FCTUC FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Sistemas Distribuídos

Licenciatura em Engenharia Informática
Turma Prática-Laboratorial 1

Docente responsável

Nuno Alexandre Martins Seixas

Grupo

Alexy de Almeida	adenis@student.dei.uc.pt	Nº2019192123
José Gonçalves	josegoncalves@student.dei.uc.pt	Nº2019223292

2 abril de 2022

1. Arquitetura de Software

O servidor ucDrive é composto por 2 classes principais: ***Server.java*** e ***Client.java***. Os dois são programas independentes. A classe ***Connection.java*** contém todos os métodos e threads principais à realização das tarefas. No ficheiro ***users.txt*** está contida a informação de todos os utilizadores, dividida da seguinte forma: *username,password,directory.* Dentro da pasta *server*, que contém todas as classes que dizem respeito ao funcionamento dos servidores, encontra-se uma pasta *clients* que contém as diretorias de cada utilizador que se autenticou.

No que diz respeito ao cliente, ***Client.java***, este inicia-se com uma thread principal na qual é estabelecida a ligação ao servidor através de uma *client* socket TCP, através da qual envia comandos e recebe as respetivas respostas. Quando é iniciado um *upload* ou *download* de um ficheiro, ocorre a criação de uma nova socket que se liga a uma nova porta disponibilizada pelo servidor. No final destas duas operações, estas sockets são fechadas.

Em relação ao servidor, ***Server.java***, este começa com uma main thread onde ocorre a leitura do ficheiro de configuração, ***addresses.txt***, e a criação de duas outras threads: ***FileSyncReceiver.java***, responsável pela receção de ficheiros duplicados a partir do servidor primário, e ***HeartbeatSender.java***, que envia heartbeats ao servidor primário através de uma socket UDP e aguarda resposta de modo a poder assumir o papel de primário se este falhar. Ainda em relação ao ***FileSyncReceiver.java***, este sempre que recebe um pedido para receber um novo ficheiro através da socket UDP que abre no início, cria uma nova thread, ***FileReceiver.java***, na qual ocorre a transferência do ficheiro a replicar via uma nova socket UDP.

Quando o server muda para funcionar em modo primário por falta de resposta do anterior servidor primário, as duas threads anteriores são terminadas e abra-se uma nova *server* socket TCP e uma nova thread, ***HeartbeatReceiver.java***, responsável por responder aos *heartbeats* do servidor secundário através de uma socket UDP. Quando a *server* socket TCP a correr na *main thread* aceita uma nova ligação de um cliente, passa essa *client* socket TCP para uma nova thread, ***Connection.java***. Nesta thread são lidos os comandos recebidos do cliente (cada thread ***Connection.java*** lida com um cliente apenas), que a seguir são processados, sendo tomada a ação apropriada e enviada a correspondente resposta.

No caso do download de um, ficheiro, ocorre a criação de uma nova thread, ***FileDownload.java***, que cria uma nova *server* socket TCP ao qual se liga a *client* socket TCP criada na thread ***Client.java*** anteriormente mencionada. De modo análogo, quando se inicia a transferência de um ficheiro com origem no cliente para o servidor, ocorre a criação de uma thread ***FileUpload.java***, onde uma nova server socket TCP aceita a ligação se uma client socket TCP do cliente dedicada para esta operação. No entanto, após a completude do *upload*, cria-se uma nova thread, ***FileSyncSender.java***, na qual uma nova socket UDP envia uma cópia do novo ficheiro para a socket UDP criada pelo ***FileReceiverUDP.java*** do servidor secundário para tentar manter replicado o estado dos dois servidores.

Quando o cliente altera a sua palavra-passe ou o diretório ativo no servidor, também ocorre a instanciação de uma nova thread ***FileSynSender.java***, que irá igualmente sincronizar através de uma socket UDP o ficheiro onde são guardados os utilizadores registados, as respectivas palavras-passe e o último diretório visitado no server, ***users.txt***.

