A

Project ReportOn

“Client Server based conversion (prefix infix postfix) application using UDP”

Department of Computer Science & Engineering

NATIONAL INSTITUE OF TECHNOLOGY PATNA

University Campus, Bihar - 800005

Submitted By:

Name	Roll No.
SUDDAPU SANDEEP	2006092
YALLAMPATI HEMAVARDHINI	2006074
PARA VENKATA AKSHITH	2006108

Course Code: CSL5403

Course Title: Computer Networks

Branch: CSE-A2

TABLE OF CONTENTS

LIST OF FIGURES

AIM OF PROJECT

Design and implement a Client Server based prefix to postfix, postfix to prefix, prefix to infix, infix to prefix, postfix to infix and infix to postfix conversion application using UDP.

1.INTRODUCTION

1.1 Socket programming:

- Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection.
- Sockets allow communication between two different processes on the same or different machines.
- **Where is Socket used**:

  A Unix Socket is used in a client server application framework. A server is a process which does some function on request from a client. Most of the application level protocols like FTP, SMTP and POP3 make use of Sockets to establish connection between client and server and then for exchanging data.
- There are four types of sockets available to the users. The first two are most commonly used and last two are rarely used.
  - Stream sockets
  - Datagram sockets
  - Raw socket
  - Sequenced packet sockets

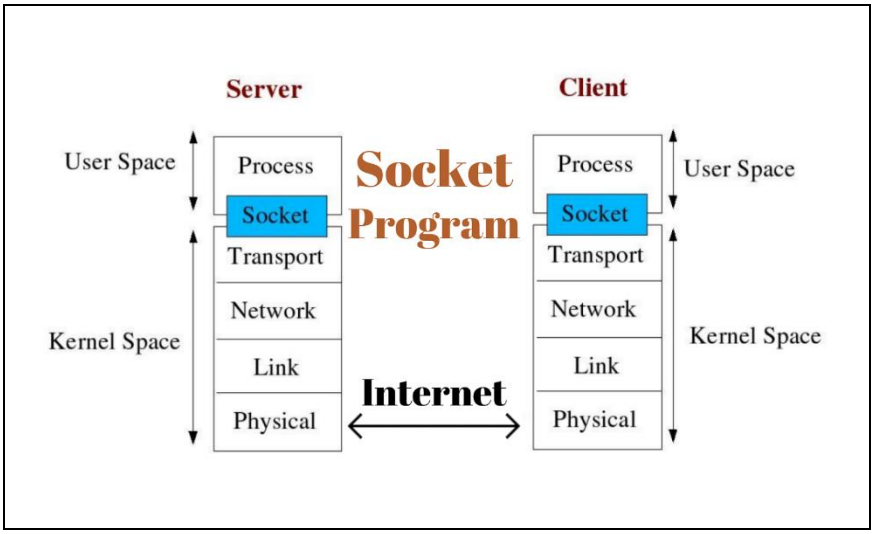


Fig 1.1: Socket program

- Sockets are mostly used in client-server architecture for communication between multiple applications.
- Socket programming tells us how we can use socket API for creating communication between local and remote processes.
- The socket is created by the combination of the IP address and port number of the software. With this combination, the process knows the system address and address of the application where data is to be sent.

1.2 Client:

- A client is a program that runs on the local machine requesting service from the server. A client program is a finite program means that the service started by the user and terminates when the service is completed.
- A client is a machine or program sending requests to another client or a server in order to take action
- A server is an entry points for multiple clients that will handle their requests

- **How to make client:** The system calls for establishing a connection are somewhat different for the client and the server, but both involve the basic construct of a socket. The two processes each establish their own sockets. The steps involved in establishing a socket on the client side are as follows:

    - Create a socket with the socket() system call.
    - Connect the socket to the address of the server using the connect() system call.
    - Send and receive data. There are a number of ways to do this, but the simplest is to use the read() and write() system calls.

1.3 Server:

- A server is a program that runs on the remote machine providing services to the clients. When the client requests for a service, then the server opens the door for the incoming requests, but it never initiates the service.
- A server program is an infinite program means that when it starts, it runs infinitely unless the problem arises. The server waits for the incoming requests from the clients. When the request arrives at the server, then it responds to the request.

Types of Server: There are two types of servers

1. **Iterative Server**: This is the simplest form of server where a server process serves one client and after completing first request then it takes request from another client. Meanwhile another client keeps waiting.

2. **Concurrent Servers**: This type of server runs multiple concurrent processes to serve many requests at a time. Because one process may take longer and another client cannot wait for so long. The simplest way to write a concurrent server under Unix is to fork a child process to handle each client separately.

- **How to make a server:** The steps involved in establishing a socket on the server side are as follows:

    - Create a socket with the socket() system call.
    - Bind the socket to an address using the bind() system call. For a server socket on the Internet, an address consists of a port number on the host machine.
    - Listen for connections with the listen() system call.
    - 4. Accept a connection with the accept() system call. This call typically blocks until a client connects with the server.
    - 5. Send and receive data using the read() and write() system calls

1.4 Client Server Model:

- A client and server networking model is a model in which computers such as servers provide the network services to the other computers such as clients to perform a user based tasks. This model is known as client-server networking model.
- An application program is known as a client program, running on the local machine that requests for a service from an application program known as a server program, running on the remote machine.

