UNIVERSIDADE DE COIMBRA

SD – ucDrive

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D COIMBRA

2019216646 – António Correia
2019216764 – João Monteiro
2019216809 – Miguel Faria

Índice

Introdução 3
Arquitetura 4
Armazenamento de dados 4
Acesso sincronizado aos ficheiros 5
Funcionamento da consola Cliente 5
Ligação TCP entre clientes e servidor primário 6
Ligação UDP entre servidores para replicação (backup) de dados 6
Ligação UDP entre servidores para realização de sistema de Failover (HearBeat) 7
Ligação UDP do servidor secundário para fins de verificação de integridade 7
Ligação RMI dos Admins 8
Funcionamento UcDrive 8
Distribuição de tarefas 8
Testes 9
Conclusão 12
Bibliografia 13

Introdução

Este projeto tem por objetivo desenvolver e adquirir conhecimentos sobre protocolos TCP e UDP, sincronização de *Threads*, conexão RMI, com recurso à linguagem de programação Java. Foi desenvolvida a aplicação *UcDrive*, que pretende simular uma plataforma para que clientes possam guardar e transferir ficheiros num servidor remoto.

O seguinte relatório tem por objetivo identificar e explicar as funcionalidades desenvolvidas, bem como a utilização da plataforma. Está presente também as tarefas desenvolvidas com os respetivos autores e por fim existe identificados os testes realizados à plataforma bem como os resultados obtidos.

