SD

Repositório de ficheiros: ucDrive

Relatório

PL5

Departamento de Engenharia Informática

Realizado por:

Edgar Duarte 2019216077 edgarduarte@student.dei.uc.pt

Miguel Barroso 2019219010 miguelbarroso@student.dei.uc.pt

1. Introdução

O seguinte trabalho foi desenvolvido com o intuito de criar uma plataforma cuja funcionalidade principal é o armazenamento permanente de ficheiros num servidor principal, sendo fundamental que nada se perca caso este servidor principal falhe. Para isto, existe um servidor secundário que serve de backup que, uma vez que o servidor primário falhe, assume o papel de servidor primário. Assim, os clientes, que se ligam aos servidores por TCP, uma vez que o servidor primário caia, vão-se ligar automaticamente ao servidor secundário.

2. Arquitetura

A arquitetura implementada foi a pedida no enunciado do projeto. Temos um servidor primário que tem ligações em multicanal via TCP com clientes (cada cliente gera uma *thread* no servidor primário). Um canal serve para o cliente enviar comandos ao server enquanto o outro serve para enviar dados (ficheiros). O cliente não contém *threads*.

Existe ainda um servidor secundário cuja função é servir de backup para o servidor primário. Quando se inicia o server, cria-se uma *thread* para a espera de ficheiros vindos do servidor primário enquanto o processo pai fica a enviar *heartbeats* para o servidor principal de forma a verificar se este ainda está a correr. Se houver perda de *heartbeats,* o servidor secundário assume que o primário falhou e assume o papel de servidor primário. A *thread* fica à espera de ficheiros vindos do servidor primário. Para isto, conecta-se ao servidor primário via UDP. Sempre que o servidor primário receber um novo ficheiro, este adiciona-o a uma *queue* de ficheiros para enviar ao servidor secundário. Depois, o servidor primário tem uma *thread* que a cada 5 segundos verifica se existe um novo ficheiro na *queue* para enviar. Se existir, envia o ficheiro via UDP para o servidor secundário. Este recebe-o em blocos e no final é verificado se o *checksum* do ficheiro é idêntico ao ficheiro inicial. Se não for ou se existir uma perda de pacotes a meio da transmissão, o ficheiro é novamente adicionado ao fim da queue, ou seja, vai ser reenviado.

O servidor que está a servir de servidor principal tem ainda acesso ao ficheiro de configurações de forma a conseguir autenticar novos clientes. O ficheiro de configurações é um ficheiro *JSON* onde cada objeto tem o nome de utilizador, a palavra-passe, o seu id único e a última pasta a que acedeu no server.

3. Funcionamento do servidor ucDrive

O processo cliente comunica com o server por via de *strings* no formato *numero_funcionalidade#var1#var2#...* .

Primeiramente, um cliente precisa de se autenticar. Para se confirmar a autenticação de um cliente, o processo cliente envia ao servidor uma *string* que lhe diz se o cliente quer dar *login* ou se quer dar *sign up*, lhe dá o nome de utilizador e a palavra-passe. O servidor verifica se os dados são aceitáveis e responde ao processo cliente com um booleano que diz se o cliente tem autenticação válida (*true* significa que a autenticação foi válida).

O servidor oferece 7 funcionalidades aos clientes autenticados:

1. Mudar a palavra-passe: para mudar a palavra-passe o processo cliente envia uma *string* ao servidor com o nome de utilizador e a palavra-passe. Com essa informação o servidor atualiza a palavra-passe no ficheiro de configuração.
2. Mudar a diretoria local e mudar diretoria do servidor: para mudar a diretoria local, o processo cliente não precisa de enviar informação ao servidor visto que esta nunca é utilizada pelo mesmo. Já para mudar a diretoria do servidor o processo cliente envia uma *string* com o nome de utilizador, uma vez que, o server, uma vez que mude a diretoria onde o cliente está, vai guardar esse caminho no ficheiro de configuração no objeto do cliente para que, numa sessão futura, o cliente entre imediatamente na última diretoria acedida na sessão anterior.
3. Listar ficheiros locais e listar ficheiros do servidor: para listar os ficheiros locais, o processo cliente não envia informação ao servidor. Para listar ficheiros no servidor, o processo cliente envia uma *string* apenas para informar o servidor da opção escolhida (não existem variáveis). O servidor devolve uma *string* com todos os ficheiros e diretorias na diretoria atual do servidor.
4. Upload de ficheiros para o servidor: para dar *upload* de um ficheiro para o servidor o cliente primeiro envia uma *string* para avisar o servidor da opção escolhida pelo cliente para este preparar uma nova socket para envio de dados. O servidor envia ao cliente a porta à qual ele tem de se ligar. O cliente liga-se a essa porta e envia o ficheiro num *buffer* na sua totalidade para o servidor. O servidor depois vai lendo o pacote enviado pelo cliente em blocos de 8Kb – notemos que a leitura é que é feita em blocos e não a sua transmissão manual visto que o TCP já faz o envio por blocos com respetivos ACKs e checksums. Quando o ficheiro tiver sido lido por completo o cliente e o servidor fecham os *sockets* e voltam a aceitar comandos.
5. Download de ficheiros do servidor: o processo de dar download de um ficheiro do servidor é análogo ao de *upload,* mas espelhado, isto é, em vez de o cliente enviar o ficheiro é o servidor. Assim, tal como no upload, primeiramente o cliente envia uma *string* com a opção escolhida e o servidor abre um *socket* para comunicação de dados. Agora o servidor envia o ficheiro inteiro num *buffer* e o cliente vai lendo-o em blocos de 8Kb.

