EP 1 - Redes Chat Socket-P2P-Server

Índice

Grupo

8623910 - Adriano dos Santos Rodrigues da Silva
8598631 - Guilherme Hernandes do Nascimento
8623865 - João Pedro Nardari dos Santos

1 - Classes

1.1 - entity.User

Nesta classe estão agrupadas todos os atributos de cada usuário, como seu nome, senha, IP, porta em que espera conexões e sua lista de amigos.

1.2 - gui.WindowLogin

Esta classe representa a tela de login, contendo os campos Usuário (nome), senha e ip do servidor. Todos os dados são informados pelo usuário e só realiza o logon quando o usuário dgitar um usuário e senha válido e um ip de servidor que esteja rodando e esperando conexão na porta 10000.

1.3 - gui.WindowListFriends

Aqui temos a lista dos usuários que são amigos do usuário que está logado no cliente. A lista exibe todos os contatos e informa quais estão online e quais estão offline. Só é possível conversar com os que estão online, clicando na lista. Algumas vezes há uma certa demora em atulizar o status, porém isto é devido ao método de atualização do componente que usamos (JList) da classe Swing não ser thread-safe, e portanto há necessidade da thread que atualiza esperar em uma fila. Contudo, sempre em menos de 30 segundos.

1.4 - gui.WindowTalk

Esta tela implementa a conversa, com o usuário digitando as mensagens que deseja enviar para o outro e exibindo todas as mensagens da conversa (recebidas e enviadas). Caso ocorra um erro pelo usuário estar offline e a lista ainda não estar atualizada, a mensagem não é enviada.

1.5 - socket.client.KeepAlive

Aqui temos a thread do cliente que fica enviando mensagens de se o cliente está no ar. Enviamos a cada 0,5 segundo uma requisição, de maneira que o cliente não fique por muito tempo desatualizado com o status dos amigos. A cada requisição abrimos uma nova conexão, para não deixar a conexão aberta por muito tempo, já que o servidor só atualiza o status como offline depois de não receber 10 mensagens de keep alive (5 segundos),e manter muitas conexões simultâneas pode acarretar em perda de performance.

1.6 - socket.client.ReceiveMsg

Esta é a thread responsável por receber todas as mensagens enviadas pelos outros peers. Ela possui uma lista com todas as janelas de chat abertas e redireciona para a janela correta. Caso não houver janela entre estes peers, esta thread abre uma nova janela de conversa.

1.7 - socket.PutUserInactive

Esta é a thread do servidor que verifica há quanto tempo os clientes não enviaram uma mensagem de keep alive, removendo-o da lista de usuários online

1.8 - socket.server.RequestType

Esta classe é só um enumerador para cada tipo de mensagem, somente para abstrair parte do protocolo.

1.9 - socket.server.Server

Este é o servidor, responsável por enviar para os usuários e amigos em qual porta escutar/enviar as mensagens, receber as conexões dos usuários e enviar os amigos online/offline de cada peer.

2 - Padrões de Requisição e Resposta (Protocolos)

2.1 - Login

Requisição	Resposta
WindowLogin envia o cabeçalho (LOGIN) e na linha abaixo separando por virgulas o username, a senha e o número ip do client	O servidor responde 0 caso o usuário ou senha estejam incorretos. Caso OK o servidor responde 1, envia a lista de amigos daquele usuário e por último a porta que será usada pelo cliente para receber mensagens.
LOGIN <username>,<password>,<ip>	1 <friendlistsize> <username>,<status>,<ip>,<port> <username>,<status>,<ip>,<port> ... <port>

2.2 - Keep Alive

Requisição	Resposta
Thread KeepAlive envia cabeçalho através do socket (KEEPALIVE) e o username como forma de o servidor conseguir detectar o usuário que está requisitando o keepalive e retornar sua lista de amigos atualizada.	Servidor retorna lista de amigos do usuário que fez requisição keepalive atualizada. Utilizamos conexão não persistente para facilitar o gerenciamento
KEEPALIVE <username>	<friendlistsize> <username>,<status>,<ip>,<port> <username>,<status>,<ip>,<port>

	...

2.3 - Logout

Requisição	Resposta
Através da tela gui.WindowListFriends é feita a requisição para desconectar o cliente. Envia-se o cabeçalho (LOGOUT) e o username	Servidor responde -1 para caso o usuário tenha sido desconectado
LOGOUT <username>	-1

2.4 - Data

Requisição	Resposta
Para atualizar o usuário e porta da WindowTalk a mesma faz uma requisição Data enviando o cabeçalho (DATA) e o username que quer atualizar o ip e a porta, (no caso o que está mantendo conversa) naquela WindowTalk	Servidor responde o ip e a porta atualizados do username solicitado separados por virgula
DATA <selectedusername>	<selecteduserip>,<selecteduserport>

2.5 - Send Message

Requisição	Resposta
WindowTalk para enviar mensagem via socket ao outro user apenas envia a mensagem no corpo da requisição e na linha abaixo o seu username para que o outro que está recebendo a mensagem saiba quem enviou a mesma.	Como apenas necessita enviar e não precisa de confirmação de recebimento não há nada no retorno desta requisição. As conexões são não-persistentes, para não corrermos o risco dos peers ficarem com a conexão muito tempo aberta sem envio de dados, congestionando a rede
<msg> <username>	

3 - Como compilar e executar

3.1 - Servidor

A implementação do servidor foi feita para rodar em linha de comando. Antes de iniciarmos a compilação e execução do mesmo certifique-se de ter disponível em seu Prompt de comando os comandos: ‘javac’ e ‘java’. Para maiores informações de como configurar os comandos nas variaveis de ambiente verifique este link.

