Computer Network Laboratory

Basic Applications of Network Programming (II)

Jiawei Chang

Dept. of Computer Science and Information Engineering

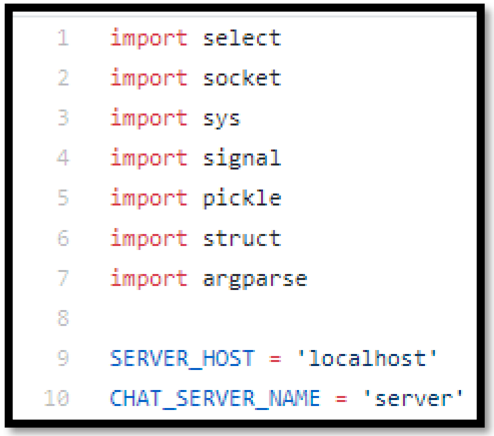
National Taichung University of Science and Technology

Outline

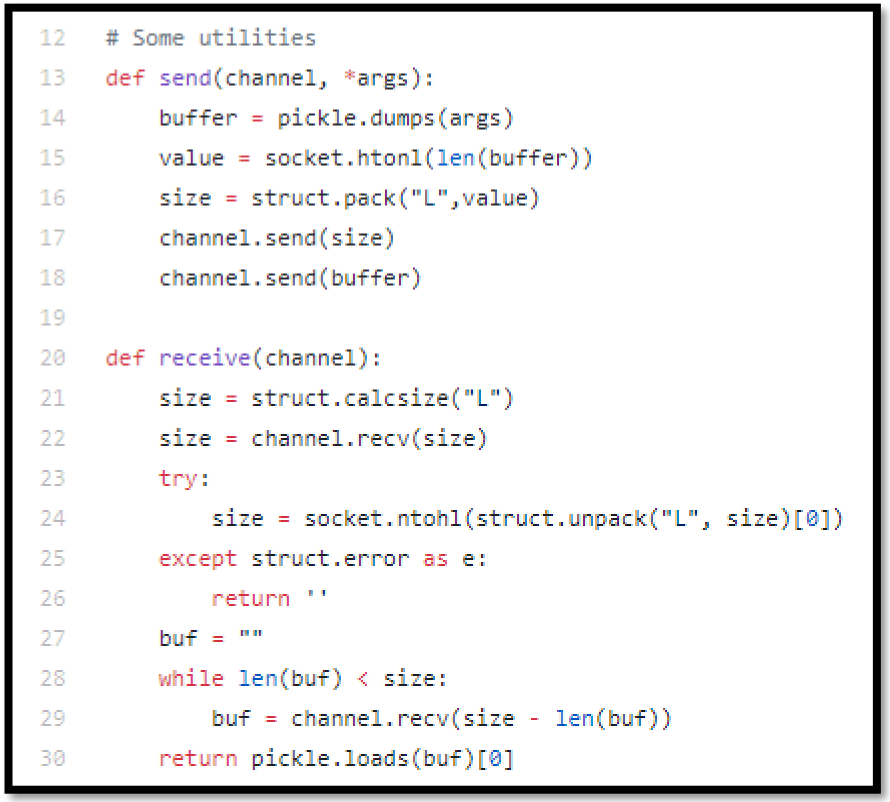
1. chat_server_with_select
2. wait_for_remote_service
3. ipc_using_socketpairs
4. unix_domain_socket_server
5. unix_domain_socket_client

CHAT_SERVER_WITH_SELECT

chat_server_with_select



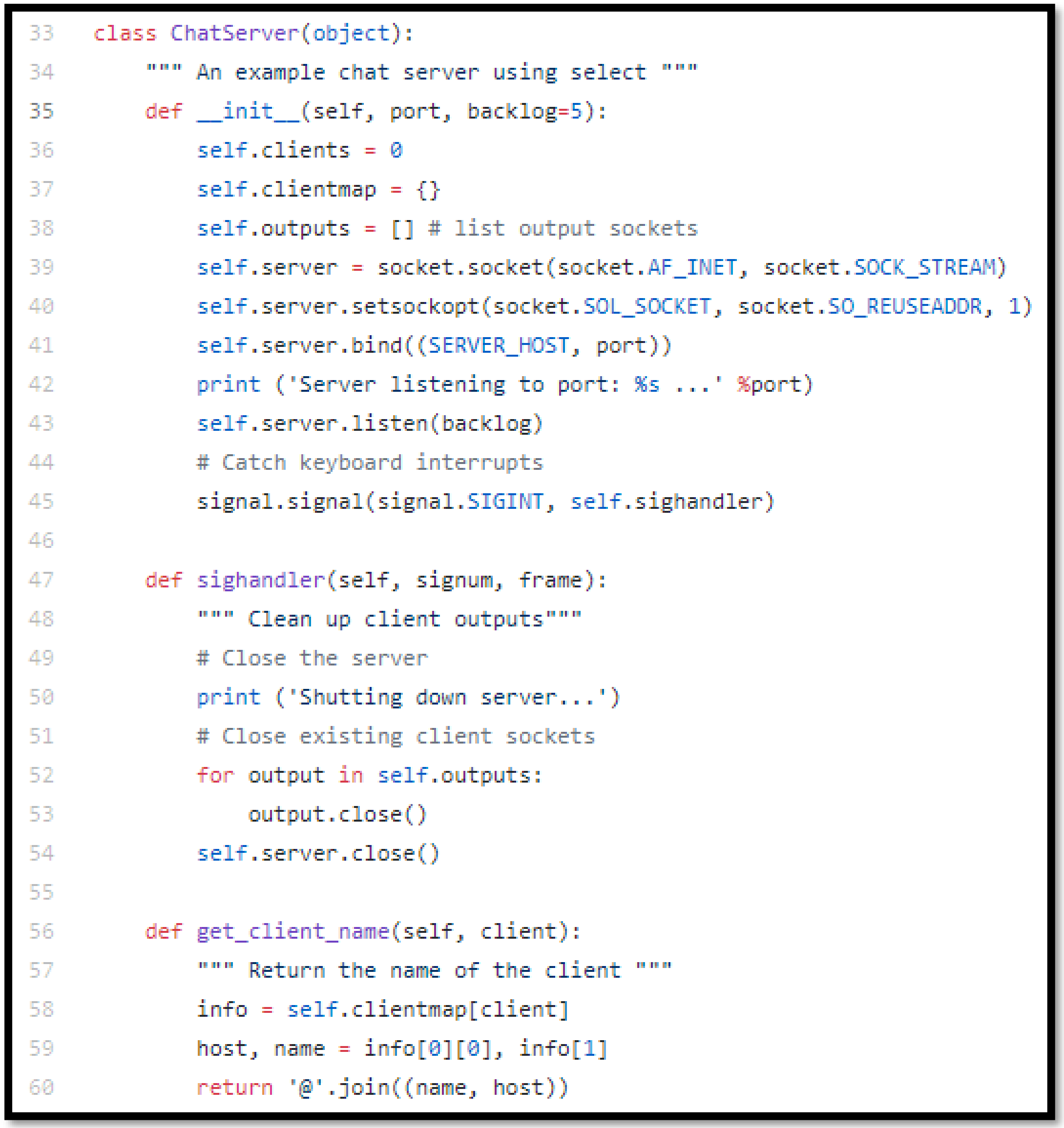
```python
import select
import socket
import sys
import signal
import pickle
import struct
import argparse

SERVER_HOST = 'localhost'
CHAT_SERVER_NAME = 'server'
```



```python
# 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]
```

