CS 3205: Computer Networks **(Due: 09/03/20)**

Assignment #3 : HTTP Proxy Server

Instructor: Manikantan Srinivasan

Name: Vimarsh Sathia, Arjun Bharat
Roll: CS17B046, CS17B006

1 Aim

The aim of the assignment was to implement a concurrent http proxy server in the C/C++ language, which receives and parses http GET requests from clients, forwards them to the remote servers, and returns the data from the remote server back to the clients.

2 Websites Used

We used the following websites for benchmarking our proxy server.

- www.cse.iitm.ac.in/
- www.cc.iitm.ac.in/

3 Experiments

We were given 2 Python test scripts for testing our code. The description of each test script is given below as follows:

- proxy_tester.py
  For the same http GET request, this script compares the output of our proxy server with that of a direct telnet session.
- proxy_tester_conc.py
  For the same http GET request, this script runs multiple tests, like fetches(both single and concurrent), split fetches(both single and concurrent), and benchmarks the proxy server by using **ab**, the Apache http server benchmarking tool.

The results are summarized and explained in the observations below.

4 Observations

4.1 For proxy_tester.py

```
arjun@arjun-XPS-13-9360:~/Documents$ python proxy_tester.py proxy
Binary: proxy
Running on port 41610
### Testing: http://www.cse.iitm.ac.in/
http://www.cse.iitm.ac.in/: [PASSED]

### Testing: HTTP://www.cc.iitm.ac.in/
HTTP://www.cc.iitm.ac.in/: [PASSED]

Summary:
        2 of 2 tests passed.
arjun@arjun-XPS-13-9360:~/Documents$
```

Figure 1: Testing simple GET requests

```
    # Retrieve directly
    direct_data = get_direct(host, hostport, urldata[2])

    passed = True
    for (proxy, direct) in zip(proxy_data, direct_data):
        if proxy != direct and not (proxy.startswith('Date') and direct.startswith('Date')) and not (proxy.startswith('Expires') and direct.startswith
        ('Expires')):
            print 'Proxy: %s' % proxy
            print 'Direct: %s'  % direct
            passed = False

    return passed

def get_direct(host, port, url):
    '''Retrieve a URL using direct HTTP/1.0 GET.'''
    getstring = 'GET %s HTTP/1.0\r\nHost: %s\r\nConnection: close\r\n\r\n'
    data = http_exchange(host, port,  getstring % (url, host))
    return data.split('\n')

def get_data(host, port, url):
    '''Retrieve a URL using proxy HTTP/1.0 GET.'''
    getstring = 'GET %s HTTP/1.0\r\nConnection: close\r\n\r\n'
    data = http_exchange(host, port,  getstring % url)
    return data.split('\n')

def http_exchange(host, port, data):
    conn = telnetlib.Telnet()
    conn.open(host, port)
    conn.write(data)
    ret_data = conn.read_all()
    conn.close()
    return ret_data
```

Figure 2: Relevant code from the proxy_tester.py

The proxy server accepts the incoming connection from the client, forwards the request to the remote server and returns the result of the remote server back to the client. The connection with the client is established using a streaming TCP socket.
The test-script compares the output of the proxy to a direct connection, and checks equality. The **telnet** protocol is used to connect to the remote and proxy server.

4.2 For proxy_tester_conc.py - Concurrent and split fetches

```
arjun@arjun-XPS-13-9360:~/Documents/CS3205/Assignment 3/HTTP-Proxy-Server/additional_materials$ python proxy_tester_conc.py ../proxy
Binary: ../proxy
Running on port 12375
### Testing: http://www.cse.iitm.ac.in/
http://www.cse.iitm.ac.in/: [PASSED]

### Testing: http://www.cc.iitm.ac.in/
http://www.cc.iitm.ac.in/: [PASSED]

### Testing 2 concurrent connects to http://www.cc.iitm.ac.in/
Connect to http://www.cc.iitm.ac.in/, 2 concurrently: [PASSED]

### Testing 10 concurrent connects to http://www.cc.iitm.ac.in/
invalid buflen 0invalid buflen 0invalid buflen 0invalid buflen 0invalid buflen 0invalid buflen 0invalid buflen 0invalid buflen 0Connect to http://www.cc.iitm.ac.in/, 1
0 concurrently: [PASSED]

### Testing 2 concurrent fetches to http://www.cc.iitm.ac.in/
invalid buflen 0invalid buflen 0invalid buflen 0invalid buflen 0Fetch to http://www.cc.iitm.ac.in/, 2 concurrently: [PASSED]

### Testing 10 concurrent fetches to http://www.cc.iitm.ac.in/
Fetch to http://www.cc.iitm.ac.in/, 10 concurrently: [PASSED]

### Testing 2 concurrent split fetches
Fetch to http://www.cc.iitm.ac.in/, 2 concurrently: [PASSED]

### Testing 10 concurrent split fetches
Fetch to http://www.cc.iitm.ac.in/, 10 concurrently: [PASSED]
```

Figure 3: Concurrent and split fetch requests

Since the proxy server forks a new process for every new incoming connection, it supports concurrent requests from multiple clients. Figure 3 gives details about concurrent and split requests to the proxy server.
A **split** fetch is one where the http request is transmitted in 2 different messages to the proxy server. The server has to wait for the client to send a message which terminates with a **CRLF**(Carriage Return Line Feed), to then preprocess and send to the remote server.