Arquitetura

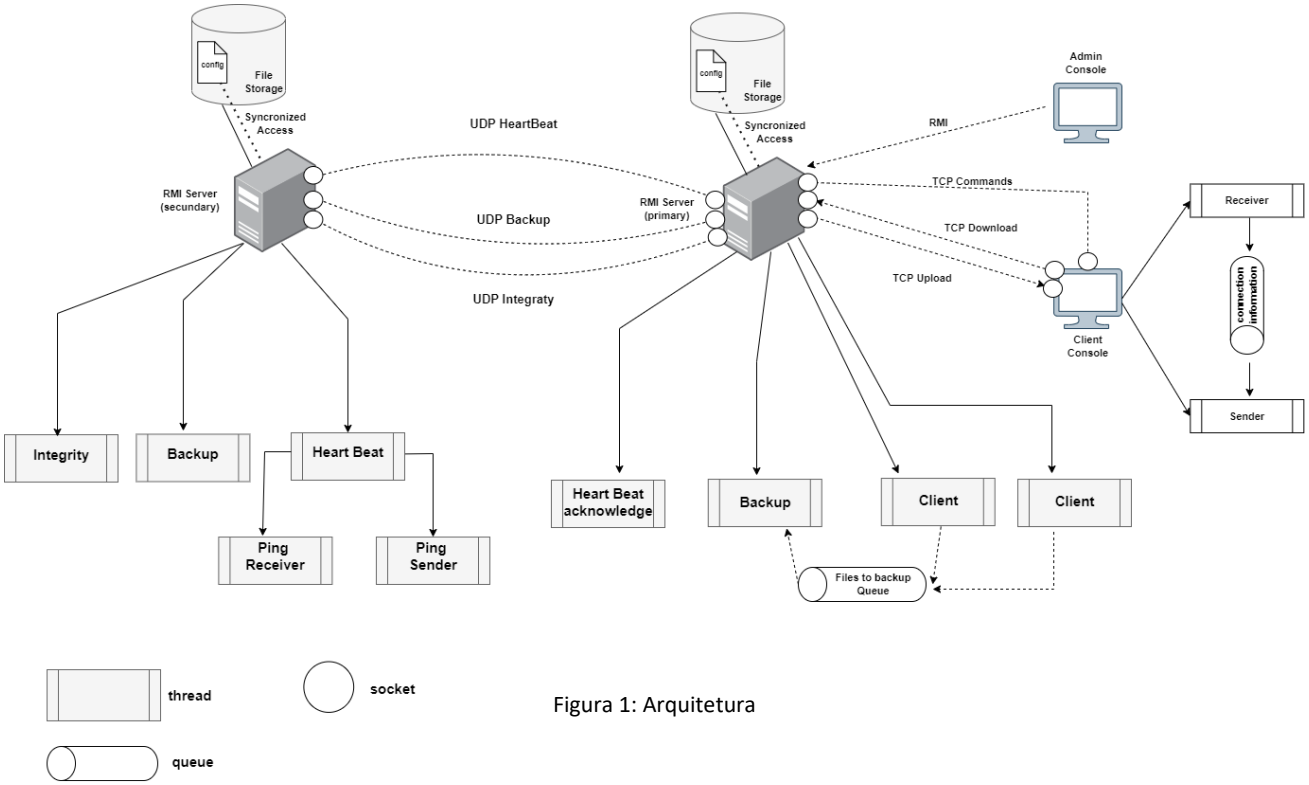

Figura 1: Arquitetura

De uma maneira geral, a arquitetura do sistema consiste seguintes componentes:

- Dois servidores, onde um realiza todas as funcionalidades e pedidos pelo cliente (primário) e outro onde é feito backup dos ficheiros, verificação do estado do servidor que faz a conexão ao cliente.
- Clientes, que através da sua consola se conectam ao servidor para realizar operações
- Admins, que, usando uma consola com suporte por RMI, conseguem visualizar o estado dos servidores e alterar os seus dados.

Armazenamento de dados

Os servidores e clientes possuem uma estrutura própria para o armazenamento de dados. Tanto nos servidores como nos clientes assume-se que existe uma pasta *home* criada previamente, que irá conter toda a árvore de ficheiros de cada um desses intervenientes.

Nos servidores, dentro desta pasta, para além das diretorias dedicadas a cada cliente, terão que existir dois ficheiros: *config.txt,* que irá possuir toda a informação necessária para um funcionamento

correto do servidor, e *clients.txt,* que possuirá informações de cada cliente, tanto para login do mesmo como informação pessoal. Estes ficheiros devem obedecer à seguinte estrutura:

```
testuser1 / password1 / testuser1/home / faculty1 / address1 / number1 / cc1
testuser2 / password2 / testuser2/home / faculty2 / address2 / number2 / cc2
testuser3 / password3 / testuser3/home / faculty3 / address3 / number3 / cc3
```

Figura 2: Ficheiro clientes.txt

```
// THIS SERVER
IP: 127.0.0.1
PORT: 1234
HEARTBEAT PORT: 5555
BACKUP PORT: 8888
// OTHER SERVER
IP: 127.0.0.1
HEARTBEAT PORT: 6666
BACKUP PORT: 9999
// HEARTBEAT CONFIG
DELAY: 1000
MAX FAILED: 5
HIERACHY: 1
// INTEGRITY
OURPORT: 4444
OTHERPORT: 3333
```

Figura 3: Ficheiro *config.txt* Servidor 1

```
// THIS SERVER
IP: 127.0.0.1
PORT: 4321
HEARTBEAT PORT: 6666
BACKUP PORT: 9999
// OTHER SERVER
IP: 127.0.0.1
HEARTBEAT PORT: 5555
BACKUP PORT: 8888
// HEARTBEAT CONFIG
DELAY: 1000
MAX FAILED: 5
HIERACHY: 2
// INTEGRITY
OURPORT: 3333
OTHERPORT: 4444
```

Figura 4: Ficheiro *config.txt* Servidor 2

Os valores presentes relativos aos ficheiros *config.txt* presentes nas figuras a cima podem ser manipulados, contudo devem manter a associação que existe entre os dois ficheiros.

Acesso sincronizado aos ficheiros

O acesso sincronizado aos ficheiros é feito para garantir que não ocorre concorrência entre acessos a clientes.txt e config.txt. Todas as funções de e acesso estão dentro de uma *Class* onde todos os métodos possuem a *syncronized keyword*. Deste modo pode se garantir que uma *thread* de comunicação com cliente não lê o ficheiro com informação do mesmo ao mesmo tempo que outro, e que um *admin* não elimina alterações feitas por um outro que esteja a aceder ao mesmo tempo.

