Sistemas Operativos 2

2021/22

Deadlocks

1

Tópicos

Conceitos principais

Prevenção e evitamento

Detecção e recuperação

Algoritmos

Bibliografia específica para este capítulo:

- *Operating Systems Concepts*, 5th Ed.; Silberschatz & Galvin – Capítulo 7
- *Operating Systems: Internals and Design Principles*; William Stallings – Capítulo 6

2

Deadlocks – Conceitos

Definição rápida

- Bloqueio permanente de dois ou mais processos.

Definição geral

- Um conjunto de processos está numa situação de deadlock quando cada um dos processos desse conjunto está à espera da ocorrência de um evento que só poderá ser desencadeado por um dos outros processos do conjunto

Situação habitual

- Competição por recursos
- O evento corresponde à libertação de um recurso

Deadlocks – Conceitos

Do ponto de vista dos processos, o problema reside no facto de, normalmente, o algoritmo apenas avançar num sentido:

1 – Pedido do recurso
- Se o pedido não puder ser satisfeito imediatamente, o processo terá que esperar (bloqueando-se) até que o recurso esteja disponível.

  Mecanismo de espera – Implementado pelo SO (ex. semáforos)

2 – Utilização do recurso

3 – Libertação do recurso

Dois processos a seguirem a mesma lógica podem entrar em conflito se utilizarem os mesmos recursos

Deadlocks – Conceitos – Condições necessárias

Existem 4 condições necessárias para a ocorrência de deadlocks

1 – Exclusão mútua

Significa que os recursos não são partilháveis

- Se ocorrer um pedido de um recurso não partilhável e este estiver ocupado, o processo que efectuou o pedido fica em espera.

2 – Detenção e espera (*hold and wait*)

- Consiste em manter um recurso enquanto se espera pela libertação de outro recurso detido por outro processo

Deadlocks – Conceitos – Condições necessárias

3 – Recursos não preemptíveis

(atenção: "preempção"no contexto de recursos e não de processos)

- Os recursos só são libertados por iniciativa própria e voluntária do processo que o detém (eventualmente após a tarefa que levantou a sua necessidade estar concluída)

4 – Espera circular

- Consiste na existência de um conjunto $\{ P_1, P_2, \ldots P_n\}$ em que P_1 está à espera de um recurso detido por P_2; P_2 está à espera de um recurso detido por P_3; ... e P_n está à espera de um recurso detido por P_1

➔ Num dado sistema, é necessário que todas estas quatro condições sejam verdadeiras para que possa ocorrer um deadlock

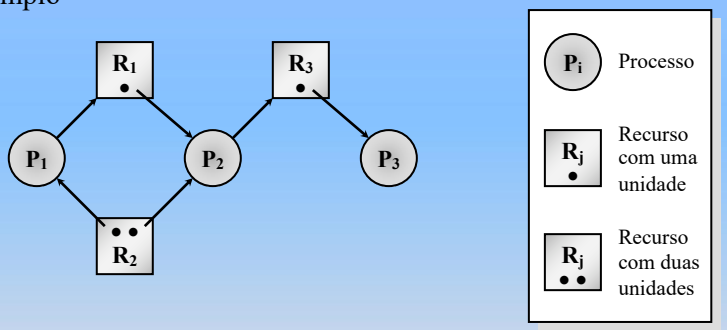
Deadlocks – Grafo de atribuição de recursos

Grafo de atribuição de recursos

- Ferramenta para descrição da situação de processos e recursos
- É um grafo dirigido
- Conjunto de vértices
  - R – Recursos no sistema
  - P – Processos no sistema
- Arcos
  - De requisição $P_i \rightarrow R_j$
    - O Processo P_i requisitou o recurso R_j e está de momento à espera dele
  - De atribuição $R_i \rightarrow P_j$
    - O recurso R_i (ou uma unidade dele) está atribuído ao processo P_j

Deadlocks – Grafos de atribuição

Exemplo



As várias unidades do mesmo recurso são indistintas

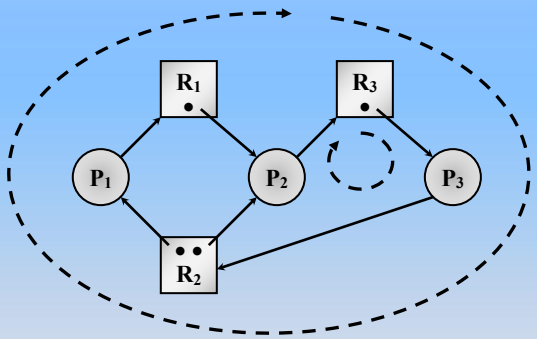
Deadlocks – Grafos de atribuição

Detecção de deadlocks através do grafo de atribuição

- Se o grafo tiver ciclos
  - Pode existir um deadlock
  - Se só existir uma unidade de cada um dos recursos envolvidos, então existe mesmo um deadlock
- Se o grafo de não contiver ciclos
  - Não existem deadlocks no sistema

Deadlocks – Grafos de atribuição

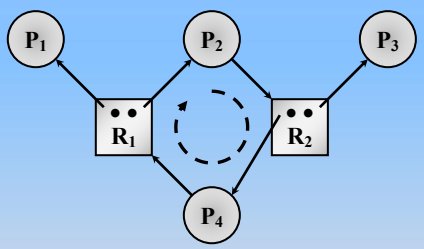
Exemplo de um grafo com ciclos com uma situação de deadlock



