TCP/IP Client and Server¶

pymotw.com (http://getpocket.com
/redirect?url=http%3A%2F
%2Fpymotw.com%2F2%2Fsocket%2Ftcp.html)

February 21st, 2013

View Original

Echo Server¶ (http://pymotw.com/2/socket/tcp.html#echo-server)

This sample program, based on the one in the standard library documentation, receives incoming messages and echos them back to the sender. It starts by creating a TCP/IP socket.

```
import socket
import sys

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Then `bind()` is used to associate the socket with the server address. In this case, the address is `localhost`, referring to the current server, and the port number is 10000.

```
# Bind the socket to the port
server_address = ('localhost', 10000)
print >>sys.stderr, 'starting up on %s port %s' %
server_address
sock.bind(server_address)
```

Calling `listen()` puts the socket into server mode, and `accept()` waits for an incoming connection.

```
# Listen for incoming connections
sock.listen(1)

while True:
    # Wait for a connection
    print >>sys.stderr, 'waiting for a connection'
    connection, client_address = sock.accept()
```

`accept()` returns an open connection between the server and client, along with the address of the client. The connection is actually a different socket on another port (assigned by the kernel). Data is read from the connection with `recv()` and transmitted with `sendall()`.

```
    try:
        print >>sys.stderr, 'connection from',
client_address

        # Receive the data in small chunks and
retransmit it
        while True:
            data = connection.recv(16)
            print >>sys.stderr, 'received "%s"' % data
            if data:
                print >>sys.stderr, 'sending data back
to the client'
                connection.sendall(data)
            else:
                print >>sys.stderr, 'no more data
from', client_address
                break

    finally:
        # Clean up the connection
        connection.close()
```

When communication with a client is finished, the connection needs to be cleaned up using `close()`. This example uses a `try:finally` block to ensure that `close()` is always called, even in the event of an error.

Echo Client¶ (http://pymotw.com/2/socket/tcp.html#echo-client)

The client program sets up its `socket` (http://pymotw.com/2/socket/index.html#module-socket) differently from the way a server does. Instead of binding to a port and listening, it uses `connect()` to attach the socket directly to the remote address.

```
import socket
import sys

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect the socket to the port where the server is
listening
server_address = ('localhost', 10000)
print >>sys.stderr, 'connecting to %s port %s' %
server_address
sock.connect(server_address)
```

After the connection is established, data can be sent through the `socket` (http://pymotw.com/2/socket/index.html#module-socket) with `sendall()` and received with `recv()`, just as in the server.

```
try:

    # Send data
    message = 'This is the message.  It will be
repeated.'
    print >>sys.stderr, 'sending "%s"' % message
    sock.sendall(message)

    # Look for the response
    amount_received = 0
    amount_expected = len(message)

    while amount_received < amount_expected:
        data = sock.recv(16)
        amount_received += len(data)
        print >>sys.stderr, 'received "%s"' % data

finally:
    print >>sys.stderr, 'closing socket'
    sock.close()
```

When the entire message is sent and a copy received, the socket is closed to free up the port.

Client and Server Together¶ (http://pymotw.com/2/socket/tcp.html#client-and-server-together)

The client and server should be run in separate terminal windows, so they can communicate with each other. The server output is:

The client output is:

Easy Client Connections¶ (http://pymotw.com/2/socket

/tcp.html#easy-client-connections)

TCP/IP clients can save a few steps by using the convenience function `create_connection()` to connect to a server. The function takes one argument, a two-value tuple containing the address of the server, and derives the best address to use for the connection.

```
import socket
import sys

def get_constants(prefix):
    """Create a dictionary mapping socket module
constants to their names."""
    return dict( (getattr(socket, n), n)
                 for n in dir(socket)
                 if n.startswith(prefix)
                 )

families = get_constants('AF_')
types = get_constants('SOCK_')
protocols = get_constants('IPPROTO_')

# Create a TCP/IP socket
sock = socket.create_connection(('localhost', 10000))

print >>sys.stderr, 'Family  :', families[sock.family]
print >>sys.stderr, 'Type    :', types[sock.type]
print >>sys.stderr, 'Protocol:', protocols[sock.proto]
print >>sys.stderr

try:

    # Send data
    message = 'This is the message.  It will be
repeated.'
    print >>sys.stderr, 'sending "%s"' % message
    sock.sendall(message)

    amount_received = 0
    amount_expected = len(message)
    while amount_received < amount_expected:
        data = sock.recv(16)
        amount_received += len(data)
        print >>sys.stderr, 'received "%s"' % data

finally:
    print >>sys.stderr, 'closing socket'
    sock.close()
```

`create_connection()` uses `getaddrinfo()` to find candidate connection parameters, and returns a `socket` (http://pymotw.com/2/socket/index.html#module-socket) opened with the first configuration that creates a successful connection. The `family`, `type`, and `proto` attributes can be examined to determine the type of `socket` (http://pymotw.com/2/socket/index.html#module-socket) being returned.