DCN LAB PROGRAMMING

# SINGLE PARITY SENDER:

print("\_" \* 30 + "19MID0017" + "\_" \* 30)

n = 0

print("At the Sender End...")

n = int(input("Enter the standard that sender wants to follow: '1' for 'Odd\_parity' and '2' for 'Even\_parity': "))

data = input("Enter the data of sender in binary : ")

list\_1 = list(data)

list\_data = [int(i) for i in list\_1]

add = sum(list\_data)

if (n == 1):

if (add % 2 == 0):

list\_data.append(1)

else:

list\_data.append(0)

else:

if (add % 2 == 0):

list\_data.append(0)

else:

list\_data.append(1)

string = ''.join([str(i) for i in list\_data])

print(f"Code generated by sender is {string}")

# SINGLE PARITY RECEIVER

print("\_" \* 30 + "19MID0017" + "\_" \* 30)

print("At the receiver end...")

n = int(input("Enter the standard that receiver wants to follow:'1' for 'Odd\_parity' and '2' for 'Even\_parity': "))

data = input("Enter the received data in binary : ")

print(f"Received data is '{data}'")

print("Checking....")

check = [int(i) for i in data]

sum\_check = sum(check)

if(n == 2):

if(sum\_check % 2 == 0):

print("Data received correctly...")

else:

print("Data received with some error")

else:

if(sum\_check % 2 != 0):

print("Data received correctly...")

else:

print("Data received with some error")

# 2D PARITY SENDER:

def get\_even(list1):

if sum(list1) % 2 != 0:

list1.append(1)

else:

list1.append(0)

return list1

def get\_odd(list1):

if sum(list1) % 2 == 0:

list1.append(1)

else:

list1.append(0)

return list1

def check\_column(f\_list):

global nop, n

list\_re = []

for i in range(nop + 1):

list\_re.append(sum([x[i] for x in f\_list]))

if(n == 2):

for i in range(len(list\_re)):

list\_re[i] = list\_re[i] % 2

else:

for i in range(len(list\_re)):

list\_re[i] = (list\_re[i] + 1) % 2

return list\_re

def to\_string(listfin, m):

string2 = m.join([str(y) for y in listfin])

return string2

n = 0

n = int(input("Enter the standard that sender and receiver are gonna follow :'1' for 'Odd\_parity' and '2' for 'Even\_parity' : "))

print("At the Sender End...")

data = input("Enter the data to send in binary : ")

nop = int(

input("Enter the no of bits per packets for your data have to be encrypted : "))

extra = (nop - ((len(data)) % nop)) % nop

extra\_bits = str('0' \* extra)

data = extra\_bits + data

list\_data = [[int(x) for x in data[i:i + nop]]

for i in range(0, len(data), nop)]

if (n == 2):

for i in list\_data:

get\_even(i)

else:

for i in list\_data:

get\_odd(i)

list\_data.append(check\_column(list\_data))

for l in range(len(list\_data)):

list\_data[l] = to\_string(list\_data[l], '')

Sender\_data = to\_string(list\_data, ' ')

print("Code generated by the Sender is : ", Sender\_data)

# 2D PARITY RECEIVER:

print("At the receiver End : ")

n = 0

n = int(input("Enter the standard that receiver wants to follow :'1' for 'Odd\_parity' and '2' for 'Even\_parity' : "))

data = input("Enter the data to send in binary : ")

nop = int(

input("Enter the no of bits per packets which sender used to encrypt : ")) + 1

list\_data = [[int(x) for x in data[i:i + nop]]

for i in range(0, len(data), nop)]

print("Checking....")

if n == 1:

result = [sum(i) % 2 == 1 for i in list\_data]

else:

result = [sum(i) % 2 == 0 for i in list\_data]

if all(result):

print("Data received without error ")

else:

print("Incorrect Data")

# CHECKSUM SENDER:

def checksum(list1, list2):

L2 = list2[::-1]

global L3

L3.append(0)

L4 = []

for num, x in enumerate(zip(list1, L2, L3)):

if (sum(x) >= 2):

L3[num + 1] = 1

L3[num] = 0

L4.append(sum(x) % 2)

else:

L4.append(sum(x))

L3[num] = 0

if(L3[-1] == 1):

