Network Programming in Python

Maram Bani Younes

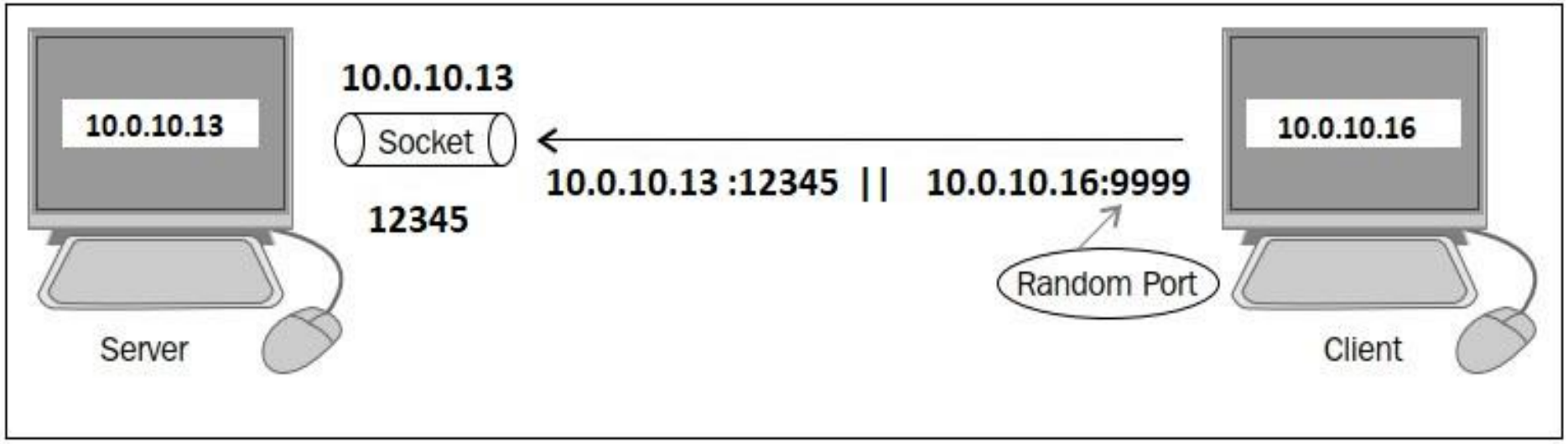
IP Addresses and Ports

- IP addresses are the addresses which help to uniquely identify a device over the internet.
- Port is an endpoint for the communication in operating systems.
- There are total of **0 – 65535** ports on a system.
- Port numbers are sometimes seen in web or other uniform resource locators (URLs) as well.
- By default**, HTTP uses port 80** and **HTTPS uses port 443**, but a URL like http://www.example.com:8080/path/ specifies that the web browser connects instead to port 8080 of the HTTP server.

Network sockets

- A network socket is one endpoint in a communication flow between two programs running over a computer network such as the Internet.
- Sockets are used to send messages across the network.
- Sockets can be used over different channels such as TCP, UDP.
- The socket connection can be used in the following two modes:

  - **Server**
  - **Client**



Socket functions

- **gethostbyname()** Retunes the IP address of a website or name device on the network.

```
import socket
ip= socket.gethostbyname("www.google.com")
print(ip)
```

```
import socket
ip2= socket.gethostbyname("localhost")
print(ip2)
```

Socket functions

- gethostbyaddr() returns the name of a website or device on the network.

```
host=socket.gethostbyaddr(str(ip))
print(host)
```

```
host2=socket.gethostbyaddr("127.0.0.1")
print(host2)
```

Socket functions

- getservbyname() returns the port number that provides the service.

```
port = socket.getservbyname("ftp") # http, ftp,telnet, ftp-data
print(port)
```

Socket functions

- getservbyport() returns the service of the port number.

```
serv = socket.getservbyport(20) # 20, 21, 80,
print(serv)
```

Network sockets

- Network socket can be identified by a unique combination of an IP address and port number
- In a very simple way, a socket is a way to talk to other computers. By means of a socket, a process can communicate with another process over the network.
- In order to create a socket, use the **socket.socket()** function that is available in the socket module. The general syntax of a socket function is as follows:

s = socket.socket (socket_family, socket_type, protocol=0)

Network sockets

s = socket.socket (socket_family, socket_type, protocol=0)

- To import the socket library and make a simple socket.
- Following are the different parameters used while making socket

- **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, defaulting to 0.

