Department of Computer Science
University of Pretoria

Computer Networks
COS 332

Study Guide (Practical Assignments)
Version 2023.00

2023

Contents

Chapter A

Some general requirements

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 B

Finding a host for your servers

B.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.

B.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.

B.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.

B.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.

B.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 1

Practical assignment 1

1.1 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). However, no browser-side active content should be used in any of the assignments of this module. Note that the use of CGI programs will be required in this assignment (and you are looowed to use CGI programs in nay of the other assignments, when meaningful. (Most assignments will not be web-based, so CGI would only be an option in a few of the assignments.)

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 B 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 dynamic content the early web servers provided a mechanism that was (and still is) called CGI (*Common Gateway Interface*). A CGI program is a program that 'generates' output in the form of HTML; this HTML will then be sent to the browser, were it would be rendered.

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 input. Another simplifying assumption for this assignment is that the back-end will consist program (without any need for a database). 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 any 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.

Package the files for this assignment (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.[1] 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.

[1]Many students have in the past struggled to figure out how to use absolute paths. Figure this out early, because it is really simple once you are aware of the concept!

1.2 Your assignment

Write one or more CGI programs and/or static HTML files that display two random numbers, along with the instruction that the user should click on the larger of the two numbers, Each of the numbers is displayed as a hyperlink (that is, between `<a ...>` and `</a>` tags). The larger of the two numbers will then be linked to, say, a file called `right.htm`, while the other may be linked to a file called `wrong.htm`. The file `right.htm` contains an appropriate congratulatory message, while `wrong.htm` contains whatever you want to display to anyone unable to select the correct answer. Both these files contain a hyperlink that allows the user to try again.

You do not have to use these naming conventions or set of files — a single CGI script may suffice.

Obviously the suggested solution contains a major flaw: If the user hovers above a number, it may be obvious that it is a link to `right.htm` or `wrong.htm`. This may mean that the user does not really think about the challenge that needs to be solved. For this assignment this flaw is acceptable — unless your personality does not allow you to live with such a flawed solution.

Note that you should consider the possibility that the browser may cache a web page, and that your newly generated numbers are not displayed; the browser just sticks to one set of numbers. Insert an appropriate meta-tag to ensure that this does not happen.

In case you missed it: The system is dynamic since the CGI program generates a new pair of numbers for each round, and it should be unpredictable whether the first or second number in any round would be the larger of the two.

Is it best to write a program such as this to execute in an infinite loop, or should there be some option to 'exit'?

1.3 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.

Chapter 2

Practical assignment 2

2.1 Background

A Telnet client is available for all well-known operating systems. To activate it, one simply enters the command

```
telnet <computer name>
```

or, sometimes,

```
telnet <computer name> <port number>
```

The Telnet client is in the first place meant for communication with a Telnet server. One may, for example, use Telnet to work on a Unix computer located in any place in the world as if one were working from a terminal directly connected to the computer. Telnet may, however, communicate with any server. Essentially it simply sends each character that is entered on the keyboard to the server — and displays each character that the server sends to the client on the client's screen.

In the basic configuration even the characters typed on the keyboard are not displayed, but sent to the server which 'echoes' each character so that one sees what one types. This has the advantage that one knows while you are communicating with the server since each character that one sees has been sent to the server and returned by it.

In contrast, the default setting of some Telnet clients are to display whatever is typed, and then *not* display the character when (and if) it is echoed. This is determined by a property known as *localecho*, if *localecho* is on, then the Telnet client will locally 'echo' everything that is typed, without waiting for it to be echoed by the server.

In a number of instances we will use Telnet to interact with a server that was never intended to be used via Telnet. In those cases one wants *localecho* to be on, since the server has no reason to echo any input it receives. If the default setting of your client is *localecho off*, it needs to be set to on. To do this, establish the con-

nection with your server. Then press `^]` (hold `Ctrl` and press ']'). This causes you to 'escape' into the client shell, and you will be presented with a > prompt. Then enter
```
set localecho
```
and you will be informed that "Local echo [is now] on." If you want to turn it off, use the command
```
unset localecho
```
To "escape" from the client shell, enter the command `quit` and you will be communicating with the connected server (rather than the local client) again.

For this assignment you are expected to write a tiny, special-purpose Telnet server. Ideally it should work with a Telnet client irrespective of the setting of *localecho*. Hence it is advisable to echo characters that the user is supposed to see. However, test it with a client where *localecho* is on: you do not want to see every character twice — once when it is locally echoed and once when the server echoes it...

