Computer Network Laboratory

Basic Applications of Network Programming

Jiawei Chang

Dept. of Computer Science and Information Engineering
National Taichung University of Science and Technology

Outline

1. echo_server and echo_client by TCP
2. echo_server and echo_client by UDP
3. forking_mixin_socket_server
4. threading_mixin_socket_server
5. chat_server_with_select

echo_server by TCP

```python
import socket
import sys
import argparse

host = 'localhost'
data_payload = 2048
backlog = 5


def echo_server(port):
    """ A simple echo server """
    # Create a TCP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Enable reuse address/port
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    # Bind the socket to the port
    server_address = (host, port)
    print ("Starting up echo server  on %s port %s" % server_address)
    sock.bind(server_address)
    # Listen to clients, backlog argument specifies the max no. of queued connections
    sock.listen(backlog)
    while True:
        print ("Waiting to receive message from client")
        client, address = sock.accept()
        data = client.recv(data_payload)
        if data:
            print ("Data: %s" %data)
            client.send(data)
            print ("sent %s bytes back to %s" % (data, address))
        # end connection
        client.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_server(port)
```

```
python 12_echo_server_TCP.py --port=<PORT>
```

echo_client by TCP

```python
import socket
import sys

import argparse

host = 'localhost'

def echo_client(port):
    """ A simple echo client """
    # Create a TCP/IP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # Connect the socket to the server
    server_address = (host, port)
    print ("Connecting to %s port %s" % server_address)
    sock.connect(server_address)

    # Send data
    try:
        # Send data
        message = "Test message. This will be echoed"
        print ("Sending %s" % message)
        sock.sendall(message.encode('utf-8'))
        # Look for the response
        amount_received = 0
        amount_expected = len(message)
        while amount_received < amount_expected:
            data = sock.recv(16)
            amount_received += len(data)
            print ("Received: %s" % data)
    except socket.error as e:
        print ("Socket error: %s" %str(e))
    except Exception as e:
        print ("Other exception: %s" %str(e))
    finally:
        print ("Closing connection to the server")
        sock.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_client(port)
```

```
python 12_echo_client_TCP.py --port=<PORT>
```

echo_server by UDP

```python
import socket
import sys
import argparse

host = 'localhost'
data_payload = 2048

def echo_server(port):
    """ A simple echo server """
    # Create a UDP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    # Bind the socket to the port
    server_address = (host, port)
    print ("Starting up echo server on %s port %s" % server_address)

    sock.bind(server_address)

    while True:
        print ("Waiting to receive message from client")
        data, address = sock.recvfrom(data_payload)

        print ("received %s bytes from %s" % (len(data), address))
        print ("Data: %s" %data)

        if data:
            sent = sock.sendto(data, address)
            print ("sent %s bytes back to %s" % (sent, address))


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_server(port)
```

python 13_echo_server_UDP.py --port=<PORT>

echo_client by UDP

```python
import socket
import sys
import argparse

host = 'localhost'
data_payload = 2048

def echo_client(port):
    """ A simple echo client """
    # Create a UDP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    server_address = (host, port)
    print ("Connecting to %s port %s" % server_address)
    message = 'This is the message.  It will be repeated.'

    try:

        # Send data
        message = "Test message. This will be echoed"
        print ("Sending %s" % message)
        sent = sock.sendto(message.encode('utf-8'), server_address)

        # Receive response
        data, server = sock.recvfrom(data_payload)
        print ("received %s" % data)

    finally:
        print ("Closing connection to the server")
        sock.close()

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Server Example')
    parser.add_argument('--port', action="store", dest="port", type=int, required=True)
    given_args = parser.parse_args()
    port = given_args.port
    echo_client(port)
```

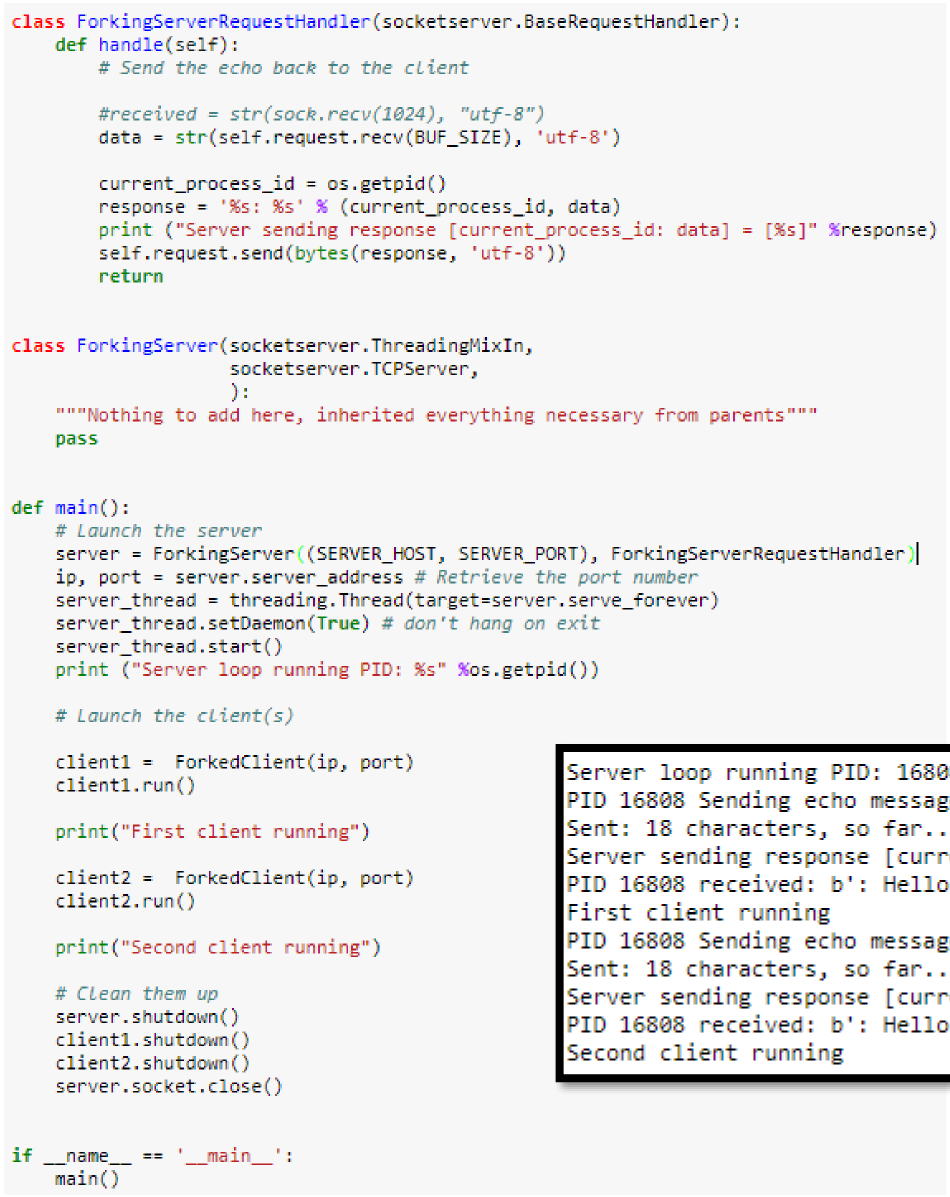
python 13_echo_client_UDP.py --port=<PORT>

