

Computer Communications and Networks (ITCS 6166)

Project Part 1

HTTP Client and Server

Language used: C#

Introduction:

The objective of this project is to implement the HTTP/1.1 request commands (GET and PUT) using client server framework.

Socket programming was used to accomplish the task of communication between the client and the server. HTTP commands such as GET and PUT were used as the communication protocols.

HTTP GET and PUT Methods:

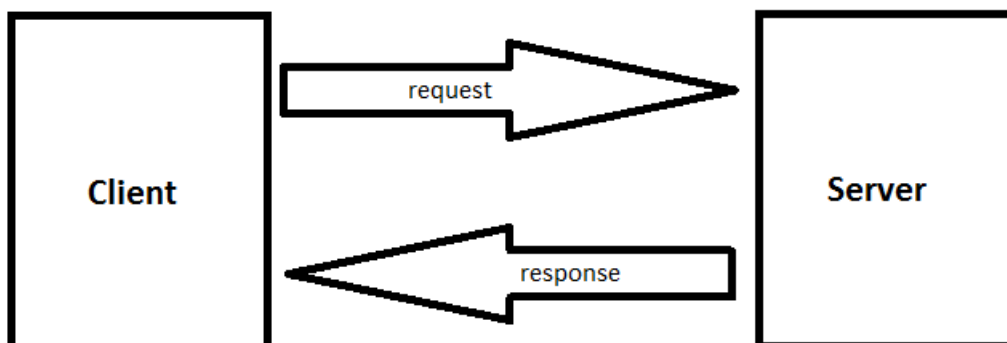
Two commonly used methods for a request-response between a client and server in HTTP/1.1 protocol are: GET and PUT.

- **GET** - Requests data from a specified resource (Server)
- **PUT** - Submits data to be processed to a specified resource (Server)

Components in the project:

The two major components of the project are: Client and Server. They communicate using the request response protocol, where the server is waiting for the client connections on a port and when the server requests something from the client the connection is established between the two. When the communication is complete between the two the connection is closed.

We use TCP connection to connect the client and server.



HTTP Server:

- It is multithreaded application which listens to multiple clients on its port.
- We need to pass a port number as the command line parameter to the server code (example: myserver.exe 8090). The server will then listen to requests on this port.
- The HTTP server handles the following commands.
 - **GET** – The GET command checks if the file is present or not and gives back an appropriate response message (“200 OK” or “404 Not Found”)

Syntax: myclient.exe host_ip port GET filename

Example: myclient.exe 127.0.0.1 8090 GET C:\USERS\TEJASWI\DESKTOP\TEST.TXT
 - **PUT** – The PUT command sends a file to the server and the server downloads the file and returns a response “200 OK FILE CREATED” if file successfully saved, otherwise a response “204 FILE NOT SAVED” will be returned.

Syntax: myclient.exe host_ip port PUT filename

Example: myclient.exe 127.0.0.1 8090 PUT C:\USERS\TEJASWI\DESKTOP\TEST.TXT
 - **QUIT** – The server is listening to requests continuously in a loop, so when we send a QUIT command from the client, the server stops listening and exits gracefully.

