KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTION)

Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH,
Narayanguda, Hyderabad – 500029

DEPARTMENT OF INFORMATION TECHNOLOGY

LAB RECORD

COMPUTER NETWORKS LAB

B.Tech. III YEAR I SEM (RKR21)
ACADEMIC YEAR 2024-25

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
(AN AUTONOMOUS INSTITUTE)
Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Hyderabad
Department of Information Technology

Certificate

This is to certify that following is a Bonafide Record of the workbook task done by

_____bearing Roll No_____of _____

Branch of _____year B.Tech Course in the_____

Subject during the Academic year _____& _____under our supervision.

Number of week tasks completed: _____

Signature of Internal Examiner

Signature of Head of the Dept.

Signature of External Examiner

Daily Laboratory Assessment Sheet

Name of the Lab: Name of the Student:

Class: HT.No:

S.No.	Name of the Experiment	Date	Observation Marks (3M)	Record Marks (4M)	Viva Voice Marks (3M)	Total Marks (10M)	Signature of Faculty
	TOTAL						

Faculty Incharge

INDEX

8	Implementation of concurrent chat server that allows current logged in users to communicate one with other.	
9	Implementation of DNS.	
10	Implementation of Ping service.	

Department of Information Technology

Vision of the Institution:

To be the fountainhead of latest technologies, producing highly skilled, globally competent engineers.

Mission of the Institution:

- To provide a learning environment that inculcates problem solving skills, professional, ethical responsibilities, lifelong learning through multi modal platforms and prepare students to become successful professionals.
- To establish Industry Institute Interaction to make students ready for the industry.
- To provide exposure to students on latest hardware and software tools.
- To promote research-based projects/activities in the emerging areas of technology convergence.
- To encourage and enable students to not merely seek jobs from the industry but also to create new enterprises
- To induce a spirit of nationalism which will enable the student to develop, understand India's challenges and to encourage them to develop effective solutions.
- To support the faculty to accelerate their learning curve to deliver excellent service to students.

Department of Information Technology

Vision of the Department:

To produce globally competent graduates to meet the modern challenges through contemporary knowledge and moral values committed to build a vibrant nation.

Mission of the Department:

- To create an academic environment, which promotes the intellectual and professional development of students and faculty.
- To impart skills beyond university prescribed to transform students into a well-rounded IT professional.
- To nurture the students to be dynamic, industry ready and to have multidisciplinary skills including e-learning, blended learning and remote testing as an individual and as a team.
- To continuously engage in research and projects development, strategic use of emerging technologies to attain self-sustainability

Department of Information Technology

PROGRAM OUTCOMES (POs)

PO1: Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: Problem Analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3: Design/Development of Solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4: Conduct Investigations of Complex Problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern Tool Usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6: The Engineer and Society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and Sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and Team Work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project Management and Finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long Learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: An ability to analyze the common business functions to design and develop appropriate Computer Science solutions for social upliftment.

PSO2: Shall have expertise on the evolving technologies like Python, Machine Learning, Deep Learning, Internet of Things (IOT), Data Science, Full stack development, Social Networks, Cyber Security, Big Data, Mobile Apps, CRM, ERP etc.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Graduates will have successful careers in computer related engineering fields or will be able to successfully pursue advanced higher education degrees.

PEO2: Graduates will try and provide solutions to challenging problems in their profession by applying computer engineering principles.

PEO3: Graduates will engage in life-long learning and professional development by rapidly adapting changing work environment.

PEO4: Graduates will communicate effectively, work collaboratively and exhibit high levels of professionalism and ethical responsibility.

B. Tech. in Information Technology

III Year I Semester Syllabus (RKR21)
COMPUTER NETWORKS LAB (21IT501PC)

L	T	P	C
0	0	3	1.5

Pre-requisite/ Co-requisites:

1. PP102ES - Programming for problem solving Course.
2. 21CS401PC - Java Programming Course.
3. 21CS501PC – Computer Networks Course.

Course Objectives: The course will help to

1. Introduce CRC Mechanism.
2. Understand the concepts of Data link layer.
3. Gain the knowledge on network layer.
4. Understand the concepts of TCP and UDP Protocols.
5. Learn the concepts of sockets and DNS.

Course Outcomes: After learning the concepts of this course, the student is able to

1. Compute CRC Mechanisms.
2. Demonstrate and implement the Go-Back-N mechanism.
3. Demonstrate and Apply routing algorithms.
4. Illustrate and implement TCP and UDP Client and server Applications.
5. Develop DNS and Ping service.

Software to be used: The students can use any OS with Java.

List of Programs: Using C/Java programming

1. Write a program to compute CRC code for the polynomials CRC-12, CRC-16 and CRC CCIP
2. Develop a simple data link layer that performs the flow control using the sliding window protocol, and loss recovery using the Go-Back-N mechanism.
3. Take an example subnet of hosts and obtain a broadcast tree for the subnet.

4. Implement distance vector routing algorithm for obtaining routing tables at each node.
5. Design the following
    a. TCP iterative Client and server application to reverse the given input sentence.
    b. TCP client and server application to transfer file.
    c. TCP concurrent server to convert a given text into upper case using multiplexing system call "select".
    d. TCP concurrent server to echo given set of sentences using poll functions.
6. Design the following
    a. UDP Client and server application to reverse the given input sentence. 2018-2019 173.
    b. UDP Client server to transfer a file.
7. Programs to demonstrate the usage of Advanced socket system calls like getsockopt( ), setsockopt(), getpeername ( ),getsockname( ),readv( ) and writev( ).
8. Implementation of concurrent chat server that allows current logged in users to communicate one with other.
9. Implementation of DNS.
10. Implementation of Ping service.

