# Sistemas Distribuídos

Aula 7 – Sincronização (cont.)

DCC/IM/UFRRJ
Marcel William Rocha da Silva

# Objetivos da aula

- **Aula anterior**
  - Relógios físicos
  - Relógios lógicos
    - Relógios de Lamport
    - Relógios vetoriais
- **Aula de hoje**
  - Algoritmos de exclusão mútua
  - Algoritmos de eleição

# Sincronização

- Sincronização em relação ao tempo
  - **Relógios Físicos e Lógicos**
- Sincronização em relação ao compartilhamento de recursos
  - **Exclusão Mútua**
- Sincronização em relação ao coordenador de um grupo
  - **Algoritmos de Eleição de Líder**

# Exclusão mútua

- Questão importante em SDs:
  - Concorrência e colaboração entre vários processos
  - Processos vão precisar acessar simultaneamente os mesmos recursos
  - Acessos concorrentes podem corromper o recurso ou torna-lo inconsistente → **acesso mutuamente exclusivos**

# Exclusão mútua

- Dois tipos de solução:
  - **Baseadas em token**
    - Passagem de uma mensagem especial (*token*) entre os processos
    - De posse do *token*, o recurso pode ser acessado
    - Ao término, *token* é repassado
    - Evita inanição (*starvation*) e bloqueios (*deadlocks*)
    - Problema: perda da mensagem de *token*
  - **Baseadas em permissão**
    - Antes de acessar um recurso, o processo primeiramente deve solicitar permissão aos outros processos
      - Versões centralizada e distribuída

# Exclusão mútua: centralizada

- Modo mais direto de conseguir exclusão mútua: simular sistema monoprocessador

- Coordenador (líder):
  - Para um processo acessar um recurso compartilhado → envia mensagem de requisição ao líder
  - Se recurso estiver livre → líder devolve uma resposta (Ok) concedendo permissão
  - Se recurso já estiver em uso → pedido é armazenado em uma fila

# Exclusão mútua: centralizada

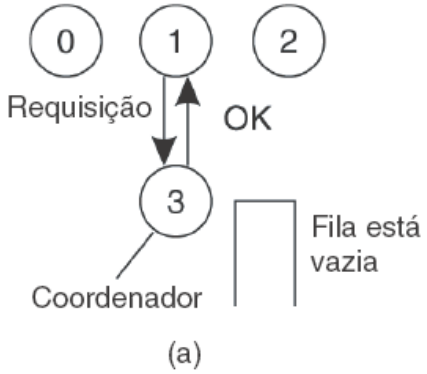



**Figura 6.14** *(a) O processo 1 solicita ao coordenador permissão para acessar um recurso compartilhado. A permissão é concedida. (b) Depois, o processo 2 solicita permissão para acessar o mesmo recurso. O coordenador não responde. (c) Quando o processo 1 libera o recurso, informa ao coordenador, que então responde a 2.*

# Exclusão mútua: centralizada

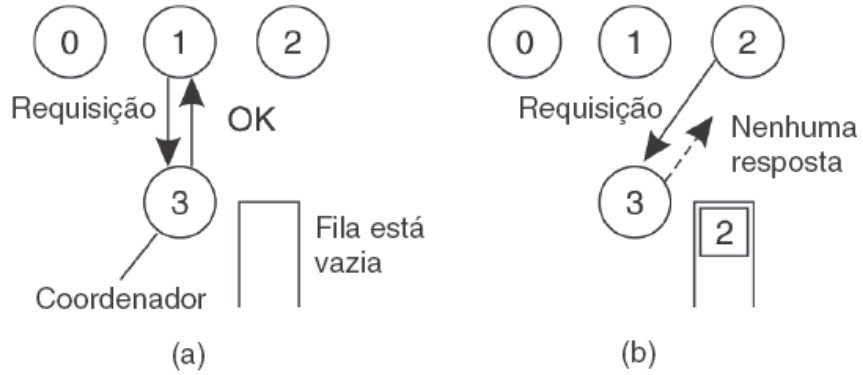



**Figura 6.14** *(a) O processo 1 solicita ao coordenador permissão para acessar um recurso compartilhado. A permissão é concedida. (b) Depois, o processo 2 solicita permissão para acessar o mesmo recurso. O coordenador não responde. (c) Quando o processo 1 libera o recurso, informa ao coordenador, que então responde a 2.*