chat_server_with_select

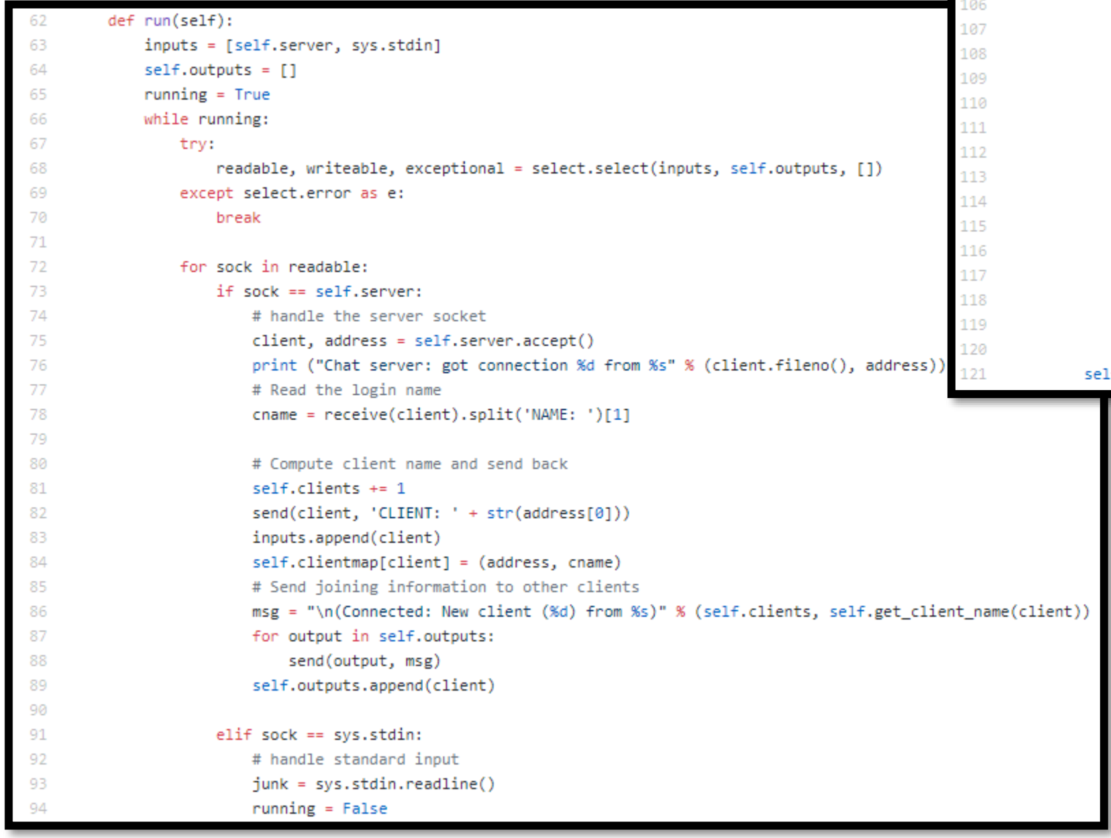
Server Side



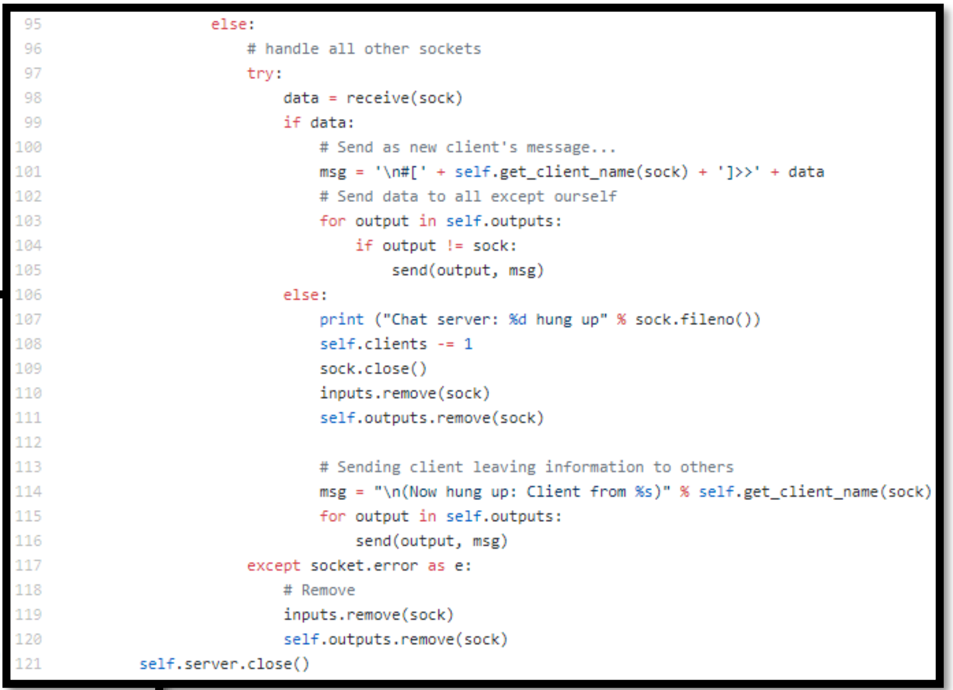
```python
class ChatServer(object):
    """ An example chat server using select """
    def __init__(self, port, backlog=5):
        self.clients = 0
        self.clientmap = {}
        self.outputs = [] # list output sockets
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        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))
```

