|  |  |
| --- | --- |
| **CN**  COMPUTER NETWORK | **[ Vatsal parsaniya ]** **[ 17BIT028 ]** |
|  |  |

# COMPUTER NETWORK

# LAB FILE

## INDEX :-

|  |  |
| --- | --- |
| Experiment 1 | To terminate a cat5e cable |
| Experiment 2 | : IP forwarding |
| Experiment 3 | To Setup a DHCP server and client in ubuntu |
| Experiment 4 | Socket Programming - TCP |
| Experiment 5 | Socket Programming - UDP |
| Experiment 6 | To Setup a DNS server in ubuntu |
| Experiment 7 | To install Cisco Packet Tracer and simulate one network |
| Experiment 8 | Aim: Implementation of Stop and Wait,CRC , and Checksum |

## **EXPERIMENT - 1**

**Aim:- TO TERMINATE A CAT5E CABLE.**

**MATERIALS REQUIRED:**

1. CAT 5E/CAT 6 Cable
2. RJ45 Connectors (8P8C connector)
3. Crimping Tool
4. Network Tester

PROCEDURE:

1. Pull about one foot of wire from the spool, add about 4 Inches for the plug on each end.

2. Measure out 2 inches from one end of the wire and place the wire in the cable strippers at that location. The cable should be snug in the strippers, but not tight. The second notch inward is appropriate.

3. Ensure the blade of the wire stripper is perpendicular to the wire and turn the wire stripper around the cable once, which will score the sheathing of the wire. (The goal is to only remove the outer sheath and not damage the inner delicate cables)

4. Remove the cable stripper and gently bend the cable along the score line. This should break the sheathing which can be pulled off the wire and thrown away. When removing the sheathing you should expose 8 twisted cables.

5. Separate the individual cables so they are no longer twisted together.

6. Now arrange the Cable in one of the following configuration.

7. T-568B Configuration

i. White – Orange

ii. Orange

iii. White – Green iv. Blue

v. White - Blue

vi. Green

vii. White – Brown viii. Brown

8. Firmly grasp all the cables near the sheathing and slide your fingers up, collecting all the cables into a flat line. Make sure the cables from left to right are the same configuration as described above.

9. Straighten the cables. Do not worry if the cables aren’t an even length, just straighten them as best you can.

10. Using the cable cutters, trim the tips of the cables so they are even. Make sure the cut is perpendicular to the cables.

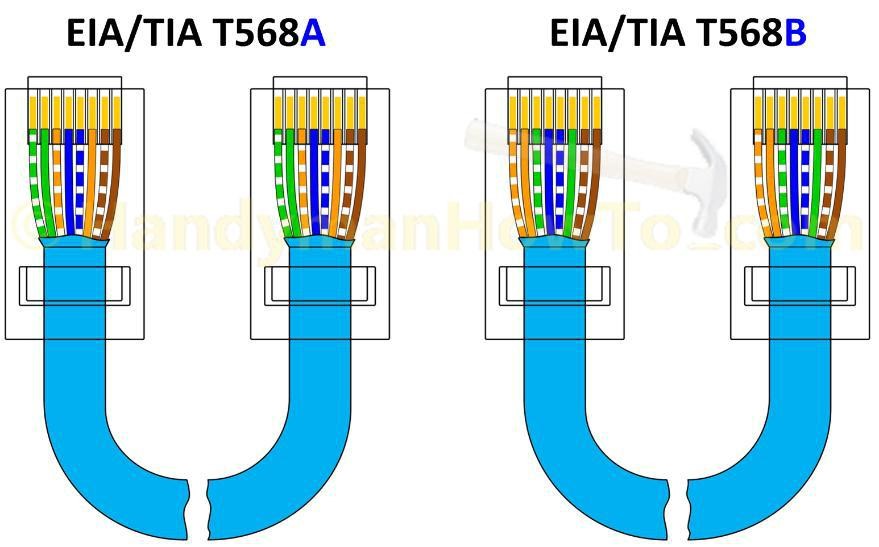
11. Insert the cables in the connector as shown in the images above. The tab on the connector should be facing the floor White-Orange (for T – 568B configuration) cable should be the furthest left in the connector. When inserted, the sheathing should be just inside the end of the data plug. If the wires are too long, remove the data plug, cut a very small length of wire from the end, and reinsert the cable into the data plug.

