EXPERIMENT No. 01: Program for Error Detecting Code using CRC - CCITT

1. 1Objective

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

1.1 Software Required

Java Programming Language and Linux

1.2 Pre-Requisite

Basic C Programming, Error detecting code

1.3 Introduction

Error detection is crucial in data communication to ensure the integrity of transmitted data. One widely used method for error detection is **Cyclic Redundancy Check (CRC)**, which helps detect accidental changes to raw data. Among various CRC algorithms, **CRC-CCITT (16-bit)** is commonly used in telecommunications, XMODEM, and other network protocols.

Understanding CRC -CCITT(16-bit)

- CRC (Cyclic Redundancy Check) is used for error detection in digital networks.
- The corresponding **CRC-CCITT 16-bit polynomial** is **0x1021** (Binary: 0001 0000 0010 0001).

Basic Steps in CRC Calculation

1. Append 16 zero bits (since CRC-CCITT is 16-bit).
2. Perform bitwise division using the polynomial (0x1021).
3. The remainder from the division is the CRC checksum.
4. The receiver verifies the CRC by re-computing it and checking if the remainder is zero.

1.4 Procedure

```
import java.util.*;
public class crc
{
            public static void main(String args[])
           {
                      Scanner s=new Scanner(System.in);
                      int n;
                      System.out.println("enter the size of the data:");
                      n=s.nextInt();
                      int data[]=new int[n];
                      System.out.println("enter the data ,bit by bit:");
                      for(int i=0; i < n; i++)
                      {
                            data[i]=s.nextInt();
                      }
                      System.out.println("enter ther size of the divisor:");
                      n=s.nextInt();
                      int divisor[]=new int[n];
                      System.out.println("enter the divisor,bit by bit:");
                      for(int i=0; i < n; i++)
                            divisor[i]=s.nextInt();
                      int remainder[]=divide(data,divisor);
                      System.out.println("\n the crc code generated is:");
                      for(int i=0; i < data.length; i++)
                            System.out.print(data[i]);
                      for(int i=0; i < remainder.length-1; i++)
                           System.out.print(remainder[i]);
                           System.out.println();
                           int sent_data[]=new int[data.length+remainder.length-1];
                           System.out.println("enter the data to be sent:");
                           for(int i=0; i < sent_data.length; i++)
                               sent_data[i]=s.nextInt();
                               recieve(sent_data,divisor);
             }
            static int[] divide(int old_data[],int divisor[])
            {
                        int remainder[],i;
                        int data[]=new int[old_data.length+divisor.length];
```

```
                System.arraycopy(old_data, 0,data, 0,old_data.length);
                System.out.println("message bits after appending divisor_length-1       0's:");
                for(i=0; i<data.length-1; i++)
                    System.out.println(data[i]);
                    remainder=new int[divisor.length];
                    System.arraycopy(data, 0, remainder, 0, divisor.length);
                   for(i=0; i < old_data.length; i++)
                   {
                       if(remainder[0]==1)
                       {
                              for(int j=1; j<divisor.length; j++)
                              {
                                   remainder[j-1]=exor(remainder[j],divisor[j]);
                              }
                       }
                       else
                       {
                              for(int j=1; j < divisor.length; j++)
                              remainder[j-1]=exor(remainder[j],0);
                       }
                       remainder[divisor.length-1]=data[i+divisor.length];
                }
                   return remainder;
      }
      static int exor(int a,int b)
      {
               if(a==b)
               return 0;
               return 1;
     }
     static void recieve(int data[],int divisor[])
     {
               int remainder[]=divide(data,divisor);
               for(int i=0; i < remainder.length; i++)
               {
                       if(remainder[i] != 0)
                         {
                              System.out.println("there is an error in recieved data");
                              return;
                         }
               }
```

```
                        System.out.println("data was recieved without any error");
              }
  }
```

1.5 Results

```
student@student-H61M-S2-B3:~/DCN_24-25$ gedit crc.java &
student@student-H61M-S2-B3:~/DCN_24-25$ javac crc.java
student@student-H61M-S2-B3:~ /DCN_24-25$ java crc
enter the size of the data:
7
enter the data ,bit by bit:
1 0 1 1 0 0 1
enter ther size of the divisor:
3
enter the divisor,bit by bit:
1
0
1
message bits after appending divisor_length-1   0's:
1 0 1 1 0 0 1 0 0

 the crc code generated is:
101100111
enter the data to be sent:
1
0
1
1
0
```

```
0
1
1
1
message bits after appending divisor_length-1   0's:
1 0 1 1 0 0 1 1 1 0 0
data was recieved without any error
```

1.6 Pre-Requisite Questions

1. What is error detection, and why is it important in data communication?
2. What are the common error detection techniques?
3. What is a Cyclic Redundancy Check (CRC)?
4. What are the common error detection techniques?

