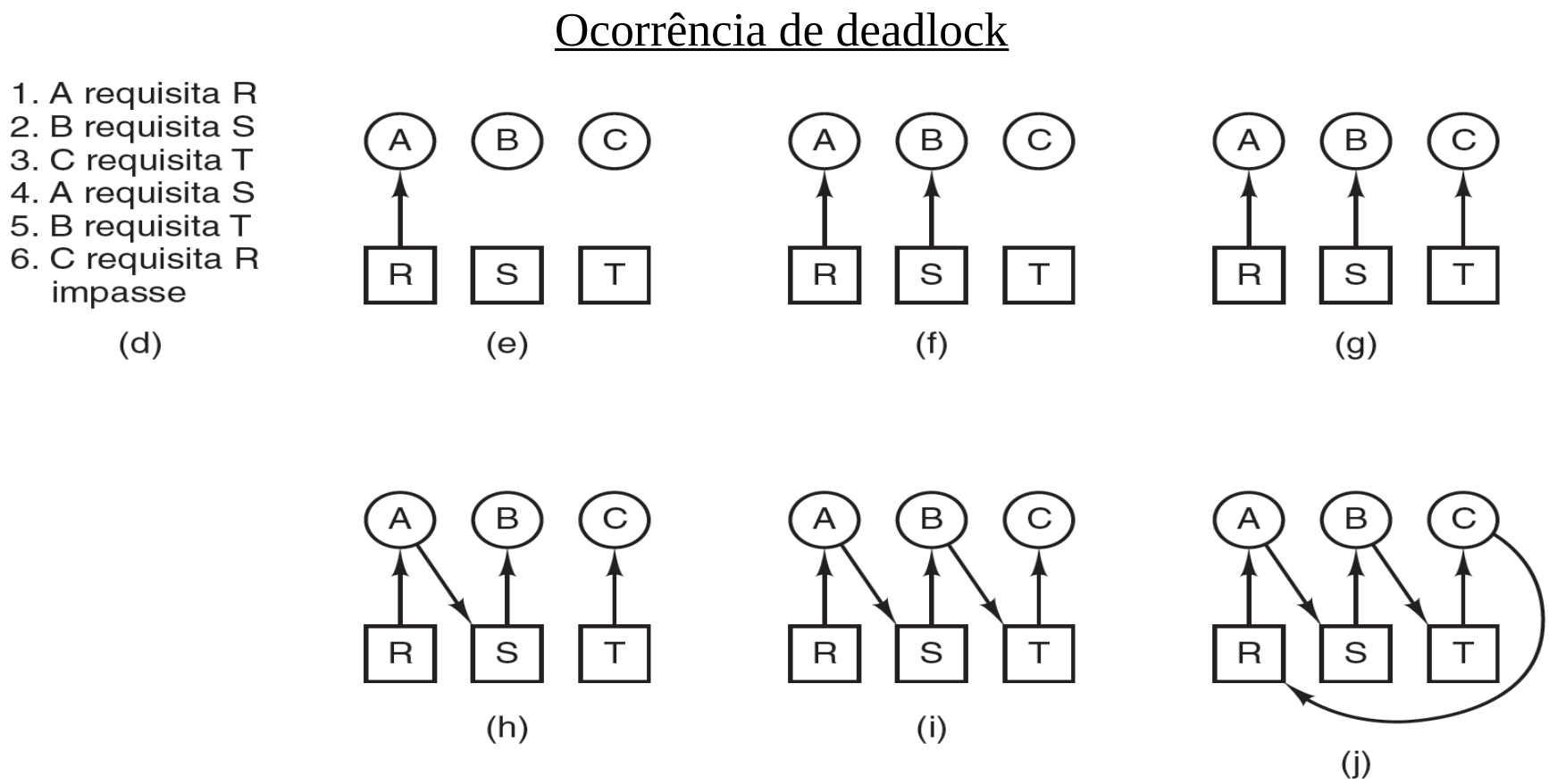


Figura 6.4 Exemplo de como um impasse ocorre e como pode ser evitado.

Modelagem de impasses

Deadlock pode ocorrer dependendo da ordem que os recursos são solicitados.

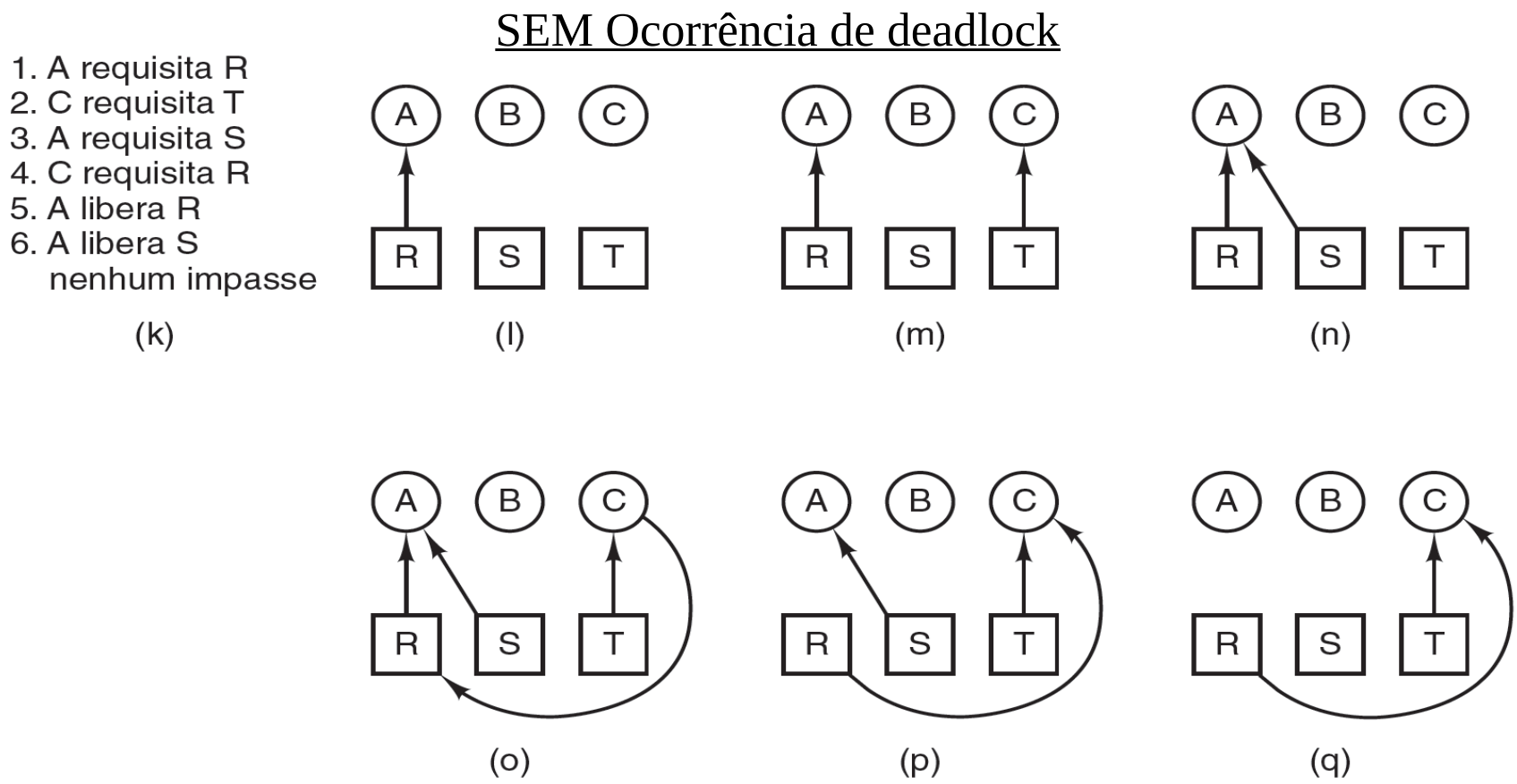


Figura 6.4 Exemplo de como um impasse ocorre e como pode ser evitado.

Detecção de impasse com um recurso de cada tipo

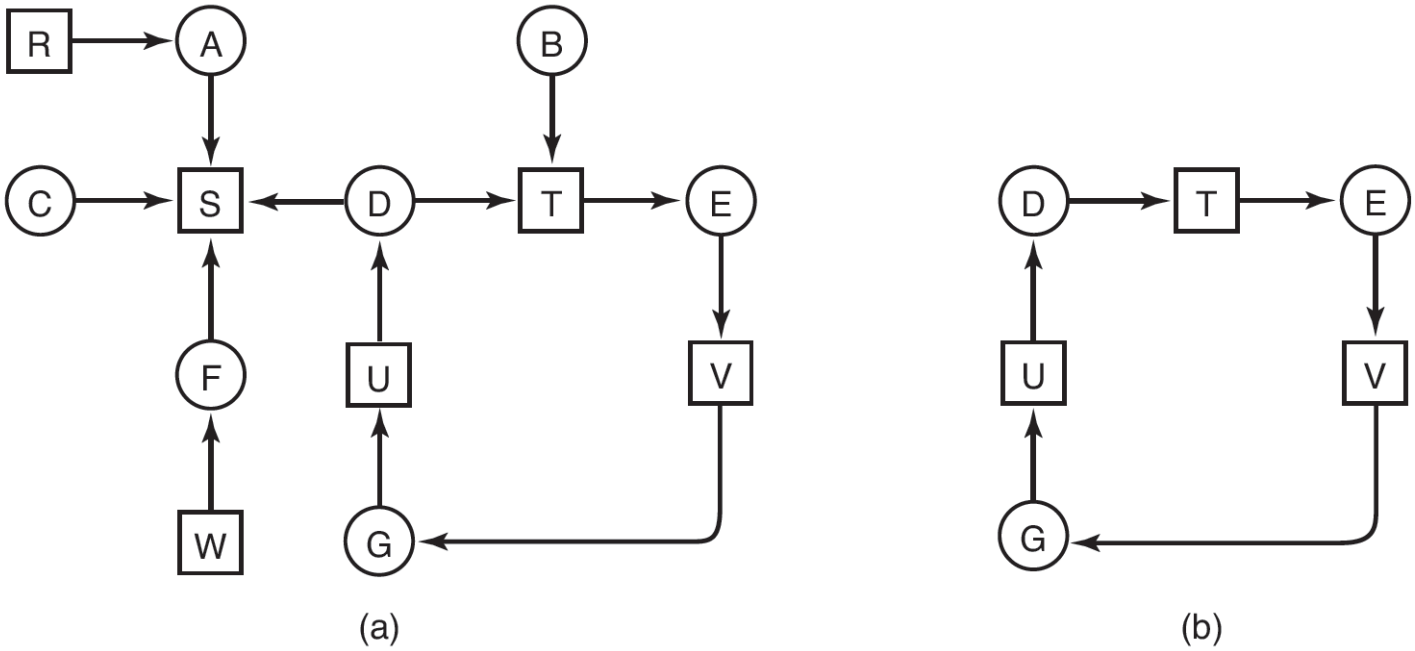


Figura 6.5 (a) Um gráfico de recursos. (b) Um ciclo extraído de (a).

Detecção de impasse com um recurso de cada tipo

Algoritmo para detecção de impasse:

1. Para cada nó, N no gráfico, realize os cinco passos seguintes, usando N como nó inicial.
2. Inicialize uma lista vazia L que conterá os nós visitados e assinale todos os arcos como não percorridos.
3. Insira o nó atual no final de L, **verifique se o nó aparece em L duas vezes. Se sim, o gráfico contém um ciclo** e o algoritmo termina.
4. A partir do nó dado, verifique se existe algum arco de saída desmarcado. Em caso afirmativo, vá para o passo 5; do contrário, vá para o passo 6.
5. Escolha aleatoriamente um arco de saída desmarcado e marque-o. Então, siga esse arco para obter o novo nó atual e vá para o passo 3.
6. Se esse nó for o inicial, o gráfico não conterá ciclo algum e o algoritmo terminará. Senão, o final foi alcançado. Remova-o e volte para o nó anterior — isto é, aquele que era atual antes desse —, marque-o como atual e vá para o passo 3.

