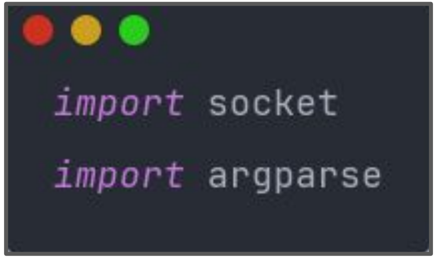
LAB 05 Session Layer

Jennessa Sierra & Andres Hung
CMPS1192 Networking Fundamentals
October 17, 2024

Creating a Python Socket Server:

Step 1: Import Libraries

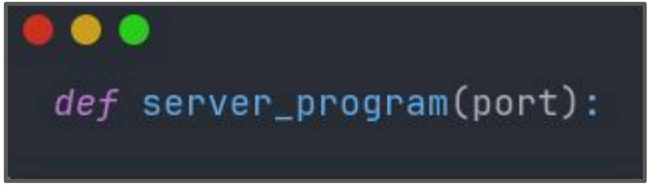


```
import socket

import argparse
```

socket: creating and managing network connections

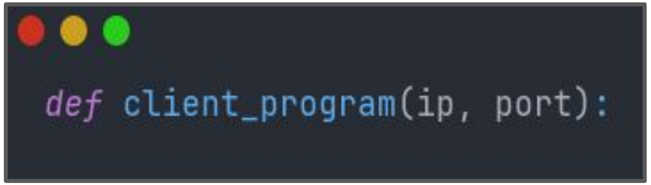
argparse: parsing command-line arguments

Step 2: Define Program (Server/Client)



```
def server_program(port):
```

port: port number to bind to the server



```
def client_program(ip, port):
```

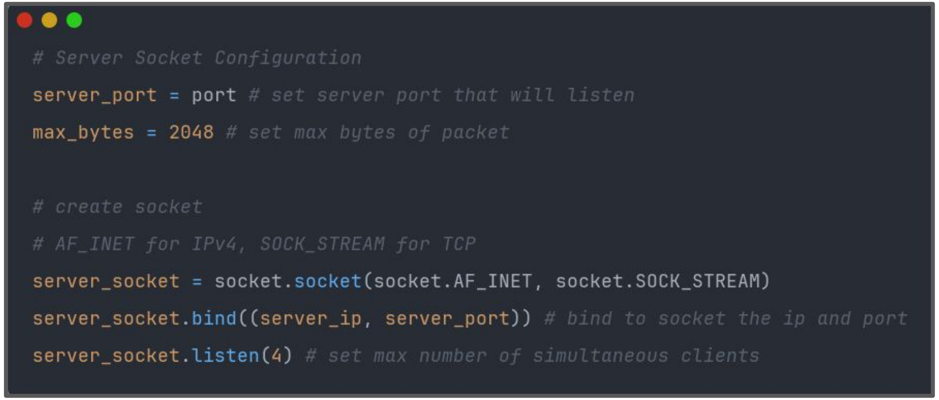
ip: ip address of the server

port: port number of the server

Creating a Python Socket Server:

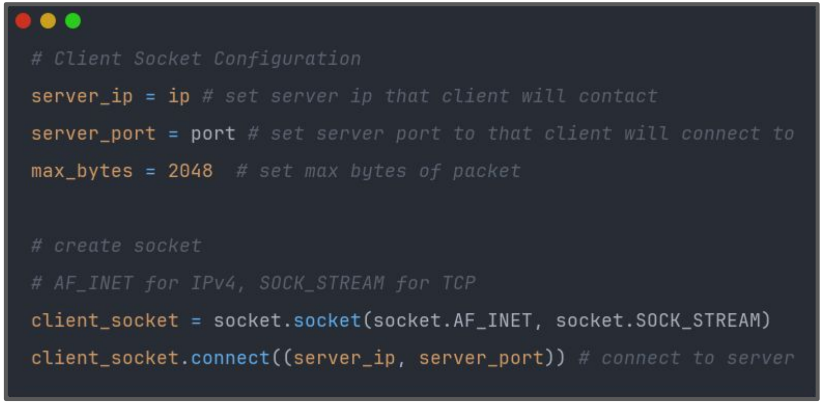
Step 3: Create the Socket

Server-Specific Configuration



```
# Server Socket Configuration
server_port = port # set server port that will listen
max_bytes = 2048 # set max bytes of packet

# create socket
# AF_INET for IPv4, SOCK_STREAM for TCP
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((server_ip, server_port)) # bind to socket the ip and port
server_socket.listen(4) # set max number of simultaneous clients
```

