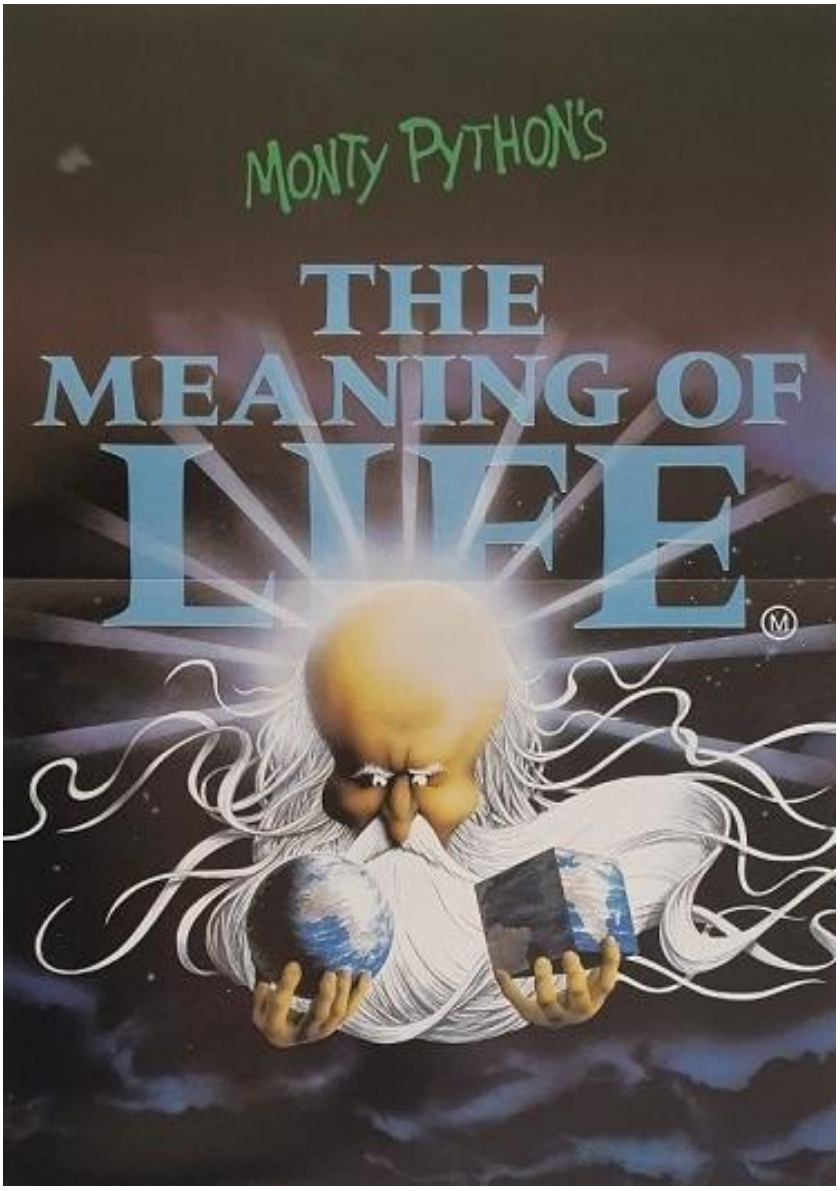
RELATÓRIO 3
TFTPY: CLIENTE TFTP



REALIZADO POR:
MARÍLIA PINHO
RODRIGO SEQUEIRA

PARA A:
UFCD 5119, CET 21180
FORMADOR – JOÃO GALAMBA

Índice

Índice de Ilustrações

INTRODUÇÃO E OBJETIVOS

Neste projeto iremos desenvolver uma aplicação que permite a transferência de ficheiros, entre um cliente e um servidor, utilizando para o efeito o TFTP.
Este protocolo vai permitir que um cliente receba ou envie um ficheiro para um servidor remoto. Neste caso o protocolo de transporte que iremos utilizar será o protocolo UDP, da camada de transporte do modelo TCP/IP.

Análise

Nesta secção apresentamos os diagramas de envio e receção de ficheiros entre cliente e servidor remoto.