Detecção de impasse com um recurso de cada tipo

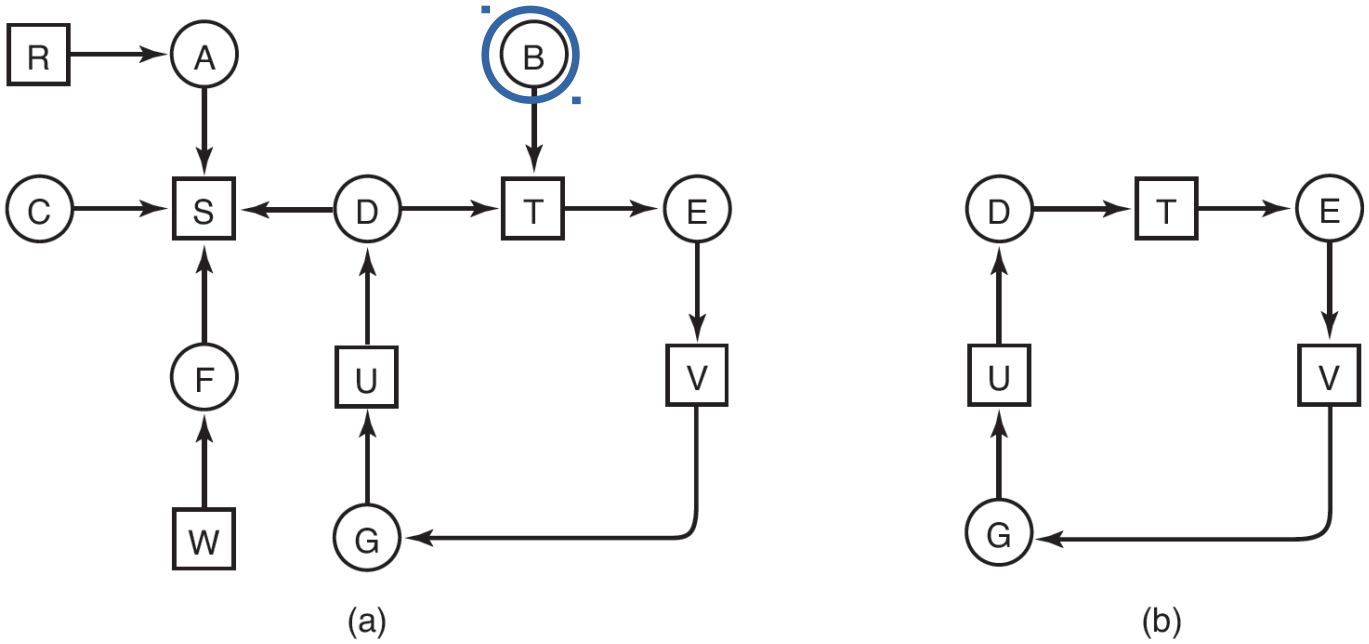


Figura 6.5 (a) Um gráfico de recursos. (b) Um ciclo extraído de (a).

L=[B]

Detecção de impasse com um recurso de cada tipo

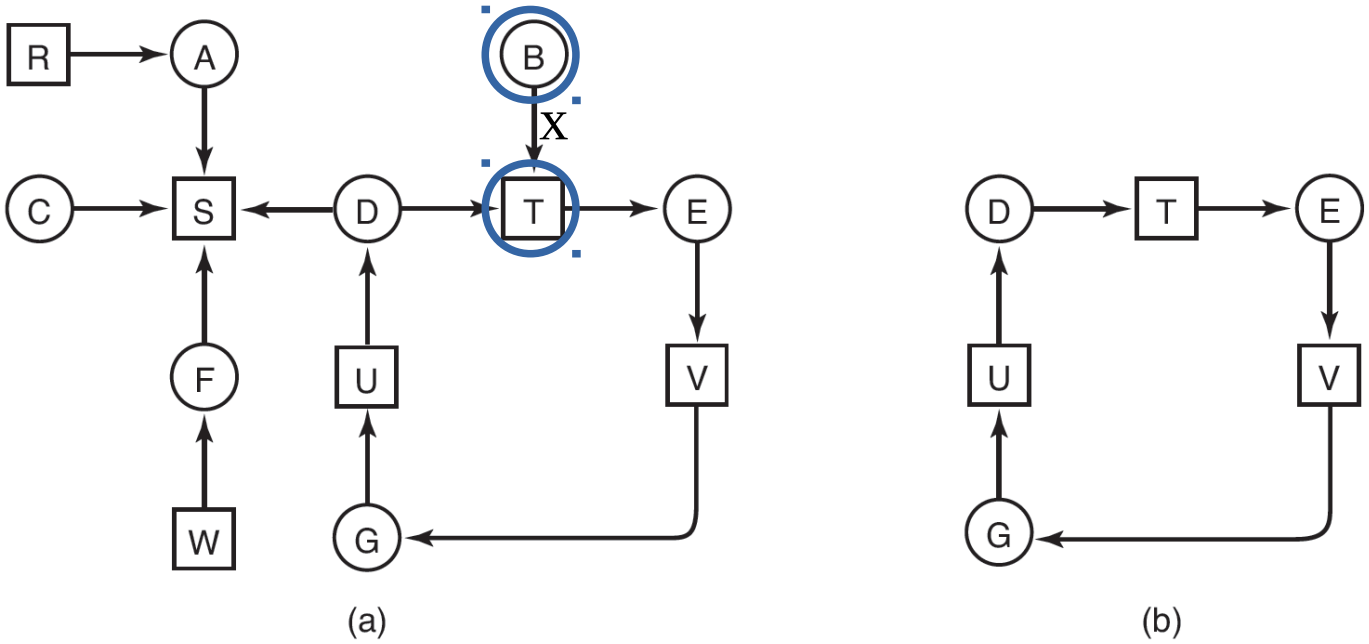


Figura 6.5 (a) Um gráfico de recursos. (b) Um ciclo extraído de (a).

$$L=[B,T]$$

Detecção de impasse com um recurso de cada tipo

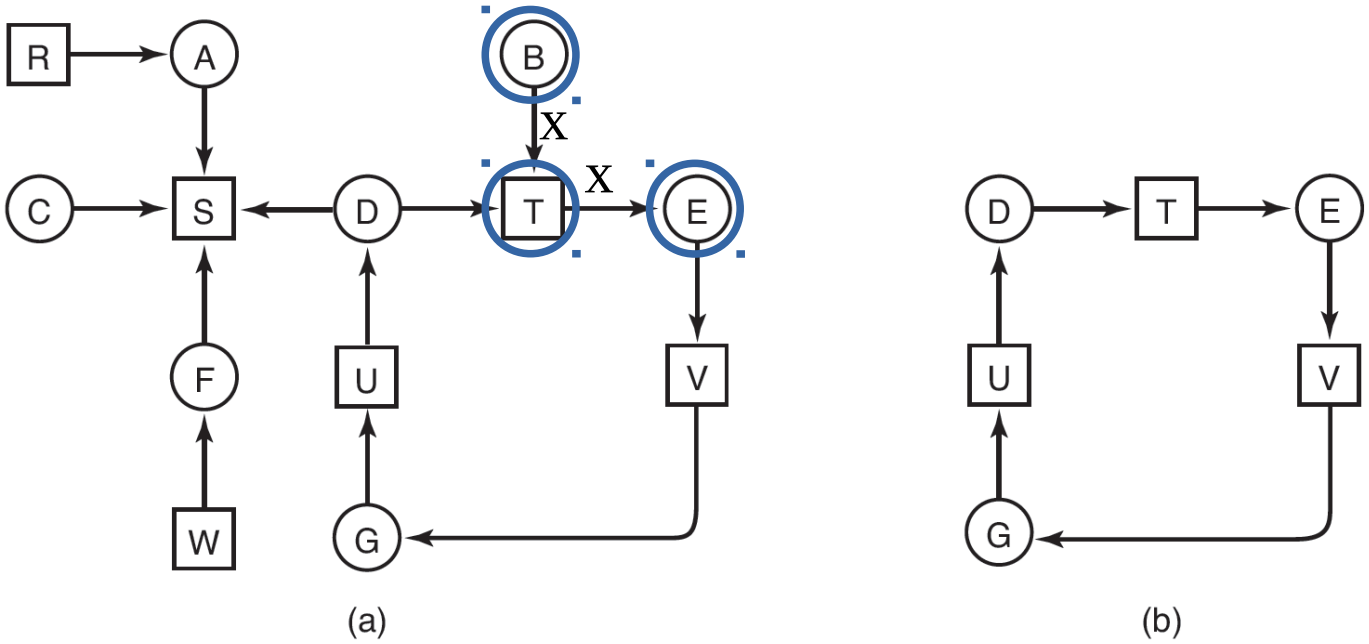


Figura 6.5 (a) Um gráfico de recursos. (b) Um ciclo extraído de (a).

$$L=[B,T,E]$$

Detecção de impasse com um recurso de cada tipo

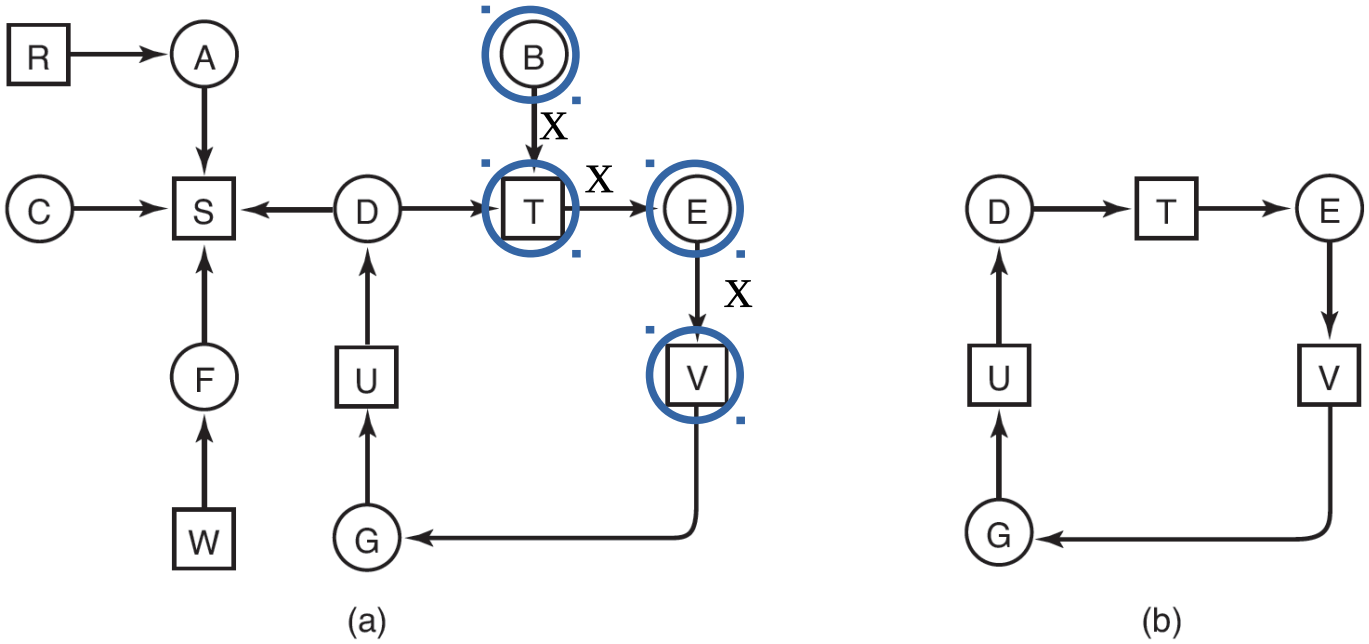


Figura 6.5 (a) Um gráfico de recursos. (b) Um ciclo extraído de (a).

