Sockets and Networking with Python

Dr Ian Cornelius

Hello

Hello (1)

Learning Outcomes

1. Understand the concept of networking and sockets in Python
2. Demonstrate knowledge on how to use sockets in a body of work

Socket Programming

Socket Programming (1)

What is Socket Programming?

- An abstract principle whereby two programs can share a data stream
- Commonly done using an **Application Programming Interface (API)**
  - uses different protocols available in the internet TCP/IP stack
- Sockets are used to exploit the capabilities of an operating system to interact with the network
- Network sockets are used to establish a connection between processes on the same machines, or different ones
- A socket address is composed of two things:
  1. an **IP address**
  2. a **port number**
- Sockets consist of two primary properties, controlling how they send data:

1. Address Family
2. Socket Type

Socket Programming (2)

Address Family

- This controls the Open Systems Interconnection (OSI) network layer protocol that is used
- There are three address families in Python:

- `AF_INET`
  - the most common and is used for IPv4 addresses
  - the majority of networking is done using IPv4
  - use IP addresses and port numbers
    - an IP can be represented in its IP form (`127.0.0.1`) or in its 32-bit form (`0x7F000001'`)
- `AF_INET6`
  - used for IPv6 addresses and represented as 128-bit (16 byte) address
    - e.g. `127.0.0.1` is represented as

      `0000:0000:0000:0000:0000:ffff:7f00:0001`
  - considered to be the next generation of internet protocol
  - not as common as IPv4, but it is growing

- `AF_UNIX`
  - used for **Unix Domain Sockets (UDS)**
  - an interprocess communication protocol on POSIX-compliant systems
  - allows an operating system to pass data from process to process, without going through the network stack

Socket Programming (3)

Socket Type

- There are two socket types in Python:
  1. `SOCK_DGRAM`
     - used for **User Datagram Protocol (UDP)**
     - it does not require a transmission handshake or other setup
     - offers lower reliability of delivery
     - UDP messages may be delivered out of order, more than once, or not at all
     - commonly used for protocols where order is less important or multicasting
  2. `SOCK_STREAM`
     - used for **Transmission Control Protocol (TCP)**
     - ensures each message is delivered exactly once, and in the correct order
     - applications that deliver large amounts of data (such as HTTP) use TCP

Using Sockets in Python

Using Sockets in Python (1)

- A socket will be specified by:
  - a machine's IP address
  - a port it is listening to
  - a protocol it uses
- Sockets in Python used a module known as `socket`

```
import socket
```

- Creating a socket is done by using the function `socket()` from the module
- The general syntax for the `socket()` function is the following:

```
x = socket.socket(socket_family, socket_type, protocol=0)
```

- The syntax above shows:
  - `family`: refers to the address family
  - `type`: refers to the type of the socket
  - `proto`: this number is usually zero; but can be different in other use-cases

Using Sockets in Python (2)

The `socket` Module i

- Provides all functionality for writing TCP and UDP clients and servers
- Most applications use the concept of client/server:
  - **server**: represents an application that is waiting for connection by a client
  - **client**: represents an application that connects to the server
- These functions are commonly used for both clients and servers:
  - `socket.recv(buflen)`
    - receives data from the socket; argument indicates the maximum amount of data to be received
  - `sockwt.recfrom(buflen)`
    - receives data and the address of the sender
  - `socket.rec_into(buffer)`
    - receives data and is placed into a buffer
  - `socket.recfrom_into(buffer)`
    - receives data and is placed into a buffer, also returns the address of the sender
  - `socket.send(bytes)`
    - sends bytes of data to a specified target

Using Sockets in Python (3)

The `socket` Module ii

- These functions are commonly used for both clients and servers: (continued)
  - `socket.sendto(data, address)`
    - sends data to a given address
  - `socket.sendall(data)`
    - sends all the data in the buffer to the socket
  - `socket.close()`
    - releases the memory and closes the connection
- Want more information about the `socket` module?