chat_server_with_select

Server Side



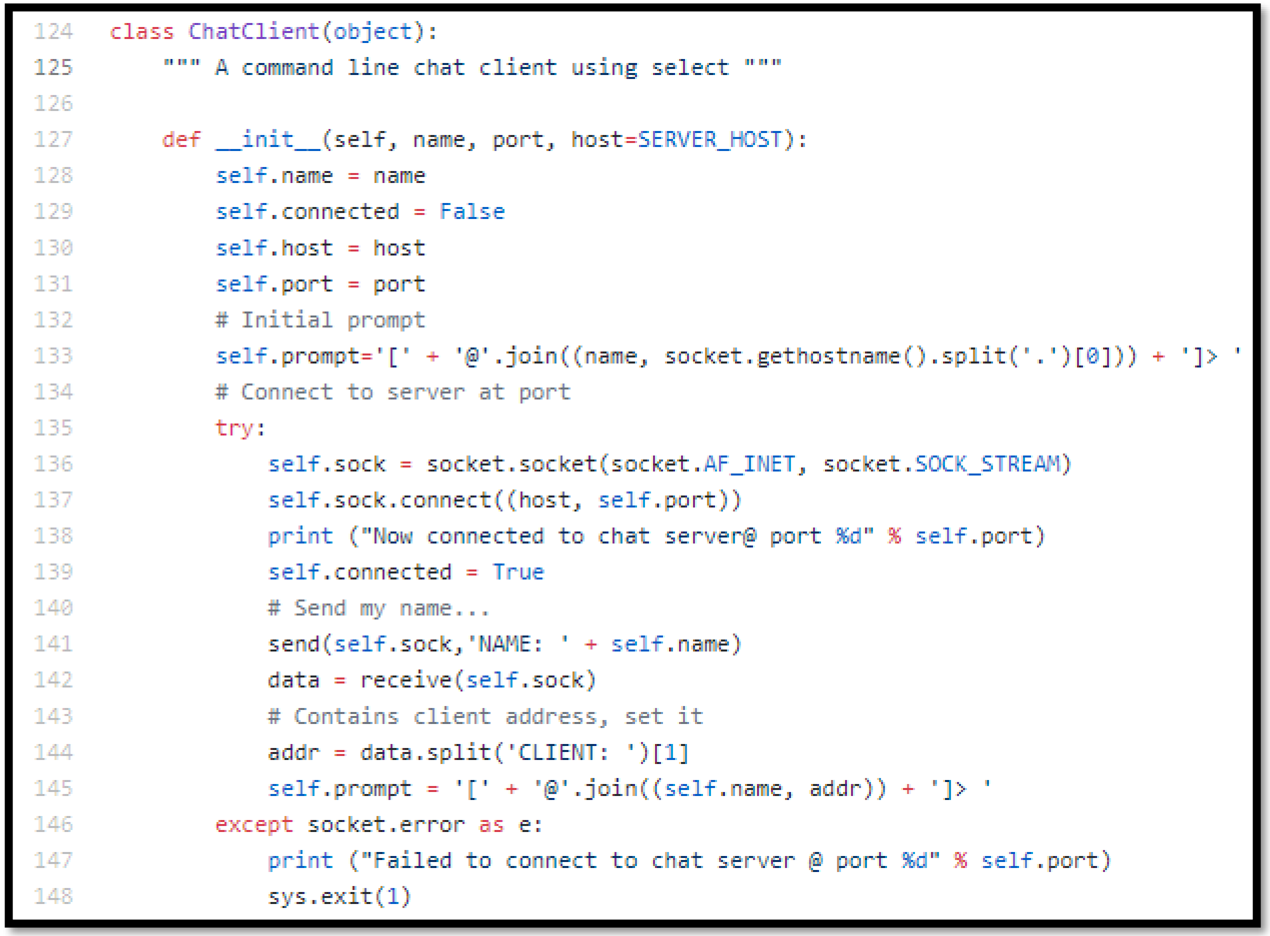
```python
    def run(self):
        inputs = [self.server, sys.stdin]
        self.outputs = []
        running = True
        while running:
            try:
                readable, writeable, exceptional = select.select(inputs, self.outputs, [])
            except select.error as e:
                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
```



```python
                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()
```

chat_server_with_select

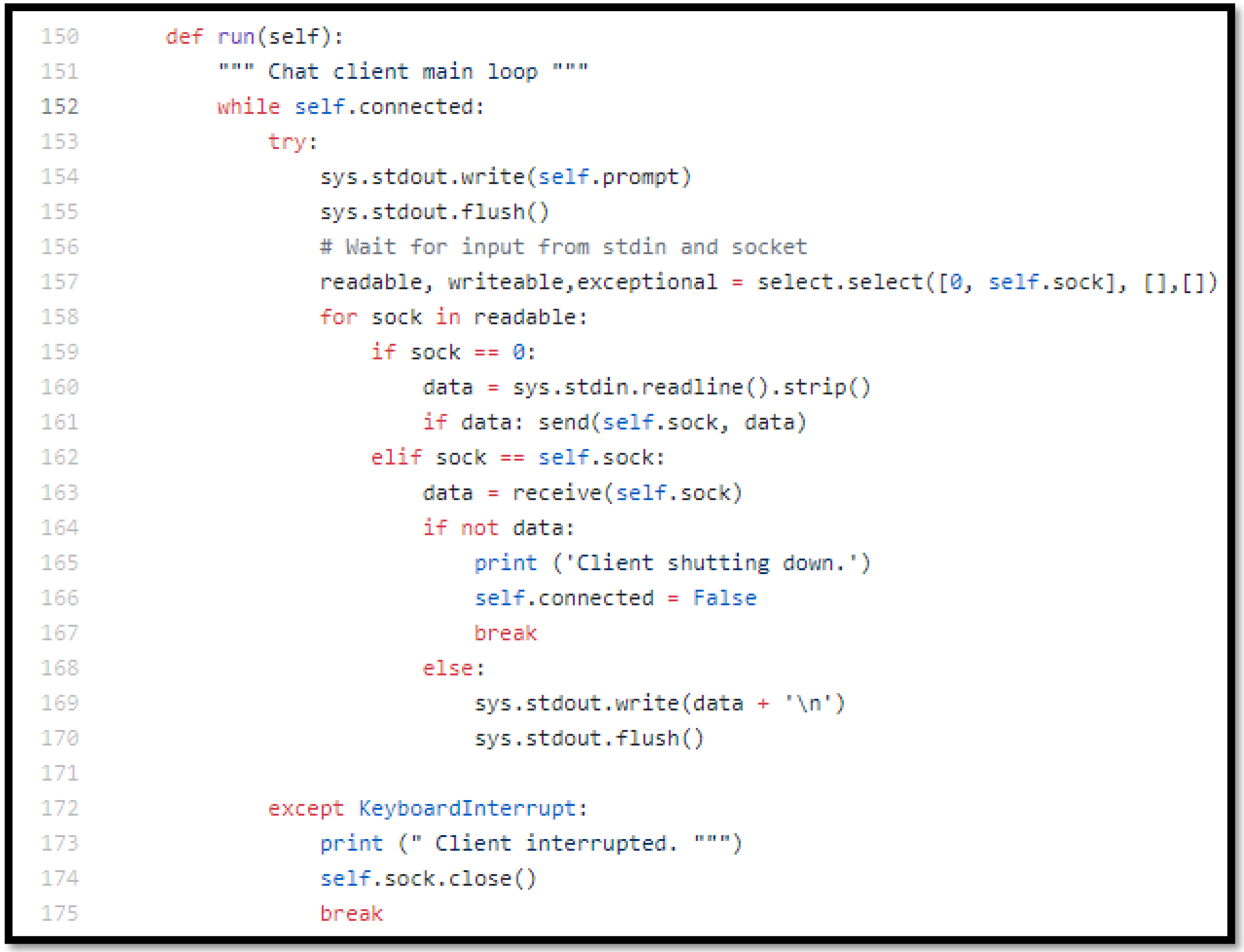
Client Side



```
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)
```

