Relatório do Trabalho 1 - Transmissão de Dados Desenvolvimento de um sistema Dropbox-*like*

Luis Fernando Arruda Marques
Universidade de Brasília
Brasíla, DF
e-mail: luisfernandomarques@outlook.com

***Resumo*—Desenvolvimento de um sistema cliente-servidor de compartilhamento de arquivos, com gerenciamento de diretório e controle de acesso.**

I. Introdução

Com o advento e a popularização da internet, os serviços de compartilhamento de arquivos em nuvem se tornaram um grande e indispensável facilitador de armazenamento e compartilhamento de informações na rede. Com este propósito, hoje milhares de empresas são especializadas em prover estes serviços ao seus usuários que armazenam, de maneira confiável, alta quantidade de informações.

Estes serviços é apenas um das centenas de milhares disponíveis na internet e que fazem parte da camada de aplicação. É nesta camada da internet que toda a preocupação está no serviço que se é oferecido ao usuário, deixando a cargo das camadas inferiores tudo que deve ser feito para que o sistema como um todo funcione [1].

Este trabalho consiste no desenvolvimento de um sistema servidor-cliente capaz de gerenciar os diretórios e arquivos de usuário cadastrados remotamente. As operações de gerenciamento são semelhantes àquelas encontradas no *shell* do *Linux*. Um sistema de pastas compartilhadas foi posteriormente implementando, permitindo o inter-compartilhamento de arquivos entre usuários distintos.

II. Metodologia

O sistema desenvolvido foi implementado em dois programas distintos: o cliente `(client.py)` e o servidor `(server.py)`, em linguagem Python versão 2.7.13+. O programa cliente é responsável pela interface com o usuário, fazer um pré-processamento, invalidar entradas indevidas e comunicar, através de um protocolo, a informação de comando a se realizar no servidor. O programa servidor, por sua vez, é responsável por receber a informação do cliente, fazer o seu processamento e retornar o *status* da operação.

A. Protocolo

O protocolo utilizado consiste no envio de operandos, separados por uma barra vertical "`|`", numa mensagem de até 1024 bytes de comprimento. O primeiro operando é o identificador da operação, seguido dos operandos desta operação (que podem ser mais um ou dois operandos, dependendo da instrução). O protocolo, portanto, se assemelha a um código de três endereços de linguagens Assembly, onde os primeiros bits representam a instrução, e os demais bits representam os operandos desta instrução.

Uma vez recebido esta mensagem de operandos, uma troca de mensagens correspondente à execução da operação é realizada. No *login*, por exemplo, o cliente envia `2|username`. O servidor identifica ser operação de login através do *opcode* `2`, chama a função `signin` relativa a este procedimento, que por sua vez verifica no arquivo de banco de dados `db.txt` se o usuário de fato existe e, caso verdadeiro, retorna ao cliente a mensagem `ask_pwd`. O cliente, então, ao receber esta requisição, espera do usuário que este digite a sua senha, verifica se esta pode ser válida e a envia ao servidor. Este recebe esta mensagem, checa se a senha é de fato correspondente àquela cadastrada e vinculada ao `username` e finalmente retorna ao cliente `"signin_ok"` ou `"wrong_pwd"`, para caso senha correta ou incorreta, respectivamente.

O protocolo definido tem um comportamento semelhante para os outros tipos de instruções: o cliente envia a requisição através da mensagem de até três operandos, o servidor requisita/informa novas dados necessárias a sua execução ou simplesmente retorna o *status* do procedimento caso não seja necessário mais nenhum dado, permitindo ao usuário a inserção de novas operações (Figura 1).

B. Cadastro e acesso de usuários

O gerenciamento de usuários é feito por três comandos básicos: registro (*sign up*), acesso (*sign in*) e *logout* de contas de usuário. Estas informações são lidas ou gravadas diretamente de um arquivo texto `db.txt` no servidor, que recebe as mensagens de cadastro (Figura 2) ou autenticação (Figura 3) da máquina cliente.

