CS305 Lab Tutorial
Lecture 4 Socket & HTTP

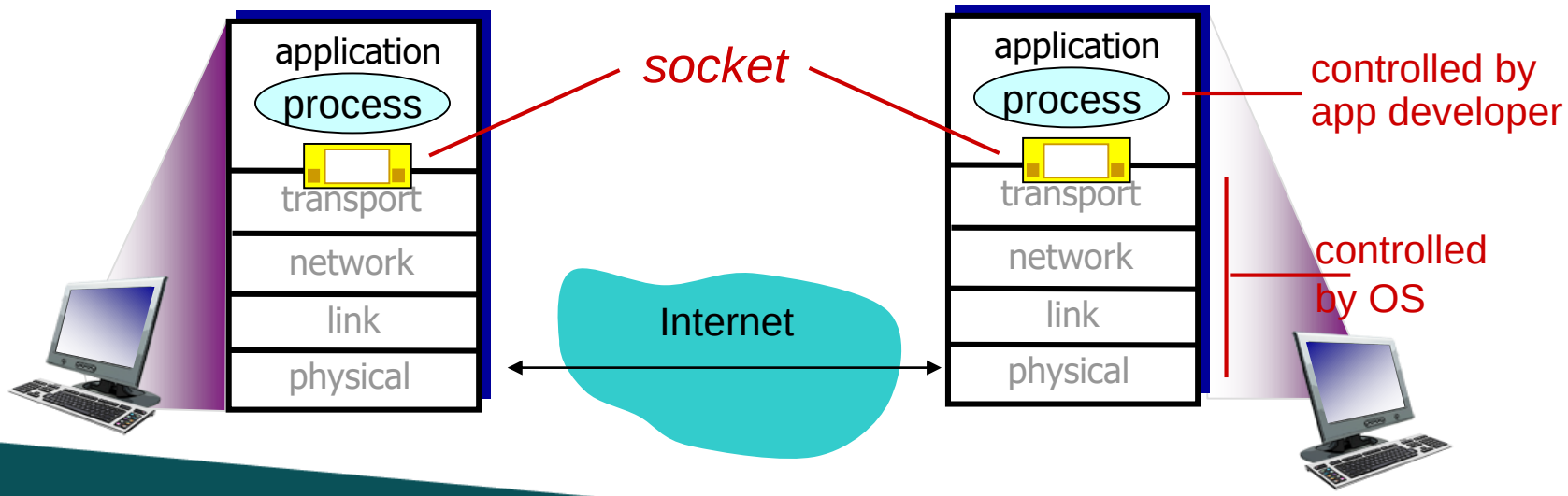
HHQ. ZHANG
Dept. Computer Science and Engineering
Southern University of Science and Technology

SUSTech
Southern University
of Science and Technology

Socket programming

goal: learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol



Socket programming

Two socket types for two transport services:

- *UDP:* unreliable datagram
- *TCP:* reliable, byte stream-oriented

Application Example:

1. client reads a line of characters (data) from its keyboard and sends data to server
2. server receives the data and converts characters to uppercase
3. server sends modified data to client
4. client receives modified data and displays line on its screen

