

Computer Communications and Networks – Project 1

HTTP Client and Server

Sampath Kumar Gunasekaran

800954726

HTTP client and server implementation

HTTP client and server which runs a simplified version of HTTP/1.1 is implemented in java using socket programming. As part of this project, only GET and PUT methods of HTTP is implemented.

Overview of GET and PUT

GET Method: Client requests data from the server by making use of GET requests. Following are the steps involved in GET requests:

1. Firstly, client establishes a TCP connection with the server.
2. Client requests for the file by sending the name of the file as part of request header and submits a valid HTTP/1.1 GET request to the server.
3. Server listens for any open client connections. When a client connection is accepted, reads the HTTP request from the client, server checks whether the requested file is present in server database or not.
4. If the requested file is not present in the server's directory, server will respond with a 400-status code.
5. If the requested file is present, server will respond by sending the requested file with a 200-status code.

PUT Method: Client sends files to the server and server saves the files locally. Following are the steps involved in PUT requests:

1. Firstly, client establishes a TCP connection with the server.
2. Client sends the file contents along with the request it wants to save to the server.
3. Server listens for any open client connections. When a client connection is accepted, reads the HTTP request from the client, server saves the file and sends 200-status code as a response.

Implementation details

Following java files are created to implement the HTTP client server.

1. MyClient.java

- a. Performs the functions of a HTTP client.
- b. MyClient class takes Server name, Port, HTTP command and the path of the file as command line arguments and submits a request to the server.

2. ClientProcess.java

- a. ClientProcess class takes Server name, Port, and the path of the file as arguments.
- b. Separated the GET and PUT requests in this class. It constructs a request header based on the HTTP command, submits a request to the server and displays the response.

3. MyServer.java

- a. Performs the functions of a HTTP server.
- b. MyServer class is a multi-threaded server to handle multiple client connections at the same time. It takes Port as command line argument and sends the response over the established connection for all the accepted clients.

4. ServerProcess.java

- a. ServerProcess class constructs the response header based on the HTTP request.
- b. Server sends the requested object if the request is a GET or saves the files locally if the request is a PUT.

HTTP Client Implementation

1. MyClient class takes Server name, Port, HTTP command and the path of the file as command line arguments.
Example: MyClient Hostname Port Method filename
2. If the number of command line arguments exceeds four, exception will be thrown saying "Invalid number of arguments. Expecting four arguments" and the program will be halted.
3. If the command is GET, getMethod() in ClientProcess class is invoked. Else if the command is PUT, putMethod() in ClientProcess class is invoked. Else, exception will be thrown saying "HTTP Command is not valid! System accepts only GET or PUT as commands." and the program will be halted.
4. ClientProcess class constructs a request header based on the HTTP command.

5. If it is a GET command, it opens a socket connection based on the hostname and port number. Then it declares a writer to this URL, builds the HTTP GET request header using `PrintStream`, sends the request, and displays the response using `DataInputStream`. I/P, O/P streams and socket connections are closed upon successful response.
6. If it is a PUT command, it opens a connection based on the host and port number, declares a write to this URL, builds the HTTP POST request header using `PrintStream` and the contents needed to be uploaded is read using `FileInputStream` and copied on using `PrintStream`, sends the request, and displays the response using `DataInputStream`. I/P, O/P streams and socket connections are closed upon successful response.

HTTP Server Implementation

1. `MyServer` class is a multi-threaded server which takes Port number as command line argument and sends the response over the established connection for all the accepted clients.
Example: `MyServer port`
2. Create `ServerSocket` at the port number received as a command line argument and listen for the clients to connect.
3. `MyServer` is a multi-threaded server which will have an infinite loop to listen for client connections. For each client accepted, it creates a separate thread to handle the connection. Hence, multiple clients can be handled by the server at the same time.
4. Request coming from each client is handled by `ServerProcess` class which constructs the response header based on the HTTP request. Server sends the requested object if the request is a GET or saves the files locally if the request is a PUT.
5. After receiving the request from the client, `processRequest()` method in `ServerProcess` class takes the responsibility to process the request based on the type of request and responds with a response.
6. If it is a GET request, it checks whether the requested file exists in the server directory. It constructs a response header with a 404-status code if the file is not present. Else, it responds with a 200-status code and file contents the client requested by writing the response to the client's output stream.
7. If it is a PUT request, server writes the contents to a file by using `FileOutputStream`.
8. If the file is successfully created, server responds with a 200-status code. Else, it responds with a 301-status code, indicating a bad request.

9. Server is shutdown gracefully on termination signal (ctrl+c) from the terminal. Resources are cleaned up, closes all the open connections and sockets before exiting.

