Department of Computer Science
University of Pretoria

Computer Networks
COS 332

Study Guide (Practical Assignments)
Version 2021.01 (Draft; see page 3)

2021

Contents

Chapter 1

Some general requirements

> Note that tis is a draft of the final document; the subversion indicates the number of the final assignment in this version of the draft.
>
> The essence of the assignments will not change as new versions are released, but typos may be corrected.

Assignments need to be uploaded to the CS server by the due date. However, marks will only be awarded after the software has been demonstrated by its author(s). Dates of these demonstrations will be communicated in due course.

All programming assignments should be demonstrated as finished products: While you should be able to show your code on request, the development environment should normally not be visible during a practical demonstration. Do not hardcode any facets of your program that may have to change if you demonstrate it in a different environment (such as mail servers, IP addresses, network masks, etc.)

You are only allowed to use code that you found on the Web and elsewhere if the licence conditions of that code allows you to use it. If you use such code, it should be clearly documented that you used the code (as well as where you got it from). Furthermore, that code should not provide the core functionality of the program you are submitting — in this course this means that such code should not provide any data communications functionality at all.

Note that plagiarism and/or violation of copyright are serious offences. We accept that you are familiar with the remarks about plagiarism as set out in Part B of the COS332 study guide and the University Regulations in general.

All the assignments in this guide may be completed by an individual student or as a team consisting of two students. When two students collaborate, they are allowed to upload identical submissions. The names of both team members should be stated as part of the code they uploaded. Both should participate in the single demonstration of a given assignment. Under normal conditions both team members will be awarded the same mark.

Chapter 2

Finding a host for your servers

2.1 Servers and clients

Network services are provided by servers that are accessed by clients. In some cases servers talk to one another in order to complete a request.

The term *server* is used with two distinct meanings. One the one hand a server may be a computer which runs the necessary software to service requests. On the other hand a server may be the software running on such a computer. Where necessary, we will refer to the computer as a *host* and the specific software as a *dæmon* (though both terms need further disambiguation to use them correctly).

To illustrate, consider a company who buys a computer to use as 'server' and then installs software on it to serve web pages, handle email and act as an FTP repository. They may refer to this computer as the 'server'. The software that handles web requests will often be `httpd`, where the `http` part indicates that it deals with the HTTP protocol, and the `d` suffix indicates that it is the dæmon (server) for this protocol. Similarly, FTP requests may be handled by a program called `ftpd` — for similar reasons. On the other hand, the dæmon used to forward email may be `sendmail`, without any suffix to indicate that it is in fact a dæmon.

In this module you will be expected to developing software that, in some cases, work as dæmons to be used by specific client software. The client software will be readily available; since you will develop the dæmon, you will have everything you require to complete a working system.

In other cases you will be expected to write client software that should communicate with a dæmon. Unfortunately for this module (but, generally fortunately) organisations restrict access to their servers. Hence you will most probably need to supply your own server to complete a number of the assignments. This chapter is intended to guide you through the process of setting up your own server.

2.2 Options

A number of options exist to obtain such a server (computer). You may already have a Linux installation available. Or your computer may use a variant of Unix as (the foundation of) its operating system, that allows one to install server software. Microsoft Windows has traditionally been challenging in this regard: while server software is typically available, they often cost money, and it may be challenging to find information about them on the web.

Whatever solution you choose, verify that you are able to install well-known server software on your platform.

2.2.1 Standalone Unix

As already mentioned, your one option is to use a (standalone) Linux installation. (This includes the case of computers that are configured to dual boot Linux and some other operating system; to complete the assignments, such a machine should be booted as a Linux computer.)

In general Ubuntu is a good choice; the 'standard' option for the module is deemed to be the server edition of `Ubuntu 20.04 LTS`. If you choose another distribution, less assistance may be available. However, feel free to use other options. The lecturer is quite fond of his Mint installations.

2.2.2 Unix from the Microsoft store

Note that a number of Linux for Windows distributions are available in the Microsoft Store. The well-known ones are all free.