Socket programming *with UDP*

UDP: no "connection" between client & server

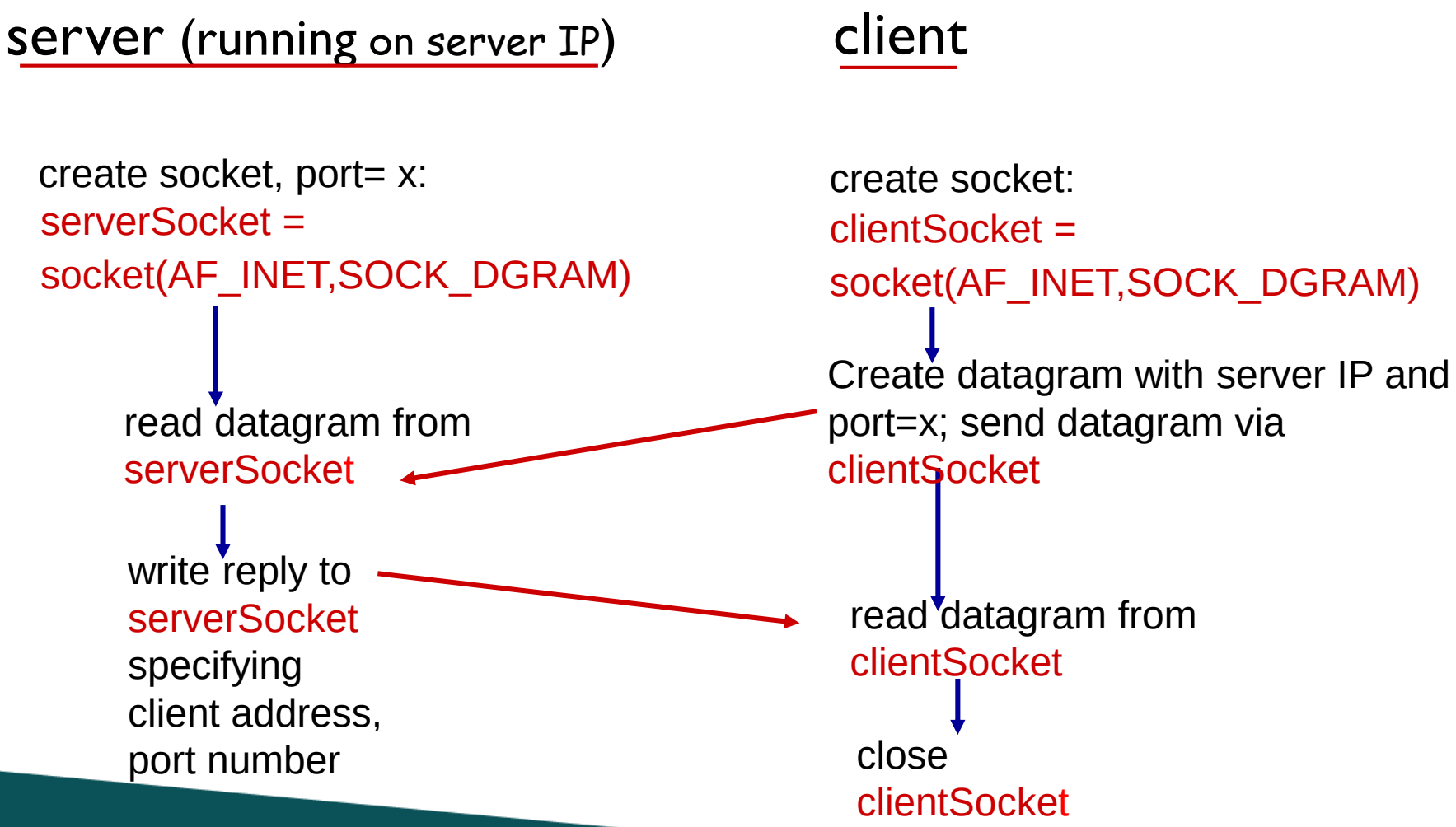
- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

- UDP provides *unreliable* transfer of groups of bytes ("datagrams") between client and server

Client/server socket interaction: UDP



Python UDPClient

include Python's socket library →

```
from socket import *
serverName = 'hostname'
serverPort = 12000
```

create UDP socket for client →

```
clientSocket = socket(AF_INET,
                      SOCK_DGRAM)
```

get user keyboard input →

```
message = input('Input lowercase sentence:')
```

Attach server name, port to message; send into socket →

```
clientSocket.sendto(message.encode(),
                    (serverName, serverPort))
```

read reply characters from socket into string →

```
modifiedMessage, serverAddress =
                    clientSocket.recvfrom(2048)
```

print out received string and close socket →

```
print(modifiedMessage.decode())
clientSocket.close()
```

