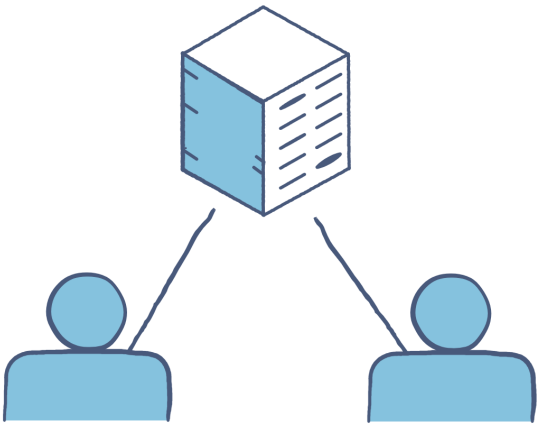
Universidade de Coimbra
Faculdade de Ciências e Tecnologia Universidade de Coimbra
Licenciatura em Engenharia Informática

Sistemas Distribuídos

ucDrive: Repositório de ficheiros na UC

2021/2021



Projeto realizado por:
Rodrigo Ferreira - 2019220060
Sofia Alves – 2019227240

Professor orientador:
Hugo Amaro

1. Arquitetura do software

Neste capítulo iremos descrever a arquitetura do software desenvolvido.

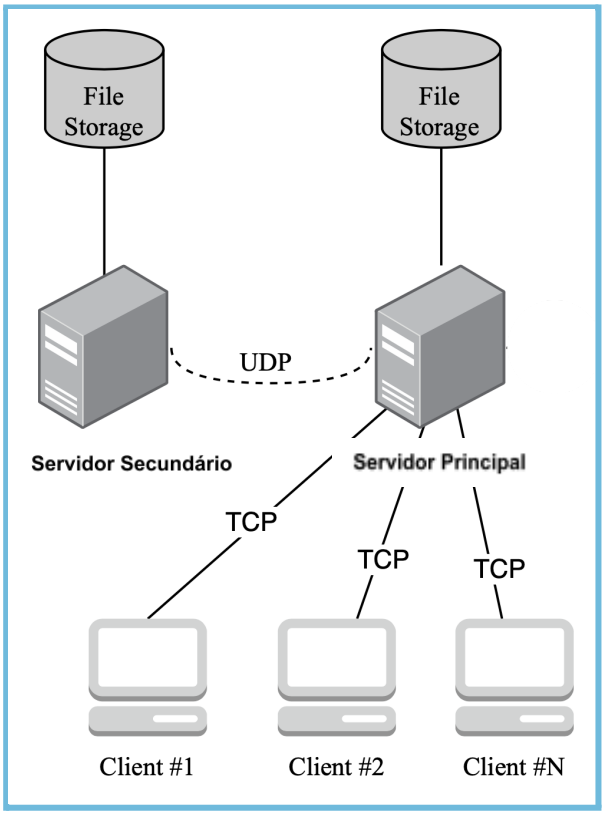


Figura 1 - Arquitetura do software

1.1. Descrição Geral

O projeto é composto por 3 partes, as quais desempenham as seguintes funções:

- **Servidor Principal** (*UcDrive Server*)
  - Responsável por aceitar ligações TCP dos clientes;
  - Comunicar com os clientes através de TCP, respondendo aos seguintes comandos:
    - listagem de ficheiros na diretoria atual do servidor;
    - alterar a diretoria do servidor;
    - *login*;
    - *logout*;
    - alteração da *palavra passe* do cliente;
    - envio e receção de ficheiros.
  - Responsável por receber e responder aos *heartbeats* enviados pelo servidor secundário via UDP;
  - Responsável por enviar ficheiros recebidos dos clientes para o servidor secundário, como forma de *backup*, mantendo a informação entre ambos os servidores sincronizada;

- Guardar a informação de *login* dos diferentes usuários, bem como a última diretoria acedida;

- **Servidor Secundário**
  - Periodicamente enviar *heartbeats* para o servidor principal de modo a verificar se o mesmo ainda se encontra operacional.
  - Capacidade de assumir o papel de servidor principal, caso o mesmo não responda aos *heartbeats*.
  - Receber ficheiros do servidor principal como forma de realizar o seu *backup*.