2.2.3 Virtual machines

A solution that is known to work well is based on Oracle's VirtualBox virtual machine. Some additional details are therefore provided for this option.

In order to create your own server on VirtualBox you will need a computer and a disc. VirtualBox should be installed on the computers in the Informatorium. To install VirtualBox on your own computer, you can download it from `https://www.virtualbox.org`

One good option is to use the server edition of `Ubuntu 20.04 LTS` (or one of its subversions) as the operating system of our virtual host. The system may be downloaded from various mirrors around the Internet. Expect a download of a few megabytes.

As noted, you also need a disc for your host. Any USB drive that can hold a file of about 16GB should work fine. (The contents of this drive will not be erased;

a new set of files will simply be added.) Proceed to install the operating system on this removable disc after you created a new virtual machine with VirtualBox. Do not install any of the servers (dæmons) that the installation process offers to install. Note that this installation process takes a few minutes that tend to feel longer than it is. Also be prepared to repeat the installation if necessary - such processes do not always proceed as planned. There are copious amounts of information available on the web that should guide you through any problems you encounter.

Be aware of the different uses of the term *host* in networking in general, and in VirtualBox. In networking a host is simply a computer connected to a network (possibly acting as a server). In VirtualBox, the computer that you create is known as a *guest*, which is 'hosted' on the physical computer. Hence, our `Ubuntu` computer will in `VirtualBox` be known as the guest, while the real computer (possibly running Windows) will be known as the *host*. Install your virtual computer with networking configured such that the host (Windows?) can talk to your Ubuntu computer. If possible, other real computers on the network should also be able to talk to your Ubuntu computer. For details, read more about "virtual networking" on VirtualBox. To repeat: In this context you want the host to be able to talk to the guest. The guest should also be able to talk to the host, but that is not a challenge.

Note that, after you have set up your network adapter, we will revert to talking about our Ubuntu computer as the host.

Once your installation is complete, you should be able to boot your virtual host, and log in. To shut it down, simply enter the following command:

```
sudo shutdown -h now
```

To boot your host on another (physical) computer you may have to create a new virtual machine, pointing to the disc you already have. It's best to try it early in the process to be certain that you can retrieve your host whenever required.

Chapter 3

Practical assignment 1

3.1 Background

When the web was invented, web pages were encoded as static HTML pages. In addition, web pages could be generated by CGI programs. As time progressed a number of mechanisms were developed to shift (some) execution from the web server to the browser. Examples include Java applets, Javascript and a variety of other mechanisms to enable web servers to serve ‘active’ content. However, HTML remains the major notation used to encode content (and active content is typically incorporated into the HTML encoded pages). In this assignment you are allowed to use active content, but static HTML pages will suffice. In subsequent assignments active content will not be allowed. However, in all assignments you may use CGI programs (if the assignment expects or allows you to produce web-based output).

HTML has been revised several times and HTML 5 is the current standard. You are expected to use (correct) HTML 5 for all assignments where HTML is used.

Web pages may be served by general purpose servers, or servers designed for a specific application. Apache is (and has for a long time been) the most popular general purpose web server. Chapter 2 describes the manner in which a virtual computer could be instantiated for practical assignments in this module. After your virtual computer has been built it should be simple to install Apache on it (if not already present after initial installation of the selected Linux distribution). Any search engine will point one to further instructions — if required.

In order to use the HTTP server one typically simply has to copy the HTML files to the appropriate ‘home’ directory and the server will start serving those.

To enable interaction the early web servers provided a mechanism that was (and still is) called CGI (*Common Gateway Interface*). A CGI program generates its output in HTML. A simple Hello World CGI program, may, look as follows (using some imaginary programming language):

```
print "<!DOCTYPE HTML>"
print "<HEAD>"
print "<TITLE>Example</TITLE>"
print "</HEAD>"
print "<BODY>"
print "<P>Hello world</P>"
print "</BODY>"
print </hHTML>"
end
```

If you run this program on a console it will simply print out the marked up "Hello world" lines. However, when installed in the CGI directory of a web server, it will 'print' its output to the pipe that connects the web server to the browser, and the page will be rendered by the web browser. For it to work, the program should be executable code and the web server must be authorised to execute it. Since the CGI and 'home' directories are not always treated equally, you may need a file called `index.htm`, `index.html`, or similar suitable 'start' file that will be displayed when you open the home directory of the server. This (static) file may include an `<a href="..."` link that points to the CGI program. When one clicks on the link, the CGI program will be executed.

