# KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

**(AN AUTONOMOUS INSTITUTION)**

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


## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# LAB MANUAL

# COMPUTER NETWROKS

# LAB

**B.Tech. II YEAR II SEM (KR21)**

# INDEX

# KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

**(AN AUTONOMOUS INSTITUTE)**

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

## Department of Computer Science & Engineering

**Vision of the Institution:**

To be the fountain head 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 Computer Science & Engineering

**Vision of the Department**:

To be among the region's premier teaching and research Computer Science and Engineering departments producing globally competent and socially responsible graduates in the most conducive academic environment.

**Mission of the Department**:

- To provide faculty with state of the art facilities for continuous professional development and research, both in foundational aspects and of relevance to emerging computing trends.
- To impart skills that transform students to develop technical solutions for societal needs and inculcate entrepreneurial talents.
- To inculcate an ability in students to pursue the advancement of knowledge in various specializations of Computer Science and Engineering and make them industry-ready.
- To engage in collaborative research with academia and industry and generate adequate resources for research activities for seamless transfer of knowledge resulting in sponsored projects and consultancy.
- To cultivate responsibility through sharing of knowledge and innovative computing solutions that benefits the society-at-large.
- To collaborate with academia, industry and community to set high standards in academic excellence and in fulfilling societal responsibilities.

## Department of Computer Science & Engineering

**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.

# KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

**(AN AUTONOMOUS INSTITUTE)**

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

**Department of Computer Science & Engineering**
**III Year I Semester Course Syllabus (KR21)**
**COMPUTER NETWORKS LAB(CS507PC)**

| L | T | P | C |
|---|---|---|---|
| 0 | 0 | 3 | 1.5 |

**Prerequisite/ Corequisites:**

1. CS203ES - Programming for problem solving Course.
2. CS401PC - Java Programming 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 socketsand 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.

**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, PearsonEducation.
3. Understanding communications and Networks- W.A. Shay, CengageLearning 3rd Edition.
4. ComputerNetworking: ATop-DownApproachFeaturingtheInternet-JamesF.Kurose&Keith W. Ross, 3rd Edition, Pearson Education.
5. DataandComputerCommunication-William Stallings,PearsonEducation,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, Narayanguda, Hyderabad – 500029**

## Department of Computer Science & Engineering

## Course Outcomes and CO-PO-PSO Mapping

### Course Objectives

1. To understand the working principle of various communication protocols.
2. To understand the network simulator environment and visualize a network topology and observe its performance
3. To analyze the traffic flow and the contents of protocol frames

### Course Outcomes

1. Implement data link layer farming methods
2. Analyze error detection and error correction codes.
3. Implement and analyze routing and congestion issues in network design.
4. Implement Encoding and Decoding techniques used in presentation layer
5. To be able to work with different network tools

**CO -PO MAPPING:**

| | PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CO1 | 1 | 2 | | 3 | | | | | | | | |
| CO2 | | 1 | | 2 | | | | | | | | |
| CO3 | | 2 | 1 | 2 | 3 | | | | | | | |
| CO4 | | 1 | | | 2 | | | | | | | |
| CO5 | 1 | 1 | | 2 | | | | | | | | |
| Avg | 1 | 1 | 1 | 2 | 3 | | | | | | | |

**CO-PSO Matrix:**

| | PSO1 | PSO2 |
|---|---|---|
| **CO1** | **1** | |
| **CO2** | | |
| **CO3** | | **2** |
| **CO4** | | |
| **CO5** | **2** | |
| **Average** | **1.5** | **2** |

Experiment 1: **Implement on a data set of characters the CRC .**

**AIM:**

**Program:**

```
#include<stdlib.h>
#include<conio.h>
#include<stdio.h>
void main()
{
int i,j,n,g,a,arr[20],gen[20],b[20],q[20],s;
clrscr();
printf("Transmitter side:");
printf("\nEnter no. of data bits:");
scanf("%d",&n);
printf("Enter data:");
for(i=0;i<n;i++)
scanf("%d",&arr[i]);
printf("Enter size of generator:");
scanf("%d",&g);
do{
printf("Enter generator:");
for(j=0;j<g;j++)
scanf("%d",&gen[j]);
}
while(gen[0]!=1);
printf("\n\tThe generator matrix:");
for(j=0;j<g;j++)
printf("%d",gen[j]);
a=n+(g-1);
printf("\n\tThe appended matrix is:");
for(i=0;i< j;++i)
arr[n+i]=0;
for(i=0;i< a;++i)
printf("%d",arr[i]);
for(i=0;i< n;++i)
q[i]= arr[i];
for(i=0;i< n;++i)
{
if(arr[i]==0)
{
for(j=i;j<g+i;++j)
arr[j] = arr[j]^0;
}
else
{
arr[i] = arr[i]^gen[0];
arr[i+1]=arr[i+1]^gen[1];
arr[i+2]=arr[i+2]^gen[2];
arr[i+3]=arr[i+3]^gen[3];
} }
printf("\n\tThe CRC is :");
for(i=n;i< a;++i)
printf("%d",arr[i]);
s=n+a;
for(i=n;i<s;i++)
q[i]=arr[i];
```

```
printf("\n");
for(i=0;i<a;i++)
printf("%d",q[i]);
getch();
}
```

**Output:**

```
Transmitter side:
Enter no. of data bits:8
Enter data:1 0 1 0 0 0 0 1
Enter size of generator:4
Enter generator:1 0 0 1
The generator matrix:1001
The appended matrix is:10100001000
The CRC is :111
10100001111
```

**Implement CRC 12 bit**
**AIM:**
**Program:**

```
#include<stdio.h>
const char * bindiv(const char *,const char *);
const char * binsub(const char *,const char *);
int f=0,ll=0;
void main()
{
char *a,p[13]="1100000001011",g[30],g1[30],yy[30]="",td[30],*aa;
int l=0,i;
clrscr();
printf("Enter transfered data : ");
scanf("%s",g);
printf("Enter received data : ");
scanf("%s",td);
strcpy(g1,g);
strcat(g,"000000000000");
printf("\n%s ) %s (",p,g);
a=bindiv(g,p);
if(strlen(a)<12)
{
for(i=strlen(a);i<12;i++)
            {
           yy[l++]='0';
            }
yy[l]='\0';
}
strcat(yy,a);
strcat(g1,yy);
printf("\ncrc is          %s",yy);
printf("\n               ------------------");
strcat(td,yy);
printf("\n\n%s ) %s (",p,td);
ll=0;
aa=bindiv(td,p);
strcpy(a,aa);
printf("\n               %s",a);
printf("\n               -----------------");
if(f==1)
printf("\nDatatransfered correctly");
else
printf("\nDatatransfered incorrectly");
getch();
}
const char * bindiv(const char *s,const char *d)
{
int i,j,k=0,x=13,h,p=0,l;
char q[15]="",b[30],*w;
for(i=0;i<strlen(s);i++)
{
           if((i+x)>strlen(s))
              x=(i+x)-strlen(s)+1;
           for(j=i;j<(i+x);j++)
           {              b[k++]=s[j];              }
```

```
b[k]='\0';
if(ll!=0)
        printf("\n              %s",b);
ll=1;
if(strlen(b)==12)
          {
            break;
          }
        printf("\n              %s",d);
          printf("\n              ----------");
w=binsub(b,d);
k=0;i=j-1;
for(l=0;l<strlen(w);l++)
          {
            if(w[l]=='1')
              break;
          }
if(l==strlen(w))
          {
           f=1;
           return(w);
          }
for(h=l;h<strlen(w);h++)
        {
          q[p++]=w[h];
        }
q[p]='\0';
x=13-strlen(q);
strcpy(b,"");
strcat(b,q);
k=strlen(q); p=0;
}
return(b);
}
const char * binsub(const char *x,const char *y)
{
int i,j=0;
char w[15]="",e[3],f[3],n[3];
e[0]='1';
e[1]='\0';
f[0]='0';
f[1]='\0';
for(i=0;i<strlen(x);i++)
{
          if((x[i]=='1')&&(y[i]=='1'))
          strcat(w,f);
           else
            if((x[i]=='0')&&(y[i]=='0'))
          strcat(w,f);
           else
          strcat(w,e);
}
n[0]='\0';
n[1]='\0';
strcat(w,n);
```

```
return(w);
 }
```
**Output:**

```
Enter transferred data: 11011  , Enter received data: 11011
10001000000100001 ) 11011000000000000000 (
10001000000100001
-------------------------
10100000001000010
10001000000100001
-------------------------
10100000110001100
10001000000100001
-------------------------
1010001101011010
-------------------------
crc is    1010001101011010
10001000000100001 ) 11011101000110101101 0 (
10001000000100001
-------------------------
10101010000101001
10001000000100001
-------------------------
10001000000100001
10001000000100001
-------------------------
00000000000000000
-------------------------  Data transferred correctly
```

