Chapter 1

Practical assignment 1

Note that all assignments consist of three sections: One that provides relevant background, another that provides the assignment itself and, finally, some remarks about assessment. Ensure that your submission addresses the issues discussed in the *assignment* part.

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 allowed to use CGI programs in any 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 of a 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.)

While you may use any programming language to develop your CGI program, ‘conventional’ languages such as Java, C++, Pascal or COBOL are preferred. You do not need any network functionality in the programming language; you just need the ability to write text to the screen. For that reason, even a language such as COBOL would work. (COBOL is not recommended for subsequent assignments, though.)

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.

1.2 Your assignment

As you are probably aware, a Fibonacci sequence is a sequence of numbers such that $f_n = f_{n-1} + f_{n-2}$ with $f_0 = 0$ and $f_1 = 1$.

For this assignment you are expected to write two CGI programs. (They will be so similar that it may be easier to write the one and then create the second from a copy.)

One of the programs will read three numbers from a text file. Those three values are initialised by you, the programmer, to be Fibonacci numbers. When the program runs, it reads the three numbers in the file, adds the second and the third number to produce a new number in the Fibonacci sequence, overwrites the numbers in the file with the second, third and new number and also displays these three new numbers. In essence, this program is a 'next' program in the sense that it displays the next set of three Fibonacci numbers. When it is executed the first time, it may display 0, 1, 1 (because $f_0 = 0$, $f_1 = 1$ and $f_2 = 1$). If one executes the program n times in sequence the values of f_{n-1}, f_n and f_{n-1} would be displayed. (Given this behaviour the file should obviously not be initialised to 0, 1, 1, because the first execution would then display 1, 1, 2, which is not what is expected.)

When we say that the program *displays* the three numbers, we mean that the program outputs text that forms valid HTML that would be rendered by a browser as a webpage with these three numbers. Additionally, the webpage should contain two hyperlinks: One would display the word 'Next' and be linked to this program that has been described thus far. When the program is installed in the correct directory and first invoked (possibly from a static initial webpage), it would cause 0, 1, 1 to be displayed in the browser window. Successively clicking 'Next' would (each time) cause the program to display the next Fibonacci triple.

[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!

There may be one tiny catch. If your browser caches the output from your CGI program it may think that it already knows the page that should be displayed, without executing the CGI program each time. Consider whether you need a metadata tag in your webpage (or some similar mechanism) that prevents such caching.

The other hyperlink on the webpage that your program generates should be called ‘Previous’ and be linked to the second CGI program you are expected to write.

This second CGI program is almost identical to the first program described above. However, when it reads (x, y, z) from the file, it overwrites the file with $(y-x, x, y)$ and displays this latter triple in the browser’s window. It also displays the two hyperlinks that the first program displays.

There are interesting boundary cases. If the file contains $(0, 1, 1)$ and ‘Previous is clicked, it may make sense to ‘go to’ $(---, 0, 1)$. Arguably, ‘Previous’ cannot be clicked again (and ideally should not even be listed as a clickable link). Does it make sense to consider some boundary case at the high end, where ‘Next’ does not make sense anymore?

To create a webpage that literally consists of three tiny numbers and two tiny hyperlinks would meet the requirements, but little effort is required to make it a bit more aesthetically pleasing. However, always remember that this module is about networking and aesthetics will not count much — if anything at all. However, a display that is really bland may or may not be commensurate with your self-image. That being said, spending more than about 10 minutes on aesthetics would be a waste.

A more meaningful question would be the following: Does one really need two programs when they are so similar? It would be easier to maintain one program that accepts a parameter. Given, say, a + moves it forward and a - moves it backwards.

One simple solution would be to write this program that is, say, called `fib`. Then create two scripts called `next` and `prev`.

`next` may look at follows:

```
!bash
fib +
```

The `prev` script looks almost identical, with just one character changed. Once these scripts are placed in the CGI directory, they should be made executable (with `chmod`) that that the web server is authorised to execute it.

It seems an even better solution would be to simply make the parameter part of the hyperlink linked to the `fib` program. This is indeed the case. However, the challenge is: how does one obtain the parameter from within the `fib` program? Yes, there are library functions that enable you to extract such parameters. In many scripting languages the ability to retrieve parameters is built in. However, **recall the prohibition of using any functions or built-in features to achieve network functionality** that applies to this module.

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.

Warning

Using any functions or built-in features in this module to achieve network functionality will lead to a mark of 0 for the assignment. The only exception is using functions to open and close network sockets.

