

Sistemas Operacionais Distribuídos e de Redes

Comunicação em grupo

Aula 05

- ❑ Tolerância a falhas baseada em replicação de serviços
 - ▶ Serviço replicado é feito por um grupo de servidores
 - ▶ Requisição do cliente é enviado ao grupo (todos executam mesma operação)
- ❑ Serviços de descoberta
 - ▶ Publicar interfaces
 - ▶ Localização de serviços providos
- ❑ Desempenho via replicação de dados
 - ▶ Multicast é usado para atualização de valores nas réplicas
- ❑ Notificação de eventos

Introdução

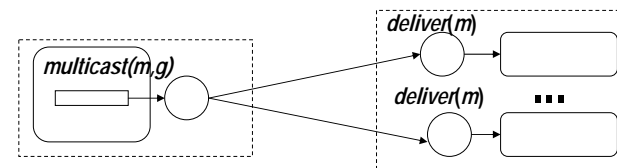
- ❑ Comunicação ponto a ponto nem sempre é a melhor opção
 - ▶ Ex: serviço realizado por um conjunto de máquinas (tolerância a falhas e/ou disponibilidade)
- ❑ *Multicast*: envio de uma única mensagem para um grupo de processos
 - ▶ Identificação individual dos membros não é importante (nem desejável)
 - ▶ Participação em um grupo é transparente
- ❑ Questões importantes:
 - ▶ Garantia de entrega de mensagens e de ordem (depende aplicação)
 - ▶ Rede física oferece suporte nativo a *multicast* (*IP multicast*, ethernet)
 - ▶ Alternativas: envio de n mensagens ponto a ponto; distribuição em árvore

Porque comunicação em grupo? (alguns exemplos)

- ❑ Tolerância a falhas baseada em replicação de serviços
 - ▶ Serviço replicado é feito por um grupo de servidores
 - ▶ Requisição do cliente é enviado ao grupo (todos executam mesma operação)
- ❑ Serviços de descoberta
 - ▶ Publicar interfaces
 - ▶ Localização de serviços providos
- ❑ Desempenho via replicação de dados
 - ▶ Multicast é usado para atualização de valores nas réplicas
- ❑ Notificação de eventos

Modelo de sistema

- ❑ Sistema é uma coleção de processos
 - ▶ Processos podem pertencer a grupos
- ❑ Primitivas
 - ▶ $multicast(g, m)$: envia mensagem m para processos do grupo g
 - ▶ Processo de envio depende do suporte sistema (rede física + sisop)
 - ▶ $deliver(m)$: entrega mensagem m para processo destino
 - ▶ Processo de entrega depende da semântica (confiabilidade x ordenação)



Gerenciamento de grupo

- ❑ Primitivas necessárias:
 - ▶ Criação e destruição de grupos
 - ▶ Inclusão e exclusão de membros de um grupo
- ❑ Soluções possíveis:
 - ▶ Centralizada: um processo é o gerente do grupo
 - ▶ Problemas: confiabilidade baixa, escalabilidade baixa
 - ▶ Distribuída
 - ▶ Problemas: consistência e desempenho (consenso)
- ❑ Grupo fechado:
 - ▶ Apenas membros do grupo podem enviar mensagens para o grupo
 - ▶ Possível um processo fora do grupo enviar para um membro individualmente
- ❑ Grupo aberto:
 - ▶ Processo fora do grupo pode enviar mensagens para o grupo

Multicast básico

- ❑ Primitiva $multicast(g,m)$ que garanta que a mensagem será entregue ao destino
- ❑ Primitivas:
 - ▶ $B-multicast(g,m)$: para cada processo $p \in g$, $send(p, m)$
 - ▶ $B-deliver(m)$: entrega m para $p \in g$

Multicast confiável

- ❑ Integridade
 - ▶ Qualquer mensagem é idêntica a que foi enviada e que nenhuma mensagem é entregue duas vezes
- ❑ Validade
 - ▶ Qualquer mensagem é aceita por ser entregue em seu destino, se estiver correta
- ❑ Atomicidade (acordo)
 - ▶ Se um processo pertencente ao grupo g executa com sucesso $deliver(m)$, todos os demais também devem fazê-lo
 - ▶ É a propriedade "todos ou nenhum"

