Universidade Estadual de Mato Grosso dos Sul
Curso de Ciência da Computação
Disciplina de Algoritmos Paralelos e Distribuídos

Aula 7 – Sincronismo: Relógios Físicos, Relógios Lógicos, Exclusão Mútua e Eleição de Líder.

Relógios Físicos

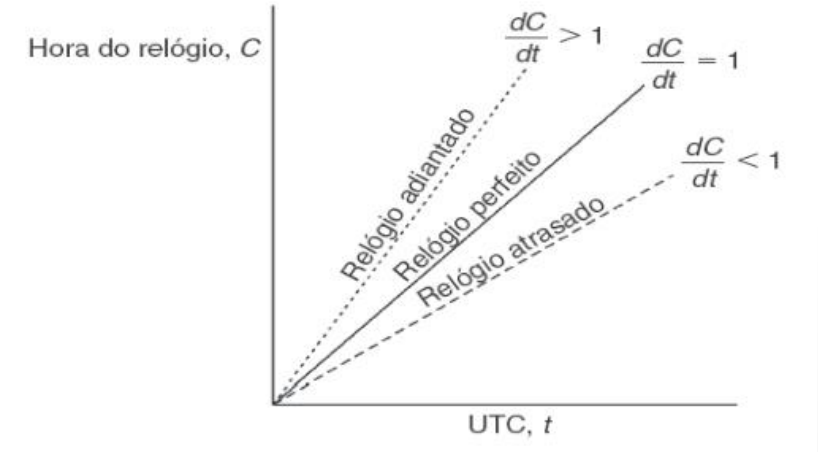
- A hora atômica internacional é baseada em relógios muito precisos (taxa de variação de 10^{-13})
- A hora coordenada internacional (HCI) é o padrão usado para a medição do tempo.
- É baseada na hora atômica, mas ocasionalmente é ajustada pela hora astronômica.
- É feito um BROADCAST da hora através de estações de rádio (www) e satélites (GPS).
- Computadores que recebem o sinal sincronizam os relógios.

Relógios Físicos

- Seja **C** o valor do relógio 'ideal' (hora UTC – Universal Coordinate Time)
- Seja **Cp** o valor do relógio no computador **p**
- Se **t** é o tempo no relógio em sincronia com a hora UTC, então temos o tempo em **p** dado por **Cp(t)**
- Idealmente, para todo **p** e para todo **t**, **Cp(t) = t** → **C'p(t) = dC/dt = 1**
- *Clock Skew* (defasagem do relógio): denota a magnitude de diferença entre as frequências de dois relógios.

Relógios Físicos

- Seja x a taxa máxima que deriva (que especifica se um relógio está atrasado ou adiantado).
- Como fazer a sincronização periódica entre os relógios??
- Exemplo: Relação entre a hora do relógio e a hora UTC quando as taxas de ciclos dos relógios são diferentes.



Sincronização de Relógios

- Se existir um "servidor de tempo" (um receptor wwc ou relógio de precisão)
  - Existem alguns algoritmos para isso: Cristian - 1998
- Se não existe uma fonte que disponibilize a hora coordenada universal (UCT)
  - Algoritmo de Berkeley
- O tempo exato não importa.
  - Relógios lógicos → Algoritmo de Lamport e relógios vetoriais.

Algoritmo de Cristian - 1998

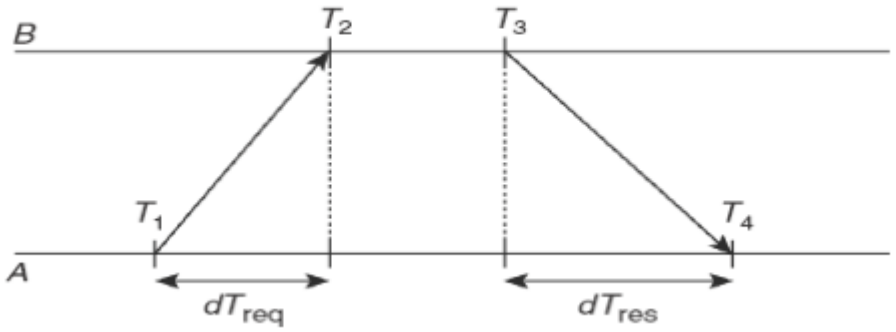
- Possibilidade de consultar servidores de tempo equipados com um receptor www ou um relógio de alta precisão.
- Problema: Atrasos de mensagens farão com que a hora fornecida seja desatualizada.
- Variações na estimativa do atraso entre as máquinas.

Algoritmo de Cristian

- Computador A consulta a hora no computador B
- Computador B inclui na resposta o valor do seu relógio → T3
- Computador A atualiza o seu relógio com o valor do relógio em B adicionado ao retardo da mensagem entre B e A (*one way delay*)
  - Estimativa para o one way delay = RTT / 2, caso atrasos em ambas as direções sejam aproximadas.

Algoritmo de Cristian

- Obtenção da hora corrente por meio de um servidor de tempo.



