Name :- Rushikesh Karbhari Palve
Roll No - 31258

: Date :	Page No :
/ /20	

Assignment No - 8 (B4)

DOS :- 03-12-2021

Problem statement :-

Write a program using UDP sockets to enable file transfer (Script, Text, Audio and Video one file each) between two machines.

Objectives :-

(i) To learn UDP Socket.
(ii) To implement a program using UDP socket for data transmission & communication.

Learning Outcomes :-

(i) Students will be able to understand UDP protocol & socket.
(ii) Students will be able to write a program using UDP socket for wired network.

Software & Hardware Requirements :-

- Softwares -
  C/C++ compiler -

Mathematical Model :-

Consider a set S consisting of all elements related to a program. The mathematical model is given as below;

$S = \{s, e, X, Y, Fme, DD, NDD, Mem\ shared\}$

Where,

s = Initial state
e = End state
X = Input given
Y = Output obtained
Fme = Algorithm/Function
DD = Deterministic data
NDD = Non-deterministic data
Mem shared = Memory shared by processor

$X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$

Where,

x_1 = Source IP Address
x_2 = Destination IP Address
x_3 = Source port number
x_4 = Destination port number
x_5 = Sequence number
x_6 = Window size

$Y = \{y_1, y_2, y_3, y_4\}$

Where,

y_1 = IP header
(version, Header length, Total length, Flags, TTL, Protocol, Checksum, source IP, destination IP)

Scanned by TapScanner

y_2 = TCP header
(SrcPort, DstPort, SeqNumber, AckNumber, DataOffset, Flags, WindowSize, CheckSum, UrgPointer)

THEORY :-

TCP Header :-

The TCP consists of a 20 to 60 byte header, followed by data from the application program. The header is 20 bytes if there are no options and up to 60 byte if it contains options.

1. Source Port Address -
This is a 16-bit field that defines the port number of the application program in the host that is sending the segment.

2. Destination Port Address -
This is a 16-bit field that defines the port number of the application program in the host that is receiving the segment.

3. Sequence Number -
The 32-bit field defines the number assigned to the first byte of data contained in this segment.

4. Acknowledgment Number -
This 32-bit field defines the byte number

that the receiver of the segment is expecting to receive from the outer party.

5. Header length -

This 4-bit field indicates the number of 4-byte words in the TCP header. The value of this field can be between 5 (5x4 = 20) and 15(15x4=60).

6. Reserved -

This is a 6-bit field reserved for future use.

7. Control -

This field defines 6 different control bits or flags as shown in following table -

URG	ACK	PSH	RST	SYN	FIN

Flag	Description
URG	The value of the Urgent pointer field is valid -
ACK	the value of the acknowledgement pointer field is valid -
PSH	push the data -
RST	Reset the connection
SYN	Synchronize sequence numbers during connection
FIN	Terminate the connection.

Scanned by TapScanner

8. Checksum -

this 16-bit field contains the checksum for ERROR Detection -

9. Urgent Pointer -

This 16-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data.

10. Options -

there can be up to 40 bytes of optional information in the TCP header -

. I

IP Header :-

The IP header is 20 to 60 bytes in length and contains information essential to routing and delivery -

1. Version (VER) -

This 4-bit defines the version of the IPv4 protocol - Currently the Version is 4.

2. Header length (HLEN) -

This 4-bit defines the total length of the datagram header in 4-byte words -

3. Services -

The field, previously called service type, is now called differentiated services -

4. Total length -

The total length field defines the total length of the datagram including the header -

5. Identification, Flags and Fragmentation Offset -

this field is used in fragmentation -

6. Time to live -

this field was originally designed to hold a timestamp, which was decremented by each visited router -

7. Protocol -

This 8-bit field defines the higher-level protocol (such as TCP, UDP, ICMP and IGMP) that uses the services of the IPv4 layer -

8. Checksum -

this 16-bit field contains the checksum for ERROR Detection -

9. Source Address -

this 32-bit field defines the IPv4 address of the source -

10. Destination Address -

This 32-bit field defines the IPv4 address of the destination -

Wireshark :-

Wireshark, a network analysis tool formerly known as Ethereal, captures packets in real time and display them in human-readable format. Wireshark includes filters, color-coding and other features that let you dig deep into network traffic and inspect individual packets.

ALGORITHM :-

1. Start
2. Set the interface
3. Get the IP address associated to the interface
4. Create an IP Header.
5. Set the source and Destination IP address.
6. Create TCP header.
7. Set source and Destination port numbers, sequence number, flags and window size.
8. Create a packet which has IP and TCP header along with payload.
9. Write the packet on the wire.
10. Print the packet.
11. Stop.