L3[0] = 1

for num, x in enumerate(zip(L4, L3)):

if (sum(x) == 2):

L3[num + 1] = 1

L4[num] = L3[num] = 0

else:

L4[num] = sum(x)

L3[num] = 0

L3.pop()

return L4

def complement():

comp = {0: 1, 1: 0}

global L1, nop

for i in range(nop):

L1[i] = str(comp.get(L1[i]))

return ''.join(L1)

data = input("Enter the data to send in binary : ")

nop = int(

input("Enter the no of bits per packet for your data have to be encrypted : "))

print("Processing...")

print("-" \* 75)

extra = (nop - ((len(data)) % nop)) % nop

extra\_bits = str('0' \* extra)

data = extra\_bits + data

list\_data = [[int(x) for x in data[i:i + nop]]

for i in range(0, len(data), nop)]

L1 = list(map(int, '0' \* nop))

L3 = L1

for i in list\_data:

L1 = checksum(L1, i)

L1 = L1[::-1]

string = complement()

print(f"Checksum produced by the sender is {string}")

print("\_" \* 75)

data = data + string

print(f"Data generated by the sender with checksum is {data}")

print("=" \* 75)

# CHECKSUM RECEIVER:

def checksum(list1, list2):

L2 = list2[::-1]

global L3

L3.append(0)

L4 = []

for num, x in enumerate(zip(list1, L2, L3)):

if (sum(x) >= 2):

L3[num + 1] = 1

L3[num] = 0

L4.append(sum(x) % 2)

else:

L4.append(sum(x))

L3[num] = 0

if(L3[-1] == 1):

L3[0] = 1

for num, x in enumerate(zip(L4, L3)):

if (sum(x) == 2):

L3[num + 1] = 1

L4[num] = L3[num] = 0

else:

L4[num] = sum(x)

L3[num] = 0

L3.pop()

return L4

def complement():

comp = {0: 1, 1: 0}

global L1, nop

for i in range(nop):

L1[i] = str(comp.get(L1[i]))

return ''.join(L1)

print('At the receiver end')

data = input("Enter the data sent by sender in binary : ")

nop = int(

input("Enter the no of bits per packet for your data have to be encrypted : "))

print("Processing...")

print("\_" \* 75)

list\_data = [[int(x) for x in data[i:i + nop]]

for i in range(0, len(data), nop)]

L1 = list(map(int, '0' \* nop))

L3 = L1

for i in list\_data:

L1 = checksum(L1, i)

print(f"Final sum of the received data is : {''.join(map(str,L1))}")

receiver\_string = complement()

print(f"complemet of the sum is {receiver\_string}")

print("\_"\*75)

filtered = list(filter(lambda x: x == '1', receiver\_string))

if (any(filtered)):

print("Error occured")

else:

print("Data received without any error")

# CRC SENDER:

from collections import deque as dq

div\_1 = []

div\_2 = []

def polynomial(ch):

global div\_1, div\_2

if(ch == 1):

string1 = "1101"

div\_1 = list(string1)

string2 = "0000"

div\_2 = list(string2)

opt = 4

elif(ch == 2):

string1 = "10100111"

div\_1 = list(string1)

string2 = "0" \* 8

div\_2 = list(string2)

opt = 8

elif(ch == 3):

string1 = "11" + "0" \* 7 + "1011"

div\_1 = list(string1)

string2 = "0" \* 13

div\_2 = list(string2)

opt = 13

else:

string1 = "11" + "0" \* 12 + "101"

div\_1 = list(string1)

string2 = "0" \* 17

div\_2 = list(string2)

opt = 17

div\_1 = list(map(int, div\_1))

div\_2 = list(map(int, div\_2))

return opt

def crc\_code():

global list1

list1 = list(map(int, list1))

i = opt

while(i < len(list1)):

division()

q.popleft()

q.append(list1[i])

i += 1

division()

q.popleft()

def division():

global q, div\_1, div\_2

if(q[0] == 1):

for i in range(4):

q[i] = q[i] ^ div\_1[i]

else:

for i in range(4):

q[i] = q[i] ^ div\_2[i]

data = input("Enter the data : ")

list1 = [int(i) for i in data]

print("\nSelect a generator polynomial from the following..\n")

print(" 1) Cubic polynomial ( x^3+x^2+1 ) \n 2) Septic polynomial ( x^7+x^5+x^2+x+1)")

print(" 3) CRC-12 ( x^12+x^5+x^2+x+1 ) \n 4) CRC-16 ( x^16+x^15+x^2+1 )\n")

ch = int(input("Please enter your choice : "))

opt = polynomial(ch)

add\_on = "0" \* (opt - 1)

list1.extend(list(add\_on))

q = dq(list1[:opt])

crc\_code()

string = list(map(lambda x: str(x), q))

final\_string = ''.join(data) + ''.join(string)

print(f"Code generated by sender is '{final\_string}'")