TEXT BOOKS:

1. Data Communications and Networking-Behrouz A. Forouzan, Fourth Edition TMH,2006.
2. Computer Networks- Andrew S Tanenbaum, 4th Edition. Pearson Education,PHI.
3. UNIX Network Programming-W.Richard Stevens, Bill Fenner, Andrew M. Rudoff, Pearson Education.
4. UNIX Network Programming- – W. Richard Stevens, PHI 1st Edition.

REFERENCE BOOKS:

1. Data communications and Computer Networks- P.C Gupta, PHI.
2. An Engineering Approach to Computer Networks-S. Keshav, 2nd Edition, Pearson Education.
3. Understanding communications and Networks- W.A. Shay, C engage Learning 3rd Edition.
4. Computer Networking: A Top-Down Approach Featuring the Internet-James F.Kurose & Keith W. Ross, 3rdEdition, Pearson Education.
5. Data and Computer Communication-William Stallings, Pearson Education,6thEdition ,2000.
6. UNIX for Programmers and Users- Graham GLASS, King abls, Pearson Education 3rd Edition.
7. Advanced UNIX Programming- M. J. ROCHKIND, Pearson Education, 2nd Edition.

KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTE)

Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Hyderabad

DEPARTMENT OF INFORMATION TECHNOLOGY

Course Objectives:

- Introduce CRC Mechanism.
- Understand the concepts of Data link layer.
- Gain the knowledge on network layer.
- Understand the concepts of TCP and UDP Protocols.
- Learn the concepts of sockets and DNS.

Course Outcomes:

After learning the contents of this course, the student is able to

CO 1: Compute CRC Mechanisms.

CO 2: Demonstrate and implement the Go-Back-N mechanism.

CO 3: Demonstrate and Apply routing algorithms.

CO 4: Illustrate and implement TCP and UDP Client and server Applications.

CO 5: Develop DNS and Ping service.

CO-PO MAPPING:

	CO	PO1	PO2	PO3	PO4	PO5	PO6	PO 7	PO8	PO9	PO10	PO11	PO12
Computer Networks Lab	**CO1**	1	2		3								
	CO2		1		2								
	CO3		2	1	2	3							
	CO4		1			2							
	CO5	1	1		2								

CO-PSO MAPPING:

	PSO-1	PSO-2
CO1	1	
CO2		
CO3		2
CO4		
CO5	2	

EXPERIMENT-1

Write a program to compute CRC code for the polynomials

```
//  Include  headers
#include<stdio.h>
#include<string.h>
// length of the generator polynomial
#define N strlen(gen_poly)
// data to be transmitted and received
char data[28];
// CRC value
char check_value[28];
// generator polynomial
char gen_poly[10];
// variables
int data_length,i,j;
// function that performs XOR operation
void XOR(){
   // if both bits are the same, the output is 0
   // if the bits are different the output is 1
   for(j = 1;j < N; j++)
   check_value[j] = (( check_value[j] == gen_poly[j])?'0':'1');
}
void crc(){
   // initializing check_value
   for(i=0;i<N;i++)
       check_value[i]=data[i];
```

```
    do{
    // check if the first bit is 1 and calls XOR function
        if(check_value[0]=='1')
            XOR();
// Move the bits by 1 position for the next computation
        for(j=0;j<N-1;j++)
            check_value[j]=check_value[j+1];
        // appending a bit from data
        check_value[j]=data[i++];
    }while(i<=data_length+N-1);
// loop until the data ends
}
// Function to check for errors on the receiver side
void receiver(){
// get the received data
    printf("Enter the received data: ");
   scanf("%s", data);
    printf("\n---------------------------------\n");
    printf("Data received: %s", data);
// Cyclic Redundancy Check
    crc();
// Check if the remainder is zero to find the error
    for(i=0;(i<N-1) && (check_value[i]!='1');i++);
if(i<N-1)
            printf("\nError detected\n\n");
        else
```

```
        printf("\nNo error detected\n\n");
}
int main()
{
    // get the data to be transmitted
    printf("\nEnter data to be transmitted: ");
    scanf("%s",data);
    printf("\n Enter the Generating polynomial: ");
    // get the generator polynomial
    scanf("%s",gen_poly);
    // find the length of data
    data_length=strlen(data);
    // appending n-1 zeros to the data
    for(i=data_length;i<data_length+N-1;i++)
        data[i]='0';
    printf("\n----------------------------------------------");
// print the data with padded zeros
    printf("\n Data padded with n-1 zeros : %s",data);
    printf("\n----------------------------------------------");
// Cyclic Redundancy Check
    crc();
// print the computed check value
    printf("\nCRC or Check value is : %s",check_value);
// Append data with check_value(CRC)
    for(i=data_length;i<data_length+N-1;i++)
        data[i]=check_value[i-data_length];
```

```
    printf("\n-----------------------------------------");
// printing the final data to be sent
    printf("\n Final data to be sent : %s",data);
    printf("\n-----------------------------------------\n");
// Calling the receiver function to check errors
    receiver();
        return 0;
}
```

