13. Networking & Multithreading

Table of Contents

|  |  |  |
| --- | --- | --- |
| Sl. No | Topics | Page No |
| 13.1 | **Networking** | **2** |
| 13.2 | **Multithreading** | **9** |

## 13.1 Networking

**Network Programming**

* Python provides two levels of access to network services. At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols.
* Python also has libraries that provide higher-level access to specific application-level network protocols, such as FTP, HTTP, and so on.

**Socket Module**

**What are Sockets?**

* Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.
* Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on.
* The *socket* library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

**Terms used In Socket**

**Domain**

The family of protocols that is used as the transport mechanism. These values are constants such as AF\_INET, PF\_INET, PF\_UNIX, PF\_X25, and so on.

**type**

The type of communications between the two endpoints, typically SOCK\_STREAM for connection-oriented protocols and SOCK\_DGRAM for connectionless protocols.

**protocol**

Typically zero, this may be used to identify a variant of a protocol within a domain and type.

**port**

Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service.

**hostname**

The identifier of a network interface −

* + A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation
  + A string "<broadcast>", which specifies an INADDR\_BROADCAST address.
  + A zero-length string, which specifies INADDR\_ANY, or
  + An Integer, interpreted as a binary address in host byte order.

**Socket Module**

* To create a socket, you must use the socket.socket() function available in socket module, which has the general syntax - s = socket.socket (socket\_family, socket\_type, protocol=0)

Here is the description of the parameters -

* + socket\_family - This is either AF\_UNIX or AF\_INET, as explained earlier.
  + socket\_type - This is either SOCK\_STREAM or SOCK\_DGRAM.
  + protocol - This is usually left out, defaulting to 0.
* Once you have socket object, then you can use required functions to create your client or server program.

**Server Socket Methods**

**s.bind()**

This method binds address (hostname, port number pair) to socket.

**s.listen()**

This method sets up and start TCP listener.

**s.accept()**

This passively accept TCP client connection, waiting until connection arrives (blocking).

**Client Socket Methods**

**s.connect()**

This method actively initiates TCP server connection.

**General Socket Methods**

**s.recv()**

This method receives TCP message

**s.send()**

This method transmits TCP message

**s.recvfrom()**

This method receives UDP message

**s.sendto()**

This method transmits UDP message

**s.close()**

This method closes socket

**socket.gethostname()**

Returns the hostname.

**Server**

* To write Internet servers, we use the **socket** function available in socket module to create a socket object. A socket object is then used to call other functions to setup a socket server.
* Now call **bind(hostname, port)** function to specify a *port* for your service on the given host.
* Next, call the *accept* method of the returned object. This method waits until a client connects to the port you specified, and then returns a *connection* object that represents the connection to that client.

#!/usr/bin/python # This is server.py file

import socket # Import socket module s = socket.socket()

# Create a socket object host = socket.gethostname()

# Get local machine name port = 12345

# Reserve a port for your service. s.bind((host, port))

# Bind to the port

s.listen(5) # Now wait for client connection.

while True: c, addr = s.accept() # Establish connection with client.

print 'Got connection from', addr

c.send('Thank you for connecting')

c.close() # Close the connection

**Client**

* Let us write a very simple client program which opens a connection to a given port 12345 and given host. This is very simple to create a socket client using Python's socket module function.
* The socket.connect(hosname, port ) opens a TCP connection to hostname on the port. Once you have a socket open, you can read from it like any IO object. When done, remember to close it, as you would close a file.
* The following code is a very simple client that connects to a given host and port, reads any available data from the socket, and then exits -

#!/usr/bin/python

# This is client.py file import socket

# Import socket module s = socket.socket()

# Create a socket object host = socket.gethostname()

# Get local machine name

port = 12345 # Reserve a port for your service.

s.connect((host, port))

print s.recv(1024)

s.close() # Close the socket when done

**Program Execution**

Now run this server.py in background and then run above client.py to see the result.

**# Following would start a server in background.**

$ python server.py &

**# Once server is started run client as follows:**

$ python client.py

**This would produce following result −**

Got connection from ('127.0.0.1', 48437)

Thank you for connecting

**Sending Emails**

* Simple Mail Transfer Protocol (SMTP) is a protocol, which handles sending e-mail and routing e-mail between mail servers.
* Python provides **smtplib** module, which defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.
* Here is a simple syntax to create one SMTP object, which can later be used to send an e-mail −

import smtplib

smtpObj = smtplib.SMTP( [host [, port [, local\_hostname]]] )

Here is the detail of the parameters −

* host − This is the host running your SMTP server. You can specify IP address of the host or a domain name like tutorialspoint.com. This is optional argument.
* port − If you are providing host argument, then you need to specify a port, where SMTP server is listening. Usually this port would be 25.
* local\_hostname − If your SMTP server is running on your local machine, then you can specify just localhost as of this option.