**Implement on a data set of characters the CRC Polynomial CRC 16**
**AIM:**
**Program:**

```
#include<stdio.h>
const char * bindiv(const char *,const char *);
const char * binsub(const char *,const char *);
int f=0,ll=0;
void main()
{
char *a,p[20]="10001000000100001",
g[30],g1[30],yy[30]="",td[30],*aa;
int l=0,i;
clrscr();
printf("Enter transfered data: ");
scanf("%s",g);
printf("Enter received data: ");
scanf("%s",td);
strcpy(g1,g);
strcat(g,"0000000000000000");
printf("\n%s ) %s (",p,g);
a=bindiv(g,p);
if(strlen(a)<16)
{
for(i=strlen(a);i<16;i++)
            {
           yy[l++]='0';
            }
yy[l]='\0';
}
strcat(yy,a);
strcat(g1,yy);
printf("\n                  ----------");
printf("\ncrc is              %s",yy);
strcat(td,yy);
printf("\n\n%s ) %s (",p,td);
ll=0;
aa=bindiv(td,p);
strcpy(a,aa);
printf("\n                  %s",a);
printf("\n                  ----------");
if(f==1)
printf("\nDatatransfered correctly");
else
printf("\nDatatransfered incorrectly");
getch();
}
const char * bindiv(const char *s,const char *d)
{
int i,j,k=0,x=17,h,p=0,l;
char q[25]="",b[30],*w;
for(i=0;i<strlen(s);i++)
{
          if((i+x)>strlen(s))
             x=(i+x)-strlen(s)+1;
```

```
            for(j=i;j<(i+x);j++)
              {
                b[k++]=s[j];
              }
b[k]='\0';
if(ll!=0)
            printf("\n                    %s",b);
ll=1;
if(strlen(b)==16)
             {
               break;
             }
            printf("\n                    %s",d);
              printf("\n                    ----------");
w=binsub(b,d);
k=0;i=j-1;
for(l=0;l<strlen(w);l++)
             {
               if(w[l]=='1')
                  break;
             }
if(l==strlen(w))
{
              f=1;
              return(w);
             }
for(h=l;h<strlen(w);h++)
            {
              q[p++]=w[h];
            }
q[p]='\0';
x=17-strlen(q);
strcpy(b,"");
strcat(b,q);
k=strlen(q); p=0;
}
return(b);
}
const char * binsub(const char *x,const char *y)
{
int i,j=0;
char w[25]="",e[3],f[3],n[3];
e[0]='1';
e[1]='\0';
f[0]='0';
f[1]='\0';
for(i=0;i<strlen(x);i++)
{
             if((x[i]=='1')&&(y[i]=='1'))
            strcat(w,f);
             else
              if((x[i]=='0')&&(y[i]=='0'))
            strcat(w,f);
             else
            strcat(w,e);
```

```
}
n[0]='\0';
n[1]='\0';
strcat(w,n);    return(w);
}
```

**Output:**

```
Enter transferred data: 11011 , Enter received data: 11011
10001000000100001) 11011000000000000000 (
10001000000100001
-------------------------
10100000001000010
10001000000100001
-------------------------
10100000110001100
10001000000100001
-------------------------
1010001101011010
-------------------------
crc is            1010001101011010
10001000000100001) 11011101000110101101 0 (
10001000000100001
-------------------------
10101010000101001
10001000000100001
-------------------------
10001000000100001
10001000000100001
-------------------------
00000000000000000
------------------------- Data transferred correctly
```

**Implement program for CRC-CCITT**

**AIM:**

**Program:**

```
#include<stdio.h>
const char * bindiv(const char *,const char *);
const char * binsub(const char *,const char *);
int f=0,ll=0;
void main()
{
char *a,p[20]="10001000000100001 ",
```

```
g[30],g1[30],yy[30]="",td[30],*aa;
int l=0,i;
clrscr();
printf("Enter transfered data: ");
scanf("%s",g);
printf("Enter received data: ");
scanf("%s",td);
strcpy(g1,g);
strcat(g,"0000000000000000");
printf("\n%s ) %s (",p,g);
a=bindiv(g,p);
if(strlen(a)<16)
{
for(i=strlen(a);i<16;i++)
            {
            yy[l++]='0';
             }
yy[l]='\0';
}
strcat(yy,a);
strcat(g1,yy);
printf("\n                  ------------------");
printf("\ncrc is              %s",yy);
strcat(td,yy);
printf("\n\n%s ) %s (",p,td);
ll=0;
aa=bindiv(td,p);
strcpy(a,aa);
printf("\n                  %s",a);
printf("\n                  ------------------");
if(f==1)
printf("\nDatatransfered correctly");
else
printf("\nDatatransfered incorrectly");
getch();
}
const char * bindiv(const char *s,const char *d)
{
int i,j,k=0,x=17,h,p=0,l;
char q[25]="",b[30],*w;
for(i=0;i<strlen(s);i++)
{
            if((i+x)>strlen(s))
                x=(i+x)-strlen(s)+1;
            for(j=i;j<(i+x);j++)
              {
                b[k++]=s[j];
              }
b[k]='\0';
if(ll!=0)
            printf("\n                    %s",b);
ll=1;
if(strlen(b)==16)
            {
              break;
```

```
            }
           printf("\n                %s",d);
             printf("\n                  ----------");
w=binsub(b,d);
k=0;i=j-1;
for(l=0;l<strlen(w);l++)
           {
             if(w[l]=='1')
               break;
           }
if(l==strlen(w))
{
            f=1;
            return(w);
           }
for(h=l;h<strlen(w);h++)
           {
            q[p++]=w[h];
           }
q[p]='\0';
x=17-strlen(q);
strcpy(b,"");
strcat(b,q);
k=strlen(q); p=0;
}
return(b);
}
const char * binsub(const char *x,const char *y)
{
int i,j=0;
char w[25]="",e[3],f[3],n[3];
e[0]='1';
e[1]='\0';
f[0]='0';
f[1]='\0';
for(i=0;i<strlen(x);i++)
{
            if((x[i]=='1')&&(y[i]=='1'))
           strcat(w,f);
            else
             if((x[i]=='0')&&(y[i]=='0'))
           strcat(w,f);
            else
           strcat(w,e);
}
n[0]='\0';
n[1]='\0';
strcat(w,n);
return(w);
}
```

**Output:**

```
Enter transferred data: 11011 , Enter received data: 11011
10001000000100001) 110110000000000000000 (
10001000000100001
-------------------------
```

```
10100000001000010
10001000000100001
-------------------------
10100000110001100
10001000000100001
-------------------------
1010001101011010
-------------------------
crc is            1010001101011010
10001000000100001) 110111010001101011010 (
10001000000100001
-------------------------
10101010000101001
10001000000100001
-------------------------
10001000000100001
10001000000100001
-------------------------
00000000000000000
------------------------- Data transferred correctly
```

Experiment 2 :

**A)Develop a simple data link layer that performs the flow control using the sliding window protocol**

**AIM:**

**Program:**

```
#include<stdio.h>
 int main()
{
   int w,i,f,frames[50];
printf("Enter window size: ");
scanf("%d",&w);
printf("\nEnter number of frames to transmit: ");
scanf("%d",&f);
printf("\nEnter %d frames: ",f);
    for(i=1;i<=f;i++)
scanf("%d",&frames[i]);
printf("\nWith sliding window protocol the frames will be sent in the following manner (assuming no corruption of frames)\n\n");
printf("After sending %d frames at each stage sender waits for acknowledgement sent by the receiver\n\n",w);
    for(i=1;i<=f;i++)
   {
      if(i%w==0)
      {
printf("%d\n",frames[i]);
printf("Acknowledgement of above frames sent is received by sender\n\n");
}       else
printf("%d ",frames[i]);
   }
    if(f%w!=0)
printf("\nAcknowledgement of above frames sent is received by sender\n");
    return 0; }
```

**Output:**

```
Enter window size: 3
Enter number of frames to transmit: 5
Enter 5 frames: 12 5 89 4 6
With sliding window protocol the frames will be sent in the following manner (assuming no corruption of frames)
After sending 3 frames at each stage sender waits for acknowledgement sent by the receiver
12 5 89
Acknowledgement of above frames sent is received by sender
4 6
Acknowledgement of above frames sent is received by sender
```

**B. Develop a simple data link layer that performs the flow control using the Go Back N protocol in c**

**AIM:**

**Program:**

```
#include<stdio.h>
int main()
{
        int windowsize,sent=0,ack,i;
        printf("enter window size\n");
        scanf("%d",&windowsize);
        while(1)
        {
                for( i = 0; i<windowsize; i++)
                        {
                        printf("Frame %d has been transmitted.\n",sent);
                                sent++;
                                if(sent == windowsize)
                                        break;
                        }
        printf("\nPlease enter the last Acknowledgement received.\n");
                        scanf("%d",&ack);
                        if(ack == windowsize)
                                break;
                        else
                                sent = ack;
        }
return 0;
}
```

**Output:**

```
enter window size
8
Frame 0 has been transmitted.
Frame 1 has been transmitted.
Frame 2 has been transmitted.
Frame 3 has been transmitted.
Frame 4 has been transmitted.
Frame 5 has been transmitted.
Frame 6 has been transmitted.
Frame 7 has been transmitted.
Please enter the last Acknowledgement received.
2
Frame 2 has been transmitted.
Frame 3 has been transmitted.
Frame 4 has been transmitted.
Frame 5 has been transmitted.
```

Frame 6 has been transmitted.
Frame 7 has been transmitted.

