



PROJETO 3 – TFTPy: CLIENTE TFTP

PROGRAMAÇÃO UFCD 5119

Índice:

PROJETO 3 – TFTP: CLIENTE TFTP	
Introdução	3
Análise	4-6
→ Diagrama de mensagens a ilustrar o envio de um ficheiro com 1730 bytes	4
→ Diagrama de mensagens a ilustrar o envio de um ficheiro com 1536 bytes	5
→ Diagrama de mensagens a ilustrar a recepção de um ficheiro com 2100 bytes	6
Desenho e Estrutura	7-8
Implementação	8-16
Conclusão	16
Webgrafia	16

Introdução

O Trivial File Transfer Protocol (TFTP) é uma ferramenta crucial para a transferência de arquivos dentro de redes locais. Este protocolo, embora rápido em comparação com outros, muitas vezes sacrifica segurança em prol da eficiência. Sua utilização é mais comum para transferência de arquivos de pequeno porte, e ele depende do User Datagram Protocol (UDP) devido à sua natureza sem conexão.

- Objetivos do trabalho:

O trabalho proposto, denominado CLIENTE TFTP, visa o desenvolvimento de uma aplicação de transferência de arquivos baseada no TFTP. Ao longo do projeto, os objetivos principais incluem:

Compreender os conceitos fundamentais de programação em redes, como o uso de sockets para comunicação entre processos.

Aprofundar o conhecimento funcional do protocolo TFTP e sua implementação.

Desenvolver uma aplicação cliente TFTP funcional e eficiente para transferência de arquivos dentro de uma rede local.

Opcionalmente, explorar a implementação de um servidor TFTP para aumentar a compreensão global do protocolo e suas funcionalidades.

Esses objetivos proporcionarão aos participantes uma compreensão abrangente de como o TFTP funciona e como implementar soluções eficazes para transferência de arquivos em ambientes de rede local.

Para garantir a colaboração e o controle de versões do código-fonte do projeto, optamos por utilizar o GitHub como plataforma de hospedagem. O GitHub é uma ferramenta amplamente reconhecida que permite o compartilhamento e gerenciamento eficiente de projetos de software.

