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)

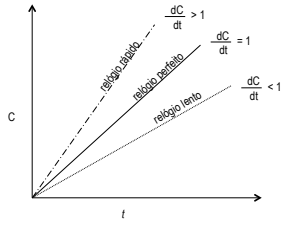
- **Relógio real**
- Era calculado baseado na rotação da terra em torno de seu eixo
- Terra com o passar do tempo está diminuindo sua velocidade de rotação
- Na idade média diversos dias (10 em 1582) tiveram que ser retirados do calendário devido a este problema
- Desde 1958, o relógio real é calculado 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 3 mseg. a cada 86.400 segundos – *leap second* (quando atinge 800 mseg.)

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)

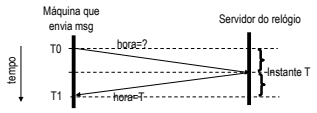
- Considerações
  - se dois relógios desviam em direções opostas, depois de um Δt da última sincronização, o desvio máximo entre eles é de: $2\rho.\Delta t$
  - para assumir que dois relógios nunca se diferenciam mais do que um valor σ (assumindo que relógios estão sincronizados se a diferença entre 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)

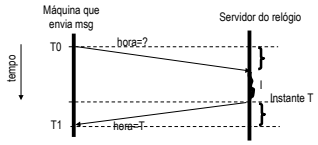
- Algoritmo centralizado para sincronização: **servidor passivo**
  - Servidor espera ser perguntado pela hora, e.g. "hora=?"
  - Após receber a mensagem informa "hora=T", onde T é a hora no servidor
  - Se cliente enviou a mensagem no tempo T0 e recebeu a resposta no tempo T1, então novo horário no 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: alguns recursos não podem ser usados simultaneamente por diversos processos (ex.: arquivos)
- Exclusividade de acesso deve ser garantida pelo sistema
  - esta exclusividade é conhecida como *exclusão mútua*
- Um algoritmo que implementa exclusão mútua deve satisfazer os seguintes critérios:
  - *exclusão mútua*: dado um recurso compartilhado que pode ser requisitado por diversos processos ao mesmo tempo, **somente um processo pode acessar aquele recurso** a qualquer momento
  - *no-starvation*: cada processo que requisita o recurso deve recebê-lo em algum momento
- Seção crítica: parte do código dos processos em que é realizado o acesso a algum recurso compartilhado que deve garantir acesso exclusivo.

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)

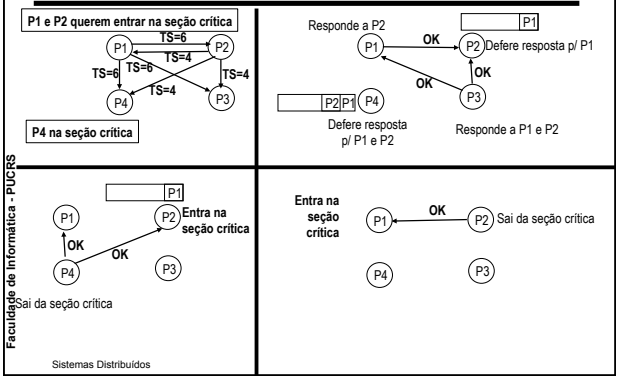
Algoritmos Distribuídos (exclusão mútua)

- **Algoritmo Distribuído**
  - Se processo deseja acessar SC, então ele envia mensagem para todos os outros processos
  - Mensagem contém:
    - 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)



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