Pontifícia Universidade Católica do Rio de Janeiro  
INF 2545 – Sistemas Distribuídos Professora: Noemi 2010.1

Trabalho 3  
Sincronização

Alunos:  
Danilo Moret  
Thiago Manhente de C. Marques

# Arquitetura

A rede montada para desenvolver e testar a biblioteca t3lua consiste de uma rede alua com daemons e processos conforme descritos na documentação da biblioteca alua. Na rede t3lua, no entanto, cada daemon da rede recebe funções para gerência de grupos e repasse das mensagens. Sendo assim todos os daemons linkados devem incluir o código existente em t3daemonscode.lua . Isto é realizado no script que monta uma rede inicial em t3daemons.lua , que abre cinco processos lua e inicia um daemon em cada um.

Na arquitetura escolhida são os daemons quem mantém o mapeamento dos processos e grupos existentes, replicando a informação por todos eles. Desta forma sempre que um novo processo se comunica através do t3lua, a biblioteca repassa a comunicação ao daemon do processo e este é responsável por repetir para todos os processos do grupo, ou informar os outros daemons da entrada ou saída do processo em um grupo.

Esta decisão foi tomada para simplificar o trabalho de manutenção das tabelas de grupos. Apesar de existir o problema de replicação da informação por todos os daemons (por exemplo, em uma rede onde a proporção entre daemons e processos seja alta esta estratégia poderia ser repensada), isto evita a centralização desta informação em um único processo ou daemon e a complexidade de localizar onde a tabela de grupos está. Além disso a replicação entre daemons é mais simples, uma vez que a biblioteca alua permite a listagem de daemons da rede, mas não dos processos.

# Desenho da API

A construção da API foi feita concentrando nos métodos para gerência de grupos. O desenho levou então para a construção dos seguintes métodos:

***t3lua.init(listenFunction[, cbf, debugMode, bogusMode])***

Este método deve ser chamado antes de usar a API. Ele inicia a conexão do processo com um dos daemons conhecidos listados em t3hosts.lua, escolhido aleatoriamente. O método inicia o loop de eventos ao final da conexão, bloqueando o fluxo de execução. Portanto as funcionalidades do programa devem ser desenvolvidas usando o método de escuta passado ou a função de callback opcional. Após ser iniciado o parâmetro t3lua.id fica disponível, sendo uma cópia do valor do identificador único do processo alua correspondente. Os outros métodos irão sair com erro caso esta função não tenha sido chamada antes, ou se a conexão tiver falhado.

Parâmetros:

listenFunction, tipo function, obrigatório - função que recebe as comunicações enviadas para este processo. A mensagem é recebida como um único parâmetro no formato de uma tabela com os seguintes campos:

data - tipo string - a mensagem transmitida.

group - tipo string - nome do grupo onde a mensagem foi transmitida.

src - tipo string - id do processo que enviou a mensagem.

cbf, tipo function - função de callback executada ao final da conexão e configuração. Não recebe nenhum parâmetro.

debugMode, tipo boolean - indica se a biblioteca deve entrar no modo debug, imprimindo mais mensagens que o normal.

bogusMode, tipo boolean - indica se a biblioteca deve entrar no modo comparativo, desviando as chamadas “reais” das funções de sincronização por chamadas fictícias, permitindo notar que o comportamento esperado está realmente sendo observado.

***t3lua.join(group[, cbf])***

Adiciona o processo atual ao grupo indicado. Caso o grupo não exista ele será criado.

Parâmetros:

group, tipo string, obrigatório - o nome do grupo, que será criado caso não exista na rede.

cbf, tipo function - função de callback executada ao final de se unir ao grupo. Não recebe nenhum parâmetro.

***t3lua.initAndJoin(group, listenFunction[, cbf, debugMode, bogusMode])***

Executa em sequencia os métodos de início e adicionar ao grupo atual. Veja os parâmetros nas funções init e join.

***t3lua.leave(group[, cbf])***

Retira o processo atual do grupo indicado.

Parâmetros:

group, tipo string, obrigatório - o nome do grupo do qual o processo atual que sair.

cbf, tipo function - função de callback executada ao final de se unir ao grupo. Não recebe nenhum parâmetro.

***t3lua.send(group, data[, cbf])***

Método de broadcast de mensagem dentro do grupo. Envia a mensagem sem nenhum controle de sincronização para todos os processos membros do grupo.

