## Programming with Python (Multiplexing sockets I/O modules):

## **[11]Write a simple web server that can return a single line/multiple line of text to any connected web browser.**

## **CODE -**

## import socket

## s = socket.socket()

## host = socket.getfqdn()

## port = 9081

## s.bind((host, port))

## print 'Starting server on', host, port

## print 'The Web server URL for this would be http://%s:%d/' % (host, port)

## s.listen(5)

## print 'Entering infinite loop; Terminate manually to exit'

## track = dict()

## while True:

## c, (client\_host, client\_port) = s.accept()

## track[client\_host] = track.get(client\_host, 0) + 1

## c.recv(1000)

## c.send('HTTP/1.0 200 OK\n')

## c.send('Content-Type: text/html\n')

## c.send('\n')

## c.send("""

## <html>

## <body>

## <h1>Hello.</h1> Server Address : """+host+"""

## </body>

## </html>

## """)

## c.close()

## print 'Successfully Connected. '

## 

## 

## **[12]Write an efficient chart server that can handle several hundred or a large number of client connections. The chart server initializes with a few data attributes. It stores the count of clients, map of each client, and output sockets. The chart client initializes with a name argument and sends this name to the chart server.**

## **CODE -**

## import select

## import socket

## import sys

## import signal

## import pickle

## import struct

## import argparse

## SERVER\_HOST = 'localhost'

## CHAT\_SERVER\_NAME = 'server'

## # Some utilities

## def send(channel, \*args):

## buffer = pickle.dumps(args)

## value = socket.htonl(len(buffer))

## size = struct.pack("L",value)

## channel.send(size)

## channel.send(buffer)

## def receive(channel):

## size = struct.calcsize("L")

## size = channel.recv(size)

## try:

## size = socket.ntohl(struct.unpack("L", size)[0])

## except struct.error as e:

## return ''

## buf = ""

## while len(buf) < size:

## buf = channel.recv(size - len(buf))

## return pickle.loads(buf)[0]

## class ChatServer(object):

## """ An example chat server using select """

## def \_\_init\_\_(self, port, backlog=5):

## self.clients = 0

## self.clientmap = {}

## self.outputs = [] # list output sockets

## self.server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

## self.server.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

## self.server.bind((SERVER\_HOST, port))

## print ('Server listening to port: %s ...' %port)

## self.server.listen(backlog)

## # Catch keyboard interrupts

## signal.signal(signal.SIGINT, self.sighandler)

## 

## def sighandler(self, signum, frame):

## """ Clean up client outputs"""

## # Close the server

## print ('Shutting down server...')

## # Close existing client sockets

## for output in self.outputs:

## output.close()

## self.server.close()

## def get\_client\_name(self, client):

## """ Return the name of the client """

## info = self.clientmap[client]

## host, name = info[0][0], info[1]

## return '@'.join((name, host))

## 

## def run(self):

## inputs = [self.server, sys.stdin]

## self.outputs = []

## running = True

## while running:

## try:

## readable, writeable, exceptional = select.select(inputs, self.outputs, [])

## except select.error as e:

## break

## for sock in readable:

## if sock == self.server:

## # handle the server socket

## client, address = self.server.accept()

## print ("Chat server: got connection %d from %s" % (client.fileno(), address))

## # Read the login name

## cname = receive(client).split('NAME: ')[1]

## 

## # Compute client name and send back