Análise

→ Diagrama de mensagens a ilustrar o envio de um ficheiro com 1730 bytes

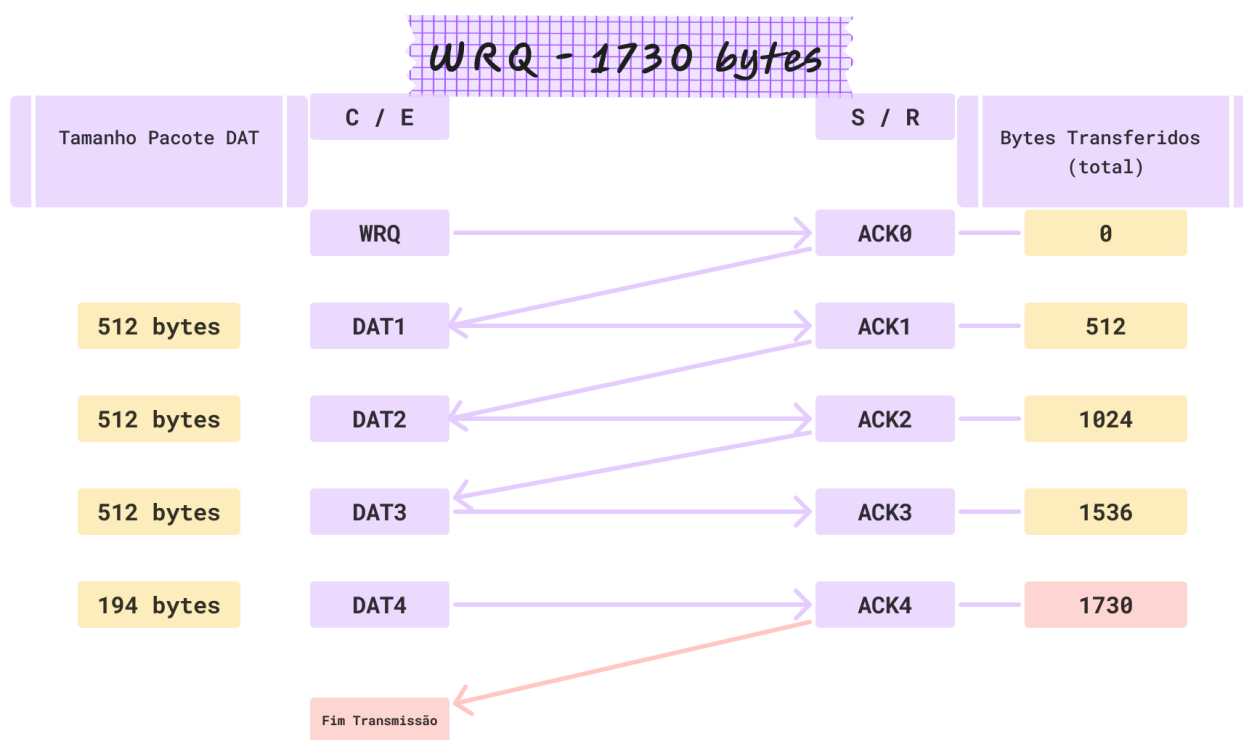


Figura 1 – Diagrama de mensagens

LEGENDA
C --> CLIENTE
S --> SERVIDOR
E --> EMISSOR
R --> RECEPTOR
WRQ --> WRITE REQUEST
ACK --> ACKNOWLEDGE

Figura 2 - Legenda

→ Diagrama de mensagens a ilustrar o envio de um ficheiro com 1536 bytes

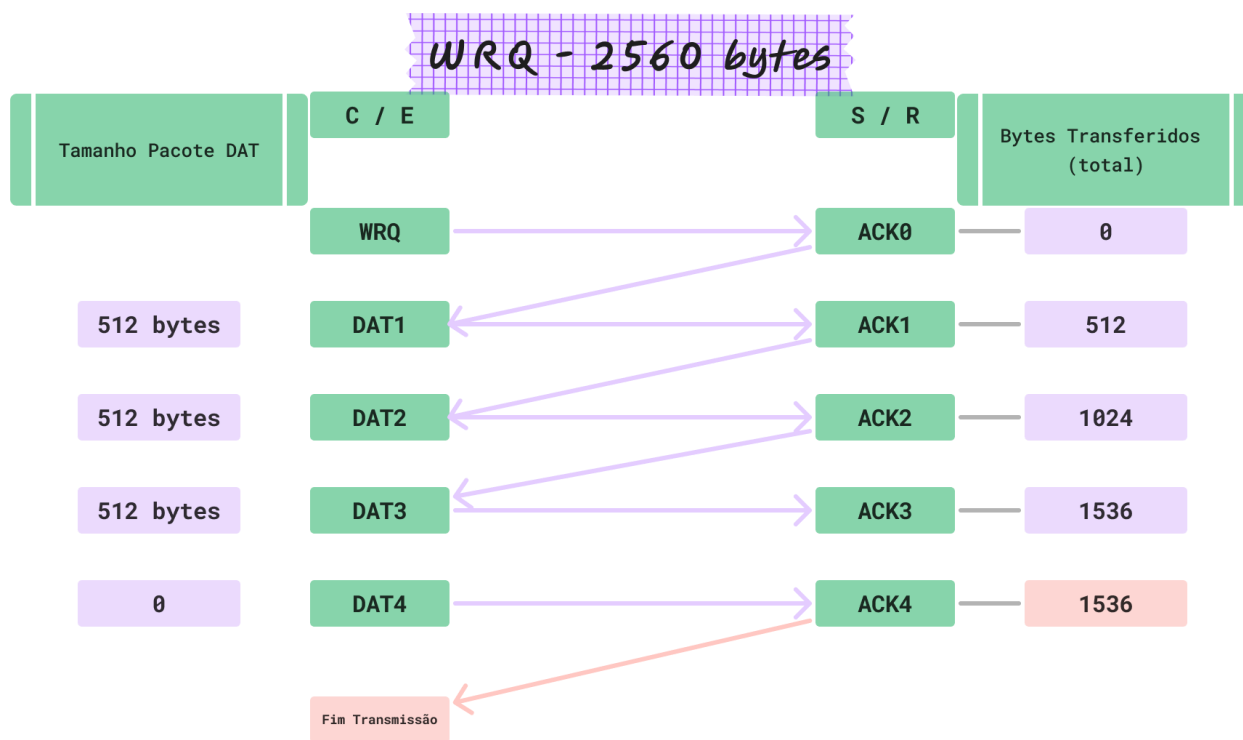


Figura 3 – Diagrama de mensagem

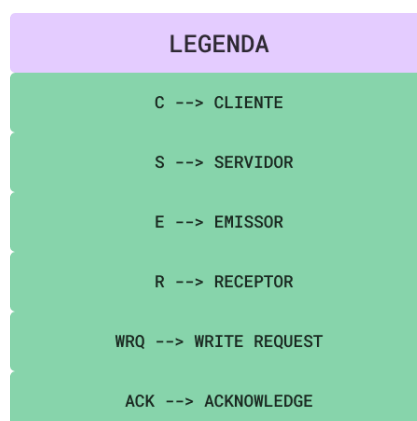


Figura 4 - Legenda

→ Diagrama de mensagens a ilustrar a recepção de um ficheiro com 2100 bytes

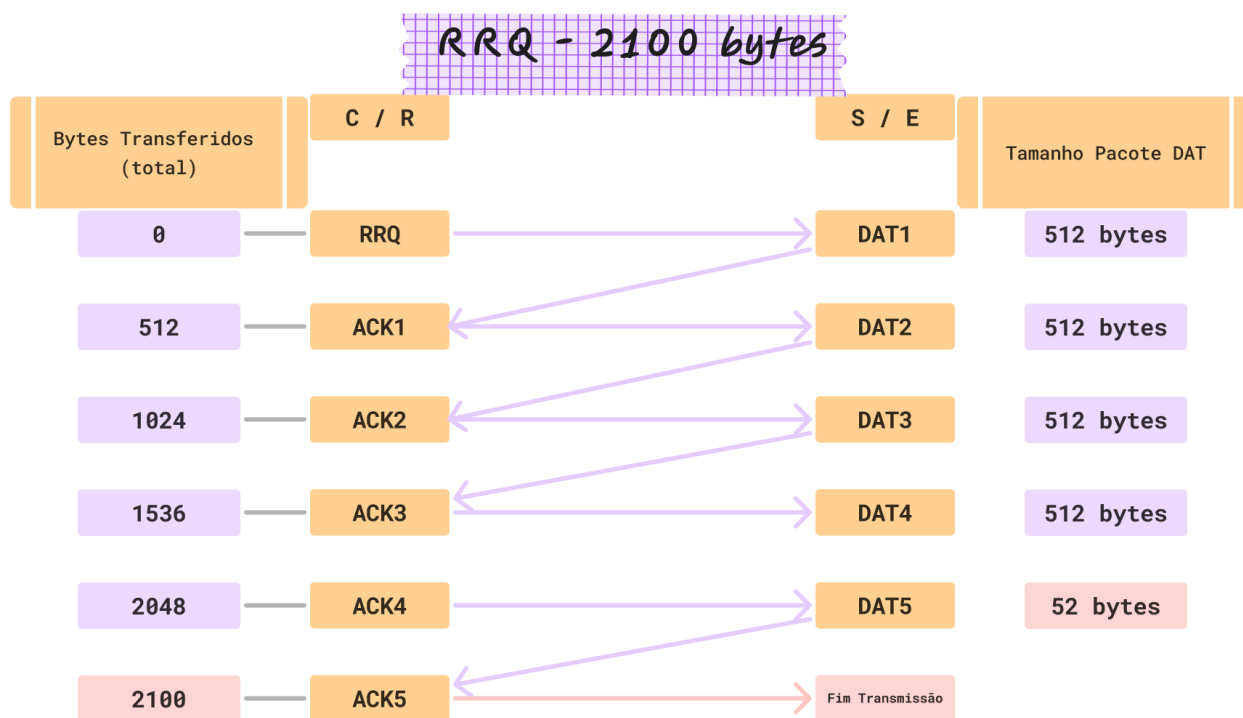


