ALGORITMOS DISTRIBUÍDOS

Algoritmos Distribuídos

- Processos em um sistema distribuído geralmente buscam atingir cooperação e para tanto utilizam mecanismos de sincronização para que esta cooperação seja realizada de maneira correta.
- Esta sincronização pode envolver problemas de:
  - sincronização de relógio
  - exclusão mútua
  - *deadlock*
  - algoritmos de eleição

ALGORITMOS DISTRIBUÍDOS
Sincronização de relógio

Algoritmos Distribuídos (relógio)

- Cada computador possui seu próprio relógio
- Diversos processos executando necessitam uma forma de saber qual operação foi executada primeiro
  - Exemplo: sistema de reserva de passagens: último assento em um vôo deve ser reservado pelo cliente que fez a requisição antes
- ou mesmo medir o tempo necessário para tarefas distribuídas, que começam em um nodo e acabam em outro
  - Exemplo: como saber o tempo de transmissão de uma mensagem?
- Existe a necessidade de sincronização de relógios dos diferentes nodos.

Algoritmos Distribuídos (relógio)

- Um relógio sempre trabalha de maneira constante, pois o cristal oscila em uma freqüência fixa
- Entretanto, cristais diferentes podem oscilar diferentemente
- Diferença pode ser pequena, mas com o passar do tempo pode causar sérios problemas
- O relógio do computador pode assim se afastar do relógio real

Algoritmos Distribuídos (relógio)

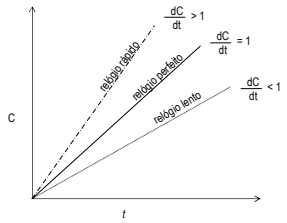
- **Relógio real**
- Era calculado baseado na rotação da terra em torno de seu eixo
- Terra com o passar do tempo está diminuindo sua velocidade de rotação
- Na idade média diversos dias tiveram que ser retirados do calendário devido a este problema
- Desde 1958, o relógio real é calculado por um certo número de transições do Césio 133
- Até 1995 houve uma diferença entre o relógio calculado através do período de rotação da terra com o do Césio 133 de mseg. a cada 86.400 segundos

Algoritmos Distribuídos (relógio)

- Relógios baseados em cristal podem ter uma diferença entre si de 1 seg. a cada 1.000.000 segs., ou seja 1 seg. a cada 11,6 dias.
- Desta forma o relógio do computador deve ser resincronizado periodicamente.
- Suponha *t* hora real, e tempo de um relógio *p* como $C_p(t)$ *("C" de clock)*
- Se todos relógios forem perfeitamente sincronizados então $C_p(t) = t$ para todos *p* e todos *t*
- Idealmente dC/dt = 1
- Como isto não é possível, define-se um ρ que representa o máximo que o relógio desvia de dC/dt, ou seja $(1- \rho) \leq (dC/dt) \leq (1+ \rho)$

Algoritmos Distribuídos (relógio)

- Relógio perfeito, relógio lento, e relógio rápido



Algoritmos Distribuídos (relógio)

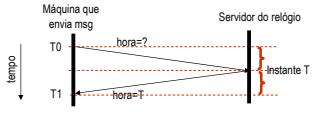
- Considerações
  - se dois relógios desviam em direções opostas, depois de um Δt da última sincronização, o desvio máximo entre eles é de: $2\rho.\Delta t$
  - para assumir que dois relógios nunca se diferenciam mais do que um valor σ (assumindo que relógios estão sincronizados se a diferença entre os dois não for maior que uma constante definida σ), então
    - $2\rho.\Delta t < \sigma \Rightarrow \Delta t < \sigma/2\rho$
    - ou seja, deve-se ressincronizar os relógios no máximo a cada $\sigma/2\rho$
  - exemplo: (u.t. = unidade de tempo)
    - diferença aceitável de 1 u.t.
    - relógio tem ρ = 0,005 (desvio de 5 u.t. em 1000 u.t.)
    - ressincronizar a cada 1/(2 . 0,005) = 100 u.t.