## self.clients += 1

## send(client, 'CLIENT: ' + str(address[0]))

## inputs.append(client)

## self.clientmap[client] = (address, cname)

## # Send joining information to other clients

## msg = "\n(Connected: New client (%d) from %s)" % (self.clients, self.get\_client\_name(client))

## for output in self.outputs:

## send(output, msg)

## self.outputs.append(client)

## elif sock == sys.stdin:

## # handle standard input

## junk = sys.stdin.readline()

## running = False

## else:

## # handle all other sockets

## try:

## data = receive(sock)

## if data:

## # Send as new client's message...

## msg = '\n#[' + self.get\_client\_name(sock) + ']>>' + data

## # Send data to all except ourself

## for output in self.outputs:

## if output != sock:

## send(output, msg)

## else:

## print ("Chat server: %d hung up" % sock.fileno())

## self.clients -= 1

## sock.close()

## inputs.remove(sock)

## self.outputs.remove(sock)

## # Sending client leaving information to others

## msg = "\n(Now hung up: Client from %s)" % self.get\_client\_name(sock)

## for output in self.outputs:

## send(output, msg)

## except socket.error as e:

## # Remove

## inputs.remove(sock)

## self.outputs.remove(sock)

## self.server.close()

## class ChatClient(object):

## """ A command line chat client using select """

## def \_\_init\_\_(self, name, port, host=SERVER\_HOST):

## self.name = name

## self.connected = False

## self.host = host

## self.port = port

## # Initial prompt

## self.prompt='[' + '@'.join((name, socket.gethostname().split('.')[0])) + ']> '

## # Connect to server at port

## try:

## self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

## self.sock.connect((host, self.port))

## print ("Now connected to chat server@ port %d" % self.port)

## self.connected = True

## # Send my name...

## send(self.sock,'NAME: ' + self.name)

## data = receive(self.sock)

## # Contains client address, set it

## addr = data.split('CLIENT: ')[1]

## self.prompt = '[' + '@'.join((self.name, addr)) + ']> '

## except socket.error as e:

## print ("Failed to connect to chat server @ port %d" % self.port)

## sys.exit(1)

## def run(self):

## """ Chat client main loop """

## while self.connected:

## try:

## sys.stdout.write(self.prompt)

## sys.stdout.flush()

## # Wait for input from stdin and socket

## readable, writeable,exceptional = select.select([0, self.sock], [],[])

## for sock in readable:

## if sock == 0:

## data = sys.stdin.readline().strip()

## if data: send(self.sock, data)

## elif sock == self.sock:

## data = receive(self.sock)

## if not data:

## print ('Client shutting down.')

## self.connected = False

## break

## else:

## sys.stdout.write(data + '\n')

## sys.stdout.flush()

## 

## except KeyboardInterrupt:

## print (" Client interrupted. """)

## self.sock.close()

## break

## if \_\_name\_\_ == "\_\_main\_\_":

## parser = argparse.ArgumentParser(description='Chat Server')

## parser.add\_argument('--name', action="store", dest="name", required=True)

## parser.add\_argument('--port', action="store", dest="port", type=int, required=True)

## given\_args = parser.parse\_args()

## port = given\_args.port

## name = given\_args.name

## if name == CHAT\_SERVER\_NAME:

## server = ChatServer(port)

## server.run()

## else:

## client = ChatClient(name=name, port=port)

## client.run()

## 

## **SERVER**

## 

## **CLIENT 1**

## **CLIENT 2**

## 

## **[13]Write program for local port forwarder, that will redirect all traffic from a local port to a particular remote host?**