Client-Specific Configuration

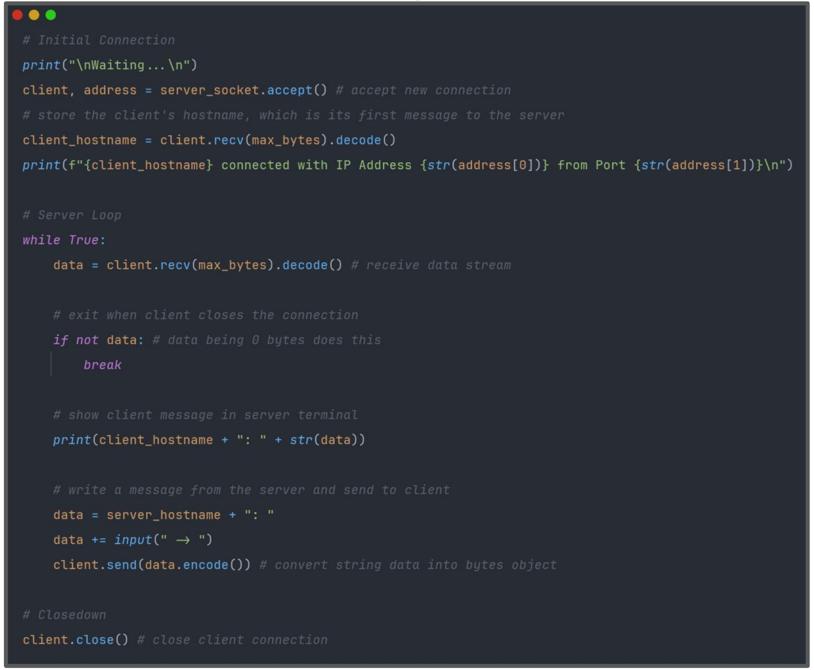


```
# Client Socket Configuration
server_ip = ip # set server ip that client will contact
server_port = port # set server port to that client will connect to
max_bytes = 2048  # set max bytes of packet

# create socket
# AF_INET for IPv4, SOCK_STREAM for TCP
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((server_ip, server_port)) # connect to server
```

Creating a Python Socket Server:

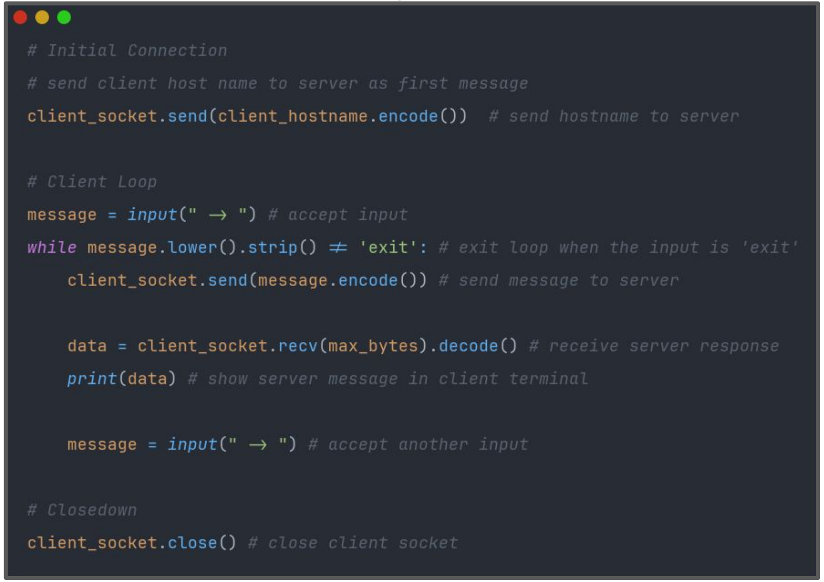
Step 4: Handle Message Logic

Server-Specific Configuration



```
# Initial Connection
print("\nWaiting...\n")
client, address = server_socket.accept() # accept new connection
# store the client's hostname, which is its first message to the server
client_hostname = client.recv(max_bytes).decode()
print(f"{client_hostname} connected with IP Address {str(address[0])} from Port {str(address[1])}\n")

# Server Loop
while True:
    data = client.recv(max_bytes).decode() # receive data stream

    # exit when client closes the connection
    if not data: # data being 0 bytes does this
        break

    # show client message in server terminal
    print(client_hostname + ": " + str(data))

    # write a message from the server and send to client
    data = server_hostname + ": "
    data += input(" → ")
    client.send(data.encode()) # convert string data into bytes object

# Closedown
client.close() # close client connection
```

Client-Specific Configuration

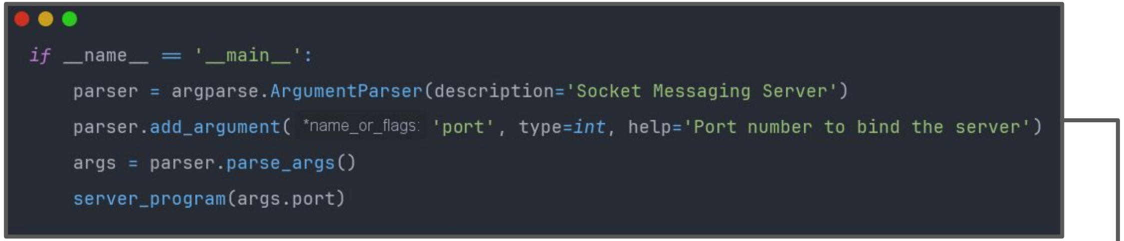


```
# Initial Connection
# send client host name to server as first message
client_socket.send(client_hostname.encode())  # send hostname to server

# Client Loop
message = input(" → ") # accept input
while message.lower().strip() ≠ 'exit': # exit loop when the input is 'exit'
    client_socket.send(message.encode()) # send message to server

    data = client_socket.recv(max_bytes).decode() # receive server response
    print(data) # show server message in client terminal

    message = input(" → ") # accept another input

# Closedown
client_socket.close() # close client socket
```

Creating a Python Socket Server:

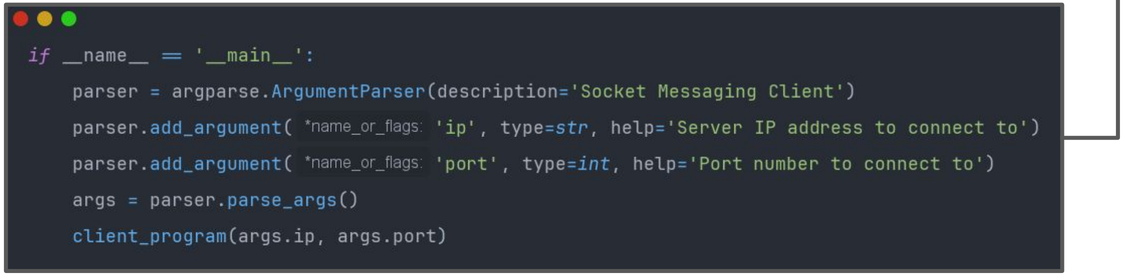
Step 5: Set Up Command-Line Arguments

Server-Specific Configuration



```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Messaging Server')
    parser.add_argument( *name_or_flags: 'port', type=int, help='Port number to bind the server')
    args = parser.parse_args()
    server_program(args.port)
```