Syntax: myclient.exe host_ip port QUIT

Example: myclient.exe 127.0.0.1 8090 QUIT

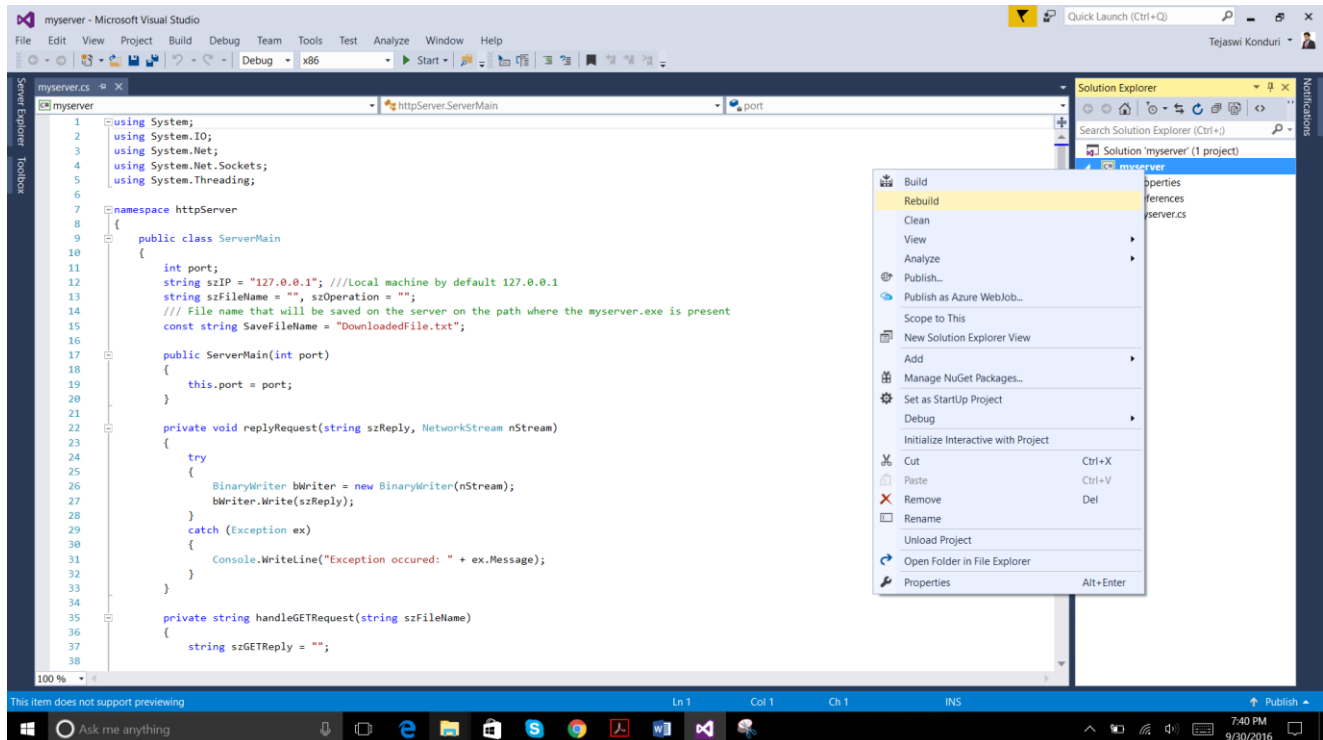
HTTP Client:

- It sends a request message to the server, the server in turn responds to its requests by sending status codes and messages.
- Client is the initiator of the request and hence the connection.
- After successful transactions between the client and the server when the server is supposed to be closed, the client sends a QUIT request. The QUIT request does not have the file path parameter.
- The client takes the following parameters:
 - Ip address/hostname (Example: 127.0.0.1)
 - Port number (Example: 8090)
 - Command (Example: GET/PUT/QUIT)
 - File path
- **Syntax of command line:** myclient.exe host_ip port command file_path
- **Example:** myclient.exe 127.0.0.1 8090 PUT C:\USERS\TEJASWI\DESKTOP\TEST.TXT