12. Looking into the connector check all the cables are as far forward as they can be making sure they are past the metal spikes which will come down when crimped.

13. Insert the prepared plug in the crimping tool and squeeze the handle, crimping the wires.

14. Repeat the process for the other end.

15. Now it is time to test your lead, all testers are different but on my one the lights, light up on after the other and they all worked fine.



Note:Network cables have a maximum length, depending on which type is being used, a common rule of thumb is Solid cable you should not exceed 100 meters. Stranded Cable you should not exceed 10 meters. If the length of the wire between two powered network devices exceeds this length, signal degradation and data loss may occur. Short patch leads should be made using stranded cable as it is more flexible but at longer lengths solid cable will be required.

## Experiment 2

##### AIM: TO SEND PACKET FROM ONE NETWORK TO ANOTHER NETWORK BY IP FORWARDING

## Theory :

It is not possible to send packets from one network to another, therefore we use IPv4 forwarding to enable sending packets from one class of network to another using an Intermediate device.

## Procedure :

1. Take 3 PCs and consider them as PC1, PC2 & PC3. Make sure the PC named PC2 (intermediate) has two NIC cards so that we can assign it two different IP addresses at the same time.
2. Now, Create two networks, one having PC1 and PC2 and the other having PC2 and PC3 by assigning a class A IP address to PC1 & another Class A IP address to one NIC of PC2, & for second network, assign class B IP addresses to PC3 & the other NIC of PC2. Hence now we have two different networks.
3. Now, ping PC1 from PC2 and vice-versa. Then do the same for PC3 and PC2. You will notice that it is possible to ping respective devices in all four cases. This is because PC1 and PC2 are in the same network, and as are PC2 and PC3.
4. Now, try to ping PC1 from PC3 and vice-versa. You will not be able to ping from PC1 to PC3 or from PC3 to PC1. This is because they are in two different networks.
5. In order to send a packet from one network to another network, we need to enable packet forwarding in the intermediate PC (PC2), which can be done using the following procedure/commands in Ubuntu Terminal :
6. Open sysctl.conf using following command :

## sudo nano /etc/sysctl.conf

1. Now, search for following line and uncomment it, if not found add the following line in sysctl.conf file to enable IP forwarding :

## net.ipv4.ip\_forward = 1

1. Save the file and exit nano text editor.
2. Restart the Computer
3. Now, it is possible to ping/transfer packets from PC1 to PC3 and vice-versa.

## Result :

Using the above procedure it is possible to send packets from device on one class of network to another.

## Experiment 3

# AIM: TO SETUP A DHCP SERVER AND CLIENT IN UBUNTU

##### THEORY:

DHCP, abbreviation of Dynamic Host Control Protocol, is a network protocol that assigns IP addresses automatically to client systems in the network. This reduces the tedious task of manually assigning IP addresses in a large network that has hundreds of systems. We can define the IP range (Scopes) in the DHCP server and distribute them across the network. The client systems in the network will automatically get the IP address.