EXPERIMENT-2

Develop a simple data link layer that performs the flow control using the sliding window protocol,and loss recovery using the Go-Back-N mechanism.

```
#include   <stdio.h>
#include  <stdlib.h>
#include  <string.h>
#include <unistd.h>
#include <time.h>
#define WINDOW_SIZE 4 // Size of the sliding window
#define TOTAL_FRAMES 10 // Total number of frames to be sent
#define TIMEOUT 3 // Timeout duration in seconds
int send_frame(int frame) {
    // Simulate random loss of frames (e.g., frame 2 is lost)
    if (rand() % 5 == 0) {
        printf("[Sender] Frame %d lost during transmission.\n", frame);
        return 0; // Simulate frame loss
    }
    printf("[Sender] Frame %d sent successfully.\n", frame);
    return 1; // Frame sent successfully
}
int receive_ack(int frame) {
    // Simulate random loss of ACKs (e.g., ACK for frame 3 is lost)
    if (rand() % 7 == 0) {
        printf("[Receiver] ACK for Frame %d lost.\n", frame);
        return 0; // Simulate ACK loss
    }
    printf("[Receiver] ACK for Frame %d received successfully.\n", frame);
```

```
    return 1; // ACK received successfully
}
void sliding_window_protocol() {
    int base = 0;      // Oldest unacknowledged frame
    int next_frame = 0; // Next frame to be sent
    int acks[TOTAL_FRAMES] = {0}; // Array to track acknowledgments
    time_t timers[TOTAL_FRAMES]; // Timers for each frame
    while (base < TOTAL_FRAMES) {
        // Send frames within the window
        while (next_frame < base + WINDOW_SIZE && next_frame < TOTAL_FRAMES) {
            if (!acks[next_frame]) {
                send_frame(next_frame);
                timers[next_frame] = time(NULL); // Start the timer
            }
            next_frame++;
        }
        // Check for ACKs or timeouts
        for (int i = base; i < next_frame; i++) {
            if (acks[i]) {
                continue; // Skip acknowledged frames
            }
            // Simulate receiving an acknowledgment
            if (receive_ack(i)) {
                acks[i] = 1; // Mark frame as acknowledged
                // Slide the window if the base frame is acknowledged
                while (base < TOTAL_FRAMES && acks[base]) {
```

```
            printf("[Sender] Sliding window. Base frame is now %d.\n", base + 1)
        base++;
          }
        } else {
          // Check for timeout
          if (difftime(time(NULL), timers[i]) > TIMEOUT) {
            printf("[Sender] Timeout for Frame %d. Resending all frames from %d.\n", i, base);
            // Resend all frames from the base frame
            next_frame = base;
            break;
          }
        }
      }
  }
  printf("[Sender] All frames sent and acknowledged successfully.\n");
}
int main() {
  srand(time(NULL)); // Seed for random number generation

  printf("[Sender] Starting Sliding Window Protocol with Go-Back-N.\n");
  sliding_window_protocol();
  printf("[Sender] Transmission completed.\n");
  return 0;
}
```

sent is received by sender

EXPERIMENT-3

Take an example subnet of hosts and obtain a broadcast tree for the subnet.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// Function to convert an IP address to a 32-bit integer
unsigned int ipToInt(char* ip) {
    unsigned int a, b, c, d;
    sscanf(ip, "%u.%u.%u.%u", &a, &b, &c, &d);
    return (a << 24) | (b << 16) | (c << 8) | d;
}
// Function to convert a 32-bit integer to an IP address
void intToIp(unsigned int ip, char* buffer) {
    sprintf(buffer, "%u.%u.%u.%u", (ip >> 24) & 0xFF, (ip >> 16) & 0xFF, (ip >> 8) & 0xFF, ip & 0xFF);
}
// Function to calculate the subnet mask from a prefix length
unsigned int calculateSubnetMask(int prefixLength) {
    return prefixLength == 0 ? 0 : ~((1 << (32 - prefixLength)) - 1);
}
int main() {
    char ip[16];
    int prefixLength, newPrefixLength;
    unsigned int subnetMask, newSubnetMask, ipInt;
    char buffer[16];
    // Input IP address and prefix length
    printf("Enter IP address (e.g., 192.168.1.0): ");
    scanf("%s", ip);
    printf("Enter current prefix length (e.g., 24): ");
    scanf("%d", &prefixLength);
```

```
    // New prefix length for creating two subnets
    newPrefixLength = prefixLength + 1;
    // Convert IP address to integer
    ipInt = ipToInt(ip);
    // Calculate original subnet mask and new subnet mask
    subnetMask = calculateSubnetMask(prefixLength);
    newSubnetMask = calculateSubnetMask(newPrefixLength);
    // Calculate the number of hosts per subnet
    int hostsPerSubnet = (1 << (32 - newPrefixLength)) - 2; // subtract 2 for network and broadcast addresses
    printf("\nNumber of subnets: 2\n");
    printf("Number of hosts per subnet: %d\n", hostsPerSubnet);
    // Generate subnets
    for (int i = 0; i < 2; i++) {
        unsigned int subnetNetwork = (ipInt & subnetMask) | (i << (32 - newPrefixLength));
        unsigned int subnetBroadcast = subnetNetwork | ~newSubnetMask;
        unsigned int firstHost = subnetNetwork + 1;
        unsigned int lastHost = subnetBroadcast - 1;
        printf("\nSubnet %d:\n", i + 1);
        printf("Network Address: ");
        intToIp(subnetNetwork, buffer);
        printf("%s\n", buffer);
        printf("Broadcast Address: ");
        intToIp(subnetBroadcast, buffer);
        printf("%s\n", buffer);
        printf("Subnet Mask: ");
        intToIp(newSubnetMask, buffer);
        printf("%s\n", buffer);
        printf("First Host: ");
        intToIp(firstHost, buffer);
```

```
        printf("%s\n", buffer);
        printf("Last Host: ");
        intToIp(lastHost, buffer);
        printf("%s\n", buffer);
    }
    return 0;
}
```

EXPERIMENT-4

Implement distance vector routing algorithm for obtaining routing tables at each node.