Figura 5 – Diagrama de mensagens



Figura 6 - Legenda

Desenho e Estrutura

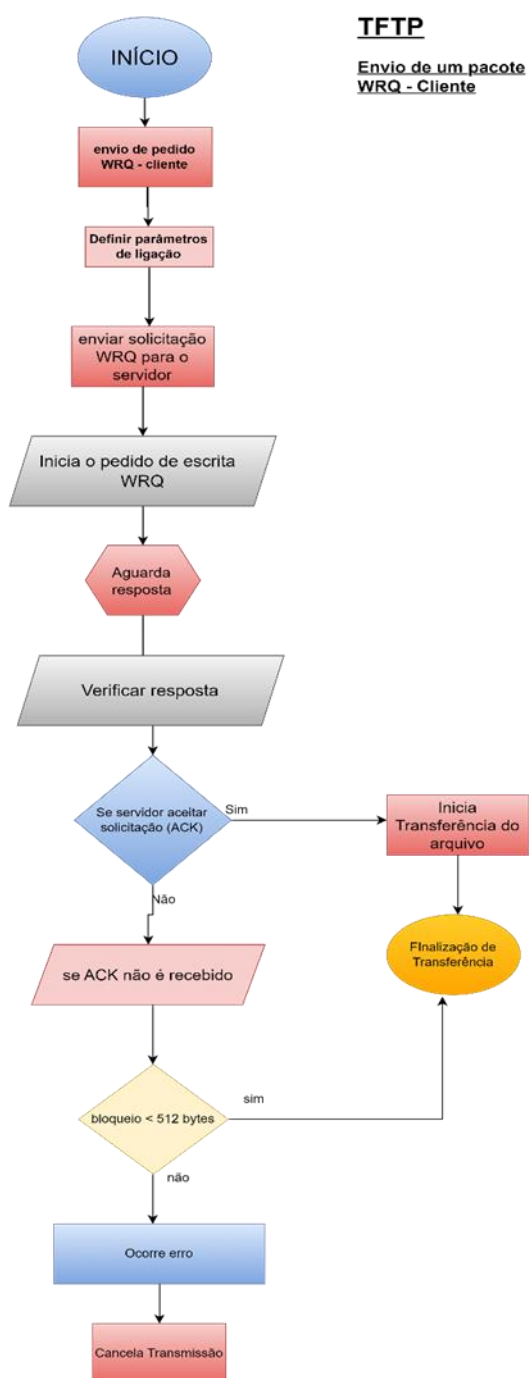


Figura 7 – Fluxograma WRQ

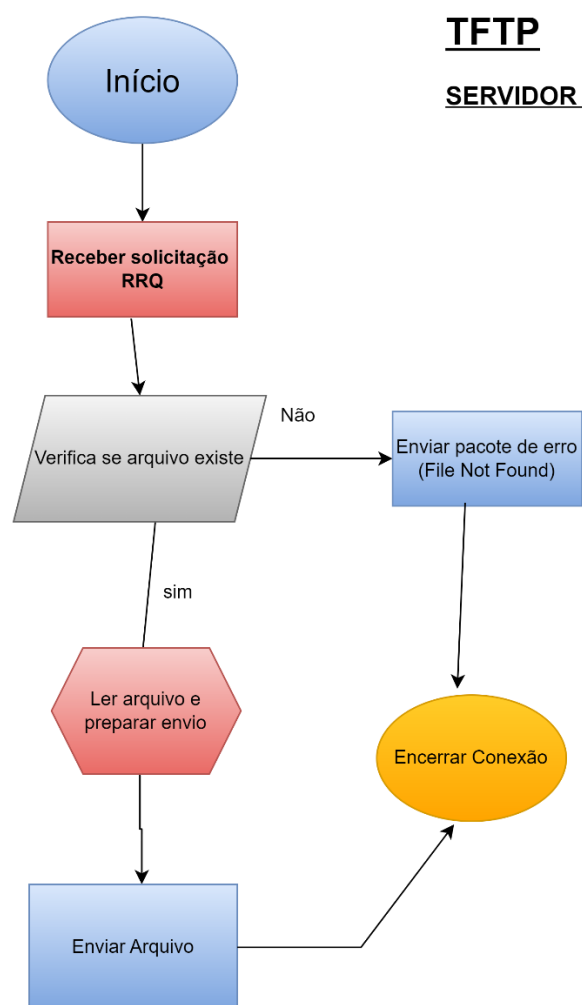


Figura 8 -Fluxograma RRQ

Implementação

Neste projeto implementamos um modo interativo para facilitar a interação com o cliente, adicionamos as mensagens de erro detalhadas como estão no enunciado do projeto de forma a ajudar o utilizador para comandos inválidos ou erros durante a execução do programa, para além disso é fornecida mensagem ao utilizador se os ficheiros em questão existem tanto no lado remoto como no lado local.

