Project #2: A Parallel FTP Client

The project is due Monday, November 19th at 11:59pm. All assignments are to be carried out individually. No collaboration is allowed. The detailed project overview was presented in Lecture 9.

The Assignment
In this assignment, you will implement a parallel FTP **client** application using the socket API. The application is called ***pftp***, and must be implemented in C or Python. The FTP protocol is specified in RFC 959. The assignment can be broken down into two components: basic ftp operation and parallel download. Both components must be completed by the due date.

Part 1: Basic FTP Operation (65%)

The client application will take an FTP server and a file name as command-line parameters. When executed, the client will simply retrieve the file specified following the FTP protocol and save it to a local file with the same name.
Note: you will not need to implement the put operation; and each FTP session operates in the default passive mode, which we have introduced in Lecture 9.

Usage:
```
 pftp  [-s hostname]  [-f file] [options]
 pftp -h | --help
 pftp -v | --version
```

For example, by executing:

```
% pftp -s ftp://mirror.keystealth.org/gnu/  -f /ProgramIndex
```
The file named ProgramIndex will be created in the current directory.

On success, the ***pftp*** application exits with code 0. If an error occurs, the application exits with the appropriate code and prints a human-readable error message on standard error.

More details on the command line:
-h or --help
Prints a synopsis of the application usage and exits with return code 0.
-v or --version
Prints the name of the application, the version number (in this case the version has to be 0.1), the author, and exits, returning 0.
[-f *file*] or [--file *file*]
Specifies the file to download.
[-s *hostname*] or [--server *hostname*]
Specifies the server to download the file from

Options:

[-p *port*] or [--port *port*]
Specifies the port to be used when contacting the server. (default value: 21).
[-n *user*] or [--username *user*]
Uses the username *user* when logging into the FTP server (default value: anonymous).
[-P *password*] or [--password *password*]
Uses the password *password* when logging into the FTP server (default value:user@localhost.localnet).
[-m *mode*] or [--mode *mode*]
Specifies the mode to be used for the transfer (ASCII or binary) (default value: binary).
[-l *logfile*] or [--log *logfile*]
Logs all the FTP commands exchanged with the server and the corresponding replies to file *logfile*. If the filename is "-" then the commands are printed to the standard output. The lines must be prepended by C->S: or S->C: for client-to-server and server-to-client lines, respectively. For example:

```
S->C: 220 Server (Uchicago) [128.135.164.133]
C->S: USER anonymous
S->C: 331 Anonymous login ok, send your complete email address as your password.
C->S: PASS user@localhost.localnet
```

Part 2: Parallel FTP Download (35%)

The next part of the assignment is to implement a parallel (multi-track) ftp download. The idea is that the ***pftp*** client uses separate threads to simultaneously connect to multiple servers, and download the same file from different positions in the same file. On the server side, this functionality is already supported by current ftp servers through the "restart marker" option, i.e. REST. A REST request sets the start position.

There's no intuitive user interface for this type of functionality. So we will use something very simple and straightforward. ***pftp*** takes in an additional command line argument:
[-t para-config-file] or [--thread config-file]

Usage:
```
  pftp  [-t para-config_file] [options]
```

Each line in the config-file specifies the login, password, hostname and absolute path to the file. Each line should be separated by a Line Feed (LF) character.

When ***pftp*** reads in the para-config-file, it parses the N servers on the list. For each of the N servers, ***pftp*** spawns a separate thread, connects to the server, and moves to the specified directory. To download a file of S bytes, ***pftp*** performs a joint download, where each thread downloads a segment of S/N bytes. The downloaded segments should be put together to form the entire file. Note: ***pftp*** can use the --thread option simultaneously with the -l option. And ***pftp*** *only supports binary model not ASCII mode.*

For each line of the para-config file, the format should be:
ftp://username:password@servername/file-path.

An example of the para-configuration file is as follows (which will be used as a test for your program)
ftp://cs23300:youcandoit@ftp1.cs.uchicago.edu/rfc959.pdf
ftp://socketprogramming:rocks@ftp2.cs.uchicago.edu/rfc959.pdf

Note that using a para-config_file, any arguments specified for username and password through options will be overridden by the values specified in the file.

Discussion
This application must be *extremely* resilient to wrong/unexpected input data, repeated values, etc. Part of your assignment is to understand how the ***pftp*** application could be abused by a user and to provide meaningful error messages if something goes wrong. You have to program defensively and with user-friendliness in mind. Segmentation fault is not an option!

You **CANNOT** use any **existing library** to retrieve the files from the server. You have to write your own code that will open connections, play the protocol, parse the data received from the server, etc.

The application should return meaningful error messages if an error occurs. In addition, the exit value should give some information about what happened. By returning different exit values in association with different errors the application becomes easily scriptable (that is, an external script can invoke the application and manage error conditions). Obviously, it is impossible to foresee all possible error conditions, therefore the following list is obviously incomplete, but this is expected from your program.

Exit code	Explanation
0	Operation successfully completed
1	Can't connect to server
2	Authentication failed
3	File not found
4	Syntax error in client request
5	Command not implemented by server
6	Operation not allowed by server
7	Generic error

Hints and Additional Notes

- Note that when invoked without any parameters the application should behave as if it were invoked with the --help option.
- All error message must be printed to standard error.
- If a parameter is specified multiple times, then the last value assigned to the parameter is the value accepted.
- List of FTP raw commands: http://www.nsftools.com/tips/RawFTP.htm
  Hint: the key commands that you should be using for this project include: PASV, REST, RETR, TYPE, USER, PASS, QUIT, etc.
- **Additional info on FTP can also be found in the lecture9.pdf**
- Tutorial for Multi-thread programming in C:
  https://www.geeksforgeeks.org/multithreading-c-2/

FTP Servers for testing your program:

We have set up two ftp servers:

ftp1.cs.uchicago.edu	username: cs23300.	Password: youcandoit
ftp2.cs.uchicago.edu	username: socketprogramming	Password:rocks

Two files were placed on both servers: rfc959.pdf, lecture8.pdf

Submission

Use the following dropbox upload link. You can submit multiple times but only the last submission will be used for grading.

https://www.dropbox.com/request/9CvMbG9PvJyzX4aq448G