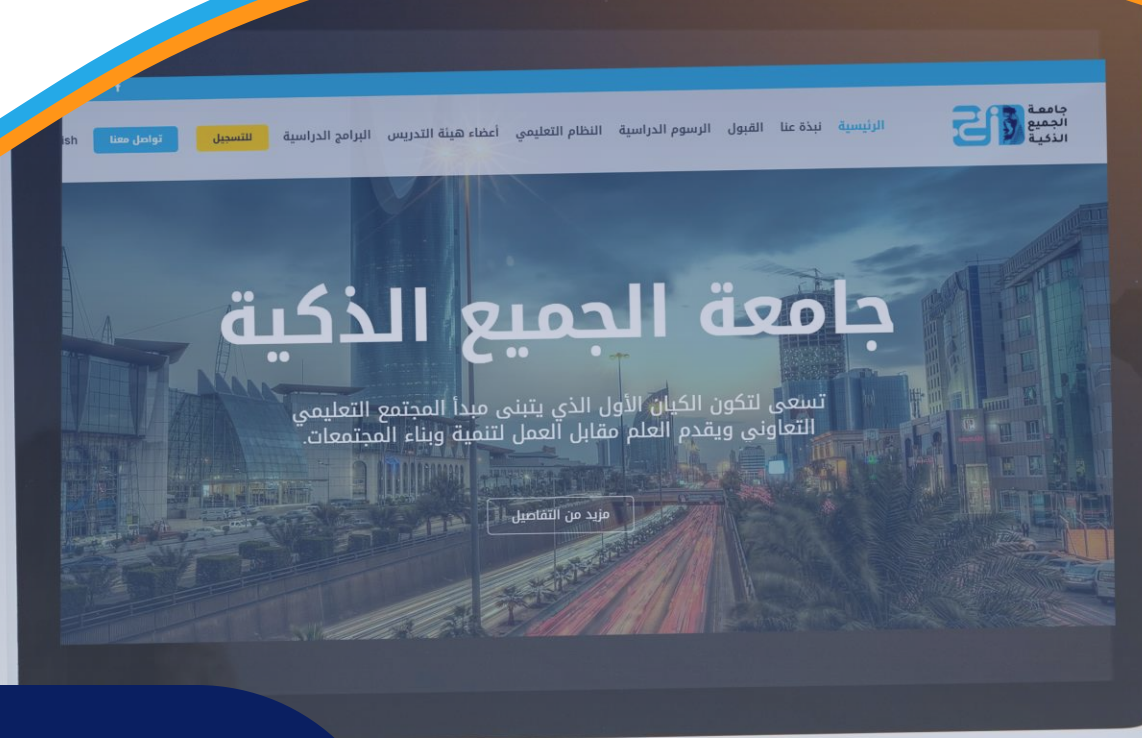




جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

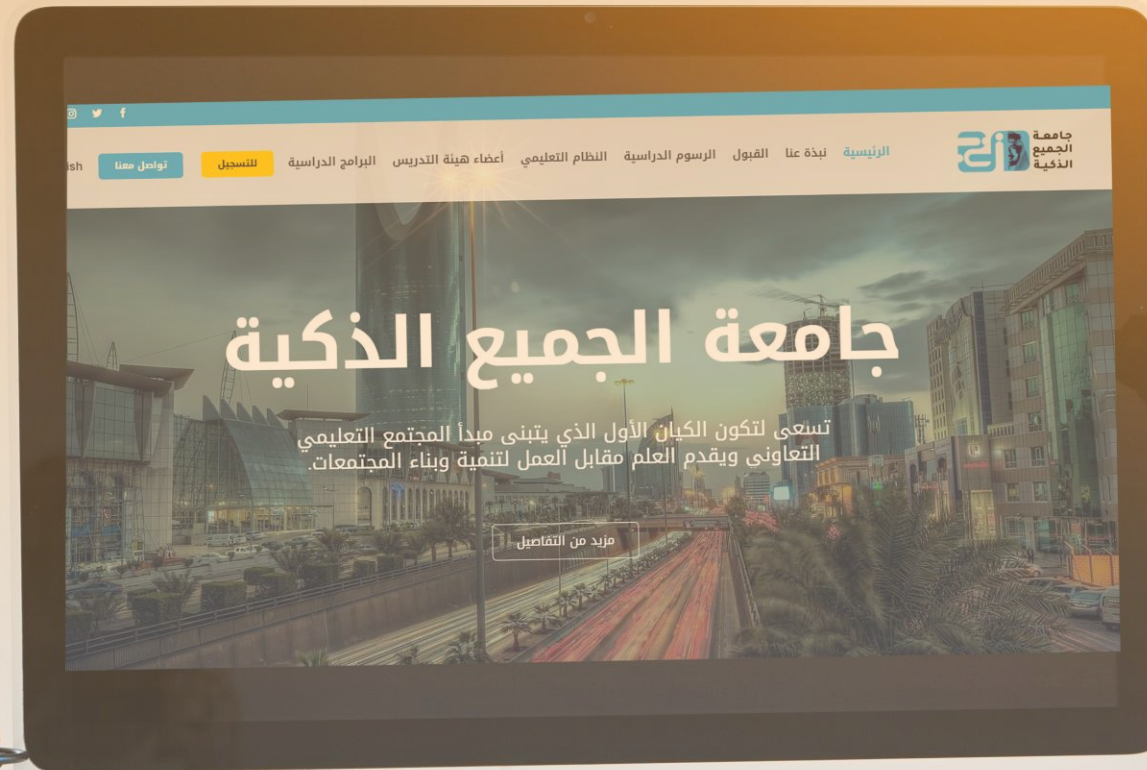


Prepared by
Dr. Hamada I. AbdulWakel
Lecture 5

CYB615
Network Programming