Parâmetros:

group, tipo string, obrigatório - o nome do grupo onde a mensagem será transmitida.

data, tipo string, obrigatório - a mensagem a ser transmitida, em texto puro. *Atenção, a mensagem será concatenada ao método send do alua, ela não está protegida, portanto deve se tomar cuidado para ela não fechar a função.*

*cbf, tipo function - função de callback executada ao final de se unir ao grupo. Não recebe nenhum parâmetro.*

***t3lua.sendTotal(group, data[, cbf])***

Método de broadcast de mensagem com ordenação total dentro do grupo. Envia a mensagem com controle de sincronização do tipo total, ou seja, garantindo que todos os membros do grupo receberão as mensagens na mesma ordem.

Parâmetros:

group, tipo string, obrigatório - o nome do grupo onde a mensagem será transmitida.

data, tipo string, obrigatório - a mensagem a ser transmitida, em texto puro. *Atenção, a mensagem será concatenada ao método send do alua, ela não está protegida, portanto deve se tomar cuidado para ela não fechar a função.*

*cbf, tipo function - função de callback executada ao final de se unir ao grupo. Não recebe nenhum parâmetro.*

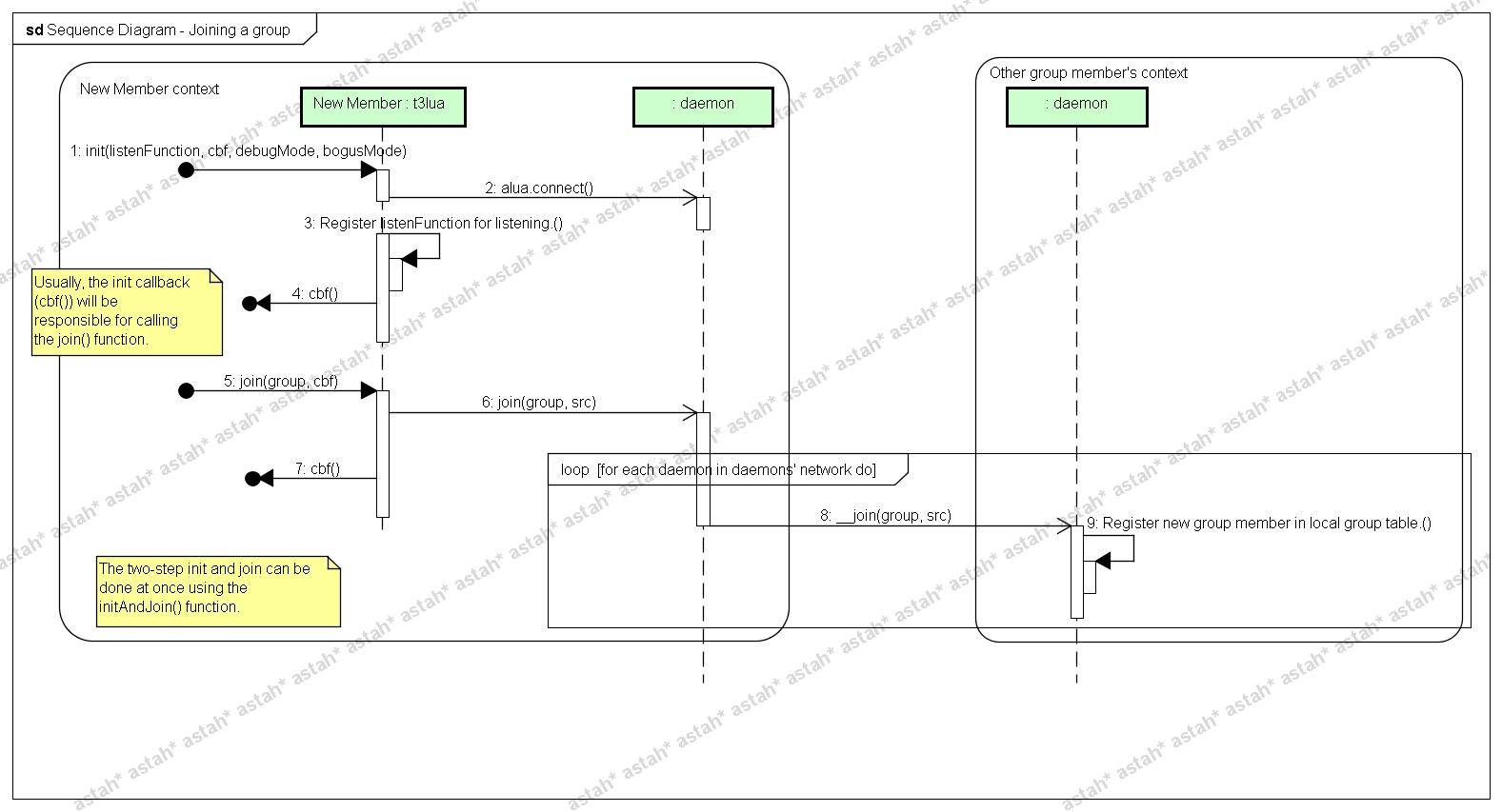
***t3lua.id***

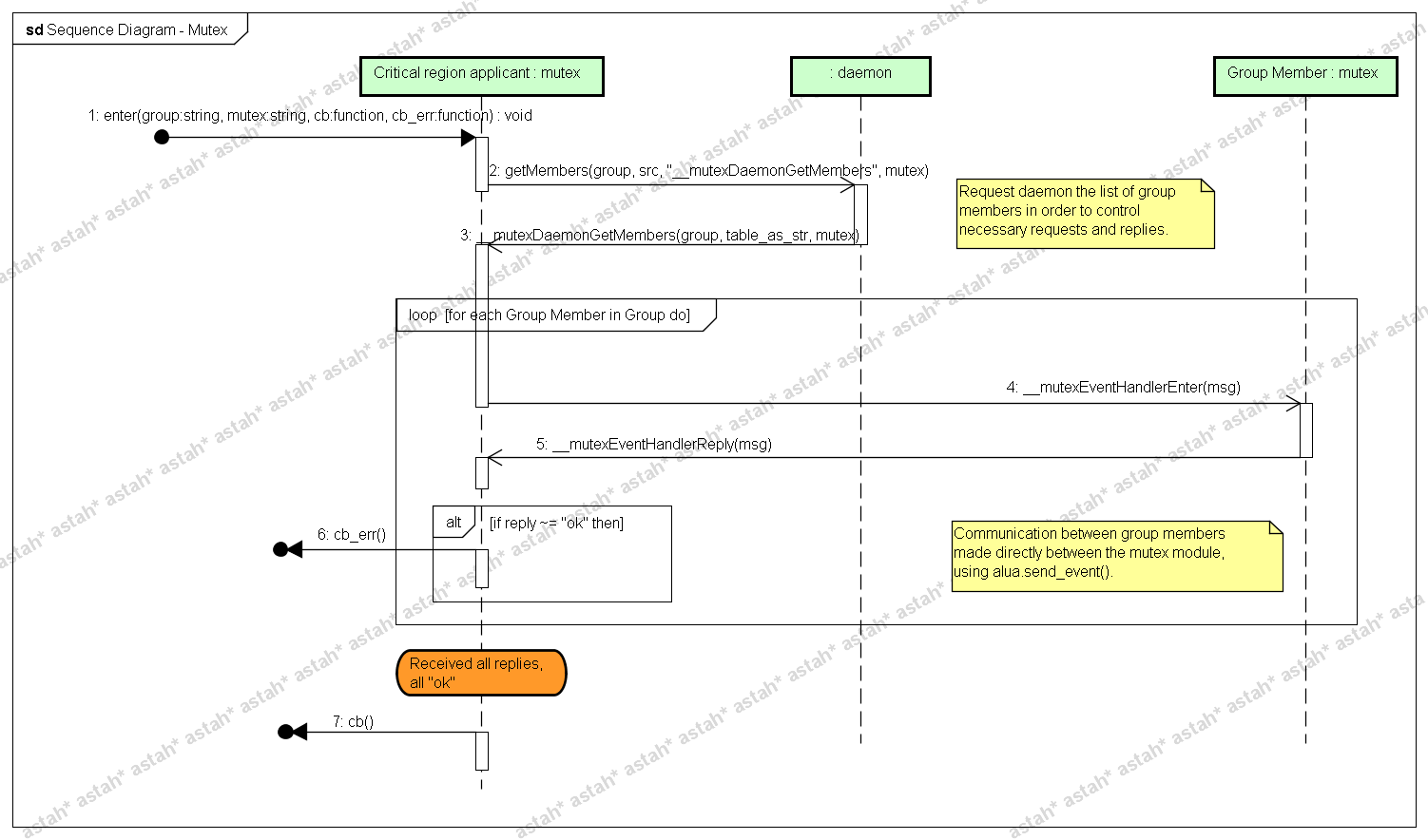
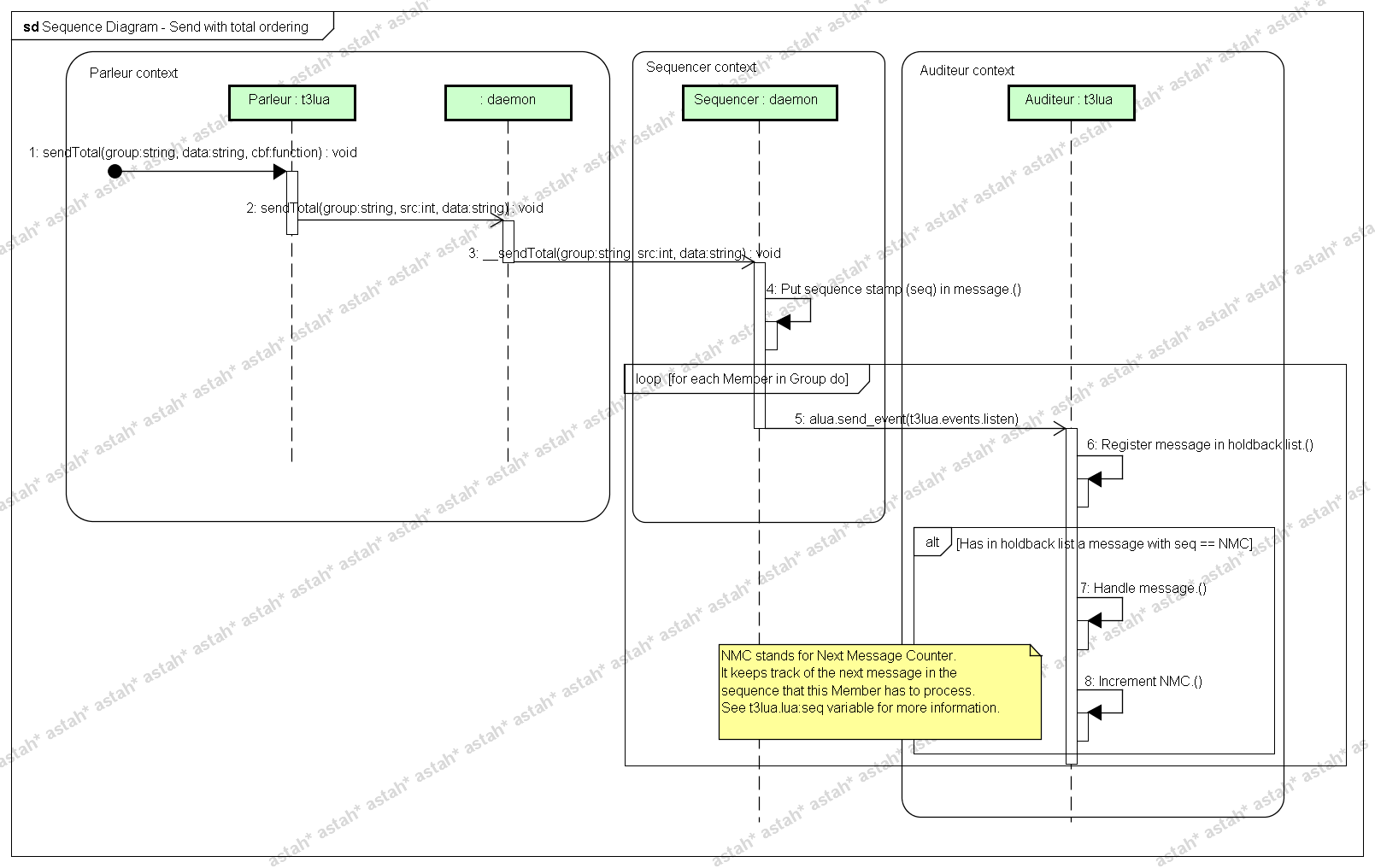
Parâmetro preenchido após o processo ter sido conectado ao daemon. Ele recebe o identificador único do processo, sendo uma cópia do id recebido na biblioteca alua.

