



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE
COIMBRA

Projeto de Sistemas Distribuídos

Plataforma ucDrive

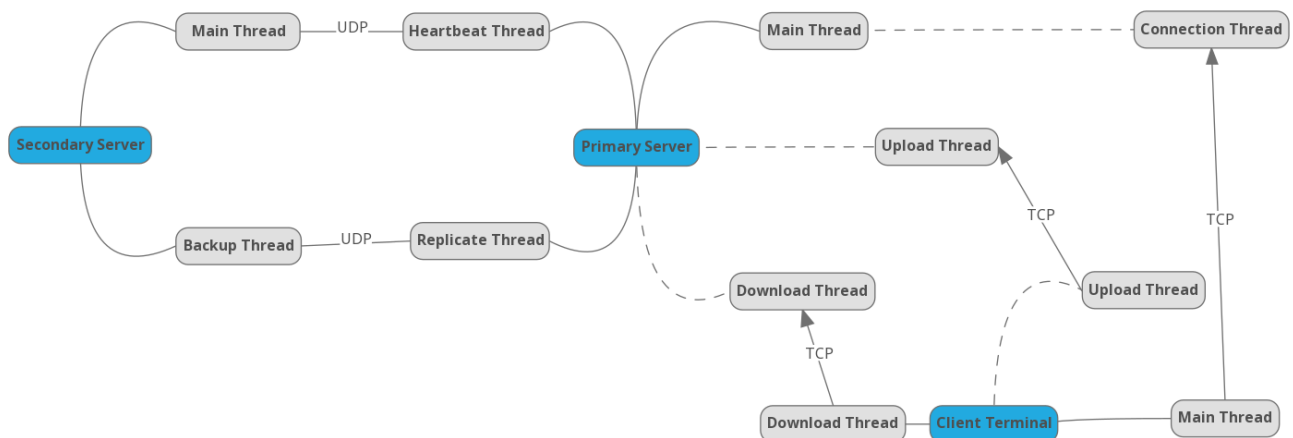
Inês Martins Marçal 2019215917

Noémia Quintano Mora Gonçalves 2019219433

Índice

Descrição da arquitetura de software:	3
Legenda da imagem representativa da arquitetura ucDrive:	3
Threads.....	3
Sockets	3
Server Side.....	3
Client Side.....	4
Organização do código:.....	4
Decisões tomadas na implementação:	4
Condições de funcionamento e limitações do programa	5
Funcionamento do servidor ucDriver:	5
Métodos implementados nos ficheiros Client.java e Server.java	5
Configurar os portos e hosts do servidor primário e secundário	5
Autenticação do cliente	6
Alterar a password de utilizador	6
Listar o conteúdo da diretoria atual do cliente e a deste no servidor	6
Mudar a diretoria atual local do cliente e a deste no servidor	6
Descarregar do servidor	7
Carregar um ficheiro para o servidor	7
Failover	8
Mecanismo de HeatBeat implementado	8
Mecanismo de replicação implementado.....	8
Testes efetuados à plataforma	9
Testes de autenticação	9
Testes de configuração	9
Testes de listagem de ficheiros e alteração da diretoria atual tanto no servidor como no cliente.....	10
Testes às funcionalidades <i>Upload</i> e <i>Download</i>	10
Testes do mecanismo de HeartBeats.....	11
Testes do mecanismo de Replicação	12
Multiplos Clientes.....	12
Divisão de tarefas.....	12

Descrição da arquitetura de software:



Legenda da imagem representativa da arquitetura ucDrive:

- As ligações contínuas indicam as *threads* permanentes;
- As ligações a tracejado indicam as *threads* que surgem conforme os pedidos dos clientes e após a conclusão da ação pretendida pelos mesmos são eliminadas.

