Sistema Dropbox-like

Bruno Fernandes Carvalho - 15/0007159
Dep. Ciência da Computação - Universidade de Brasília (UnB)
Transmissão de Dados - Turma A
Data de realização: 18/06/2017
brunofcarvalho1996@gmail.com

Abstract

Sistemas de armazenamento e sincronização em nuvem são amplamente utilizados e representam uma importante parcela na descentralização e no compartilhamento os dados. Foi desenvolvido nesse trabalho um sistema que realizar as principais operações do Dropbox, permitindo armazenar arquivos remotamente e acessá-los de diferentes máquinas, buscando aplicar os conceitos da disciplina de conectividade e relações cliente-servidor.

1. Objetivos

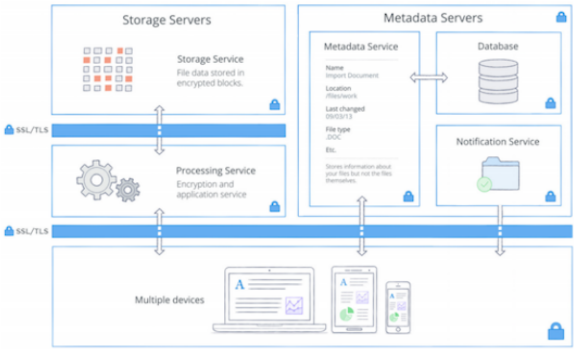
O objetivo do trabalho é usar os conceitos de conexão cliente-servidor usando protocolo TCP/IP para desenvolver um sistema capaz armazenar e sincronizar seus dados entre os mesmos usuários. Tanto o lado cliente quanto o lado servidor devem ser desenvolvidos.

2. Introdução

Para entender melhor o sistema proposto, deve ser compreendido o que é o Dropbox. Este é uma aplicação de armazenamento em nuvem, onde é possível ter o acesso aos arquivos de qualquer máquina que executar o programa cliente. Assim, é possível garantir também a sincronização dos dados entre as várias máquinas que estão acessando essa conta.

Para que esse sistema seja realizável, é necessário usar a internet para transmitir dados e garantir essa sincronização. Isso se tornou possível com a expansão da internet e com a necessidade de existir cada vez mais dados para serem armazenados. Ainda, o Dropbox conta com um protocolo de segurança para impedir ataques e fraude dos arquivos, garantindo proteção aos seus usuários. Um resumo dessa aplicação contendo suas camadas de desenvolvimento são mostradas na figura 1

Figura 1. Esquemático mostrando as camadas do Dropbox



3. Metodologia

O sistema consiste no desenvolvimento do programa cliente e servidor capaz de simular uma aplicação Dropbox-like. Para isso, será usado a linguagem de programação Python 3.5, assim como diversas bibliotecas que suportam criar Sockets, realizar múltiplas conexões, implementar protocolo de comunicação JSON e codificar arquivos. A tabela 1 mostra as principais bibliotecas de programação utilizadas nos dois programas (cliente e servidor):

Tabela 1. Bibliotecas utilizadas

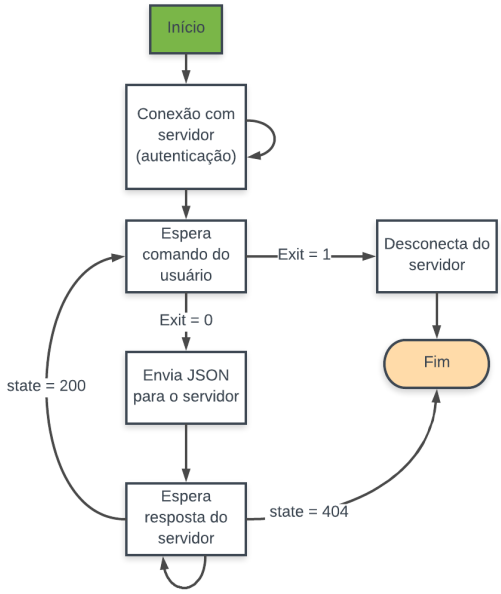
Biblioteca	Descrição
Socket	Possibilita criar sockets TCP
Json	Implementa protocolo Json
Threading	Conexão Multithreading
Base64	Codificação/decodificação de arquivos
OS/Subprocess	Possibilita operar no SO

Antes de executar o programa, recomenda-se que leia o arquivo README.txt para saber alguns detalhes de como rodar o programa.