Algoritmo de Cristian

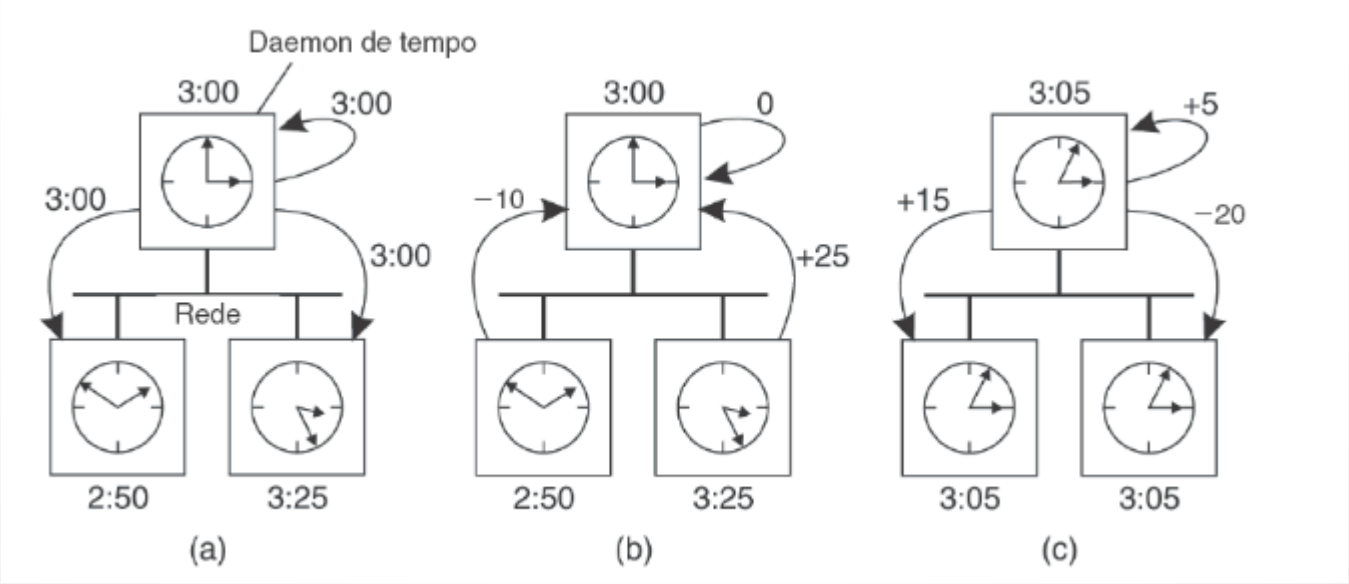
- Como estimar o *one way delay* (atraso em uma direção?)
  - RTT / 2
  - Protocolo *Network Time Protocol* (NTP)
    - Divisão dos computadores

Algoritmo de Berkeley

- Algoritmo usado para a sincronização interna de um grupo de computadores.
- "Servidor de tempo" é ativo (mestre) e coleta os valores de relógios de outras máquinas (escravos)
- O mestre usa as estimativas do RTT para estimar o valor dos relógios dos computadores dentro dos grupos.
- Hora atual é resultante de uma média
- Mestre envia aos escravos o total de tempo em que os relógios devem adiantar ou atrasar.
- Caso o mestre falhe, um novo computador mestre é eleito.

Algoritmo de Berkeley

- a) O *daemon* de tempo pergunta a todas as outras máquinas os valores marcados por seus relógios.
- b) As máquinas respondem
- c) O *daemon* de tempo informa a todas as máquinas como devem ajustar os seus relógios.



Considerações

- Algoritmos de Cristian e Berkeley são algoritmos centralizados.
- Algoritmos descentralizados também existem, como por exemplo, o *Network Time Protocol* (NTP)
  - Organizado em um estrutura hierárquica → estratos (evitar dependências cíclicas).
  - Um computador somente atualiza a sua hora, caso esteja em um estrato de nível inferior ao daquele que está trocando informações no momento.

Relógios Lógicos

- Sincronização baseada em tempo relativo.
- Tempo relativo não possui nenhuma relação com o tempo real.
- O mais importante é que os processos do Sistema Distribuído concordem com a ordem em que os eventos ocorrem.
  - Algoritmo de Lamport
  - Relógios Vetoriais

Algumas Definições

- Um sistema distribuído pode ser visto como uma coleção ***P*** de ***N*** processos p_i, $i = 1,2,3,...,N$
- Cada processo p_i possui um estado consistente s_i
  Os processos se comunicam através de mensagens
- Ações de um processo: enviar e receber mensagens, mudar o próprio *status*.
- Evento: ocorrência de uma ação associada ao processo.
- Eventos dentro de um processo p_i podem ser totalmente ordenados pela relação "acontece antes" ou seja, **a** → **b**, se e somente se, **a** ocorre antes de **b** em p_i.

Relógios Lógicos Lamport

- Ao invés da sincronização de relógios, aplica-se a ordenação dos eventos:

1. Se dois eventos ocorrem no mesmo processo, então eles na ordem observada pelo processo p_i
2. Quando uma mensagem **m** é trocada entre dois processos, e **a** é o evento de envio e **b** o de recebimento, então **a → b**
3. Relação **"acontece antes"** é transitiva.