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

Detecção de impasse com um recurso de cada tipo

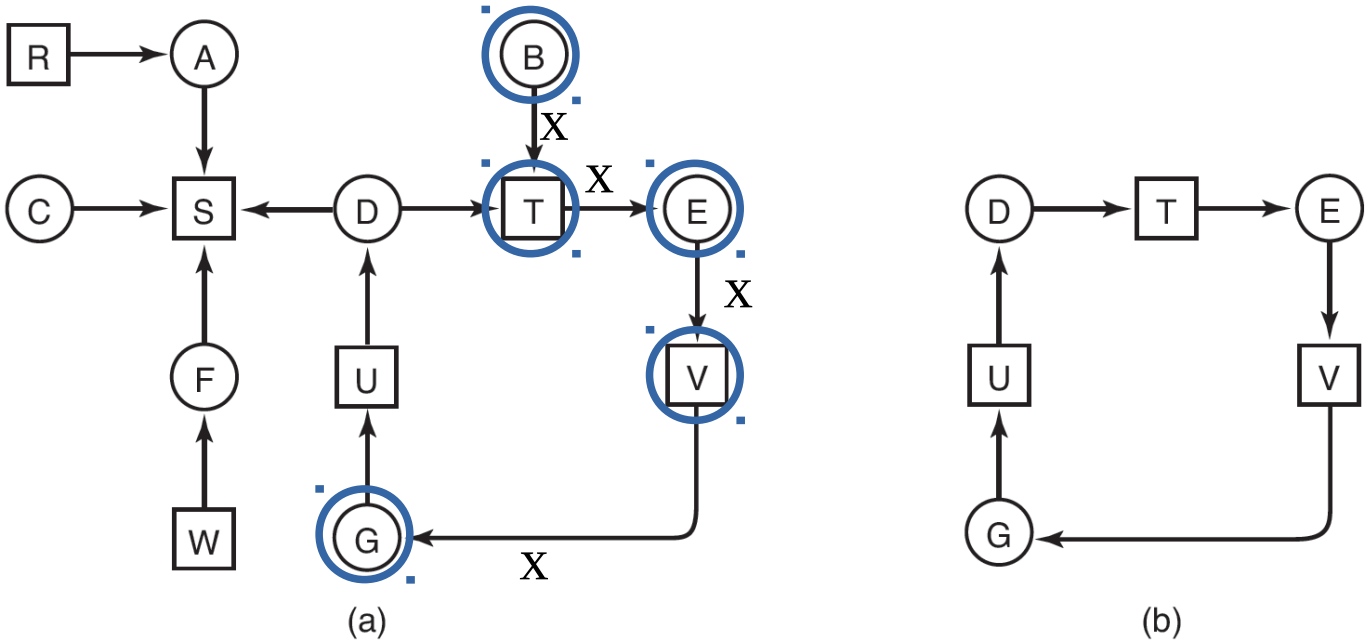


Figura 6.5 (a) Um gráfico de recursos. (b) Um ciclo extraído de (a).

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

Detecção de impasse com um recurso de cada tipo

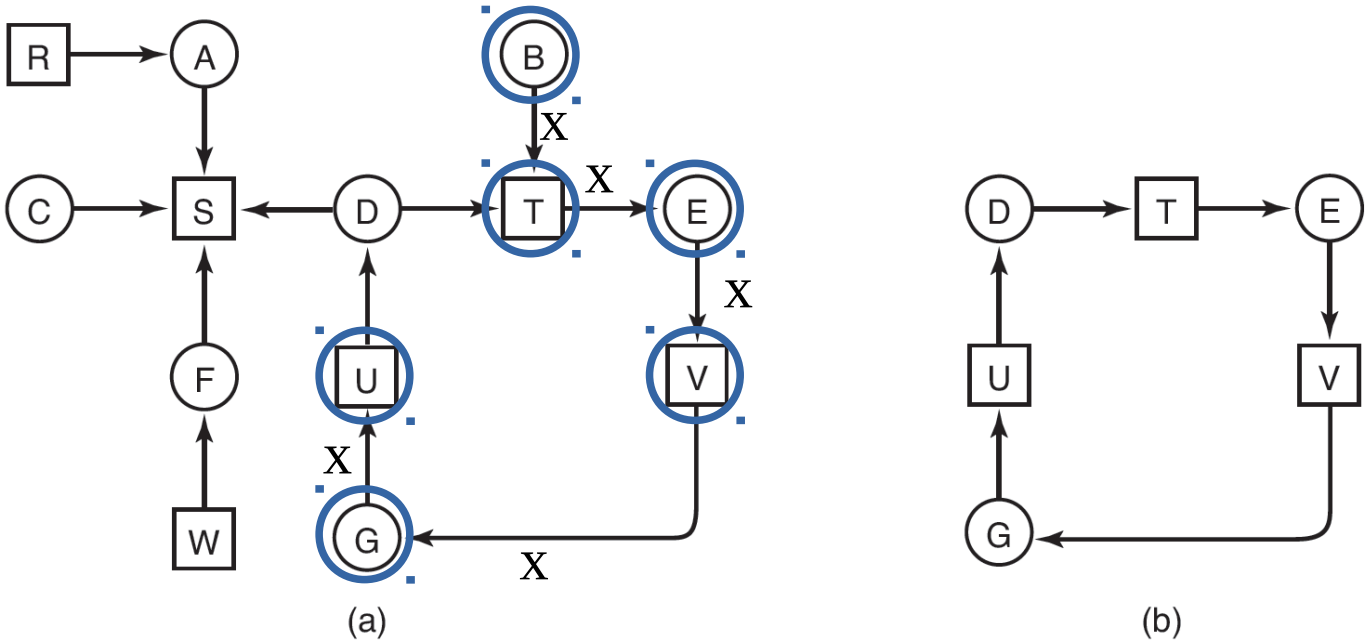


Figura 6.5 (a) Um gráfico de recursos. (b) Um ciclo extraído de (a).

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

Detecção de impasse com um recurso de cada tipo

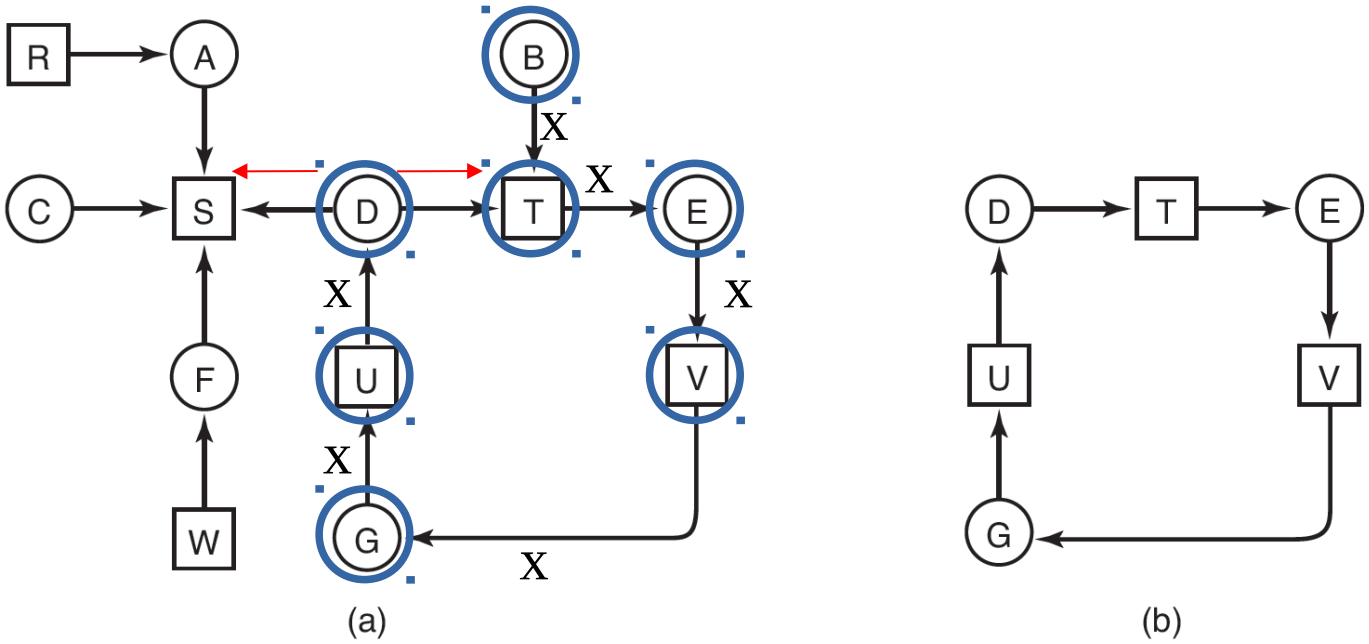


Figura 6.5 (a) Um gráfico de recursos. (b) Um ciclo extraído de (a).

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

Detecção de impasse com um recurso de cada tipo

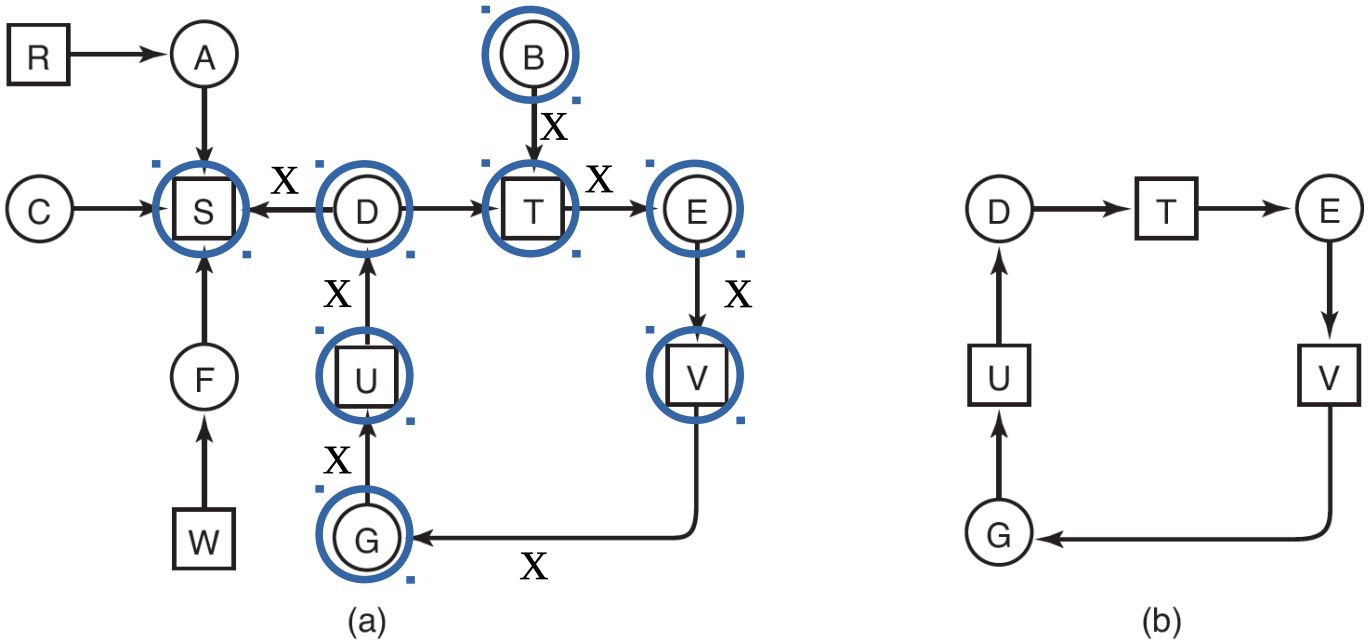


Figura 6.5 (a) Um gráfico de recursos. (b) Um ciclo extraído de (a).

