SimpFT protocol

Kenan Augsburger & Mário Ferreira

November 20, 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> \n` where `\n` 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 sens `Actions` to interact with files on the server.

> **Task 2**
>
> If there is enough time, add an authentication step in the connection process

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

> **Task 3**
>
> If there is enough time, add a move action

The `Status` values use the values defined by the c standard library in errno.h°

When an invalid message is received, the server should answer with `ENOTSUP` and flush its buffer.

1.3. Section 3 - Messages

1.3.1. List files

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

```
1 LIST <PATH>                                                    text
```

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

1.3.1.2. Response

```
1 <CODE>                                                         text
```

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

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. The code matches one of:
- EACCES
- ENOENT

1.3.2. Get file

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

1.3.2.1. Request

```
1 GET <PATH>                                                     text
```

1.3.2.2. Response

```
1 <CODE>                                                         text
```

```
1 <DATA>                                                         binary
```

On a successful request, the server answers with the code `0`, followed by the content of the file in binary form.

On error, only the error code is sent. The code matches one of:
- EACCES
- ENOENT
- EISDIR

1.3.3. Put file

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

1.3.3.1. Request

```
1  PUT <PATH>                                                    text
```

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.

```
1  <DATA>                                                        binary
```

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

1.3.3.2. Response

```
1  <CODE>                                                        text
```

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

On error, the code matches one of:

- EACCES
- EFBIG
- EISDIR
- ENOENT

1.3.4. Delete file

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

1.3.4.1. Request

```
1  DELETE <PATH>                                                 text
```

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

```
1  <CODE>                                                        text
```

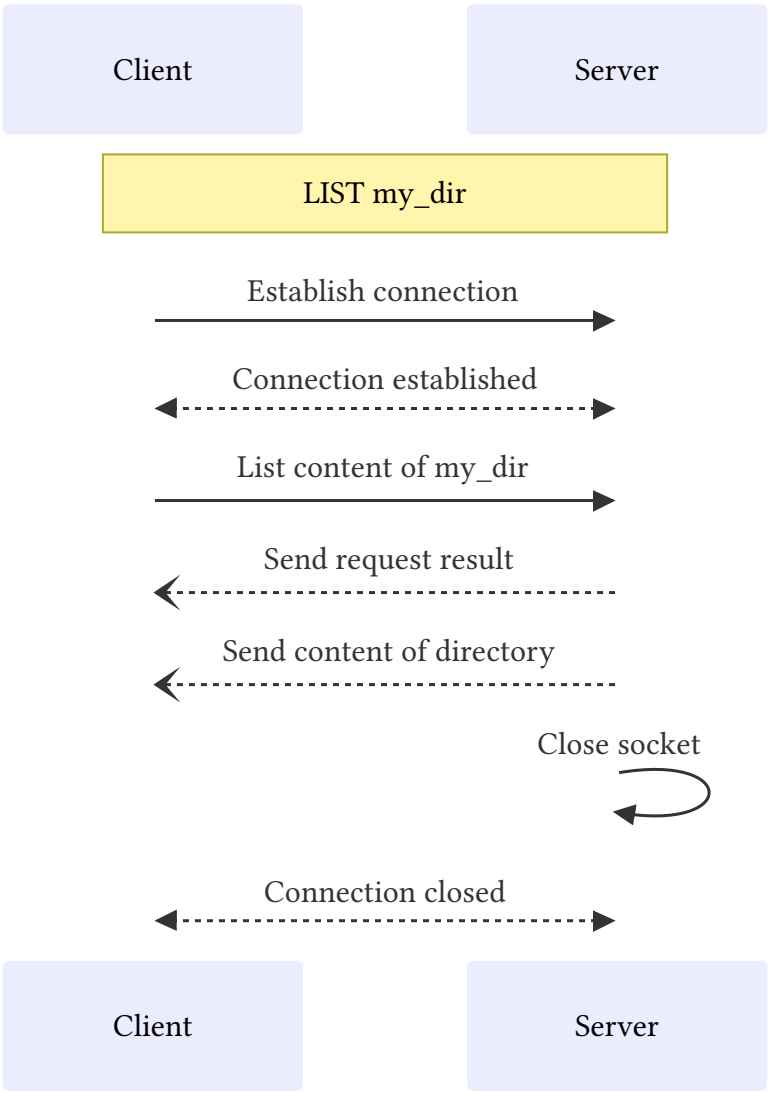
On a successful request, the server answers with the code `0` indicating that the file or folder was removed successfully.