Please enter the last Acknowledgement received.
8
---

Experiment 3 :

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

**AIM:**

**Program:**

```
#include <stdio.h>
int max();
int distance[20];
int n;

main()
{
 int adj[20][20],adj1[20][20],flag[30];
 int i,j,root,x;
 int source,count=1,y=0;
printf("enter no of nodes");
scanf("%d",&n);

 for(i=0;i<n;i++)
  {
   for(j=0;j<n;j++)
    {
scanf("%d",&adj[i][j]);
     }
    }

printf("enter the source for broadcasting");
scanf("%d",&source);

  for(i=0;i<n;i++)
   {
    flag[i]=0;
    }

for(root=0;root<n;root++)
   {
    for(i=0;i<N;I++)
     {
      distance[i]=adj[root][i];
      }
      x=min();

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

     if(distance[i]==x)
             {
              adj1[root][i]=x;
              adj1[i][root]=x;
              }
            else
             {
              adj1[root][i]=0;
```

```
            }
           }
    }

    for(i=0;i<n;i++)
      {
       for(j=0;j<n;j++)
       {
            if(adj1[i][j]!=0)
            {
             adj1[j][i]=adj[i][j];
             }
       }
      }

printf("given adjacency matrix is");

 for(i=0;i<n;i++)
      {
       for(j=0;j<n;j++)
            {
            printf("%d",adj[i][j]);
             }
            printf("\n");
       }
printf("minimal spanning tree");

 for(i=0;i<n;i++)
      {
     for(j=0;j<n;j++)
       {
            printf("%d ",adj1[i][j]);
            }
            printf("\n");
      }
      root=source;
      flag[root]=1;
     while(count!=y)
   {
            for(i=0;i<n;i++)
             {
              if(adj1[root][i]!=0 && flag[root]==1 && flag[i]!=1)
               {
            printf("%d sends message to %d \n",root,i);
                flag[i]=1;
               }
              }
             if(root<n-1)
              {
               root++;
               }
               else
```

```
            {
             root=0;
             }
           for(i=0;i<n;i++)
            {
             if(flag[i]==0)
              {
               break;
              }
   }
           if(i==n)
            {
             count=y;
             }
    }
  }

int min()
  {
     int i,j=0;
     int mini;
     int distance1[10];
     for(i=0;i<n;i++)
     {
          if(distance[i]!=0)
           {
            distance1[j]=distance[i];
          j++;
           }
      }
     mini=distance1[0];

    for(i=1;i<j;i++)
         {
           if(distance1[i]<mini)
            {
             mini=distance1[i];
            }
          }
           return(mini);
   }
```

**Output:**

```
enter no of nodes2
enter the adjacency matrix
0 2
2 0
enter the source for broadcasting1
given adjacency matrix is
```

0 2
2 0

minimal spanning tree is
0 2
2 0
1 sends message to 0

enter no of nodes3
enter the adjacency matrix
0 1 2
1 0 5
2 5 0
enter the source for broadcasting2
given adjacency matrix is
0 1 2
1 0 5
2 5 0

1

2 5

minimal spanning tree is
0 1 2
1 0 0
2 0 0

1
2

2 sends message to 0
0 sends message to 1

enter no of nodes4
enter the adjacency matrix
0 6 8 7
6 0 5 0
8 5 0 4
7 0 4 0
enter the source for broadcasting2

given adjacency matrix is
0 6 8 7
6 0 5 0
8 5 0 4
7 0 4 0

4

7 8 5

6

minimal spanning tree is
0 6 0 0
6 0 5 0
0 5 0 4
0 0 4 0

4

5

6

2 sends message to 1
2 sends message to 3
1 sends message to 0

Experiment 4 :

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

**AIM:**

**Program:**

```
#include <stdio.h>
int ja,ji,jh,jk;
void calc( );
int repa[20],repi[20],reph[20],repk[20],rsp[20];
char line[20];
int main( )
{
int i=0;
printf("Enter a delay time\n");
scanf("%d %d %d %d",&ja,&ji,&jh,&jk);
printf("Enter response time\n");
for(i=0;i<12;i++)
{
printf("Enter value \n");
scanf("%d %d %d %d",&repa[i],&repi[i],&reph[i],&repk[i]);
}
calc( );
printf("Least response\n");
for(i=0;i<12;i++)
{
printf("%d\t%c\n",rsp[i],line[i]);
}}
void calc( )
{
int i;
for(i=0;i<12;i++)
{
repa[i]=repa[i]+ja;
repi[i]=repi[i]+ji;
reph[i]=reph[i]+jh;
repk[i]=repk[i]+jk;
}
for(i=0;i<12;i++)
{
if(i==9)
{
rsp[i]=0;
line[i]='J';
}
else
{
rsp[i]=min(repa[i],repi[i],reph[i],repk[i]);
if(rsp[i]==repa[i])
line[i]='A';
else if(rsp[i]==repi[i])
line[i]='I';
else if(rsp[i]==reph[i])
line[i]='H';
```

```
else
line[i]='K';
}
}
}
int min(int repa,intrepi,intreph,intrepk)
{
int i,j;
if(repa<repi)
i=repa;
else
i=repi;
if(reph<repk)
j=reph;
else
j=reph;
if(i<j)
return i;
else
return j;
}
```

**Output:**

```
Enter a delay time
8
10
12
6
Enter response time
Enter value
0
24
20
21
Enter value
12
36
31
28
Enter value
25
18
19
36
Enter value
40
27
8
24
Enter value
14
7
30
```

22
Enter value
23
20
19
40
Enter value
18
31
6
31
Enter value
17
20
0
19
Enter value
21
0
14
22
Enter value
9
11
7
10
Enter value
24
22
22
0
Enter value
29
33
9
9
Least response
8 A
20 A
28 I
20 H
17 I
30 I
18 H
12 H
10 I
J
32 A
21 H

**Experiment 5:**
**Design the following**
a. TCP iterative Client and server application to reverse the given input sentence.

Server Program

```
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/socket.h>
#include<sys/types.h>
#define MAXLINE 20
#define SERV_PORT 5777
main(int argc,char *argv)
{
int i,j;
ssize_t n;
char line[MAXLINE];
char revline[MAXLINE];
int listenfd,connfd,clilen;
struct sockaddr_in servaddr,cliaddr;
listenfd=socket(AF_INET,SOCK_STREAM,0);
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET; servaddr.sin_port=htons(SERV_PORT);
bind(listenfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
listen(listenfd,1);
for( ; ; )
{
clilen=sizeof(cliaddr);
connfd=accept(listenfd,(struct sockaddr*)&cliaddr,&clilen);
printf("connect to client");
while(1)
{
if((n=read(connfd,line,MAXLINE))==0)
break;
line[n-1]='\0';
j=0;
for(i=n-2;i>=0;i--)
revline[j++]=line[i];
revline[j]='\0';
write(connfd,revline,n);
}
}
}
```

Client Program

```
#include<string.h>
#include<stdio.h>
#include<stdlib.h>
```

```
#include<unistd.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<sys/types.h>
#define MAXLINE 20
#define SERV_PORT 5777
main(int argc,char *argv)
{
char sendline[MAXLINE],revline[MAXLINE];
int sockfd;
struct sockaddr_in servaddr;
sockfd=socket(AF_INET,SOCK_STREAM,0);
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_port=ntohs(SERV_PORT);
connect(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
printf("\n enter the data to be send");
while(fgets(sendline,MAXLINE,stdin)!=NULL)
{
write(sockfd,sendline,strlen(sendline));
printf("\n line send");
read(sockfd,revline,MAXLINE);
printf("\n reverse of the given sentence is : %s",revline);
printf("\n");
}
exit(0);
}
```

