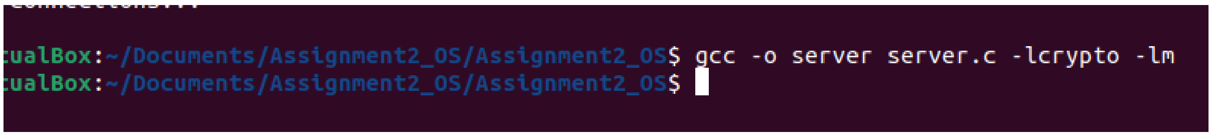
Assignment2_OS

Part A:

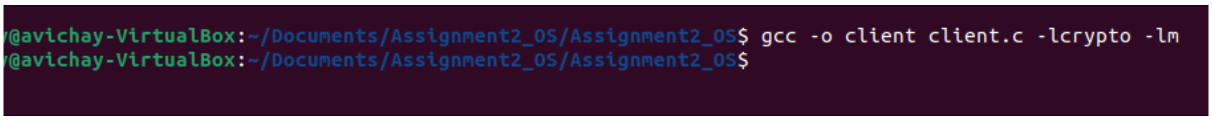
First, you need to compile the server.c file with this command:



```
ualBox:~/Documents/Assignment2_OS/Assignment2_OS$ gcc -o server server.c -lcrypto -lm
ualBox:~/Documents/Assignment2_OS/Assignment2_OS$ 
```

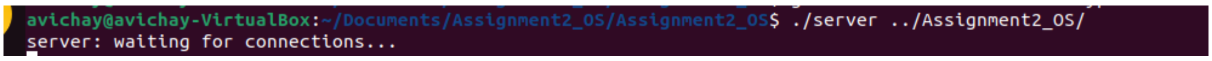
`'-lcrypto -lm'` ensures that your program, upon compilation, has access to and can use the functionalities provided by the OpenSSL cryptographic library and the standard math library, enabling it to perform cryptographic operations and complex mathematical calculations.

After that, compile in another terminal the client.c with this command:



```
@avichay-VirtualBox:~/Documents/Assignment2_OS/Assignment2_OS$ gcc -o client client.c -lcrypto -lm
@avichay-VirtualBox:~/Documents/Assignment2_OS/Assignment2_OS$
```

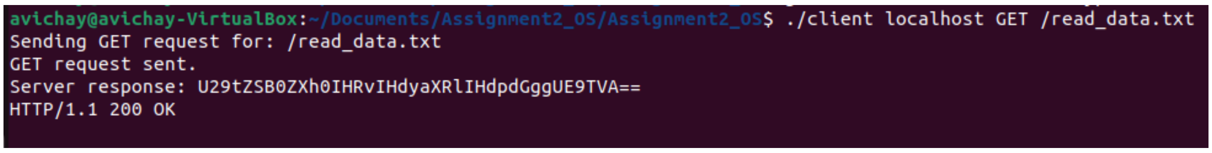
Next, run the server with this command:



```
avichay@avichay-VirtualBox:~/Documents/Assignment2_OS/Assignment2_OS$ ./server ../Assignment2_OS/
server: waiting for connections...
```

Now the server is waiting for connections.

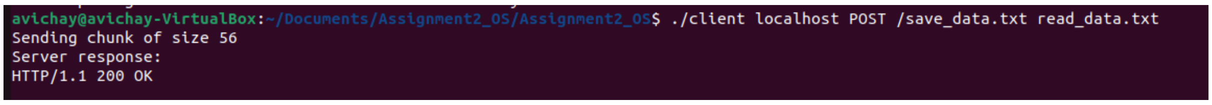
Now go to the second terminal and run this command:



```
avichay@avichay-VirtualBox:~/Documents/Assignment2_OS/Assignment2_OS$ ./client localhost GET /read_data.txt
Sending GET request for: /read_data.txt
GET request sent.
Server response: U29tZSB0ZXh0IHRvIHdyaXRlIHdpdGggUE9TVA==
HTTP/1.1 200 OK
```

Note that read_data.txt can be replaced by any file that can be converted to Base64.

