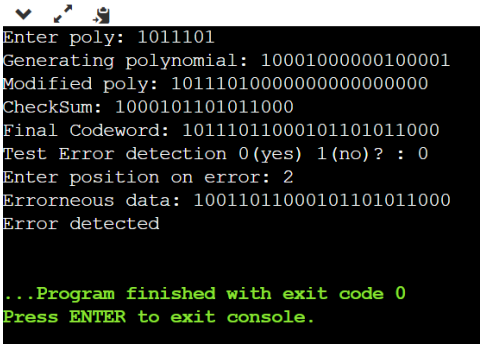
CN LAB CYCLE 2

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

```
import java.util.*;
public class Main{
   public static int n;
   public static void main(String[] args){
      Scanner in=new Scanner(System.in);
      Main ob=new Main();
      String code, copy, rec,zero="0000000000000000";
      System.out.print("Enter poly: ");
      code=in.nextLine();
      System.out.println("Generating polynomial: 10001000000100001");
      n=code.length();
      copy=code;
      code+=zero;
      System.out.println("Modified poly: "+code);
      code=ob.divide(code);
      System.out.println("CheckSum: "+code.substring(n));
      copy=copy.substring(0,n)+code.substring(n);
      System.out.println("Final Codeword: "+copy);
      // System.out.print("\nEnter recived data: ");
      // rec=in.nextLine();
      // if(zero.equals(ob.divide(rec).substring(n)))
      //    System.out.println("Correct bits recieved");
      // else
      //    System.out.println("Recieved frame contains one or more
errors");
      System.out.print("Test Error detection 0(yes) 1(no)? : ");
      int choice = in.nextInt();
      if(choice == 0){
         System.out.print("Enter position on error: ");
         int errorPos = in.nextInt();
         if(copy.charAt(errorPos) == '1')
        copy = copy.substring(0,errorPos) + "0" + copy.substring(errorPos+1);
```

```
        else
            copy = copy.substring(0,errorPos) + "1" +
copy.substring(errorPos+1);
          System.out.println("Errorneous data: "+copy);
          System.out.println("Error detected");
       }
       else
          System.out.println("No Error detection");
    }
    public String divide(String s){
       int i,j;
       char x;
       String div="10001000000100001";
       for(i=0;i<n;i++){
          x=s.charAt(i);
          for(j=0;j<17;j++){
             if(x=='1'){
                if(s.charAt(i+j)!=div.charAt(j))
                   s=s.substring(0,i+j)+"1"+s.substring(i+j+1);
                else
                   s=s.substring(0,i+j)+"0"+s.substring(i+j+1);
             }
          }
       }
       return s;
    }
}
```

OUTPUT:



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

```
#include<stdlib.h>
#include<stdio.h>
#define NUL 1000
#define NODES 10
struct node
{
int t[NODES][3];
};
struct node n[NODES];
typedef struct node NOD;
int main()
{
void init(int,int);
void inp(int,int);
void caller(int,int);
void op1(int,int,int);
void find(int,int);
int i,j,x,y,no;
do{
printf("\n Enter the no of nodes required:");
scanf("%d",&no);
}while(no>10||no<0);
for(i=0;i<no;i++)
{
init(no,i);
inp(no,i);
}
printf("\nThe configuration of the nodes after initalization is as follows:");
for(i=0;i<no;i++)
op1(no,i,0);
for(j=0;j<no;j++)
{
for(i=0;i<no;i++)
```

