User Datagram Client and Server

The user datagram protocol (UDP) works differently from TCP/IP. Where TCP is a *stream oriented* protocol, ensuring that all of the data is transmitted in the right order, UDP is a *message oriented* protocol. UDP does not require a long-lived connection, so setting up a UDP socket is a little simpler. On the other hand, UDP messages must fit within a single datagram (for IPv4, that means they can only hold 65,507 bytes because the 65,535 byte packet also includes header information) and delivery is not guaranteed as it is with TCP.

Echo Server

Since there is no connection, per se, the server does not need to listen for and accept connections. It only needs to use `bind()` to associate its socket with a port, and then wait for individual messages.

```
# socket_echo_server_dgram.py

import socket
import sys

# Create a UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind the socket to the port
server_address = ('localhost', 10000)
print('starting up on {} port {}'.format(*server_address))
sock.bind(server_address)

while True:
    print('\nwaiting to receive message')
    data, address = sock.recvfrom(4096)

    print('received {} bytes from {}'.format(
        len(data), address))
    print(data)

    if data:
        sent = sock.sendto(data, address)
        print('sent {} bytes back to {}'.format(
            sent, address))
```

Messages are read from the socket using `recvfrom()`, which returns the data as well as the address of the client from which it was sent.

Echo Client

The UDP echo client is similar the server, but does not use `bind()` to attach its socket to an address. It uses `sendto()` to deliver its message directly to the server, and `recvfrom()` to receive the response.

```
# socket_echo_client_dgram.py

import socket
import sys

# Create a UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

server_address = ('localhost', 10000)
message = b'This is the message.  It will be repeated.'

try:

    # Send data
```

```
    print('sending {!r}'.format(message))
    sent = sock.sendto(message, server_address)

    # Receive response
    print('waiting to receive')
    data, server = sock.recvfrom(4096)
    print('received {!r}'.format(data))

finally:
    print('closing socket')
    sock.close()
```

Client and Server Together

Running the server produces:

```
$ python3 socket_echo_server_dgram.py
starting up on localhost port 10000

waiting to receive message
received 42 bytes from ('127.0.0.1', 57870)
b'This is the message.  It will be repeated.'
sent 42 bytes back to ('127.0.0.1', 57870)

waiting to receive message
```

The client output is:

```
$ python3 socket_echo_client_dgram.py
sending b'This is the message.  It will be repeated.'
waiting to receive
received b'This is the message.  It will be repeated.'
closing socket
```

TCP/IP Client and Server

Unix Domain Sockets

Quick Links
Echo Server
Echo Client
Client and Server Together

This page was last updated 2017-01-01.

Navigation
TCP/IP Client and Server
Unix Domain Sockets

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

Module Index
Index

Other Writing

Blog
The Python Standard Library By Example