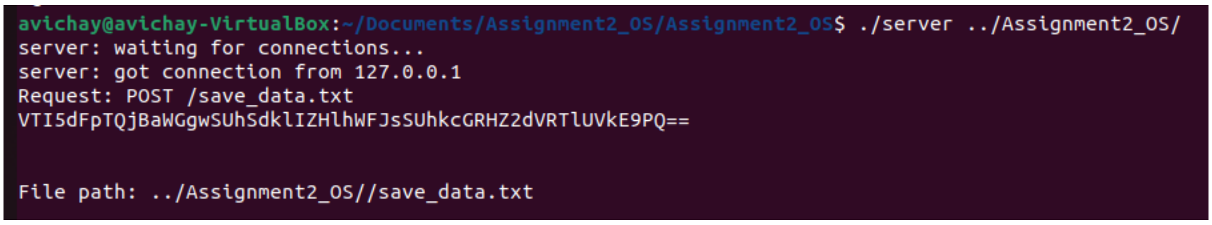
For POST you just need to run in the client terminal this next command instead of the last command:



```
avichay@avichay-VirtualBox:~/Documents/Assignment2_OS/Assignment2_OS$ ./client localhost POST /save_data.txt read_data.txt
Sending chunk of size 56
Server response:
HTTP/1.1 200 OK
```

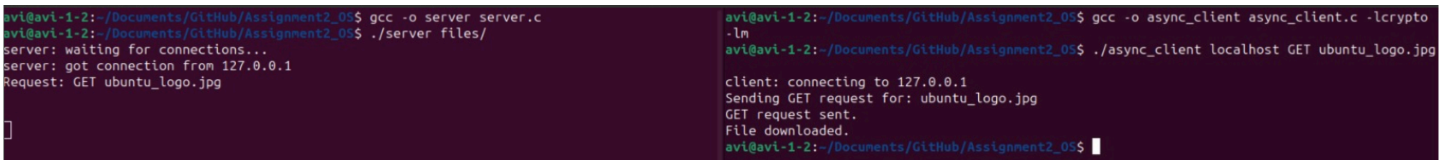
note that save_data.txt is where you want to save the data you reading, and the read_data.txt is from where you read the data.

In the terminal of the server you supposed to get:



```
avichay@avichay-VirtualBox:~/Documents/Assignment2_OS/Assignment2_OS$ ./server ../Assignment2_OS/
server: waiting for connections...
server: got connection from 127.0.0.1
Request: POST /save_data.txt
VTI5dFpTQjBaWGgwSUhSdklIZHlhWFJsSUhkcGRHZ2dVRTlUVkE9PQ==


File path: ../Assignment2_OS//save_data.txt
```

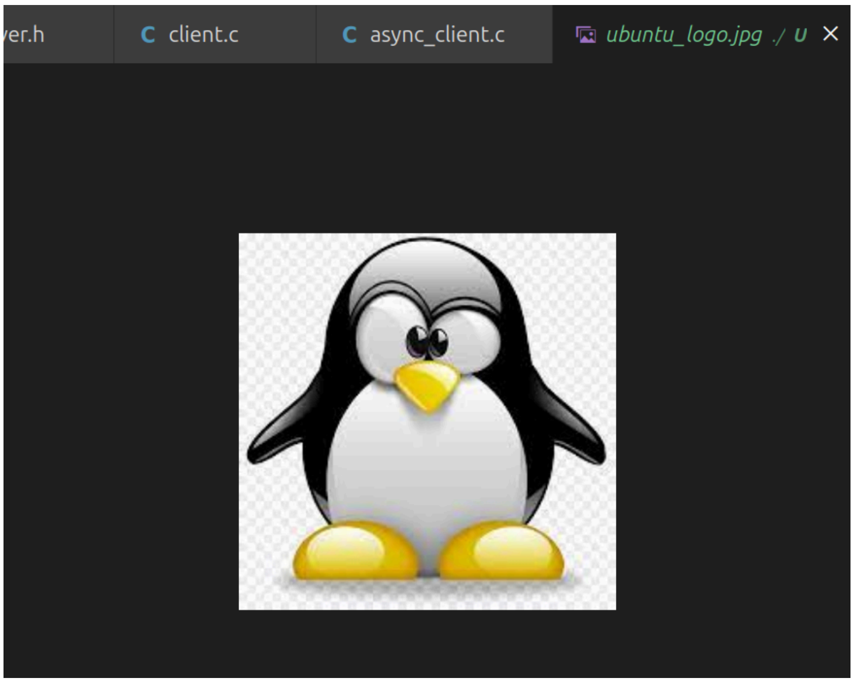
Part B:
In this part, we have the async client that can listen to files asynchronously, i.e., for each file, the server will open a socket for the client to listen to that file.



