Sistemas Distribuídos

Deadlocks

Referência

- “Sistemas operacionais modernos” Andrew S. TANENBAUM Prentice-Hall, 1995
  - Seção 11.5 pág. 340 - 344

Conteúdo

- Detecção de deadlock distribuído
  - Detecção centralizada
  - Detecção distribuída
- Prevenção de deadlock distribuído

Deadlocks em Sistemas distribuídos

- Problema semelhante aos sistemas com um único processador porém mais complexo
- Definição:
  - Um conjunto de processos está em uma situação de deadlock, se cada processo do conjunto estiver esperando por um evento que somente outro processo pertencente ao conjunto poderá fazer acontecer

Deadlocks em Sistemas distribuídos

- Condições para ocorrer deadlock:
  - Condição de exclusão mútua: cada recurso ou está alocado a exatamente um processo ou está disponível
  - Condição de posse e espera: processos que estejam de posse de recursos obtidos anteriormente podem solicitar novos recursos
  - Condição de não-preempção: recursos já alocados a processos não podem ser tomados a força. Eles precisam ser liberados explicitamente pelo processo que detém sua posse
  - Condição de espera circular: deve existir uma cadeia circular de 2 ou + processos, cada um deles esperando por um recurso que está com o próximo membro da cadeia

Deadlocks em Sistemas distribuídos

- Dois tipos:
  - Comunicação entre processos: processo A tenta enviar msg para B, que tenta enviar msg para C que tenta enviar msg para A (ex.: indisponibilidade de buffer)
  - Alocação de recursos: vários processos competem por recurso como dispositivos de E/S, arquivos etc

Deadlocks em Sistemas distribuídos

- Estratégias para tratar deadlocks:
  - algoritmo do avestruz: ignorar o problema
  - detecção: permitir o deadlock e tentar a recuperação
  - prevenção: tornar estruturalmente impossível a ocorrência de deadlocks
  - evitar deadlocks com uma política de alocação de recursos cuidadosa

Detecção de deadlocks

- Quando um deadlock é detectado em um sistema operacional convencional a forma convencional de resolvê-lo é matando um ou mais processos
- Em um sistema baseado em transações atômicas pode-se abortar uma ou mais transações.

Detecção centralizada de deadlocks

- Cada máquina possui um grafo para seus próprios processos e recursos.
- Um coordenador central deve manter o grafo de recursos para todo sistema. Quando o coordenador detecta o deadlock ele mata um dos processos envolvidos.