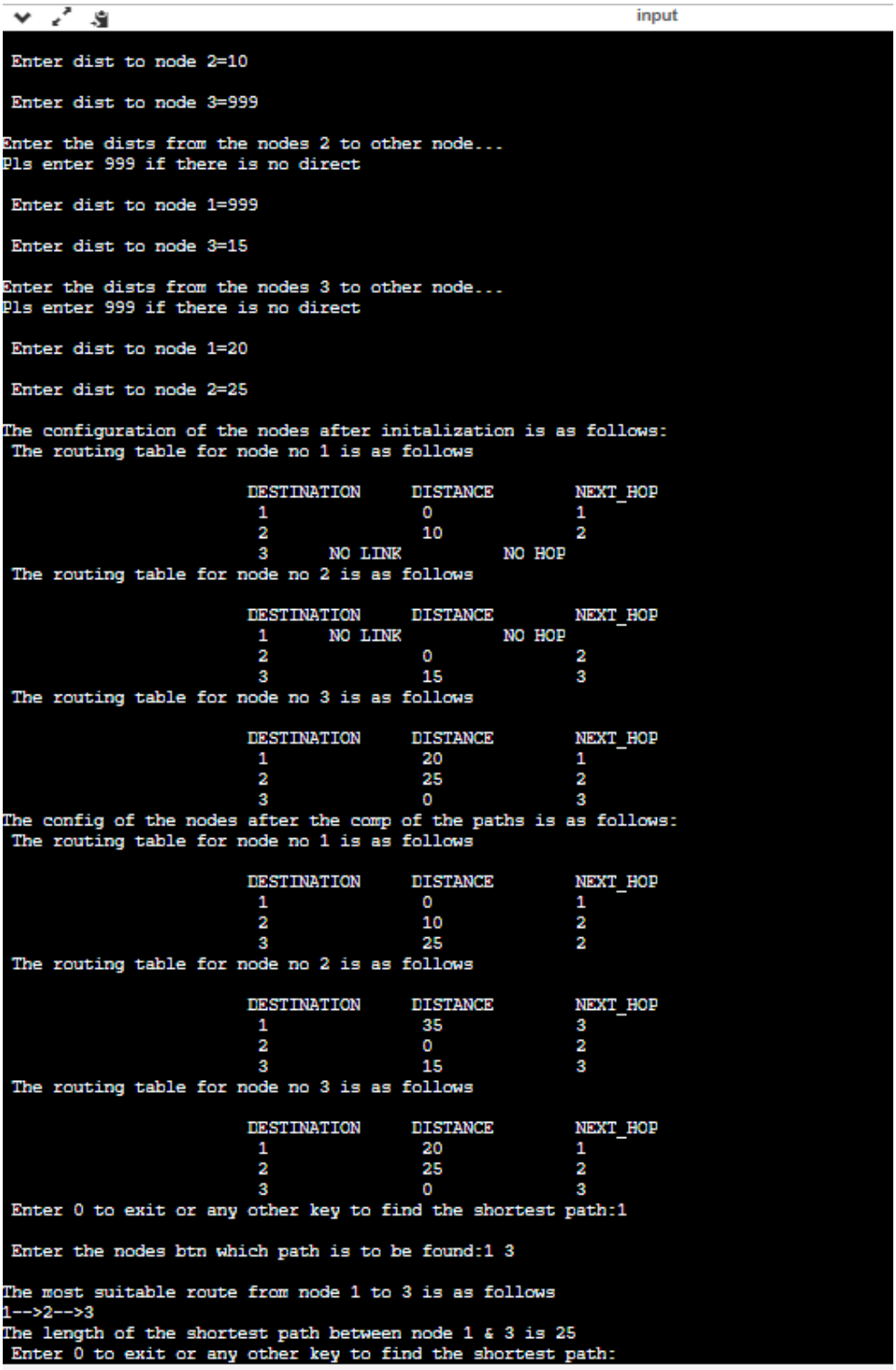
```
caller(no,i);
}
printf("\nThe config of the nodes after the comp of the paths is as
follows:");
for(i=0;i<no;i++)
op1(no,i,1);
while(1)
{
printf("\n Enter 0 to exit or any other key to find the shortest path:");
scanf("%d",&j);
if(!j)
break;
do{
printf("\n Enter the nodes btn which path is to be found:");
scanf("%d%d",&x,&y);
}while((x<0||x>no) && (y<0||y>no));
printf("\nThe most suitable route from node %d to %d is as follows\n",x,y);
find(x,y);
printf("%d",y);
printf("\nThe length of the shortest path between node %d & %d is
%d",x,y,n[x-1].t[y-1][2]);
}
}
void init(int no,int x)
{
int i;
for(i=0;i<no;i++)
{
n[x].t[i][1]=i;
n[x].t[i][2]=999;
n[x].t[i][3]=NUL;
}
n[x].t[x][2]=0;
n[x].t[x][3]=x;
}
```

```
void inp(int no,int x)
{
int i;
printf("\nEnter the dists from the nodes %d to other node...",x+1);
printf("\nPls enter 999 if there is no direct \n");
for(i=0;i<no;i++)
{
if(i!=x)
{
do
{
printf("\n Enter dist to node %d=",i+1);
scanf("%d",&n[x].t[i][2]);
}while(n[x].t[i][2]<0|| n[x].t[i][2]>999);
if(n[x].t[i][2]!=999)
n[x].t[i][3]=i;
}}
}
void caller(int no,int x)
{
void compar(int,int,int);
int i;
for(i=0;i<no;i++)
{
if(n[x].t[i][2]!=999 && n[x].t[i][2]!=0)
{
compar(x,i,no);
}
}
}
void compar(int x,int y,int no)
{
int i,z;
for(i=0;i<no;i++)
{
```

```
z=n[x].t[y][2]+n[y].t[i][2];
if(n[x].t[i][2]>z)
{
n[x].t[i][2]=z;
n[x].t[i][3]=y;
}
}
}
void op1(int no,int x,int z)
{
int i,j;
printf("\n The routing table for node no %d is as follows",x+1);
printf("\n\n\t\t\tDESTINATION\tDISTANCE\tNEXT_HOP");
for(i=0;i<no;i++)
{
if((!z && n[x].t[i][2]>=999) ||(n[x].t[i][2]>=(999*no)))
printf("\n\t\t\t %d \tNO LINK \t NO HOP",n[x].t[i][1]+1);
else
if(n[x].t[i][3]==NUL)
printf("\n\t\t\t %d \t\t %d \t\t NO HOP",n[x].t[i][1]+1,n[x].t[i][2]);
else
printf("\n\t\t\t %d \t\t %d \t\t%d",n[x].t[i][1]+1,n[x].t[i][2],n[x].t[i][3]+1);
}
}
void find(int x,int y)

{
int i,j;
i=x-1;
j=y-1;
printf("%d-->",x);
if(n[i].t[j][3]!=j)
{
find(n[i].t[j][3]+1,y);
return;
```