On error, the code matches one of:

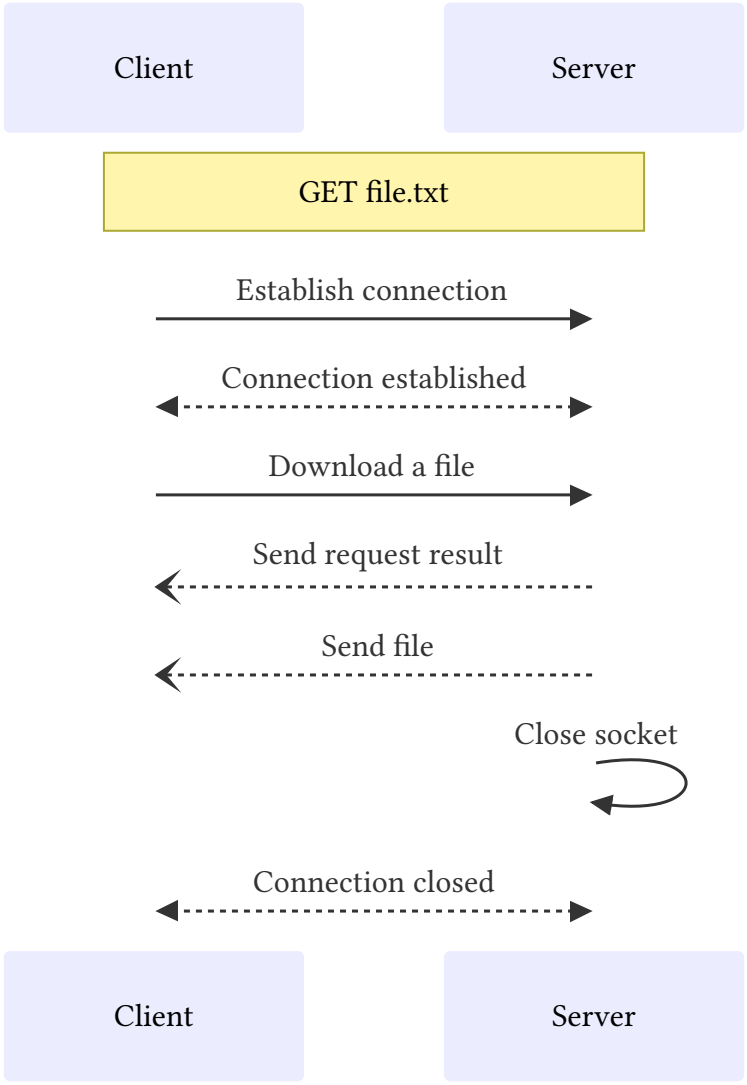
- EACCES
- ENOENT

1.4. Section 4 - Examples

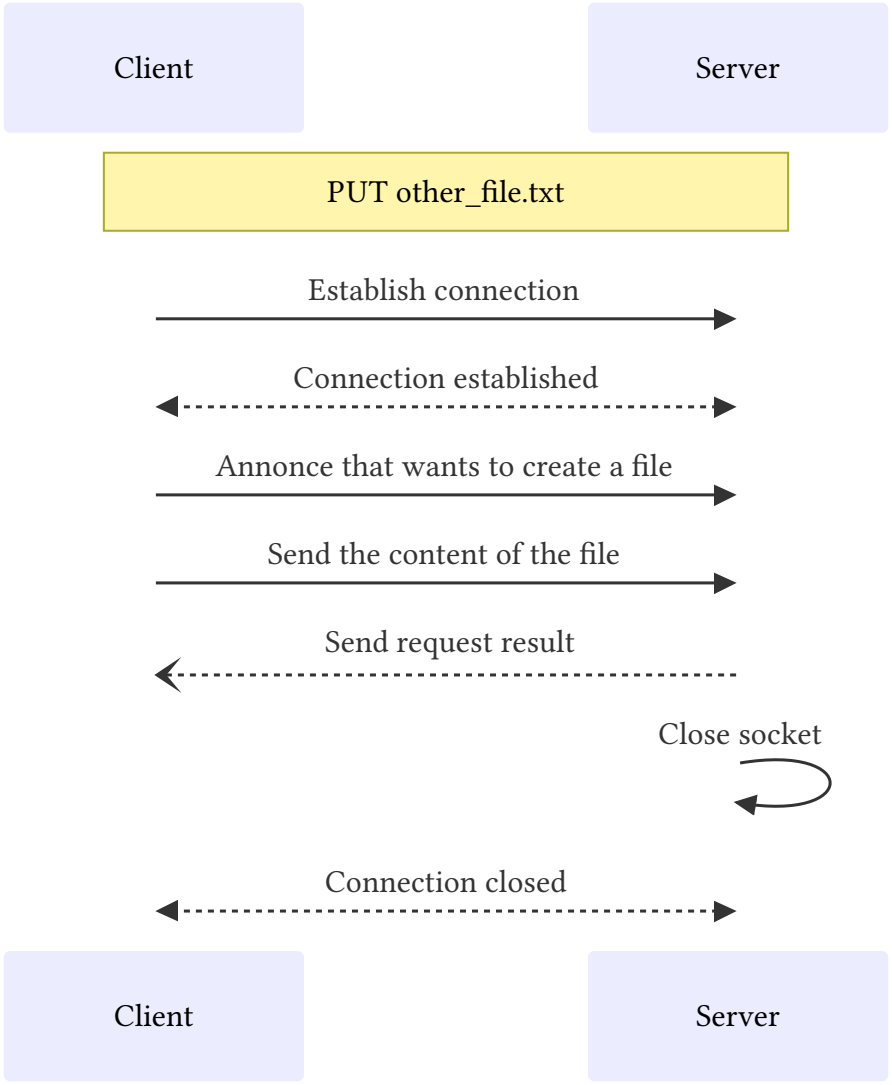
1.4.1. List



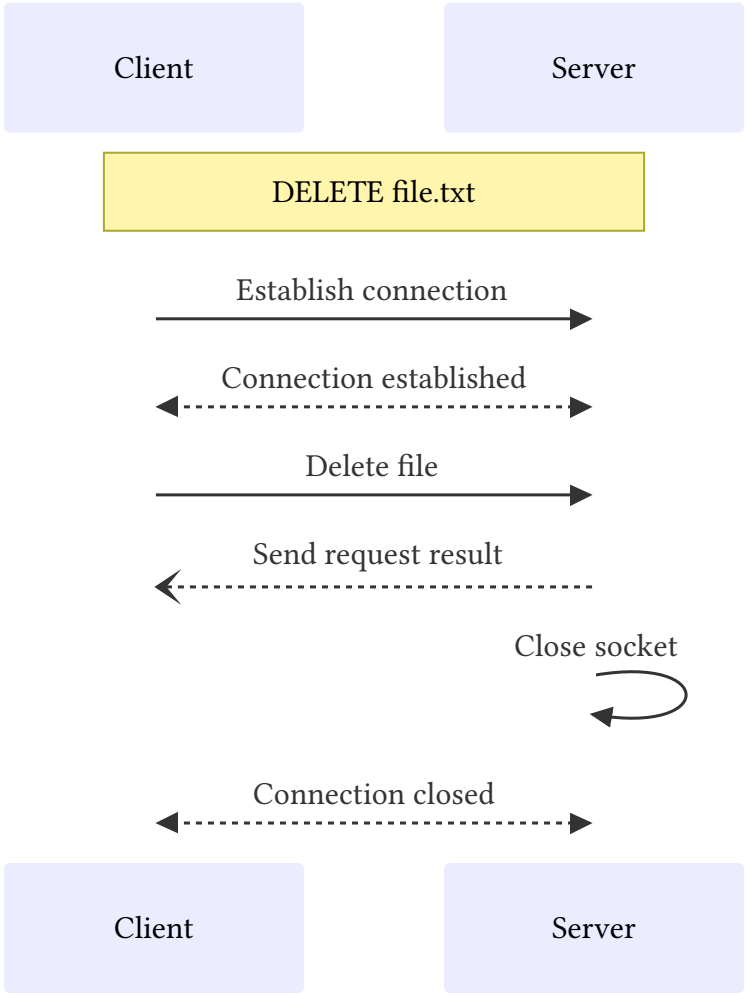
1.4.2. Get


Client
Server
GET file.txt
Establish connection
Connection established
Download a file
Send request result
Send file
Close socket
Connection closed
Client
Server

1.4.3. Put


Client
Server
PUT other_file.txt
Establish connection
Connection established
Annonce that wants to create a file
Send the content of the file
Send request result
Close socket
Connection closed
Client
Server

1.4.4. Delete


Client
Server
DELETE file.txt
Establish connection
Connection established
Delete file
Send request result
Close socket
Connection closed
Client
Server