Output

```
enter the data to be sendhellow
line send
reverse of the given sentence is : wolleh
```

5 B) TCP client and server application to transfer file.

Source program

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
int main(void)
{
 int listenfd = 0;
 int connfd = 0;
 struct sockaddr_in serv_addr;
 char sendBuff[1025];
 int numrv;
 listenfd = socket(AF_INET, SOCK_STREAM, 0);
 printf("Socket retrieve success\n");
 memset(&serv_addr, '0', sizeof(serv_addr));
 memset(sendBuff, '0', sizeof(sendBuff));
```

```
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(5000);
```

[Type text] [Type text] [Type text]

```
bind(listenfd, (structsockaddr*)&serv_addr,sizeof(serv_addr));
if(listen(listenfd, 10) == -1)
{
printf("Failed to listen\n");
return -1;
}
while(1)
{
connfd = accept(listenfd, (struct sockaddr*)NULL ,NULL);
/* Open the file that we wish to transfer */
FILE *fp = fopen("fifoserver.c","rb");
if(fp==NULL)
{
printf("File opern error");
return 1;
}
/* Read data from file and send it */
while(1)
```

```
 {
 /* First read file in chunks of 256 bytes */
```

[Type text] [Type text] [Type text]

```
 unsigned char buff[256]={0};
 int nread = fread(buff,1,256,fp);
 printf("Bytes read %d \n", nread);
 /* If read was success, send data. */
 if(nread > 0)
 {
 printf("Sending \n");
 write(connfd, buff, nread);
 }
```

OutPut:

Client Program:

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>
int main(void)
{
```

```
int sockfd = 0;
int bytesReceived = 0;
char recvBuff[256];
memset(recvBuff, '0', sizeof(recvBuff));
struct sockaddr_in serv_addr;
/* Create a socket first */
if((sockfd = socket(AF_INET, SOCK_STREAM, 0))< 0)
{
printf("\n Error : Could not create socket \n");
return 1;
}
/* Initialize sockaddr_in data structure */
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(5000); // port
serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
/* Attempt a connection */
if(connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr))<0)
{
printf("\n Error : Connect Failed \n");
```



```
return 1;
}
/* Create file where data will be stored */
FILE *fp;
fp = fopen("fifoserver.c","ab");
if(NULL == fp)
{
printf("Error opening file");
return 1;
}
/* Receive data in chunks of 256 bytes */
while((bytesReceived = read(sockfd, recvBuff, 256)) > 0)
```

```
 {
 printf("Bytes received %d\n",bytesReceived);
 // recvBuff[n] = 0;
 fwrite(recvBuff, 1,bytesReceived,fp);
 // printf("%s \n", recvBuff);
 }
 if(bytesReceived < 0)
 {
 printf("\n Read Error \n");
 }
 return 0;
}
```

Out put

Server side

Socket retrieve success

Bytes read 256

Sending

Bytes read 256

Sending

Bytes read 256

Sending

Bytes read 28

Sending

End of file

Client side

Bytes received 256

Bytes received 256

Bytes received 256

Bytes received 28

5 C) TCP concurrent server to convert a given text into upper case using multiplexing system call “select”.

Server program:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/select.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#define MAXLINE 100
#define SERV_PORT 13153
int main(int argc, char **argv)
{
int k, i, maxi, maxfd, listenfd, connfd, sockfd;
int nready, client[FD_SETSIZE];
ssize_t n;
fd_set rset, allset;
char line[MAXLINE],buf[100];
socklen_t clilen;
struct sockaddr_in cliaddr, servaddr;
listenfd = socket(AF_INET, SOCK_STREAM, 0);
if (listenfd < 0 )
{
perror("socket" );
exit(1);
```

```
}
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(SERV_PORT);
bind(listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr));
listen(listenfd,5);
maxfd = listenfd; /* initialize */
maxi = -1; /* index into client[] array */
for (i = 0; i < FD_SETSIZE; i++)
client[i] = -1; /* -1 indicates available entry */
FD_ZERO(&allset);
FD_SET(listenfd, &allset);
/* end fig01 */
/* include fig02 */
for ( ; ; ) {
printf("Server:I am waiting-----Start of Main Loop\n");
rset = allset; /* structure assignment */
nready = select(maxfd+1, &rset, NULL, NULL, NULL);
if (FD_ISSET(listenfd, &rset)) { /* new client connection */
clilen = sizeof(cliaddr);
connfd = accept(listenfd, (struct sockaddr *) &cliaddr, &clilen);

#ifdef NOTDEF
printf("new client: %s, port %d\n",
inet_ntop(AF_INET, &cliaddr.sin_addr, buf, NULL),
ntohs(cliaddr.sin_port));
#endif
for (i = 0; i < FD_SETSIZE; i++)
if (client[i] < 0) {
client[i] = connfd; /* save descriptor */
break;
```

```
}
if (i == FD_SETSIZE)
 {
 printf("too many clients");
 exit(0);
 }
FD_SET(connfd, &allset); /* add new descriptor to set */
if (connfd > maxfd)
maxfd = connfd; /* for select */
if (i > maxi)
maxi = i; /* max index in client[] array */
if (--nready <= 0)
continue; /* no more readable descriptors
*/
}
for (i = 0; i <= maxi; i++) { /* check all clients for data */
if ( (sockfd = client[i]) < 0)
continue;
if (FD_ISSET(sockfd, &rset)) {
if ( (n = read(sockfd, line, MAXLINE)) == 0) {
/*4connection closed by client */
close(sockfd);
FD_CLR(sockfd, &allset);
client[i] = -1;
} else
{
printf("\n output at server\n");
for(k=0;line[k]!='\0';k++)
printf("%c",toupper(line[k]));
write(sockfd, line, n);
}
if (--nready <= 0)
```

```
break; /* no more readable descriptors
*/
}
}
}
}
/
```

Client program :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#define MAXBUFFER 1024
void sendstring(int , char *);
int main( int C, char *V[] )
{
int sd,fd;
char c;
struct sockaddr_in serveraddress;
char text[100];
int i=0;
sd = socket( AF_INET, SOCK_STREAM, 0 );
if( sd < 0 ) {
perror( "socket" );
exit( 1 );
```

```
}
if (V[1] == NULL ) {
printf ("PL specfiy the server's IP Address \n");
exit(0);
}
if (V[2] == NULL ) {
printf ("PL specify the server's Port No \n");
exit(0);
}
// if (V[3] == NULL ) {
// printf ("PL specfiy the string to be send to the server \n");
// exit(0);
// }
memset( &serveraddress, 0, sizeof(serveraddress) );
serveraddress.sin_family = AF_INET;
serveraddress.sin_port = htons(atoi(V[2]));//PORT NO
serveraddress.sin_addr.s_addr = inet_addr(V[1]);//ADDRESS
if (connect(sd,(struct sockaddr*)&serveraddress,
sizeof(serveraddress))<0)
{
printf("Cannot Connect to server");
exit(1);
}
printf("enter sentence to end enter #");
while(1)
{
c=getchar();
if(c=='#')
break;
text[i++]=c;
}
text[i]='\0';
```

```
sendstring(sd,text);
close(sd);
return 0;
}
/*************************************************************************
* FUNCTION NAME:sendstring
* DESCRIPTION: sends a string over the socket .
* NOTES : No Error Checking is done .
* RETURNS :void
*************************************************************************/
void sendstring(
int sd, /*Socket Descriptor*/
char *fname) /*Array Containing the string */
/*************************************************************************/
{ int n , byteswritten=0 , written ;
char buffer[MAXBUFFER];
strcpy(buffer , fname);
n=strlen(buffer);
while (byteswritten<n)
{
written=write(sd , buffer+byteswritten,(n-byteswritten));
byteswritten+=written;
29
}
printf("String : %s sent to server \n",buffer);
}
```

Execution Steps:

b) Concurrent Server Application Using Select.

