

Socket programming - FTP server

Introduction

Your task is to create a very simple **FTP server** program using sockets.

1 – The server needs to be able to interpret the protocol commands, including: **LIST**, **RETR**, **STOR**

2 - The client should be a standard command line FTP client application (available on your OS). The **application** commands that should work from the client are:

```
dir (or ls)
get <filename>
put <filename>
```

Assume that the files are text only (ASCII rather than binaries). There is no need to implement text X binary modes.

Guidelines

The *server* needs to interpret requests that the client sends. To successfully implement **LIST**, **STOR** and **RETR**, the *server* has also to interpret the following protocol commands: **USER**, **PASS**, **SYST**, **PASV**, **PORT**, **QUIT**. Learn about the protocol commands and response messages that need to be issued. Refer to the material given in class, and find additional information about FTP at <http://www.freesoft.org/CIE/RFC/959/>.

Remember that the FTP protocol requires **two connections**, one in port 21 and a data connection in port 20 (actually any other random port). The machines in the lab may not allow users to bind applications on these ports. Alternatively, use ports **1121** and **1120**. You will need to start the ftp client indicating that you want port 1121 to be used:

```
$ftp 127.0.0.1 1121          OR
$ftp
ftp>open 127.0.0.1 1121
```

The interpretation of the protocol commands can be coded as the following example:

```
//USER
if (strncmp(rbuffer,"USER",4)==0) {
    printf("Logging in \n");
    sprintf(sbuffer,"331 Password required (anything will do really... :-) \r\n");
    bytes = send(ns, sbuffer, strlen(sbuffer), 0);
}
//PASS
if (strncmp(rbuffer,"PASS",4)==0) {
    printf("Typing password (anything will do... \n");
    sprintf(sbuffer, "230 Public logging in ok, type your email for password \r\n");
    bytes = send(ns, sbuffer, strlen(sbuffer), 0);
}
```

The assignment is going to be marked based on functionality. *The server should not crash if the user tries to issue commands that are not implemented.* The marks are distributed as follows:

- 2 marks for correct connection/disconnection (the server should allow reconnection after errors, disconnections, etc) and **LIST** request (**dir**)
- 3 marks for the **RETR** request (**get <filename>**)
- 3 marks for the **STOR** request (**put <filename>**)
- 2 marks for **robustness** (i.e., the server does not crash or lose the connection during my attempts to upload or download)

Notes:

- 1 - Submit your file *.c (**the server only**) electronically via stream.
- 2 - Your submission has to be a single source file (server) compatible with GCC (if using other compilers to develop the assignment, **make sure the final source compiles on GCC**).
- 3 - This assignment is worth **10 marks**.
- 4 - You may lose marks for late assignments.