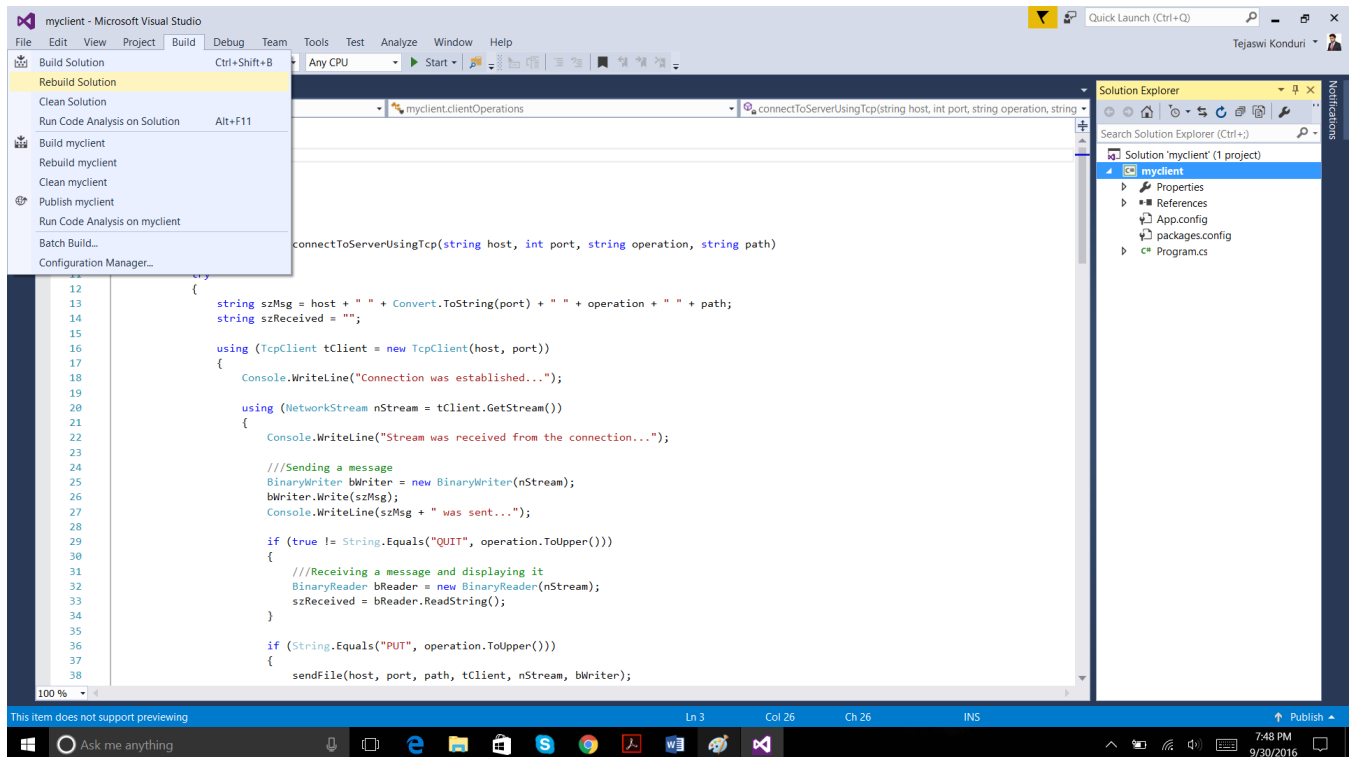
Instructions to build and run the application in C#:

Steps to run the application:

- 1) Unzip the project folder (PP1_Tejaswi)
- 2) Build the server project (PP1_Tejaswi\myserver\myserver.sln)
 - a. Open the myserver.sln file in Visual Studio
 - b. Rebuild the project from the Menu bar or Solution Explorer



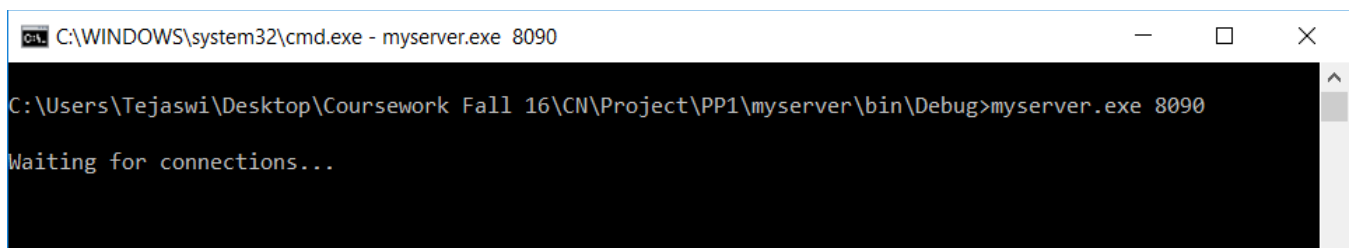
- c. Open the path in the folder explorer to view the executable file (myserver.exe).
Example in our case: PP1_Tejaswi\myserver\bin\Debug\
- 3) Build the client project (PP1_Tejaswi\myclient\myclient.sln)
 - a. Open the myclient.sln file in Visual Studio
 - b. Rebuild the project from the Menu bar or Solution Explorer