No modo interativo os comandos 'get', 'put', 'help' e 'quit' são suportados, num loop infinito que permite ao utilizador executar as operações correspondentes.

No geral são utilizadas as bibliotecas socket para comunicação de rede usando sockets UDP; o sys para manipulação de argumentos na linha de comandos e saída de erros; e uso do docopt para processamento de argumentos na linha de comandos.

Código TFTP

Fizemos uso do módulo socket, utilizado para criar um socket UDP, permitindo assim a comunicação e transferência de pacotes entre o cliente e o servidor TFTP. A manipulação de strings é facilitada pelo módulo string, que fornece constantes e funções para operações comuns de manipulação de strings. Para a codificação e decodificação de dados binários, usamos o módulo struct. Ele possibilita o empacotamento e desempacotamento de dados, é essencial para lidar com os formatos binários que são frequentemente encontrados em operações de transferência de ficheiros.

O módulo ipaddress é utilizado para representar e manipular endereços IP e redes IP, o módulo sys é utilizado para acessar variáveis e funções específicas do sistema, é usado para obter argumentos da linha de comandos e fornecer mensagens de erro caso os argumentos não sejam fornecidos corretamente.

A função `verif_ficheiro_existente` é definida para verificar a existência de um ficheiro no servidor TFTP antes de iniciar uma operação de transferência. Ela recebe dois parâmetros: `server_addr`, um tuple a representar o endereço IPv4 do servidor TFTP e o número do porto, e `remote_filename`, o nome do ficheiro remoto a ser verificado. A função devolve um valor booleano indicando se o arquivo existe ou não.

Dentro da função, um socket UDP é criado para comunicação com o servidor TFTP, e é definido um tempo limite para operações de leitura e gravação no socket. Em seguida, é feita uma solicitação de leitura (RRQ) que é montada para o ficheiro remoto especificado e enviada para o servidor. Um loop infinito é iniciado para aguardar a resposta do servidor.

Durante cada iteração do loop, um pacote é recebido do servidor e o opcode (código de operação) é extraído. Se o opcode indicar que o arquivo existe (DAT), a função retorna True. Se indicar que houve um erro (ERR), verifica-se se o erro foi do ficheiro não encontrado.

Técnico Especialista em Gestão de Redes e Sistemas Informáticos – TEGRSI12

Senão, uma exceção é lançada. Se não houver resposta do servidor dentro do tempo limite especificado, a função volta como False.

A função `get_file` é definida para transferir um arquivo remoto do servidor TFTP e guardá-lo num ficheiro local. Ela recebe três parâmetros: `server_addr`, um tuplo que representa o endereço IPv4 do servidor TFTP e o número do porto, `source_filename`, o nome do ficheiro remoto a ser transferido, e `destination_filename`, o nome do ficheiro local para guardar o conteúdo transferido.

Dentro da função, um socket UDP é criado para comunicação com o servidor TFTP, e é definido tempo limite para operações de leitura e gravação no socket. De seguida, um ficheiro local é aberto para escrita em modo binário. Uma solicitação de leitura (RRQ) é montada para o ficheiro remoto especificado e enviada para o servidor. Um loop infinito é iniciado enquanto esperamos a resposta do servidor. Durante cada iteração do loop, um pacote é recebido do servidor e o opcode é extraído. Se o opcode indicar que os dados do arquivo estão a ser enviados (DAT), os dados são escritos no ficheiro local e um acknowledge (ACK) é enviado de volta para o servidor. O progresso da transferência é exibido. Se todos os dados do arquivo forem recebidos, a transferência é concluída. Se houver um erro durante a transferência (ERR), uma exceção é levantada. Se não houver resposta do servidor dentro do tempo limite especificado, a função retorna False.

A função `put_file` é definida para enviar um ficheiro local para um servidor TFTP e lidar com erros durante a transferência. Ela recebe três parâmetros: `server_addr`, um tuplo que representa o endereço IPv4 do servidor TFTP e o seu número do porto, `source_filename`, o nome do ficheiro local a ser enviado, e `destination_filename`, o nome do ficheiro remoto no servidor TFTP (opcional, se não fornecido, o ficheiro será enviado com o mesmo nome do ficheiro local).

Dentro da função, um socket UDP é criado para comunicação com o servidor TFTP, e é definido um tempo limite para operações de leitura e gravação no socket. O ficheiro local é aberto para leitura em modo binário. Uma solicitação de escrita (WRQ) é montada para o ficheiro remoto especificado e enviada para o servidor. O número do bloco é inicializado como 0 e o tamanho do ficheiro local é obtido em bytes. Uma mensagem a indicar o início do envio do arquivo é exibida. Um loop infinito é iniciado para aguardar a resposta do servidor. Durante cada iteração do loop, um pacote de acknowledge (ACK) é recebido do servidor e o opcode é extraído. Se o opcode indicar que o servidor está pronto para receber dados adicionais (ACK), verifica-se se o número do bloco recebido corresponde ao número do bloco que é esperado. Se corresponder, os dados do arquivo local são lidos em pedaços de tamanho máximo e enviados ao servidor em pacotes de transferência de dados (DAT).

