NOVA SCHOOL OF SCIENCE & TECHNOLOGY
DEPARTAMENTO DE INFORMÁTICA

Transfer File Service
(TPC1 – Redes de Computadores)

1 Introduction

The objective of this TPC is to implement the client and server programs of a one-way **File Transfer Service**. In short, the client sends commands (**dir** and **get**) to the server to identify and download available files. The server should be able to handle several clients simultaneously. Files that can be downloaded by clients (using the "**get**" command) are stored in a directory on the server that clients can consult (using the "**dir**" command). Files are transferred by the server in 512-byte blocks.

The server is launched before the client using the following command:

```
python3 server.py server_port
```

where **server_port** is the TCP port number that the server listens to for client connections.

Once started, it simply runs in an infinite loop waiting for incoming connections until it is terminated.

Clients are invoked by a command line such as:

```
python client.py server_addr  server_port
```

where **server_addr** is the IP address of the server, **server_port** is the server's port.

2 Client-Server Interaction Protocol

The packets exchanged between clients and the server contain more complex Python data structures than the simple data types you have worked with so far. The serialization and deserialization methods available in the Python **pickle** module - namely **dumps** and **loads** - make it easy to send such data structures to and retrieve them from network sockets.

For more information and examples on the use of the pickle module you can check the following references:

https://docs.python.org/3/library/pickle.html

https://realpython.com/python-pickle-module/

The client and server communicate with each other by exchanging data packets. The figure below illustrates the format of these packets.

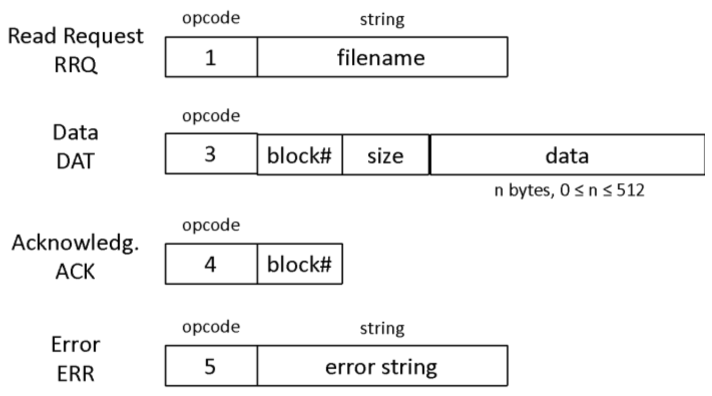

Figure 1 - Packet format.

Each packet begins with an opcode (integer). Filenames and error messages are specified in ASCII. Block number and size are represented by integers. Data is sent in blocks of 512 bytes or less.

2.1 Downloading files (**get** command)

When the client wants to retrieve a file, it must send to the server an **RRQ** packet. The receive operation works as follows:

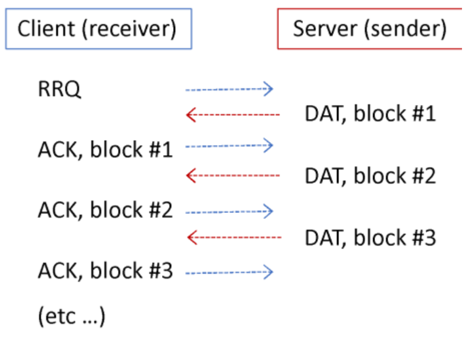

Figure 2- Client asks to receive a file from server (download).

If the amount of data sent in each packet is equal to the block size (512 bytes), it means that more packets will follow. If it is less than the block size, then that packet is the final one. Receiving any other type of packet during file transmission means that an error occurred or that a protocol error of some sort occurred.

The connection should be closed if:

- The expected block number of a DAT or ACK packet is incorrect.
- There is a protocol error (an unexpected packet type is received).

2.2 Directory listing (**dir** command)

If the server receives an **RRQ** packet with an empty filename (“”), it should generate an ASCII formatted directory listing and send it to the client as an ordinary file. The client should redirect the incoming directory data to the standard output.

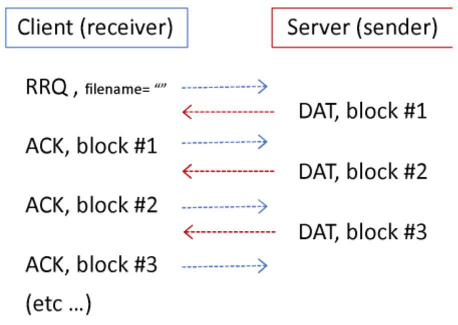

Figure 3 - client asks for a directory from server

Note: If the server directory contains three files, this means that four DAT blocks should be sent by the server—three with the file names (one block per file), and one additional block with empty data. The last block is used by the client to signal that all the directory information has been transmitted.