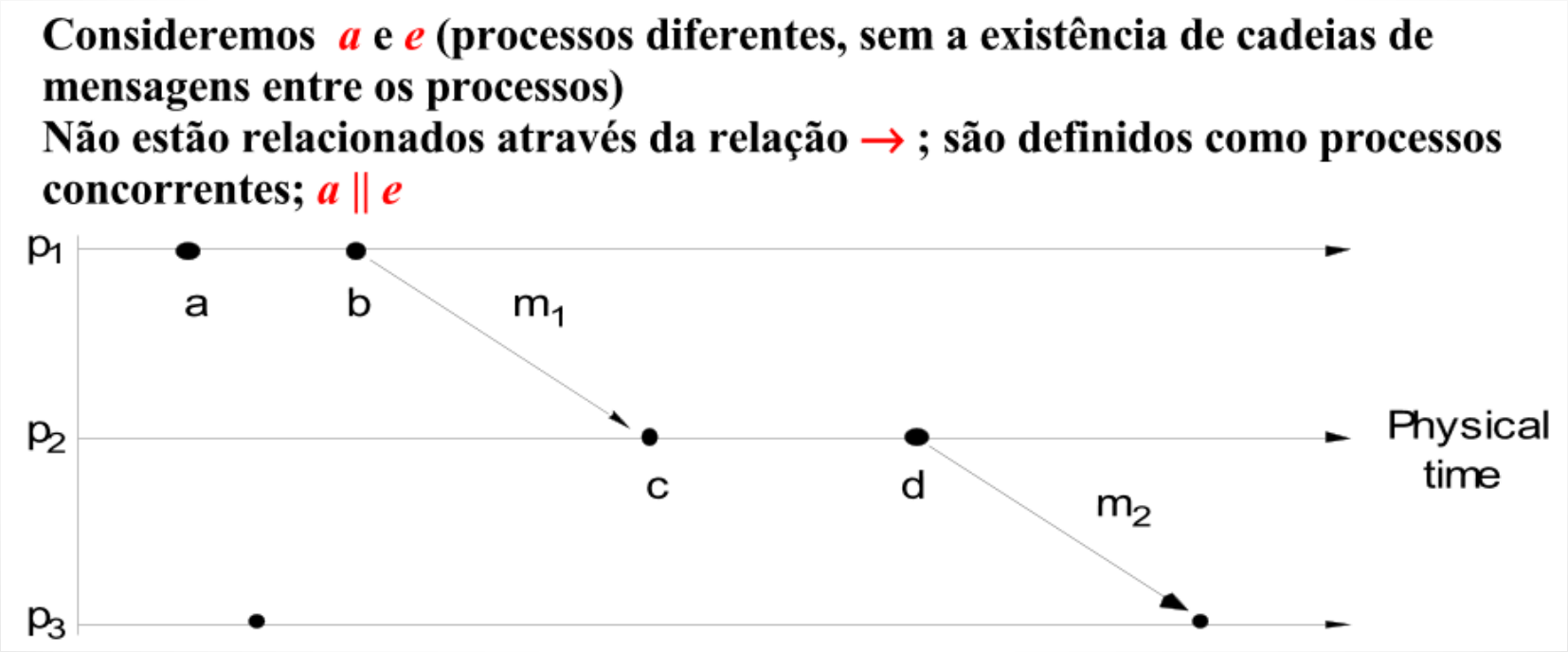
$a \rightarrow b\ (p1)\ c \rightarrow d\ \ (p2)$

$b \rightarrow c$ dado $m1$

$d \rightarrow f$ dado $m2$

Relógios Lógicos Lamport

- Nem todos os eventos podem ser relacionados através da relação "acontece antes" →



Relógios Lógicos Lamport

- Um relógio lógico é um contador monotonicamente crescente.
  - Não precisa estar relacionado com o relógio físico.

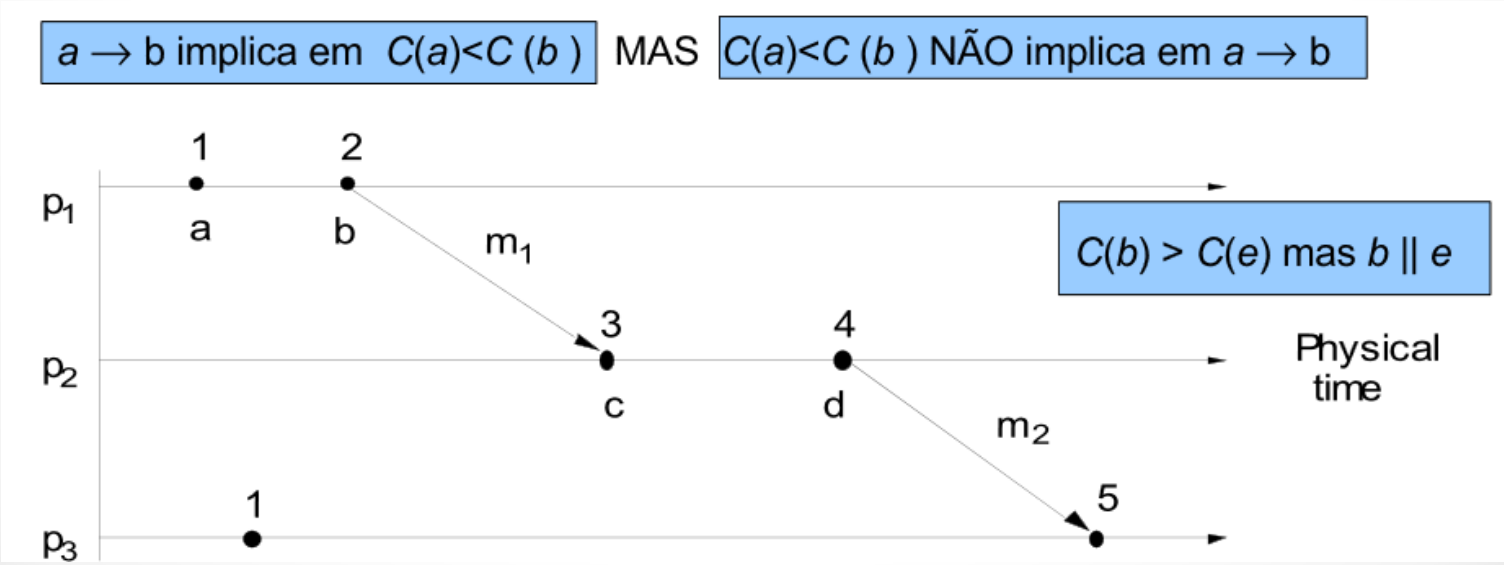
- Funcionamento:

Cada processo p_i tem o seu relógico lógico, C_i que pode ser usado para aplicar timestamps lógicos aos eventos

1) C_i é incrementado de 1 antes de cada evento no processo p_i
2) Quando um processo p_i envia mensagem m, o tempo $t = C_i$ é anexado a mensagem
3) Quando p_i recebe (m,t), o relógio é atualizado para $C_i := max(C_j, t)$ antes de aplicar 1)

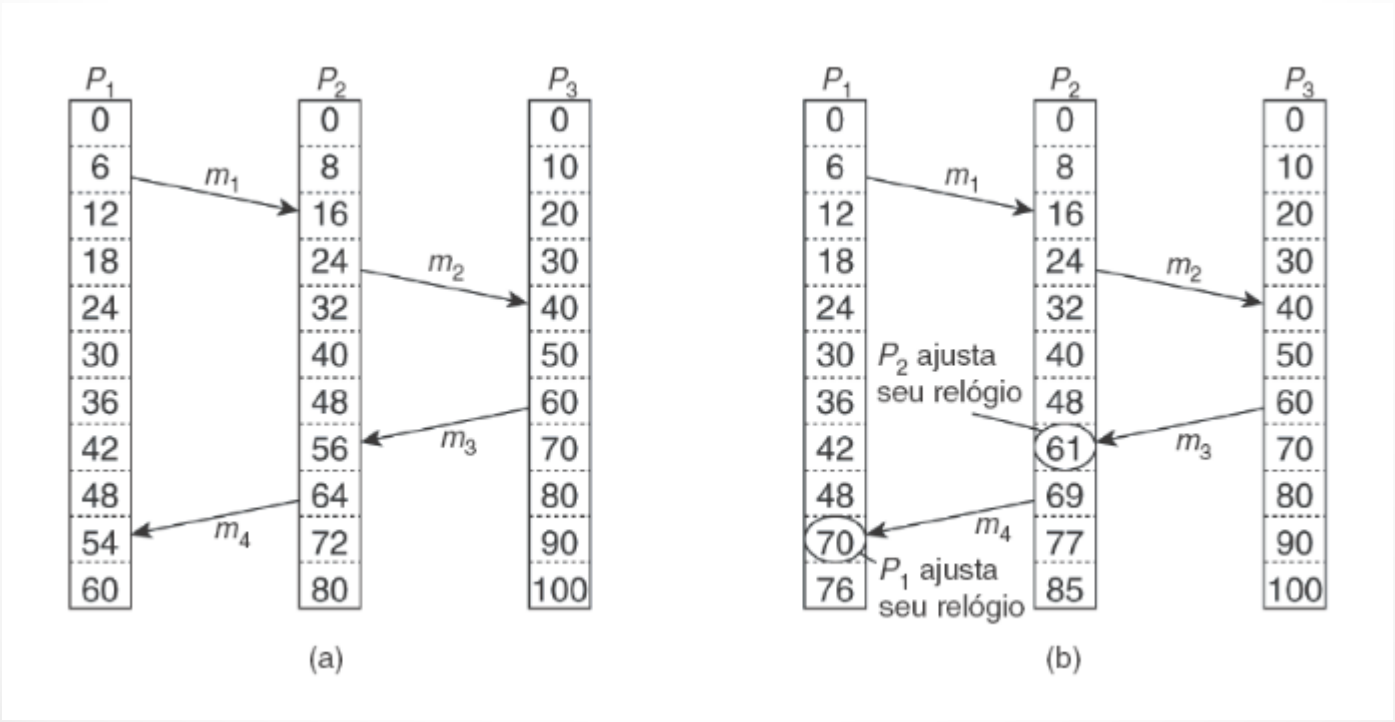
Relógios Lógicos Lamport

- Em cada um dos processos p1, p2 e p3 o relógio lógico é inicializado com zero.
- Os valores dos relógios lógicos são aqueles imediatamente após o evento, por exemplo, 1 para a, e 2 para b.
- Juntamente com m1, o valor 2 é enviado e o relógio em p2, após o evento c, recebe o max(0,2)+1=3

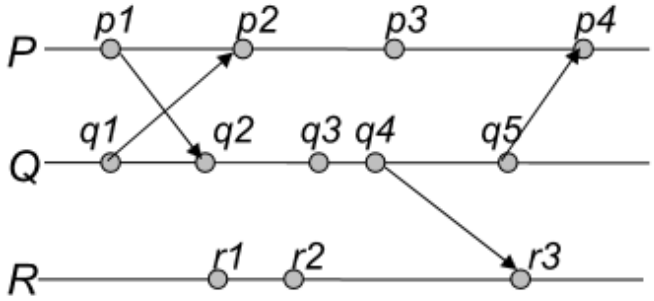


Relógios Lógicos Lamport

- a) três processos, cada um com seu próprio relógio. Os relógios funcionam a taxas diferentes.
- b) o algoritmo de Lamport corrige os relógios.



