Universidade de Évora - Colégio Luís António Verney

2022-2023

Relatório

- Redes de Computadores -

2º ano da licenciatura de Engenharia Informática

Henrique Rosa nº 51923 Diogo Matos nº 54466

Índice

Introdução .....pág 3
Instruções .....pág 4
Ficheiros .....pág 6
Funcionamento do Programa .....pág 8
Escolhas técnicas .....pág 11
Problemas técnicos .....pág 12
Conclusão .....pág 13
Bibliografia .....pág 14

Introdução

O trabalho prático consta da criação de um servidor TCP/IP cujo funcionamento simula o funcionamento de uma aula. Optamos por programar o Servidor em Java. O servidor que pretendemos criar contém sistemas para: marcação de presenças; Autenticação por meio de número de aluno e password; Fazer perguntas e administrar respostas a estas; Fazer *upload* e *download* de ficheiros; Calcular a média de presenças de um aluno; Guardar os dados armazenados dos diferentes sistemas listados em ficheiros para possibilitar a permanência de dados. Todos os comandos e todos outputs em terminal do server foram escritos em inglês para que a apresentação estética do servidor fosse idêntica à do enunciado do trabalho

Este relatório tem o propósito de fornecer informação sobre o funcionamento do Servidor e das suas várias componentes, bem como expor a sua arquitetura. Serão dados exemplos de testes que realizamos, em visa a dar contexto a certos problemas técnicos. Também serão listados as instruções escritas para que o Servidor funcione de acordo com o previsto.

Instruções

[!] Todos os comandos listados deverão ser inseridos em formato de texto no terminal do Cliente.
[!] Não são aceites quaisquer instruções no terminal do Servidor

IAM [número com "l")] **withpass** [password] - Realiza a entrada na aula do aluno cujo número é inserido, caso a autenticação com a sua password seja válida (não é possível entrar com dados incorretos). Aquando da sua entrada, a presença do aluno é automaticamente marcada, e o aluno passa a poder usar o resto dos comandos listados a seguir, o qual seria impossível sem a sua autenticação.

ASK [pergunta] - A pergunta inserida é adicionada à lista de perguntas para que possa ser respondida por outros clientes. A essa pergunta será atribuído um número.

ANSWER [número da pergunta] - A resposta inserida é adicionada à lista de respostas da pergunta com o número inserido.

ERASE [número da pergunta] - Para a pergunta cujo o número é indicado, a resposta mais recente feita pelo cliente é apagada. Não é possível apagar respostas de outros clientes. Este comando foi implementado para que haja uma forma de corrigir um possível erro na inserção de uma resposta.

LISTQUESTIONS - Apresenta uma lista com todas as perguntas feitas até ao momento, acompanhadas dos seus respectivos números. Também são

apresentadas, por baixo de cada pergunta, as respectivas respostas, cada uma acompanhada do número do aluno que “mandou” a resposta.

PUTFILE [nome do ficheiro] - lê o ficheiro cujo o nome é indicado (caso um ficheiro com esse exista na diretoria) e envia o seu nome e o seu conteúdo* para o server, onde o ficheiro vai ser “clonado” - vai ser criado um ficheiro com o mesmo nome e o mesmo conteúdo.

GETFILE [número do ficheiro] [tipo de ficheiro] - Inversamente à instrução imediatamente acima, importa , do servidor, o conteúdo* do ficheiro cujo o número é indicado (ver instruções abaixo) e cria um ficheiro na diretoria do cliente com o mesmo conteúdo e com o nome “ficheiro_recebido.ext” sendo “ext” a extensão do tipo de ficheiro indicado na instrução. Esta instrução requer que o número de ficheiros exista na lista de ficheiros.

LISTFILES - Apresenta uma lista com o nome de todos os ficheiros presentes no servidor até ao momento, cada um desses nomes acompanhados com um número para que possam ser identificados quando for necessário executar a instrução imediatamente acima.