Funcionamento da consola Cliente

A consola do cliente, após estabelecer a conexão ao servidor, irá originar duas *threads*, *Receiver* e *Sender*, as quais irão receber mensagens do servidor e enviar comandos para o mesmo, respetivamente. Foi utilizada esta estrutura com a finalidade de evitar ter de controlar a quantidade de receções e envios de mensagens e ordem das mesmas. Deste modo o servidor pode enviar várias mensagens seguidas. Uma *queue* é utilizada entre estas *threads* com objetivo de comunicar para determinar as ações que devem ser executadas em casos onde: se perca a conexão ao servidor, seja

necessário reintroduzir dados de login após uma alteração de password (onde o cliente automaticamente reconecta por TCP) ou em que o cliente queira terminar a conexão. Estas comunicações indicarão se as *threads* devem terminar e as ações que executam a alteração de variáveis que determinarão como deve proceder a consola.

Ligação TCP entre clientes e servidor primário

As ligações entre clientes e servidor são feitas através do protocolo TCP. O servidor, após cada conexão, irá criar uma *socket* e uma *thread* destinada à comunicação com o cliente conectado. Nesta *socket* são recebidos todos os comandos escritos pelo cliente e enviadas ações que possam ser necessárias executar pela consola do utilizador. Na situação de download e upload, a transferência de ficheiros é feita a partir de uma outra *socket* num porto disponível. Esta está apenas aberta durante o processo de transferência; caso outro pedido seja feito, uma nova *socket* num outro porto será criada.

Ligação UDP entre servidores para replicação (backup) de dados

Com o intuito de realizar a replicação de dados do servidor primário para o secundário, foi implementada uma ligação UDP entre eles. Deste modo, o servidor primário irá possuir uma *thread* que permitirá enviar ficheiros/pastas, enquanto o secundário irá ter uma *thread* para os receber.

De maneira a saber quais os ficheiros que devem ser enviados para backup, estes são selecionados com o uso de uma *queue* (que corresponde na Figura 1 a "Files to backup *queue*"), onde sempre que é realizada uma alteração no ficheiro "clientes.txt" ou é acrescentado um ficheiro/pasta, é colocado nessa *queue* o tipo de ficheiro bem como a *path* a partir da diretoria *home* do server. Assim, a *thread* de envio fica bloqueada até algo ser colocado na fila (devido ao facto de ser uma *Blocking Queue*).

Para o envio de qualquer tipo de dado por UDP é tido em conta o seguinte princípio: sempre que é enviado um pacote, a *thread* que realizou o envio fica em espera até receber de volta uma confirmação da outra *thread* de que o pacote foi recebido corretamente. Isto é conseguido devido ao uso de bytes de controlo no início de cada um deles: quando se pretende enviar uma *path* o primeiro byte é definido para garantir a integridade do pacote; quando se pretende enviar um ficheiro este é subdividido em pacotes onde o primeiro byte assegura a sua integridade, o segundo a sua ordem e o terceiro se o EOF foi alcançado.

Deste modo, para o envio de um ficheiro, é inicialmente enviado uma *string* contendo o seu tipo e a sua path no servidor (que irá ter relevância na parte de receção, para saber se inicializa um ficheiro e o processo de receção dos seus sub-pacotes ou se cria uma pasta com a path desejada). Caso não seja uma diretoria, após receber confirmação, esse ficheiro é transformado para um *byte array*, decomposto em sub-pacotes e o seu envio é também inicializado.

Relativamente à receção de ficheiros, a *thread* correspondente abre a sua *socket* durante o dobro do tempo do número máximo de *pings* que podem ocorrer sem resposta por parte da *thread Heartbeat*.

Caso este tempo se esgote sem ter recebido nenhum pacote, é verificado se o *Heartbeat* ainda está ativo: caso esteja, a *thread* de receção volta a repetir o processo; caso contrário a *thread* de *backup* também termina.

Ligação UDP entre servidores para realização de sistema de Failover (HearBeat)

A funcionalidade de *Failover* (*HeartBeat*) foi criada com a finalidade de no ocorrer de uma falha no servidor com estatuto primário, o secundário possuir capacidade de detetar esta anomalia e assumir o estatuto de primário.