1.7 Post-Experimental Questions

1. What were the key steps followed in the CRC-CCITT error detection experiment?
2. How was the CRC remainder calculated, and how did it change with different input data?
3. How did the system handle errors in transmitted data?
4. What was the role of the generator polynomial in the CRC computation?

EXPERIMENT No. 02: Leaky Bucket Algorithm using JAVA Program.

2.1Objective

Write a program for congestion control using a leaky bucket algorithm using JAVA program.

2.2 Software Required

Java Programming Language and Linux

2.3 Pre-Requisite

Basics of Java Programming, Concept of Selecion Sort

2.4 Introduction

The **Leaky Bucket Algorithm** is a traffic shaping and rate-limiting algorithm used in computer networks to control data flow. It ensures that data is transmitted at a consistent rate, preventing congestion and packet loss. The algorithm works by maintaining a bucket (buffer) with a fixed capacity. When data packets arrive, they are added to the bucket. The bucket leaks (transmits) data at a constant rate, and if it overflows, excess packets are discarded.

Key Concepts of the Leaky Bucket Algorithm

1. **Bucket Capacity**: The buffer has a fixed size to hold incoming packets.
2. **Constant Output Rate**: Packets are transmitted (leaked) at a fixed rate.
3. **Overflow Handling**: If the bucket is full, incoming packets are dropped.
4. **Efficient Flow Control**: Prevents sudden bursts of data from overwhelming the network.

2.5 Procedure

```
import java.math.*;
import java.util.*;
import java.util.Random;
import java.io.*;
import java.lang.*;
```

```
public class leaky_bucket
{
        public static void main(String[]args)
        {
            int drop=0,mini,i,o_rate,b_size,nsec,p_remain=0;
            int packet[]=new int[100];
            Scanner in=new Scanner(System.in);
            System.out.print("Enter the bucket size:\n");
            b_size=in.nextInt();
            System.out.print("Enter output rate:\n");
             o_rate=in.nextInt();
            System.out.print("Enter the number of seconds to simulate:\n");
            nsec=in.nextInt();
            Random rand=new Random();
            for(i=0;i<nsec;i++)
            packet[i]=(rand.nextInt(1000));
            System.out.println("seconds|packet received|packet sent| packets left|packets
            dropped\n");
            System.out.println("                    ");
            for(i=0;i<nsec;i++)
            {
                p_remain+=packet[i];
                if(p_remain>b_size)
                {
                    drop=p_remain-b_size;
                    p_remain=b_size;
                    System.out.print(i+1          +"       ");
                    System.out.print(packet[i] +"                ");
                    mini=Math.min(p_remain,o_rate);
                    System.out.print(mini +"                    ");
                    p_remain=p_remain-mini;
```

```
                        System.out.print(p_remain+"          ");
                        System.out.println(drop +"          ");
                        System.out.print("               \n");
                        drop=0;
                  }
            }
            while(p_remain!=0)
            {
                  if(p_remain>b_size)
                  {
                        drop=p_remain-b_size;
                  }
                  mini=Math.min(p_remain,o_rate);
                  System.out.print("            "+p_remain +"           " +mini);
                  p_remain=p_remain-mini;
                  System.out.println("          "+p_remain +"          " +drop);
                  drop=0;
            }
      }
}
```

2.6. Results

```
student@student-H61M-S2-B3:~/DCN_24-25$ gedit leaky_bucket.java
student@student-H61M-S2-B3:~/DCN_24-25$ javac leaky_bucket.java
student@student-H61M-S2-B3:~/DCN_24-25$ java leaky_bucket
Enter the bucket size:
10
Enter output rate:
4
Enter the number of seconds to simulate:
10
```

```
seconds|packet received|packet sent| packets left|packets dropped

1       517             4           6           507
2       103             4           6           99
3       536             4           6           532
4       139             4           6           135
5       976             4           6           972
6       811             4           6           807
7       921             4           6           917
8       827             4           6           823
9       273             4           6           269
10      625             4           6           621


            6     4       2   0
            2     2       0   0
```

2.7 Pre-Requisite Questions

1. What is the Leaky Bucket Algorithm, and how does it work?
2. Why is the Leaky Bucket Algorithm used in networking?
3. How does the Leaky Bucket Algorithm help in traffic shaping?
4. What happens if the bucket overflows in the Leaky Bucket Algorithm?