Existe deadlock

Deadlocks – Grafos de atribuição

Exemplo de um grafo com ciclos mas sem deadlocks



Não existe nenhum deadlock

Deadlocks

Tratamento de deadlocks

Estratégias para lidar com deadlocks

- **Prevenção / evitamento**
  Tentar impedir que ocorram deadlocks
- **Detecção e recuperação**
  Conseguir detectar e recuperar quando ocorrem deadlocks
- **Ignorar**
  Assumir que nunca acontecem, ou, assumir que não vale a pena impedir que aconteçam, detectar nem recuperar

Deadlocks – Prevenção e Evitamento

Prevenção

- A estratégia consiste em fazer com que **pelo menos uma das quatro condições necessárias ao deadlock nunca se verifique**

Evitamento

- A estratégia consiste em **fornecer ao SO informação acerca da utilização de recursos** por parte de cada processo durante toda a sua execução futura.
- Com base nesta informação, **o SO pode prever situações em que podem vir a verificar-se situações de deadlocks e evitá-las**, recusando alguns pedidos mesmo que haja nesse instante recursos suficientes para satisfazer esses pedidos

Deadlocks – Detecção e Recuperação

Quando um sistema não faz nem prevenção nem evitamento

- Podem ocorrer deadlocks

Torna-se necessário:

- Um algoritmo para analisar o estado do sistema e detectar ocorrências de deadlocks
  = Algoritmo de detecção
- Um algoritmo para recuperar situações de deadlocks (desfazer o deadlock)
  = Algoritmo de recuperação

Deadlocks – Estratégia de ignorar deadlocks

Quando um sistema não implementa

- Nem prevenção nem evitamento
- Nem detecção nem recuperação

➔Podem ocorrer deadlocks e estes nunca serão detectados nem recuperados

Consequências para o sistema

- Vai ocorrer uma degradação do sistema
  - Existem recursos que ficam permanentemente bloqueados (=indisponíveis)
  - Processos importantes podem ficar "congelados"
- A situação tende a alastrar a todo o sistema
  - Mais processos vão sendo adicionados ao conjunto que forma o deadlock
- Os processos (eventualmente o sistema) têm que ser terminados e recomeçados

Deadlocks

Quando é que a estratégia de ignorar é viável

- Em sistemas onde as os deadlocks são raros e custam menos (em termos de produtividade) que a sua prevenção ou recuperação

Por exemplo:

- Se num dado sistema for mais frequente o recomeço (*reboot*) da máquina por causas diversas que devido à necessária pela degradação por deadlocks, poderá compensar ignorar os deadlocks

Deadlocks – Prevenção

1 – Impedir a condição "Exclusão mútua"

Existem dois tipos de recursos

- Partilháveis
  - Estes não levam a deadlocks pois o facto de um processo o deter não impede que outros o obtenham
  - → Não existem deadlocks por aqui = problema resolvido
- Não partilháveis
  - Exemplos: impressora, tape
  - Não se pode atribuir um recurso não partilhável a dois processos ao mesmo tempo – As operações efectuadas deixariam de ser consistentes
  - A característica partilhável / não-partilhável dos recursos depende da natureza dos recursos e não dos processos

Geralmente não é viável a prevenção de deadlocks pela negação da condição "exclusão mútua"

Deadlocks – Prevenção

2 – Impedir a condição "Detenção e espera"

Objectivo:

- Impedir que um processo que esteja bloqueado à espera de um recurso mantenha outros recursos, uma vez que não os pode utilizar
- Libertando esses recursos, outros processos poderão prosseguir, eventualmente libertando os recursos que o primeiro processo necessita para prosseguir.

Estratégia 1

Obrigar os processos a pedir todos os recursos que vão necessitar no inicio da execução.

- Ou os obtêm todos, ou não obtêm nenhum.
- Pode ser conseguido através de informação contida em ficheiros executáveis de formato específico.

Deadlocks – Prevenção

2 – Impedir a condição "Detenção e espera"

Estratégia 2

Só permitir a atribuição de recursos a um processo se este não detiver nenhum no momento em que faz o pedido.

Um processo que queira manter um recurso que já tem e obter outro terá que libertar o que já tem e voltar a pedi-lo juntamente com o novo recurso.

- Pode ser visto como uma variante da primeira estratégia (na medida em que pede tudo de uma só vez)
- Mais flexível pois não obriga à imobilização dos recursos ao longo de toda a execução do processo
- Os processos podem pedir mais do que um recurso, mas têm de o fazer de uma forma atómica (ou todos, ou nenhum)
- Aumenta a complexidade na programação

Deadlocks – Prevenção

Desvantagens da prevenção pela negação da condição "Detenção e espera"

- Ambas as estratégias aumentam a possibilidade de *starvation*
- Diminuição da utilização dos recursos
  - Leva à diminuição da eficiência do sistema

Exemplo

- Processo que copia um ficheiro de uma tape para o disco, ordenando-o, e de seguida o copia do disco para a impressora.

  Recursos envolvidos: tape, disco, impressora

  Sequência de operações:

  1 – Requisita tape e disco
  2 – Copia o ficheiro da tape para disco, ordenando-o
  3 – Liberta tape e disco
  4 – Requisita disco e impressora
  5 – Copia o ficheiro do disco para a impressora
  6 – Liberta disco e impressora

Deadlocks – Prevenção

3 – Impedir a condição "Recursos não preemptíveis"

Estratégia 1

Quando um processo requisitar um recurso que não pode obter de imediato fica bloqueado à espera desse recurso e para além disso perde todos os outros recursos que já detinha, ficando também à espera deles.

O processo só se voltará a executar quando puder requisitar todos os recursos (os que já tinha + o que pediu em último lugar)

Esta estratégia pode ser implementada a nível de SO de forma transparente para o processo

Pode não ser seguro retirar alguns recursos ao processo (exemplo: impressora a meio de uma listagem)

Deadlocks – Prevenção

3 – Impedir a condição "Recursos não preemptíveis"

Estratégia 2 (menos "drástica")

Quando um processo requisitar um recurso que não pode obter de imediato, é averiguado se esse recurso é detido por um processo que está também bloqueado.