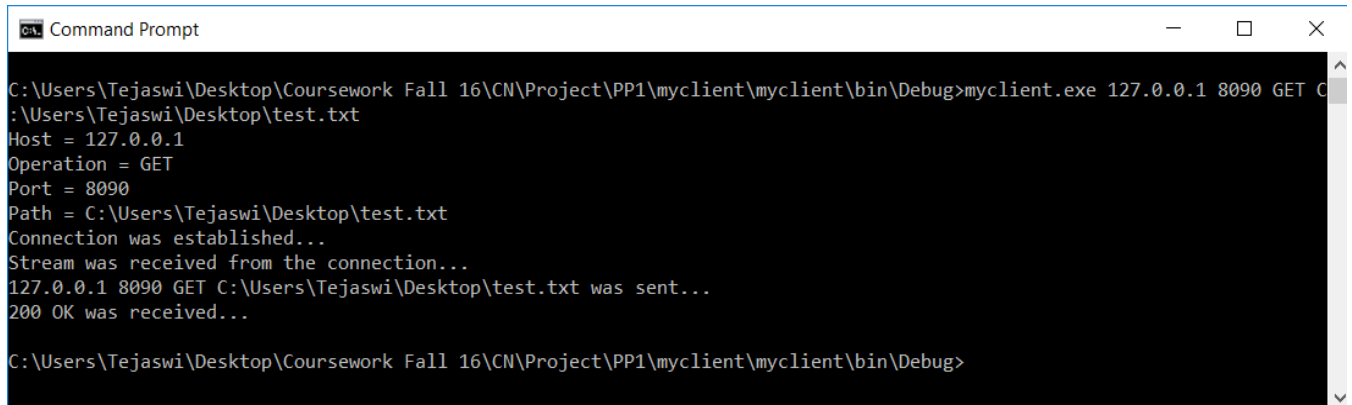


- c. Open the path in the folder explorer to view the executable file (myclient.exe).
Example in our case: PP1_Tejaswi\myclient\myclient\bin\Debug
- 4) First run the server executable (myserver.exe) in command prompt with appropriate command line arguments, it will start listening to incoming requests.
- 5) Then, run the client executable (myclient.exe) in command prompt with appropriate command line arguments, it will connect to the server, make sure you are connecting to the server with the same port number (ex. 8090).
- 6) For PUT command, the file 'DownloadedFile.txt' will be saved in the same folder where myserver.exe is placed. In our case it will be saved at PP1_Tejaswi\myserver\bin\Debug\
- 7) Refer the Screenshots section of the Project Report (this document) for the syntax and example of each operation for client and server.

Screenshots of the various operations:

Initial state of server:



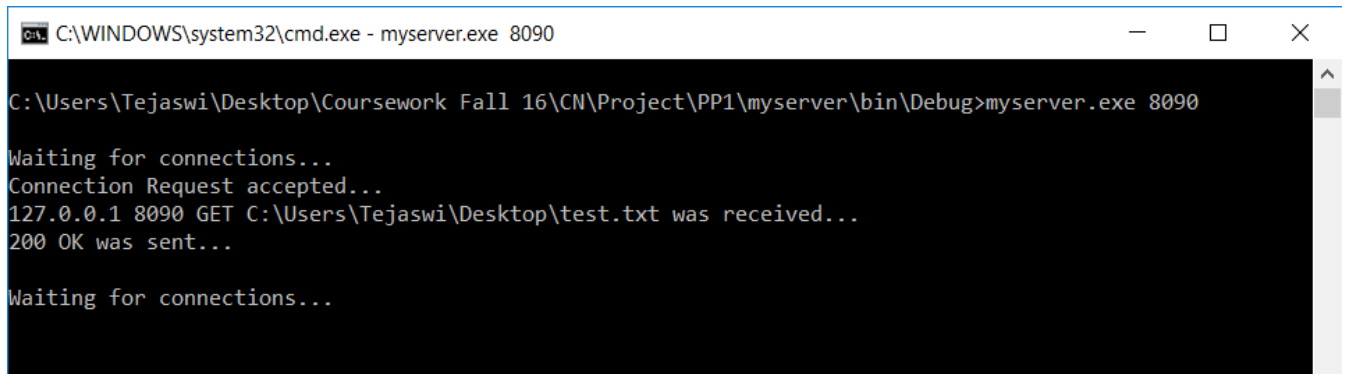
1) GET command**a) 200 OK response****Client side command:**


```

C:\Users\Tejaswi\Desktop\Coursework Fall 16\CN\Project\PP1\myclient\myclient\bin\Debug>myclient.exe 127.0.0.1 8090 GET C:\Users\Tejaswi\Desktop\test.txt
Host = 127.0.0.1
Operation = GET
Port = 8090
Path = C:\Users\Tejaswi\Desktop\test.txt
Connection was established...
Stream was received from the connection...
127.0.0.1 8090 GET C:\Users\Tejaswi\Desktop\test.txt was sent...
200 OK was received...

C:\Users\Tejaswi\Desktop\Coursework Fall 16\CN\Project\PP1\myclient\myclient\bin\Debug>