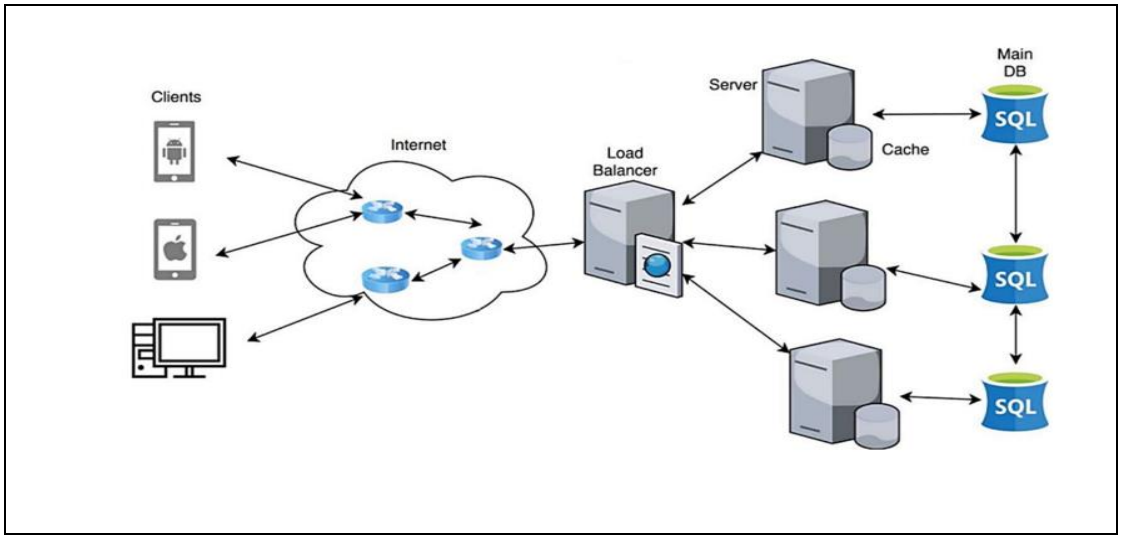


Fig 1.2: Client Server Model

- A client program runs only when it requests for a service from the server while the server program runs all time as it does not know when its service is required.
- A server provides a service for many clients not just for a single client. Therefore, we can say that client-server follows the many-to-one relationship. Many clients can use the service of one server.
- Examples of Client-Server Model are Email, World Wide Web, etc.

1.5 Advantages of Client-server networks:

- **Centralized:** Centralized back-up is possible in client-server networks, stored in server
- **Security:** These networks are more secure as all the shared resources are centrally administered.
- **Performance:** The use of the dedicated server increases the speed of sharing resources. This increases the performance of the overall system
- **Scalability:** We can increase the number of clients and servers separately, i.e., the new element can be added, or we can add a new node in a network at any time.

1.6 Disadvantages of Client-Server network:

- **Traffic Congestion** is a big problem in Client/Server networks. When a large number of clients send requests to the same server may cause the problem of Traffic congestion.
- It does not have a robustness of a network, i.e., when the server is down, then the client requests cannot be met.

1.7 USER DATAGRAM PROTCOL (UDP):

- **User Datagram Protocol (UDP)** is a Transport Layer protocol. UDP is a part of the Internet Protocol suite, referred to as UDP/IP suite. Unlike TCP, it is an **unreliable and connectionless protocol.** So, there is no need to establish a connection prior to data transfer. The UDP helps to establish low-latency and loss-tolerating connections establish over the network. The UDP enables process to process communication.
- The User Datagram Protocol (UDP) is a lightweight transport-layer protocol that works on top of IP. UDP provides a mechanism to detect corrupt data in packets, but it does not attempt to solve other problems that arise with packets, such as lost or out of order packets. That's why UDP is sometimes known as the Unreliable Data Protocol.
- UDP is a connectionless, unreliable, datagram protocol, quite unlike the connection-oriented, reliable byte stream provided by TCP. Nevertheless, there are instances when it makes sense to use UDP instead of TCP. Some popular applications are built using UDP: DNS, NFS, and SNMP.
- Here, UDP comes into the picture. For real-time services like computer gaming, voice or video communication, live conferences; we need UDP. Since high performance is needed, UDP permits packets to be dropped instead of processing delayed packets.

- Features:
  - UDP is used when acknowledgement of data does not hold any significance.
  - UDP is good protocol for data flowing in one direction.
  - UDP is simple and suitable for Query based communications.
  - UDP is not connection oriented.
  - UDP does not provide congestion control mechanism.
  - UDP does not guarantee ordered delivery of data.
  - UDP is stateless.
  - UDP is suitable protocol for streaming applications such as VoIP, multimedia streaming.

UDP Header –

- UDP header is an **8-bytes** fixed and simple header, while for TCP it may vary from 20 bytes to 60 bytes. The first 8 Bytes contains all necessary header information and the remaining part consist of data.
- UDP port number fields are each 16 bits long, therefore the range for port numbers is defined from 0 to 65535; port number 0 is reserved. Port numbers help to distinguish different user requests or processes.

The UDP header contains four fields:

- **Source port number:** It is 16-bit information that identifies which port is going t send the packet.
- **Destination port number:** It identifies which port is going to accept the information. It is 16-bit information which is used to identify application-level service

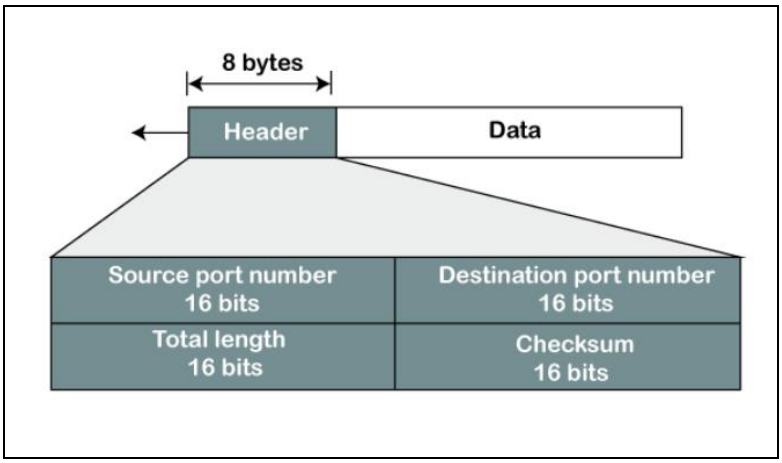


Fig 1.3: UDP Header Format

- **Length:** It is 16-bit field that specifies the entire length of the UDP packet that includes the header also. The minimum value would be 8-byte as the size of the header is 8 bytes.
- **Checksum:** It is a 16-bits field, and it is an optional field. This checksum field checks whether the information is accurate or not as there is the possibility that the information can be corrupted while transmission. In UDP, the checksum field is applied to the entire packet.