A comunicação é iniciada quando ocorre a conexão do cliente ao servidor primário através de TCP (conexão entre *Main Thread* e *Connection Thread* representada na figura acima). Caso sejam solicitados um upload ou download é criada uma *thread* para a respetiva operação (*Upload Thread* ou *Download Thread*), que comunica através de TCP com as respetivas *threads* do cliente. Paralelamente, o *Secondary Server* comunica com o *Primary Server* enviando *heartBeats*, que serão devolvidos por este último de modo a sinalizar a sua presença. Por sua vez, a *Thread Backup* espera que a *Thread Replicate* detete alterações e as comunique através do UDP.

Threads

O *Primary Server* ao iniciar cria as *Threads Heartbeat e Replicate*, esperando assim que os clientes se conectem. Conforme estes vão estabelecendo a ligação com o mesmo vai sendo criada uma *thread* para efetuar a comunicação com cada um deles. Posteriormente, quando algum destes solicita um upload a *Thread Upload* é iniciada e caso este seja interrompido ou concluído com sucesso a *Thread* irá ser eliminada. Quanto à *Replicate Thread*, esta verifica de 1 em 1 minuto se ocorreu algum upload que modificou o estado dos ficheiros no server, reencaminhando essa informação da alteração para a *Thread BackUp*. Por sua vez, a receção, por parte desta, da informação dos ficheiros que foram atualizados irá permitir que o *Secondary Server* contenha exatamente os ficheiros com o mesmo conteúdo em relação aos que se encontram no *Primary Server*. A *MainThread* do *Secondary Server* envia *heartBeats* para a *HeartBeat Thread* do *Primary Server*, permitindo que o *Secondary* tome conhecimento de quando tem de substituir o *Primary*.

Sockets

Server Side

Os *sockets* TCP do *Primary Server* e os *sockets* UDP da *HeartBeat Thread* e do *Secondary Server* (*thread de Backup*) são definidos e posteriormente lidos de um ficheiro de configurações quando o server inicia, mantendo-se posteriormente durante a execução.

De forma dinâmica são atribuídos pelo sistema operativo *sockets* TCP às *Threads de Download e upload* dos diferentes clientes, uma vez que estas *threads* assim que o *download ou upload* é concluído irão ser eliminadas. Além destes referidos anteriormente, também é atribuído dinamicamente o *DatagramSocket* do *Secondary Server* enquanto envia *heartbeats* para a *HeartBeat Thread* do *Primary*.

As *Connection Threads* obtêm os seus portos através do *accept* do *socket* TCP do *Primary Server*.

Client Side

No *client* é possível configurar os *sockets* que permitem vir a efetuar uma conexão com o servidor. Quando é requisitado um download ou upload, o sistema operativo atribui portos disponíveis que posteriormente irão ser utilizados pelas *threads* para se conectarem a um *server socket* durante a sua operação.

Organização do código:

- Foram criados dois ficheiros .java, um para o cliente (Client.java) e outro para o servidor (Server.java).
- Foi ainda criado a classe User.java no sentido de representar os *users*.
- No ficheiro Client.java foi criada uma *class* Client.java onde se encontram os métodos que permitem implementar as funcionalidades do ucDrive assim como a *class* Download e Upload. Estas duas últimas representam as *threads* definidas para efetuar a funcionalidade download de ficheiros do servidor por parte do cliente e upload de ficheiros para o servidor. No main da *class* Client é implementado o mecanismo que permite que o cliente se conecte ao servidor e aguarde comandos. Aqui também é definido os comandos, explicados mais à frente, que irão permitir ao cliente efetuar as operações pretendidas, sendo que cada comando tem uma função associada.
- No ficheiro Server.java foi definido a *class* Server onde o *socket* do servidor espera por uma conexão por parte do cliente. Ao mesmo tempo são criadas as *Threads* HeartBeat, Replicate e Backup (cujo o mecanismo de funcionamento será explicado mais adiante). Relativamente à criação de cada uma destas *threads* procedeu-se à definição de uma classe para cada uma. Tal como no Client.java também foi implementado uma *class* Download e uma *class* Upload. Neste ficheiro foi ainda criado a *class* Connection que irá permitir ao cliente estabelecer ligações com o servidor em causa. Por outro lado, é nesta *class* que também são implementadas a maior parte das funcionalidades que irão permitir a realização das operações pretendidas pelo cliente. Se o Server não se conseguir conectar em modo primário irá assumir o cargo de secundário no main da *class* Server e procede ao envio de *HeartBeats* para prevenir total *Failover* do serviço.