Confiabilidade do multicast

- ❑ O-confiável:
 - ▶ Remetente da mensagem não espera por confirmação dos membros
- ❑ 1-confiável:
 - ▶ Remetente espera uma confirmação de qualquer um dos membros
- ❑ m -de- n confiável
 - ▶ Remetente espera confirmação de m membros de um grupo de n ($1 < m < n$)
- ❑ Totalmente confiável
 - ▶ Remetente espera confirmação de todos os membros do grupo

Multicast confiável

On initialization

$Received := \{\};$

For process p to R -multicast message m to group g

B -multicast(g, m); // $p \in g$ is included as a destination

On B -deliver(m) at process q with $g = group(m)$

if ($m \notin Received$)

then

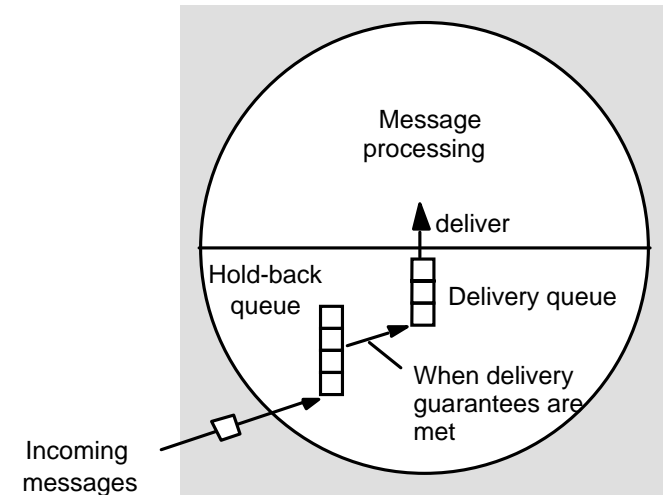
$Received := Received \cup \{m\};$

if ($q \neq p$) then B -multicast(g, m); end if

R -deliver m ;

end if

Fila de espera para mensagens *multicast*

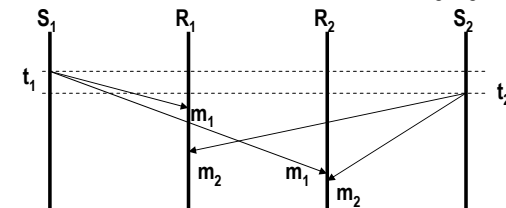


Problema de ordem na entrega das mensagens

- ❑ Garantir que todas as mensagens são entregues a todos os destinatários em uma ordem aceitável pela aplicação
- ❑ Semânticas possíveis:
 - ▶ Ordenação absoluta
 - ▶ Ordenação FIFO
 - ▶ Ordenação de causa
 - ▶ Ordenação consistente ou total
- ❑ Importante: garantia de ordem não implica em confiabilidade
 - ▶ É possível entregar duas mensagens em ordem, perdendo uma terceira intermediária a elas
 - ▶ Pode-se agregar a propriedade de atomicidade a essas semânticas

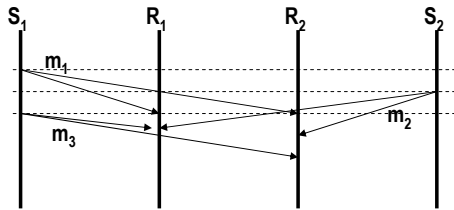
Ordenação absoluta

- ❑ As mensagens são entregues a todos os destinatários na mesma ordem em que foram emitidas (enviadas)
- ❑ Baseado em um *timestamp* da hora da criação (envio) da mensagem
 - ▶ Problema: dificuldade de manutenção de um relógio global