1.8 UDP SERVER & CLIENT:

- In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of the sender which the server uses to send data to the correct client.
- A UDP server is always listening. A UDP client is only listening after sending a message, for a response

UDP Server:

1. Create a UDP socket.
2. Bind the socket to the server address.
3. Wait until the datagram packet arrives from the client.
4. Process the datagram packet and send a reply to the client.
5. Go back to step 2.

UDP Client:

1. Create a UDP socket.
2. Send a message to the server.
3. Wait until response from the server is received.
4. Process reply and go back to step 2, if necessary.
5. Close socket descriptor and exit

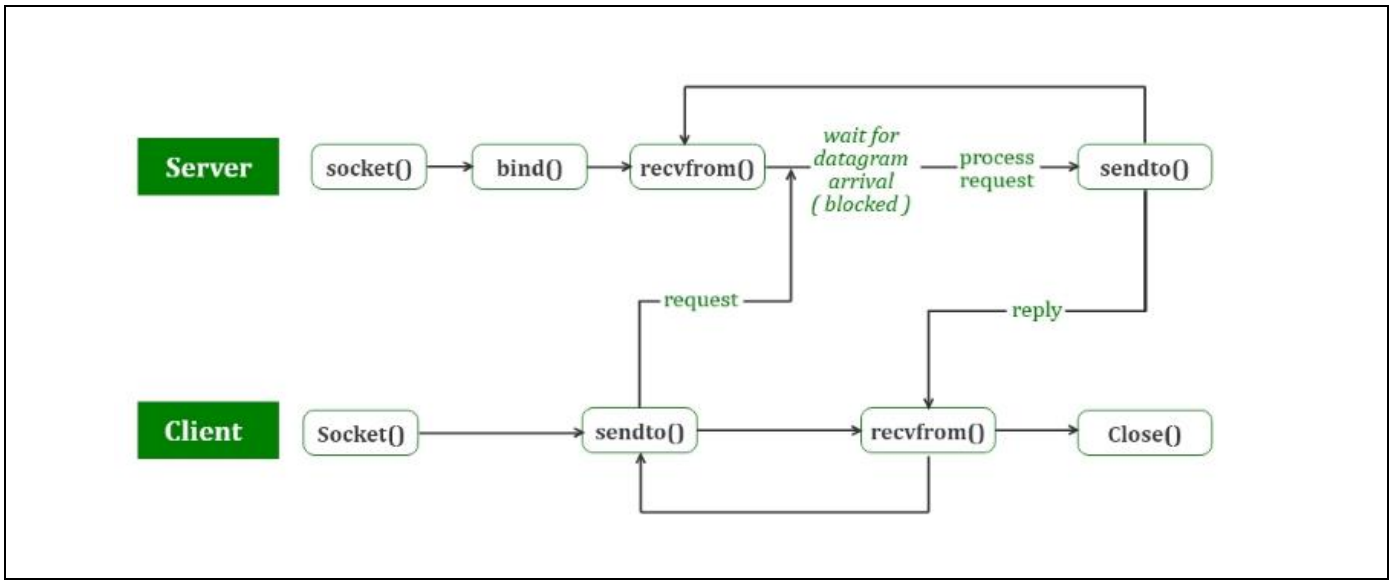


Fig 1.4: UDP CLIENT SERVER

1.9 Advantages of UDP:

UDP has a number of benefits for different types of applications, including:

- **No retransmission delays** – UDP is suitable for time-sensitive applications that can't afford retransmission delays for dropped packets. Examples include Voice over IP (VoIP), online games, and media streaming.
- **Speed** – UDP's speed makes it useful for query-response protocols such as DNS, in which data packets are small and transactional.
- **Suitable for broadcasts** – UDP's lack of end-to-end communication makes it suitable for broadcasts, in which transmitted data packets are addressed as receivable by all devices on the internet. UDP broadcasts can be received by large numbers of clients without server-side overhead.

At the same time, UDP's lack of connection requirements and data verification can create a number of issues when transmitting packets. These include:

- No guaranteed ordering of packets.
- No verification of the readiness of the computer receiving the message.
- No protection against duplicate packets.
- No guarantee the destination will receive all transmitted bytes. UDP, however, does provide a checksum to verify individual packet integrity.

1.10 DISADVANTAGES OF UDP:

- UDP is **unreliable** protocol.
- UDP protocol does not provide **congestion control service**.
- It does not guarantee the order of data received as there is no concept of windowing in UDP.
- Flow control is also not provided by UDP protocol.
- There is **no acknowledgment** mechanism in UDP, so the receiver will not acknowledge the sender for the received packet.