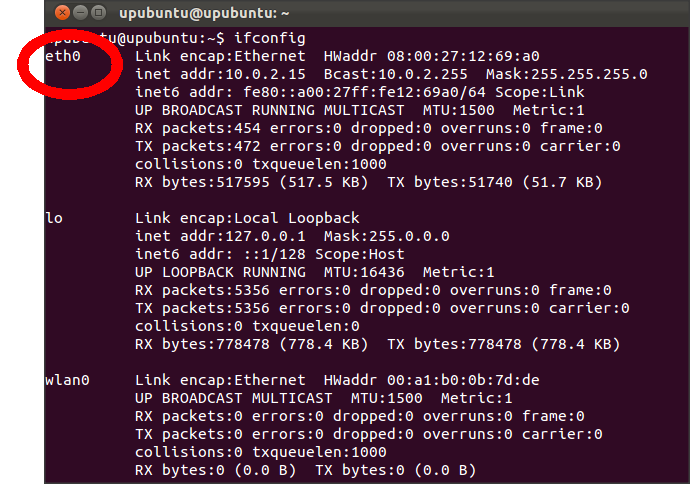
##### PROCEDURE:

1. **Install DHCP Server in Ubuntu 16.04 LTS:**

a. Open Terminal (Ctrl + Alt +T) and run the following command to install DHCP server

sudo apt-get install isc-dhcp-server

1. **Configure DHCP server:**
   1. The default configuration file of DHCP server is /etc/default/isc-dhcp-server. We need to edit and modify it as per our requirements.
   2. If you have more than one Network interface card in your DHCP server, you need to mention on which interface should the DHCP server serve DHCP requests.
   3. Use the “ifconfig” command to get the interface name of your network card. It is the address on the first column.



d.To do so, edit /etc/default/isc-dhcp-server configuration file:

sudonano/etc/default/isc-dhcp-server

e.Assign the network interface:

INTERFACES="enp0s3"

1. Then edit the dhcpd.conf file:

sudonano/etc/dhcp/dhcpd.conf

Modify it as shown below.

[...]

* option definitions common to all supported networks...

option domain-name "pdpu.lan";

ption domain-name-servers ubuntuserver.pdpu.lan;

[...]

1. To make this server as official DHCP for your clients, find and uncomment the following line that says “authoritative;”
2. Scroll down a little bit , and define the subnet, IP range, domain and domain name servers like below.

[...]

* A slightly different configuration for an internal subnet.

subnet 192.168.1.0 netmask 255.255.255.0 { range 192.168.1.20 192.168.1.30;

option domain-name-servers ubuntuserver.pdpu.lan; option domain-name "pdpu.lan";

option routers 192.168.1.1;

option broadcast-address 192.168.1.255; default-lease-time 600; max-lease-time 7200;

}

[...]

You can change the subnet and the IP range as per your requirement

1. Now, restart dhcp service:

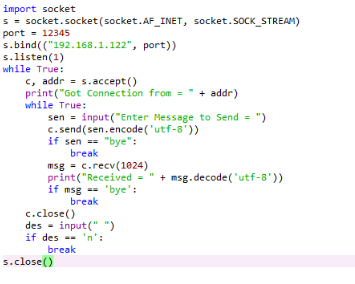
sudosystemctl restart isc-dhcp-server

1. **Configure DHCP Clients:**
   1. Open Network Connections either from Unity dash or Menu.
   2. In the Network connections window, select your Ethernet card and click Edit.
   3. Click IPv4 Settings and select “Automatic (DHCP)” option. Finally click Save.
   4. Now, restart your client system, and check the IP address of your client system.
   5. Use the “ifconfig” command to check the ip address of the client system.

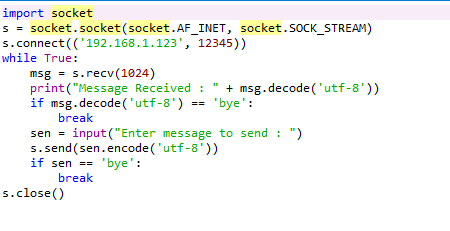
**OBSERVATIONS:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Experiment 4AIM: TCP Socket Programming |  |  |

