



SOCKET PROGRAMMING



OVERVIEW

01

What are sockets?

02

Socket
Programming in
Python?

03

Client-Server
Architecture

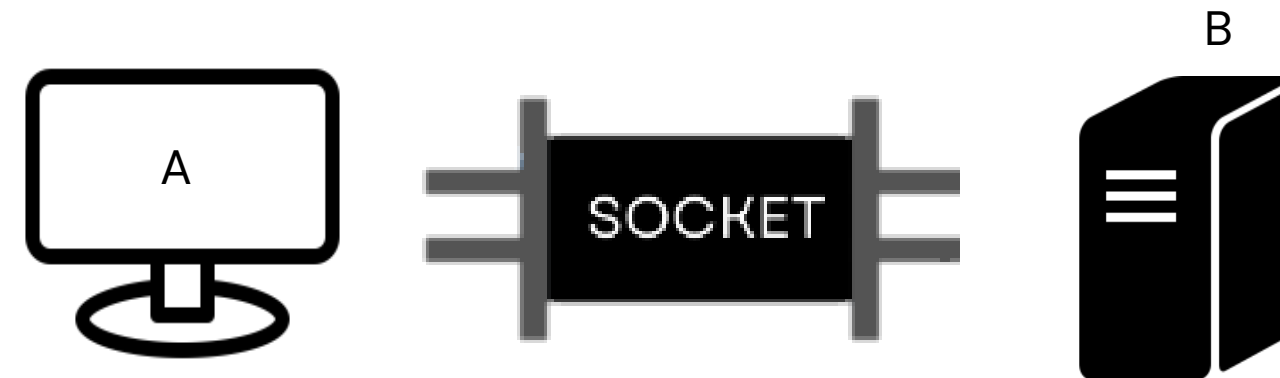
04

Implementation
examples

What are sockets?

01

Let's suppose we have Point A that would like to communicate with Point B, sockets are communication endpoints built for sending and receiving data



02

There are several types of sockets: an Internet socket, sockets designed for Bluetooth, Infrared, or even at the operating system level.

03

An internet socket would be specified as `socket.AF_INET`, while a Bluetooth socket would be `socket.AF_BLUETOOTH`

Internet socket

We first specify the `AF_INET` in case of an internet socket and after we specify if it is :

Stream socket

Used for reliable, connection-oriented socket where there is an exchange of data that we can terminate when we want.

Error-free, no out-of-order packets

SSH/ TELNET/HTTP

Datagram socket

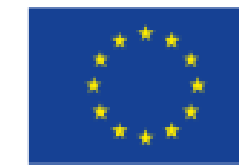
Used for unreliable connectionless socket

Packets may be lost; may arrive out of order. However it is more real time and less network and PC stress

Streaming audio/video

Common port Numbers

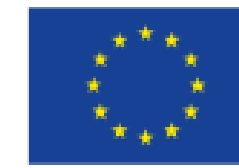
Protocol	Port Number	Python Library	Function
HTTP	80	httplib, urllib,xmllrpclib	Web pages
FTP	20	ftplib, urllib	File transfers
NNTP	119	nntplib	Unsent news
SMTP	25	smtplib	Sending email
Telnet	23	telnetlib	Command lines
POP3	110	poplib	Fetching email
Gopher	70	gopherlib	Document transfer



```
C:\Users\asus>netstat -a
```

```
Active Connections
```

Proto	Local Address	Foreign Address	State
TCP	0.0.0.0:135	DESKTOP-DVG6VG6:0	LISTENING
TCP	0.0.0.0:445	DESKTOP-DVG6VG6:0	LISTENING
TCP	0.0.0.0:5040	DESKTOP-DVG6VG6:0	LISTENING
TCP	0.0.0.0:27000	DESKTOP-DVG6VG6:0	LISTENING
TCP	0.0.0.0:49664	DESKTOP-DVG6VG6:0	LISTENING
TCP	0.0.0.0:49665	DESKTOP-DVG6VG6:0	LISTENING
TCP	0.0.0.0:49666	DESKTOP-DVG6VG6:0	LISTENING
TCP	0.0.0.0:49667	DESKTOP-DVG6VG6:0	LISTENING
TCP	0.0.0.0:49668	DESKTOP-DVG6VG6:0	LISTENING
TCP	0.0.0.0:49671	DESKTOP-DVG6VG6:0	LISTENING
TCP	0.0.0.0:50116	DESKTOP-DVG6VG6:0	LISTENING
TCP	0.0.0.0:50877	DESKTOP-DVG6VG6:0	LISTENING
TCP	10.38.224.109:139	DESKTOP-DVG6VG6:0	LISTENING
TCP	10.38.224.109:50845	wm-in-f188:5228	ESTABLISHED
TCP	10.38.224.109:50846	104.16.103.112:https	ESTABLISHED
TCP	10.38.224.109:50847	whatsapp-cdn-shv-01-bcn1:https	ESTABLISHED
TCP	10.38.224.109:50850	ec2-34-237-73-95:https	ESTABLISHED