}
}
OUTPUT:


input
Enter dist to node 2=10
Enter dist to node 3=999
Enter the dists from the nodes 2 to other node...
Pls enter 999 if there is no direct
Enter dist to node 1=999
Enter dist to node 3=15
Enter the dists from the nodes 3 to other node...
Pls enter 999 if there is no direct
Enter dist to node 1=20
Enter dist to node 2=25
The configuration of the nodes after initalization is as follows:
The routing table for node no 1 is as follows
DESTINATION DISTANCE NEXT_HOP
1 0 1
2 10 2
3 NO LINK NO HOP
The routing table for node no 2 is as follows
DESTINATION DISTANCE NEXT_HOP
1 NO LINK NO HOP
2 0 2
3 15 3
The routing table for node no 3 is as follows
DESTINATION DISTANCE NEXT_HOP
1 20 1
2 25 2
3 0 3
The config of the nodes after the comp of the paths is as follows:
The routing table for node no 1 is as follows
DESTINATION DISTANCE NEXT_HOP
1 0 1
2 10 2
3 25 2
The routing table for node no 2 is as follows
DESTINATION DISTANCE NEXT_HOP
1 35 3
2 0 2
3 15 3
The routing table for node no 3 is as follows
DESTINATION DISTANCE NEXT_HOP
1 20 1
2 25 2
3 0 3
Enter 0 to exit or any other key to find the shortest path:1
Enter the nodes btn which path is to be found:1 3
The most suitable route from node 1 to 3 is as follows
1-->2-->3
The length of the shortest path between node 1 & 3 is 25
Enter 0 to exit or any other key to find the shortest path:

3. **Implement Djikstra's algorithm to compute the shortest path for a given topology.**

```
import java.util.*;
class Edge{
   int src, dest, w;
   public Edge(int src, int dest, int w){
      this.src = src;
      this.dest = dest;
      this.w = w;
   }
}

class Node {
   int vertex, w;
   public Node(int vertex, int w) {
      this.vertex = vertex;
      this.w = w;
   }
}


class Graph{
   List<List<Edge>> edgeList = null;
   Graph(List<Edge> edges, int N){
      edgeList = new ArrayList<>();
      for (int i = 0; i < N; i++) {
         edgeList.add(new ArrayList<>());
      }
      for (Edge edge: edges){
         edgeList.get(edge.src).add(edge);
      }
   }
}

class Main{
```

```
   private static void getPath(int[] prev, int i, List<Integer> route){
      if (i >= 0){
         getPath(prev, prev[i], route);
         route.add(i);
      }
   }

   public static void getShortestPath(Graph graph, int src, int N){
      PriorityQueue<Node> minHeap;
      minHeap = new PriorityQueue<>(Comparator.comparingInt(node ->
node.w));
      minHeap.add(new Node(src, 0));
      List<Integer> dist = new ArrayList<>(Collections.nCopies(N,
Integer.MAX_VALUE));
      dist.set(src, 0);
      boolean[] done = new boolean[N];
      done[src] = true;
      int[] prev = new int[N];
      prev[src] = -1;
      List<Integer> route = new ArrayList<>();
      while (!minHeap.isEmpty()){
         Node node = minHeap.poll();
         int u = node.vertex;
         for (Edge edge: graph.edgeList.get(u)){
            int v = edge.dest;
            int w = edge.w;
            if (!done[v] && (dist.get(u) + w) < dist.get(v)){
               dist.set(v, dist.get(u) + w);
               prev[v] = u;
               minHeap.add(new Node(v, dist.get(v)));
            }
         }
         done[u] = true;
      }
```

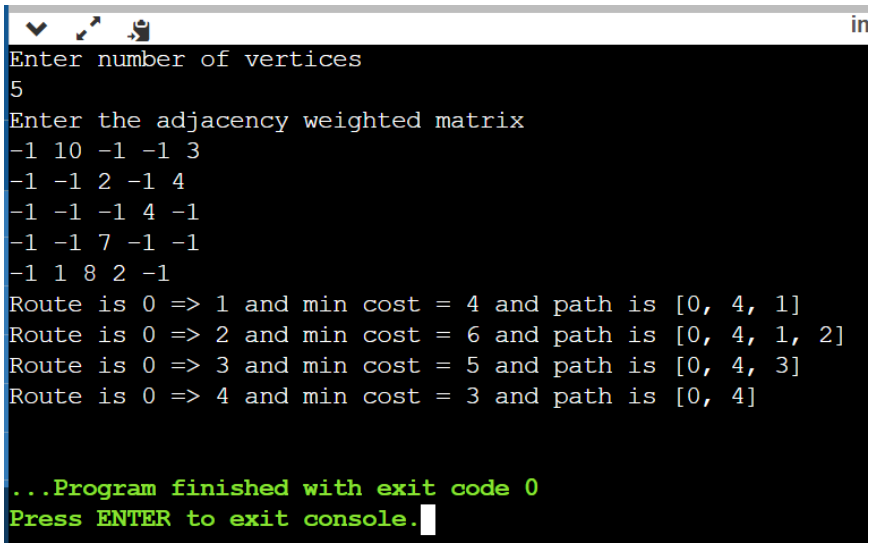
```
        for(int i = 1; i < N; ++i){
            if (i != src && dist.get(i) != Integer.MAX_VALUE) {
                getPath(prev, i, route);
                System.out.printf("Route is %d => %d and min cost = %d and path
is %s\n",
                        src, i, dist.get(i), route);
                route.clear();
            }
        }
    }

    public static void main(String[] args){
        Scanner s = new Scanner(System.in);
        List<Edge> edges = new ArrayList<>();
        System.out.println("Enter number of vertices");
        int n = s.nextInt();
        System.out.println("Enter the adjacency weighted matrix");
        int[][] mat = new int[n][n];
        for(int i=0; i<n; i++){
            for(int j=0; j<n; j++){
                mat[i][j] = s.nextInt();
            }
        }

        for(int i=0; i<n; i++){
            for(int j=0; j<n; j++){
                if(i == j) continue;
                if(mat[i][j] != -1){
                    edges.add(new Edge(i, j, mat[i][j]));
                }
            }
        }
        Graph graph = new Graph(edges, n);
        int src = 0;
        getShortestPath(graph, src, n);
```

```
        s.close();
      }
    }
```