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print ("The server is ready to receive")
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(),
                        clientAddress)
```

create UDP socket

bind socket to local port number 12000

loop forever

Read from UDP socket into message, getting client's address (client IP and port)

send upper case string back to this client

Socket programming *with TCP*

client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

client contacts server by:

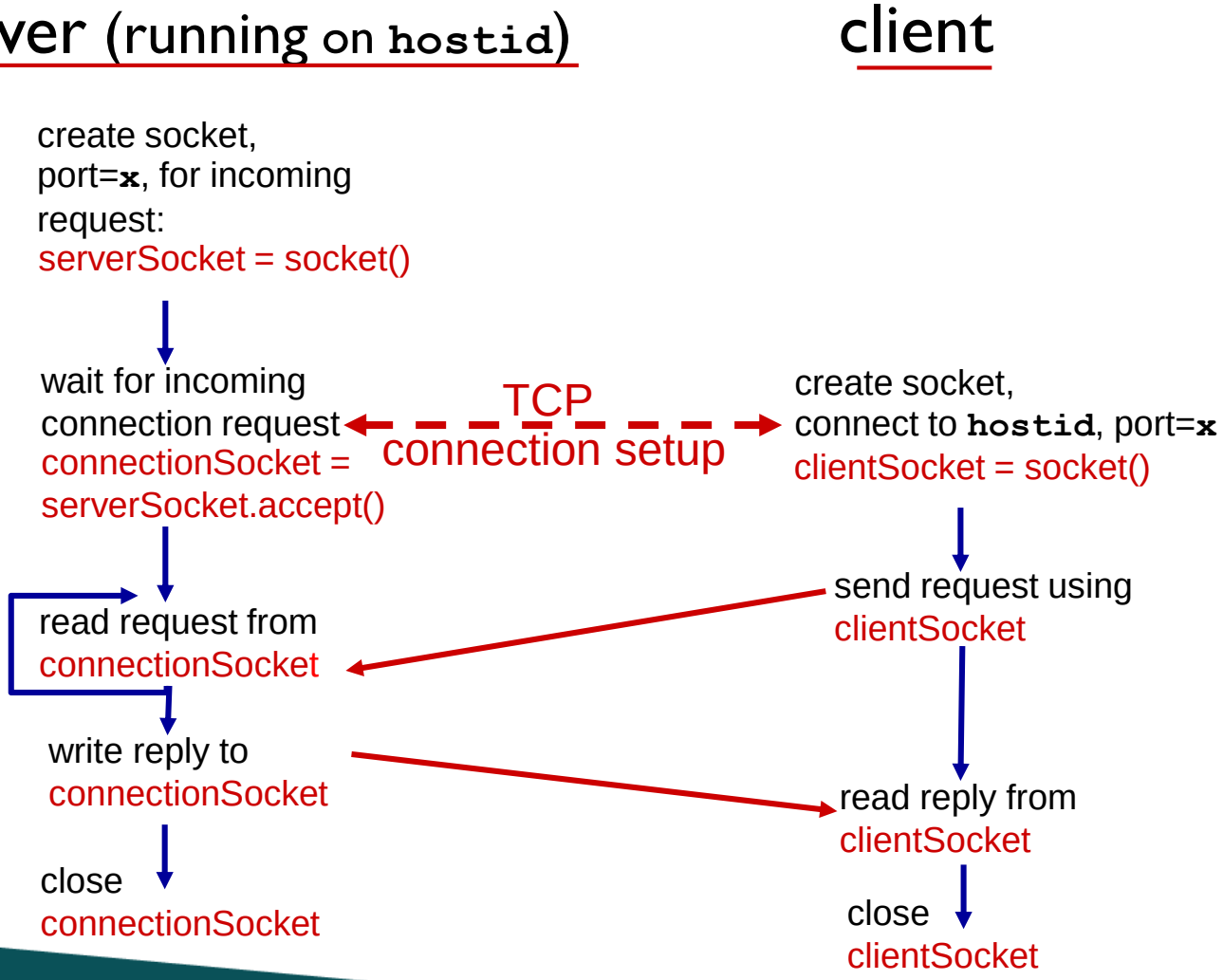
- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients

application viewpoint:

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

Client/server socket interaction: TCP



Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for server, remote port 12000

No need to attach server name, port

Python TCPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind((‘’,serverPort))
serverSocket.listen(1)
print (‘The server is ready to receive’)
while True:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.encode())
    connectionSocket.close()
```