2. Funcionamento do servidor ucDrive

Para o programa funcionar, é feito primeiro o arranque do servidor primário e, de seguida, do servidor secundário e clientes. O cliente estabelece uma ligação ao servidor e, se essa for bem sucedida, surge um menu com os vários comandos que o utilizador pode inserir para realizar determinadas tarefas. Não fizemos uma distinção dos comandos dependendo se o utilizador já esteja autenticado ou não, no entanto, são feitas verificações para o utilizador não poder utilizar determinados comandos se este ainda não se encontrar autenticado.

Todos os comandos que o utilizador escrever na linha de comandos relativos ao servidor, seja "lss" (listar ficheiros na diretoria atual do cliente no servidor) ou "cds <directory>" (mudar diretoria atual do cliente no servidor), são tratados na classe ***Connection.java*** pelos métodos correspondentes. O cliente envia o que escreveu na linha de comandos para o servidor e o servidor verifica se se trata de um comando válido. Por exemplo, se o cliente enviar "au squalexy aaa", que permite fazer a autenticação do cliente, o servidor irá verificar primeiro se "au" é um comando válido e se no ficheiro ***users.txt*** existe um username "squalexy" com a password "aaa". Se o comando for aceite, envia uma mensagem de confirmação ao utilizador que irá aparecer no seu ecrã; caso contrário, envia uma mensagem de erro e, em ambos os casos, o utilizador pode continuar a inserir comandos para enviar ao servidor.

Por outro lado, todos os comandos que o utilizador escrever na linha de comandos relativos ao próprio utilizador serão processados pelos métodos correspondentes na própria classe do cliente, não sendo necessário enviar informação nenhuma ao servidor. No entanto, o utilizador precisa de se autenticar para poder usar esses comandos locais, uma vez que estes só serão úteis para operações que necessitem que o servidor esteja ativo.

A pasta local dos clientes diz respeito à pasta onde é corrido o programa ***Client.java***, ou seja, se vários clientes correrem o programa na mesma pasta, então a pasta local será igual para todos, apesar de não ser partilhada. Ou seja, um cliente pode estar na pasta de raiz "." enquanto outro cliente pode estar na pasta "./Testes". Por outro lado, cada cliente tem a sua própria pasta no servidor, sendo previsto o caso de estes quererem aceder às pastas uns dos outros ou à pasta que contém a informação de cada cliente.

No que toca às tarefas de *download* e *upload*, estas funcionam de modo análogo. Ao receber o comando, o servidor irá criar uma nova socket TCP à qual o cliente que enviou o comando se irá ligar. Sendo estabelecida, são criados os canais que permitem a leitura da stream de input (um ficheiro no caso do download ou a socket no caso do upload), e a escrita na stream de output (o inverso). A transferência ocorre de forma faseada, distribuída ao longo de vários pacotes de tamanho máximo fixo. Sendo este processo feito sobre sockets TCP, temos garantia de receber todos os pacotes na ordem correta, o que facilita a reconstrução do pacote. Além disso, no caso do upload também ocorre posteriormente a cópia do novo ficheiro para o servidor secundário.

Em relação à sincronização do estado entre servidor primário e servidor secundário, tal responsabilidade é assente sobre as classes ***FileSyncSender.Java***, ***FileSyncReceiver.Java*** e ***FileReceiver.java***. O primeiro é chamado quando se pretende iniciar o envio de um ficheiro do primário para o secundário. É criada então uma socket UDP que envia para o outro servidor o

caminho do ficheiro novo em relação à raiz do servidor. Ao receber esta mensagem na sua socket, o ***FileSyncReceiver.java*** cria uma thread dedicada para aquela transferência em específico. Nesta thread é gerada uma resposta de *acknowledgement*, isto é, um ACK, que é enviada através de uma nova socket UDP para o ***FileSyncSender.java***. Este ao receber o ACK, envia o tamanho do ficheiro a ser enviado, aguardando por outro ACK antes de começar a enviar porções do ficheiro.