3.1. Cliente

O programa cliente deve ser capaz de realizar as seguintes operações: se autenticar no servidor e, se não for possível, deve ser capaz de criar um novo login e senha para o usuário; se comunicar corretamente com o servidor e esperar uma resposta que deve ser apresentada ao usuário; interpretar comandos de operação do usuário. Uma máquina de estados resume esse programa, mostrado na figura 2

Figura 2. Operação do cliente



Foram encontrados dois principais problemas durante o upload de arquivos: o primeiro deles acontece quando o arquivo é muito grande, sendo necessário quebrá-lo em várias partes e enviá-las separadamente ao servidor. Isso ocorre pois há um limite máximo que o socket suporta para enviar dados. O segundo foi o fato de diversos arquivos não serem text-plain (.pdf e.png, por exemplo), o que impossibilita o uso da biblioteca Json e Socket. Para isso, foi necessário codificá-los utilizando a biblioteca que implementa o algoritmo base64.

Para se comunicar com o servidor, utilizou-se o protocolo Json com os seguintes campos:

{
operation: define qual a operação que o usuário deseja realizar
name: parâmetros auxiliares que contêm informações sobre o nome do diretório/arquivo
data: contém os dados dos arquivos durante o upload
}

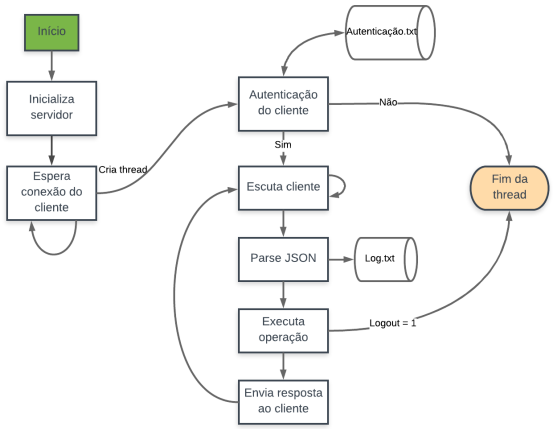
3.2. Servidor

O servidor deve ser capaz de interpretar as requisições do cliente, sendo capaz de confirmar uma autenticação ou recusá-la, realizar os comandos diretamente no SO para criar arquivos/pastas, assim como movê-las e removê-las. Todas essas operações são especificadas com o comando 'help' durante a execução do cliente. Além disso, foi utilizado o protocolo simples Json para se comunicar com o cliente. Este contém os seguintes campos:

{
state: define o estado da comunicação com o cliente: se for 200, a conexão deve ser mantida. 404, a conexão deve ser fechada.
message: mensagem que deve ser mostrada ao usuário
data: contém os dados dos arquivos durante o download
}

Para realizar a autenticação, foi criado um arquivo "autenticação.txt"que armazena o login e senha de todos os usuários já cadastrados. Já para realizar o armazenamento, utiliza-se a biblioteca OS para operar diretamente no sistema operacional. Cria-se uma pasta por usuário no diretório em que o servidor é executado, e todas as operações que o cliente requisitar são feitas dentro da respectiva pasta. Isso garante a sincronização do mesmo usuário acessando o servidor Dropbox de várias máquinas diferentes ao mesmo tempo. Além disso, é feito um log de todas as operações que o servidor realizar em um arquivo "log.txt". A máquina de estados da figura 3 simplifica esse processo.

Figura 3. Operação do servidor



Os problemas encontrados foram os mesmos do cliente, só que agora no download de arquivos, já que essa ponta da conexão lida com o envio deles. Outro ponto importante a ser destacado é o download de uma pasta, que deve enviar ao cliente todos os arquivos e pastas dentro dela de forma

a ser reconstruída corretamente no cliente. Para solucionar esse problema, foi definido um protocolo que indica em qual diretório cada arquivo/pasta estão localizados, possibilitando que os dados sejam mostrados de forma correta ao cliente.

4. Resultados

Executando ambos cliente e servidor na mesma máquina, é possível validar se os programas estão rodando corretamente. A seguir será apresentado uma sequência de imagens que mostram o funcionamento do sistema.

