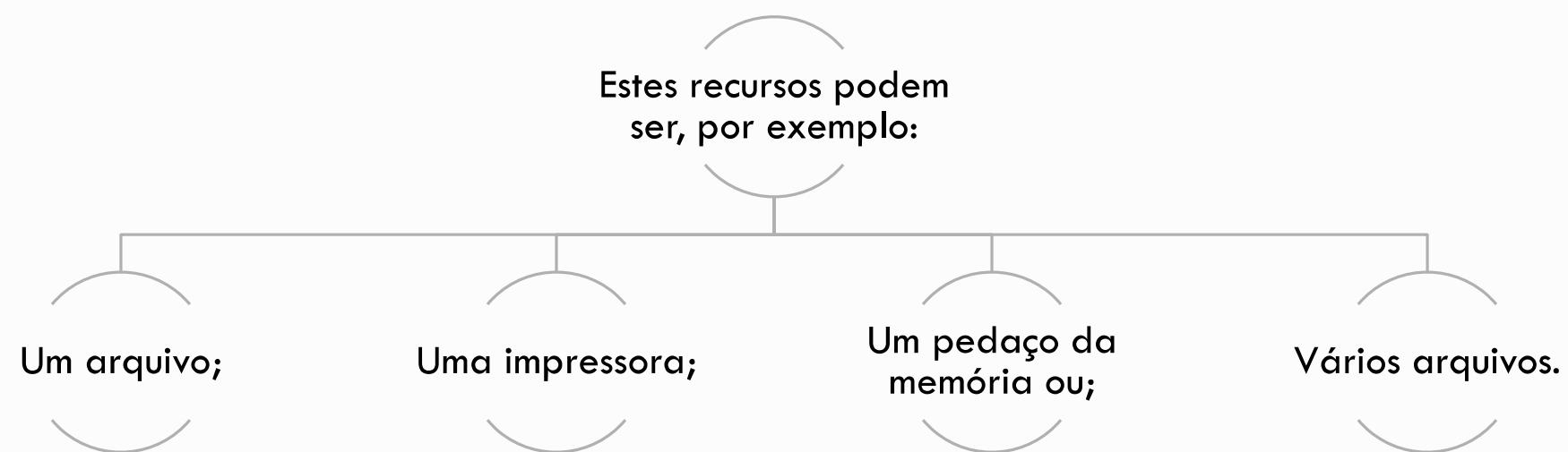


Sistemas Operacionais

Deadlocks (impasses)

Deadlocks

Os **deadlocks** acontecem quando um conjunto de processos/threads disputam por um ou mais recursos.



Deadlocks

Se não existe o controle de acessos ao recurso comum, danos irreparáveis podem ocorrer.

Exemplos:

- Arquivos corrompidos.
- Perda de documentos na fila da impressora.
- Erros lógicos em programas de usuário (chave primária incremental em BD, por exemplo).
- Inconsistência nos dados.

Deadlocks

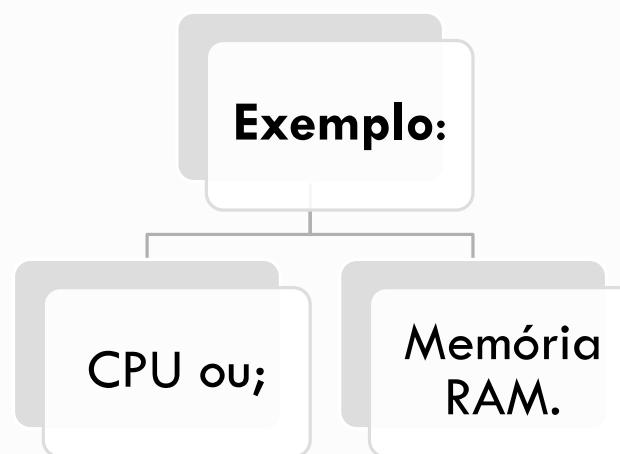
Definição formal:

“Um conjunto de processos/threads estará em situação de deadlock se todo processo/threads pertencente ao conjunto estiver esperando por um evento que somente um outro processo/threads desse conjunto poderá fazer acontecer.”



Recursos Preemptíveis e Não Preemptíveis

Um recurso **preemptível** é aquele que pode ser retirado do processo proprietário sem nenhum prejuízo.



Recursos Preemptíveis e Não Preemptíveis

Um recurso **não preemptível**, ao contrário, é aquele que não pode ser retirado do atual processo proprietário sem que a computação apresente falha.



Exemplo:

Se um processo começou a gravar um CD/DVD, fornecer o acesso da gravadora a outro processo gera um CD/DVD com erros.

Deadlocks envolvem necessariamente recursos não preemptíveis.

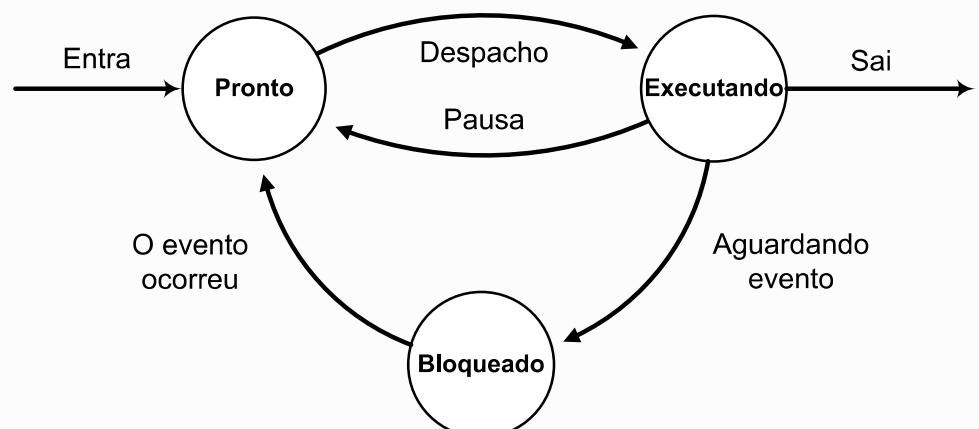
Recursos Preemptíveis e Não Preemptíveis

Os eventos necessários para utilização de recursos não preemptíveis são:

Requisitar o recurso.

Usar o recurso.

