

Sistemas Distribuídos
Relatório do Projeto - Meta 1



ucDrive: Repositório de ficheiros na UC

2021/2022

Licenciatura em Engenharia Informática

PL4	Davide Figueiredo Areias	2019219422	uc2019219422@student.uc.pt
PL4	Francisco Faria	2019227649	uc2019219422@student.uc.pt
PL4	Iago Bebiano	2019219422	uc2019219422@student.uc.pt

Arquitetura do Software	3
Descrição Geral	3
Estrutura de Threads e Sockets	4
Organização do Código	5
Organização da Base de dados	5
Funcionamento dos Servidores	6
Funcionamento Geral	6
Funcionamento do comunicação RMI	7
Funcionamento Geral	7
Métodos Disponibilizados	7
Funcionamento do comunicação TCP	8
Funcionamento Geral	8
Métodos Disponibilizados	8
Funcionamento do comunicação UDP	8
Funcionamento Geral	8
Failover	9
Distribuição de Tarefas	9
Testes	9

1. Arquitetura do Software

Neste capítulo apresentamos a arquitetura do software, em todas as suas vertentes.

1.1. Descrição Geral

Em termos arquiteturais, o projeto contém 3 entidades que importam salientar:

- Consola de Administração:
 - Responsável pelas mais várias funções de administração:
 - Registar utilizador;
 - Listar as directorias/ficheiros por utilizador;
 - Configurar o mecanismo de failover;
 - Listar detalhes sobre o armazenamento;
 - Validar a replicação dos dados entre os vários servidores.
 - Implementação: **ucDriveAdmin.java**
- Aplicação cliente:
 - Contém todas as funcionalidades de Cliente:
 - Autenticação do utilizador no terminal de cliente;
 - Alterar a password do utilizador;
 - Configurar endereços e portos de servidores primário e secundário;
 - Listar os ficheiros que existem na directoria atual do Servidor/Cliente;
 - Mudar a directoria atual do Servidor/Cliente;
 - Carregar/Descarregar um ficheiro do servidor.
 - Pode haver múltiplas Consolas de Clientes
 - Implementação: **ucDriveClient.java**
- Servidor:
 - Responsável por fornecer métodos remotos tanto à consola de administração como aos terminais de Clientes para as mais diversas funções e por gerir uma base de dados por cada servidor com pastas pessoais por pessoas, entre outros documentos.
 - Deve haver 2 Consolas de Servidores (primário e secundário)
 - Implementação: **ucDriveServer.java/ucDriveServer2.java**

1.2. Estrutura de Threads e Sockets

Tanto na implementação da infraestrutura de rede como na arquitetura cliente-servidor foi seguida a proposta presente no enunciado.

Neste propunha-se que para cada servidor existiriam 3 tipos de comunicação. O primeiro tipo é RMI que serve para comunicação entre o admin e o server para efeitos de administração e configuração da plataforma. O segundo tipo é TCP que serve para comunicação entre o cliente e o server para interações com a plataforma. Por último uma comunicação UDP que é utilizada para troca de informação entre os dois servidores tanto para aplicação de mecanismos de failover como para replicação de ficheiros.

Devido à existência de 3 tipos de comunicação foram criados 4 sockets, um RMI, um TCP e dois UDP um para mecanismos de failover e outro para envio de ficheiros.

Em termos de threads as aplicações cliente são no seu global single threaded em todo o seu código tirando quando está a ocorrer um download para o qual se cria uma thread.

O admin é single threaded visto que nunca é necessário comunicação simultânea através de canais diferentes.

Os servidores são multithreaded, visto que precisam de simultaneamente receber clientes, admins, failover e replicação dos ficheiros no caso de ser primário e failover e replicação de ficheiros no caso de secundário. É importante referir que mesmo para o caso em que o servidor apenas tivesse de lidar com clientes continuaria a ser multi-threaded visto que este cria uma nova thread por cliente.

1.3. Organização do Código

A organização do código tem a seguinte estrutura:

- ucDrive : classe usada para implementação do rmi;
- ucDriveAdmin : código correspondente ao admin;
- ucDriveClient : código correspondente ao cliente;
- ucDriveServer: código correspondente ao servidor primário;
- ucDriveServer2: código correspondente ao servidor secundário;
- Ainda foram criadas duas classes para comunicação que se encontram dentro da pasta classes:
 - *loginPacket* : estrutura usado no login é constituído por duas string, uma com o username e outra com a password
 - *User*: Estrutura usada para guardar toda a informação relativa ao utilizador, isto é: username, palavra passe, universidade, cc, data de validade cc, e diretoria.