- **Clientes**
  - Capazes de se conectarem ao servidor principal através de uma ligação TCP.
  - Enviam comandos obtidos por terminal para o servidor e mostram ao utilizador a resposta do servidor
  - Capazes de se tentarem conectar a outro servidor caso se desconecte do servidor atual.

1.2. Funcionamento detalhado da estrutura de Threads e Sockets

Neste subcapítulo iremos descrever com detalhe o comportamento do programa e decisões tomadas ao longo do desenvolvimento do projeto.

- **Servidor *ucDrive***

O servidor *ucDrive* não arranca automaticamente como servidor principal, isto acontece para prevenir a existência de várias instâncias de servidores principais a correr simultaneamente, o que iria gerar conflitos. Sendo assim, um servidor ao arrancar envia *heartbeats* para saber se já existe algum com o papel de servidor principal. Caso não obtenha resposta a 3 *heartbeats* assume-se que não existe um servidor principal de momento e o próprio assume o papel.

- **Servidor Principal**

Quando um servidor assume o papel de servidor principal, este cria inicialmente uma *thread* para responder aos *heartbeats* do servidor secundário. Esta *thread* recebe *pacotes* para o porto UDP, definido no ficheiro de configuração, porto 5000. Os pacotes dos *heartbeats* contêm 2 valores possíveis, 200 ou 101.

No caso de ler 200, código de *heartbeat*, o servidor principal irá responder mandando um arraylist com os nomes dos ficheiros que ainda não foram enviados para o servidor secundário.

No caso de ler 101, código de pedido de ficheiro do servidor secundário, o servidor principal irá criar uma *socket* na porta de ficheiros UDP, porta 1000 no ficheiro de configuração. Após isso irá esperar, com *timeout*, para receber o nome do ficheiro que o servidor secundário quer fazer backup. Em resposta, o servidor primário envia o tamanho do ficheiro e começa a enviar pacotes de 1KB com o

conteúdo do ficheiro. Após o envio dos pacotes com o conteúdo do ficheiro o servidor principal aguarda pela resposta do servidor secundário com o valor do *checksum*. Caso o valor seja igual ao calculado pelo servidor principal, este marca o ficheiro como tendo sido enviado com sucesso e remove-o do *arraylist* de ficheiros que ainda não foram enviados para o servidor secundário. Se o valor do *checksum* for diferente, o nome do ficheiro será mantido no arraylist e no próximo *heartbeat* tentar-se-á novamente enviar para o servidor secundário.

Para além da thread criada para responder aos heartbeats do servidor secundário, o servidor principal escuta por ligações TCP do cliente e ao aceitar a ligação cria uma nova thread para a comunicação cliente-servidor.

Na ligação TCP com o cliente, o servidor pode responder a vários comandos enviados pelo utilizador, explicados no subcapítulo 1.3. No caso do envio e receção de ficheiros entre o cliente e servidor é criada uma nova *socket* numa porta disponível e transferido o ficheiro por lá, quem recebe o ficheiro, recebe-o em blocos visto não saber o tamanho do ficheiro a receber.

- **Servidor Secundário**

O servidor ao assumir o papel de servidor secundário irá criar uma thread para o envio de *heartbeats* periódicos. Os *heartbeats* são representados por um inteiro de valor 200 e são enviados via UDP para o porto do servidor principal, definido no ficheiro de configuração.

Após o envio do *heartbeat*, o servidor secundário aguarda pela resposta ao *heartbeat,* com *timeout.* Caso o servidor secundário não obtenha resposta nesse intervalo de tempo, incrementa o número de *heartbeats* perdidos e envia um novo *heartbeat*. Se falharem 3 *heartbeats* seguidos o servidor secundário vai assumir que o servidor principal se encontra *offline* e assume o seu papel, realizando as funções descritas acima.

Se o servidor secundário obtém resposta ao *heartbeat*, este verifica se a resposta contém algum nome de ficheiro. A resposta contém o nome dos ficheiros que foram recebidos recentemente e ainda não foram enviados para o servidor secundário. Caso contenha, irá pedir ao servidor principal cada um desses ficheiros e realizar a sua cópia.

Como forma de fazer cópia do ficheiro para o servidor secundário via UDP, inicialmente o servidor secundário envia o código 101, código de pedido de ficheiro, para o servidor principal. Após o envio do código o servidor secundário irá enviar o nome do ficheiro que deseja obter para a porta de transferência via UDP, definida também no ficheiro de configuração. De seguida, aguarda pela resposta do servidor principal com o tamanho do ficheiro a receber e inicia um ciclo onde lê pacotes com o conteúdo do ficheiro e vai escrevendo-os o conteúdo para um ficheiro. Em qualquer parte da receção do ficheiro via UDP existe um *timeout* de modo a garantir o bom funcionamento do servidor.