# CRC RECEIVER:

from collections import deque as dq

div\_1 = []

div\_2 = []

def polynomial(ch):

global div\_1, div\_2

if(ch == 1):

string1 = "1101"

div\_1 = list(string1)

string2 = "0000"

div\_2 = list(string2)

opt = 4

elif(ch == 2):

string1 = "10100111"

div\_1 = list(string1)

string2 = "0" \* 8

div\_2 = list(string2)

opt = 8

elif(ch == 3):

string1 = "11" + "0" \* 7 + "1011"

div\_1 = list(string1)

string2 = "0" \* 13

div\_2 = list(string2)

opt = 13

else:

string1 = "11" + "0" \* 12 + "101"

div\_1 = list(string1)

string2 = "0" \* 17

div\_2 = list(string2)

opt = 17

div\_1 = list(map(int, div\_1))

div\_2 = list(map(int, div\_2))

return opt

def crc\_code():

global list1

list1 = list(map(int, list1))

i = opt

while(i < len(list1)):

division()

q.popleft()

q.append(list1[i])

i += 1

division()

q.popleft()

def division():

global q, div\_1, div\_2

if(q[0] == 1):

for i in range(4):

q[i] = q[i] ^ div\_1[i]

else:

for i in range(4):

q[i] = q[i] ^ div\_2[i]

data = input("Enter the data : ")

list1 = [int(i) for i in data]

print("\nSelect a generator polynomial from the following..\n")

print(" 1) Cubic polynomial ( x^3+x^2+1 ) \n 2) Septic polynomial ( x^7+x^5+x^2+x+1)")

print(" 3) CRC-12 ( x^12+x^5+x^2+x+1 ) \n 4) CRC-16 ( x^16+x^15+x^2+1 )\n")

ch = int(input("Please enter your choice : "))

opt = polynomial(ch)

q = dq(list1[:opt])

crc\_code()

string = list(map(lambda x: str(x), q))

print("Remainder : ", ''.join(string))

if(any(q)):

print("Some error has occured while transmission..")

else:

print("Data received successfully!!!")

# STOP AND WAIT ARQ:

import threading

import time

import random

print("-" \* 10 + "Mothishwaran.C 19MID0017" + "-" \* 10)

receiverlist = []

frames = [0, 1, 2, 3, 4, 5, 6]

sf = 0

rn = 1

def thread\_make(y):

x = 0

while(x < (4)):

time.sleep(1)

x += 1

def receiver(z):

global rn

time.sleep(2)

y = random.randrange(0, 50) % 7

if (y != 0):

print(f"Ack{z+1}")

rn += 1

receiverlist.append(z)

else:

print(f"\_\_\_\_ -->Acknowledgement lost")

while(sf < len(frames)):

if(sf < rn):

print(f"Transmitting frame-{sf}....")

else:

sf -= 1

print(f"ReTransmitting frame-{sf}....")

t = threading.Thread(target=thread\_make, args=(sf,))

t.start()

r = threading.Thread(target=receiver, args=(sf,))

r.start()

t.join()

r.join()

# time.sleep(1)

sf += 1

print(f"Receiver received : {receiverlist}")

