Department of Computer Science
University of Pretoria

Computer Networks
COS 332

Study Guide (Practical Assignments)
Version 2025.01

2025

Contents

Important information for 2025

Programming language to use

In the past I did not care which programming language(s) students used to complete the projects described in this guide. However, there has always been a fundamental rule: Do not use library, or built-in functionality to perform the communication that these assignments require. This rule will be repeated later in this guide, but, given that so many students burn their fingers, let us also briefly mention it here: If you, for example, ever in your professional career have to write software that sends an email , you will use library, or built-in functionality to accomplish it. Our purpose is not to get the email out (if the assignment involves sending an email. Our purpose is to 'explicitly talk the protocol that gets an email sent'. It starts by opening a socket, waiting for the email server to greet you, telling the email server who the recipients are, waiting for the server to state that it will accept mail to those recipients, and so on. Clearly, if your environment has a function `sendEmail`, that accepts the destination email address, email header and email body, and then does all the work to actually send the email, then you have not done anything that that the assignment really called for. Using such a library or built-in functionality means that your assignment will not earn *any* marks.

The rule above has been emphasised over many years, but the odd student still commits this cardinal sin. However, as the popularity of scripting languages increased, more and more students fell into this trap. Often, it seems, the functionality was so easy to achieve that they did not think about it as significant built-in functionality (and it obviously did not come from a library that they had to import). In some cases they started out on the right track, couldn't get it to work and then, as a final resort, looked at help from the web, which obviously recommended use of the built-in functionality.

A 0 for an assignment spoils the day for both the student(s) and the person who marks their assignment. Let's try the following for 2025, with the hope that this leads to fewer cases of a 0 mark:

- **Before using any scripting language for these assignments in 2025, double check that you understand the requirements of the assignment about having to 'explicitly speak the protocol'.**
- **The use of Python to complete these assignments is prohibited in 2025.** This prohibition is not based on a dislike of Python, but because the vast majority of students who incorrectly used built-in functionality in 2023 and 2024, did so in Python.

Java is generally a good choice for these assignments. This is also true for many other programming languages.

Using an SBC for your practical assignments

Networking is about getting two (or more) computers to communicate. In years gone past, we were able to find servers on the Internet, and (at least) our clients spoke to distant computers. As computer security became more important, the available servers have decreased, firewalls block communication with those that exist, and those that exist often enforces the use of encryption that adds a layer of complexity to the software you have to write. The solutions later in this guide will mention a number of options that enable one to communicate on a single computer. However, that removes a lost of the inherent excitement of getting computers to really talk to one another.

The most realistic option to experience proper communication is to get a *nix computer, connect your regular computer (running MacOS, Windows or *nix) and connect the two with a UTP cable. Some of you will complete the assignment in pairs, and may have access to two computers. However, even in those cases, please consider the following option: If you do not have a tiny single board computer (SBC) get one and install *nix on it. Your cheapest option to buy such a computer new is probably the Raspberry Pi Zero W, which is currently available for R310. However, on it you will have to use WiFi (which complicates matters) or get a USB network adapter. If you plan on getting a USB network adapter, the Raspberry Pi Zero at about R250 is a cheaper option. USB to Ethernet adapters are widely available for less than R150. (Note that the Pi Zero provides a micro-USB slot and most USB to Ethernet adapters come with a USB A plug, so a conversion plug — and, possibly a USB hub — will most likely also be required.) To avoid having to obtain multiple components, a Raspberry Pi 1, 2, 3, 4 or 5 is a good option. Even the smallest with the least memory will suffice. There is only one 'model' of the Raspberry Pi 5. For the earlier versions, the 'model B' provides the network functionality you need. It would presumably be hard to find a Pi 1 or Pi 2. The Pi 3 is getting scarce, but (its model B Plus) sells at about R740. The Pi 4 starts at about R1000 and the top-of-the range Pi 5 now sells for just less than R2700.

The raspberry Pi does not come with a keyboard, mouse, screen or power supply. In most cases an old cell phone charger will be good enough to use as a power supply. You don't need a mouse, and old keyboards are usually easy to find. You will only need a screen while installing the operating system; from then on headless operation would be typical.

In addition, you will need a micro-SD card to use as a disc.

Note that it is very simple to share a Raspberry Pi. You remove the disc, by removing the SD card; a friend can then install an SD card, and have an entirely different computer to use.

The logic of strongly recommending that you use such an SBC is based on

two facts:

1. Networking is more fun when communication occurs between different computers;

2. Communication between two different computers is often easier to manage, than managing communication between a computer and a virtual host on the computer;

3. Using a computer in headless mode develops useful networking skills that we do not have time to address explicitly during the course.

I looked at the PiShop (`https://pishop.co.za`) to confirm most prices and stock levels while writing the above.

Note that `I highly recommend getting an SBC for the module, but do not require it.` You marks will in no way be affected by your use of an SBC or not. If you complete you assignments in pairs, only one SBC per pair is necessary.

Uploading MD5 digests

In 2025 you will be expected to upload an MD5 digest of your executable for each assignment in addition to uploading the usual files that constitute your submission. The *executable* is the output from the compiler (terminating with `.exe` on Windows, and usually without a suffix on *nix). In the case of Java, the byte code that is interpreted will be deemed to be the executable. For a 'pure' interpreted language, the source code is also the executable.

The intention is that you should be able to calculate the MD5 digest of your program at the time you demonstrate it, and the digest will then be compared with the value that you submitted online.

Chapter A

Some general requirements

Assignments need to be uploaded to clickUP 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). i**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 develop 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.)

One of the more exciting options is to use a Raspberry Pi as your server. Any Raspberry Pi would suffice for these assignments. In the case of the earlier versions of the Pi, it is better to get the B version — that is, a 1B, 2B, 3B or 4B — since these versions are, amongst others, intended to be used as general-purpose computers. The Pi 5 is only available as a model that is similar to the earlier B versions, so any Pi 5 will suffice.

The Pi 1 to 5 provides an Ethernet port, which dramatically simplifies connection between your client computer (such as your laptop) and your server (the Pi). Some versions of the Pi Zero include a WiFi interface, and with enough patience, it is not too hard to get your client computer to talk to the Pi Zero as a server. At the time of writing this, the Pi Zero W is available for about R310.

If you plan to buy a Pi, note that some shops sell Pi boards at exorbitant prices. As a general rule I recommend that you compare prices (and availability) of boards or kits that are of interest with offers on `pishop.co.za` — one of the four official distributors of the Pi in South Africa. As a computer science student the odds are that you will only need a board and a micro-SD card, since you probably have access to spare keyboards and other components. (For most of our work you won't even need a monitor for the Pi.) Note that typical cell phone chargers are sufficient to use as a PSU for most of the Pi computers. Note that you need a somewhat more powerful charger (2A) for the Pi 4B and an above-average one (3A) for the Pi 5.

The other authorised South African distributors are

- DigiKey (`https://www.digikey.co.za/`),

- Farnell (`https://export.farnell.com/`) and
- Mouser(`https://www.mouser.co.za/`),

where prices and stock levels are somewhat harder to determine.

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 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

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

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

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

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

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

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

or

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

Package the files (static HTML and executable CGI programs) in an archive that will install the files into the correct directory irrespective of the current directory from which the archive is unpacked. This requires you to use `tar`, `tgz` and/or `zip` to package the files using absolute paths. The correct absolute paths

[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!

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.

1.3 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 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' (such as an array read from and written to a text file) of friends and their telephone numbers. All the usual operations such as searching, addition, deletion, etcetera, must be available. After your server has been activated, *all* interaction with it has to occur through 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 emulates a calculator: It should display all ten digits, the four common operators and the answer thus far. The program should function as a special-purpose HTTP server and be used via a browser.

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 record of friends' telephone numbers 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).

As a challenge you may try to not only store names and numbers, but also photographs or other pictures. 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

Suppose you know quite a few people and you want to maintain a list of their telephone numbers. Suppose further that you classify these people into "organisational units" that you call Friends, Acquaintances, and (possibly) Enemies. For this assignment we are only interested in Friends.

Create an appropriate Directory Information Tree on your server (containing Friends' names and telephone numbers) using PLA. Populate the tree with some entries.

Now write a client that will ask you for a Friend name, query the LDAP database, and display the telephone number (if it 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

Assume your house has a permanent Internet connection. Then it is possible to link your alarm system such that it sends you e-mail whenever it becomes suspicious. (It is, after all, possible to send e-mail to a cell phone so that you are informed immediately when there is a problem.)

Write a program that simulates such an alarm system. Assume that the sensors are connected to keys on the keyboard and that movement or a door that opens causes contact to be made on the keyboard as if a specific key has been pressed. Your program then has to send a suitable e-mail message to a specific address (quite possibly of a user who has an account on your own server). You have to be able to show that the message has indeed reached the target mailbox. To read the email message you are allowed to use any suitable (existing) email client.

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

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

You are expected to set up the following software on your virtual server:

- The Apache server; and
- A suitable FTP dæmon (that is able to write to a directory served by Apache)

You are expected to *write* a program that monitors some HTML file on your workstation. Whenever this file 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.

To demonstrate your program, point some browser to the appropriate URL on your server computer, such that it will display the file in question. Then you edit the HTML file on your workstation. Once you save it, it should automatically be uploaded by the program you wrote. The browser (possibly after pressing 'refresh') will display the new version of the file.

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

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 Telnet proxy. Your organisation has decided that employees are allowed to access some *specific* Telnet server. However, they do not want employees to ever execute the `ps` command to see which processes are executing on that computer. (Note that this means that the server address may be hard-coded into the proxy, but see the challenges below.)

In practical terms, this means the following:

1. You have to find and install a Telnet server on your own server machine — see Chapter B again. You will not modify this Telnet server in any way.

2. You need access to a Telnet client. (If you do not have such access at this stage of the module, you have not been paying attention to the lectures...) You are not allowed to modify this Telnet client in any way.

3. Modifying the settings or configuration of the Telnet server and/or client is allowed.

4. You have to write the code for the Telnet proxy.

Let us assume the (standard) server is executing one some machine which we will refer to as S, where it listens on, say, port 23 for requests. Your client will execute on whatever machine you enter the `telnet` command; let's call this machine C. The proxy that you have to write will execute on a computer that we will call P, listening on some port (perhaps 55555) for requests. It is possible that

hosts C, P and H are the same host, but it will be more fun if they are different hosts.

When the proxy server receives a request, it simply checks to see whether it is the `ps` command. Whenever it sees the `ps` command, it discards it; in all other cases it passes the command to the server on host S.

Remember that the `ps` command may be used with options and/or arguments. A number of options exist. You may inspect the incoming request and pass every character to S; however, when the proxy sees a `p` it waits for the next character. If it is an `s` is starts discarding characters (by not forwarding them to S) until it sees an incoming newline character. If the character following the `p` is not an `s` the proxy sends the `p` as well as the character following it to S and then continues sending every character to S. Note, however, that this may cause the following to happen:

```
$ echo Oops!
Oo
```

or even worse

```
$ echo Oops!
$ ls -l
Ools -l
```

You should be able to correct this problem without too much effort.

9.3 Challenges

If your proxy server works as described above (but without discarding anything in the example, since the `ps` command was not used in the example) you will earn a rather low mark — see Section 9.4 below.

Here are some challenges to increase your mark.

1. The command `ps` may be echoed locally by the client. With the proper configuration, the `ps` would not even be displayed. It would be as if the user is unable to even type the command.

2. The proxy server, as described this far, is hard-coded to connect to S. It is trivial to set the address of S in a configuration file. However, is it possible to build the proxy server such that it is able to connect to an arbitrary server, specified by the user when using the proxy server?

3. It is possible to have multiple commands on a *nix command line; consider, for example `ls & ps & cd /etc` (or the same commands separated by semicolons). Ideally the proxy should allow the `ls` and `cd` to execute, but strip out the `ps`.

Note that I deem examples such as the following beyond the scope of this project:

```
$ echo ps > x
$ chmod 777 x
$ ./x
```

Stated differently, there is no way in which you can follow all the instructions and deliver a perfect solution. (Also note that we have not blocked users from using the `top` command...)

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.

Chapter 10

Practical schedule

Practical assignments have to be uploaded to clickUP by the due date. The assignment then has to be demonstrated during the indicated practical sessions to earn marks.

The practical assignments are due on the following dates:

Wednesday, 6 March 2024 Assignment 1

Wednesday, 3 April 2024 Assignment 2

Wednesday, 10 April 2024 Assignment 3

Wednesday, 17 April 2024 Assignment 4

Wednesday, 24 April 2024 Assignment 5

Wednesday, 1 May* **2024** Assignment 6

Wednesday, 8 May 2024 Assignment 7

Wednesday, 22 May 2024 Assignment 8

Wednesday, 29 May 2024 Assignment 9

Assignments are due at 13:00 on the days noted. In all cases an automatic extension of 24 hours will be granted. That means the assignment will be accepted for another 24 hours without any penalty. However, beyond that, assignments are too late to be accepted. (Note, due to technical limitations, this means that clickUP will display the due date and time for Assignment 1 as 7 March 2024 by 13:00. In the lecturer's mind it is actually due on 6 March 2024 by 13:00.)

Since 1 May is a public holiday, submissions will be accepted until 22:00 on 2 May without penalty.

Submissions that are due on a Wednesday are to be demonstrated during the practical period on the next Friday (10:30–12:30) *or* the immediately following Tuesday (14:30–17:30).

Demonstrations not only involve showing what your software does, but also include answering some questions about reasons you decided to solve a problem in some particular way, or about the way an aspect of your program works. Answers to such questions may affect your mark. (In practice, many students 'volunteer' the expected information during the demonstrations, such that additional questions about the work are not necessary.)

These tight schedules explain why extensions cannot be granted beyond the automatic 24-hour extension.

You are allowed to upload a given assignment any number of times and only the final submission will be deemed to have been submitted. Rather submit work as soon as it is almost ready, and then again before the due date, than missing the opportunity to upload your work.

When two students collaborate on a project, both have to be present simultaneously to demonstrate their project.