Sockets All Versions

Many programming languages use sockets to communicate across processes or between devices. This topic explains proper usage the the sockets module in Python to facilitate sending and receiving data over common networking protocols.

Examples

Multi-threaded TCP Socket Server

When run with no arguments, this program starts a TCP socket server that listens for connections to `127.0.0.1` on port `5000`. The server handles each connection in a separate thread.

When run with the `-c` argument, this program connects to the server, reads the client list, and prints it out. The client list is transferred as a JSON string. The client name may be specified by passing the `-n` argument. By passing different names, the effect on the client list may be observed.

client_list.py

```
import argparse
import json
import socket
import threading

def handle_client(client_list, conn, address):
    name = conn.recv(1024)
    entry = dict(zip(['name', 'address', 'port'], [name, address[0], address[1]]))
    client_list[name] = entry
    conn.sendall(json.dumps(client_list))
    conn.shutdown(socket.SHUT_RDWR)
    conn.close()

def server(client_list):
    print "Starting server..."
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.bind(('127.0.0.1', 5000))
    s.listen(5)
    while True:
        (conn, address) = s.accept()
        t = threading.Thread(target=handle_client, args=(client_list, conn, address))
        t.daemon = True
        t.start()

def client(name):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(('127.0.0.1', 5000))
    s.send(name)
    data = s.recv(1024)
    result = json.loads(data)
    print json.dumps(result, indent=4)

def parse_arguments():
    parser = argparse.ArgumentParser()
    parser.add_argument('-c', dest='client', action='store_true')
    parser.add_argument('-n', dest='name', type=str, default='name')
```

Server Output

```
$ python client_list.py
Starting server...
```

Client Output

```
$ python client_list.py -c -n name1
{
    "name1": {
        "address": "127.0.0.1",
        "port": 62210,
        "name": "name1"
    }
}
```

The receive buffers are limited to 1024 bytes. If the JSON string representation of the client list exceeds this size, it will be truncated. This will cause the following exception to be raised:

```
ValueError: Unterminated string starting at: line 1 column 1023 (char 1022)
```

Raw Sockets on Linux

First you disable your network card's automatic checksumming:

```
sudo ethtool -K eth1 tx off
```

Then send your packet, using a SOCK_RAW socket:

```
#!/usr/bin/env python
from socket import socket, AF_PACKET, SOCK_RAW
s = socket(AF_PACKET, SOCK_RAW)
s.bind(("eth1", 0))

# We're putting together an ethernet frame here,
# but you could have anything you want instead
# Have a look at the 'struct' module for more
# flexible packing/unpacking of binary data
# and 'binascii' for 32 bit CRC
src_addr = "\x01\x02\x03\x04\x05\x06"
dst_addr = "\x01\x02\x03\x04\x05\x06"
payload = ("["*30)+"PAYLOAD"+("]"*30)
checksum = "\x1a\x2b\x3c\x4d"
ethertype = "\x08\x01"

s.send(dst_addr+src_addr+ethertype+payload+checksum)
```

Receiving data via UDP

UDP is a connectionless protocol. This means that peers sending messages do not require establishing a connection before sending messages. `socket.recvfrom` thus returns a tuple (`msg` [the message the socket received], `addr` [the address of the sender])

A UDP server using solely the `socket` module:

```
from socket import socket, AF_INET, SOCK_DGRAM
sock = socket(AF_INET, SOCK_DGRAM)
sock.bind(('localhost', 6667))

while True:
    msg, addr = sock.recvfrom(8192)  # This is the amount of bytes to read at maximum
    print("Got message from %s: %s" % (addr, msg))
```

Below is an alternative implementation using `socketserver.UDPServer` :

```
from socketserver import BaseRequestHandler, UDPServer

class MyHandler(BaseRequestHandler):
    def handle(self):
        print("Got connection from: %s" % self.client_address)
        msg, sock = self.request
        print("It said: %s" % msg)
        sock.sendto("Got your message!".encode(), self.client_address) # Send reply

serv = UDPServer(('localhost', 6667), MyHandler)
serv.serve_forever()
```

By default, `sockets` block. This means that execution of the script will wait until the socket receives data.

Sending data via TCP

Sending data over the internet is made possible using multiple modules. The sockets module provides low-level access to the underlying Operating System operations responsible for sending or receiving data from other computers or processes.

The following code sends the byte string `b'Hello'` to a TCP server listening on port 6667 on the host localhost and closes the connection when finished:

```
from socket import socket, AF_INET, SOCK_STREAM
s = socket(AF_INET, SOCK_STREAM)
s.connect(('localhost', 6667))  # The address of the TCP server listening
s.send(b'Hello')
s.close()
```

Socket output is blocking by default, that means that the program will wait in the connect and send calls until the action is 'completed'. For connect that means the server actually accepting the connection. For send it only means that the operating system has enough buffer space to queue the data to be send later.

Sockets should always be closed after use

Sockets should always be closed after use.

Sending data via UDP

UDP is a connectionless protocol. Messages to other processes or computers are sent without establishing any sort of connection. There is no automatic confirmation if your message has been received. UDP is usually used in latency sensitive applications or in applications sending network wide broadcasts.

The following code sends a message to a process listening on localhost port 6667 using UDP

Note that there is no need to "close" the socket after the send, because UDP is connectionless .

```
from socket import socket, AF_INET, SOCK_DGRAM
s = socket(AF_INET, SOCK_DGRAM)
msg = ("Hello you there!").encode('utf-8')  # socket.sendto() takes bytes as input, hence we mus
s.sendto(msg, ('localhost', 6667))
```

Syntax

Parameters

Parameter	Description
socket.AF_UNIX	UNIX Socket
socket.AF_INET	IPv4
socket.AF_INET6	IPv6
socket.SOCK_STREAM	TCP
socket.SOCK_DGRAM	UDP

Remarks