O progresso da transferência é exibido. Se todos os dados do ficheiro forem enviados, a transferência é concluída e uma mensagem a indicar o envio completo é exibida. Se houver um erro durante a transferência (ERR), uma exceção é levantada. Se não houver resposta do servidor dentro do tempo limite especificado, a função retorna False. Essa função é útil para enviar ficheiros locais para um servidor TFTP de forma confiável e lidar com possíveis erros durante a transferência, além de exibir um indicador de progresso para o utilizador.

Técnico Especialista em Gestão de Redes e Sistemas Informáticos – TEGRSI12

pack_rrq(filename: str, mode: str = DEFAULT_MODE) -> bytes: Esta função empacota uma solicitação de leitura (Read Request - RRQ) para o ficheiro especificado com o modo de transferência especificado, devolvendo os bytes empacotados.

_pack_rrq_wrq(opcode: int, filename: str, mode: str = DEFAULT_MODE) -> bytes: Função interna usada para empacotar solicitações de leitura (RRQ) ou gravação (WRQ). Recebe o opcode da solicitação, o nome do ficheiro e o modo de transferência, devolvendo os bytes empacotados.

unpack_rrq(packet: bytes) -> tuple[str, str]: Desempacota uma solicitação de leitura (RRQ) de um pacote recebido, retornando uma tupla contendo o nome do arquivo e o modo de transferência.

unpack_wrq(packet: bytes) -> tuple[str, str]: Desempacota uma solicitação de gravação (WRQ) de um pacote recebido, devolvendo um tuplo contendo o nome do transferência e o modo de transferência.

pack_dat(block_number: int, data: bytes) -> bytes: Empacota um bloco de dados para transferência. Recebe o número do bloco e os dados a serem enviados, retornando os bytes empacotados.

unpack_dat(packet: bytes) -> tuple[int, bytes]: Desempacota um bloco de dados de um pacote recebido, retornando um tuplo contendo o número do bloco e os dados.

pack_ack(block_number: int) -> bytes: Empacota um acknowledge (ACK) para confirmar a recepção de um bloco de dados. Recebe o número do bloco recebido e retorna os bytes empacotados do ACK.

unpack_ack(packet: bytes) -> int: Desempacota um acknowledge (ACK) de um pacote recebido, retornando o número do bloco confirmado pelo ACK.

pack_err(error_code: int, error_msg: str | None = None) -> bytes: Empacota uma mensagem de erro (ERR) com o código de erro e a mensagem de erro especificados. Se a mensagem de erro não for fornecida, usa a mensagem padrão associada ao código de erro, retornando os bytes empacotados da mensagem de erro.

unpack_err(packet: bytes) -> tuple[int, str]: Desempacota uma mensagem de erro (ERR) de um pacote recebido, retornando um tuplo contendo o código de erro e a mensagem de erro.

unpack_opcode(packet: bytes) -> int: Desempacota o opcode (código de operação) de um pacote recebido, retornando o opcode.

TFTPValueError(ValueError): Esta classe é uma subclasse de ValueError e é utilizada para representar erros relacionados a valores inválidos no protocolo TFTP.

NetworkError(Exception): É uma subclasse de Exception e serve para representar erros de rede, como host não encontrado, tempos de espera expirados.

Técnico Especialista em Gestão de Redes e Sistemas Informáticos – TEGRSI12

ProtocolError(NetworkError): Subclasse de NetworkError, esta classe representa erros de protocolo, como opcode inesperado ou inválido, número de bloco incorreto, ou qualquer outro parâmetro de protocolo inválido.

Err(Exception): É uma subclasse de Exception e é utilizada para representar erros enviados pelo servidor TFTP. Pode ser causada porque uma operação de leitura/escrita não pode ser processada. Erros de leitura e escrita durante a transmissão de arquivos também causam o envio desta mensagem, e a transmissão é então terminada. O número do erro fornece um código de erro numérico, seguido de uma mensagem de erro ASCII que pode conter informações adicionais específicas do sistema operacional.

Além disso, o código apresenta uma função interna `_make_is_valid_hostname`, que devolve uma função `is_valid_hostname` para verificar se uma string é um nome de host válido de acordo com as regras especificadas. Estas classes de exceção são essenciais para capturar e manipular diferentes tipos de erros que podem ocorrer durante a execução de operações de transferência de ficheiros usando o protocolo TFTP.

A função `get_host_info` recebe um endereço de servidor como argumento e devolve o endereço IP e o nome do host correspondente.

try:: Este bloco try é utilizado para capturar exceções que podem ocorrer durante a execução do código.

ipaddress.ip_address(server_addr):: Tenta criar um objeto `ip_address` com a string `server_addr`. Se `server_addr` for um endereço IP válido, esta linha não levantará uma exceção `ValueError`.

server_ip = server_addr:: Se `server_addr` for um endereço IP válido, atribui `server_addr` à variável `server_ip`.

try:: Inicia outro bloco try para lidar com possíveis exceções ao tentar obter o nome do host correspondente ao endereço IP.

server_name = socket.gethostbyaddr(server_ip)[0]: Tenta obter o nome do host usando o método `gethostbyaddr` do módulo `socket`, passando o endereço IP como argumento. Aqui, `[0]` é usado para obter o nome do host a partir do tuplo devolvido.

except socket.error:: Se ocorrer uma exceção ao tentar obter o nome do host, atribui uma string vazia à variável `server_name`.

except ValueError:: Se a tentativa de criar um objeto `ip_address` falhar, a exceção `ValueError` será capturada. Isso significa que `server_addr` não é um endereço IP válido.

if not is_valid_hostname(server_addr):: Verifica se `server_addr` não é um nome de host válido usando a função `is_valid_hostname`. Se não for, levanta uma exceção `ValueError` indicando que o nome do host é inválido.

server_name = server_addr:: Se `server_addr` for um nome de host válido, atribui `server_addr` à variável `server_name`.

