Especificação do Trabalho II – ELC1018 Sistemas Distribuídos
Programação de um middleware para
Comunicação *multicast* com Ordenamento Causal de mensagens e Mecanismo de Estabilização para descarte de mensagens do buffer ambos baseados em vetor de relógios lógicos
Data de entrega (código e documentação) e apresentação: 24/09/2020

O trabalho consiste em **implementar um *middleware* para envio de mensagens *multicast* obedecendo à ordem causal**, bem como implementar no middleware um controle de estabilização para descarte de mensagens. Consulte o material teórico e os slides para compreender o **algoritmo para tratar o ordenamento de mensagens** e o **algoritmo para tratar estabilização de mensagens**.

O *middleware* deve ser um único *packet* Java nomeado `CausalMulticast` que deve ser importado por algum programa Java (usuário) e que oferecerá através de sua API a facilidade de enviar uma mensagem *multicast* para um conjunto de destinatários (grupo) com ordenamento causal. Vide detalhes da API abaixo. Funcionalmente, o *middleware* deve implementar a comunicação *multicast* entre suas instâncias (programas que usam o *packet* e que executam em diferentes nodos/computadores) através do envio de mensagens *unicast* não confiável (sockets UDP) a todos os membros do grupo destinatário. O uso de sockets será necessário em dois momentos: 1) para realizar a descoberta de qual computador tem um middleware executando; e 2) para trocar mensagens do usuário. No caso 1) use IP *multicast* para realizar a descoberta dos computadores/middlewares e manter a lista de computadores que executam o middleware. No caso 2) use sockets UDP para uma instância do middleware enviar as mensagens da aplicação do usuário a outras instâncias do middleware (em outras máquinas).

A interface do *middleware* (API do packet `CausalMulticast`) deve oferecer apenas um método `mcsend(msg, this)`, para envio de uma mensagem `msg` às aplicações vinculadas a outras instâncias do middleware. Para receber mensagens, os usuários do middleware (aplicações do usuário que usam o middleware) devem implementar um método `deliver(msg)`, que será invocado pelo middleware para entregar por *callback* a mensagem `msg`. Note que o parâmetro `this` do método `mcsend` é quem viabilizará a *callback*. O serviço de descoberta deve permanecer sempre ativo, a fim de permitir atualização dinâmica dos membros do grupo. Como cada instância do *middleware* deve conter as informações sobre quem participa do grupo (resultado da etapa de descoberta), e a atualização é realizada dinamicamente, o encaminhamento das mensagens multicast será realizado ao grupo corrente.

Importação do pacote `CausalMulticast`:
```
import CausalMulticast;
```

API oferecida pelo pacote `CausalMulticast`:
```
public void mcsend(String msg, Object cliente);
```

Interface que deve ser implementada por todo usuário do pacote `CausalMulticast`:
```
Public interface ICausalMulticast {
    public void deliver(String msg);
}
```

Na aplicação do usuário, o pacote `CausalMulticast` (middleware) permite a instanciação de um canal *multicast*:
```
CMChannel canal = new CMChannel(this);
```
OBS: a instanciação não deve ser realizada no método `main`!

Para possibilitar a correção do trabalho, faça o envio de cada mensagem *unicast* ser controlado via teclado, ou seja, deve haver uma pergunta antes de cada envio *unicast* (controle) questionando se é para enviar a todos ou não. O caso "não" deve permitir o envio um a um no *unicast*, aguardando posteriormente para que o professor decida sobre o envio da mensagem. Por exemplo, com três processos, deve-se poder escolher pelo menos um para envio posterior (atrasado). O conteúdo do buffer e dos relógios lógicos também precisam ser permanentemente demonstrados na tela. Não é necessário implementar uma GUI na aplicação do usuário.

O algoritmo de ordenação causal a ser implementado pode ser o proposto no artigo *Fundamentals of Distributed Computing: A Practical Tour of Vector Clock Systems*, disponível em https://www.computer.org/csdl/mags/ds/2002/02/o2001.pdf. Porém outros algoritmos com vetores de relógios lógicos também podem ser implementados (indique isto na documentação, se ocorrer).