# Exclusão mútua: centralizada

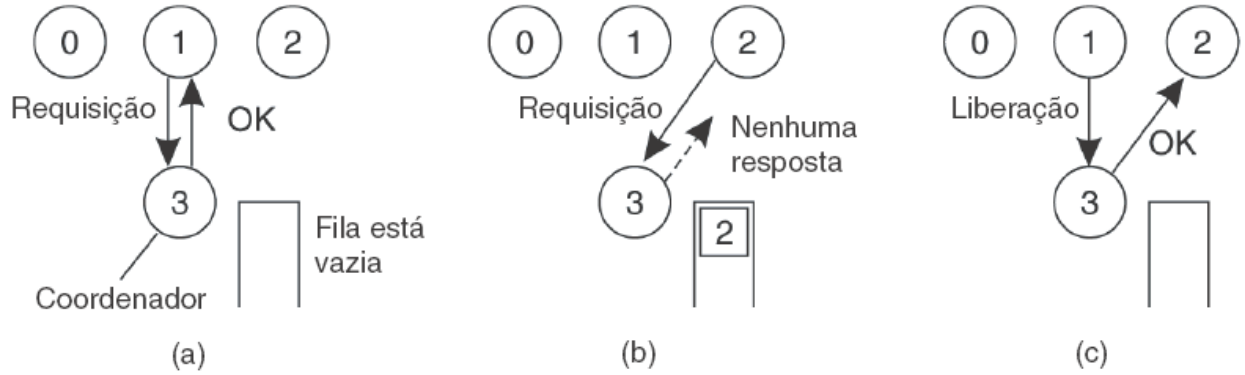



**Figura 6.14** *(a) O processo 1 solicita ao coordenador permissão para acessar um recurso compartilhado. A permissão é concedida. (b) Depois, o processo 2 solicita permissão para acessar o mesmo recurso. O coordenador não responde. (c) Quando o processo 1 libera o recurso, informa ao coordenador, que então responde a 2.*

# Exclusão mútua: centralizada

- Discussão sobre o caso centralizado:
  - Garante exclusão mútua: apenas um processo acessa o recurso por vez
  - Garante justiça: permissões são concedidas na ordem em que as requisições foram recebidas
  - Simples de implementar: requer somente três mensagens a cada utilização do recurso (requisição, concessão, liberação)
  - Problema: coordenador é um único ponto de falha!
    - Como identificar um coordenador inativo de uma permissão negada?

# Exclusão mútua: distribuída

- Trabalho de Ricart e Agrawala (1981)
  - Depende da ordenação total dos eventos: uso do algoritmo de Lamport!
- Quando um processo quer acessar um recurso compartilhado:
  - Cria uma mensagem de **requisição** que contém o **nome do recurso**, seu **número de processo** e a **hora corrente** (relógio de Lamport)
  - Envia a **requisição** para todos os outros processos
  - Premissa: mensagens não se perdem

# Exclusão mútua: distribuída

- Quando um processo recebe uma **requisição**
  - Se o receptor não estiver acessando o recurso e não quiser acessá-lo: envia mensagem de **OK**
  - Se o receptor já tiver acesso ao recurso: apenas coloca a requisição em uma **fila**
  - Se o receptor também quiser acessar o recurso: compara a hora da msg recebida com a hora da msg que enviou para todos (vence a menor)
    - Caso a hora da msg que chegou seja menor, envia **OK**
    - Caso contrário, enfileira a requisição

# Exclusão mútua: distribuída

- Após enviar uma **requisição**, processo espera pelo **OK** de todos os outros processos
  - Ao receber todos os **OK's**: acessa o recurso

- Logo após o uso do recurso:
  - Envia mensagem de **OK** e remove cada um dos processos que estão em sua fila