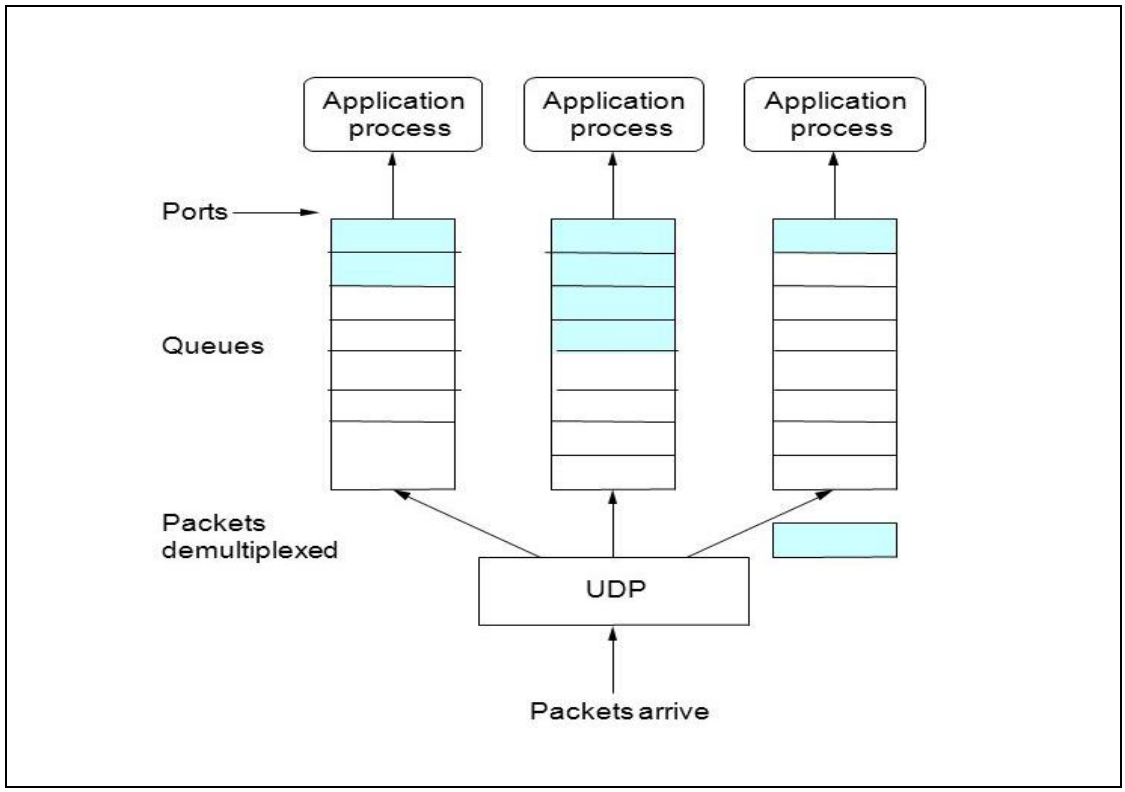


Fig 1.5: UDP Message Queue

1.11 UDP SOCKETS:

- Implementation of conversions on postfix prefix infix using UDP by creating datagram packets in a socket.
- UDP socket routines enable simple IP communication using the user datagram protocol (UDP).
- The User Datagram Protocol (UDP) runs on top of the Internet Protocol (IP) and was developed for applications that do not require reliability, acknowledgment, or flow control features at the transport layer. This **simple protocol** provides transport layer addressing in the form of UDP ports and an optional checksum capability.

1.12 Java Socket Programming:

- Java Socket programming is used for communication between the applications running on different JRE. Java Socket programming can be connection-oriented or connection-less.
- Socket and Server Socket classes are used for connection-oriented socket programming and Datagram Socket and Datagram Packet classes are used for connection-less socket programming.
- The client in socket programming must know two information:

Creating Server:

```
1. Server Socket ss=new Server Socket (6666);
2. Socket s=ss. Accept () ;//establishes connection and waits for the client
```

Creating Client:

```
1. Socket s=new Socket("localhost",6666);
```

Java Datagram Packet Class:

- **Java Datagram Packet** is a message that can be sent or received. It is a data container. If you send multiple Packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Here, commonly used Constructors of Datagram Packet class

- **Datagram Packet (byte [] Barr, int length):** it creates a datagram packet. This constructor is used to receive the packets.
- **Datagram Packet (byte [] Barr, int length, Inet Address address, int port):** it creates a datagram packet. This constructor is used to send the packets

1.13 INFIX PREFIX POSTFIX

- **INFIX NOTATION:** When the operator is written in between the operands, then it is known as **infix notation**. Operand does not have to be always a constant or a variable; it can also be an expression itself.

  Syntax of infix notation is given below:

  <operand> <operator> <operand>

- **PREFIX NOTATION:** A prefix notation is another form of expression but it does not require other information such as precedence and associativity, whereas an infix notation requires information of precedence and associativity. It is also known as **polish notation**. In prefix notation, an operator comes before the operands.

  The syntax of prefix notation is given below:

  <operator> <operand> <operand>

- **POSTFIX NOTATION:** If we move the operators after the operands then it is known as a postfix expression. In other words, postfix expression can be defined as an expression in which all the operators are present after the operands.

  The syntax of postfix notation is given below:

  <operand> <operand> <operator>

Examples:

Infix	Prefix	Postfix	Notes
A * B + C / D	+ * A B / C D	A B * C D / +	multiply A and B, divide C by D, add the results
A * (B + C) / D	/ * A + B C D	A B C + * D /	add B and C, multiply by A, divide by D
A * (B + C / D)	* A + B / C D	A B C D / + *	divide C by D, add B, multiply by A

2. SYSTEM REQUIREMENTS

Hardware Requirements:

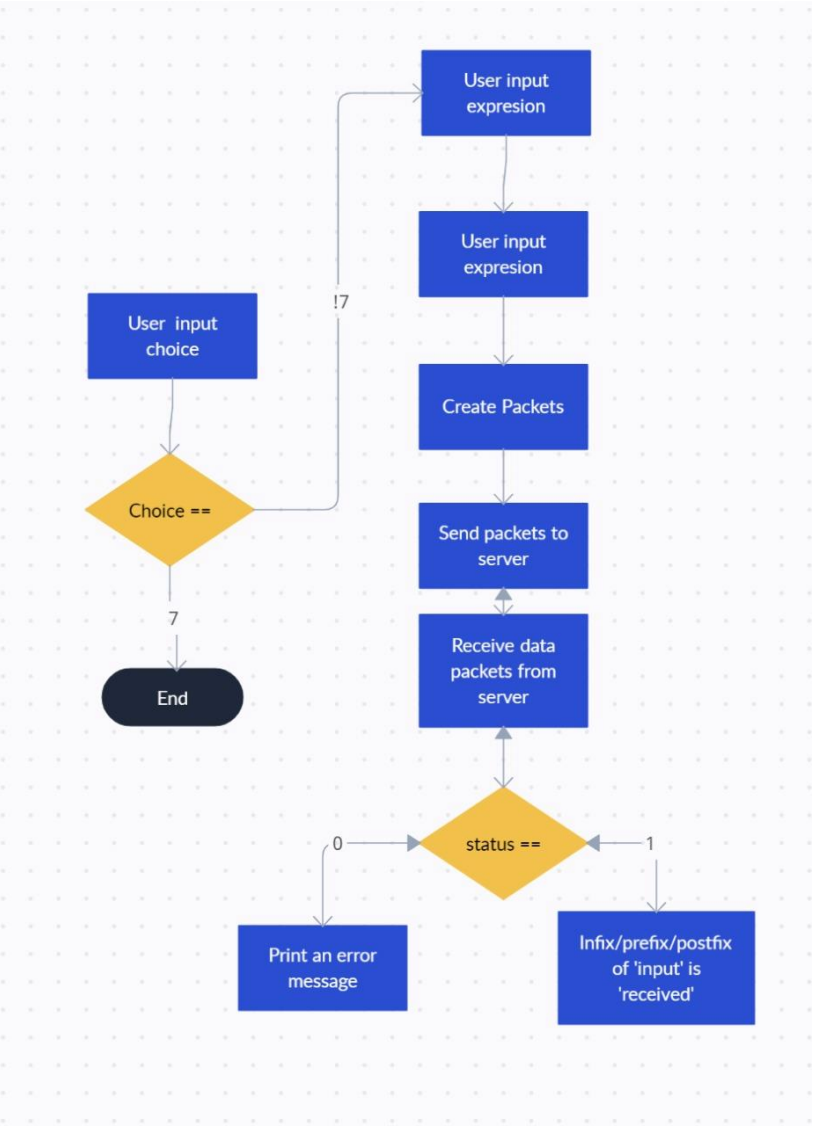
- 2 GHz x 86 processor
- 256 MB of system memory (RAM)
- 100 MB of hard drive space
- Keyboard/Mouse for data input

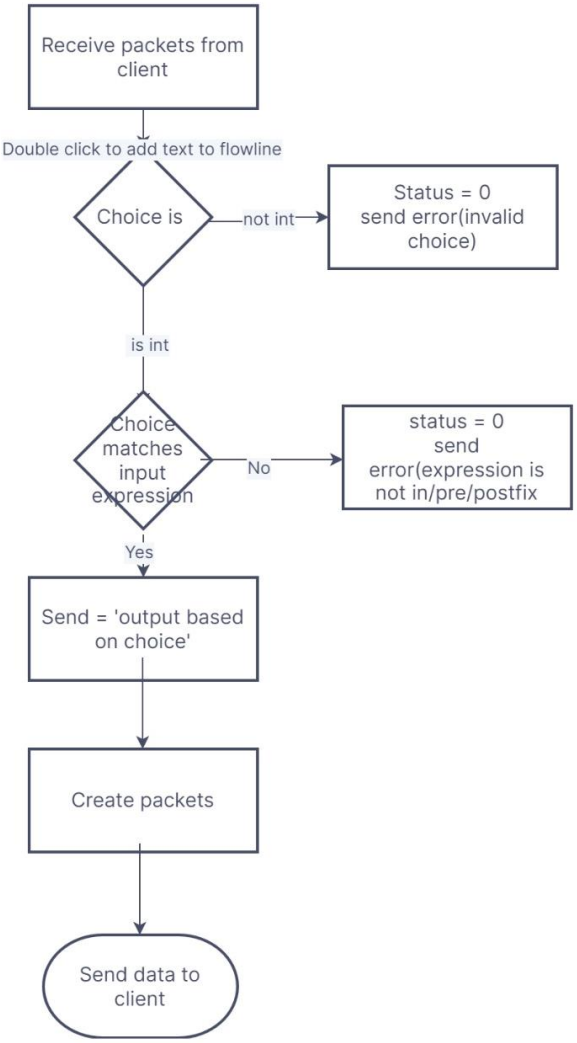
Software Requirements:

- Java compiler Command prompt
- MS Word (Documentation)

3.FLOW CHART/DATA FLOW DIAGRAM

Client **Server**





4.CODE

4.1 CODE TO IMPLEMENT UDP CLIENT USING JAVA:

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


public class UDPClient
{
   public static void main(String args[]) throws IOException
   {
      Scanner scan = new Scanner(System.in);
      DatagramSocket soc = new DatagramSocket();                //Creating sockets
      DatagramSocket soc2 = new DatagramSocket(5000);
      InetAddress ip = InetAddress.getLocalHost();

      while(true){
      System.out.println("\nChoices : \n\n1 - Infix to Prefix\n2 - Infix to Postfix\n3 - Prefix to
Infix\n4 - Prefix to Postfix\n5 - Postfix to Infix\n6 - Postfix to Prefix\n7 - Exit");

      System.out.print("\nEnter choice: ");

      String choice=scan.nextLine();
      if(choice.equals('7')) java.lang.System.exit(0);
      else{
         System.out.print("Enter the expression: ");
         String str = scan.nextLine();
         byte[] data=str.getBytes();          //Creating byte arrays of input strings
         byte[] ch = choice.getBytes();