Compiling and running server.

```
root@localhost week7and8]# cc tcpservselect01.c
[root@localhost week7and8]# mv a.out tcpservselect1
[root@localhost week7and8]# ./tcpservselect1
```

Server:I am waiting-----Start of Main Loop

Server:I am waiting-----Start of Main Loop

output at server

A B C DServer:I am waiting-----Start of Main Loop

output at server

A B C DServer:I am waiting-----Start of Main Loop

Server:I am waiting-----Start of Main Loop

Compiling and running Client.

root@localhost week7and8]# ./tcpclient 127.0.0.1 13153

enter sentence to end enter #abcd#

String : abcd sent to server

5 D) TCP concurrent server to echo given set of sentences using poll functions.

server program

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <limits.h> /* for OPEN_MAX */
#include <poll.h>
#include <errno.h>
#define MAXLINE 100
#define SERV_PORT 13154
#define POLLRDNORM 5
#define INFTIM 5
#define OPEN_MAX 5
int main(int argc, char **argv)
{
int k, i, maxi, listenfd, connfd, sockfd;
int nready;
ssize_t n;
char line[MAXLINE];
socklen_t clilen;
struct pollfd client[OPEN_MAX];
struct sockaddr_in cliaddr, servaddr;
listenfd = socket(AF_INET, SOCK_STREAM, 0);
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(SERV_PORT);
bind(listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr));
listen(listenfd, 5);
client[0].fd = listenfd;
```

```
client[0].events = POLLRDNORM;
for (i = 1; i < OPEN_MAX; i++)
client[i].fd = -1; /* -1 indicates available entry */
maxi = 0; /* max index into client[] array */
/* end fig01 */
/* include fig02 */
for ( ; ; ) {
nready = poll(client, maxi+1, INFTIM);
if (client[0].revents & POLLRDNORM) { /* new client connection */
clilen = sizeof(cliaddr);
connfd = accept(listenfd, (struct sockaddr *) &cliaddr, &clilen);
#ifdef NOTDEF
printf("new client: %s\n", sock_ntop((struct sockaddr *) &cliaddr,
clilen));
#endif
for (i = 1; i < OPEN_MAX; i++)
if (client[i].fd < 0) {
client[i].fd = connfd; /* save descriptor */
break;
}
if (i == OPEN_MAX)
{
printf("too many clients");
exit(0);
}
client[i].events = POLLRDNORM;
if (i > maxi)
maxi = i; /* max index in client[] array */
if (--nready <= 0)
continue; /* no more readable descriptors
*/
}
for (i = 1; i <= maxi; i++) { /* check all clients for data */
if ( (sockfd = client[i].fd) < 0)
continue;
if (client[i].revents & (POLLRDNORM | POLLERR)) {
if ( (n = read(sockfd, line, MAXLINE)) < 0) {
if (errno == ECONNRESET) {
```

```
/*4connection reset by client */
#ifdef NOTDEF
printf("client[%d] aborted connection\n", i);
#endif
close(sockfd);
client[i].fd = -1;
} else
printf("readline error");
} else if (n == 0) {
/*4connection closed by client */
#ifdef NOTDEF
printf("client[%d] closed connection\n", i);
#endif
close(sockfd);
client[i].fd = -1;
} else{ printf("\n data from client is \n");
 k=strlen(line);
printf(" length=%d data = %s\n", k,line);
//write(sockfd, line, n);

 strcpy(line," ");
}
if (--nready <= 0)
break; /* no more readable descriptors
*/
}
}
}
}
```

client program:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
```

```
#include <sys/stat.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#define MAXBUFFER 1024
void sendstring(int , char *);
int main( int C, char *V[] )
{
int sd,fd;
char c;
struct sockaddr_in serveraddress;
char text[100];
int i=0;
sd = socket( AF_INET, SOCK_STREAM, 0 );
if( sd < 0 ) {
perror( "socket" );
exit( 1 );
}
if (V[1] == NULL ) {
printf ("PL specfiy the server's IP Address \n");
exit(0);
}
if (V[2] == NULL ) {
printf ("PL specify the server's Port No \n");
exit(0);
}
// if (V[3] == NULL ) {
// printf ("PL specfiy the string to be send to the server \n");
// exit(0);
// }
memset( &serveraddress, 0, sizeof(serveraddress) );
serveraddress.sin_family = AF_INET;
serveraddress.sin_port = htons(atoi(V[2]));//PORT NO
serveraddress.sin_addr.s_addr = inet_addr(V[1]);//ADDRESS
if (connect(sd,(struct sockaddr*)&serveraddress,
sizeof(serveraddress))<0)
{
```

```
printf("Cannot Connect to server");
exit(1);
}
printf("enter sentence to end enter #");
while(1)
{
c=getchar();
if(c=='#')
break;
text[i++]=c;
}
text[i]='\0';
sendstring(sd,text);
close(sd);
return 0;
}
/*****************************************************************************
* FUNCTION NAME:sendstring
* DESCRIPTION: sends a string over the socket .
* NOTES : No Error Checking is done .
* RETURNS :void
*****************************************************************************/
void sendstring(
int sd, /*Socket Descriptor*/
char *fname) /*Array Containing the string */
/*****************************************************************************/
{ int n , byteswritten=0 , written ;
char buffer[MAXBUFFER];
strcpy(buffer , fname);
n=strlen(buffer);
while (byteswritten<n)
{
written=write(sd , buffer+byteswritten,(n-byteswritten));
byteswritten+=written;
}
printf("String : %s sent to server \n",buffer);
}
```

Concurrent Server Application Using Poll.

Compiling and running server.

```
[root@localhost week8]# cc tcpservpoll01.c
[root@localhost week8]# mv a.out pollserv
[root@localhost week8]# ./pollserv
data from client is
data = aaaaaaaaaaaaaaaaaaaaaa
Compiling and running Client.
[root@localhost week8]#cc democlient.c
[root@localhost week8]#mv a.out client
[root@localhost week8]# ./client 127.0.0.1 13153
enter sentence to end enter #aaaaaaaaaaaaaaaaaaaaaa#
String : aaaaaaaaaaaaaaaaaaaaaa sent to server
```

---

## Experiment 6 :

**Design the following**

**6 A) UDP Client and server application to reverse the given input sentence. 2018-2019 173.**

**Server program :**

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <signal.h>
#define BUFSIZE 512
#define MYPORT 11710
#define MAXNAME 100
int main(int C, char **V )
{
int sd,n,ret;
struct sockaddr_in
serveraddress,cliaddr;
socklen_t length;
char clientname[MAXNAME],datareceived[BUFSIZE];
sd = socket( AF_INET, SOCK_DGRAM, 0 );
if( sd < 0 ) {
perror( "socket" );
exit( 1 );
}
memset( &serveraddress, 0, sizeof(serveraddress) );
memset( &cliaddr, 0, sizeof(cliaddr) );
serveraddress.sin_family = AF_INET;
serveraddress.sin_port = htons(MYPORT);//PORT NO
serveraddress.sin_addr.s_addr = htonl(INADDR_ANY);//IP ADDRESS
ret=bind(sd,(struct sockaddr*)&serveraddress,sizeof(serveraddress));
if(ret<0)
{
perror("BIND FAILS");
```

```
exit(1);
}
for(;;)
{
printf("I am waiting\n");
/*Received a datagram*/
length=sizeof(cliaddr);
n=recvfrom(sd,datareceived,BUFSIZE,0,
(struct sockaddr*)&cliaddr , &length);
printf("Data Received from %s\n",
inet_ntop(AF_INET,&cliaddr.sin_addr,
clientname,sizeof(clientname)));
/*Sending the Received datagram back*/
datareceived[n]='\0';
printf("I have received %s\n",datareceived);
sendto(sd,datareceived,n,0,(struct sockaddr *)&cliaddr,length);
}
}
```

client program :

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <signal.h>
#include <unistd.h>
#define BUFSIZE 512
static void sig_usr(int);
void str_cli(FILE *fp , int sockfd , struct sockaddr *server , socklen_t len);
int main( int C, char *argv[] )
{
```

```
int sd;
struct sockaddr_in
serveraddress;
/*Installing signal Handlers*/
signal(SIGPIPE,sig_usr);
signal(SIGINT,sig_usr);
if (NULL==argv[1])
{
printf("Please enter the IP Address of the server\n");
exit(0);
}
if (NULL==argv[2])
{
printf("Please enter the Port Number of the server\n");
exit(0);
}
sd = socket( AF_INET, SOCK_DGRAM, 0 );
if( sd < 0 )
{
perror( "socket" );
exit( 1 );
}
memset( &serveraddress, 0, sizeof(serveraddress) );
serveraddress.sin_family = AF_INET;
serveraddress.sin_port = htons(atoi(argv[2]));//PORT NO
serveraddress.sin_addr.s_addr = inet_addr(argv[1]);//ADDRESS
printf("Client Starting service\n");
printf("Enter Data For the server\n");
str_cli(stdin,sd ,(struct sockaddr *)&serveraddress,
sizeof(serveraddress));
}

void str_cli(FILE *fp, /*Here to be used as stdin as argument*/
int sockfd ,
struct sockaddr *to ,socklen_t length) /*Connection Socket */
/*************************************************************************/
{
int maxdes,n;
```

```
fd_set rset;
char sendbuf[BUFSIZE] , recvbuf[BUFSIZE] ,servername[100];
struct sockaddr_in serveraddr;
socklen_t slen;
FD_ZERO(&rset);
maxdes=(sockfd>fileno(fp)?sockfd+1:fileno(fp)+1);
for(;;){
FD_SET(fileno(fp) , &rset);
FD_SET(sockfd , &rset);
select(maxdes,&rset,NULL,NULL,NULL);
if(FD_ISSET(sockfd , & rset))
{
slen=sizeof(serveraddr);
n=recvfrom(sockfd,recvbuf,BUFSIZE,0,
(struct sockaddr*)&serveraddr,&slen);
printf("Data Received from server %s:\n",
inet_ntop(AF_INET,&serveraddr.sin_addr,
servername,sizeof(servername)));
write(1,recvbuf,n);
printf("Enter Data For the server\n");
}
if(FD_ISSET(fileno(fp) , & rset))
{
/*Reading data from the keyboard*/
fgets(sendbuf,BUFSIZE,fp);
n = strlen (sendbuf);
/*Sending the read data over socket*/
sendto(sockfd,sendbuf,n,0,to,length);
printf("Data Sent To Server\n");
}
}
}
```

