

**Universidade do Minho**

Escola de Engenharia, Departamento de Informática

LEI 3º ano – Comunicações por Computador

“Music Game”

**Desenho e implementação de um jogo distribuído na Internet**

**Abstract.**

Este relatório visa a documentação de todo o processo e decisões implementadas durante a construção e implementação de um serviço de distribuição de conteúdos por transmissão de informação através do uso de *sockets* TCP e UDP. Os servidores terão acesso à informação através de comunicação por *sockets* TCP e, os clientes, receberão os dados enviados pelo servidor por *sockets* UDP. O desafio é, então, implementar um procedimento fiável, com controlo de fluxo e de erros, que possibilite a comunicação entre os diferentes participantes do jogo (servidores e clientes).

Bruno Pereira, 69303 João Mano 69854 Patrícia Rocha 67636

1. Introdução

O avanço tecnológico permitiu a introdução de aplicações únicas suportadas pela internet, que permitem o fornecimento de um dado serviço. Nestas aplicações, a distribuição de informação dos servidores para os clientes (habitualmente sempre em maior número), é das maiores preocupações a ter aquando o desenvolvimento dos programas.

Para que a comunicação entre as diversas entidades envolvidas seja possível e para que a sua eficácia e desempenho seja também, tão elevada quanto possível, é necessário projetar, instalar e operar redes otimizadas de comunicação que têm em conta paradigmas como a fiabilidade, disponibilidade, atrasos e débitos dos serviços fornecidos.

Assim sendo, como objetivo deste projeto, será construído uma aplicação que fornecerá um jogo sobre música a todos os seus clientes, implementando elementos como rankings, pontuações, desafios entre jogadores, etc. Esta aplicação, sendo suportada pela internet, exige a construção de um protocolo de comunicação que será então o objetivo principal, ao longo deste documento será explicitado o processo de construção do mesmo.

1. **Arquitetura da Aplicação**

A solução desenvolvida implementa a seguinte estrutura, existe um servidor denominado MusicServer que, quando é inicializado “parte-se” em duas threads de escuta, a thread AtendimentoCliente que estará sempre à escuta de pedidos UDP, direcionada, como o próprio nome indica, para atender clientes, e a thread AtendimentoServidor que estará à escuta para pedidos TCP, de outros servidores. Esta última thread ainda se encarrega de, caso o servidor não seja primário, registar-se perante o servidor primário.

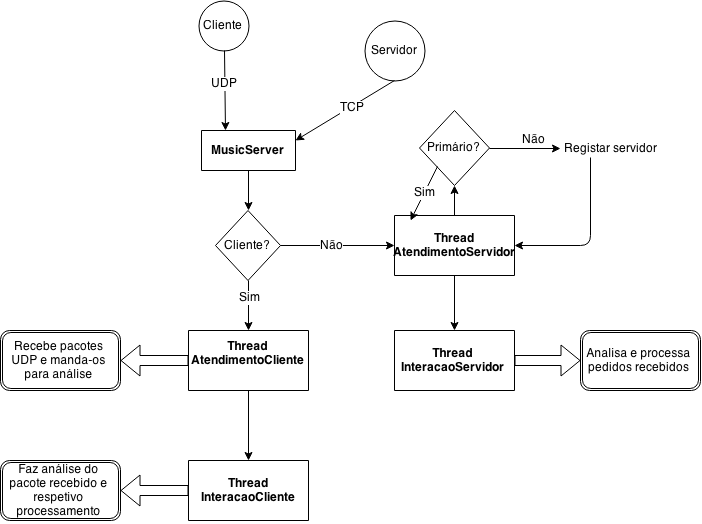


Figura 1 – Processamento de pedidos

Após a criação das threads, sempre que é recebido um pacote, visto que é preciso estar sempre “à escuta”, são criadas a thread de análise e processamento dos pedidos, denominadas InteracaoCliente para pedidos provenientes dos clientes e, InteracaoServidor para pedidos provenientes de outros servidores. No entanto existe um caso particular em que a thread InteracaoCliente tem de enviar e receber pacotes por TCP, este caso aparece quando o utilizador pede para jogar um jogo não local, neste caso o servidor, na sua thread InteracaoCliente enviará um pedido por TCP ao servidor dono do desafio e este retorna-lhe todas as informações necessárias para o desafio, o jogo decorre depois normalente (abordado mais à frente).

