# 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.