Programação para Redes de Computadores

Lucas Piacenti Graciano
Gustavo Kikey Kakinohana
Gabriel Costa Silva

Sistema de arquivo de rede via Socket

Introdução

O projeto foi totalmente desenvolvido com Python 3.11.2 e utilizando apenas ferramentas built-in da linguagem, como socket para estabelecer a conexão cliente e servidor, e shutil para realizar operações em arquivos de alto nível.

Desenvolvimento

Durante o decorrer do trabalho optamos por manter tanto o módulo do servidor quanto do cliente no mesmo projeto para facilitar desenvolvimento e os testes. Sendo assim, basta fornecer o argumento 'server' para executar o módulo do servidor e 'client' para executar o do cliente.

Além do mais, foi inserida uma pasta chamada 'workspace' a qual representa o ambiente de trabalho do cliente. Sendo assim, todos os arquivos e diretórios que serão manipulados no decorrer da aplicação, serão manuseados dentro dessa pasta.

Como utilizar

Para compilar o servidor basta executar o comando: *python3 start.py server.* Já para o cliente basta executar o comando: *python3 start.py client*

Em ambos cenários o programa irá requisitar um host e uma porta para realizar a comunicação cliente-servidor.

Exemplo:

> Enter a host: *127.0.0.1*
> Enter a port: *3000*

Após estabelecer uma conexão com sucesso, o servidor irá fornecer um log "Client connected at: 127.0.0.1:3000" no terminal e irá enviar uma mensagem para o cliente de boas vindas.

Exemplo:

> Welcome to my TCP file system!
> Type 'help' to see all commands.
> To exit application, type 'exit'

Servidor

Em relação ao lado do servidor, é onde se encontram todas as regras de negócio do projeto. Neste módulo foi criado o arquivo main.py, no qual se concentra o núcleo da aplicação, que se responsabiliza por identificar o input do usuário recebido pelo cliente e direcioná-lo para o comando respectivo.

Ademais, criamos mais dois pacotes para modularizar os comandos e os utilitários da aplicação, sendo eles 'command' e 'utils' respectivamente. O módulo de comando como o nome indica se responsabiliza por alocar as classes de comandos que o cliente pode consumir, sendo eles:

Criar diretório

Insere uma nova pasta a partir do nome fornecido.

Input:
    mkdir <nome do diretório>
Output:
    Directory created succesfully!

Exemplo:
    mkdir fotos
Observação:
    Este comando pode criar diretórios recursivamente.

Remover diretório ou arquivo

Remove uma pasta ou arquivo a partir de um caminho fornecido.

Input:
    rm <caminho do diretório/arquivo>
Output:
    File removed succesfully!
    ou
    Directory removed succesfully!

Exemplo: *rm documents/pictures*

Observação: Este comando pode remover arquivos e diretórios recursivamente.

Listar conteúdo de diretório

Lista todos os arquivos e pastas contidos dentro de um dado diretório.

Input:
ls <caminho do diretório>
Output:
Files found:
…
…

Exemplo:
ls pictures/me.png

Observação:
Para listar os arquivos da raiz, apenas coloque um '.' como nome do arquivo.

Enviar arquivo do cliente para o servidor

Envia um determinado arquivo contido na máquina do cliente para o workspace do servidor.

Input:
put <nome do arquivo a ser criado no servidor com a extensão>
Enter a file path: <caminho do arquivo a ser enviado com a extensão>
Output:
File received succesfully!

Exemplo:
put prog_redes.pdf
Enter a file path: /Users/john.doe/Documents/prog_redes_report.pdf

Observação:
É possível já enviar arquivos para dentro de um determinado diretório basta fornecer o caminho do diretório escolhido por exemplo: *put documents/reports/prog_redes.pd*.

Ajuda

Comando auxiliar que fornece informações dos comandos.

Input:

help

Output:

Commands:
mkdir <directory name> - Create a new directory in the server
rm <path> - Remove a directory or a file in the server
ls <directory name or . for root> - List all files in given directory
put <file> - Send a file to the server
exit - Exit application

Sair

Comando para finalizar a aplicação:

Input:

exit

Output:

Goodbye!