Algoritmos Distribuídos (relógio)

- Considerações
  - relógios estão sincronizados se a diferença entre os dois não for maior que uma constante definida σ
  - computadores devem conhecer o valor de relógios de outros computadores no sistema
  - ao ler o valor, problemas podem acontecer durante a comunicação do valor
  - Importante: relógios não devem voltar no tempo nunca, nem dar saltos muito grandes
    - aumenta-se ou diminui-se a velocidade do relógio
    - exemplo: se interrupção adiciona 8 mseg. então pode adicionar 9 ou 7 mseg.

Algoritmos Distribuídos (relógio)

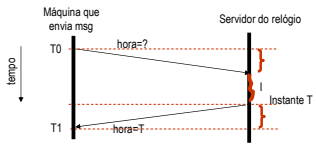
- Algoritmo centralizado para sincronização: **servidor passivo**
  - Servidor espera ser perguntado pela hora, e.g. "hora=?"
  - Após receber a mensagem informa "hora=T", onde T é a hora no servidor
  - Se cliente enviou a mensagem no tempo T0 e recebeu a resposta no tempo T1, então novo horário no cliente = T+(T1-T0)/2



  - supõe-se que tempo de tráfego do request = tempo de tráfego do reply e que tempo de serviço = 0 ...

Algoritmos Distribuídos (relógio)

- Algoritmo centralizado para sincronização: **servidor passivo**
  - Outra estimativa pode levar em conta o tempo (I) que o servidor levou para processar a solicitação, ou seja novo horário no cliente = T+(T1-T0-I)/2 (importante é contabilizar a diferença que acontece na transmissão)



Algoritmos Distribuídos (relógio)

- Algoritmo centralizado para sincronização: **servidor ativo**
  - sevidor *broadcasts* o valor da hora para todos computadores periodicamente
  - em geral servidor sabe o tempo de transmissão entre servidor e computadores
  - desta forma periodicamente ele envia "hora=T+Ta", onde Ta é o tempo de transmissão para cada um dos computadores
- Variação: Algoritmo *Berkeley*

Algoritmos Distribuídos (relógio)

- Algoritmo centralizado para sincronização: **servidor ativo**
- Variação: Algoritmo *Berkeley*
  - periodicamente o servidor pergunta o horário dos computadores
  - computadores respondem seu horário
  - servidor tem conhecimento do tempo de propagação de cada computador ao servidor
  - quando tem os valores, faz uma média e informa os computadores a diferença dos relógios deles em relação a nova média
  - para fazer esta média ele considera somente relógios que não diferem mais de um certo valor (intervalo) - elimina problemas de relógios errados que possam causar grande efeito no horário global

Algoritmos Distribuídos (relógio)

- Algoritmo distribuído para sincronização: **média global**
  - Neste algoritmo cada processo *broadcasts* uma mensagem com o valor de seu relógio em um T0+iR, para todos os outros processos no sistema
    - T0 é um valor combinado no passado
    - i representa os intervalos que os processos irão sincronizar seus relógios
    - R representa o tamanho do intervalo para a sincronização acontecer
  - Após fazer o *broadcast*, o processo espera um intervalo T para receber as mensagens de outros processos
  - para cada mensagem, guarda o tempo local de sua recepção
  - Após o intervalo T, o processo então recalcula o seu relógio utilizando os valores que recebeu dos outros processos
    - calcula a diferença de si para cada um dos outros
    - calcula média destas diferenças e a usa para corrigir seu relógio
  - variações: descarte de valores fora de limite aceitável (altos ou baixos)

Algoritmos Distribuídos (relógio)

- Algoritmo distribuído para sincronização: **média local**
  - O algoritmo anterior necessita um sistema de *broadcast* para funcionar
  - Assim ele é bom para pequenas redes
  - No algoritmo de média local, cada processo troca informações com os processos vizinhos e calcula o seu novo horário a partir dos valores que recebeu de seus vizinhos

ALGORITMOS DISTRIBUÍDOS
Exclusão mútua

Algoritmos Distribuídos (exclusão mútua)

- Problema: recursos que não podem ser usados simultaneamente por diversos processos
- Acesso exclusivo deve ser provido pelo sistema
- Esta exclusividade é conhecida como *exclusão mútua*

Algoritmos Distribuídos (exclusão mútua)