O algoritmo do artigo para ordenação causal está descrito a seguir e o seu comportamento resumido está ilustrado na figura 1.

```
% realizado por cada Pi
procedure mcsend(msg)
      msg.VC ← VCi                                  % constrói o timestamp da msg
      ∀x ∈ {1,...,n} do send(msg) to Px enddo       % multicast msg
      VCi[i] ← VCi[i]+1                             % Pi fez mais um broadcast

% realizado por cada Pi
when Pi receives msg from Pj                        % msg traz msg.VC por piggyback
      % verifica e realiza o ordenamento causal
      atrasa entrega até (x ∈ {1,...,n} : msg.VC[x] ≤ VCi[x])

      % atualiza variável de controle e entrega msg
      if i ≠ j then VCi[j] ← VCi[j]+1
      entrega msg para a aplicação
```

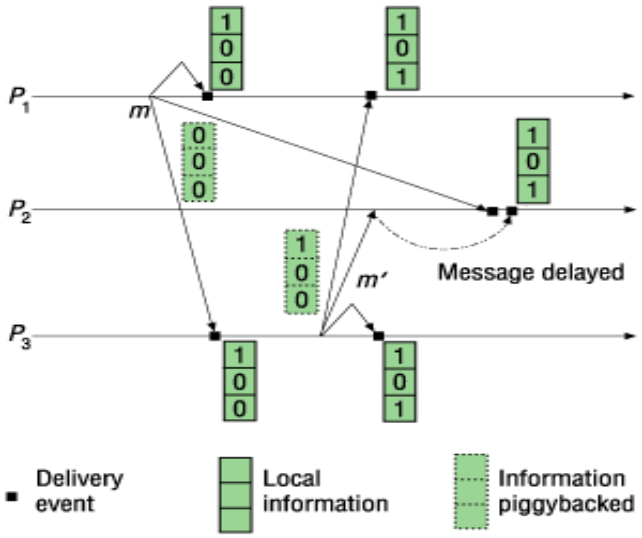

Figura 1 – Funcionamento do algoritmo para ordenamento causal.

O algoritmo do artigo para estabilização das mensagens está descrito a seguir e o seu comportamento resumido está ilustrado na figura 2.

```
procedure mcsend(msg)                               % realizado por Pi
    msg.VC ← MCi[i][*]                              % constrói o timestamp de msg
    msg.sender ← i
    for all P do send(msg) to Pj enddo              % multicast
    MCi[i][i] ← MCi[i][i]+1                         % Pi fez mais um multicast

when Pi receives msg from Pj                        % msg traz msg.VC por piggyback
    deposit(msg)                                    % coloca msg no buffer
    MCi[j][*] ← msg.VC                    % atualiza visão do Pi com visão de Pj
    if i ≠ j then MCi[i][j] ← MCi[i][j]+1           % mais 1 msg de Pj entregue
    deliver msg to the upper layer                  % evento de entrega

when (existe msg no buffer_i AND msg.VC[msg.sender] ≤ min_{1≤x≤n}(MCi[x][msg.sender])
    discart(msg)                                    % elimina msg do buffer local
```

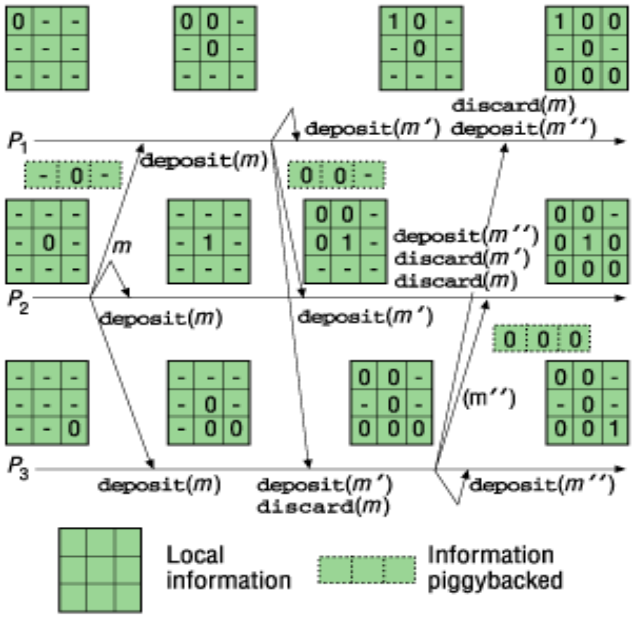

Figura 2 – Funcionamento do algoritmo para estabilização de mensagens.

A entrega do trabalho deve ser no Moodle e deve inclui código e documentação no formato Oxigen ou JavaDoc.