Client-Specific Configuration



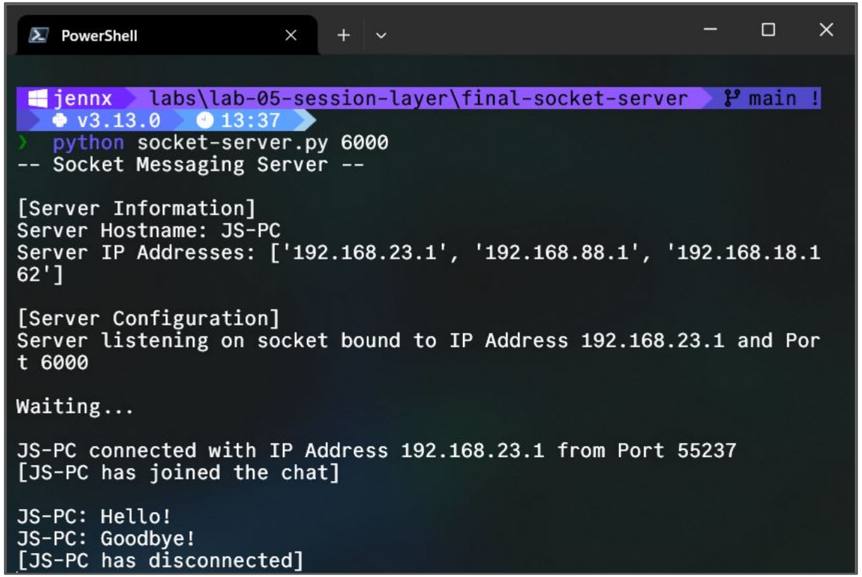
```
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Socket Messaging Client')
    parser.add_argument( *name_or_flags: 'ip', type=str, help='Server IP address to connect to')
    parser.add_argument( *name_or_flags: 'port', type=int, help='Port number to connect to')
    args = parser.parse_args()
    client_program(args.ip, args.port)
```

`main:` calls the server/client function and passes the parameters as arguments

Testing the Socket Server:

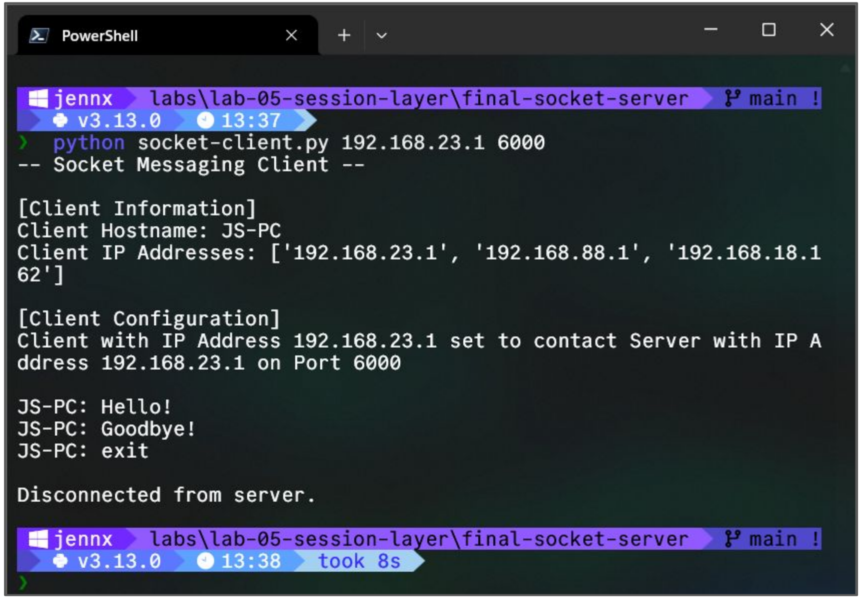
Open Terminal & Run Script

Server-Side



```
jennx  labs\lab-05-session-layer\final-socket-server  main !
 v3.13.0  13:37
› python socket-server.py 6000
-- Socket Messaging Server --

[Server Information]
Server Hostname: JS-PC
Server IP Addresses: ['192.168.23.1', '192.168.88.1', '192.168.18.1
62']

[Server Configuration]
Server listening on socket bound to IP Address 192.168.23.1 and Por
t 6000

Waiting...

JS-PC connected with IP Address 192.168.23.1 from Port 55237
[JS-PC has joined the chat]

JS-PC: Hello!
JS-PC: Goodbye!
[JS-PC has disconnected]
```

Client-Side



```
jennx  labs\lab-05-session-layer\final-socket-server  main !
 v3.13.0  13:37
› python socket-client.py 192.168.23.1 6000
-- Socket Messaging Client --

[Client Information]
Client Hostname: JS-PC
Client IP Addresses: ['192.168.23.1', '192.168.88.1', '192.168.18.1
62']

[Client Configuration]
Client with IP Address 192.168.23.1 set to contact Server with IP A
ddress 192.168.23.1 on Port 6000

JS-PC: Hello!
JS-PC: Goodbye!
JS-PC: exit

Disconnected from server.

jennx  labs\lab-05-session-layer\final-socket-server  main !
 v3.13.0  13:38  took 8s
›
```

What is a Socket?