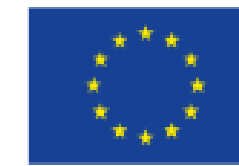
Socket programming in python

01

First you need to import the 'socket' module to use socket functions

02

This module consists of several built-in methods that are required for creating sockets and help them associate with each other



Socket methods

Function Call	Description
Socket()	To create a socket
Bind()	It's a socket identification like a telephone number to contact
Listen()	Ready to receive a connection
Connect()	Ready to act as a sender
Accept()	Confirmation, it is like accepting to receive a call from a sender
Write()	To send data
Read()	To receive data
Close()	To close a connection

```
# Create a socket object  
s = socket.socket(socket_family, socket_type, protocol)
```

Socket_family

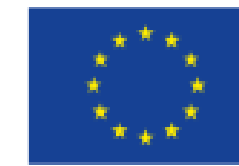
This is either
AF_UNIX or
AF_INET

Socket_type

This is either
SOCK_STREAM or
SOCK_DGRAM

Protocol

This is usually left
out, by default 0

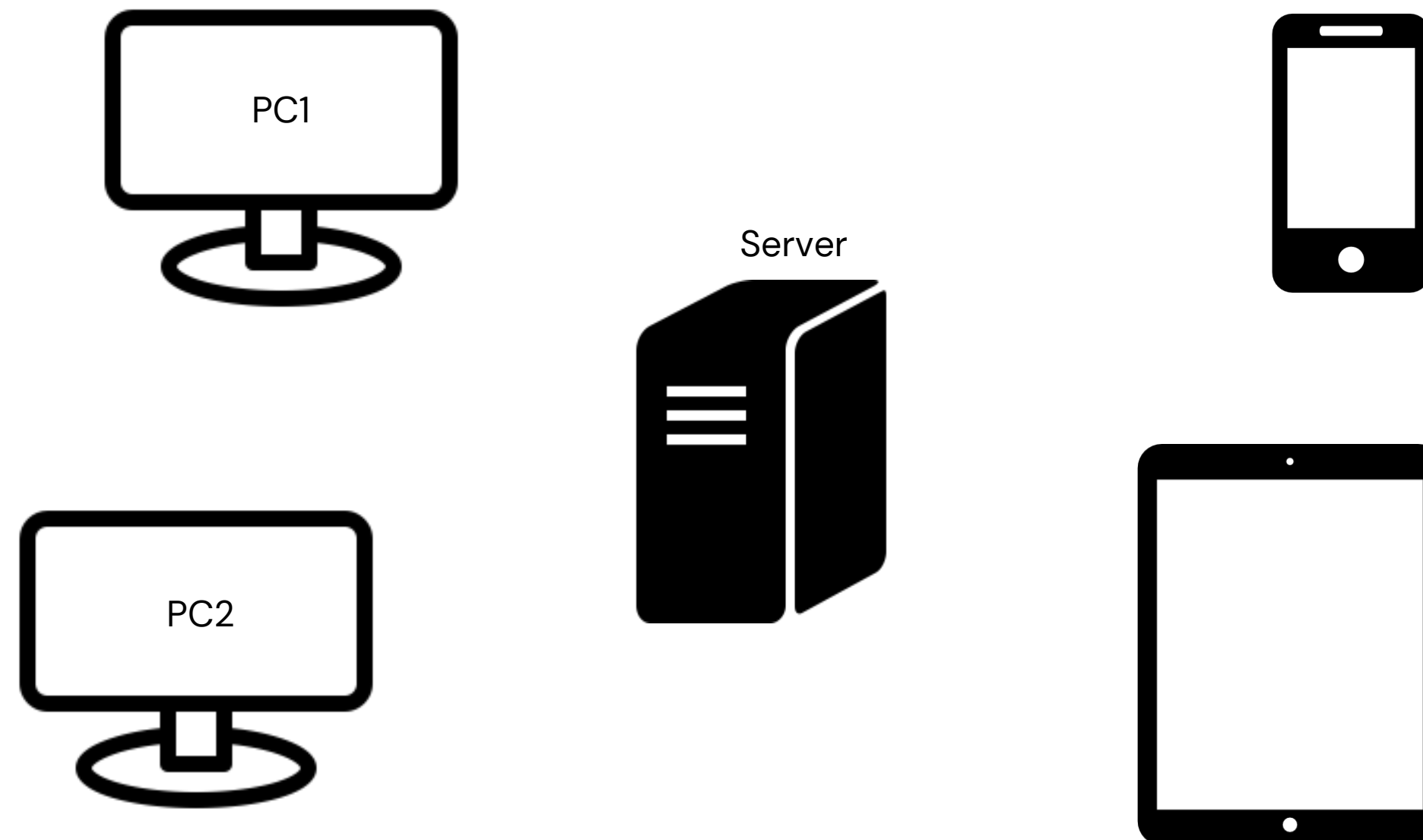


The `SO_REUSEADDR` option enables a socket to bind to a port that is currently in use by a socket in the `TIME_WAIT` state. This is particularly helpful when you need to restart a server and reuse the same port without waiting for the previous socket to fully close.

```
#set the REUSEADDR option  
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

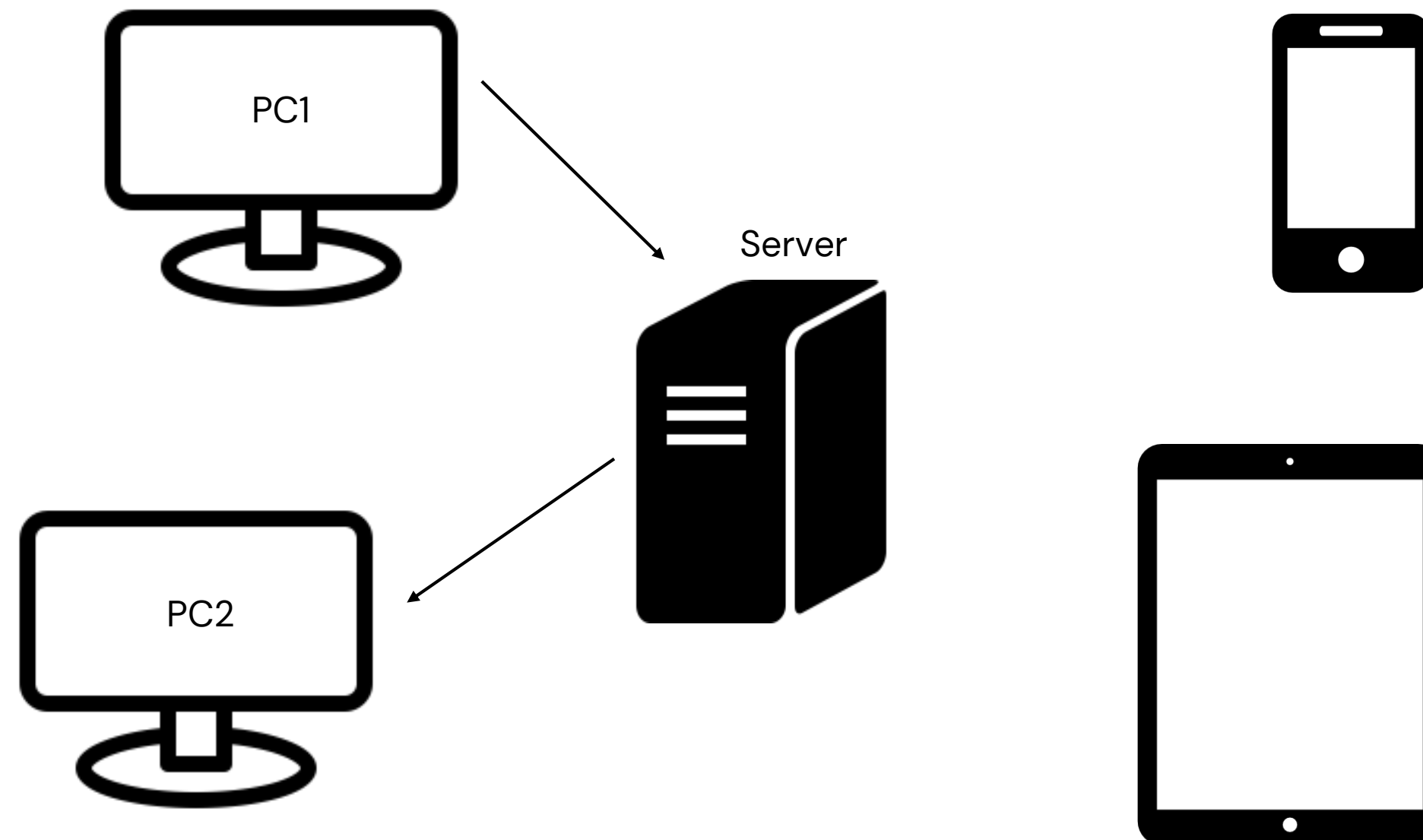
Client-Server Architecture

It is basically the way we structure a program with clients and servers. A simple example:



PC1 has to send a request to the server that it would like to communicate with PC2. PC2 will receive this from the server.

The important thing is that PC1 is not sending directly the message to PC2.



Servers

01

Servers are either a program, a device or a computer that are devoted to managing network resources.