MULTIPLEXING SOCKET I/O FOR BETTER PERFORMANCE

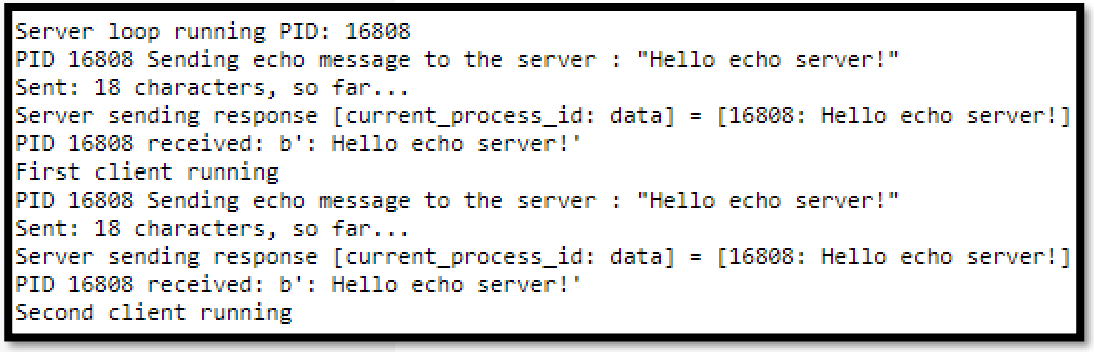
forking_mixin_socket_server

```python
import os
import socket
import threading
import socketserver


SERVER_HOST = 'localhost'
SERVER_PORT = 0 # tells the kernel to pickup a port dynamically
BUF_SIZE = 1024
ECHO_MSG = 'Hello echo server!'


class ForkedClient():
    """ A client to test forking server"""
    def __init__(self, ip, port):
        # Create a socket
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # Connect to the server
        self.sock.connect((ip, port))

    def run(self):
        """ Client playing with the server"""
        # Send the data to server
        current_process_id = os.getpid()
        print ('PID %s Sending echo message to the server : "%s"' % (current_process_id, ECHO_MSG))

        sent_data_length = self.sock.send(bytes(ECHO_MSG, 'utf-8'))

        print ("Sent: %d characters, so far..." %sent_data_length)

        # Display server response
        response = self.sock.recv(BUF_SIZE)
        print ("PID %s received: %s" % (current_process_id, response[5:]))

    def shutdown(self):
        """ Cleanup the client socket """
        self.sock.close()
```