L=[B,T,E,V,G,U,D,S] Sem saída

Detecção de impasse com um recurso de cada tipo

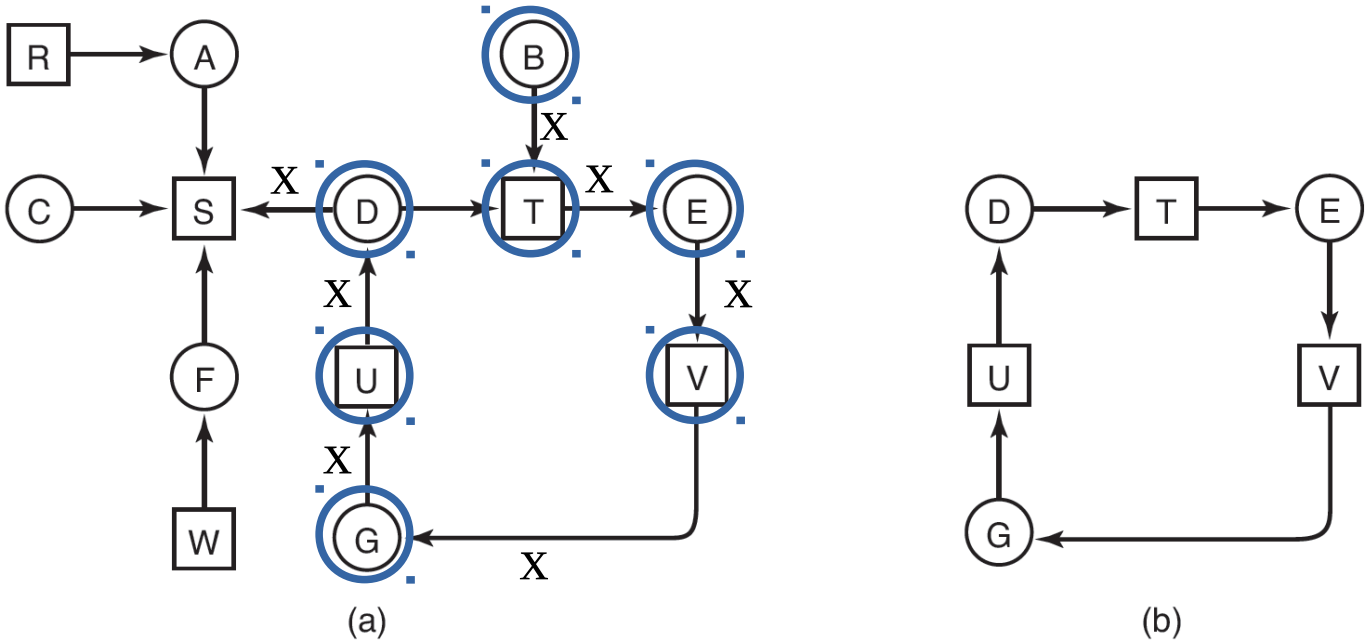


Figura 6.5 (a) Um gráfico de recursos. (b) Um ciclo extraído de (a).

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

Detecção de impasse com um recurso de cada tipo

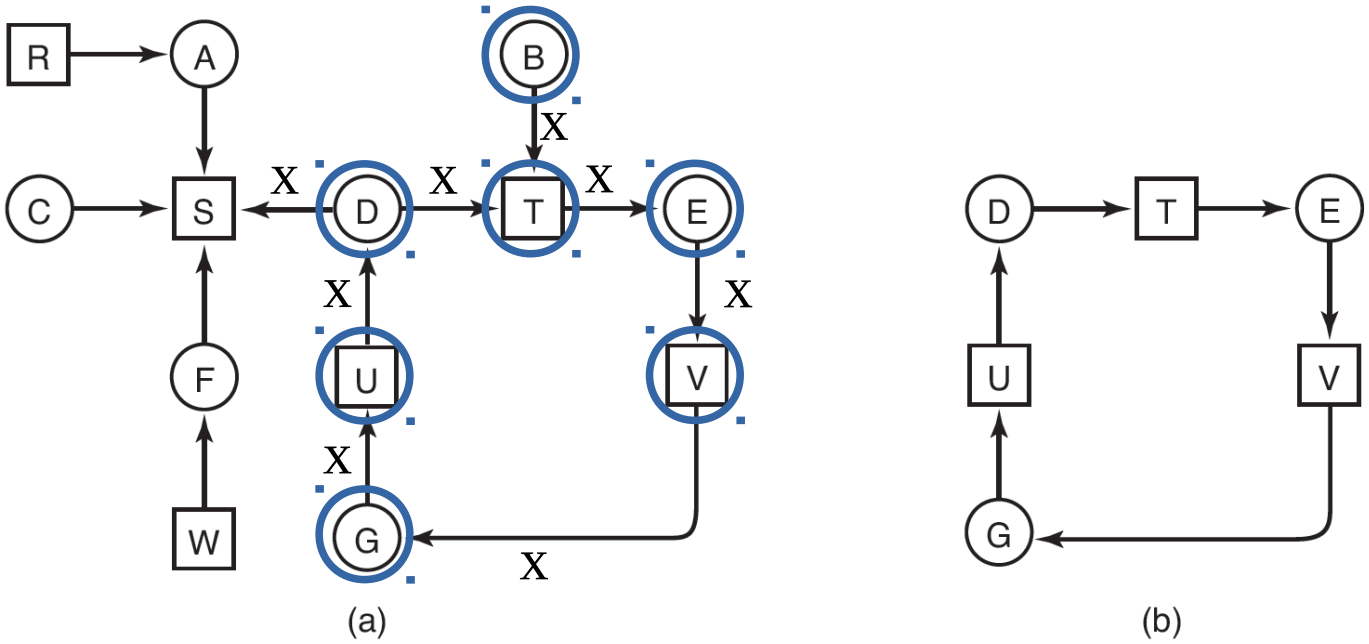


Figura 6.5 (a) Um gráfico de recursos. (b) Um ciclo extraído de (a).

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

Detecção de impasse com um recurso de cada tipo

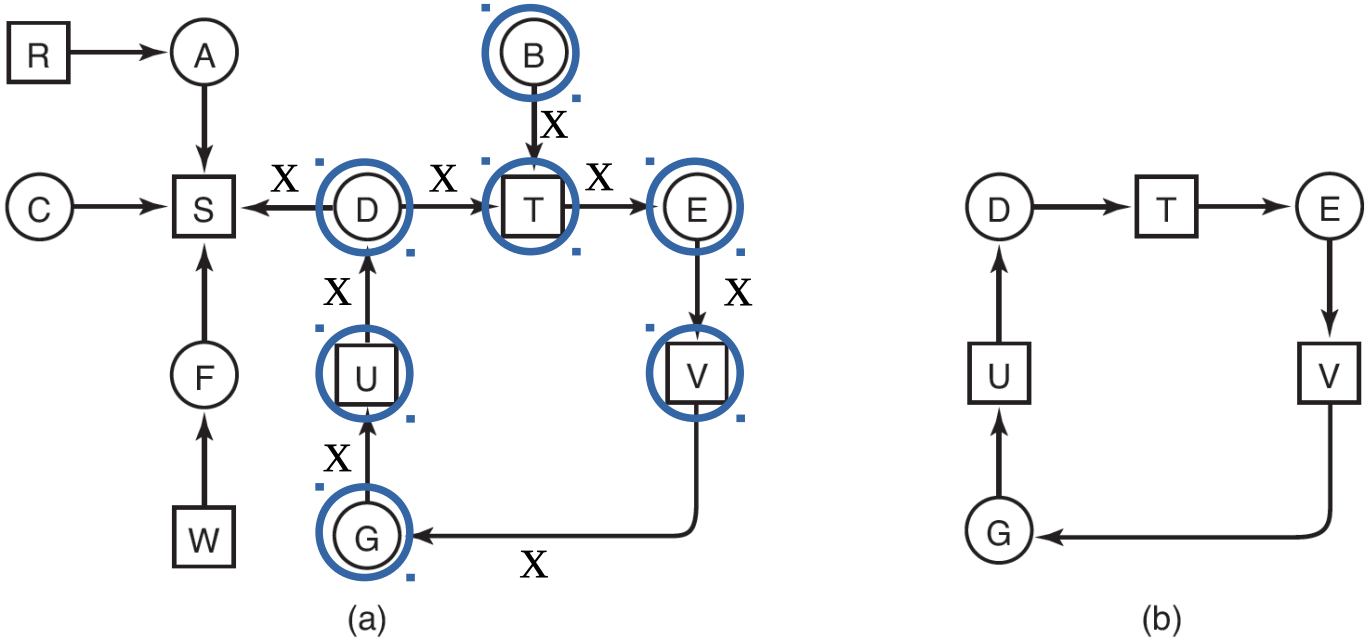


Figura 6.5 (a) Um gráfico de recursos. (b) Um ciclo extraído de (a).

- O algoritmo para detecção de ciclos pode ser implementado pelo SO?
- Com qual frequência o processo que detecta ciclos pode ser executado? (ex. 1 vez a cada 100 ms)
- O processo que detecta ciclos pode ser interrompido durante a execução?

Detecção de impasse com múltiplos recursos de cada tipo

Recursos existentes
$(E_1, E_2, E_3, \ldots, E_m)$

Matriz de alocação atual

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix}$$

Linha n é a alocação atual para o processo n

Recursos disponíveis
$(A_1, A_2, A_3, \ldots, A_m)$

Matriz de requisições

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$$

Linha 2 informa qual é a necessidade do processo 2

Detecção de impasse com múltiplos recursos de cada tipo

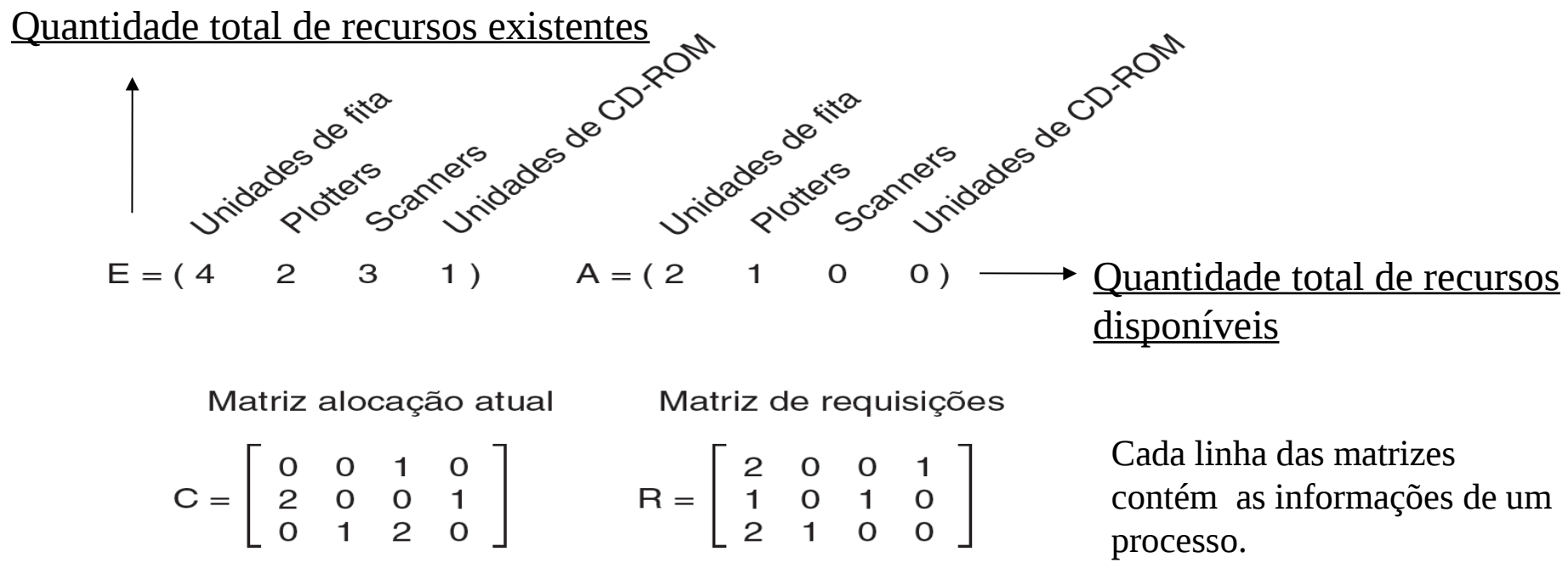


Figura 6.7 Um exemplo para o algoritmo de detecção de impasses.

