CIS 41B
Advanced Python Programming

Network

De Anza College
Instructor: Clare Nguyen

The OSI Network Model

- When 2 devices such as a client and server need to communicate with each other, their communication can be separated into layers, starting from the high level application layer all the way down to the hardware layer.
- These layers can be described with the OSI model or Open System Interconnection model. There are 7 layers in the OSI model:

7. Application layer: interacts directly with the user through tools such as a browser, Skype, email, etc.
   Common protocols in this layer are: HTTP, FTP, SMTP, DNS, Telnet
6. Presentation layer: translation of application data to / from network data, and encryption / decryption of data.
5. Session layer: set up, coordinate, and end a session of communication
4. Transport layer: determine data packet size, destination device address, transfer rate.
   Common protocols in this layer are TCP and UDP.
3. Network layer: how to route data packets in the most efficient way
   Common protocol in this layer is IP.
2. Data link layer: actual data transfer and error correction
1. Physical layer: hardware, cable, switches that support the transmission of data.

© 2021 C. Nguyen

Networking in Python

- Python provides access to 2 of the 7 layers of network services:
  - At the higher level are the modules that work with the application layer protocols such as FTP, HTTP, SMTP, etc.
  - At the lower transport layer is the socket module, which uses the TCP, UDP and IP protocol.
- Here is the list of application layer protocols and corresponding Python modules.
- In module 3 we've used urllib or requests at the application level to work with URLs.
- Now in this module we will work at the transport level or with the socket module.
- Unlike the application layer modules, the socket module is dependent on the specific system that we run on: Mac, Linux, or Windows, because it interacts with the socket API of the OS.
- Sockets had their start in the Unix system, and modern day sockets can still belong to the Unix family of sockets.
- In addition there are 2 main types of sockets, depending on the connection protocol.

© 2021 C. Nguyen

Socket Family and Type

- The 2 main families of sockets are: AF_LOCAL and AF_INET.
- AF is for Address Family
  - LOCAL is a UNIX protocol, it was the protocol of the very first sockets.
  - INET is for Internet
    AF_INET is currently the most widely used protocol for internet connection.
- There are 2 types of sockets: SOCK_STREAM and SOCK_DGRAM
- SOCK_STREAM is connection oriented. This means a connection must be established before communication can occur. A common protocol of this type is TCP or Transmission Control Protocol.
  - Connection oriented communication is similar to a phone call. Both parties are connected before the talking begins.
- SOCK_DGRAM is connectionless. This means communication can occur before a connection is established. A common protocol of this type is UDP or User Datagram Protocol.
  - Connectionless communication is similar to leaving a voicemail. The recipient of the message is not connected directly to the sender of the message.

© 2021 C. Nguyen

Socket Programming

- A socket must be created before any of the higher layer of network communication can take place. When we used urllib or requests in module 3, Python created a socket for us to send and receive data with the URL.
- A socket is a data structure that serves as a connection point between a client and a server.
- A server is a system that provides data or service for its clients.
  - An example of a server that provides data is the web server that provides the web page data that we access with the requests module.
  - An example of a server that provides a service is a print server that prints hard copy of files.
- The server typically runs in an infinite loop to wait for a client request, respond to that request, and then wait for more client requests.
- A client connects to a server on an as needed basis.
- When the client needs the data or service of the server, it connects to the server, sends the request and any necessary data, and then waits for the response from the server.
- Each of the client and server code is a process that runs on a system.

© 2021 C. Nguyen

Socket Address - Hostname

- Since a socket is a connection point for the client and server, a socket has an address to identify it.
- A socket address is made up of 2 parts: a hostname and a port number.
- A hostname is a unique name of a device on the network. The socket hostname identifies that the socket belongs to a particular device.
- The most common form of hostname is a string, such as: voyager.deanza.edu
- A hostname is resolved into an IP address, such as: 125.16.254.1
  or in IPV6 notation such as: 2001:00B8:AC10:FF01:0000:0000:0000:0000
- Less common is the string “<broadcast>” or an empty string, or an integer that is the binary value of the address.