02

A server is a program that waits for connections from clients.

Server

Socket()

Bind()

Listen()

Accept()

Recv()

Send()

Close()

A server does the following:

Socket()

The server creates a socket to listen for incoming connections.

Bind()

The server binds the socket to a specific IP address and port number, which clients will use to connect

Listen()

The server puts the socket into listening mode, waiting for clients to connect.

Accept()

When a client tries to connect, the server accepts the connection and creates a new socket to communicate with the client.

Recv() & Send()

The server and client can now send and receive data through the connected sockets.

Close()

After the communication is complete, the server closes the socket.

Clients

01

A client is a program that connects to a server to send or receive data.

02

The best example is a web browser

Client

Socket()

Connect()

Send()

Recv()

Close()

A client does the following:

Socket()

The client creates a socket to connect to the server

Connect()

The client uses the server's IP address and port number to establish a connection.

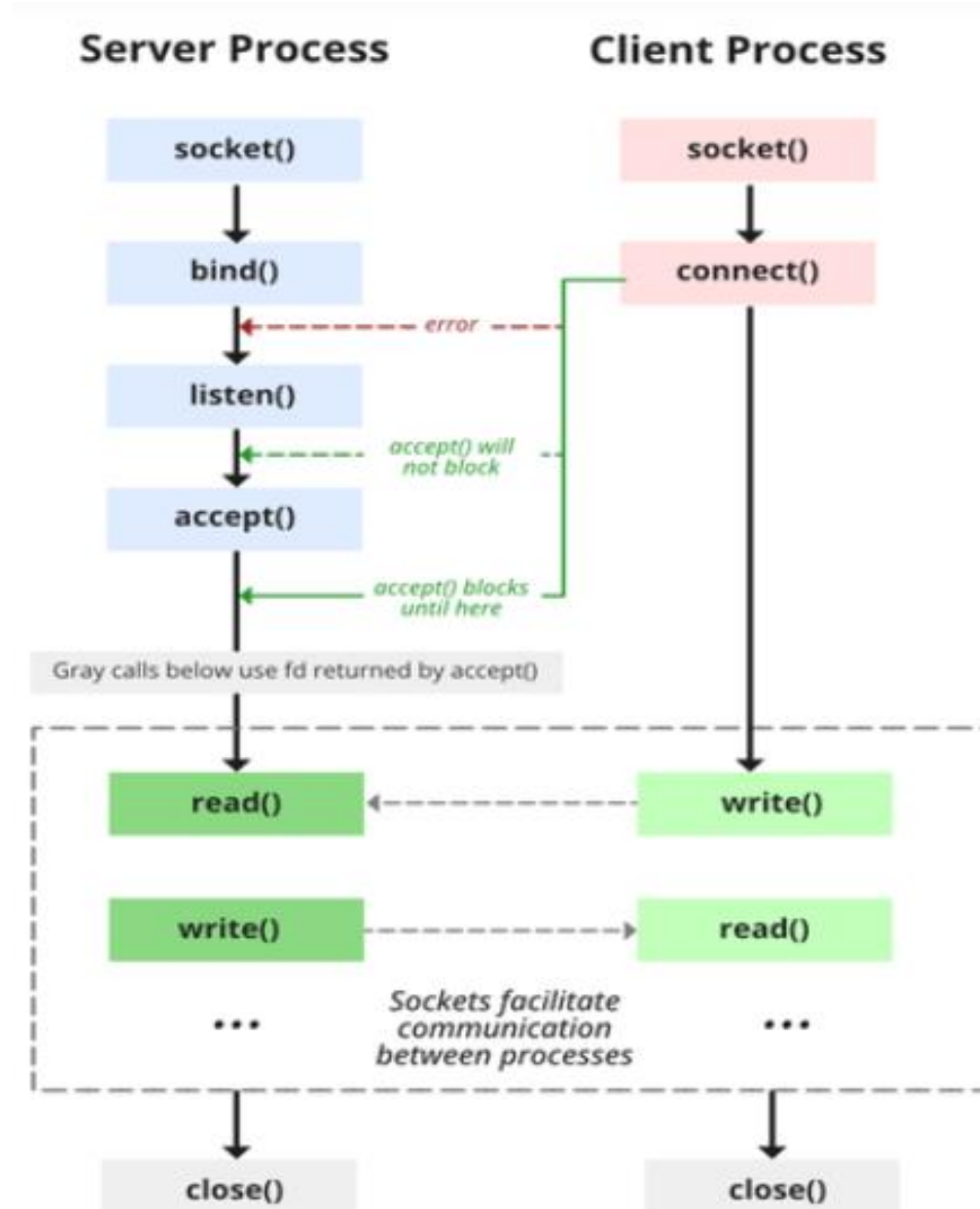
Send() & Recv()

Once connected, the client and server can send and receive data.

Close()

After the communication is done, the client closes the socket.

State diagram for client-server





Implementation examples

In this section we will show several examples of where we are exchanging :

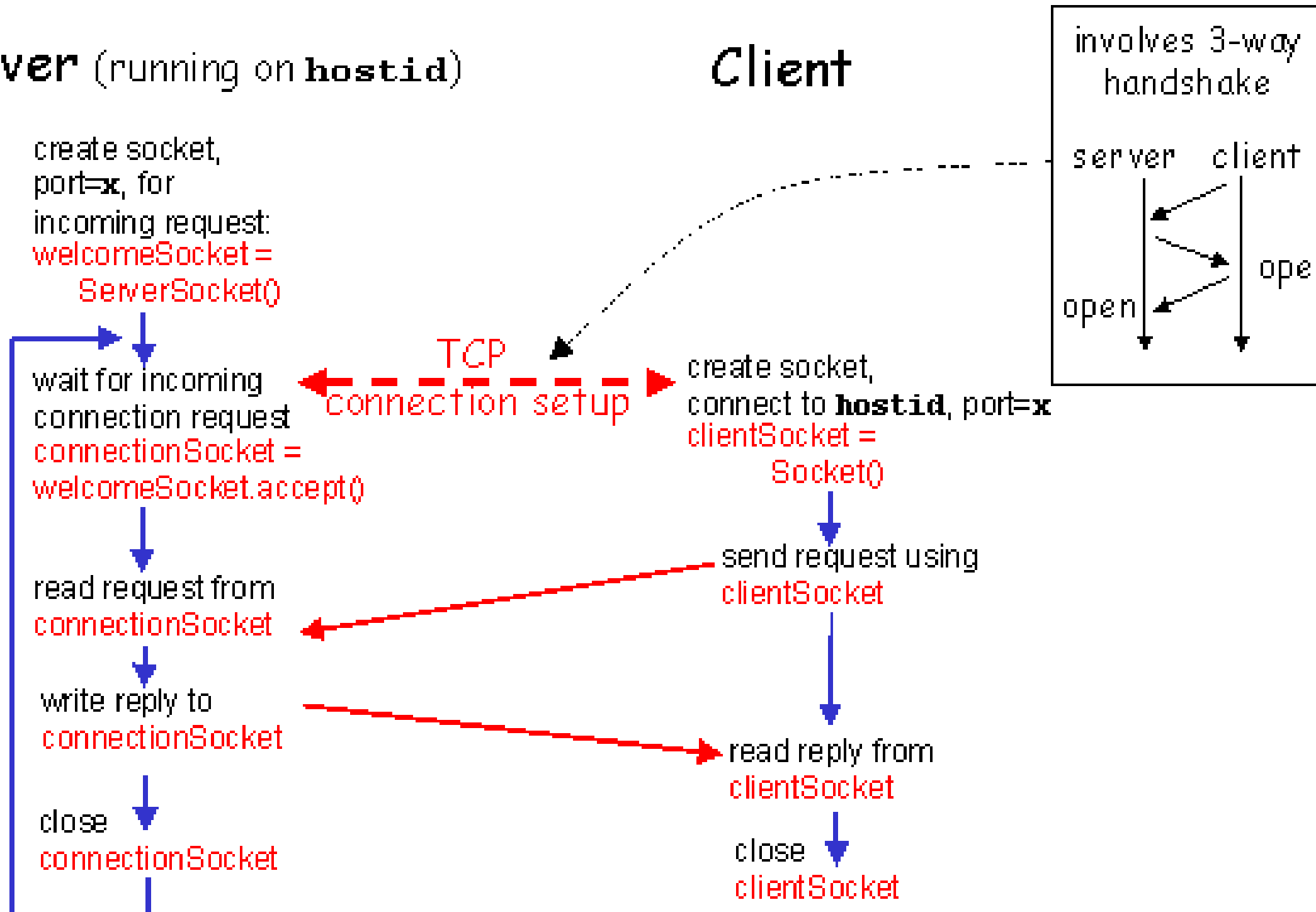
- ▶ one hello message,
- ▶ An image file
- ▶ A document file

between server and client to demonstrate the client/server model

Hello message example

Server (running on `hostid`)

Client



Hello message example – Server Code

