Redes de Computadores

Computer Networks

Lab 2 – Stream oriented network programming with TCP sockets

Summary

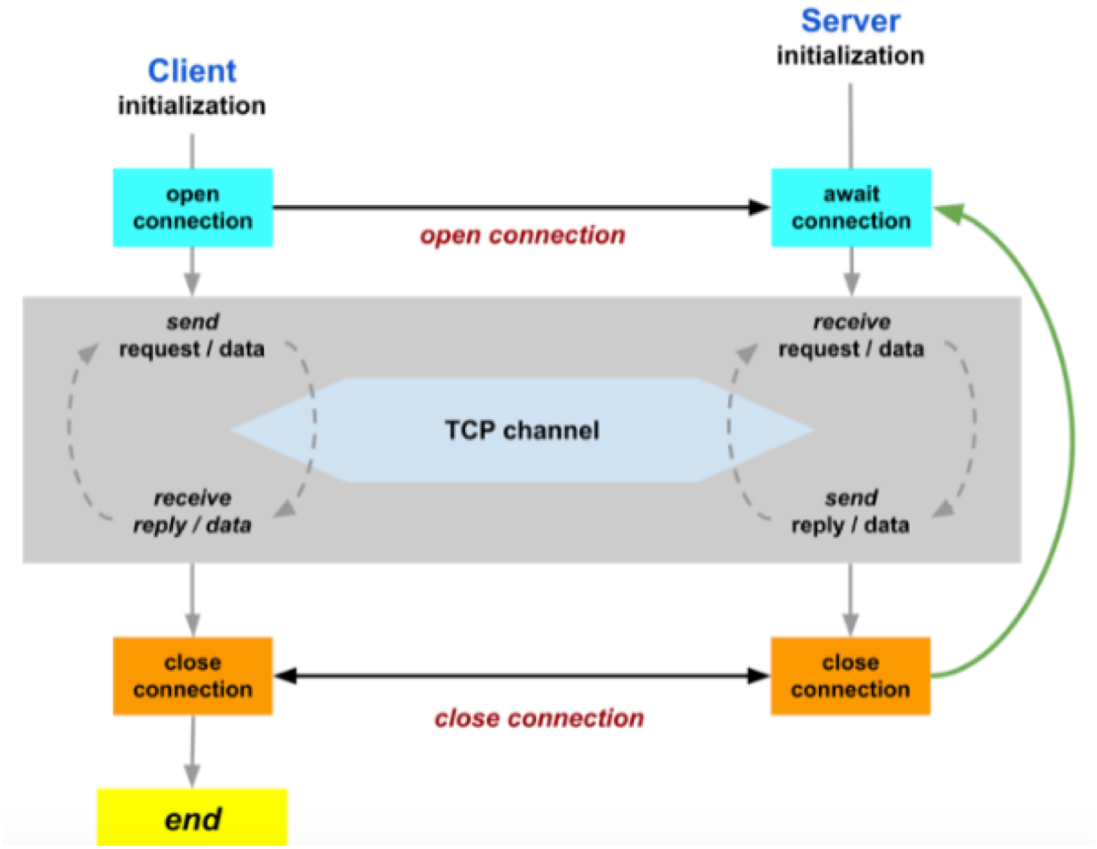
- Client/Server model
- Example (Echo Cliente and Server)
- Assigments

Client/Server Model

- Two autonomous components:
  - **server** - first to run and usually always running
  - **client** - usually started by user to request a service

- TCP follows the same abstract client/server model that characterizes UDP communication

Client/Server execution diagram



TCP Channel

- **Reliable** connection between two processes;
- **Initiated** by the client, addressed to a machine and port, where the server **already** awaits connection requests;
- Enables a continuous and bidirectional flow of data (bytes) between the two processes;
- Closed at the request of **either** the client or the server;
- Both the client and server **are aware** when a connection is established or broken/closed.

TCP oriented communication

- Communication is based on **sockets**
  - TCP sockets for accepting connections (server)
  - TCP sockets for data exchange (client and server)

- Addresses identify connection endpoints (origin and destination processes/socket)
  - **Host** (IP Address) 10.1.233.67, 127.0.0.1, 192.168.1.1, etc.
  - **Port** (16 bits) 80, 443, 8000, 8080, etc.

TCP Client

TCPclient.py

TCPclient.py 〉No Selection

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
@author: cpm
"""
from socket import *

serverName = "localhost"            # server name
serverPort = 12000                  # socket server port number
sockBuffer = 2048                   # socket buffer size

def main():
    clientSocket = socket(AF_INET,SOCK_STREAM)       # create TCP socket
    clientSocket.connect((serverName, serverPort))   # open TCP connection

    sentence = input("Input lowercase sequence: ")   # take input

    clientSocket.send(sentence.encode())             # send user's sentence
                                                  # over TCP connection

    modifiedSentence = clientSocket.recv(sockBuffer).decode()  # receive response
    print("Message received: ", modifiedSentence)