Técnico Especialista em Gestão de Redes e Sistemas Informáticos – TEGRSI12

server_ip = socket.gethostbyname_ex(server_name)[2][0]: Obtém o endereço IP correspondente ao nome do host usando o método `gethostbyname_ex` do módulo `socket`. Aqui, `[2][0]` é usado para obter o primeiro endereço IP da lista retornada.

except socket.gaierror:: Se ocorrer uma exceção ao tentar obter o endereço IP correspondente ao nome do host, levanta uma exceção `NetworkError` indicando que o servidor é desconhecido.

return server_ip, server_name:: Retorna um tuple que contém o endereço IP e o nome do host correspondente ao endereço de servidor fornecido.

Código cliente

A solução implementada utiliza diversas bibliotecas e módulos do Python para manipulação de ficheiros e interação com o sistema operacional, em conjunto com uma abordagem de linha de comando e um modo interativo para execução de operações TFTP (Trivial File Transfer Protocol).

As bibliotecas importadas incluem:

os: Fornecer funções para interagir com o sistema operacional, como manipulação de arquivos e leitura de variáveis de ambiente.

textwrap: Fornecer funções para formatar strings de texto em parágrafos ou blocos de texto com largura específica.

sys: Utilizada para acessar argumentos de linha de comando e manipular exceções.

docopt: Facilita o processamento de argumentos de linha de comando, simplificando a criação de interfaces de linha de comando. Analisa a linha de comando do usuário com base na descrição e devolve um dicionário com os argumentos fornecidos.

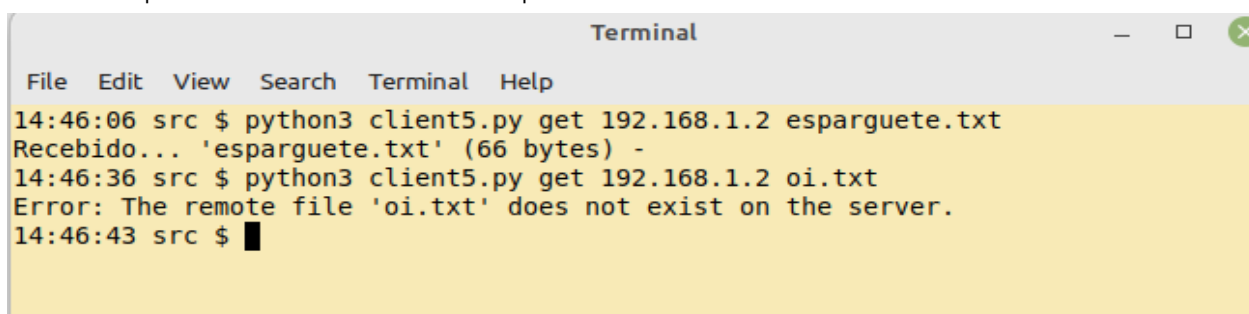
A função `docopt(doc)` é utilizada para analisar os argumentos da linha de comandos com base nessa documentação, devolvendo um dicionário contendo os argumentos e opções fornecidos pelo user. Após a análise dos argumentos, o código adquire o nome do servidor fornecido pelo user, acedendo a chave correspondente no dicionário de argumentos. Além disso, a porta do servidor é obtida e convertida para um inteiro, garantindo consistência no tipo de dado.

Em seguida, a função `verif_estado_server(server_name, server_port)` é chamada para verificar o estado do servidor, ou seja, se ele está disponível. Se o servidor estiver disponível, o endereço do servidor (compreendendo o endereço IP e a porta) é armazenado na variável `server_address` para uso posterior.

A função principal, `main()`, inicia o programa, definindo uma mensagem de ajuda, obtendo o nome e porta do servidor e verificando seu estado. Com base nos argumentos da linha de comando, determina a operação a ser executada e chama a função apropriada (`get`, `put` ou inicia um shell TFTP).

Técnico Especialista em Gestão de Redes e Sistemas Informáticos – TEGRSI12

Se o usuário especificou a operação "get" na linha de comandos, se essa condição for verdadeira, o programa prossegue com a extração do nome do ficheiro remoto e do nome do ficheiro local dos argumentos analisados. Se o utilizador não forneceu um nome de ficheiro local, o código verifica e utiliza o nome do ficheiro remoto como nome do ficheiro local. Posteriormente, o programa verifica se o ficheiro remoto existe no servidor TFTP. Se o arquivo não existir, uma mensagem de erro é exibida e o programa é encerrado com um código de saída 1. Caso o ficheiro remoto exista, a função `get_file` é chamada para transferir o ficheiro do servidor TFTP para o cliente. Os argumentos passados para essa função são o endereço do servidor, o nome do ficheiro remoto e o nome do ficheiro local. Este bloco de código garante que, se o user solicitou a operação "get", o programa tentará transferir o ficheiro especificado do servidor TFTP para o cliente.



