Norges teknisk-naturvitenskapelige universitet

Institutt for informasjonssikkerhet og kommunikasjonsteknologi

TTM4100 – Kommunikasjon – Tjenester og nett

Praksisøving 2

HTTP Web Server

Denne praksisøvingen er én av seks praksisøvinger i emnet. Du må levere og få godkjent minst fire av disse praksisøvingene, i tillegg til seks av åtte teorivinger, for å kunne gå opp til eksamen. Hvis du lurer på noe angående øvingen kan du stille spørsmål på Piazza eller få veiledning av læringsassistent. Veiledning foregår på Teams. Se emnesiden på Blackboard for tidspunkter og fremgangsmåte.

Oppgavene er ment å fokusere på læring og forståelse, og ikke debugging av kode. Du kan velge om du vil svare på norsk eller engelsk.

Les oppgaven og kildekoden nøye. Det anbefales på det sterkeste at du leser og forstår kapittel 2.7.2 i pensumboken før du starter øvingen.

Lever svarene dine som **én enkelt PDF** på Blackboard innen fristen. For å få svaret i én PDF kan du f.eks. kopiere kildekoden og bildet av responsen i nettleseren inn i et Word-dokument og lagre som PDF.

Innleveringsfrist: **søndag 14. februar 2020, kl. 23:59.**

The goal of this exercise is to understand how the application code sends and receives data through a socket. You will learn the basics of socket programming for TCP connections in Python: 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.

N.B. It is strongly recommended to read **Sect. 2.7.2** of the textbook before (and while) doing this programming lab.

Code

You will find the skeleton code for the Web server in the file "Skeleton WebServer.py". You are to complete the skeleton code with any Python version of your choice. Beware there are some syntax differences in Python 2.7 and Python 3. The places where you need to fill in code are marked with `#FILL IN START` and `#FILL IN END`. Each place may require one or more lines of code. The comments in the skeleton explain what the code you are to fill in is supposed to do.

Running the Server

Put an HTML file (e.g., HelloWorld.html) in the same directory that the server is in. If you do not know how to create an HTML file, there are many examples in the web; a possible one is http://www.w3schools.com/html/html_editors.asp.

Run the server program. Determine the IP address of the host that is running the server (e.g., 128.238.251.26). You can do this locally by using the default localhost IP (127.0.0.1). From another host, open a browser and provide the corresponding URL. For example:

http://128.238.251.26:6789/HTML_files/HelloWorld.html

'HTML_files' and 'HelloWorld.html' are the names of the folder and file you placed in the server directory. Note also the use of the port number after the colon. You need to replace this port number with whatever port you have used in the server code. In the above example, we have used the port number 6789. The browser should then

display the contents of HelloWorld.html. If you omit ":6789", the browser will assume port 80 and you will get the web page from the server only if your server is listening to port 80.

Then try to get a file that is not present at the server. You should get a “404 Not Found” message.

Tips: If you cannot connect to your HTTP server from a remote computer, make sure your two computers are in the same network, or try disabling firewall on the server computer.

What to Hand in

You will hand in the complete server code along with the screenshots of your client browser, verifying that you actually receive the contents of the HTML file from the server.

Optional Exercises

1. Currently, the web server handles only one HTTP request at a time. Implement a multithreaded server that is capable of serving multiple requests simultaneously. Using threading, first create a main thread in which your modified server listens for clients at a fixed port. When it receives a TCP connection request from a client, it will set up the TCP connection through another port and services the client request in a separate thread. There will be a separate TCP connection in a separate thread for each request/response pair.
2. Instead of using a browser, write your own HTTP client to test your server. Your client will connect to the server using a TCP connection, send an HTTP request to the server, and display the server response as an output. You can assume that the HTTP request sent is a GET method.
   The client should take command line arguments specifying the server IP address or host name, the port at which the server is listening, and the path at which the requested object is stored at the server. The following is an input command format to run the client:

   ```
   client.py server_host server_port filename
   ```