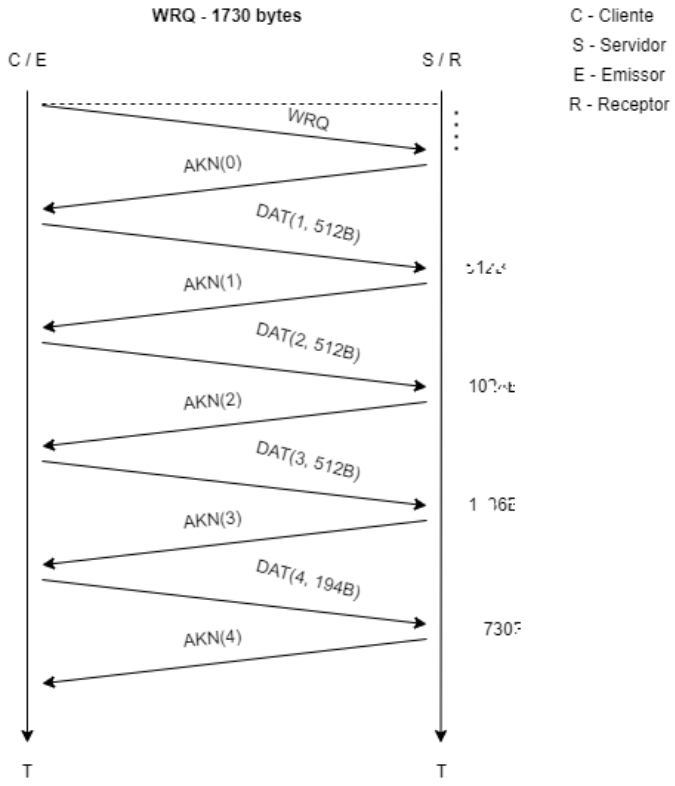


Figura 1 – Diagrama WRQ 1730 Bytes

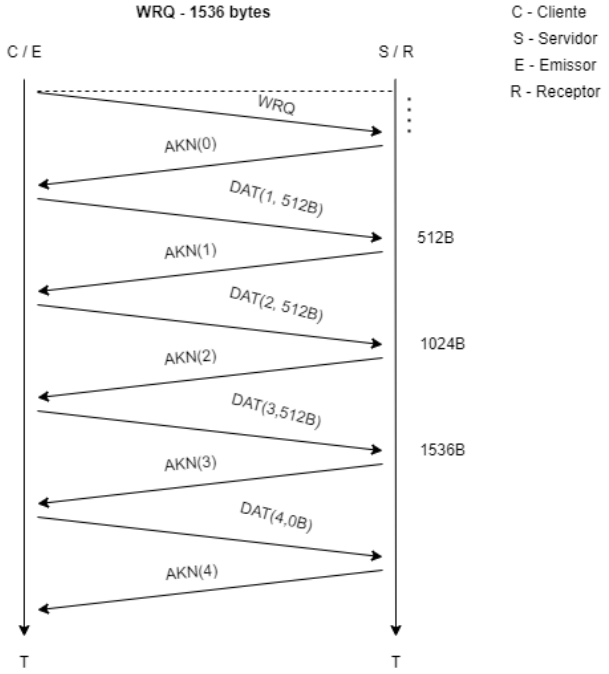


Figura 2 – Diagrama WRQ 1536 bytes

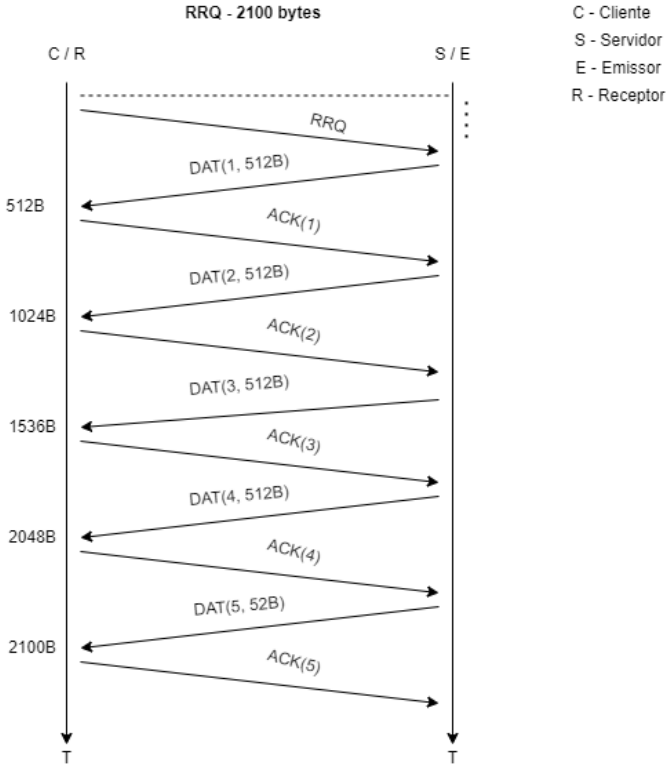


Figura 3 – Diagrama RRQ 2100 bytes

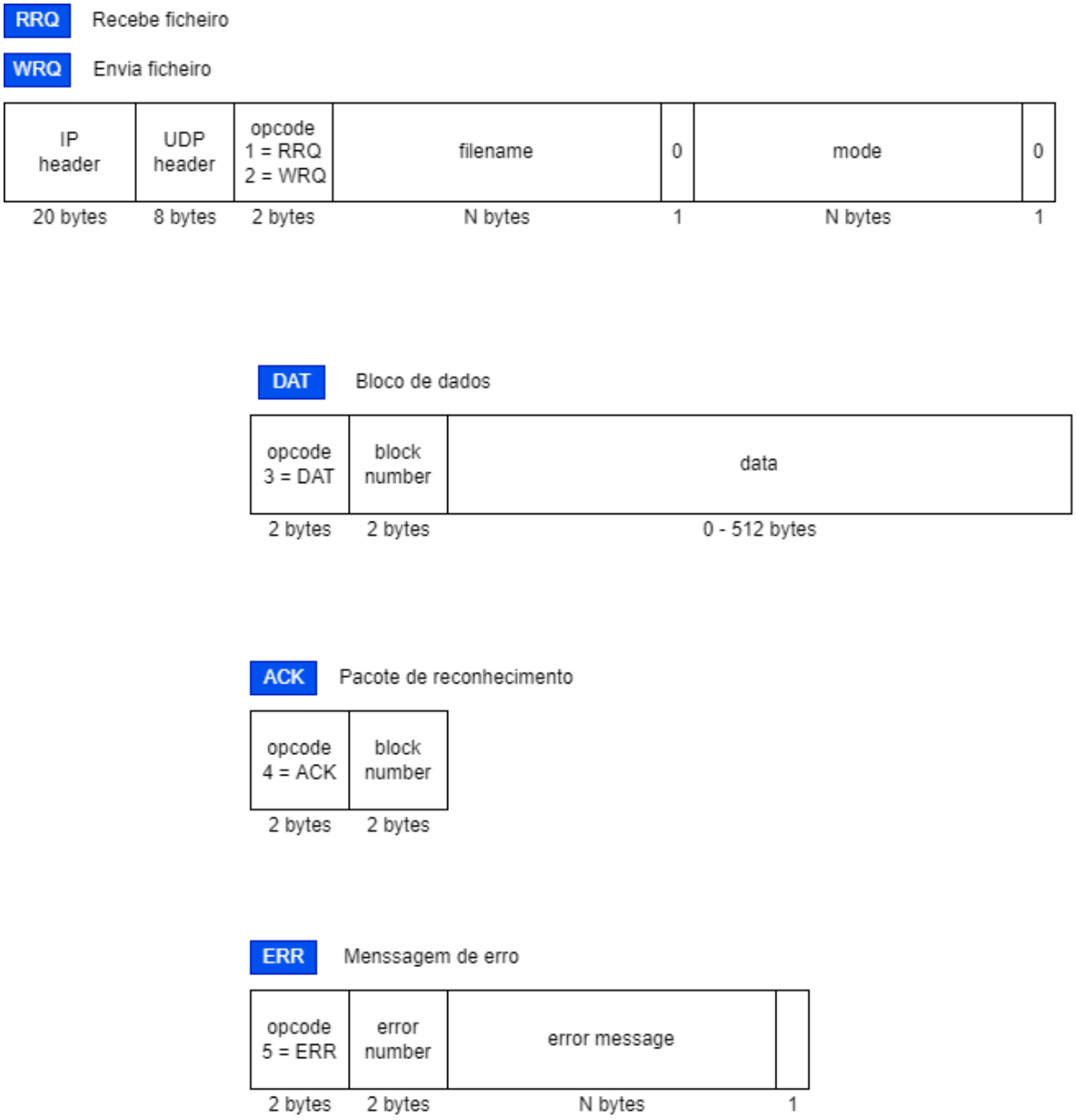


Figura 4 – Diagrama dos pacotes

Legenda dos campos dos pacotes do diagrama acima

RRQ

Opcode : O campo de opcode do pacote TFTP RRQ tem 2 bytes de comprimento. Os pacotes TFTP RRQ têm um opcode de "1".
Nome de arquivo : O pacote TFTP RRQ tem um nome de ficheiro, que é o nome do ficheiro solicitado. O nome do ficheiro é uma cadeia de tamanho variável. O nome do ficheiro é uma sequência de bytes ASCII.
0s : O nome de arquivo é terminado por um byte de todos os zeros. Uma vez que o campo de nome de ficheiro é variável em comprimento, um byte de todos os 0s indica o fim do nome de arquivo.
Modo : O modo é outro campo de comprimento variável no pacote TFTP RRQ. O campo de modo contém informações sobre o modo de transferência de dados. Três modos de transferência de dados são netascii (o ficheiro é transferido como linhas de caracteres para um ficheiro ASCII, cada um terminado por uma devolução de transporte), octeto (cru 8-bits) e correio (para envio de ficheiros para um endereço de e-mail).