UDP Client Server Application.

Compiling and running server.

```
[user@localhost week9]$ cc udp_server.c
[user@localhost week9]$ mv a.out udp_server
```

[user@localhost week9]$ ./ udp_server

I am waiting

Data Received from 127.0.0.1

I have received abcd efgh

rev is

hgfe dcba

I am waiting

Compiling and running client.

user@localhost week9]$ cc udp_client.c

[user@localhost week9]$ mv a.out udp_client

[user@localhost week9]$ ./ udp_client 127.0.0.1 11710

Client Starting service

Enter Data For the server

abcd efgh

Data Sent To Server

Data Received from server 127.0.0.1:

abcd efgh

Enter Data For the server

**6 B) UDP Client and server to transfer file**

```
// server code for UDP socket programming
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define IP_PROTOCOL 0
#define PORT_NO 15050
#define NET_BUF_SIZE 32
#define cipherKey 'S'
#define sendrecvflag 0
#define nofile "File Not Found!"

// function to clear buffer
void clearBuf(char* b)
{
	int i;
	for (i = 0; i < NET_BUF_SIZE; i++)
		b[i] = '\0';
}

// function to encrypt
char Cipher(char ch)
{
	return ch ^ cipherKey;
}

// function sending file
int sendFile(FILE* fp, char* buf, int s)
{
	int i, len;
	if (fp == NULL) {
		strcpy(buf, nofile);
		len = strlen(nofile);
		buf[len] = EOF;
		for (i = 0; i <= len; i++)
			buf[i] = Cipher(buf[i]);
		return 1;
	}

	char ch, ch2;
	for (i = 0; i < s; i++) {
```

```
                ch = fgetc(fp);
                ch2 = Cipher(ch);
                buf[i] = ch2;
                if (ch == EOF)
                        return 1;
        }
        return 0;
}

// driver code
int main()
{
        int sockfd, nBytes;
        struct sockaddr_in addr_con;
        int addrlen = sizeof(addr_con);
        addr_con.sin_family = AF_INET;
        addr_con.sin_port = htons(PORT_NO);
        addr_con.sin_addr.s_addr = INADDR_ANY;
        char net_buf[NET_BUF_SIZE];
        FILE* fp;

        // socket()
        sockfd = socket(AF_INET, SOCK_DGRAM, IP_PROTOCOL);

        if (sockfd < 0)
                printf("\nfile descriptor not received!!\n");
        else
                printf("\nfile descriptor %d received\n", sockfd);

        // bind()
        if (bind(sockfd, (struct sockaddr*)&addr_con, sizeof(addr_con)) == 0)
                printf("\nSuccessfully binded!\n");
        else
                printf("\nBinding Failed!\n");

        while (1) {
                printf("\nWaiting for file name...\n");

                // receive file name
                clearBuf(net_buf);

                nBytes = recvfrom(sockfd, net_buf,
                                        NET_BUF_SIZE, sendrecvflag,
                                        (struct sockaddr*)&addr_con, &addrlen);

                fp = fopen(net_buf, "r");
                printf("\nFile Name Received: %s\n", net_buf);
```

```
		if (fp == NULL)
			printf("\nFile open failed!\n");
		else
			printf("\nFile Successfully opened!\n");

		while (1) {

			// process
			if (sendFile(fp, net_buf, NET_BUF_SIZE)) {
				sendto(sockfd, net_buf, NET_BUF_SIZE,
					sendrecvflag,
					(struct sockaddr*)&addr_con, addrlen);
				break;
			}

			// send
			sendto(sockfd, net_buf, NET_BUF_SIZE,
				sendrecvflag,
				(struct sockaddr*)&addr_con, addrlen);
			clearBuf(net_buf);
		}
		if (fp != NULL)
			fclose(fp);
	}
	return 0;
}


// client code for UDP socket programming
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define IP_PROTOCOL 0
#define IP_ADDRESS "127.0.0.1" // localhost
#define PORT_NO 15050
#define NET_BUF_SIZE 32
#define cipherKey 'S'
#define sendrecvflag 0

// function to clear buffer
void clearBuf(char* b)
```

```
{
	int i;
	for (i = 0; i < NET_BUF_SIZE; i++)
		b[i] = '\0';
}

// function for decryption
char Cipher(char ch)
{
	return ch ^ cipherKey;
}

// function to receive file
int recvFile(char* buf, int s)
{
	int i;
	char ch;
	for (i = 0; i < s; i++) {
		ch = buf[i];
		ch = Cipher(ch);
		if (ch == EOF)
			return 1;
		else
			printf("%c", ch);
	}
	return 0;
}

// driver code
int main()
{
	int sockfd, nBytes;
	struct sockaddr_in addr_con;
	int addrlen = sizeof(addr_con);
	addr_con.sin_family = AF_INET;
	addr_con.sin_port = htons(PORT_NO);
	addr_con.sin_addr.s_addr = inet_addr(IP_ADDRESS);
	char net_buf[NET_BUF_SIZE];
	FILE* fp;

	// socket()
	sockfd = socket(AF_INET, SOCK_DGRAM,
					IP_PROTOCOL);

	if (sockfd < 0)
		printf("\nfile descriptor not received!!\n");
	else
```

```
            printf("\nfile descriptor %d received\n", sockfd);

        while (1) {
            printf("\nPlease enter file name to receive:\n");
            scanf("%s", net_buf);
            sendto(sockfd, net_buf, NET_BUF_SIZE,
                sendrecvflag, (struct sockaddr*)&addr_con,
                addrlen);

            printf("\n---------Data Received---------\n");

            while (1) {
                // receive
                clearBuf(net_buf);
                nBytes = recvfrom(sockfd, net_buf, NET_BUF_SIZE,
                                        sendrecvflag, (struct sockaddr*)&addr_con,
                                        &addrlen);

                // process
                if (recvFile(net_buf, NET_BUF_SIZE)) {
                    break;
                }
            }
            printf("\n-------------------------------\n");
        }
        return 0;
}
```

Server:

```
Socket file descriptor 3 received

Successfully binded!

Waiting for file name...

File Name Received: dm.txt

File Successfully opened!

Waiting for file name...

File Name Received: /home/dmayank/Documents/dm.txt

File Successfully opened!
```

Client :

```
Socket file descriptor 3 received

Please enter file name to receive:
dm.txt

---------Data Received---------
30
-------------------------------

Please enter file name to receive:
/home/dmayank/Documents/dm.txt

---------Data Received---------
30
-------------------------------
```

**Experiment 7:** Programs to demonstrate the usage of Advanced socket system calls like getsockopt( ), setsockopt(), getpeername ( ),getsockname( ),readv( ) and writev( ).

Elementary Socket System Calls:

**Socket:**

To do network I/O, the first thing a process must do is to call the socket system call, specifying the
type of communication protocol desired.

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int family, int type, int protocol);
```

The socket type is one of the following:
SOCK_STREAM stream socket
SOCK_DGRAM datagram socket
SOCK_RAW raw socket
SOCK_SEQPACKET sequenced packet socket
SOCK_RDM reliably delivered message socket (not implemented yet)

**1) readv and writev system calls:**

These two functions are similar to read and write , but readv and writev let us read into or write from one or more buffers with a single function call. These operations are called scatter read (since the input data is scattered into multiple application buffers) and gather write (since multiple buffers are gathered for a single output operation).