- Um algoritmo que implementa exclusão mútua deve satisfazer os seguintes critérios:
  - *exclusão mútua*: dado um recurso compartilhado que pode ser acessado por diversos processos ao mesmo tempo, somente um processo pode acessar aquele recurso a qualquer momento
  - *starvation*: cada processo que requisita o recurso deve recebe-lo em algum momento

Algoritmos Distribuídos (exclusão mútua)

- **Algoritmo centralizado**
- Neste algoritmo, um processo do sistema é eleito como o coordenador e coordena as entradas na seção crítica (SC)
- Cada processo que deseja entrar em uma SC deve antes pedir autorização para o coordenador
- Se não existe processo acessando a SC, então o coordenador pode imediatamente garantir acesso ao processo que fez a requisição
- Se mais de um pede acesso à SC, então só um ganha acesso
- Após término do uso, processo informa coordenador
- Coordenador pode então liberar SC para outro processo (se existir)

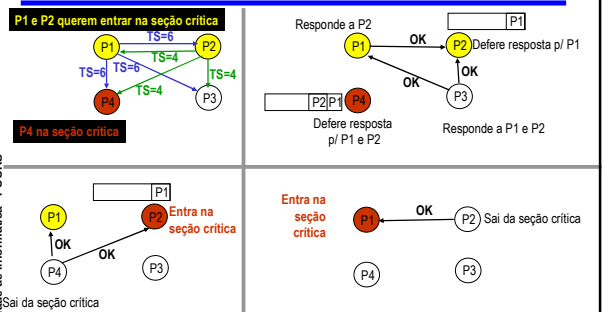
Algoritmos Distribuídos (exclusão mútua)

- **Algoritmo Distribuído**
- Se processo deseja acessar SC, então ele envia mensagem para todos os outros processos
- Mensagem contem:
  - identificador do processo
  - nome da SC que ele deseja acessar
  - um *timestamp* único gerado pelo processo que enviou a mensagem

Algoritmos Distribuídos (exclusão mútua)

- Ao receber mensagem, processo:
  - responde ao processo que enviou msg e garante acesso à SC se:
    - não quer acessar SC
    - quer acessar SC mas seu *timestamp* é maior que o *timestamp* do processo que enviou a mensagem
  - não responde se:
    - processo que recebeu mensagem está executando na SC
    - processo está esperando para acessar SC e seu *timestamp* é menor que o *timestamp* do processo que enviou a mensagem

Algoritmos Distribuídos (exclusão mútua)



Enunciado do 1ro Trabalho Prático

- Implementar o algoritmo de exclusão mútua distribuída baseado na garantia de acesso a partir da autorização de todos
- Supor relógios das máquinas sincronizados
- Trabalho deve ser feito na linguagem C, utilizando RPC
- O recurso compartilhado pelos diversos processos é um arquivo que está em uma área comum das diversas máquinas onde os processos vão ser executados. No início este arquivo tem um valor aleatório. Cada processo deve ler o último valor que se encontra neste arquivo, realizar uma operação sobre o valor, e armazenar o resultado na próxima posição (append). A seqüência de operações (o conteúdo do arquivo) deve constar no documento final a ser entregue. Cada vez que o processo acessa o recurso (arquivo) ele deve imprimir o valor que lá se encontrava, a operação que foi realizada, e o novo valor que lá foi armazenado. Cada processo deve realizar no mínimo 5 operações sobre o valor do arquivo. Cada processo deve ter uma operação diferente dos outros, de tal forma que o resultado de dois processos com base na mesma entrada é sempre diferente.
- Devem existir no mínimo 5 processos executando

Algoritmos Distribuídos (exclusão mútua)

- **Algoritmo baseado na passagem de *token***
  - Neste método, exclusão mútua é conseguida pelo uso de um *token* único que circula entre os processos do sistema
  - Um *token* é uma mensagem especial que dá ao detentor da mensagem direito de acesso à SC
  - Para que o algoritmo seja justo, os processos são organizados em um anel
  - O *token* circula entre os processos no anel sempre na mesma direção

