FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

ucDrive: Repositório de ficheiros na UC – Meta 1

Sistemas Distribuídos

Licenciatura em Engenharia Informática

2021/2022

Duarte Emanuel Ramos Meneses – 2019216949 – duartemeneses@student.dei.uc.pt

Patrícia Beatriz Silva Costa – 2019213995 – patriciacosta@student.dei.uc.pt

Índice

Introdução

Neste trabalho prático foi-nos pedido que desenvolvêssemos uma plataforma capaz de replicar um sistema de armazenamento de ficheiros em servidores acedidos por alunos e professores. Foi também imposto que as ligações entre clientes e servidores fossem feitas através de TCP e a comunicação entre servidores feitas em UDP.

Ao longo deste relatório vamos explicar detalhadamente a arquitetura de software, o funcionamento do servidor, os testes que realizamos à plataforma e o que implementamos para assegurar fiabilidade e consistência à nossa aplicação.

Arquitetura de software

Sendo que nos era pedido para ter clientes, um servidor primário e outro secundário, decidimos criar um ficheiro java para cada um. Apesar de os servidores terem funções idênticas, entendemos que fazia sentido ter dois ficheiros distintos pois cada um lê ficheiros de texto diferentes.

Era-nos requerido também que as ligações entre os clientes e o servidor primário fossem via TCP e que as ligações entre os servidores fossem através de UDP. Deste modo, o servidor primário ao ser ligado e tendo os dados nos ficheiros de texto todos corretos, fica à espera de ligações TCP dos clientes. Para cada ligação, vai criar uma thread para tratar individualmente de cada cliente.

Cada servidor está preparado para ser primário ou secundário. Quando se inicia cada um, estes leem as suas configurações de um ficheiro de texto (o seu endereço, porto TCP para comunicar com os clientes, porto UDP para comunicar entre si, porto UDP para troca de ficheiros, o endereço absoluto para as pastas do servidor, o número máximo possível de pings falhados e o tempo entre cada ping). Tendo isto, o servidor a ser ligado vai tentar criar uma conexão UDP no porto especificado para comunicar com o outro servidor. Se não conseguir, é sinal que o outro se conectou primeiro sendo primário. Deste modo, o que se estava a tentar conectar agora passa a ser secundário.

Se o servidor for secundário, vai ter apenas duas ligações UDP para comunicar com o outro servidor: uma para troca de heartbeats (para garantir que o servidor primário não tem problemas e deixa os clientes sem acesso à plataforma) e outra para receber os ficheiros que são colocados no servidor primário e o ficheiro de configuração sobre os clientes (para ter a mesma informação nos dois servidores, mantendo-a persistente). Para receber ficheiros criamos uma thread que irá tratar dos ficheiros de cada cliente. Já para trocar heartbeats com o servidor primário, fazemos sem thread uma vez que esta funcionalidade não está relacionada com os clientes, não necessitando de existir uma troca por cada cliente.

Já se o servidor for primário, vai ter também as duas ligações já referidas acima, e outras duas TCP. Estas duas ligações via TCP servem para comunicar com os clientes. Uma é iniciada mal o utilizador se liga e serve para troca de comandos. A outra apenas é aberta caso o cliente pretenda fazer o upload ou download de ficheiros e serve para esta troca de informação.

Cada cliente no servidor primário é definido como uma thread para conseguir ter tratamento independente dos outros. Logicamente, a troca de ficheiros no servidor com o utilizador é feita através de threads para evitar conflitos de informação entre clientes. Também a resposta aos heartbits é feita com uma thread.

Na aplicação cliente existem no máximo dois sockets em simultâneo. Esta vai ler de um ficheiro de texto os portos e endereços aos quais se deve tentar conectar (de ambos os servidores). Conseguindo conectar-se com o servidor primário via TCP, apenas volta a criar um novo socket TCP caso pretenda fazer troca de ficheiros (socket esse criado no porto “TCP que já se está a utilizar + 1”).

Sendo que cada cliente terá a sua própria aplicação a correr, entendemos que não seria necessário ter threads aqui uma vez que cada cliente já vai estar isolado dos outros, não necessitando de paralelismo.