# Uso da API

A seguir temos a descrição de três usos da API, acompanhados de diagramas.

O primeiro cenário é o de iniciar a conexão e entrar em grupo. Ao realizar esta operação o processo cliente está acionando a biblioteca t3lua, que por usa vez irá conectar o processo a um daemon escolhido aleatóriamente na lista de daemons conhecidos. Tanto o processo t3lua quanto o daemons t3lua são por sua vez construídos diretamente sobre um processo e um daemon alua. Pelas características da própria biblioteca alua, isso quer dizer que ao final da conexão para começar a receber as mensagens e eventos de outros processos é necessário entrar no loop de eventos. O loop de eventos do alua não retorna quando chamado, e as funcionalidades do programa devem ser implementadas no método de callback da conexão ou nos eventos cadastrados. Como o foco é o envio de mensagens, a biblioteca t3lua manteve o callback e usou a API de eventos para cadastrar um evento de transmissão. Sendo assim quando o processo t3lua conecta na rede ele indica um método de escuta que receberá as mensagens quando o evento de recebimento for disparado e as mensagens puderem sair da fila de holdback, abstraindo o conceito de eventos para a camada do processo.

O segundo cenário de exemplo é o do broadcast total. Assumindo que o processo já se uniu ao grupo, para fazer a transmissão garantindo entrega com ordenação total ele chama o método sendTotal da API. Ele indica para este método o grupo e passa a mensagem como string. A biblioteca t3lua então irá repassar a mensagem para o daemon onde o processo está conectado, que por sua vez irá repassa-la para o daemon da rede que foi eleito o sequenciador da rede. O daemon irá numerar a mensagem na ordem que foi recebida e então retransmiti-la para os membros do grupo. Note que o método sendTotal pode receber uma função de callback opcionalmente, mas ela não será chamada somente na recepção da mensagem, e sim logo após que a mensagem for retransmitida para o daemon, permitindo apenas saber na implementação atual se a mensagem foi encaminhada e prosseguir o processamento. Ao invés disso o processo irá ouvir o eco de sua própria mensagem, entregue ordenadamente, através da função de escuta que ele passou como referência ao conectar na rede.

Finalmente o mutex ...

# Experimentos e demonstração

Como não seria necessário mais simular a performance de máquinas isoladas, apenas confirmar a sequencia de mensagens enviadas ou a exclusão mútua de execução, os experimentos desta vez foram todos realizados na mesma máquina, executando os processos manualmente e em seguida comparando a saída tabulada de cada um. Pela arquitetura escolhida os testes envolvem iniciar um conjunto de daemons usando o script t3daemons.lua e em seguida chamar os scripts que iniciam os processos, entram nos grupos e seguem o roteiro de comunicação para testar cada tipo de comunicação, conforme indicando a seguir.

A principal estratégia de teste foi a sugerida na definição do trabalho. A biblioteca recebe em seu método de início um parâmetro, *bogusMode, indicando se a biblioteca deve usar os métodos de efetivos (bogusMode = false, valor padrão) ou desviar para métodos que não fazem os controles (bogusMode = true). O roteiro de transmissão então irá fazer várias comunicações entre os processos, e o resultado quando a biblioteca estiver no modo efetivo deve ser igual ao esperado na definição de cada tipo de sincronização. Já no modo não efetivo deve ser possível encontrar situações onde as condições garantidas pelas sincronizações são violadas.*

*Para testar o multicast total o script dos processos transmite uma mensagem contendo um nome designando cada processo (alice, bob, carl) e um número serial das mensagens. Ao receber uma mensagem da biblioteca o processo imprime a mensagem para um arquivo próprio (alice.log, bob.log, carl.log) e se a mensagem veio de outro processo o script aguarda entre 1 e 3 segundos (usando uma chamada não bloqueante aproveitando a função socket.select) e então envia outra mensagem para o grupo. A saída dos processos deve ter todas as mensagens recebidas por todos os processos na mesma ordem, mas quando a biblioteca for colocada no modo não efetivo devem existir mensagens recebidas em ordens diferentes.*

xx