# Lab Exercise 1: Concurrent Web Server

The goal of this exercise is to help introduce you to basic client-server programming using sockets with python. You are given starter code that implements a simple "echo" client server program. You will read this code to familiarize yourself with socket programming. Then, you will modify the code to implement a (highly simplified) concurrent Web server.

## Background

### HTTP (HyperText Transfer Protocol) (RFC1945)

HTTP has been in use by the World-Wide Web global information initiative since 1990. On the Internet, HTTP communications generally take place over TCP/IP connections. The default port is TCP 80, but other ports can be used.

There are two basic operations in HTTP: request and respond. A request message from a client to a server includes, within the first line of that message, the method to be applied to the resource (GET, HEAD, or POST. See RFC1945 for details), the identifier of the resource, and the protocol version in use. After receiving and interpreting a request message, a server responds in the form of an HTTP response message. A traditional HTTP GET request will request a file from a server using the format listed below:

> GET <file path> HTTP/1.0<CR><LF><CR><LF>

This command will be sent to the server as simple text (bytes). You may assume the control command for <CR> and <LF> (Carriage Return and Line Feed) are \r and \n respectively. As an example, consider that the client sends the following string to the server.

> GET file.txt HTTP/1.0\r\n\r\n

This implies that the client wishes to get the file <server current directory>/file.txt from the server using the HTTP protocol.

## Concurrent Web Server

An HTTP server must also be able to handle multiple client requests, i.e., more than one client can connect to the server and be served simultaneously. The server must retain the same ability to interpret GET commands and appropriately respond with the requested file.

In order to accomplish this, the server must be able to spawn a new thread to service a new client request. The new thread is responsible for receiving and processing the client request and responding appropriately.

Note that in order to service multiple clients that may send a request anytime, the server should not exit/close. (Hint: Infinite loops?)

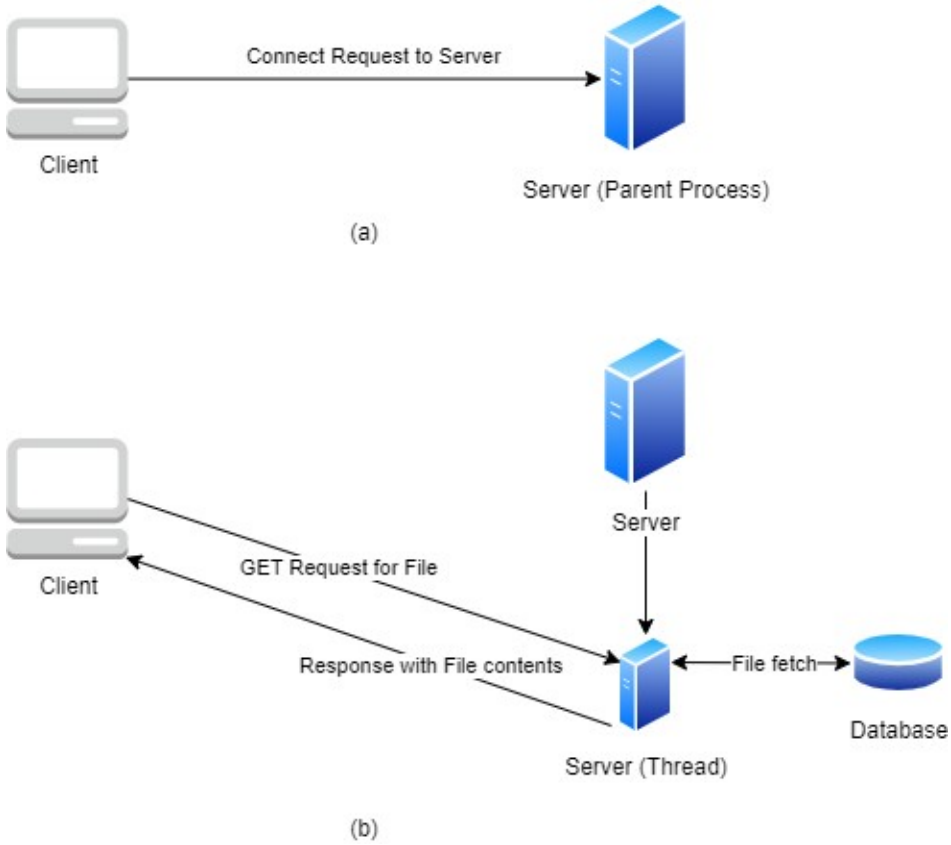


Fig 1. Concurrent Server Working. (a) Client sends a “connect” request to the server. (b) The server launches a new thread to service the client. It receives the GET request, fetches the file from the database and responds with the file contents.

# What you need to do in this exercise.

In this exercise, you are provided a starter code, which implements an echo client and echo server. You must modify this code to create a concurrent Web server.

The echo client first connects with the echo server, sends a pre-decided message to the server, and waits for the response. The echo server opens a listening socket and waits for the client to send a request. On receiving a message from a client, the server sends it back to the client. A small processing delay is added at the server, controlled by a configurable parameter (Database_Delay). This can be interpreted as delays incurred by the server back-end in the real world. You are urged to first run this code and understand the various socket programming constructs being used.

You must then modify the server code to (i) parse an incoming GET message, identify the file requested by the client, and send the contents of the file to the client; and (ii) handle multiple clients simultaneously using threads. A minor modification is also needed to the client code. Rather than send an arbitrary text message to the server, the client must be modified to send a GET message in the format described above. The client should fetch the file “testfile”, and it is acceptable to hardcode this filename as part of the GET message for the purposes of this exercise. The client should print the response from the server as is.

## Running your code

You can run the client or server python files by running:

> python client.py [server hostname] [server port] [client port]

> python server.py [server port]

The server should be executed first in a separate terminal. While it is running, each client should be launched in a separate terminal with the server hostname and the port.

**Notes:**

- The server hostname is generally “localhost” if the server and client are running on the same machine.
- The server port and client port should be different if they are launched on the same machine otherwise it can lead to bind errors.
- While Web servers traditionally run on port 80, you need to use a non-reserved port (1024 or higher) for both the client and server in this exercise.

## What you need to submit

You must submit the modified client and server codes. Please do not change the file names of the client and server since the auto-grading code assumes these names. Please also ensure the original test file is present in the directory.
***Do not submit any binaries*. Your git repo should only contain source files; no products of compilation.**