2.2 Your assignment

You are expected to write a server that maintains a 'database' of your appointments (date, time, with whom, etc). All the usual operations such as searching, addition and deletion must be available. After your server has been activated, ¡b¿all¡/b¿ interaction with it has to occur through Telnet. Your server is also responsible to echo everything that is entered. Note that the 'database' may be an array that is read from or written to an ordinary file.

To make your program more visually pleasing, you may use ANSI escape sequences that are supported by ANSI and VT100 (and other) emulations. Two of the more useful ANSI escape sequences are:

- `ESC[2J` to clear the screen; and
- `ESC[y;xH` to move the cursor to position `(x,y)` on the screen;

Here `ESC` is ASCII character 27_{10}; `x` and `y` are numbers in string format. If `screen` is a suitable Java stream object, then

```
  screen.write( 27 );
  screen.print( "[20;5HHello" );
```

will display the message `Hello` in line 5 from position 20 on the screen. Experiment to ensure that you understand the concept before you write your program.

(Naturally the best solution is to write your own class and methods to hide this level of detail from the rest of your program.)

Remember that your program is a server that is to be used via the network. After you have activated the server, all interaction with the server has to occur via Telnet: during development it may be a good idea to build a server that outputs debugging information in the server window; however, once it is demonstrated, the server will not output any values on its window or display.

2.3 Assessment

A working program (that uses screen control) will earn 8 out of 10. To earn higher marks your program will be expected to do more than just the basics, such as to allow more than one user to use your program simultaneously or to simply use colour. However, since colour has now been mentioned, it is no longer an original idea and won't earn many additional marks.

Chapter 3

Practical assignment 3

3.1 Background

The HTTP protocol is used to transfer packets between a web server and a web client (or browser, such as *Chrome*, *FireFox* or *Safari*). Normally these packets contain data encoded in HTML.

The web client marks all the links in a web page by underlining it and/or displaying it in a different colour. When one clicks on such a link, an HTTP (or HTTPS) request is sent to the server concerned — usually to fetch the page identified in the HTML. The format of such an HTTP request to fetch, for example, `/index.htm` from `www.cs.up.ac.za`, is as follows:

```
GET /index.htm HTTP/1.1
Host: www.cs.up.ac.za
```

Each line is followed by a CR character (ASCII 13_{10}).

When the server receives a `GET` request it simply sends the required data to the client as a stream of characters. Normally the stream of data will contain HTML encoded data that will be interpreted by the web client.

Note that the server may also receive requests other than `GET`; technically it is supposed to respond at least to the `GET` and `HEAD` requests. (The latter request is not considered further in this assignment.)

The description of HTTP given above has obviously been oversimplified, but it is sufficient for this exercise. Those who require more information may consult RFC 2616. (It consists of 176 pages in the text version, so do not procrastinate.)

Web pages are constructed by marking content up using HTML. To create a link, one uses the `<a>` tag:
`<a href=...> ...  </a>`
The text between the `<a>` and `</a>` tags is displayed as the link text. The portion

that follows `href=` is used to construct the `GET` message that the browser will send to the web server. As noted, it will often be the name of a file that should be fetched. It may also be the name of a CGI program that is to be executed.

However, it is possible to construct a special-purpose web server that interprets this parameter very differently. Let us use an example to illustrate this idea. Suppose one builds a server that accepts 'normal' HTTP requests that look as follows:

```
GET <string> HTTP/1.1
...
...
<blank line>
```

When it receives the request, it may check that it starts with the string `GET`. Then it extracts the `string`; let's call that string s. It may interpret and/or ignore anything that follows, up to the blank line, which demarcates the end of the header (and therefore also the end of the `GET` request).

It now considers the string s. If it happens to be '`/`' (typically meaning the root), it 'prints' the following to the standard output:

```
<some appropriate header line>
<some further appropriate header lines>
<blank line>
<html>
<body>
<p>X</p><br>
<p>
<a href="L">Left</a>
<a href="R">Right</a>
<a href="U">Up</a>
<a href="D">Down</a>
<a href="/">Home</a>
</p>
</body>
```

The receiving web client will display an 'X' in its upper left corner, and the links *Left*, *Right*, *Up*, *Down* and *Home* in the next line.