# TCP Server



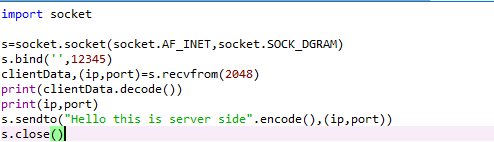
# TCP Client



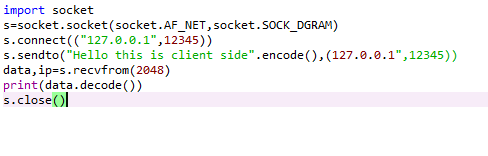
## Experiment 5

##### AIM: UDP Socket Programming

# UDP Server



# UDP Client



## Experiment 6

##### AIM: TO SETUP A DNS SERVER IN UBUNTU

# THEORY:

A DNS server is a computer server that contains a database of public IP addresses and their associated hostnames, and in most cases, serves to resolve, or translate, those common names to IP addresses as requested.Caching name server saves the DNS query results locally for a particular period of time. It reduces the DNS server’s traffic by saving the queries locally, therefore it improves the performance and efficiency of the DNS server.

##### PROCEDURE:

1. **Install DNS Server in Ubuntu 16.04 LTS:**

a. Open Terminal (Ctrl + Alt +T) and run the following command to install DNS server

sudo apt-get install bind9 bind9utils bind9-doc

1. **Configure DHCP server:**
   1. The default configuration file of DNS server is **/etc/bind/named.conf.options**. We need to edit and modify it as per our requirements.

sudonano/etc/bind/named.conf.options

1. Uncomment the following lines. And then, add your ISP or Google public DNS server IP addresses:

forwarders {

8.8.8.8;

};

c.Restart bind9 service to take effect the changes:

sudosystemctl restart bind9

**6.Install and configure Primary DNS server:**

1. The default configuration file of DNS server is **/etc/bind/**directory. We need to edit the **named.conf** file and as per our requirements.

sudonano/etc/bind/named.conf

b.This file should have the following lines in it. If the lines are not there, just add them.

include "/etc/bind/named.conf.options";

include "/etc/bind/named.conf.local";

include "/etc/bind/named.conf.default-zones";

c.Now we need to define the forward and reverse zone files, edit **named.conf.local** file:

sudonano/etc/bind/named.conf

Define the forward and reverse files as shown below.

zone "pdpu.lan" {

type master;

file "/etc/bind/for.pdpu.lan";

};

zone "1.168.192.in-addr.arpa" { type master;

file "/etc/bind/rev.pdpu.lan";

};

1. Here, for.pdpu.lan is the forward zone file. rev.pdpu.lan is the reverse zone files. Let us now create the zone files which we defined in the previous step. First let us create forward zone file as shown below.

sudonano /etc/bind/for.pdpu.lan

Add the following line:

$TTL 86400

@IN SOA pri.pdpu.lan. root.pdpu.lan. ( 2011071001 ;Serial

3600;Refresh

1800;Retry

604800;Expire

|  |  |  |  |
| --- | --- | --- | --- |
|  | 86400 |  | Minimum TTL |
|  |  |  |  |
| @ | IN | NS | pri.pdpu.lan. |
| @ | IN | A | 192.168.1.200 |
| @ | IN | A | 192.168.1.202 |
| pri | IN | A | 192.168.1.200 |
| client | IN | A | 192.168.1.202 |

1. Similarly, you can add the other client records as defined in the above file. Save and close the file. Next create reverse zone.

sudonano /etc/bind/rev.pdpu.lan

Add the following line:

|  |  |  |  |
| --- | --- | --- | --- |
| $TTL 86400 |  |  |  |
| @ IN | SOA |  | pri.pdpu.lan. root.pdpu.lan. ( |
|  | 2011071002 ;Serial |  |  |
|  | 3600 |  | ;Refresh |
|  | 1800 |  | ;Retry |
|  | 604800 |  | ;Expire |
|  | 86400 |  | ;Minimum TTL |
| ) |  |  |  |
| @ | IN | NS | pri.pdpu.lan. |
| @ | IN | NS | sec.pdpu.lan. |
| @ | IN | PTR | pdpu.lan. |
| pri | IN | A | 192.168.1.200 |
| client | IN | A | 192.168.1.202 |
| 200 | IN | PTR | pri.pdpu.lan. |
| 202 | IN | PTR | client.pdpu.lan. |
|  |  |  |  |