2.3 End session (**end** command)

The client execution terminates when the user gives the command **end**. It should also close the connection to the server.

2.4 Transmission Errors

The client and the server should be prepared to handle error cases.

For example:

- The client requests a file that does not exist on the server.

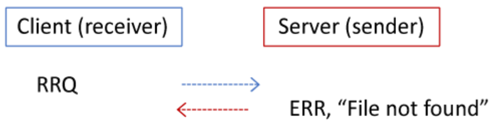

Figure 4 - Client asks for a non-existest file from the server

- A client protocol error.

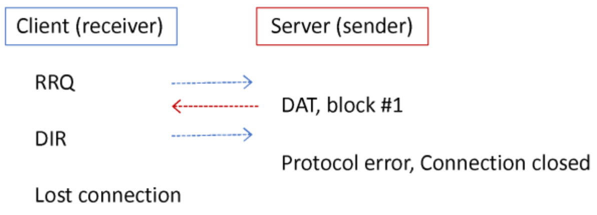

Figure 5 - A client protocol error

3 The Server

When the server starts, it should open a socket on the given port. If successful it should print the message: "*Server is running*". If an error occurs and the server is unable to start, it should display the error message "*Unable to start server*" and exit.

When a client first connects to the server, the server should launch a new **thread** to handle the client.

For information regarding threads consult lecture **T03-16Sep.pdf** (additional information https://realpython.com/intro-to-python-threading/)

After establishing a connection and starting a client thread, the server should send a message to the client identifying itself. It should be done by sending a DAT packet with a greeting message ("*Welcome to <ip_addr> file server*"). The client should then respond by sending an ACK.

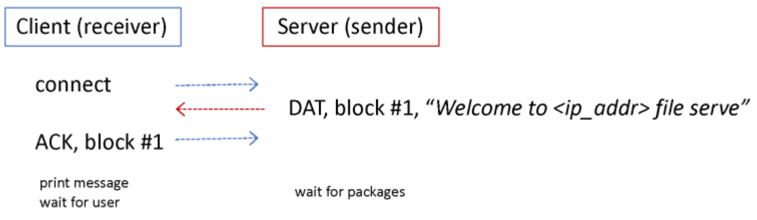

Figure 6 - Client initial connection to the server.

4 The Client

The client must support the following commands:

- **dir** – this command obtains the contents of the directory where the server stores files and lists it one per line.
  Syntax: *dir*
- **end** – this command ends the execution of the client, closing the connection to the server.
  Syntax: *end*
- **get** – this command downloads a file from the server, this means to transfer a file from the server file system to the client´s file system.
  Syntax: *get <remote_filename> <local_filename>*
  Errors: (1) invalid number of arguments;
  (2) a file with the indicated name already exists on the client;
  (3) the indicated file does not exist on the server;

  Example: *get test1.pdf my_test1.pdf*

If an invalid command is typed, the client should display the error "*Unknown command*".

The client tries to establish a connection with the server. If a connection is not established, an error message should be generated ("*Unable to connect with the server*"). If the connection is successful, the client should print a connection message ("*Connect to server*"), followed by the server's greeting message, and prompt the user for a command.

Example of a client execution:

```
%python3 Tclient.py 127.0.0.1 12000
Connect to server
Welcome to 127.0.0.1 file server
client>
...
client> xxx
Unknown command
...
client> get test.pd test1.pdf
File not found
...
client> get test.pdf test1.pdf
File transfer completed
...
client> end
Connection close, client ended
```

The above example illustrates the expected behavior of your client program. The client should start by printing a connection message ("*Connect to server*"), then the server's greeting message, and finally, it should prompt the user for a command.

Your client program will have a structure like this:

1- Open a socket connection with the server.
2- Establish the connection by receiving the server message and sending the ACK packet.
3- Read a command from the user.
4- Send data to the server, using the protocol.
5- Wait for the server response.
6- Handle data sent by the server (e.g., a directory listing, an error message, or a file)
7- Repeat steps 3-7 until the user ends the program

5 Delivery

Delivery details will be sent later. The deadline for delivery is 23:59 on 29st September, 2025.