



COSC 328 – LAB 3
Introduction to Networks
2020 Winter Term 1

Deadline: October 9th, 2020 at 11:59 PM Pacific Time (Sharp). Delayed assignments will receive penalty as described in the course outline.

Introduction

In this lab, we will have some fun on writing a simple FTP server.

Simple FTP Server (100 Marks)

We will explore a bit of TCP. As we don't want to delve into the topic of threading, this will be a simple File Transfer Protocol (FTP) server. File Transfer Protocol is a bit too much so we will limit our simple version to just 5 commands and these commands will be slightly different from the real ones.

Because you are running the client and the server on the same machine, you should create two directories in your working directory. One called client and the other called server. These are the directories that files from the ftp commands will be stored.

OPEN #

Attempts to open an FTP connection to 127.0.0.1 using the port number specified. Note that when testing your program the TAs will be using ports of their choosing.

GET filename

Requests a file from the server. This file will be loaded from the server directory, transferred by the server to the client, and saved by the client to the same name in the client directory.

PUT filename

Sends a file to the server. This file will be loaded from the client directory, transferred by the client to the server, and saved by the server to the same name in the server directory.

CLOSE

This closes the current connection to the server but keeps the client running so that the client can connect to another server if they wish to (and the TAs will wish to)

QUIT

Exits the client, closing any open connections

As a starting point here is a simple TCP client

```
from socket import *
HOST = '127.0.0.1'
PORT = 12000
```



a place of mind
THE UNIVERSITY OF BRITISH COLUMBIA

```
# set up the tcp socket
sock = socket(AF_INET, SOCK_STREAM)
sock.connect((HOST, PORT))

while (True):
    s = input("Message: ")
    sock.sendall(s.encode("utf-8"))
    data = sock.recv(1024).decode("utf-8")
    if data == "QUIT":
        break
    print ("Received: ", data)
sock.close()
```

And the server:

```
from socket import *
```

```
HOST = '127.0.0.1'
```

```
PORT = 12000
```

```
# set up the tcp socket
sock = socket(AF_INET, SOCK_STREAM)
sock.bind((HOST, PORT))
sock.listen()

# listen for a connection
conn, addr = sock.accept()
print("Connected to " , addr)
while (True):
    data = conn.recv(1024).decode("utf-8").upper()
    print(data)
    conn.sendall(data.encode("utf-8"))
    if data == "QUIT":
        break
conn.close()
sock.close()
```

Submission Requirements:

A zip file containing the client and the server programs. The sample is in Python, but you are welcome to use Java if you prefer. Screenshots of the client and server running will help if there are issues running your programs.



a place of mind
THE UNIVERSITY OF BRITISH COLUMBIA

NOTES:

Python has an FTP library (ftplib). The aim of this assignment is NOT to use the ftplib, rather, you write your own simple library. Please do NOT use the ftplib (or similar existing ones in other programming languages).

The way you should see if your program is working is as follows:

- run your server
- run your client
- open a connection using the OPEN PortNumber on the client side
- run other commands such as GET or PUT (followed by the file name)
- close the connection

Of course, this is not the proper way to test your program. But, a simple way to ensure it works.

You can send proper commands (such as `print("Received", data)`) to see the commands are properly received.

Please write a professional program! You need to catch exceptions if the file does not exist. Use classes for client and server, etc. ...

Advanced practices (0 points)

For those of you who want to take things to the next level, you can explore the threading options of Python or Java and create a server that is able to support multiple connections at the same time.