*As instruções PUTFILE e GETFILE padecem de alguns erros técnicos, pelo que é importante ver a secção de “Problemas Técnicos” antes de as executar e avaliar os seus resultados. Para localização desta secção, consulte o Índice no início do relatório

Ficheiros

[!] No que toca a todas as funcionalidades que operem com ficheiros, o servidor só vê os ficheiros na sua diretoria (ver secção sobre o funcionamento do programa, sobre as diretorias), e o mesmo acontece ao cliente.

Na diretoria do **Servidor**, dentro da pasta "files" estarão presentes os ficheiros com os seguintes nomes e estruturas:

Data_files.txt - Contém o número total de ficheiros, bem como o nome e tamanho de cada um dos ficheiros enviados pelos clientes para o servidor, em formato de texto simples, em visa a estabelecer a permanência de dados no que toca aos ficheiros do servidor. Na primeira linha tem o número de ficheiros. Nas linhas seguintes estarão sequências de duas linhas: uma linha com o nome do ficheiro (e.g data.txt) e outra linha com o tamanho deste.

Data_questions_ans.txt - Contém as informações necessárias à persistência de dados das perguntas e respostas da aula, em formato de texto simples. A primeira linha contém o número de questões, seguido de *n* linhas com uma pergunta em cada uma, sendo n o número de perguntas. Depois teremos sequências começadas pelo número de respostas a uma determinada pergunta (determinada pela leitura sequencial do ficheiro), ao qual segue o dobro desse número de linhas: uma linha com o número de aluno que escreveu a resposta e outra linha com a resposta. Se o número de respostas para uma pergunta for zero (não houve respostas), passar-se-á para a próxima sequência de respostas, para a próxima pergunta.

Data_user.txt - Contém as informações necessárias para a persistência de dados quanto às presenças dos alunos. A primeira linha vai conter o número de presenças marcadas seguida desse numero de sequencias de três linhas: a primeira com o nome do aluno para o qual a presença foi marcada, a segunda com a pass desse aluno, e a terceira com o valor da presença, que pode variar entre "PRESENT", "LATE" e "MISS", sendo estas as representações de "presente", "atrasado" e "faltou", respectivamente.

Na diretoria do **servidor** também serão criados e guardados os ficheiros enviados pelos vários clientes, identificados pelo nome importado aquando do seu envio para o servidor.

Na diretoria do **Cliente**, dentro da pasta "files", aquando da execução da instrução "GETFILE", vai ser criado (se já existente, modificado) um ficheiro chamado "**ficheiro_recebido",** (consultar a instrução "GETFILE" na secção de instruções). É requerido pelo funcionamento do Servidor que, dentro da mesma pasta "Files", estejam presentes os ficheiros (com o nome correto) a ser enviados por meio da instrução "PUTFILE", para que estes possam ser enviados (Consultar a instrução "PUTFILE" na secção de Instruções) .

Funcionamento do Programa

- Diretorias -

O programa está estruturado em duas diretorias independentes: Uma pasta chamada “Server” e outra chamada “Client”. Também haverá uma pasta chamada “Media” que contém o código da apresentação da média do aluno (ver o terceiro parágrafo da secção de problemas técnicos)

Dentro da pasta “Server” estarão duas pastas: a pasta “Files” e a pasta ”src”. na pasta “Files” estarão os ficheiros a ser enviados e recebidos pelo server e ficheiros necessários ao funcionamento do server. Na pasta “src” encontrar-se-á o código do server em dois ficheiros com os nomes “Server.java” e “ClientHandler.java”.

A Pasta “Client” tem uma estrutura idêntica, com uma pasta “Files” - que irá irá eventualmente conter os ficheiros que o cliente queira enviar/receber. - e uma pasta “src” que irá conter o código do cliente num ficheiro com o nome “Client.java”

A Pasta “Media” apenas irá conter o código no ficheiro Media_Pres.java e, eventualmente, o ficheiro com as presenças para poder ser feita a média