Warning

This assignment deals with server-side processing. **Using client-side computation (for example, using JavaScript) would lead to a mark of 0 for this assignment (and will be heavily penalised in any other assignment in this module).**

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 connection 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

Oine of the more onerous tasks of a lecturer is the compilation of tests, but a computer can be used to ease this task. Assume a file exists that contains questions in the following format:

```
?The successor of IPv4 is:
-TCP
-Ethernet
-IPv5
+IPv6
-None of the above
?DNS normally uses the following transport layer protocol
-TCP
+UDP
-IEEE 802.3
-All of the above
...
```

In such a file

? indicates that the remainder of the line contains a question;

- indicates that the remainder of the line contains a (wrong) alternative answer for the preceding question, while

`+` indicates that the remainder of the line contains the correct answer.

Write a program that reads such a file and then waits on port 55555 for a Telnet connection. When the connection is made, the program randomly selects one of the questions and displays it on the virtual terminal. The different alternative answers are displayed underneath, each with a suitable letter to identify the alternative. (Obviously the program should not give any indication of which answer is the correct one.) After this, the program, again using the virtual terminal, asks the user to enter the correct answer. If the entered answer is correct, the user should be congratulated; if not, the correct answer should be displayed.

Next the user should be asked whether the program should ask another question. Depending on the user's answer, the program should ask the next question or display the user's score.

If a question contains no correct (+) answer, the server should automatically add a None of the above following the other options, and assume that it is the correct answer. If a question has more than one correct (+) answer, the program should automatically add More than one of the above following the other options and assume that it is the correct answer.

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.

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 repeats the functionality of assignment 1: It should again display Fibonacci triples. However, rather than implementing the functionality using an existing server and CGI, the software that you write for this assignment should be a dedicated 'Fibonacci server'. Hence it would open a server socket (say 55555) and wait for connections. Once a browser (such as Firefox) connects, your program should interpret the `GET` request that the browser sends. The response that your server sends should not only be the HTML that your scripts sent in assignment 1, but also the appropriate HTTP headers, such that the browser is able to interpret the HTTP message correctly (and render the HTML correctly).

Implement all the commands that an HTTP server *must* implement.

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

Modify the program that you wrote for assignment 2 so that it can be used with a browser (rather than Telnet).

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 other information. In this assignment we will use LDAP in a rather atypical manner. A typical directory may use countries as the first nodes in the directory tree. Organisations within those countries may be placed at the next level. Departments (or 'organisation units') within each company may be at the next level. Finally, details of employees working in such a department (such as their names, work addresses and telephone numbers) are stored as the leaves. It is , however, not necessary to implement the tree using all the levels. A company that wants to create a directory may, for example, deem the company to be at the root, and put departments (and with departmental information) one link lower in the tree.

Similar to the directory described above, the DNS forms a tree, starting at the root nodes.

Now suppose, just because we are able to do it, we want to store DNS information about the organisations within the `za.` TLD in LDAP. (This is an educational exercise — do not think too hard about the logic of what we are doing.)

Hence, the root will be the TLD. Our 'organisational units' in the tree will be the second-level domains, such as `ac`, `co` and `gov`. The organisations about which we want to store details will be one level lower — where details if individuals would fit more naturally.

The information we want to record about our (individual) organisations is the information stored on the third layer of the DNS tree about them. To use `up.ac.za` as an example, the following information is relevant:

```
C:\>nslookup

> set type=any
> up.ac.za

Non-authoritative answer:
up.ac.za
    primary name server = ns2.up.ac.za
    responsible mail addr = dnsadmin.up.ac.za
    serial  = 3991808205
    refresh = 10800 (3 hours)
    retry   = 3600 (1 hour)
    expire  = 1728000 (20 days)
    default TTL = 900 (15 mins)
up.ac.za    internet address = 137.215.10.220
up.ac.za    MX preference = 1, mail exchanger = aspmx.l.google.com
up.ac.za    MX preference = 10, mail exchanger = alt3.aspmx.l.google.com
up.ac.za    MX preference = 5, mail exchanger = alt2.aspmx.l.google.com
up.ac.za    MX preference = 10, mail exchanger = alt4.aspmx.l.google.com
up.ac.za    nameserver = ns2.up.ac.za
up.ac.za    nameserver = ns3.up.ac.za
up.ac.za    nameserver = ns1.up.ac.za
>
```

(Not all the information returned by this query is included above.)