```

Server side response:


```

C:\WINDOWS\system32\cmd.exe - myserver.exe 8090

C:\Users\Tejaswi\Desktop\Coursework Fall 16\CN\Project\PP1\myserver\bin\Debug>myserver.exe 8090
Waiting for connections...
Connection Request accepted...
127.0.0.1 8090 GET C:\Users\Tejaswi\Desktop\test.txt was received...
200 OK was sent...

Waiting for connections...

```

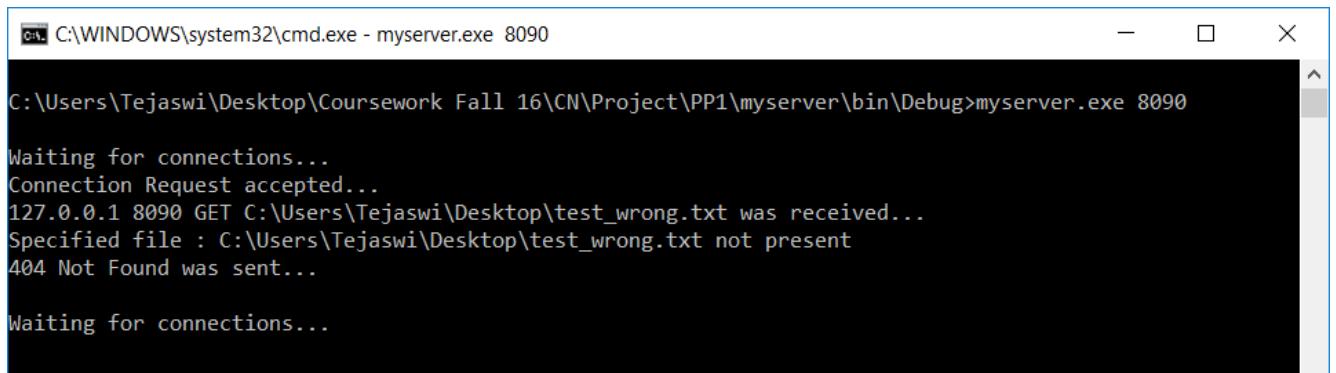
b) 404 Not Found**Client side command:**


```

C:\Users\Tejaswi\Desktop\Coursework Fall 16\CN\Project\PP1\myclient\myclient\bin\Debug>myclient.exe 127.0.0.1 8090 GET C:\Users\Tejaswi\Desktop\test_wrong.txt
Host = 127.0.0.1
Operation = GET
Port = 8090
Path = C:\Users\Tejaswi\Desktop\test_wrong.txt
Connection was established...
Stream was received from the connection...
127.0.0.1 8090 GET C:\Users\Tejaswi\Desktop\test_wrong.txt was sent...
404 Not Found was received...

C:\Users\Tejaswi\Desktop\Coursework Fall 16\CN\Project\PP1\myclient\myclient\bin\Debug>