forking_mixin_socket_server

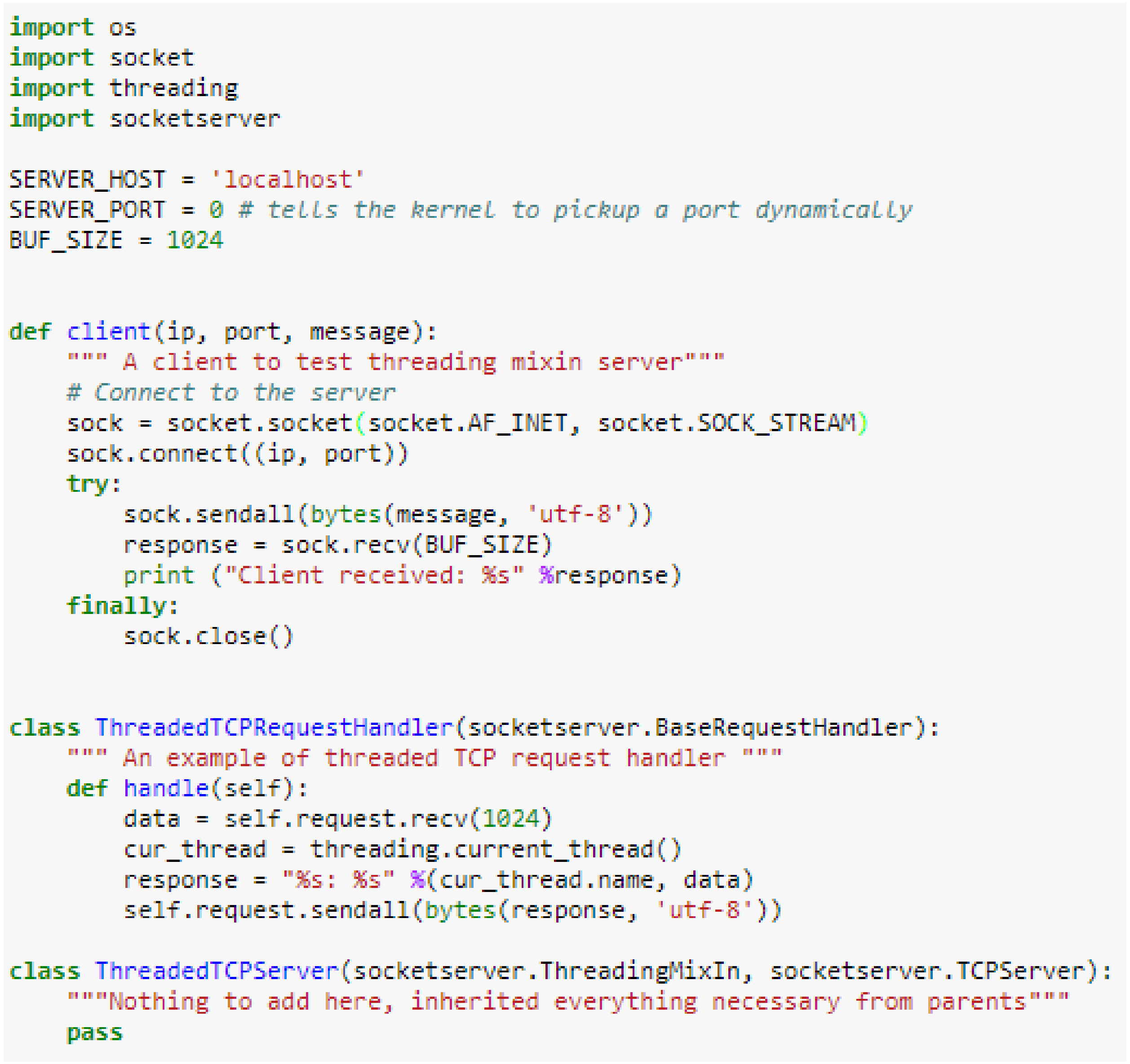


```python
class ForkingServerRequestHandler(socketserver.BaseRequestHandler):
    def handle(self):
        # Send the echo back to the client

        #received = str(sock.recv(1024), "utf-8")
        data = str(self.request.recv(BUF_SIZE), 'utf-8')

        current_process_id = os.getpid()
        response = '%s: %s' % (current_process_id, data)
        print ("Server sending response [current_process_id: data] = [%s]" %response)
        self.request.send(bytes(response, 'utf-8'))
        return


class ForkingServer(socketserver.ThreadingMixIn,
                    socketserver.TCPServer,
                    ):
    """Nothing to add here, inherited everything necessary from parents"""
    pass


def main():
    # Launch the server
    server = ForkingServer((SERVER_HOST, SERVER_PORT), ForkingServerRequestHandler)
    ip, port = server.server_address # Retrieve the port number
    server_thread = threading.Thread(target=server.serve_forever)
    server_thread.setDaemon(True) # don't hang on exit
    server_thread.start()
    print ("Server loop running PID: %s" %os.getpid())

    # Launch the client(s)

    client1 =  ForkedClient(ip, port)
    client1.run()

    print("First client running")

    client2 =  ForkedClient(ip, port)
    client2.run()

    print("Second client running")

    # Clean them up
    server.shutdown()
    client1.shutdown()
    client2.shutdown()
    server.socket.close()


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



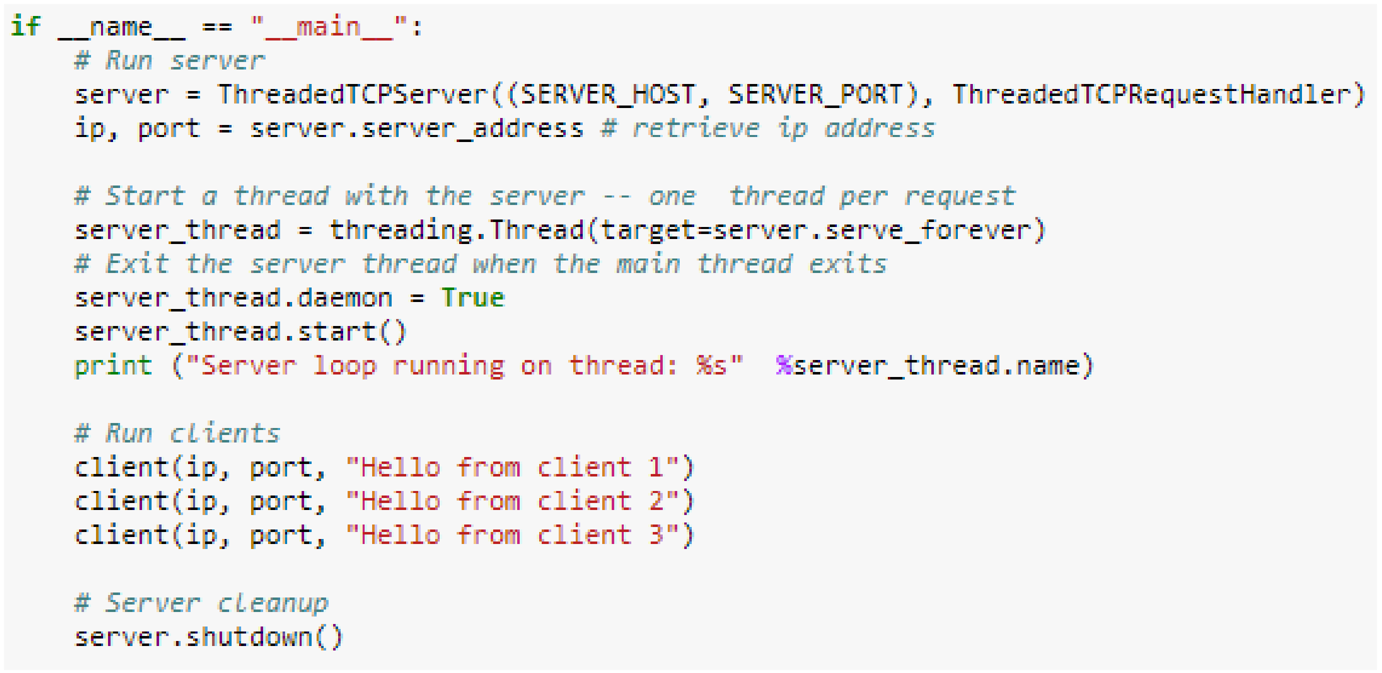
```
Server loop running PID: 16808
PID 16808 Sending echo message to the server : "Hello echo server!"
Sent: 18 characters, so far...
Server sending response [current_process_id: data] = [16808: Hello echo server!]
PID 16808 received: b': Hello echo server!'
First client running
PID 16808 Sending echo message to the server : "Hello echo server!"
Sent: 18 characters, so far...
Server sending response [current_process_id: data] = [16808: Hello echo server!]
PID 16808 received: b': Hello echo server!'
Second client running
```

threading_mixin_socket_server

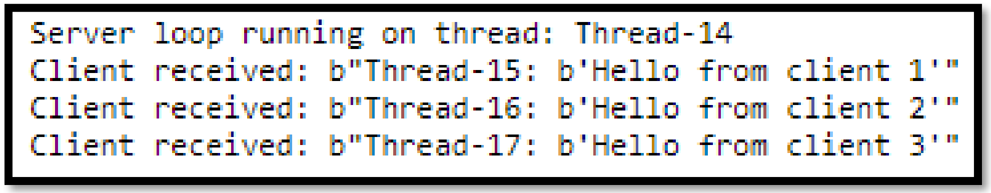


