Network Architecture and Protocol

Project 1 – A Client -Server Application

Date – 5th October 2017

Report by – Abhinuv Nitin Pitale (906039146)

Tools Used – Python 2.7.13, Command Prompt(Windows command Line utility)

Operation System Tested on – Windows 10

Implementation

1. Server

Design Considerations:

BUFFERSIZE = 1024

Uses a polling type of server, which is always on and gets activated for incoming requests

Does not involve multithreading, hence only sequential sockets can be accepted.

Algorithm/Code Flow:

1. Get user input for selecting the port number for the server to be set-up on.
2. Create a TCP socket and bind it to the port.
3. Wait for the server to listen to incoming requests.
4. Accept the incoming socket and create a new connection socket for receiving and sending data.
5. Receive a HTTP query from the incoming socket. (Assumption : The incoming query length will be less than 1024 bits.)
6. Decode the query –
   1. Split the query into 3 parts
      1. Method Used (for eg – GET,POST
      2. Filename Requested (for eg – docTest.txt)
      3. HTTP version
   2. Check if the method used and the Http version is valid. If not, set the state as 400
   3. Check if the filename queried exists on the server. If not, set the state as 404
   4. If b and c is true, set the state as 200
   5. Return state and filename(if exists)
7. Create the Http Response to be sent over the server.
8. Check state and return the query to be sent over the server.
   1. If it is 400, output the following query

HTTP/1.0 400 Bad Request

* 1. If it is 404, output the following query

HTTP/1.0 404 Not Found

* 1. If it is 200,
     1. Read the file as binary.
     2. Compute it’s length
     3. Send the query in this particular format-

HTTP/1.0 404 Not Found

Content-Length: *length*

*CONTENT*

1. Send the data over the socket by computing it’s length and sending it by parts, if the buffersize is overloaded.
2. Close the connection socket and go back to step 3.
3. CLIENT

Design Considerations:

BUFFERSIZE = 1024

CodeFlow/Algorithm:

1. Query the user for host and port number.
2. Create a TCP socket connected to that port.
3. Display connection as success, if it passes step 2.
4. Query user for a filename to retrieve from the server.
5. Create the Http Request:

query = GET /filename HTTP/1.0

1. Send the query via socket
2. Receive the server response and parse it
   1. Read the state from the response
      1. If 400, display ‘Bad Request’
      2. If 404, display ‘File Not Found’
      3. If 200, parse the data in the request.

Open a file with the same name and store the *filecontent* in it.

1. Close the socket.