chat_server_with_select

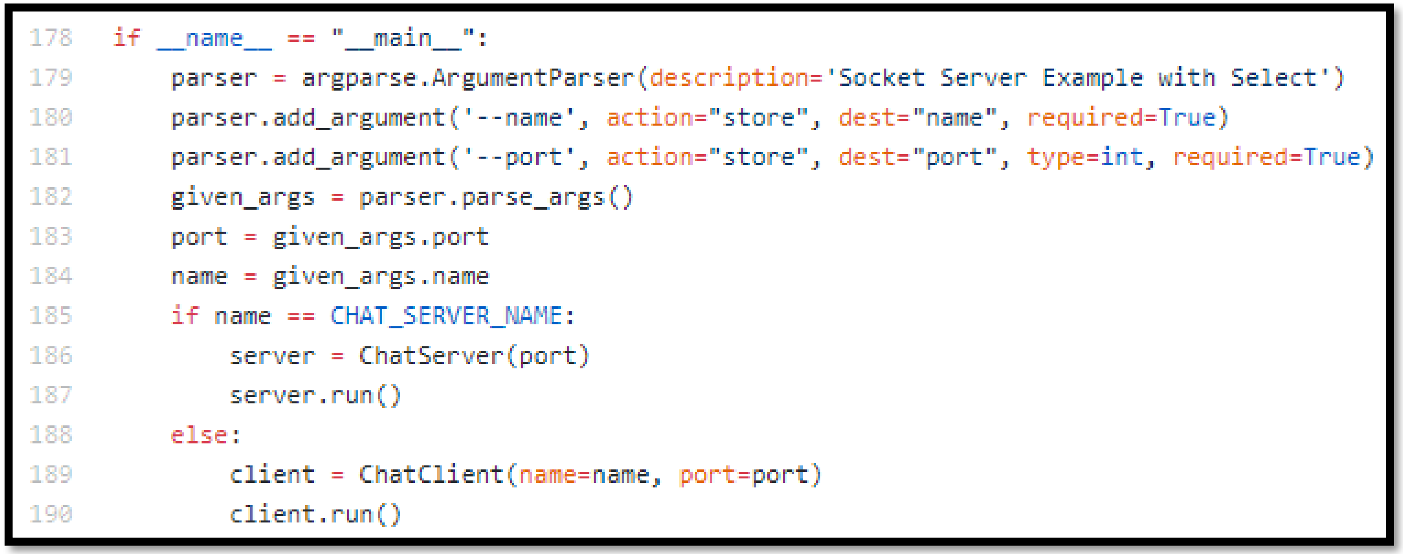
Client Side



```
    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
```

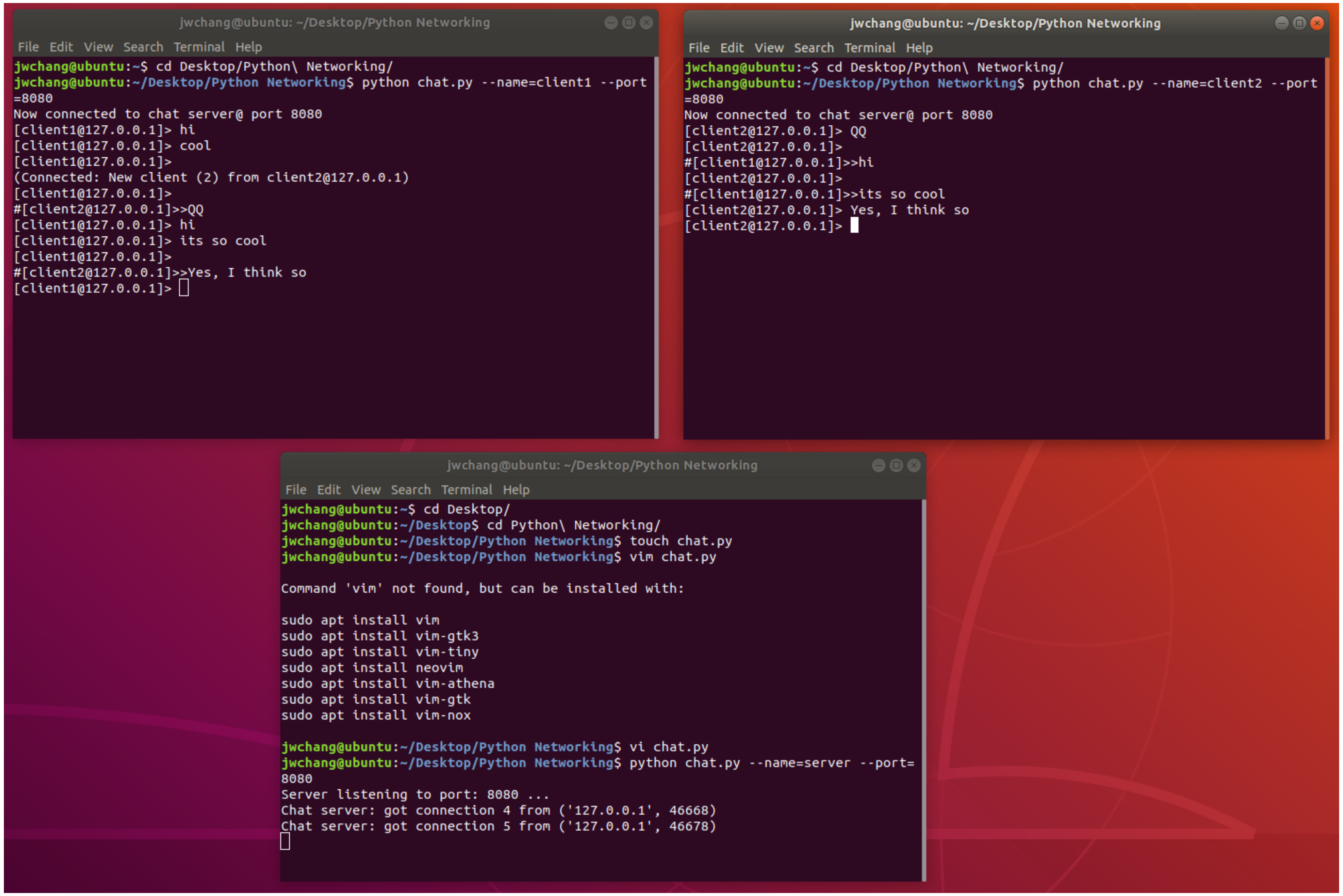
chat_server_with_select

Main



```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Socket Server Example with Select')
    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()
```

chat_server_with_select



jwchang@ubuntu: ~/Desktop/Python Networking
File Edit View Search Terminal Help

```
jwchang@ubuntu:~$ cd Desktop/Python\ Networking/
jwchang@ubuntu:~/Desktop/Python Networking$ python chat.py --name=client1 --port
=8080
Now connected to chat server@ port 8080
[client1@127.0.0.1]> hi
[client1@127.0.0.1]> cool
[client1@127.0.0.1]>
(Connected: New client (2) from client2@127.0.0.1)
[client1@127.0.0.1]>
#[client2@127.0.0.1]>>QQ
[client1@127.0.0.1]> hi
[client1@127.0.0.1]> its so cool
[client1@127.0.0.1]>
#[client2@127.0.0.1]>>Yes, I think so
[client1@127.0.0.1]>
```