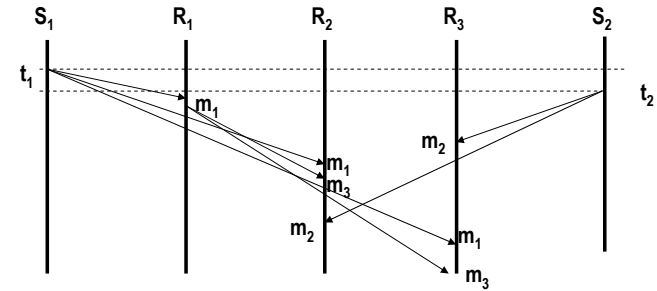
Ordenação FIFO

- Se um processo executa $multicast(g,m)$ e após $multicast(g,m')$, cada processo $q \in g$ realizará $deliver(m')$ após ter executado $deliver(m)$



Ordenação causal

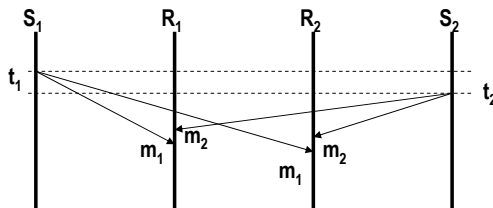
- Se $multicast(g,m) \rightarrow multicast(g,m')$ então $deliver(m')$ será executado após $deliver(m)$



Por causa da recepção de m_1 , R_1 envia a mensagem m_3 para R_2 e R_3 . A ordem m_1 e m_3 deve ser mantida nesses dois destinos.

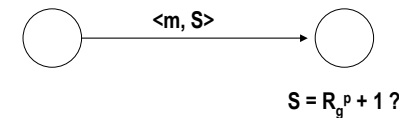
Ordenação consistente (total)

- Se um processo $p \in g$ executa $deliver(m)$ antes de $deliver(m')$, então qualquer outro processo $q \in g$ entregará m' após m
 - Garantia que todas as mensagens são entregues na mesma ordem em todos os processos mesmo que difira da ordem de emissão



Algoritmo para ordenação FIFO

- Baseado em números de seqüência
- Dois contadores em cada processo:
 - S_g^p : número de seqüência das mensagens que processo envia a g
 - R_g^q : número de seqüência da última mensagem recebida de um processo q

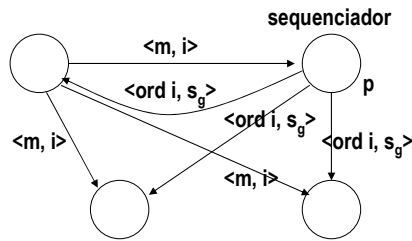


Sim:
- $R_g^p = S$
- Entrega a mensagem

Não:
- $S > R_g^p + 1$
- Enfileira a mensagem

Algoritmo ordenação total (centralizado)

- Definição de um processo $p \in g$ como sequenciador
 - ▶ Recebe mensagem de um processo q e a retransmite a todos os membros com um identificador único sequencial
 - ▶ Para cada mensagem m' enviada após m : $deliver(m')$ após ter sido executado $deliver(m)$



- Desvantagens:**
- ▶ Ponto único de falha
 - ▶ Escalabilidade

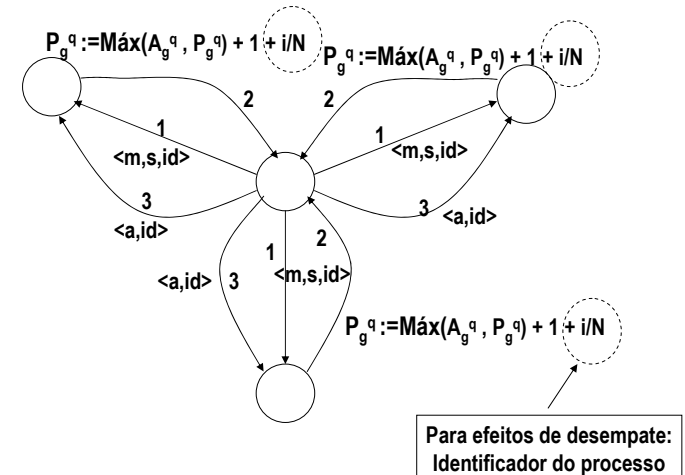
Algoritmo ordenação total (distribuído)