Say the user clicks on *Down*, then the server will get the request
`GET D HTTP/1.1`
followed by other header lines. Our special-purpose web server interprets this as meaning that the 'X' should move down one line. It therefore returns the same HTML response as it did above, but this time a new line consisting of `<br>` is

inserted before the `<p>X</p><br>` line. In fact, every time the server sees a `GET D ...` it adds another `<br>` line to its output. Hence, every time the user clicks on *Down*, the 'X' will move down one line.

The actions when a user clicks *Up* are the opposite: The server removes one of these `<br>` lines, until none of these `<br>` lines remains, in which case the server just produces the same output it previously did.

Similarly, whenever the user clicks *Right*, another space is inserted before the 'X', and when the user clicks *Left* one space preceding the 'X' (if any) is removed from the output produced by the server.

In essence, this (silly) server just isolates the character s, and produces output that differs from its previous output in the appropriate manner.

The example illustrates that the content of a `GET` request may be interpreted in a non-standard manner to build a special-purpose HTTP server, that can be used via an ordinary browser.

There are some caveats that should be noted from the example above. A server must, at least, support `GET` and `HEAD` requests. How does our server deal with `HEAD`? What should our server do when it receives a `GET` request that is not followed by one of the five characters that have some meaning for it? Should it return a 404 error? (Note that the typical browser will attempt to send a `GET /favicon.ico HTTP/1.1` request in addition to the request it is expected to send. Note that the addition of spaces in front of the 'X' in our example may be interpreted as whitespace by the browser, and the 'X' may not move to the right, as one would hope in this example. And finally, the browser may need to be coaxed to clear its content and replace it with the new content. Despite these issues, the example hopefully serves its intended purpose.

3.2 Your assignment

Write a program that emulates a world clock: Normally, it will only display the local time (in South Africa), with a list of major world cities. The program itself should, however, not do its own displaying, but operate as a sever that uses a browser to display all its interaction. When one clicks on the name of a city, the current time in that city, together with the current time in South Africa (as well as the list of other cities on which one may click) should be displayed. See the `<meta http-equiv="REFRESH">` HTML tag to determine how your program will be able to continuously display the current time.

The (simplified) process will therefore be as follows: One should select a port — say 55555. Then one should activate the server that listens to this port. After that, one activates *Chrome* or *FireFox* and enters `http://127.0.0.1:55555` as the URL — if the server executes on the local computer. This will cause a `GET` to be sent to the server which should then respond with the appropriate data — in HTML format. Each digit or operator should be a link and each click on such a link will then send a `GET` to the server, which should respond appropriately.

3.3 Assessment

A working program will be awarded 8 out of 10. To earn a higher mark your program has to do more than just the basics - in particular should it demonstrate that you understand something of the HTTP RFC.

Chapter 4

Practical assignment 4

4.1 Background

HTML makes provision for forms that enable one to enter data into fields and then send the data to the server. Consider the following HTML:

```
<form method="get" action="http://www.cs.up.ac.za/">
Number: <input type="text"  name="n" size="20">
<input type="submit" value="Do it">
</form>
```

This will display a form on the screen with a field in which one may enter a number. When one clicks on the 'Do it' button the specified action will be performed; in this case the home page for Computer Science will simply be loaded. As has been explained in practical assignment 3, a `GET` request will be sent to the server concerned (`www.cs.up.ac.za` in this case). Since there is a data field, the content specification (`/` in the `GET` request will be followed by a question mark, which will in turn be followed by the name of the field (`n`), an equals sign and then the value that has been entered by the user. If the user, for example, enters 33 and clicks on 'Do it' the following `GET` request will be sent to the server:

```
GET /?n=33
Host: www.cs.up.ac.za
```

Do your own experiments to see how forms with more than one field work. For more information search the web using your favourite search engine; keep your eyes open for tutorials. Also look again at RFC 2616 if necessary. Those who want to do more, note the `POST` method as an alternative for `GET`.

4.2 Your assignment

Write (another) program that keeps track of appointments with all the usual operations such as insertion, searching and deletion. In this case, however, your program has to be a server that should be used via a standard browser (such as *Chrome*; except for the initial activation there must be no direct interaction with the server via the command line or console).

As a challenge you may try to not only store times and appointment descriptors, but also a picture of the person with whom you intend to meet (when you have such a picture and when it may be appropriate). Note that, while it is possible to direct a browser to fetch a picture from the client machine, the real challenge is to transmit the picture from the server to the browser using HTTP. This involves encoding and serving appropriate headers; these are the reasons why serving a picture may warrant an additional mark or two.

