Sistemas Distribuídos

Trabalho Prático: iBei

Meta I

João Feliciano - jamaia@student.dei.uc.pt - Nº 2009112631

Luís Coimbra - lcoimbra@student.dei.uc.pt - Nº 2014232313

Luís Silva - lfgsilva@student.dei.uc.pt - Nº 2008103138

---

29 de Outubro de 2016

1 Arquitetura de software

A arquitectura interna da aplicação está dividida em 4 partes, servidor TCP, servidor RMI, consola de administrador e base de dados.

1.1 Servidor RMI

O servidor RMI cria as suas próprias Threads quando os métodos são invocados pelos servidores TCP, contudo foi necessário criarmos outras Threads de modo a termos funcionalidades adicionais em tempo real:

- Thread UdpSlaveRMIServer - corre no servidor RMI secundário, permite comunicar com o servidor principal e detecta a falha desse servidor;
- Thread UdpMasterRMIServer - está à escuta de *pings* originados no servidor secundário, respondendo aos mesmos.

1.2 Servidor TCP

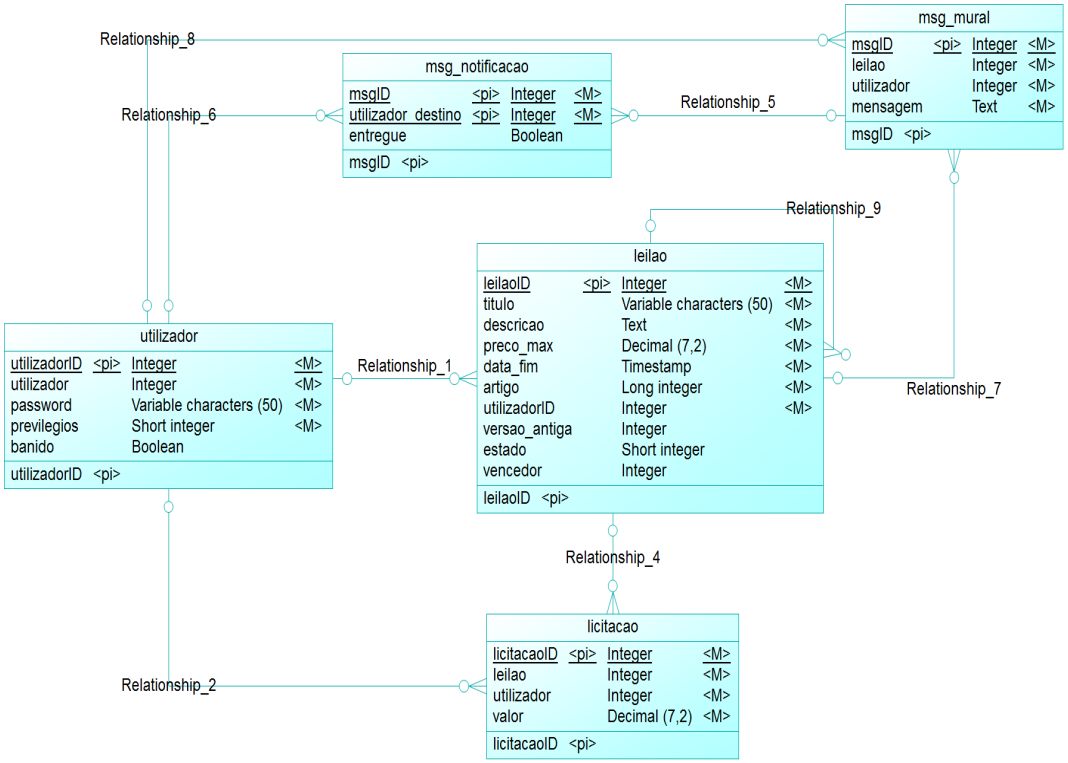
O servidor TCP é constituído por 4 Threads principais:

- Thread para aceitar clientes que vai criar uma nova *thread* por cada cliente que vai tratar os comandos que o respectivo cliente envia, e que terminam quando a ligação termina.
- Thread que através de *Multicast* envia a carga do servidor aos restantes servidores TCP.
- Thread que através de *Multicast* recebe as cargas enviadas pelos restantes servidores TCP.
- Thread que envia a cada utilizador a carga de todos os servidores TCP online, de minuto a minuto.

1.3 Consola de Administrador (Extra para grupos de 3 elementos)

A consola de administrador é constituida por uma Thread que vai executar os comandos introduzidos pelo administrador comunicando tanto com o Servidor TCP, como com o Servidor RMI.

1.4 Base de Dados



Para armazenar os dados da aplicação implentámos uma Base de Dados com 5 tabelas, que através de relações permite a vizualição e manipulação da informação guardada.

2 Detalhes do funcionamento do servidor TCP

Cada Servidor TCP começa por se ligar ao Servidor RMI. Em seguida junta-se a um grupo multicast por onde irá enviar a sua carga e irá receber a carga de todos os servidores TCP ligados à rede. É iniciada uma thread que de minuto a minuto envia uma mensagem para todos os clientes a informar da carga dos utilizadores, aqui optámos por contar o número de sockets de clientes abertas. Por fim fica à espera que os clientes se liguem, e quando isto acontece é criada uma thread para cada cliente onde os pedidos serão tratados.

3 Detalhes do funcionamento do servidor RMI

Quando um servidor RMI executa este verifica se já existe algum servidor RMI ligado.