    clientSocket.close()            # close TCP connection

main()

```

TCP Server

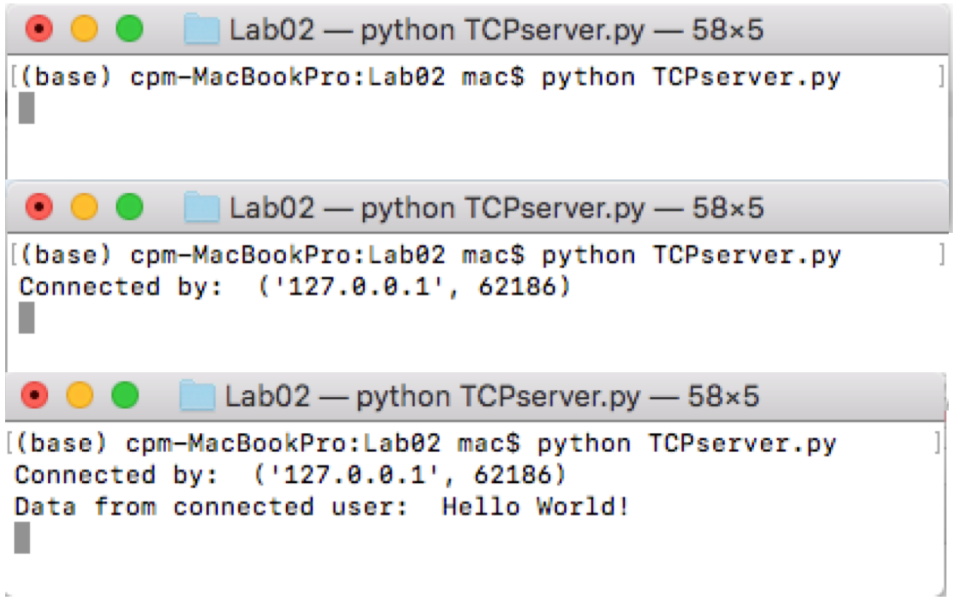
TCPserver.py

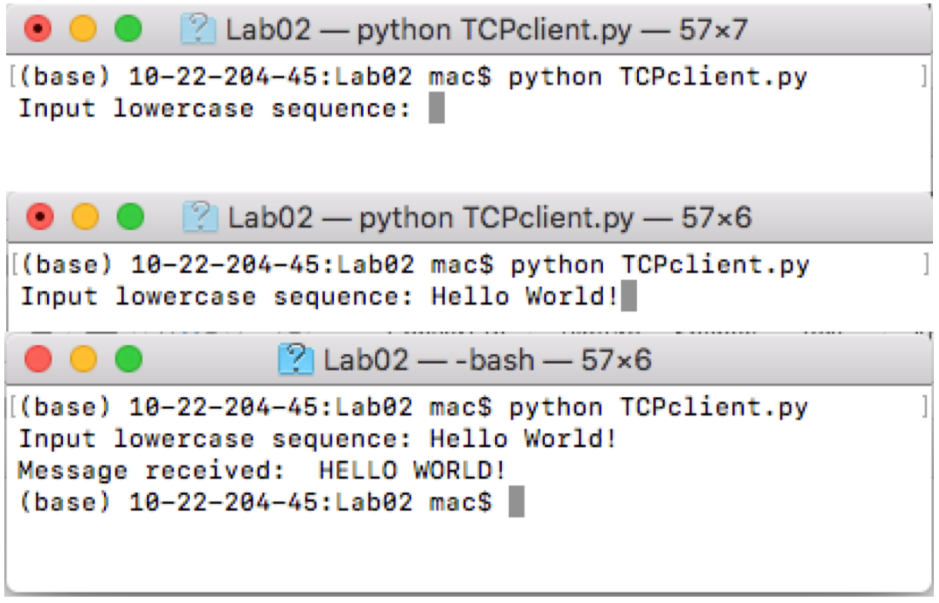
TCPserver.py 〉 No Selection

```
"""
@author: cpm
"""
from socket import *

serverPort = 12000
sockBuffer = 2048

def main():

    serverSocket = socket(AF_INET,SOCK_STREAM)   # create TCP welcoming socket
    serverSocket.bind(("", serverPort))

    serverSocket.listen(1)              # begin listening for incoming TCP requests
    print("Server is running")

    while True:
        connSocket, addr = serverSocket.accept()    # waits for incoming requests:
                                                    # new socket created on return
        print("Connected by: ", str(addr))

        sentence = connSocket.recv(sockBuffer).decode()     # read a sentence of bytes
                                                            # received from client

        print("Data from connected user: ", sentence)       # display received sentence

        sentenceUpper = sentence.upper()                    # convert to upper case

        connSocket.send(sentenceUpper.encode())             # send modified sentence over
                                                            # TCP connection

        connSocket.close()      # close TCP connection:
                                # the welcoming socket continues