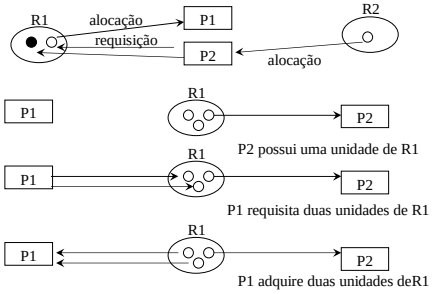
ALGORITMOS DISTRIBUÍDOS
Deadlock

Algoritmos Distribuídos *(deadlock)*

- Um *deadlock* é causado pela situação onde um conjunto de processos está bloqueado permanentemente, i.e., não conseguem prosseguir a execução, esperando um evento que somente outro processo do conjunto pode causar.
- Situação de *deadlock*:
  alocação de recursos formando ciclo
- Outra Situação de *deadlock:*
  - vários processos tentam alocar recursos para realizar suas tarefas, alocando o total de recursos de uma máquina.
  - Não podem acabar as tarefas por não terem recursos suficientes, não liberam recursos por não terem acabado as tarefas.
  - serializar a execução dos processos, não permitindo concorrência de processos que utilizem na soma mais que a quantidade de recursos disponíveis

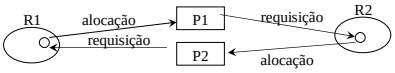
Algoritmos Distribuídos *(deadlock)*

- **Representação da relação entre processos e recursos**



Algoritmos Distribuídos *(deadlock)*

- **Deadlock**



Algoritmos Distribuídos *(deadlock)*

- **Condições necessárias para *deadlock*** [Coffman, et.al., 1971]
  - *exclusão mútua*: se recurso bloqueado por processo P, então outros processos tem que esperar processo P para usar o recurso;
  - *segura e espera*: processos podem requerer uso de novos recursos sem liberar recursos em uso;
  - *não preempção*: recurso se torna disponível somente pela liberação do recurso pelo processo;
  - *espera circular*: 2 ou mais processos formam uma cadeia circular na qual cada processo está a espera de um recurso bloqueado pelo próximo membro da cadeia.
- As quatro condições devem ser válidas ao mesmo tempo
- Se uma delas for quebrada, então *deadlock* não acontece no sistema

Algoritmos Distribuídos *(deadlock)*

- **Estratégias de tratamento de deadlock**
  - prevenir
    - estaticamente faz com que *deadlocks* não ocorram
  - detectar
    - permite que o *deadlock* ocorra, detecta e tenta recuperar

Algoritmos Distribuídos *(deadlock)*

- **Métodos para prevenir *deadlocks***
  - projetar o sistema de tal maneira que *deadlocks* sejam impossíveis
  - não necessita teste durante *run-time*
  - condições necessárias e suficientes:
    - exclusão mútua
    - segura e espera
    - não preempção
    - espera circular
  - técnicas: garantir que ao menos uma das condições não é nunca satisfeita
  - métodos: *collective requestes*, *ordered requests*, preempção

Algoritmos Distribuídos *(deadlock)*

- **Métodos para prevenir deadlocks**
  - *collective requests*
    - evita que condição segura e espera possa ser satisfeita
    - se um processo tem um recurso ele não pode ter outros
    - políticas
      - a) fazer *request* de todos recursos antes de sua execução: se todos recursos estão disponíveis, o processo pode executar senão, nenhum recurso é alocado e o processo espera
      - b) processo pode requerer recursos durante execução se ele liberar outros - tem que liberar todos e alocar recursos necessários até poder novamente liberar para depois alocar
    - *b* melhor que *a* pois processo pode não saber quantos recursos são necessários antes de começar a execução;

Algoritmos Distribuídos *(deadlock)*

- **Métodos para prevenir deadlocks**
  - *collective requests*
    - má utilização de recursos: processo pode ter vários recursos alocados e não usar alguns por longos períodos
    - *starvation*: se processo precisa de muitos recursos, cada vez que faz pedido pode encontrar um ou mais já alocados, tem que esperar e voltar a pedir .