A estrutura utilizada para armazenar os dados, denominada BD é constituída por:

* Um HashMap de Utilizadores (chave é a alcunha);
* Um HashMap para o ranking local e outro para o ranking global (mapeiam a alcunha de um jogador para a sua pontuação local ou global respetivamente);
* Um HashMap de desafios locais e outro para os desafios globais (mapeiam a string com o nome do desafio para a data de início);
* ACHO QUE VAI MUDARUm HashMap listaDesafiosServidores para conseguir associar ao nome do desafio não local o InetAddress e a porta do servidor a quem é preciso pedir as perguntas;
* Um ArrayList de perguntas;
* Um HashMap de servidores (mapeia um InetAddress de um servidor à sua porta);
* Uma string com o nome da pasta de músicas e outra com o nome da pasta de imagens.

Esta estrutura é comum a todos os servidores e a todas as threads pertencentes ao servidor (threads de atendimento, processamento).

Note-se que o uso de threads é mais que necessário para a “libertação” dos servidores, de modo a que estes estejam constantemente a receber pedidos e a reencaminhá-los para outra thread, tal como um rececionista, caso assim não fosse o servidor teria uma capacidade muito menor de atender pedidos!

* 1. **Cliente**

Após a entrada de um pedido na thread AtendimentoCliente é criada então, tal como já foi referido, a thread InteracaoCliente. Esta thread analisa o tipo de pedido recebido (login, registo, aceitar jogo, etc.) e encarrega-se de tratar do seu processamento.

De todos os diversos pedidos abordaremos apenas aqueles que achamos de maior relevância, outros como Login, Registo, etc. são bastante simples e por isso não mencionados.

* + 1. **Jogar Desafio**

Tanto a aceitação como a criação de um desafio implicam a criação de uma nova thread, Jogo, para fazer os clientes aguardarem pelo jogo.

Ao criar um desafio é introduzido o desafio localmente (no HashMap de desafios locais e no de globais) e enviada a informação de um novo desafio para os restantes servidores, estes vão atualizar a estrutura de desafios globais. É ainda introduzido o jogador no desafio.

De seguida, no servidor, é criada a thread Jogo que, espera até à hora do desafio chegar e, posteriormente envia a primeira pergunta do desafio ao jogador respetivo se estiver tudo bem. Isto acontece também no pedido de aceitar desafio.

Assim quando o utilizador recebe a pergunta espera até receber o último pacote e realiza uma validação para verificar se todos os blocos foram recebidos, caso não tenham sido, é pedido um retransmit do bloco em falta.

Tanto os pedidos de retransmit como as respostas às questões são recebidas pela thread de escuta, AtendimentoCliente reencaminhadas à thread InteracaoCliente e, caso seja resposta à pergunta e não seja a última é enviada a nova pergunta e o processo repetido, caso seja a última é esperado que todos os utilizadores terminem e então enviadas as cotações do desafio para todos os utilizadores.

Utilizamos uma thread neste ponto para, de alguma forma, estruturar a aplicação, assim esta thread torna-se responsável por fazer a espera do utilizador e iniciar o jogo.

É ainda importante explicar o que se sucede quando o desafio não é local, o início do processo, como já tinha sido referido, a thread InteracaoCliente pede por TCP tudo o que precisa para a realização do desafio e recebe, guardando a informação na pasta correta. Depois o processo decorre normalmente, ou seja, é criada a thread Jogo.

* + 1. **Controlo de perdas**

O controlo de perdas de pacote é feito na parte do cliente, sempre que existe um timeout, isto é quando uma resposta não chega entre um determinado período de tempo, é lançado uma TimeOutException que é apanhada e despoleta um retransmit do último pacote enviado. Desta forma é requerido que o último pacote enviado seja armazenado por parte do cliente.

Note-se que o retransmit não é infinito, ou seja, após um número estipulado de retransmit’s não se volta a pedir uma retransmissão e é lançada a exceção ServerUnreachable.

Ao retransmitir o pacote enviado será obtida uma nova resposta do servidor.

* + 1. **Gestão de Concorrência**

A gestão de concorrência é feita com a aplicação de synchronized aos métodos de alteração de conteúdo das estruturas partilhadas.

Apesar de não ser a solução mais eficaz é uma solução funcional e rápida na execução/implementação.

* 1. **Servidor**
     1. **Registo de Servidor Secundário**

Quando um novo servidor é criado, é feita a pergunta se este é primário ou secundário, se for secundário é necessário que este se registe perante os servidor primário e, por sua vez o primário envia a informação do novo servidor aos restantes. Para isto é pedido o Ip e porta Tcp do servidor primário e, enviado com a função **registaServidor** da thread AtendimentoServidor ao servidor primário um pedido de registo com o código regista servidor com resposta. O servidor primário regista então o servidor na sua base de dados e, envia para os restantes servidores que conhece os dados do novo servidor com um código de registo de servidor sem resposta, ou seja estes apena adicionam o novo servidor na sua base de dados.

Ainda ao receber o pedido de registo o servidor envia toda a informação que tem ao servidor que se está a registar, assim este passa a ter conhecimento de toda a rede (servidores existentes, desafios, utilizadores, etc) e portanto passa a estar funcional.