         DatagramPacket DPch = new DatagramPacket(ch,ch.length,ip,5001);
         soc.send(DPch);
         DatagramPacket DPsend = new DatagramPacket(data,data.length,ip,5001);

         soc.send(DPsend);                    //Sending the datagram packets
         System.out.println("\nSent to server");


         DatagramPacket DPrec = null;
         DatagramPacket DPst = null;
         byte[] receive=new byte[1000];     //Packets to receive data from server
         byte[] st = new byte[16];
```

```
        System.out.println("\nWaiting for server....");

        DPrec = new DatagramPacket(receive,receive.length);
        DPst = new DatagramPacket(st,st.length);
        soc2.receive(DPrec);
        soc2.receive(DPst);

       //Receiving packets from the server
        System.out.println("Received");

        String out = new String(DPrec.getData(), 0, DPrec.getLength());
        String sta = new String(DPst.getData(),0,DPst.getLength());
        if(sta.equals(""+'1')){
           switch(choice){
           case ""+'3' :
           case ""+'5' : System.out.print("Infix of "+str+" is "+out);break;
           case "" +'1':
           case ""+'6' : System.out.print("Prefix of "+str+" is "+out);break;
           case "" +'2':
           case ""+'4' : System.out.print("Postfix of "+str+" is "+out);      //Printing the outputs
           }
        }
        else System.out.println(out);

        System.out.println("\nEnter '0' if you want to exit!");

        String a = scan.nextLine();
        if(a.equals("0"))java.lang.System.exit(0);
        else continue;
        }
      }
   }
}
```

4.2 CODE TO IMPLEMENT UDP SERVER USING JAVA:

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

public class UDP_S {
    public static void main(String args[]) throws IOException {
        DatagramSocket soc = new DatagramSocket(5001);
        byte[] rec = new byte[1000];
        byte[] ch = new byte[16];
        String st = "" + '1';

        while (true) {
            DatagramPacket DPrec = null;
            DatagramPacket DPch = null;
            DPch = new DatagramPacket(ch, ch.length);
            System.out.println("\n\nWaiting for client...");
            soc.receive(DPch);
            DPrec = new DatagramPacket(rec, rec.length);
            soc.receive(DPrec); // Receiving data from a client
            InetAddress ip = DPrec.getAddress();
            System.out.println("\tReceived:" + data(rec) + "\n\tchoice is " + data(ch)
+ "\n");
// data method returns data in the byte[]arrays
            String str = data(rec).toString();
            String choice = data(ch).toString();
            String send = new String();
            int chint = Integer.parseInt(choice) + 1;

            String type = null;
// Getting the type of input sent by client ; chint is 'Choice int'
            switch (Math.round(chint / 2)) {
                case 1:
                    type = "Infix";
                    break;
                case 2:
                    type = "Prefix";
                    break;
                case 3:
                    type = "Postfix";
                    break;
                default:
                    type = "__";
            }

            if (chint / 2 == type(str)) {
```

```
// Checking if the client choice matches with the expression received
                switch (choice) {
                    case "" + '1':
                        send = infix_to_prefix(str).toString();
                        break;
                    case "" + '2':
                        send = infix_to_postfix(str).toString();
                        break;
                    case "" + '3':
                        send = prefix_to_infix(str);
                        break;
                    case "" + '4':
                        send = prefix_to_postfix(str);
                        break;
                    case "" + '5':
                        send = postfix_to_infix(str);
                        break;
                    case "" + '6':
                        send = postfix_to_prefix(str);
                        break;
                    case "" + '7':
                        java.lang.System.exit(0);
                    default:
                        System.out.println("\tInvalid choice.\nExiting..");
                        java.lang.System.exit(0);
// Accessing the methods based on the client choice
                }
            }

            else {
                send = "Entered expression is not " + type;
                st = "" + '0';
                System.out.println("\tEntered expression is not " + type);
// if the choice doesn't match the expression, sending an error message and
 setting the conversion status to '0'
            }

            byte[] Bsend = send.getBytes();
            byte[] Bst = st.getBytes();
            DatagramPacket DPsend = new DatagramPacket(Bsend, Bsend.length, ip, 5000);
            DatagramPacket DPst = new DatagramPacket(Bst, Bst.length, ip, 5000);
            soc.send(DPsend);
            soc.send(DPst); // Sending data back to the client
            System.out.println("\tSent to client");
        }
    }
```

```
    static double type(String exp) { // Method to check whether the received
expression is Infix or Prefix or Postfix
        int i = 0;
        double v = 1.0;
        while (i < exp.length() - 1) {
            if (isOperator(exp.charAt(i)) && isOperator(exp.charAt(i + 1))) {
                v = 0.0;
                break;
            }
            i++;
        }
        if (isOperator(exp.charAt(0)))
            v = 2.0;
        if (isOperator(exp.charAt(exp.length() - 1)))
            v = 3.0;
        return v;
    }

    public static StringBuilder data(byte[] a)
// data method returns data in the byte[] arraysl
    {
        if (a == null)
            return null;