Relógio de Lamport Exemplo



Condição Inicial: $C(P) = 0$, $C(Q) = 2$, $C(R) = 0$
Processadores: $pid(P) = 0$, $pid(Q) = 1$, $pid(R) = 2$

Ordenamento parcial

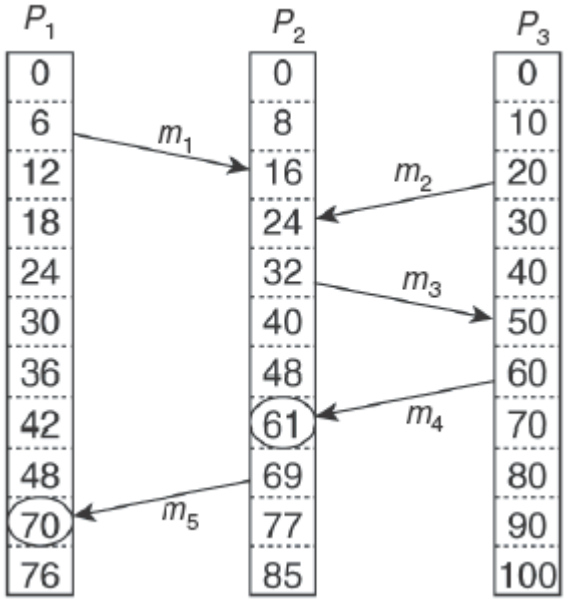
$p1 = 1,\ p2 = 4,\ p3 = 5,\ p4 = 8$
$q1 = 3,\ q2 = 4,\ q3 = 5,\ q4 = 6,\ q5 = 7$
$r1 = 1,\ r2 = 2,\ r3 = 7$

Ordenamento total

$p1 = 10,\ p2 = 40,\ p3 = 50,\ p4 = 80$
$q1 = 31,\ q2 = 41,\ q3 = 51,\ q4 = 61,\ q5 = 71$
$r1 = 12,\ r2 = 22,\ r3 = 72$

Relógio de Lamport Problema

- Exemplo: Transmissão de mensagens concorrentes com utilização de relógios lógicos.
- C(a) < C(b): não podemos inferir que a → b



Relógios Vetoriais

- Os relógios vetoriais foram implementados para evitar a limitação do relógio de Lamport: C(a)<C(b) não implica a "acontece antes" de b.
- Vetores com marcas de tempo são usados para os eventos locais em cada processo.
- Seja VC [i] o número de eventos ocorridos em p_i até o instante de tempo em questão.
- Seja VC [j] o número de eventos que ocorreram em p_j, portanto p_i sabe quantos eventos ocorreram em p_j.

Relógios Vetoriais

- Funcionamento:
  - Vetor de relógios VC_i no processo p_i é um vetor de N inteiros.
- Algoritmo:

1) Inicialmente $CV_i[j] = 0$ for $i, j = 1, 2, \ldots N$

***2)antes de cada evento,* p_i *executa* $CV_i[i] := CV_i[i] + 1$**

3) p_i envia $t = CV_i$ em cada mensagem transmitida

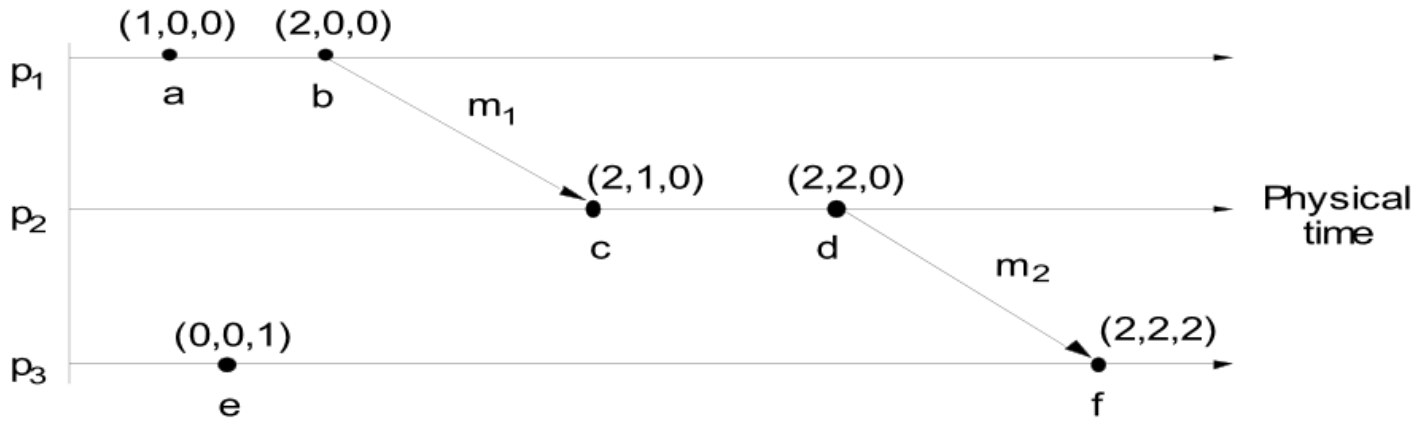
4) quando p_i recebe (m,t), o processo ajusta $CV_i[j] :=$ $\max(CV_i[j], t[j])$ $j = 1, 2, \ldots N$ *(antes do próximo evento adiciona 1 ao seu próprio contador de eventos)*

Relógios Vetoriais

***p*1: *a*(1,0,0); *b* (2,0,0) envia (2,0,0) juntamente com a mensagem *m*1**

Em *p*2, no recebimento de *m*1, o vetor de relógios é modificado para *max* ((0,0,0), (2,0,0)) = (2, 0, 0) adicionando 1 ao seu próprio relógio = (2,1,0)
Neste caso, o evento c 'sabe' que ocorreram 2 eventos no processo *p*1 antes da ocorrência do evento c em *p2*

=,<=, max: devem ser realizadas entre pares de elementos



Considerações

- Os algoritmos de Cristian e Berkeley sincronizam os relógios físicos, apesar da defasagem entre os relógios e os retardos das mensagens.
- Para ordenar eventos em computadores diferentes, nem sempre será possível fazer a sincronização dos processos.
- A relação "acontece antes" resulta em uma ordenação parcial dos eventos.
- Relógios de Lamport são contadores que mudam de acordo com o relacionamento de "acontece antes" entre os eventos.
- Relógios vetoriais representam uma melhora nos relógios de Lamport, onde dois eventos são ordenados pela relação "acontece antes" ou são concorrentes através da comparação dos vetores com marcas de tempo.