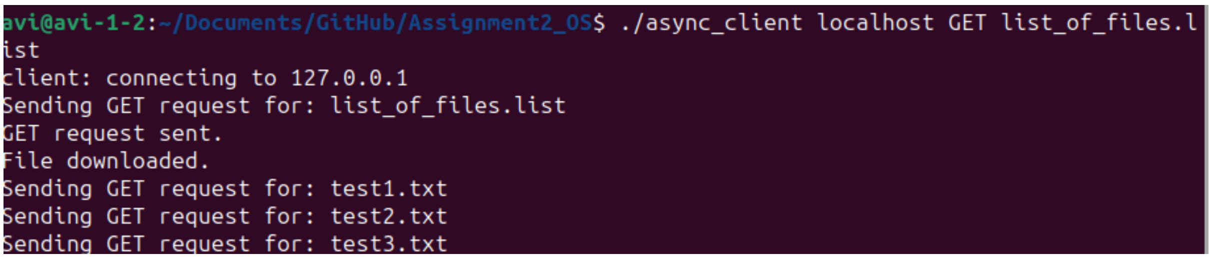
```
avi@avi-1-2:~/Documents/GitHub/Assignment2_OS$ gcc -o server server.c
avi@avi-1-2:~/Documents/GitHub/Assignment2_OS$ ./server files/
server: waiting for connections...
server: got connection from 127.0.0.1
Request: GET ubuntu_logo.jpg
```

```
avi@avi-1-2:~/Documents/GitHub/Assignment2_OS$ gcc -o async_client async_client.c -lcrypto
-lm
avi@avi-1-2:~/Documents/GitHub/Assignment2_OS$ ./async_client localhost GET ubuntu_logo.jpg

client: connecting to 127.0.0.1
Sending GET request for: ubuntu_logo.jpg
GET request sent.
File downloaded.
avi@avi-1-2:~/Documents/GitHub/Assignment2_OS$
```

In this example, we first executed the server and the async client as before (Part A), then run the executable files such that the server would be listening to the files folder and the client would send a request to get the “ubuntu_logo.jpg” image from the “files” folder.
As we can see, the response was good and “File downloaded”, so if we go back to the files of the client, we can see that new image:

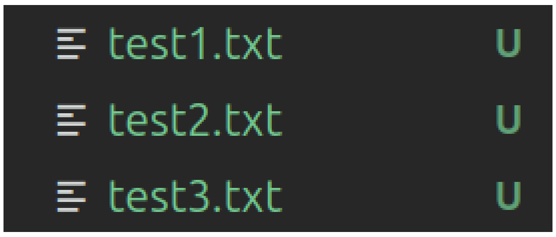


which is the same image as the image on the server (encoded in base64).
Here is another example:

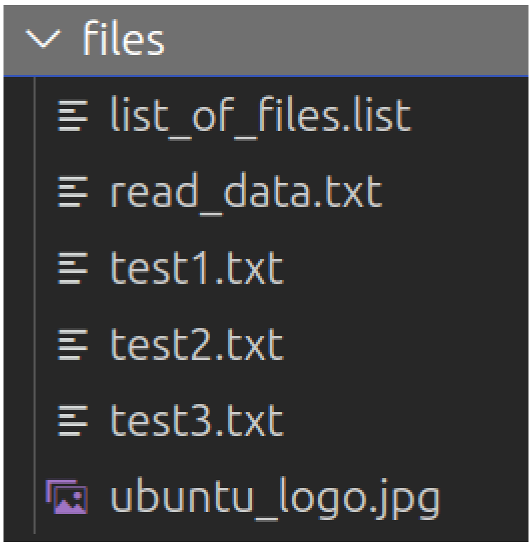


```
avi@avi-1-2:~/Documents/GitHub/Assignment2_OS$ ./async_client localhost GET list_of_files.l
ist
client: connecting to 127.0.0.1
Sending GET request for: list_of_files.list
GET request sent.
File downloaded.
Sending GET request for: test1.txt
Sending GET request for: test2.txt
Sending GET request for: test3.txt
```