CONCLUSION :-

Thus, we have studied the UDP sockets to transfer file from client to server.

CODE :-

Server.py

```python
import socket,sys,datetime

class UDPFileTransfer:
    server_address = ("127.0.0.1", 12345)
    def __init__(self,type):
        self.type=type
        self.s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
        if type=="server":
            self.s.bind(UDPFileTransfer.server_address)
            print("UDP server running: ", UDPFileTransfer.server_address)
            self.client_address=None

    def notifyServer(self):
        if self.type=="server":
            return
        msg="Hello, UDP server im client"
        msg=bytes(msg.encode('utf-8'))
        self.s.sendto(msg,UDPFileTransfer.server_address)
        print("server was notified!!!")
        print("awaiting data...")

    def wasNotifiedBy(self):
        if self.type=="client":
            return
        while True:
            data, address = self.s.recvfrom(4096)
            data = data.decode('utf-8')
            if data == "Hello, UDP server im client":
                print("client located: ", address)
                self.client_address=address
                return

    def SendText(self,file):
        sys.stdin = open(file, 'r')
        prev = datetime.datetime.now()
        while True:
            try:
                msg = input()
                msg = bytes(msg.encode('utf-8'))
                self.s.sendto(msg, self.client_address)
            except:
                msg = "sab kuch toh de diya!!!"
                msg = bytes(msg.encode('utf-8'))
                self.s.sendto(msg, self.client_address)
                break
        curr = datetime.datetime.now()
        print("All text sent!!!")
        print("Time taken: ", curr - prev)

    def SendAudio(self,file):
        sys.stdin = open(file,'rb')
        prev = datetime.datetime.now()
        # audio
        msg = sys.stdin.read()
        print(len(msg))
        for i in range(0,len(msg),1024):
            self.s.sendto(msg[i:i+1024],self.client_address)
        msg = "sab kuch toh de diya!!!"
        msg = bytes(msg.encode('utf-8'))
        self.s.sendto(msg, self.client_address)
        curr = datetime.datetime.now()
        print("All audio sent!!!")
        print("Time taken: ", curr - prev)

    def SendVideo(self,file):
        pass

    def SendFile(self,file,ext):
        if self.type=="client":
            return
        msg = bytes(ext.encode('utf-8'))
        self.s.sendto(msg, self.client_address)
        if ext=="txt":
            self.SendText(file)
        elif ext=="mp3":
            self.SendAudio(file)
        elif ext=="mp4":
            self.SendVideo(file)

    def RecvText(self):
        sys.stdout = open('file_received.txt', 'w')
        prev = datetime.datetime.now()
        while True:
            data, address = self.s.recvfrom(4096)
            data = data.decode('utf-8')
            if address == UDPFileTransfer.server_address:
                if data == "sab kuch toh de diya!!!":
                    curr = datetime.datetime.now()
                    sys.stdout = sys.__stdout__
                    print("All text received!!!")
                    print("Time taken: ", curr - prev)
                    break
                print(data)

    def RecvAudio(self):
        sys.stdout = open('received.mp3', 'wb')
        prev = datetime.datetime.now()
        while True:
            data, address = self.s.recvfrom(4096)
            if address == UDPFileTransfer.server_address:
                try:
                    data = data.decode('utf-8')
                    curr = datetime.datetime.now()
                    sys.stdout = sys.__stdout__
                    print("All audio received!!!")
                    print("Time taken: ", curr - prev)
                    break
                except:
                    sys.stdout.write(data)

    def RecvVideo(self):
```

```python
        pass

    def RecvFile(self):
        ext, address = self.s.recvfrom(4096)
        ext = ext.decode('utf-8')
        if address!=UDPFileTransfer.server_address:
            return
        if ext == "txt":
            self.RecvText()
        elif ext == "mp3":
            self.RecvAudio()
        elif ext == "mp4":
            self.RecvVideo()


server=UDPFileTransfer("server")
server.wasNotifiedBy()
while True:
    sys.stdin=sys.__stdin__
    c=input("""Send file
    1. text
    2. audio
    3. video
    0. quit
    """)
    if c=="0":
        msg = bytes('q'.encode('utf-8'))
        server.s.sendto(msg, server.client_address)
        break
    elif c=='1':
        server.SendFile("sending/file_send.txt","txt")
    elif c=='2':
        server.SendFile("sending/music_send.mp3","mp3")
    elif c=='3':
        server.SendFile("sending/video_send.mp4","mp4")
server.s.close()
```