OUTPUT:



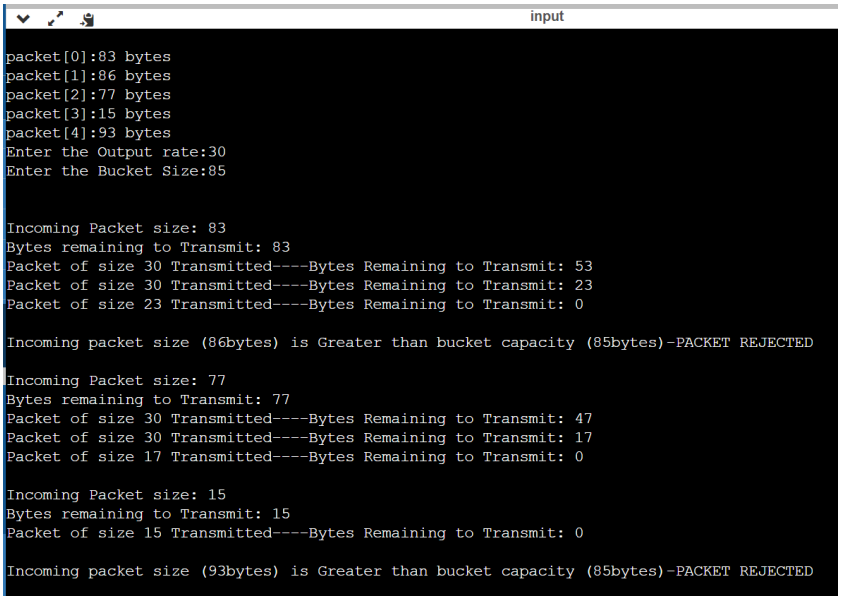
4. **Write a program for congestion control using Leaky Bucket algorithm.**

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#define NOF_PACKETS 5
/*
int rand (int a)
{
int rn = (random() % 10) % a;
return rn == 0 ? 1 : rn;
}
*/
/*
#include &lt;stdlib.h&gt;
long int random(void);
```