```python
import os
import socket
import threading
import socketserver

SERVER_HOST = 'localhost'
SERVER_PORT = 0 # tells the kernel to pickup a port dynamically
BUF_SIZE = 1024


def client(ip, port, message):
    """ A client to test threading mixin server"""
    # Connect to the server
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.connect((ip, port))
    try:
        sock.sendall(bytes(message, 'utf-8'))
        response = sock.recv(BUF_SIZE)
        print ("Client received: %s" %response)
    finally:
        sock.close()


class ThreadedTCPRequestHandler(socketserver.BaseRequestHandler):
    """ An example of threaded TCP request handler """
    def handle(self):
        data = self.request.recv(1024)
        cur_thread = threading.current_thread()
        response = "%s: %s" %(cur_thread.name, data)
        self.request.sendall(bytes(response, 'utf-8'))

class ThreadedTCPServer(socketserver.ThreadingMixIn, socketserver.TCPServer):
    """Nothing to add here, inherited everything necessary from parents"""
    pass
```

threading_mixin_socket_server



```python
if __name__ == "__main__":
    # Run server
    server = ThreadedTCPServer((SERVER_HOST, SERVER_PORT), ThreadedTCPRequestHandler)
    ip, port = server.server_address # retrieve ip address

    # Start a thread with the server -- one  thread per request
    server_thread = threading.Thread(target=server.serve_forever)
    # Exit the server thread when the main thread exits
    server_thread.daemon = True
    server_thread.start()
    print ("Server loop running on thread: %s"  %server_thread.name)

    # Run clients
    client(ip, port, "Hello from client 1")
    client(ip, port, "Hello from client 2")
    client(ip, port, "Hello from client 3")

