CHAT CLIENT

# Python program to implement client side of chat room.

import socket

import select

import sys

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

#server.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

IP\_address = '127.0.0.1'

Port = 12345

s.connect((IP\_address, Port))

while True:

# maintains a list of possible input streams

inputStream\_list = [sys.stdin, s]

""" There are two possible input situations. Either the

user wants to give manual input to send to other people,

or the server is sending a message to be printed on the

screen. Select returns from sockets\_list, the stream that

is reader for input. So for example, if the server wants

to send a message, then the if condition will hold true

below.If the user wants to send a message, the else

condition will evaluate as true"""

read\_sockets,write\_socket, error\_socket = select.select(inputStream\_list,[],[])

for socks in read\_sockets:

if socks == s:

message = socks.recv(2048)

print "<received>" + message

else:

message = raw\_input()

s.send(message)

print "<You>" + message

if message == 'bye'

s.close()

inputStream\_list.remove(s)

break

if s not in inputStream\_list:

break

s.close()

CHAT CLIENT BASIC

#!/usr/bin/python

# This is client.py file

import socket # Import socket module

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Create a socket object

#host = socket.gethostname() # Get local machine name

port = 12345 # Reserve a port for your service.

s.connect(("127.0.0.1", port))

While True:

buf = raw\_input('Enter a message to send: ')

s.send(buf)

if buf == 'bye':

break

msg= s.recv(1024)

print 'received ' + msg

if msg == 'bye':

break

CHAT ROOM CLIENT

# Python program to implement client side of chat room.

import socket

import select

import sys

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

IP\_address = '127.0.0.1'

Port = 12345

server.connect((IP\_address, Port))

while True:

# maintains a list of possible input streams

sockets\_list = [sys.stdin, server]

""" There are two possible input situations. Either the

user wants to give manual input to send to other people,

or the server is sending a message to be printed on the

screen. Select returns from sockets\_list, the stream that

is reader for input. So for example, if the server wants

to send a message, then the if condition will hold true

below.If the user wants to send a message, the else

condition will evaluate as true"""

read\_sockets,write\_socket, error\_socket = select.select(sockets\_list,[],[])

for socks in read\_sockets:

if socks == server:

message = socks.recv(2048)

print message

else:

message = sys.stdin.readline()

server.send(message)

print "<You>" + message

server.close()

CHAT ROOM SERVER

import socket

import select

import sys

from thread import \*

def clientthread(conn, addr):

# sends a message to the client whose user object is conn

conn.send("Welcome to this chatroom!")

while True:

try:

message = conn.recv(2048)

if message:

"""prints the message and address of the

user who just sent the message on the server

terminal"""

print "<" + addr[0] + "> " + message

# Calls broadcast function to send message to all

message\_to\_send = "<" + addr[0] + "> " + message

broadcast(message\_to\_send, conn)

else:

"""message may have no content if the connection

is broken, in this case we remove the connection"""

remove(conn)

except:

continue

"""Using the below function, we broadcast the message to all

clients who's object is not the same as the one sending

the message """

def broadcast(message, connection):

for conn in list\_of\_conns:

if conn!=connection

try:

conn.send(message)

except:

conn.close()

# if the link is broken, we remove the client

remove(conn)

"""The following function simply removes the object

from the list that was created at the beginning of

the program"""

def remove(connection):

if connection in list\_of\_conns:

list\_of\_conns.remove(connection)

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

#server.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

Port = 12345

"""

binds the server to specified port number.

The client must be aware of these parameters

"""

server.bind(('', Port))

"""

listens for 100 active connections. This number can be

increased as per convenience.

"""

server.listen(100)

list\_of\_conns = []

while True:

"""Accepts a connection request and stores two parameters,

conn which is a socket object for that user, and addr

which contains the IP address of the client that just

connected"""

conn, addr = server.accept()

"""Maintains a list of clients for ease of broadcasting

a message to all available people in the chatroom"""

list\_of\_conns.append(conn)

# prints the address of the user that just connected

print str(addr) + " connected"

# creates and individual thread for every user

# that connects

start\_new\_thread(clientthread,(conn,addr))

conn.close()

server.close()

CHAT SERVER

import socket

import select

import sys

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

Port = 12345

server.bind(('', Port))

server.listen(5)

while True:

conn, addr = server.accept()