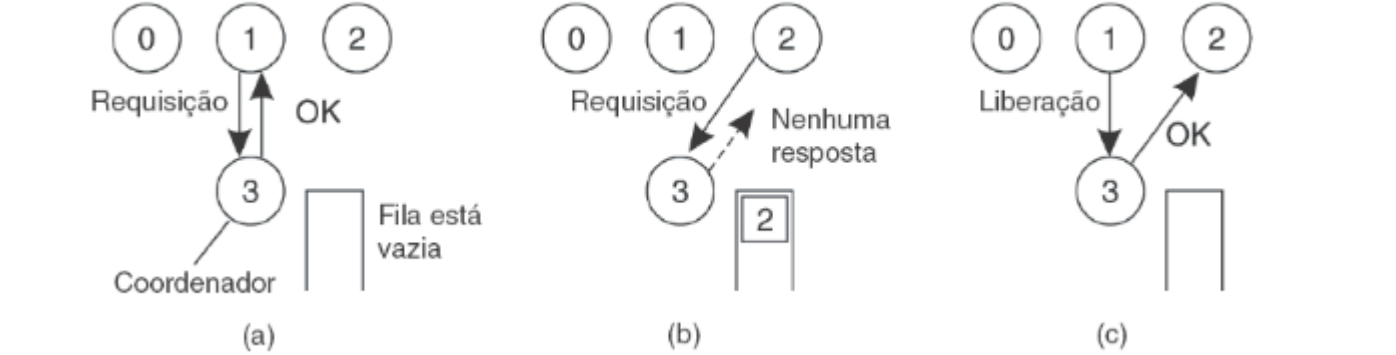
Exclusão Mútua

- A exclusão mútua surge diante de um problema onde um conjunto de processos, em um sistema distribuído, necessitam garantir acesso exclusivo a um recurso.

- Algumas soluções:
  - Servidor centralizado;
  - Completamente distribuído sem topologia específica
  - Completamente distribuído com uma topologia lógica em anel.

Exclusão Mútua Servidor Centralizado

- a) processo 1 solicita ao coordenador permissão para acessar um recurso compartilhado. A permissão é concedida.
- b)Depois, o processo 2 solicita permissão para acessar o mesmo recurso. O coordenador não responde.
- c) Quando o processo 1libera o recurso, informa ao coordenador, que então responde a 2.

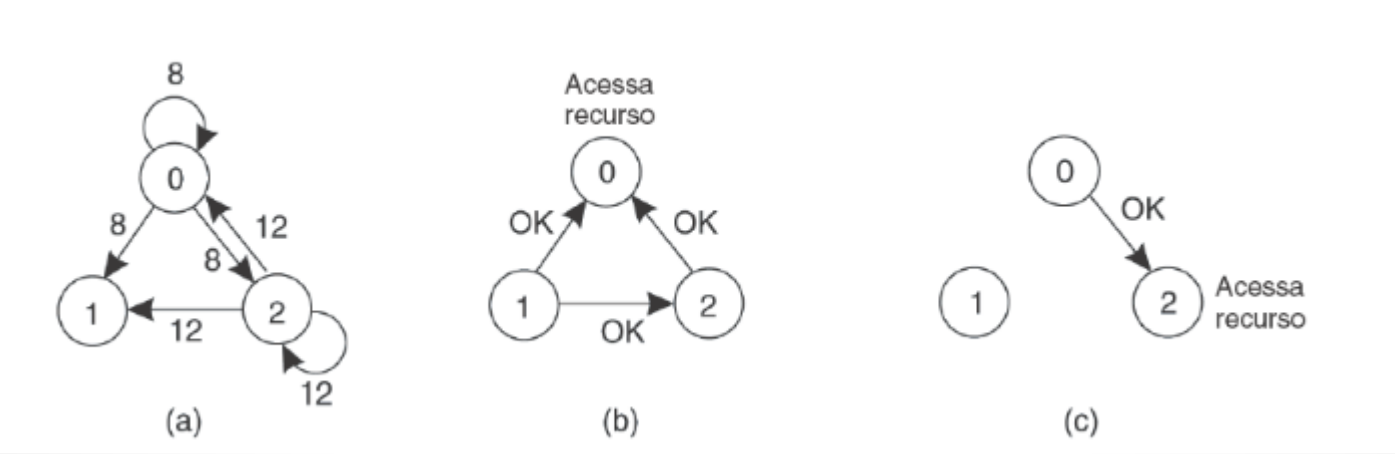


Exclusão Mútua Algoritmo Distribuído

- Funcionamento:
- A) Processo precisa acessar uma região crítica
  - Envia uma mensagem a todos os outros processos:
    - Nome do recurso, número do processo e tempo corrente.
  - Todas as mensagens devem ser confirmadas.
- B) Processo recebe uma requisição:
  - A ação depende do estado no qual se encontra o processo ao receber a requisição
    - Se o processo não está acessando o recurso e nem pretende fazê-lo, envia um OK.
    - Se estiver na região crítica, a requisição é enviada a uma fila.
    - Se não está acessando o recurso, mas pretende fazê-lo, então é feita uma comparação entre os tempos de geração dos pedidos. O menor ganha o acesso.

