Faculdade de Ciências e Tecnologia
Universidade de Coimbra

Licenciatura em Engenharia Informática

Relatório Sistemas Distribuídos
ucDrive: Repositório de ficheiros na UC

Trabalho realizado por:

Miguel Ferreira | 2019214567 | PL5
Thomas Fresco | 2019219057 | PL5

Hugo Dinis Pereirinha da Silva Amaro
Filipe João Boavida Mendonça Machado de Araújo

2021/2022

Índice

Arquitetura do software

O código fonte do projeto encontra-se dividido entre 3 ficheiros *.java*: TCPClient.java, MainServer.java e BackupServer.java. Os últimos 2 ficheiros são muito semelhantes, diferindo apenas em alguns prints (meramente estéticos) e na definição de portos para alguns sockets.

Servidores (MainServer.java e BackupServer.java)

Classes usadas:

- **Class Queue -** A classe principal (MainServer/BackupServer) possui um objeto desta classe, uma estrutura auxiliar ao envio de ficheiros do Servidor Principal para o Secundário. Como se trata de um envio com recurso ao protocolo UDP e um servidor pode ter n clientes a enviar-lhe ficheiros ao mesmo tempo, este objeto atua como FIFO dos ficheiros a enviar para o secundário, sendo da responsabilidade de uma thread isolada "despachar" todos os ficheiros por ordem, evitando a sua corrupção.

    - **int front, rear -** posição de início e de fim da fila, respetivamente;
    - i**nt capacity -** capacidade da fila;
    - **String queue[] -** array que representa a fila propriamente dita.

- **Class MainServer/BackupServer -** Class que possui o método *main*. Lê a informação dos ficheiros *.txt* de configuração e avalia se se deve comportar como servidor principal ou secundário, seguindo com o comportamento escolhido.

    - **int serverPort -** porto onde é aberta a socket de troca de comandos entre os clientes e o servidor.

- **Class Connection -** É criado um objeto desta classe cada vez que um cliente se liga ao servidor. Usada para tratar todos os comandos vindos do cliente, os métodos desta classe fazem várias verificações e proteções, como por exemplo, verificar se o comando enviado é válido ou tem a estrutura correta. É dentro desta classe que são criados os objetos *FilesPipe*, usados para o envio e receção de ficheiros entre servidor e cliente.

    - **DataInputStream in -** usado para leitura de uma socket TCP;
    - **DataInputStream out .** usado para a escrita para uma socket TCP;
    - **Socket clientSocket -** variável usada para inicialização e identificação da socket;
    - **int thread_number -** contador para o número de clientes ligados ao servidor;
    - **Queue q -** variável usada para identificar e referenciar a Queue;
    - **Semaphore semaphorePass, semaphoreLog -** semáforos para controlar leituras e escritas nos ficheiros com os dados dos clientes e no ficheiro de log.

- **Class FilesPipe -** é criado um objeto desta classe sempre que é recebido, por parte do cliente, um comando válido do tipo *sendfile* ou *getfile*. Dependendo do tipo de comando, o objeto trata de receber ou de enviar o ficheiro em causa.

    - **InputStream in, OutputStream out -** representam o ficheiro a enviar/receber;
    - **String mode -** define se o servidor vai receber ou vai enviar um ficheiro;
    - **String file -** nome do ficheiro a enviar/receber;
    - **Socket clientSocket -** socket para troca de ficheiros.

Como os dois ficheiros *.java*, dependendo do contexto, podem correr tanto como principal ou secundário, as threads e sockets criadas/utilizadas dependem do comportamento assumido pelo servidor.

Quando funcionam como servidor principal:

Threads usadas:

- 1 thread para a receção e resposta aos pings enviados pelo secundário
- 1 thread para envio de ficheiros para o servidor secundário
- 1 thread por cliente para receção de comandos
- 1 thread por ordem de envio/receção de um ficheiro

Sockets usados:

- 1 socket TCP para o listen de comandos;
- 1 socket UDP para receção (e envio) de pings;
- 1 socket UDP para envio de ficheiros para o servidor secundário;
- 1 socket TCP por cada envio (ou receção) de ficheiros entre o servidor principal e um cliente.