```
#include  <stdio.h>

#include <stdlib.h>

#define INF 9999

#define MAX_NODES 10

// Function to initialize distance vector and routing table

void initialize(int numNodes, int costMatrix[MAX_NODES][MAX_NODES], int
distVector[MAX_NODES][MAX_NODES], int nextHop[MAX_NODES][MAX_NODES]) {

   for (int i = 0; i < numNodes; i++) {

      for (int j = 0; j < numNodes; j++) {

         distVector[i][j] = costMatrix[i][j];

         if (costMatrix[i][j] != INF && i != j) {

            nextHop[i][j] = j;

         } else {

            nextHop[i][j] = -1;

         }

      }

   }

}

// Function to print routing table for each node

void printRoutingTable(int numNodes, int distVector[MAX_NODES][MAX_NODES], int
nextHop[MAX_NODES][MAX_NODES]) {

   for (int i = 0; i < numNodes; i++) {

      printf("Routing table for node %d:\n", i);

      printf("Destination\tNext Hop\tDistance\n");

      for (int j = 0; j < numNodes; j++) {

         if (distVector[i][j] == INF) {

            printf("%d\t\t-\t\tINF\n", j);

         } else {
```

```
            printf("%d\t\t%d\t\t%d\n", j, nextHop[i][j], distVector[i][j]);
        }
    }
    printf("\n");
  }
}
// Function to implement Distance Vector Routing algorithm
void distanceVectorRouting(int numNodes, int costMatrix[MAX_NODES][MAX_NODES], int
distVector[MAX_NODES][MAX_NODES], int nextHop[MAX_NODES][MAX_NODES]) {
  int updated;
  do {
    updated = 0;
    for (int i = 0; i < numNodes; i++) {
      for (int j = 0; j < numNodes; j++) {
        for (int k = 0; k < numNodes; k++) {
          if (distVector[i][k] + distVector[k][j] < distVector[i][j]) {
            distVector[i][j] = distVector[i][k] + distVector[k][j];
            nextHop[i][j] = nextHop[i][k];
            updated = 1;
          }
        }
      }
    }
  } while (updated);
}
int main() {
  int numNodes, costMatrix[MAX_NODES][MAX_NODES];
  int distVector[MAX_NODES][MAX_NODES];
  int nextHop[MAX_NODES][MAX_NODES];
  printf("Enter the number of nodes: ");
```

```
    scanf("%d", &numNodes);
    printf("Enter the cost matrix (use %d for INF):\n", INF);
    for (int i = 0; i < numNodes; i++) {
        for (int j = 0; j < numNodes; j++) {
            scanf("%d", &costMatrix[i][j]);
        }
    }
    initialize(numNodes, costMatrix, distVector, nextHop);
    distanceVectorRouting(numNodes, costMatrix, distVector, nextHop);
    printRoutingTable(numNodes, distVector, nextHop);
    return 0;
}
```

EXPERIMENT-5

Design the following

a. TCP iterative Client and server application to reverse the given input sentence.

Server Program

```
import   java.io.*;

import java.net.*;


public class server {

    public static void main(String[] args) {

        int port = 65432;


        try (ServerSocket serverSocket = new ServerSocket(port)) {

            System.out.println("Server is listening on port " + port);


            while (true) {

                Socket socket = serverSocket.accept();

                System.out.println("New client connected");


                // Create input and output streams for communication

                InputStream input = socket.getInputStream();

                BufferedReader reader = new BufferedReader(new InputStreamReader(input));

                OutputStream output = socket.getOutputStream();

                PrintWriter writer = new PrintWriter(output, true);

                String text;

                while ((text = reader.readLine()) != null) {

                    System.out.println("Received: " + text);

                    // Reverse the string

                    String reversedText = new StringBuilder(text).reverse().toString();

                    System.out.println("Sending: " + reversedText);
```

```
                // Send the reversed string back to the client
                writer.println(reversedText);
            }
            socket.close();
            System.out.println("Client disconnected");
        }
    } catch (IOException ex) {
        System.out.println("Server exception: " + ex.getMessage());
        ex.printStackTrace();
    }
  }
}
```

Client Program

```
import java.io.*;
import java.net.*;
public class Client {
   public static void main(String[] args) {
      String hostname = "127.0.0.1";
      int port = 65432;
      try (Socket socket = new Socket(hostname, port)) {
         System.out.println("Connected to the server");
         // Create input and output streams for communication
         OutputStream output = socket.getOutputStream();
         PrintWriter writer = new PrintWriter(output, true);
         InputStream input = socket.getInputStream();
         BufferedReader reader = new BufferedReader(new InputStreamReader(input));
         // Read user input
         BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));
```

```
            String text;
            System.out.println("Enter a sentence to reverse (or type 'exit' to quit):");
            while (true) {
                System.out.print(">> ");
                text = consoleReader.readLine();
                if ("exit".equalsIgnoreCase(text)) {
                    System.out.println("Closing connection...");
                    break;
                }
                // Send the sentence to the server
                writer.println(text);
                // Receive the reversed sentence from the server
                String reversedText = reader.readLine();
                System.out.println("Reversed sentence: " + reversedText);
            }
        }
catch (UnknownHostException ex) {
            System.out.println("Server not found: " + ex.getMessage());
        } catch (IOException ex) {
            System.out.println("I/O error: " + ex.getMessage());
        }
    }
}
```

b. TCP client and server application to transfer file.

Server Program

```
import   java.io.*;
import java.net.*;
public class FileServer {
    public static void main(String[] args) {
        int port = 65432;
        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Server is listening on port " + port);
            while (true) {
                Socket socket = serverSocket.accept();
                System.out.println("Client connected.");
                try (
                    InputStream inputStream = socket.getInputStream();
                    DataInputStream dataInputStream = new DataInputStream(inputStream);
                    FileOutputStream fileOutputStream = new FileOutputStream("received_file.txt")
                ) {
                    System.out.println("Receiving file...");

                    // Read file data from the client
                    long fileSize = dataInputStream.readLong();
                    byte[] buffer = new byte[4096];
                    int bytesRead;
                    long totalBytesRead = 0;
                    while (totalBytesRead < fileSize && (bytesRead = dataInputStream.read(buffer)) != -1) {
                        fileOutputStream.write(buffer, 0, bytesRead);
                        totalBytesRead += bytesRead;
                    }
```