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

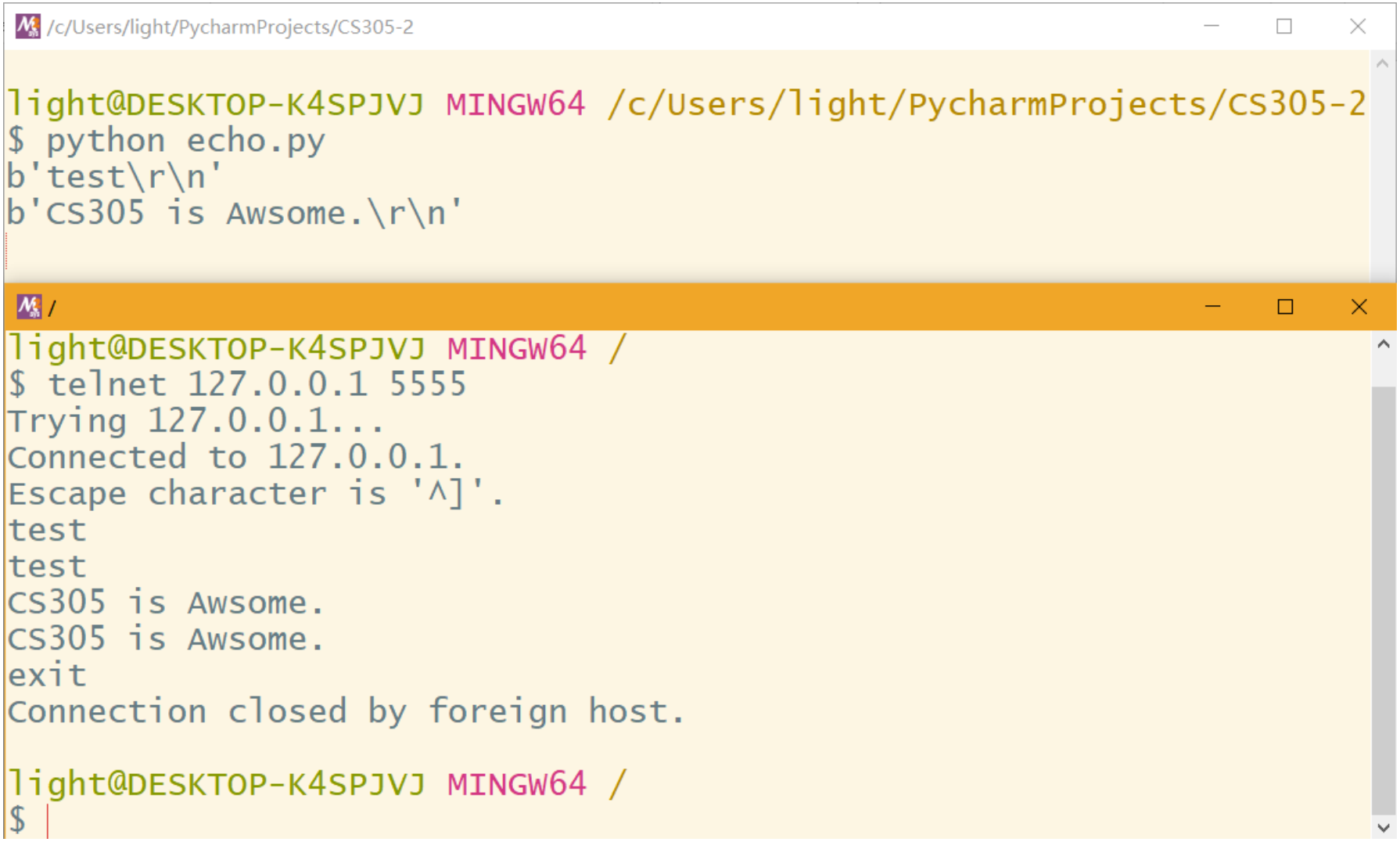
server waits on accept() for incoming requests, new socket created on return

read bytes from socket (but not address as in UDP)

close connection to this client (but *not* welcoming socket)

Example: Echo Server

```
import socket

def echo():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(('127.0.0.1', 5555))
    sock.listen(10)
    while True:
        conn, address = sock.accept()
        while True:
            data = conn.recv(2048)
            if data and data != b'exit\r\n':
                conn.send(data)
                print(data)
            else:
                conn.close()
                break

if __name__ == "__main__":
    try:
        echo()
    except KeyboardInterrupt:
        exit()
```