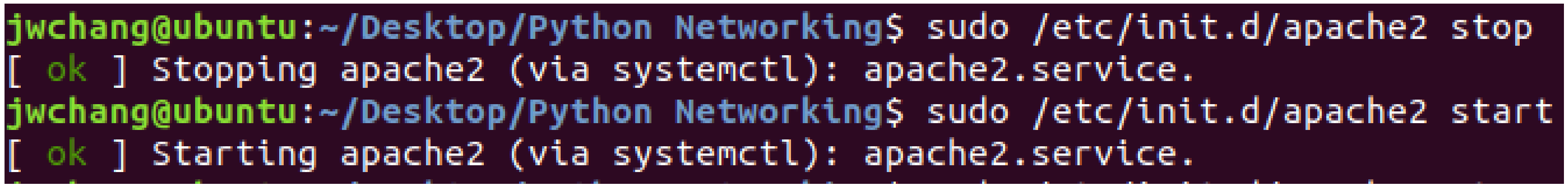
```
jwchang@ubuntu:~$ cd Desktop/Python\ Networking/
jwchang@ubuntu:~/Desktop/Python Networking$ python chat.py --name=client2 --port
=8080
Now connected to chat server@ port 8080
[client2@127.0.0.1]> QQ
[client2@127.0.0.1]>
#[client1@127.0.0.1]>>hi
[client2@127.0.0.1]>
#[client1@127.0.0.1]>>its so cool
[client2@127.0.0.1]> Yes, I think so
[client2@127.0.0.1]>
```

```
jwchang@ubuntu:~$ cd Desktop/
jwchang@ubuntu:~/Desktop$ cd Python\ Networking/
jwchang@ubuntu:~/Desktop/Python Networking$ touch chat.py
jwchang@ubuntu:~/Desktop/Python Networking$ vim chat.py

Command 'vim' not found, but can be installed with:

sudo apt install vim
sudo apt install vim-gtk3
sudo apt install vim-tiny
sudo apt install neovim
sudo apt install vim-athena
sudo apt install vim-gtk
sudo apt install vim-nox

jwchang@ubuntu:~/Desktop/Python Networking$ vi chat.py
jwchang@ubuntu:~/Desktop/Python Networking$ python chat.py --name=server --port=
8080
Server listening to port: 8080 ...
Chat server: got connection 4 from ('127.0.0.1', 46668)
Chat server: got connection 5 from ('127.0.0.1', 46678)
```

WAIT_FOR_REMOTE_SERVICE

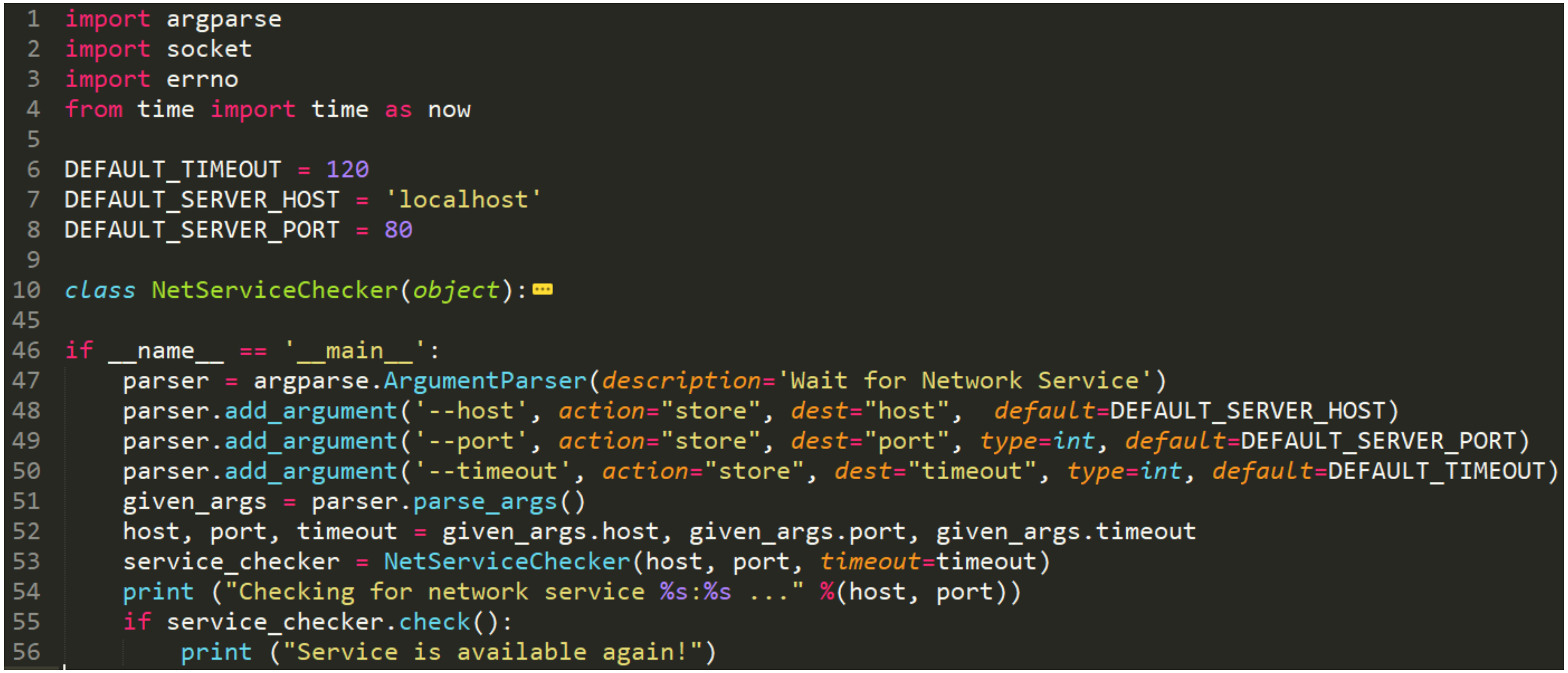
wait_for_remote_service

1. sudo apt install apache2
2. sudo /etc/init.d/apache2 stop
3. sudo /etc/init.d/apache2 start