© 2021 C. Nguyen

Socket Address – Port Number

- A server listens for its clients' requests at one or more ports. A port is typically identified with a port number.
- Port numbers range in theory from 0 – 65535.
- Port numbers less than 1024 and those larger than 50,000 are already used by computer systems and should be avoided by application programs that are creating their own connections.
- Here are some common port number and the associated services that are used by servers:

21	FTP	80	HTTP (Web)
22	SSH	110	POP3 (Mail)
23	Telnet	119	NNTP (News)
25	SMTP (Mail)	443	HTTPS (Web)

- A complete list of port numbers and their availability is at https://www.iana.org/assignments/service-names-port-numbers/
- For our applications we are free to choose any port that is unassigned on the list, typically in the 5000 range and above, or starting at around page 95 of the above website.

© 2021 C. Nguyen

Socket Creation

- A socket is created to initiate a connection, listen for incoming messages, and send responses.
- Python has a socket module that we need to import:

```
import socket
```

- To create a socket:

```
s = socket.socket( socket_family, socket_type )
```

  - The default socket_family is AF_INET, and the default socket_type is SOCK_STREAM, which are the most commonly used family and type.
  - Therefore, we typically only need to create a socket with:

```
s = socket.socket()
```

- It is good practice to close the connection when done

```
s.close()
```

- Just like with file open and lock acquire, a more convenient way is to use the "with" construct:

```
with socket.socket() as s :
```

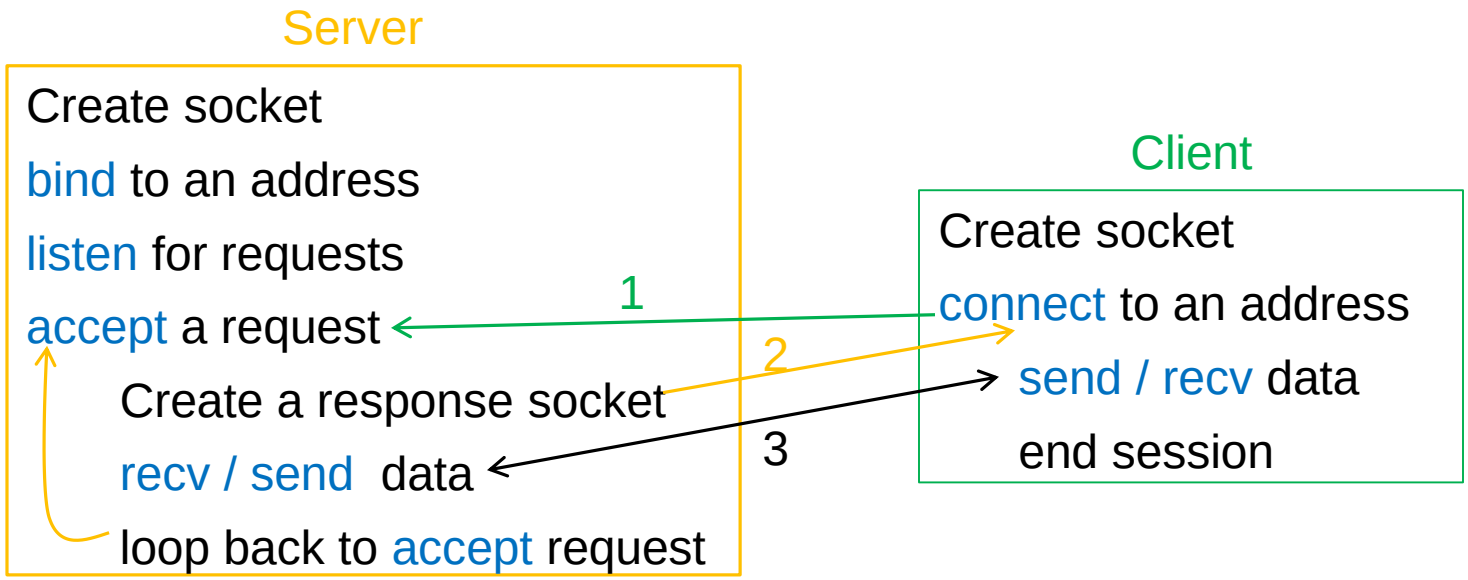
so that the connection is automatically closed when execution is outside of the with block.