The information provided clearly derives from SOA, A, MX and NS resource records in the authoritative zone file. For the assignment you only have to include information that pertain to A, NS and MX records. One would expect a single IPv4 address (if any). Record only the MX record with the highest priority (lowest number) — if MX records are present. Use any one NS record — if NS records are present.

Create the Directory Information Tree on your server and populate it with three or four second-level domains. Pick any that you like. Pick any of these second-level domains and add information about three to four organisations at the appropriate places in the tree. Use the pre-existing tools discussed above to populate this (tiny) part of the tree.

Now write a client that will ask you for an organisation name, query the LDAP database, and display the resource records (if the organisation exists in your tree).

If `up.ac.za` is in your tree, are you able to query it using just `up`? Are you able to query it using `up.ac.za`? The underlying question is the following: If Jane Doe works in the design department of ACME, will we (where LDAP is used for its intended purpose) be able to search for Jane Doe, Jane Doe at ACME, the employees at ACME, or any other such complete or incomplete information.

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 mechanisms 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 previous assignments you created a system that used multiple-choice questions to test a person's subject knowledge. Modify that such that it emails the test-taker's results to an appropriate email address.

If you use assignment 4 as the basis, ending the test on the web browser may offer a button that may be clicked to email the results. You may also use the Telnet-based solution and somehow add a command (at the end of testing) to email the results in a similar manner. (Obviously you have to make provision for entering your email address so that the results will be sent to the correct recipient.)

You have to be able to show that the message indeed reaches the target mailbox.

As usual, remember to implement sending email by using native SMTP commands rather than any pre-existing mailing functionality.

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

In some cases an email sender sends a blind copy to some party whom they want to share the information with, but they do not want to let the indicated recipients know that this party was included as a sender. If you receive such a blind copy, you may accidentally reply to all — and suddenly the other recipients of the email would know that you were included as a recipient of the original email. This may cause embarrassment (or worse) for the original sender who included you as a BCC recipient.

Hence a program that 'warns' you when you received an email as a blind copy would be useful. Let's think about how such a tool may work. Suppose your tool monitors your email inbox using POP3. It downloads every email (or, if possible, enough lines from the top of the email). It then searches for the first `BCC:` header (with some caveats). If the BCC header is found, only your address would be expected next to it. If your program finds an email with a `BCC:` header it sends a warning email to your email account, using a header such as

```
[BCC Warning] <original header>
```

Hopefully, when you open your inbox there will be a warning waiting for you above the email that you received as a blind carbon copied recipient — and you can avoid embarrassment.

Think about the logical point up to which you should search for a `BCC:` header. If any email includes the text `BCC:` it may cause a false alarm.

How would automatic forwarding of an BCC email present? What would happen when a mailing list is the BCC recipient of a message?

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

A computer may stop working if an important system file is (maliciously or accidentally) modified or deleted. Suppose one has a background process on such a computer that monitors all important system files. Whenever it determines that any one of these files are changed or missing, or simply downloads the correct version from an online repository and replaces the compromised file with a copy that is known to be good.

For this assignment you are expected to implement a prototype of such a system. It will monitor only one file (which should ideally be a text file that has simply been created for demonstration purposes). The known-good file is stored on a site (or 'server') with a working FTP client that you should obtain from any of the usual places to obtain such server software.

The program that you will write will monitor the 'protected' text file. Whenever loss or change is detected, your program will use the FTP protocol to download the known-good file from the server to ensure the integrity of the file.

To facilitate testing the known-good file on the server may, for example, contain the text 'Good'. If one deletes the protected file (er edits it to, for example, 'Bad') it should quickly be back in a state where it contains the text 'Good'. To demonstrate that the file is really downloaded (and not recreated by your program, you will be expected to edit the file on the server to something like 'Very good'. Tampering with the local protected file should then replace it with a file that contains the updated content (such as 'Very good'). This requirement poses some challenges of its own: How do you know when a file has been modified? There are several options. One is to compute some hash (such as MD5) over the file and compare the protected file with the known-good hash. However, changing the file on the server, changes the known-good hash. One solution is to store both the original file and its hash on the server. When the known-good file is updated, the hash is updated too. (If you decide to use a hash you are allowed to use a library function to calculate the hash; on the server you will probably simply use a built-in command to update the hash.) An alternative option may be to use the date of the protected file to determine whether it has been modified. This would require your program to either reset the date of the protected file to its 'original' date or to remember the last date it reset it. (To access and/or modify file creation dates you are — obviously — allowed to use built-in functionality and/or library functions).

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 local computer. If you poll, say, once a minute changes to the protected file may take a minute before it is corrected. This is expected behaviour.

One minor (?) issue to keep in mind is the following: Some editors lock a file while editing. Your program may notice that it has been changed and try to overwrite it with a correct version while it is still locked for writing. Your program (1) should not fail due to this sharing violation, and (2) update the protected file once it becomes writeable. Your program will probably handle requirement (2): Automatically: While the protected file differs from the original, it will (during each polling round) notice the difference, and try to replace it with the original.

This may continue until it succeeds. Hence, only requirement (1) needs your explicit consideration.

Ideally, your monitoring program will quietly run in the background and record any events in the system log. However, for demonstration purposes, it is simpler to run the program in its own window where it can ‘log’ events verbosely. During the demonstration it should therefore be obvious what is happening, without a need to inspect any log files.

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

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 a POP3 proxy. The logic (or that of your boss) is the following. The corporate customer email service is hosted somewhere in the cloud. Employees are allowed to read customer service emails only when they are in the office. Hence it has been decided that the password of the corporate service will not be given to any employees. They would rather be given usernames and passwords that are checked by a proxy server that is only accessible from the network in the office. This proxy server 'knows' the real credentials for the cloud email service. Once the user has been authenticated via the initial part of the POP3 conversation by the proxy server, the proxy server initiates a POP3 conversation with the cloud server to be authenticated using the real credentials. Once authenticated, the proxy server merely relays messages between the user and the cloud server back and forth, until the POP3 session is terminated.

Obviously, this 'solution' raises a number of questions about its effectiveness. Would it have any benefit if employees use laptops that they can take home? Perhaps only installed desktop computers are allowed to be used in the office... But users are then able to copy emails to memory stick that they can take home... What can be done to make this scheme effective? Does the idea have any merit at all? Please ponder these questions. However, as the organisation's programmer your main task is to implement the software as specified by management. So, start designing the program despite any concerns you may have.

To simplify matters the 'corporate POP3 server' does not have to be in the cloud. It may be any server that you can access via POP3. One option is to use a

POP3 server installed on the server computer you used for other assignments.

The specification suggested that multiple users, each with an own password, may be able to access the proxy server. For your assignment, a single used id with password (that both differ from the real access credentials) will suffice.

Ideally the proxy server should execute on its own server computer. However, use either your server computer or your client computer to host it.

You should use any mail client (such as Thunderbird) to read received email. This presents you with the challenge: To which SMTP server do you connect. A nice solution would be to also write an SMTP proxy, but that is not required for this assignment. You may therefore provide the details of the 'real customer service' SMTP server to your email client. *However, you have to provide the (different) POP3 proxy credentials to your client.*

To demonstrate that your proxy works, send emails to the 'corporate' email account, and show that your proxy can access the emails via the proxy using POP3.

Ideally such a proxy would print event messages to a log file. To simplify demonstration, let the proxy run in its own window and print verbose 'log' messages to the screen. This should enable an observer to quickly notice what is happening at any given moment — such as when the vatious connections are established and terminated, when commands are relayed (or blocked), etc. However, do not print entire email messages to this 'log', because such text would obscure rather than highlight events as the happen.

9.3 Challenges

Here are some challenges to increase your mark.

1. If you implement an appropriate SMTP proxy you may, as a special bonus, earn up to 2 additional marks!

2. There may be confidential emails that the 'normal' employees are not supposed to see. Assume that any email that contains the word 'Confidential' in the subject line should be hidden from users who access the mailbox via the proxy. Note that this challenge is rather hard. One way of simplifying it somewhat is to replace the confidential email with some cover email ('Just testing' from a fake email address [perhaps test@example.com] or a message selected from spam that some spam filter recently caught).

3. Insert a line 'Handled by <*username*> where *username* is the proxy username of the individual who downloaded the email. The intention is that, if a user, for example, forwards or prints the email, it may be possible to

determine which user forwarded or printed the email. (Obviously a prudent user will be able to delete such an insertion before forwarding or printing.) However, just seeing the inserted line is sufficient for a mark or two. The fact that the assignment only calls for one proxy username does not matter. Keep in mind that many emails are sent in both text and HTML formats (with `multipart/alternative` MIME header.)

4. Use more than one username on the proxy and only allow one user to DELEte emails.

5. Most importantly: use your own creativity.

9.4 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 increase your mark; with sufficient additional work, a maximum of 20 will be awarded. However, note that fewer marks may be awarded than specified above if your solutions are not completely correct.