Para cadastro, o usuário deve inserir "1". Em seguida, é pedido o nome de usuário e a senha, e então isto é enviado numa única mensagem ao servidor, que checa se o nome de usuário está disponível e, caso afirmativo, registra em `db.txt` o nome seguido da senha, separadas por "`|`", e por fim retorna ao cliente se o usuário foi cadastrado ou não (já existente). Neste arquivo, cada linha possuirá um usuário diferente, pois o separador de usuários é uma quebra de linha.

Para se realizar login, o usuário deve inserir "2"na tela inicial do programa. O nome de usuário então é pedido, e o servidor então verifica se este de fato existe e pede a sua senha caso positivo. Este procedimento foi detalhado na seção anterior, ao exemplificar o funcionamento do protocolo.

Figura 1. Diagrama do funcionamento do protocolo

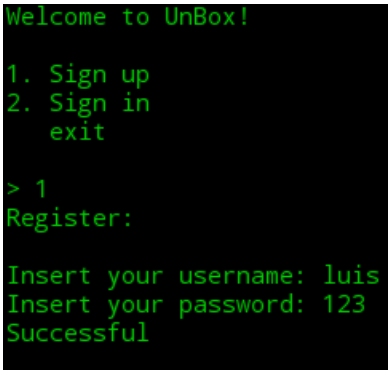

Figura 2. Procedimento de cadastro de usuário.

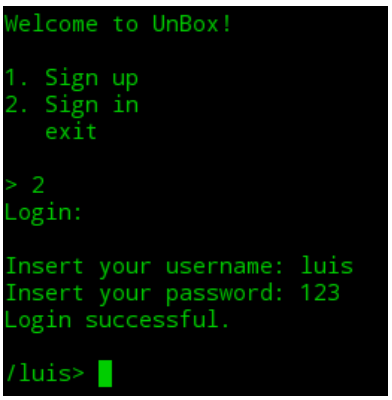

Figura 3. Procedimento de login de usuário.

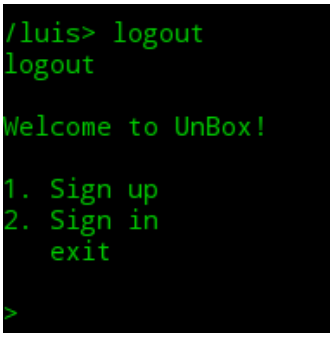

Figura 4. Procedimento de logout.

Após o login, o usuário então fica na parte de operações do sistema. O comando *logout* é o utilizado para encerrar a sessão e voltar pra tela inicial (Figura 4). Após o usuário inserir `logout`, esta requisição é enviada ao servidor que faz ambos os programas (servidor e cliente) saírem do laço de operações e voltarem pra seção de autenticação e cadastro.

C. Operações

O sistema cliente-servidor implementado permite várias operações básicas de gerenciamento remoto de arquivos e diretórios. Elas podem ser descobertas através do comando `help`.

Estas operações, como podem ser vistas na Figura 5, são em maioria de sintaxe idênticas àquelas que podem ser encontradas no *shell* do sistema operacional *Linux*, com exceção das instrução de `share` e a utilização do prefixo `sh` para a criação e uso de pastas compartilhadas (detalhadas mais a frente).

1) ls: O operação de listagem de arquivos e subdiretórios de um diretório corrente é feita utilizando-se do comando `ls`.

A Figura 6 mostra o retorno desta operação de listagem. Neste exemplo, a pasta `/luis/Musicas` possui outras quatro subpastas.

A mensagem enviada pelo cliente é apenas `ls`, uma vez que esta operação não admite operandos. O servidor a recebe, executa a função `os.listdir` e salva os nomes em uma lista. Depois, envia iterativamente cada elemento desta lista

```
/luis> help
Commands avaliable:

ls
cd <dst>
cd sh:<dst> (used for shared folders)
mv <src> <dst>
rm <src>
mkdir <src>
upload <file or folder>
download <file or folder>
logout
share (share current folder and its subdirectories and files)

/luis>
```

Figura 5. Listagem de operações disponíveis.

```
/luis/Musicas> ls
Motorhead   The Beatles   Metallica   Creedence Clearwater Revival
```

Figura 6. Utilização do comando `ls`

ao cliente, e por fim envia um EOF para indicar ao cliente que foi o último nome de arquivo/pasta. O cliente, ao receber, vai concatenando os nomes numa string e por fim a imprime em tela.