1.4. Organização da Base de dados

A nossa base de dados está dividida em 3 partes:

- base de dados primária/secundária:
 - o servidor primário utiliza a pasta db;
 - o servidor secundario utiliza a pasta db2;
 - Possui um ficheiro chamado users.txt onde se encontra toda a informação relativa aos utilizadores e uma pasta chamada dir na qual se encontram as pastas de cada utilizador com o nome do cartão de cidadão do mesmo, dentro das quais estão todos os ficheiros associados ao utilizador.
- ficheiros partilhados:
 - heartbeats.txt: contém informação relativa ao número máximo de pings falhados e o tempo entre cada ping;
 - primary.txt: contém informação que indica qual é o servidor primário e qual é o servidor secundário.

2. Funcionamento dos Servidores

2.1. Funcionamento Geral

Para explicar o funcionamento geral do servidor é necessário explicar o que cada uma das suas threads faz, para tal segue-se uma explicação do que ocorre em cada uma.

- **main-thread** - Esta thread é responsável pela inicialização do servidor no seu global, visto que é esta que inicializa todas as outras threads. Inicialmente esta verifica o conteúdo do `primary.txt`, de modo a saber que threads tem de ser executadas, se este for o servidor primário serão executadas as threads responsáveis pelos clientes, pelos admins, pela replicação de ficheiro e verificação de failover. De seguida esta espera que as threads acabem e termina a sua execução.
- **Client-thread** - Esta thread é responsável pela conexão com os diversos clientes via tcp e por criar uma nova thread para cada um, para tal este começa por inicializar um socket no porto 6000. De seguida chama-se a função `listenSocket.accept()` dentro de um `while` de modo a que nunca para de receber novos clientes ,ainda dentro deste `while` cria-se uma instância da classe `ucDriveCliente` que disponibiliza todas as funcionalidades ao cliente.
- **failover-thread primary** - Esta thread é responsável por receber e enviar mensagens de e para o outro servidor em ciclo. Começa por criar um `dataDatagramSocket` que irá esperar mensagens na porta 6001, de seguida espera por uma mensagem e após receber esta envia uma de volta.
- **replication-thread primary** - Esta thread é responsável por enviar os ficheiros do servidor primário para o secundário. Começa por esperar até que haja diretorias para replicar através de um ciclo `while` e um `wait` de modo a que não tenha espera ativa. Quando tem ficheiros para replicar este primeiro envia a diretorias e de seguida envia o conteúdo do ficheiro, voltando a entrar de novo no `while` até que seja necessário replicar ficheiros novamente.
- **failover-thread secondary** - Esta thread é executada quando o servidor é o secundário, sendo responsável por receber e enviar mensagens de e para o outro servidor em ciclo. Começa por adicionar inicializar um `datagram socket` e definir um `timeout`, de seguida envia uma mensagem para o servidor primário e espera por resposta, se está demorar mais do que 1 segundo é considerado erro e adiciona-se 1 ao número de pings falhados quando este alcançar um máximo definido previamente assume-se que o primário falhou e inicializa-se este como primário. Caso o ping não tenha falhado a thread entra em `sleep` durante um período definido pelo admin;

- **replication-thread secondary** - Esta thread é executada quando do servidor e o secundário sendo responsável por receber os ficheiros do servidor primário. Começa por criar um *dataDatagramSocket* que irá esperar mensagens na porta 6002, de seguida espera por duas mensagens por ficheiro sendo a primeira a diretoria e a segunda o conteúdo do ficheiro ficando a espera novamente de outro ficheiro.

3. Classe BD/BD2

3.1. Funcionamento Geral

Esta classe é extremamente importante pois é através desta que todas as alterações a base de dados e a replicação de dados são efetuadas. Tendo isto em conta consideramos relevante descrever todas as funções implementadas na mesma.

