SimpFT protocol

Kenan Augsburger & Mário Ferreira

December 03, 2024

Contents

1. SimpFTP

Task 1

add context

1.1. Section 1 - Overview

The SimpFTP (Simple File Transfer Protocol) is a communication protocol that allows a client to interact with files on a server.

1.2. Section 2 - Transport protocol

The SimpFT protocol is a text based protocol. It uses TCP to ensure reliability. The default port is `1234`.

Thee protocol has three kinds of messages:

- `Actions` which are encoded in UTF-8 and use the following pattern `<ACTION> <ARG>\n` where `\n` is used as a delimiter.
- `Statuses` which are encoded in UTF-8 and use the following pattern `<CODE><EOT>` where `EOT` (`0x04` character in ASCII table) is used as a delimiter
- `Datas` which is the binary content of a transferred file delimited by an end of transmission character `EOT`.

The initial connection must be established by the client.

Once the server accepts the connection, the client can send `Actions` to interact with files on the server.

When an `Action` is used to transfer a file from the server to the client, the server response should be a `Status` followed by the `Data` of the file if there is no error.

When an `Action` is used to transfer a file from the client to the server, the `Data` should follow right away and the server responds with a status once the file is sent.

The client can do the following actions:
- List the files and folders
- Get a file from the server
- Store a file on the server
- Delete a file from the server

The `Status` values use the values defined by the c standard library in errno.h° or `0` to indicate success.

When an invalid message is received, the server should answer with `ENOTSUP`.

When the status represents an error, the server terminates the connection by sending `<CODE>\x04` asfsfd `\x04` is the `EOT` (End Of Transmission) character.

1.3. Section 3 - Messages

Even though you will find in the examples bellow the name of the actions in uppercase, the server accepts them in any form (upper, lower, mix of both, etc…).

The valid messages are:

- `LIST` - List the contents of directories
- `GET` - Downloads a file
- `PUT` - Create a new file
- `DELETE` - Delete a file

1.3.1. LIST

The client sends a list request to the server to show the list of files and folders at the specified path.

1.3.1.1. Request

```text
LIST <PATH>
```

If the path is empty, the working directory of the server will be used.

1.3.1.2. Response

```text
<CODE>
```

```text
foldera/:folderb/:filea:fileb
```

On a successful request, the server answers with the code `0`, followed by a colon separated list of files and folders. Each folders have a trailing `/` appended to them.

On error, only the error code is sent. `<CODE>` matches one of:

- `EACCES`
- `ENOENT`
- `ENOTDIR`
- `EINVAL`

1.3.2. GET

The client sends a get request to the server to download a file.

Downloading directories

Notice that directories can not be downloaded. If you want to download the content of a directory you have to to list its contents to fetch the name of the files and then download them.

1.3.2.1. Request

```text
GET <REMOTE_PATH>
```

- `REMOTE_PATH` : The path of the file to be downloaded

1.3.2.2. Response

```text
<CODE>
```

```text
<FILE_SIZE>
```

```binary
<DATA>
```

On a successful request, the server answers with the code `0`, followed by the size (a non-negative integer value) of the file as well as its content in binary form. All the 3 connections are delimited by the `EOT` character.

On error, only the error code is sent. `<CODE>` matches one of:

- `EACCES`
- `ENOENT`
- `EISDIR`
- `EINVAL`

1.3.3. PUT

The client sends a put request to the server to upload a file or create a directory.

1.3.3.1. Request

```text
PUT <PATH> <FILE_SIZE>
```

The first part of the request provides the path to the file or directory on the server. A trailing `/` indicates that a directory should be created and no size should be included.

```binary
<DATA>
```

If the path doesn't end with a `/`, the rest of the request contains the file content in binary.

Warning

Notice that the file is sent in two requests, the first one to create the file and the second one to send its content.

1.3.3.2. Response

```text
<CODE>
```

On a successful request, the server answers with the code `0` indicating that the file or directory was created successfully.

On error, only the error code is sent. `<CODE>` matches one of:
- `EACCES`
- `EFBIG`
- `EISDIR`
- `ENOENT`
- `EINVAL`

1.3.4. DELETE

The client sends a delete request to the server to delete a file.

1.3.4.1. Request

```text
DELETE <PATH>
```

Where path is the path to the file or directory to delete.

If the path points to a directory, the whole directory is removed recursively.

1.3.4.2. Response

```text
<CODE>
```

On a successful request, the server answers with the code `0` indicating that the file or folder was removed successfully.