    # Server cleanup
    server.shutdown()
```



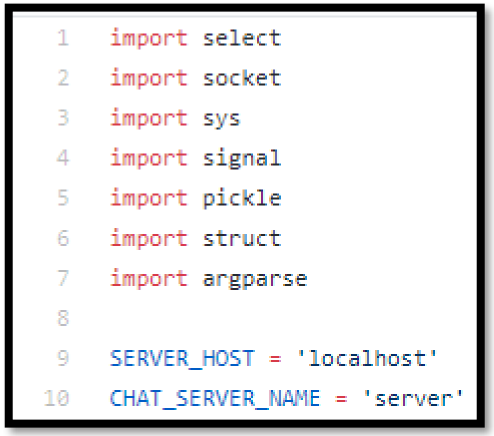
```
Server loop running on thread: Thread-14
Client received: b"Thread-15: b'Hello from client 1'"
Client received: b"Thread-16: b'Hello from client 2'"
Client received: b"Thread-17: b'Hello from client 3'"
```

Thinking Time

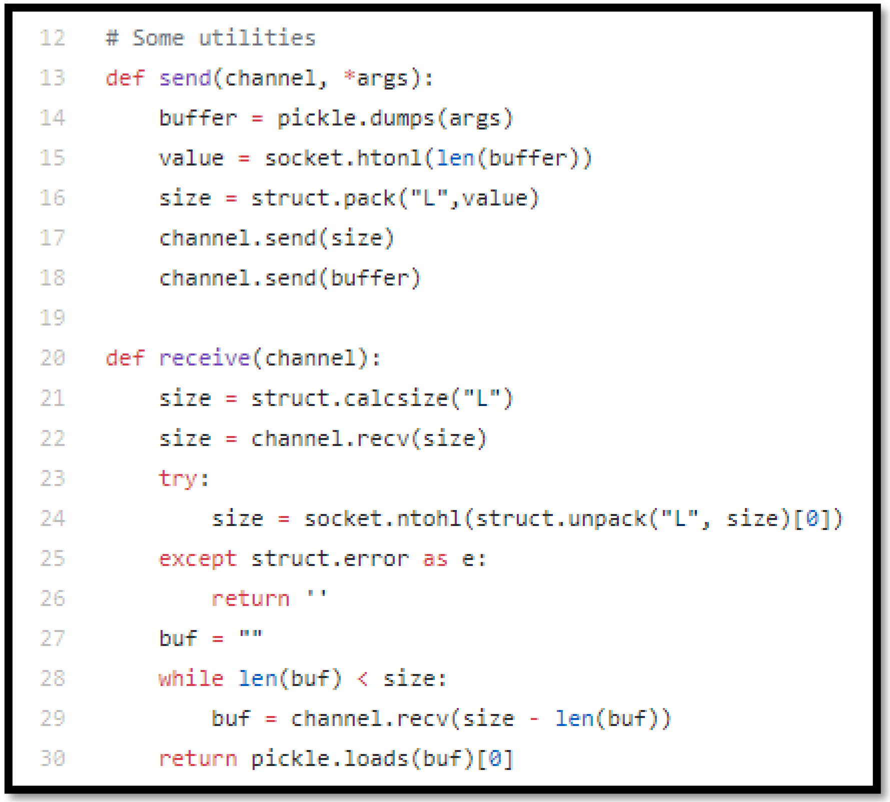
Process v.s. Thread

What is relation between them?

chat_server_with_select



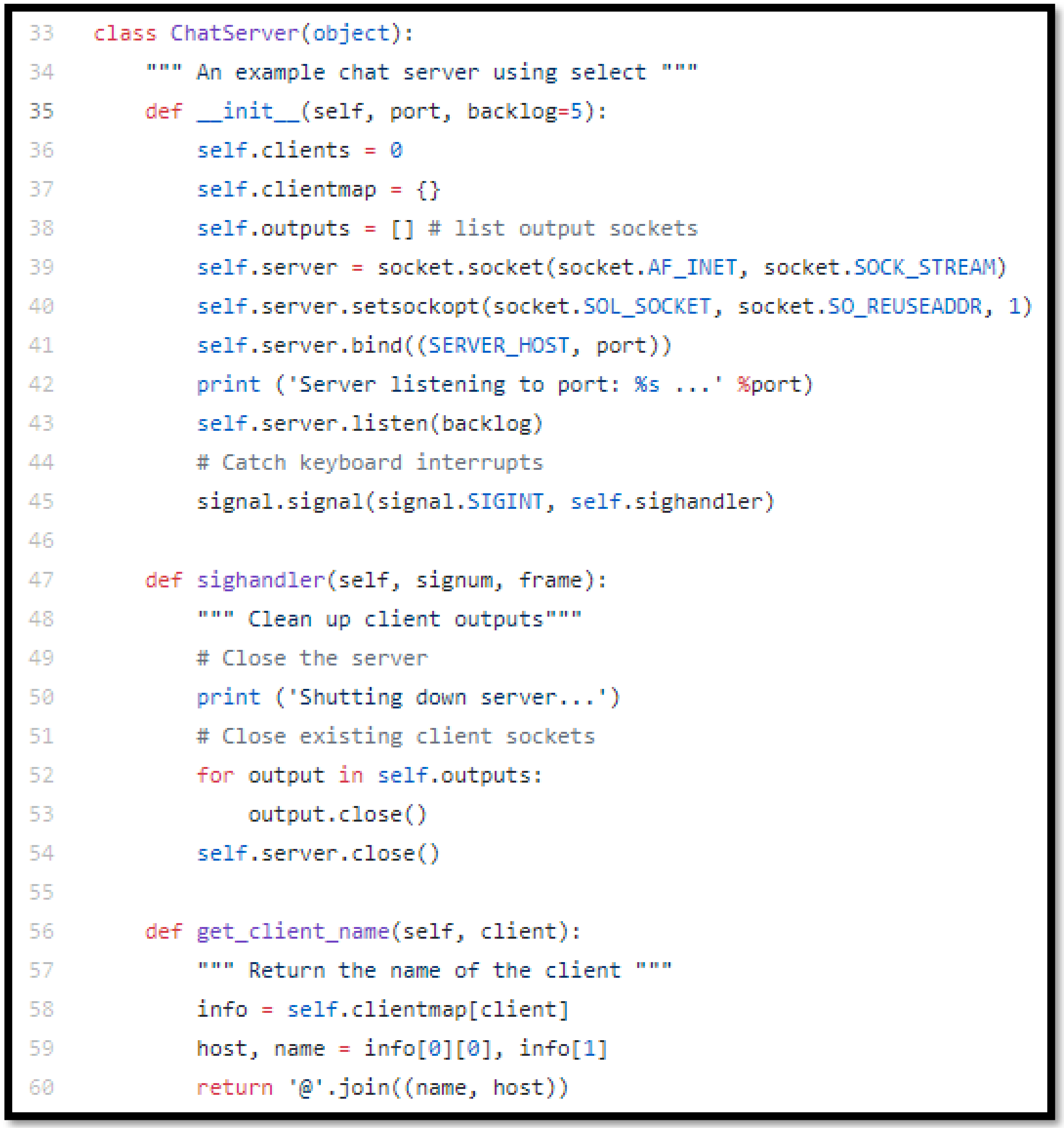
```python
import select
import socket
import sys
import signal
import pickle
import struct
import argparse