Compilation and Execution

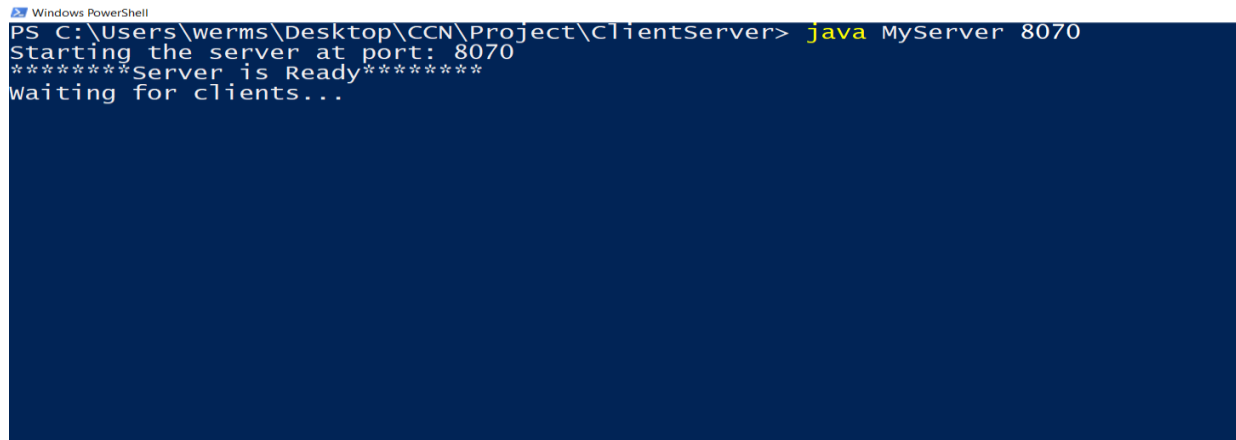
1. Compile both client and server classes using the following commands.

```
javac MyClient.java
```

```
javac MyServer.java
```

2. Open a new terminal and start the server by using the following command.

```
java MyServer 8070
```

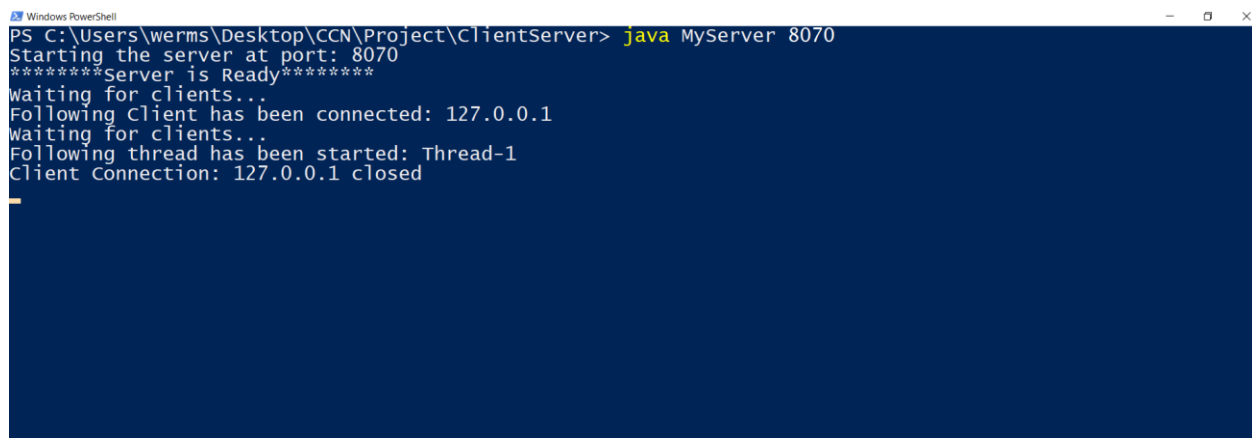


```
Windows PowerShell
PS C:\Users\werms\Desktop\CCN\Project\ClientServer> java MyServer 8070
Starting the server at port: 8070
*****Server is Ready*****
waiting for clients...
```

3. Create a GET request to read the contents of the file getdata.txt which is present in the server's directory.

```
java MyClient localhost 8070 GET getdata.txt
```

Server's screenshot showing that the following client has been connected.



```
Windows PowerShell
PS C:\Users\werms\Desktop\CCN\Project\ClientServer> java MyServer 8070
Starting the server at port: 8070
*****Server is Ready*****
Waiting for clients...
Following Client has been connected: 127.0.0.1
Waiting for clients...
Following thread has been started: Thread-1
Client Connection: 127.0.0.1 closed
```

Client's screenshot showing the successful GET request.

```
Windows PowerShell
PS C:\Users\werms\Desktop\CCN\Project\ClientServer> java MyClient localhost 8070 GET getdata.txt
Trying to connect: localhost on port number: 8070
*****
Connected to the server
*****
Creating a GET request... Trying to get the file: getdata.txt
GET Request sent!
*****
Response from the Server:
HTTP/1.1 200 OK
Date:Sat Jun 10 14:21:58 EDT 2017
Server:localhost

Testing the client server get connection!!!

Response Received!
*****
PS C:\Users\werms\Desktop\CCN\Project\ClientServer> _
```