Detecção de impasse com múltiplos recursos de cada tipo

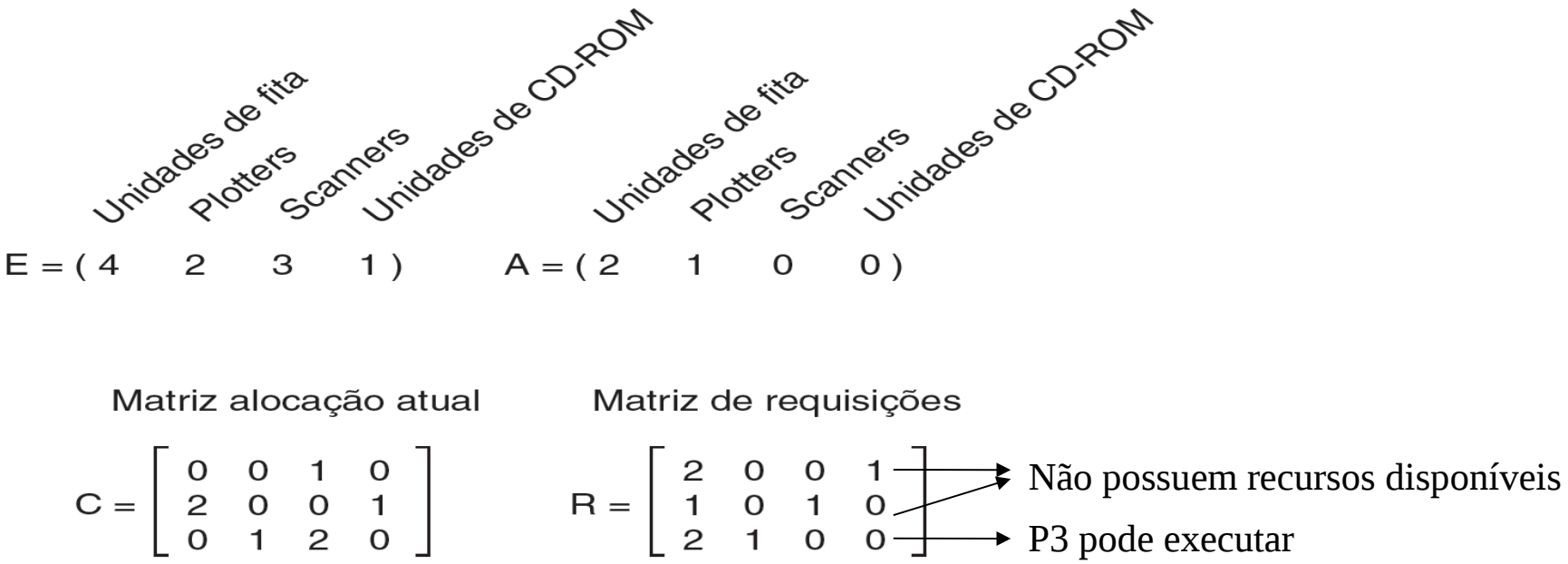


Figura 6.7 Um exemplo para o algoritmo de detecção de impasses.

Detecção de impasse com múltiplos recursos de cada tipo

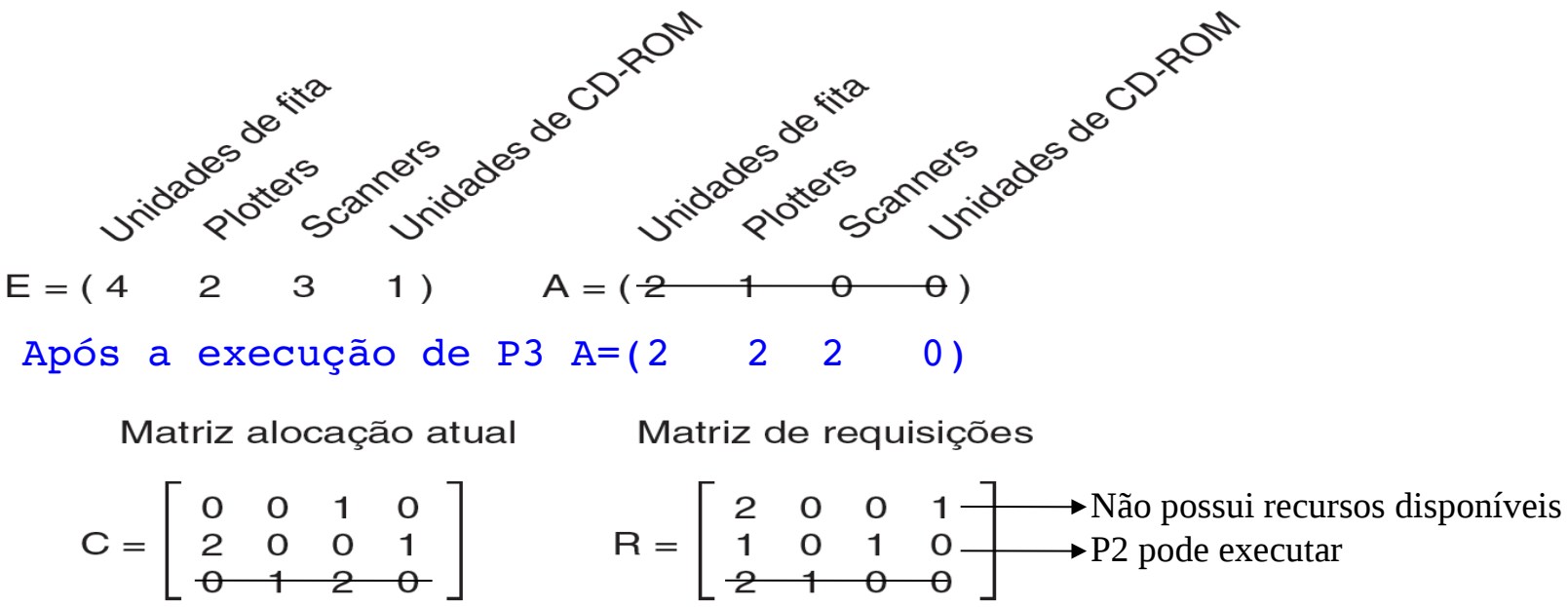


Figura 6.7 Um exemplo para o algoritmo de detecção de impasses.

Detecção de impasse com múltiplos recursos de cada tipo

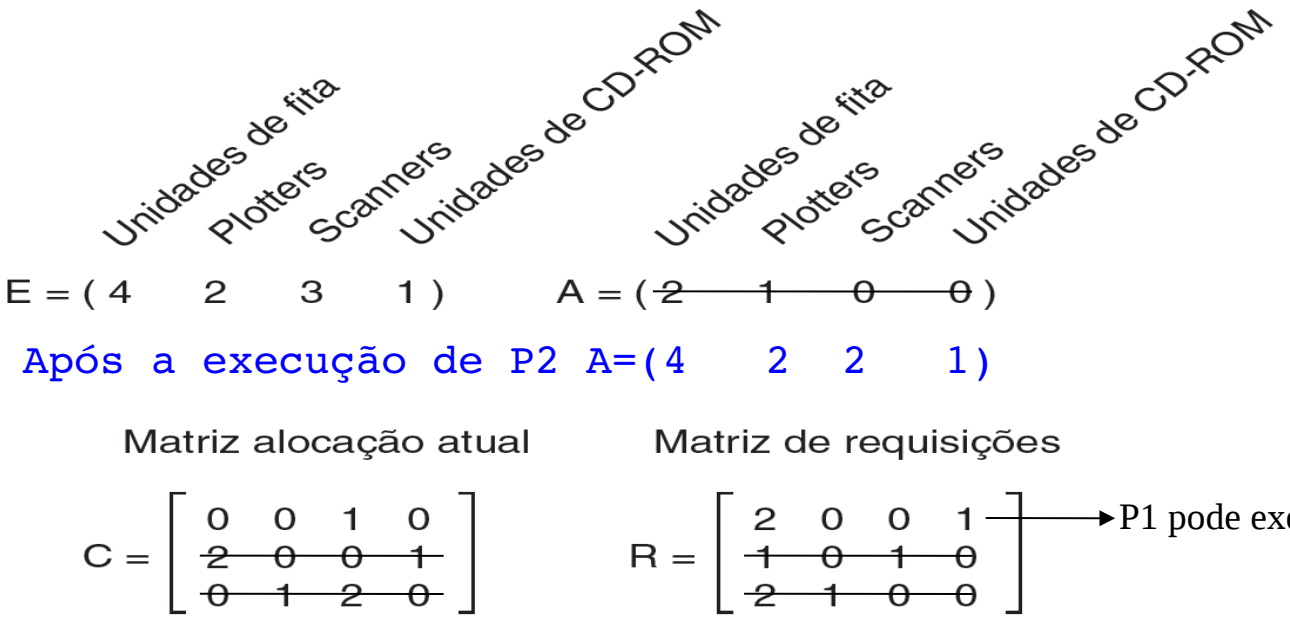


Figura 6.7 Um exemplo para o algoritmo de detecção de impasses.