0s - O fim dos pacotes TFTP RRQ é com um campo de byte comprimento de todos os 0s.

WRQ

Opcode : O campo de opcode do pacote TFTP WRQ tem 2 bytes de comprimento. Os pacotes TFTP WRQ têm um opcode de "2".
Nome de arquivo : O pacote TFTP WRQ tem um nome de ficheiro, que é o nome do ficheiro solicitado. O nome do ficheiro é uma cadeia de tamanho variável. O nome do ficheiro é uma sequência de bytes ASCII.
0s : O nome de arquivo é terminado por um byte de todos os zeros. Uma vez que o campo de nome de ficheiro é variável em comprimento, um byte de todos os 0s indica o fim do nome de arquivo.
Modo : O modo é outro campo de comprimento variável no pacote TFTP WRQ. O campo de modo contém informações sobre o modo de transferência de dados. Três modos de transferência de dados são netascii (o ficheiro é transferido como linhas de caracteres para um ficheiro ASCII, cada um terminado por uma devolução de transporte), octeto (cru 8-bits) e correio (para envio de ficheiros para um endereço de e-mail).
0s - O fim dos pacotes TFTP WRQ é com um campo de byte comprimento de todos os 0s.

DAT

Opcode : O campo de opcode do pacote TFTP DATA tem 16 bits (2 bytes) de comprimento. Os pacotes TFTP DATA têm um valor opcode de "3".