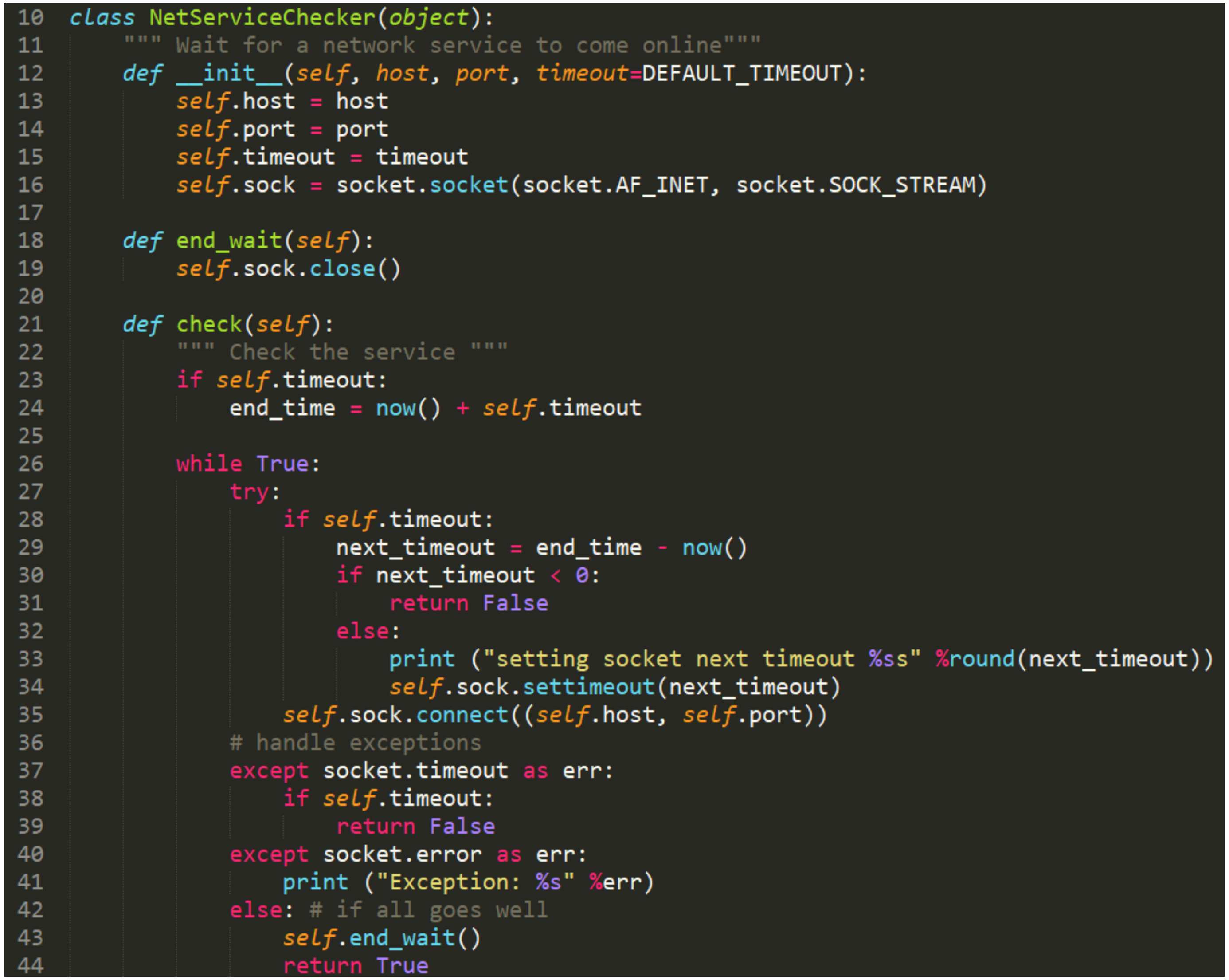
```
jwchang@ubuntu:~/Desktop/Python Networking$ sudo /etc/init.d/apache2 stop
[ ok ] Stopping apache2 (via systemctl): apache2.service.
jwchang@ubuntu:~/Desktop/Python Networking$ sudo /etc/init.d/apache2 start
[ ok ] Starting apache2 (via systemctl): apache2.service.
```

wait_for_remote_service



```python
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):

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Wait for Network Service')
    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!")
```

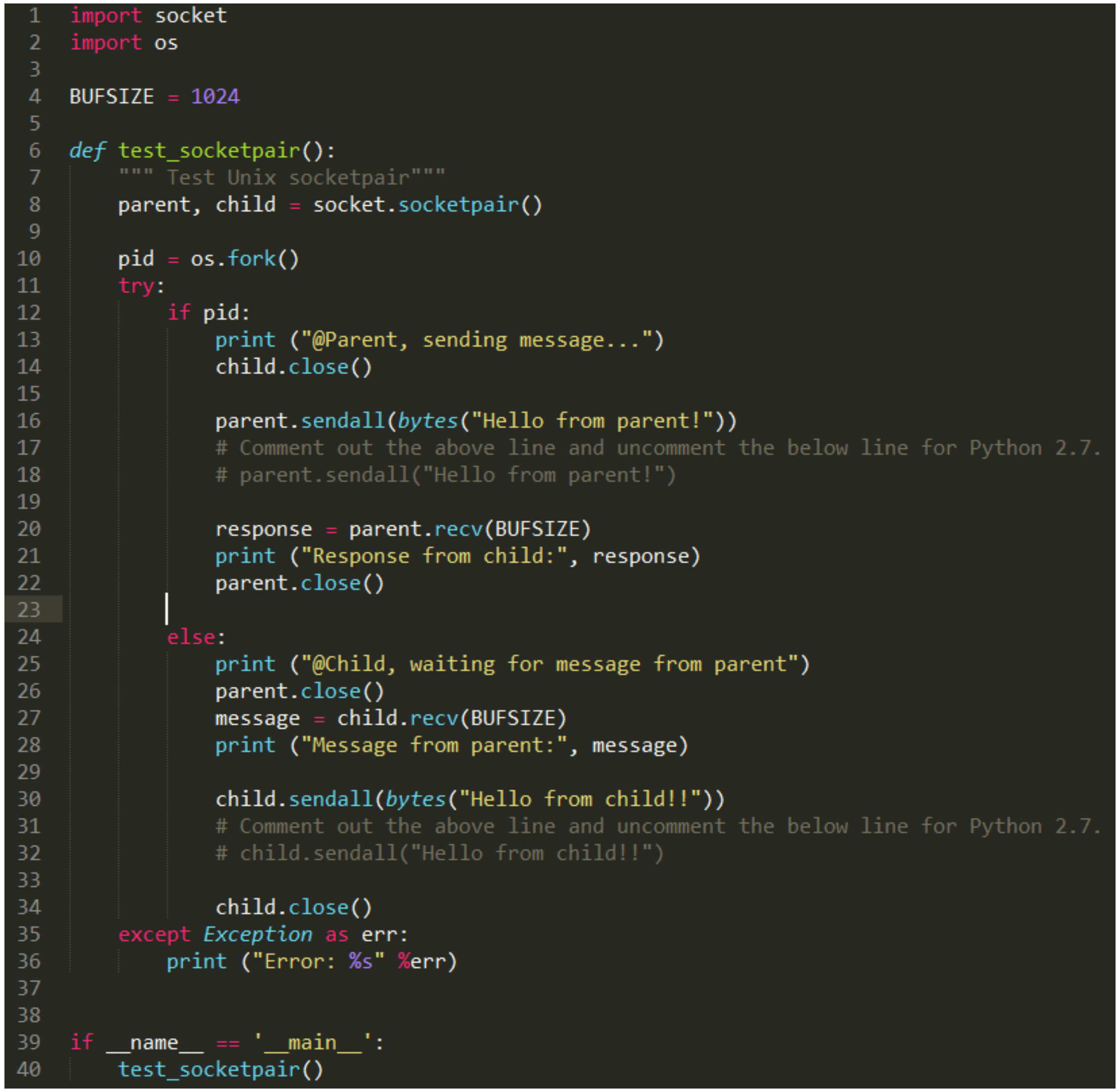
wait_for_remote_service



```python
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
```

