# P1: Threading

In this project we demonstrate the interaction between a server that accepts more than one connection from the clients. For each of those connections, the client sends to numbers separated by a space. The first number is the initial number that the server will start counting from. The second number is the number of increments to be done by the server. For example, if the user inputs: `2 10`, this means that it asks the server to start counting from `2` and it will increment the number `10` times. So, after the end of the program, the final result returned by the server is `12`.

The code consists of three main files, as follows:

- `client.py`: The code executed by a client. Their can be many clients running in the same time.
- `server.py`: The code executed by a server. The server can receive many connections from the clients at the same time.
- `constCS.py`: Contains the constant configurations needed by the server and the client. For example the host IP and the server's listening port number. In addition to that, it has the variable `MAX_NUM_CLIENT_CONNECTIONS_TO_SERVER` that defines the total number of connections that can run by the server simultaneously.

To show the code running, a video is recorded when sharing my screen and uploaded to youtube for easier access. When opening the video, the server is shown on the left side of the screen and three clients are sending requests concurrently to the server. On top of that, the program allows the clients to send `STOP` messages instead of sending messages to the server to close the connection with the server.

## Some Highlights from Server Code

The part responsible for receiving new connections from users and execute the process responsible for handling each user.

```
s = socket(AF_INET, SOCK_STREAM)
  s.bind((HOST, PORT))  # AB: Bind the socket to specific IP address and
port.
  s.listen(MAX_NUM_CLIENT_CONNECTIONS_TO_SERVER)  # AB: maximum number of
connection to be accepted by siumltaneously by the server.
  while True:                # forever
    print("Server listening on port: {} ...".format(PORT))
    (conn, addr) = s.accept()  # returns new socket and addr. of the client
    print("Accepted connection from {}:{}".format(addr[0], addr[1]))
    client_handler = Process(target=handle_client_request, args=(conn,
addr))
```

The part responsible for handling the number and the number of increments sent to the server by the client.

```
  while True:
    data = conn.recv(1024)  # receive data from client
    if not data:
```

```
      break  # stop if client stopped
    msg = data.decode()  # process the incoming data into a response
    split_res = msg.split(' ')
    num = int(split_res[0])
    repetition = int(split_res[1])
    sleeplist = list()
    for i in range(repetition):
      thread_id = "{}:{}.{}".format(addr[0], addr[1], i)
      subsleeper = Thread(target=client_thread_worker, args=(thread_id,
addr))
```

During the loop of increments, it is implemented using non-blocking threads so that the server keeps receiving requests from users while counting the numbers from the users that it has received previous requests from. This part is shown in the following code snippet.

```
global num
  print("Server counts {} in thread {} for connection {}:{}".format(num,
thread_id, addr[0], addr[1]))
  x = 0.0
  t = time.gmtime()  # -
  num = int(num)
  s = random.randint(1, 10)
  txt = str(t.tm_min) + ':' + str(t.tm_sec) + ' ' + thread_id + ' is going
to sleep for ' + str(s) + ' seconds'  # -
  print(txt)  # -
  t = time.time()
  c = s * 8000000
  for i in range(c):
    x = x + 1.0
  print(time.time() - t)
  t = time.gmtime()  # -
  num = num + 1
  txt = str(t.tm_min) + ':' + str(t.tm_sec) + ' ' + thread_id + ' has woken
up, seeing shared x being ' + str(num)  # -
  print(txt)
```

The code snippet that is executed by the client and closes the connection with the server only when it sends the string `STOP` is shown below.

```
s = socket(AF_INET, SOCK_STREAM)
s.connect((HOST, PORT)) # connect to server (block until accepted)
while True:
    msg = input("Please enter a number, space, then number of increments to
be done by the server: ")
    if msg == "STOP":
        break
    split_res = msg.split(' ')
    init_num = split_res[0]
    repetition = split_res[1]
```

```
    print("Number received entered: {} to be repeated {} times", init_num,
repetition)

    s.send(msg.encode())    # send the message
    print("Client sent {} to the server".format(msg))
    data = s.recv(1024)     # receive the response
    print("Client received result: {}".format(data))
    print(data.decode())    # print the result
s.close()               # close the connection
```