Em suma, a arquitetura da nossa plataforma está representada na imagem abaixo:

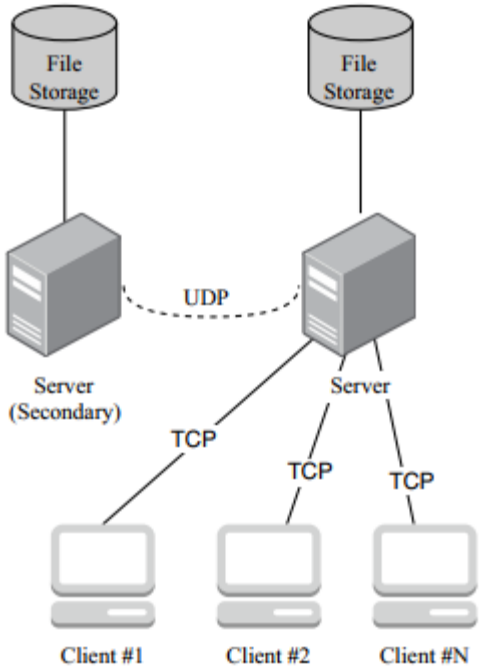

Figura 1 - Arquitetura da plataforma

Fica assim evidente que cada servidor tem armazenamento de informação. Esse armazenamento é feito através de sistema de ficheiros, sendo o endereço absoluto para a diretoria em questão definido nos ficheiros de configuração dos servidores.

O código do servidor primário está estruturado de forma a que, após a leitura das informações vindas dos ficheiros de configuração, crie uma thread para comunicar com o servidor secundário para troca de heartbits e fique num ciclo infinito para aceitar clientes e, aquando desta ação, criar a thread responsável por cada cliente no servidor. Decidimos definir também a tarefa de responder aos heartbits numa thread pois precisávamos que essa ação fosse feita em paralelo com outras.

No que diz respeito à thread responsável por cada cliente no servidor principal, recebe sempre a opção que o cliente escolheu e, a partir daí, direciona a execução para a parte pretendida. Apenas cria outra thread caso o utilizador necessite de fazer upload ou download de ficheiros. Neste caso, como a comunicação tem de ser num outro canal e convém ser em paralelo com outras operações, decidimos criar thread para esta operação.

Já no servidor secundário, criamos a thread para receber eventuais ficheiros para operar em simultâneo com a resposta aos heartbits feita sem thread.

Quanto à parte do código do cliente, o programa começa e corre segundo as opções que o cliente escolhe.

Funcionamento do servidor ucDriver

Tal como já referimos na secção anterior, cada servidor está preparado para ser primário ou secundário. Quando se inicia cada um, estes leem as suas configurações de um ficheiro de texto (o seu endereço, porto TCP para comunicar com os clientes, porto UDP para comunicar entre si, porto UDP para troca de ficheiros, o endereço absoluto para as pastas do servidor, o número máximo possível de pings falhados e o tempo entre cada ping). Tendo isto, o servidor a ser ligado vai tentar criar uma conexão UDP no porto especificado para comunicar com o outro servidor. Se não conseguir, é sinal que o outro se conectou primeiro sendo primário. Deste modo, o que se estava a tentar conectar passa a ser secundário.

O servidor secundário comunica com o primário para duas questões: para receber resposta aos heartbeats e para receber os ficheiros carregados para o primário ou o de clientes caso exista alguma alteração. A parte dos heartbeats é feita através da tal conexão UDP que tenta criar ao iniciar. Quanto a isto, vai enviá-los para o servidor primário à espera de resposta. Se não receber X respostas seguidas (X sendo o número definido nas configurações), passa a ser o primário.

Na parte em que recebe ficheiros, o servidor secundário começa por receber o endereço absoluto em que o primário colocou o ficheiro em questão. Como o que varia nos endereços é pouco (no caso de ser um ficheiro que foi carregado para o servidor muda apenas uma pasta, em vez de ser “server” passa a ser “server2”, ou vice-versa, e no caso de ser o ficheiro que tem as informações dos clientes muda apenas o nome, “clients1.txt” ou “clients2.txt”), o servidor secundário calcula logo o endereço absoluto no qual tem de colocar o ficheiro. De seguida, vai receber o tamanho do mesmo, uma vez que necessita de saber quando deve parar de estar à espera de informação.