Algoritmos Distribuídos *(deadlock)*

- **Métodos para prevenir deadlocks**
  - *ordered requests*
    - evita que espera circular aconteça
    - associa número a recurso
    - processo só pode alocar recursos em uma determinada ordem

Algoritmos Distribuídos *(deadlock)*

- **Métodos para prevenir deadlocks**
  - *ordered requests*

r = { r1, r2, r3, ... ,rn} // recursos
função f = r → N onde N é o conjunto dos números naturais
f (disco) = 1
f (fita) = 2
f (impressora) = 3

Processo que requisita Ri somente pode requisitar Rj se e somente se f (Rj) > f (Ri), caso contrário somente requisita Rj após liberar Ri

Algoritmos Distribuídos *(deadlock)*

- **Métodos para prevenir deadlocks**
  - preempção
    - possibilita preempção de recurso
    - recurso preemptável: estado pode ser facilmente salvo para ser restaurado depois (ex.: CPU e memória)
    - se processo precisa de recurso, faz *request*, se outro processo que detém recurso está bloqueado a espera de outro recurso, então recurso é preemptado e passado ao primeiro processo, caso contrário primeiro processo espera
    - durante espera recursos podem ser preemptados deste processo para que outros progridam
    - processo é desbloqueado quando recurso requerido, e qualquer outro recurso preemptado dele, possam ser bloqueados para o processo

Algoritmos Distribuídos *(deadlock)*

- **Métodos para detecção de deadlocks**
  - não evita nem previne, deixa que aconteçam e depois detecta para corrigir
  - algoritmo examina estado do sistema para determinar se existe *deadlock*
  - se existe - toma ação corretiva
  - técnica equivalente a sistema centralizado
  - mantém informação sobre alocação de recursos, formando grafo de alocação de recursos, e procurando ciclos neste grafo (grafo WFG - wait for graph)

Algoritmos Distribuídos *(deadlock)*

- **Algoritmo centralizado de detecção**
  - Em cada máquina se mantém um grafo de alocação de recursos pelos processos
  - Um coordenador centralizado mantém um grafo completo do sistema (a união dos grafos locais)
  - Quando o coordenador detecta um ciclo, ele mata um dos processos e acaba com o *deadlock*

Algoritmos Distribuídos *(deadlock)*

- Diferente de sistemas centralizados, onde as máquinas estão disponíveis automaticamente, em um sistema distribuído estas informações devem ser enviadas ao coordenador explicitamente
- Como cada máquina possui um grafo local, quando um arco é incluído ou excluído do grafo local, uma mensagem deve ser enviada para o coordenador
- Problema: quando uma mensagem demora para chegar, pode causar um **falso deadlock**

Algoritmos Distribuídos *(deadlock)*

- **Algoritmo distribuído de detecção**
  - Proposto por Chandy-Misra-Haas
  - Funciona da seguinte forma:
    - iniciado quando um processo tem que esperar um recurso alocado por outro processo
    - processo envia msg (*probe message*) para processo (ou processos) que está(ao) utilizando recurso
    - msg contém 3 informações:
      - número do processo que está bloqueado
      - número do processo que enviou a msg
      - número do processo que está recebendo a msg
    - quando a msg chega a um processo, ele verifica se está esperando por recurso
      - se sim a msg é atualizada e enviada para o processo que está usando o recurso
    - se a msg dá toda a volta e chega ao processo que iniciou a msg, um ciclo existe e o sistema está em *deadlock*

ALGORITMOS DISTRIBUÍDOS
Algoritmos de eleição

Algoritmos Distribuídos (eleição)

- Em sistemas distribuídos, diversos algoritmos necessitam que um processo funcione como coordenador, inicializador, sequenciador, enfim, ter um papel especial
- exemplos.:
  - coordenador de exclusão mútua com controle centralizado
  - coordenador para detecção de deadlock distribuído
  - seqüenciador de eventos para ordenação consistente centralizada
  - etc.
- falha do coordenador compromete serviço para vários processos
- novo coordenador deve assumir - eleição!
  - Objetivo: eleger ***um*** processo, entre os ativos, para desempenhar função especial

Algoritmos Distribuídos (eleição)

- Existem algoritmos especiais para escolha de um processo que assumirá o papel de coordenador
- Veremos dois *algoritmos de eleição*:
  - algoritmo do valentão (*bully algorithm*)
  - algoritmo em anel (*ring algorithm*)

Algoritmos Distribuídos (eleição)