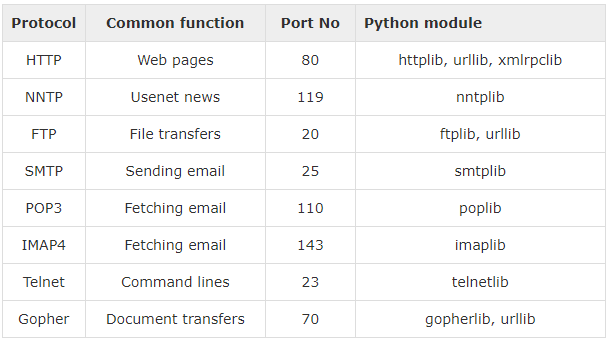
An SMTP object has an instance method called sendmail, which is typically used to do the work of mailing a message. It takes three parameters −

* The sender − A string with the address of the sender.
* The receivers − A list of strings, one for each recipient.
* The message − A message as a string formatted as specified in the various RFCs.
* To send an e-mail with mixed content requires to set **Content-type** header to **multipart/mixed**. Then, text and attachment sections can be specified within **boundaries**.
* A boundary is started with two hyphens followed by a unique number, which cannot appear in the message part of the e-mail. A final boundary denoting the e-mail's final section must also end with two hyphens.

Attached files should be encoded with the **pack("m")** function to have base64 encoding before transmission

**Python Internet Modules**

A list of some important modules in Python Network/Internet programming.



## 13.2 Multithreading

**What is Multithreading?**

Running several threads is similar to running several different programs concurrently, but with the following benefits −

* Multiple threads within a process share the same data space with the main thread and can therefore share information or communicate with each other more easily than if they were separate processes.
* Threads sometimes called light-weight processes and they do not require much memory overhead; they are cheaper than processes.

A thread has a beginning, an execution sequence, and a conclusion. It has an instruction pointer that keeps track of where within its context it is currently running.

* It can be pre-empted (interrupted)
* It can temporarily be put on hold (also known as sleeping) while other threads are running - this is called yielding.

**What is Threading Module?**

The newer threading module included with Python 2.4 provides much more powerful, high-level support for threads than the thread module discussed in the previous section.

The *threading* module exposes all the methods of the *thread* module and provides some additional methods −

* **threading.activeCount()** − Returns the number of thread objects that are active.
* **threading.currentThread()** − Returns the number of thread objects in the caller's thread control.
* **threading.enumerate()** − Returns a list of all thread objects that are currently active.

**What are the Methods of Thread Class?**

In addition to the methods, the threading module has the *Thread* class that implements threading. The methods provided by the *Thread* class are as follows −

* **run()** − The run() method is the entry point for a thread.
* **start()** − The start() method starts a thread by calling the run method.
* **join([time])** − The join() waits for threads to terminate.
* **isAlive()** − The isAlive() method checks whether a thread is still executing.
* **getName()** − The getName() method returns the name of a thread.
* **setName()** − The setName() method sets the name of a thread.

**Share the steps in Creating Thread Using Threading Module?**

To implement a new thread using the threading module, you have to do the following –

* Define a new subclass of the *Thread* class.
* Override the *\_\_init\_\_(self [,args])* method to add additional arguments.
* Then, override the run(self [,args]) method to implement what the thread should do when started.

Once you have created the new *Thread* subclass, you can create an instance of it and then start a new thread by invoking the *start()*, which in turn calls *run()* method.

**What is Thread Synchronization?**

* The threading module provided with Python includes a simple-to-implement locking mechanism that allows you to synchronize threads. A new lock is created by calling the *Lock ()* method, which returns the new lock.
* The *acquire(blocking)* method of the new lock object is used to force threads to run synchronously. The optional *blocking* parameter enables you to control whether the thread waits to acquire the lock.
* If *blocking* is set to 0, the thread returns immediately with a 0 value if the lock cannot be acquired and with a 1 if the lock was acquired. If blocking is set to 1, the thread blocks and wait for the lock to be released.
* The *release()* method of the new lock object is used to release the lock when it is no longer required.

**How about Priority Queue Multithread?**

The *Queue* module allows you to create a new queue object that can hold a specific number of items.

There are following methods to control the Queue −

* **get()** − The get() removes and returns an item from the queue.
* **put()** − The put adds item to a queue.
* **qsize()** − The qsize() returns the number of items that are currently in the queue.
* **empty()** − The empty( ) returns True if queue is empty; otherwise, False.
* **full()** − the full() returns True if queue is full; otherwise, False.