4.3 For proxy_tester_conc.py - Apache benchmark

```
### Testing apache benchmark on args [-n 20 -c 1]
    This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
    Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
    Licensed to The Apache Software Foundation, http://www.apache.org/

    Benchmarking www.cse.iitm.ac.in [through 127.0.0.1:12375] (be patient).....done


    Server Software:        Apache/2.4.10
    Server Hostname:        www.cse.iitm.ac.in
    Server Port:            80

    Document Path:          /
    Document Length:        54155 bytes

    Concurrency Level:      1
    Time taken for tests:   0.881 seconds
    Complete requests:      20
    Failed requests:        0
    Total transferred:      1086500 bytes
    HTML transferred:       1083100 bytes
    Requests per second:    22.71 [#/sec] (mean)
    Time per request:       44.035 [ms] (mean)
    Time per request:       44.035 [ms] (mean, across all concurrent requests)
    Transfer rate:          1204.76 [Kbytes/sec] received

    Connection Times (ms)
    min  mean[+/-sd] median   max
    Connect:        0    0   0.0      0       0
    Processing:    42   44   1.9     43      49
    Waiting:       21   23   1.8     23      29
    Total:         42   44   1.9     43      50

    Percentage of the requests served within a certain time (ms)
    50%     43
    66%     45
    75%     45
    80%     45
    90%     47
    95%     50
    98%     50
    99%     50
    100%     50 (longest request)
http://www.cse.iitm.ac.in/ with args -n 20 -c 1: [PASSED]
```

Figure 4: Apache benchmark for 20 requests, concurrency 1

```
### Testing apache benchmark on args [-n 200 -c 10]
    This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
    Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
    Licensed to The Apache Software Foundation, http://www.apache.org/

    Benchmarking www.cse.iitm.ac.in [through 127.0.0.1:12375] (be patient)
    Completed 100 requests
    Completed 200 requests
    Finished 200 requests


    Server Software:        Apache/2.4.10
    Server Hostname:        www.cse.iitm.ac.in
    Server Port:            80
invalid buflen 0
    Document Path:          /
    Document Length:        54155 bytes

    Concurrency Level:      10
    Time taken for tests:   1.187 seconds
    Complete requests:      200
    Failed requests:        0
    Total transferred:      10865000 bytes
    HTML transferred:       10831000 bytes
    Requests per second:    168.46 [#/sec] (mean)
    Time per request:       59.362 [ms] (mean)
    Time per request:       5.936 [ms] (mean, across all concurrent requests)
    Transfer rate:          8936.91 [Kbytes/sec] received

    Connection Times (ms)
    min  mean[+/-sd] median   max
    Connect:        0    0   0.1      0       0
    Processing:    45   58  16.5     53     137
    Waiting:       22   31  14.5     26     110
    Total:         45   58  16.5     53     138

    Percentage of the requests served within a certain time (ms)
    50%     53
    66%     56
    75%     58
    80%     59
    90%     76
    95%    102
    98%    118
    99%    136
    100%    138 (longest request)
http://www.cse.iitm.ac.in/ with args -n 200 -c 10: [PASSED]
```

Figure 5: Apache benchmark for 200 requests, concurrency 10

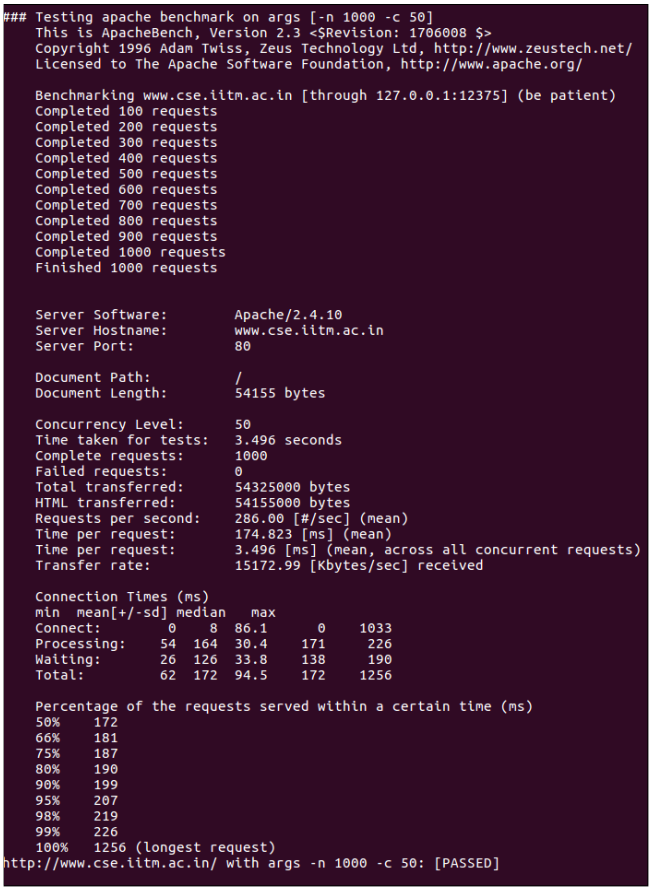

Figure 6: Apache benchmark for 1000 requests, concurrency 50

The Apache bench tools is used for load testing a http web-server. This involves concurrently requesting http GET requests, and retrieving them, and performing some calculations like average time taken per request, length of received http message, number of requests and so on.
All the metrics after 3 Apache bench test are shown in Figures 4,5 and 6. The arguments for every test are shown in the first line of the screenshot.

5 Conclusion

We learnt the following things from this assignment:

- Implementation of a proxy server and raw parsing of http messages.
- A brief idea of how concurrent clients are serviced in real-time servers (by using a client-specific thread or process)
- How to do load testing and web page retrieval for server using **apache2-utils** and **telnet**.