Comunicação de grupo

Enviar diversas mensagens únicas para um grupo de processos não é o melhor modelo de comunicação, por exemplo quando se deseja implementar um serviço para aumentar a disponibilidade ou confiabilidade de um sistema.
Multicast messages são mais apropriadas, i.e. mensagens enviadas por um processo para membros de um grupo. Outra forma seria a utilização de *broadcast messages*, onde uma mensagem é enviada para todos os processos, e aqueles envolvidos aceitam a mensagem e os demais descartam tais mensagens (isto é feito em redes de computadores). *Broadcast messages* tem a desvantagem de consumir tempo, pois uma camada deve analisar cada mensagem enviada.
Em geral *multicast messages* são muito úteis para construir sistemas distribuidos com as seguintes características:

1. Tolerância a falhas baseada em replicação de um serviço: grupo de servidores, mesmo que um servidor esteja fora do ar, o serviço será atendido.

2. Localizar objetos em serviços distribuídos: servidor de arquivos distribuídos, uma mensagem é enviada para todos os servidores e somente aquele que possui o arquivo responde.

3. Melhoria de performance através de dados replicados: uso de replicas, onde se mantem uma cópia na estação de trabalho do usuário.

4. atualizações múltiplas: quando alguma informação chega então uma mensagem é enviada para todos os interessados, e.g. *news service*.

Atomicidade

No caso 1 acima, requer-se que todos os servidores recebam todas as requisições de forma que todos servidores façam as mesmas atualizações. Isto requer *atomic multicast*:

Atomic Multicast: uma mensagem transmitida é recebida por todos os processos ou por nenhum processo.

Atomic multicast nem sempre é necessário. Exemplo, se estiver sendo feito uma consulta, muitas vezes a resposta de um só servidor já é o suficiente para que o cliente continue trabalhando. Para este tipo de situação *reliable multicast* já resolve o problema.

Reliable Multicast: é o processo onde é feito o possível para enviar a mensagem para todos os envolvidos, mas isto não é garantido. Em um *unreliable multicast* a mensagem só é transmitida uma vez.

Ordenação

Ambos *reliable* e *atomic multicast* usam uma fila para ordenar mensagens entre pares de processos. Em uma ordenação de fila, as mensagens de um cliente para um servidor são entregues na ordem que foram enviadas. Isto é resolvido através de um número (ou vetor) anexado a cada mensagem. No caso 1 acima, necessita-se que todos os servidores executem as operações recebidas na mesma ordem, considerando que as mensagens podem ser enviadas por clientes diferentes.
Sem um mecanismo que garanta a entrega ordenada de mensagens, quando dois originadores de uma mensagem para um grupo no mesmo tempo, suas mensagens podem não chegar ao mesmo tempo. Por exemplo, isto pode acontecer se uma mensagem é perdida e depois retransmitida.
A forma mais forte de ordenação é chamada *ordenação absoluta*:

Ordenação absoluta: quando diversas mensagens são transmitidas para um grupo, as mensagens chegam para todos os membros do grupo na mesma ordem que foram enviadas (isto pode ser atingido utilizando sincronização entre os diversos relógios do sistema).

Outra forma de ordenação é a *ordenação consistente*:

Ordenação consistente: quando diversas mensagens são transmitidas para um grupo as mensagens chegam para todos os membros do grupo na mesma ordem (que pode ser diferente da ordem que foram enviadas).

Para algumas aplicações as semânticas acima não são necessárias, e uma semântica mais fraca é aceitável. Uma forma de semântica mais fraca é a *ordenação de causa*.

Ordenação de causa: garante que se o evento de envio de uma mensagem causa o evento de envio de outra mensagem, então as mensagens são enviadas a todos receptores na mesma ordem. Entretanto se duas mensagens não possuem relação alguma de causalidade, então elas podem ser entregues em qualquer ordem.

CBCAST - Ordenação de causa

O protocolo de comunicação de grupo CBCAST funciona da seguinte forma:

- Se um grupo tem n membros, cada processo mantem um vetor com n elementos, um por membro do grupo.
- O *i-ésimo* elemento do vetor representa o número da última mensagem recebida do processo i.