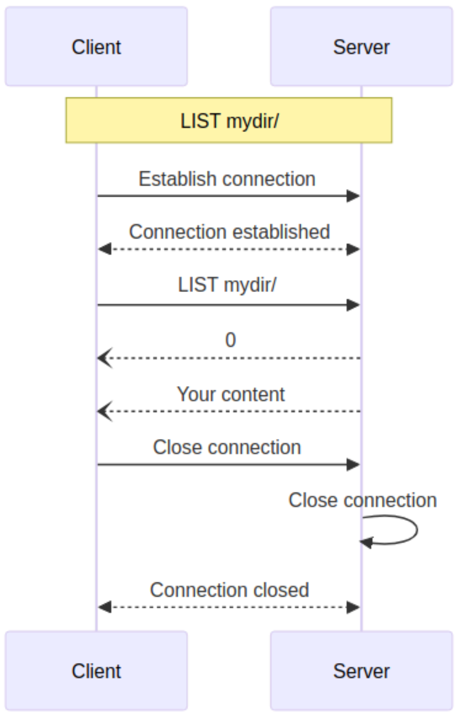
On error, only the error code is sent. `<CODE>` matches one of:

- `EACCES`
- `ENOENT`
- `EINVAL`

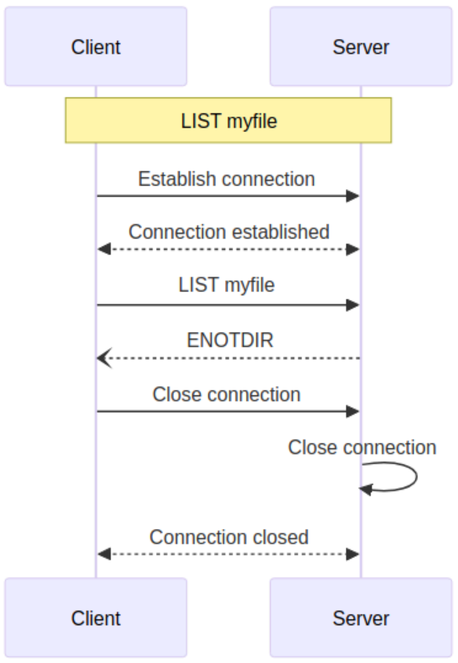
1.4. Section 4 - Examples

1.4.1. LIST

1.4.1.1. OK

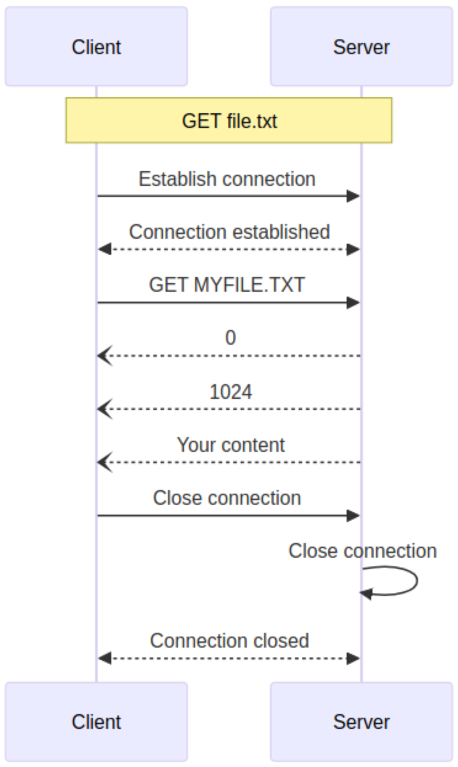


1.4.1.2. ERROR

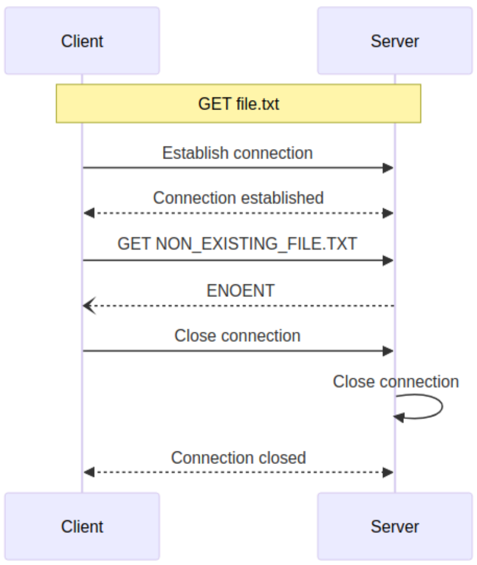


1.4.2. GET

1.4.2.1. OK

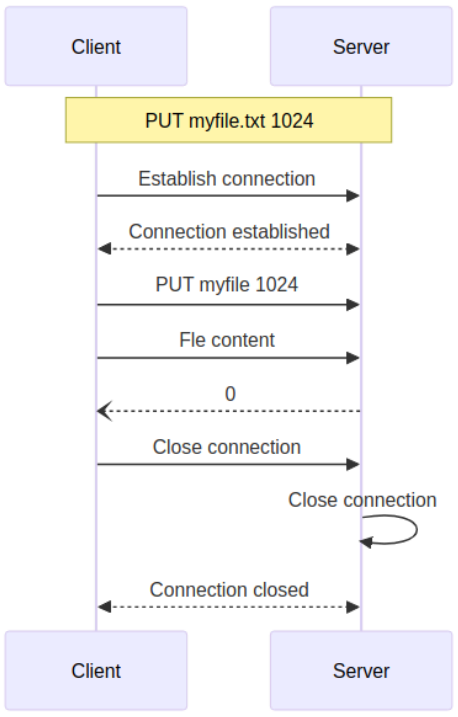


1.4.2.2. ERROR