Network sockets

- The description of the parameters:

socket_family: socket.AF_INET, PF_PACKET

- **AF_INET** is the address family for IPv4.
- **AF_UINX** is the address family for IPv6

- These arguments represent the address families and the protocol of the transport layer:

Socket Type :

1. **socket.SOCK_DGRAM**
2. **socket.SOCK_RAW**
3. **socket.SOCK_STREAM**

Network sockets

Socket Methods

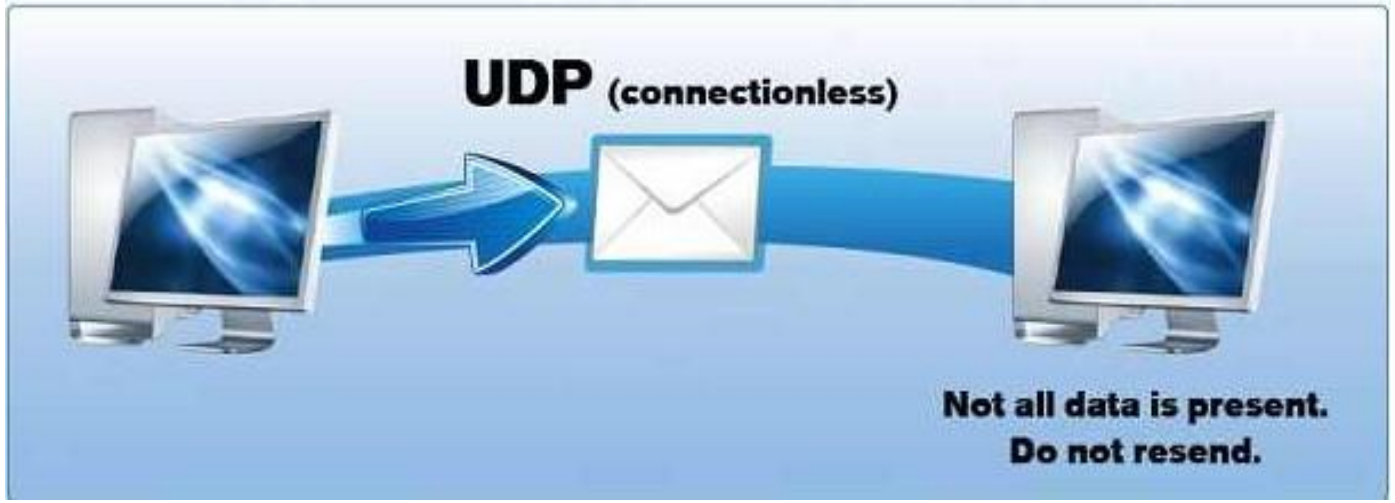
The three different set of socket methods are:

- Server Socket Methods
- Client Socket Methods
- General Socket Methods

Network sockets

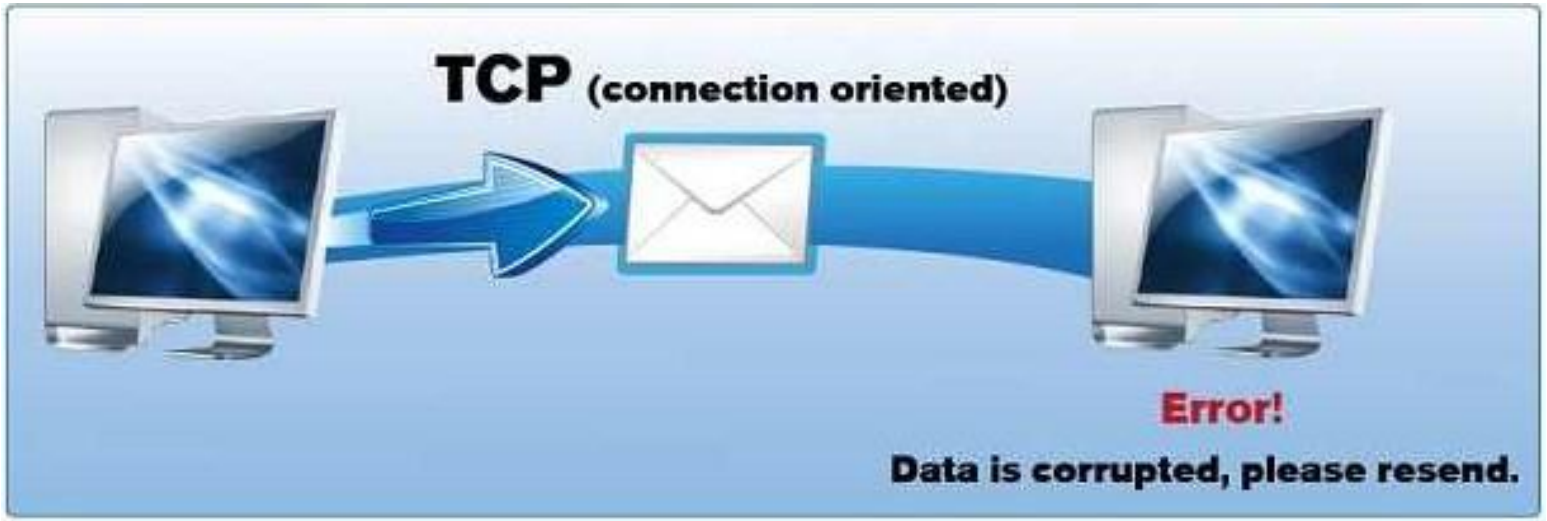
- Type means the kind of communication between two endpoints.

- **socket.SOCK_DGRAM :**depicts that UDP is unreliable and for connectionless protocols.



Network sockets

- **socket.SOCK_STREAM :**depicts TCP is reliable and is a two-way, connection-based service, and for connection-oriented protocols .



Network sockets

- There are two type of sockets: SOCK_STREAM and SOCK_DGRAM. Below we have a comparison of both types of sockets.

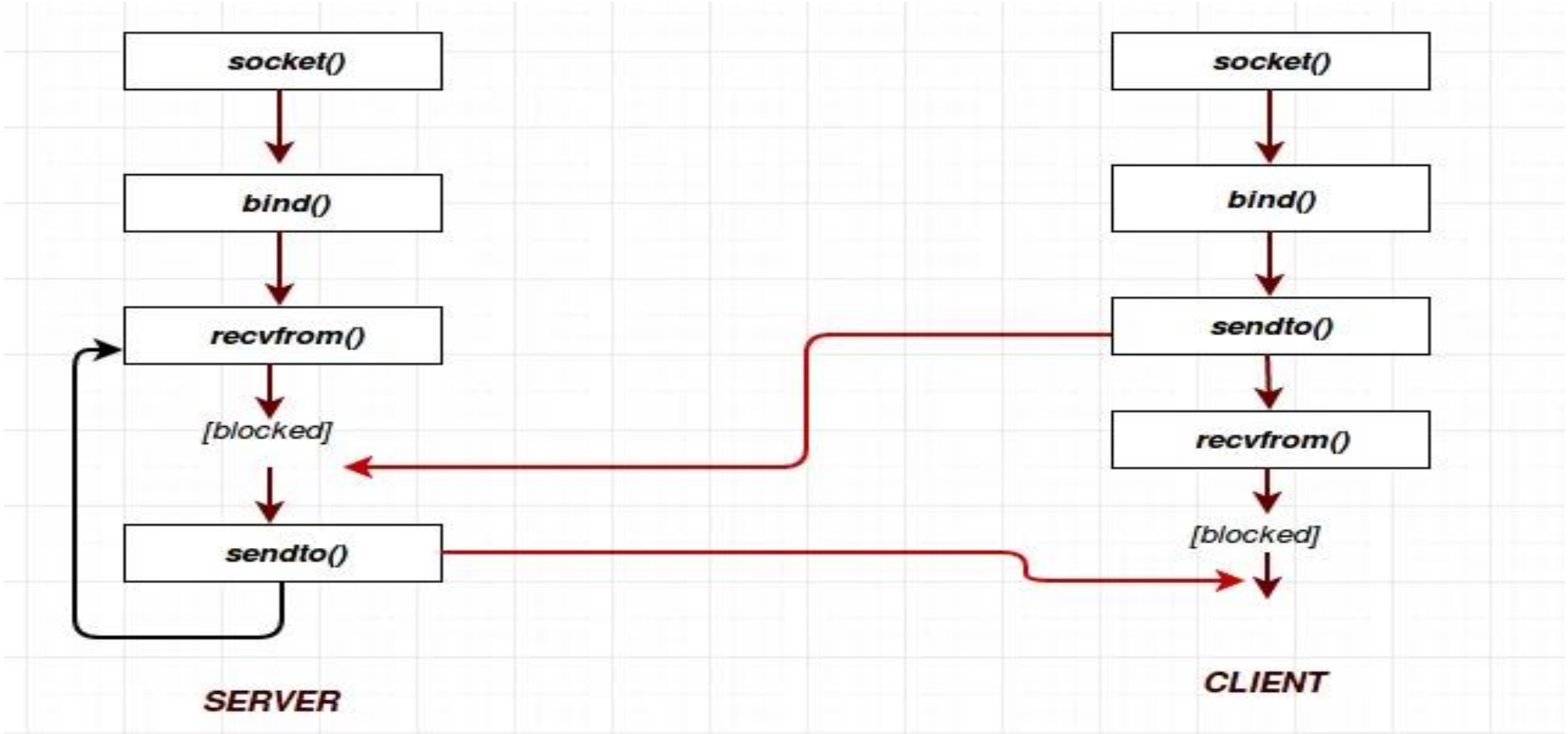
SOCK_STREAM	SOCK_DGRAM
For TCP protocols	For UDP protocols
Reliable delivery	Unrelible delivery
Guaranteed correct ordering of packets	No order guaranteed
Connection-oriented	No notion of connection(UDP)
Bidirectional	Not Bidirectional