2) cd: O comando `cd` entra no diretório especificado como operando Figura 7. Caso o operando seja `..`, a operação retrocede um diretório no servidor (Figura 7).

```
/luis> cd Musicas
/luis/Musicas> cd ..
/luis> cd ..
Can't.
/luis>
```

Figura 7. Utilização do comando `cd`

Ao enviar como argumento o nome da pasta que deseja acessar, o servidor a converte para um diretório equivalente no servidor e atualiza a variável `path`, que registra o diretório atual do cliente. Se `path` é um diretório válido ou não, o servidor retorna uma mensagem correspondente ao cliente. É importante ressaltar que o servidor nunca de fato sai de sua pasta raiz, pois caso outro usuário deseja logar no servidor após isto, o mesmo estaria já previamente acessando uma pasta incorreta. Na implementação, toda a movimentação de diretórios é relativa à pasta raiz do servidor (diretório onde foi executado o `server.py`).

3) mv: A operação de movimentação de arquivos ou pastas é realizada pelo comando `mv`. Este comando aceita dois operandos: arquivo/diretório de origem e diretório de destino.

O funcionamento segue os das demais operações: o cliente envia o comando com os operandos, o servidor checa se o arquivo/diretório de origem de fato existe e em seguida faz uma faz uma interrupção pro sistema operacional realizar a movimentação através do comando `subprocess`. Este procedimento emula uma entrada no *shell*, que no caso é de mesma sintaxe que a digitada que a implementada neste

```
/luis> ls
controle_dinamico.pdf   PPP.pdf   Imagens   Musicas
/luis> mv Imagens Musicas
Moved.
/luis> ls
controle_dinamico.pdf   PPP.pdf   Musicas
/luis> cd Musicas
/luis/Musicas> ls
Motorhead   The Beatles   Metallica   Imagens   Creedence Clearwater Revival
/luis/Musicas>
```

Figura 8. Utilização do comando `mv`

sistema. As mensagens de erro retornada pelo S.O são obtidas caso existentes, e informa ao usuário que houve uma falha de movimentação se ocorrida, ou que a movimentação de fato ocorreu.

4) rm: O comando `mv` é o responsável por remover um arquivo ou diretório. Este comando aceita apenas um único operando: arquivo/diretório que se deseja apagar.

```
/luis> ls
controle_dinamico.pdf   PPP.pdf   Imagens   Musicas
/luis> rm PPP.pdf
Removed.
/luis> ls
controle_dinamico.pdf   Imagens   Musicas
/luis>
```

Figura 9. Utilização do comando `rm`

Novamente, se é utilizada uma interrupção de S.O, assim como comando `mv`. O diretório informado pelo cliente é convertido para o seu equivalente no servidor. O *status* é retornado ao cliente é de acordo com o retorno do *shell* do Linux. Na implementação, um possível problema de sincronismo foi levado em consideração: quando um usuário tentar apagar um diretório que outro usuário está acessando, ou o diretório-pai deste, uma mensagem de erro é exibida e a operação não é executada (Figura 10). Esta implementação foi realizada utilizando-se de uma lista auxiliar `cdir` que armazena os diretórios que estão sendo acessados no momento.

```
/luis> rm /luis/IPI
Another user in that directory.
/luis>
```

Figura 10. Mensagem exibida quando tenta remover um diretório que está sendo acessado por outro usuário.

5) mkdir: O comando `mkdir` cria um subdiretório. Aceita um único operando (nome do diretório).

```
/luis> ls
controle_dinamico.pdf   Imagens   Musicas
/luis> mkdir teste
Created.
/luis> mkdir 'nome composto'
Created.
/luis> ls
teste   controle_dinamico.pdf   nome composto   Imagens   Musicas
/luis>
```

Figura 11. Utilização do comando `mkdir`

Novamente, se é utilizado o próprio S.O para criação deste diretório. O diretório informado pelo cliente é convertido para o seu equivalente no servidor e então informa ao cliente o *status* da execução conforme o retorno do *shell*.