```

Server side response:


```

C:\WINDOWS\system32\cmd.exe - myserver.exe 8090

C:\Users\Tejaswi\Desktop\Coursework Fall 16\CN\Project\PP1\myserver\bin\Debug>myserver.exe 8090

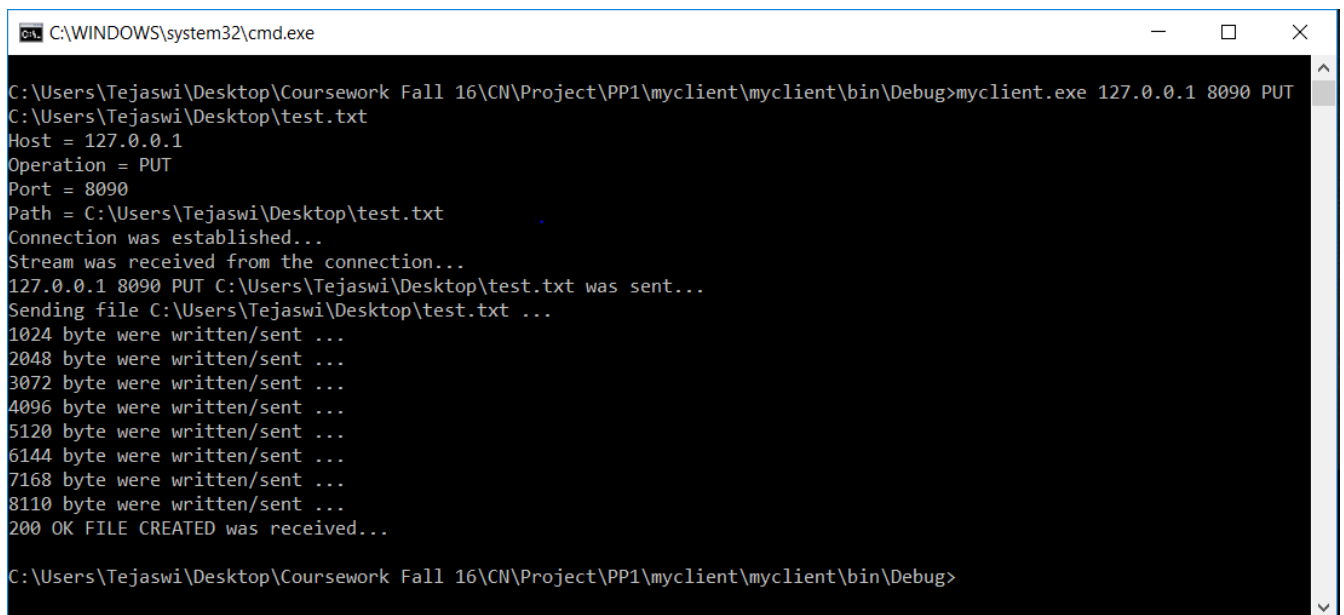
Waiting for connections...
Connection Request accepted...
127.0.0.1 8090 GET C:\Users\Tejaswi\Desktop\test_wrong.txt was received...
Specified file : C:\Users\Tejaswi\Desktop\test_wrong.txt not present
404 Not Found was sent...

Waiting for connections...

```

2) PUT Command

Note: While entering path for the file make sure you write it in double quotes (""), this will help if there is space in the folder name. Example: "C:\a b\test.txt"

