**CN LAB REPORT**

**SAKSHI ROY**

**5-C**

**CYCLE-2**

# **Write a program for error detecting code using CRC-CCITT (16-bits).**

# #include<stdio.h>

# char m[50],g[50],r[50],q[50],temp[50];

# void caltrans(int);

# void crc(int);

# void calram();

# void shiftl();

# int main()

# {

# int n,i=0;

# char ch,flag=0;

# printf("Enter the frame bits:");

# while((ch=getc(stdin))!='\n')

# m[i++]=ch;

# n=i;

# for(i=0;i<16;i++)

# m[n++]='0';

# m[n]='\0';

# printf("Message after appending 16 zeros:%s",m);

# for(i=0;i<=16;i++)

# g[i]='0';

# g[0]=g[4]=g[11]=g[16]='1';g[17]='\0';

# printf("\ngenerator:%s\n",g);

# crc(n);

# printf("\n\nquotient:%s",q);

# caltrans(n);

# printf("\ntransmitted frame:%s",m);

# printf("\nEnter transmitted frame:");

# scanf("\n%s",m);

# printf("CRC checking\n");

# crc(n);

# printf("\n\nlast remainder:%s",r);

# for(i=0;i<16;i++)

# if(r[i]!='0')

# flag=1;

# else

# continue;

# if(flag==1)

# printf("Error during transmission");

# else

# printf("\n\nReceived frame is correct");

# }

# void crc(int n)

# {

# int i,j;

# for(i=0;i<n;i++)

# temp[i]=m[i];

# for(i=0;i<16;i++)

# r[i]=m[i];

# printf("\nintermediate remainder\n");

# for(i=0;i<n-16;i++)

# {

# if(r[0]=='1')

# {

# q[i]='1';

# calram();

# }

# else

# {

# q[i]='0';

# shiftl();

# }

# r[16]=m[17+i];

# r[17]='\0';

# printf("\nremainder %d:%s",i+1,r);

# for(j=0;j<=17;j++)

# temp[j]=r[j];

# }

# q[n-16]='\0';

# }

# void calram()

# {

# int i,j;

# for(i=1;i<=16;i++)

# r[i-1]=((int)temp[i]-48)^((int)g[i]-48)+48;

# }

# void shiftl()

# {

# int i;

# for(i=1;i<=16;i++)

# r[i-1]=r[i];

# }

# void caltrans(int n)

# {

# int i,k=0;

# for(i=n-16;i<n;i++)

# m[i]=((int)m[i]-48)^((int)r[k++]-48)+48;

# m[i]='\0';

# 

1. **Write a program for distance vector algorithm to find suitable path for transmission.**