- Old definition of socket by ARPANET is a combination of IP Address and Port. Now this is called an **endpoint**.
- New definition is a software term that is a combination
  - **IP Address** (Layer 3), **Port** (Layer 4), and **Protocol** (Layer 4)
- Allows **bidirectional communication** for process-to-process communication.
- **Inter-Process Communication (IPC)** on same or different hosts

Stream Sockets
SOCK_STREAM
TCP

Datagram Sockets
SOCK_DGRAM
UDP

Raw Sockets
SOCK_RAW
ICMP

Sockets in Client/Server Architecture

.bind()

.listen()

.accept()

.connect()

.send()

.recv()

.close()

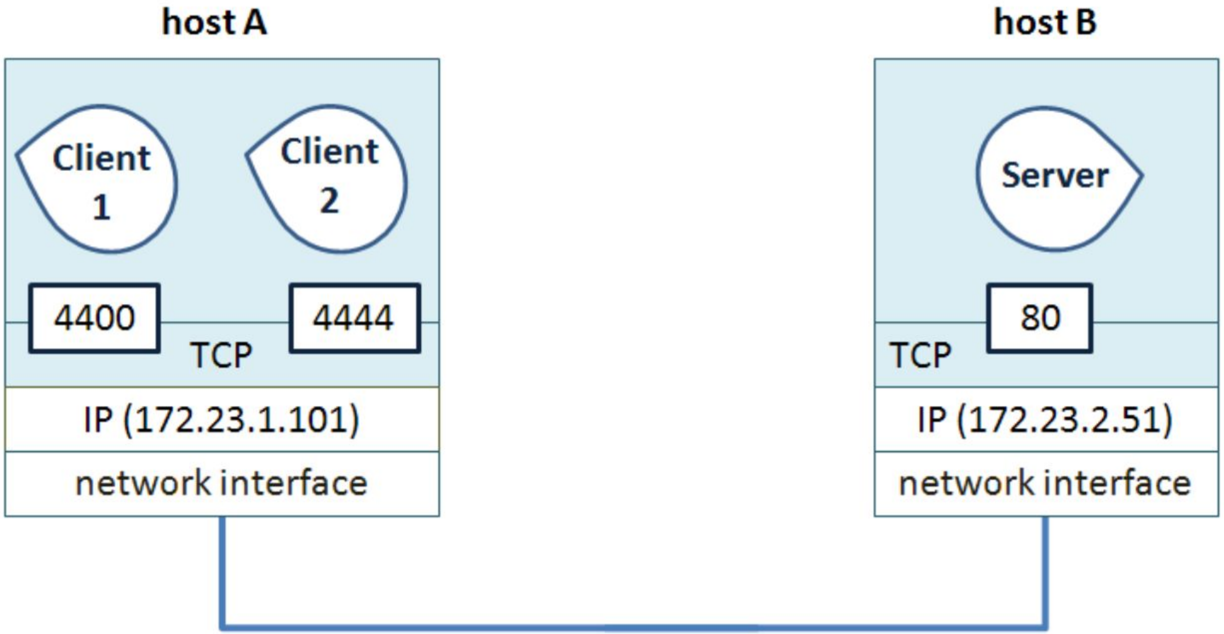


Figure 3.17: Client Server model over TCP/IP

Network buffers

Blocking

Multiple Clients

Concurrency

Designing Headers

Lab Activity Client and Server

	Client Computer (Jennessa)	Server Computer (Andres)
Protocol	TCP	TCP
Source Port	**54321**	6000
Destination Port	6000	0
Source IP	10.0.56.23	10.0.73.221
Destination IP	10.0.73.221	0.0.0.0

Well-known Ports (0 - 1023)
Registered Ports (1024 - 49151)
Dynamic/Private Ports (49152 - 65535)

Works for 1 client at a time

The Socket API

- Internets Sockets span from Layer 4 (Transport) to Layer 7 (Application), mainly dealing with those two layers.
- Socket programming done in Application Layer.
- A session (layer 5) is technically established with sockets.
- Layer 5 (Session) in the OSI model is more theoretical, with many of its functions happening in Layer 4 (e.g. TCP three-way handshake).