# GO BACK NARQ :

import threading

import time

from collections import deque as que

import random

print("-" \* 10 + "Mothishwaran.C 19MID0017" + "-" \* 10)

window = que([0, 1, 2, 3])

frames = [4, 5, 6, -1, -1, -1, -1]

rn = que([0] \* 10)

s = 0

finalreceive = [0] \* 7

frn = 0

def add\_parity(p\_list):

if sum(p\_list) % 2 == 0:

p\_list.append(0)

else:

p\_list.append(1)

return p\_list

def framing(list1):

m = len(bin(max(list1)).replace("0b", ""))

list1 = map(lambda x: format(x, '0' + str(m) + 'b'), list1)

list1 = [list(map(int, i)) for i in list1]

list1 = list(map(add\_parity, list1))

return list1

def send(y, sv):

x = 0

while(x < (sv + 4)):

time.sleep(1)

x += 1

def receiver(z, sv):

global finalreceive, window, frames, rn, s, frn

time.sleep(sv + 2)

y = random.randrange(0, 50) % 7

if ((y != 0) and (rn[0] == 0)):

print(f"Ack {z+1} -->confirms the frame-{z} has received")

finalreceive[frn] = z

window.popleft()

window.append(frames[s])

s += 1

frn += 1

elif (y != 0):

print(f"Ack {z+1} But discard it ")

else:

print("\_\_\_\_\_ -->Acknowledgement lost !!!")

rn[0] = -1

# framed\_list=framing()

while(sum(window) > -4):

threadList = []

ReceiveList = []

sleepvar = 0

if(rn[0] == -1):

print("Retransmitting the current window....")

else:

print("Transmitting the current window.....")

rn[0] = 0

print("frames in current window:", end=" ")

[print(frame, end=" ") for frame in window if frame != -1]

print()

for i in window:

if(rn[i] == 0) and i >= 0:

t = threading.Thread(target=send, args=(i, sleepvar))

threadList.append(t)

t.start()

r = threading.Thread(target=receiver, args=(i, sleepvar))

ReceiveList.append(r)

r.start()

sleepvar += 1

#print("for loop done")

for i in threadList:

i.join()

for r1 in ReceiveList:

r1.join()

print(f"Receiver received : {finalreceive}")

# SELECTIVE REPEAT:

import threading

import time

from collections import deque as que

import random

print("-" \* 10 + "Mothishwaran.C 19MID0017" + "-" \* 10)

window = que([0, 1, 2, 3])

frames = [4, 5, 6, -1, -1, -1, -1]

rn = que([])

s = 0

finalreceive = [0] \* 7

frn = 0

def send(y, sv):

x = 0

while(x < (sv + 4)):

time.sleep(1)

x += 1

def receiver(z, sv):

global finalreceive, window, frames, rn, s, frn

time.sleep(sv + 2)

y = random.randrange(0, 14) % 3

if (y != 0):

print(f"Ack {z+1} -->confirms the frame-{z} has received")

finalreceive[z] = z

window.append(frames[s])

s += 1

frn += 1

else:

print(f"NAK {z} -->frame- {z} has to be send again")

rn.appendleft(z)

window.popleft()

while(sum(window) > -4):

threadList = []

ReceiveList = []

sleepvar = 0

while(rn):

print(f"Retransmitting....{rn[0]}")

window.appendleft(rn[0])

rn.popleft()

print("Transmitting.....")

print("frames in current window:", end=" ")

[print(frame, end=" ") for frame in window if frame != -1]

print()

for i in window:

if i >= 0:

t = threading.Thread(target=send, args=(i, sleepvar))

threadList.append(t)

t.start()

r = threading.Thread(target=receiver, args=(i, sleepvar))

ReceiveList.append(r)

r.start()

sleepvar += 1

for i in threadList:

i.join()

for r1 in ReceiveList:

r1.join()

print(f"Receiver received : {finalreceive}")

# Dijkstra with routing table (link state):

def intConversion(a):

if a == '\_':

return 9999

else:

return int(a)

def select(list1, sv):

global bool\_list, V

minimum\_val = 9999

for i in range(V):

if bool\_list[i] is False and list1[i] < minimum\_val:

minimum\_val = list1[i]

if minimum\_val != 9999:

bool\_list[list1.index(minimum\_val)] = True

return minimum\_val

else:

return sv

def minimum(a, b, index, ind\_a, ind\_b):

global precedence

if a < b:

return a

else:

precedence[index] = ind\_b

return b

def routing\_table(sv):

print(f"Routing table of {sv} is ")

print("Node" + "\t" + "Cost" + "\t" + "Next Router")

global G, precedence, V

for i in range(V):

print(f"{i}\t\t{G[sv][i]}\t\t{precedence[i]}")

print("-" \* 10 + " Mothishwaran C -19MID0017" + "-" \* 10)