```
                System.out.println("File received successfully.");
            } catch (IOException e) {
                System.out.println("Error handling file transfer: " + e.getMessage());
            }
            socket.close();
            System.out.println("Connection closed.");
        }
    }
 catch (IOException e) {
        System.out.println("Server exception: " + e.getMessage());
        e.printStackTrace();
    }
  }
}
```

Client Program

```
import   java.io.*;
import java.net.*;
public class FileClient {
   public static void main(String[] args) {
      String hostname = "127.0.0.1";
      int port = 65432;
      try (Socket socket = new Socket(hostname, port)) {
         System.out.println("Connected to the server.");
         // Specify the file to send
         File file = new File("C:\\Users\\sreel\\OneDrive\\Desktop\\fts.txt");
         if (!file.exists()) {
            System.out.println("File does not exist.");
            return;
         }
```

```
            try (
                FileInputStream fileInputStream = new FileInputStream(file);
                OutputStream outputStream = socket.getOutputStream();
                DataOutputStream dataOutputStream = new DataOutputStream(outputStream)
            ) {
                System.out.println("Sending file: " + file.getName());
                // Send file size to the server
                dataOutputStream.writeLong(file.length());
                // Send file data
                byte[] buffer = new byte[4096];
                int bytesRead;
                while ((bytesRead = fileInputStream.read(buffer)) != -1) {
                    dataOutputStream.write(buffer, 0, bytesRead);
                }
                System.out.println("File sent successfully.");
            } catch (IOException e) {
                System.out.println("Error sending file: " + e.getMessage());
            }
        }
    catch (UnknownHostException e) {
            System.out.println("Server not found: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("I/O error: " + e.getMessage());
        }
    }
}
```

C. TCP concurrent server to convert a given text into upper case using multiplexing system call "select" in java

Server Program

```
import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.channels.*;
import java.util.Iterator;
public class Uccs {
    public static void main(String[] args) {
        int port = 65432;
        try (Selector selector = Selector.open();
             ServerSocketChannel serverChannel = ServerSocketChannel.open()) {
            serverChannel.configureBlocking(false);
            serverChannel.bind(new InetSocketAddress(port));
            serverChannel.register(selector, SelectionKey.OP_ACCEPT);
            System.out.println("Server is listening on port " + port);
            while (true) {
                // Block until there is a ready channel
                selector.select();
                // Iterate over the ready keys
                Iterator<SelectionKey> keys = selector.selectedKeys().iterator();
                while (keys.hasNext()) {
                    SelectionKey key = keys.next();
                    keys.remove();
                    if (key.isAcceptable()) {
                        // Accept the new client connection
                        ServerSocketChannel server = (ServerSocketChannel) key.channel();
                        SocketChannel clientChannel = server.accept();
                        clientChannel.configureBlocking(false);
```

```
                clientChannel.register(selector, SelectionKey.OP_READ);
                System.out.println("New client connected: " + clientChannel.getRemoteAddress());

            } else if (key.isReadable()) {
                // Read data from the client
                SocketChannel clientChannel = (SocketChannel) key.channel();
                ByteBuffer buffer = ByteBuffer.allocate(1024);

                int bytesRead = clientChannel.read(buffer);
                if (bytesRead == -1) {
                    // Client has disconnected
                    System.out.println("Client disconnected: " + clientChannel.getRemoteAddress());
                    clientChannel.close();
                } else {
                    buffer.flip();
                    String receivedText = new String(buffer.array(), 0, buffer.limit()).trim();
                    System.out.println("Received: " + receivedText + " from " +
clientChannel.getRemoteAddress());

                    // Convert text to uppercase
                    String upperCaseText = receivedText.toUpperCase();

                    // Send the uppercase text back to the client
                    buffer.clear();
                    buffer.put(upperCaseText.getBytes());
                    buffer.flip();
                    clientChannel.write(buffer);
                    System.out.println("Sent: " + upperCaseText + " to " +
clientChannel.getRemoteAddress());
                }
```

```
                }
            }
        }
    } catch (IOException e) {
        System.out.println("Server error: " + e.getMessage());
        e.printStackTrace();
    }
  }
}
```

Client Program

```
import java.io.*;
import java.net.Socket;
public class Ucasec {
  public static void main(String[] args) {
    String hostname = "127.0.0.1";
    int port = 65432;
    try (Socket socket = new Socket(hostname, port)) {
      System.out.println("Connected to the server.");
      BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));
      PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
      BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
      System.out.println("Enter text to convert to uppercase (type 'exit' to quit):");
      String input;
      while (true) {
        System.out.print(">> ");
        input = consoleReader.readLine();
        if ("exit".equalsIgnoreCase(input)) {
          System.out.println("Closing connection...");
```

```
                break;
            }
            // Send input to the server
            out.println(input);
            // Receive and print the uppercase text from the server
            String response = in.readLine();
            System.out.println("Server response: " + response);
        }
    } catch (IOException e) {
        System.out.println("Client error: " + e.getMessage());
    }
  }
}
```

d. TCP concurrent server to echo given set of sentences using poll functions.

Server Program

```
import   java.io.*;
import java.net.*;
import java.nio.ByteBuffer;
import  java.nio.channels.*;
import java.util.*;
public class ConcurrentServer {
    public static void main(String[] args) {
        int port = 65432;
        List<SocketChannel> clients = new ArrayList<>();
        try (ServerSocketChannel serverChannel = ServerSocketChannel.open()) {
            serverChannel.configureBlocking(false);
            serverChannel.socket().bind(new InetSocketAddress(port));
            System.out.println("Server is listening on port " + port);
            while (true) {
                // Accept new connections
                SocketChannel clientChannel = serverChannel.accept();
                if (clientChannel != null) {
                    clientChannel.configureBlocking(false);
                    clients.add(clientChannel);
                    System.out.println("New client connected: " + clientChannel.getRemoteAddress());
                }
                // Handle client I/O using poll
                List<SocketChannel> readyClients = new ArrayList<>();
                for (SocketChannel client : clients) {
                    try {
                        ByteBuffer buffer = ByteBuffer.allocate(1024);
```