Decisões tomadas na implementação:

- Na diretoria do servidor encontra-se um ficheiro users.txt com os dados dos clientes, onde são colocados em primeiro lugar os mais utilizados e modificados ao longo da implementação, mais especificamente o *username*, *password* e ainda o número de cartão de cidadão. Cada linha deste ficheiro encontra-se estruturada da seguinte forma:
[userID];[password];[ccNumber];[nome];[CCValidade];[morada];[departamento];[telefone];[diretoriaAtual]
- Para uma melhor gestão dos dados dos clientes foi criado um número de identificação único (primeiro de cada linha do ficheiro) que irá ser utilizado como *username* na autenticação e assim como identificador da respetiva pasta do cliente no server.
- Procedeu-se à criação de uma diretoria Data no lado do servidor onde se encontram as pastas de cada cliente identificadas pelo *username*, tal como referido acima.
- Cada cliente tem uma pasta *home* (definida como diretoria *default*) e encontra-se limitado a ela. Nesta encontram-se os ficheiros sobre os quais é possível efetuar o *download*.
- Foi criado um ficheiro log.txt no server, onde são colocadas as diretorias dos ficheiros que sofreram alterações, para posteriormente serem assim replicados no *Secondary Server*. Cada linha deste ficheiro encontra-se estruturada da seguinte forma:
[primaryPort];[secondaryPort];[heartbeatPort]
- Foi criado um ficheiro config.txt no qual se encontram os portos do *Primary Server*, os do *Secondary* e ainda o do *heartBeat*.
- Foi definido que o cliente terá de realizar uma nova autenticação se proceder à alteração da password.

- Se o *Secondary Server* assumir o cargo de *Primary Server*, o cliente terá de realizar uma nova autenticação para poder realizar as ações pretendidas.

-O *boolean primaryConection* na *class* do *Client* indica se o cliente está conectado ao *Primary* or *Secondary Server*, para apenas se atualizar a conexão ao qual o cliente está ligado atualmente.

Condições de funcionamento e limitações do programa

O server tem de ser inicializado numa pasta que contenha os ficheiros: *config.txt* e *users.txt* uma vez que é nesta que se encontram as configurações de arranque e os dados dos utilizadores. A diretoria "*Data*" também deverá encontrar-se criada e com as respetivas pastas dos utilizadores também já criadas, uma vez que o server irá arrancar assumindo já que tudo se encontra em conformidade.

Assume-se ainda que o utilizador apenas tem permissões sobre a sua pasta do *server* e que o *upload* de ficheiros locais está limitado a ficheiros colocados na diretoria "*home*", previamente criada.

O *download* e *upload* apenas pode ser efetuado sobre ficheiros e não sobre diretorias.

O projeto plataforma ucDrive foi realizado usando java 17.

Funcionamento do servidor ucDriver:

No sentido de tornar a utilização da plataforma ucDrive mais realista foi escolhido estabelecer comandos que permitissem vir a realizar as operações pretendidas pelo cliente. Por outras palavras, criou-se uma interface baseada num aspeto de um terminal em que em cada linha do mesmo é indicado o *username* do utilizador a que este pertence e a diretoria em que este se encontra localmente. Deste modo, o cliente após estabelecer a conexão TCP com o servidor e efetuar a autenticação, explicada mais à frente, poderá digitar os seguintes comandos:

- **cp [oldpassword] [newpassword] [newpassword]**--> o cliente pode modificar a sua password para a nova que pretende;
- **config [primary/secondary] [host] [port]**--> permite configurar endereços e portos do cliente que permitem a sua conexão ao servidor;
- **ls [server/local]** --> permite listar o conteúdo da diretoria em que o cliente se encontra localmente ou no servidor, de acordo com o segundo argumento utilizado no comando;
- **cd [server/local] [directory]** --> permite mudar a diretoria em que o cliente se encontra localmente ou no server.