6) Upload: Para se enviar um arquivo da máquina do cliente para o servidor, utiliza-se o comando `upload`. O argumento desta função é o endereço absoluto do arquivo/diretório de envio na máquina do cliente.

```
/luis> ls
teste   controle_dinamico.pdf   nome composto   Imagens   Musicas
/luis> upload /home/luisfernando/Documents/envio/
envio
Upload completed.
/luis> ls
envio   teste   controle_dinamico.pdf   nome composto   Imagens   Musicas
/luis> cd envio
/luis/envio> ls
Lab02-2-2017.pdf
/luis/envio>
```

Figura 12. Utilização do comando `upload`

A operação de upload de arquivos é realizada em duas etapas: primeiramente o arquivo de origem é comprimido, para otimizar o uso da rede na transmissão, e depois é realizada o procedimento de envio. A compressão para formato `.zip`, é realizada utilizando `shutil.make_archive` disponível no Python. Para o envio, primeiramente é obtido o tamanho do arquivo a ser enviado (já comprimido) e informado ao servidor. O servidor então armazena esta informação e autoriza o upload (seria possível, com isso, implementar um controle de espaço de armazenamento negando o envio do arquivo caso este ultrapasse o espaço disponível). O lado cliente abre o arquivo comprimido como binário e envia sequências de 1024 bytes até todo o arquivo ser lido e enviado. O servidor, munido da informação da quantidade de bytes a receber, interrompe o recebimento do lado cliente quando a quantidade de bytes recebida for igual ou superior ao tamanho do arquivo (pode ser superior uma vez que se é enviado blocos de 1024 bytes, logo o tamanho recebido será um múltiplo de 1024 igual ou superior ao tamanho do arquivo). O arquivo então é descomprimido e uma mensagem de confirmação é enviada ao cliente e por sua vez mostrada ao usuário. Em ambos os lados da comunicação, o arquivo comprimido é apagado após o envio e recebimento.

A vantagem de se realizar a compressão do arquivo a ser enviada é ainda mais justificada quando se deseja enviar um diretório: o procedimento de enviar vários arquivos e subdiretórios é idêntico de se enviar um único, uma vez que um único arquivo comprimido é gerado. Em termos de otimização de banda de rede, o resultado disso também é muito satisfatório, uma vez que a compressão realizada pode diminuir significativamente o tamanho do envio.

Um possível problema de sincronismo também foi tratado nesta operação: caso outro usuário esteja logado neste mesmo diretório e deseja enviar um arquivo de mesmo nome simultaneamente, uma mensagem negando esta operação é exibida (Figura 13). Isto se justifica pelo fato de que não pode haver dois arquivos com mesmo nome num mesmo diretório, portanto somente quem está enviando primeiro tem a prioridade. A implementação desta *feature* foi feita utilizando uma estrutura dicionário, que relaciona quem está enviando (número de conexão) com o nome do arquivo e diretório. Desta forma, ao se realizar um *upload*, esta estrutura é checada e, caso esteja disponível o envio, o dicionário é preenchido com as informações que o diretório está recebendo o arquivo e por quem está fazendo o envio. Após este término de envio pelo primeiro usuário, esta entrada é removida da estrutura dicionário, não mais impedindo o segundo usuário de enviar este arquivo. Caso se deseje enviar um arquivo de mesmo nome a um já existe no servidor, este último será sobrescrito/atualizado, assim como é feito no Dropbox. O impedimento é, portanto, apenas para envios simultâneos, para evitar que um sobrescreva o envio do outro e que nenhum dos dois envios seja de fato realizado de maneira devida.

```
/luis> upload /home/luisfernando/Documents/CDIN
Another user is uploading with this same filename to same directory.
/luis>
```

Figura 13. Mensagem exibida quando um segundo usuário tenta fazer `upload` de um arquivo de mesmo nome no mesmo diretório simultaneamente.