# prints the address of the user that just connected

print addr[0] + " connected"

while True:

inputStream\_list = [sys.stdin, conn]

read\_sockets,write\_socket, error\_socket = select.select(inputStream\_list,[],[])

for socks in read\_sockets:

if socks == conn:

message = socks.recv(2048)

print "<received>" + message

else:

message = raw\_input()

conn.send(message)

print "<You>" + message

if message == 'bye'

conn.close()

inputStream\_list.remove(conn)

break

if conn not in inputStream\_list:

break

conn.close()

# serve the next client

server.close()

CHAT SERVER BASIC

#!/usr/bin/python

# This is server.py file

import socket # Import socket module

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) # Create a socket object

port = 12345 # Reserve a port for your service.

s.bind(('', port)) # Bind to the port

s.listen(5) # Now wait for client connection.

while True:

c, addr = s.accept() # Establish connection with client.

print 'Got connection from', addr

while True:

buf=c.recv(1024)

print 'received ' + buf + '\n'

if buf == 'bye':

break

msg=raw\_input("Enter a msg")

c.send(msg)

if msg == 'bye':

break

c.close() # Close the connection with this client

# start next interaction to connect with next client

ECHO CLIENT

#!/usr/bin/python

# This is client.py file

# Import socket module

# Import socket module

import socket

# Create a socket object

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Reserve a port for your service.

port = 12345

# connect with server running on localhost

s.connect(("127.0.0.1", port))

buf = raw\_input('Enter a message to send: ')

s.send(buf)

print s.recv(1024)

ECHO SERVER

#!/usr/bin/python

# This is server.py file

# Import socket module

import socket

# Create a socket object

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Reserve a port for your service.

port = 12345

# Bind to the port

s.bind(('', port))

Now wait for client connection.

s.listen(5)

while True:

# Establish connection with client.

c, addr = s.accept()

print 'Got connection from', addr

buf=c.recv(1024)

print 'received ' + buf + '\n'

print 'echoing it back\n'

c.send(buf)

c.close() # Close the connection

SERVER

#!/usr/bin/python

# This is server.py file

# Import socket module

import socket

# Create a socket object

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Reserve a port for your service.

port = 12345

# Bind to the port, permit connections from any node in network

s.bind(('', port))

# Now wait for client connection.

s.listen(5)

while True:

# Establish connection with client.

c, addr = s.accept()

print 'Got connection from', addr

c.send('Thank you for connecting\n')

# Close the connection

c.close()

UDP BASIC CHAT CLIENT

#!/usr/bin/python

import socket

# Create a UDP socket at client side

udpSocket = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_DGRAM)

ipAddress="127.0.0.1"

port = 12345

while True:

buf=raw\_input('Enter a message to send:')

udpSocket.sendto (buf, (ipAddress, port))

if buf == 'bye':

break

# receive message from a UDP server

message = udpSocket.recvfrom(1024)

print 'Message from Server:' + str(message)

if message[0]== 'bye':

break

UDP BASIC CHAT SERVER

import socket

udpSocket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

# Bind to address and ip

udpSocket.bind(('', 12345))

print("UDP server up and listening")

while(True):

message = udpSocket.recvfrom(1024)

print "received message from client:" + str (message)

# message [0] is the message content and message[1] is address of sender

clientMessage = message[0]

clientAddress = message[1]

if clientMessage == 'bye':

continue

# Sending a reply to client

buf=raw\_input ("Enter a message for client:")

udpSocket.sendto(buf, clientAddress)

UDP CHAT CLIENT

#!/usr/bin/python

import socket

import select

import sys

# Create a UDP socket at client side

udpSocket = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_DGRAM)

ipAddress="127.0.0.1"

port = 12345

#as server is not connected to specific client, it can talk to any one who interacts

while True:

inputStream\_list = [sys.stdin, udpSocket]

read\_sockets,write\_socket, error\_socket = select.select(inputStream\_list,[],[])

for socks in read\_sockets:

if socks == udpSocket:

message = udpSocket.recvfrom(1024)

print "<received>" + str(message)

if message[0]== 'bye':

udpSocket.close()

inputStream\_list.remove(udpSocket)

break

else:

buf=raw\_input()

udpSocket.sendto (buf, (ipAddress, port))

print "<You>" + buf

if buf== 'bye':

udpSocket.close()

inputStream\_list.remove(udpSocket)

break

if udpSocket not in inputStream\_list:

break

UDP CHAT SERVER

import socket

import select

import sys

udpSocket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

# Bind to address and ip

udpSocket.bind(('', 12345))

print("UDP server up and listening")

clientInfo=[]

while(True):

inputStream\_list = [sys.stdin, udpSocket]

read\_sockets,write\_socket, error\_socket = select.select(inputStream\_list,[],[])

for socks in read\_sockets:

if socks == udpSocket:

message = udpSocket.recvfrom(1024)

print "<received>" + str(message)

clientInfo = message[1]

if message[0]== 'bye':

inputStream\_list.remove(udpSocket)

break

else:

buf=raw\_input()

udpSocket.sendto (buf, clientInfo)

print "<You>" + buf

if buf== 'bye':

inputStream\_list.remove(udpSocket)

break

UDP ECHO CLIENT

#!/usr/bin/python

import socket

# Create a UDP socket at client side

udpSocket = socket.socket(family=socket.AF\_INET, type=socket.SOCK\_DGRAM)

ipAddress="127.0.0.1"

port = 12345

buf=raw\_input('Enter a message to send:')

udpSocket.sendto (buf, (ipAddress, port))

# receive message from a UDP server

message = udpSocket.recvfrom(1024)

print 'Message from Server:' + str(message)

UDP ECHO SERVER

import socket

udpSocket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

# Bind to address and ip

udpSocket.bind(('', 12345))

print("UDP server up and listening")

while(True):

message = udpSocket.recvfrom(1024)

print "received message from client:" + str (message)

# message [0] is the message content and message[1] is address of sender

clientMessage = message[0]

clientAddress = message[1]

print 'echoing it back'

# Sending a reply to client

udpSocket.sendto(clientMessage, clientAddress)

CHECKSUM

#include<iostream>

#include<string.h>

using namespace std;

int main()

{

char a[20],b[20];

char sum[20],complement[20];

int i;

cout<<"Enter first binary string\n";

cin>>a;

cout<<"Enter second binary string\n";

cin>>b;

if(strlen(a)==strlen(b))

{

char carry='0';

int length=strlen(a);

for(i=length-1;i>=0;i--)

{

if(a[i]=='0' && b[i]=='0' && carry=='0')

{

sum[i]='0';

carry='0';

}

else if(a[i]=='0' && b[i]=='0' && carry=='1')

{

sum[i]='1';

carry='0';

}

else if(a[i]=='0' && b[i]=='1' && carry=='0')

{

sum[i]='1';

carry='0';

}

else if(a[i]=='0' && b[i]=='1' && carry=='1')

{

sum[i]='0';

carry='1';

}

else if(a[i]=='1' && b[i]=='0' && carry=='0')

{

sum[i]='1';

carry='0';

}

else if(a[i]=='1' && b[i]=='0' && carry=='1')

{

sum[i]='0';

carry='1';

}

else if(a[i]=='1' && b[i]=='1' && carry=='0')

{

sum[i]='0';

carry='1';

}

else if(a[i]=='1' && b[i]=='1' && carry=='1')

{

sum[i]='1';

carry='1';

}

else

break;

}

sum[length]='\0';

cout<<"\nSum="<<carry<<sum;

for(i=0;i<length;i++)

{

if(sum[i]=='0')

complement[i]='1';

else

complement[i]='0';

}

complement[length]='\0';

if(carry=='1')

carry='0';

else

carry='1';

cout<<"\nChecksum="<<carry<<complement;