- Protocolo ABCAST (sistema ISIS)
- Passos:
 - ▶ Emissor atribui um número de seqüência S e envia a mensagem para todos os membros do grupo $\langle m, s, id \rangle$
 - ▶ Cada membro do grupo, responde com um número de seqüência proposto:
 - ▶ $Máx(A_{max}, P_{max}) + 1 + i/N$
 - ▶ Emissor seleciona o maior número de seqüência proposto pelos membros e envia uma nova mensagem com esse número de seqüência associando-o a mensagem previamente enviada $\langle a, id \rangle$
 - ▶ Membros associam número de seqüência a mensagem id
 - ▶ Membros realizam entrega ($deliver$) baseado no número de seqüência

Algoritmo ordenação total (distribuído)

- Protocolo ABCAST (sistema ISIS)
- ▶ Cada processo p armazena o maior número de seqüência acordado até o momento (A_p^q) e proposto p_p^q
- Passos:
 - ▶ Processo p envia $\langle m, id \rangle$ para g (id é um identificador de m)
 - ▶ Cada processo q responde a p enviando um número de seqüência $P_p^q := Máx(A_p^q, P_p^q) + 1$
 - ▶ Emissor seleciona o maior número de seqüência proposto pelos membros e envia uma nova mensagem com esse número de seqüência associando-o a mensagem previamente enviada $\langle a, id \rangle$
 - ▶ Membros associam número de seqüência a mensagem id
 - ▶ Membros realizam entrega ($deliver$) baseado no número de seqüência

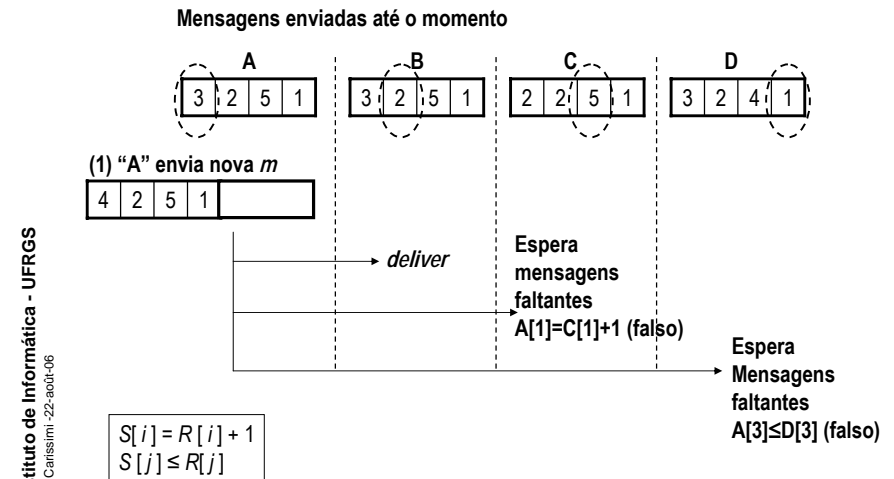
Algoritmo de ordenação total (cont.)



Algoritmo ordenação causal

- ❑ Protocolo CBCAST (Sistema ISIS)
- ❑ Passos:
 - ▶ Cada membro mantém um vetor de n elementos (membro i , i -ésima entrada)
 - ▶ Entrada é o número de seqüência da última mensagem recebida de i
 - ▶ Ao enviar uma mensagem, cada membro incrementa sua própria entrada no vetor e envia o vetor junto com a mensagem
 - ▶ *deliver* é executada no destino quando for satisfeito (simultaneamente):
 - ▶ $S[i] = R[i] + 1 \rightarrow$ destino não perdeu nenhuma mensagem do remetente
 - ▶ $S[j] \leq R[j]$ para todo $j \neq i \rightarrow$ remetente não recebeu nenhuma mensagem que ainda não tenha sido recebida pelo destino
 - ▶ onde S é vetor recebido junto a mensagem, R é o vetor local

Algoritmo ordenação causal (cont.)



Leituras adicionais

- ❑ Coulouris, G; Dollimore, J; Kindberg, T. *Distributed Systems: Concepts and Design* (3th edition), Addison Wesley, 2001
 - ▶ Capítulo 4 (seção 4.5)
 - ▶ Capítulo 11 (seção 11.4)