```
                int bytesRead = client.read(buffer);
                if (bytesRead > 0) {
                    buffer.flip();
                    String message = new String(buffer.array(), 0, buffer.limit()).trim();
                    System.out.println("Received: " + message + " from " + client.getRemoteAddress());

                    // Echo the message back to the client
                    buffer.flip();
                    client.write(buffer);
                    System.out.println("Echoed: " + message);
                } else if (bytesRead == -1) {
                    // Client disconnected
                    readyClients.add(client);
                }
            } catch (IOException e) {
                // Handle client disconnection
                readyClients.add(client);
            }
        }

        // Remove disconnected clients
        clients.removeAll(readyClients);
        for (SocketChannel client : readyClients) {
            System.out.println("Client disconnected: " + client.getRemoteAddress());
            client.close();
        }
    }
} catch (IOException e) {
    System.out.println("Server error: " + e.getMessage());
```

```
            e.printStackTrace();
        }
    }
}
```

Client Prog1

```
import java.io.*;
import java.net.*;
public class EchoClient {
    public static void main(String[] args) {
        String hostname = "127.0.0.1";
        int port = 65432;
        try (Socket socket = new Socket(hostname, port)) {
            System.out.println("Connected to the server.");
            // Create input and output streams for communication
            OutputStream output = socket.getOutputStream();
            PrintWriter writer = new PrintWriter(output, true);
            InputStream input = socket.getInputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(input));
            // Read user input
            BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));
            String text;
            System.out.println("Enter sentences to echo (or type 'exit' to quit):");
            while (true) {
                System.out.print(">> ");
                text = consoleReader.readLine();
                if ("exit".equalsIgnoreCase(text)) {
                    System.out.println("Closing connection...");
                    break;
                }
```

```
            // Send the sentence to the server
            writer.println(text);
            // Receive the echoed sentence from the server
            String echoedText = reader.readLine();
            System.out.println("Server echoed: " + echoedText);
        }
    } catch (UnknownHostException e) {
        System.out.println("Server not found: " + e.getMessage());
    } catch (IOException e) {
        System.out.println("I/O error: " + e.getMessage());
    }
  }
}
```

Client Prog2

```
import java.io.*;
import java.net.*;
public class EchoClient1 {
  public static void main(String[] args) {
    String hostname = "127.0.0.1";
    int port = 65432;
    try (Socket socket = new Socket(hostname, port)) {
        System.out.println("Connected to the server.");
        // Create input and output streams for communication
        OutputStream output = socket.getOutputStream();
        PrintWriter writer = new PrintWriter(output, true);
        InputStream input = socket.getInputStream();
        BufferedReader reader = new BufferedReader(new InputStreamReader(input));
        // Read user input
        BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));
```

```
            String text;
            System.out.println("Enter sentences to echo (or type 'exit' to quit):");
            while (true) {
                System.out.print(">> ");
                text = consoleReader.readLine();
                if ("exit".equalsIgnoreCase(text)) {
                    System.out.println("Closing connection...");
                    break;
                }
                // Send the sentence to the server
                writer.println(text);
                // Receive the echoed sentence from the server
                String echoedText = reader.readLine();
                System.out.println("Server echoed: " + echoedText);
            }
        } catch (UnknownHostException e) {
            System.out.println("Server not found: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("I/O error: " + e.getMessage());
        }
    }
}
```

EXPERIMENT-6

Design the following

a. UDP Client and server application to reverse the given input sentence. 2018-2019 173.

Server Program

```
import java.net.*;
public class UDPRS {
    public static void main(String[] args) {
        int port = 9876;
        try (DatagramSocket serverSocket = new DatagramSocket(port)) {
            System.out.println("Server is running on port " + port);
            byte[] receiveBuffer = new byte[1024];
            byte[] sendBuffer;
            while (true) {
                // Receive packet
                DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);
                serverSocket.receive(receivePacket);
                String receivedSentence = new String(receivePacket.getData(), 0, receivePacket.getLength());
                System.out.println("Received: " + receivedSentence);
                // Reverse the sentence
                String reversedSentence = new StringBuilder(receivedSentence).reverse().toString();
                // Send the reversed sentence back
                sendBuffer = reversedSentence.getBytes();
                InetAddress clientAddress = receivePacket.getAddress();
                int clientPort = receivePacket.getPort();
                DatagramPacket sendPacket = new DatagramPacket(sendBuffer, sendBuffer.length,
clientAddress, clientPort);
                serverSocket.send(sendPacket);
                System.out.println("Sent: " + reversedSentence);
            }
        } catch (Exception e) {
```

```
            System.out.println("Server error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Client Program

```
import java.net.*;
import java.util.Scanner;
public class UDPRC {
    public static void main(String[] args) {
        String serverHostname = "127.0.0.1";
        int serverPort = 9876;
        try (DatagramSocket clientSocket = new DatagramSocket();
             Scanner scanner = new Scanner(System.in)) {
            InetAddress serverAddress = InetAddress.getByName(serverHostname);
            byte[] sendBuffer;
            byte[] receiveBuffer = new byte[1024];
            System.out.println("Enter sentences to reverse (type 'exit' to quit):");
            while (true) {
                System.out.print(">> ");
                String inputSentence = scanner.nextLine();
                if ("exit".equalsIgnoreCase(inputSentence)) {
                    System.out.println("Exiting...");
                    break;
                }
                // Send input sentence to server
                sendBuffer = inputSentence.getBytes();
                DatagramPacket sendPacket = new DatagramPacket(sendBuffer, sendBuffer.length, serverAddress, serverPort);
                clientSocket.send(sendPacket);
```

```
            // Receive reversed sentence from server
            DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);
            clientSocket.receive(receivePacket);
            String reversedSentence = new String(receivePacket.getData(), 0, receivePacket.getLength());
            System.out.println("Server response: " + reversedSentence);
        }
    } catch (Exception e) {
        System.out.println("Client error: " + e.getMessage());
        e.printStackTrace();
    }
  }
}
```

b. UDP Client server to transfer a file.