1. Set the proper permissions and ownership to the bind9 directory.

sudochmod -R 755 /etc/bind sudochown -R bind:bind /etc/bind

1. Next, we need to verify the DNS configuration files and zone files.
2. Check the DNS configuration files with commands:

sudo named-checkconf /etc/bind/named.conf

sudo named-checkconf /etc/bind/named.conf.local

1. If the above commands returns nothing, it means DNS configuration is valid.
2. Next, check the zone files using commands:

sudo named-checkzonepdpu.lan /etc/bind/for.pdpu.lan

k.Check the reverse zone file:

sudo named-checkzonepdpu.lan /etc/bind/rev.pdpu.lan sudosystemctl restart bind9

## Experiment 7

##### AIM: TO INSTALL CISCO PACKET TRACER AND SIMULATE ONE NETWORK

# THEORY:

Cisco developed Packet Tracer to help Networking Academy students achieve the most optimal learning experience while gaining practical networking technology skills.Packet Tracer is a powerful network simulation platform inspiring student to experiment with network behavior and ask 'what if' questions. It supplements physical equipment in the classroom by allowing students to create a network with an almost unlimited number of devices, encouraging practice, discovery and troubleshooting.About Networking Academy: Started in 1997, Cisco Networking Academy is an IT & Networking skills and career building program for learning institutions and individuals worldwide.

##### PROCEDURE:

1. **Install DHCP Server in Ubuntu 16.04 LTS:**
   1. Open Terminal (Ctrl + Alt +T) and run the following command in the same directory as you downloaded the cisco packet tracer file.

tar –zxfvPacketTracer7.tar.gz

b.Now some output will appear, then change file directory (cd) as follows:

cd PacketTracer70

c.The needed file is install. So now type the command

sudo bash install

d.Now some output will appear, then change file directory (cd) as follows

cd PacketTracer70

e.Then type the password and press Enter and select the default location for installation.

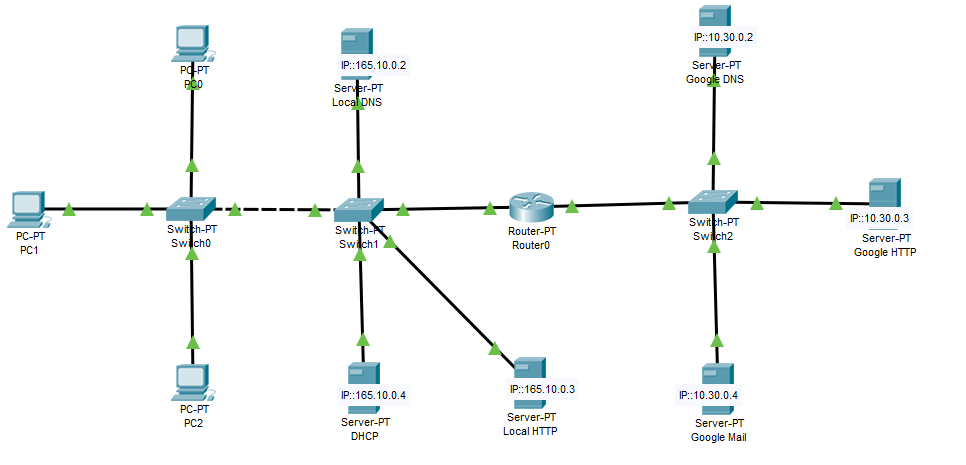
1. We need to set up the packet tracer environment parameters so use the following command to run the script.