Número do bloco : O campo número de bloco nos pacotes de dados começa com um e, em seguida, aumenta sequencialmente um para cada novo pacote. Este tipo de numerações permite que as aplicações TFTP se identifiquem entre novos pacotes de dados e duplicados.
DADOS : O tamanho do campo DATA é de 0 a 512 bytes. O tamanho do campo DATA de todos os pacotes são 512 bytes de comprimento, exceto o último pacote. Se o tamanho do campo DATA for entre 0 a 511 bytes, este é o último pacote DATA pertence à transmissão atual.

ACK

Opcode : O campo de opcode do pacote TFTP ACK tem 2 bytes de comprimento. Os pacotes TFTP ACK têm um opcode de "4".
Número do bloco : O campo número de bloco do pacote TFTP ACK também tem 2 bytes de tamanho. O campo número de bloco do pacote TFTP ACK contém o número do bloco DATA recebido. O número de bloco do ACK de um pacote DE TFTP WRQ é 0.

ERR

Opcode : O campo de opcode do pacote TFTP ERROR tem 2 bytes de comprimento. Os pacotes TFTP ERROR têm um valor opcode de "5".
Código de erro : O código de erro do pacote TFTP ERROR é um inteiro que mostra o tipo de erro.
Error Message : O campo Error Message tem um tamanho variável. O campo error Message está no ASCII.
0s : O pacote TFTP ERROR é terminado com um byte de todos os zeros.

DESENHO E ESTRUTURA

Nesta secção apresentamos dois fluxogramas de alto nível.