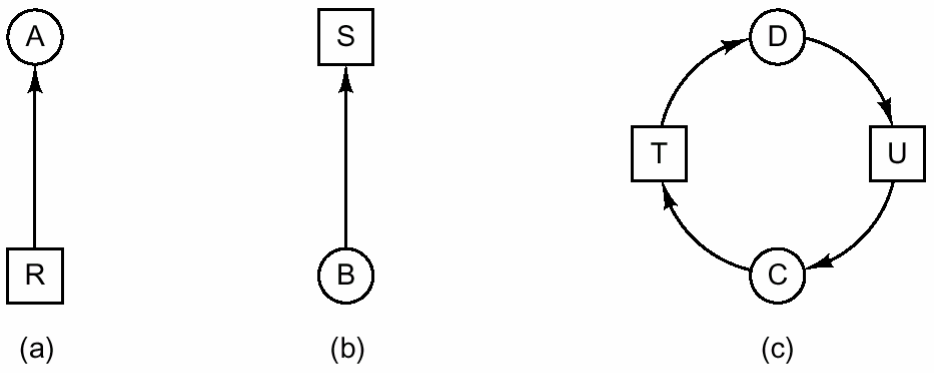
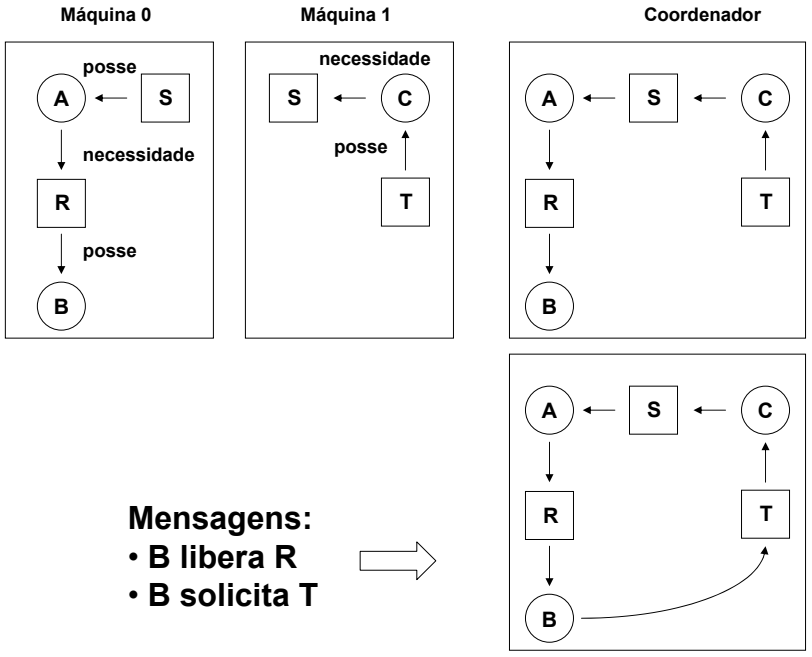


Fig. 3-3. Resource allocation graphs. (a) Holding a resource. (b) Requesting a resource. (c) Deadlock.

Detecção centralizada de deadlocks

- Manutenção da informação:
  - Sempre que um arco for adicionado ou removido o coordenador é informado
  - Cada máquina envia periodicamente uma lista de arcos adicionados e removidos
  - O coordenador solicita as informações quando forem necessárias

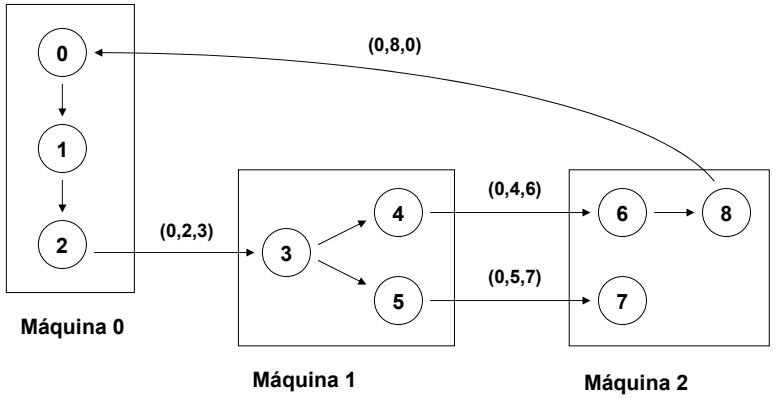


Detecção centralizada de deadlocks

- Pseudo-Deadlock
  - Situação onde ocorre um ciclo no grafo devido ao atraso de informações.
  - Falso deadlock → algoritmo de Lamport

Detecção distribuída de deadlocks

- Algoritmo de Chandy-Misra-Haas (1983)
- Os processos podem requisitar vários recursos
- É distribuído, portanto não há um processo coordenador .
- É assumido:
  - que o tempo de transmissão de uma mensagem é arbitrário mas finito.
  - não há perda de mensagens, são recebidas na ordem em que são enviadas.



Detecção distribuída de deadlocks

- Mensagens de dois tipos são usadas neste algoritmo de detecção.
  - do tipo pergunta : inicia a procura no grafo do iniciante para identificar os processos no conjunto de dependentes
  - do tipo resposta : relata os resultados desta busca para o iniciante
- Cada processo *Pi* usa certas variáveis locais para gravar sua própria visão do estado global.