Caso não exista, este torna-se o servidor primário iniciando uma thread que irá servir de sinalizador por udp para o RMI secundário poder verificar que este está activo. Por fim está preparado para poderem ser chamados métodos.

Caso já exista um servidor RMI primário este torna-se backup. Chama uma thread que irá verificar periodicamente se o RMI server primário está activo e após verificar cinco vezes que ele não está activo, irá tentar se tornar primário. Em caso de sucesso este torna-se primario e se o outro servidor conseguir recuperar irá se tornar num de backup.

4 Detalhes do funcionamento da Consola

Para a construção da Consola criámos um programa que se liga ao RMI server, recebe os pedidos do administrador e os executa no servidor RMI. Quando o administrador pede para testar o servidor TCP (ponto 7.4 do enunciado), este irá estabelecer uma ligação com o servidor TCP e executar uma série de comandos automaticamente.

Para o Administrador realizar pedidos, optámos por implemetar um protocolo parecido ao implementado pela comunicação entre o cliente e o TCP server, sendo os comandos utilizados os seguintes:

- Cancelar um leilão
  Exemplo - type: cancel_auction, id: 101
- Banir um utilizador
  Exemplo - type: ban_user, username: pierre
- Consultar estatísticas
  Exemplo - type: stats

Para testar o TCP server usamos.

- Exemplo - type: test_server, ip: <endereço ip>, port: <porto>

5 Como Executar

1º Criar a base de dados com o código SQL.

2º Configurar o ficheiro iBei.properties para a rede que se vai utilizar.

3º Correr o Servidor RMI que irá ficar como primário. 4º Correr o Servidor RMI que irá ficar como *Backup*. (Nota: Consideramos que os dois servidores RMI correm na mesma máquina). 5º Correr os Servidores TCP

6º Ligar os clientes(java, telnet, netcat, etc.)

6 Distribuição de tarefas

A distribuição de tarefas foi efectiada da seguinte forma:

- João Feliciano
  - Desenvolvimento do TCP server
  - Desenvolvimento da consola de Administrador
  - Implementação do Multicast
  - Implementação do Callback para notificações
  - Teste do programa
  - Criação do relatório
- Luís Coimbra
  - Idealização da base de dados
  - Criação de métodos para comunicação com a Base de Dados
  - Criação de métodos remotos do RMI
  - Teste do programa
  - Criação do relatório
- Luís Silva
  - Processo Fail-Over
  - Desenvolvimento do RMI
  - Desenvolvimento parcial da duplicação de dados
  - Teste do programa
  - Criação do relatório

7 Script de Testes

Executámos o ficheiro de testes fornecido e verificámos alguns casos em que o nosso código não passa por motivos de implementação:

- search_auction - Dependendo do valor do código de artigo, como usamos um Long de até 13 digitos para o guardar, quando este apresenta zeros à esquerda ao imprimir resposta este não imprimirá os zeros sendo as comparações de dados inválidas.
- edit_auction - Pelos mesmos motivos do ponto anterior não pode ser editado um código de artigo que não seja numérico. Strings não serão aceites.
- bid - não é premitido licitar no próprio leilão.

- immediate_message_notification - nos nossos testes conseguimos receber notificações em tempo real, mas no script não.
- offline_bid_notification - Ao emplementar as bids segundo o ponto 6.7 do enunciado, apenas estamos a notificar os utilizadores que efectuaram licitações e estão online no momento, não guardando referencias para os que estão offline

8 Testes de software

Requisitos Funcionais	**Resultado**
Registar novo utilizador	OK
Login protegido com password	OK
Criar um leilão	OK
Pesquisar leilões por código EAN/ISBN	OK
Ver todos os detalhes de um leilão (incl. histórico de licitações)	OK
Listar todos os leilões em que o utilizador está/esteve ativo	OK
Licitar um valor num leilão	OK
Editar propriedades de um leilão, guardando versões anteriores	OK
Escrever mensagem no mural de um leilão	OK
Utilizadores online leem novas mensagens imediatamente	OK
Utilizadores offline leem novas mensagens assim que se ligam	OK
Listar utilizadores online	OK
Notificação imediata de licitação melhor de outro utilizador	OK
Leilão termina corretamente na data, hora e minuto marcados	OK
Grupos de 3: cancelar um leilão (-3)	OK
Grupos de 3: banir utilizador, cancelando leilões e licitações (-3)	OK
Grupos de 3: mostrar estatísticas de atividade (-3)	OK
Grupos de 3: realizar teste a um servidor TCP (-3)	OK

Tratamento de Exceções e Balanceamento de Carga	**Resultado**
Avaria de um servidor RMI não tem qualquer efeito nos clientes	OK
Não se perde/duplica licitações se os servidores RMI falharem	passou no caso de perda
Avarias temporárias (¡30s) no RMI são invisíveis para clientes	OK
Servidores TCP trocam via UDP o número de clientes ligados	OK
Clientes são informados a cada 60s da carga dos servidores TCP	OK

Failover	**Resultado**
O servidor RMI secundário testa periodicamente o primário	OK
Em caso de avaria longa os servidores TCP ligam ao secundário	OK
Servidor RMI secundário substitui o primário em caso de avaria	OK
Os dados são os mesmos em ambos os servidores RMI	OK
O failover é invisível para utilizadores (não perdem o login)	OK
O servidor original, quando recupera, torna-se secundário	OK