LAB 4 – SOCKET PROGRAMMING – HTTP

DATE : 17.08.2024

BATCH : N,P,Q

Exercise 4

OBSERVATION :

1.List down the uses of basic commands in networking

- ○ Netstat
- ○ Tcpdump
- ○ Ipconfig
- ○ Nslookup
- ○ Tracert
- ○ Ping
- ○ Arp-a

2. What is HTTP and WWW?

3. To which OSI layer does IP belong?

4. What HTTP response headers do?

5. What is HTTP session state?

6. What is Secure HTTP?

7. What is the current version of HTML?

8. What is HTTP session and what is session id?

9. What is the main usage of session id?

10. Define cookie and list some real time examples where the cookies are used.

EXECUTION :

1. Write a HTTP web client program to download a web page using TCP sockets.

2. A server available in www.students.com/marks/index.html, maintains a database of student marks. On entering the register number of a student in the page, the list of courses taken by the student is sent by the server to the client who has requested for the student marks. The marks secured by the student in each course is available as an individual object in www.students.com/marks/<register-number>/<course-code>. Simulate the HTTP protocol for the transfer of messages between the client and the server for fetching the marks of a student using each of the following types of connections:

1. non-persistent (TCP connection opens and closes for transfer of each object)

2. persistent (TCP connection remains open until all objects are transferred)

Compare the time taken by each type of connection for fetching all the marks of a student.

Implementing HTTP from socket

HTTP stands for Hyper Text Transfer protocol. It is an application layer protocol for communicating data between client and server using browser. We can use sockets to implement HTTP. We have to decode/encode messages according to the HTTP specification (RFC2616) which is just a Text. In this article, we will implement the basics of HTTP protocol.

HTTP server

We will create a TCP socket. The default port is 80 for HTTP. But we will be using 8080 for running our test.

You can use request module to receive response from custom HTTP server.

```
>>> import requests

>>> data = requests.get("http://127.0.0.1:8080")

>>> data.status_code

200

>>> data.text

'Hello World'

>>> data.headers

{}
```

On the server you can see:

```
GET / HTTP/1.1

Host: 127.0.0.1:8080

User-Agent: python-requests/2.22.0

Accept-Encoding: gzip, deflate

Accept: */*

Connection: keep-alive
```

Request

A HTTP request consists of following.

Request line: One line identifying the request type and path.

Headers: An optional set of RFC-822-style headers

Data: An optional data.

Request line

To generalize it we can say the request line consists of

METHOD REQUEST-URL HTTP-VERSION CRLF

The request line of above request is:GET / HTTP/1.1

The method is GET. which means we want to retrieve data from the server. Note that the method is case sensitive. Here the path to resource is /. HTTP/1.0 is the HTTP protocol version we are using. And \r\n is the CRLF. CR-LF means carriage return. It signifies end of line.

Headers

Host, User-Agent, Accept, Accept-Encoding, Connection are request header fields.

Data

In the above request we haven't sent any data. Lets send another request to understand things little better.

```
>>requests.post(url="http://127.0.0.1:8080?a=1", headers= {"header1": "value1", "header2": "value2"},data={"key1": "value1", "key2": "value2"})
```

In the print we can see :

```
POST /?a=1 HTTP/1.1

Host: 127.0.0.1:8080

User-Agent: python-requests/2.22.0

Accept-Encoding: gzip, deflate

Accept: */*

Connection: keep-alive

header1: value1

header2: value2

Content-Length: 23

Content-Type: application/x-www-form-urlencoded

key1=value1&key2=value2
```

Here the method has been changed to post. We can see that the request url has been modified to ?a=1. Our custom headers header1 and header2 have been added. And the data which key1 and key2 we sent is also present.

Browser

We can access the server from browser. Open your browser and type 127.0.0.1:8080

In the terminal, you will see the following result.

```
GET / HTTP/1.1

Host: 127.0.0.1:8080

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:89.0) Gecko/20100101
Firefox/89.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Connection: keep-alive
```

Upgrade-Insecure-Requests: 1

Cache-Control: max-age=0

HTTP client

We can implement our own HTTP client using the same principle. It is a simple socket program.

The above code will print:

HTTP/1.1 200 OK

Hello World

HTTP/1.1 is the protocol we are using.

200 is the status code which means the request was success. Status code is three digits.

OK It is a simple one line phase that tells reason.

Hello World is the body of the message. It is the part that is modifies.

Response

The response has:

One line giving the response code

An optional set of RFC-822-style headers

Optional data