4.3 Assessment

A working program will be awarded 8 out of 10. To earn a higher mark your program has to do more than just the basics - in particular should it demonstrate that you understand something of the HTTP RFC.

Chapter 5

Practical assignment 5

5.1 Background

LDAP is a protocol that provides access to a distributed directory service. It is standardised by RFC 4511. RFC 4510 provides a road map of a number of standards that may be relevant for this assignment (and indicates where RFC 4511 fits into the bigger picture). The Wikipedia entry on LDAP provides a nice overview of the protocol and its use; it may assist you to navigate RFC 4511 (and other relevant standards).

At an early stage in your preparation for this assignment it is important to understand how an LDAP tree (a *Directory Information Tree* is structured. Sections 2.1 to 2.3 of RFC 4512 (LDAP Models) should provide you with all the essential details.

In order to gain some experience with the use of LDAP install `OpenLDAP` on your server computer. Although one should almost always use the most secure options when configuring LDAP, for this assignment it is acceptable to send passwords as cleartext and not to use SSL/TLS. Without encryption, sniffing traffic is possible and much can be learnt from such sniffing.

It is possible to access the LDAP tree using the commands (on the command line) that are installed along with `OpenLDAP`. However, rather install `phpLDAPadmin` (often abbreviated as PLA) as well. It provides one with a web-based interface through which one is able to add, delete and modify LDAP entries.

Note that PLA executes on the server and you will not learn much about the operation of LDAP by sniffing traffic between your browser and PLA. Many LDAP clients exist that use the LDAP protocol. You are encouraged to find such a client and sniff traffic between it and your server.

Note that the protocols required to complete this assignment use different approaches to specify the representation of messages. You will encounter (at least)

ASN.1 and Augmented Backus-Naur Form (ABNF). Use this as a learning opportunity.

Also note that, where previous assignments used protocols where requests and responses were sent as plain text, using LDAP will often require you to send messages encoded in binary.

5.2 Your assignment

While LDAP is intended to serve as a directory of people, it may also be used to store information about resources. For this assignment, we will go beyond the usual application of LDAP and store information about our assets.

Suppose that you classify these assets into "organisational units" (OUs) that you call Planes, Trains, and Automobiles. For this assignment we are only interested in one OU — decide which one you prefer. (Note that this means your assignment will deal with Plains *or* Trains *or* Automobiles) For any asset we may (in principle) want to store its current location, cost price, next service date and value. For this assignment we will only consider one value: maximum speed in km/h.

Create an appropriate Directory Information Tree on your server (containing the value of the specific asset and its maximum speed) using PLA. Populate the tree with some entries.

Now write a client that will ask you for an asset name, query the LDAP database, and display the telephone maximum speed (if the asset exists). Note that your client should establish a connection with port 389 on your server, formulate and write all requests to the socket, and read and interpret all responses from the server. As always you are not allowed to use library functions or any other mechanism that handles communication between the client and server on a more abstract level. To be more specific, your client should construct the query packet and write it to the appropriate socket using byte or string level operations. Similarly, your client should read a response packet from the appropriate port, unpack it using string or byte array operations, and process the unpacked message as appropriate.

5.3 Assessment

A working program will be awarded 8 out of 10. To earn a higher mark your program has to do more than just the basics - in particular should it demonstrate that you have some deeper knowledge of the protocols as they are defined in the various LDAP-related RFCs.

Chapter 6

Practical assignment 6

6.1 Background

Many personal computers receive e-mail using the POP3 protocol *(Post Office Protocol 3)* and send mail by using the SMTP protocol *(Simple Mail Transfer Protocol)*

The SMTP protocol (RFC 821) is a rather old protocol (it dates from 1982) but is still commonly used — in some cases with a few extensions. The basic operation of the protocol is explained in your textbook, but you will have to read parts of the RFC to understand the operation better. Try to grasp at least the following commands:

```
HELO
MAIL
RCPT
DATA
QUIT
```

The SMTP server usually listens to TCP port 25.