## **CODE -**

## import argparse

## import asyncore

## import socket

## LOCAL\_SERVER\_HOST = 'localhost'

## REMOTE\_SERVER\_HOST = 'www.nitrkl.ac.in'

## BUFSIZE = 4096

## class PortForwarder(asyncore.dispatcher):

## def \_\_init\_\_(self, ip, port, remoteip,remoteport,backlog=5):

## asyncore.dispatcher.\_\_init\_\_(self)

## self.remoteip=remoteip

## self.remoteport=remoteport

## self.create\_socket(socket.AF\_INET,socket.SOCK\_STREAM)

## self.set\_reuse\_addr()

## self.bind((ip,port))

## self.listen(backlog)

## def handle\_accept(self):

## conn, addr = self.accept()

## print ("Connected to:",addr)

## Sender(Receiver(conn),self.remoteip,self.remoteport)

## class Receiver(asyncore.dispatcher):

## def \_\_init\_\_(self,conn):

## asyncore.dispatcher.\_\_init\_\_(self,conn)

## self.from\_remote\_buffer=''

## self.to\_remote\_buffer=''

## self.sender=None

## def handle\_connect(self):

## pass

## def handle\_read(self):

## read = self.recv(BUFSIZE)

## self.from\_remote\_buffer += read

## def writable(self):

## return (len(self.to\_remote\_buffer) > 0)

## def handle\_write(self):

## sent = self.send(self.to\_remote\_buffer)

## self.to\_remote\_buffer = self.to\_remote\_buffer[sent:]

## def handle\_close(self):

## self.close()

## if self.sender:

## self.sender.close()

## class Sender(asyncore.dispatcher):

## def \_\_init\_\_(self, receiver, remoteaddr,remoteport):

## asyncore.dispatcher.\_\_init\_\_(self)

## self.receiver=receiver

## receiver.sender=self

## self.create\_socket(socket.AF\_INET, socket.SOCK\_STREAM)

## self.connect((remoteaddr, remoteport))

## 

## def handle\_connect(self):

## pass

## 

## def handle\_read(self):

## read = self.recv(BUFSIZE)

## self.receiver.to\_remote\_buffer += read

## 

## def writable(self):

## return (len(self.receiver.from\_remote\_buffer) > 0)

## 

## def handle\_write(self):

## sent = self.send(self.receiver.from\_remote\_buffer)

## self.receiver.from\_remote\_buffer = self.receiver.from\_remote\_buffer[sent:]

## 

## def handle\_close(self):

## self.close()

## self.receiver.close()

## if \_\_name\_\_ == "\_\_main\_\_":

## parser = argparse.ArgumentParser(description='Local Port Forwarder.')

## parser.add\_argument('--local-host', action="store", dest="local\_host", default=LOCAL\_SERVER\_HOST)

## parser.add\_argument('--local-port', action="store", dest="local\_port", type=int, required=True)

## parser.add\_argument('--remote-host', action="store", dest="remote\_host", default=REMOTE\_SERVER\_HOST)

## parser.add\_argument('--remote-port', action="store", dest="remote\_port", type=int, default=80)

## given\_args = parser.parse\_args()

## local\_host, remote\_host = given\_args.local\_host, given\_args.remote\_host

## local\_port, remote\_port = given\_args.local\_port, given\_args.remote\_port

## 

## print ("Starting port forwarding local %s:%s => remote %s:%s" % (local\_host, local\_port, remote\_host, remote\_port))

## PortForwarder(local\_host, local\_port, remote\_host, remote\_port)

## asyncore.loop()

## 

## **[14] Write a client that will wait for a particular network service forever or for a time out?**

## **CODE -**

## import argparse

## import socket

## import errno

## from time import time as now

## DEFAULT\_TIMEOUT = 120

## DEFAULT\_SERVER\_HOST = 'localhost'

## DEFAULT\_SERVER\_PORT = 80

## class NetServiceChecker(object):

## """ Wait for a network service to come online"""

## def \_\_init\_\_(self, host, port, timeout=DEFAULT\_TIMEOUT):

## self.host = host

## self.port = port

## self.timeout = timeout

## self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

## 

## def end\_wait(self):

## self.sock.close()

## def check(self):

## """ Check the service """

## if self.timeout:

## end\_time = now() + self.timeout

## 

## while True:

## try:

## if self.timeout:

## next\_timeout = end\_time - now()

## if next\_timeout < 0:

## return False

## else:

## print ("setting socket next timeout %ss" %round(next\_timeout))

## self.sock.settimeout(next\_timeout)

## self.sock.connect((self.host, self.port))

## # handle exceptions

## except socket.timeout as err:

## if self.timeout:

## return False

## except socket.error as err:

## print ("Exception: %s" %err)

## else: # if all goes well

## self.end\_wait()

## return True

## if \_\_name\_\_ == '\_\_main\_\_':

## parser = argparse.ArgumentParser(description='Waiting for remote server')

## parser.add\_argument('--host', action="store", dest="host", default=DEFAULT\_SERVER\_HOST)

## parser.add\_argument('--port', action="store", dest="port", type=int, default=DEFAULT\_SERVER\_PORT)

## parser.add\_argument('--timeout', action="store", dest="timeout", type=int, default=DEFAULT\_TIMEOUT)

## given\_args = parser.parse\_args()

## host, port, timeout = given\_args.host, given\_args.port, given\_args.timeout

## service\_checker = NetServiceChecker(host, port, timeout=timeout)

## print ("Checking for network service %s:%s ..." %(host, port))

## if service\_checker.check():

## print ("Service is available again!")

## **APACHE SERVER IS RUNNING**

## **APACHE SERVER IS STOPPED**

## 

## **[15]Write a program to list the network interfaces present in your machine?**

## **CODE -**

## import os

## print('List of network interfaces : ')

## print(os.listdir('/sys/class/net/'))

## 