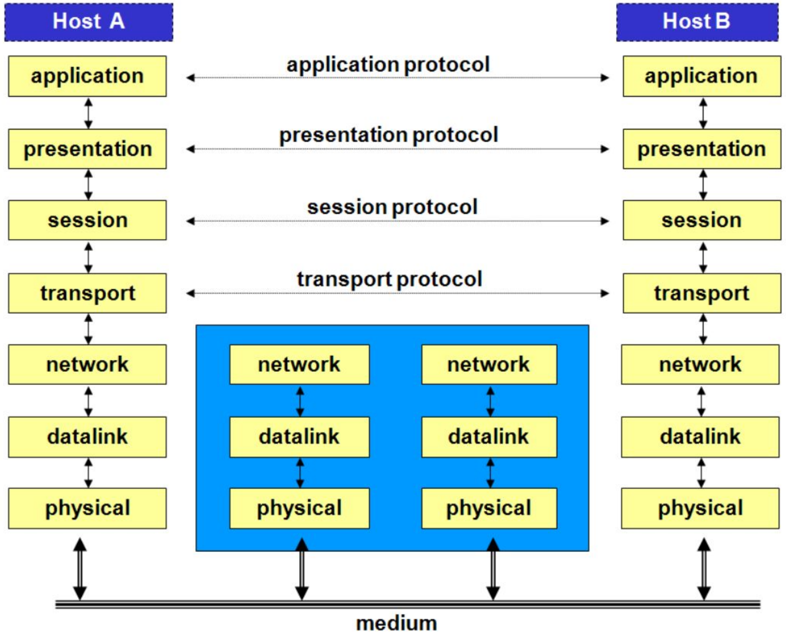


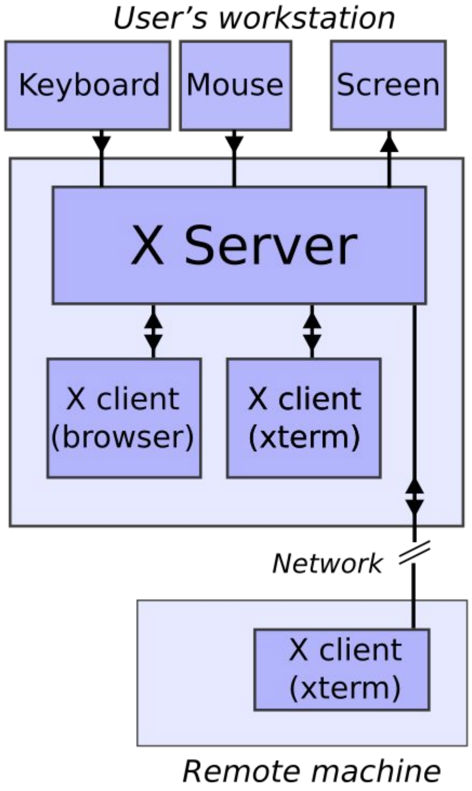
Figure 1.1: The OSI model

The Session Layer (Layer 5)

- Session management (establishment, authentication, recovery)
- Synchronization (resume data transfer e.g. resume downloads)
- Dialog control (half-duplex, full-duplex)

Protocols

- Remote Procedure Call (RPC) (Windows Active Directory)
- AppleTalk Session Protocol (ASP) (Apple device file sharing)
- X Window System (Linux desktop environments)
- Point-to-Point Tunneling Protocol (PPTP) (VPN services)


User's workstation
Keyboard
Mouse
Screen
X Server
X client (browser)
X client (xterm)
Network
X client (xterm)
Remote machine