3.2. Métodos DB

- **addUser(User user)** - adiciona user ao array *usersInfo*, recorrendo de seguida a função *setUsers()* de modo a dar update ao *users.txt* file;
- **changeDir(User currentUser, String dir)** - troca a diretoria atual do utilizador *currentUser* pela *dir* dada como parâmetro, recorrendo de seguida à função *setUsers()* de modo a dar update ao *users.txt* file;
- **changePassword(User currentUser, String pass)** - troca a palavra passe do utilizador *currentUser* pela *pass* dada como parâmetro, recorrendo de seguida à função *setUsers()* de modo a dar update ao *users.txt* file;
- **changePingValues(int v1, int v2)** - troca os valores associados ao ping trocando *maxFailedRounds* por *v1* e *period* por *v2*;
- **setUsers()** - atualiza o ficheiro *users.txt* substituindo a informação pela que está atualmente no array *usersInfo*.
- **needupdate(String dir, int v) :**
 - aplicação de espera passiva no início de modo a que apenas uma thread esteja a alterar estes valores;
 - se o *v* é igual a 1 então verifica-se se já existe o *users.txt* no array das diretorias e adiciona-se caso não haja, saindo de seguida;
 - para *v* igual a 0 adiciona o *dir* ao array *needUpdateDir*.
- **userUpdate()** - função chamada pela thread de replicação de dados de modo a alterar a variável *needUserUpdate* para 0.
- **checkWhenNedded()** - ciclo while para espera passiva do qual a thread só sai quando o array das diretorias não estiver vazio.
- **getPingInfo()** - Função que atualiza as variáveis *maxFailedPing* and *period*, trocando-os pelos que estão no ficheiro *heartbeats.txt*.

4. Funcionamento do comunicação RMI

4.1. Funcionamento Geral

Como referido anteriormente na arquitetura apresentada, as comunicações RMI são responsáveis pelos efeitos de administração e configuração da plataforma. De seguida estão descritos os diversos métodos disponibilizados ao administrador.

4.2. Métodos Disponibilizados - Admin Side

Quando é dito que a função *example* comunica com o servidor, significa que a função chama um método remoto com o mesmo nome a partir do objeto remoto *h*, e recebe uma resposta.

- **register(ucDrive h)** - A função *Register* recebe os dados via input e é comunicado ao servidor o desejo de se criar um novo registo, este por sua vez devolve uma resposta, de modo registar um utilizador na base de dados.
- **listFiles(ucDrive h)** - Esta função pede ao administrador por um *user_id*. De seguida comunica com o servidor e recebe um array chamado de *FileList*, este é passado como argumento para a função *printDir*, que vai imprimindo recursivamente as pastas e ficheiros pertencentes ao utilizador selecionado.
- **storageInfo(ucDrive h)** - Esta função, dá a escolher ao administrador entre ver a memória alocada por cada utilizador ou por todos. De seguida comunica com o servidor e recebe o número em Bytes que é processado e se necessário convertido em KB ou MB.
- **changeFailover(ucDrive h)** - Aqui é pedido ao administrador que insira dois valores, o *Max HeartBeats Failed* e o outro é o tempo que leva cada *Ping* ou *HeartBeat* em ms. De seguida é comunicada a intenção de alterar estes valores ao servidor, e recebe uma resposta, se os valores foram alterados com sucesso ou não.

Existem ainda duas funções importantes:

- **connectSocket()** - Obtém o objeto Remoto, *h*, usado nas funções anteriores, e chama a função *menu*.
- **menu(ucDrive h)** - Tal como o nome indica é imprimido um menu onde o administrador pode escolher as várias opções, dadas pelas funções acima tratadas.

4.3. Métodos Disponibilizados - Server Side

- **register(User userData)** - A função Register recebe os dados do User e verifica se já existe algum utilizador com o mesmo número de cartão de cidadão, caso não exista o utilizador é adicionada à base de dados e é criada uma pasta pessoal em que o nome é o número do cartão de cidadão.
- **listFiles(String user_id)** - Recebe um user_id e devolve um array com todas as pastas e ficheiros que o utilizador selecionado contém na sua pasta pessoal.
- **storageInfo(String user_id)** - Recebe um user_id, se o user_id for igual a "all" é devolvido o valor em Bytes de todas as pastas pessoais, senão é devolvido apenas de uma pasta pessoal em específico.
- **changePing(String v1, String v2)** - Recebe dois valores, o maxHeartBeats Failed e o tempo que leva cada Ping ou HeartBeat em ms, e altera-os na base de dados.

