NSSA 220
Task Automation with Interpreted Languages

Network Programming in Python

Instructor: Dr. Fahed Jubair

RIT DUBAI

Network Programming and Sockets

- Sockets are logical endpoints for communication between machines
- To develop a server-client application, we need to create two sockets, one at the server side and one at the client side
- Communication between sockets is bi-directional
- Python provides the Socket module for developing server-client applications

The Socket Module

- Syntax:

```
import socket
s = socket.socket(address_family, socket_type)
```

- Address family refers to the type of the internet address. By default, it is AF_INET, which is IPv4 addressing.
- Socket type refers to the communication protocol type, which can be either SOCK_STREAM (TCP/IP), or SOCK_DGRAM (UDP). SOCK_STREAM is the default type.

Socket Methods

Server-side methods

bind()	binds address (hostname, port number pair) to socket.
listen()	sets up and start TCP listener.
accept()	Waits for a connection request from the client

Client-side methods

connect()	initiates TCP server connection.

General methods

send()	transmits TCP message
recv()	receives TCP message
gethostname()	gets hostname
close()	closes socket

Example

```python
#server.py
import socket

host_name = socket.gethostname()
port = 12345
server_socket = socket.socket()
server_socket.bind((host_name,port))
server_socket.listen()
conn, addr = server_socket.accept()
print("connection from", str(addr))

while True:
  data = conn.recv(1024)
  if not data:
     break
  print("received data:", data.decode())
  msg = 'hello’
  conn.send(msg.encode())
conn.close()
```

```python
#client.py
import socket

host_name = socket.gethostname()
port = 12345

client_socket = socket.socket()
client_socket.connect((host_name,port))
msg = 'hello'
client_socket.send(msg.encode())

reply = client_socket.recv(1024)
print("reply is", reply.decode())

client_socket.close()
```

Note: from two different terminals, run the server program first, then run the client program

Another Example

```python
#server.py
import socket

host_name = socket.gethostname()
port = 12345
server_socket = socket.socket()
server_socket.bind((host_name,port))
server_socket.listen()
conn, addr = server_socket.accept()
print("connection from", str(addr))

data = conn.recv(1024)
print(data.decode())

conn.close()
```

```python
#client.py
import socket

host_name = socket.gethostname()
port = 12345

client_socket = socket.socket()
client_socket.connect((host_name,port))

msg = open('story.txt').read().strip()
client_socket.send(msg.encode())

client_socket.close()
```

HTTP

- Python has the following modules:
  - http.client (for implementing http protocols on client-side)
  - http.server (for implementing http protocols on server-side)
- Upon establishing an HTTP connection between a client and a server, the client can send a **request** to the server using either a GET method, or a POST method
  - GET requests are typically used to request data from the server
  - POST requests are typically used to send data to the server
- Upon receiving a request, a server will send a **response** to the client

HTTP Example with GET Method

```
import http.client

conn = http.client.HTTPSConnection("www.httpbin.org")
conn.request("GET","/")
response = conn.getresponse()
print(response.status, response.reason)
print(response.getheaders())
conn.close()



// output
200 OK
[('Date', 'Fri, 18 Nov 2022 17:25:13 GMT'), ('Content-Type', 'text/html;
charset=utf-8'), ('Content-Length', '9593'), ('Connection', 'keep-
alive'), ('Server', 'gunicorn/19.9.0'), ('Access-Control-Allow-Origin',
'*'), ('Access-Control-Allow-Credentials', 'true')]
```

HTTP Example with POST Method

```
import http.client
import json

conn = http.client.HTTPSConnection("www.httpbin.org")
headers_list = {'content-type':'application/json'}
post_text = {'text':'Hello World !!'}
json_data = json.dumps(post_text)
conn.request("POST","/post",json_data, headers_list)
response = conn.getresponse()
print(response.read().decode())
conn.close()
```

See output in next slide

HTTP Example with POST Method Output

```
{
  "args": {},
  "data": "{\"text\": \"Hello World !!\"}",
  "files": {},
  "form": {},
  "headers": {
    "Accept-Encoding": "identity",
    "Content-Length": "26",
    "Content-Type": "application/json",
    "Host": "www.httpbin.org",
    "X-Amzn-Trace-Id": "Root=1-6377d21c-6f8ffce679c3b4c70b9b8cf4"
  },
  "json": {
    "text": "Hello World !!"
  },
  "origin": "92.253.28.169",
  "url": "https://www.httpbin.org/post"
}
```

HTTP Server

- The following command can be used to run the http server on the designated port. If no port is specified, then port 8000 is used.

```
$ python –m http.server 9000
```

- Now, go to the browser and type “localhost:9000” in the address bar

The urllib Package

- urllib is a package in Python that is used for accessing websites, as well as downloading and parsing data from websites

- Some of the following modules in urllib are the following:
  - urllib.request: used for opening and reading URLs
  - urllib.error: contains exceptions raised by urllib.request
  - urllib.parse: used for parsing URLs

urllib Example 1

```python
import urllib.request
response = urllib.request.urlopen("https://www.httpbin.org/")

print(response.read())
print(response.info())
print(type(response))

response.close()
```

Let us run the program to see the output

urllib Example 2

```python
import urllib.request
import urllib.parse
url = 'https://www.google.com/search'
values = {'query' : 'python tutorial'}

data = urllib.parse.urlencode(values)
data = data.encode('utf-8') # data should be bytes
req = urllib.request.Request(url, data)
resp = urllib.request.urlopen(req)
respData = resp.read()
print(respData)
```

Summary

- We learned how to develop simple client-server application in python using sockets over TCP/IP
- In addition to sockets, Python provides other useful packages and modules (such as *http* and *urllib*) for requesting and sending data to websites and http servers