Liberar o recurso.



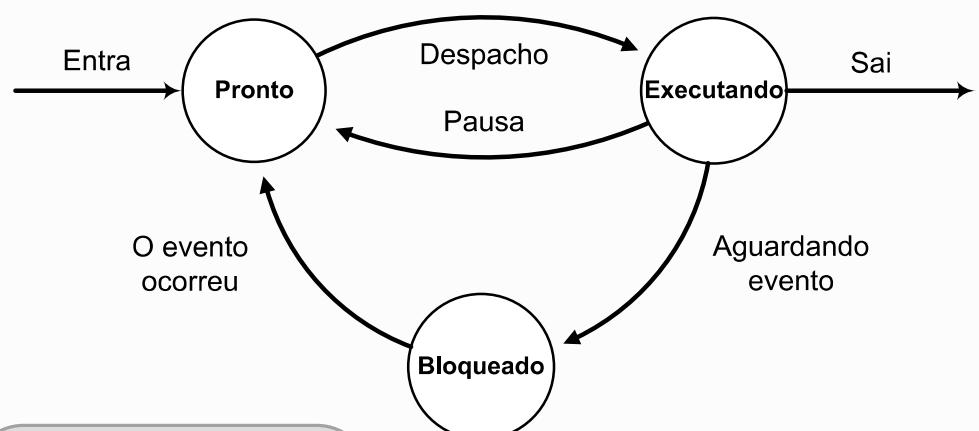
Recursos Preemptíveis e Não Preemptíveis

Os eventos necessários para utilização de recursos não preemptíveis são:

Requisitar o recurso.

Usar o recurso.

Liberar o recurso.



Se o recurso não estiver disponível, o processo é obrigado a esperar (dormindo – estado bloqueado);

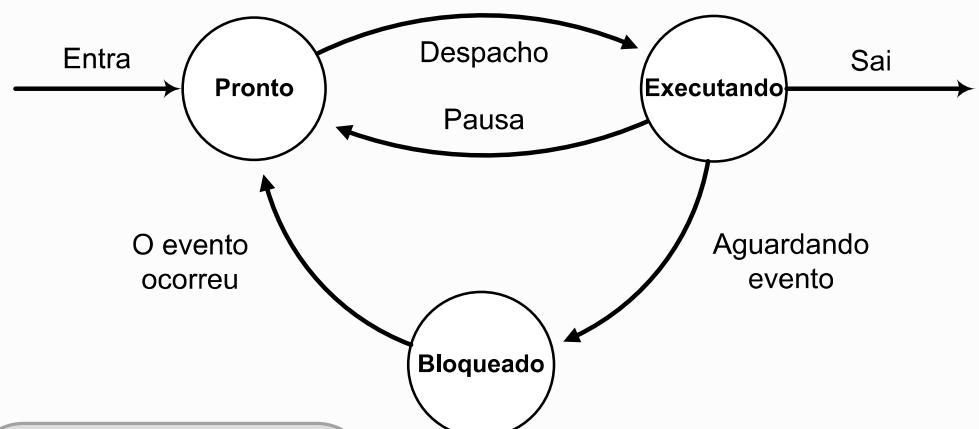
Recursos Preemptíveis e Não Preemptíveis

Os eventos necessários para utilização de recursos não preemptíveis são:

Requisitar o recurso.

Usar o recurso.

Liberar o recurso.



Após a confirmação de acesso individual, o processo pode efetuar suas operações sobre o recurso.

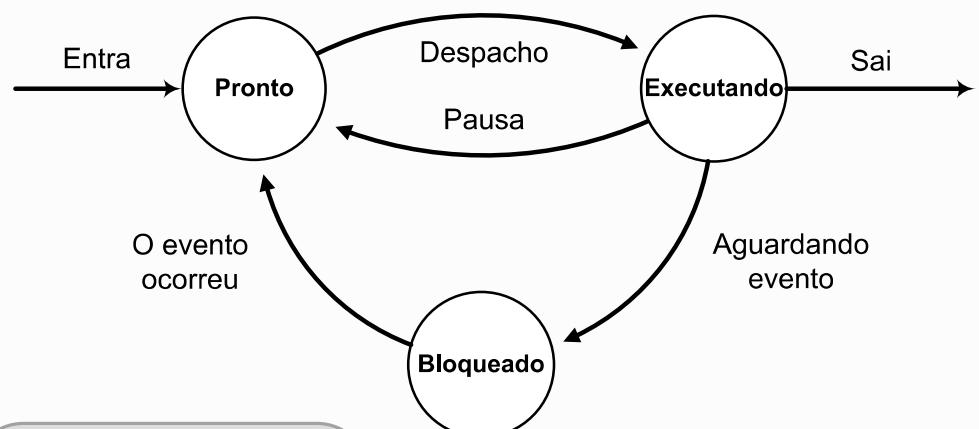
Recursos Preemptíveis e Não Preemptíveis

Os eventos necessários para utilização de recursos não preemptíveis são:

Requisitar o recurso.

Usar o recurso.

Liberar o recurso.



Após a utilizar o recurso, o processo deve liberar o acesso, para outros processos concorrentes, e avisar tais processos da liberação

Recursos Preemptíveis e Não Preemptíveis

Os eventos necessários para utilização de recursos não preemptíveis são:

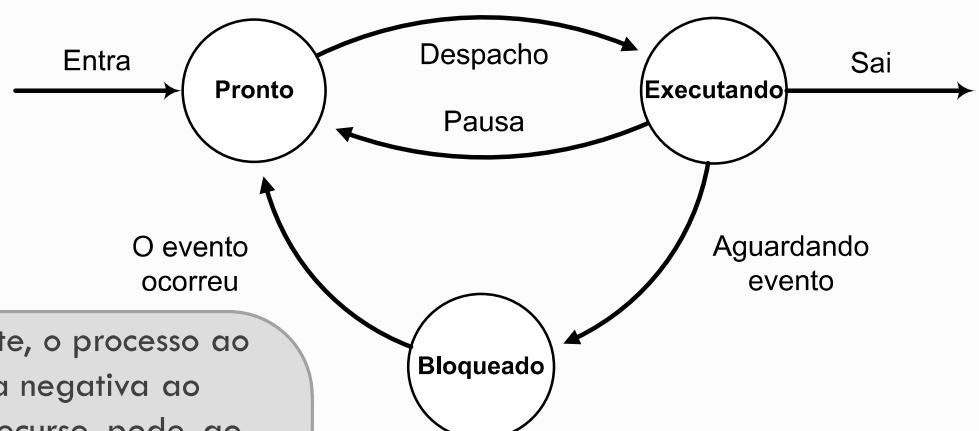
Requisitar o recurso.