5. Funcionamento do comunicação TCP

5.1. Funcionamento Geral

Como referido anteriormente na arquitetura apresentada, as comunicações RMI são responsáveis pelos efeitos de administração e configuração da plataforma. De seguida estão descritos os diversos métodos disponibilizados ao administrador.

5.2. Métodos Disponibilizados - Client Side

- **login()** - Função que recebe como input o username e a palavra passe, insere estes strings no loginPacket e envia este para o server. De seguida recebe um int correspondente ao sucesso da operação, caso tenha sido bem sucedida inicializa todas as variáveis necessárias e sai da função. Caso contrário pede as informações de login novamente;
- **menu()** - Função que imprime uma lista de todas as funcionalidades, recebendo um int que representa a funcionalidade desejada, de seguida encaminha este para o server, seguindo para o corpo da função desejada posteriormente;

- **changePass()** - Esta função recebe a nova palavra passe via input, e redireciona-a para o servidor através do objecto output stream, de seguida recebe a resposta do servidor correspondente ao sucesso da operação, caso não tenha sido bem sucedida informa-se o utilizador e volta-se ao menu. Caso contrário informa-se o utilizador e pede-se para fazer login novamente;
- **ListDirectoryServer()** - Esta função recebe um string do server que corresponde a lista de ficheiros;
- **changeServerDirec()** - Esta função recebe a nova directoria via input, e redireciona-a para o servidor através do objecto output stream, de seguida recebe a resposta do servidor correspondente ao tipo de operação efectuada (só mudança de directoria ou criação de uma pasta) para ambos os casos dá-se print do tipo de operação realizada e atualiza-se o *currentServerDir*;
- **ListDirectoryLocal()** - Esta função recorre à função *getFiles()* de modo a obter um string que contém todos os ficheiros presentes na *currentLocalDir* imprimindo-os de seguida;
- **changeLocalDirec()** - Esta função recebe a nova directoria via input, caso esta seja igual a "--" significa que se pretende andar para trás na directoria, este só altera a directoria caso esteja em alguma pasta para além do dir definido inicialmente *minDir*. Caso não estejamos na situação referida anteriormente verifica-se se a directoria desejada existe alterando-se em caso afirmativo e informando o utilizador caso contrário;
- **sendFile():**
 - Começa por receber como input o nome do ficheiro que se pretende dar download e o nome para o novo ficheiro.
 - Cria-se uma nova thread a qual se conecta com um socket novo do server
 - Envia-se o nome do ficheiro para download ao server e recebe-se uma resposta que identifica se este ficheiro existe ou não. Caso não exista sai-se.
 - Recebe-se o tamanho do ficheiro e através de um ciclo while lê-se este ficheiro até que se tenha lido todo o ficheiro.
- **uploadFile():**
 - Recebe-se o nome do ficheiro local e o nome do novo ficheiro no server via input
 - Cria-se uma nova thread a qual se conecta com um socket novo do server
 - Verifica-se se o ficheiro existe e informa-se o server desta verificação
 - Caso exista, envia-se o tamanho total do ficheiro e de seguida o conteúdo do ficheiro em blocos.

5.3. Métodos Disponibilizados - Server Side

- **login()** - recebe o loginPacket utilizador, utilizando a informação deste packet percorre todos os utilizadores a verificar se as informações estão corretas, caso não estejam informa o utilizador e espera por uma nova tentativa de login, caso contrário inicializa todas as variáveis e avisa o utilizador do sucesso;
- **menu()** - lê um int do utilizador e segue para a função associada a esse int;
- **changePass()** - recebe password do utilizador, percorre todo o array dos usuários até encontrar o correto e substitui a palavra passe;
- **ListDirectoryServer()** - Recorrendo a função listFiles() obtém-se uma lista de todos os ficheiro e envia-se a mesma ao utilizador;
- **changeServerDirec()** - Esta função recebe a nova diretoria do utilizador, caso esta seja igual a "--" significa que se pretende andar para trás na diretoria, este só altera a diretoria caso esteja em alguma pasta para além do dir Home. Caso não estejamos na situação referida anteriormente verifica-se se a diretoria desejada existe alterando-se em caso afirmativo, criando a pasta e informado o utilizador caso contrário;
- **ListDirectoryLocal()** - apenas aqui para o menu;
- **changeLocalDirec()** - apenas aqui para o menu;
- **sendFile()**
 - cria-se uma nova thread para executar download
 - cria socket e aceita conexão;
 - Recebe-se o nome do ficheiro e verifica-se se este existe, informando o utilizador do resultado;
 - caso exista envia-se o tamanho e de seguida todo o conteúdo do ficheiro em blocos, caso contrário para-se a operação.
- **uploadFile()**
 - cria-se uma nova thread para executar download;
 - cria socket e aceita conexão;
 - recebe um int que diz se o download pretende continuar ou não;
 - de seguida recebe o nome do novo ficheiro;
 - inicializa-se um file output stream;
 - Recebe-se o tamanho total do ficheiro e utilizando este valor recebe-se todo o conteúdo do ficheiro.