/c/Users/light/PycharmProjects/CS305-2
light@DESKTOP-K4SPJVJ MINGW64 /c/Users/light/PycharmProjects/CS305-2
$ python echo.py
b'test\r\n'
b'CS305 is Awsome.\r\n'
/
light@DESKTOP-K4SPJVJ MINGW64 /
$ telnet 127.0.0.1 5555
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
test
test
CS305 is Awsome.
CS305 is Awsome.
exit
Connection closed by foreign host.
light@DESKTOP-K4SPJVJ MINGW64 /
$

Example: Hello World Web Server

```
import socket

hello = [b'HTTP/1.0 200 OK\r\n',
    b'Connection: close'
    b'Content-Type:text/html; charset=utf-8\r\n',
    b'\r\n',
    b'<html><body>Hello World!<body></html>\r\n',
    b'\r\n']

err404 = [b'HTTP/1.0 404 Not Found\r\n',
    b'Connection: close'
    b'Content-Type:text/html; charset=utf-8\r\n',
    b'\r\n',
    b'<html><body>404 Not Found<body></html>\r\n',
    b'\r\n']
```

Example: Hello World Web Server

```
def web():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(('127.0.0.1', 8080))
    sock.listen(10)
    while True:
        conn, address = sock.accept()
        data = conn.recv(2048).decode().split('\r\n')
        print(data[0].split(' '))
        res = err404
        if data[0].split(' ')[1] == '/':
            res = hello
        for line in res :
            conn.send(line)
        conn.close()

if __name__ == "__main__":
    try:
        web()
    except KeyboardInterrupt:
        exit()
```

Example: Hello World Web Server

```
/c/Users/light/PycharmProjects/CS305-2

light@DESKTOP-K4SPJVJ MINGW64 /c/Users/light/PycharmProjects/CS305-2
$ python web_hello.py
['GET', '/', 'HTTP/1.1']
['GET', '/not-exist', 'HTTP/1.1']
```

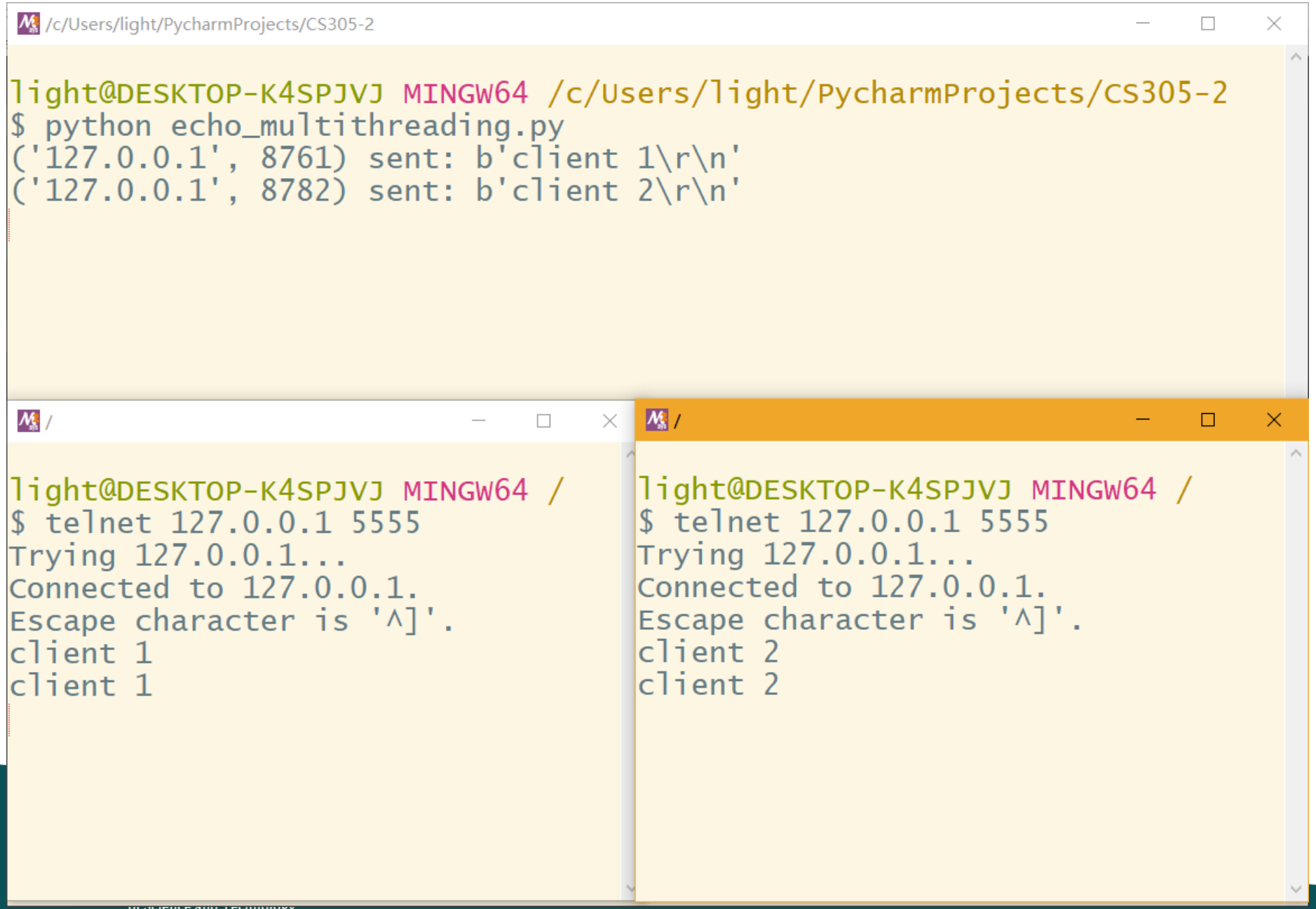
```
/

light@DESKTOP-K4SPJVJ MINGW64 /
$ curl 127.0.0.1:8080
<html><body>Hello World!<body></html>


light@DESKTOP-K4SPJVJ MINGW64 /
$ curl 127.0.0.1:8080/not-exist
<html><body>404 Not Found<body></html>
```