Se sim, o recurso passa para o processo que acabou de o pedir. Uma vez que o outro processo já estava bloqueado, não foi muito prejudicado, ficando agora à espera de mais um recurso.

Se não, o processo que fez o pedido fica bloqueado, podendo perder alguns dos recursos que detém (se vierem a ser precisos por outros processos).

A preempção de recursos só é aplicável a um sub-conjunto dos recursos, nomeadamente àqueles cuja coerência das operações efectuadas não é posta em perigo com a re-atribuição do recurso.

Deadlocks – Prevenção

4 – Impedir a condição "Espera circular"

Estratégia

Impor a todos os processos sempre a mesma ordem de requisição dos recursos

Implementação

Numeram-se todos os recursos por ordem crescente levando em conta eventuais recursos com unidades múltiplas.

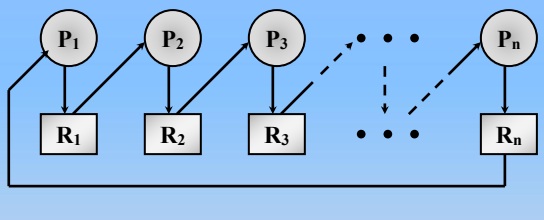
A ordem deve reflectir a sequência normal de utilização dos recursos

Só permitir que um processo obtenha um novo recurso se:

O recurso estiver disponível
E
(O recurso tem uma ordem superior à de qualquer recurso detido pelo processo
OU
O processo não detém nenhum recurso)

Deadlocks – Prevenção

Porque é que esta estratégia é eficaz



A situação descrita no grafo é impossível de acontecer se se obrigar a que os recursos sejam atribuídos por ordem crescente

- Não é possível que P_1 esteja bloqueado a pedir R_1 detendo R_n
- A cadeia circular é quebrada. Os processos obtém os recursos que necessitam de P_n para P_1

Deadlocks – Prevenção

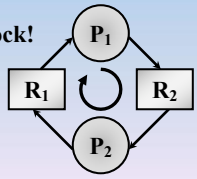
Exemplo de prevenção de deadlock através desta estratégia

Processos P_1, P_2 Recursos R_1, R_2

Pedidos por qualquer ordem

1 – P_1 pede e obtém R_1
2 – P_2 pede e obtém R_2
3 – P_1 pede R_2 e fica bloqueado
4 – P_2 pede R_1 e fica bloqueado

Deadlock!



Pedidos pela mesma ordem

1 – P_1 pede e obtém R_1
2 – P_2 pede R_1 e fica bloqueado
3 – P_1 pede e obtém R_2
4 – P_1 termina e liberta R_1 e R_2
5 – P_2 desbloqueia e obtém R_1
6 – P_2 pede e obtém R_2
7 – P_2 termina e liberta R_1 e R_2

Ambos os processos terminaram!

Deadlocks – Prevenção

Desvantagens na prevenção de deadlocks

- Baixa a utilização dos recursos
  Leva a uma diminuição do *throughput* (completam-se menos processos por unidade de tempo)
- Como o sistema não sabe qual vai a ser a utilização futura dos recursos (quais, em que altura, durante quanto tempo) por parte dos processos, tem que assumir o pior cenário e prevenir todas as hipóteses de ocorrência de deadlocks

  Exemplo
  Processos P_1 e P_2; Recursos R_1, R_2, R_3, R_4
  P_1 requisita R_1 e R_2 (apenas estes e por esta ordem)
  P_2 requisita R_4 e R_3 (apenas estes por esta ordem)
- Com prevenção de deadlock pela estratégia da negação da condição de espera circular, P_2 seria impedido de obter os recursos, apesar de não existir qualquer possibilidade de deadlock (P_1 e P_2 usam recursos diferentes)

Deadlocks – Evitamento

Evitamento

Ideia base:

- Se o sistema tiver informação acerca da utilização futura de cada recurso por parte de todos os processos apenas precisará de actuar (negar o acesso a um recurso por parte dum processo) no caso do pedido poder mesmo gerar um deadlock

Exemplo anterior → não acontece deadlock

- Sabendo que $\mathbf{P_1}$ nunca vai requisitar $\mathbf{R_3}$ ou $\mathbf{R_4}$ e que $\mathbf{P_2}$ nunca vai requisitar $\mathbf{R_1}$ ou $\mathbf{R_2}$, não há possibilidade de ocorrência de deadlock apesar dos processos estarem a requisitar recursos por uma ordem que não é crescente (não é igual para ambos)
- O sistema apercebe-se da impossibilidade de ocorrência de deadlock e deixa $\mathbf{P_2}$ obter os recursos

Deadlocks – Evitamento

Principais dificuldades

- Como saber se, num dado pedido, vai ocorrer deadlock ou não ?
- Como saber a utilização futura de recursos pelos processos ?

Algoritmo mais simples

- Cada processo "declara" quantas unidades de cada recurso irá precisar (sem especificar se é logo no início, a meio da execução ou no fim).
- O sistema fica a saber que, que recursos irão (eventualmente) ser necessários durante a execução de cada processo.

Conceitos principais

- Estado de atribuição de recursos
- Estado seguro / não seguro

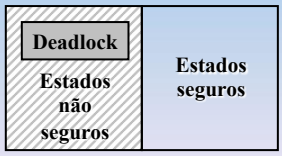
Deadlocks – Evitamento – Conceitos principais

Estado de atribuição de recursos

- Nº de recursos disponíveis
- Nº de recursos atribuídos
- Nº máximo de pedidos pelos processos

Estado seguro

- Quando existe pelo menos uma maneira do sistema atribuir os recursos a todos os processos até ao máximo conhecido para cada um sem causar deadlock.