6. Funcionamento do comunicação UDP

6.1. Funcionamento Geral

A comunicação UDP que é utilizada para troca de informação entre os dois servidores tanto para aplicação de mecanismos de failover como para replicação de ficheiros. Esta comunicação é aplicada através de 4 threads, 2 por servidor já descritas anteriormente.

7. Failover

O **failover** dos servidores são implementados da seguinte forma:

- Ambos os servidores começam por aceder aos valores presentes no ficheiro `heartbeats.txt`, onde se encontram o valor correspondente ao *maxFailedPing* and *period*, valores cruciais para o comportamento do failover;
- De seguida o primário cria um datagram e espera por mensagem do secundário após receber esta retorna a mesma mensagem. Caso o servidor secundário não receber resposta passado 1 segundo do envio este incrementa a número de fails ping. Este processo ocorre até que o número de pings falhados exceda o `maxFailedPing`, o que faz com que o servidor secundário seja primário.

8. Distribuição de Tarefas

Davide - 1 Francisco - 2 Iago - 3

Estrutura de comunicação RMI (Interface, registo no registry: bind e lookup)	1,3
Registar utilizador	1,3
Listar as directorias/ficheiros por utilizador	1,3
Configurar o mecanismo de failover	2,3
Listar detalhes sobre o armazenamento	1
Validar a replicação dos dados entre os vários servidores	2
Estrutura de comunicação TCP e troca de mensagens	2
Autenticar o do utilizador no terminal de cliente	1
Alterar a password do utilizador	3
Configurar endereços e portos de servidores primário e secundário	1,2
Listar os ficheiros que existem na diretoria atual do servidor	1,2
Mudar a diretoria atual do servidor	1,2
Listar os ficheiros que existem na diretoria atual do cliente	3
Mudar a diretoria atual do cliente	1,2
Descarregar um ficheiro do servidor	2
Carregar um ficheiro para o servidor	2
Tratamento de exceções	1,2
Failover - Estrutura de comunicação UDP	1,2,3
Failover - Heartbeat e deteção de falha no servidor primário	1,2,3
Failover - Ficheiros carregados para o primário são copiados para o secundário	1,2
Failover - Clientes conseguem operar com o secundário quando o primário está em baixo	1,2
Relatório	1,2,3

9. Testes

9.1. Consola de Administração

- Registrar utilizador ✓
 - É verificado se já existe algum utilizador com o mesmo número de cartão de cidadão.
- Listar as diretorias/ficheiros por utilizador ✓
- Testar mecanismo Failover ✓
- Listar detalhes sobre o armazenamento ✓
- Validar a replicação de dados entre vários servidores ✓

9.2. Consola Cliente

- Estrutura de comunicação TCP e troca de mensagens ✓
- Autenticar o utilizador no terminal de cliente ✓
- Alterar a password do utilizador ✓
- Listar os ficheiros que existem na diretoria atual do servidor ✓
- Mudar a diretoria atual do servidor ✓
- Listar os ficheiros que existem na diretoria atual do cliente ✓
- Mudar a diretoria atual do cliente ✓
- Descarregar um ficheiro do servidor ✓
- Carregar um ficheiro para o servidor ✓

9.3. Atributos de Qualidade

- Tratamento de exceções +-
 - Caso o servidor vá abaixo cliente reconecta-se
 - Não foi implementado o retry nos downloads
- Failover - Estrutura de comunicação UDP ✓
- Failover - Heartbeat e deteção de falha no servidor primário ✓
- Failover - Ficheiros carregados para o primário são copiados para o secundário +-
 - Ficheiros pequenos funciona
 - Maiores da erro
- Failover - Clientes conseguem operar com o secundário quando o primário está em baixo ✓