# #include <iostream>

# #include <stdio.h>

# using namespace std;

# struct router

# {

# int dist[10];

# int next[10];

# } router[10];

# int main()

# {

# int no;

# cout << "Enter number of router : " << endl;

# cin >> no;

# cout << "Enter adjacency matrix : " << endl;

# int vt[no][no];

# for (int i = 0; i < no; i++)

# {

# for (int j = 0; j < no; j++)

# {

# cin >> router[i].dist[j];

# router[i].next[j] = j;

# }

# cout << endl;

# }

# for (int i = 0; i < no; i++)

# {

# for (int j = 0; j < no; j++)

# {

# for (int k = 0; k < no; k++)

# {

# if (router[i].dist[j] > router[i].dist[k] + router[k].dist[j])

# {

# router[i].dist[j] = router[i].dist[k] + router[j].dist[k];

# router[i].next[j] = k;

# }

# }

# }

# }

# for (int i = 0; i < no; i++)

# {

# cout << "Router info for router: " << i + 1 << endl;

# cout << "Dest\tNext Hop\tDist" << endl;

# for (int j = 0; j < no; j++)

# printf("%d\t%d\t\t%d\n", j + 1, router[i].next[j] + 1, router[i].dist[j]);

# }

# return 0;

# }

# 

# **Implement Dijkstra’s algorithm to compute the shortest path for a given topology.**

# #include <bits/stdc++.h>

# using namespace std;

# int a[30][30], source, dist[30], path[30];

# void dijkstar(int a[][30], int n)

# {

# set<pair<int, int>> s;

# for (int i = 0; i < n; i++)

# {

# dist[i] = a[source][i];

# path[i] = source;

# s.insert({dist[i], i});

# }

# while (!s.empty())

# {

# pair<int, int> t = \*s.begin();

# s.erase(s.begin());

# for (int i = 0; i < n; i++)

# {

# if (dist[i] > t.first + a[t.second][i])

# {

# dist[i] = dist[t.second] + a[t.second][i];

# path[i] = t.second;

# s.insert({dist[i], i});

# }

# }

# }

# }

# int main()

# {

# int n;

# cout << "Enter the no. of vertices :" << endl;

# cin >> n;

# cout << "Enter the adjacency matrix(Enter 9999 for infinity):" << endl;

# for (int i = 0; i < n; i++)

# {

# for (int j = 0; j < n; j++)

# {

# cin >> a[i][j];

# }

# }

# cout << "Enter the source vertex :" << endl;

# cin >> source;

# cout << "The shortest paths from vertex ' " << source << " ' are :" << endl;

# cout << "Vertex paths" << endl;

# dijkstar(a, n);

# for (int i = 0; i < n; i++)

# {

# int k = i;

# while (k != source)

# {

# cout << k << " <- ";

# k = path[k];

# }

# cout << source << " = ";

# cout << "Path cost:" << dist[i] << endl;

# }

# return 0;

# }

# 

# **Write a program for congestion control using Leaky bucket algorithm.**

# #include <Windows.h>

# using namespace std;

# #define bucketSize 500

# void bucketInput(int a, int b)

# {

# if (a > bucketSize)

# cout << "\n\t\tBucket overflow";

# else

# {

# Sleep(5);

# while (a > b)

# {

# cout << "\n\t\t" << b << " bytes outputted.";

# a -= b;

# Sleep(5);

# }

# if (a > 0)

# cout << "\n\t\tLast " << a << " bytes sent\t";

# cout << "\n\t\tBucket output successful";

# }

# }

# int main()

# {

# int op, pktSize;

# cout << "Enter output rate : ";

# cin >> op;

# for (int i = 1; i <= 5; i++)

# {

# Sleep(rand() % 10);

# pktSize = rand() % 700;

# cout << "\nPacket no " << i << "\tPacket size = " << pktSize;

# bucketInput(pktSize, op);

# }

# cout << endl;

# return 0;

# }

# 

1. **Using TCP/IP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.**

**from socket import \***

serverName = "127.0.0.1"

serverPort = 12000

serverSocket = socket(AF\_INET, SOCK\_STREAM)

serverSocket.bind((serverName, serverPort))

serverSocket.listen(1)

while 1:

print("The server is ready to receive")

connectionSocket, addr = serverSocket.accept()

sentence = connectionSocket.recv(1024).decode()

file = open(sentence, "r")

l = file.read(1024)

connectionSocket.send(l.encode())

print('\nSent contents of ' + sentence)

file.close()

connectionSocket.close()

# from socket import \*

# serverName = '127.0.0.1'

# serverPort = 12000

# clientSocket = socket(AF\_INET, SOCK\_STREAM)

# clientSocket.connect((serverName, serverPort))

# sentence = input("\nEnter file name: ")

# clientSocket.send(sentence.encode())

# filecontents = clientSocket.recv(1024).decode()

# print('\nFrom Server:\n')

# print(filecontents)

# clientSocket.close()

# 

1. **Using UDP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.**

from socket import \*

serverName = "127.0.0.1"

serverPort = 12000

clientSocket = socket(AF\_INET, SOCK\_DGRAM)

sentence = input("\nEnter file name: ")

clientSocket.sendto(bytes(sentence,"utf-8"),(serverName, serverPort))

filecontents,serverAddress = clientSocket.recvfrom(2048)

print ('\nReply from Server:\n')

print (filecontents.decode("utf-8"))

# for i in filecontents:

# print(str(i), end = '')

clientSocket.close()

clientSocket.close()

from socket import \*

serverPort = 12000

serverSocket = socket(AF\_INET, SOCK\_DGRAM)

serverSocket.bind(("127.0.0.1", serverPort))

print ("The server is ready to receive")

while 1:

sentence, clientAddress = serverSocket.recvfrom(2048)

sentence = sentence.decode("utf-8")

file=open(sentence,"r")

l=file.read(2048)

serverSocket.sendto(bytes(l,"utf-8"),clientAddress)

print ('\nSent contents of ', end = ' ')

print (sentence)

# for i in sentence:

# print (str(i), end = '')

file.close()