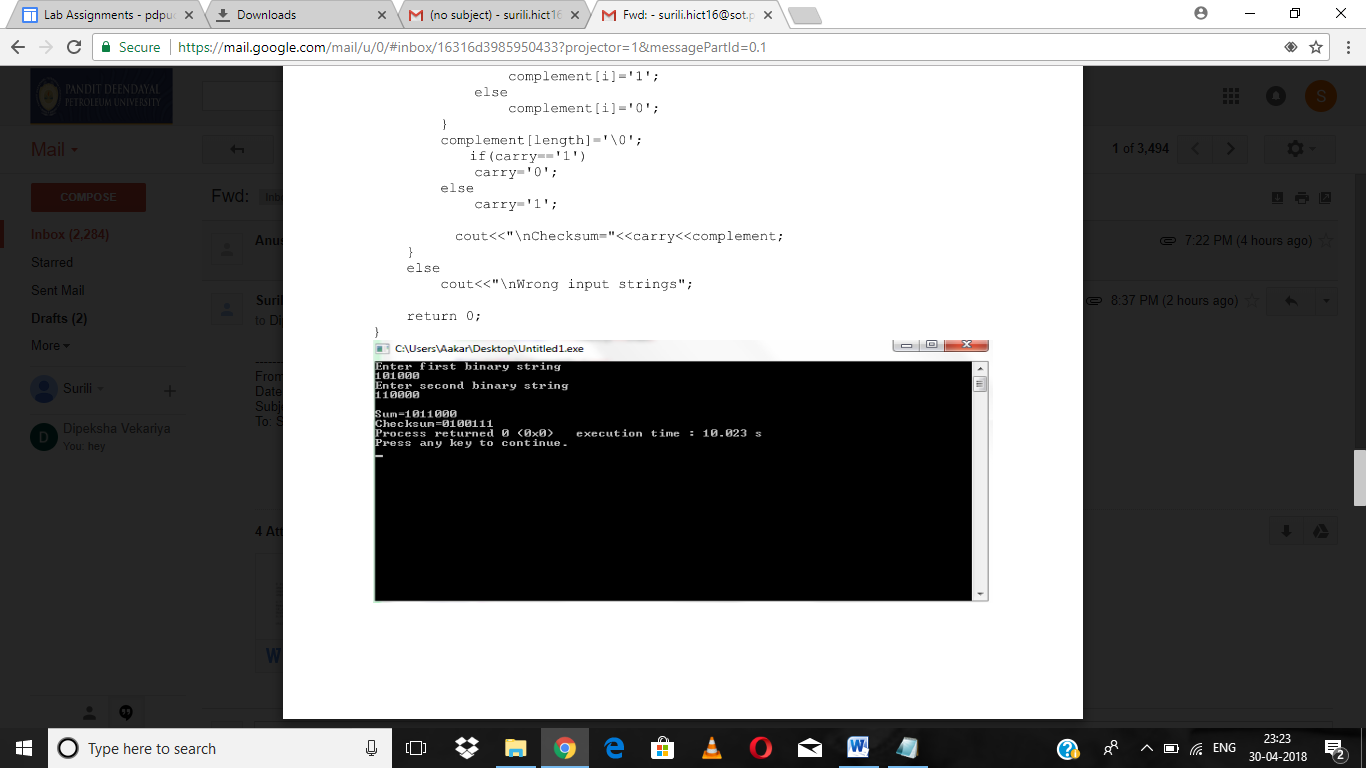
}

else

cout<<"\nWrong input strings";

return 0;

}



CRC-RECEIVER

# First of all import the socket library

import socket

def xor(a, b):

# initialize result

result = []

# Traverse all bits, if bits are

# same, then XOR is 0, else 1

for i in range(1, len(b)):

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

result.append('0')

else:

result.append('1')

return ''.join(result)

# Performs Modulo-2 division

def mod2div(divident, divisor):

# Number of bits to be XORed at a time.

pick = len(divisor)

# Slicing the divident to appropriate

# length for particular step

tmp = divident[0 : pick]

while pick < len(divident):

if tmp[0] == '1':

# replace the divident by the result

# of XOR and pull 1 bit down

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

else: # If leftmost bit is '0'

# If the leftmost bit of the dividend (or the

# part used in each step) is 0, the step cannot

# use the regular divisor; we need to use an

# all-0s divisor.

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

# increment pick to move further

pick += 1

# For the last n bits, we have to carry it out

# normally as increased value of pick will cause

# Index Out of Bounds.

if tmp[0] == '1':

tmp = xor(divisor, tmp)

else:

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

checkword = tmp

return checkword

# Function used at the receiver side to decode

# data received by sender

def decodeData(data, key):

l\_key = len(key)

# Appends n-1 zeroes at end of data

appended\_data = data + '0'\*(l\_key-1)

remainder = mod2div(appended\_data, key)

return remainder

# Creating Socket

s = socket.socket()

print ("Socket successfully created")

# reserve a port on your computer in our

# case it is 12345 but it can be anything

port = 12345

s.bind(('', port))

print ("socket binded to %s" % (port))

# put the socket into listening mode

s.listen(5)

print ("socket is listening")

while True:

# Establish connection with client.

c, addr = s.accept()

print('Got connection from', addr)