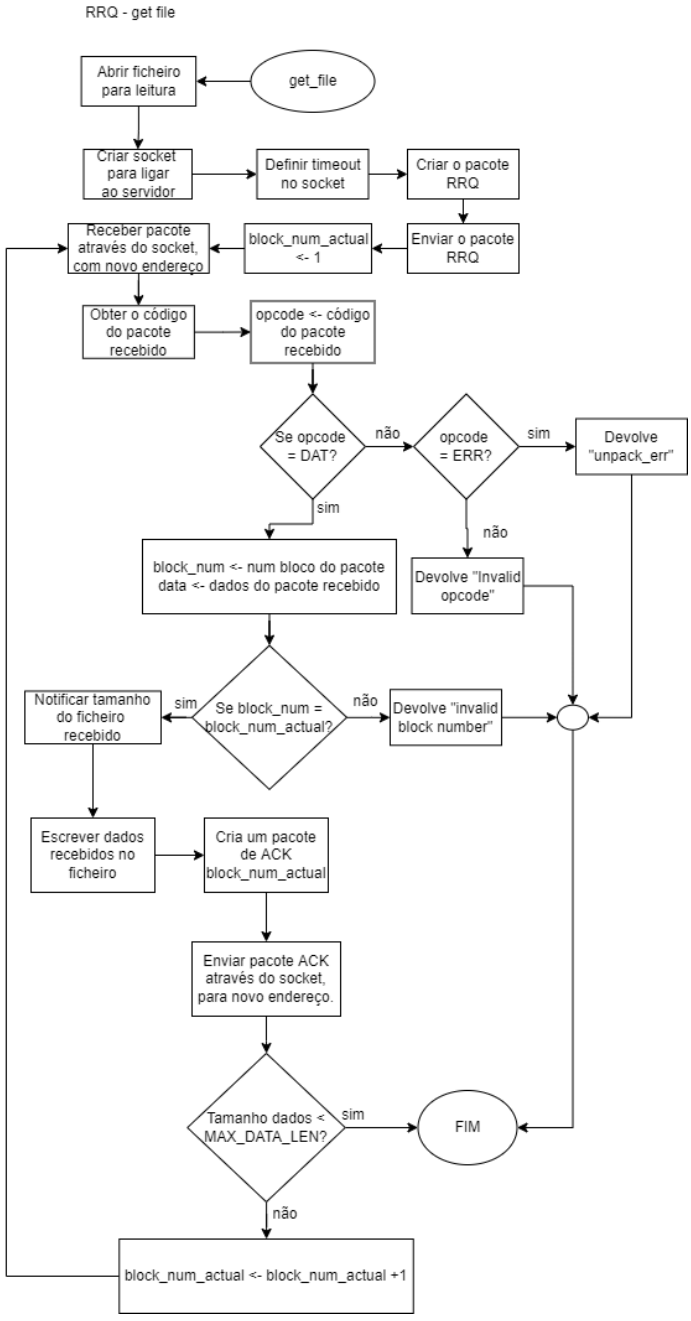


Figura 5 - receção de um ficheiro (RRQ) – cliente » servidor.