Server Program

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
public class UDPFS {
    public static void main(String[] args) {
        int port = 9876;
        String outputFileName = "received_file.txt";
        try (DatagramSocket serverSocket = new DatagramSocket(port);
             FileOutputStream fileOutputStream = new FileOutputStream(outputFileName)) {
            System.out.println("Server is running on port " + port);
            byte[] receiveBuffer = new byte[1024];
            DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);
            System.out.println("Waiting for file...");
            while (true) {
                serverSocket.receive(receivePacket);
                // Check if it's the end of the file
                String endSignal = new String(receivePacket.getData(), 0, receivePacket.getLength());
                if ("EOF".equals(endSignal)) {
                    System.out.println("File transfer complete.");
                    break;
                }
                // Write data to the file
                fileOutputStream.write(receivePacket.getData(), 0, receivePacket.getLength());
            }
        } catch (IOException e) {
            System.out.println("Server error: " + e.getMessage());
```

```
            e.printStackTrace();
        }
    }
}
```

Client Program

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
public class UDPFC {
    public static void main(String[] args) {
        String serverAddress = "127.0.0.1";
        int serverPort = 9876;
        String filePath = "C:\\Users\\sreel\\OneDrive\\Desktop\\fts.txt";
        try (DatagramSocket clientSocket = new DatagramSocket();
             FileInputStream fileInputStream = new FileInputStream(new File(filePath))) {
            InetAddress serverInetAddress = InetAddress.getByName(serverAddress);
            byte[] sendBuffer = new byte[1024];
            int bytesRead;
            System.out.println("Sending file...");
            while ((bytesRead = fileInputStream.read(sendBuffer)) != -1) {
                DatagramPacket sendPacket = new DatagramPacket(sendBuffer, bytesRead, serverInetAddress,
serverPort);
                clientSocket.send(sendPacket);
            }
            // Send EOF signal
            String endSignal = "EOF";
            sendBuffer = endSignal.getBytes();
```

```
            DatagramPacket endPacket = new DatagramPacket(sendBuffer, sendBuffer.length, serverInetAddress, serverPort);

            clientSocket.send(endPacket);

            System.out.println("File sent successfully.");

        } catch (IOException e) {

            System.out.println("Client error: " + e.getMessage());

            e.printStackTrace();

        }

    }

}
```

EXPERIMENT-7

Programs to demonstrate the usage of Advanced socket system calls like getsockopt( ), setsockopt(), getpeername ( ),getsockname( ),readv( ) and writev( ).

Server Program

```
import   java.io.*;

import java.net.*;

public   class   AdvancedSocketServer   {

   public static void main(String[] args) {

      int port = 9876;

      try (ServerSocket serverSocket = new ServerSocket(port)) {

         System.out.println("Server is listening on port " + port);

         while (true) {

            Socket socket = serverSocket.accept();

            System.out.println("Client connected.");

            // Retrieve and print socket options

            System.out.println("Server Socket Options:");

            System.out.println("SO_RCVBUF (Receive Buffer Size): " + socket.getReceiveBufferSize());

            System.out.println("SO_SNDBUF (Send Buffer Size): " + socket.getSendBufferSize());

            System.out.println("SO_TIMEOUT (Socket Timeout): " + socket.getSoTimeout());

            // Echo back received data

            BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

            PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);

            String line;

            while ((line = reader.readLine()) != null) {

               System.out.println("Received: " + line);

               writer.println("Echo: " + line);

            }
```

```
            socket.close();
            System.out.println("Client disconnected.");
        }
    } catch (IOException e) {
        System.out.println("Server error: " + e.getMessage());
        e.printStackTrace();
    }
  }
}
```

Client Program

```
import java.io.*;
import java.net.*;
public class AdvancedSocketClient {
  public static void main(String[] args) {
    String serverAddress = "127.0.0.1";
    int serverPort = 9876;
    try (Socket socket = new Socket(serverAddress, serverPort)) {
      System.out.println("Connected to the server.");
      // Retrieve and print socket options
      System.out.println("Client Socket Options:");
      System.out.println("SO_RCVBUF (Receive Buffer Size): " + socket.getReceiveBufferSize());
      System.out.println("SO_SNDBUF (Send Buffer Size): " + socket.getSendBufferSize());
      System.out.println("TCP_NODELAY (Disable Nagle's Algorithm): " + socket.getTcpNoDelay());
      // Send messages to the server
      PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
      BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
      BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));
```

```
            System.out.println("Type messages to send to the server (type 'exit' to quit):");
            String message;
            while (!(message = consoleReader.readLine()).equalsIgnoreCase("exit")) {
                writer.println(message);
                String response = reader.readLine();
                System.out.println("Server response: " + response);
            }
        } catch (IOException e) {
            System.out.println("Client error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

EXPERIMENT-8

Implementation of concurrent chat server that allows current logged in users to communicate one with other.

Server Program