Usar o recurso.

Liberar o recurso.

Alternativamente, o processo ao receber uma negativa ao requisitar um recurso, pode, ao invés de se bloquear, suspender sua execução por um determinado período.
Exemplo: `sleep(100)`.

Logo depois, é necessário verificar novamente se o recurso foi liberado.



Recursos Preemptíveis e Não Preemptíveis

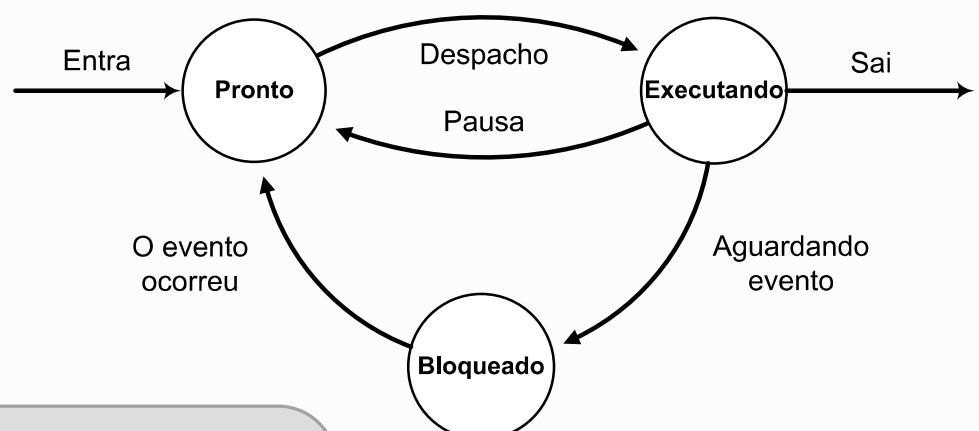
Os eventos necessários para utilização de recursos não preemptíveis são:

Requisitar o recurso.

Usar o recurso.

Liberar o recurso.

Nesta situação (*sleep*), ao acabar de utilizar o recurso, o processo não precisa avisar aos seus concorrentes da liberação da região crítica, pois o S.O. vai acordá-los periodicamente.



Aquisição de Recursos

Como já estudamos, uma forma de garantir a exclusão mútua é com a utilização de semáforos.

```
typedef int semaphore;  
semaphore resource1;  
  
void processA(void) {  
    down(&resource1);  
    useResource1();  
    up(&resource1);  
}
```



Utilização de apenas 1
recurso
não preemptível

Aquisição de Recursos

```
typedef int semaphore;  
semaphore resource1;  
semaphore resource2;  
  
void processA(void) {  
    down(&resource1);  
    down(&resource2);  
    useBothResources();  
    up(&resource2);  
    up(&resource1);  
}  
  
void processB(void) {  
    down(&resource1);  
    down(&resource2);  
    useBothResources();  
    up(&resource2);  
    up(&resource1);  
}
```

Concorrência por dois recursos programa (a)

```
typedef int semaphore;  
semaphore resource1;  
semaphore resource2;  
  
void processA(void) {  
    down(&resource1);  
    down(&resource2);  
    useBothResources();  
    up(&resource2);  
    up(&resource1);  
}  
  
void processB(void) {  
    down(&resource2);  
    down(&resource1);  
    useBothResources();  
    up(&resource1);  
    up(&resource2);  
}
```

Algum dos algoritmos
possui falha visível?

Concorrência por dois recursos programa (b)

Aquisição de Recursos

O Processo A captura o passe do semáforo do *resource1*;

Preempção feita pelo S.O.;

O processo B captura o passe do *resource2*;

B é bloqueado pois o passe do *resource1* está com o Processo A;

Preempção;

O Processo A é bloqueado porque o passe do *resource2* está com o Processo B;

DEADLOCK!

Concorrência por dois recursos programa (a)

```
typedef int semaphore;  
semaphore resource1;  
semaphore resource2;  
  
void processA(void) {  
    down(&resource1);  
    down(&resource2);  
    useBothResources();  
    up(&resource2);  
    up(&resource1);  
}  
  
void processB(void) {  
    down(&resource2);  
    down(&resource1);  
    useBothResources();  
    up(&resource1);  
    up(&resource2);  
}
```

Concorrência por dois recursos programa (b)

Condições de Ocorrência - Deadlocks

Existem 4 condições que devem estar presentes para que um deadlock ocorra:

Condição de exclusão mútua: em cada instante, cada recurso está:

- Sendo utilizado por um único processo, ou
- Está disponível.

Condição de posse e espera: processos que, em um determinado instante, retêm recursos concedidos anteriormente e podem requisitar novos recursos.

Condição de Não preempção: recursos sendo utilizados não podem ser tomados para utilização em outro processo.

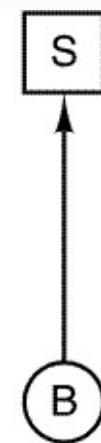
Condição de espera circular: Deve existir um encadeamento circular de dois ou mais processos. Cada um deles encontra-se a espera de um recurso que está sendo utilizado por outro membro desta cadeia.

Modelagem de um *Deadlock*

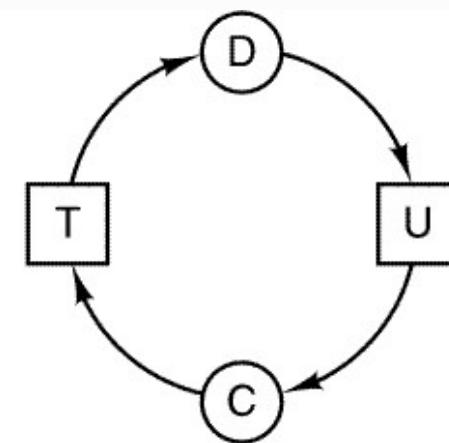
Para facilitar a visualização, podemos utilizar grafos para entender um *deadlock*.