Para além dos comandos de configuração acima indicados, é ainda dado a opção ao cliente através do **comando sh** de poder visualizar as configurações efetuadas. Por outro lado, no sentido de o servidor perceber de que tipo de ação se trata, à frente de cada um destes comandos enviados para o mesmo é indicado o número do *case* implementado no *switch* para cada um.

Métodos implementados nos ficheiros Client.java e Server.java

Configurar os portos e hosts do servidor primário e secundário

O cliente inicialmente no main tenta se conectar ao *Primary Server*, no entanto se esta conexão falhar o mesmo tentará se conectar ao *Secondary Server*. Por sua vez se não for permitido a conexão a nenhum destes é dado a possibilidade **ao cliente de configurar os portos e o host com que este se conecta ao servidor primário e secundário ou sair através do comando "q" do programa**. Esta funcionalidade é implementada pelo **método configureHostPorts()** através do comando indicado acima – **config [primary/secondary] [host] [port]**. Este por sua vez ao receber os dados passados por parâmetro através do *data[]* verifica se o porto é válido. Foi implementado um *switch*, neste método ainda, para que fosse possível o cliente escolher qual dos dois pretendia configurar.

Autenticação do cliente

Posteriormente, após a criação do *socket*, antes de ser permitida qualquer ação é pedida a **autenticação do cliente**. Este método no lado do cliente recebe como parâmetros o *DataInputStream* in e o *DataOutputStream* que permite a comunicação entre o servidor e o cliente. Nele é pedido ao utilizador o *username* e a *password* que se encontram no ficheiro de dados *users.txt*. De seguida, estes dados são enviados para o *server* através do *DataOutputStream* na seguinte estrutura: "0;" + *studentId* + ";" + *password*, sendo o "0" o indicador de que case/comando se trata, permitindo assim que o servidor saber a ação em causa. Com o objetivo de se verificar a veracidade da informação introduzida pelo cliente na linha de comandos relativamente ao ficheiro *users.txt* foi implementado o mesmo **método authentication()** no servidor. Este por sua vez recebe como parâmetros o *username* e a *password* provenientes do cliente. Além de ser efetuado neste método a verificação dos dados é criado o *user* com base nos dados presentes no ficheiro *users.txt*. Foi também decidido definir uma variável *ServerDirectory* que permite guardar a diretoria a que o cliente se encontra (esta é atualizada no ficheiro *users.txt* sempre que a conexão do cliente é fechada, sendo substituída pela última que foi acedida), no sentido de ser possível sempre que o cliente efetua login continuar na diretoria em que se encontrava anteriormente localmente antes de se ter desconectado. Neste método foi também criado uma variável contadora que permite guardar a linha do ficheiro onde os dados do *user* se encontram, facilitando o futuro acesso aos mesmos. No momento de criação do *user* na autenticação no servidor é verificado se o cliente apresenta mais algum parâmetro que o suposto no ficheiro *users.txt*. Caso isto aconteça, significa que um download ou upload não ocorreu com sucesso e deste modo é notificado o respetivo cliente que o tentou efetuar que o download/upload do insucesso da operação. Esta função consoante o resultado da ação autenticação efetua o envio de uma mensagem "OK" (quando se autenticou com sucesso), "PENDING" (quando se autenticou com sucesso, mas ficou alguma operação upload/download pendente) ou "Invalid username/password" (quando houve algum problema na autenticação) ao cliente.