V = int(input("Enter the number of Vertices : "))

print("Please Enter the respected details : ")

print("""Caution:

1) For self-loops enter '0'

2)If there is no path is between the two vertices enter '\_'

3)Enter the distance from a vertice to all other vertices seperated by comma """)

Adj\_list = [list(input(f"Enter the distance from vertex {i} to all : ").split(

' ')) for i in range(V)]

G = [list(map(intConversion, i[:V])) for i in Adj\_list]

print("The Adjacent Matrix of the Graph is ")

for i in G:

print(i)

for k in range(V):

visited = [k]

bool\_list = [False] \* V

bool\_list[k] = True

precedence = ['-'] \* V

nv = []

for i in range(V):

if i not in visited:

nv.append(i)

for \_ in range(V):

index\_w = select(G[k], k)

w = G[k].index(index\_w)

if w not in visited:

visited.append(w)

nv.remove(w)

for j in nv:

G[k][j] = minimum(G[k][j], (G[k][w] + G[w][j]), j, k, w)

routing\_table(k)

# SOCKET:

## SERVER SIDE:

import socket as soc

import threading as th

'''

1) Pick the port

2) Pick the server

3) Pick the socket

4) Bind the socket

'''

HEADER = 64

PORT = 9999

# or use loopback address [localhost]

SERVER = soc.gethostbyname(soc.gethostname())

# gethostname returns device name gethostby name returns it ip address

ADDR = (SERVER, PORT)

FORMAT = "utf-8"

DISCONNECT\_MESSAGE = "Disconnect !"

RESPONSE\_MESSAGE = "Message received successfully...."

# Socket creation

# first argument --> famiily (categories) 2nd -->ways of sending data (protocols)

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

# the first argument tells the what type of ip address or type of address that we gonna be accepting or looking for specific connections

server.setsockopt(soc.SOL\_SOCKET, soc.SO\_REUSEADDR, 1) #for immediate use of that port

server.bind(ADDR)

def handle\_client(client\_conn, addr):

print(f"[NEW CONNECTION] {addr} connected.")

connected = True

while connected:

msg\_length = client\_conn.recv(HEADER).decode(FORMAT)

# conn.recv() takes argument of length of the msg

# every time we send msg need to encode in byte format

if msg\_length:

msg\_length = int(msg\_length)

msg = client\_conn.recv(msg\_length).decode(FORMAT)

if msg == DISCONNECT\_MESSAGE:

connected = False

print(f"[{addr}] {msg}")

response\_msg\_length = len(RESPONSE\_MESSAGE)

send\_length = str(response\_msg\_length).encode(FORMAT)

send\_length += b' ' \* (HEADER - len(send\_length))

client\_conn.send(send\_length)

client\_conn.send(RESPONSE\_MESSAGE.encode(FORMAT))

client\_conn.close()

def start():

# listen

server.listen()

print(f"[LISTENING] Server is listening on {SERVER}")

while True: # continue to listen until turn off or it crashes

# this line waits for new connection to occur & it stores the address as well as the port

client\_conn, addr = server.accept()

# this function will return the socket\_object & address-port tuple

thread = th.Thread(target=handle\_client, args=(client\_conn, addr))

thread.start()

# to exclude the main thread (this paython program ) we sub 1

print(f"[ACTIVE CONNECTION] {th.activeCount()-1}")

print("[STARTING] server is starting...")

start()

## CLIENT:

import socket as soc

HEADER = 64

PORT = 9999

FORMAT = "utf-8"

DISCONNECT\_MESSAGE = "Disconnect !"

SERVER = soc.gethostbyname(soc.gethostname())

ADDR = (SERVER, PORT)

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

client.connect(ADDR)

def send(msg):

message = msg.encode(FORMAT)

msg\_length = len(message)

send\_length = str(msg\_length).encode(FORMAT)

send\_length += b' ' \* (HEADER - len(send\_length))

# b' ' -->byte format specifier padding

client.send(send\_length)

client.send(message)

msg\_length = client.recv(HEADER).decode(FORMAT)

msg\_length = int(msg\_length)

msg = client.recv(msg\_length).decode(FORMAT)

print(msg)

send("Hello Mothish !")

input()

send("Success !")

input()

send("Done !")

send(DISCONNECT\_MESSAGE)