Primeiramente, é necessário que o usuário se autentique. Se ele criar uma conta com login já existente, uma mensagem de erro é apresentado. A figura 4 mostra algumas tentativas de acessar o servidor, assim como o log do servidor mostrando as operações que foram realizadas.

Figura 4. Autenticação do cliente no servidor

```
bruno@bruno-Lenovo-G40-80:~/Documentos/UNB/TD/trabalhoTD/cliente$ python client.p
y
Execucao invalida. Faça conforme umas das opcoes abaixo
        1) python client.py host port login password
        2) python client.py host port
bruno@bruno-Lenovo-G40-80:~/Documentos/UNB/TD/trabalhoTD/cliente$ python client.p
y 127.0.0.1 5005
Necessário criar login e senha para acessar o servidor
Login: bruno
Senha: 123
Usuario cadastrado com sucesso. Bem vindo ao dropbox bruno
Digite 'help' para mais informacoes
/bruno >>> exit
Saindo do dropbox...
bruno@bruno-Lenovo-G40-80:~/Documentos/UNB/TD/trabalhoTD/cliente$ python client.p
y 127.0.0.1 5005
Necessário criar login e senha para acessar o servidor
Login: bruno
Senha: 123
Usuario ja existente. Nao foi possivel fazer o cadastro
bruno@bruno-Lenovo-G40-80:~/Documentos/UNB/TD/trabalhoTD/cliente$ python client.p
y 127.0.0.1 5005 bruno 123
Bem vindo ao dropbox bruno
Digite 'help' para mais informacoes
/bruno >>>
```

```
bruno@bruno-Lenovo-G40-80:~/Documentos/UNB/TD/trabalhoTD$ python server.py
{"operation": "cadastro", "params": ["bruno", "123"]}
{"operation": "logout", "params": ["", ""]}
bruno se desconectou
{"operation": "cadastro", "params": ["bruno", "123"]}
{"params": ["bruno", "123"], "operation": "login"}
```

Após estar conectado no servidor, é possível ver quais operações estão disponíveis a serem realizadas, mostradas na figura 5

Figura 5. Execução do comando 'help'

```
bruno@bruno-Lenovo-G40-80:~/Documentos/UNB/TD/trabalhoTD/cliente$ python client.p
y 127.0.0.1 5005 brunoviske 123
Bem vindo ao dropbox brunoviske
Digite 'help' para mais informacoes
/brunoviske >>> help
Lista de comandos:
    1) ls: verifica arquivos e pastas do diretorio atual
    2) cd path: muda para o diretorio 'path'
    3) mv path_file dest_dir: move o arquivo ou diretorio em 'path_file' para  o
diretorio 'dest_dir'
    4) rm file: remove arquivo ou diretorio de nome 'file'\
    5) mkdir dir_name: cria um diretorio com nome 'dir_name'
    6) upload path_file: faz o upload de um arquivo ou diretorio com nome 'path_f
ile'
    7) download file: faz o download de um arquivo ou diretorio com nome 'file'
    8) exit: sair do dropbox

/brunoviske >>>
```

Mostra-se também algumas operações sendo realizadas pelo usuário, como criar diretório, upload de arquivos/pastas e remoção dos mesmos. A figura 6 mostra como isso é realizado na prática, além do log do servidor mostrando as operações realizadas. Nesse exemplo, foi criado um diretório 'pasta' e feito o upload do arquivo 'ola.txt'. Após isso, esse diretório foi removido.