IPC_USING_SOCKETPAIRS

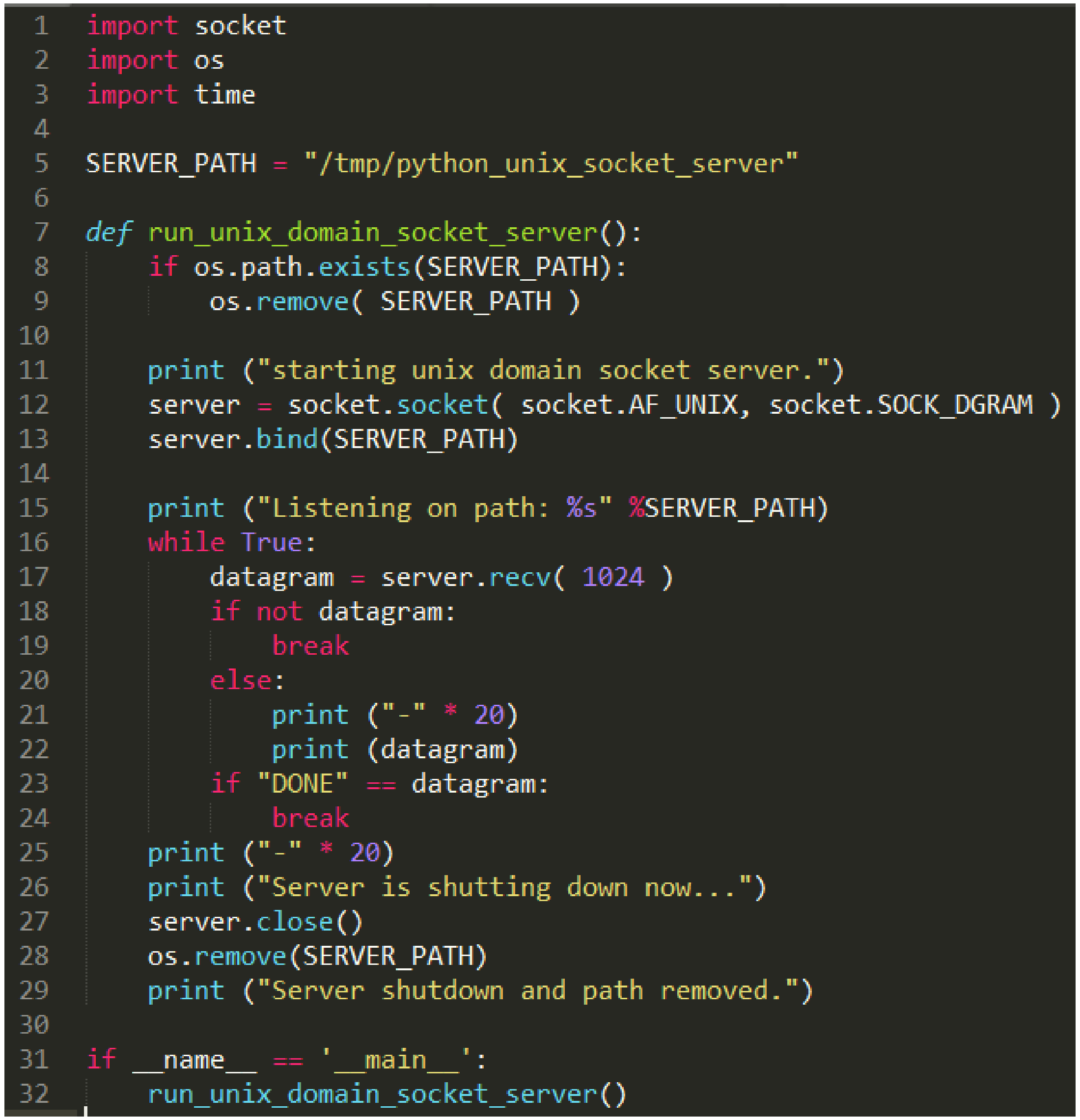
ipc_using_socketpairs



```python
import socket
import os

BUFSIZE = 1024

def test_socketpair():
    """ Test Unix socketpair"""
    parent, child = socket.socketpair()

    pid = os.fork()
    try:
        if pid:
            print ("@Parent, sending message...")
            child.close()

            parent.sendall(bytes("Hello from parent!"))
            # Comment out the above line and uncomment the below line for Python 2.7.
            # parent.sendall("Hello from parent!")

            response = parent.recv(BUFSIZE)
            print ("Response from child:", response)
            parent.close()

        else:
            print ("@Child, waiting for message from parent")
            parent.close()
            message = child.recv(BUFSIZE)
            print ("Message from parent:", message)

            child.sendall(bytes("Hello from child!!"))
            # Comment out the above line and uncomment the below line for Python 2.7.
            # child.sendall("Hello from child!!")

            child.close()
    except Exception as err:
        print ("Error: %s" %err)


if __name__ == '__main__':
    test_socketpair()
```