© 2021 C. Nguyen

Roles of Client and Server Sockets

Steps for client and server set up and communication:

- The server creates a socket, binds it to an address, and starts an infinite loop to listen for and accept requests.
- A client creates a socket and connects to the server through the address.
- The server accepts the client's request and creates a response socket to communicate with the client.
- The server and client exchange data.
- When the client is done the client closes the connection.
- The server continues to listen for and accept another client request.



© 2021 C. Nguyen

Server Socket

- A server socket starts by identifying itself so the clients can refer to it.
- The AF_INET protocol requires a (hostname, port) tuple as the ID.
- We use the socket bind method to pass in the hostname and the port:

```
s.bind( (hostname, port_number) )
```

  - For our class we will use sockets within our own machine, so the hostname is "localhost" or IP address "127.0.0.1".
  - The port number is any of the unassigned ports as discussed on slide 7.
- It is good practice to declare 2 constants at the top of the source file for the hostname and port number. This way it is easy to change them as the need arises (such as when the port is no longer available).
- Next, enable the server socket to listen for client requests:

```
s.listen()
```

- Then, to accept client requests:

```
( conn, addr ) = s.accept()
```

- The accept method is blocking, it waits until there is a request from a client. When a client request arrives:
  - 'conn' is a new socket object that is used by the server to send and receive data with the client
  - 'addr' is the address bound to the client socket

© 2021 C. Nguyen

Client Socket

- After a socket is created, the client makes a request to a server:

```
s.connect( (hostname, port) )
```

  where: hostname is the hostname of the server
  port is the port number of the server

- The connect method is blocking, which means the client waits until the connection is established.
- If there is error with the connection, connect will keep trying to establish the connection. If the server is down, then an exception will be raised.

© 2021 C. Nguyen

Setting a Timer

- Both the accept method of the server and the connect method of the client will block, therefore we can set a timer to release the socket if needed.
- The socket object method: settimeout(n)
  sets the timer for n seconds
- And when the timer times out, it raises the exception socket.timeout
- Example

```
try :
      s.settimeout(3)                    # set timer to 3 seconds
      ...
      (conn, addr) = s.accept()          # this method blocks
      …
 except socket.timeout :       # exception when the timer times out
       …
```

© 2021 C. Nguyen

Sending and Receiving Data

- Once the connection between the client and the server's communication sockets is established, both sides can send and receive data.
- To send data:

```
num = s.send( data.encode('utf-8') )
```

  - If data is a Python str data type, it needs to be encoded into a byte string as shown above.

    If data is any other Python data structure, it needs to be pickled into a byte string.
  - Returns the number of bytes sent.
- To receive data:

```
data = s.recv( size ).decode('utf-8')
```

  - size is recommended to be a small power-of-2 (512, 1024, 2048, 4096) bytes.
  - The returned value is a byte string that can be converted into to a Python str type, as shown above, or un-pickled into a Python data structure.

© 2021 C. Nguyen

Going further…

- The sockets we've discussed so far is for a single client – server connection and can be useful if we're connecting our standard computer to another computer that is used as a controller for other hardware devices, such as a raspberry pi.
- To implement a server that can handle multiple clients we use threads, where each thread is to serve one client.
  - We can move the accept method inside the infinite loop of the server, and with every new connection we start a new thread.
  - Then the thread runs a function that has an infinite loop to process requests from the client.
- It is also possible to set up a server with several server sockets, and use a polling mechanism to keep checking the sockets for activity. The polling mechanism is done with the select module, which relies on the OS polling functions.

Up Next: Timing

© 2021 C. Nguyen