Detecção de impasse com múltiplos recursos de cada tipo

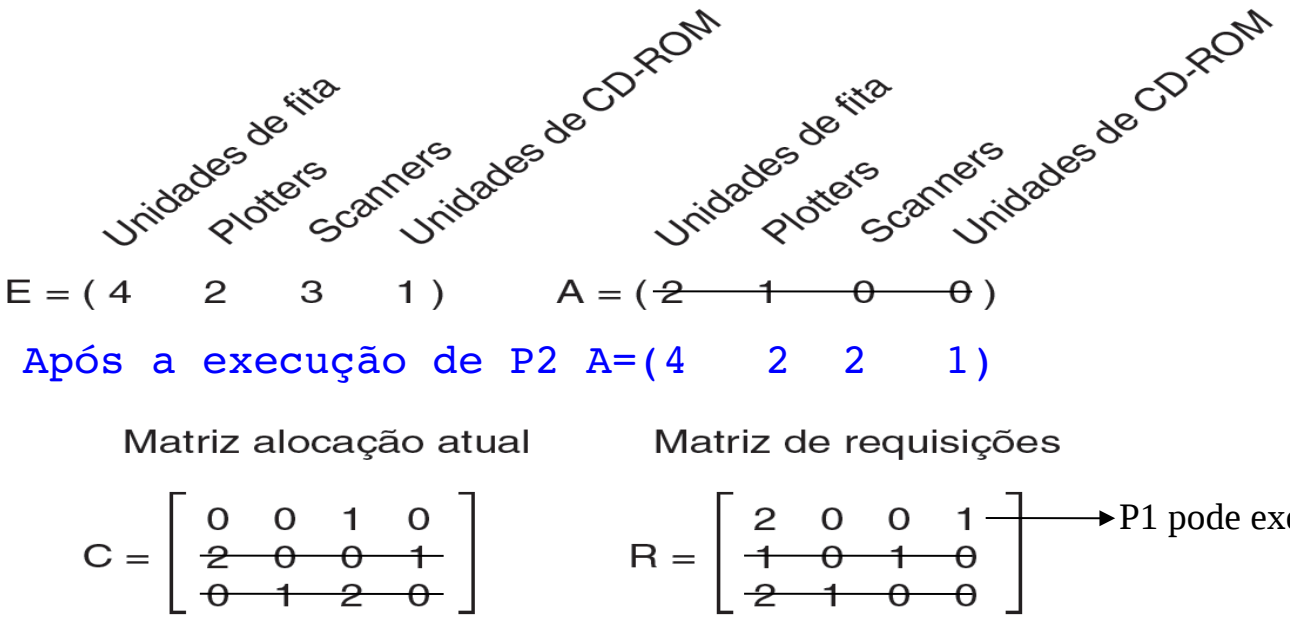


Figura 6.7 Um exemplo para o algoritmo de detecção de impasses.

Se não existir uma linha em R que seja menor ou igual à correspondente de *A* → **deadlock**

Detecção de impasse com múltiplos recursos de cada tipo

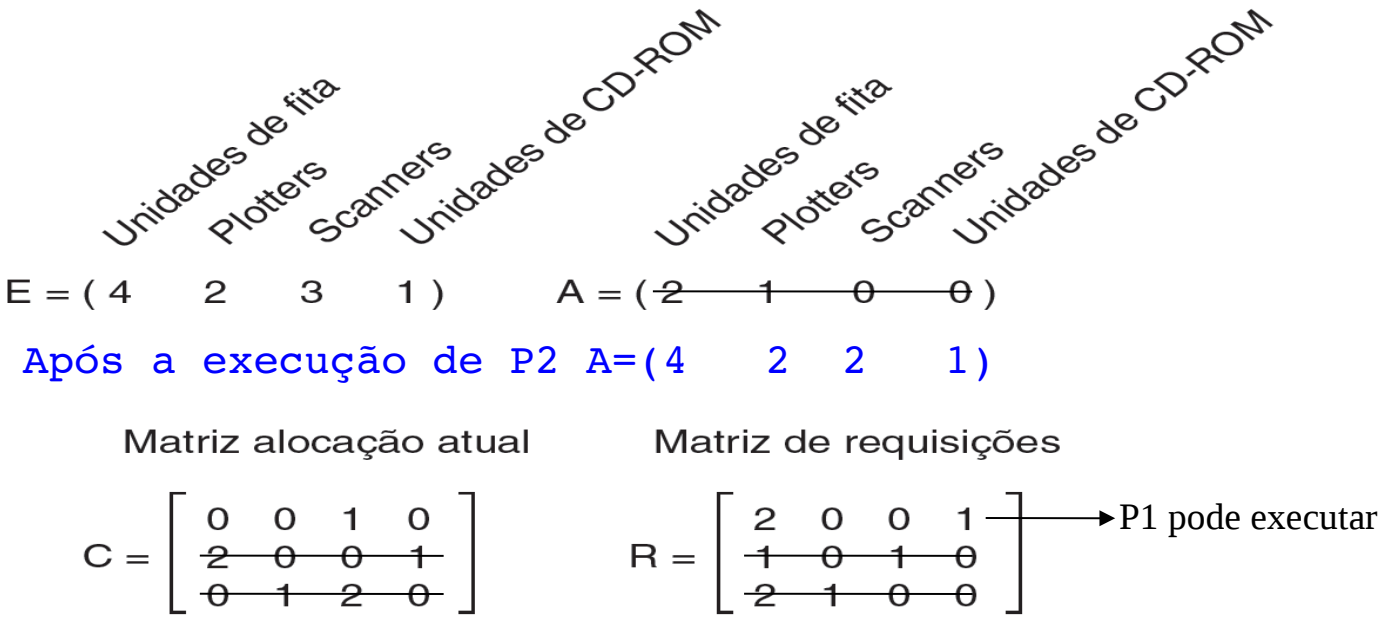


Figura 6.7 Um exemplo para o algoritmo de detecção de impasses.

<u>Quando procurar pelos impassses?</u>

- toda vez que um recurso for solicitado (custo alto)
- a cada k minutos
- quando o uso da CPU estiver baixo (o que pode indicar um impasse)

Detecção de impasse com múltiplos recursos de cada tipo

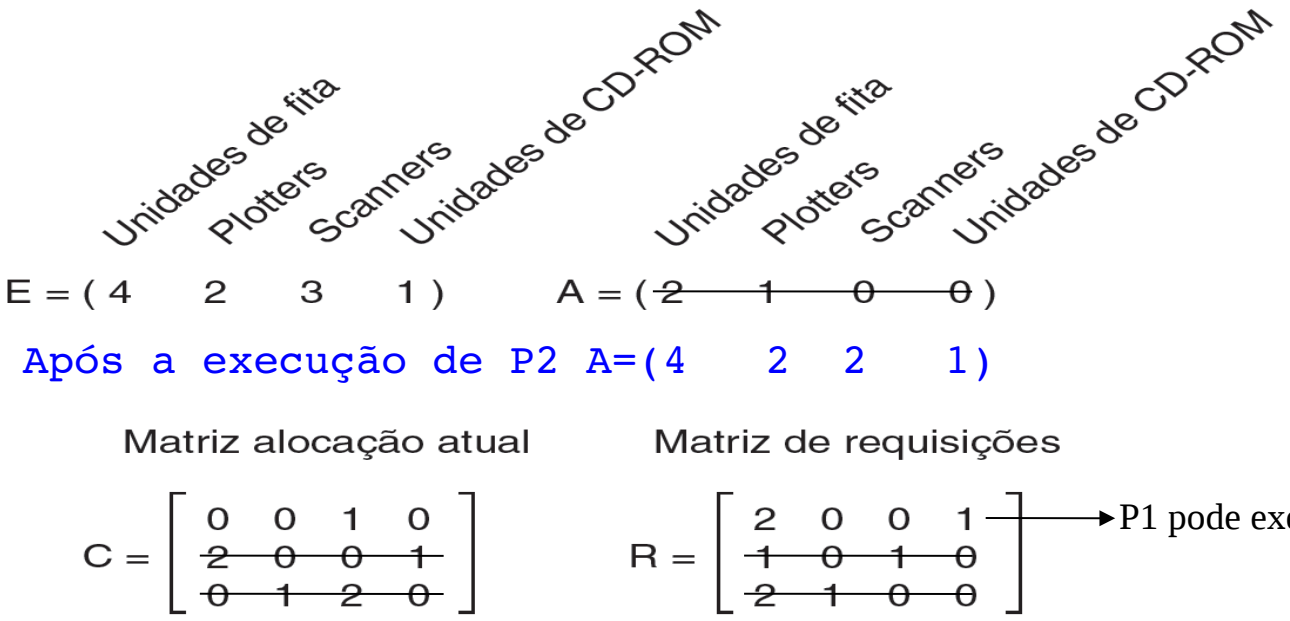


Figura 6.7 Um exemplo para o algoritmo de detecção de impasses.

Quando incrementar um valor na matriz de requisições? O valor pode ser incrementado toda vez que um processo der um *lock* no recurso?

As atualizações do vetor A e das matrizes C e R podem ser feitas após um certo intervalo de tempo específico (ex. 1 vez a cada 100 ms)?

Detecção de impasse com múltiplos recursos de cada tipo

Unidades de fita, Plotters, Scanners, Unidades de CD-ROM

E = (4 2 3 1) A = (2 1 0 0)

Unidades de fita, Plotters, Scanners, Unidades de CD-ROM

Após a execução de P2 A=(4 2 2 1)

Matriz alocação atual

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

Matriz de requisições

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

→ P1 pode executar

Figura 6.7 Um exemplo para o algoritmo de detecção de impasses.

Este método pode ser usado para prevenir impasses?

O que fazer após a detecção de um impasse?

Recuperação de impasses

Recuperação de impasses

Alguns métodos para se recuperar um de deadlock são:

- retirar o recurso do processo que está gerando o deadlock (**recuperação por preempção**). Esta solução depende da natureza do recurso.

- Retroceder o processo ao estado anterior à solicitação do recurso. (**retrocesso por meio de *checkpoints – roll back*)**
  - Como escolher o processo que será retrocedido?

Recuperação de impasses

Alguns métodos para se recuperar um de deadlock são:

- retirar o recurso do processo que está gerando o deadlock (**recuperação por preempção**). Esta solução depende da natureza do recurso.

- Retroceder o processo ao estado anterior à solicitação do recurso. (**retrocesso por meio de *checkpoints – roll back*)**
  - Como escolher o processo que será retrocedido?
    - O processo que possui a menor quantidade de *locks*.
    - O processo que realizou a menor quantidade de trabalho.
    - O processo que está mais longe de ser completado.

Recuperação de impasses

Alguns métodos para se recuperar um de deadlock são:

- retirar o recurso do processo que está gerando o deadlock (**recuperação por preempção**). Esta solução depende da natureza do recurso.

- Retroceder o processo ao estado anterior à solicitação do recurso. (**retrocesso por meio de *checkpoints – roll back*)**
  - Como escolher o processo que será retrocedido?
    - O processo que possui a menor quantidade de *locks*.
    - O processo que realizou a menor quantidade de trabalho.
    - O processo que está mais longe de ser completado.
  - Dependendo do método utilizado para escolher o processo que irá retroceder, poderá ocorrer inanição?

Recuperação de impasses

Alguns métodos para se recuperar um de deadlock são:

- retirar o recurso do processo que está gerando o deadlock (**recuperação por preempção**). Esta solução depende da natureza do recurso.

- Retroceder o processo ao estado anterior à solicitação do recurso. (**retrocesso por meio de *checkpoints – roll back*)**
  - Como escolher o processo que será retrocedido?
    - O processo que possui a menor quantidade de *locks*.
    - O processo que realizou a menor quantidade de trabalho.
    - O processo que está mais longe de ser completado.
  - Dependendo do método utilizado para escolher o processo que irá retroceder, poderá ocorrer inanição?
    - Sim, caso o mesmo processo seja sempre escolhido.

Recuperação de impasses

Alguns métodos para se recuperar um de deadlock são:

- Eliminar o processo que está gerando o deadlock.
  - A escolha do processo pode seguir os mesmos critérios utilizados no retrocesso por meio de checkpoints.
  - Por um lado, eliminar o processo como um todo elimina o custo de manter checkpoints. Por outro lado, um longo tempo de processamento pode ser perdido.