Trying to retrieve a file which is not present in the server's directory. Server responded with a 404-error message.

```
PS C:\Users\werms\Desktop\CCN\Project\ClientServer> java MyClient localhost 8070 GET random.txt
Trying to connect: localhost on port number: 8070
*****
Connected to the server
*****
Creating a GET request... Trying to get the file: random.txt
GET Request sent!
*****
Response from the Server:
HTTP/1.1 404 Not Found
Date:Sat Jun 10 14:37:35 EDT 2017
Server:localhost

Response Received!
*****
PS C:\Users\werms\Desktop\CCN\Project\ClientServer>
```

4. Creating a PUT request to the server. Uploading an image to the server.

```
Windows PowerShell
PS C:\Users\werms\Desktop\CCN\Project\ClientServer> java MyClient localhost 8070 PUT clientserver.png
*****
Connected to the server
*****
Creating a PUT request... Trying to upload the file: clientserver.png
PUT Request Header sent!
*****
Response from the Server:
Response: HTTP/1.1 200 OK
Date:Sat Jun 10 14:46:55 EDT 2017
Server:localhost

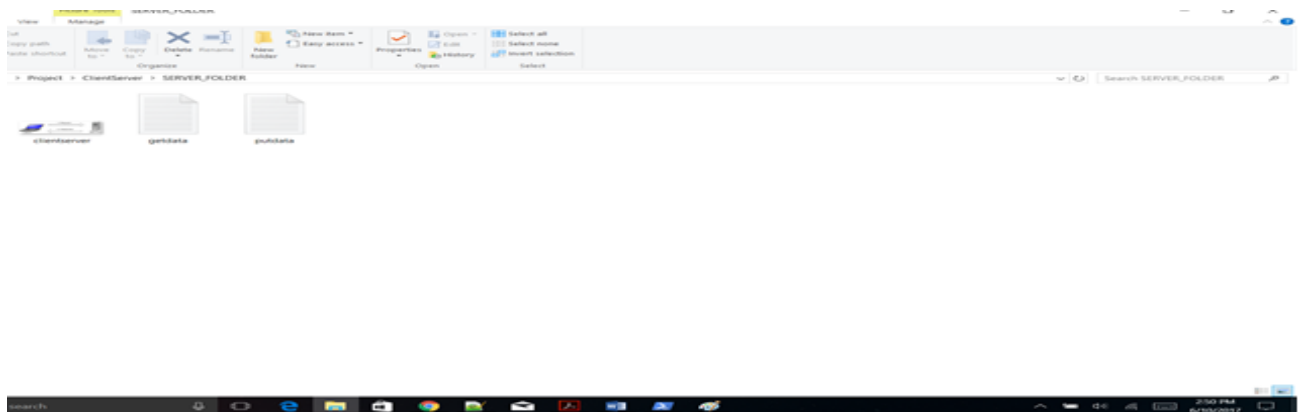
*****
PS C:\Users\werms\Desktop\CCN\Project\ClientServer>
```

Uploading a file to the server.

```
PS C:\Users\werms\Desktop\CCN\Project\ClientServer> java MyClient localhost 8070 PUT putdata.txt
*****
Connected to the server
*****
Creating a PUT request... Trying to upload the file: putdata.txt
PUT Request Header sent!
*****
Response from the Server:
Response: HTTP/1.1 200 OK
Date: Sat Jun 10 14:49:05 EDT 2017
Server: localhost

*****
PS C:\Users\werms\Desktop\CCN\Project\ClientServer> █
```

Server's directory showing the receipt of two files uploaded by the client.



5. Server is shut down gracefully by using ctrl+c in the terminal.

```
Windows PowerShell
PS C:\Users\werms\Desktop\CCN\Project\ClientServer> java MyServer 8070
Starting the server at port: 8070
*****Server is Ready*****
Waiting for clients...
Following Client has been connected: 127.0.0.1
Waiting for clients...
Following thread has been started: Thread-1
Client Connection: 127.0.0.1 closed
Following Client has been connected: 127.0.0.1
Waiting for clients...
Following thread has been started: Thread-2
Client Connection: 127.0.0.1 closed
Following Client has been connected: 127.0.0.1
Waiting for clients...
Following thread has been started: Thread-3
Client Connection: 127.0.0.1 closed
Following Client has been connected: 127.0.0.1
Waiting for clients...
Following thread has been started: Thread-4
Client Connection: 127.0.0.1 closed
Starting to shut down
Cleaned up all the resources
The server is shut down!!!
PS C:\Users\werms\Desktop\CCN\Project\ClientServer>
```