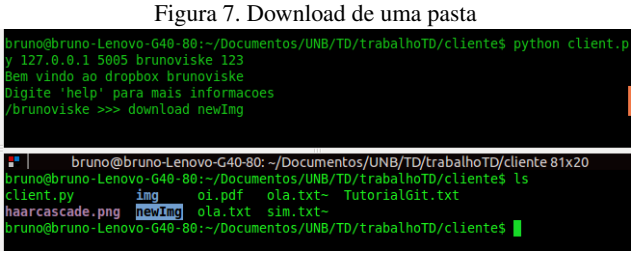
Figura 6. Operações sendo realizadas no servidor Dropbox

```
bruno@bruno-Lenovo-G40-80:~/Documentos/UNB/TD/trabalhoTD/cliente$ python client.p
y 127.0.0.1 5005 brunoviske 123
Bem vindo ao dropbox brunoviske
Digite 'help' para mais informacoes
/brunoviske >>> ls
['sim', 'newImg']
/brunoviske >>> mkdir pasta

/brunoviske >>> ls
['sim', 'newImg', 'pasta']
/brunoviske >>> cd pasta
/brunoviske/pasta >>> upload ola.txt

/brunoviske/pasta >>> ls
['ola.txt']
/brunoviske/pasta >>> cd ..
/brunoviske >>> rm pasta

/brunoviske >>> ls
['sim', 'newImg']
/brunoviske >>>
```

```
bruno@bruno-Lenovo-G40-80:~/Documentos/UNB/TD/trabalhoTD$ python server.py
{"params": ["brunoviske", "123"], "operation": "login"}
{"name": ["", ""], "operation": "ls", "data": ""}
{"name": ["pasta", ""], "operation": "mkdir", "data": ""}
{"name": ["", ""], "operation": "ls", "data": ""}
{"name": ["pasta", ""], "operation": "cd", "data": ""}
{"name": ["", "/pasta"], "operation": "upload", "data": ["b2xhYWFhYWFhYWFhYWFhYWF
hYWEK\n", "ola.txt", "0", "0", 29, 0]}
{"name": ["ola.txt", "/pasta"], "operation": "upload", "data": ""}
{"name": ["", "/pasta"], "operation": "ls", "data": ""}
{"name": ["..", "/pasta"], "operation": "cd", "data": ""}
{"name": ["pasta", ""], "operation": "rm", "data": ""}
{"name": ["", ""], "operation": "ls", "data": ""}
```

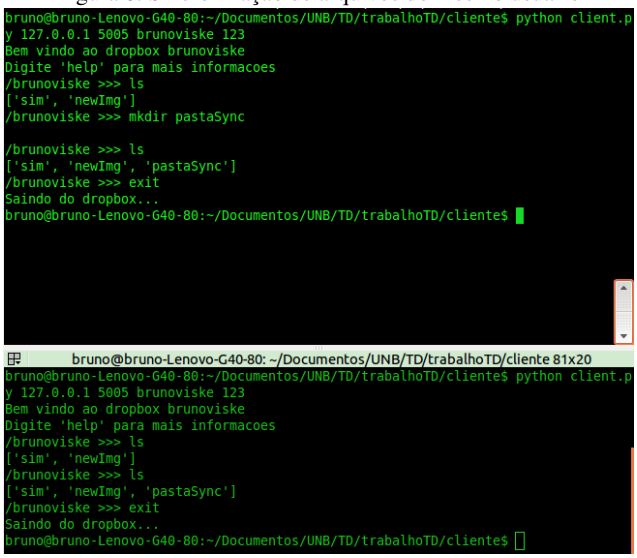
Na figura 7 observa-se o comando de download da pasta "newImg", que apareceu corretamente no diretório de execução do programa cliente (está destacada em azul).

Figura 7. Download de uma pasta



Por último, mostra-se na figura 8 a sincronização entre dois clientes executando na mesma máquina, porém em processos diferentes. Foi criada a pasta 'pastaSync' no primeiro processo, a qual foi sincronizada entre todas as conexões desse mesmo usuário.

Figura 8. Sincronização de arquivos do mesmo usuário



5. Conclusão

A partir da metodologia apresentada, foi possível concluir os principais aspectos de implementação desse sistema, tanto do lado cliente quanto do lado servidor, assim como os problemas encontrados e as soluções para cada um deles. Ainda, entendeu-se os protocolos que foram utilizados e o que cada campo representa na comunicação entre as pontas.

Além disso, os resultados do trabalho mostraram que o sistema proposto foi implementado corretamente, realizando o armazenamento e manipulação de arquivos remotos. Também foi visto que a sincronização ocorre entre mesmos usuários em máquinas diferentes, caracterizando de fato uma aplicação Dropbox-like.

Referências

[1] J. F. Kurose and K. W. Ross, *Computer Networking - A top-down approach*, 6th ed. Pearson.

[2] Python Documentation. (2018, 18 Junho). [Online]. Available: https://www.python.org/doc/

[3] M. F. Caetano, C. Camargo, and M. Makiuchi, "Roteiro do trabalho: Desenvolvimento de um sistema dropbox-like," 2018.

[1] [2] [3]