As situações de deadlock são um sub-conjunto dos estados

Deadlocks – Evitamento – Conceitos principais

Estado seguro – Outra descrição, o mesmo significado

- Quando existe pelo menos uma sequência de atribuição dos recursos ao processos de tal forma que o deadlock não ocorre.
  - Os primeiros processos na sequência terão os recursos ainda livres; os do fim da sequência, quando muito, apenas terão que esperar pela conclusão dos primeiros
  - → = conceito de sequência segura
- Se existir pelo menos uma sequência segura, o estado é seguro.

Sequência segura (definição)

< P_1, P_2 , ..., P_n > tal que,

Para cada processo P_i, os pedidos de P_i podem ser satisfeitos através dos recursos disponíveis actuais mais os recursos detidos pelos processos P_j em que j < i

- Significa que, quando muito (no pior caso), P_i apenas terá que esperar pela conclusão dos processos P_j

Deadlocks – Evitamento

Exemplo

Recursos
12 impressoras

Processos	Recursos máximos necessários
P_1	**10**
P_2	**4**
P_3	**9**

Instante t_0

Recursos detidos
- P_1 – **5** (max. **10**)
- P_2 – **2** (max. **4**)
- P_3 – **2** (max. **9**)

Recursos livres
3

→ **É um estado seguro**

Existe a sequência segura $< P_2, P_1, P_3 >$
(Não existem outras sequências seguras)

Deadlocks – Evitamento

Exemplo (cont.)

t_1 – **P_3** requisita e obtém uma impressora

Recursos detidos
- P_1 – **5** (max. 10)
- P_2 – **2** (max. 4)
- P_3 – **3** (max. 9)

Recursos livres
2

Este estado não é seguro: não existe nenhuma sequência segura

Sequências possíveis
- P_1, P_X, P_Y → "Pára" em P_1
- P_2, P_1, P_3 → "Pára" em P_1
- P_2, P_3, P_0 → "Pára" em P_3
- P_3, P_X, P_Y → "Pára" em P_3

Conclusão
Não se devia ter atribuído a impressora a P_3 em t_1
P_3 devia ter ficado bloqueado

Deadlocks – Evitamento

Estratégia para evitar deadlocks

Ficar sempre nos estados seguros

Algoritmo

Simular a atribuição do(s) recurso(s) pedido(s)
Avaliar o novo estado
Se for seguro, efectiva-se a atribuição
Se não for seguro, não se efectua a a atribuição e o processo fica bloqueado

Pior caso possível:

– Todos os processos ficam bloqueados e apenas um (de cada vez) consegue prosseguir

Perdem-se os benefícios da multiprogramação mas não há deadlock!

Deadlocks – Evitamento

Como avaliar eficientemente se um estado é seguro ou não ?

Caso 1 – Todos os recursos têm apenas uma unidade

➢Utiliza-s uma **variante do grafo de atribuição**

Novo tipo de arco de processos para recursos: "Arco de pedido possível"

– Representação: $\mathbf{P_j} \dashrightarrow \mathbf{R_k}$

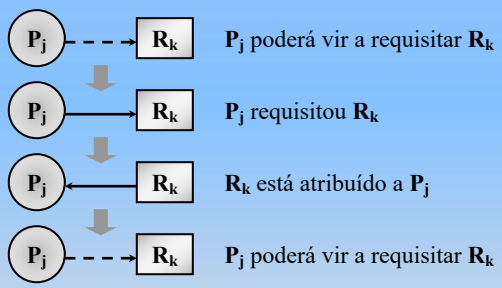
– Significado: "O processo $\mathbf{P_j}$ pode pedir o recurso $\mathbf{R_k}$ no futuro"

Todos os arcos de pedido possível devem ser conhecidos no inicio ou então só podem ser acrescentados novos arcos de pedido possível se só existirem arcos de pedido possível no grafo (vai dar ao mesmo)

➢ Consiste na concretização do requisito base do evitamento de deadlock: ter algum conhecimento acerca do comportamento futuro do processo

Deadlocks – Evitamento

Evolução dos arcos



Utilização do grafo

- Um grafo sem ciclos corresponde a um estado seguro
- Um grafo com ciclos corresponde a um estado não seguro
  → Poderá originar um deadlock (não é obrigatório que aconteça)

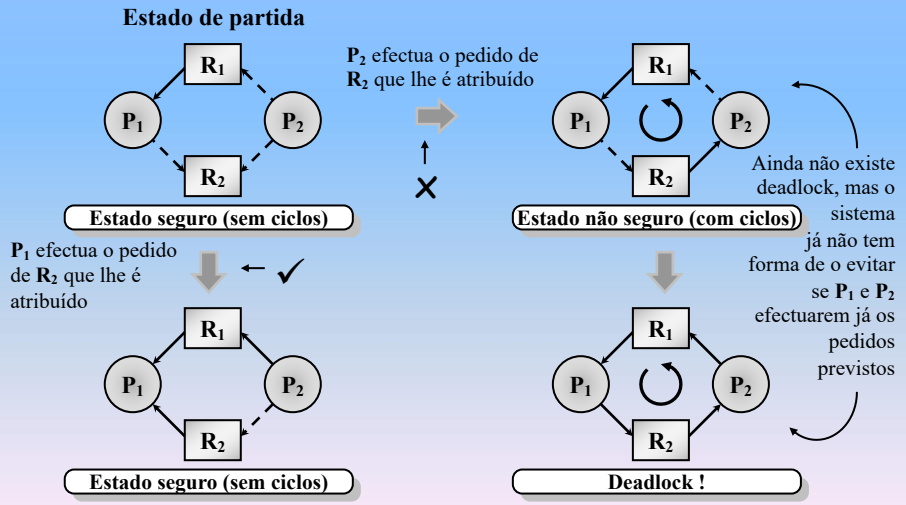
➔ Não se deve passar de um grafo sem ciclos para um com ciclos