Note that SMTP servers often place restrictions on who may use it to send mail or to whom mail may be sent via it. If neither the sender, nor the recipient has anything to do with the server one often gets the message

```
We do not relay messages
```

This is to prevent cowards who want to send junk mail via another organisation's computer from hiding their identities. If you use a connection provided by an ISP, you may be able to to use the SMTP server of your service provider. If you have a Gmail account, you may be able to use Google's mail server (depending on settings in your account). However, more and more email relays insist on using encrypted communications, as well as require one to log on. (This applies to Google's mail relays.) Implementing encryption (or even the need to sign in) pushes this assignment beyond the amount of work expected for this assignment. The recommended approach is therefore to set up your own email server on a system such as the one described in Chapter B. Note that such servers often default to installations that also prevent them from relaying email. You will have to read your email server's documentation and/or search the various Internet forums to determine how to enable email relay. Be careful that you do not host an open relay on the Internet, though. However, in the typical setup you will use, it is more likely that your email server will not talk to other email relays and you will only be able to send email on your local server, or local network. This is fine for this assignment — you do not need to be able to communicate with other email servers on the Internet to complete this assignment. There are many simple email clients for Linux and Unix that will enable you to read email on your server. This assignment is about sending email, and you have to write the software to send the email; you are not allowed to use an existing email client for sending the email as specified in this assignment.

Mail that is transferred via SMTP may, in principle, consist of any text. (See RFC 821 for more details.) However, if you want to use headings (such as *Subject*), look at RFC 561 (from 1973!). (These old standards often refer to protocols that no one knows anymore — just ignore such references.)

6.2 Your assignment

In the rat race one too often forgets the birthdays of friends and other important events. Luckily the computer can help us in this regard. If one has a program that runs once a day, scans a list of friends and reminds one of birthdays in the next week, one at least has a chance.

For this assignment you are expected to write such a program. It should read a text file with the format

```
dd/mm name or event
```

`dd` is the day (eg '21' or '07' or simply '7', without the quotation marks) and `mm` (similarly) the month. The `name or event` is a character string that describes the event.

If any of the events in the file occur exactly six days from the date on which the program is executed, the program should send you an e-mail listing all the events that occur six days later. If nothing is found that you should be reminded of, the program should simply terminate.

It is obviously possible to send yourself an SMS or other instant message on the day of the event (if you know how to do this). (For such functionality — and *only* for such functionality — you are allowed to use a pre-existing library.

For this assignment we will not schedule the program to automatically run once a day. We will rather execute it manually whenever we want to demonstrate its functionality. This may mean that the same reminder will be sent whenever the program is executed more than once on a day. You have to be able to show that the message has indeed reached the target mailbox.

6.3 Assessment

A working program will be awarded 8 out of 10. To earn a higher mark your program has to do more than just the basics - in particular should it demonstrate that you understand something of the SMTP or emails representation RFCs.

Chapter 7

Practical assignment 7

7.1 Background

POP3 (*Post Office Protocol 3*, RFC 1939) is used to download e-mail waiting at a server for a specific user.

You can (amongst others) use POP3 to download your e-mail waiting on your Gmail account (if you have one and POP3 is enabled for it). Just search the web for the appropriate port to use. (Again, encryption for POP3 is a *very* good idea, but being forced to implement it complicates, this assignment beyond what we expect you to do to complete this assignment. Hence, using your own POP3 server, where you do not enforce encryption may be you best option. Refer to Chapter B.

7.2 Your assignment

Society has become so used to immediate communication that people are often frustrated if you do not reply to e-mail immediately — even if you are on holiday. To handle this problem many people are using a 'vacation' program whenever they are away from the office to inform senders of e-mail of this fact.

Write a program that runs on your computer and that checks your mailbox every once in a while for *new* mail using POP3 and then informs the sender (using SMTP) that you are away.

Obviously, there are a number of pitfalls that you should avoid: It is sufficient to inform any given person only once that you are on vacation. (Think of the chaos that will ensue if your vacation program starts talking to that of someone else…) You may be subscribed to mailing lists and it is not necessary to inform everyone on the list that you are on vacation when the list arrives in your mailbox. Your program should also not delete the mail that it checks because you may still want to read it when you return from vacation.

It would also be a nuisance if your program informed everyone sending mail to you that you are on vacation while you are developing or demonstrating your program. Therefore you have to insert a check that will cause the program only to respond to messages that have a subject of `prac7`.

****

Some e-mail programs do not allow one *not* to download those messages that you are not interested in. In some cases such messages are very long and one spends a lot in phone costs to download them to then simply delete them. It may therefore be handy to have a program that shows one which mail messages are available so that one can delete those messages that are not required before you use your regular e-mail program to download the rest.

Write such a program that displays the sender, subject and size of all the messages waiting in your mailbox. Now make it possible to mark all those messages that you are not interested in (using, for example, a checkbox, but simpler approaches are acceptable — the user interface is not a prime concern in a network module). Your program then has to remove all those messages from the server without downloading any messages.

