

# [Sistemas Distribuídos]

ucDrive: Repositório de ficheiros na UC

## [Relatório do projeto ucDrive]

## Relatório realizado por:

## Alexandre Gameiro Leopoldo – 2019219929

## Luís Miguel Gomes Batista – 2019214869

# Estrutura da comunicação entre aplicações de Servidor

Servidor Primário

Servidor Secundário

Envia *heartbeats* para verificar se está ativo

*Socket* UDP para receção de *heartbeats*

*Socket* UDP para envio de ficheiro

*Socket* UDP para gerir portos de comunicação *failover*

*Socket* UDP para envio de *heartbeats*

*Socket* UDP para receção de ficheiro

Envia *acknowledgement* do *heartbeat* recebido

*Socket* UDP para receber portos de comunicação failover

Envia intenção de replicar ficheiro

Envia porta disponível para replicar ficheiro

Envia ficheiro a ser replicado

Envia *acknowledgement* de receção do ficheiro

# Estrutura de comunicação entre Cliente e Servidor (Primário)

Envia ficheiro para servidor

*Socket* TCP para upload de ficheiros para o servidor

Responde consoante o pedido recebido

Envia comandos para interagir com o servidor

*Socket* TCP para download de ficheiros do cliente

*Socket* TCP para upload de ficheiros para o cliente

*Socket* TCP para comunicação com clientes

*Socket* TCP para download de ficheiros do servidor

*Socket* TCP para comunicação com o servidor

Servidor Primário

Cliente

Solicita ficheiro para download

Envia ficheiro para cliente

Solicita ficheiro para upload

# Arquitetura do Servidor

No planeamento da arquitetura da aplicação do Servidor foi optado o uso de uma *thread* por conexão a cada cliente de modo a constituir uma “Thread-per-connection architecture”. A *thread* principal, após um conjunto de etapas de configuração, fica em ciclo a aceitar ligações TCP para comunicação com utilizadores. Estas comunicações são posteriormente geridas por uma *thread* da classe “ClientHandler” responsável por receber e responder a comandos do cliente.

Ao iniciar, a aplicação servidor lê um ficheiro de dados de clientes e dá opção de alterá-lo. Após isto pede endereços e portos relativos ao seu próprio servidor e a outro que faz parte do mecanismo de *failover*.

A classe “ClientHandler” está responsável por receber comandos do seu cliente e realizar todas as operações necessárias para que estes sejam concretizados. Trata também de criar as *threads* necessárias para a transferência de ficheiros com o cliente, assim como replicar todas as alterações feitas pelo cliente para o servidor secundário.

# Arquitetura do mecanismo de *failover*

Para a arquitetura do mecanismo de *failover* da aplicação Servidor, é essencial a distinção entre servidor primário e secundário. Esta diferenciação é realizada através de *heartbeats* no início do programa em que, caso haja um *acknowledgement*, a aplicação saberá que está outro servidor ligado e por isso passa a ser o servidor secundário. Caso contrário, ao fim de um dado número de *heartbeats* sem resposta, a aplicação assumirá que é o servidor primário.

A receção de *heartbeats* no servidor principal é realizada por uma *thread* da classe “UDPHeartbeat” que se encontra a correr em paralelo à escuta numa *socket* UDP. No servidor secundário o envio de *heartbeats* é feito por uma função na *thread* principal.

Estando estabelecida esta comunicação UDP, o servidor primário sabe que terá de replicar dados para o servidor secundário, sendo este passo fundamental para que não haja perdas de informação. De modo a que possa haver um envio de ficheiros, é utilizada a classe “UDPPortManager” responsável pela troca de portos para comunicações UDP e envio/receção de diretorias para manter os dados atualizados.

Assim que o servidor primário recebe um *heartbeat* de outro servidor pela primeira vez, assume que existe um servidor secundário e vai replicar toda a sua informação atual para este. A partir deste momento, qualquer alteração aos dados do servidor primário será enviada para o servidor secundário.

Para o envio e receção de um ficheiro são usadas as classes “UDPFileSender” e “UDPFileReceiver”, respetivamente. O envio é feito de forma repartida com um *buffer* de 8KB e entre cada envio é esperado um *acknowledgement* de forma a garantir que todos os pacotes são recebidos. Após o envio total do ficheiro o recetor calcula o seu *checksum* através da função SHA-256 e envia o resultado para o transmissor, que irá comparar com o seu. Se for igual, envia um sinal positivo para o recetor, acabando o processo. Caso contrário, envia um sinal negativo e reinicia o processo.

Em caso de falha do servidor primário, o servidor secundário (que até ao momento não aceitava ligações de clientes) irá assumir a responsabilidade, e tornar-se-á primário. Desta forma, os clientes conectam-se ao novo servidor primário, tendo todos os seus ficheiros e diretorias atualizadas.

# Arquitetura do Cliente

A aplicação cliente começa por pedir o endereço e porta dos dois servidores. Após isto tenta ligar-se ao primeiro servidor, sendo que se falhar tenta conectar-se ao segundo. Assim que consegue estabelecer uma ligação são lhe pedidas as credenciais de autenticação, que são enviadas ao servidor obtendo a informação acerca da sua autenticação.

De seguida, é apresentado uma diretoria com a localização atual do cliente do servidor, pronta a receber comandos.

Lista de comandos:

* **?** – Apresenta a lista de comandos disponíveis;
* **ls** – Lista os ficheiros e pastas dentro da diretorial atual;
* **cd** [nome/..] – Altera a diretoria atual para a pasta selecionada;
* **mkdir** [nome] – Cria uma pasta na diretoria atual com o nome especificado e muda a diretoria atual para essa pasta;
* **rm** [nome] – Remove a diretoria ou ficheiro selecionado;
* **pw** [password] – Altera a palavra-passe do utilizador autenticado;
* **dw** [nome] – Descarrega um ficheiro do servidor para a diretoria local atual;
* **up** [nome] – Envia um ficheiro local para a diretoria atual do servidor;
* **clear** – Limpa o terminal;
* **sv** – Altera as informações dos endereços e portos dos servidores;
* **ch** – Alterna entre as diretorias local e do servidor;
* **exit** – Sai do servidor e fecha a aplicação.

Para operações de transferência de ficheiros, a aplicação cria uma *thread* que fica responsável por enviar ou receber o ficheiro pretendido, de modo a que a aplicação continue operacional durante todo este processo.

Em caso de falha do servidor, a aplicação tenta reconectar-se e em caso de falha, conecta-se ao segundo. Neste momento são lhe pedidas as credenciais de autenticação novamente.

# Distribuição de tarefas

A distribuição do trabalho realizado foi feita de forma igual, sendo que o papel de ambos foi fundamental para o desenvolvimento do projeto e obtenção do resultado final.

# Descrição dos testes feitos à plataforma

Olasdasd