Network sockets

TCP Socket Methods	UDP Socket Methods
s.recv()→ Receives TCP messages	s.recvfrom()→ Receives UDP messages
s.send()→ Transmits TCP messages	s.sendto()→ Transmits UDP messages

Network sockets

- **UDP Sockets**



Server socket methods

- **socket.bind(address):** This method is used to connect the address (IP address, port number) to the socket. The socket must be open before connecting to the address.

- **socket.listen(q):** This method starts the TCP listener. The q argument defines the maximum number of lined-up connections.

- **socket.accept():** The use of this method is to accept the connection from the client.
  - ✓ Before using this method, the socket.bind(address) and socket.listen(q) methods must be used.
  - ✓ This method returns two values: client_socket and address:
    - ➢ client_socket : is a new socket object used to send and receive data over the connection.
    - ➢ Address: is the address of the client. You will see examples later.

Client socket methods

- The only method dedicated to the client is the following:

- **socket.connect(address):** This method connects the client to the server. The address argument is the address of the server.

General socket methods

- **socket.send(bytes):** This method is used to send data to the socket. Before sending the data, ensure that the socket is connected to a remote machine. It returns the number of bytes sent.

- **socket.sendto(data, address) :** As name implies, this method is used to send data from the socket. Two pair (data, address) value is returned by this method. Data defines the number of bytes sent and address specifies the address of the remote machine.

- **socket.sendall(data):**This method sends all the data to the socket which is connected to a remote machine. It will carelessly transfers the data until an error occurs and if it happens then it uses socket.close() method to close the socket.

General socket methods

- **socket.recvfrom(bufsize):** This method receives data from the socket. The method returns a pair of values:
  - The first value gives the received data.
  - The second value gives the address of the socket sending the data.

- **socket.recv(bufsize):** This method receives a TCP message from the socket. The bufsize argument defines the maximum data it can receive at any one time.

- **socket.gethostname():** This method will return the name of the host.

- **socket.close():** This method will close the socket.

General “UDP Client – Server” Example

- Server:

```
#server
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
print("1")
s.bind(("127.0.0.1", 5550))
print("2")
Rdata, address = s.recvfrom(1024)
print("3")
Rdata=Rdata.decode("UTF-8")
print(str(Rdata))
print("4")
Sdata="Hello Client"
Sdata=Sdata.encode("UTF-8")
print("5")
s.sendto(Sdata, address) # s.sendto(data, ("0.0.0.0",5409))
print("6")
```

- Client:

```
#client
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
print("1")
data="hello Server"
print("2")
s.sendto(data.encode("UTF-8"), ("127.0.0.1",5550)) # s.sendto(data, ("0.0.0.0",5409))
print("3")
Rdata, address = s.recvfrom(1024)
Rdata=Rdata.decode("UTF-8")
print("4")
print(Rdata, address)
print("5")
```

UDP Client – Server Examples.

- Two different devices.
  - Check IP address of your devices on the local network→ IPConfig
- Chat Client Server
- A server to provide time and date to the clients *datetime.now()*
- Server provide some services to the Client such Calculator.

Chat Client Server

```
#server
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
print("1")
s.bind(("127.0.0.1", 5550))
print("2")
while True:
    Rdata, address = s.recvfrom(1024)
    print("3")
    Rdata=Rdata.decode("UTF-8")
    print(str(Rdata))
    print("4")
    Sdata=input("enter your response\n: ")
    Sdata=Sdata.encode("UTF-8")
    print("5")
    s.sendto(Sdata, address) # s.sendto(data, ("0.0.0.0",5409))
    print("6")
```

```
#client
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
print("1")
while True:
    data=input("enter your message:\n")
    print("2")
    s.sendto(data.encode("UTF-8"), ("127.0.0.1",5550))
    print("3")
    Rdata, address = s.recvfrom(1024)
    Rdata=Rdata.decode("UTF-8")
    print("4")
    print(Rdata, address)
    print("5")
```

Time and date Server

```python
#server
import socket
from datetime import datetime
s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
print("1")
s.bind(("127.0.0.1", 5550))
print("2")
Rdata, address = s.recvfrom(1024)
print("3")
Rdata=Rdata.decode("UTF-8")
print("4")
Time=str(datetime.now())
Sdata=Time.encode("UTF-8")
print("5")
s.sendto(Sdata, address) # s.sendto(data, ("0.0.0.0",5409))
print("6")
```

```python
#client
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
print("1")
data="hey server, Set my time"
print("2")
dataE=data.encode("UTF-8")
C_address= ("127.0.0.1",5550)
s.sendto(dataE,C_address)
print("3")
Rdata, address = s.recvfrom(1024)
Rdata=str(Rdata.decode("UTF-8"))
print("4")
print("the Current time is: %s" %Rdata)
print("5")
```

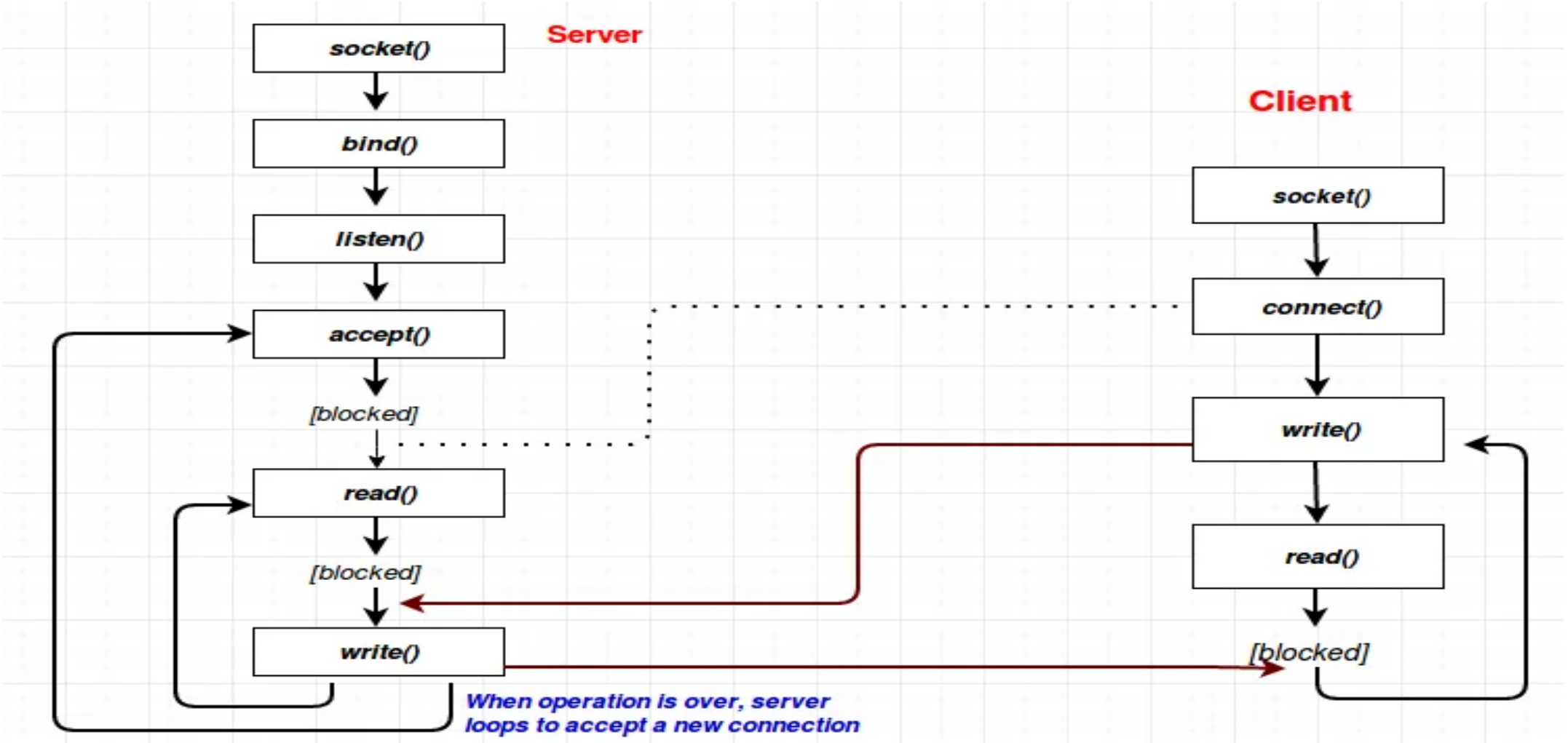
Calculator Server (HW)

- Develop a client-server code using UDP
- The client sent a simple mathematical Computation to the server e.g., 5 + 6 or 4 * 5.

  Hint: you can write (+, 5, 6)instead of 5 + 6 if you want
- The server return the result to the client. 5+6 = 11