* 1. **Outras Estruturas**
  2. **PDU**

A classe PDU representa o pacote que permite a comunicação entre os diversos componentes.

Falta aqui!!!!!!!!!!!!!!!!!!!!

* 1. **Pergunta**

1. **Mudanças em Relação ao Enunciado**

Foram aplicadas algumas alterações em relação ao enunciado, nomeadamente:

* As datas dos desafios só utilizam 5 bytes, 3 para o ano, 1 para o mês e 1 para o dia.
* As horas apenas usam 3 bytes, um para cada campo (hora, minuto, segundo).

As últimas duas mudanças foram executadas para poupar espaço.

Alterou-se ainda na transferência de informação de servidor para servidor (código 14) foram associados códigos às diferentes intenções nos pedidos:

* Registo de um novo servidor: 32
* Registo de novo desafio: 33
* Lista de desafios disponíveis localmente: 34
* Registo de aceitação do desafio: 35
* Resultados de um desafio: 36
* Ranking dos utilizadores locais: 37
* Registar um servidor sem resposta: 38

Esta mudança surgiu da necessidade de mais facilmente se conseguir processar os pacotes recebido entre as comunicações servidor – servidor.

Falta as mudanças do servidor para pedir desafios e questões criamos um novo código bem como para se registar, explicar como funciona o registar!!!!!

1. **Implementação**
   1. **Funcionalidades requeridas**

Nesta secção, tendo em conta as funcionalidades requeridas e expostas no enunciado disponibilizado, será explicado como foram implementadas essas funcionalidades, sendo que como é natural, algumas já foram explicadas nas secções anteriores e apenas são referenciadas.

1. ***“Fazerem a gestão dos utilizadores da sua rede local (para cada jogador só se precisa manter informação de autenticação*** ***com*** ***nome ou alcunha e password e do seu score global)”:***

Com o Hashmap de utilizadores existente na estrutura de dados do servidor é possível após um pedido de autenticação de um cliente (onde são passados os dados necessários para a sua autenticação), consultar esta estrutura e responder ao pedido positivamente ou com um erro. Também, num pedido de registo há a verificação se o registo é possível, ou seja, se não existem outros utilizadores com os mesmos dados, etc. e, novamente é enviada uma resposta positiva ou de erro consoante o caso.

1. ***“Fazerem a gestão dos trechos musicais, imagens, respetivas perguntas e respostas;”:***

A estrutura conceptualizada possui o caminho para a pasta de imagens e outra para a pasta de músicas e ainda um arrayList de perguntas (ver secção de Falta aqui), que permitem gerir esta informação.

1. ***Fazerem a gestão do desenrolar dos jogos ativos dos utilizadores da sua rede local;***

Este item já foi explicado com detalhe na secção 2.2.1 – Jogar Desafio.

1. ***Num sistema com multi-servidores, anunciarem/distribuírem a informação local (dados dos desafios e respetivos resultados) pelos outros servidores;***

Mal existe a criação de um desafio o servidor envia a informação de um novo desafio e, caso seja aceite num outro servidor este envia ao servidor dono um pedido de REQUESTDESAFIO ao qual o servidor dono responde enviando a informação do desafio, para mais detalhes ver secção Falta aqui

1. ***Manterem uma lista local de Ranking;***

A estrutura de dados possui um HashMap, rankingLocal que mapeia um nome de utilizador à sua pontuação.

1. ***Num sistema com multi-servidores, registar um novo servidor e anuncia-lo aos outros servidores do sistema, etc.***

Ver secção Falta aqui

* 1. **Cliente -> Servidor**

É importante mencionar que foi criada uma interface em JavaFX para tornar o programa mais apelativo ao uso. Assim sendo passaremos agora à explicação de como está implementada a classe que corresponde ao cliente, MusicClient.

Inicialmente é enviado com o **menuInit** um hello ao servidor e recebida a resposta, de seguida são apresentadas as opções de autenticação ou registo, para autenticação tem-se a função **menuLogin** que envia um pacote com a alcunha e a password e recebe uma resposta de erro ou de validação com os restantes parâmetros do utilizador (pontuação, nome, etc.). Para registo tem-se a função **menuRegista** que envia os parâmetros de registo (nome, alcunha, password) espera uma resposta e, retorna um valor inteiro igual a 0 se houver erro e 1 caso o registo seja bem-sucedido.