Alterar a password de utilizador

Relativamente ao **método changePassword()** no lado do cliente, este irá permitir que os clientes modifiquem a sua password utilizada na autenticação. Esta recebe como parâmetros os dados introduzidos pelo cliente (*String data[]*), assim como o *socket*, o *DataInputStream* e o *DataOutputStream*. Através do comando – cp [oldpassword] [newpassword] [confirmação new password] é verificado em primeiro lugar se a password nova introduzida e a sua confirmação são iguais. Perante estes dados é enviado ao servidor através do *DataOutputStream* uma mensagem com a seguinte estrutura: "1" + ";" + old password + ";" + newpassword. Deste modo, **no lado do servidor** foi implementado também um **método changePassword()** que recebe como parâmetros os dados provenientes do comando digitado pelo cliente, referido anteriormente, mais especificamente o *username* e a *password*. Neste é efetuado a validação da password antiga perante a que se encontra no ficheiro de dados *users.txt*. E só posteriormente, depois de esta validada, é que é atualizada a password no ficheiro para a nova escolhida pelo cliente. A substituição pela password nova é facilitada através do *contador* criado anteriormente (e explicado no tópico acima) que permite saber exatamente onde este cliente se encontra no ficheiro de dados *users.txt*. Após a substituição da mesma é pedido novamente a autenticação ao cliente.

Listar o conteúdo da diretoria atual do cliente e a deste no servidor

Mudar a diretoria atual local do cliente e a deste no servidor

No sentido de ser implementada a funcionalidade de **listar e mudar de diretoria** foi implementada a função **ExistDirectory()**, tanto no *Client.java* como no *User.java*. Este método recebe como parâmetros a diretoria em que o cliente se encontra atualmente, no *server* ou localmente, e a nova diretoria para a qual se pretende ir. Assim como o método referido anteriormente, o **verifyDirectory()** também é implementado tanto no *Client.java* como no

Server.java. Este método recebe como parâmetro a diretoria para a qual o cliente pretende se mudar dentro da sua pasta *home*. Caso a opção do cliente seja recuar, este apenas o poderá fazer até à sua diretoria *home*. Para que isto aconteça é contabilizado a última ocorrência do "/" no *path* de diretoria. Por outro lado, se o objetivo do cliente é avançar para uma nova diretoria, é efetuada a verificação da existência da mesma (utilizando o método **ExistDirectory()**, referido anteriormente) e perante a sua validação procede-se à atualização da diretoria variável definida. **No caso do servidor, ainda é procedida a sua atualização no ficheiro users.txt para que seja possível a manutenção de estado dos clientes no servidor.** Tanto no cliente como no servidor em caso de a operação ter sido bem-sucedida é efetuado um print informando o sucesso da mensagem, caso contrário é informado que não foi possível efetuar a mudança de diretoria. Por outro lado, o método **listLocalDirectory()** também foi implementado tanto no cliente como no servidor, no sentido de ser implementada a funcionalidade de listar o conteúdo da diretoria em que o cliente se encontra localmente ou no servidor, respetivamente. Nestes, além de ser verificado, inicialmente, se a diretoria guardada numa variável, que representa a diretoria atual no servidor ou cliente, é ainda efetuada a **distinção entre um ficheiro e uma diretoria** para se compreender melhor de que tipo de ficheiro se trata no momento da visualização do conteúdo destes.