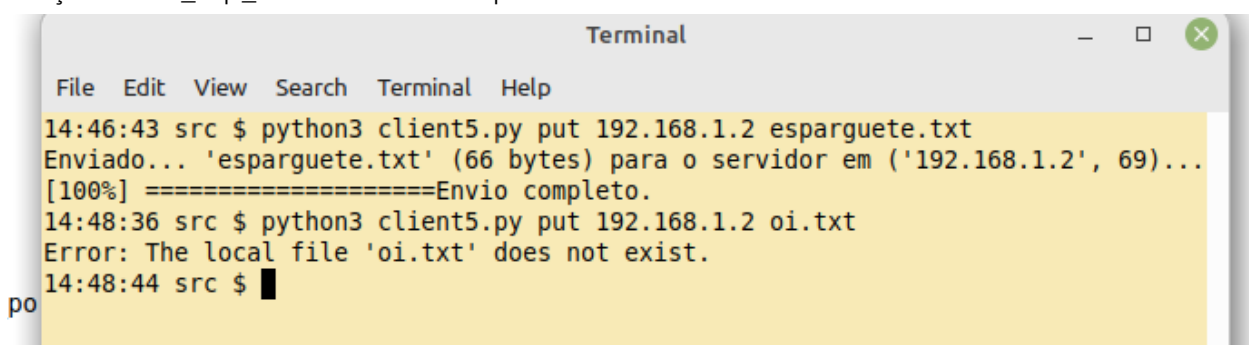
```

Terminal
File Edit View Search Terminal Help
14:46:06 src $ python3 client5.py get 192.168.1.2 esparguete.txt
Recebido... 'esparguete.txt' (66 bytes) -
14:46:36 src $ python3 client5.py get 192.168.1.2 oi.txt
Error: The remote file 'oi.txt' does not exist on the server.
14:46:43 src $

```

Figura 9 - 'GET' em modo não interativo

Se o usuário especificou a operação "put" na linha de comandos, se essa condição for verdadeira o programa prossegue obtendo o nome do ficheiro local e o nome do ficheiro remoto dos argumentos analisados. Se o user não fornecer um nome de ficheiro remoto, o código usa o nome do ficheiro local como nome do ficheiro remoto. Em seguida, verifica se o ficheiro local especificado pelo usuário existe. Se o ficheiro não existir, uma mensagem de erro é exibida e o programa é encerrado com um código de saída 1. Caso o ficheiro local exista, a função `put_file` é chamada para enviar o ficheiro do cliente para o servidor TFTP. Os argumentos passados para essa função são o endereço do servidor, o nome do ficheiro remoto e o nome do ficheiro local. Por fim, se o user não especificar nem "get" nem "put" na linha de comandos, o programa presume que a intenção é executar o shell TFTP, e então a função `exec_tftp_shell` é chamada para iniciar o shell TFTP.



```

Terminal
File Edit View Search Terminal Help
14:46:43 src $ python3 client5.py put 192.168.1.2 esparguete.txt
Enviado... 'esparguete.txt' (66 bytes) para o servidor em ('192.168.1.2', 69)...
[100%] =====Envio completo.
14:48:36 src $ python3 client5.py put 192.168.1.2 oi.txt
Error: The local file 'oi.txt' does not exist.
14:48:44 src $

```

Figura 10 - 'PUT' modo interativo

A função `exec_tftp_shell()` executa o programa em modo interativo, aceitando comandos do usuário como `get`, `put`, `help` e `quit`.

Técnico Especialista em Gestão de Redes e Sistemas Informáticos – TEGRSI12

Comandos Suportados:

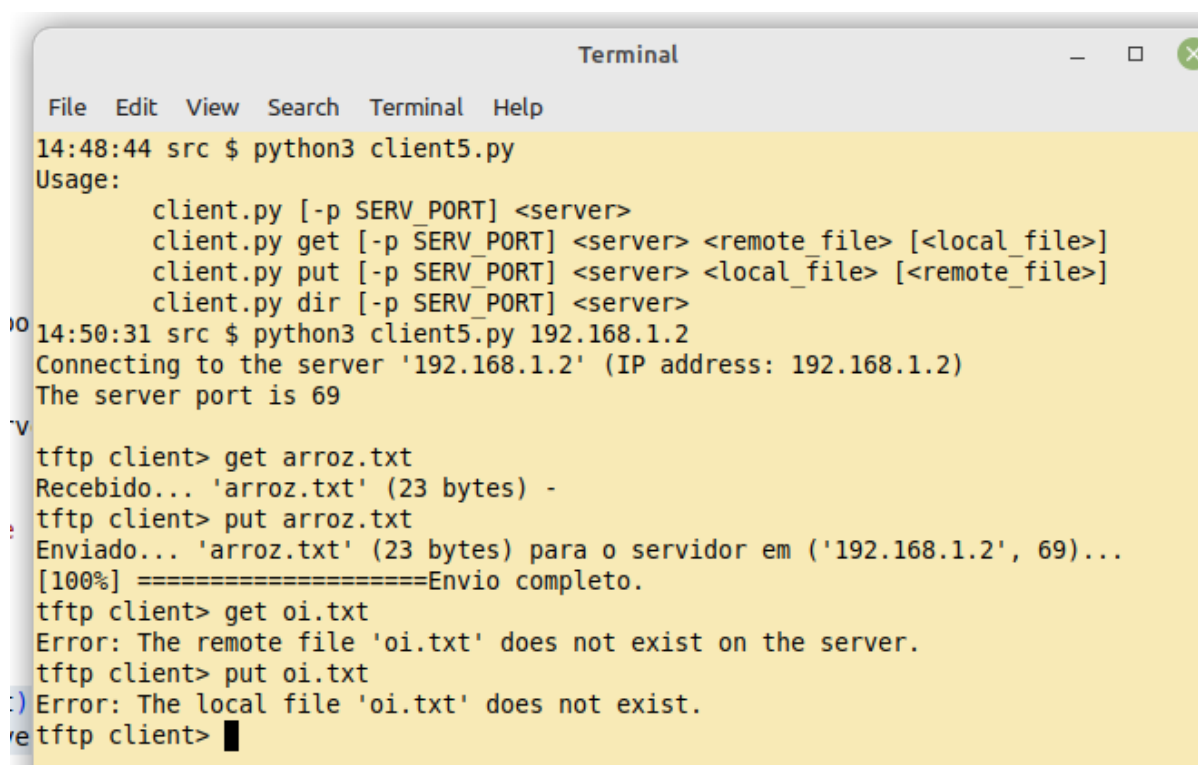
get: Permite ao usuário baixar um ficheiro do servidor.

put: Permite ao usuário enviar um ficheiro para o servidor.

help: Mostra uma mensagem de ajuda com os comandos disponíveis.

quit: Encerra o cliente TFTP.