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

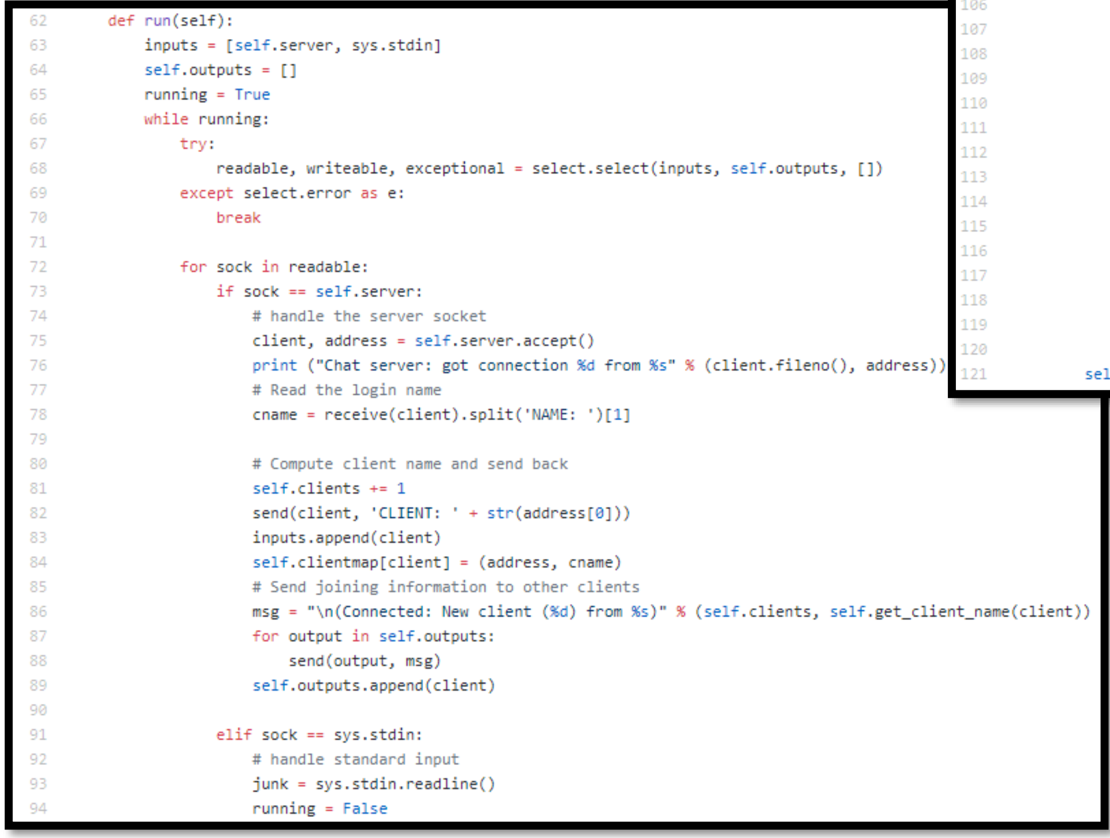
Server Side



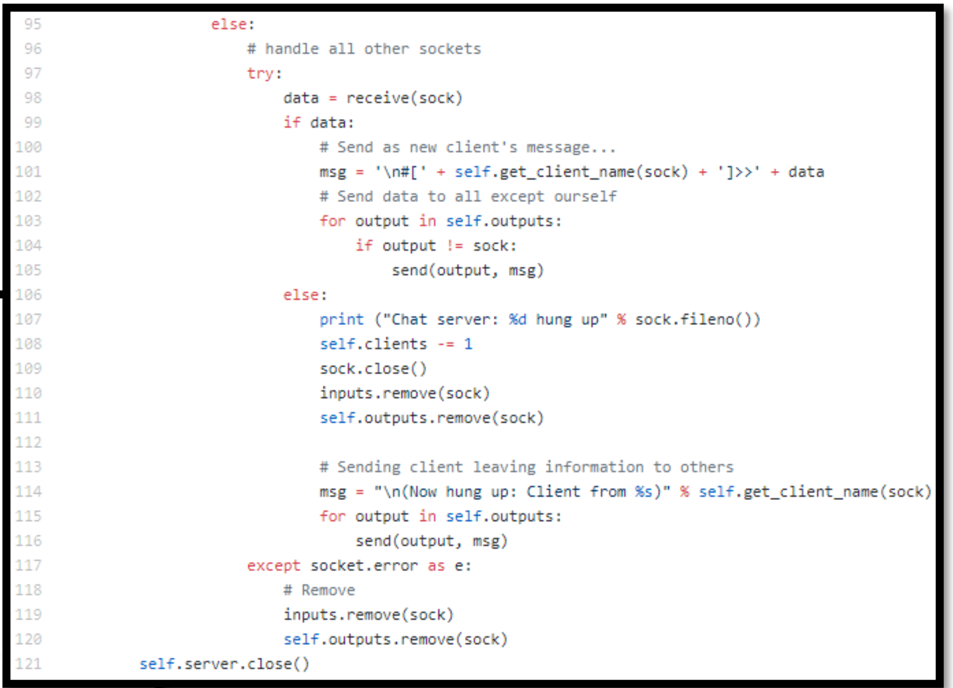
```
class ChatServer(object):
    """ An example chat server using select """
    def __init__(self, port, backlog=5):
        self.clients = 0
        self.clientmap = {}
        self.outputs = [] # list output sockets
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server.setsockopt(socket.SOL_SOCKET,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



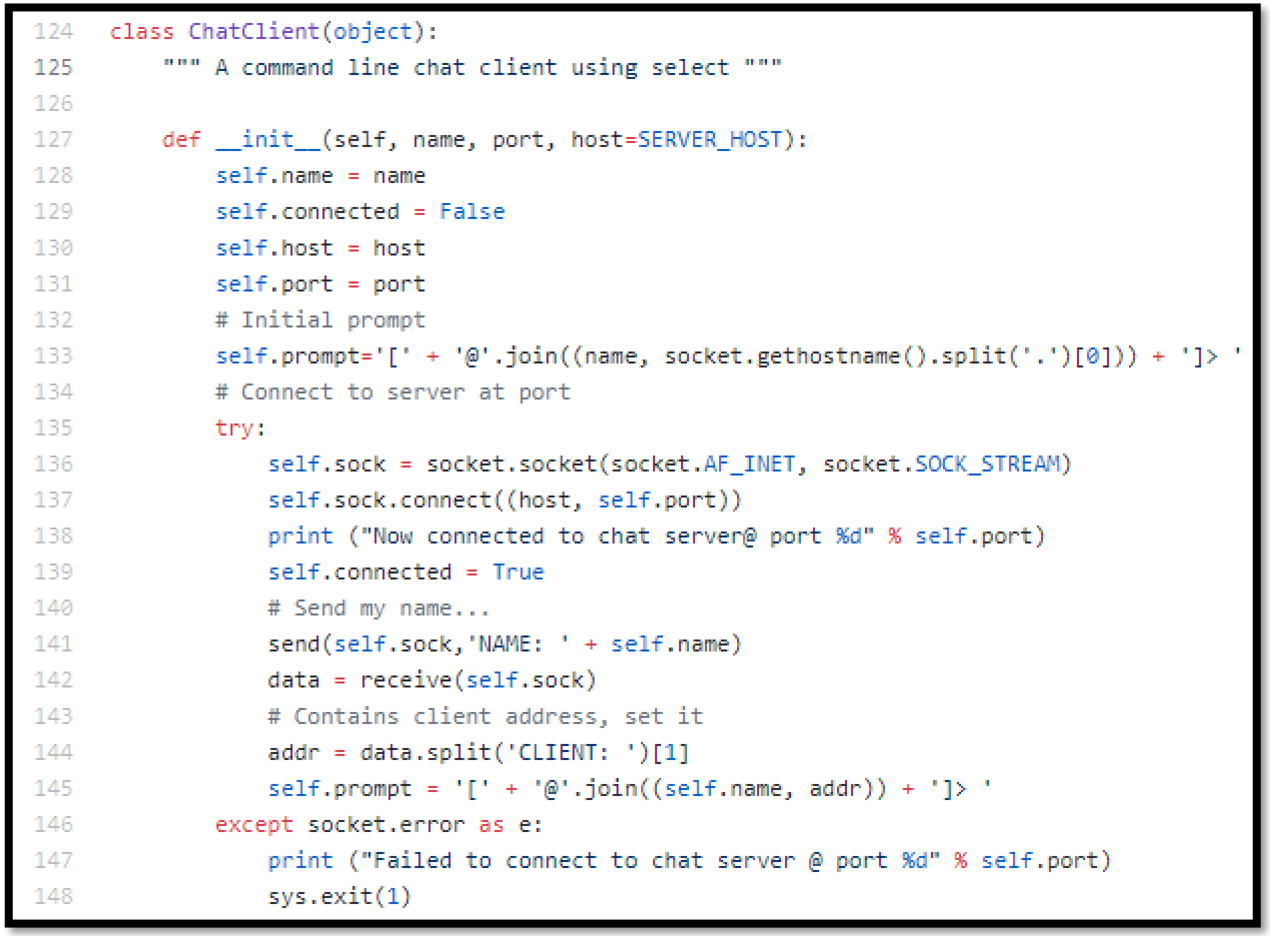
```python
    def run(self):
        inputs = [self.server, sys.stdin]
        self.outputs = []
        running = True
        while running:
            try:
                readable, writeable, exceptional = select.select(inputs, self.outputs, [])
            except select.error as e:
           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