7) Download: A operação de se obter um arquivo hospedado no servidor na máquina cliente é realizada pelo comando `download`, e aceita também um único operando: o diretório do arquivo/pasta que se deseja baixar (Figura 14)

```
/luis/envio> ls
Lab02-2-2017.pdf
/luis/envio> download Lab02-2-2017.pdf
Download Completed.
/luis/envio>
```

Figura 14. Utilização do comando `download`

O funcionamento é o espelho àquele realizado no `upload`: aqui, a máquina cliente é quem recebe os dados, e toda a parte de compressão e sequência de envio é realizada diretamente no servidor.

D. Shared Folder

Neste trabalho também foi implementada um sistema de compartilhamento de arquivos entre usuários através de pastas compartilhadas. O funcionamento consiste em um usuário logado tornar um próprio diretório público, de maneira que todos possam acessar. A implementação escolhida foi através do comando `share`, que torna a pasta corrente acessível a todos os usuários. Outros usuários, então, poderão acessar este diretório através do comando `cd` seguido do prefixo `sh:`, que indica ser uma operação relativa a *shared folder*. Desta maneira, se o usuário `luis`, que está acessando `/luis/Musicas` inserir o comando `share` (Figura 15), qualquer outro usuário logado, mesmo em outra conta, poderá acessar este diretório inserindo `cd sh:/luis/Musicas` (Figura 16). Os demais comandos são permitidos, desde que eles não acessem ou modifiquem diretório de pastas anteriores a esta, porém permite que subdiretórios filhos sejam operados.

Figura 15. Tornando uma pasta pública através do comando `share`

Figura 16. Acessando uma pasta compartilhada através do prefixo identificador `sh`

A implementação foi feita utilizando-se de um banco de dados auxiliar `share.txt`, que armazena todos os diretório que são públicos e que todos os usuários podem acessar. Assim, mesmo se desligar momentaneamente o servidor, esta informação é preservada, assim como as informações de cadastro de usuários.

E. Registro (log) de atividades

O servidor também cria um registro de atividades em um arquivo de texto `log.txt` que armazena todas as trocas de mensagens realizada e os resultados das requisições. Nele, é gravado a data e hora atual, seguida do número de conexão, e da mensagem relativa à execução atual.

Exemplo:

```
1 2018-06-23 02:06:30.561359 (40318): Connection established
      with 127.0.0.1
2 2018-06-23 02:06:32.442227 (40318): Received "2|luis" from
      client
3 2018-06-23 02:06:33.303309 (40318): User "luis" logged
4 2018-06-23 02:06:34.899829 (40318): Received "cd|teste"
      from client
5 2018-06-23 02:06:34.900193 (40318): Directory changed to /
      home/luisfernando/Documents/TD/trab1/servidor/luis/
      teste
6 2018-06-23 02:06:35.874842 (40318): Received "cd|.." from
      client
7 2018-06-23 02:06:35.875190 (40318): Directory changed to /
      home/luisfernando/Documents/TD/trab1/servidor/luis
8 2018-06-23 02:06:37.563465 (40318): Received "rm|teste"
      from client
9 2018-06-23 02:06:37.569192 (40318): Removed /home/
      luisfernando/Documents/TD/trab1/servidor/luis/teste
```

III. Resultados e Conclusões

O sistema cliente-servidor implementado funcionou como deveria para uma série de testes realizados pelo autor, se comportando conforme as especificações. O trabalho se mostrou muito útil ao aprendizado uma vez que o foi posto em prática o pensamento crítico de criação e estabelecimento de um protocolo e, por conseguinte, como duas máqunas devem se comunicar para se prover um serviço útil ao usuário, sendo isto toda a motivação da matéria estudada. Apesar da área trabalhada neste trabalho ser apenas a da camada de aplicação, o trabalho corrobora o entendimento de quão interessante é a divisão da internet em camadas. Foi visto que toda a preocupação de fazer o dado trafegado chegar ao destino se concerne as camadas inferiores, ficando a cargo do programador apenas definir o seu *payload* do fluxo de dados trafegado entre as máquinas.

Referências

[1] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach: International Edition*. Pearson Education Limited, 2013.