```
#include <sys/uio.h>
int readv(int fd , struct iovec iov[] , int iovcount) ;
int writev(int fd , struct iovec iov[] , int iovcount) ;
```

These two system calls use the following structure that is defined in
<syst/uio.h>:

```
struct iovec{
caddr_t iov_base; /*strating address of buffer*/
int iov_len; /*size of buffer in size*/
```

 The writev system call write the buffers specified by iov[0], iov[1], throughiov[iovcount-1].
 The readv sytem call does the input equivalent. It always fills one buffer (as specified but the iov_len value) before proceeding to the next buffer in the iov array.
 Both system calls return the total number of bytes read and written.

**2) getpeername** - get the name of the peer socket

```
#include <sys/socket.h>
int getpeername(int socket, struct sockaddr *address,socklen_t *address_len);
```

The getpeername() function retrieves the peer address of the specified socket, stores this address

in the sockaddr structure pointed to by the address argument, and stores the length of this address in the object pointed to by the address_len argument.

 If the actual length of the address is greater than the length of the supplied sockaddr structure, the stored address will be truncated.

 If the protocol permits connections by unbound clients, and the peer is not bound, then the value stored in the object pointed to by address is unspecified.

**3) getsockname** - get the socket name

```
#include <sys/socket.h>
int getsockname(int socket, struct sockaddr *address,socklen_t *address_len);
```

 The getsockname() function retrieves the locally-bound name of the specified socket,stores this address in the sockaddr structure pointed to by the address argument, and stores the length of this address in the object pointed to by the address_len argument.

 If the actual length of the address is greater than the length of the supplied sockaddr structure, the stored address will be truncated.

 If the socket has not been bound to a local name, the value stored in the object pointed to by address is unspecified.

4) **getsockopt and setsockopt** allow socket options values to be queried and set, respectively.

int getsockopt (sockid, level, optName, optVal, optLen);
 sockid: integer, socket descriptor
 level: integer, the layers of the protocol stack (socket, TCP, IP)
 optName: integer, option
 optVal: pointer to a buffer; upon return it contains the value of the specified option
 optLen: integer, in-out parameter it returns -1 if an error occured
int setsockopt (sockid, level, optName, optVal, optLen);
 optLen is now only an input parameter

**Experiment 8:** Implementation of concurrent chat server that allows current logged in users to communicate one with other

Program Objective: Determine the number of Users currently logged in and establish chat session with them.

Program Description:
The command that counts the number of users logged in is who |wc –l. Using this command, determine the number of users currently available for chat.

Steps
Server:
 Include appropriate header files.
 Create a TCP Socket.
 Fill in the socket address structure (with server information)
 Bind the address and port using bind() system call.
 Server executes listen() system call to indicate its willingness to receive connections.
 Accept the next completed connection from the client process by using an accept() system call.
 Receive a message from the Client using recv() system call.
 Send the reply of the message made by the client using send() system call.

Client
 Create a TCP Socket.
 Fill in the socket address structure (with server information)
 Establish connection to the Server using connect() system call.
 Send a chat message to the Server using send() system call.
 Receive the reply message made to the server using recv() system call.
 Write the result thus obtained on the standard output.

Server program :

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

// Function designed for chat between client and server.
void func(int connfd)
{
        char buff[MAX];
```

```
	int n;
	// infinite loop for chat
	for (;;) {
		bzero(buff, MAX);

		// read the message from client and copy it in buffer
		read(connfd, buff, sizeof(buff));
		// print buffer which contains the client contents
		printf("From client: %s\t To client : ", buff);
		bzero(buff, MAX);
		n = 0;
		// copy server message in the buffer
		while ((buff[n++] = getchar()) != '\n')
			;

		// and send that buffer to client
		write(connfd, buff, sizeof(buff));

		// if msg contains "Exit" then server exit and chat ended.
		if (strncmp("exit", buff, 4) == 0) {
			printf("Server Exit...\n");
			break;
		}
	}
}

// Driver function
int main()
{
	int sockfd, connfd, len;
	struct sockaddr_in servaddr, cli;

	// socket create and verification
	sockfd = socket(AF_INET, SOCK_STREAM, 0);
	if (sockfd == -1) {
		printf("socket creation failed...\n");
		exit(0);
	}
	else
		printf("Socket successfully created..\n");
	bzero(&servaddr, sizeof(servaddr));

	// assign IP, PORT
	servaddr.sin_family = AF_INET;
	servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
	servaddr.sin_port = htons(PORT);
```

```
	// Binding newly created socket to given IP and verification
	if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
		printf("socket bind failed...\n");
		exit(0);
	}
	else
		printf("Socket successfully binded..\n");

	// Now server is ready to listen and verification
	if ((listen(sockfd, 5)) != 0) {
		printf("Listen failed...\n");
		exit(0);
	}
	else
		printf("Server listening..\n");
	len = sizeof(cli);

	// Accept the data packet from client and verification
	connfd = accept(sockfd, (SA*)&cli, &len);
	if (connfd < 0) {
		printf("server accept failed...\n");
		exit(0);
	}
	else
		printf("server accept the client...\n");

	// Function for chatting between client and server
	func(connfd);

	// After chatting close the socket
	close(sockfd);
}
```

Client program :

```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
{
	char buff[MAX];
	int n;
	for (;;) {
```

```
		bzero(buff, sizeof(buff));
		printf("Enter the string : ");
		n = 0;
		while ((buff[n++] = getchar()) != '\n')
			;
		write(sockfd, buff, sizeof(buff));
		bzero(buff, sizeof(buff));
		read(sockfd, buff, sizeof(buff));
		printf("From Server : %s", buff);
		if ((strncmp(buff, "exit", 4)) == 0) {
			printf("Client Exit...\n");
			break;
		}
	}
}

int main()
{
	int sockfd, connfd;
	struct sockaddr_in servaddr, cli;

	// socket create and verification
	sockfd = socket(AF_INET, SOCK_STREAM, 0);
	if (sockfd == -1) {
		printf("socket creation failed...\n");
		exit(0);
	}
	else
		printf("Socket successfully created..\n");
	bzero(&servaddr, sizeof(servaddr));

	// assign IP, PORT
	servaddr.sin_family = AF_INET;
	servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
	servaddr.sin_port = htons(PORT);

	// connect the client socket to server socket
	if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
		printf("connection with the server failed...\n");
		exit(0);
	}
	else
		printf("connected to the server..\n");

	// function for chat
	func(sockfd);
```

```
        // close the socket
        close(sockfd);
}
```

**Compilation –**
Server side:
gcc server.c -o server
./server
Client side:
gcc client.c -o client
./client

**Output –**
Server side:
Socket successfully created..
Socket successfully binded..
Server listening..
server accept the client...
From client: hi
    To client : hello
From client: exit
    To client : exit
Server Exit...

Client side:
Socket successfully created..
connected to the server..
Enter the string : hi
From Server : hello
Enter the string : exit
From Server : exit
Client Exit...

**Experiment 9 : Implementation of DNS.**

```
#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<netdb.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>int main(int argc,char *argv[1])
{
struct hostent *hen;if(argc!=2)
{
fprintf(stderr,"Enter the hostname \n");exit(1);
}
hen=gethostbyname(argv[1]);
if(hen==NULL){fprintf(stderr,"Host not found \n");
}
printf("Hostname is %s \n",hen->h_name);
printf("IP address is %s \n",inet_ntoa(*((struct in_addr *)hen->h_addr)));
}
```

Input: ./a.out www.google.com
Output:
Host name is www.google.com
IP Address is 173.194.73.99