Client side command:


```

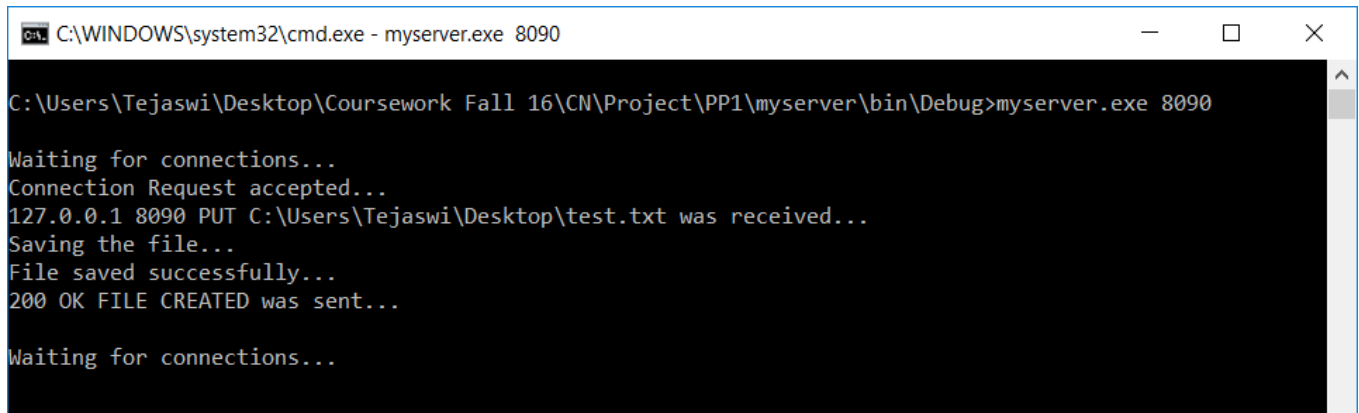
C:\WINDOWS\system32\cmd.exe

C:\Users\Tejaswi\Desktop\Coursework Fall 16\CN\Project\PP1\myclient\myclient\bin\Debug>myclient.exe 127.0.0.1 8090 PUT
C:\Users\Tejaswi\Desktop\test.txt
Host = 127.0.0.1
Operation = PUT
Port = 8090
Path = C:\Users\Tejaswi\Desktop\test.txt
Connection was established...
Stream was received from the connection...
127.0.0.1 8090 PUT C:\Users\Tejaswi\Desktop\test.txt was sent...
Sending file C:\Users\Tejaswi\Desktop\test.txt ...
1024 byte were written/sent ...
2048 byte were written/sent ...
3072 byte were written/sent ...
4096 byte were written/sent ...
5120 byte were written/sent ...
6144 byte were written/sent ...
7168 byte were written/sent ...
8110 byte were written/sent ...
200 OK FILE CREATED was received...

C:\Users\Tejaswi\Desktop\Coursework Fall 16\CN\Project\PP1\myclient\myclient\bin\Debug>

```

Server side response:



```

C:\WINDOWS\system32\cmd.exe - myserver.exe 8090

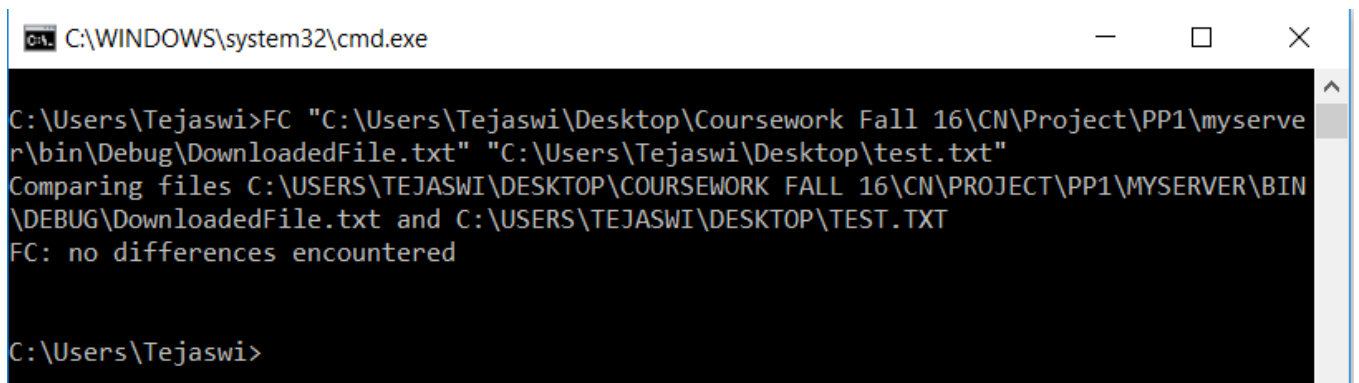
C:\Users\Tejaswi\Desktop\Coursework Fall 16\CN\Project\PP1\myserver\bin\Debug>myserver.exe 8090

Waiting for connections...
Connection Request accepted...
127.0.0.1 8090 PUT C:\Users\Tejaswi\Desktop\test.txt was received...
Saving the file...
File saved successfully...
200 OK FILE CREATED was sent...

Waiting for connections...