Após ler os pacotes com o conteúdo do ficheiro, o servidor calcula o checksum e envia para o servidor primário o valor obtido. Caso o *checksum* dê diferente no próximo *heartbeat*, o ficheiro continuará na resposta enviada pelo servidor principal e irá ser tentado o envio do ficheiro novamente, reescrevendo a versão atual do ficheiro que se encontrava incorreta.

- **Client**

O cliente inicialmente cria uma *socket* e tenta-se conectar ao servidor principal. Caso exista um servidor principal este irá aceitar a ligação TCP e o cliente pedirá ao utilizador que se autentique.

Após a autenticação com sucesso, a informação de *login* é guardada para uma autenticação automática caso o servidor atual fique *offline* e outro o substitua.

Após isso, o cliente pode enviar comandos que, dependendo do comando, o cliente pode enviar ao servidor e apresentar a resposta do mesmo ou apenas executar o comando sem a necessidade de comunicar com o servidor. Os comandos possíveis estão explicados em mais detalhe no subcapítulo 1.3.

Nos comandos de envio e receção de ficheiros, o cliente notifica o servidor e de seguida aguarda pela resposta da porta a que se conectar para receber ou enviar o ficheiro, tal como descrito no Servidor Principal.

O cliente, ao enviar o comando de alteração de diretoria do lado do cliente, conseguirá mover-se para pastas que já existam de forma livre. No entanto, apenas se consegue mover nas diretorias do lado do servidor até à pasta "*Home*", não conseguindo recuar por uma questão de segurança. Por outro lado, na diretoria do servidor caso a subpasta não exista esta será criada, permitindo assim ao cliente criar subpastas e enviar ficheiros apenas usando comandos.

Para uma melhor leitura na linha de comandos, a diretoria atual do lado do cliente apenas mostra as 3 diretorias pai. Os comandos de listagem de ficheiros e alteração da pasta do lado do servidor virão com um *output* indicando a subpasta que o cliente se encontra nesse lado.

1.3. Comandos do terminal

Aqui iremos explicar os comandos possíveis de executar no terminal do cliente e o funcionamento do terminal e do *ucDrive* ao executá-los.

Login

A autenticação do cliente é realizada ao correr o ficheiro *terminal.jar* e tendo sido possível estabelecer ligação com o servidor principal. É pedido ao utilizador o seu nome de utilizador e palavra passe, esta informação é enviada para o servidor após o envio da *keyword* "LOGIN" para informar que o cliente irá realizar o *login*. Após receber toda a informação o servidor tenta autenticar o cliente com a informação carregada do ficheiro "users_info.txt" e devolve a resposta ao cliente. Caso o cliente não consiga autenticar se, este perguntará novamente ao utilizador para se tentar autenticar.

Listagem de ficheiros na pasta do cliente - "ls client"

O comando permite visualizar os ficheiros e pastas na diretoria atual do cliente. O output no terminal será uma lista de ficheiros e pastas espaçados por espaços. Ficheiros e pastas com espaços serão colocados entre aspas para melhor perceção.

Listagem de ficheiros na pasta do servidor - “ls server”

O comando permite visualizar os ficheiros e pastas na diretoria atual do servidor. O *output* do terminal será a diretoria atual no servidor seguido da lista com os nomes de ficheiros e pastas separados por espaços. Ficheiros e pastas com espaços serão colocados entre aspas para melhor perceção.

Alteração da diretoria do cliente - “cd client *nome do ficheiro*”

O comando permite mudar a diretoria que o cliente se encontra de momento. O terceiro parâmetro corresponde à subpasta que o utilizador deseja entrar ou recuar para a pasta pai caso insira “..”. Caso a subpasta não exista o terminal irá imprimir a uma mensagem de insucesso referindo que a subpasta inserida não existe.

Alteração da diretoria do servidor - “cd server *nome do ficheiro*”

O comando permite mudar a diretoria que o cliente se encontra do lado do servidor. O terceiro parâmetro corresponde à subpasta que o utilizador deseja entrar ou recuar para a pasta pai caso insira “..”. Caso a subpasta não exista o servidor irá criar. O utilizador também se encontra limitado sendo que não consegue recuar da pasta “Home” dentro da sua pasta no servidor.