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

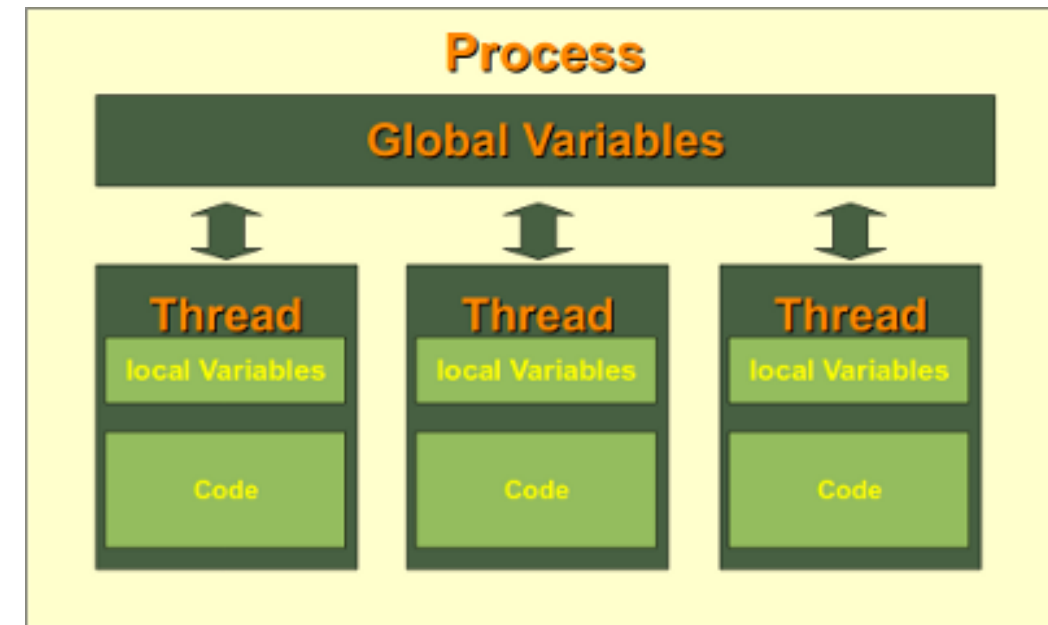


Module4: Socket Programming using Python – Part II

- Threading Programming.
- Multiple Client/Server Model.
- User Datagram Protocol.