        StringBuilder ret = new StringBuilder();
        int i = 0;
        while (a[i] != 0) {
            ret.append((char) a[i]);
            i++;
        }
        return ret;
    }

 // Method to check if the character is an operator or not
static boolean isOperator(char x) {
        switch (x) {
            case '+':
            case '-':
            case '/':
            case '*':
                return true;
        }
        return false;
    }
```

```
// Method to check the precedence of the operator
    static int precedence(char c) {
        switch (c) {
            case '+':
            case '-':
                return 1;
            case '*':
            case '/':
                return 2;
            case '^':
                return 3;
        }
        return -1;
    }
// Method to convert infix to prefix
    static StringBuilder infix_to_prefix(String expression) {
        StringBuilder result = new StringBuilder();
        StringBuilder input = new StringBuilder(expression);
        input.reverse();
        Stack<Character> stack = new Stack<Character>();

        char[] charsExp = new String(input).toCharArray();
        for (int i = 0; i < charsExp.length; i++) {

            if (charsExp[i] == '(') {
                charsExp[i] = ')';
                i++;
            } else if (charsExp[i] == ')') {
                charsExp[i] = '(';
                i++;
            }
        }
        for (int i = 0; i < charsExp.length; i++) {
            char c = charsExp[i];

            // check if char is operator or operand
            if (precedence(c) > 0) {
                while (stack.isEmpty() == false && precedence(stack.peek()) >=
precedence(c)) {
                    result.append(stack.pop());
                }
                stack.push(c);
            } else if (c == ')') {
                char x = stack.pop();
                while (x != '(') {
                    result.append(x);
                    x = stack.pop();
```

```
                }
            } else if (c == '(') {
                stack.push(c);
            } else {
                // character is neither operator nor "("
                result.append(c);
            }
        }

        for (int i = 0; i <= stack.size(); i++) {
            result.append(stack.pop());
        }
        return result.reverse();
    }
// Method to convert infix to postfix
      static String infix_to_postfix(String expression) {
        String result = "";
        Stack<Character> stack = new Stack<>();
        for (int i = 0; i < expression.length(); i++) {
            char c = expression.charAt(i);

            // check if char is operator
            if (precedence(c) > 0) {
                while (stack.isEmpty() == false && precedence(stack.peek()) >=
precedence(c)) {
                    result += stack.pop();
                }
                stack.push(c);
            } else if (c == ')') {
                char x = stack.pop();
                while (x != '(') {
                    result += x;
                    x = stack.pop();
                }
            } else if (c == '(') {
                stack.push(c);
            } else {
                // character is neither operator nor (
                result += c;
            }
        }
        for (int i = 0; i <= stack.size(); i++) {
            result += stack.pop();
        }
        return result;
    }
```

```
// Method to convert postfix to infix
    static String prefix_to_infix(String expression) {
        Stack<String> stack = new Stack<>();
        for (int i = expression.length() - 1; i >= 0; i--) {
            char c = expression.charAt(i);

            if (isOperator(c)) {
                String s1 = stack.pop();
                String s2 = stack.pop();
                String temp = s1 + c + s2;
                stack.push(temp);
            } else {
                stack.push(c + "");
            }
        }

        String result = stack.pop();

        return result;
    }
// Method to convert prefix to postfix

    static String prefix_to_postfix(String expression) {
        Stack<String> stack = new Stack<String>();
        for (int i = expression.length() - 1; i >= 0; i--) {

            char c = expression.charAt(i);

            if (isOperator(c)) {
                String s1 = stack.pop();
                String s2 = stack.pop();
                String temp = s1 + s2 + c;
                stack.push(temp);
            } else {
                stack.push(c + "");
            }
        }

        String result = stack.pop();
        return result;
    }
// Method to convert postfix to infix

    static String postfix_to_infix(String expression) {
        Stack<String> stack = new Stack<>();
        for (int i = 0; i < expression.length(); i++) {
            char c = expression.charAt(i);
```

```
            if (c == '*' || c == '/' || c == '^' || c == '+' || c == '-') {
                String s1 = stack.pop();
                String s2 = stack.pop();
                String temp = s2 + c + s1;
                stack.push(temp);
            } else {
                stack.push(c + "");
            }
        }

        String result = stack.pop();
        return result;
    }
// Method to convert postfix to prefix

    static String postfix_to_prefix(String expression) {

        Stack<String> stack = new Stack<>();
        for (int i = 0; i < expression.length(); i++) {

            char c = expression.charAt(i);

            if (isOperator(c)) {
                String s1 = stack.pop();
                String s2 = stack.pop();
                String temp = c + s2 + s1;
                stack.push(temp);
            } else {
                stack.push(c + "");
            }
        }
        String result = stack.pop();
        return result;
    }

}
```