7.3 Assessment

A working program will be awarded 8 out of 10. To earn a higher mark your program has to do more than just the basics — in particular should it demonstrate that you understand something of the RFC.

Chapter 8

Practical assignment 8

8.1 Background

The transfer of content to a server is often most easily accomplished through FTP. Recall that the FTP daemon needs to be installed on the server computer and that an FTP client is then used on the user's workstation. Note that the commands entered by the user in a typical FTP client often differ from the actual commands that are sent by the software to the daemon. Recall that the FTP protocol acts somewhat strange: When an FTP connection is established, a control connection is established from the client to the server, as one would expect. However, whenever any data is to be transferred, the client opens a server socket, to which the daemon connects (as a client) to establish the data connection. This data connection is used for the actual transfer of the data.

Many programs have FTP clients built into them: When a security camera 'notices' movement, it may take a picture and use its built-in FTP client to upload the image to some server; even if the camera is stolen or broken, the image is safely recorded on some remote server. Web development software also often has the ability to "push" a newly developed site to a web server via its built-in client.

8.2 Your assignment

One of the worst modern-day nightmares is the scenario where you have been working on an important set of files and (typically just before some deadline) the disc on which you have been working crashes.

One solution for this dilemma is a program that 'watches' your important files and copies them to some server whenever they change. For this assignment we will assume that all the important files are stored in a single directory on your workstation or laptop. The program that you are expected to write, will monitor this directory and copy any file that changes to a backup directory on your virtual server.

Ensure that you install and activate a suitable FTP dæmon on your virtual computer (that is able to write to the selected destination directory on the virtual computer)

As indicated, you are expected to *write* a program that monitors files in a specific directory on your workstation. Whenever any file in the directory changes (such as when it is edited) your program should notice the change and use the FTP protocol to send the updated version to your server computer.

One of the subproblems you will have to solve is how to determine whether a file has been modified. In the simplest case, your program can record the datestamp of all programs in the directory and it can then occasionally check the datestamps of the existing files against the stored datestamps. If a difference is noted, the file can be uploaded, and the new datestamp recorded in the list of datestamps. Better options may exist.

Another subproblem to consider is whether new files will be automatically added to your list. (This seems obvious, does it not?)

A less obvious problem is how to solve deletion. If you accidentally delete an important file it will be really sad if your backup program also promptly deletes your backup file... On the other hand, the backup directory may fill up with junk if the unnecessary files that you delete on your workstation are never deleted from the backup set... (At this stage of the discussion, we may be leaving the 8/10 scenario if you provide a meaningful solution.)

Almost as bad as accidentally deleting a file is deleting its contents and saving it, after which your mirroring program will copy this new empty file over the file that had the important details in it. One possible solution for the backup program is to append a sequence number to the backup file. When it first observes a file `x.txt` in the important directory, it backs it up as `x.txt.000` (or `x.000.txt`?). When `x.txt` changes locally, it writes the new version as `x.txt.001` (or `x.001.txt`). It is then possible to retrieve a rather old copy of the file if one makes some mistake and does not notice it for a while. (Again, this part of the discussion is beyond the 8/10 scenario.)

For the sake of simplicity, you program will be tested using `txt` files that are edited.

As always, your program is not allowed to use any pre-existing FTP client functionality: it has to open socket(s) and write FTP protocol commands and other values to the socket(s) and read responses from the socket(s). .

You may use polling to test whether the file in question has been updated on the workstation. If you poll, say, once a minute and two updates within a minute are not both uploaded, you will not be penalised.

8.3 Assessment

A working program will be awarded 8 out of 10. To earn a higher mark your program has to do more than just the basics — in particular should it demonstrate that you have some deeper knowledge of the FTP protocol.

Chapter 9

Practical assignment 9

9.1 Background

Firewalls are usually deemed to operate on layer 3 or layer 7.

9.1.1 Layer 3 firewalls

A layer 3 firewall inspects network layer packets that flow through it. The firewall is typically installed on the host(s) through which a corporate or similar network is connected to the outside world. It is therefore in a position to monitor traffic and block certain traffic from flowing into the corporate network or leaving the corporate network. (In what follows, we will talk about a *corporate* network, though the principles apply in exactly the same manner to other networks, such school, home, and NGO networks. In fact, the principles also apply to subnetworks within such an entity, such as a department or even a single computer in the bigger organisation, if a firewall is installed for the subunit.)