Vá até a pasta raiz do projeto “Socket-P2P-Network” e execute o arquivo “compileAndRunServer.bat”:

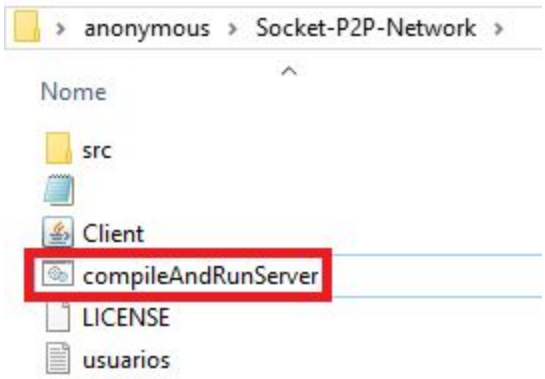

Figura 1. compileAndRunServer arquivo bat

Caso execute com sucesso:

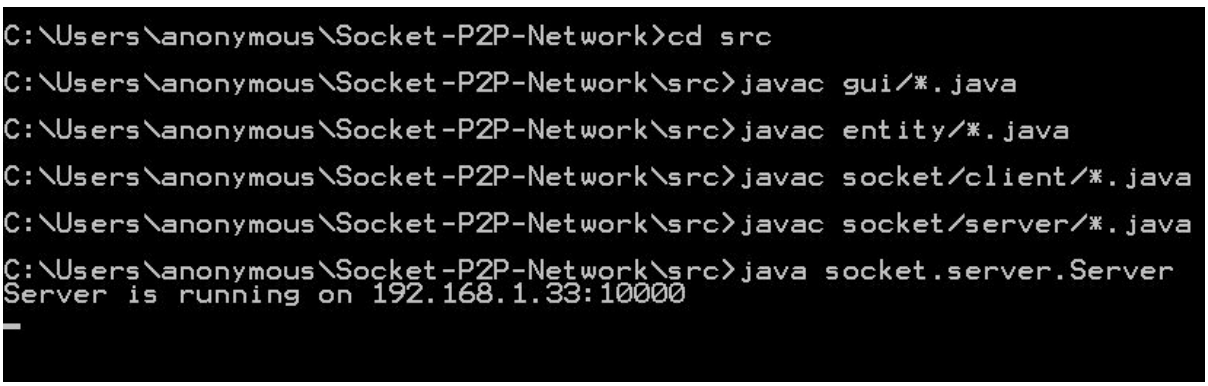

Figura 2. Servidor executado com sucesso aguardando requisições

3.2 - Cliente

Para executar nosso programa cliente para se conectar ao servidor basta na pasta raiz executar o arquivo “Client.jar”

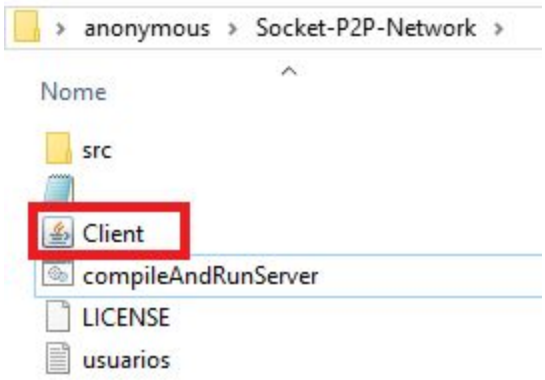

Figura 3. Arquivo Client.jar na pasta raiz

Utilize os usuários de teste listados abaixo para executar o login

Usuário	**Senha**
Adriano	123
Guilherme	123
Joao	123
Maria	123
Ana	123

Preencha todos os campos:

- usuário e senha
- IP Servidor - com o IP onde o servidor está sendo executado

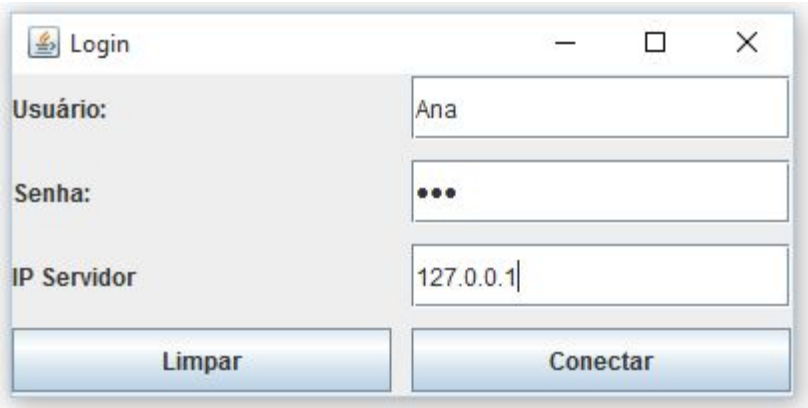

Figura 4. Tela de login

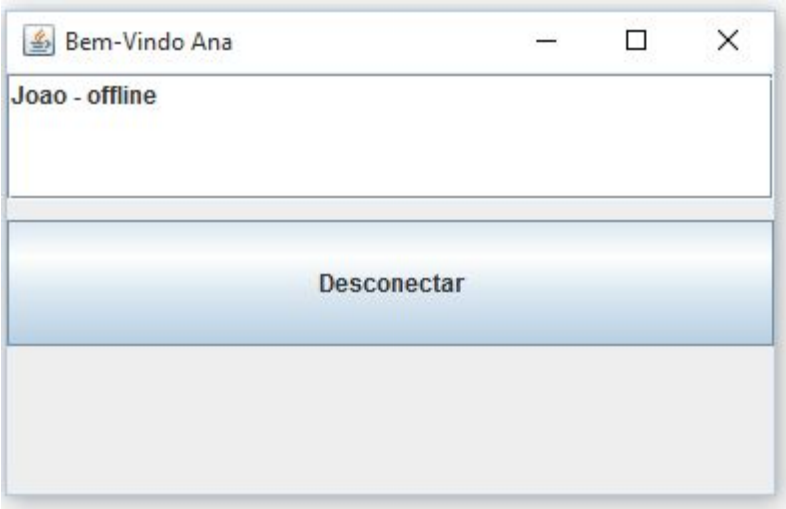

Figura 5. Tela de amigos usuário Ana

4 - Testes realizados

- Realizamos testes para conexão com o servidor, enviando os dados preenchidos de maneira incorreta ou não preenchidos e verificando se retornava mensagens de erro. Também testamos o cliente logando e deslogando para verificar se o servidor respondia a ele corretamente. Para este caso todos os testes foram OK.

- Outro teste realizado foi a atualização dos status dos amigos do cliente, logando e deslogando os amigos e vendo como ele reagia na atualização do status dos amigos. Apesar de uma certa demora (menos de 15 segundos) o status era alterado corretamente, e mesmo quando tentava-se conversar com um amigo que ficou offline mas não teve seu status atualizado, o cliente enviava uma mensagem falando que não conseguiu se conectar.

- Também testamos a parte do envio das mensagens entre os peers, ocorrendo problemas somente quando o usuário tentava enviar mensagens que terminavam com caractere vazio. Para os outros casos, não houve nenhuma manifestação de erro, mesmo testando os casos em que um usuário deslogava e logava e outro permanecia online e com os dois enviando as mensagens. A parte de gerenciamento das telas de conversa também funcionou OK.

- Foi testado tanto o caso de todos clientes e o servidor rodando em uma única máquina como em duas máquinas na mesma subrede (uma com o servidor e um cliente e outra com outro cliente). Estes casos também tiveram uma execução OK.

5 - Observações

- Optamos por utilizar conexões não-persistentes em todos as trocas de mensagens, mesmo no keep alive, em que temos um overhead de abertura de conexão, devido a

uma maior facilidade no gerenciamento e para não ficar com uma porta aberta por muito tempo sem utilização.

- A atualização da lista pode demorar alguns segundos (cerca de menos de 15 segundos) devido ao método utilizado da classe Swing não ser thread-safe

- No envio das mensagens, favor não pressionar o enter ou enviar ou barra de espaço, pois ficará com uma linha vazia e sua mensagem (antes do enter) não será enviada para o outro peer, pois o programa interpreta que é uma linha vazia, já que pega somente última linha (podendo acarretar em outros problemas de conexão).

- Acontece problemas quando há mais de um servidor rodando no mesmo ip, já que a porta é fixa, sendo lançada uma exceção por duas aplicações escutando na mesma porta.

- Rodar o .bat do servidor e o jar da aplicação no diretório que foi enviado, já que pode acarretar em problemas caso tentar rodar fora, principalmente o .bat que compila os arquivos antes de rodar.

- Um dos membros do grupo teve problema com a porta utilizada no servidor (Server.java) que já tinha outra aplicação utilizando. Neste caso, como a porta está fixa (10000), é necessário alterar manualmente em todas as conexões com o servidor e gerar novamente o .jar do cliente, pois as classes estão compiladas. Caso preferir, entre em contato com o grupo que alteramos, se necessário. Também pode remover essa aplicação que está rodando.

6 - Referências Utilizadas

http://stackoverflow.com/questions/26602338/java-p2p-multicast-chat-program-error-while-implementing-1-1-chat-between - apoio para implementação do p2p, principalmente na estrutura das classes

Redes de computadores e a Internet - James F. Kurose, Keith W. Ross

http://docs.oracle.com/javase/7/docs/api/java/net/Socket.html