➔ **Não esquecer - apenas quando todos os recursos têm apenas uma unidade**

35

Deadlocks – Evitamento

Exemplo (dois processos e dois recursos)



36

Deadlocks – Evitamento

Visualização dos estados seguros / não seguro através de um gráfico onde se representa a evolução dos processos

- A evolução de cada processo está associado a um eixo ordenado (um por cada processo) que representa a sua evolução no tempo
- Num sistema com um processador, a evolução do conjunto dos processos é marcada por linhas paralelas aos eixos (só avança um processo de cada vez)
- O algoritmo dos processos não anda para trás (situação normal). De igual modo a evolução dos processos no gráfico **apenas anda para a frente**
- A área do gráfico é dividida em regiões que correspondem aos períodos de tempo em que os processos necessitam de cada um dos recursos do sistema
- As regiões em que vários processos têm recursos mutuamente necessários pelo(s) outro(s) correspondem a estados impossíveis de atingir
- As regiões que podem conduzir aos deadlocks são estados **não seguros**
- A intersecção das fronteiras entre estados não seguros e estados impossíveis de atingir correspondem à situação de **deadlock**
- **Evitamento de deadlocks**: não se atravessa a fronteira entre estados seguros e estados não seguros

Deadlocks – Evitamento

Como avaliar eficientemente se um estado é seguro ou não ?

Caso 2 – Pelo menos um recurso tem mais que uma unidade

➢Utiliza-se o algoritmo do banqueiro (caso geral)

Estruturas de dados necessárias (com **n** processos e **m** recursos)

Matriz **disp**

- Vector de tamanho **m**
- Indica a **disponibilidade** de cada recurso
- disp[k] = número de unidades disponíveis do recurso k

Matriz **max**

- Matriz rectangular de tamanho **n** x **m**
- Indica a **procura máxima** de todos os recursos por parte dos processos
- max[j,k] = número máximo de unidades do recurso k que o processo j pode obter

Deadlocks – Evitamento – Algoritmo do banqueiro

Estruturas de dados necessárias (cont.)

Matriz **atrib**
- Matriz rectangular de tamanho **n** x **m**
- Indica quantas unidades de cada recurso estão **atribuídas** a cada processo
- atrib[j,k] = número de unidades do recurso k que o processo j detém

Matriz **proc**
- Matriz rectangular de tamanho **n** x **m**
- Indica, quantas unidades de cada recurso ainda podem ser pedidos por cada processo (**procura** possível de recursos)
- proc[j,k] = número máximo de unidades do recurso k que o processo j **ainda** poderá vir a requisitar

  proc[j,k] = max[j,k] - atrib[j,k]

Deadlocks – Evitamento – Algoritmo do banqueiro

Avaliação do estado (seguro / não seguro)

Ideia base: tentar encontrar uma sequência segura

Repetir
- Identificar os processos que podem terminar (**são aqueles cujos pedidos futuros podem ser todos satisfeitos através dos recursos livres**)
- Assumir que esses processos terminam: **adicionam-se os recursos por eles detidos aos recursos livres**

Até todos os processos terem "terminado" ou mais nenhum puder "terminar"

Se todos os processos tiverem "terminado", então a sequência pela qual terminam é uma **sequência segura** e o estado é seguro

Definição necessária ao algoritmo

Sejam x e y vectores de tamanho M

x <= y se x[i] <= y[i] para todos os i entre 1 e M

Deadlocks – Evitamento – Algoritmo do banqueiro

Avaliação do estado (seguro / não seguro)

N processos e **M** recursos

Matrizes auxiliares

w[M] Vector que indica os recursos livres ao longo da execução do algoritmo
f[N] Vector que indica se um processo já terminou ou não

1. **w= disp** // Recursos livres iniciais no estado em questão
   f[i] = falso para cada i entre 1 e N // Inicialmente nenhum processo terminou
2. **Enquanto existir algum i entre 1 e N tal que (f[i] = falso) e (proc[i] ≤= w) faz**
   w = w + atrib[i] // Assume que P_i termina e liberta os seus recursos
   f[i] = verdadeiro // Assinala que P_i consegue terminar
3. **Se (f[i] = verdade) para todos os i entre 1 e N então o estado é seguro**
   Caso contrário o estado não é seguro

Deadlocks – Evitamento – Algoritmo do banqueiro

Algoritmo de requisição de um recurso (quando ocorre um pedido)

$pedido_j$ = vector de pedidos do processo **j**
$pedido_j$[**k**] = número de unidades do recurso **k** que o processo **j** pretende

1. **Se ($pedido_j$ <= proc[j]) for falso**
   O processo **j** está a tentar obter mais recursos que o que tinha declarado inicialmente. O pedido é negado. **Fim**
2. **Se ($pedido_j$ <= disp) for falso**
   O processo **j** fica em espera porque não há recursos disponíveis. **Fim**
3. Simular a atribuição dos recursos e ver se o estado resultante é seguro
   disp = disp - pedido
   atrib[j] = atrib[j] + $pedido_j$
   proc[j] = proc[j] - $pedido_j$
   Averiguar se estado é seguro. Se for, confirma-se a atribuição dos recursos, caso contrário restauram-se as estruturas de dados e o processo fica em espera. **Fim**

Deadlocks – Evitamento – Algoritmo do banqueiro

Exemplo 1 – Construção e uso das matrizes que descrevem o estado do sistema

Num determinado sistema existem os recursos A, B, C
A tem 5 unidades, B tem 6 unidades, C tem 4 unidades

Matriz de disponibilidade (inicial)

Disp.	A	B	C
	5	6	4

Deadlocks – Evitamento – Algoritmo do banqueiro

Exemplo 1 – Construção e uso das matrizes que descrevem o estado do sistema

Nesse sistema foram colocados 4 processos a executar: P1, P2, P3, P4
Cada processo indicou inicialmente ao sistema a utilização máxima esperada de cada recurso.