Ao iniciar um servidor este é sempre iniciado no seu modo *default*, sendo este o de secundário. Neste modo, o servidor inicia a *thread HeartBeat*, servindo de controlo para outras *threads: Integrity* e *Backup*. Estas mantêm acessos periódicos a esta com o objetivo de saber se esta terminou; se for o caso, isto significa que este servidor deve assumir o estatuto principal, terminando-as. A *thread* de *HeartBeat* origina duas *threads* que são *started* e *joined* na mesma. *Ping Receiver* e *Ping Sender*, irão executar as ações de enviar e receber *pings* respetivamente, onde ao enviar um *ping* é decrementa o valor máximo estabelecido para o envio sem resposta do mesmo, e ao receber, este valor é colocado de volta ao seu valor máximo. Se não ocorrer confirmações pelo outro servidor, ambas as *threads* terminam, de seguida termina o *Heartbeat*, e por consequência, como mencionado a cima, termina *Integrity* e *Backup*, passando o servidor ao estatuto de primário.

O servidor que tiver o estatuto principal possui também a sua própria *thread* de *HeartBeat*, a qual confirma o *ping* recebido, devolvendo a mesma mensagem.

Na situação em que ambos os servidores se encontram no modo de *backup*, para determinar qual é mantido neste modo e qual é passado para principal, foi convencionado que o que cada servidor envia como *ping* é o valor estabelecido no ficheiro de configurações correspondente à hierarquia. O servidor que possuir menor valor de hierarquia irá passar a primário.

Ligação UDP do servidor secundário para fins de verificação de integridade

Com o intuito de validar a replicação de dados entre os servidores, foi realizada uma funcionalidade que permite verificar quais os elementos da árvore de ficheiros do servidor principal não se encontram no secundário. Para isto, foi usada uma ligação UDP, semelhante à de realização de backup de ficheiros, onde apenas é enviada a path de um ficheiro (por parte da thread gerada automaticamente pela ligação RMI do admin) e recebido se o ficheiro com essa path existe no servidor secundário (com essa verificação e envio a ser realizado por parte da thread iniciada no servidor de backup)

Ligação RMI dos Admins

Para estabelecer uma ligação RMI entre o terminal de administrador e o servidor foi utilizada a biblioteca *java.rmi*, a qual cria automaticamente uma *thread* e uma *socket* para poder comunicar sem interromper qualquer outro processo. Com esta biblioteca foi criado um *registry* na porta 7001 com o *lookup* "*admin*". A partir desta ligação é possível aceder à função *adminCommandHandler* do lado do servidor, a qual interpreta os comandos de administrador e devolve uma resposta adequada, sendo esta, também através do RMI, passada para o terminal de administrador.

Funcionamento UcDrive

Para utilizar a plataforma ucDrive deverão ser inicializados os dois servidores, não interessando a ordem uma vez que através das funcionalidades descritas anteriormente, estes conseguem determinar qual é que fica como primário e secundário. Após isto, vários clientes poderão se conectar. Ao conectar o cliente deve indicar o ip e porto do servidor, sendo recusado conexão ao servidor secundário. Uma vez conectado pode realizar todas as operações, em caso de dúvida o comando *help* indicará os comandos possíveis.

Distribuição de tarefas

Tarefa	Realizado por:	Auxiliado por:
Conexão TCP cliente	António Correia, João Monteiro, Miguel Faria	---
Login Cliente	António Correia, João Monteiro, Miguel Faria	---
Tratamento de Comandos	António Correia, João Monteiro, Miguel Faria	---
Download/Upload ficheiros por outra socket	João Monteiro	---
Consola Cliente	João Monteiro	---
Tratamento de comandos do lado do cliente	João Monteiro	---

Consola 'local' do cliente que para realizar upload	Miguel Faria	
Funcionalidade de failover	João Monteiro	Miguel Faria
Funcionalidade de backup	Miguel Faria	João Monteiro
Consola Admin	António Correia	---
Funcionalidade RMI número 5, integridade usando UDP	António Correia, Miguel Faria	---
Acesso sincronizado a ficheiros	António Correia, João Monteiro, Miguel Faria	---