# Exclusão mútua: distribuída

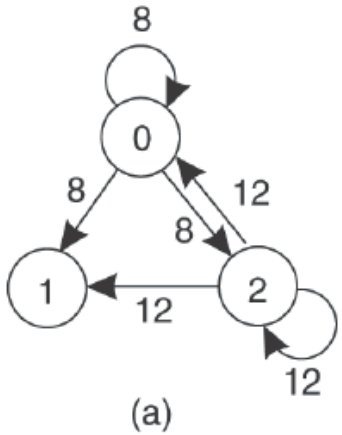



**Figura 6.15** (a) Dois processos querem acessar um recurso compartilhado no mesmo momento. (b) O processo 0 tem a marca de tempo mais baixa, portanto vence. (c) Quando o processo 0 conclui, também envia uma mensagem *OK*, portanto, agora, 2 pode seguir adiante.

# Exclusão mútua: distribuída

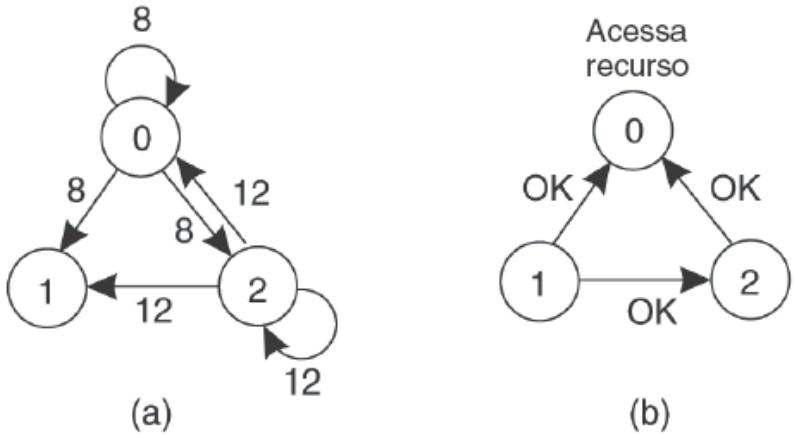



**Figura 6.15** *(a) Dois processos querem acessar um recurso compartilhado no mesmo momento. (b) O processo 0 tem a marca de tempo mais baixa, portanto vence. (c) Quando o processo 0 conclui, também envia uma mensagem OK, portanto, agora, 2 pode seguir adiante.*

# Exclusão mútua: distribuída

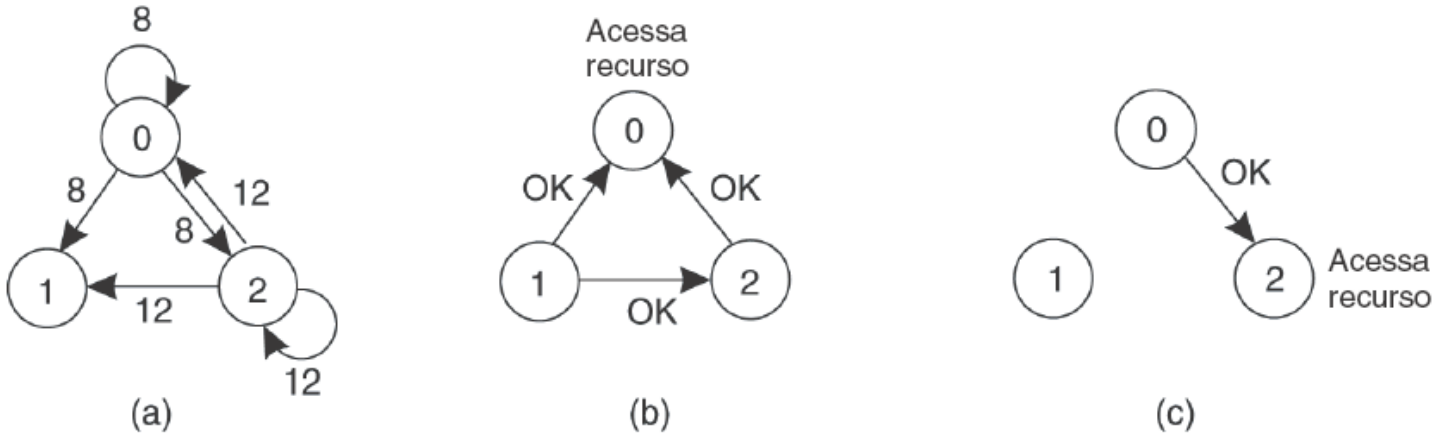



**Figura 6.15** *(a) Dois processos querem acessar um recurso compartilhado no mesmo momento. (b) O processo 0 tem a marca de tempo mais baixa, portanto vence. (c) Quando o processo 0 conclui, também envia uma mensagem OK, portanto, agora, 2 pode seguir adiante.*