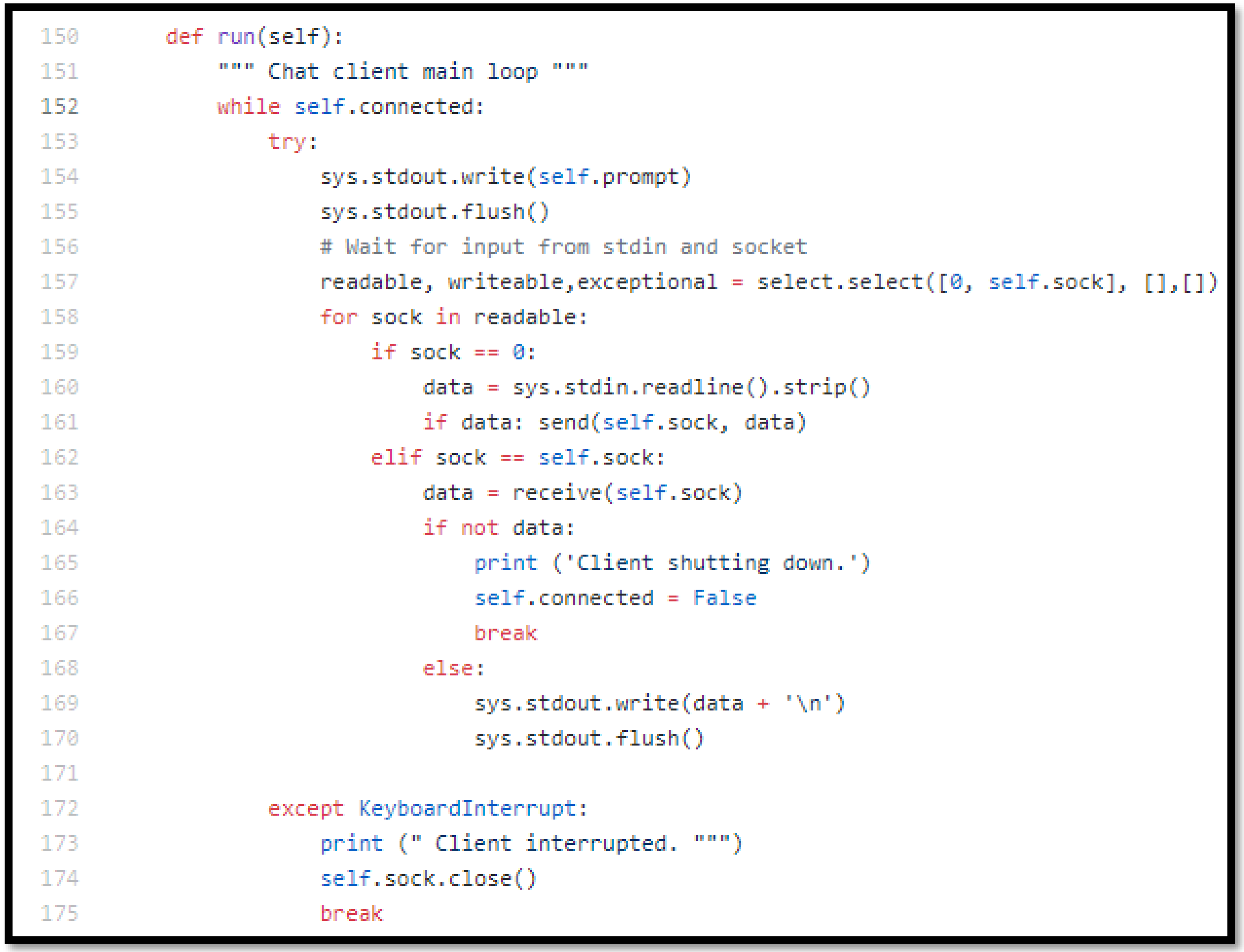
Client Side



```python
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)
```