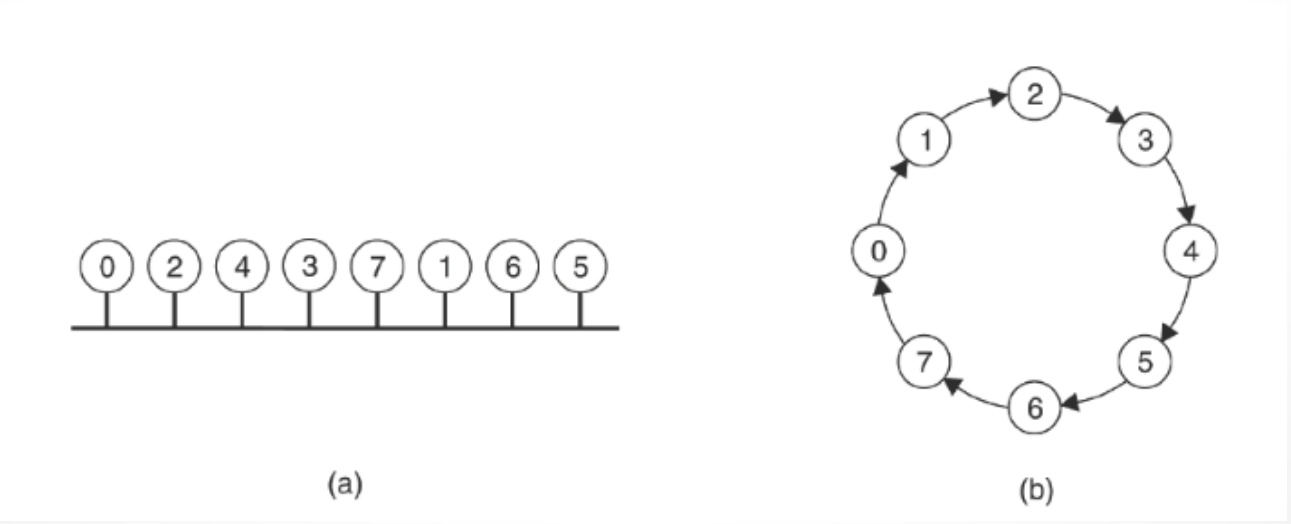
Exclusão Mútua Algoritmo Distribuído

- a) Dois processo querem acessar um recurso compartilhado no mesmo momento.
- b) O processo 0 tem a marca de tempo mais baixa, portanto vence.
- c) Quando o processo 0 conclui, também envia uma mensagem OK. O processo 2 pode seguir adiante.



Exclusão Mútua Algoritmo Token Ring

- Funcionamento:
  - Organiza os processos em um anel lógico e um token circula entre os processos que fazem parte do sistema distribuído.
  - Ao processo que está com o token é permitido utilizar o recurso 'crítico', caso queira usá-lo.
  - Exemplo: a) Grupos de processos não ordenados em uma rede
    - b) Um anel lógico é construído em software.



Algoritmos de Eleição

- Muitos SDs precisam que um processo aja como coordenador ou outro papel especial.
- Em geral, não importa qual processo assume a responsabilidade.
- Algoritmos para eleger um coordenador
  - Geralmente tentam localizar o processo com maior ID
  - Assume-se que todos os processos conhecem os IDs dos demais processos
    - Não se sabe se estão disponíveis ou indisponíveis
- Objetivo: fazer com que todos concordem com a eleição do novo coordenador.

Algoritmo do *Bully* (valentão)

- Quando o processo nota que o coordenador não está mais respondendo requisições, inicia-se a eleição:
  - Processo P convoca uma eleição:
    1. P envia mensagens de eleição para todos os processos com IDs maiores
    2. Se ninguém responde, P vence a eleição e torna-se coordenador.
    3. Se algum processo com ID maior responde, ele desiste.
  - Quando o processo recebe msgs de eleição de membros com IDs mais baixa
    - Envia OK para remetente para indicar que está vivo e assume o processo.

Algoritmo do *Bully* (valentão)

- Eventualmente, todos os processos desistem menos o processo do novo coordenador.

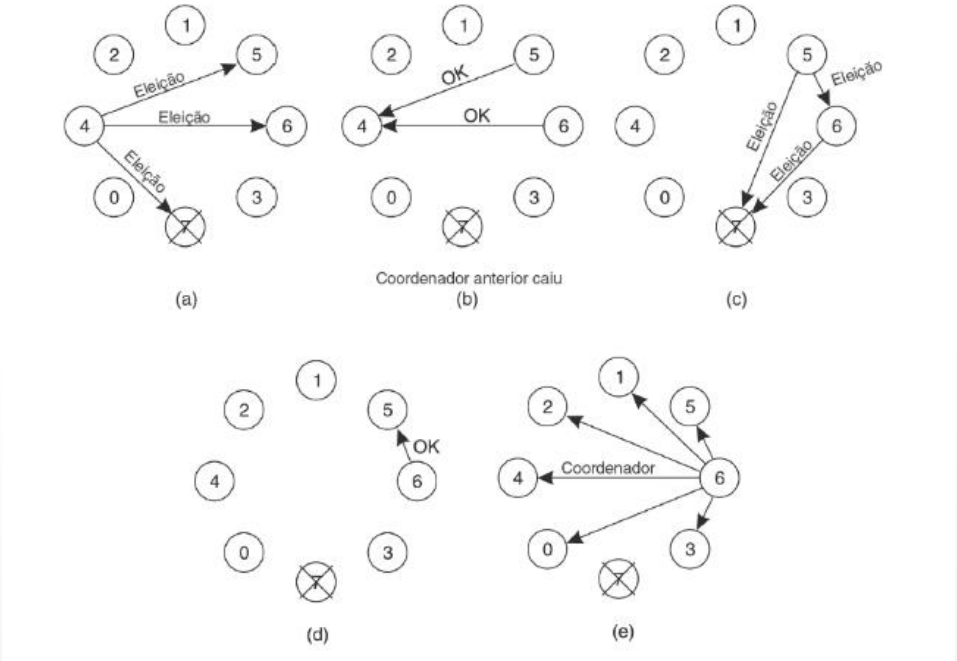
- Se o processo que estava indisponível voltar, reinicia a eleição:

  - Se for processo com maior ID, vence e retoma a coordenação

  - De modo geral o processo com o maior ID (o grandão) sempre vence.