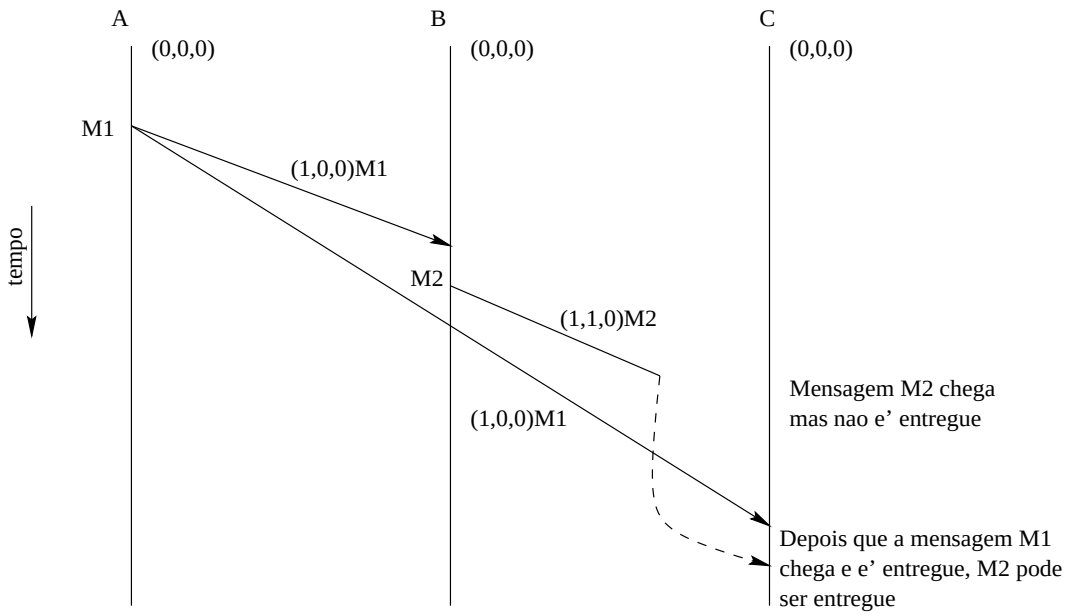

Figure 1: Entrega de mensagens usando CBCAST

- Os vetores são controlados pelo sistema de suporte, e não pelos processos, e são inicializados com zero. Veja Figura 1.
- Quando um processo tem uma mensagem para enviar, ele incrementa seu elemento no vetor, e envia o vetor como parte da mensagem.
- Quando M1 na Figura 1 chega ao processo B, verifica-se se ela depende de qualquer outra mensagem que B ainda não recebeu.
- . O primeiro elemento do vetor é maior do que o mesmo elemento no vetor armazenado em B, o que é esperado (e necessário) para uma mensagem que esta chegando de A, e os outros elementos são iguais, então a mensagem pode ser entregue para B.
- Se qualquer outro elemento do vetor que está vindo com a mensagem enviada por A fosse maior que o elemento correspondente no vetor em B, a mensagem não poderia ser entregue naquele momento.
- Na Figura 1, logo após ter recebido a mensagem M1, B envia uma mensagem para C, que chega em C antes de M1. A partir do vetor em C, C nota que B já recebeu uma outra mensagem de A antes que M2 foi enviada, e como C não recebeu coisa alguma de A, M2 é armazenada até que a mensagem M1 chegue.
- O algoritmo geral para decidir se uma mensagem que chega pode ou não ser entregue é o seguinte:

- Considere V_i como o *i-ésimo* elemento do vetor que está chegando com a mensage;
- Considere L_i como o *i-ésimo* elemento do vetor no processo que está recebendo a mensagem;
- Suponha que a mensagem foi enviada pelo processo j;
- A primeira condição é a seguinte: $V_j = L_j + 1$ (esta condição especifica que esta é a próxima mensagem vinda de j, ou seja nenhuma mensagem foi perdida - mansagens enviadas pelo mesmo processo são sempre ordenadas, ou seja uma é a causa da outra);
- A segunda condição para se aceitar a mensagem é: $v_i \leq L_i$ para todo $i \neq j$ (esta condição garante que quem enviou a mensagem recebeu todas as mensagens que o receptor já recebeu).

Exercício Imagine um grupo com 6 processo (A-F) com os vetores com os valores mostrados na Figura 2. A envia uma mensagem para todos os demais processos do grupo com o seu vetor já atualizado. Quais dos processos podem receber a mensagem e quais devem armazenar a mensagem até que outras cheguem ? Mostre por que.

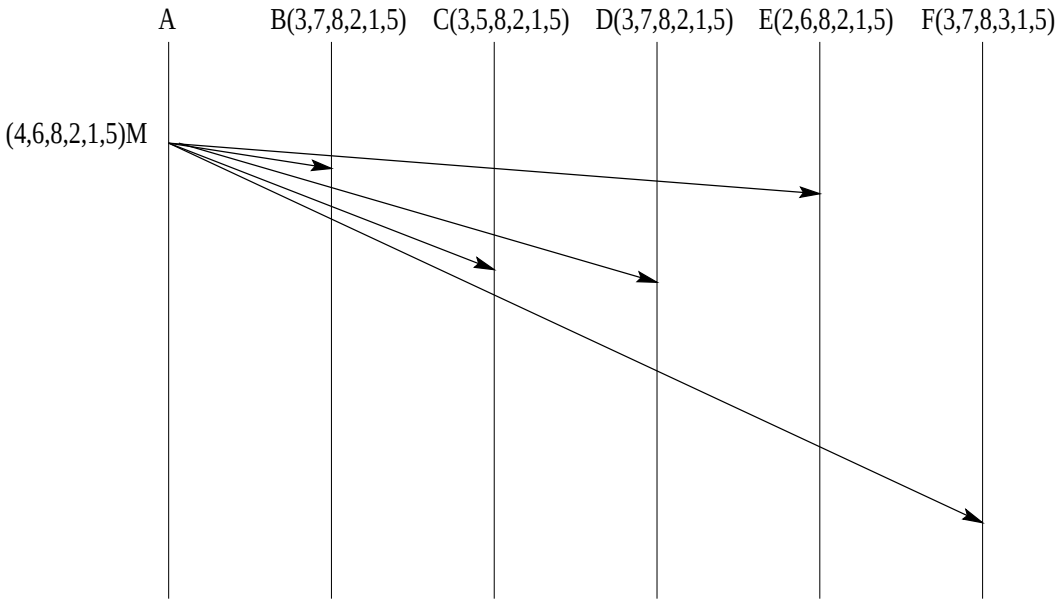

Figure 2: Exercício