Posto isto, o servidor entra num ciclo enquanto o tamanho que falta receber for superior a 0. Sendo que enviamos o ficheiro em pacotes de 1KB via UDP, convém saber se a informação chegou ao destino. Deste modo, caso a espera pelo pacote não ultrapasse o tempo limite definido, o servidor secundário envia uma mensagem afirmativa ao servidor primário em como recebeu o pacote. Se assim for, o primário envia o próximo. Caso o tempo de espera seja ultrapassado, o servidor secundário envia uma mensagem indicativa de que não recebeu e o primário reenvia o mesmo pacote.

Já o servidor primário é mais complexo. Após criar o socket UDP para comunicar com o servidor secundário, vai ler as informações dos clientes do ficheiro .txt correspondente e guardá-las num ArrayList de clientes (classe definida para conter todas as informações relativas aos clientes). Após isto, cria o socket de comunicação com os clientes e a thread que servirá para responder aos heartbits vindos do servidor secundário. De seguida, ficará num ciclo infinito à espera de conexões de clientes. Ao receber uma, cria a thread responsável por tratar de cada utilizador.

Essa thread vai começar por receber o username e a password do cliente. Como guardou ao início as informações dos utilizadores no ArrayList, basta procurar lá se existe algum objeto que tenha os atributos username e password iguais. Se sim, o servidor pode passar à próxima fase de aguardar pelas opções do cliente visto que este já se autenticou. Se não, volta a ficar à espera de receber o username e a password para validar.

Tendo o cliente autenticado, o servidor fica à espera de receber a opção que o utilizador vai escolher do menu. Mediante a opção escolhida, o servidor reage de maneira diferente. Se o cliente escolher alterar a sua palavra-passe, o servidor ficará à espera de receber a palavra-chave atual do cliente como método de segurança. Se coincidir com a já existente, comunica-lhe que pode prosseguir e fica à espera de receber a nova password. Ao receber, define-a como sendo a nova desse utilizador e chama o método update_clients para atualizar o ficheiro com as configurações dos clientes (que vai atualizar também o do outro servidor).

Caso o servidor receba a informação de que o cliente pretende visualizar os ficheiros que tem na sua diretoria atual no servidor, vai percorrê-los e enviar para o cliente.

Já se o cliente pretender mudar de diretoria no servidor, este vai fazer algumas verificações e só depois dizer ao utilizador o que pode fazer. Por exemplo, estando numa diretoria, o servidor vai ver quantas estão dentro da atual. Se não existir nenhuma, o servidor já não indica ao cliente que pode "andar em frente" e entrar numa diretoria. De igual modo, se o seu endereço atual no servidor for igual à junção do endereço da pasta onde estão as diretorias de todos os clientes no servidor e o nome de utilizador, significa que não pode "andar para trás" no endereço. Se esta condição se verificar, o servidor indica ao utilizador que não o pode fazer. Mediante as combinações destas situações, o servidor indica ao cliente que ações pode realizar e fica à espera da decisão do mesmo. Após receber indicação do cliente, faz o que este pretende, atualizando no fim o ficheiro dos mesmos uma vez que esta é uma propriedade que devemos guardar para que o cliente se ligue sempre na última diretoria acedida.

É importante realçar que sempre que acedemos a esse ficheiro com as informações dos clientes, recorremos a uma espécie de semáforo para evitar que vários clientes tentem modificá-lo em simultâneo.

No caso de o cliente pretender carregar ou descarregar um ficheiro do servidor, como esta operação deve ser realizada noutra ligação, criamos um novo socket TCP com o "porto utilizado para a passagem de comandos + 1" e chamamos a thread responsável por estas ações.