## Experiment 10 : **Implementation of Ping service.**

```
// C program to Implement Ping

// compile as -o ping
// run as sudo ./ping <hostname>

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <netinet/ip_icmp.h>
#include <time.h>
#include <fcntl.h>
#include <signal.h>
#include <time.h>

// Define the Packet Constants
// ping packet size
#define PING_PKT_S 64

// Automatic port number
#define PORT_NO 0

// Automatic port number
#define PING_SLEEP_RATE 1000000 x

// Gives the timeout delay for receiving packets
// in seconds
#define RECV_TIMEOUT 1

// Define the Ping Loop
int pingloop=1;
```

```
// ping packet structure
struct ping_pkt
{
        struct icmphdr hdr;
        char msg[PING_PKT_S-sizeof(struct icmphdr)];
};

// Calculating the Check Sum
unsigned short checksum(void *b, int len)
{ unsigned short *buf = b;
        unsigned int sum=0;
        unsigned short result;

        for ( sum = 0; len > 1; len -= 2 )
            sum += *buf++;
        if ( len == 1 )
            sum += *(unsigned char*)buf;
        sum = (sum >> 16) + (sum & 0xFFFF);
        sum += (sum >> 16);
        result = ~sum;
        return result;
}


// Interrupt handler
void intHandler(int dummy)
{
        pingloop=0;
}

// Performs a DNS lookup
char *dns_lookup(char *addr_host, struct sockaddr_in *addr_con)
{
        printf("\nResolving DNS..\n");
        struct hostent *host_entity;
        char *ip=(char*)malloc(NI_MAXHOST*sizeof(char));
        int i;
```

```
        if ((host_entity = gethostbyname(addr_host)) == NULL)
        {
            // No ip found for hostname
            return NULL;
        }

        //filling up address structure
        strcpy(ip, inet_ntoa(*(struct in_addr *)
                                                host_entity->h_addr));

        (*addr_con).sin_family = host_entity->h_addrtype;
        (*addr_con).sin_port = htons (PORT_NO);
        (*addr_con).sin_addr.s_addr = *(long*)host_entity->h_addr;

        return ip;

}

// Resolves the reverse lookup of the hostname
char* reverse_dns_lookup(char *ip_addr)
{
        struct sockaddr_in temp_addr;
        socklen_t len;
        char buf[NI_MAXHOST], *ret_buf;

        temp_addr.sin_family = AF_INET;
        temp_addr.sin_addr.s_addr = inet_addr(ip_addr);
        len = sizeof(struct sockaddr_in);

        if (getnameinfo((struct sockaddr *) &temp_addr, len, buf,
                                        sizeof(buf), NULL, 0, NI_NAMEREQD))
        {
            printf("Could not resolve reverse lookup of hostname\n");
            return NULL;
        }
        ret_buf = (char*)malloc((strlen(buf) +1)*sizeof(char) );
        strcpy(ret_buf, buf);
```

```
	return ret_buf;
}

// make a ping request
void send_ping(int ping_sockfd, struct sockaddr_in *ping_addr,
                                char *ping_dom, char *ping_ip, char *rev_host)
{
	int ttl_val=64, msg_count=0, i, addr_len, flag=1,
			msg_received_count=0;

	struct ping_pkt pckt;
	struct sockaddr_in r_addr;
	struct timespec time_start, time_end, tfs, tfe;
	long double rtt_msec=0, total_msec=0;
	struct timeval tv_out;
	tv_out.tv_sec = RECV_TIMEOUT;
	tv_out.tv_usec = 0;

	clock_gettime(CLOCK_MONOTONIC, &tfs);


	// set socket options at ip to TTL and value to 64,
	// change to what you want by setting ttl_val
	if (setsockopt(ping_sockfd, SOL_IP, IP_TTL,
			&ttl_val, sizeof(ttl_val)) != 0)
	{
	    printf("\nSetting socket options
                                to TTL failed!\n");
	    return;
	}

	else
	{
	    printf("\nSocket set to TTL..\n");
	}

	// setting timeout of recv setting
	setsockopt(ping_sockfd, SOL_SOCKET, SO_RCVTIMEO,
```

```
                    (const char*)&tv_out, sizeof tv_out);

// send icmp packet in an infinite loop
while(pingloop)
{
    // flag is whether packet was sent or not
    flag=1;

    //filling packet
    bzero(&pckt, sizeof(pckt));

    pckt.hdr.type = ICMP_ECHO;
    pckt.hdr.un.echo.id = getpid();

    for ( i = 0; i < sizeof(pckt.msg)-1; i++ )
            pckt.msg[i] = i+'0';

    pckt.msg[i] = 0;
    pckt.hdr.un.echo.sequence = msg_count++;
    pckt.hdr.checksum = checksum(&pckt, sizeof(pckt));


    usleep(PING_SLEEP_RATE);

    //send packet
    clock_gettime(CLOCK_MONOTONIC, &time_start);
    if ( sendto(ping_sockfd, &pckt, sizeof(pckt), 0,
    (struct sockaddr*) ping_addr,
            sizeof(*ping_addr)) <= 0)
    {
            printf("\nPacket Sending Failed!\n");
            flag=0;
    }

    //receive packet
    addr_len=sizeof(r_addr);

    if ( recvfrom(ping_sockfd, &pckt, sizeof(pckt), 0,
```

```
        (struct sockaddr*)&r_addr, &addr_len) <= 0
        && msg_count>1)
{
        printf("\nPacket receive failed!\n");
}

else
{
        clock_gettime(CLOCK_MONOTONIC, &time_end);

        double timeElapsed = ((double)(time_end.tv_nsec -
                                        time_start.tv_nsec))/1000000.0
        rtt_msec = (time_end.tv_sec-
                        time_start.tv_sec) * 1000.0
                        + timeElapsed;

        // if packet was not sent, don't receive
        if(flag)
        {
                if(!(pckt.hdr.type ==69 && pckt.hdr.code==0))
                {
                        printf("Error..Packet received with ICMP
                                type %d code %d\n",
                                pckt.hdr.type, pckt.hdr.code);
                }
                else
                {
                        printf("%d bytes from %s (h: %s)
                                (%s) msg_seq=%d ttl=%d
                                rtt = %Lf ms.\n",
                                PING_PKT_S, ping_dom, rev_host,
                                ping_ip, msg_count,
                                ttl_val, rtt_msec);

                        msg_received_count++;
                }
        }
}
```

```
        }
        clock_gettime(CLOCK_MONOTONIC, &tfe);
        double timeElapsed = ((double)(tfe.tv_nsec -
                                            tfs.tv_nsec))/1000000.0;

        total_msec = (tfe.tv_sec-tfs.tv_sec)*1000.0+
                                            timeElapsed

        printf("\n===%s ping statistics===\n", ping_ip);
        printf("\n%d packets sent, %d packets received, %f percent
            packet loss. Total time: %Lf ms.\n\n",
            msg_count, msg_received_count,
            ((msg_count - msg_received_count)/msg_count) * 100.0,
            total_msec);
}

// Driver Code
int main(int argc, char *argv[])
{
        int sockfd;
        char *ip_addr, *reverse_hostname;
        struct sockaddr_in addr_con;
        int addrlen = sizeof(addr_con);
        char net_buf[NI_MAXHOST];

        if(argc!=2)
        {
            printf("\nFormat %s <address>\n", argv[0]);
            return 0;
        }

        ip_addr = dns_lookup(argv[1], &addr_con);
        if(ip_addr==NULL)
        {
            printf("\nDNS lookup failed! Could
                                not resolve hostname!\n");
            return 0;
        }
```

```
        reverse_hostname = reverse_dns_lookup(ip_addr);
        printf("\nTrying to connect to '%s' IP: %s\n",
                                                        argv[1], ip_addr);
        printf("\nReverse Lookup domain: %s",
                                            reverse_hostname);

        //socket()
        sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
        if(sockfd<0)
        {
            printf("\nSocket file descriptor not received!!\n");
            return 0;
        }
        else
            printf("\nSocket file descriptor %d received\n", sockfd);

        signal(SIGINT, intHandler);//catching interrupt

        //send pings continuously
        send_ping(sockfd, &addr_con, reverse_hostname,
                                                    ip_addr, argv[1]);

        return 0;
}
```

1. **An example output:** Run sudo ./ping google.com

```
Resolving DNS..

Trying to connect to 'google.com' IP: 172.217.27.206

Reverse Lookup domain: bom07s15-in-f14.1e100.net
Socket file descriptor 3 received

Socket set to TTL..
64 bytes from bom07s15-in-f14.1e100.net (h: google.com) (172.217.27.206)
                msg_seq=1 ttl=64 rtt = 57.320584 ms.
```

64 bytes from bom07s15-in-f14.1e100.net (h: google.com) (172.217.27.206)
msg_seq=2 ttl=64 rtt = 58.666775 ms.

64 bytes from bom07s15-in-f14.1e100.net (h: google.com) (172.217.27.206)
msg_seq=3 ttl=64 rtt = 58.081148 ms.

64 bytes from bom07s15-in-f14.1e100.net (h: google.com) (172.217.27.206)
msg_seq=4 ttl=64 rtt = 58.700630 ms.

64 bytes from bom07s15-in-f14.1e100.net (h: google.com) (172.217.27.206)
msg_seq=5 ttl=64 rtt = 58.281802 ms.

64 bytes from bom07s15-in-f14.1e100.net (h: google.com) (172.217.27.206)
msg_seq=6 ttl=64 rtt = 58.360916 ms.

===172.217.27.206 ping statistics===

6 packets sent, 6 packets received, 0.000000 percent packet loss.
Total time: 6295.187804 ms.