Note that a CGI program may also 'print' `<a href="..."` lines — just as if they were embedded in a static file.

Of course, writing such a program that simply produces static content is pretty useless; it is intended to be used in a context where what it outputs will change depending on current conditions. As an example, the CGI program may be connected to a database that operates as a back-end and that, for example, contains the types and numbers of items a merchant has in stock. Running this program will then not display static data, but the current stock levels of the merchant.

More generally, the CGI program may need some inputs so that one may, for example, query the stock level of some specific item. However, in this assignment we will not assume that one can provide the CGI program with inputs. Another simplifying assumption for this assignment is that the back-end will consist of a simple data file that the CGI program can read from and write to. Our intention is that you should demonstrate that

- You are able to install and use the web server;
- You can install one or more simple static pages in the appropriate server directory;

- You can install one or more executable CGI programs in the appropriate directory; and
- Get it all to work via a browser.

You may use any language to write your CGI programs. Run them from the console / command line to test them. As an example, if your CGI program is called `test`, run `./test` from the command line; you should see the output that should look like something encoded in HTML. Better still, execute `./test > test.htm` and the output will be redirected to `test.htm`; open `test.htm` in your favourite browser and the expected output should be rendered properly. Also consider validating `test.htm` using

`https://validator.w3.org/`

to be sure that your program generates valid HTTP. (It is a good idea to also validate ant static HTML files you create for this — and other — assignments.)

Recall that the 'back-end' of your system will consist of a simple data file in an appropriate directory and with an appropriate initial value. Remember to create it once you have read the remainder of the assignment.

The assignment focusses on so-called server-side execution, so you are not allowed to execute any code in the browser — hence no JavaScript or other client-side software is allowed.

3.2 Your assignment

The time zone in South Africa is GMT+2 (or UTC+2). Time in Ghana happens to be GMT+0 (or UTC+0). Initialise your 'back-end' file to 2.

You are expected to write three CGI programs. The first program should, when executed, display the current time at the time of execution in the time zone indicated in the back-end file. It should just display the current time; the assignment does not expect you to write a running clock. (You may, if you want to, but recall that you are not allowed to use any client-side programming.)

Two links should be displayed along with the time: *Switch to South African Time* and *Switch to Ghana Time*. The first link should execute a program that sets the back-end file to 2. The second link should execute a program that sets the back-end file to 0. Both programs will have to output some HTML encoded page; you can experiment to see what works best.

You may use a back-end file with some additional data. For example, the file may, in addition to the time offset, also include the name of the country, and possibly even some other relevant facts about the country. Your CGI program(s) can then use that data to display more relevant data such as

```
The current time in <country> is <time>.
```

or

```
The current time in <capital city>, <country> is <time>.
```

Package the files (static HTML and executable CGI programs) in an archive that will install the files into the correct directory irrespective of the current directory from which the archive is unpacked. This requires you to use `tar`, `tgz` and/or `zip` to package the files using absolute paths. The correct absolute paths may be changed in the Apache configuration file(s), but you are expected to use the default locations from the Linux distribution you use for this assignment. (If your CGI programs are compiled, let the archive unpack your source code to your home directory — or a suitable subdirectory in your home directory.)

Note that the intention is not to show your prowess to build sites. It is intended to show that you are able to achieve the goals set out above A tiny site will thus suffice.

3.3 Assessment

This assignment — like most of those that will follow — will count out of 10. If you manage to execute your assignment perfectly, you will be awarded 10. (This will not be true for subsequent assignments). However, marks will be deducted for aspects that do not work correctly; examples include files that are installed in incorrect directories from your archive, a web server that does not serve the pages correctly, or output that is clearly wrong.

This is the end of the current draft; see page 3.
March 31, 2021