```
import socket

# Create a socket (welcomeSocket) to listen for incoming connections
welcomeSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
welcomeSocket.bind(('0.0.0.0', 80))
welcomeSocket.listen(1)
print("Server is listening on port 80")

# Accept a connection from a client
connectionSocket, addr = welcomeSocket.accept()
print(f"Connected to {addr}")

# Read the request from the client
request = connectionSocket.recv(1024).decode()
print(f"Received request: {request}")

# Write a reply to the client
reply = "Hello, Client! Your request was received."
connectionSocket.send(reply.encode())

# Close the connection
connectionSocket.close()
welcomeSocket.close()
```



Hello message example – Client Code

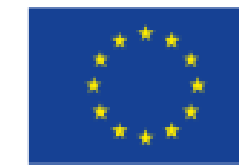
```
import socket

# Create a socket (clientSocket) to connect to the server
clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
clientSocket.connect(('10.11.34.24', 80))

# Send a request to the server
request = "Hello, Server!"
clientSocket.send(request.encode())

# Read the reply from the server
reply = clientSocket.recv(1024).decode()
print(f"Received reply: {reply}")

# Close the client socket
clientSocket.close()
```



Hello message output

01

Run your code on your server

02

Run your code on your client

Server Output

```
Server is listening on port 80  
Connected to 192.168.1.7  
Received request: Hello, Server!
```

Client Output

```
Received reply: Hello, Client! Your request was received
```



An image file example – Server Code

```
import socket
import sys

def send_image(image_path, host, port):
    # Create a socket object
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    try:

        # Bind to the address and port
        server_socket.bind((host, port))

        # Listen for incoming connections
        server_socket.listen(1)

        print("Server is listening on {}:{}".format(host, port))

        # Accept a connection
        client_socket, client_address = server_socket.accept()
        print("Connection from:", client_address)

        # Read the image file
        with open(image_path, 'rb') as f:
            image_data = f.read()

        # Send the image size
        client_socket.sendall(len(image_data).to_bytes(4, byteorder='big'))

        # Send the image data
        client_socket.sendall(image_data)

        print("Image sent successfully")

    finally:
        # Close the connection
        client_socket.close()
        server_socket.close()

if __name__ == "__main__":
    if len(sys.argv) != 2 :
        print("Usage: python server-image3.py <path file>")
        sys.exit(1)

    image_path = sys.argv[1] # Path from commandline
    host = '0.0.0.0' # Server IP address
    port = 443 # Port number

    send_image(image_path, host, port)
```



An image file example – Client Code

```
import socket
from PIL import Image
from io import BytesIO

def receive_image(host, port):
    # Create a socket object
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    try:
        # Connect to the server
        client_socket.connect((host, port))