Testes

Teste	Pass/Fail	Resultado
Registar utilizador	Pass	O cliente é registado com sucesso, sendo a sua conta duplicada para o server secundário, é criada a pasta do mesmo e a sua *home.*
Admin Listar diretorias/ficheiros por utilizador	Pass	A árvore de ficheiros é apresentada com sucesso.
Configurar o mecanismo de failover	Pass	O admin altera com sucesso os valores.
Listar detalhes de armazenamento	Pass	É apresentado com sucesso o valor em bytes na totalidade e por user.
Validar replicação de dados	Pass	Uploads, Downloads e alterações por parte de admin são replicadas.
Autenticação do Cliente	Pass	Cliente autentica-se com sucesso sendo necessário indicar o username e password certos para entrar.

Alterar password	Pass	A password é alterada, replicada entre servers, e a conexão é restabelecida de modo a pedir de novo dados de autenticação.
Cliente configurar endereço onde se conecta	Pass	Ao iniciar a consola o cliente consegue indicar o ip e porto de onde se pretende conectar, bem como quando o servidor crasha.
Listar ficheiros na diretoria atual do servidor	Pass	Todos os ficheiros e pastas são apresentados.
Mudar de diretoria	Pass	Muda de diretoria com sucesso sendo esta guardada e replicada.
Listar ficheiros locais	Pass	Todos os ficheiros são apresentados.
Mudar diretoria local	Pass	Muda com sucesso quando o modo é local.
Download de ficheiro	Pass	Download ocorre com sucesso mesmo quando a pasta definida para download não existe
Upload de ficheiro	Pass	Upload ocorre com sucesso mesmo quando a pasta definida para download não existe
Failover quando um dos servidores crasha	Pass	Servidor secundário inicia como primário quando o primeiro crasha
Backup de ficheiros	Pass/Fail	Ficheiros são duplicados com sucesso quando inferiores a 65MB
Guardar diretoria atual em caso de erro ou ao sair o cliente.	Pass	Diretoria está presente em ambos os servers.
Cliente ao perder a conexão receber o erro devido bem como	Pass	Ao perder conexão, após tentar enviar um comando é notificado do erro de conexão e é recomendado a ligar ao servidor secundário.

hipótese para nova conexão		
Terminar servidor a meio de um backup	Pass/Fail	O ficheiro é perdido na replicação, contudo é eliminado o ficheiro corrompido.
Terminar Server a meio de um upload	Fail	Cliente não recebe notificação que o da falha, mas recebe mais tarde que perdeu conexão.
Terminar Cliente a meio de um upload	Pass	É eliminado o ficheiro corrompido
Terminar Server a meio de um download	Pass	É eliminado o ficheiro corrompido
Terminar Cliente a meio de um download	Pass/Fail	O ficheiro fica corrompido. O server notifica que um erro ocorreu.
Registar Clientes com o mesmo username	Pass	Notificação ao admin que este username já existe.
Terminar servidor enquanto cliente esta no menu local	Pass	Ao perder conexão, após tentar enviar um comando não local é notificado do erro de conexão e é recomendado a ligar ao servidor secundário.
Upload de ficheiro que não existe	Pass	Notificação que ficheiro não existe.
Download de ficheiro que não existe	Pass	Notificação que ficheiro não existe.
Terminar servidor primário com ficheiros na queue por enviar para serem replicados	Fail	Se existirem ficheiros na queue do servidor primário e este crashar antes de os enviar, não é feita a replicação dos dados.

Conclusão

Com este projeto foi possível desenvolver e adquirir diversos conhecimentos, sobre a matéria mencionada. Através de ligações TCP foi possível a fácil e eficaz transferência de ficheiros, enquanto com o uso de ligações UDP esta transferência foi bastante mais complicada, sendo prejudicial a sua aplicação desta forma para o comportamento pretendido. Além disso, foi constatada a importância de sincronização entre *threads*, por exemplo no uso de ficheiros em várias instâncias.

Bibliografia

https://heptadecane.medium.com/file-transfer-via-java-sockets-e8d4f30703a5

https://gist.github.com/absalomhr/ce11c2e43df517b2571b1dfc9bc9b487