Module 08 – Sockets

What Is A Socket?

- Socket is an endpoint for communicating processes across a network. The socket mechanism is versatile, supporting both connection oriented communications (stream sockets) providing point to point communications, and connectionless communications (datagram sockets) providing the ability to broadcast information.

- An applications plugs in to the network, sending and receiving data, through a socket.

- The socket mechanism allows the programmer full control over virtually every aspect of the programming. Which protocol transport the data, the domain, etc.

Protocols types - TCP

Connection Oriented Communications — TCP:

— The Transmission Control Protocol, better known as TCP, is one of the widely used protocols.

— Applications using this protocol will connect to each and then pass information, just like when you use the phone.

— The protocol make the data to be received safely:

- In order
- Without errors
- etc

Protocols types - UDP

- Connectionless Communications And UDP

— Applications need not be “connected” in order to communicate. Just as mail can be sent to your home address, data can be sent, in messages called datagrams.

— The User Datagram Protocol, better known as UDP, is the messenger protocol for this type of communication. This protocol allows directed datagrams and broadcasting.

— Connectionless communications is generally considered to be less reliable.

  - An application might broadcast on a network that is a temporarily down or send a message to a host that is not running.
  - There is no way for the sender to verify that the data has been received on the other side, in order and without errors

Sockets protocols - Stream Sockets

- Sockets come in two main flavors, one of them being the stream socket.
- Stream socket is full duplex byte stream implementing the connection oriented model.
  - — Server sets up a socket with a well known address.
  - — Clients connect to the server and data is passed on this open connection.
  - — When the client and or server are done the connection is terminated. The socket mechanism provides an API for each step on the client and the server. The minimal steps required are:

— Server side:

- socket()
- bind()
- listen()
- accept() (possibly repeating the accept())

— Client side:

socket()
connect()

Creating sockets

- Sockets are used nearly everywhere. Sockets mechanism in python placed in socket module.

- The first step is to create an endpoint, a socket, with socket function:

socket(family=AF_INET, type= SOCK_STREAM, proto=0) Create'a new socket using the given address family, socket type and protocol number.

— The address (and protocol) family can be:

- socket.AF_INET-— for internet sockets ipv4
- socket.AF_INET6 — for internet sockets ipv6
- socket.AF_UNIX— unix domain sockets (UNIX IPC), not supported on all platforms

Creating sockets — cont’d

Socket type:

-socket.SOCK_STREAM

-socket.SOCK_DGRAM

-socket.SOCK_RAW

-socket.SOCK_RDM

-socket.SOCK_SEQPACKET

- Only sock_stream and sock_dgram appear to be generally useful
- The proto argument - this parameter specifies a particular protocol to be used with the socket
- Normally only a single protocol exists to support socket type within a given protocol family. In this case, proto stays default (zero)

— TCP for stream sockets and UDP for datagram sockets

- However, it is possible that many protocols may exist, in which case a particular protocol (protocol number) must be specified

Binding The Socket

- The server now has to “bind” the socket
- socket.bind(address) - bind the socket to address, so that clients can connect to it
- The socket must not already be bound.
- The format of address depends on the address family:
  - — In INET and INET6 address family, the address should be a tuple of ip and port
  - — In UNIX address family, the address should be a file system path

- For Example:
  — s.bind((socket.gethostname(), 8765))
  — s.bind((‘localhost', 8765))

Binding Internet Socket

- The IP

— Server application can use socket.gethostname()so that the socket would be visible to the outside world.
— Using ‘localhost’ or '127.0.0.1' the socket will be only visible within the same machine.
— Empty sting ('') specifies that the socket is reachable by.any address the machine happens to have.

- The port
  - low number ports are usually reserved for "well known” services (HTTP, ftp, telnet, etc).
  - The non-privileged ports value should be at least 4 digits number

Listen for connections on a socket

- After binding a socket to an address the server must create a queue for clients wanting to connect to the server. A connect request from a client is queued until the server accepts the connection.

- socket.listen(backlog)
  - — The backlog argument specifies the maximum number of queued connections
  - — The maximum value is system-dependent (usually 5), the minimum value is forced to 0.

Accepting Connections

- The server, having created, bound socket and created a queue for the clients, can now accept requests from clients
- socket.accept()
  - — The return value is a pair (conn, address) where
    - connis anew socket object usable to send and receive data
    - address is the address bound to the client
- accept blocks the calling process until a connection is present.

Closing Down The Connection

- The socket can be closed using
  - — close
  - — shutdown
- socket.close()- close the socket.
  - All future operations on the socket object will fail.
  - This function actually destroy the socket
- socket.shutdown(how) – shut down one or both halves of the connection.
  - — If how is SHUT_RD, further receives are disallowed.
  - — If how is SHUT_WR, further sends are disallowed.
  - — If how is SHUT_RDWR, further sends and receives are disallowed. The socket will still be able to receive pending data that already sent.

The client

- There is no difference between the client and the server as far as creating the socket is concerned.
- socket.connect(addr) — connect to remote socket at address - addr

Socket example

Demo

Socket Example — server side

```
import socket

HOST = ""  # Set the host IP address or leave it empty for localhost
PORT = 50007

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
conn, addr = s.accept()
print("Connected by {}".format(addr))

while True:
    data = conn.recv(1024)
    if not data:
        break
    conn.sendall(data)

conn.close()
```

```python
import socket

HOSTNAME = 'mydomain.com'
HOST = socket.gethostbyname(HOSTNAME)
PORT = 50007

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
s.sendall(b'Hello, world')
data = s.recv(1024)
s.close()
print('Received: {}'.format(data.decode()))
```

Receive data from the socket

- socket.recv(bufsize) - receive data from the socket.
- The return value is a string representing the data received. The maximum amount of data to be received at once is specified by *bufsize*.
- The best bufsize match with hardware and network realities should be a relatively small power of 2, for example - 1024

Send data to the socket

- socket.send(string) - send data to the socket.
  — Returns the number of bytes sent.

- socket.sendall(string) - send data to the socket. Unlike send(), this method continues to send data from string until either all data has been sent or an error occurs.
  — None is returned on success.
  — Anexception is raised on error.
  — In case of error, there is no way to determine how much data, + if any, was successfully sent.

Getting host by name

- * socket.gethostbyname(hostname) - translate a host name to IPv4 address format.

- The IPv4 address is returned as a string, such as '100.50.200.5’.

- If the host name is an IPv4 address itself it is returned unchanged.

- gethostbyname() does not support IPv6 name resolution. Use getaddrinfo() instead for

Lab 01

Lab

Questions