- A Thread is defined in computer science as **the smallest unit** that can be scheduled in an operating system.
- More than one thread can exist within the same process.
- The ability of a process to execute multiple threads parallelly is called multithreading



- Advantages:
 - Significantly improve the performance of any program.
 - A program can remain responsive while one thread waits for input.
 - All the threads of a process have access to its global variables.
- Disadvantages:
 - On 1-CPU system, multithreading won't hit the speed of computation.
 - Synchronization for preventing mutual exclusion of shared resources.
 - Increases the complexity of the program.

Thread safe problem.



- This code segment is represents a **Python Function**.
- A function is a set of instructions that could be performed n times according to function calling in the code.

```
def fun1(a, b):  
    c = a + b  
    print(c)
```



```
import threading
```

```
def fun1(a, b):
```

```
    c = a + b
```

```
    print(c)
```

```
# args must be tuple
```

```
thread1 = threading.Thread(target = fun1, args = (12, 10))
```

```
thread1.start()
```



```
import threading, time

def fun1(a, b):
    time.sleep(1) #seconds
    c = a + b
    print(c)
def fun2(a):
    c = a * 2
    print(c)
```



....

```
thread1 = threading.Thread(target = fun1, args = (12, 10))  
thread1.start()
```

```
thread2 = threading.Thread(target = fun2, args = (10,))  
thread2.start()
```

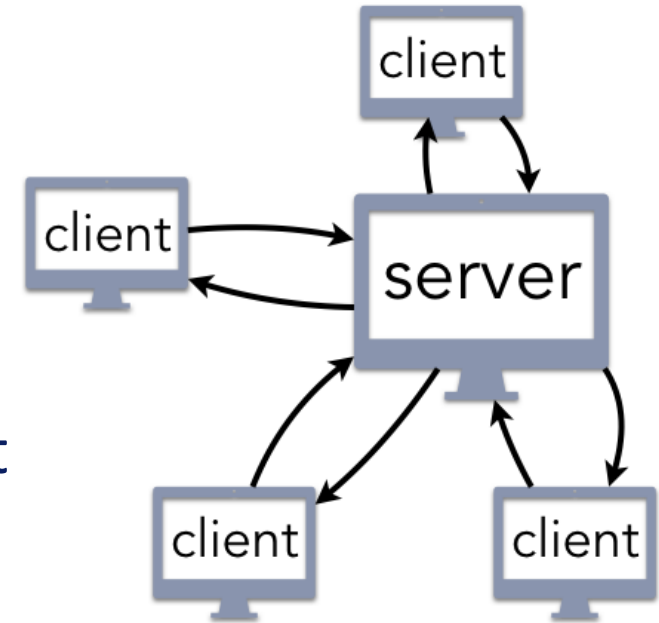
```
print("Total number of threads", threading.active_count() - 1)
```