```
The random() function uses a nonlinear additive feedback random number
generator employing a default ta-
ble of size 31 long integers to return successive pseudo-random numbers in
the
range from 0 to RAND_MAX.
The period of this random number generator is very large, approximately
16 *
((2^31) - 1).
*/
int main()
{
int packet_sz[NOF_PACKETS], i, clk, b_size, o_rate, p_sz_rm=0, p_sz,
p_time, op;
for(i = 0; i<NOF_PACKETS; ++i)
packet_sz[i] = random() % 100;
for(i = 0; i<NOF_PACKETS; ++i)
printf("\npacket[%d]:%d bytes\t", i, packet_sz[i]);
printf("\nEnter the Output rate:");
scanf("%d", &o_rate);
printf("Enter the Bucket Size:");
scanf("%d", &b_size);
for(i = 0; i<NOF_PACKETS; ++i)
{
if( (packet_sz[i] + p_sz_rm) > b_size)
if(packet_sz[i] > b_size)/*compare the packet siz with bucket size*/
printf("\n\nIncoming packet size (%dbytes) is Greater than bucket capacity
(%dbytes)-PACKET REJECTED", packet_sz[i], b_size);
else
printf("\n\nBucket capacity exceeded-PACKETS REJECTED!!");
else
{
p_sz_rm += packet_sz[i];
printf("\n\nIncoming Packet size: %d", packet_sz[i]);
printf("\nBytes remaining to Transmit: %d", p_sz_rm);
//p_time = random() * 10;
```

```
//printf(&quot;\nTime left for transmission: %d units&quot;, p_time);
//for(clk = 10; clk &lt;= p_time; clk += 10)
while(p_sz_rm>0)
{
sleep(1);
if(p_sz_rm)
{
if(p_sz_rm <= o_rate)/*packet size remaining comparing with output rate*/
op = p_sz_rm, p_sz_rm = 0;
else
op = o_rate, p_sz_rm -= o_rate;
printf("\nPacket of size %d Transmitted", op);
printf("----Bytes Remaining to Transmit: %d", p_sz_rm);
}
else
{
printf("\nNo packets to transmit!!");
}
}
}
}
}
```

OUTPUT:



5. **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.**

ClientTCP.py

```
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()
```

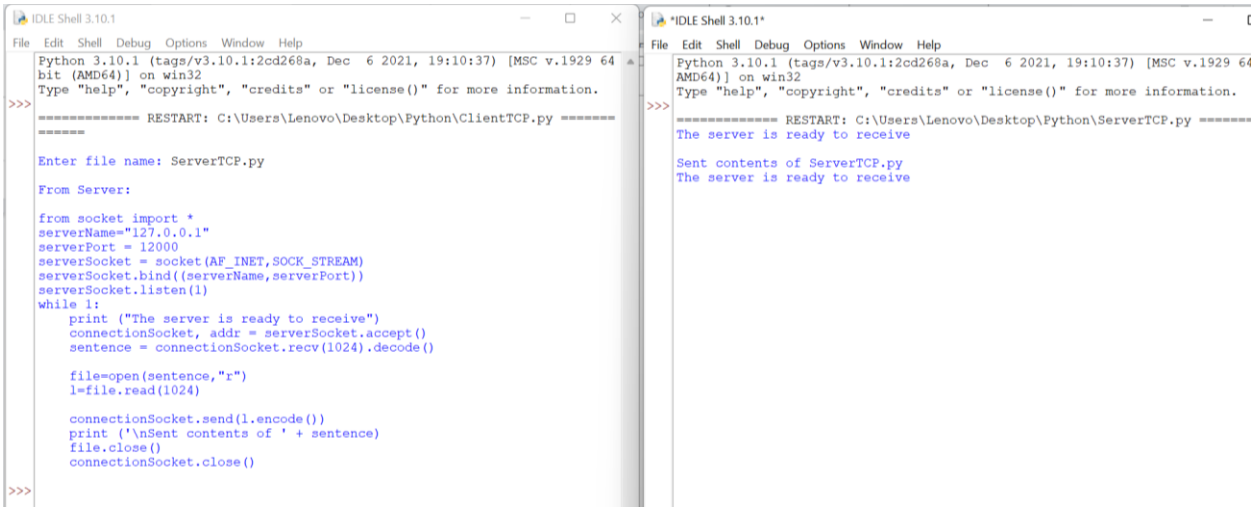
ServerTCP.py

```
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()
```

OUTPUT:



6. **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.**

ClientUDP.py

```
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()
```

ServerUDP.py

```
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()
```

OUTPUT: