Python Network Programming

CS 360 Internet Programming

Daniel Zappala

Brigham Young University
Computer Science Department

Why Use Python?

- provides direct access to the same socket API you use with C
  - simple, but powerful
  - socket addressing easier, buffer allocation done for you
  - many higher-level abstractions available
- because of all the other great features of Python (easy string parsing, simple threading, dynamic typing, overriding builtin methods), you can very quickly and easily build powerful network programs

Python Modules for Network Programs

http://docs.python.org/library/index.html

see Sections 17 - 20

httplib

- HTTP protocol client
  - implements the client side of HTTP and HTTPS
- HTTPConnection: make a connection to a server

```
connection = httplib.HTTPConnection(host)
connection.request(method,url,data,headers)
response = connection.getresponse()
```

- HTTPResponse: returned by a request

```
data = response.read()
status = response.status
reason = response.reason
```

httplib Example

- *see example code on web site*

urllib

- higher level interface to the Web
- `urlencode`: encode data as input to a GET or POST request, using appropriate escape codes as necessary

```
params = urllib.urlencode(dictionary)
```

- `urlopen`: opens a network object at the given URL, returns an object that acts like a file

```
f = urllib.urlopen(url)
response = f.read()
```

urllib Example

- *see example code on web site*

SimpleHTTPServer

- a basic web server
- `SimpleHTTPRequestHandler`: defines a request handler that serves files from the current directory and below
- `SocketServer`: defines basic TCP servers

```
handler = SimpleHTTPServer.SimpleHTTPRequestHandler
server = SocketServer.TCPServer((self.address,self.port),handler)
server.serve_forever()
```

SimpleHTTPServer Example

- *see example code on web site*

Server Socket API

1. create a socket
2. bind the socket to an address and port
3. listen for incoming connections
4. accept a client
5. send and receive data

Address Families

- **AF_UNIX**
  - communication between two processes on the same machine
  - represented as a string
- **AF_INET**
  - communication over the Internet, with IP version 4
  - represented as a tuple of *(host, port)*, *host* is a string host name, *port* is an integer port number
  - *host* can be a Internet host name (www.cnn.com) or an Ip address (64.236.24.20)
- **AF_INET6**
  - communication over the Internet, with IP version 6
  - represented using a tuple of *(host, port, flow_info, scope_id)*
    - *flow_info* is a flow identifier used for Quality of Service (e.g. low delay or guaranteed bandwidth)
    - *scope_id* is a scope identifier, which can limit packet delivery to various administrative boundaries

Create a Socket

```
socket(family,type[,protocol])
```

- returns a socket identifier
- *family* is AF_UNIX, AF_INET, or AF_INET6
- *type* is usually SOCK_STREAM for TCP, or SOCK_DGRAM for UDP
- *protocol* is ignored in most cases

```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
```

Bind the Socket

```
bind(address)
```

- *address* is a tuple defined by the address family

```
host = ''
port = 50000
s.bind((host,port))
```

- AF_INET is a (host,port) tuple
- setting host to the empty string tells the OS to use any address associated with the host
- port number must not be currently used, or else an exception is raised

Listen

```
listen(backlog)
```

- tells the server to listen for incoming connections
- *backlog* is an integer specifying the maximum number of connections the server will hold in a queue
- use a minimum of one, OS maximum is usually 5
- use threads to service the queue of connections quickly if service time for a connection is large

```
backlog = 5
s.listen(backlog)
```

Accept a Client

```
accept()
```

- returns a tuple *(socket,address)*
- *socket* is a new socket identifier for the client
- *address* is the client address, a tuple defined by the address family (host, port for AF_INET)

```
client,address = s.accept()
```

Client Socket API

1. create a socket
2. connect to the server
3. send and receive data

Connect to the Server

```
connect(address)
```

- *address* is a tuple defined by the address family

```
host = 'localhost'
port = 50000
s.connect((host,port))
```

- use a (host,port) tuple just like bind
- must use the address and port of the server, not the client
- using localhost means the server is running on the local machine – use an Internet host name or an IP address for a remote machine
- server must be listening for clients, or else an exception is raised

Sending Data

```
send(string[, flags])
```

- returns the number of bytes sent
- *string* is the data to be sent
- see Linux send man page for flags
- possible that some of the data is not sent – must check return value and resend if necessary

```
data = "Hello World"
client.send(data)
```

Receiving Data

```
recv(buffersize[, flags])
```

- returns a string representing the data received
- *buffersize* is the maximum size of the data to be received
- see Linux recv man page for flags
- possible that less data is received than the maximum

```
size = 1024
data = client.recv(size)
```

A Simple Echo Server

- *see example code on web site*
- see also exception handling in the code

Select Module

- allows an application to wait for input from multiple sockets at a time
  - does not use threads – multiplexes with kernel support
  - can interleave client requests
- polling methods
  - select - original UNIX system call (most OS)
  - poll - more improved system call (most OS)
  - epoll - Edge and Trigger Level Polling Objects (Linux)
  - kqueue - Kernel Queue Objects (BSD)
  - kevent - Kernel Event Objects (BSD)

Poll

```
poller = select.epoll()
```

- returns a polling object: supports registering and unregistering file descriptors and then polling them for I/O events

```
poller.register(fd,mask)
poller.unregister(fd)
fds = poller.poll(timeout)
```

Poll Example

- *see example code on web site*

Threaded Server Example

- *see example code on web site*
- warning – this code uses a thread per connection