- Funcionamento do server -

Server.java: Aquando da sua execução, primeiramente faz a verificação dos ficheiros e o “load” desses ficheiros para a sua correta disseminação conforme o explicitado na secção de instruções. Depois, cria um objeto do tipo “Server”, com um “server socket” na porta 5555 e “inicia” o server, definindo uma hora local de início (para a contagem do tempo da aula) e entra num ciclo que repete até que a socket do server seja fechada ou o tempo decorrido da aula seja excedente de duas horas. A cada iteração deste ciclo, aceitar-se-ão novas conexões de clientes usando o comando “accept( )", abrindo uma thread do “ClientHandler” para cada

uma delas, de modo a que o server não bloqueie a cada nova conexão, e possa aceitar vários clientes ao mesmo tempo, enviando a socket do cliente para a respetiva thread “ClientHandler” para que cada cliente possa ser “tratado”. Em cada iteração, será também criada uma instância da classe “PersistenceTask”, que vai contar o tempo decorrido (isto é feito para que a contagem do tempo não dependa do ciclo, que vai ficar bloqueado de acordo com o funcionamento da função accept( )). Essa nova “PersistenceTask” tem como função principal, salvar os dados na pasta “Files”, e possui um método “run” (para o funcionamento do qual é necessário que a classe “PersistanceTask” implemente a interface “Runnable), o qual é executado de acordo com o agendamento de execução da task de persistência (neste caso, de 5 em 5 minutos).Caso a execução do ficheiro “Server.java” seja interrompida ou uma exceção lançada, o “server socket” será fechado e o funcionamento do server pára.

Client.java: Aquando da sua execução, cria um novo objeto “Client” com uma socket ligada na porta 5555 localhost, ligando-se assim ao servidor, se este já estiver a ser executado. Depois chama as funções “messageFromServer ( )” e “sendMessage ( )”. A função “messageFromServer ( )” vai possibilitar ao cliente receber mensagens do server, por meio análogo ao funcionamento do próprio server: cria uma nova thread com um loop que repete até que a conexão da socket seja fechada, o qual, por cada iteração, lê o input vindo do server, para que ele possa ser lido e interpretado pelo “clientHandler” respectivo ao cliente atual (cada cliente tem um). A função “sendMessage ( )” vai executar um loop idêntico, o qual, a cada iteração, vai ler e interpretar o input do próprio cliente (por meio do terminal do usuário), de acordo com as especificações do enunciado, para que o funcionamento das instruções (descritas na secção de instruções) seja devidamente desempenhado. Para a correta interpretação do input do usuário, este é sujeito a verificação de conteúdo por meio de diversas condições (if’s) que garantem que o input está correto para a devida execução das funções do cliente.

ClientHandler.java: O código neste ficheiro desempenha uma função de utilidade: esta classe vai desempenhar uma função de ponte entre o servidor e o cliente, sendo nele onde maior parte da lógica operativa do nosso código reside. A classe ClientHandler vai criar uma instância sua para cada cliente conectado ao server, para que essa instância possa tratar desse cliente, disponibilizando todas as funções até agora descritas, para que o cliente possa comunicar com o server de acordo com o enunciado. Esta classe desempenha inúmeras tarefas, dentre as quais:

- Conexão com o cliente e com o server, fazendo de ponte entre estes;
- Importação da informação presente nos ficheiros de persistência para variáveis e listas de instância;
- Leitura de Input e escrita de Output;
- Autenticação do aluno, verificando se esse aluno já existe na lista das presenças (se não, cria um novo objeto do tipo aluno e adiciona-o);
- Marcação de presenças, e a sua exportação para ficheiro;
- logística de perguntas e respostas, e a sua exportação para ficheiro;
- logística de upload e download do ficheiro (por envio do conteúdo do ficheiro para cada uma das partes, dependendo do que se pretenda fazer)

Para todas as tarefas listadas, e para que estas sejam desempenhadas corretamente, este ficheiro possui métodos de utilidade com verificações e lógica funcional variada dependendo da função e utilidade que desempenham.

Nota Importante: Neste relatório, a descrição do funcionamento do server pode ser um pouco vago e não informativo, mas essa foi uma escolha consciente, já que, nos ficheiros de código, incluímos comentários que fazem uma melhor explicação de como o server funciona em detalhe, e não seria de grande uso que estivessemos a repetir informação que já está disponível e que é relativamente melhor do que qualquer tentativa possível de descrição mais detalhada feita aqui (acabaria por ser mais abstrato e de difícil compreensão para o leitor), pelo que, se for necessário uma descrição detalhada do código, recomendamos que sejam tomados em conta os comentários do próprio código.