Por forma a explicar o que realizam as restantes funções serão, de seguida, enumeradas e explicadas:

1. **menuLogout:** função que envia um pacote de logout ao servidor e capta a resposta.
2. **receivePDU:** função que se encarrega de receber um pacote do servidor e, caso exista um timeout volta a enviar o pacote, para causar uma nova resposta do servidor.
3. **receivePDUNoException:** esta função é exatamente igual à anterior mas, sem pedido de retransmit caso não receba resposta, surgiu para resolver o seguinte caso: imagine-se que um jogador é muito rápido e termina o jogo e espera por uma resposta não deveria ser posto um tempo mínimo de resposta mas sim exigido que esperasse o tempo que fosse preciso, isto poderá ser prejudicial caso o pacote se perda, exigindo ao fecho da aplicação.
4. **sendPDU:** função que recebe um conjunto de dados, constrói um pacote e envia-o ao servidor.
5. **menuQuit:** esta função envia um pacote que identifica a desistência de um desafio. OUTROS SERVIDORES DEVIAM SER INFORMADOS!!!!!!!!!!!
6. **menuEnd:** envia um pacote que simboliza o término do desafio por parte deste utilizador e recebe as pontuações do desafio.
7. **menuDelete: Falta ver se funciona!!!!!!!!!**
8. **answer:** envia a resposta ao servidor, recebe a informação sobre os pontos conseguidos, qual a resposta certa, etc. e mostra-a ao utilizador.
9. **proximaPergunta:** pede ao servidor que envie a próxima pergunta e manda-a apresentar ao utilizador.
10. **menuListRankings:** envia o pedido de listar os rankings e vai construindo a lista com a ajuda da **getNextUserRanking** que capta a pontuação do próximo utilizador.
11. **menuListChallenge:** semelhante à anterior mas direcionada para os desafios e com a ajuda da **getNextDesafio**.
12. **acceptChallenge:** envia ao servidor a informação de que o cliente quer aceitar um dado desafio.
13. **menuMakeChallenge:** envia ao servidor a informação de que o utilizador quer criar um novo desafio e todos os dados necessários desse desafio (nome).OUTROS SERVIDORES DEVIAM SER INFORMADOS!!!!!!!!!!!
14. **Jogar:** encarrega-se de receber a informação de uma questão, a música e imagem são recebidas pelo uso da função **recebeBlocos** e, logo de seguida é chamada a função **checkBlocos** que verifica se todos os blocos de música e imagem foram bem recebidos e, caso não tenham sido é utilizada a função **askBlockRetransmit,** que envia ao servidor um pedido de retransmissão de um dado bloco de música ou imagem e recebe-o armazenando-o no devido lugar.
15. **constroiFicheiroAudio:** tal como o nome indica, esta função constrói o ficheiro áudio que pega nos blocos recebidos, anteriormente armazenados num treemap, e junta-os num único byte array e cria o ficheiro áudio numa pasta temporária.
16. **constroiFicheiroImagem:** idêntica à anterior mas para a imagem.
    1. **Servidor -> Cliente**

Nesta parte do relatório serão documentadas as funções que o servidor utiliza para responder ao cliente, ou seja, as funções implementadas na classe InteracaoCliente.

1. **analisaPacote:** função que analisa o pacote recebido, dependendo do tipo de pedido, o pacote é processado de diferentes maneiras.
2. **quitDesafio:** esta função recebe e processa pacotes de desistência de um desafio, ou seja, retira o dado utilizador do desafio e este não ganha nem perde pontos.
3. **fimDesafio:** responsável pela atualização das pontuações e envio das cotações do desafio aos utilizadores.
4. **processaLogin:** verifica se os dados de login estão corretos e envia ou um pacote de erro caso não esteja tudo bem ou então um pacote com o InetAdress e com a porta associada ao utilizador.
5. **processaRegisto:** recebe as informações de registo se não houverem conflitos regista o novo utilizador e envia um pacote de OK, senão envia um pacote de erro e o utilizador não é registado.
6. **listaDesafios:** envia um pacote por cada desafio existente ainda disponível para jogar, com a indicação de continua ou não caso não existam mais desafios a enviar. Se não existirem desafios a listar é enviado um pacote de erro correspondendo a não existem desafios.
7. **listaRanking:** funciona exatamente igual à anterior mas para os utilizadores existentes em vez dos desafios, com a exceção de nunca ser enviado pacote de erro porque existe sempre pelo menos um pacote a enviar (o do próprio utilizador).
8. **criaDesafio:** cria aqui um desafio com os dados recebidos, caso já exista é enviado um pacote de erro com a mensagem de desafio existente, caso seja bem-sucedida a criação, é enviado um pacote com os dados do desafio incluindo a data de início e, posteriormente, criada a thread jogo.
9. VERIFICAR TODA A SECÇÃO POR POSSÍVEIS MUDANÇAS!!!!!!!!!!!
   1. **Servidor->Servidor**
10. **Testes e Resultados**

Aqui ficam todos os testes realizados e respetivos resultados.

**FALTAM TESTES**

1. **Conclusões e Trabalho Futuro**