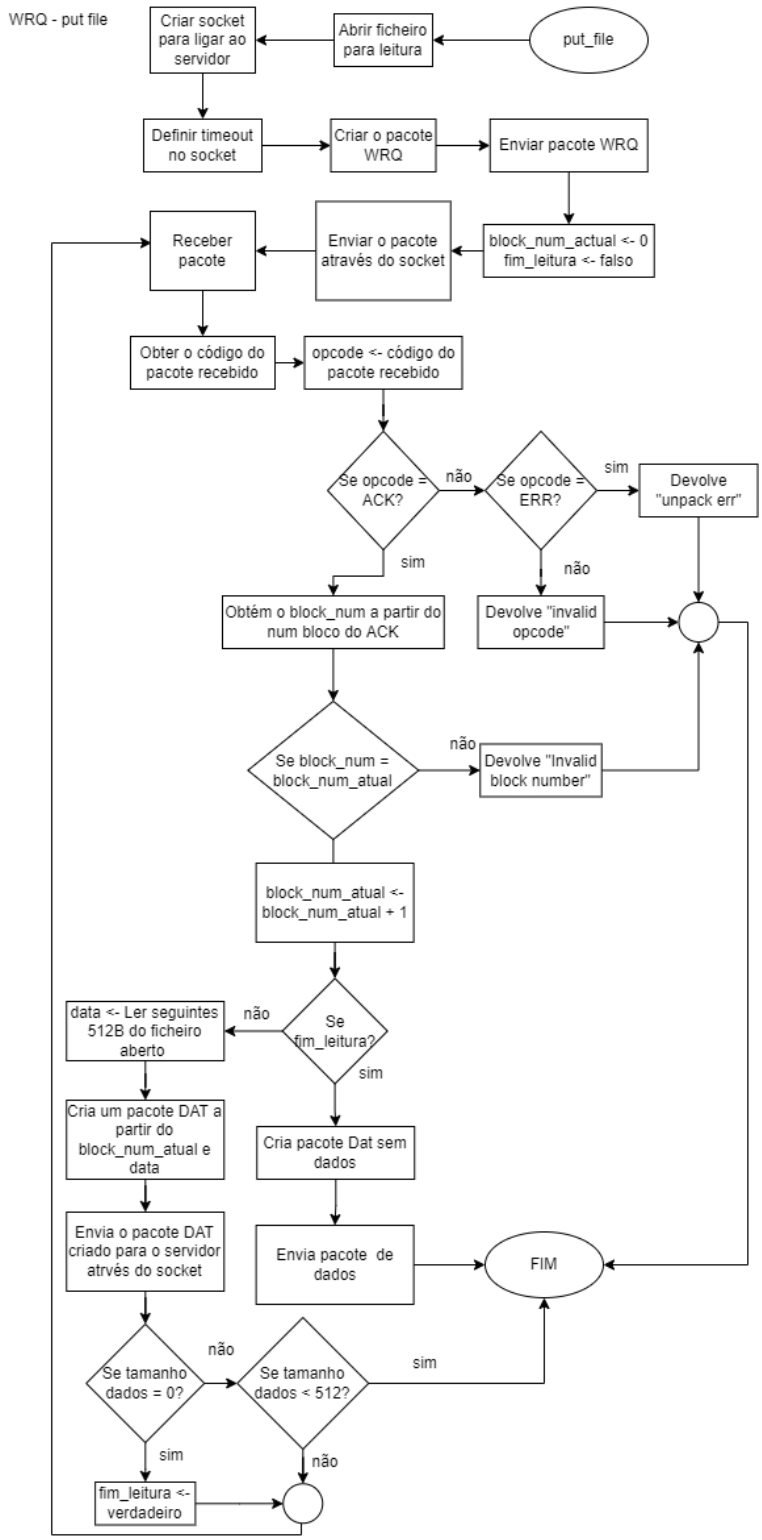


Figura 6 - Envio de um ficheiro (WRQ) – Cliente » servidor.

IMPLEMENTAÇÃO

.Estes programas foram editados no visual studio code version 1.58.2 através da VM mint_20_1_cin version ubuntu(64-bit). O interpretador de Python é o Python 3 que temos instalado na: /usr/bin/python3

Neste projeto implementamos um programa cliente que através do TFTP, envia e recebe ficheiros para e do servidor. O TFTP utiliza o protocolo UDP, mais rápido mas menos confiável, utilizando o porto 69 para iniciar a comunicação com o servidor. O porto 69 não será a utilizado pelo servidor para responder ao cliente, e no decorrer da transmissão não voltará a ser utilizado pelo cliente, pois este portal tem que ficar desocupado para pedidos de outros clientes. Assim sendo a transmissão passa a decorrer por outro portal anunciado pelo servidor, no pacote de mensagens que envia para o cliente.
No caso do TFTP iremos ter 5 tipos de pacotes, o RRQ, WRQ, DAT, ACK e ERR.
No cliente executámos os comandos **put** (envia um ficheiro), **get** (recebe um ficheiro), **help**, **dir** e o **quit**. Qualquer outro comando utilizado fará o programa emitir uma mensagem de erro.

O comando **put** exige um caminho para o ficheiro a ser enviado. Se o ficheiro não existir o programa dará um erro. Esse ficheiro será aberto para leitura em modo binário. Criámos um socket do tipo datagrama. O programa cria um pacote de dados WRQ, que será enviado ao servidor através do socket. Caso seja definido pelo utilizador o nome do ficheiro de destino, ele será também transmitido ao servidor no pacote WRQ.
Não será necessário fechar o ficheiro quando a transferência terminar, pois utilizando o “with”, para a abertura do ficheiro, o mesmo será fechado automaticamente.

No **get** exige a identificação do ficheiro a receber. Poderá receber ainda a identificação que deverá vir a ter o ficheiro local. Caso ela não exista, o ficheiro local terá o nome do ficheiro original. Este ficheiro é aberto em modo de escrita binária. Criámos um socket do tipo datagrama. O programa cria um pacote de dados RRQ, que será enviado ao servidor através do socket.
Não será necessário fechar o ficheiro quando a transferência terminar, pois utilizando o “with”, para a abertura do ficheiro, o mesmo será fechado automaticamente.

Nós ativamos o time out. No caso de ocorrer um erro de time out durante a execução da “sock.recvfrom” vamos considerar um erro de NetworkError, não fazendo distinção por ser provocado pelo time out.

Implementamos um procedimento get_dir, mas como não está implementado no lado do servidor, criámos então um get_dir_nova que apresenta o conteúdo de um documento que está no servidor O documento contém a listagem da diretoria no servidor.
O Formador tem que descomentar o get_dir quando for experimentar o programa com a implementação correspondente do lado do servidor.

Não conseguimos utilizar a função *get_server_info* , pelo que não conseguimos obter o nome do servidor a partir do seu IP, ou o IP a partir do nome do servidor.