UNIX_DOMAIN_SOCKET _SERVER

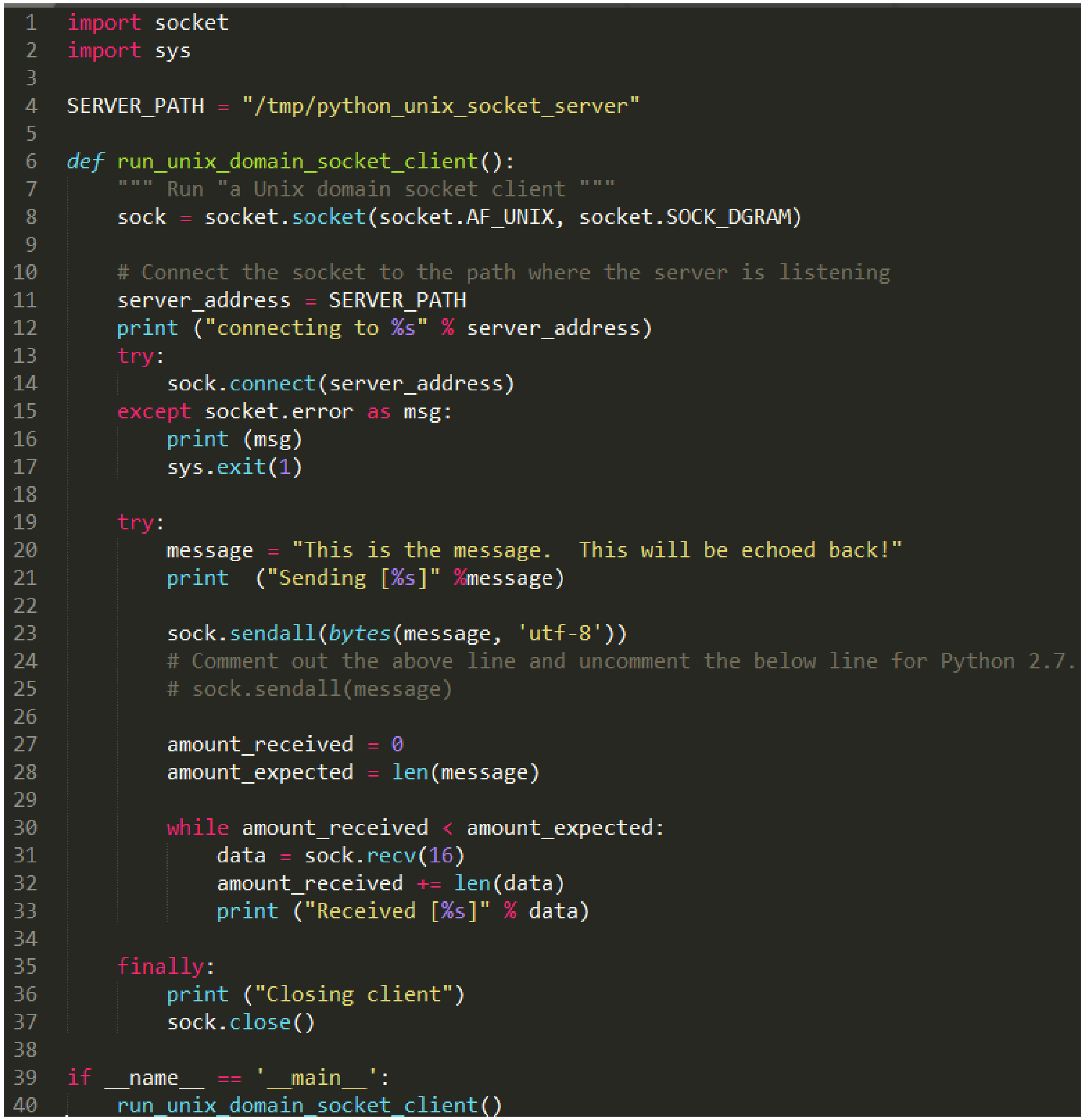
unix_domain_socket_server



```python
import socket
import os
import time

SERVER_PATH = "/tmp/python_unix_socket_server"

def run_unix_domain_socket_server():
    if os.path.exists(SERVER_PATH):
        os.remove( SERVER_PATH )

    print ("starting unix domain socket server.")
    server = socket.socket( socket.AF_UNIX, socket.SOCK_DGRAM )
    server.bind(SERVER_PATH)

    print ("Listening on path: %s" %SERVER_PATH)
    while True:
        datagram = server.recv( 1024 )
        if not datagram:
            break
        else:
            print ("-" * 20)
            print (datagram)
        if "DONE" == datagram:
            break
    print ("-" * 20)
    print ("Server is shutting down now...")
    server.close()
    os.remove(SERVER_PATH)
    print ("Server shutdown and path removed.")

if __name__ == '__main__':
    run_unix_domain_socket_server()
```

UNIX_DOMAIN_SOCKET_CLIENT

unix_domain_socket_client



```python
import socket
import sys

SERVER_PATH = "/tmp/python_unix_socket_server"

def run_unix_domain_socket_client():
    """ Run "a Unix domain socket client """
    sock = socket.socket(socket.AF_UNIX, socket.SOCK_DGRAM)

    # Connect the socket to the path where the server is listening
    server_address = SERVER_PATH
    print ("connecting to %s" % server_address)
    try:
        sock.connect(server_address)
    except socket.error as msg:
        print (msg)
        sys.exit(1)

    try:
        message = "This is the message.  This will be echoed back!"
        print  ("Sending [%s]" %message)

        sock.sendall(bytes(message, 'utf-8'))
        # Comment out the above line and uncomment the below line for Python 2.7.
        # sock.sendall(message)

        amount_received = 0
        amount_expected = len(message)

        while amount_received < amount_expected:
            data = sock.recv(16)
            amount_received += len(data)
            print ("Received [%s]" % data)

    finally:
        print ("Closing client")
        sock.close()

if __name__ == '__main__':
    run_unix_domain_socket_client()
```

Unix Domain Socket

- Unix Domain Socket (UDS) 又稱 IPC (inter-process communication)，用於實現同一主機上的程序之間的通訊。
  - Socket 原本是為了網路通訊設計的，但後來在socket的框架上發展出一種IPC機制，就是UDS。雖然 socket 也可用於同一台主機的程序通訊，如loopback地址127.0.0.1），但是UDS用於IPC更有效率，因為不需要經過 Protocol stack，省去打包拆包，計算校驗和，維護序號和應答等等工作。
  - UDS只是將應用層數據從一個程序拷貝到另一個程序。IPC機制本質上是可靠的通訊，而 Network Protocol 是為不可靠的通訊設計的。
- UDS是全雙工的，API接口語義豐富，相比其它 IPC 機制有明顯的優越性，目前已成為使用最廣泛的IPC機制，如X Window服務器和GUI程序之間就是通過UDS通訊的。

Inter-process Communication

- classical IPC 讓跑在同一台電腦上的 processes 可以互相通訊
- network IPC 則可讓跑在不同電腦的 processes 互相通訊
- network IPC 可用在
  - inter-machine communication
  - intra-machine communication

延伸閱讀

- Socket Programming in Python (Guide)
  - https://realpython.com/python-sockets/#socket-api-overview
- Python 网络编程
  - http://www.runoob.com/python/python-socket.html

Resource is available by
https://jiaweichang.github.io/biography/

THANKS