Nesta thread, caso a operação pretendida seja a de descarregar um ficheiro, o servidor vai verificar se existe algum na diretoria do cliente no servidor. Se existir, vai enviar-lhe informação de quais são, ficando a aguardar pela escolha do utilizador. Se este acabar por escolher um (recusar é com a opção 0), o servidor manda ao cliente o nome do ficheiro em questão e de seguida envia o pretendido em blocos de 1KB.

Caso o cliente queira carregar um ficheiro, o servidor vai receber o nome deste, calcular a partir disso o endereço para qual o ficheiro deve ir e começar a recebê-lo em blocos de 1KB. Após isto, ainda falta replicar no servidor secundário. Para isso, vai abrir um socket UDP no porto correspondente e enviar o necessário para o outro servidor.

Detalhes do mecanismo de failover

Uma vez que pretendíamos ter consistência na nossa plataforma, criámos um mecanismo de failover que nos permite ter mais garantias de que o serviço não falha.

Para detetar que existe algum problema, tal como já mencionámos anteriormente, o servidor secundário comunica com o primário enviando heartbeats e fica à espera de resposta. Caso não receba X respostas seguidas (X sendo o número definido nas configurações do servidor), o secundário passa a ser o primário. Caso o anteriormente primário se volte a ligar, como o socket UDP para a comunicação entre servidores já está a ser utilizado, conecta-se como secundário. Sendo que temos este mecanismo dentro de um ciclo infinito, garantimos sempre que caso o primário falhe, não existe perda de ligação pois o secundário assume o papel principal.

Sempre que ocorre mudança nos papéis dos servidores, o cliente ao reparar que já não se consegue conectar, passa automaticamente a tentar conectar-se ao outro (que em princípio assumiu o papel de primário).

Sendo que existem dois servidores, convém que a informação seja a mesma em ambos. Por exemplo, se fizermos o upload de um ficheiro para o servidor Y (que está como primário num dado momento) e, por alguma razão quisermos aceder a esse ficheiro quando o primário é o servidor Z, convém que esse ficheiro lá esteja. Deste modo, sempre que fazemos um upload de um ficheiro ou alteramos o ficheiro com as informações dos clientes, mandamos via UDP esses dados para o servidor secundário, bem como o endereço absoluto em que deve colocar tais ficheiros. Como o endereço é igual, mudando apenas o nome de uma pasta, é fácil de o servidor secundário obter, a partir daí, a diretoria em que deve colocar o ficheiro em causa.

Seria importante também retomar operações não concluídas aquando da falha de conexão. No entanto, não chegámos a implementar esta funcionalidade, ficando em mente para trabalho futuro e de melhoria da plataforma.

Distribuição de tarefas por membros do grupo

Sendo este um trabalho que convém que se vá fazendo em conjunto para se saber o que cada comunicação precisa, decidimos programar juntos. Deste modo, sempre que um de nós estava a programar, o outro elemento estava junto a ajudar e a discutir qual a melhor solução a adotar.

O único momento em que dividimos tarefas foi na parte final em que enquanto um foi elaborando o relatório, o outro foi testando exaustivamente a plataforma, tendo sido essa divisão de tarefas assim:

- Relatório: Duarte Meneses
- Testes à plataforma: Patrícia Costa

Testes realizados à plataforma

- Aplicação do cliente

	Autenticação do utilizador no terminal de cliente
Resultado esperado	Deve ser possível a autenticação de um utilizador, através do terminal. Após a colocação do respetivo nome e password, o servidor deve situar o cliente na sua diretoria *Home* ou na diretoria da última sessão caso este já se tenha autenticado no sistema no passado.
Resultado obtido	É pedido ao utilizador a colocação do seu nome e password, passando por uma verificação de que tais informações existem e estão corretas. Após a autenticação, o servidor redireciona o cliente para a sua diretoria *Home*, caso seja a primeira vez que se autentica, ou na diretoria da última sessão caso este já se tenha autenticado no sistema no passado.
Veredito	Aprovado