# Exclusão mútua: distribuída

- Discussão sobre o caso distribuído:
  - Exclusão mútua é garantida sem starvation e deadlock
  - Número de mensagens para cada pedido de permissão é de 2(n-1), onde o número total de processos no sistema é n
  - Problema: n pontos de falha → se qualquer processo falhar, não responderá as requisições, que será interpretada como recusa!

# Exclusão mútua: distribuída

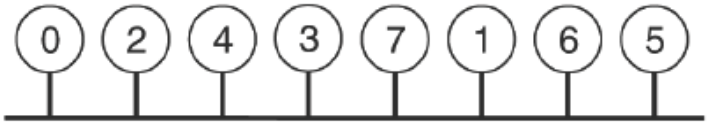
- Discussão sobre o caso distribuído:
  - Como identificar recusa ou falha?
    - Uso de temporizador
    - Toda requisição será respondida com mensagem de OK ou recusa
    - Se uma requisição ou resposta se perde → remetente esgota a temporização de espera e tenta novamente
    - Após um determinado número de tentativas o destinatário é considerado inativo

# Exclusão mútua: distribuída

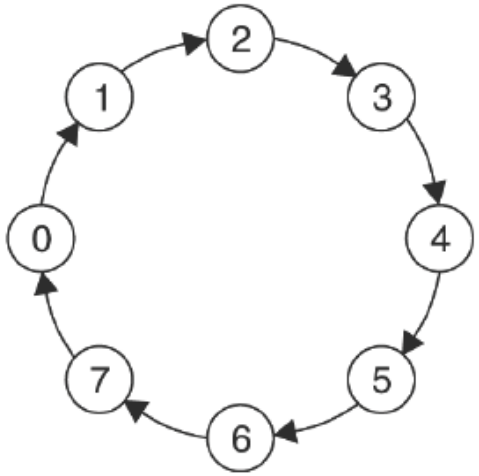
- Discussão sobre o caso distribuído:
  - Outras desvantagens:
    - Maior número de mensagens trocadas
    - Maior probabilidade de falhas
  - Apesar disso:
    - Importante para mostrar que, apesar das desvantagens, é possível ter um algoritmo distribuído que efetivamente promove a exclusão mútua

# Exclusão mútua: Token ring

- Solução distribuída usando **token**
- **Anel lógico** é construído em software
  - A cada processo é designada uma posição no anel
  - Cada processo deve saber a quem repassar o token




(a)




(b)

Figura 6.16 *(a) Grupo de processos não ordenados em uma rede. (b) Um anel lógico é construído em software.*

# Exclusão mútua: Token ring

- Quando o anel é inicializado, o processo de menor identificador recebe o token
- Token é repassado do processo **k** para o **k+1** através de msgs ponto-a-ponto (*unicast*)
- Quando um processo recebe o token: verifica se precisa acessar o recurso compartilhado
  - Acessa o recurso e repassa o token ao próximo processo
  - Caso contrário, simplesmente repassa o token

# Exclusão mútua: Token ring

- Discussão sobre o token ring:
  - Garante a exclusão mútua → só um processo tem o token a qualquer instante
  - Evita a ocorrência de inanição (*starvation*)
  - Problemas:
    - Se o token se perder, deve ser gerado novamente
    - Se um processo falha, deve ser removido do anel e o token é repassado para o próximo vizinho lógico

# Comparação entre algoritmos

| Algoritmo | Mensagens por entrada/saída | Atraso antes da entrada (em número de tempos de mensagens) | Problemas |
|---|---|---|---|
| Centralizado | 3 | 2 | Queda do coordenador |
| Distribuído | $2\ (n-1)$ | $2\ (n-1)$ | Queda de qualquer processo |
| Token Ring | 1 a $\infty$ | 0 a $n-1$ | Ficha perdida; processo cai |