# Get data from client

data = c.recv(1024)

print(data)

if not data:

break

key = "1001"

ans = decodeData(data, key)

print("Remainder after decoding is->"+ans)

# If remainder is all zeros then no error occured

temp = "0" \* (len(key) - 1)

if ans == temp:

c.sendall("THANK you Data ->"+data + " Received No error FOUND")

else:

c.sendall("Error in data")

c.close()

CRC-SENDER

Import socket module

import socket

def xor(a, b):

# initialize result

result = []

# Traverse all bits, if bits are

# same, then XOR is 0, else 1

for i in range(1, len(b)):

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

result.append('0')

else:

result.append('1')

return ''.join(result)

# Performs Modulo-2 division

def mod2div(divident, divisor):

# Number of bits to be XORed at a time.

pick = len(divisor)

# Slicing the divident to appropriate

# length for particular step

tmp = divident[0 : pick]

while pick < len(divident):

if tmp[0] == '1':

# replace the divident by the result

# of XOR and pull 1 bit down

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

else: # If leftmost bit is '0'

# If the leftmost bit of the dividend (or the

# part used in each step) is 0, the step cannot

# use the regular divisor; we need to use an

# all-0s divisor.

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

# increment pick to move further

pick += 1

# For the last n bits, we have to carry it out

# normally as increased value of pick will cause

# Index Out of Bounds.

if tmp[0] == '1':

tmp = xor(divisor, tmp)

else:

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

checkword = tmp

return checkword

# Function used at the sender side to encode

# data by appending remainder of modular divison

# at the end of data.

def encodeData(data, key):

l\_key = len(key)

# Appends n-1 zeroes at end of data

appended\_data = data + '0'\*(l\_key-1)

remainder = mod2div(appended\_data, key)

# Append remainder in the original data

codeword = data + remainder

return codeword

# Create a socket object

s = socket.socket()

# Define the port on which you want to connect

port = 12345

# connect to the server on local computer

s.connect(('127.0.0.1', port))

# Send data to server 'Hello world'

## s.sendall('Hello World')

input\_string = raw\_input("Enter data you want to send->")

#s.sendall(input\_string)

data =(''.join(format(ord(x), 'b') for x in input\_string))

print data

key = "1001"

ans = encodeData(data,key)

print(ans)

s.sendall(ans)

# receive data from the server

print s.recv(1024)

# close the connection

s.close()

HAMMING CODE

#include<stdio.h>

void main() {

int data[10];

int dataatrec[10],c,c1,c2,c3,i;

printf("Enter 4 bits of data one by one\n");

scanf("%d",&data[0]);

scanf("%d",&data[1]);

scanf("%d",&data[2]);

scanf("%d",&data[4]);

//Calculation of even parity

data[6]=data[0]^data[2]^data[4];

data[5]=data[0]^data[1]^data[4];

data[3]=data[0]^data[1]^data[2];

printf("\nEncoded data is\n");

for(i=0;i<7;i++)

printf("%d",data[i]);

printf("\n\nEnter received data bits one by one\n");

for(i=0;i<7;i++)

scanf("%d",&dataatrec[i]);

c1=dataatrec[6]^dataatrec[4]^dataatrec[2]^dataatrec[0];

c2=dataatrec[5]^dataatrec[4]^dataatrec[1]^dataatrec[0];

c3=dataatrec[3]^dataatrec[2]^dataatrec[1]^dataatrec[0];

c=c3\*4+c2\*2+c1 ;

if(c==0) {

printf("\nNo error while transmission of data\n");

}

else {

printf("\nError on position %d",c);

printf("\nData sent : ");

for(i=0;i<7;i++)

printf("%d",data[i]);

printf("\nData received : ");

for(i=0;i<7;i++)

printf("%d",dataatrec[i]);

printf("\nCorrect message is\n");

//if errorneous bit is 0 we complement it else vice versa

if(dataatrec[7-c]==0)

dataatrec[7-c]=1;

else

dataatrec[7-c]=0;