Detecção distribuída de deadlocks

- O algoritmo é executado quando um processo deve esperar por algum recurso.
- Um msg de sondagem é enviada para o(s) processo(s) detentor(es) do(s) recurso(s).
- A msg é composta por 3 números: o processo bloqueado, o processo que enviou a msg e o processo para o qual ela está sendo enviada. Ex.: o processo 0 enviaria (0,0,1).

Detecção distribuída de deadlocks

- Qndo um processo recebe a msg ele verifica se ele está aguardando algum recurso de algum processo. Se estiver atualiza a msg substituindo o segundo número pelo seu próprio e o terceiro pelo processo que ele está aguardando e envia a msg.
- Se estiver bloqueado por causa de vários, cada um deles recebe uma msg diferente.
- Se a msg voltar à origem (primeiro número) existe um deadlock.

Detecção distribuída de deadlocks

- Como desfazer o deadlock?
- Forçar o processo que iniciou a sondagem a cometer suicídio.
- E se vários executam a sondagem ao mesmo tempo?
- Identificação de cada processo na msg. Ao final matar o processo de maior número.

Detecção distribuída de deadlocks

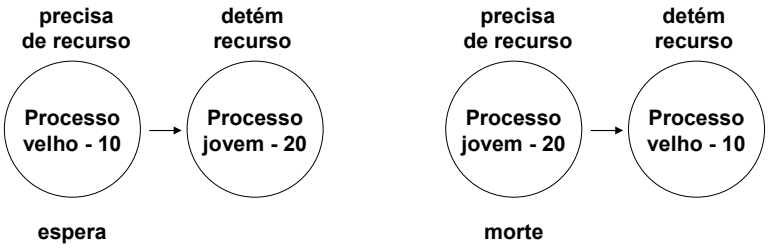
- Teoria *versus* Prática:
- Como fazer que um processo bloqueado envie msgs?
- 90% dos deadlocks envolvem apenas 2 processos.

Prevenção de deadlocks distribuídos

- Prevenção: projetar o sistema de forma a impedir a ocorrência de deadlocks. Ex.:
  - os processos só podem deter a posse de um único recurso por vez
  - os processo deve pedir todos os recursos necessários com antecedência
- Sistemas com tempo global e transações atômicas

Prevenção de deadlocks distribuídos

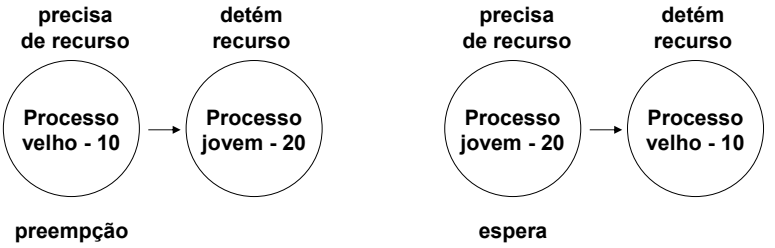
- Idéia: atribuir a cada transação um relógio global que marca o início da transação.
- Qndo um processo vai ser bloqueado por espera de recurso que outro processo está usando verifica-se o carimbo de tempo de cada transação
  - o mais velho tem maior prioridade (evita starvation)
  - o mais novo tem maior prioridade



Algoritmo esperar-morrer

Prevenção de deadlocks distribuídos

- Se o processo jovem precisar de um recurso de posse do mais velho ele deve ser morto. Provavelmente ele vai ser reiniciado e tentar novamente. Se o processo mais velho ainda não liberou o recurso ele vai ser morto novamente.
- Outra opção é fazer a preempção do processo mais jovem. Isto faz com que a transação seja reiniciada. E se o processo requisitar o recurso agora de posse do mais velho ele deve esperar.



Algoritmo wound-wait (ferir-esperar)

Exercícios

- "Sistemas operacionais modernos" Andrew S. TANENBAUM Prentice-Hall, 1995
  - Capítulo 11 Exercícios 22 e 23 (pág. 345)