# Algoritmos de eleição

- Muitos algoritmos distribuídos requerem um processo especial (coordenador ou líder)

- Considerações iniciais:
  - Todos os processos são iguais (sem característica distintiva)
  - Cada um possui um **identificador** (ID ou endereço IP)
  - Todo processo conhece os IDs dos outros processos
  - Não se sabe a priori quais os processos ativos no momento

- **Algoritmo de eleição** → localizar o processo ativo com maior identificador para atuar como líder
  - Todos devem concordar com a escolha

# Eleição: Algoritmo do valentão (bully)

- Proposto por Garcia-Molina (1982)
- Quando processo P verifica a falta do líder (falha ou condição inicial), ele inicia uma nova eleição
- Processo P convoca uma eleição:
  - P envia msg de **eleição** para todos os processos com IDs maiores
  - Se ninguém responde, P vence eleição e torna-se líder
  - Se algum processo com ID maior responde, ele desiste
- Quando processo P recebe msg de eleição de membros com ID menor:
  - Envia **OK** ao remetente indicando estar ativo
  - Convoca nova eleição

# Eleição: Algoritmo do valentão (bully)

- Eventualmente todos os processos desistem menos um → novo líder
- Se processo que estava indisponível volta → inicia nova eleição
  - Se for processo com maior ID, vence e toma coordenação
  - O maior sempre vence → valentão!

# Eleição: Algoritmo do valentão (bully)

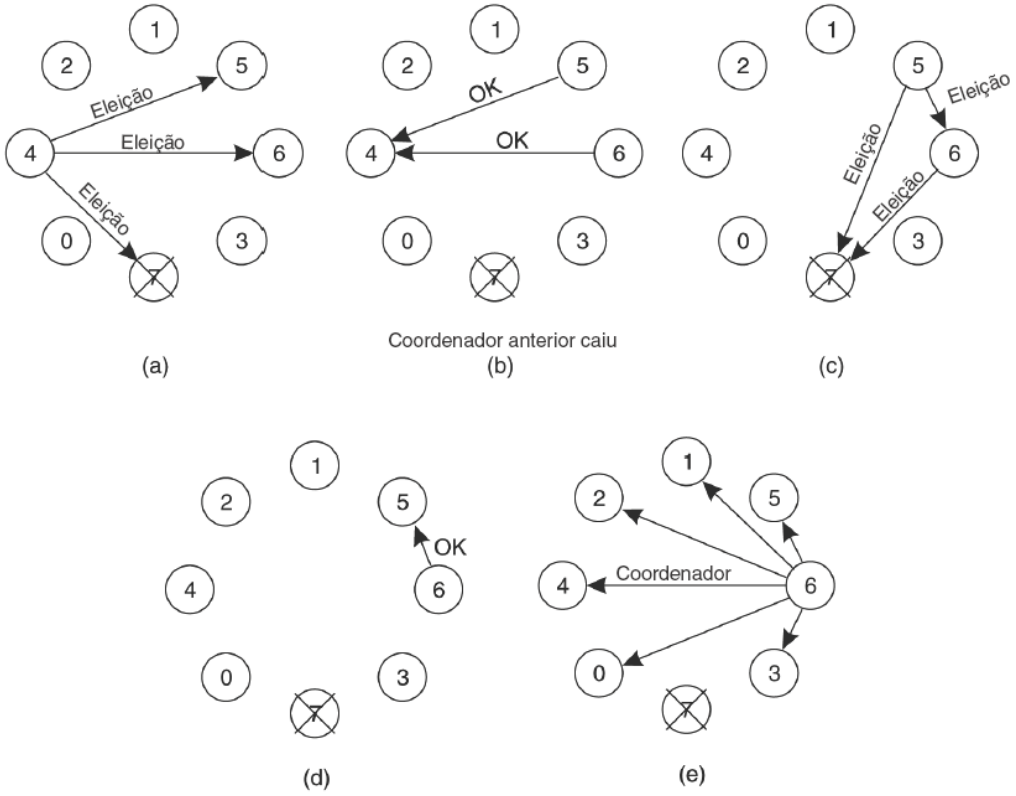



**Figura 6.19** *Algoritmo de eleição do valentão. (a) O processo 4 convoca uma eleição. (b) Os processos 5 e 6 respondem e mandam 4 parar. (c) Agora, cada um, 5 e 6, convoca uma eleição. (d) O processo 6 manda 5 parar. (e) O processo 6 vence e informa a todos.*