From the layer 3 headers it can determine the source and destination addresses of the packets. The rules associated with the firewall may determine that the communication may be allowed or should be blocked. If it is to be allowed, the packet is forwarded to the next hop on its route. If the message is to be blocked, the packet may simply be dropped. As a simple example, if it is known that crackers operate from some known addresses, any traffic to and from those addresses can be blocked. Note that this example is unrealistic.

To properly use a layer 3 firewall the knowledge that the (immediate) payload of a layer 3 packet is a layer 4 packet helps. The first information that follows the layer 3 header in the layer 3 packet is, in principle, the header of the layer 4 packet it contains. Below we will consider why this is not always true, but until then we will proceed under the assumption that it is indeed always true.

The layer 3 firewall is therefore not only able to consider the layer 3 header, but also able to inspect the layer 4 header. This means it can, amongst others, also determine the source and destination ports of the packets. If the layer 4 content contains a TCP header, the firewall can also determine from the SYN, FIN and ACK flags determine the connection status of the TCP stream.

Now it becomes possible to restrict, say, web access to only be directed at the corporate web server: if the destination address is one that belongs to the corporate network and the destination port is 80, only allow the packet to proceed if the destination address is that of the corporate web server. In a similar manner, connections to other services may be restricted to the appropriate servers. At the very least the sysadmin can now ensure that the servers are properly configured and patched against the latest vulnerabilities for the services they provide.

A challenge remains: Someone in the corporation may operate, say, an unauthorised web server on some non-standard port on the corporate network. But the firewall is also of use in this case: Add a rule that blocks any connection to be opened from outside the organisation to a computer on the corporate network by blocking the TCP handshake (that is, any message for which the SYN flag is set, the ACK flag is reset, and the destination address is somewhere on the corporate network). It should be obvious that this will only be useful if this rule is only processed *after* rules allowing traffic to proceed to the appropriate servers have been processed. Stated differently, if a web request arrives and its destination is the corporate web server, ACCEPT it. Do the same for all other legitimate service requests. Then, as a default, REJECT all new requests to open a connection to any other hosts.

This still leaves us with some questions about those cases where the payload of a layer 3 packet is a UDP packet (or anything else, for that matter). We will not discuss those in this assignment.

We did promise to talk about those cases where the TCP header does not immediately follow the layer 3 header in the packet. This may happen where, say, a long TCP stream is transmitted, and it is split into several IP datagrams. The TCP header will follow the IP header in the first IP packet, but not in the subsequent IP packets. Fortunately, this problem is easy to solve — if one blocks the TCP header, the TCP stream can never connect, and the remaining part of the TCP stream is useless. Hence, blocking the TCP header is sufficient. When one explores this a bit deeper, there are all sorts of interesting games crackers play, and security specialists needs to be aware of, but we will not explore these in the current assignment.

9.1.2 Layer 7 firewalls

A layer 3 firewall provides one with many options to protect the corporate network from the big bad Internet out there, and even from corporate insiders to access services on the outside that are not deemed to be in the corporate interest. However, layer 3 is too far removed from the application layer to provide complete protection. By inspecting the ports, the layer 3 firewall knows which service (or application) is to be accessed, but further details are hidden in too many layers of payload — one cannot use the same trick we used to let a layer 3 firewall also access layer 4 headers to reach all the way to layer 7 details. Hence, layer 7 firewalls are typically deployed in addition to layer 3 firewalls. Layer 7 firewalls are often referred to by the term *application proxies*. Note that, despite the implication earlier in this paragraph that complete security may be achieved by increasing the 'reach' of firewalls, firewalls always offer only a small — albeit important — part of network security. One should carefully think about threats that are not addressed by *any* firewall and have a broad view of network security. This assignment only considers firewalls, though.

One of the best-known examples of a layer 7 firewall is a web proxy. Browsers make provision for one to enter details about the proxy. If a web proxy is used, browsing requests will be sent to the web proxy, rather than the destination host. The web proxy will then forward the request to the destination host and receive the response. It will then relay the response to the browser that sent the request initially. The web proxy is able to inspect the request sent by the browser and block it, if the corporate policy does not allow access to the specific website. One typically has to log into the web proxy, so unauthorised parties can be prevented from using the corporate network to surf the web. In addition, if users are only allowed a certain amount of web traffic per month, a tally of the traffic relayed can be kept and requests from that user can be blocked once the threshold is reached. As another example, it becomes simple to allow employees to access social networks over the lunch break, but block it at other times, when employees are supposed to be working.