Lecture 5

Multiple Client/Server Model

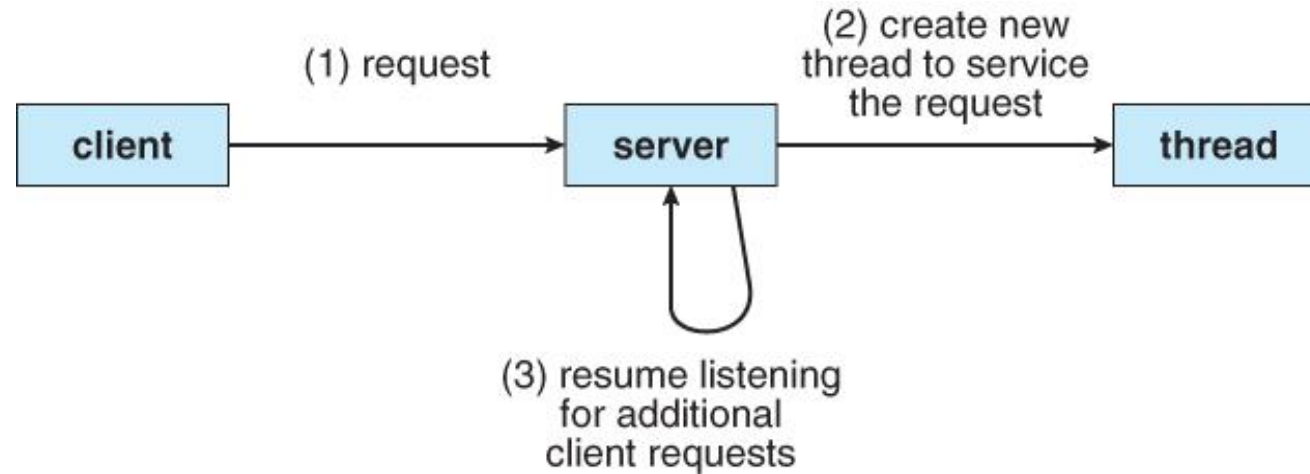
- In the basic model, server handles only one client at a time.
- Scalable server model?!
- Does not mean to implement a server for each client connection.



Lecture 5

Threading for Client/Server Model

- Thread-1: Listen and accept connections from other clients.
- Thread-N: Sending commands to an already connected clients.



Lecture 5

Server Using Threading

```
import socket, threading
```

```
PORT = 5050
```

```
SERVER = socket.gethostname(socket.gethostname())
```

```
print(SERVER)
```

```
ADDR = (SERVER, PORT)
```

```
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
server.bind(ADDR)
```

```
def send(msg, client):
```

```
client.send(msg.encode("utf-8"))
```



Lecture 5

Server Using Threading

Cont'd

```
def client(conn, addr, clients):  
    print(f"{addr} is connected.")  
    connected = True  
    while connected:  
        msg = conn.recv(1024).decode("utf-8")  
        if msg == "DES":  
            connected = False  
            continue  
        print(f"{addr} {msg}")  
        msg = f"Message received: {msg}"  
        for client in clients:  
            if client != conn:  
                send(msg, client)  
    conn.close()
```



Lecture 5

Server Using Threading

Cont'd

```
def start():  
    server.listen()  
    print(f"The server is listening to {SERVER}")  
    clients = []  
    while True:  
        conn, addr = server.accept()  
        clients.append(conn)  
        thread = threading.Thread(target=client, args=(conn, addr, clients))  
        thread.start()  
        print(f"Number of active connection: {threading.active_count() - 1}")  
  
print("The server is starting")  
start()
```



Lecture 5

Client Using Threading

```
import threading, socket
```

```
PORT = 5050
```

```
SERVER = socket.gethostname(socket.gethostname())
```

```
ADDR = (SERVER, PORT)
```

```
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
client.connect(ADDR)
```

```
def read():
```

```
    msg = client.recv(1024).decode("utf-8")
```

```
    print(msg)
```