(the server still listening.)
The client asks for all the files in the “list_of_files.list” file and gets this:

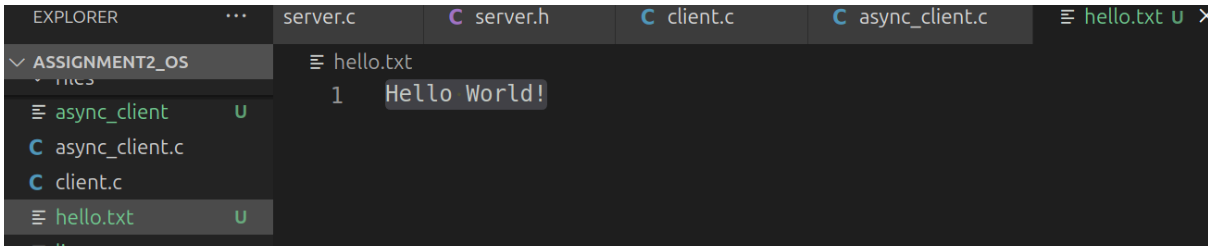


Which are the files in the files folder (on the server):



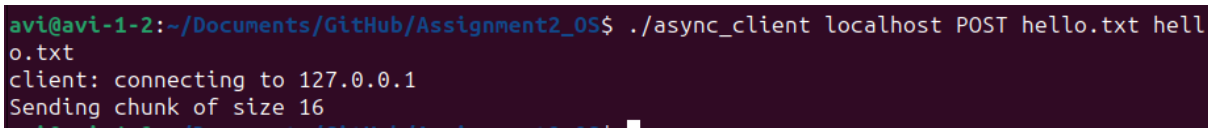
```
files
  list_of_files.list
  read_data.txt
  test1.txt
  test2.txt
  test3.txt
  ubuntu_logo.jpg
```

Inside "list_of_files.list” there is a string encoded in base64 that stores the client IP and the txt files.
In the last example, we got this text file in the client folder:



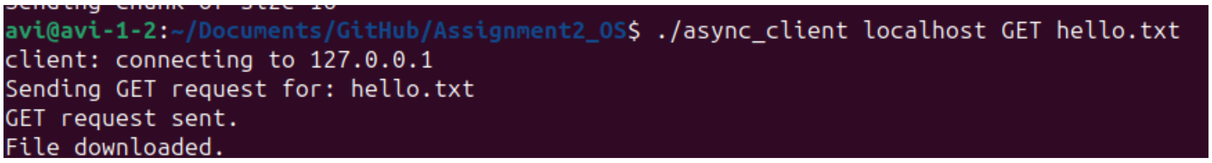
```
EXPLORER  ···     server.c    server.h    client.c    async_client.c    hello.txt U

ASSIGNMENT2_OS         hello.txt
  async_client   U       Hello World!
  async_client.c
  client.c
  hello.txt      U
```

and we want to post this into the server folder and then get it with GET to see if it’s decoding the file correctly:

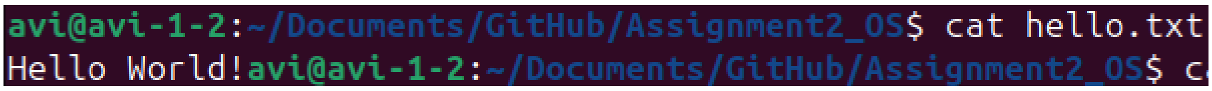


```
avi@avi-1-2:~/Documents/GitHub/Assignment2_OS$ ./async_client localhost POST hello.txt hell
o.txt
client: connecting to 127.0.0.1
Sending chunk of size 16
```

then we delete the file from the client because we assume there is no file with the same name as in the server (if there is one like this, it will overwrite the bytes and then there is a chance to see another character beside the real text)