Descarregar do servidor

Como objetivo de se proceder à implementação da funcionalidade de download foi criada a **class Download**, tanto no cliente como no servidor. No caso do cliente, este método recebe como parâmetros o comando (*download [file]*) introduzido pelo cliente assim como o *socket*, o *DataInputStream* e o *DataOutputStream*. O ficheiro, ao qual se pretende efetuar o download, é enviado para o servidor através de uma mensagem com a seguinte estrutura: "7" + ";" + file, sendo que o "7" permite indicar ao server de que tipo de ação se trata. **Assim no lado do servidor**, este método ao receber como parâmetro o ficheiro que o cliente pretende efetuar download irá permitir que seja feita a verificação da existência do mesmo na diretoria do server em que o cliente se encontra, antes de se realizar qualquer ação. Depois desta confirmação e caso a mesma se verifique é criado um *socket* com o porto 0 (porto definido pelo sistema operativo). Posteriormente, é enviado ao cliente a mensagem "OK" não só com o porto definido pelo SO (que pode ser visualizado pela função *getLocalPort()*) como também o tamanho do ficheiro, sendo de seguida criada uma *thread* responsável pelo *download*. Esta, por sua vez, recebe como parâmetros o *socket* e o ficheiro a que se pretende efetuar o *download*. A *thread* espera que o cliente efetue a sua conexão e assim que confirmada envia 1kbyte de cada vez do ficheiro para o cliente. **No lado do cliente** também foi criada uma *Thread Download* que recebe como parâmetros a diretoria atual (onde se pretende receber o ficheiro), o *socket*, o porto com o qual efetua a conexão com o server, o nome do ficheiro a receber e por último o tamanho do mesmo. Esta *thread* vai ler sempre a informação recebida de 1kbyte, no entanto se a mesma for menor que o tamanho do buffer é efetuado o mínimo entre os dois e recebido assim o resto do ficheiro que faltava sem perda de conteúdo.

Carregar um ficheiro para o servidor

Relativamente ao upload foi também criada uma **class que irá representar uma thread**. Esta funciona de modo análogo aos métodos implementados anteriormente (*Download*), no entanto neste caso é o *server* que pretende receber um ficheiro proveniente do cliente. Deste modo, no lado do *server* é criado um *socket* com o porto 0 novamente e posteriormente este é passado para o lado do cliente. De seguida é criada uma *thread Upload* que recebe como parâmetros o *socket* criado, o nome do ficheiro, o seu tamanho e a diretoria do *server* (*a atual*) que pretende receber o ficheiro. No lado do servidor é ainda utilizada a mesma técnica de receção de *kbytes* que no cliente no lado do download, enquanto que no cliente o modo de implementação é semelhante ao do envio de *kbytes* do server no download.

Failover

Mecanismo de HeartBeat implementado

De modo a que se o servidor ucdrive falhe é necessário que haja outro servidor (o secundário) para assegurar as comunicações entre servidor e cliente. Assim, o servidor secundário troca periodicamente, via UDP, *heartBeats* com o servidor primário. Estabeleceu-se, deste modo, que ao terceiro *failed heartBeat* o secundário assumiria o papel de servidor primário.

Na implementação deste mecanismo foi definido estaticamente o número máximo de *failed HeartBeats* (3), o *timeout* para que o *socket* não permaneça infinitamente à espera de algo do servidor primário e ainda o *period* que permite que a *thread* do server secundário não esteja constantemente a enviar *hearbeats* e assim haja períodos de pausa. Por outro lado, os portos (do Servidor Secundário e Primário assim como o porto da *thread HeartBeats*) encontram-se definidos num ficheiro de configuração “*config.txt*”.

No main, no servidor primário, é criado uma *thread* que irá permitir a receção dos *heartBeats* provenientes do servidor secundário. Esta é criada com um porto para posteriormente ser utilizado na criação do *socket* e assim permitir a receção destes *heartBeats*. Para ser possível demonstrar essa mesma receção por parte do servidor primário, o servidor secundário envia um *DatagramPacket* com um byte correspondente ao valor 0 e posteriormente este é convertido de byte para inteiro. Para demonstrar o sucesso da receção o servidor secundário envia como resposta novamente um byte com o valor 0. Deste modo, se o servidor secundário não conseguir comunicar com o primário durante o período definido como *timeout* é aumentando o número de *failed HeartBeats* e caso seja atingido o número máximo de *failed HeartBeats*, o servidor secundário assume o papel de primário.