Encerrar o terminal - “exit()”

O comando permite ao cliente desconectar-se do servidor e terminar o programa. Ao enviar este comando o servidor irá terminar a ligação entre o cliente e servidor e guardar a informação do utilizador e a diretoria atual do cliente no ficheiro “users_info”.

Alterar a palavra passe do utilizador - “cp *nova palavra passe*

Este comando permite ao utilizador alterar a sua palavra passe. A nova palavra passe será enviada para o servidor onde atualiza a informação do cliente. Após isso é pedido novamente ao utilizador para se autenticar.

Obtenção de um ficheiro do servidor - “get *nome do ficheiro*

O envio deste comando permitirá ao utilizador obter um ficheiro do servidor. O ficheiro precisa de se encontrar na pasta atual que o cliente se situa no servidor e será descarregado para a pasta que o cliente se encontra actualmente do lado do cliente. Durante a transferência o cliente não conseguirá enviar outros comandos, sendo apenas executados após a finalização da transferência do ficheiro.

Envio de um ficheiro para o servidor - “send *nome do ficheiro*”

Este comando permite ao cliente enviar ficheiros para o servidor. O ficheiro a enviar precisa de se encontrar na pasta atual do lado do cliente e será colocado na pasta onde o utilizador se encontra do lado do servidor. Durante o tempo de envio o cliente também não conseguirá enviar comandos.

2. Mecanismo de failover

Neste capítulo iremos descrever em detalhe os mecanismos de *failover* implementados. Como forma de garantir que existe sempre um servidor funcional capaz de responder aos pedidos dos clientes, implementamos *heartbeats* entre os servidores.

O servidor secundário periodicamente envia *heartbeats* para o servidor principal para verificar se o mesmo se encontra *online*. Caso o servidor secundário falhe 3 *heartbeats* seguidos assume que o servidor principal ficou *offline* e assume o seu papel, passando a aceitar ligações dos clientes e a responder aos seus pedidos. No caso de o servidor inicial voltar a ficar *online*, assume um papel de servidor secundário e envia *heartbeats* à procura do servidor principal, caso obtenha resposta mantém o papel de secundário.

O servidor principal mantém informação de todos os ficheiros recebidos que ainda não foram enviados para o servidor secundário, mesmo que o servidor secundário não se encontre *online* no momento da receção do ficheiro. Após o servidor secundário ficar *online*, o servidor principal envia a lista com os nomes dos ficheiros que necessitam de *backup*. O servidor secundário com esta informação irá começar a pedir os ficheiros individualmente.

Na ligação TCP entre o cliente e o servidor principal também temos implementados mecanismos de failover, o terminal guarda a informação de login do utilizador bem como o último comando enviado pelo utilizador enquanto este não for finalizado. Esta informação é guardada para o eventual caso de o servidor principal ficar *offline.*

Neste caso o terminal irá tentar conectar-se ao outro servidor. Caso esse servidor já tenha assumido a sua função de servidor principal, o terminal conseguirá se conectar, realizar login e reenviar o comando que não foi finalizado, tudo de forma automática.

3. Distribuição de tarefas

De um modo geral, a dinâmica consistiu em distribuir a realização de certas *features* por cada elemento, sendo que foram usadas *branches* para a realização das mesmas (*git*). Outras funcionalidades foram realizadas em conjunto durante reuniões realizadas à distância, incluindo a criação de sessões em que ambos os elementos do grupo programavam juntos simultaneamente.

4. Testes realizados

Para cada funcionalidade implementada foram realizados testes e verificações de modo a tornar a solução o mais robusta possível. A seguir apresentam-se os mesmos realizados.

Autenticação

- Correr o programa e realizar o *login* inserindo um nome de utilizador já registado e a respetiva palavra passe nos campos corretos.

  Resultado - Dá-se início à sessão, acedendo à diretoria *Home*, caso seja a primeira vez a autenticar-se, ou à última diretoria acedida na sessão anterior, caso contrário.

- Correr o programa e realizar o *login* inserindo um nome de utilizador que não esteja registado ou com a palavra passe errada.

  Resultado - Volta a ser solicitado o nome de utilizador e a palavra passe, não sendo dado qualquer acesso ao *software*.