Escolhas Técnicas

Por escolha de ambos os integrantes do grupo, utilizámos o IDE “ IntelliJ “ da JetBrains para a realização deste trabalho, ambos achámos mais fácil, prático, e este IDE também nos facilita em cooperação remota por meio da extensão “code together” que nos deixa editar o mesmo código em conjunto.

Por falta de conhecimento sobre a linguagem java, ambos decidimos fazer pesquisas extensivas na internet sobre métodos e lógica de código que nos ajudaram a realizar o trabalho, pelo que várias das implementações presentes no código foram inspiradas diretamente de conteúdo presente na internet (sob a forma de vídeos da plataforma youtube, excertos de livros sobre programação em java, fóruns comunitários e diversos sites de caráter didático) . No final deste documento, incluímos uma bibliografia de todos os materiais dos quais tirámos inspiração para realizar o trabalho.

Problemas Técnicos

Embora esteja especificado no enunciado do trabalho prático, não conseguimos implementar o servidor de modo a limitar o tamanho das mensagens a 1024 bytes, pelo que algumas mensagens comportam muito mais do que esse valor ,principalmente no que toca à emissão e receção de ficheiros.

No que conta a transferência de ficheiros: Apesar da transmissão de ficheiros de texto funcionar corretamente, tanto a emissão quanto a receção de ficheiro de imagens (só foram testados ficheiros do tipo.PNG) não estão a funcionar corretamente, pelo que a imagem só é parcialmente emitida/recebida - a imagem fica cortada a meio (ás vezes até menos).

Aquando da fase testes realizada, mudámos de IDE, do IntelliJ para o VSCode. Isto foi feito porque não nos foi possível testar o código no IntelliJ com diretorias diferentes, o IDE não aceitava que o código estivesse separado em pastas diferentes. Nisto, já a testar no VSCode, ocasionalmente ocorria que a criação de ficheiro para o destino não era devidamente realizada, pelo que o conteúdo do ficheiro era apenas apresentado no terminal e não escrito no ficheiro (certas vezes o ficheiro nem era criado), embora o código a testar fosse idêntico (ou até mesmo igual) ao testado no IntelliJ.

Como teríamos de mudar a base da implementação do server, a funcionalidade da apresentação da média do aluno foi feita separadamente do resto do código, pelo que esta funcionalidade pode não ter sido implementada de acordo com o pretendido no enunciado. Mesmo assim achámos pertinente incluí-la no trabalho, já que a lógica já estava feita, e seria trabalho “mandado ao lixo”

Para todos os problemas técnicos listados, foram feitos inúmeros testes, mas mesmo assim, e com grande desilusão, não nos foi possível corrigi-los.

Conclusão

Neste trabalho prático conseguimos abordar técnicas práticas da criação de servers e de programação java.sockets, pondo em prática matéria leccionada na aula, e tirando partido da informação exposta nessa matéria para criarmos um server TCP/IP multi-cliente funcional e "user-friendly". Embora tenhamos experienciado alguns problemas técnicos, o trabalho prático foi bastante educativo, e é da opinião de ambos que foi desafiante e, por falta de melhor termo, "divertido".

Bibliografia

Vídeos:

⟶ https://www.youtube.com/watch?v=vsWoSpQ6Dww

⟶ https://www.youtube.com/watch?v=GLrlwwyd1gY&t=589s

Sites/Fóruns:

⟶ https://www.programiz.com

⟶ https://stackoverflow.com

Documentos:

⟶https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ScheduledExecutorService.html

⟶https://docs.oracle.com/javase/8/docs/api/java/io/FileOutputStream.html

⟶https://www.codejava.net/java-se/networking/java-socket-server-examples-tcp-ip

⟶https://docs.oracle.com/javase/8/docs/api/java/io/InputStreamReader.html

⟶https://docs.oracle.com/javase/8/docs/api/java/io/OutputStream.html