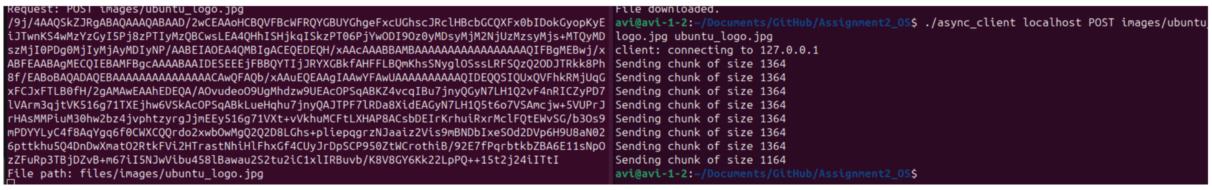


```
avi@avi-1-2:~/Documents/GitHub/Assignment2_OS$ ./async_client localhost GET hello.txt
client: connecting to 127.0.0.1
Sending GET request for: hello.txt
GET request sent.
File downloaded.
```



```
avi@avi-1-2:~/Documents/GitHub/Assignment2_OS$ cat hello.txt
Hello World!avi@avi-1-2:~/Documents/GitHub/Assignment2_OS$ c
```

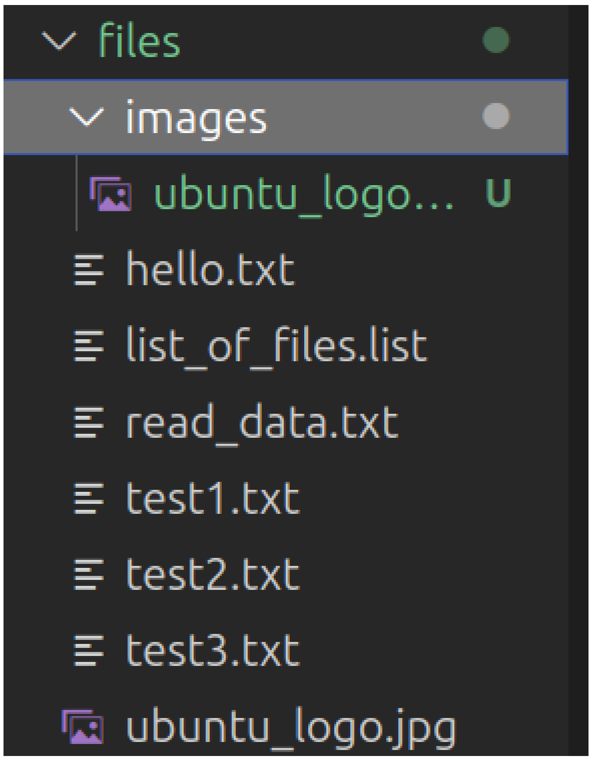
That means that we got it right.
There is also an option to create a folder. Here is how you can do it:



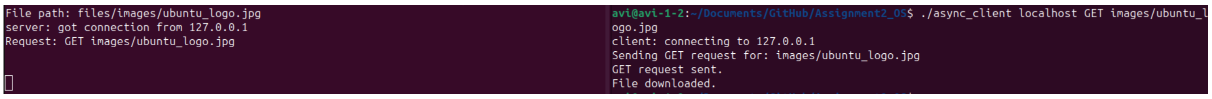
```
Request: POST images/ubuntu_logo.jpg
/9j/4AAQSkZJRgABAQAAAQABAAD/2wCEAAoHCBQVFBcWFRQYGBUYGhgeFxcUGhscJRclHBcbGCQXFx0bIDokGyopKyE
iJTwnKS4wMzYzGyI5Pj8zPTIyMzQBCwsLEA4QHhISHjkqISkzPT06PjYwODI9Oz0yMDsyMjM2NjUzMzsyMjs+MTQyMD
szMjI0PDg0MjIyMjAyMDIyNP/AABEIAOEA4QMBIgACEQEDEQH/xAAcAAABBAMBAAAAAAAAAAAAAAAAAQIFBgMEBwj/x
ABFEAABAgMECQIEBAMFBgcAAAABAAIDESEEEjFBBQYTIjJRYXGBkfAHFFLBQmKhsSNyglOSssLRFSQzQ2ODJTRkk8Ph
8f/EABoBAQADAQEBAAAAAAAAAAAAAAACAwQFAQb/xAAuEQEAAgIAAwYFAwUAAAAAAAAAAQIDEQQSIQUxQVFhkRMjUqG
xFCJxFTLB0fH/2gAMAwEAAhEDEQA/AOvudeoO9UgMhdzw9UEAcOPSqABKZ4vcqIBu7jnyQGyN7LH1Q2vF4nRICZyPD7
lVArm3qjtVK516g71TXEjhw6VkAcOPSqABkLueHqhu7jnyQAJTPF7lRDa8XidEcjw+5VUPrJ
rHAsMMPiuM30hw2bz4jvphtzyrgJjmEEy516g71VXt+vVkhuMCFtLXXNAbhsxOEIrKrhuiRxrMclFQtEWvSG/b
mPDYYLyC4f8AqYgq6f0CWXC6QCQQCQDx2LGhs+pliegqrzNJaaize2NDHS5JBPPxA4kN3N1CvD+OSbeBi2JM+ldh
6pttkhu/3YAmgz9B6fuC5W0lZ4Ulp5O2I+SEUCuuMWzbWTRP6lYwGnZRUFbR4QIArrBMIE27AP0Pc4S3kxUaAsSx+Hk9
zZFuRp3TBjZDZ+m67Dpj4ps5Hte+wMJ3o1cgBh+8IVlmHqjxc/Tm1m8r7udpRJ+lOhJLTbdOC+ekUJ/QsiCsqBNGuR4
File path: files/images/ubuntu_logo.jpg
```

```
File downloaded.
avi@avi-1-2:~/Documents/GitHub/Assignment2_OS$ ./async_client localhost POST images/ubuntu
logo.jpg ubuntu_logo.jpg
client: connecting to 127.0.0.1
Sending chunk of size 1364
Sending chunk of size 1364
Sending chunk of size 1364
Sending chunk of size 1364
Sending chunk of size 1364
Sending chunk of size 1364
Sending chunk of size 1364
Sending chunk of size 1164
avi@avi-1-2:~/Documents/GitHub/Assignment2_OS$
```