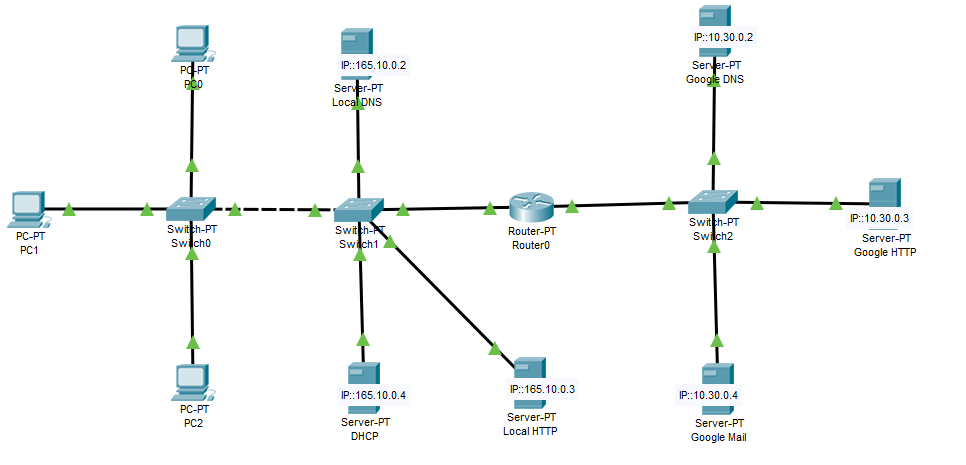
sudo bash set\_ptenv.sh

**To run the packet tracer open a terminal and use the following the command: packettracer**

1. **Simulating a network:**
   1. Use the appliances and tools available at the bottom of the screen to create a network similar to the one shown below.
2. Setup the Servers in the following configuration:
   1. Server 1:
      1. DNS Server
      2. Static IP
      3. DHCP Server
   2. Server 2:
      1. HTTP Server
      2. Mail Server
   3. Server 3:
      1. HTTP Server
      2. Mail Server
3. Also, Setup 3 clients and Server one on one class network and connect them via a switch and router to network of another class network with server 2 & server 3.

##### SCREENSHOTS:





Experiment 8

##### AIM: IMPLEMENTATION OF STOP AND WAIT, GO BACK N, AND SELECTIVE REPEAT

# STOP AND WAIT:

# THEORY:

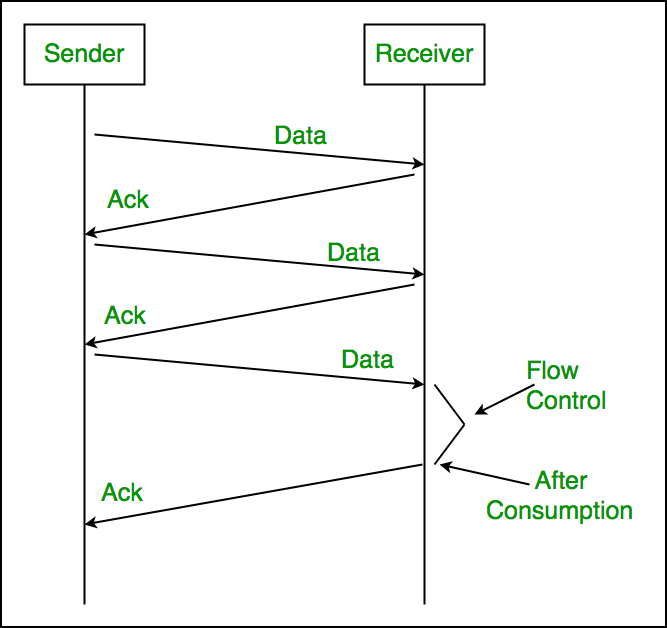
Rules for Simple Stop Wait Protocol.