	Alterar a password do utilizador
Resultado esperado	Para que o utilizador mude a sua password deve, inicialmente, autenticar-se através do terminal do cliente. Após a alteração o terminal deve desconectar-se e pedir novamente a autenticação do cliente de modo a manter a segurança na alteração da password.
Resultado obtido	Após a escolha da opção "1 - Change user password", para mudar, o cliente deve verificar a sua password antiga e só depois colocar a nova. Após a alteração o terminal desconecta-se e é pedido novamente a autenticação ao cliente.
Veredito	Aprovado

	Configurar endereços e portos de servidores primário e secundário
Resultado esperado	Na aplicação cliente deve ser possível configurar os endereços e portos dos servidores primário e secundário.
Resultado obtido	Antes da autenticação, o cliente pode escolher ou não configurar os portos e endereços com que se vai tentar ligar a um dos servidores. Caso escolha algo

	que não exista, após uma espera de poucos segundos ocorre um erro de conexão recusada.
Veredito	Ambíguo (muda os portos e endereços, mas se escolher algo que não existe gera uma exceção)

	Listar os ficheiros que existem na diretoria atual do servidor
Resultado esperado	O cliente deve conseguir listar os ficheiros existentes na diretoria atual do servidor.
Resultado obtido	Após a escolha da opção "2 - List files at the current directory of the server" é apresentado, no terminal do cliente, os ficheiros presentes na diretoria atual do servidor.
Veredito	Aprovado

	Listar os ficheiros que existem na diretoria atual do servidor sem que haja ficheiros
Resultado esperado	Não deve aparecer nada.
Resultado obtido	Após a escolha da opção "2 - List files at the current directory of the server" não é listado nada.
Veredito	Aprovado

	Mudar a diretoria atual do servidor
Resultado esperado	O utilizador deve conseguir navegar entre as várias diretorias a que tem acesso, sem conseguir mudar para diretorias fora da sua *Home*.
Resultado obtido	No terminal, é apresentado todas as opções para onde pode navegar. Poderá escolher mudar para cada uma das suas pastas, não mudar de diretoria ou voltar para trás na pasta sem que seja possível alterar a ponto de sair da sua *Home*.
Veredito	Aprovado

	Listar os ficheiros que existem na diretoria atual do cliente
Resultado esperado	Os utilizadores que estejam a aceder ao sistema através de um terminal devem conseguir listar os ficheiros existentes na diretoria corrente do cliente.
Resultado obtido	Após a escolha da opção "4 - List files at the current local directory" são apresentados, no terminal do cliente, os ficheiros presentes na diretoria atual do cliente.
Veredito	Aprovado

	Listar os ficheiros que existem na diretoria atual do servidor sem que haja ficheiros
Resultado esperado	Não deve aparecer nada.
Resultado obtido	Não é listado nada.
Veredito	Aprovado

	Mudar a diretoria atual do cliente
Resultado esperado	O utilizador deve conseguir mudar a diretoria local para que seja possível navegar entre as várias diretorias existentes no sistema local.

Resultado obtido	No terminal, é apresentado todas as opções para onde pode navegar. Poderá escolher mudar para cada uma das suas pastas, não mudar de diretoria ou voltar para trás na pasta.
Veredito	Aprovado

	Descarregar um ficheiro do servidor
Resultado esperado	O utilizador para que possa descarregar um ficheiro deve estar autenticado. Ao escolher o que descarregar deve ficar com uma cópia do ficheiro armazenado localmente.
Resultado obtido	O utilizador ao estar autenticado, poderá escolher o ficheiro que quer descarregar. Após a escolha, é feita uma cópia do ficheiro armazenado localmente.
Veredito	Aprovado.

	Descarregar um ficheiro do servidor de uma diretoria sem ficheiro
Resultado esperado	Deve aparecer uma mensagem ao utilizador a dizer que não existem ficheiros para descarregar.
Resultado obtido	Aparece uma mensagem ao utilizador a dizer que não existem ficheiros para descarregar.
Veredito	Aprovado.