Example: Echo Server Multithreading

```
import socket, threading

class Echo(threading.Thread):
    def __init__(self, conn, address):
        threading.Thread.__init__(self)
        self.conn = conn
        self.address = address

    def run(self):
        while True:
            data = self.conn.recv(2048)
            if data and data != b'exit\r\n':
                self.conn.send(data)
                print('{} sent: {}'.format(self.address, data))
            else:
                self.conn.close()
                return
```

Example: Echo Server Multithreading

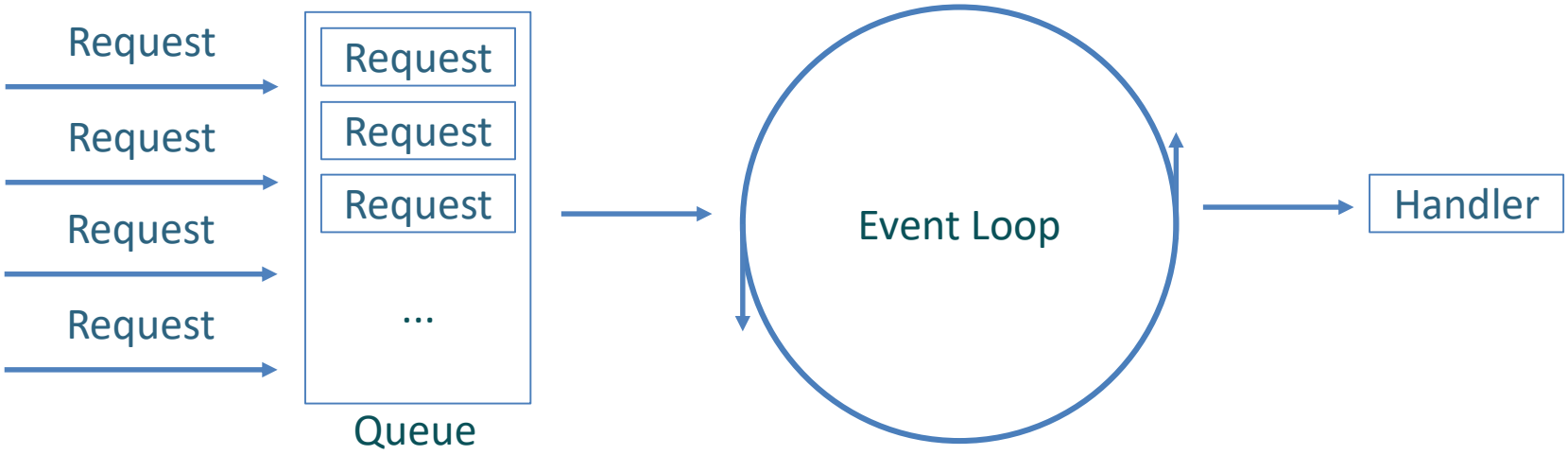
```
def echo():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(('127.0.0.1', 5555))
    sock.listen(10)
    while True:
        conn, address = sock.accept()
        Echo(conn, address).start()


if __name__ == "__main__":
    try:
        echo()
    except KeyboardInterrupt:
        exit()
```