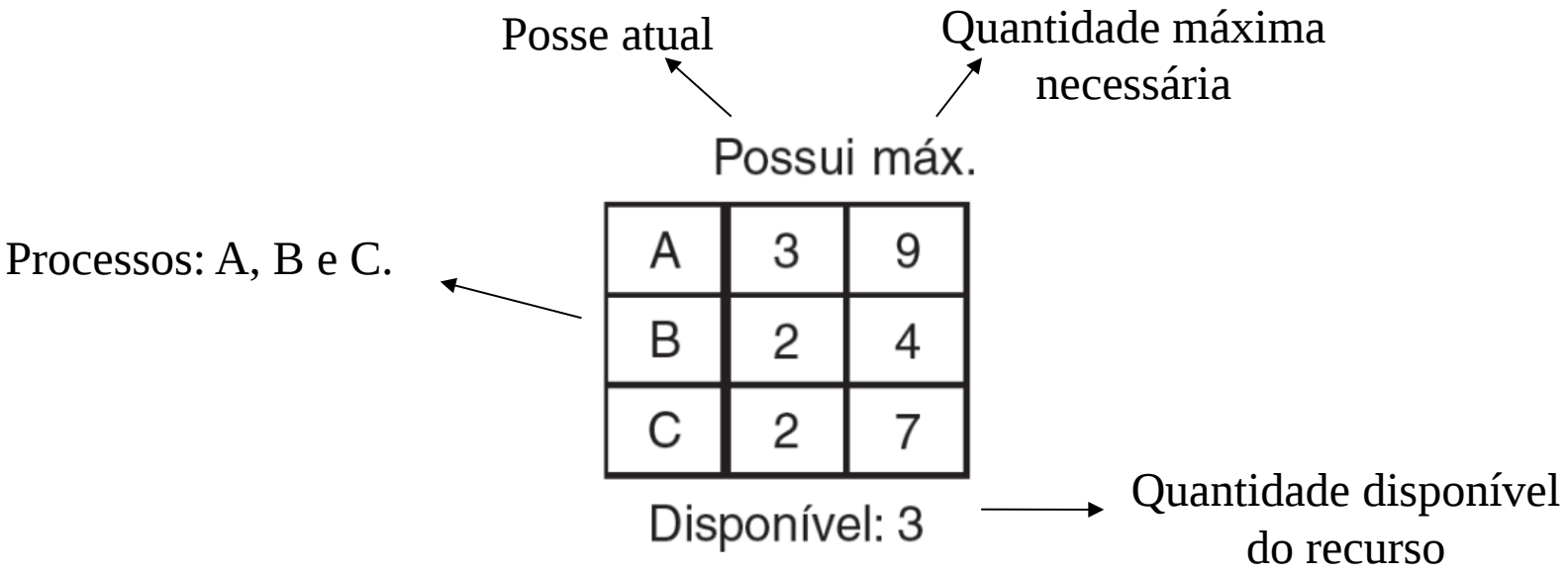
Situações:

- Detectando deadlocks (exemplo: ciclos em grafos)
- Recuperando de deadlocks
- Evitando deadlocks (anulação dinâmica): para se evitar deadlocks, podem ser identificados estados seguros e inseguros no sistema.

Evitando impasses (anulação dinâmica)

Estados seguros e inseguros

Considerando que um sistema armazene uma tabela com as informações sobre a utilização de um tipo de recurso



Como escolher o próximo processo que será executado de tal forma que um deadlock seja evitado?

Como evitar estados inseguros que podem levar a um deadlock?

Evitando impasses

Estados seguros e inseguros

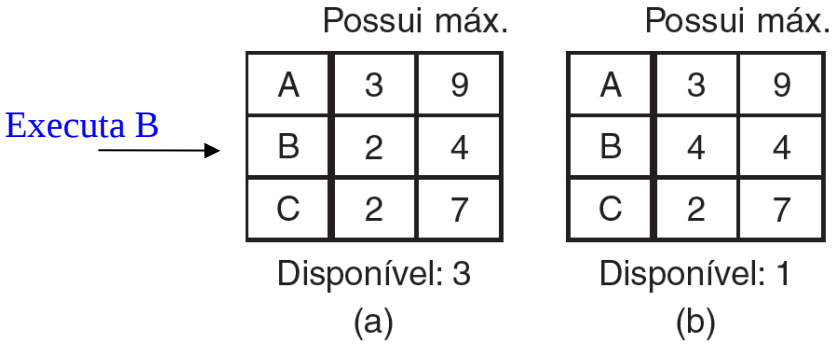


Figura 6.9 Demonstração de que o estado em (a) é seguro.

Evitando impasses

Estados seguros e inseguros

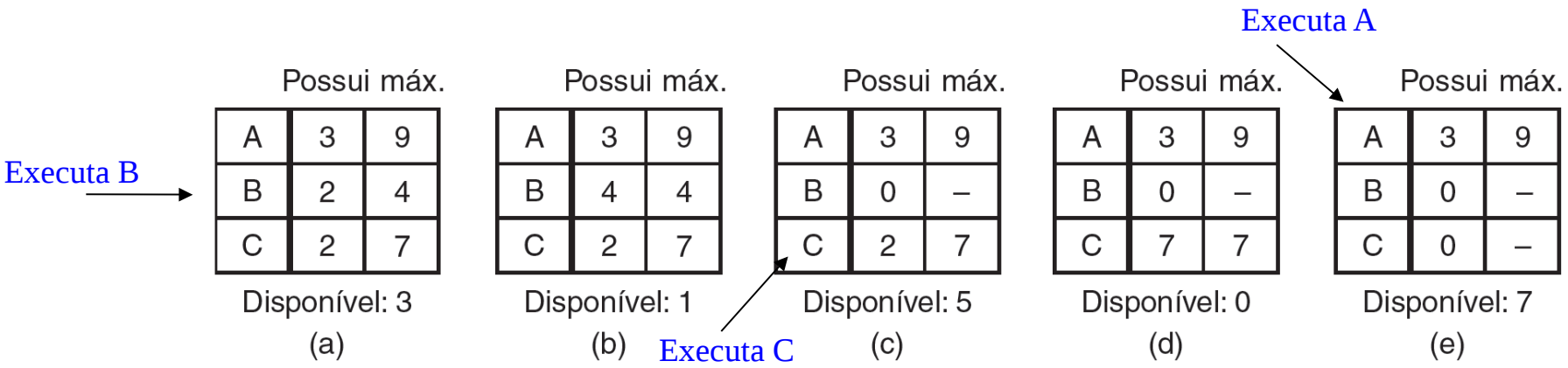


Figura 6.9 Demonstração de que o estado em (a) é seguro.

A sequência B, C e A não levou a um deadlock

Evitando impasses
Estados seguros e inseguros

Mesmo estado inicial

	Possui	máx.
A	3	9
B	2	4
C	2	7

Disponível: 3

(a)

Processo A recebeu mais um recurso

	Possui	máx.
A	4	9
B	2	4
C	2	7

Disponível: 2

(b)

Figura 6.10 Demonstração de que o estado em (b) é inseguro.

O que acontecerá se o processo “A” receber mais um recurso? O processo “B” poderá executar? O que acontecerá no sistema após a execução do processo “B”?

Evitando impasses

Estados seguros e inseguros

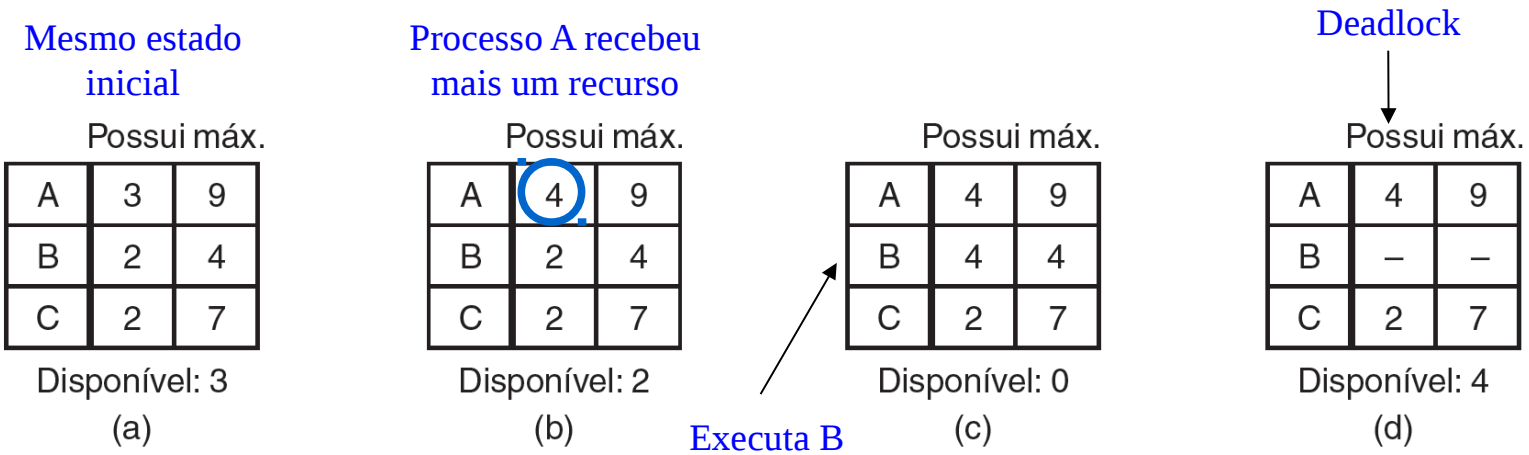


Figura 6.10 Demonstração de que o estado em (b) é inseguro.

Evitando impasses

Estados seguros e inseguros

A transição de A para B levou o sistema de seguro para inseguro.

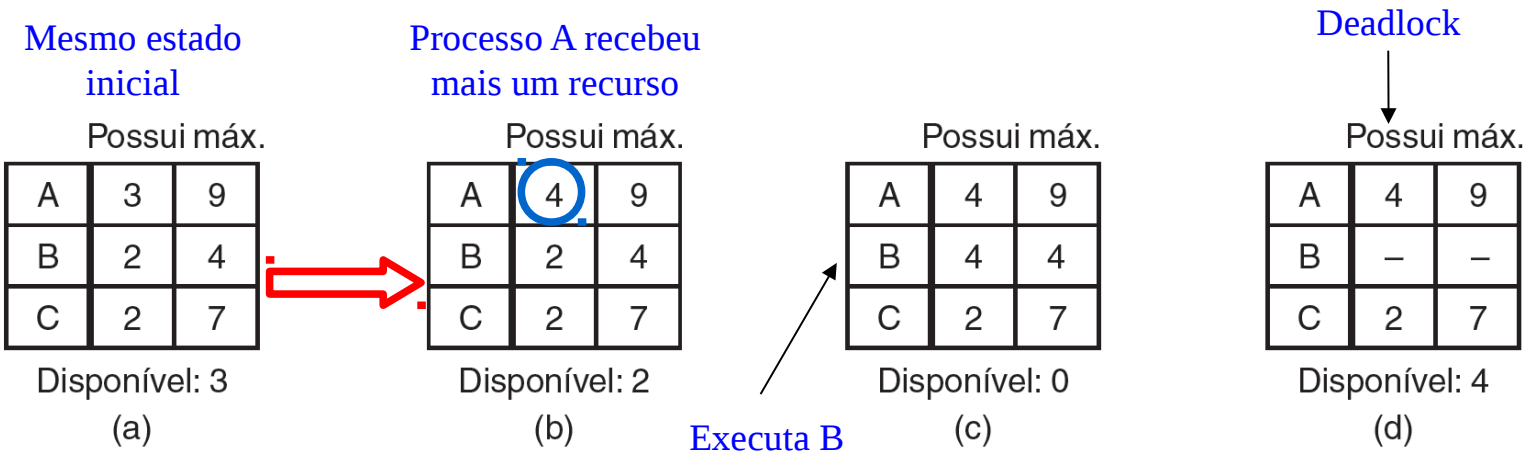


Figura 6.10 Demonstração de que o estado em (b) é inseguro.

Um estado inseguro não significa necessariamente que ocorrerá um impasse pois os processos podem não precisar de todos os recursos e também podem terminar.

Evitando impasses

Algoritmo do banqueiro para múltiplos recursos

Antes da execução, cada processo deve informar quantos recursos de cada classe serão utilizados.

Figura 6.12 O algoritmo do banqueiro com múltiplos recursos.

Quando o sistema está em deadlock? Procure um processo cuja necessidade de recurso seja inferior ou igual aos disponíveis. Se não existir tal processo e todos processos existentes estiverem esperando por mais recursos, então o sistema estará em deadlock (nenhum processo pode ser executado até o fim).

Evitando impasses

Algoritmo do banqueiro para múltiplos recursos

Antes da execução, cada processo deve informar quantos recursos de cada classe serão utilizados.

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

Figura 6.12 O algoritmo do banqueiro com múltiplos recursos.

Suponha que o processo “B” solicitou um scanner, o novo estado do sistema será ...

Evitando impasses

Algoritmo do banqueiro para múltiplos recursos

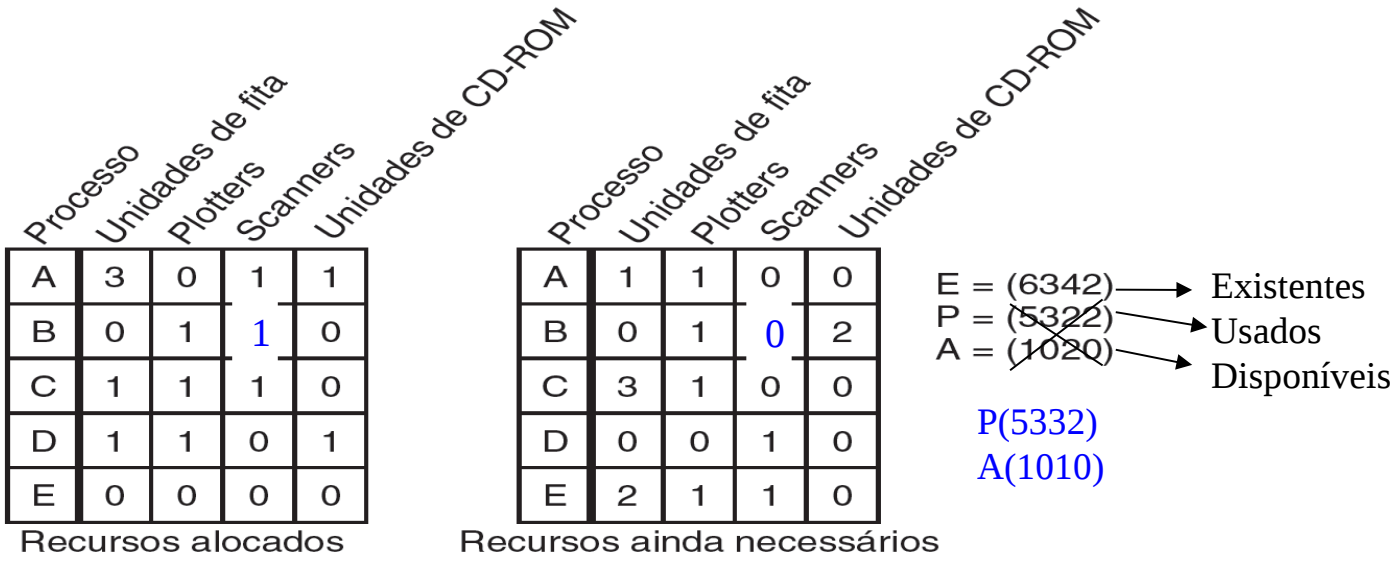


Figura 6.12 O algoritmo do banqueiro com múltiplos recursos.

Estado do sistema após a solicitação de “B”.

Este estado é seguro pois “D” ainda pode terminar.

Evitando impasses

Algoritmo do banqueiro para múltiplos recursos

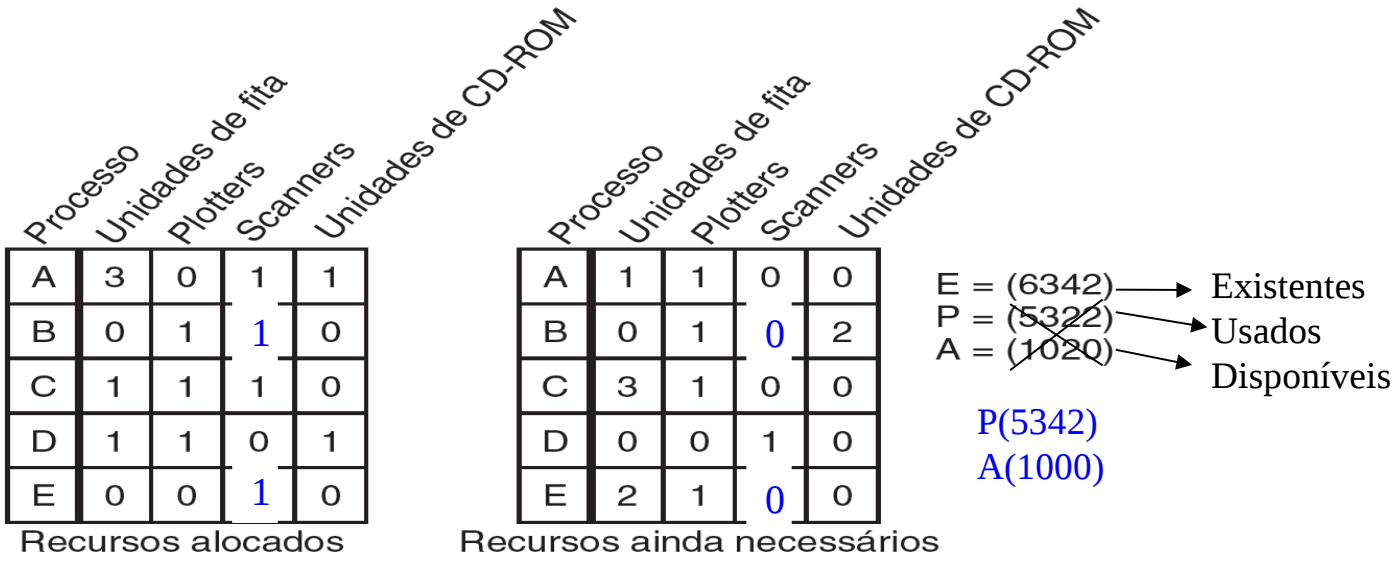


Figura 6.12 O algoritmo do banqueiro com múltiplos recursos.

Porém, se "E" solicitar o último scanner, o sistema ficará em um estado inseguro pois nenhum outro processo poderá terminar com os recursos disponíveis no momento.

A solicitação de "E" deve ser negada para se evitar um possível deadlock.

Evitando impasses

Algoritmo do banqueiro para múltiplos recursos

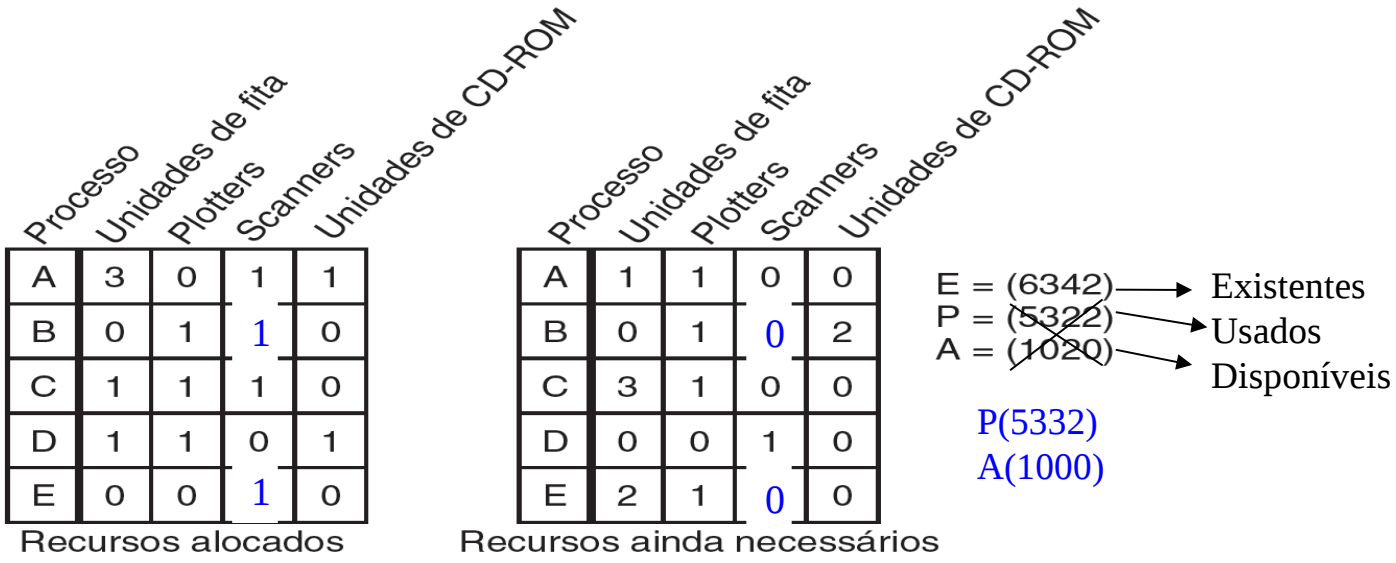


Figura 6.12 O algoritmo do banqueiro com múltiplos recursos.

O problema deste método é que na prática os processos não sabem com antecedência quantos recursos vão precisar. Além disso, o número de recursos disponíveis pode mudar com o tempo (ex. defeito em algum recurso).

Prevenção de impasses

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

Atacando a condição de exclusão mútua

- Sem exclusão mútua não há impasses.

Prevenção de impasses

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

Atacando a condição de posse e espera

- a tarefa somente se inicia após ter posse de todos os recursos.
- Problema: difícil conhecer previamente todos os recursos necessários. Se isso fosse possível, o algoritmo do banqueiro poderia ser usado.

Prevenção de impasses

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

Atacando a condição de preempção

- retira o recurso da tarefa sem que ela libere-o explicitamente.
- Difícil implementação pois pode deixar o recurso em um estado inconsistente.

Prevenção de impasses

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

Atacando a condição de espera circular

- identifica os recursos em ordem numérica.

- as requisições devem ser feitas sempre em ordem (exemplo da transferência entre as contas).

Livelock

- **Livelock**:

bloqueio do recurso usando polling (espera ocupada).

  - O processos continuam executando, mas nunca progridem, pois ambos ficam esperando pela liberação do recurso.

```
void process_A(void) {
        enter_region(&resource_1);
        enter_region(&resource_2);
        use_both_resources( );
        leave_region(&resource_2);
        leave_region(&resource_1);
}

void process_B(void) {
        enter_region(&resource_2);
        enter_region(&resource_1);
        use_both_resources( );
        leave_region(&resource_1);
        leave_region(&resource_2);
}
```

Figura 6.15 A espera ocupada que pode acarretar um livelock.

Estratégias para lidar com impasses

como lidar com impasses?

1) Detecção e recuperação. Deixe os impasses ocorrerem, detecte e recupere.
2) Anulação dinâmica por meio de alocação cuidadosa de recursos.
3) Prevenção, negando estruturalmente uma das quatro condições necessárias.
4) Ignorar o problema!!

Fim do capítulo 6 (Tanenbaum)