main()
```

Assignment 1

- Get the source code of *TCPclient.py* and *TCPserver.py* available from CLIP.
- Study the source code and run it.
  - You can invoke the Python interpreter in a shell.





Assignment 2 – Adding two numbers

In this assignment, you'll write a client that will use TCP sockets to communicate with a server that you will also write. Here's what your client and server should do:

Client

1. Accept a name and an integer between 1 and 100 from the keyboard;
2. Create a TCP socket and open a connection to the port agreed with the server;
3. Send to the server: (i) the entered name and (ii) the entered integer value and then wait for a server reply;
4. When the client receives the reply from the server it should process it and then display:
    - its name and the server's name;
    - Its integer value, the server's integer value and the sum of both;
    - The client then terminates after closing any created sockets. As a note (and as a check that you are doing things correctly) you should make sure for yourself that the values and the sums are correct!

Server

1. Create a string containing a name (e.g., "Server of John Q. Smith") and a random number;
2. Create a TCP socket, bind it to the agreed port, and wait for a connection;
3. On accepting a client connection, the server should:
    - Display the client's received name and the server's name;
    - Display the client's number, its number, and the sum of those numbers;
    - Reply to the client with the server name and the server number;
    - If your server receives an integer value that is out of range, it should terminate after releasing any created sockets. You can use this to shut down your server.

Assignment 3 – File Transfer

You should write a file transfer program where:

- The server is invoked with the following command line:

  ```
  >python TCPserver.py port
  ```

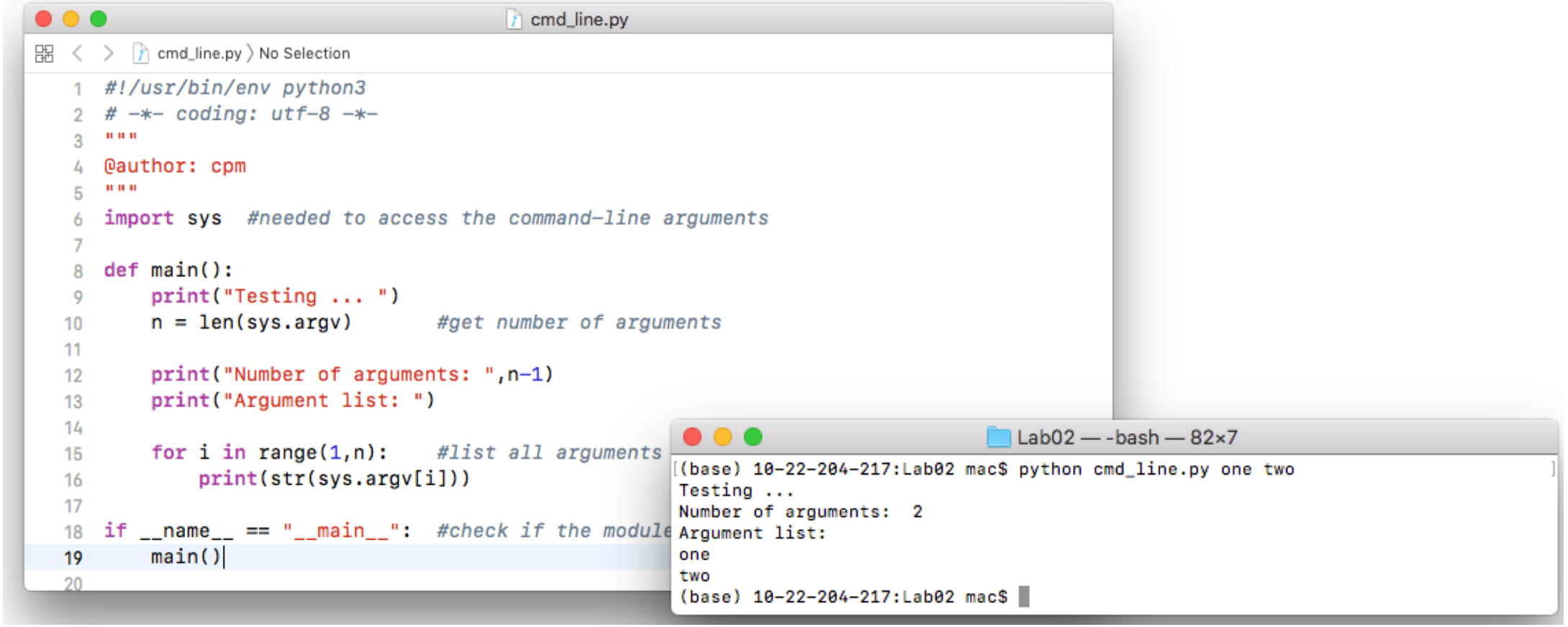
- The client is invoked with the following command line:

  ```
  >python TCPclient.py server:port:fNameInServer fNameInClient
  ```

- The client connects to the server
- The client sends the file name to be transferred
- The server sends the file to the client in 1024 bytes blocks
- Client receives the blocks, sent by the server, and writes them to its local disk.
- Compare the contents of the two files (windows: fc command; linux and macOs: diff command).

Assignment 3

Python - command line commands



Assignment 3

Python - command line commands (Spyder)

Run -> Configuration per file