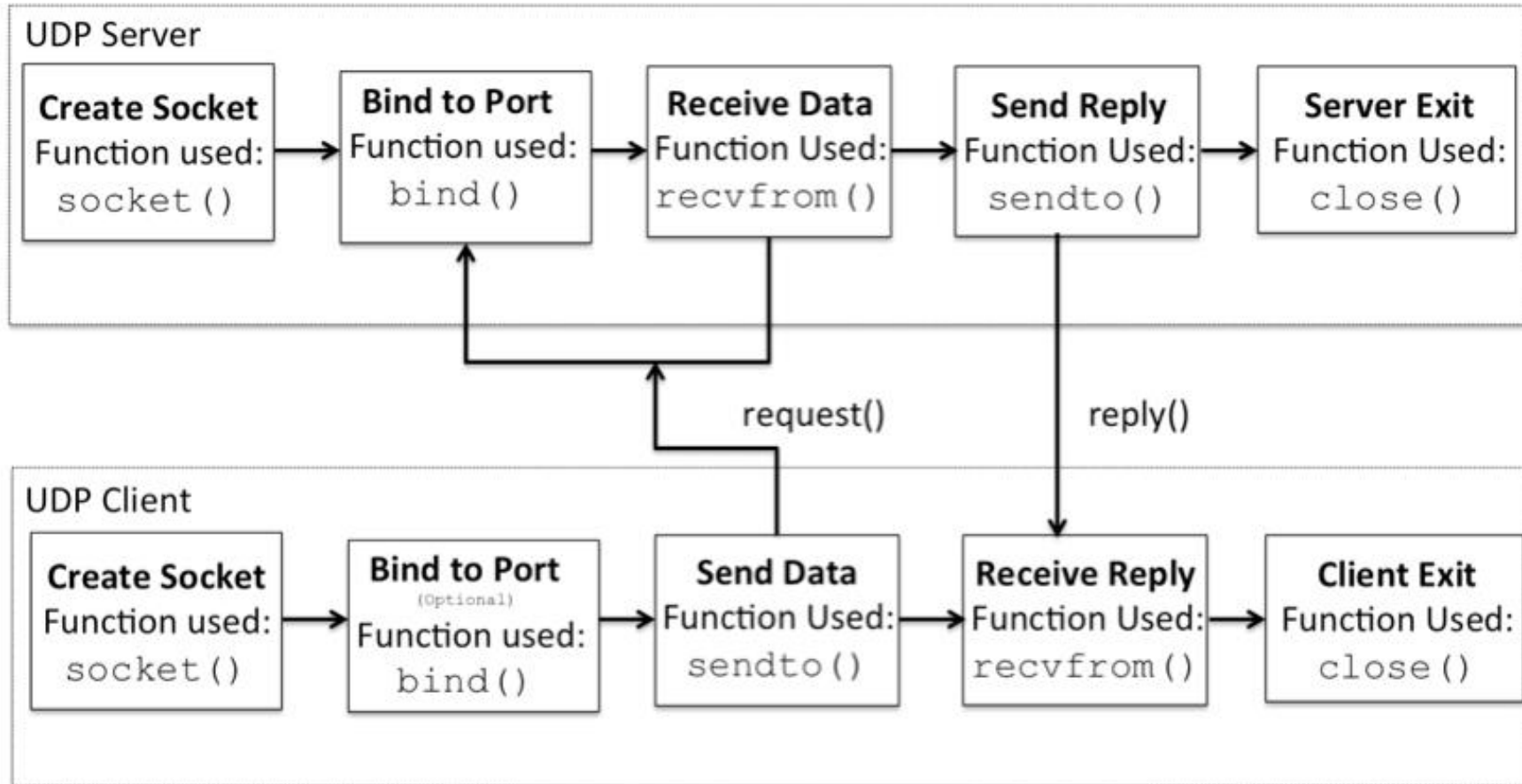


```
def send():
    connect = True
    while connect:
        msg = input()
        client.send(msg.encode("utf-8"))
        if msg == "DES":
            connect = False
        else:
            thread = threading.Thread(target=read)
            thread.start()
            continue
```



Lecture 5

User Datagram Protocol



Lecture 5

Simple Datagram Server

```
import socket
localIP = "127.0.0.1";    localPort = 20001
UDPServerSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
UDPServerSocket.bind((localIP, localPort))
print("UDP server up and listening")
while True:
    bytesAddressPair = UDPServerSocket.recvfrom(1024)
    message = bytesAddressPair[0].decode()
    address = bytesAddressPair[1]
    print(f"Message from Client: {message}")
    print(f"Client IP Address: {address}")
    message = message.upper().encode()
    UDPServerSocket.sendto(message, address)
```