Note that such an application proxy needs to be used in conjunction with a layer 3 firewall: The layer 3 firewall needs to block any outgoing traffic to port 80 (or other relevant ports), except when the traffic comes from the web proxy.

This introduction quickly leads to opinions about 'corporate censorship'; we will, however, not succumb to that temptation in this assignment. For this assignment, we accept that the corporate network belongs to the corporation, and they are free to formulate policies regarding the network and enforce them.

Note again that web browsers make provision for web proxies and the user is often unaware of the fact that a web proxy is in use. The web browser would normally send its request to the web server as specified by the URL entered into the browser. However, when a proxy is in use, that request will be sent to the proxy. The proxy will perform whatever inspection it is configured to perform, and then forward the request to the server — or send an error message to the web browser. The response from the server is similarly relayed to the browser by the proxy. For other applications, the ability to redirect messages to a proxy are typically not build into the client. If one wants to use a proxy, some non-standard process to relay the messages via an application proxy may be required.

9.2 Your assignment

For this assignment you are expected to build an application proxy. Note that the application proxy will act as both a server and a client: The usual client will connect to the proxy, as if the proxy were the server. This requires server functionality: It will have to wait on some specified port (such as 55555) to accept requests from 'normal' clients for that application. The application proxy will then perform whatever checks it needs to perform on the request. After completing the checks, it has to connect to the intended server, and act as a client to that server.

For this assignment you are expected to write an SMTP proxy. Your organisation has decided that all outgoing emails have to include a disclaimer at the bottom that reads: "Please do not take anything in this email seriously!"

Your organisation has also read George Orwell's *Nineteen Eighty-Four* and accepted *Newspeak* as a recommendation, rather than a warning. As a first step, they decided to eliminate the words *warm* and *bad* from usage, and resplace them by *uncold* and *ungood*, respectively. Your proxy is supposed to scan outgoing emails and replace words according to this policy. (However, be careful to not, for example, change *swarm* to *suncold*...)

The following changes in outgoing emails are also expected:

Old	**Substituted by**
fast	speedful
rapid	speedful
quick	speedful
slow	unspeedful
ran	runned
stole	stealed
better	gooder
best	goodest

While not required for the initial version, management would be very pleased (sorry, pluspleased), if you could make the following substitutions:

Old	Substituted by
very good	plusgood
very fast	plusfast
very bad	plusungood

Although not from George Orwell's book, management decided that any email that uses the word *Illuminati* should not be sent. It should be replaces by an email that simply says "Hello world".

In order to get this you work, you will select a client that can send emails. Configure it not to use TLS/SSL. Configure it to use, say, port 55555 on your virtual computer as its SMTP port. Install SMTP and POP3 servers on your vrtual server, and use the 'normal' port to connect your email client to the POP3 server.

Install the proxy server that you have written to accept incomping traffic at the appropriate port (55555, if you used 55555 as suggested above). This proxy should be developed such that it can talk to the SMTP port of the installed email server.

The proxy would normally simply copy whatever it receives from the email client to the real SMTP server and copy whatever is received from the email server to the email client. The exceptions are the following:

- After the `DATA` command has been forwarded to the SMTP server, the proxy will examine all strings being forwarded. If it notices any ungood words (those in the lists above), it would substitute the ungood word by its good equivalent.
- When the proxy sees the full stop on a line of its own (in other words, the end of the message), it first send the prescribed disclaimer to the server. Then it send the full stop to terminate the message.

In addition, the proxy should monitor communication and take appropriate action when necessary (if needed ever) when error conditions occur.

To simplify this task, configure the email client to only send emails as ASCII text. Alternatively, consider it a challenge to handle messages that may be encoded in various MIME representations. (Consider `base64` as one example.)

9.3 Assessment

This project counts 20 marks. Writing a working proxy server as set out in section 9.2 will earn you a mark of 12. Solving one or more of the challenges will push your mark up to 20. However, note that fewer marks may be awarded than specified above if your solutions are not completely correct.

Chapter 10

Practical schedule

The practical schedule will be posted on clickUP.

The first assignment will be due on the first Monday, two weeks after this guide has been released online. Subsequent assignments will be due approximately every week after that. However, weeks in which semester tests are written, or where public holidays impact on available time will be taken into account.