Listar conteúdo da diretoria atual do cliente

- Correr o comando “ls client”.

  Resultado - Do lado do cliente verificam-se os ficheiros presentes na diretoria atual e imprime-se o seu *path* e os respetivos ficheiros no ecrã separados por espaços. Ficheiros com espaços no nome aparecem entre aspas.

- Inserir o comando de maneira incorreta, seja escrever apenas “ls” ou escrever incorretamente o segundo parâmetro.

  Resultado - O mesmo será descartado aparecendo apenas mensagem de erro no ecrã “Wrong Syntax: ls client|server”

Listar conteúdo da diretoria atual do servidor

- Correr o comando “ls server”.

  Sucesso - Do lado do servidor verificam-se os ficheiros presentes na diretoria atual e imprime-se o seu *path* e os respetivos ficheiros no ecrã separados por espaços. Ficheiros com espaços no nome aparecem entre aspas.

- Inserir o comando de maneira incorreta, seja escrever apenas “ls”, escrever incorretamente o segundo parâmetro.

  Insucesso - O mesmo será descartado aparecendo apenas mensagem de erro no ecrã “Wrong Syntax: ls client|server”

Mudar a diretoria atual do cliente

- Correr o comando “cd client *diretoria*”.

  Resultado - Do lado do cliente, ao inserir a pasta para onde se quer mover, este irá atualizar a pasta atual para a mesma e irá mudar a *command prompt* para a diretoria atual e as 2 diretorias anteriores. Exemplo: ClientProject/src/files> cd client test
  src/files/test>

- Correr o comando “cd client ..” .

  Resultado - Permite mudar para uma diretoria pai. O utilizador pode mover-se livremente pelas diretorias.

- Inserir o comando de forma incorreta, seja escrever apenas “cd”, “cd client” ou escrever incorretamente o segundo parâmetro.

  Resultado - O mesmo será descartado aparecendo apenas a mensagem de erro no ecrã “Wrong Syntax: cd client|server *new_dir*”.

- Inserir o comando corretamente mas escolher uma diretoria que não exista.

  Resultado - O mesmo será descartado aparecendo apenas a mensagem de erro no ecrã “folder doesn't exist”

Mudar a diretoria atual do servidor

- Correr o comando “cd server *diretoria*”.

  Resultado - Do lado do servidor, ao inserir a pasta para onde se quer mover, este irá atualizar a pasta atual para a mesma e irá mudar a *command prompt* para a diretoria atual e as 2 diretorias anteriores.

- Correr o comando “cd server ..” .

  Resultado - Permite mudar para uma diretoria pai. Este ficará limitado à diretoria do utilizador atual, não podendo navegar para as diretorias de outros utilizadores.

- Correr o comando cd com um caminho, por exemplo “cd server *diretoria1/diretoria2/diretoria3*, sendo que todas as diretorias existem.

  Resultado - O comando é realizado com sucesso, mudando para a última diretoria referida e atualizando a *command prompt*.

- Inserir o comando corretamente mas escolher uma diretoria que não exista.

  Resultado - Cria a diretoria referida, mudando para a mesma.

Alterar palavra passe do cliente

- Correr o comando “cp *nova palavra passe*”.

  Resultado - O servidor recebe o nome do utilizador e a nova palavra passe e atualiza os dados do cliente, alterando assim a sua palavra passe. O cliente é

desconectado da sessão, sendo solicitado que realize um novo *login*, desta vez com a palavra passe nova. O cliente procede ao mesmo, autenticando-se com sucesso com a nova palavra passe.

- Inserir o comando de forma incorreta, escrevendo apenas “cp” ou inserindo uma palavra passe com espaços.

  Resultado - Não é alterada a palavra passe e a sessão permanece iniciada, aparecendo apenas no ecrã a mensagem de erro “Wrong Syntax: cp *new_password*”.

Descarregar um ficheiro do servidor

- Correr o comando “get *nome do ficheiro*”.

  Resultado - Primeiramente é verificado se o ficheiro pedido existe. Caso exista, é criada uma nova socket onde o terminal se irá conectar e receberá o ficheiro via TCP.

- Servidor principal desconectar-se a meio da transferência.

  Resultado - O secundário toma o seu papel e o cliente continua a ler e a escrever do seu *buffer* para um novo ficheiro, conseguindo receber o mesmo completamente.