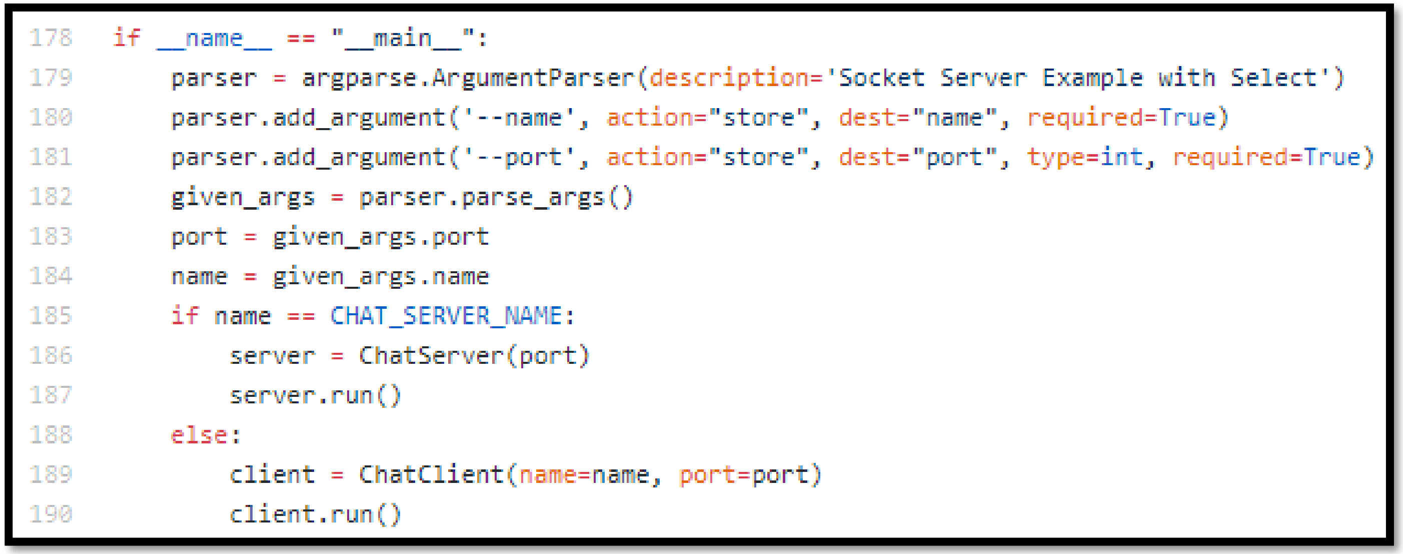
chat_server_with_select

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
```

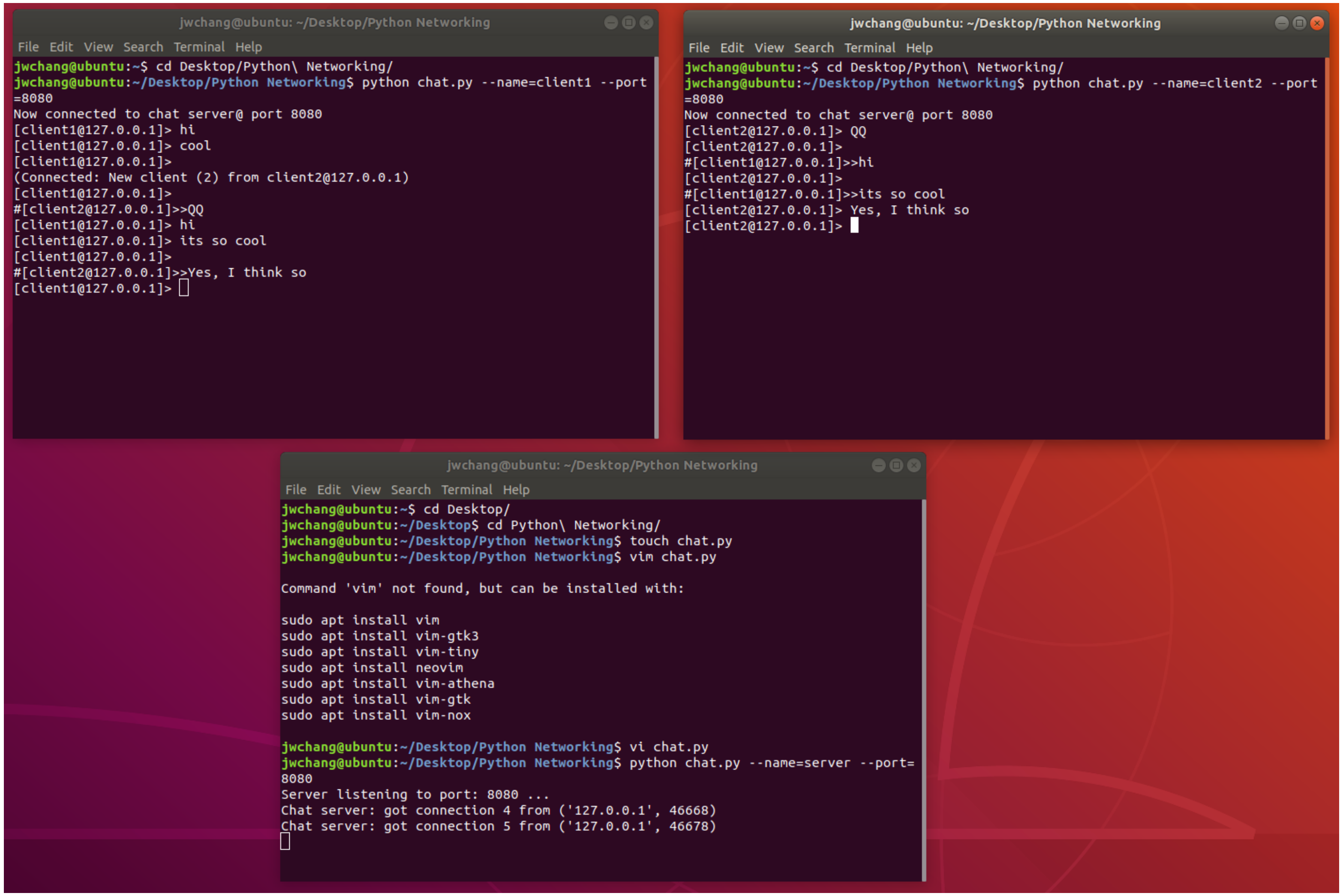
chat_server_with_select

Main



```python
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)
```

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