And now we can see this in the files folder:

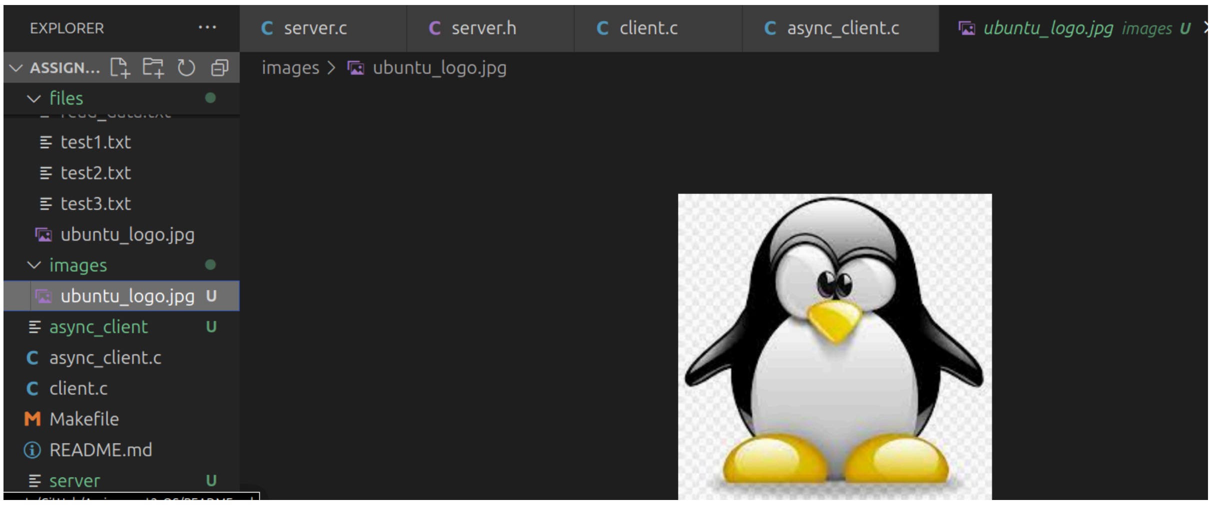


which indicates that there is a folder named “images” and “ubuntu_logo.jpg” inside it. To check if it’s sent all bytes, we now delete the image from the client and then get it with the GET method and see if the image is complete:



```
File path: files/images/ubuntu_logo.jpg
server: got connection from 127.0.0.1
Request: GET images/ubuntu_logo.jpg
```

```
avi@avi-1-2:~/Documents/GitHub/Assignment2_OS$ ./async_client localhost GET images/ubuntu_l
ogo.jpg
client: connecting to 127.0.0.1
Sending GET request for: images/ubuntu_logo.jpg
GET request sent.
File downloaded.
```



that it means that all data came as expected.