        # Receive the image size
        image_size_bytes = client_socket.recv(4)
        image_size = int.from_bytes(image_size_bytes, byteorder='big')

        # Receive the image data
        image_data = b''
        while len(image_data) < image_size:
            chunk = client_socket.recv(min(image_size - len(image_data), 4096))
            if not chunk:
                break
            image_data += chunk

        # Convert image data to PIL Image object
        img = Image.open(BytesIO(image_data))
        # Get image format
        image_format = img.format

        # Write the image data to a file
        save_path = "received_image."+image_format;

        with open(save_path, 'wb') as f:
            f.write(image_data)

        print("Image received and saved successfully")

    finally:
        # Close the connection
        client_socket.close()

if __name__ == "__main__":
    host = '10.11.34.19' # Server IP address
    port = 443 # Port number
    receive_image(host, port)
```

An image file output

01

Run the server-side script from the command line with the path to the image

```
python server-image.py path/to/image.jpg
```

02

Run the client-side script on your client

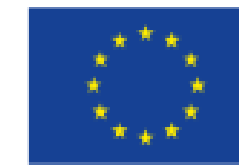
Server Output

```
Server is listening on 0.0.0.0:443  
Connection from: ('10.11.34.19', 443)  
Image sent successfully
```

Client Output

```
Image received and saved successfully
```

The client code connects to the server, receives the image, and saves it as “received_image.[appropriate format]”



A document file example – Server Code

```
import socket

def send_file(filename, host, port):
    try:
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # Bind to the address and port
        server_socket.bind((host, port))

        # Listen for incoming connections
        server_socket.listen(1)

        print("Server is listening on {}:{}".format(host, port))

        client_socket, client_address = server_socket.accept()
        print("Connection from:", client_address)

        with open(filename, 'rb') as file:
            data = file.read(1024)
            while data:
                client_socket.send(data)
                data = file.read(1024)

        print("File sent successfully!")
        client_socket.close()

    except Exception as e:
        print("Error:", e)

    finally:
        server_socket.close()

# Usage

if __name__ == "__main__":
    host = '0.0.0.0' # Server IP address
    port = 8080 # Port number
    send_file("winter.txt", host, port) # Replace IP with Raspberry Pi's IP
```



A document file example – Client Code

```
import socket

def receive_file(filename, server_ip, server_port):
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

    try:
        client_socket.connect((server_ip, server_port))

        with open(filename, 'wb') as file:
            while True:
                file_data = client_socket.recv(1024)
                if not file_data:
                    break
                file.write(file_data)

        print("File received successfully!")

    except Exception as e:
        print("Error:", e)

    finally:
        client_socket.close()

if __name__ == "__main__":
    host = '10.11.34.19' # Server IP address
    port = 8080 # Port number
    receive_file("received_file.txt", host, port) # Replace IP with Kali Linux's IP
```

An image file output

01

Run the server-side script from the command line with the path to the image

```
python server-file.py
```

02

Run the client-side script on your client

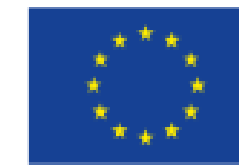
Server Output

```
Server is listening on 0.0.0.0:8080  
Connection from: 10.11.34.19  
File sent successfully!
```

Client Output

```
File received successfully!
```

The client code connects to the server, receives the .txt and saves it as "received_file.txt"



Resources

1. **"Python Network Programming Cookbook" by Dr. M. O. Faruque Sarker**
2. **"Computer Networking: A Top-Down Approach" by James F. Kurose and Keith W. Ross**
3. **"Unix Network Programming" by W. Richard Stevens**
4. **"Foundations of Python Network Programming" by Brandon Rhodes and John Goerzen**
5. **Python.org - Socket Programming HOWTO (<https://docs.python.org/3/howto/sockets.html>)**



**FACULTY OF
ENGINEERING**

RL4Eng



Co-funded by the
Erasmus+ Programme
of the European Union

THANK YOU