O código implementa verificação de erros, como a falta de ficheiros locais ou a inexistência de ficheiros remotos no servidor, exibindo mensagens de erro apropriadas. Além disso, garante uma interação contínua com o utilizador, verificando a disponibilidade do servidor e a existência de ficheiros remotos antes de realizar operações de transferência de ficheiros. Se ocorrerem exceções durante a execução, uma mensagem de erro genérica é exibida para o utilizador.

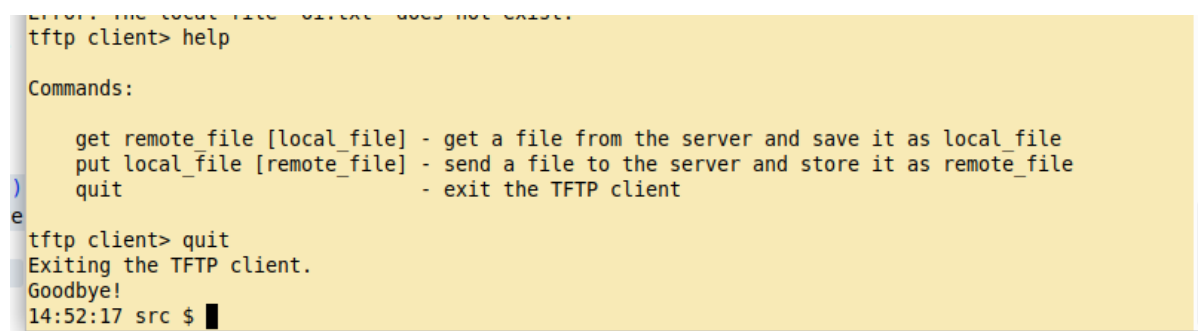


```

Terminal
File Edit View Search Terminal Help
14:48:44 src $ python3 client5.py
Usage:
    client.py [-p SERV_PORT] <server>
    client.py get [-p SERV_PORT] <server> <remote_file> [<local_file>]
    client.py put [-p SERV_PORT] <server> <local_file> [<remote_file>]
    client.py dir [-p SERV_PORT] <server>
14:50:31 src $ python3 client5.py 192.168.1.2
Connecting to the server '192.168.1.2' (IP address: 192.168.1.2)
The server port is 69
tftp client> get arroz.txt
Recebido... 'arroz.txt' (23 bytes) -
tftp client> put arroz.txt
Enviado... 'arroz.txt' (23 bytes) para o servidor em ('192.168.1.2', 69)...
[100%] =====Envio completo.
tftp client> get oi.txt
Error: The remote file 'oi.txt' does not exist on the server.
tftp client> put oi.txt
Error: The local file 'oi.txt' does not exist.
tftp client>

```

Figura 11 - Modo interativo



```

tftp client> help
Commands:
    get remote_file [local_file] - get a file from the server and save it as local_file
    put local_file [remote_file] - send a file to the server and store it as remote_file
    quit                          - exit the TFTP client
tftp client> quit
Exiting the TFTP client.
Goodbye!
14:52:17 src $

```

Figura 12 - Modo interativo

Técnico Especialista em Gestão de Redes e Sistemas Informáticos – TEGRSI12**O Que Ficou por Implementar:**

--> Suporte a Comando dir, a funcionalidade de listar arquivos no servidor não está implementada diretamente.

--> Temporizadores de Pacotes DAT, Não fizemos implementação explícita de temporizadores para retransmissão de pacotes DAT. .

--> Não foi realizado código para o servidor

Conclusão

O desenvolvimento do Cliente TFTP foi uma oportunidade valiosa para explorar os conceitos fundamentais de programação em redes, especialmente ao entender o protocolo TFTP e sua implementação. Conseguimos criar uma aplicação funcional para transferência de arquivos dentro de uma rede local, dentro dos possíveis e não de uma forma tão complexa o quanto desejaríamos.

Reconhecemos que o prazo limitado para a implementação deste projeto foi um desafio adicional. A complexidade do trabalho exigiu um equilíbrio entre aprofundar nosso entendimento do protocolo TFTP e atender a outras responsabilidades do curso. Essa restrição de tempo ressalta a importância de gerenciar efetivamente os recursos disponíveis e estabelecer prioridades claras ao enfrentar projetos complexos dentro de prazos definidos.



Webgrafia

<https://www.python.org/>

<https://github.com/> - beaisasousafernandes@gmail.com

<https://docs.python.org/pt-br/3.7/howto/sockets.html>

<https://datatracker.ietf.org/doc/html/rfc1350>

<https://docs.python.org/3/library/struct.html>

<https://medium.com/@urapython.community/introdu%C3%A7%C3%A3o-a-sockets-em-python-44d3d55c60d0>



Técnico Especialista em Gestão de Redes e Sistemas Informáticos – TEGRSI12