Neste projeto não foi implementado o servidor.

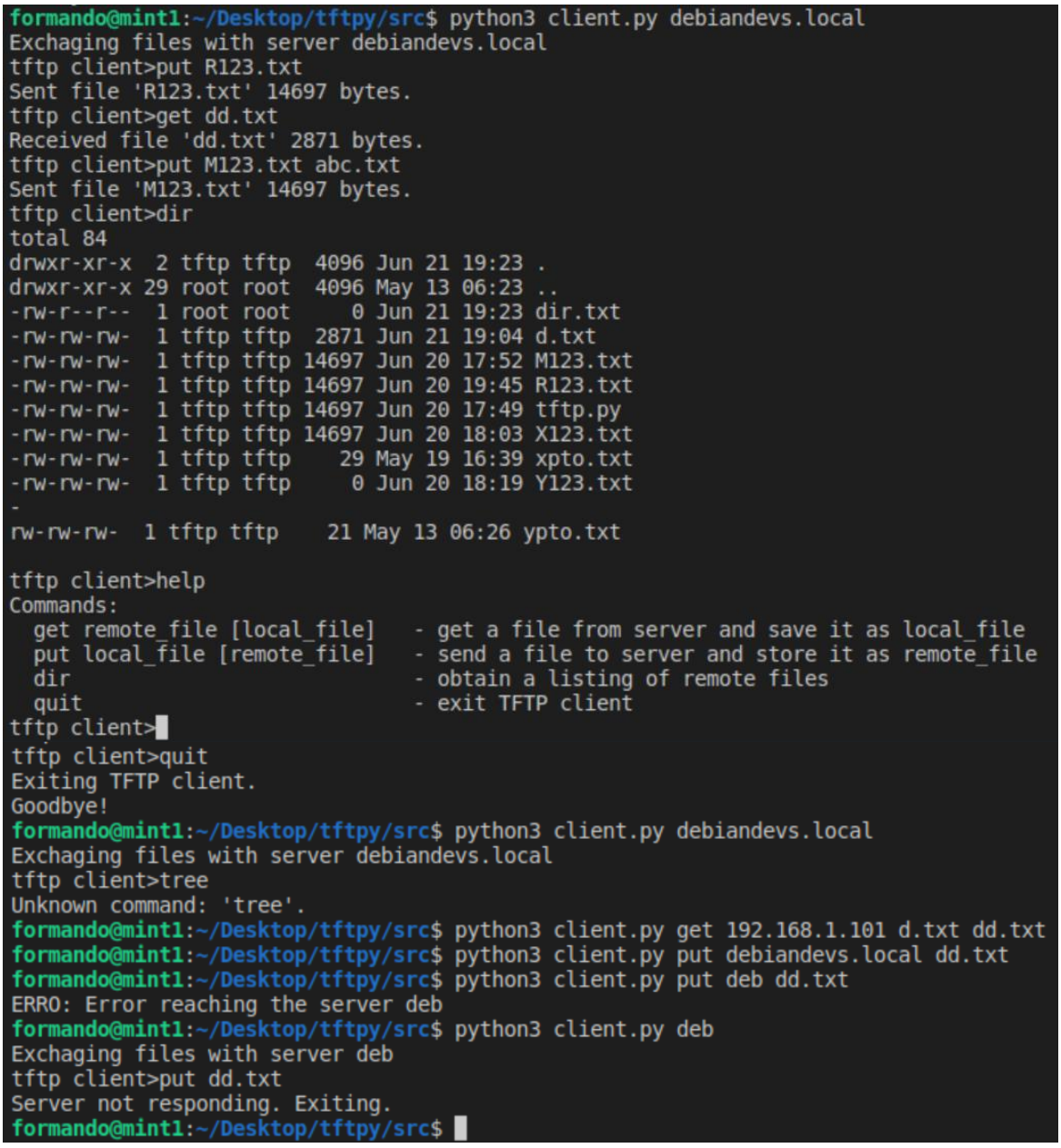


Figura 7 – Exemplo da utilização do programa cliente.py

Cliente TFTP

Log de Erros de sistema:

Sempre que ocorre um erro de parâmetros, o programa mostra a respetiva mensagem e termina com um código de erro. Eis os seus valores e causas:

- Quando o número do bloco é diferente do número do bloco esperado assinala o erro **"Invalid block number".**
- Quando o valor do opcode não é um dos esperados assinala o erro **"Invalid opcode".**
- Quando o valor do opcode é um pacote de erro assinala o erro **"unpack_err".**
- Se o ficheiro não for uma string binária assinala o erro **"Invalid filename"**
- Se o modo for diferente do *octet* assinala o erro **"Invalid mode"**
- Se o comprimento do campo *data* do pacote for superior ao estipulado assinala o erro **"Invalid data lenght"**.
- Se o número de campos não tiver a dimensão esperada assinala o erro **"Invalid packet lenght".**
- Se o nome do opcode não for o esperado assinala o erro **"Unrecognized opcode".**
- Se o nome do hostname não estiver correto assinala o erro **"Invalid hostname".**
- Se o endereço IP não estiver acessível a partir do servidor assinala o erro **"Unknown server".**

Bibliotecas

Procedemos à utilização das seguintes bibliotecas:

Sys (version 3.9.6) - O módulo sys permite-nos utilizar o sys.argv e o sys.exc_info.

sys.argv - É uma função que permite que possamos usar os argumentos passados para o Python a partir da linha de comando.

sys.exc_info – É uma função que devolve um 3-tuple com a exceção, o parâmetro da exceção, e um objeto de retrocesso que identifica a linha de Python que elevou a exceção

ipaddress - é um módulo para inspecionar e manipular endereços IP, a primeira coisa que temos de fazer é criar alguns objetos. Podemos usar o ipaddress para criar objetos a partir de strings e inteiros.

re - Este módulo fornece operações para correspondência de expressões regulares semelhantes às encontradas em Perl. O nome do módulo vem das iniciais do termo em inglês *regular expressions.*

struct – executa conversões entre valores Python e C structs representados como objetos Python bytes. Isto pode ser utilizado no tratamento de dados binários armazenados em ficheiros ou a partir de ligações à rede, entre outras fontes. Utiliza cordas de formato como descrições compactas do layout das estruturas C e da conversão pretendida a partir dos valores Python.

socket – São essências para o envio de mensagens através de uma rede, Sendo as aplicações mais comuns as de cliente-servidor. Nesta biblioteca podemos encontrar os seguintes métodos: socket.sendto, socket.recvfrom, socket.settimeout, entre outras.

sock.sendto – Método que transmite mensagem UDP.

sock.recvfrom – Método que recebe mensagem UDP.

socket.gethostbyname_ex – Método que devolve o nome do host.

socket.gethostbyaddr – Método que devolve o endereço IP do host.

sock.settimeout – Método que ativa o time out do socket.

socket.herror – Exceção que pode ocorrer no socket quando se tenta buscar o hostname do servidor a partir do IP. No nosso caso acontece sempre, por isso não conseguimos usar *socket.gethostbyaddr.*

socket.gaierror - Exceção que pode ocorrer no socket quando se tenta buscar o IP do servidor a partir do hostname. No nosso caso acontece sempre, por isso não conseguimos usar *socket.gethostbyname_ex.*

Usámos a biblioteca **docopt** para fazer o reconhecimento dos parâmetros. Para a instalar tivemos que utilizar o comando:
$ pip install docopt==0.6.2.

CONCLUSÃO

Neste projeto executámos um programa em que um cliente, através do protocolo TFTP envia e recebe ficheiros de um servidor. Seguimos o que é proposto no enunciado, ficando por implementar o servidor

Anexo I - UDP

O protocolo UDP, assim como o protocolo TCP pertencem à camada 3, no modelo TCP/IP, ou seja à camada de transporte. No entanto existe uma diferença entre estes dois protocolos, o protocolo UDP é mais simples e não fornece garantia na entrega de pacotes. Na transmissão de pacotes entre emissor e recetor, não vai existir a confirmação de receção dos dados transmitidos. Assim se um pacote se perder, não vai haver a solicitação de reenvio, e para o recetor esse pacote simplesmente não existe. No entanto podemos colmatar essa falha na camada de aplicação, onde poderemos definir algumas regras que tornarão o envio mais seguro.
Já no caso do protocolo TCP, o mesmo garante-nos uma entrega de todos os pacotes ordenadamente, através do three way handshake (SYN, SYN-ACK, ACK). Sendo assim sempre que uma comunicação, entre um cliente emissor e um servidor recetor, é iniciada o servidor terá sempre de responder a um SYN do cliente, com um SYN-ACK, ao qual o cliente responderá com um ACK, que terá um número de confirmação igual ao número de sequência do recetor acrescido de 1. Sem este sistema a funcionar entre cliente e servidor, a comunicação falhará.

WEBGRAFIA

https://www.omnisecu.com/

https://docs.python.org/