/c/Users/light/PycharmProjects/CS305-2
light@DESKTOP-K4SPJVJ MINGW64 /c/Users/light/PycharmProjects/CS305-2
$ python echo_multithreading.py
('127.0.0.1', 8761) sent: b'client 1\r\n'
('127.0.0.1', 8782) sent: b'client 2\r\n'
/
light@DESKTOP-K4SPJVJ MINGW64 /
$ telnet 127.0.0.1 5555
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
client 1
client 1
/
light@DESKTOP-K4SPJVJ MINGW64 /
$ telnet 127.0.0.1 5555
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
client 2
client 2

Reference: https://docs.python.org/3/library/asyncio.html

asyncio

- Handle requests concurrently with a single thread.

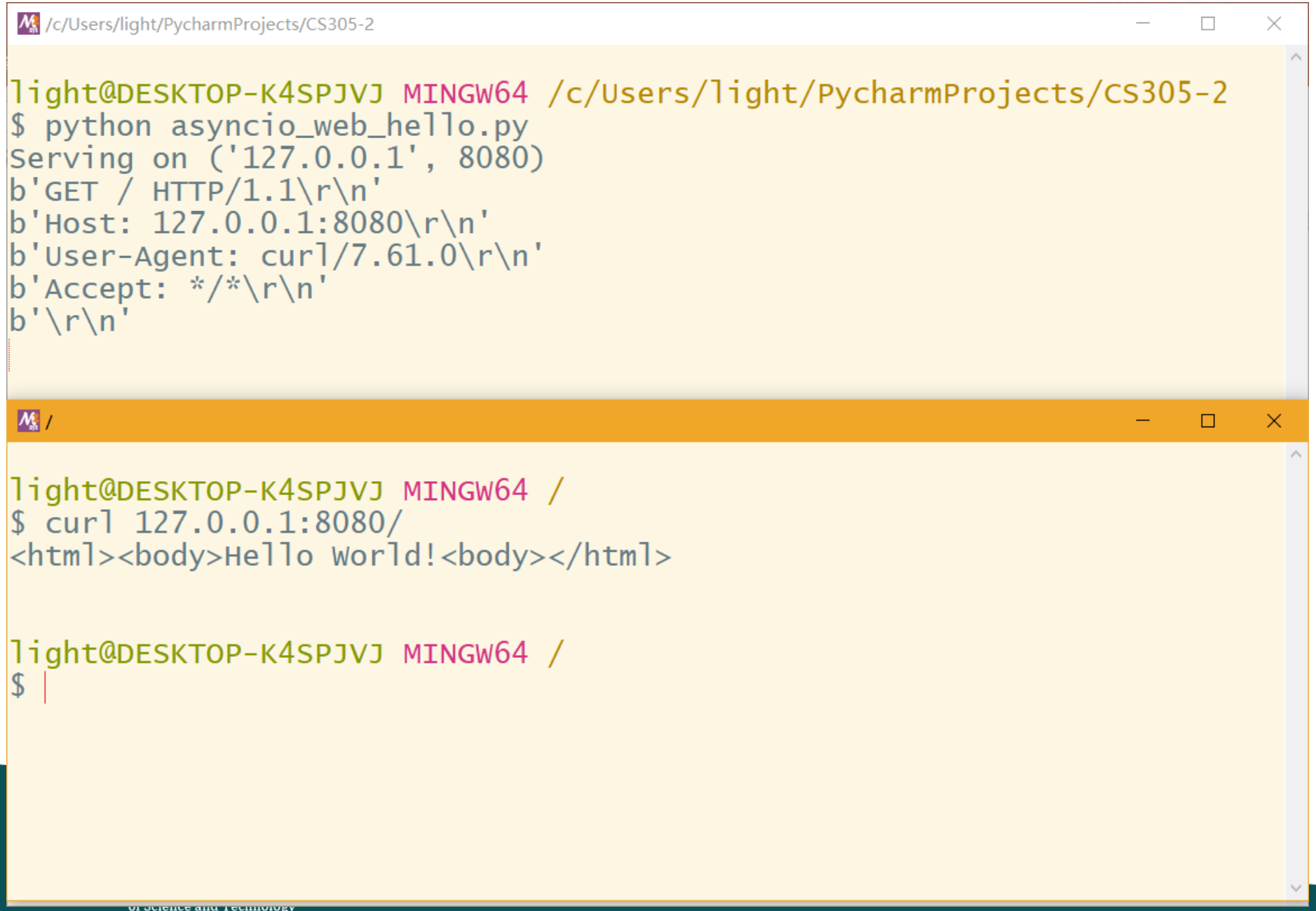


Example: asyncio Web Hello

```
import asyncio

async def dispatch(reader, writer):
    while True:
        data = await reader.readline()
        message = data.decode().split(' ')
        print(data)
        if data == b'\r\n':
            break
    writer.writelines([
        b'HTTP/1.0 200 OK\r\n',
        b'Content-Type:text/html; charset=utf-8\r\n’,
        b'Connection: close\r\n',
        b'\r\n',
        b'<html><body>Hello World!<body></html>\r\n',
        b'\r\n'
    ])
    await writer.drain()
    writer.close()
```

Example: asyncio Web Hello

```
if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    coro = asyncio.start_server(dispatch, '127.0.0.1', 8080, loop=loop)
    server = loop.run_until_complete(coro)

    # Serve requests until Ctrl+C is pressed
    print('Serving on {}'.format(server.sockets[0].getsockname()))
    try:
        loop.run_forever()
    except KeyboardInterrupt:
        pass

    # Close the server
    server.close()
    loop.run_until_complete(server.wait_closed())
    loop.close()
```