2.8 Post- Experimental Questions

1. How did the algorithm manage varying packet arrival rates?
2. What happened when the packet arrival rate exceeded the bucket capacity?
3. How did the output (leak) rate affect the overall traffic flow?
4. Were there scenarios where packets were dropped? What conditions led to packet loss?

EXPERIMENT No.03: JAVA using TCP - Iterative Client-Server Program

3.1 Objectives:

Write a socket programming in JAVA using TCP -iterative client-server program.

3.2 Software Required:

JAva Programming Language and Linux

3.3 Pre-Requisite:

3.4 Introduction:

Socket programming in Java allows communication between two applications over a network using the **Transmission Control Protocol (TCP)** or **User Datagram Protocol (UDP)**. TCP provides reliable, connection-oriented communication, making it suitable for applications where data integrity is crucial, such as file transfer and messaging applications.

In the **iterative client-server model**, the server handles one client at a time. When a client connects, the server processes the request, sends a response, and then closes the connection before accepting another client. This model is simple but may not scale well for handling multiple clients concurrently.

How TCP Socket Programming Works

1. **Server Process**
    - Listens for incoming client requests on a specific port.
    - Accepts a connection request from a client.
    - Reads data from the client, processes it, and sends a response.
    - Closes the connection before handling another client.
2. **Client Process**
    - Connects to the server on a specified port.
    - Sends a request (e.g., a message).
    - Receives and processes the server's response.
    - Closes the connection.

3.5 Procedure:

TCP Server program:

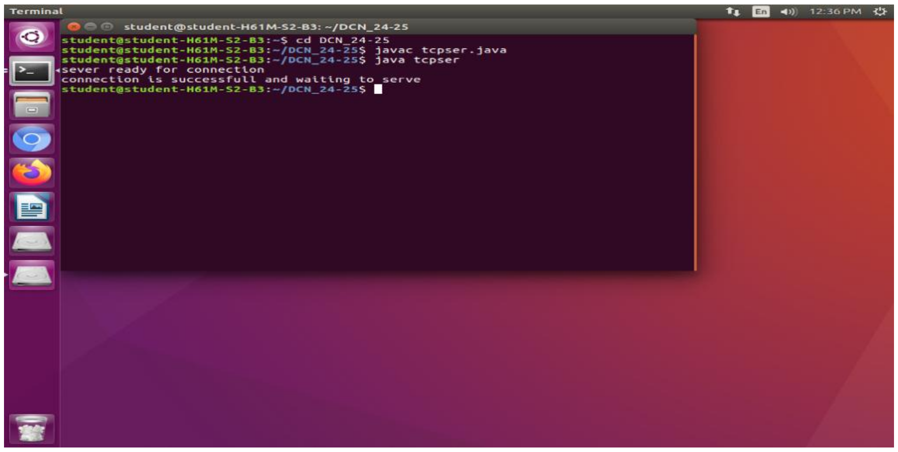
```
import java.net.*;
import java.io.*;
public class tcpser
{
        public static void main(String args[])throws Exception
        {
              ServerSocket sersock= new ServerSocket(4000);
              System.out.println("sever ready for connection");
              Socket sock=sersock.accept();
              System.out.println("connection is successfull and waiting to serve");
              InputStream istream=sock.getInputStream();
              BufferedReader fileRead=new BufferedReader(new InputStreamReader(istream));
              String fname=fileRead.readLine();
              BufferedReader contentRead=new BufferedReader(new FileReader(fname));
              OutputStream ostream=sock.getOutputStream();
```

```
                PrintWriter pwrite=new PrintWriter(ostream,true);
                String str;
                while((str=contentRead.readLine())!=null)
                {
                        pwrite.println(str);


                }

        }
}
```