(a)



(b)



(c)

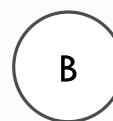
Recurso R alocado
ao processo A

Processo B está
solicitando/esperando
pelo recurso S

Processos C e D estão
em *deadlock* sobre
recursos T e U

Modelagem de um *Deadlock*

Processos:



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

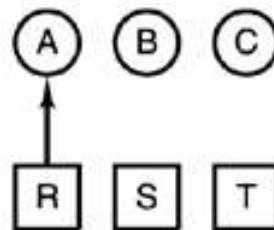
(c)

Recursos:

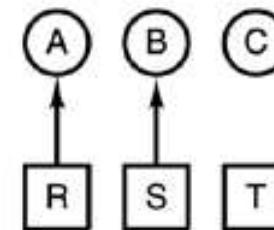


Modelagem de um Deadlock

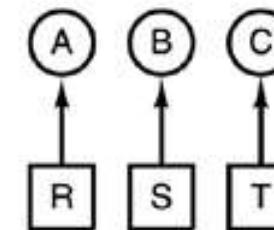
1. A requisita R
2. B requisita S
3. C requisita T
4. A requisita S
5. B requisita T
6. C requisita R
deadlock



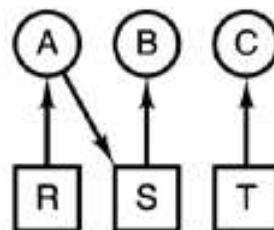
(e)



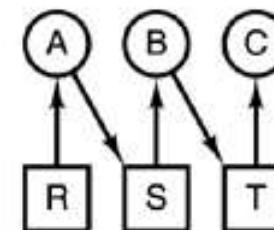
(f)



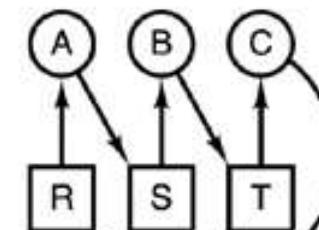
(g)



(h)



(i)

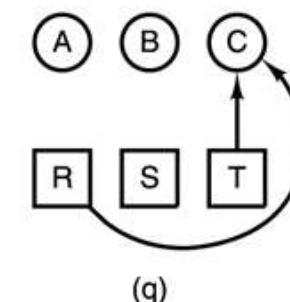
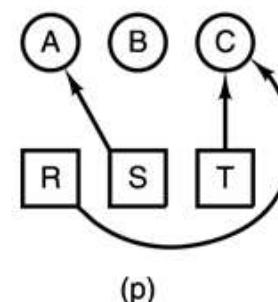
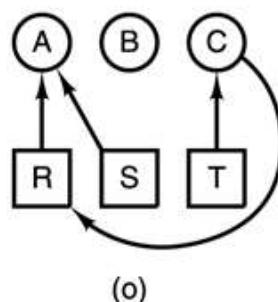
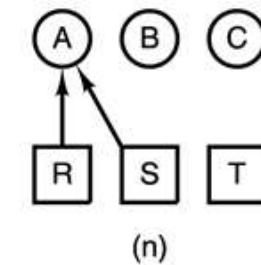
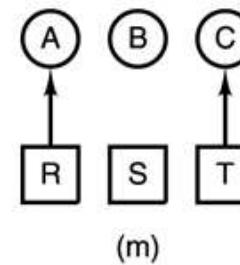
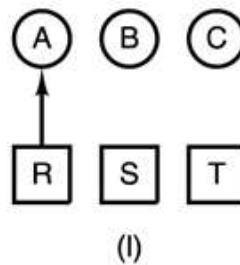


(j)

Modelagem de um Deadlock

Se o Sistema Operacional suspende o processo B, o deadlock não acontece! (Anulação dinâmica)

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



Modelagem de um *Deadlock*

Estratégias para tratar deadlocks

Ignorar por completo o problema

Detecção e recuperação

Anulação dinâmica (alocação cuidadosa de recursos)

Prevenção (negação de uma das quatro condições necessárias)

Ignorar por completo o problema

Algoritmo do Avestruz

Conhecido por **Algoritmo do Avestruz**, é um método (se é que podemos chamá-lo desta forma) mais simples, que consiste em fingir que nada está acontecendo.

Mesmo parecendo um absurdo, esta pode ser uma saída interessante.

- Matemáticos consideram totalmente inaceitável.
- Engenheiros perguntam com que frequência o deadlock ocorre.

Ignorar por completo o problema

Algoritmo do Avestruz

Questionamentos importantes:

- Qual é importância do software que está sendo analisado.
 - Ele pode parar?
- Com que frequência o deadlock ocorre? (em média)
 - Isso está relacionado com a concorrência:
 - Quantas vezes os processos paralelos tentam acessar os recursos em uma unidade de tempo?
 - Se esta concorrência for alta, a chance de deadlock é alta.

Ignorar por completo o problema

Algoritmo do Avestruz

A maioria dos S.O.s, incluindo Unix e Windows, ignoram situações de deadlock:

- Isso porque consideraram que o custo de seu tratamento é alto demais.

Evidentemente, em S.O.s com outros propósitos, existe a inversão deste valor:

- Exemplo: trabalhando com Sistemas Militares.

Ignorar por completo o problema

Algoritmo do Avestruz

Situações:

- Um sistema médico que controla a máquina de bombeia o sangue de um paciente no tratamento da hemodiálise pode parar?
- Um sistema que controla um medidor de temperatura do quarto de um hospital pode parar?
- Um sistema militar que controla um míssil durante seu voo pode parar?



Detecção e Recuperação de deadlocks

Na detecção e recuperação de deadlocks, o sistema não tenta prevenir a ocorrência dos Deadlocks.

O algoritmo se resume a:

 Detectar que um
 deadlock ocorreu.
(Operações em grafos)

 Detectar quais
 processos estão
 envolvidos no deadlock.
(Operações em grafos)

 Corrigir o problema.
(Várias formas)

Detecção e Recuperação de deadlocks

Para detectar que um *deadlock* ocorreu basta saber se o grafo, que representa a interação entre recursos e processos, possui ciclo(s).

Obrigações do S.O.:

Sempre que um processo/recurso é criado, um nó do tipo “circulo”/“quadrado” é inserido no grafo.

Sempre que um processo/recurso é finalizado/removido, seu nó é removido do grafo (incluindo suas arestas).

Sempre que um processo aguarda um recurso, uma aresta do nó do processo é criada em direção ao nó do recurso.

Sempre que um processo tem a posse de um recurso, uma aresta do nó do recurso é criada em direção ao nó do processo.

Detecção e Recuperação de deadlocks

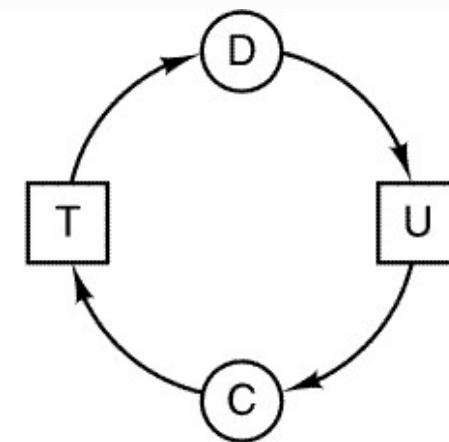
Para facilitar a visualização, podemos utilizar grafos para entender um deadlock.



(a)



(b)



(c)

Recurso R alocado
ao processo A

Processo B está
solicitando/esperando
pelo recurso S

Processos C e D estão
em deadlock sobre
recursos T e U

Detecção e Recuperação de deadlocks

Consideremos um grafo qualquer representando a interação entre processos e recursos.

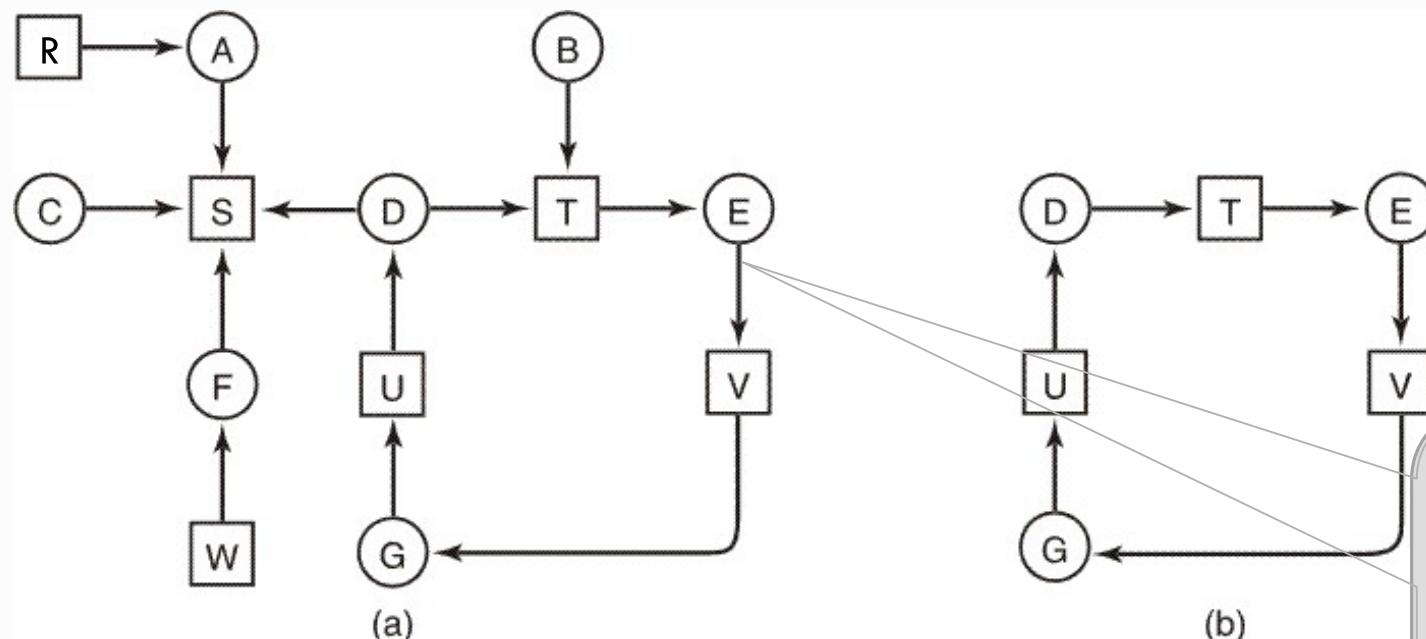
- Um deadlock existe se somente se o grafo possui **ciclo(s)**.
- Todo processo que faz parte do ciclo, está dormindo
 - Sem possibilidade de ser acordado.
- Os outros processos do grafo continuam em situação normal de operação.
 - Estes processos que estão funcionando podem a fazer parte a qualquer momento do ciclo que caracteriza o deadlock.

Detecção e Recuperação de deadlocks

Sobre os processos de A a G e recursos de R a W

- O processo A possui o recurso R e requisita S.
- O processo B requisita T.
- O processo C requisita S.
- O processo D possui o recurso U e requisita S e T.
- O processo E possui o recurso T e requisita V.
- O processo F possui o recurso W e requisita S.
- O processo G possui o recurso V e requisita U.

Detecção e Recuperação de deadlocks



Visualizando com um grafo planar:

Observe a posse e solicitações de recursos

Um ciclo pode ser encontrado dentro do grafo, denotando deadlock.

Detecção e Recuperação de deadlocks

O algoritmo de busca em profundidade pode ser utilizado para detecção de ciclos em grafos.

- Ou seja, pode ser utilizado para detectar se existe um deadlock na interação entre processos/recursos.

Para detalhes, consulte o livro: *Algoritmos, Teoria e Prática*, de Thomas Cormen, 2^a edição, página 430.

- O algoritmo do livro nesta edição possui um pequeno erro. Cuidado – Aviso do professor Humberto.

Detecção e Recuperação de deadlocks

$DFS(G)$

1 para cada vértice $u \leftarrow V[G]$

2 $cor[u] \leftarrow BRANCO$

3 $tempo \leftarrow 0$

4 para cada vértice $u \in V[G]$

5 se $cor[u] = BRANCO$

6 $DFS - VISIT(u)$

É necessário realizar uma pequena adaptação para encontrar Deadlocks utilizando Busca em profundidade.

$DFS - VISIT(u)$

1 $cor[u] \leftarrow CINZA$

2 $tempo \leftarrow tempo + 1$

3 $d[u] \leftarrow tempo$

4 para cada vértice $v \in Adj(u)$

5 se $cor[v] = BRANCO$

6 $DFS - VISIT(v)$

7 $cor[u] \leftarrow PRETO$

8 $f[u] \leftarrow tempo \leftarrow (tempo + 1)$

Detecção e Recuperação de deadlocks

Busca em Profundidade – adaptado ao problema do deadlock

$DFS(G)$

1 para cada vértice $u \leftarrow V[G]$

2 $cor[u] \leftarrow BRANCO$

3 $tempo \leftarrow 0$

4 para cada vértice $u \in V[G]$

5 se $cor[u] = BRANCO$

6 $DFS - VISIT(u)$

$DFS - VISIT(u)$

1 $cor[u] \leftarrow CINZA$

2 $tempo \leftarrow tempo + 1$

3 $d[u] \leftarrow tempo$

4 para cada vértice $v \in Adj(u)$

5 se $cor[v] = CINZA$

6 CICLO ENCONTRADO;

7 se $cor[v] = BRANCO$

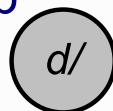
8 $DFS - VISIT(v)$

9 $cor[u] \leftarrow PRETO$

10 $f[u] \leftarrow tempo \leftarrow (tempo + 1)$

Detecção e Recuperação de deadlocks

Busca em Profundidade

- a  Vértice desconhecido
- b  Vértice encontrado
- c  Vértice encontrado, com fecho positivo totalmente visitado

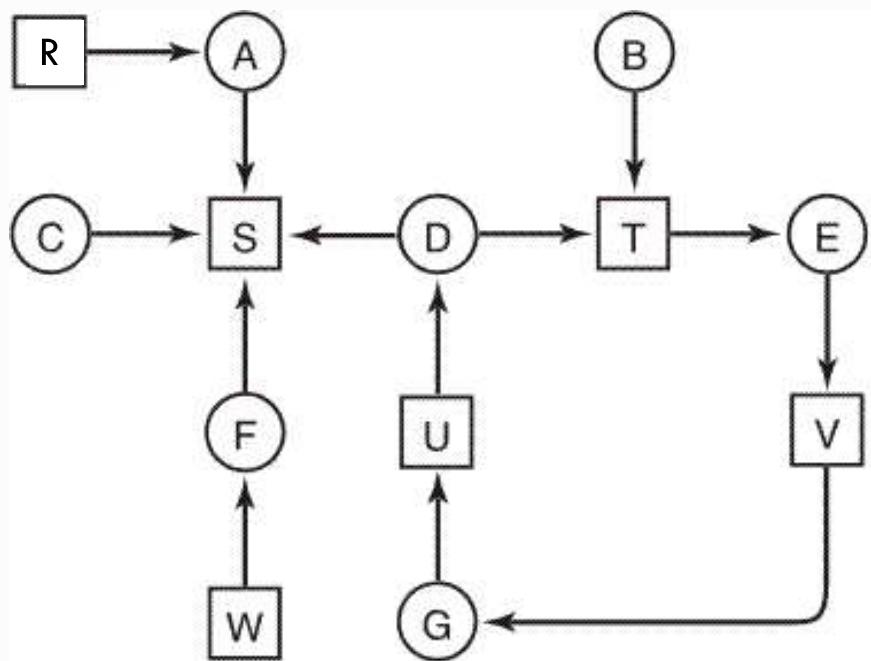
Legenda

d: marcador do instante que o vértice c foi encontrado;

f: marcador do instante que os fecho transitivo do vértice c foi totalmente visitado.

Detecção e Recuperação de deadlocks

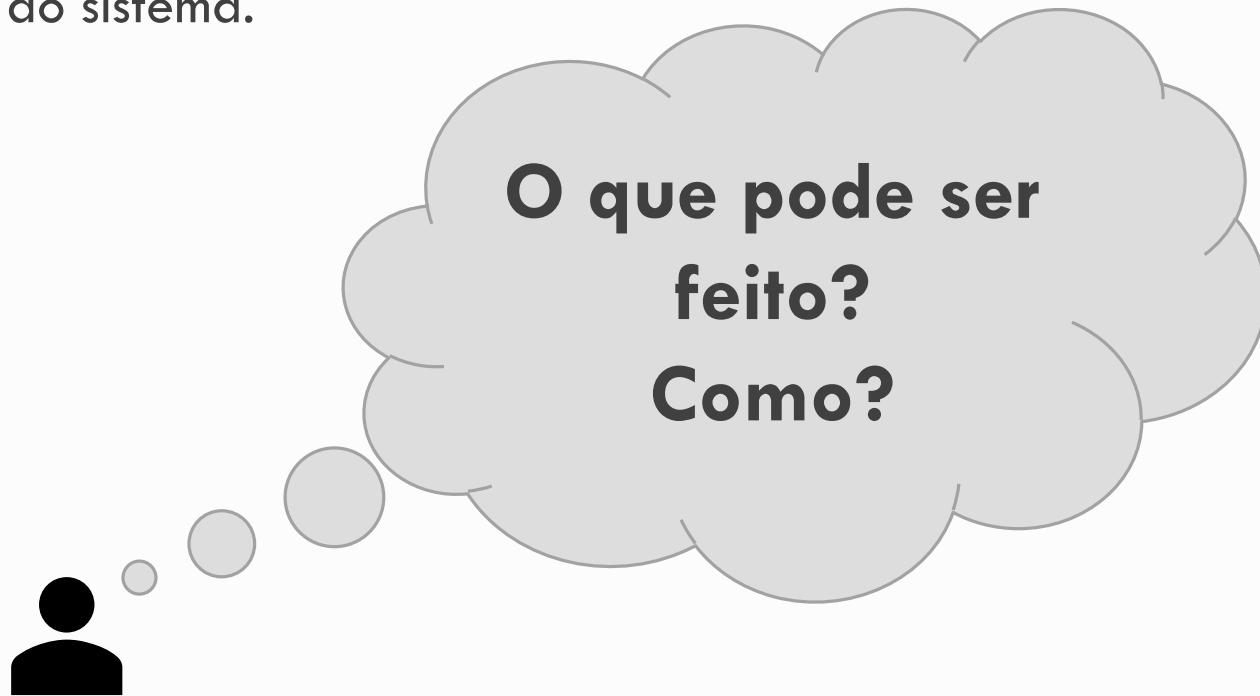
Busca em Profundidade para detectar deadlocks



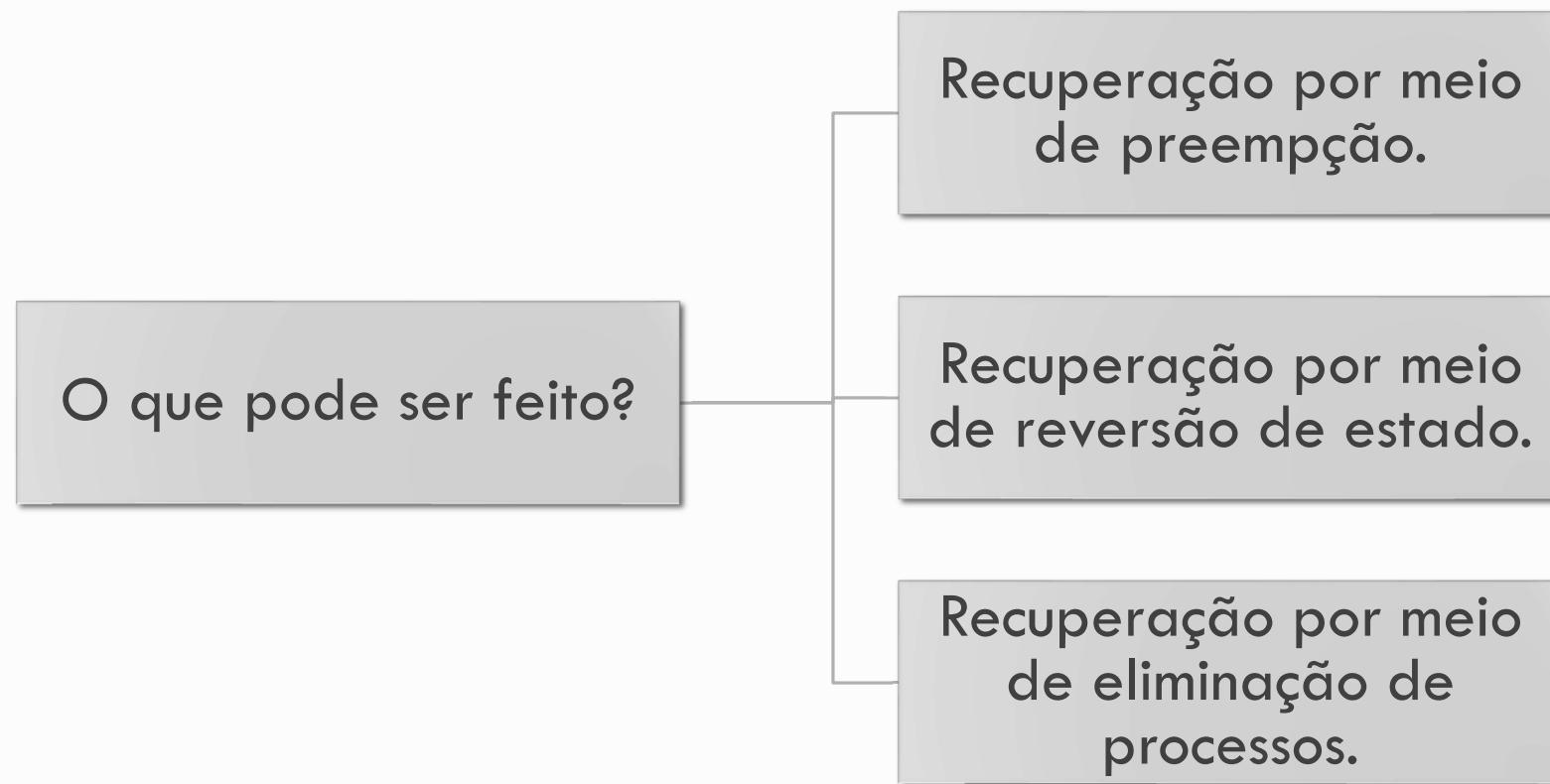
Atividade extra.

Detecção e Recuperação de deadlocks

Após a detecção de um deadlock, o S.O. deve restabelecer a normalidade na operação do sistema.



Detecção e Recuperação de deadlocks



Detecção e Recuperação de deadlocks

Recuperação por meio de preempção:

Em alguns casos, pode ser possível tomar provisoriamente um recurso de seu proprietário atual para fornecê-lo a outro processo.

Por exemplo:

Se a impressora R for tomada do processo A, a impressora deve parar a impressão de A, empilhar o conteúdo de sua bandeja em outro compartimento, para iniciar sua operação com seu novo processo proprietário. Ao final, deve desempilhar as folhas da impressão de A, e continuar sua impressão.

Portanto, a recuperação por meio de preempção está diretamente relacionada com a natureza do recurso.

Visando eliminar o deadlock, O S.O. deveria conhecer previamente todos os recursos para efetuar, se possível, tal operação.

É possível?

Detecção e Recuperação de deadlocks

Recuperação por meio de reversão de estado:

Para entender este algoritmo, imagine, “via grafo”, que o processo possui um determinado estado em cada instante de tempo.

O S.O. então controlaria uma Máquina de Estados Finitos.

Sempre, antes de requisitar um recurso, o processo deve armazenar em um arquivo temporário seu estado atual.

É uma técnica semelhante a utilizada pelo MS-Windows para restaurar o sistema.

