2228 BLAKE ST #100
DENVER, CO 80205
800 445 0024 X 1

SecureSet Academy

CORE Network 500
Twisted Server
Version: November 17, 2020

Table of Contents

Environment Setup

No special environment setups required.

Abstract

Synchronous coding executes one task at a time. The task must complete and pass control back before the next task can complete. Any delays in completion of task 1 will result in task 2 and 3 being unable to start until task 1 complete. Batch scripts typically operate in this manner.

Asynchronous coding allows tasks to be split up and allows the CPU to split the processing between tasks. For example, if task 1 initiates a database query, the idle time during which it is waiting for the query to be returned is otherwise unused. In Asynchronous processing, the CPU can perform task 2 during that time.

Similarly, a web server that can only serve web pages to a single client at a time would be a synchronous web server. Since web servers serve multiple clients simultaneously, they act asynchronously, maintaining connection information on the multiple connections (sockets).

Look at http://krondo.com/an-introduction-to-asynchronous-programming-and-twisted/ for more information on how this works. Part 1: In Which We Begin at the Beginning goes into more detail.

Objective(s)

The objectives of this lab are:

- Gain a functional understanding of blocking and non-blocking sockets
- Gain a functional understanding of threads in python

Pre & Post conditions of lab

No changes to the current lab environment are expected

System Requirements & Configuration

System Requirements

Set up a fresh instance of Ubuntu (tested on Ubuntu 16.04, 64 bit). Ensure the system has python2.7 installed.

Ensure you have updated all Ubuntu components.

Network Requirements

This lab should be done on local Virtual Machines with internet connectivity (NAT or NAT Network).

Software Requirements

No additional software required. Twisted will be installed during the lab.

Data Requirements

receiver.py
thread_recv.py
tcpserver.py
twistedchofactory.py
twistedechoudp.py

Procedure – Detailed Lab Steps

Setup

No special lab setup required.

Lab Execution

Threaded vs non-threaded sockets

The file receiver.py has a simple example of a line receiver in Python2. It handles a single connection at a time.

Run the server with

```
python receiver.py
```

1. Examine the while loop in the main function. What is the exit condition for the loop?
2. Examine the while loop in the handler function. What is the exit condition for the loop?
3. What is the purpose of the handler function?
4. The socket is set to bind to 0.0.0.0. Is this a valid IP address?
    a. If not, why not. Fix this.
    b. If so, what happens when the socket binds to that address?

Connect to the server in a terminal with

```
nc localhost 5000
```

The server doesn't do anything besides receive lines. When the client disconnects, the server continues to listen for another connection.

Remember that a socket is defined as an IP address, protocol, and port. What IP addresses, protocol and ports are in use for this connection? There should be two, one at each end.

This server can never serve multiple concurrent connections. Try this by opening multiple terminal windows and executing **nc localhost 5000** on each. You should notice that the first connection displays a message on the receiver.py application. However, subsequent connections do not, but they appear to be attempting to connect. Disconnect the first connection (ctrl-c or close terminal) and note what happens.

The file thread_recv.py has a threaded example of a line receiver in Python. It can handle multiple connections concurrently.

Run the server with

```
python thread_recv.py
```

As you examine this code, note that the only difference is the additional threading commands. These commands control how the application will work with the threads and keep track of which thread is associated with which socket.

Connect to the server multiple times in different terminals with

```
nc localhost 5000
```

Again, the server doesn't do anything besides receive lines. This time, however, the server can accept multiple connections. It uses threads for asynchronous execution.

If your network is a NAT Network, try connecting from a different VM or put your connection in bridge mode and connect from the host. Is there any difference?

Use Wireshark to view these communications.

Twisted Installation

Open the Ubuntu 16.04 Virtual Machine, and go to the terminal:

```
sudo apt update
sudo apt upgrade
sudo apt install python-twisted virtualenv python-configparser
   python-pip
sudo pip install pycrypto
```

In the Twisted Server, create this Python script (available in the Student Resources Folder, twistedechofactory.py):

```
#!/usr/bin/python

from twisted.internet import protocol, reactor

class Echo(protocol.Protocol):
  def dataReceived(self, data):
    print data
    self.transport.write(data)

class EchoFactory(protocol.Factory):
  def buildProtocol(self, addr):
    return Echo()
if __name__ == '__main__':
  PORT = 8080
  print "Listening on port", PORT
  reactor.listenTCP(PORT, EchoFactory())
  reactor.run()
```

This script is influenced by https://twistedmatrix.com/trac/.

Run the script and use telnet or netcat to connect to the server. Anything you type in will simply be echoed back to your session.

View this connection in Wireshark and verify that the information is being sent in both directions.

Notice the connection type is TCP. Can this simply be changed to udp? If not, why not?

If this were a UDP connection, what would be different?

Examine at twistedechoudp.py (from https://twistedmatrix.com/documents/current/core/howto/udp.html). Load this into your server and use Wireshark to view the connection. You will need to use the following on your client to connect:

```
echo "Hello World" | nc -u localhost 9999
```

The -u (for UDP) on the netcat connection.

1. What was returned to the client? Does this make sense?
2. What shows on the server?

Advanced Lab

Use the chatserver.py and view the resulting communications with Wireshark. You may want to pair up with another student and set up communications across VM/hosts.

Use the information at https://twistedmatrix.com/trac/wiki to build a Web Server and Publish/Subscribe server. This will help get you used to using the Twisted environment. Again, use Wireshark to view the communications.

Lab Results

Submit a screenshot showing both sides of the TCP traffic.

Lab “Tear-down”

Do not tear this server down. While we will stand up another Twisted server for the Honeypot lab, it will be useful to have this server set up, as well. You will be able to clone this lab as a starting point for the Honeypot lab.

Questions/Responses

Student: Please record anything that was unclear about this lab.

Appendix

Lab Assistance

N/A

Terminology

N/A

References

- https://twistedmatrix.com/trac/
- http://krondo.com/in-which-we-begin-at-the-beginning/
- https://twistedmatrix.com/documents/current/core/howto/udp.html