```

File Compare (FC) command result:



```

C:\WINDOWS\system32\cmd.exe

C:\Users\Tejaswi>FC "C:\Users\Tejaswi\Desktop\Coursework Fall 16\CN\Project\PP1\myserver\bin\Debug\DownloadedFile.txt" "C:\Users\Tejaswi\Desktop\test.txt"
Comparing files C:\USERS\TEJASWI\DESKTOP\COURSEWORK FALL 16\CN\PROJECT\PP1\MYSERVER\BIN\DEBUG\DownloadedFile.txt and C:\USERS\TEJASWI\DESKTOP\TEST.TXT
FC: no differences encountered

C:\Users\Tejaswi>

```

3) QUIT command

Client side command:



```

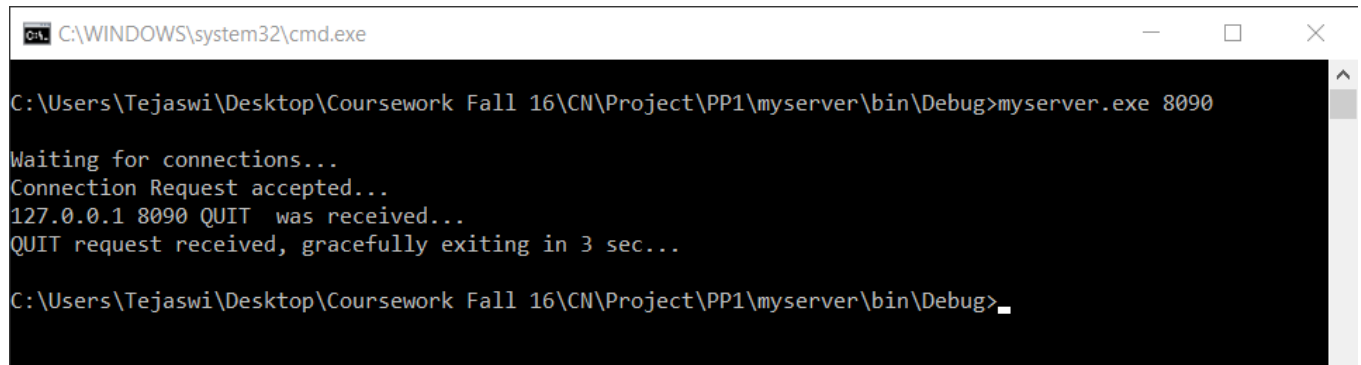
Command Prompt

C:\Users\Tejaswi\Desktop\Coursework Fall 16\CN\Project\PP1\myclient\myclient\bin\Debug>myclient.exe 127.0.0.1 8090 QUIT
Host = 127.0.0.1
Operation = QUIT
Port = 8090
Connection was established...
Stream was received from the connection...
127.0.0.1 8090 QUIT was sent...

C:\Users\Tejaswi\Desktop\Coursework Fall 16\CN\Project\PP1\myclient\myclient\bin\Debug>

```

Server side response:



A screenshot of a Windows command prompt window. The title bar shows the file path "C:\WINDOWS\system32\cmd.exe". The command prompt displays the following text: "C:\Users\Tejaswi\Desktop\Coursework Fall 16\CN\Project\PP1\myserver\bin\Debug>myserver.exe 8090", "Waiting for connections...", "Connection Request accepted...", "127.0.0.1 8090 QUIT was received...", "QUIT request received, gracefully exiting in 3 sec...", and "C:\Users\Tejaswi\Desktop\Coursework Fall 16\CN\Project\PP1\myserver\bin\Debug>".

```
C:\WINDOWS\system32\cmd.exe
C:\Users\Tejaswi\Desktop\Coursework Fall 16\CN\Project\PP1\myserver\bin\Debug>myserver.exe 8090
Waiting for connections...
Connection Request accepted...
127.0.0.1 8090 QUIT was received...
QUIT request received, gracefully exiting in 3 sec...
C:\Users\Tejaswi\Desktop\Coursework Fall 16\CN\Project\PP1\myserver\bin\Debug>
```