Sistemas Distribuídos – COS470

2018/1

Trabalho Prático 3

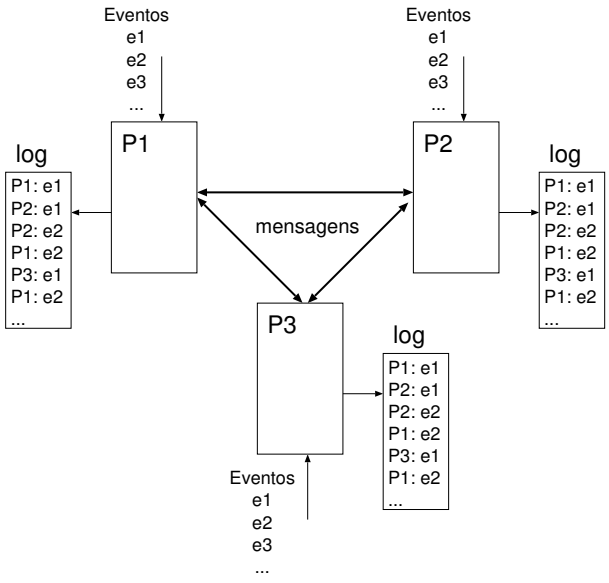
1 Objetivos

O objetivo deste trabalho é implementar um mecanismo distribuído de ordenação total de eventos. Além da implementação, você deve testar seu programa, rodando os estudos de casos. Você pode escolher qualquer linguagem de programação para implementar este trabalho, pois o foco está no mecanismo distribuído de ordenação total e no relógio lógico (ver abaixo).

Você deve preparar um relatório, com no máximo 6 páginas, com as decisões de projeto e implementação das funcionalidades especificadas, assim como a avaliação dos estudos de caso. Por fim, você deve preparar uma apresentação de no máximo 10 minutos sobre seu trabalho (como no relatório), a ser realizada no horário da aula. O trabalho pode ser individual ou em dupla.

2 Cenário

Considere um cenário onde um conjunto fixo de processos executando em diferentes máquinas recebem eventos gerados localmente e assincronamente. Por exemplo, os eventos podem ser citações (frases) que chegam aos processos por algum mecanismo externo. Um evento que ocorre em um processo deve ser enviado pelo processo a todos os outros processos do conjunto. Cada processo deve armazenar todos os eventos (os gerados localmente e os recebidos de outros processos) em um arquivo de log local. Entretanto, os arquivos de log devem ser todos idênticos, ou seja, os eventos devem ser registrados na mesma ordem em todos os logs. A figura abaixo ilustra o cenário com três processos.



3 Tarefa

Para que o sistema distribuído acima funcione conforme desejado, é necessário implementar um mecanismo de ordenação total de eventos. Em particular, iremos adotar o *Totally Ordered Multicast*, visto em aula, que não faz uso de relógios reais. A principal tarefa deste trabalho é implementar este mecanismo, levando em conta os seguintes aspectos:

- O número de processos que participam do sistema é fixo, dado por n, e cada processo tem um identificador único $P_i = i$, para $i = 1, \ldots, n$. Tanto o identificador quanto o número de processos são parâmetros de entrada do programa.

- Cada processo deve ser *multi-threaded*, sendo uma thread responsável por gerar os eventos locais (entre outras coisas); e uma outra thread para receber os eventos de outros processos (entre outras coisas). Você pode usar mais threads, se achar apropriado.

- Implementar o relógio lógico de Lamport para os eventos e mensagens do sistema. Para isto, cada processo deve manter um relógio lógico local que deve ser atualizado de acordo com as regras definidas pelo relógio lógico de Lamport.

- Implementar o mecanismo de troca de mensagens do *Totally Ordered Multicast*. Lembrando que o mecanismo tem dois tipos de mensagens: (1) notificação de evento, (2) confirmação de recebimento de evento. A mensagem de notificação de evento deve incluir o evento (no caso, frases), o identificador do processo onde o evento foi gerado, e o relógio lógico associado ao evento.

- Usar uma estrutura de dados com prioridades para incluir os eventos na fila e remover para gravar no log (cuidado com condição de corridas).

- Utilizar uma biblioteca de *sockets* para estabelecer a comunicação entre os processos. Para facilitar, o processo com menor identificador deve inicializar a conexão para um processo com maior identificador. Além disso, crie um *socket* do tipo TCP para cada par de processos, que deve ser usado para a comunicação nos dois sentidos. Você pode usar outra ideia, se achar apropriado.

- Gerar um arquivo em disco com os eventos seguindo a ordenação global. Cada linha do arquivo de log deve ter o identificador do processo que gerou o evento, o relógio lógico associado ao evento, e o evento em si (ex. uma frase).

4 Estudo de caso

Para testar o sistema acima, desenvolva um gerador de frases (eventos). As frases podem ser aleatórias, tanto em seu comprimento quanto em seus caracteres. Por exemplo, frases de tamanho entre 50 e 100 apenas com caracteres minúsculos e espaço (tudo escolhido aleatoriamente). Outra opção é usar um conjunto de frases que façam sentido (ex. 100 frases de renomado pensadores), carregar na memória uma tabela com as frases (ex. ler as frases de um arquivo), e escolher aleatoriamente uma frase desta tabela.

A taxa de geração de frases deve ser controlada por um parâmetro de entrada λ, que determina o número de frases a ser gerada por segundo. Implemente um gerador que utilize sinais (*alarm*) para gerar frases a uma taxa λ. O processo de geração termina quando k frases forem geradas, outro parâmetro de entrada.

Para realizar os testes, incie todos os processos (de preferência em mais de uma máquina), aguardando um tempo para iniciar o processo de geração de frases. Cada processo deve então gerar k frases, e aguardar (não é necessário implementar um mecanismo de término de processo). Compare os arquivos logs gerados para ver se todos são idênticos.

Teste o sistema com $n = 2, 4, 8$ e $k = 100, 1000, 10000$ e $lambda = 1, 2, 10$. Quais foram os resultados obtidos?