Quando funcionam como servidor secundário:

Threads usada:

- 1 thread para receção de ficheiros do servidor principal

Sockets usados:

- 1 socket TCP por cada envio (ou receção) de ficheiros entre o servidor principal e o cliente
- 1 socket UDP para envio (e receção da resposta) de pings
- 1 socket UDP para a receção de ficheiros vindos do servidor principal

Cliente (TCPClient.java)

Classes usadas:

- **TCPClient** - única classe presente no ficheiro. Trata de ler os inputs do cliente, validar os comandos, ler o ficheiro de configuração com as informações sobre os servidores (endereços e portos). As credenciais de login são pedidas em sucessão dentro de um ciclo até que o cliente efetue o login com sucesso. Tenta ligar ao servidor de backup caso o atual deixe de funcionar e, caso este não responda, termina o programa.

Threads usadas:
- 1 Thread para o envio de ficheiros para o servidor, criada apenas quando necessário
- 1 Thread para a receção de ficheiros do servidor, criada apenas quando necessário

Sockets usados:
- 1 Socket para o envio de ficheiros para o servidor, aberta apenas quando necessário
- 1 Socket para a receção de ficheiros do servidor, aberta apenas quando necessário

Funcionamento do Servidor

Quando o servidor inicia, é chamada a função *startUp()*, cujo modo de funcionamento é descrito na secção de *Failover*. Além disto é criada uma queue para controlar o envio de ficheiros para o servidor secundário e são inicializados dois semáforos, que serão usados para controlar leituras e escritas nos ficheiros com os dados dos clientes e no ficheiro de log.

Caso esta função retorne *True* significa que já existe outro servidor ligado, e que este deve tomar o comportamento de servidor de Backup, inicializando uma thread que vai receber ficheiros do servidor principal assim como a função *heartbeat()*, que envia pings para o servidor principal e cujo funcionamento está descrito na secção de *Failover*.

Caso a função *startUp()* retorne *False*, o que significa que não há outro servidor ligado, este servidor vai tomar o comportamento de servidor principal, chamando a função *recebeHeartbeat()*, que vai ficar à espera que um servidor secundário envie pings e cujo funcionamento também está descrito na secção de *Failover*. É chamada a função *readData()* que lê o ficheiro *data.txt* e cria uma diretoria para cada utilizador registado, caso ela ainda não exista. Depois é criada outra thread e dentro desta é chamada a função *enviaFile()*, que vai ficar à espera que um ou mais ficheiros sejam adicionados à queue para, de seguida o enviar para o servidor secundário.

Seguidamente, cria um socket para escuta de comandos e, por cada cliente que é aceite nesse porto, é criado um novo objeto *Connection* e é lançada uma nova *thread*, que irá tratar de analisar os vários comandos recebidos e satisfazer as necessidades de cada cliente. Sempre que um cliente envia um comando do tipo “sendfile” ou “getfile”, o servidor cria pontualmente uma nova socket e uma thread para tratar da receção (ou envio) do ficheiro identificado, sendo o seu path adicionado à queue de ficheiros a enviar para o secundário.

Caso exista um servidor a atuar como secundário, existe uma thread isolada que irá tratar de retirar da queue o ficheiro adicionado e enviar para o servidor de *backup*, como acima referido.

Funcionamento do Cliente

O cliente começa por se tentar conectar ao servidor principal. Se o principal não estiver ligado, tenta ligar-se ao secundário. Se ambos estiverem desligados, o cliente fecha. Se se conseguir conectar a um dos servidores, irá mostrar "Insira o nome de utilizador e a palavra-passe:". Aí, deve-se inserir um nome de utilizador válido, assim como a password associada, da seguinte forma: "<utilizador> <password>". Após o login bem sucedido, é mostrada a mensagem "Login efetuado com sucesso!" e a consola é "desbloqueada", sendo agora possível enviar comandos para o servidor. Os comandos permitidos são os seguintes:

- **sls ->** listar ficheiros da diretoria atual no servidor
- **sls -all ->** lista todos os ficheiros dentro da pasta do utilizador no servidor
- **scd ->** mostrar o path atual no servidor
- **scd <path> ->** ir para a diretoria <path> no servidor
- **scd . ->** ir para a diretoria "home" da pasta do cliente no servidor
- **ls ->** listar ficheiros da diretoria atual no cliente
- **ls -all ->** lista todos os ficheiros dentro da pasta do utilizador no cliente
- **cd ->** mostrar o path atual no cliente
- **cd <path> ->** ir para a diretoria <path> no cliente
- **cd . ->** ir para a diretoria "home" da pasta do cliente no cliente
- **ps -set <password> ->** alterar password
- **main -set <endereço> <porto> ->** definir endereço IP e porto para o servidor principal
- **backup -set <endereço> <porto> ->** definir endereço IP e porto para o servidor secundário
- **main ->** mostra endereço IP e porto para o servidor principal
- **backup ->** mostra endereço IP e porto para o servidor secundário
- **sendfile <nome> ->** envia uma copia de um ficheiro da diretoria atual do cliente para a diretoria atual do servidor.
- **getfile <nome> ->** envia uma copia de um ficheiro da diretoria atual do servidor para a diretoria atual do cliente.

Mecanismo de *Failover*

Ambos os servidores têm o código fonte igual, diferindo apenas na definição das portas com as quais os servidores vão abrir os seus sockets e em alguns prints (meramente estéticos).

O servidor é iniciado com a função *startUp()*. Este método lê o ficheiro *config.txt* e guarda o IP e porto do outro servidor. Guarda também o número de pings necessários para considerar que o outro servidor falhou e o tempo de espera entre cada um. Definiu-se cada um destes pings como o envio de uma string "*ping*" e a receção de uma resposta "*pong*". De seguida, este servidor tenta contactar o outro usando o protocolo UDP, tendo como limite de tentativas falhadas o número de pings acima referido.

Se nenhum desses pings receber resposta, o servidor em causa vai assumir que nenhum outro está ligado, tomando o comportamento de servidor principal. Aí, inicializa a função *recebeHearbeat()*, que vai abrir uma thread e ficar à espera que um servidor de backup envie pings. Depois, segue o comportamento normal de um servidor principal.

Se em qualquer um destes pings ele obtiver uma resposta, o servidor vai assumir que já existe outro ligado, tomando o comportamento de servidor de backup, iniciando a função *heartbeat()*. Esta função vai ler o ficheiro "*config.txt"*, que de forma análoga à função *recebeHeartbeat()*, vai ler o IP e porto do outro servidor, assim como o número máximo de pings que podem falhar e o tempo que os separa. De seguida, vai começar a enviar pings e a receber resposta do servidor principal. Se, durante este processo, 5 pings seguidos não obtiverem resposta, este servidor assume que o outro foi desligado e vai tomar o comportamento de servidor principal descrito acima.

Do lado do cliente, quando a conexão a um dos servidores fecha, é recebida uma exceção. Nessa exceção, os valores das variáveis que indicam o IP e porto do servidor atual são trocados com os valores as variáveis que indicam os do secundário. Após este processo, é tentada uma nova conexão e, se esta não for bem sucedida, o cliente admite que nenhum dos servidores está ligado e desliga-se. O scanner para ler o input do cliente é criado no *main*, de forma a evitar o fecho do *System.in*, que acontecia quando o cliente entrava na exceção referida acima.

Distribuição das Tarefas

- M**iguel Ferreira -** especial atenção ao Failover e ao tratamento de exceções.
- **Thomas Fresco -** especial atenção à aplicação cliente e ao envio de ficheiros.

No entanto, a maior parte do projeto foi desenvolvida em conjunto, com recurso a pair programing e a um repositório partilhado no GitHub.

Testes Feitos à Plataforma

NOTA: O código não funciona com diretorias/ficheiros cujo nome apresenta o caracter " " (espaço).

Teste	Comp. esperado	Comp. Observado	pass / fail
Login com credenciais válidas	Mensagem “Login efetuado com sucesso!”	Mensagem “Login efetuado com sucesso!”	
Login com palavra-passe inválida	Mensagem “Password invalida para este username!”	Mensagem “Password invalida para este username!”	
Login com username não registado	Mensagem “Username nao existe!"	Mensagem “Username nao existe!"	
Comando com o número errado de argumentos	Mensagem “Comando Inválido!”	Mensagem “Comando Inválido!”	
Inserção de um comando enquanto o servidor principal está desligado e o de backup ligado	Mensagem “Servidor Indisponível! Trocando de servidor...”	Mensagem “Servidor Indisponível! Trocando de servidor...”	
Listagem dos ficheiros da diretoria atual do servidor	Lista dos ficheiros e das diretorias do servidor	Lista dos ficheiros e das diretorias do servidor	
Listagem de todos os ficheiros dentro da pasta do utilizador no servidor	Lista dos ficheiros e das diretorias do servidor	Lista dos ficheiros e das diretorias do servidor	
Mostrar o path atual no servidor	Mensagem com o path atual	Mensagem com o path atual	
Mudar de diretoria no servidor	Mensagem com o path da nova diretoria	Mensagem com o path da nova diretoria	
Mudar para a diretoria “home” no servidor	Mensagem com o path da diretoria “home”	Mensagem com o path da diretoria “home”	
Listagem dos ficheiros na diretoria atual do cliente	Lista dos ficheiros e das diretorias do cliente	Lista dos ficheiros e das diretorias do cliente	