Network sockets

- **TCP Sockets**



“TCP Client – Server” Example

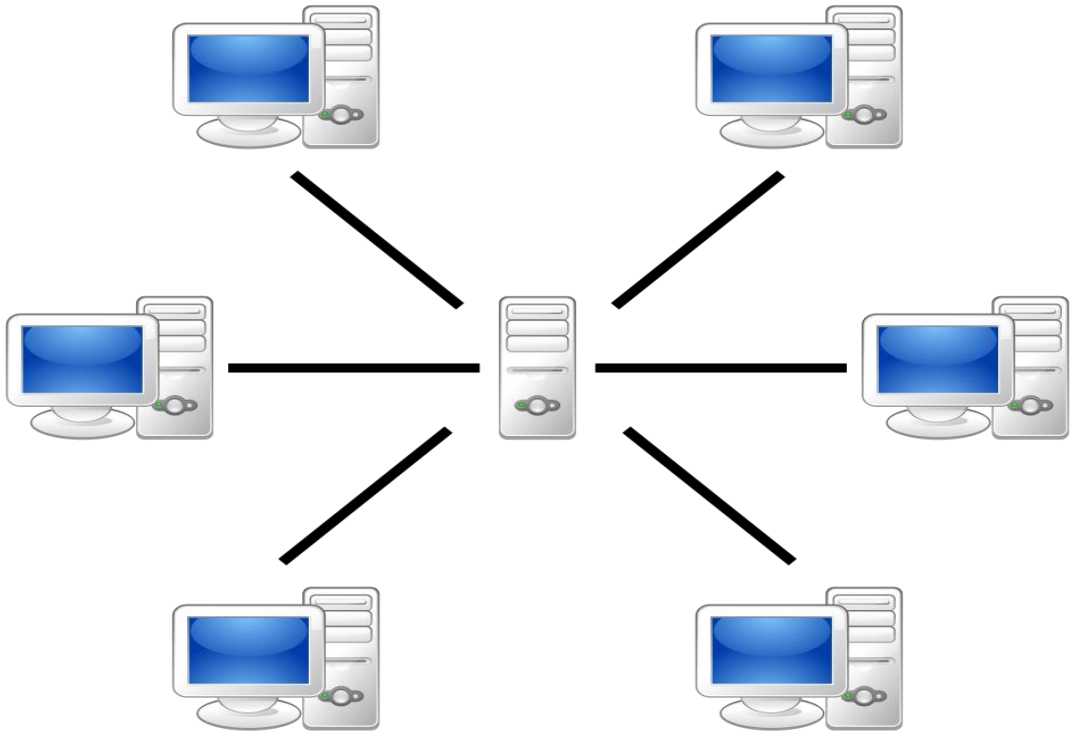
```python
#server
import socket
from datetime import datetime
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("1")
s.bind(("127.0.0.1", 5550))
print("2")
s.listen(5) # number of clients at the moment
client, address = s.accept()
Request = client.recv(1024)
print("3")
Rdata=Request.decode("UTF-8")
print("4")
print(str(Rdata))
Time=str(datetime.now().time())
print(Time)
Sdata=Time.encode("UTF-8")
print("5")
client.send(Sdata) # s.sendto(data, ("0.0.0.0",5409))
print("6")
client.close()
```

```python
#client
import socket
client=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("1")
client.connect(("127.0.0.1",5550))
data="hey server, Set my time"
print("2")
dataE=data.encode("UTF-8")

client.send(dataE)
print("3")
Rdata = client.recv(1024)
Rdata=str(Rdata.decode("UTF-8"))
print("4")
print("the Current time is: %s" %Rdata)
print("5")
client.close()
```

Server socket methods

- In a client-server architecture, there is one centralized server that provides service, and many clients request and receive service from the centralized server.



TCP Example (HW)

- Develop a TCP client-server program where :
- Four clients send different string words to the server
- The server reply by the number of characters in each word and the beginning character of that word.

DNS servers and records.
What is DNS

- DNS: Domain Name System is the phonebook of the Internet.
- Humans access information online through domain names, like www.google.com or www.yahoo.com.
- Web browsers interact through Internet Protocol (IP) addresses.
- DNS translates domain names to IP addresses so browsers can load Internet resources.

What is a DNS record?

- DNS records (aka zone files) are instructions that live in authoritative DNS servers.
- It provides information about a domain including what IP address is associated with that domain and how to handle requests for that domain.
- These records consist of a series of text files written in what is known as DNS syntax.
- DNS syntax is just a string of characters used as commands that tell the DNS server what to do.
- All DNS records also have a ‘TTL’, which stands for time-to-live, and indicates how often a DNS server will refresh that record.

What are the most common types of DNS record?

- **A record** - The record that holds the IP address of a domain.
- **AAAA record** - The record that contains the IPv6 address for a domain .
- **CNAME record** - Forwards one domain or subdomain to another domain, does NOT provide an IP address.
- **MX** - Directs mail to an email server.
- **SOA record** - Stores admin information about a domain.
- **SRV record** - Specifies a port for specific services.
- **PTR record** - Provides a domain name in reverse-lookups.

Programming DNS records Examples

- To install the required library
    - pip install dnspython

DNS Example

```
import dns.resolver
R= dns.resolver.resolve("Yahoo.com","A", raise_on_no_answer=False)
for value in R:
    print("Host-->>%s" %value.to_text())
print(R)
```

```
Host-->>74.6.143.26
Host-->>74.6.231.20
Host-->>98.137.11.164
Host-->>98.137.11.163
Host-->>74.6.143.25
Host-->>74.6.231.21
<dns.resolver.Answer object at 0x000002DAA82B5160>
```