/c/Users/light/PycharmProjects/CS305-2
light@DESKTOP-K4SPJVJ MINGW64 /c/Users/light/PycharmProjects/CS305-2
$ python asyncio_web_hello.py
Serving on ('127.0.0.1', 8080)
b'GET / HTTP/1.1\r\n'
b'Host: 127.0.0.1:8080\r\n'
b'User-Agent: curl/7.61.0\r\n'
b'Accept: */*\r\n'
b'\r\n'
/
light@DESKTOP-K4SPJVJ MINGW64 /
$ curl 127.0.0.1:8080/
<html><body>Hello World!<body></html>
light@DESKTOP-K4SPJVJ MINGW64 /
$

Parse HTTP Header (Example Code)

```
keys = ('method', 'path')

class HTTPHeader:
    def __init__(self):
        self.headers = {key: None for key in keys}

    def parse_header(self, line):
        fileds = line.split(' ')
        if fileds[0] == 'GET' or fileds[0] == 'POST' or fileds[0] == 'HEAD':
            self.headers['method'] = fileds[0]
            self.headers['path'] = fileds[1]

    def get(self, key):
        return self.headers.get(key)
```

```
/c/Users/light/PycharmProjects/CS305-2

light@DESKTOP-K4SPJVJ MINGW64 /c/Users/light/PycharmProjects/CS305-2
$ python asyncio_web_withparse.py
Serving on ('127.0.0.1', 8080)
/
/test
```

```
/

light@DESKTOP-K4SPJVJ MINGW64 /
$ curl 127.0.0.1:8080/
<html><body>Hello World!<body></html>


light@DESKTOP-K4SPJVJ MINGW64 /
$ curl 127.0.0.1:8080/test
<html><body>404 Not Found<body></html>


light@DESKTOP-K4SPJVJ MINGW64 /
$
```

Assignment 4.1

- Using asyncio to implement an Echo Server with the same function as *Example: Echo Server*
  - *References: Echo Server Multithreading Example*

Assignment 4.2

- Using asyncio implement a HTTP/1.0 web file browser (nginx autoindex style) according to the pdf document on Sakai.
  - The functions should includes: browsing directory, jumping, and open files. Editing directory and files are not asked to supported.