Listagem de todos os ficheiros dentro da pasta do utilizador do cliente	**Lista de todos os ficheiros, pastas, subpastas e ficheiros contidos nestas subpastas**	**Lista de todos os ficheiros, pastas, subpastas e ficheiros contidos nestas subpastas**	
Mostrar o path atual no cliente	**Mensagem com o path atual**	**Mensagem com o path atual**	
Mudar o path atual no cliente	**Mensagem com o path da nova diretoria**	**Mensagem com o path da nova diretoria**	
Mudar para a diretoria “home” no cliente	**Mensagem com o path da diretoria “home”**	**Mensagem com o path da diretoria “home”**	
Mudar a palavra passe	**Mensagem “Password atualizada!” e pedido para fazer login de novo**	**Mensagem “Password atualizada!” e pedido para fazer login de novo**	
Mudar o IP e porto do servidor principal	**Mudança do IP e do porto no ficheiro “config.txt” na pasta do cliente**	**Mudança do IP e do porto no ficheiro “config.txt” na pasta do cliente**	
Mudar o IP e porto do servidor secundário	**Mudança do IP e do porto no ficheiro “config.txt” na pasta do cliente**	**Mudança do IP e do porto no ficheiro “config.txt” na pasta do cliente**	
Mostrar o IP e porto do servidor principal	**Mensagem com o IP e porto do servidor principal**	**Mensagem com o IP e porto do servidor principal**	
Mostrar o IP e porto do servidor secundário	**Mensagem com o IP e porto do servidor secundário**	**Mensagem com o IP e porto do servidor secundário**	
Envio de ficheiros para o servidor quando nenhuma das diretorias contém o caracter “ ” (espaço)	**Mensagem “A enviar teste.txt para <diretoria> Ficheiro enviado: <diretoria com o ficheiro>”**	**Mensagem “A enviar teste.txt para <diretoria> Ficheiro enviado: <diretoria com o ficheiro>**	
Download de ficheiros para o servidor quando nenhuma das diretorias contém o caracter	**Mensagem “A transferir ficheiro <nome> para <diretoria> Ficheiro recebido: <diretoria com o ficheiro>”**	**Mensagem “A transferir ficheiro <nome> para <diretoria> Ficheiro recebido: <diretoria com o ficheiro>”**	
Envio de ficheiros quando uma das diretorias contém o	**Envio normal do ficheiro**	**Erro e crash no terminal do cliente. o servidor também deixa de**	

caracter “ ” (espaço)		**funcionar.**	
Download de ficheiros quando uma das diretorias contém o caracter “ ” (espaço)	**Download normal do ficheiro**	**Erro e crash no terminal do cliente. o servidor também deixa de funcionar.**	

Referências Bibliográficas

S. Jon, disponível a 20 de março em: https://stackoverflow.com/questions/13910512/passing-parameter-to-java-thread
javatpoint, disponível a 25 de março em: https://www.javatpoint.com/java-queue
WhiteFang34, disponível a 25 de março em: https://stackoverflow.com/questions/6099636/sending-files-through-sockets
A. Alvin, disponível a 25 de março em: https://alvinalexander.com/blog/post/java/how-set-timeout-time-out-java-socket-client/
G. Martin, disponível a 26 de março em: https://stackoverflow.com/questions/9941296/how-do-i-make-a-jar-from-a-java-file
Oracle, disponível a 26 de março em: https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Semaphore.html
M. Sam, disponível a 30 de março em: https://www.baeldung.com/java-checksums
user1605892, disponível a 2 de abril em: https://stackoverflow.com/questions/12023490/inner-classes-not-being-included-in-jar-file