Mecanismo de replicação implementado

No caso em que o servidor primário deixe de poder efetuar a comunicação com os clientes é necessário que haja um servidor secundário exatamente com os mesmos ficheiros que o servidor secundário. Pelo que quando é efetuado upload de algum ficheiro no servidor, na diretoria do cliente, é necessário também que este passe a existir no servidor secundário de modo a não haver perda de ficheiros.




Deste modo, foram utilizadas duas *threads* para o devido efeito, mas especificamente uma *Thread Backup* que pertence ao secundário e outra *Thread Replicate* implementada no servidor primário. Para a implementação desta técnica foi criado um ficheiro “*log.txt*” onde é colocado, sempre que ocorre um upload, a diretoria do ficheiro que sofreu alterações. Perante esta informação a *Thread Replicate* efetua o envio da mesma para o servidor Secundário através de UDP. Deste modo, a *Thread Backup* ao receber a diretoria onde o ficheiro se encontra no servidor primário cria também o mesmo no servidor secundário. Para o mecanismo de envio do conteúdo do ficheiro foi escolhido dividir o mesmo em blocos de bytes, no sentido de haver um maior controlo do envio de pacotes que o UDP não disponibiliza. Mais especificamente é efetuado o mínimo entre o tamanho do buffer de 1024 bytes e o que falta de enviar do ficheiro em bytes para que se saiba a última posição do bloco em causa no ficheiro e assim proceder ao envio do pacote *Datagram*.

Este mecanismo é efetuado até que o tamanho do ficheiro enviado seja igual ao que tamanho do que deu origem a esta ação. Perante o envio de cada bloco de bytes, a *Thread Backup* procede à sua escrita no ficheiro e para demonstrar a receção do mesmo envia um “ACK” com destino ao *socket* da *Thread Replicate*. Este procedimento ocorre também até que o tamanho da informação recebida seja igual ao tamanho do ficheiro que sofreu alteração. Por fim, se houver a receção deste “ACK” por parte da *Thread Backup* a operação ocorreu com sucesso, caso contrário é enviado novamente este pacote (descontando assim ao valor do tamanho que já tinha sido enviado, o tamanho do que não ocorreu com sucesso).




Testes efetuados à plataforma

Os testes foram efetuados assumindo que as configurações se encontravam bem realizadas (referenciadas nas condições de funcionamento). Os que passaram com sucesso são apresentados com um *check*, os restantes com uma *cross*.

Testes de autenticação

Descrição	Passed	Failed
Colocar uma <i>password</i> errada	O cliente não se consegue autenticar 	O cliente consegue autenticar
Colocar a <i>password</i> de um user noutra diferente	O cliente não se consegue autenticar 	O cliente consegue-se autenticar
Colocar a <i>password</i> antiga após ter sido alterada a mesma	O cliente não se consegue autenticar 	O cliente consegue-se autenticar


Testes de configuração







Descrição	Passed	Failed
O cliente conectou-se ao Primary Server, posteriormente é colocado configurações inválidas nos host/portos do cliente para causar a desconexão do mesmo. De seguida, são colocadas as configurações válidas no cliente para o mesmo se reconectar.	O cliente recupera da desconexão e depois da configuração dos portos reconecta-se com sucesso 	O cliente é desconectado e não recupera da exceção gerada
Mudança de portos do servidor pelo ficheiro de configurações	Os clientes conseguem conectar-se ao novo porto 	Os clientes não se conseguem conectar ao novo porto
Número de parâmetros errado da linha de comando que permite a configuração	O cliente não consegue efetuar a ação prevista e é apresentado as indicações do uso correto do comando 	O cliente conseguia efetuar a ação prevista

Testes de listagem de ficheiros e alteração da diretoria atual tanto no servidor como no cliente