Sender:

1. Send one data packet at a time.
2. Send next packet only after receiving acknowledgement for previous.

Receiver:

1. Send acknowledgement after receiving and consuming of data packet.
2. After consuming packet acknowledgement need to be sent (Flow Control)



# Stop & Wait Client and Server:

##### Server side :

import socket

import select

import sys

def getACK(ack,Client\_socket,i\_s,number):

while not ack:

msg ="packet :"+str(number)

Client\_socket.send(msg.encode("utf-8"))

r = select.select(i\_s,[],[],5)

if r[0]:

Client\_socket.recv(1024).decode("utf-8")

ack = True;

print("GOT ack packet " + str(number))

number += 1

return True,number

server\_socket = socket.socket()

server\_socket.bind(('',12345))

server\_socket.listen(1)

Client\_socket , address = server\_socket.accept()

print(address)

number = 1

ack = False

i\_s = [Client\_socket,sys.stdin]

while True:

ack,number = getACK(False,Client\_socket,i\_s,number)

## Client side :

import socket

import sys

import select

client\_socket = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

client\_socket.connect(('',12345))

while True:

while True:

Input\_Stream = [client\_socket,sys.stdin]

r,w,e= select.select(Input\_Stream,[],[])

for soc in r:

if soc == client\_socket:

msg = client\_socket.recv(1024).decode('utf-8')

print(msg)

if soc == sys.stdin:

msg = raw\_input()

client\_socket.send(msg.encode('utf-8'))

if msg == 'bye':

input\_stream.remove(client\_socket)

if client\_socket in input\_stream:

break

client\_socket.close()

## CRC SENDER

import socket

import sys

import select

def binary(string\_data):

binary = ' '.join(format(ord(x), 'b') for x in string\_data)

return binary

def xor(a,b):

result = list(b)

for i in range(0,len(b)):

if a[i] == b[i]:

result[i] ='0'

else:

result[i] ='1'

return "".join(result)

def mod2div(divident,divisor):

pick = len(divisor)

temp = divident[0:pick]

while pick < len(divident):

if temp[0] == '1':

temp = xor(temp,divisor) + divident[pick]

temp = temp[1:]

else:

temp = xor('0'\*pick,temp) + divident[pick]

temp =temp[1:]

pick = pick + 1

if temp[0] == '1':

temp = xor(divisor, temp)

else:

temp = xor('0'\*pick, temp)

return temp

def encodeData(data,key):

lengthOfKey = len(key)

addZerotoData = data + '0'\*(lengthOfKey-1)

remainder = mod2div(addZerotoData,key)

return data+remainder[(-1)\*(lengthOfKey-1):],remainder[(-1)\*(lengthOfKey-1):]

server\_socket = socket.socket()

server\_socket.bind(('',12345))

server\_socket.listen(1)

client\_socket , address = server\_socket.accept()

print("connected to :",address)

key = "1001"

while True:

input\_s = [sys.stdin,client\_socket]

r,w,e = select.select(input\_s,[],[])

for soc in r:

if soc == sys.stdin:

msg = raw\_input()

print("Message :",msg)

binary\_msg = binary(msg)

print("Message in binary : ",binary\_msg)

encode\_msg,crc = encodeData(binary\_msg,key)

print("Message after apply CRC :",encode\_msg)

print("CRC :",crc)

client\_socket.send(encode\_msg.encode("utf-8"))

else:

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

print("Message from Client :",msg)

check = mod2div(msg,key)

print("check CRC : ",check)

if msg == "bye":

input\_s.remove(client\_socket)

if client\_socket not in input\_s:

break

server\_socket.close()

CRC RECEIVER

import socket

import select

import sys

def binary(string\_data):

binary = ' '.join(format(ord(x), 'b') for x in string\_data)

return binary

def xor(a,b):

result = list(b)

for i in range(0,len(b)):

if a[i] == b[i]:

result[i] ='0'

else:

result[i] ='1'

return "".join(result)

def mod2div(divident,divisor):

pick = len(divisor)

temp = divident[0:pick]

while pick < len(divident):

if temp[0] == '1':

temp = xor(temp,divisor) + divident[pick]

temp = temp[1:]

else:

temp = xor('0'\*pick,temp) + divident[pick]

temp =temp[1:]

pick = pick + 1

if temp[0] == '1':

temp = xor(divisor, temp)

else:

temp = xor('0'\*pick, temp)

