# Programming Assignment 1 CPSC-471, Spring 2023

## Simplified FTP Server and Client Protocol Design

## 1. Message format (type: byte)

| Data size (10 bytes) | Data payload (bytes) |
|---|---|

A typical message comprises two parts:

- The first 10 bytes of a message carry information about the size of the message so that the application can reassemble the original message at the destination.
    - Data size converted from int to byte representation.
        - Using [int_var].to bytes(10, “big”)
- The data payload carries the actual message (type of byte). There are two types of messages in the application:
    - Message specifies the request.
    - Message carries the response/data in a file.

## 2. Message Exchange (expected message with no error)

| Commands | Client | Server |
|---|---|---|
| 1. get | 1a. ftp> get [filename]<br><br>1c. [client connects to the new data channel]*<br><br>1f. Receiving file: [filename]<br><br>File transfer complete<br>Data connection closed | 1b. [Data connection object created. Inform the client about the new channel]*<br><br>1d. [Accept client connection. Send 1024 bytes of data at a time]*<br><br>1e. File sent successfully.<br>Data connection closed. |

| | | |
|---|---|---|
| 2. put | 2a. ftp> put [filename]<br><br>2c. [client connects to the new data channel]*<br><br>2e. [Send 1024 bytes of data at a time]*<br>Sending [filename]<br><br>File transfer complete.<br>Send [total] of bytes.<br>Data connection closed. | 2b. [Data connection object created.<br>Inform the client about the new channel]*<br><br>2d. [Accept client connection]*<br><br>2f. Receiving file: [filename].<br><br>File received successfully.<br>Data connection closed. |
| 3. ls | 3a. ftp> ls<br><br>3c. [client connects to the new data channel]*<br><br>3e. [List of files shown]<br><br>File list transfer complete.<br>Data connection closed. | 3b. [Data connection object created.<br>Inform the client about the new channel]*<br><br>3d. [Accept client connection.<br>Query the list of available files in the server's directory.<br>Send the whole list]*<br><br>List of files sent successfully.<br>Data connection closed. |
| 4. quit | 4a. ftp> quit<br><br>4c. Server closed connection.<br>FTP client closed. | 4b. [inform client about its closing]<br>Control connection closed.<br><br>Waiting for new connection … |

* Response indicated by [ ]* are hidden under the program application

If an error happens during any step, the program will notify the user.

## 3. Connection Class encapsulates socket connection.

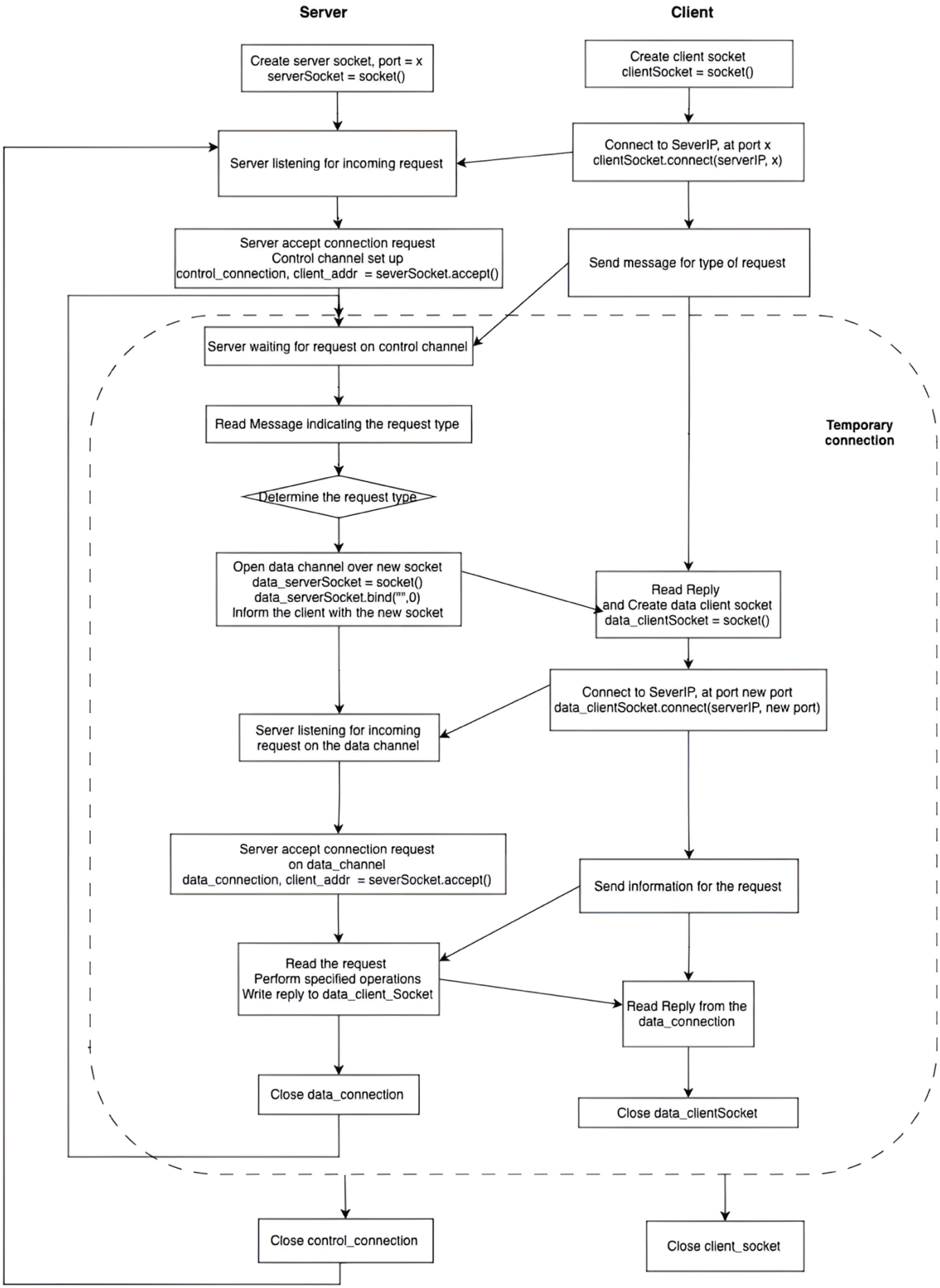
| Connection | clientConnection | serverConnection |
|---|---|---|
| **Fields:**<br>- conn: socket connection | **Extends Connection** | **Extends Connection** |