- Ambos os servidores desconectarem-se a meio da transferência.

  Resultado - Caso o servidor tenha conseguido enviar o ficheiro todo, o cliente continua a ler e a escrever do seu *buffer* para um novo ficheiro, conseguindo receber o mesmo completamente.Caso contrário, o ficheiro ficará incompleto.

- O terminal do cliente desconectar-se a meio da transferência.

  Resultado - Este chega a receber o ficheiro com o conteúdo até ao momento enviado.

- Requisitar um ficheiro que não exista do lado do servidor.

  Resultado - Não será enviado qualquer ficheiro para o cliente, imprimindo apenas no ecrã a mensagem de erro “File not found”.

- Inserir o comando de forma incorreta, escrevendo apenas “get”.

  Resultado - Não será enviado qualquer ficheiro para o cliente, imprimindo apenas no ecrã a mensagem de erro “Wrong Syntax: get *file_name*”.

Carregar um ficheiro para o servidor

- Correr o comando “send *nome do ficheiro*”.

  Resultado - É criado uma *socket* do lado do cliente onde o servidor se irá conectar para poder enviar o ficheiro via TCP.

- Servidor principal desconectar a meio da transferência.

  Resultado - É disparada uma exceção do lado do cliente, voltando a tentar enviar o ficheiro para o outro servidor. Ao mesmo tempo, o servidor secundário toma o papel do principal, recebendo assim o conteúdo do ficheiro que será escrito para um novo ficheiro, recebendo com sucesso o mesmo.

- Ambos os servidores desconectarem-se a meio da transferência.

  Resultado - O servidor principal chega a receber o ficheiro com o conteúdo até ao momento enviado. O ficheiro ficará incompleto.

- O terminal do cliente desconectar a meio da transferência

  Resultado - Caso o terminal consiga enviar o ficheiro completo o servidor principal continua a ler e a escrever do seu *buffer* para um novo ficheiro, conseguindo receber o mesmo completamente. Caso contrário, o ficheiro ficará incompleto.

- Requisitar um ficheiro que já exista do lado do cliente.

  Resultado - O ficheiro é enviado na mesma, escrevendo por cima do ficheiro que contenha o mesmo nome do lado do servidor.

- Requisitar o envio de um ficheiro que não existe do lado do cliente.

  Resultado - Não será enviado qualquer ficheiro para o servidor, imprimindo apenas no ecrã a mensagem de erro “File not found”.

- Inserir o comando de forma incorreta, escrevendo apenas “send”.

  Resultado - Não será enviado qualquer ficheiro para o servidor, imprimindo apenas no ecrã a mensagem de erro “Wrong Syntax: send *file_name*”.

Configurar os ip’s e os portos e abrir consola do cliente

- Ao correr o programa correspondente ao cliente, indicar os ip’s e portos correndo o comando “java -jar terminal.jar *ip do servidor principal* *porto do servidor principal* *ip do servidor secundário* *porto do servidor secundário*

  Resultado - O programa é iniciado sem quaisquer constrangimentos, abrindo *sockets* com esses portos/ip’s

- Inserir o comando de forma incorreta, seja não escrevendo apenas quatro argumentos ou não respeitar o formato dos portos (quatro dígitos).

  Resultado - O comando é descartado, aparecendo no ecrã a mensagem de erro “Wrong Syntax: java -jar terminal.jar *mainServerIP* *mainServerPort* *secServerIP* *secServerPort*”.

- Inserir o comando quando ainda não existe um servidor primário.

  Resultado - A conexão será rejeitada, não conseguindo conectar-se a qualquer servidor.

Logout

- Correr o comando “exit()”.

  Resultado - O cliente termina a sessão corretamente, notificando o servidor do mesmo. Este fecha a conexão com o cliente e guarda as informações sobre a sessão.

Outros testes

- Múltiplos clientes conectarem-se a servidor primário.

  Resultado - É feita a conexão sem quaisquer constrangimentos.

- Servidor primário deixar de funcionar.

  Resultado - Ao fim de 3 *heartbeats* sem resposta, o servidor secundário assume o seu papel.

- Múltiplos clientes conectarem-se ao servidor primário. De seguida, este desconectar-se e o secundário passa a tomar o seu papel.

  Resultado - É realizada a conexão ao novo servidor com sucesso, permitindo a realização de todas as operações sem quaisquer constrangimentos.