Como o UDP não tem garantias de entrega ou ordem, a cada datagrama enviado o servidor espera pelo um ACK seguido do número de sequência correto antes de enviar o próximo, reenviando o mesmo byte array caso contrário. Chegando ao final da transferência, o servidor primário envia um ACK ao secundário, aguardando por um ACK como resposta do secundário para dar a thread como terminada. O ACK que o secundário envia apenas ocorre caso o número de bytes recebidos corresponda ao tamanho do ficheiro, conforme havia sido enviado anteriormente.

3. Mecanismo de failover

Nesta situação, foi implementado um mecanismo de heartbeats que funciona da seguinte forma: cada servidor contém a thread, ***HeartbeatSender.java***, que envia heartbeats ao servidor primário através de uma socket UDP. Como o servidor primário não está ainda ativo no primeiro arranque do programa, ambos os servidores vão começar como secundários até o primeiro se tornar primário, fazendo com que o outro se mantenha secundário.

Caso um dos heartbeats não seja recebido, incrementa a variável ***failedheartbeats*** até o valor desta ser igual a 5. Caso o servidor secundário consiga receber uma resposta do servidor primário, então a contagem do ***failedheartbeats*** é reinicializada e o processo é repetido; caso contrário, se o ***failedheartbeats*** chegar ao valor 5, então o servidor secundário assume-se como servidor primário.

Esta incrementação existe para mostrar que o servidor primário não está a funcionar como é esperado, seja porque a ligação não está a ser bem efetuada, seja porque o servidor foi abaixo, seja por qualquer outro motivo que impeça o seu funcionamento normal. Nesse momento, aparece uma mensagem nos clientes que o servidor foi abaixo e/ou que está a tentar ser estabelecida uma conexão ao servidor. Quando o servidor secundário assume-se como servidor primário, o cliente fica ligado a esse servidor, prosseguindo-se então o normal funcionamento do programa.

A principal falha do nosso mecanismo é que, quando é feito o download ou upload de um ficheiro e o servidor for abaixo, esse mesmo download ou upload não irá ser retomado quando o cliente se conectar ao outro servidor, uma vez que não são guardados esses dados. Isto acontece também porque quando um servidor vai abaixo, o cliente deixa de estar autenticado, tendo que fazer novamente essa autenticação quando se encontra ligado ao novo servidor.

4. Distribuição das tarefas

Este trabalho foi dividido paralelamente em duas componentes principais: a escrita das funções para a realização das tarefas e os mecanismos de sincronização/ligação.

Um dos membros tratou de escrever as funções que dizem respeito às tarefas a serem realizadas, isto é, funções para listar ficheiros da diretoria remota, mudar de diretoria, alterar o conteúdo de certos ficheiros, etc, assim como testar o programa e verificar a presença de bugs ou resultados inesperados na execução de cada programa, como por exemplo, verificar se ao escrever o comando de download sem passar argumentos apresenta um erro.

O outro membro tratou de implementar a lógica por detrás das sockets TCP, usadas para a comunicação entre o servidor e os clientes que este serve, enviando comandos e respectivas respostas; e as UDP, utilizadas para o mecanismo de *failover* via *heartbeats* e consequente mudança de papel quando o primário não respondia, assim como para a replicação de ficheiros entre servidor primário e secundário.

No final, ambos os membros chegaram ao ponto da arquitetura UDP onde ambos implementaram a tarefa de gerir uma cópia do ficheiro que foi carregado para o servidor primário e transferi-lo via UDP para o servidor secundário, passando por fases de sincronização, testagem, entre outros.

5. Testes efetuados

Todos os testes que efetuamos possuem uma caixa colorida à esquerda. Uma caixa verde corresponde a um teste bem sucedido, com o resultado esperado, enquanto uma caixa vermelha corresponde a um teste mal sucedido, com o resultado contrário ao esperado.

- **Ligação aos servidores**

	Servidor primário liga e um cliente conecta-se a este.
	Servidor secundário liga e manda heartbeats ao servidor primário.
	Vários clientes conectam-se ao servidor e todas as ligações são aceites.
	Quando o servidor primário vai abaixo, o cliente perde a ligação mas o programa não acaba, tentando conectar-se ao servidor.
	Quando o servidor primário vai abaixo, o servidor secundário assume-se como primário e todos os clientes conectam-se a esse.

