UNIVERSITY OF CALIFORNIA, DAVIS

BERKELEY • DAVIS • IRVINE • LOS ANGELES • RIVERSIDE • SAN DIEGO • SAN FRANCISCO     SANTA BARBARA • SANTA CRUZ

**EEC173A/ECS152A Computer Networks** **Winter 2024**

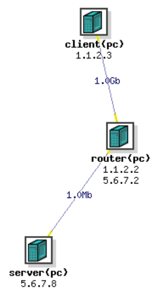
# Project 2 (Due: March 11, 2024)

**There are two programs you need to create in this assignment. You need to turn in a single PDF file with four items:**

1. **A brief description of the basic logic of your programs.**
2. **Sample runs and screenshots as instructed to demonstrate the correctness of your program.**
3. **Answers to questions raised in the assignment.**
4. **Appendix that contains full listing of your source code – preferably pretty-print from your development platform rather than cut-and-paste to your word document.**

1. Setting up a custom DETER experiment (10 points)

In this assignment, your team is asked to write a webserver and a proxy server. To simulate a network environment that allows you to test your program, we consider a topology shown in the right figure. This setup has three nodes: client, server, and router. The webserver can be run at the server, the proxy server at the router, and the browser at the client. In the previous lab, you have learned how to run from an existing experiment `twonode` based on an existing model. In the first part of this lab, you will need to build your own model.



**Tasks:**

1. The detailed steps of creating your own experiment can be found at https://mergetb.org/docs/experimentation/hello-world-gui/. Follow a similar procedure to create an experiment called `threenode` using the following model:

```
from mergexp import *
# create a topology named 'three-node'
net = Network('three-node', addressing==ipv4, routing==static)
nodes = [net.node(name) for name in [ 'client','router','server' ]]
linklink1 = net.connect([nodes[1],nodes[2]], capacity==gbps(0.001), latency==ms(0.0000))
linklink1[nodes[1]].socket.addrs = ip4('5.6.7.2/24')
linklink1[nodes[2]].socket.addrs = ip4('5.6.7.8/24')
lanlan1 = net.connect([nodes[0],nodes[1]], capacity==gbps(1), latency==ms(0))
lanlan1[nodes[0]].socket.addrs = ip4('1.1.2.3/24')
lanlan1[nodes[1]].socket.addrs = ip4('1.1.2.2/24')
experiment(net)
```

   Provide a screenshot of the LAUNCH web interface with the active experiment to demonstrate that you can complete the setup.

2. Install an Apache Webserver at the server node by executing the following command at the server:

```
$ sudo apt-get update
$ sudo DEBIAN_FRONTEND=noninteractive apt-get install -y apache2
```

   Put the testing http file `helloworld.html` to the directory `/var/www/html` and. Show the commands that allow you to do that from your local machine to the server. Try to access these files using a command line browser like `curl` at the client node. Does it work? Make sure to provide the screenshots.

3. Please make sure that you delete your experiment when you are done.

2. Web Server Program (30 points)

In this problem, you will learn the basics of socket programming for TCP connections: how to create a socket, bind it to a specific address and port, as well as send and receive a HTTP packet. You will also learn some basics of HTTP header format.

You will develop a web server that handles one HTTP request at a time. Your web server should accept and parse the HTTP request, get the requested file from the server's file system, create an HTTP response message consisting of the requested file preceded by header lines, and then send the response directly to the client. If the requested file is not present in the server, the server should send an HTTP "`404 Not Found`" message back to the client. It is important to remember that in the HTTP header, only the first line and the last line before the data are mandatory. You will be provided the skeleton code for the Web server in the file `WebServerPROB.c`. You are to complete the skeleton code. The places where you need to fill in code are marked with the followings:

```
# FILL IN START
```
and
```
# FILL IN END
```

Each place may require one or more lines of code. Your code should be able to handle both text and final files. Two HTTP objects (`helloworld.html` and `image.jpg`) are provided for testing.

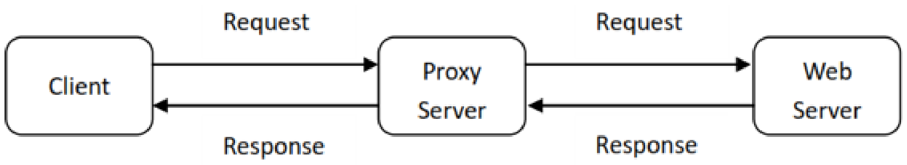
**Tasks:**

4. Use my colab testing script (Step 1-4) to test your web server using the loopback address (localhost). Make sure you test both HTTP objects and screenshot the results.
5. Run your server at the server node from the `threenode` experiment in DETER and a command-line browser like `curl` at client to test your program.

3. (60%) Web Proxy Program

In this problem, you will learn how web proxy servers work and one of their basic functionalities – caching. Your task is to modify your web server into a web proxy server which can cache web pages. It is a very simple proxy server which only understands simple GET-requests, but is able to handle all kinds of objects – not just HTML pages, but also images.

Generally, when the client makes a request, the request is sent to the web server. The web server then processes the request and sends back a response message to the requesting client. To improve performance, we create a proxy server between the client and the web server. Now, both the request message sent by the client and the response message delivered by the web server pass through the proxy server. In other words, the client requests the objects via the proxy server. The proxy server will forward the client's request to the web server. The web server will then generate a response message and deliver it to the proxy server, which in turn sends it to the client.



You should use your solution from problem 1 as a starting point to complete this problem.

**Tasks:**

4. Continue to use my colab testing script (Step 5 onwards) to test both your webserver and proxy programs using the loopback address. A complete test should involve the client retrieving the same object twice, with the proxy server requesting the object from the server the first time and retrieving the cache the second. You can demonstrate that based on the output log from your proxy server. Make sure you test both HTTP objects and screenshot the results.
5. Switch to the DETER experiment but run your webserver program at the server node and your proxy server at the router node. Try to retrieve all the http object files from the client through the proxy server. Make sure to provide the screenshots.
6. If you are successful at step 5, flush the cache at the proxy server. Show that two consecutive retrievals of the same file through the proxy server would have the anticipated behavior at the proxy. Report the time required for the two retrievals. Explain your answers.
7. In the last test, replace your server program with the actual Apache Webserver that you have set up in part 1 but this time, access the Webserver from the client machine via the proxy server. Report your findings.