5.OUTPUT

5.1 OUTPUT FOR UDP SERVER:

```
c:\Users\AADRITH\Desktop\Akshith\Studies\Sem 5\Comp Network\Project\Work>javac UDPserver
.java

c:\Users\AADRITH\Desktop\Akshith\Studies\Sem 5\Comp Network\Project\Work>java UDPserver.
java


Waiting for client...
        Received:a+b-c*d
        choice is 1

        Sent to client


Waiting for client...
        Received:a+b-c*d
        choice is 2

        Sent to client


Waiting for client...
        Received:+a-b*cd
        choice is 3

        Sent to client


Waiting for client...
        Received:+a-b*cd
        choice is 4

        Sent to client


Waiting for client...
        Received:abcd*-+
        choice is 5

        Sent to client
```

5.2 OUTPUT FOR UDP CLIENT:

```
c:\Users\AADRITH\Desktop\Akshith\Studies\Sem 5\Comp Network\Project\Work>java UDPclient.
java

Choices :

1 - Infix to Prefix
2 - Infix to Postfix
3 - Prefix to Infix
4 - Prefix to Postfix
5 - Postfix to Infix
6 - Postfix to Prefix
7 - Exit

Enter choice: 1
Enter the expression: a+b-c*d

Sent to server

Waiting for server....
Received
Prefix of a+b-c*d is +a-b*cd
Enter '0' if you want to exit!
abcd

Choices :

1 - Infix to Prefix
2 - Infix to Postfix
3 - Prefix to Infix
4 - Prefix to Postfix
5 - Postfix to Infix
6 - Postfix to Prefix
7 - Exit

Enter choice: 2
Enter the expression: a+b-c*d

Sent to server

Waiting for server....
Received
Postfix of a+b-c*d is ab+cd*-
Enter '0' if you want to exit!
1
```

```
Choices :

1 - Infix to Prefix
2 - Infix to Postfix
3 - Prefix to Infix
4 - Prefix to Postfix
5 - Postfix to Infix
6 - Postfix to Prefix
7 - Exit

Enter choice: 3
Enter the expression: +a-b*cd

Sent to server

Waiting for server....
Received
Infix of +a-b*cd is a+b-c*d
Enter '0' if you want to exit!
1

Choices :

1 - Infix to Prefix
2 - Infix to Postfix
3 - Prefix to Infix
4 - Prefix to Postfix
5 - Postfix to Infix
6 - Postfix to Prefix
7 - Exit

Enter choice: 4
Enter the expression: +a-b*cd

Sent to server

Waiting for server....
Received
Postfix of +a-b*cd is abcd*-+
Enter '0' if you want to exit!
1
```

```
Enter choice: 5
Enter the expression: abcd*-+

Sent to server

Waiting for server....
Received
Infix of abcd*-+ is a+b-c*d
Enter '0' if you want to exit!
dfghj

Choices :

1 - Infix to Prefix
2 - Infix to Postfix
3 - Prefix to Infix
4 - Prefix to Postfix
5 - Postfix to Infix
6 - Postfix to Prefix
7 - Exit
```

4.3 output for Entering wrong choice:

UDP server:

```
Waiting for client...
        Received:a+B
        choice is abcdefg

        Sent to client
```

UDP client:

```
Enter choice: abcdefg
Enter the expression: a+B

Sent to server

Waiting for server....
Received
Invalid choice.
```

6. OBSERVATIONS

- **Sockets** in computer networks are used for allowing the transmission of information between two processes of the same machines or different machines in the network. The socket is the combination of IP address and software port number used for communication between multiple processes. Socket helps to recognize the address of the application to which data is to be sent using the IP address and port number.

- The **client-server** network model is one of the most widely used networking models. In the client-server network, the files are not stored on the hard drive of each computer system.

- A server is always ON so client machines can access the files and resources without caring whether the server computer system is ON or not. One of the major drawbacks of the client-server model is that if the server is turned OFF (due to any certain reason), the resources present on the server will not be available to the clients.

- In the client-server model, the files are centrally stored and backed up on a specialized computer known as a server. A client does not share any of its resources, but it requests data or services from a server.
- UDP protocol can be utilized for simple request-response communication when there is a smaller size data since there is very less concern about the error and flow control. UDP carries packet switching, so UDP is considered suitable protocol for multicasting.

- Some of the implementations that use UDP as its transport layer protocol are NTP, DNS, TFTP, RTSP, RIP.

- UDP is a simplest transport layer protocol. UDP protocol is a connection-less, unreliable transport layer protocol. User Datagram Protocol packets are called user datagram.

- UDP does not provide any error and flow control and acknowledgment mechanism.

7.CONCLUSION

- This mini project details the socket programming. A Socket is a communications connection point (endpoint) that you can name and address in a network. Socket programming shows how to use socket APIs to establish communication links between remote and local processes.
- This project includes the how the client and server works explained by the client server model. Normally, client server architecture is a basic arrangement where clients are located at the workstations while servers are located far away on the powerful machines in the network. This model is beneficial in the office environments where the clients and servers usually perform routine tasks.
- In this project using UDP for conversions, for sure, the development of UDP (User Datagram Protocol) is revolutionary. It allows fast delivery, which is highly valuable for a number of applications.
- The main purpose of the UDP protocol enables computer applications to send messages, referred to as datagrams, to other systems on an IP (Internet Protocol) network. UDP is a minimal, connectionless protocol and doesn't require any handshake communication before setting up communication channels or data routes.
- Using client must specify the server's IP address and port number for the call to sendto. using UDP converting infix to postfix, infix to prefix, postfix to prefix, postfix to infix, prefix to postfix, prefix to infix based on client server architecture. Stack data structure used in this Conversions.
- This study helps us to realize the advantages and disadvantages of client server model and TCP vs UDP.UDP can form duplicates but in TCP does not possible for duplicates.
- The solution and implementation for this project is based on UDP client server model. That we were able to execute the conversions of postfix, prefix, infix.

8. LEARNING OUTCOMES

Following are the learning outcomes of us by this mini project that is" implement a Client-Server based prefix to postfix, postfix to prefix, prefix to infix, infix to prefix, postfix to infix and infix to postfix conversion application using UDP.

- After this experiment, we are familiar with Socket programming it's different types of sockets, where is socket used in programming.[1]
- We are also able to understand client server model, how it's working in TCP and UDP.[2]
- Before this Experiment mostly we used TCP client Server architecture for implementing programs on socket but, after this experiment we are able to implement codes in TCP as well as UDP.[5]
- We got an idea on how UDP client & UDP server works how they get connected.[3]
- We learnt the advantages and disadvantages of UDP from that we conclude how UDP works and draw backs of UDP. [4]
- After this project we are able to understand the how infix converted to prefix, postfix converted to infix, prefix converted to postfix ….. using user datagram protocol.[5]
- Finally, we learnt UDP client server working model and have a detailed knowledge of the UDP Sockets.[6]

9.REFERENCES

[1] **UNIX® Network Programming Volume 1, Third Edition**: The Sockets Networking API By W. Richard Stevens, Bill Fenner, Andrew M. Rudoff.

[2] Lectures on distributed system.pdf: Client-server communication UDP/TCP By Paul Krzyzanowski.

[3] Geeks For Geeks: https://www.geeksforgeeks.org/udp-server-client-implementation-c/

[4] https://www.scaler.com/topics/computer-network/

[5] Website: https://www.cs.man.ac.uk/~pjj/cs212/fix.html

[6] TCP/IP illustrated, volume 1: The protocols, 2nd Edition: **By Kevin R. Fall and Author: W. Richard Stevens.**