	Carregar um ficheiro para o servidor
Resultado esperado	O utilizador para que possa carregar um ficheiro deve estar autenticado. Ao escolher o que carregar para o servidor deve ficar com uma cópia do ficheiro armazenado na diretoria.
Resultado obtido	O utilizador ao estar autenticado, poderá escolher o ficheiro que quer carregar. Após a escolha, é feita uma cópia do ficheiro armazenado na diretoria.
Veredito	Aprovado

	Carregar um ficheiro para o servidor de uma diretoria sem ficheiro
Resultado esperado	Deve aparecer uma mensagem ao utilizador a dizer que não existem ficheiros para carregar.
Resultado obtido	Aparece uma mensagem ao utilizador a dizer que não existem ficheiros para carregar.
Veredito	Aprovado

- Atributos de qualidade

	Tratamento de exceções
Resultado esperado	Caso o hardware falhe, é preciso garantir que os ficheiros não se perdem, nem aparecem duplicados. É preciso que do lado do cliente nenhuma operação fique a meio e que sejam retomadas as operações no mesmo estado em que se encontravam antes da falha.
Resultado obtido	Caso o hardware falhe, as operações também falham. Se porventura estiver a ser feito um carregamento, este é interrompido e é necessário voltar a carregá-lo novamente para que o cliente tenha o pretendido. O mesmo ocorre para o descarregar de ficheiros.
Veredito	Reprovado

	Failover - Heartbeat e deteção de falha no servidor primário
Resultado esperado	O servidor secundário deve trocar periodicamente, via UDP, pings ou heartbeats com o servidor primário. Se algumas destas mensagens forem perdidas, um certo número de vezes, o secundário deve assumir que o primário avariou e passar a exercer essa função.
Resultado obtido	O servidor secundário troca periodicamente, via UDP, pings ou heartbeats com o servidor primário. Após um certo número de mensagens perdidas, o secundário assume a função de primário.
Veredito	Aprovado

	Failover - Ficheiros carregados e descarregados para o primário são copiados para o secundário
Resultado esperado	Ambos os servidores devem conter os mesmos ficheiros para que o servidor de backup esteja pronto a substituir o outro em caso de falha. Ou seja, sempre que se carregar um ficheiro, o servidor primário deve enviar este ficheiro para o servidor secundário, via UDP, para que ambos tenham a mesma informação.
Resultado obtido	Ambos os servidores contêm os mesmos ficheiros para que o servidor de backup esteja pronto a substituir o outro em caso de falha. Quando é necessário carregar um ficheiro, o servidor primário envia-o para o servidor secundário, via UDP, para que ambos tenham a mesma informação. É feita uma cópia que é enviada ao servidor secundário para que coloque na mesma diretoria em que se encontrava no primário.
Veredito	Aprovado

	Failover - Clientes conseguem operar com o secundário quando o primário está em baixo
Resultado esperado	Quando o secundário assume que o primário avariou, passa a exercer essa função. Caso o antigo primário recupere, este passa a assumir a função de secundário. O servidor com o papel de secundário, enquanto o primário está ativo, não aceita ligações dos clientes. O cliente opera com aquele que assumir a função de primário.
Resultado obtido	Quando o secundário assume que o primário avariou, passa a exercer essa função. No momento em que o antigo primário recupera, assume a função de secundário. O servidor com o papel de secundário, enquanto o primário está ativo, não aceita ligações dos clientes. O cliente opera com aquele que assumir a função de primário.
Veredito	Aprovado

	Failover – Alterar a password
Resultado esperado	Caso o cliente mude a password, ambos os servidores devem conter a informação da alteração. Deste modo, o primário deve enviar a informação para o secundário, via UDP, para que em caso de falha, o cliente possa autenticar-se com a nova password no secundário (que teria assumido a função de primário)
Resultado obtido	Caso o cliente mude a password, ambos os servidores contêm a informação da alteração. O primário envia a informação para o secundário, via UDP, para

	que em caso de falha, o cliente possa autenticar-se com a nova password no secundário (que teria assumido a função de primário)
Veredito	Aprovado

Foram testadas as mesmas funcionalidades para cada um dos servidores enquanto assumem função de primário e secundário, assim como a falha de cada um deles como primários. Ambos tiveram os resultados demonstratados nas tabelas anteriores.