"Utilização máxima" = "quantos recursos precisa em simultâneo"

≠ "quantas vezes pode usar" (pode usar quantas vezes quiser desde que não ultrapasse a quantidade máxima em simultâneo que indicou - ou seja, se libertar um recurso pode sempre pedi-lo novamente mais tarde)

Neste exemplo: P1 pode vir a usar: 2 de A, 3 de B, 1 de C
P2: 3 de A, 1 de B, 2 de C
P3: 1 de A, 2 de B, 1 de C
P4: 1 de A, 1 de B, 2 de C

Matriz de pedidos máximos (iniciais) ----->

Proc Max (inicial)	A	B	C
P1	2	3	1
P2	3	1	2
P3	1	2	1
P4	1	1	2

Deadlocks – Evitamento – Algoritmo do banqueiro

Exemplo 1 – Construção e uso das matrizes que descrevem o estado do sistema

No instante t_0 nenhum processo tem ainda recursos. A disponibilidade actual coincide com a inicial, a matriz de atribuição está toda a zeros, e a procura máxima coincide com a procura máxima inicial

Disp (t_0)

A	B	C
5	6	4

Atrib (t_0)

	A	B	C
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	0

Proc (t_0)

	A	B	C
P1	2	3	1
P2	3	1	2
P3	1	2	1
P4	1	1	2

Deadlocks – Evitamento – Algoritmo do banqueiro

Exemplo 1 – Construção e uso das matrizes que descrevem o estado do sistema

Ainda no instante t_0 o processo P2 efectua um pedido de 2 unidades de A, 0 de B e 1 de C

Vector de pedido de P2 ----->

	A	B	C
P2:	2	0	1

O sistema vai então verificar as seguintes condições

1. O processo P2 "pode" pedir isso?
   Consiste em verificar se o vector de pedido de P2 tem valores menores ou iguais à linha correspondente a esse processo na matriz proc. Se não for verdade, o pedido é negado (o processo não fica à espera – é-lhe indicado um código de erro)
2. Os recursos estão disponíveis neste momento?
   Consiste em verificar se o vector de pedido de P2 tem valores iguais ou menores ao vector ***disp***. Se não for verdade, o processo fica à espera
3. Se o processo puder fazer esse pedido e os recursos estiverem disponíveis, se forem atribuídos, o estado resultante é seguro? Se não for, o processo fica à espera. Para responder a esta questão o sistema simula a atribuição de recursos e depois tenta encontrar uma sequência segura.

Deadlocks – Evitamento – Algoritmo do banqueiro

Exemplo 1 – Construção e uso das matrizes que descrevem o estado do sistema

Ainda no instante t_0 o processo P2 efectua um pedido de 2 unidades de A, 0 de B e 1 de C

Vector de pedido de P2 -->

	A	B	C
P2:	2	0	1

Simulação da atribuição – matrizes resultantes

Disp ($t_{0'}$)

A	B	C
3	6	3

Atrib ($t_{0'}$)

	A	B	C
P1	0	0	0
P2	2	0	1
P3	0	0	0
P4	0	0	0

Proc ($t_{0'}$)

	A	B	C
P1	2	3	1
P2	1	1	1
P3	1	2	1
P4	1	1	2

Existe a sequência segura (entre outras): P1 -> P2 -> P3 -> P4 portanto o estado resultante é seguro

=> a atribuição dos recursos é efectuada e este estado passa a ser o estado actual, a ser usado como base nas considerações a fazer no próximo pedido

Deadlocks – Evitamento – Algoritmo do banqueiro

Exemplo 1 – Construção e uso das matrizes que descrevem o estado do sistema

Após várias iterações de pedidos e devoluções, o sistema encontra-se no instante *n*. A matriz ***disp*** descreve os recursos disponíveis, a matriz ***atrib*** descreve os recursos atribuídos a cada processo, e a matriz ***proc*** descreve os pedidos que ainda são esperados e admitidos por parte de cada processo.

Exemplo instante *n*

Disp (t_n)

A	B	C
1	4	2

Atrib (t_n)

	A	B	C
P1	2	2	1
P2	2	0	1
P3	0	0	0
P4	0	0	0

Proc (t_n)

	A	B	C
P1	0	1	0
P2	1	1	1
P3	1	2	1
P4	1	1	2

O exemplo seguinte (independente deste) repete estas ideias mas mostra também uma situação em que em que um pedido é rejeitado porque o estado resultante seria não-seguro

Deadlocks – Evitamento – Algoritmo do banqueiro

Exemplo 2 – Utilização das matrizes no algoritmo do banqueiro

Processos: P_1, P_2, P_3, P_4, P_5

Recursos: R_1, R_2, R_3

Dados iniciais (invariantes ao longo da execução dos processos)

Unidades existentes de cada recurso

R1	R2	R3
10	5	7

Procura máxima declara pelos processos (matriz **max**)

	R1	R2	R3
P1	7	5	3
P2	3	2	2
P3	9	0	2
P4	2	2	2
P5	4	3	3

Deadlocks – Evitamento – Algoritmo do banqueiro

Exemplo 2 (cont.)

No instante t_1 observa-se

Atrib

	R1	R2	R3
P1	0	1	0
P2	2	0	0
P3	3	0	2
P4	2	1	1
P5	0	0	2

Disp

R1	R2	R3
3	3	2

Proc

	R1	R2	R3
P1	7	4	3
P2	1	2	2
P3	6	0	0
P4	0	1	1
P5	4	3	1

Max - Atrib = Proc

Os processos podem terminar pela sequência < P_2 , P_4 , P_5 , P_3 , P_1 >

⇒ O estado é seguro

Deadlocks – Evitamento – Algoritmo do banqueiro

Exemplo 2 (cont.)

P_2 pede uma unidade de R_1 e duas de R_3

$pedido_2 = (1, 0, 2)$

Aplicação do algoritmo

1. $pedido_2 \leq proc[2] \rightarrow (1, 0, 2) \leq (1, 2, 2) \rightarrow$ Sim
2. $pedido_2 \leq disp \rightarrow (1, 0, 2) \leq (3, 3, 2) \rightarrow$ Sim
3. Assumir que o pedido é satisfeito.

Disp

R_1	R_2	R_3
2	3	0

Atrib

	R_1	R_2	R_3
P_1	0	1	0
P_2	3	0	2
P_3	3	0	2
P_4	2	1	1
P_5	0	0	2

Proc

	R_1	R_2	R_3
P_1	7	4	3
P_2	0	2	0
P_3	6	0	0
P_4	0	1	1
P_5	4	3	1

Verificar se o estado resultante é seguro

→ Sim ($<P_2, P_4, P_5, P_1, P_3>$)

⇒ O pedido de P_2 é satisfeito

51

Deadlocks – Evitamento – Algoritmo do banqueiro

Exemplo 2 (cont.)

Pedido posterior (ao pedido satisfeito no slide anterior) por P1 de (0, 2, 0)

$pedido_1$ = (0, 2, 0)

Aplicação do algoritmo

1. $pedido_1 \leq proc[1] \rightarrow (0, 2, 0) \leq (7, 4, 3) \rightarrow$ Sim
2. $pedido_1 \leq disp \rightarrow (0, 2, 0) \leq (2, 3, 0) \rightarrow$ Sim
3. Assumir que o pedido é satisfeito.

Disp

R_1	R_2	R_3
2	1	0

Atrib

	R_1	R_2	R_3
P_1	0	3	0
P_2	3	0	2
P_3	3	0	2
P_4	2	1	1
P_5	0	0	2

Proc

	R_1	R_2	R_3
P_1	7	2	3
P_2	0	2	0
P_3	6	0	0
P_4	0	1	1
P_5	4	3	1

Verificar se o estado resultante é seguro

→ Não é

⇒ O pedido de P_1 não é satisfeito

52

Deadlocks – Detecção

Situação – Estará o sistema com deadlocks ?

O que há a fazer

- Detectar
- Resolver (recuperar) se for detectado algum deadlock

Detecção – Caso 1 – Recursos com apenas uma unidade

➔ Através da análise do grafo de atribuição de recursos

- Se existir um ciclo, existe deadlock
- Por vezes utiliza-se uma variante do grafo de atribuição de recursos

  ➔grafo de *espera-por (wait-for)*

  - Os recursos desaparecem (logo menos nós para percorrer)
  - Os pares de arcos $P_X \rightarrow R_Y \rightarrow P_Z$ dão origem ao arco $P_X \rightarrow P_Y$
  - Não existem mais arcos
  - A análise do grafo é a mesma: ciclo ⇒ deadlock

Deadlocks – Detecção

Detecção – Caso 2 – Recursos com mais que uma unidade

N processos e **M** recursos

➔Utilização do algoritmo banqueiro

Utilização semelhante à do evitamento de deadlocks, mas em que se está apenas a avaliar se o estado é de facto um deadlock, e não se é seguro.

- Estruturas de dados

Disp	vector de comprimento M indica os recursos disponíveis de momento
Atrib	matriz N x M Indica os recursos detidos por cada processo
Proc	matriz N x M indica a procura **actual** por parte de cada processo (no evitamento significava procura máxima!)

Deadlocks – Detecção

Aplicação do algoritmo para a detecção de deadlock

N processos e **M** recursos

Matrizes auxiliares

w[M] Vector que indica os recursos livres ao longo da execução do algoritmo

f[N] Vector que indica se um processo já terminou ou não

1. **w= disp** // Recursos livres iniciais no estado em questão
   f[i] = falso para cada i entre 1 e N // Inicialmente nenhum processo terminou
2. **Enquanto existir algum i entre 1 e N tal que (f[i] = falso) e (proc[i] ≤ w) faz**
   w = w + atrib[i] // Assume que P_i termina e liberta os seus recursos
   f[i] = verdadeiro // Assinala que P_i consegue terminar
3. **Se existir algum i tal que (f[i] = falso) então existe um deadlock**
   Caso contrário não existe de momento um deadlock (por enquanto todos os processos podem terminar segundo uma certa ordem)

Deadlocks – Detecção

No instante t em que se aplica a detecção:

- O algoritmo apenas detecta a existência de deadlocks no instante da sua aplicação, não é aplicado para fazer previsões (estado seguro/não seguro) tal como acontecia no evitamento
- Se não for detectado um deadlock não implica que não possa ocorrer mais tarde
- Um deadlock que seja inevitável mas que não esteja a ocorrer de momento não é detectado, mas é garantidamente detectado num instante futuro em que ocorra e se execute o algoritmo

Deadlocks – Detecção

Exemplo de detecção de deadlock

Processos: P_1 , P_2 , P_3 , P_4 , P_5 Recursos: R_1 , R_2 , R_3

Unidades existentes de cada recurso:

R1	R2	R3
7	2	6

No instante t_1 observa-se a seguinte situação

atrib

	R1	R2	R3
P1	0	1	0
P2	2	0	0
P3	3	0	3
P4	2	1	1
P5	0	0	2

proc

	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	0
P4	1	0	0
P5	0	0	2

disp

R1	R2	R3
0	0	0

- O sistema não está em deadlock

A sequência < P_1 , P_3 , P_4 , P_2 , P_5 > permite completar todos os processos

Deadlocks – Detecção

Exemplo de detecção de deadlock (cont.)

Supondo que P_3 estava a pedir mais uma unidade do recurso R_3, ter-se-ía:

atrib

	R1	R2	R3
P1	0	1	0
P2	2	0	0
P3	3	0	3
P4	2	1	1
P5	0	0	2

proc

	R1	R2	R3
P1	0	0	0
P2	2	0	2
P3	0	0	1
P4	1	0	0
P5	0	0	2

disp

R1	R2	R3
0	0	0

- Agora o sistema contém um deadlock envolvendo P_2 , P_3 , P_4 e P_5

Deadlocks – Recuperação

Quando um deadlock acontece e é detectado

- O sistema deve eliminar o deadlock

Existem duas categorias de métodos para o conseguir

- Terminação de processos
  - Significa terminar um ou mais processos "à força"
  - O processo termina, não por o seu algoritmo assim o determinar, mas sim por intervenção do sistema operativo
  - Provavelmente perde-se (parte do) trabalho já efectuado pelo processo
- Preempção de recursos
  - Significa retirar um ou mais recursos a um processo
  - São retirados recursos até o deadlock desaparecer
  - O recurso pode ficar num estado incoerente → O SO pode ter que reinicializar o recurso

Deadlocks – Recuperação

Recuperação através de terminação de processos

Alternativas

- Terminar todos os processos envolvidos no deadlock
  - Solução demasiado custosa
    - Os processos podem ter já efectuado grande quantidade de trabalho
- Terminar um processo de cada vez até que o deadlock desapareça
  (De entre os processos envolvidos no deadlock)
  - Solução pesada em termos de eficiência
    - Obriga à execução do algoritmo de detecção deadlocks por cada processo que se termina

Deadlocks – Recuperação

Recuperação através de terminação de um processo de cada vez

→ Qual o (próximo) processo a ser terminado ?

– Deve ser aquele que envolve um custo mínimo

Factores que influenciam o custo de terminação do processo

- Prioridade de processo
- Quanto trabalho é que o processo já tinha efectuado e quanto é que ainda lhe falta fazer
- Quantos recursos é que o processo detém
- Qual o tipo de recursos que são detidos pelo processo
- Quantos recursos mais o processo precisa para se completar
- O processo é interactivo ?

Deadlocks – Recuperação

Recuperação através de preempção de recursos

Questões envolvidas

1 – Selecção da "vítima"

Qual o recurso e qual o processo que o vai perder ?

- Deve-se escolher o recurso/processo cuja preempção tem o menor custo

2 – Reposição (*rollback*)

O que acontece ao processo que ficou sem o recurso ?

Alternativas

- É terminado e recomeçado (reposição total: + simples e + drástico)
- É reposto num estado coerente anterior e recomeçado a partir daí
  - Como saber qual é esse estado ?
  - É necessária mais informação do algoritmo

Deadlocks – Recuperação

Recuperação através de preempção de recursos

Questões envolvidas (cont.)

3 – *Starvation*

Corre-se o perigo de ser sempre o mesmo processo a sofrer a preempção

- O processo acabaria por ficar em *starvation* – nunca obteria o recuros durante o tempo suficiente para efectuar trabalho necessário

Solução: Levar em consideração o número de vezes que um processo já foi vítima de uma preempção de recurso

Deadlocks – Combinação das técnicas

Prevenção, evitamento, detecção e recuperação

- São as técnicas principais
- Utilizadas de forma isolada não produzem resultados muito satisfatórios

Solução

➔ Utilização combinada das técnicas anteriores

- Particionamento dos recursos por categorias ordenadas
  - Para evitar à partida que os deadlocks ocorreram com recursos de mais de uma classe: **prevenção através da negação da condição "espera circular"**
- Dentro de cada categoria, utiliza-se uma das técnicas vistas
- A técnica a utilizar dentro de cada classe depende da natureza dos recursos dessa classe

Deadlocks – Combinação das técnicas

Categorias típicas de recursos e técnicas utilizadas

- Recursos internos usados pelo sistema
  Exemplo: entradas em tabelas de sistema
  - ➢ Prevenção através da negação da condição de espera circular
    Ordenam-se os recursos e obrigam-se os processos a obtê-los por uma certa ordem
- Memória central
  Memória usada pelos processos
  - ➢ Prevenção através da negação da condição "recursos não preemptíveis
    O processo pode ser transferido para memória secundária, libertando-se assim espaço em memória central
    Corresponde a retirar o recurso "memória central" ao processo e atribuí-lo a outro

Deadlocks – Combinação das técnicas

Categorias típicas de recursos e técnicas utilizadas

- Recursos de trabalho
  Exemplos: ficheiros e dispositivos
  - ➢ Evitamento (avaliação de estados seguros/não seguros)
    A utilização deste tipo de recursos pode ser obtida no inicio do processo
- Memória de *swap*
  Espaço dos processos em memória secundária
  - ➢ Prevenção através da negação da condição "detenção e espera"
    O processo só se iniciará se existir espaço em memória secundária para que em qualquer instante da sua execução possa ser *swapped-out*
    Geralmente o requisito máximo em termos de memória é facilmente conhecido pelo sistema