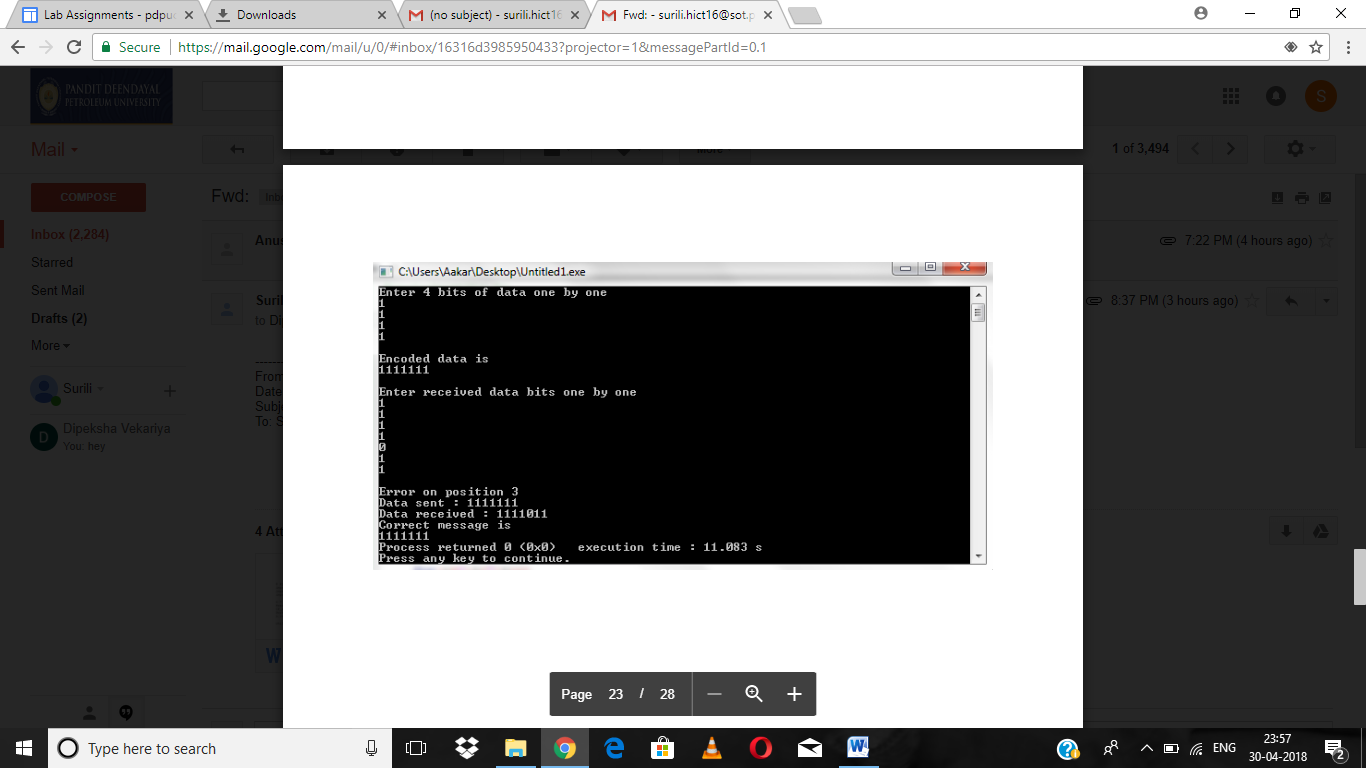
for (i=0;i<7;i++) {

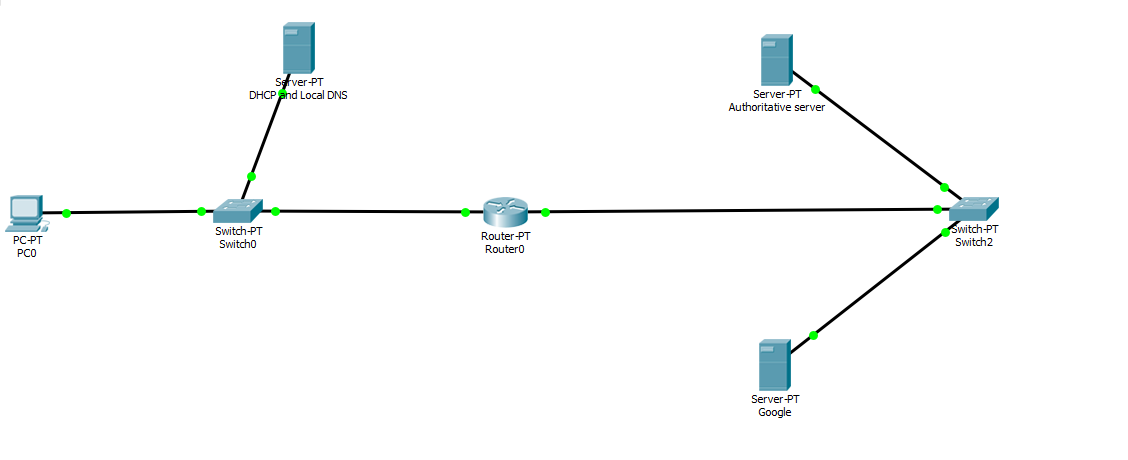
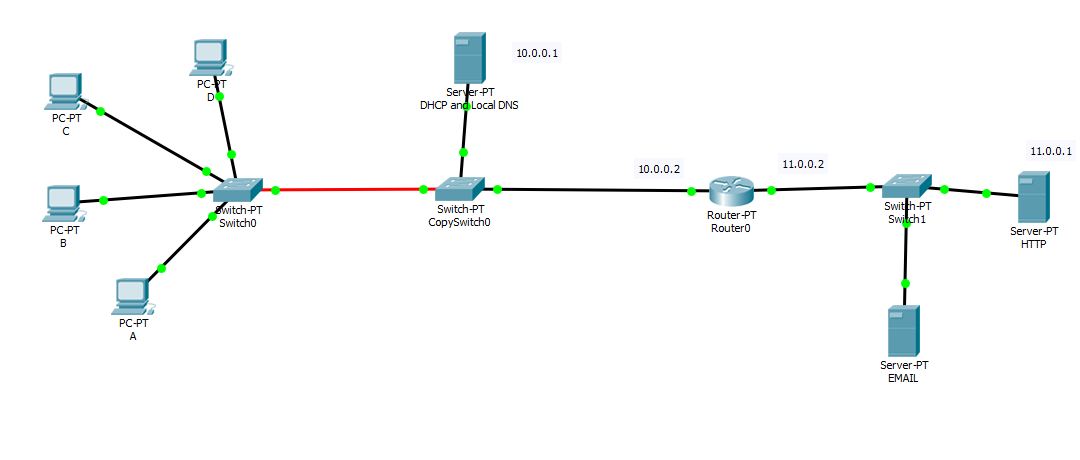
printf("%d",dataatrec[i]);

}

}

}





GO BACK N

#include<stdio.h>

int main()

{

int windowsize,sent=0,ack,i;

printf("enter window size\n");

scanf("%d",&windowsize);

while(1)

{

for( i = 0; i < windowsize; i++)

{

printf("Frame %d has been transmitted.\n",sent);

sent++;

if(sent == windowsize)

break;

}

printf("\nPlease enter the last Acknowledgement received.\n");

scanf("%d",&ack);

if(ack == windowsize)

break;