Algoritmo do *Bully*

- Exemplo: Algoritmo de eleição do valentão
  - a) O processo 4 convoca uma eleição.
  - b) Os processos 5 e 6 respondem e mandam 4 parar.
  - c) Agora, cada um, 5 e 6 convoca uma eleição
  - d) O processo 6 manda 5 parar
  - e) O processo 6 vence e informa a todos.

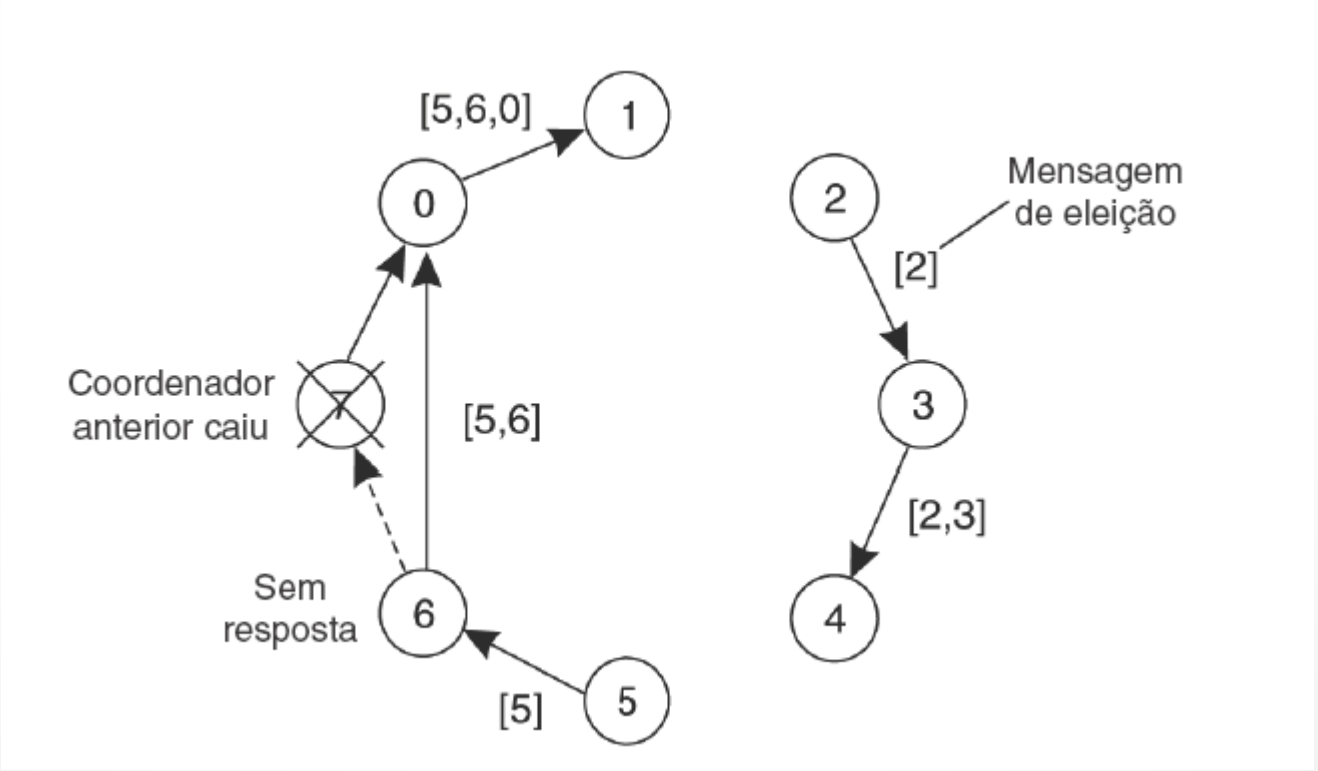


Algoritmo em Anel

- Uso de uma topologia em anel, porém sem token
- Processos fisicamente ou logicamente ordenados
  - Desta forma, eles conhecem sempre o sucessor.
- Um processo identifica que o coordenador não funciona
  - Envia uma msg de eleição ao sucessor
  - Msg contém seu ID
  - Se sucessor indisponível, manda para o próximo e, assim, sucessivamente
  - A cada passo, o processo que recebe a msg adiciona o seu ID e repassa para o sucessor.
- Eventualmente, a msg retorna para aquele processo que começou a eleição.
  - Isto é reconhecido porque a msg de eleição contém o seu ID
  - Neste ponto, ele muda msg para coordenador e a circula novamente para informar a todos
    - Novo coordenador: processo com maior ID
    - Nova configuração do anel
  - Quando a msg coordenador volta para o processo que iniciou a eleição, a msg é retirada do anel e todos voltam a trabalhar.

Algoritmo em Anel

- Exemplo de algoritmo de eleição que usa uma topologia em anel.



Leitura, Estudo e Exercícios

- Este material tem sua fonte no livro:
  - Sistemas Distribuídos: Princípios e Paradigmas.
  - Andrew Tanenbaum e Marteen Steen
  - Capítulo 6

  - Distributed System – Concepts and Design
  - George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair.
  - Capítulos 14