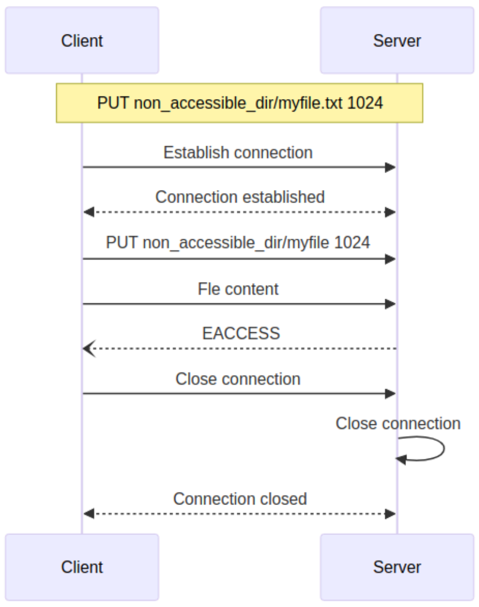


1.4.3. PUT

1.4.3.1. OK

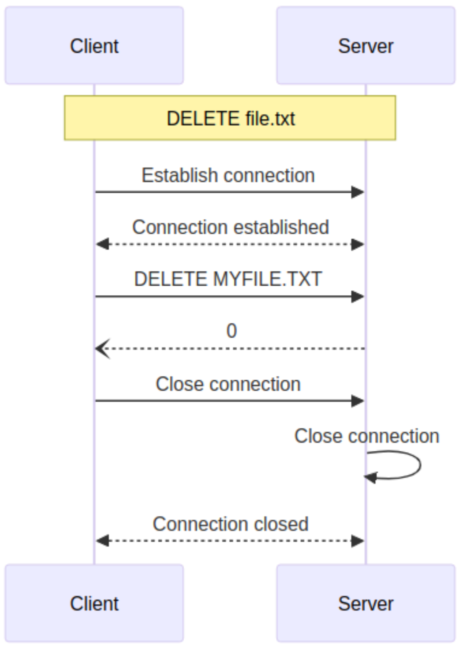


1.4.3.2. ERROR

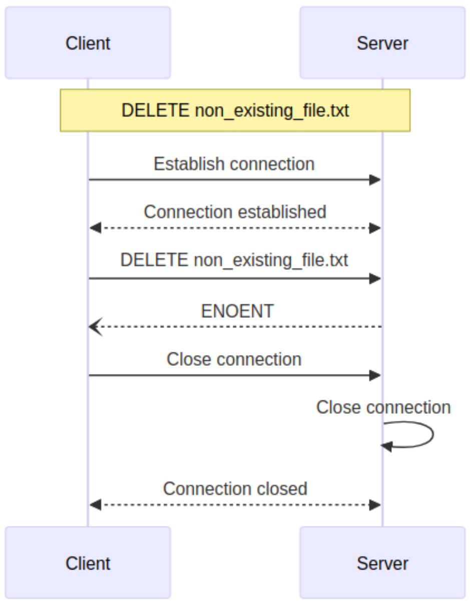


1.4.4. DELETE

1.4.4.1. OK



1.4.4.2. ERROR


Client
Server
DELETE non_existing_file.txt
Establish connection
Connection established
DELETE non_existing_file.txt
ENOENT
Close connection
Close connection
Connection closed
Client
Server