Descrição	Passed	Failed
Listar localmente uma pasta vazia	Mensagem a informar que a pasta se encontra vazia 	Mostrar conteúdo que não se encontra na pasta
Recuar para uma diretoria anterior à "home" localmente	Mensagem a informar que não é possível recuar na diretoria 	O cliente consegue recuar para uma pasta anterior à "home"
Listar localmente uma diretoria com conteúdo	É mostrado o conteúdo correto da diretoria 	É mostrado algo que não corresponde ao conteúdo da diretoria
Mudar para uma diretoria válida dentro da pasta "home" localmente	É permitido a mudança 	Não é permitido a mudança
Mudar para uma diretoria que não existe no servidor	Não é permitido a mudança 	É permitido a mudança
Tentar recuar para uma diretoria anterior à pasta do cliente	Não é permitido a mudança 	É permitido a mudança
Listar o conteúdo de uma diretoria dentro da do utilizador no servidor	É mostrado o conteúdo da diretoria 	É mostrado o conteúdo errado da diretoria em causa
Listar o conteúdo de uma diretoria no servidor dentro da pasta do cliente	Mensagem a informar que a diretoria se encontra vazia 	É mostrado conteúdo não existente nessa diretoria

Testes às funcionalidades Upload e Download


Descrição	Passed	Failed
Efetuar o <i>upload</i> de um ficheiro para a diretoria em que o cliente se encontra no servidor	É efetuado o <i>upload</i> com sucesso 	Não foi possível efetuar o <i>upload</i>

Interromper o <i>upload</i> de um ficheiro para a diretoria a que o cliente se encontra no servidor	O ficheiro corrompido é eliminado e o cliente recebe uma mensagem assim que se reconecta que não foi possível realizar o mesmo 	O ficheiro mantém-se corrompido na diretoria
O servidor <i>crasha</i> enquanto está a ser feito um <i>upload</i> por parte do cliente	O cliente reconecta-se e recebe a mensagem de que o upload não ocorreu com sucesso 	O cliente experiênci problemas posteriores a este erro
Efetuar o <i>download</i> de um ficheiro existente na diretoria em que o cliente se encontra	O ficheiro é descarregado para a diretoria atual do cliente 	O ficheiro não se encontra na diretoria atual do cliente após o <i>download</i> ter sido finalizado
Efetuar o <i>download</i> de um ficheiro existente na diretoria em que o cliente não se encontra	É apresentado uma mensagem que o ficheiro não se encontra na diretoria atual do servidor 	É efetuado o download mesmo que o cliente não se encontre na pasta atual
O servidor <i>crasha</i> enquanto está a ser feito um <i>download</i> por parte do cliente	O ficheiro é eliminado e o cliente recebe uma mensagem de que o servidor se desconectou, ocorrendo um erro a efetuar o <i>download</i> 	O cliente experiênci problemas posteriores a este erro
O cliente <i>crasha</i> enquanto está a ser efetuado um <i>download</i>	O ficheiro não é eliminado e o cliente recebe uma mensagem de que não foi possível efetuar o <i>download</i> pretendido 	O cliente não é notificado de que não foi possível efetuar o <i>download</i>


Testes do mecanismo de HeartBeats

Descrição	Passed	Failed
Ligar o <i>Primary Server</i> e crashar este	O <i>Secondary Server</i> passa a assegurar as funcionalidades do <i>Primary</i> 	O <i>Secondary Server</i> não substitui o <i>Primary Server</i>

Teste do mecanismo de Replicação

Descrição	Passed	Failed
Efetuar o upload de um ficheiro para o <i>Primary Server</i>	O ficheiro de upload encontra-se também no <i>Secondary Server</i> 	O ficheiro de upload não se encontra no <i>Secondary Server</i>

Múltiplos Clientes

Descrição	Passed	Failed
Ter vários clientes ligados ao servidor	Todos os clientes conseguem efetuar as operações pretendidas, ou seja, estarem conectados em simultâneo 	Os clientes não se conseguem conectar em simultâneo

Divisão de tarefas

Para a implementação da plataforma ucDrive foram divididas as tarefas pelos dois elementos pertencentes ao grupo e sempre que se terminava alguma procedia-se à integração da mesma no projeto.