# Eleição: Algoritmo do anel

- Também utiliza um anel lógico (mas não usa token)
- Qualquer processo inicia a eleição
  - Monta uma mensagem ELEIÇÃO, com o seu próprio identificador e envia ao seu sucessor no anel
  - Caso o sucessor esteja inativo, o remetente tenta o próximo processo no anel
  - Cada processo adiciona o seu número a lista na msg ELEIÇÃO ao repassar
  - Ao retornar a origem, líder é definido (maior ID dentre os processos na lista)
  - Msg COORDENADOR é enviada no anel, indicando novo líder e a lista de processos ativos

# Eleição: Algoritmo do anel

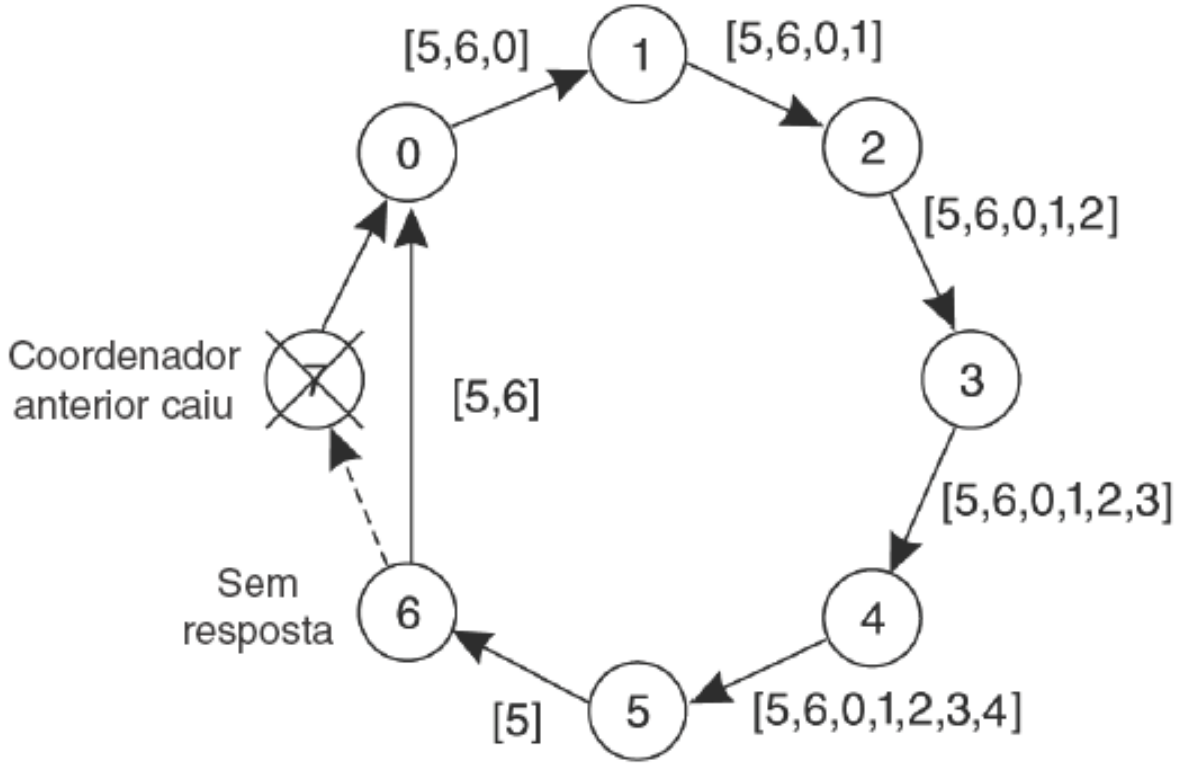



Figura 6.20 *Algoritmo de eleição que usa um anel.*