Client.py

```python
import socket,sys,datetime

class UDPFileTransfer:
    server_address = ("127.0.0.1", 12345)
    def __init__(self,type):
        self.type=type
        self.s=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
        if type=="server":
            self.s.bind(UDPFileTransfer.server_address)
            print("UDP server running: ", UDPFileTransfer.server_address)
            self.client_address=None

    def notifyServer(self):
        if self.type=="server":
            return
        msg="Hello, UDP server im client"
        msg=bytes(msg.encode('utf-8'))
        self.s.sendto(msg,UDPFileTransfer.server_address)
        print("server was notified!!!")
        print("awaiting data...")
    def wasNotifiedBy(self):
        if self.type=="client":
            return
        while True:
            data, address = self.s.recvfrom(4096)
            data = data.decode('utf-8')
            if data == "Hello, UDP server im client":
                print("client located: ", address)
                self.client_address=address
            return

    def SendText(self,file):
        sys.stdin = open(file, 'r')
        prev = datetime.datetime.now()
        while True:
            try:
                msg = input()
                msg = bytes(msg.encode('utf-8'))
                self.s.sendto(msg, self.client_address)
            except:
                msg = "sab kuch toh de diya!!!"
                msg = bytes(msg.encode('utf-8'))
                self.s.sendto(msg, self.client_address)
                break
        curr = datetime.datetime.now()
        print("All text sent!!!")
        print("Time taken: ", curr - prev)

    def SendAudio(self,file):
        sys.stdin = open(file,'rb')
        prev = datetime.datetime.now()
        # audio
        msg = sys.stdin.read()
        print(len(msg))
        for i in range(0,len(msg),1024):
            self.s.sendto(msg[i:i+1024],self.client_address)
        msg = "sab kuch toh de diya!!!"
        msg = bytes(msg.encode('utf-8'))
        self.s.sendto(msg, self.client_address)
        curr = datetime.datetime.now()
        print("All audio sent!!!")
        print("Time taken: ", curr - prev)

    def SendVideo(self,file):
        pass

    def SendFile(self,file,ext):
        if self.type=="client":
            return
        msg = bytes(ext.encode('utf-8'))
        self.s.sendto(msg, self.client_address)
        if ext=="txt":
            self.SendText(file)
        elif ext=="mp3":
            self.SendAudio(file)
        elif ext=="mp4":
            self.SendVideo(file)
    def RecvText(self):
        sys.stdout = open('receiving/file_received.txt', 'w')
        prev = datetime.datetime.now()
        while True:
            data, address = self.s.recvfrom(4096)
            data = data.decode('utf-8')
            if address == UDPFileTransfer.server_address:
                if data == "sab kuch toh de diya!!!":
                    curr = datetime.datetime.now()
                    sys.stdout = sys.__stdout__
                    print("All text received!!!")
                    print("Time taken: ", curr - prev)
                    break
                print(data)

    def RecvAudio(self):
        sys.stdout = open('receiving/music_received.mp3', 'wb')
        prev = datetime.datetime.now()
        while True:
            data, address = self.s.recvfrom(4096)
            if address == UDPFileTransfer.server_address:
                try:
                    data = data.decode('utf-8')
                    curr = datetime.datetime.now()
                    sys.stdout = sys.__stdout__
                    print("All audio received!!!")
                    print("Time taken: ", curr - prev)
                    break
                except:
                    sys.stdout.write(data)

    def RecvVideo(self):
        pass

    def RecvFile(self):
        ext, address = self.s.recvfrom(4096)
        ext = ext.decode('utf-8')
        if address!=UDPFileTransfer.server_address:
            return
        if ext == "txt":
            self.RecvText()
        elif ext == "mp3":
            self.RecvAudio()
        elif ext == "mp4":
            self.RecvVideo()
        elif ext=="q":
            return ext

client=UDPFileTransfer("client")
client.notifyServer()
while True:
    q=client.RecvFile()
    if q=="q":
        break
client.s.close()
```

OUTPUT :-

Server

```
UDP server running:  ('127.0.0.1', 12345)
client located:  ('127.0.0.1', 55167)
Send file
    1. text
    2. audio
    3. video
    0. quit
1
All text sent!!!
Time taken:  0:00:00.001994
Send file
    1. text
    2. audio
    3. video
    0. quit
    2
2977540
All audio sent!!!
Time taken:  0:00:00.028003
Send file
    1. text
    2. audio
    3. video
    0. quit
    3
Send file
    1. text
    2. audio
    3. video
    0. quit
    0
```

Client

```
server was notified!!!
awaiting data...
All text received!!!
Time taken:  0:00:00.000995
All audio received!!!
Time taken:  0:00:00.027004
```