| | | |
|---|---|---|
| - identity: private string<br><br>**Functions:**<br>- send_message(data): send the whole formatted message<br><br>- recvAll(bytes): receives the number of bytes<br><br>- recv_data_payload():<br><br>call recvAll(10) to extract the length of data load<br><br>call recvAll(data_size) to get the actual data<br><br>- abstract function: init_data_channel() | **Functions:**<br>- init_data_channel():<br><br>create new socket connection<br><br>wait for server's response about the new connection<br><br>connect to new server's socket<br><br>return new connection | **Functions:**<br>- init_data_channel():<br><br>create new socket connection<br><br>inform the client about the new socket<br><br>listen<br><br>return new connection |

## 4. TCP Buffer considerations

- Considering the situation of a simplified application, buffer overflow should not be a big issue:
  - From perspective of the transport layer and beyond: already handled by TCP.
  - From perspective of the application:
    - TCP buffer of modern computer is generally $\geq 4096$ bytes.
    - Send and receive only 1024 bytes of data at a time should not overflow the TCP buffer size.
    - However, list of files under the server directory may be large from real life perspective, still considering the simplified scenario for the assignment, it would be sufficient to simply send the whole message

## 5. General diagram of operations taking place in the protocol



## APPENDIX

### OUTPUT samples:

1. **Server started**

```
16:32:35 python3 serv.py 1235
Waiting for new connection...
```

2. **Client started - client control channel established**

```
16:33:09 python3 cli.py localhost 1235
Server address: localhost, server port: 1235
FTP client control channel established. Ready to take requests.
ftp>
```

3. **Server welcomed client - server control channel established to service**

```
16:32:35 python3 serv.py 1235
Waiting for new connection...
FTP server control channel established. Ready to take requests.
Accepted connection from client:  ('127.0.0.1', 59026)
```

4. **Client - Server message exchange** `ls request`
`Data Channel set up and torn down.`

```
ftp> ls
testing.txt
File list transfer complete.
Data connection closed.
```

```
List of files sent successfully
Data connection closed.
```

5. **Client - Server message exchange** `get request`
`Data Channel set up and torn down.`

```
ftp> get testing.txt
Receiving file: testing.txt
File transfer complete.
Data connection closed.
```

```
File sent successfully
Data connection closed.
```

6. **Client - Server message exchange** `put request`

Data Channel set up and torn down.

```
ftp> put kitten.png
Sending file: kitten.png
File transfer complete. Send total of 4744 bytes.
Data connection closed.
```

```
Receiving file: kitten.png
File received successfully
Data connection closed.
```

Checking if the server has stored the file from the client in the server directory.

```
ftp> ls
kitten.png
testing.txt
File list transfer complete.
Data connection closed.
```

7. **Client - Server message exchange** `quit request.`

Client request quit; server acknowledges its closing.
Client then proceed to close its connection and quit.
Server waits for a new connection.

```
ftp> quit
Server closed connection
FTP client closed.
Control connection closed.
```

```
Control connection closed.
Waiting for new connection...
```