return temp

def encodeData(data,key):

lengthOfKey = len(key)

addZerotoData = data + '0'\*(lengthOfKey-1)

remainder = mod2div(addZerotoData,key)

return data+remainder[(-1)\*(lengthOfKey-1):],remainder[(-1)\*(lengthOfKey-1):]

s = socket.socket()

s.connect(('',12345))

print("Connected")

key = "1001"

while True:

input\_s = [sys.stdin,s]

r,w,e = select.select(input\_s,[],[])

for soc in r:

if soc == s :

msg = s.recv(1024).decode('utf-8')

print("Message From Server :",msg)

print("Check CRC :",mod2div(msg,key))

if soc == sys.stdin:

msg = raw\_input()

print("Message : ",msg)

binary\_msg = binary(msg)

print("Message in Binary : ",binary\_msg)

encode\_msg,crc = encodeData(binary\_msg,key)

print("Message after apply CRC : ",encode\_msg)

print("CRC :" , crc)

s.send(encode\_msg.encode("utf-8"))

if msg == "bye":

input\_s.remove(s)

if s not in input\_s:

break

s.close()

## Checksum Client and Server

Server side:

import socket

import sys

def ones\_complement(n):

one = ""

for c in n:

if c == "0":

one += "1"

else:

one += "0"

return one

server = socket.socket()

server.bind(('192.168.50.234', 12345))

server.listen(1)

checksumSize = 16

while True:

connection, addr = server.accept()

msg = input("Enter message : ")

if len(msg) % checksumSize != 0:

msg += "0" \* (checksumSize - len(msg) % checksumSize)

print("After padding : " + msg)

checksum = 0

j = 0

for i in range(checksumSize, len(msg)+1, checksumSize):

checksum += int(msg[j:i], 2)

j += checksumSize

checksumB = str("{0:b}".format(checksum))

if len(checksumB) > checksumSize:

e = len(checksumB) - checksumSize

checksum = int(checksumB[0:e], 2) + int(checksumB[e:], 2)

checksumB = str("{0:b}".format(checksum))

if len(checksumB) < checksumSize:

checksumB = "0" \* (checksumSize - len(checksumB)) + checksumB

checksumB = ones\_complement(checksumB)

print("Binary Checksum : " + checksumB)

print("Checksum : " + str(int(checksumB, 2)))

connection.send((msg + checksumB).encode('utf-8'))

ack = connection.recv(1024)

if ack.decode('utf-8') == "ACK":

print("Message successfully received by client")

break

else:

print("Error")

server.close()

sys.exit()

## Checksum Client

import socket

import sys

def ones\_complement(n):

one = ""

for c in n:

if c == "0":

one += "1"

else:

one += "0"

return one

server = socket.socket()

server.bind(('192.168.50.234', 12345))

server.listen(1)

checksumSize = 16

while True:

connection, addr = server.accept()

msg = input("Enter message : ")

if len(msg) % checksumSize != 0:

msg += "0" \* (checksumSize - len(msg) % checksumSize)

print("After padding : " + msg)

checksum = 0

j = 0

for i in range(checksumSize, len(msg)+1, checksumSize):

checksum += int(msg[j:i], 2)

j += checksumSize

checksumB = str("{0:b}".format(checksum))

if len(checksumB) > checksumSize:

e = len(checksumB) - checksumSize

checksum = int(checksumB[0:e], 2) + int(checksumB[e:], 2)

checksumB = str("{0:b}".format(checksum))

if len(checksumB) < checksumSize:

checksumB = "0" \* (checksumSize - len(checksumB)) + checksumB

checksumB = ones\_complement(checksumB)

print("Binary Checksum : " + checksumB)

print("Checksum : " + str(int(checksumB, 2)))

connection.send((msg + checksumB).encode('utf-8'))

ack = connection.recv(1024)

if ack.decode('utf-8') == "ACK":

print("Message successfully received by client")

break

else:

print("Error")

server.close()

sys.exit()