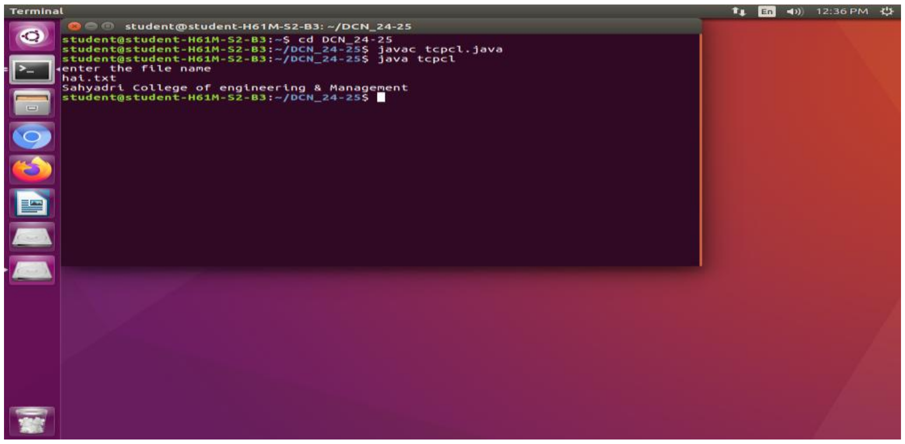
TCP Client program:

```
import java.net.*;
import java.io.*;

public class tcpcl
{
        public static void main(String args[])throws Exception
        {
                Socket sock=new Socket("127.0.0.1",4000);
                System.out.println("enter the file name");
                BufferedReader keyRead=new BufferedReader(new InputStreamReader(System.in));
                String fname=keyRead.readLine();
                OutputStream ostream=sock.getOutputStream();
                PrintWriter pwrite=new PrintWriter(ostream,true);
                pwrite.println(fname);
                InputStream istream=sock.getInputStream();
                BufferedReader SocketRead=new BufferedReader(new InputStreamReader(istream));
                String str;
                while((str=SocketRead.readLine())!=null)
                {
                        System.out.println(str);
                }
                pwrite.close();
                SocketRead.close();
                keyRead.close();
        }
}
```

3.6 Results:



```
student@student-H61M-S2-B3:~$ cd DCN_24-25
student@student-H61M-S2-B3:~/DCN_24-25$ javac tcpser.java
student@student-H61M-S2-B3:~/DCN_24-25$ java tcpser
sever ready for connection
connection is successfull and waiting to serve
student@student-H61M-S2-B3:~/DCN_24-25$
```



```
student@student-H61M-S2-B3:~$ cd DCN_24-25
student@student-H61M-S2-B3:~/DCN_24-25$ javac tcpcl.java
student@student-H61M-S2-B3:~/DCN_24-25$ java tcpcl
enter the file name
hai.txt
Sahyadri College of engineering & Management
student@student-H61M-S2-B3:~/DCN_24-25$
```

3.7 Post-Requisite Questions

1. What are the key steps in setting up a TCP server using sockets?
2. How does a server handle multiple clients in a TCP connection?
3. What steps are involved in setting up a TCP client connection?
4. What are the key differences between a TCP client and a TCP server?

3.8 Pre-Experimental Questions

1. What is TCP, and how does it differ from UDP?

2. Why is TCP considered a reliable protocol?
3. What are the key components required to establish a TCP connection?
4. What are the main steps required to create a TCP server?

EXPERIMENT No.04: JAVA using UDP - Iterative Client-Server Program

4.1 Objective

Write a socket programming in JAVA using UDP -iterative client-server program.

4.2 Software Required

4.3 Pre-Requisite

4.4 Introduction

User Datagram Protocol (UDP) is a connectionless communication protocol used for fast, lightweight network communication. Unlike **TCP (Transmission Control Protocol)**, UDP does not establish a connection before transmitting data, making it faster but less reliable. It is commonly used in applications like **video streaming, VoIP, online gaming, and DNS resolution**, where speed is more critical than guaranteed delivery.

How UDP Works in Socket Programming

- **No Connection Establishment**: The client sends data (datagram packets) to the server without setting up a connection.
- **Unreliable Data Transfer**: There is no acknowledgment mechanism, meaning data may be lost or received out of order.
- **Faster and Lightweight**: Since there is no connection setup overhead, it is more efficient for real-time applications.

4.5 Procedure

UDP Server Program

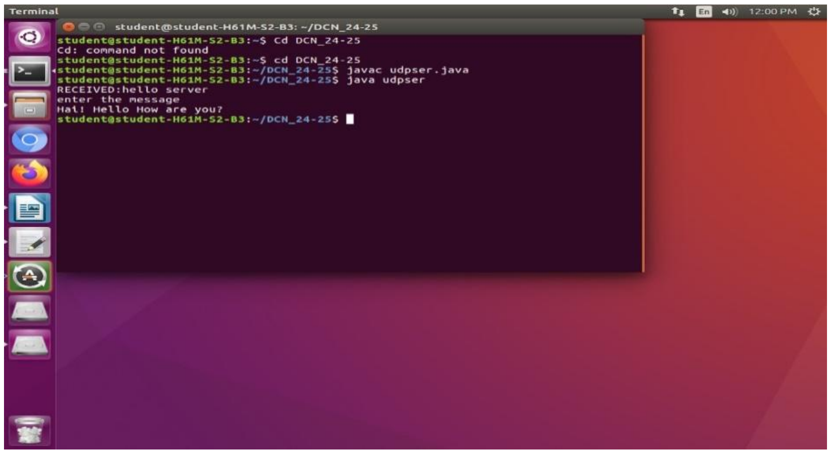
```
import java.io.*;
import java.net.*;
class udpser
{
  public static void main(String args[]) throws Exception
  {
        DatagramSocket serversocket=new DatagramSocket(9876);
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        byte[] receiveData=new byte[1024];
        byte[] sendData=new byte[1024];
        DatagramPacket receivePacket=new DatagramPacket (receiveData,receiveData.length);
        serversocket.receive(receivePacket);
        String sentence=new String(receivePacket.getData());
        System.out.println("RECEIVED:"+sentence);
        InetAddress IPaddress=receivePacket.getAddress();
        int port=receivePacket.getPort();
        System.out.println("enter the message");
        String data=br.readLine();
        sendData=data.getBytes();
        DatagramPacket sendPacket=new  DatagramPacket(sendData,sendData.length,IPaddress,port);
```

```
            serversocket.send(sendPacket);
            serversocket.close();
        }
    }
```

