Deadlocks

INF151 – Sistemas Operacionais II

Prof. Marcelo Johann

UFRGS – 2006/2

Introdução

Processo:

Componente ativo que solicita recursos

Recurso:

Qualquer entidade que pode ocasionar o bloqueio de um processo

Processo fica bloqueado quando solicita um recurso que está indisponível nesse momento

Exemplo:

P1: requisita(R1); ... requisita(R2); ... libera(R1,R2);

P2: requisita(R2); ... requisita(R1); ... libera(R1,R2);

Deadlocks

***Deadlock*:**

Um conbjunto de *N* processos está em *deadlock* quando cada um dos *N* processos está bloqueado à espera de um recursos que somente pode ser liberado por um processo desse conjunto

Recursos serialmente reusáveis:

Número fixo de unidades idênticas usadas independente e temporariamente. Ex: região crítica

Recursos consumíveis:

Número variável de unidades criadas e consumidas dinamicamente pelos processos. Ex: semáforos

Condições necessárias para *Deadlocks*

Só pode haver *deadlock* se:

1 - processo que tem um recurso pode solicitar outro;

Alocação estática evita isso.

2 – recursos alocados não podem ser confiscados, mesmo que seja temporariamente, ou seja, só podem ser liberados pelo próprio processo que os detêm;

Processos podem devolver recursos ao bloquearem-se.

3 – for possível a formação de um ciclo envolvendo alocações e requisições entre processos e recursos;

Hierarquia e ordenamento evitam ciclos.

Atitudes perante *Deadlocks*

Abordagens:

Atitudes perante *Deadlocks*

Abordagens:

1) Não fazer nada! Dá Ctrl-Alt-Del se acontecer!

Atitudes perante *Deadlocks*

Abordagens:

1) Não fazer nada! Dá Ctrl-Alt-Del se acontecer!

2) Programar o sistema de modo que sejam impossíveis...

Atitudes perante *Deadlocks*

Abordagens:

1) Não fazer nada! Dá Ctrl-Alt-Del se acontecer!

2) Programar o sistema de modo que sejam impossíveis...

3) Detecção: deixar o sistema livre em relação à utilização dos recursos, mas detectar as situações de *deadlock* e matar alguns processos para resolver. Em geral, mata-se todos, mas primeiro os de menor prioridade e mais distantes de sua conclusão.

Atitudes perante *Deadlocks*

Abordagens:

1) Não fazer nada! Dá Ctrl-Alt-Del se acontecer!

2) Programar o sistema de modo que sejam impossíveis...

3) Detecção: deixar o sistema livre em relação à utilização dos recursos, mas detectar as situações de *deadlock* e matar alguns processos para resolver. Em geral, mata-se todos, mas primeiro os de menor prioridade e mais distantes de sua conclusão.

4) Prevenção: usar um algoritmo de alocação de recursos inteligente que evita dinamicamente colocar o sistema em situações que potencialmente levem a *deadlock*.

2) Quebrar ciclos

Com recursos ou processos

Wait-die - Mais velho espera, mais novo morre.

Se p pede um recurso que está com q:

Se idp< idq (p é mais velho), p vai para fila esperar

Caso contrário (p mais novo), p libera recursos e morre

Wound-wait - Mais velho mata, mais novo espera.

Se p pede um recurso que está com q:

Se idp< idq (p é mais velho), p mata q e usa seus recursos

Caso contrário (p mais novo), p vai para a fila de espera

3) Detecção de *Deadlocks*

Modelo de Holt

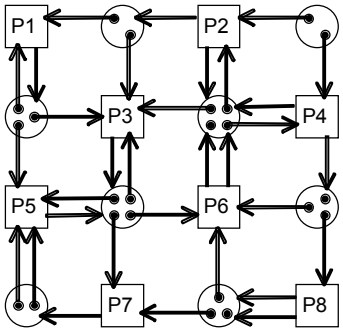
Representa: processos, recursos, alocações, requisições.

Se há *deadlock* não há maneira de resolvê-lo.

Então simula-se a melhor seqüência de operações do estado atual:

Para cada processo não bloqueado, dá os recursos que ele pede e supõe que ele termina a execução e libera todos os recursos que possui.

Modelo de Holt



4) Prevenção de *Deadlocks*

Algoritmo do Banqueiro

(1) Supõe o atendimento da requisição.

(2) Então supõe que todos os processos não bloqueados requisitem o máximo de recursos que podem.

(3) Aplica o algoritmo de detecção de *deadlocks* sobre essa situação hipotética.

(4) Se há *deadlock*, não faz o atendimento, mesmo que haja unidades para fazê-lo, pois isso pode conduzir a *deadlock*.

Algoritmo do Banqueiro

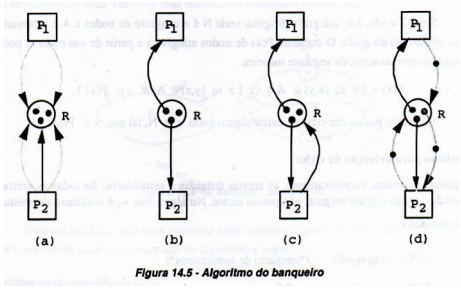


Figura 14.5 - Algoritmo do banqueiro