Foi verificado também o que poderia acontecer caso vários clientes se ligassem em simultâneo a estes servidores. Os resultados foram igualmente positivos e ambos os utilizadores conseguiram obter o que pretendiam. Os clientes foram capazes de realizar as operações desejadas, mostrando assim que a nossa plataforma é capaz de lidar com vários clientes ligados simultaneamente.

Conclusão

Era-nos pedido para desenvolver uma plataforma capaz de replicar um sistema de armazenamento de ficheiros em servidores acedidos por alunos e professores.

De um modo geral, pensamos ter cumprido bem o objetivo embora nem tudo o que era pedido tenha sido implementado. Devíamos ainda ter garantido mais fiabilidade ao evitar que operações se perdessem caso alguma ligação falhasse. Este é sem dúvida um ponto a ter em conta em trabalho futuro para melhorar a plataforma.

No entanto, e tal como podemos ver pelos testes realizados à plataforma, na grande maioria destes a nossa plataforma teve resultados positivos, pelo que podemos concluir que acabámos por cumprir a maioria do que era requerido.

Referências

- Material fornecido pelos docentes
- https://www.geeksforgeeks.org/path-tostring-method-in-java-with-examples/ [acedido em 26/03/2022]
- https://reactgo.com/java-convert-string-to-path/ [acedido em 26/03/2022]
- https://docs.oracle.com/javase/tutorial/essential/io/dirs.html [acedido em 26/03/2022]
- https://stackoverflow.com/questions/30249324/how-to-get-java-to-wait-for-user-input [acedido em 26/03/2022]
- http://www.beginwithjava.com/java/file-input-output/writing-text-file.html [acedido em 26/03/2022]
- https://stackoverflow.com/questions/7209110/java-util-nosuchelementexception-no-line-found [acedido em 26/03/2022]
- https://www.geeksforgeeks.org/compare-two-strings-in-java/ [acedido em 26/03/2022]
- https://www.tutorialspoint.com/how-to-search-for-a-string-in-an-arraylist-in-java [acedido em 26/03/2022]
- https://www.w3schools.com/java/java_arraylist.asp [acedido em 26/03/2022]
- https://attacomsian.com/blog/how-to-read-a-file-line-by-line-in-java [acedido em 26/03/2022]
- https://stackoverflow.com/questions/5125242/java-list-only-subdirectories-from-a-directory-not-files [acedido em 27/03/2022]
- https://stackoverflow.com/questions/4871051/how-to-get-the-current-working-directory-in-java [acedido em 27/03/2022]
- https://www.yawintutor.com/java-util-regex-patternsyntaxexception-unexpected-internal-error-near-index-1/ [acedido em 27/03/2022]
- https://www.geeksforgeeks.org/file-isfile-method-in-java-with-examples/ [acedido em 28/03/2022]
- https://srikarthiks.files.wordpress.com/2019/07/file-transfer-using-tcp.pdf [acedido em 28/03/2022]
- https://heptadecane.medium.com/file-transfer-via-java-sockets-e8d4f30703a5 [acedido em 28/03/2022]
- https://www.geeksforgeeks.org/returning-multiple-values-in-java/ [acedido em 28/03/2022]
- https://gist.github.com/absalomhr/ce11c2e43df517b2571b1dfc9bc9b487 [acedido em 30/03/2022]
- http://underpop.online.fr/j/java/help/datagrampacket-class-network-dev-java-programming-language.html [acedido em 30/03/2022]
- https://stackoverflow.com/questions/8721262/how-to-get-file-size-in-java [acedido em 30/03/2022]
- http://bethecoder.com/applications/articles/java/basics/how-to-convert-long-to-byte-array.html [acedido em 30/03/2022]
- https://stackoverflow.com/questions/8721262/how-to-get-file-size-in-java [acedido em 30/03/2022]
- https://www.geeksforgeeks.org/java-program-to-convert-file-to-a-byte-array/ [acedido em 31/03/2022]
- https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html [acedido em 01/04/2022]

- https://stackoverflow.com/questions/12089967/find-difference-between-two-strings [acedido em 01/04/2022]