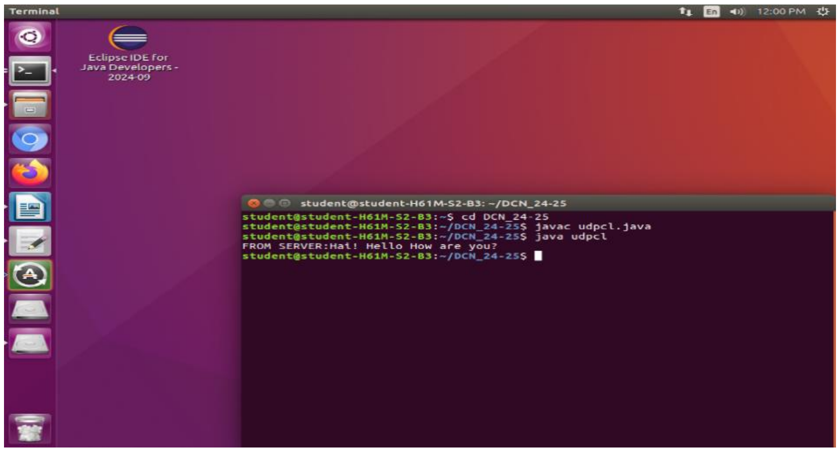
UDP Client Program

```
  import java.io.*;
  import java.net.*;
  class udpcl
  {
            public static void main(String[]args)throws Exception
            {
                  BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
                  DatagramSocket clientsocket=new DatagramSocket();
                  InetAddress IPaddress=InetAddress.getByName("localhost");
                  byte[] sendData=new byte[1024];
                  byte[] receiveData=new byte[1024];
                  String sentence="hello server";
                  sendData=sentence.getBytes();
                  DatagramPacket sendPacket=new DatagramPacket (
                  sendData,sendData.length,IPaddress,9876);
                  clientsocket.send(sendPacket);
                  DatagramPacket receivePacket=new DatagramPacket(receiveData,receiveData.length);
                  clientsocket.receive(receivePacket);
                  String modifiedsentence=new String(receivePacket.getData());
                  System.out.println("FROM SERVER:" +modifiedsentence);
                  clientsocket.close();
            }
  }
```

4.6 Results



```
student@student-H61M-S2-B3:~$ Cd DCN_24-25
Cd: command not found
student@student-H61M-S2-B3:~$ cd DCN_24-25
student@student-H61M-S2-B3:~/DCN_24-25$ javac udpser.java
student@student-H61M-S2-B3:~/DCN_24-25$ java udpser
RECEIVED:hello server
enter the message
Hai! Hello How are you?
student@student-H61M-S2-B3:~/DCN_24-25$
```



```
student@student-H61M-S2-B3:~$ cd DCN_24-25
student@student-H61M-S2-B3:~/DCN_24-25$ javac udpcl.java
student@student-H61M-S2-B3:~/DCN_24-25$ java udpcl
FROM SERVER:Hai! Hello How are you?
student@student-H61M-S2-B3:~/DCN_24-25$
```

4.7 Pre-Requisite Questions

1. What are the key steps in setting up a UDP server?
2. How does a UDP server receive data, and how does it respond to a client?
3. How does a UDP client send data to a server?
4. What function is used to receive a response from a UDP server?

4.8 Post-Experimental Questions

1. What is UDP, and how does it differ from TCP?
2. Why is UDP considered a connectionless protocol?
3. What are the advantages and disadvantages of using UDP over TCP?
4. What are the main steps required to create a UDP server