4. Failover

Para garantir ter sempre um servidor funcional para clientes foi necessário implementar mecanismos de *failover*.

Quando a aplicação inicia, o server secundário assume o papel de servidor de backup. Assim, quando um ficheiro é enviado para o servidor principal é crucial que o servidor secundário também o receba. Assim, via UDP, sempre que o servidor principal recebe um novo ficheiro, envia o ficheiro para o servidor secundário. Como a ligação é UDP existe a chance de o ficheiro vir com erros. Assim, se ocorrer um erro a meio da transmissão ou se o *checksum* final não for igual ao checksum do ficheiro original, o ficheiro é reenviado.

O servidor secundário precisa também de estar constantemente a verificar se o servidor primário ainda está vivo. Para o efeito, a cada 2.5 segundos o secundário manda um *ping* para o servidor primário para verificar o seu estado. Se receber uma resposta, é porque o servidor primário ainda está *up*. Senão, assume o papel de servidor primário. Quando esta transição ocorre, quando um processo cliente tentar executar qualquer funcionalidade que utilize o socket do antigo servidor primário, vai-se levantar uma exceção (*java.net.SocketException*) que vai levar o processo a tentar se ligar ao socket do servidor secundário que agora atua como primário. Se esta ligação for bem-sucedida, pede-se uma nova autenticação e o cliente está pronto para mandar de novo comandos.

5. Testes feitos à plataforma

Foram realizados diversos testes à plataforma. Apresenta-se uma tabela com os resultados dos testes mais relevantes:

Teste	Servidor primário	Servidor secundário	Descrição
Nomes de utilizador com carateres especiais (@$.)	Success	Success	Verificamos que a plataforma aceita nomes de utilizador com carateres que potenciavam erros. Verificamos que tal não acontecia.
Upload/download de ficheiros <= 70MB server-cliente TCP	Success	Success	Verificamos que ficheiros até cerca de 70MB são enviados corretamente. Como é TCP, pensamos que qualquer tamanho de ficheiro seja enviado corretamente embora o tempo de envio possa ser muito grande.
Upload de ficheiros <=6MB do servidor primário para servidor secundário via UDP (failover)	-	Success	Verificamos que ficheiros até cerca de 6 MB são transferidos via UDP corretamente, embora, para ficheiros

			mais perto de 6MB, seja preciso, por vezes, alguns reenvios.
Upload de ficheiros > 6MB do servidor primário para servidor secundário via UDP (failover)	-	Fail	Verificamos que ficheiros maiores do que 6MB, embora por vezes enviados, têm uma taxa de reenvio muito alta. Assim, não consideramos que o programa suceda neste teste.
Servidor secundário e clientes corretamente identificam quando o servidor primário falha	-	Success	Verificamos que, quando o servidor primário cai, os clientes e o servidor secundário corretamente identificam esta falha e se ligam entre si.
Não permitir que clientes saiam da pasta /home ou /home2 do server	Success	Success	Verificamos que por mais que o cliente tente, e devido à forma como foi implementado, é impossível o cliente conseguir sair da pasta /home do server
Permitir vários clientes utilizarem a plataforma incluindo upload de ficheiros	Success	Success	Verificamos que a plataforma suporta vários utilizadores em simultâneo e podem fazer upload de ficheiros simultaneamente sem que haja algum problema visto que o servidor primário adiciona os ficheiros (a serem enviados) a uma *quele* que é tratada por uma thread independente.

6. Distribuição de tarefas

Numa fase inicial, optamos por dividir o trabalho em 2 metades. Enquanto um de nós trabalhava no script do servidor (Miguel) o outro trabalhava no script do cliente (Edgar), embora isto não signifique que o outro membro não tenha trabalhado no script oposto. Numa fase mais avançada, nomeadamente do *failover*, enquanto um dos membros fazia o envio de ficheiros para o servidor secundário (Miguel) o outro fazia a receção desses mesmos ficheiros (Edgar). Um dos membros também desenvolveu a classe do ficheiro de configuração (Edgar) enquanto o outro tratou dos *heartbeats* (Miguel). Depois, com o recurso ao GitHub, progressivamente, juntávamos os códigos. Desta forma conseguimos paralelizar eficientemente o trabalho. Estimamos que cada aluno tenha gastado cerca de 25 horas.