```
import   java.io.*;
import java.net.*;
import java.util.*;
public class ChatServer {
   private static final int PORT = 9876;
   private static Set<ClientHandler> clientHandlers = new HashSet<>();
   public static void main(String[] args) {
      System.out.println("Chat server is running on port " + PORT);
      try (ServerSocket serverSocket = new ServerSocket(PORT)) {
         while (true) {
            Socket clientSocket = serverSocket.accept();
            System.out.println("New client connected: " + clientSocket);
            ClientHandler clientHandler = new ClientHandler(clientSocket);
            clientHandlers.add(clientHandler);
            new Thread(clientHandler).start();
         }
      } catch (IOException e) {
         System.out.println("Server error: " + e.getMessage());
         e.printStackTrace();
      }
   }
   // Broadcast message to all clients
   public static synchronized void broadcast(String message, ClientHandler sender) {
      for (ClientHandler client : clientHandlers) {
         if (client != sender) { // Don't send the message back to the sender
            client.sendMessage(message);
```

```
            }
        }
    }
    // Remove a client handler when a client disconnects
    public static synchronized void removeClient(ClientHandler clientHandler) {
        clientHandlers.remove(clientHandler);
    }
}
class ClientHandler implements Runnable {
    private Socket socket;
    private PrintWriter out;
    private String clientName;
    public ClientHandler(Socket socket) {
        this.socket = socket;
    }
    @Override
    public void run() {
        try (
            InputStream input = socket.getInputStream();
            OutputStream output = socket.getOutputStream();
            BufferedReader reader = new BufferedReader(new InputStreamReader(input));
        ) {
            out = new PrintWriter(output, true);
            // Ask the client for their name
            out.println("Enter your name:");
            clientName = reader.readLine();
            System.out.println(clientName + " has joined the chat.");
            ChatServer.broadcast(clientName + " has joined the chat.", this);
```

```
            String message;
            while ((message = reader.readLine()) != null) {
                System.out.println(clientName + ": " + message);
                ChatServer.broadcast(clientName + ": " + message, this);
            }
        } catch (IOException e) {
            System.out.println("Error handling client: " + e.getMessage());
        } finally {
            try {
                socket.close();
            } catch (IOException e) {
                System.out.println("Error closing socket: " + e.getMessage());
            }
            ChatServer.removeClient(this);
            ChatServer.broadcast(clientName + " has left the chat.", this);
            System.out.println(clientName + " has disconnected.");
        }
    }
    public void sendMessage(String message) {
        if (out != null) {
            out.println(message);
        }
    }
}
```

Client Program

```
import   java.io.*;
import java.net.*;
public class ChatClient {
    private static final String SERVER_ADDRESS = "127.0.0.1";
    private static final int SERVER_PORT = 9876;
    public static void main(String[] args) {
        try (Socket socket = new Socket(SERVER_ADDRESS, SERVER_PORT)) {
            System.out.println("Connected to the chat server.");
            new Thread(new ReadHandler(socket)).start();
            new Thread(new WriteHandler(socket)).start();
        } catch (IOException e) {
            System.out.println("Client error: " + e.getMessage());
        }
    }
}
class ReadHandler implements Runnable {
    private Socket socket;
    public ReadHandler(Socket socket) {
        this.socket = socket;
    }
    @Override
    public void run() {
        try (BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()))) {
            String message;
            while ((message = reader.readLine()) != null) {
                System.out.println(message);
            }
        } catch (IOException e) {
```

```
            System.out.println("Error reading from server: " + e.getMessage());
        }
    }
}
class WriteHandler implements Runnable {
    private Socket socket;
    public WriteHandler(Socket socket) {
        this.socket = socket;
    }
    @Override
    public void run() {
        try (
            BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));
            PrintWriter writer = new PrintWriter(socket.getOutputStream(), true)
        ) {
            String message;
            while ((message = consoleReader.readLine()) != null) {
                writer.println(message);
            }
        } catch (IOException e) {
            System.out.println("Error writing to server: " + e.getMessage());
        }
    }
}
```

EXPERIMENT-9

Implementation of DNS.

```
import java.net.*;
import java.util.Scanner;
public class SimpleDNSResolver {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Welcome to the Simple DNS Resolver!");
        System.out.println("Enter a domain name to resolve (or type 'exit' to quit):");
        while (true) {
            System.out.print("Domain: ");
            String domain = scanner.nextLine();
            if ("exit".equalsIgnoreCase(domain)) {
                System.out.println("Exiting DNS Resolver. Goodbye!");
                break;
            }
            try {
                InetAddress[] addresses = InetAddress.getAllByName(domain);
                System.out.println("IP addresses for " + domain + ":");
                for (InetAddress address : addresses) {
                    System.out.println(" - " + address.getHostAddress());
                }
            } catch (UnknownHostException e) {
                System.out.println("Could not resolve domain: " + e.getMessage());
            }
        }
        scanner.close();
    }
}
```

EXPERIMENT-10

Implementation of Ping service.

```
import java.net.*;
import java.io.*;
public class PingService {
   public static void main(String[] args) {
      // Check if the user provided a hostname as an argument
      if (args.length != 1) {
         System.out.println("Usage: java PingService <hostname>");
         return;
      }
      String hostname = args[0]; // Get the hostname from command-line arguments
      try {
         System.out.println("Pinging " + hostname + "...");
         InetAddress inetAddress = InetAddress.getByName(hostname);
         boolean isReachable = inetAddress.isReachable(5000);
         if (isReachable) {
            System.out.println("Host " + hostname + " is reachable.");
            System.out.println("IP Address: " + inetAddress.getHostAddress());
         } else {
            System.out.println("Host " + hostname + " is not reachable.");
         }
      } catch (UnknownHostException e) {
         System.out.println("Unknown host: " + hostname);
      } catch (IOException e) {
         System.out.println("Error occurred while pinging " + hostname + ": " + e.getMessage());
      }
   }
}
```