Python Socket Documentation

Using Sockets in Python (4)

The `socket` Module iii

Example 1: Retrieving Data from a Local Website

- Create two variables, `ip` and `port` with the values of the local address/site you wish to connect to
  - i.e. `ip = 127.0.0.1` and `port = 80`
- A socket is created using:
  - `AF_INET` - the type of address family used, in this case IPv4
  - `SOCK_STREAM` - the type of connection being made, in this case enabling us to send and receive a message to a web server
- A connection is then made to the IP address and port number
  - passed through the `connect` function as a tuple argument
  - e.g. `s.connect( (ip, port) )`
- The message is sent to the server using the `sendall` function
  - e.g. `s.sendall("GET /5062CEM.php\r\n")`
- Data is returned from the server, and captured using the `recv` function
  - `recv` accepts a single argument, and it is the number of bytes to retrieve from the web server
  - e..g `s.recv(1024)`

```
import socket
ip = '127.0.0.1'
port = 80
message = "GET /5062CEM.php\r\n"
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((ip, port))
    s.sendall(message.encode())
    data = s.recv(1024)
```

```
data -> b'<html>\n<head>\n\t<title>5062CEM - Networking
Demo</title>\n</head>\n<body>\nHello, welcome to 5062CEM!</body>\n'
```

```
data2 = s.recv(10)
```

```
data2 -> b'<html>\n<he'
```

Using Sockets in Python (5)

The `socket` Module iv

Example 2: Sending Data to a Local Website

- Create two variables, `ip` and `port` with the values of the local address/site you wish to connect to
  - i.e. `ip = 127.0.0.1` and `port = 80`
- A socket is created using:
  - `AF_INET` - the type of address family used, in this case IPv4
  - `SOCK_STREAM` - the type of connection being made, in this case enabling us to send and receive a message to a web server
- A connection is then made to the IP address and port number
  - passed through the `connect` function as a tuple argument
  - e.g. `s.connect( (ip, port) )`
- The message is sent to the server using the `sendall` function
  - e.g. `s.sendall("GET /5062CEM.php?lecturer=Ian%20Cornelius \r\n")`
- Data is returned from the server, and captured using the `recv` function
  - `recv` accepts a single argument, and it is the number of bytes to retrieve from the web server
  - e.g. `s.recv(136)`

```
import socket
ip = '127.0.0.1'
port = 80
message = "GET /5062CEM.php?lecturer=Ian%20Cornelius \r\n"
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((ip, port))
    s.sendall(message.encode())
    data = s.recv(136)
```

```
data -> b'<html>\n<head>\n\t<title>5062CEM - Networking Demo</title>\n</head>\n<body>\nHello Ian Cornelius, and welcome to 5062CEM!</body>\n'
```

```
message = "GET /5062CEM.php?lecturer=Terry%20Richards \r\n"
```

```
data -> b'<html>\n<head>\n\t<title>5062CEM - Networking Demo</title>\n</head>\n<body>\nHello Terry Richards, and welcome to 5062CEM!</body>\n'
```

Using Sockets in Python (6)

Server `socket` Functions

- With client-server architecture, there is a *central server*
- The central server provides services to a set of machines that are connected to it
- These functions are commonly used as point of view from the server:
  - `socket.bind(address)`
    - connects to the address with the socket
    - the requirement is that the socket must be open before establishing a connection with the address
  - `socket.listen(count)`
    - the argument denotes the maximum number of connections from clients
    - it starts the TCP listener for incoming connections
  - `socket.accept()`
    - accepts client connections and returns a tuple representing the *client socket* and *client address*
    - `bind()` and `listen()` should be called before using this function

Using Sockets in Python (7)

Client `socket` Functions

- These are functions that are used on a client machine to connect with the central server:
  - `socket.connect(address)`
    - connects to the server IP address
  - `socket.connect_ex(address)`
    - same functionality as `connect()`
    - offers the possibility of returning an error if not able to connect

Goodbye

Goodbye (1)

Questions and Support

- Questions? Post them on the **Community Page** on Aula
- Additional Support? Visit the Module Support Page
- Contact Details:
  - Dr Ian Cornelius, ab6459@coventry.ac.uk