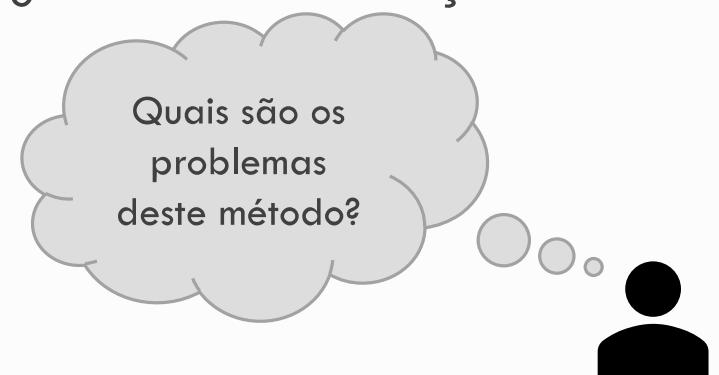
Ele mantém o estado atual dos arquivos (para o usuário não perder nenhum arquivo recentemente alterado) e restaura os arquivos de configuração do S.O. através de uma reversão de estados em uma máquina de estados finitos.

Detecção e Recuperação de deadlocks

Recuperação por meio de reversão de estado:

Sempre que um deadlock ocorrer, o S.O. escolhe um processo para “regredir” no tempo, assumindo seu estado anterior ao pedido do recurso.

Com isso, seu processo concorrente pode utilizar o recurso sem restrições, e o processo que regrediu entra no estado bloqueado, aguardando a liberação do recurso.



Detecção e Recuperação de deadlocks

Recuperação por meio de reversão de estado:

O custo para guardar as imagens pode ser alto demais:

- Processo que ocupa muita memória, ou;
- Processo que efetua muitas requisições invocará muitas vezes a função de imagem no disco.

Processos que interagem com usuários finais:

- Imagine perder tudo que digitou no editor de textos nos últimos 10 minutos.

Processos que interagem com outros processos:

- Exemplo (comunicação em uma rede): Se um processo regredir, a comunicação pode ser totalmente perdida. Go-back-n, Selective Reject.
- Depende da natureza da comunicação.

Detecção e Recuperação de deadlocks

Recuperação por meio de eliminação de processos:

Método radical e extremista – sempre que um ciclo é identificado no grafo, o S.O. mata um processo.

Se o ciclo ainda existir, ele mata outro, até quebrar o ciclo, eliminando consequentemente o deadlock.

O S.O. tem que ter o cuidado de não matar processos que não pertencem ao ciclo.

Anulação Dinâmica

Alocação individual de recursos

À medida que o processo necessita.
Soluções também utilizam matrizes.

Escalonamento cuidadoso tem um alto custo computacional.

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

Inseguros.

Anulação Dinâmica

Estados seguros:

Não provocam *deadlocks* e há uma maneira de atender a todas as requisições pendentes finalizando normalmente todos os processos.

Estados inseguros:

Podem provocar *deadlocks*, mas não necessariamente provocam.

Prevenção de Deadlock

Existem 4 condições que devem estar presentes para que um deadlock ocorra:

Condição de exclusão mútua: em cada instante, cada recurso está:

- (i) sendo utilizado por um único processo, ou
- (ii) está disponível.

Condição de posse e espera: processos que, em um determinado instante, retêm recursos concedidos anteriormente e podem requisitar novos recursos.

Condição de não preempção: Recursos sendo utilizados não podem ser tomados para utilização em outro processo.

Condição de espera circular: Deve existir um encadeamento circular de dois ou mais processos. Cada um deles encontra-se a espera de um recursos que está sendo utilizado por outro membro desta cadeia.

Prevenção de Deadlock

Se nunca acontecer de um recurso ser alocado exclusivamente a um único processo, nunca teremos deadlock.

Obviamente, que para alguns recursos a quebra desta condição levaria a falhas.

Exemplo: Utilização de uma impressora por 3 editores de texto ao mesmo tempo.

Repare que, na prática, vários editores solicitam impressão, mas as requisições são gerenciadas pelo daemon de impressão, através da técnica de *spooling*.

O spooler de impressão é quem garante o acesso correto a impressora (garante a exclusão mútua no caso).

Prevenção de Deadlock

Se pudermos impedir que processos que já mantêm recursos esperem por mais recursos, seremos capazes de eliminar os deadlocks.

Veja o caso:

Suponha que os filósofos (do problema do Jantar dos Filósofos) pegam os garfos do lado esquerdo, e não soltam até comer. Se todos pegarem o garfo do lado esquerdo, todos os filósofos irão dormir ao tentar pegar os garfos do lado direito, porque todos os garfos estão ocupados.

Uma solução seria o processo pegar todos os recursos que vai precisar de uma só vez.

Suponha uma thread que manipula uma região de memória, imprime um documento na impressora, e após a impressão ele deve alterar novamente o endereço de memória.

Com a solução proposta, a região crítica da memória ficaria travada até todo o processo terminar, não podendo ser alterada por outra thread.

Prevenção de Deadlock

Condição de posse e espera

Desvantagens:

Este método exige do programador o conhecimento de todos os recursos que precisam ser utilizados antes da execução do processo/thread.

Não utiliza os recursos de forma otimizada.

Vantagem:

Evita os deadlocks.

Prevenção de Deadlock

Se um processo tomar a força um recurso de outro processo que tinha posse do mesmo, evitamos os *deadlocks*.

Ou seja, nenhum processo precisa aguardar a liberação de recursos.

Parecido com a primeira estratégia (atacando a exclusão mútua) . Mas aqui é considerado que o recurso é tomado a força. E o processo que tinha a posse do recurso, termina sua utilização.

Na primeira estratégia é considerado que o recurso é utilizado ao mesmo tempo por vários processos.

Prevenção de Deadlock

Condição de
não preempção

Desvantagem:



Seu processo não tem garantia na utilização dos recursos. A qualquer instante o recurso pode lhe ser tomado.

Vantagem:



Acaba com deadlock.

Prevenção de Deadlock

Este é o conceito do ciclo no grafo. Se não existir ciclo, não existe deadlock.

Forma simples de atacar a espera circular:

Se existir a restrição de que um processo pode apenas ter a posse de um recurso, a cada instante de tempo, este grafo nunca possuirá ciclos.

Imagine que um processo precisa copiar um arquivo de um disco para uma impressora, o arquivo é grande e não cabe inteiro na memória.

Ele pode bloquear o disco, copiar o máximo que a memória suporta. Liberar o disco, bloquear a impressora, imprimir esta parte, e repetir este processo o números de vezes que for necessário, até imprimir todo o arquivo.

Quais são os problemas desta abordagem?