Lecture 5

Simple Datagram Client

```
import socket

serverAddressPort = ("127.0.0.1", 20001)
UDPClientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
while True:
    data = input("Enter a word: ")
    data = data.encode()
    UDPClientSocket.sendto(data, serverAddressPort)
    msgFromServer = UDPClientSocket.recvfrom(1024)
    msg = f"Message from Server {msgFromServer[0].decode()}"
    print(msg)
```



- Exercise 1:

Create a UDP client/server program that calculate and send the factorial of a given number send by a client.

$$1! = 1$$

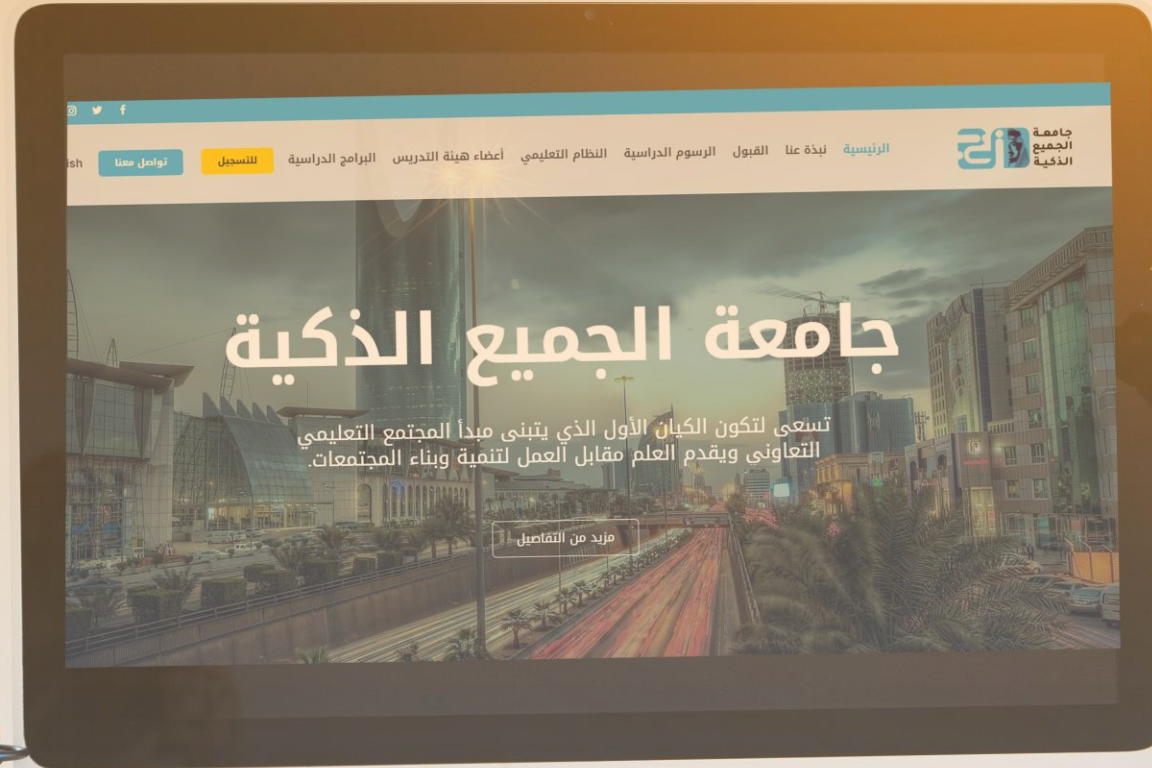
$$2! = 2(1) = 2$$

$$3! = 3(2)(1) = 6$$

$$4! = 4(3)(2)(1) = 24$$

$$5! = 5(4)(3)(2)(1) = 120$$





جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY

THANK YOU