- **Autenticação**

"au <username> <password>", permite autenticar o utilizador.

	Inserir um nº de argumentos diferente de 3 apresenta mensagem de erro.
	Inserir o username ou password errados apresenta mensagem de erro.

	Se o utilizador já se encontrar autenticado, não se pode autenticar novamente e apresenta mensagem de erro.
	Inserir um comando, à exceção de "au <username> <password>", sem que o utilizador esteja autenticado apresenta mensagem de erro.
	Quando o utilizador se autentica, se não existir uma diretoria para esse, então criada no ficheiro.

- **Modificar password**

  "pw <old password> <new password>", permite modificar a password do utilizador.

	Inserir um nº de argumentos diferente de 3 apresenta mensagem de erro.
	Não efetua o comando se o utilizador não se encontrar autenticado, apresentando mensagem de erro.
	Inserir a antiga password errada não efetua alteração nenhuma e apresenta mensagem de erro.
	Quando o utilizador muda a password, este é desconectado do servidor e necessita de fazer novo login.
	A antiga password substituída pela nova password com sucesso.
	Quando é alterada a password, a diretoria remota atual do cliente é alterada para "/home".

- **Listagem de ficheiros**

  lss, permite listar ficheiros do servidor.

  lsc, permite listar ficheiros locais.

	Inserir um nº de argumentos diferente de 1 apresenta mensagem de erro.
	Não efetua o comando se o utilizador não se encontrar autenticado, apresentando mensagem de erro.
	lsc faz listagem dos ficheiros e pastas na diretoria local onde se encontra o cliente.
	lss faz listagem dos ficheiros e pastas na diretoria remota onde se encontra o cliente.
	Se a pasta se encontrar vazia, a listagem apresenta no ecrã "Empty".

- **Mudança de diretoria**

“cds <directory>”, permite mudar de diretoria no servidor.

“cdc <directory>”, permite mudar de diretoria localmente.

	Inserir um nº de argumentos diferente de 2 apresenta mensagem de erro.
	Não efetua o comando se o utilizador não se encontrar autenticado, apresentando mensagem de erro.
	Mudar para uma diretoria local/remota inexistente ou para um ficheiro que não seja diretoria apresenta mensagem de erro.
	Mudar para uma diretoria local/remota existente é bem sucedido (sendo case sensitive, caso contrário apresenta mensagem de erro).
	Mudar para a diretoria local/remota anterior é bem sucedido.
	Se o utilizador se encontrar na diretoria remota “/home”, não consegue mudar para a diretoria anterior.
	Se o utilizador se encontrar na diretoria local de raiz não consegue mudar para a diretoria “anterior”.

- **Download e upload de ficheiros**

“dn <filename>”, faz download de um ficheiro.

“up <filename>”, faz upload de um ficheiro.

	Inserir um nº de argumentos diferente de 2 apresenta mensagem de erro.
	Não efetua o comando se o utilizador não se encontrar autenticado, apresentando mensagem de erro.
	Se o ficheiro a ser descarregado não existir na diretoria remota atual apresenta mensagem de erro.
	Se o ficheiro a ser carregado não existir na diretoria local atual apresenta mensagem de erro.
	O ficheiro a ser descarregado da diretoria remota atual do cliente passa a existir na pasta atual local do cliente.
	O ficheiro a ser carregado da diretoria local atual do cliente passa a existir na diretoria remota atual do cliente.
	Se o servidor principal for a baixo e estiver a ser feito um download/upload, esta tarefa continua a ser feita no novo servidor primário.
	O servidor primário efetua uma cópia do ficheiro carregado para o servidor secundário.

	A cópia do ficheiro carregado para o servidor secundário encontra-se na sua íntegra.
	Se o servidor primário não conseguiu fazer a cópia inteira do ficheiro carregado para o servidor secundário, tenta novamente.

- **Outros**

	Se o utilizador inserir um comando inválido apresenta uma mensagem de erro, quer esteja autenticado ou não.
	Quando o utilizador muda de diretoria no servidor ou de password, a informação é modificada no ficheiro users.txt.