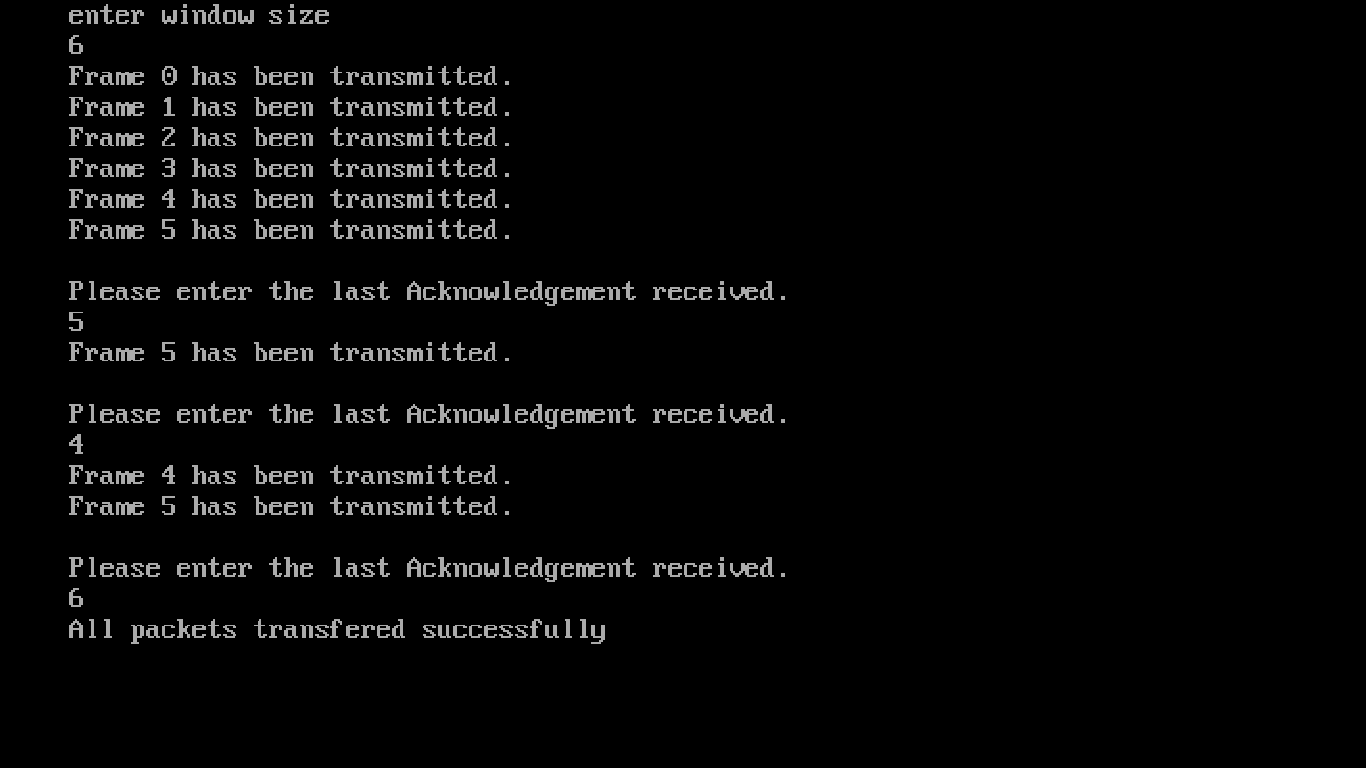
else

sent = ack;

}

return 0;

}



STOP AND WAIT

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

void main()

{

int i,j,noframes,x,x1=10,x2;

clrscr();

for(i=0;i<200;i++)

rand();

noframes=rand()/200;

i=1;

j=1;

noframes = noframes / 8;

printf("\n number of frames is %d",noframes);

getch();

while(noframes>0)

{

printf("\nsending frame %d",i);

srand(x1++);

x = rand()%10;

if(x%2 == 0)

{

for (x2=1; x2<2; x2++)

{

printf("waiting for %d seconds\n", x2);

sleep(x2);

}

printf("\nsending frame %d",i);

srand(x1++);

x = rand()%10;

}

printf("\nack for frame %d",j);

noframes-=1; i++; j++;

}

printf("\n end of stop and wait protocol");

getch();

}

SELECTIVE REPEAT

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

void main()

{

int temp1,temp2,temp3,temp4,temp5,i,winsize=8,noframes,moreframes;

char c;

int reciever(int);

int simulate(int);

int nack(int);

clrscr();

temp4=0,temp1=0,temp2=0,temp3=0,temp5 = 0;

for(i=0;i<200;i++)

rand();

noframes=rand()/200;

printf("\n number of frames is %d",noframes);

getch();

moreframes=noframes;

while(moreframes>=0)

{

temp1=simulate(winsize);

winsize-=temp1;

temp4+=temp1;

if(temp4 >noframes)

temp4 = noframes;

for(i=noframes - moreframes;i<=temp4;i++)

printf("\nsending frame %d",i);

getch();

temp2=reciever(temp1);

temp3+=temp2;

if(temp3 > noframes)

temp3 = noframes;

temp2 = nack(temp1);

temp5+=temp2;

if (temp5 !=0)

{

printf("\n No acknowledgement for the frame %d",temp5);

getch();

for(i=1;i<temp5;i++) ;

printf("\n Retransmitting frame %d",temp5);

getch();

}

moreframes-=temp1;

if(winsize<=0)

winsize=8;

}

printf("\n end of sliding window protocol Selective Reject");

getch();

}

int reciever(int temp1)

{

int i; for(i=1;i<100;i++)

rand();

i=rand()%temp1; return i;

}

int nack(int temp1)

{

int i;

for(i=1;i<100;i++)

rand();

i=rand()%temp1;

return i;

}

int simulate(int winsize)

{

int temp1,i;

for(i=1;i<50;i++)

temp1=rand();

if(temp1==0)

temp1=simulate(winsize);

i = temp1%winsize;

if(i==0)

return winsize;

else

return temp1%winsize;

}