- Para ambos os algoritmos, assume-se que: (*general assumptions*)
  - 1) Todo processo no sistema tem uma *prioridade única*
  - 2) Quando eleição acontece, o processo com maior prioridade *entre os processos ativos* é eleito como coordenador
  - 3) Na recuperação (volta à atividade), um processo falho pode tomar ações para juntar-se ao grupo de processos ativos

Algoritmos Distribuídos (eleição)

- ***Bully algorithm*** [Garcia-Molina em 1982]
  - Assume que um processo sabe a prioridade de todos outros processos no sistema
- Algoritmo
  - Quando um processo Pi detecta que o coordenador não está respondendo a um pedido de serviço, ele inicia uma eleição da seguinte forma:

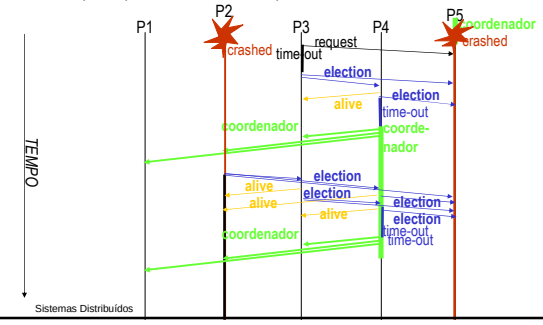
    a) Pi envia uma msg ELEIÇÃO para todos processos com prioridade maior que a sua

    b) se nenhum processo responde
    Pi vence a eleição e torna-se coordenador
    // significa que não há processos ativos com maior prioridade que a sua
    Pi manda uma msg COORDENADOR para os processos de menor prioridade informando que é o coordenador deste momento em diante.

    se processo Pj com prioridade maior que Pi responde (msg *ALIVE*)
    Pi não faz mais nada, Pj assume o controle
    Pj age como Pi nos passos **a)** e **b)**

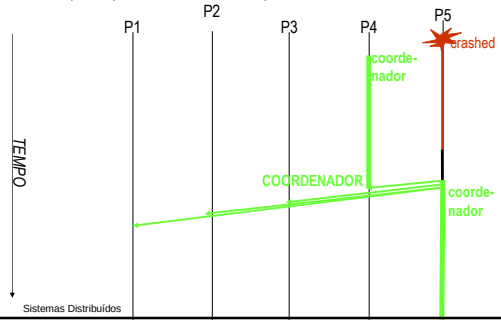
Algoritmos Distribuídos (eleição)

- ***Bully algorithm - exemplo [Pradeep, pg 334]***
  - suponha processos P1 a P5 com prioridade conforme seu identificador



Algoritmos Distribuídos (eleição)

- ***Bully algorithm - exemplo [Pradeep, pg 334]***
  - suponha processos P1 a P5 com prioridade conforme seu identificador



Algoritmos Distribuídos (eleição)

- ***Ring algorithm***
  - Baseado no uso de um anel lógico, sem uso de *token*
  - Cada processo conhece o anel inteiro, mas manda mensagens somente para o próximo processo ativo na direção do anel
  - Quando processo detecta que o coord. não está ativo, ele constrói uma msg ELEIÇÃO contendo seu id., e manda para o próximo do anel
  - A cada passo, o processo que recebe a msg inclui seu id na msg e envia para o próximo do anel
  - No final o processo que iniciou a eleição recebe a msg e escolhe aquele que tem maior id.
  - Nova msg é enviada novamente através do anel para todos contendo o novo coordenador
  - Uma vez que a msg passou por todos processos e chegou no originador, ela é retirada do sistema

Algoritmos Distribuídos (eleição)

- **Análise**
  - algoritmo do valentão (bully)
    - se processo com prioridade mais baixa detecta a falha do coordenador, em um sistema com n processos, então n-1 eleições acontecem
    - cada eleição tem mensagens conforme número de processos - O(n2) mensagens - *no pior caso*
    - se processo que detecta falha é o ativo de maior prioridade precisa só de n-2 mensagens - *melhor caso*
  - algoritmo do anel
    - eleição sempre precisa de 2(n-1) mensagens
    - n-1 mensagens para rotação da mensagem de eleição
    - n-1 mensagens para rotação da mensagem de coordenador