

SRI VASAVI ENGINEERING COLLEGE (Autonomous)
PEDATADEPALLI, TADEPALLIGUDEM.



Certificate

*This is to certify that this is a bonafide record of Practical Work done in
Computer Networks LAB by Mr./Miss _____ bearing -
RollNo _____ of CSE Branch of VI Semester during the academic year 2022 -23.*

No. of Experiments Done: 10

Faculty In charge of the Laboratory

Head of the Department

EXTERNAL EXAMINER

INDEX

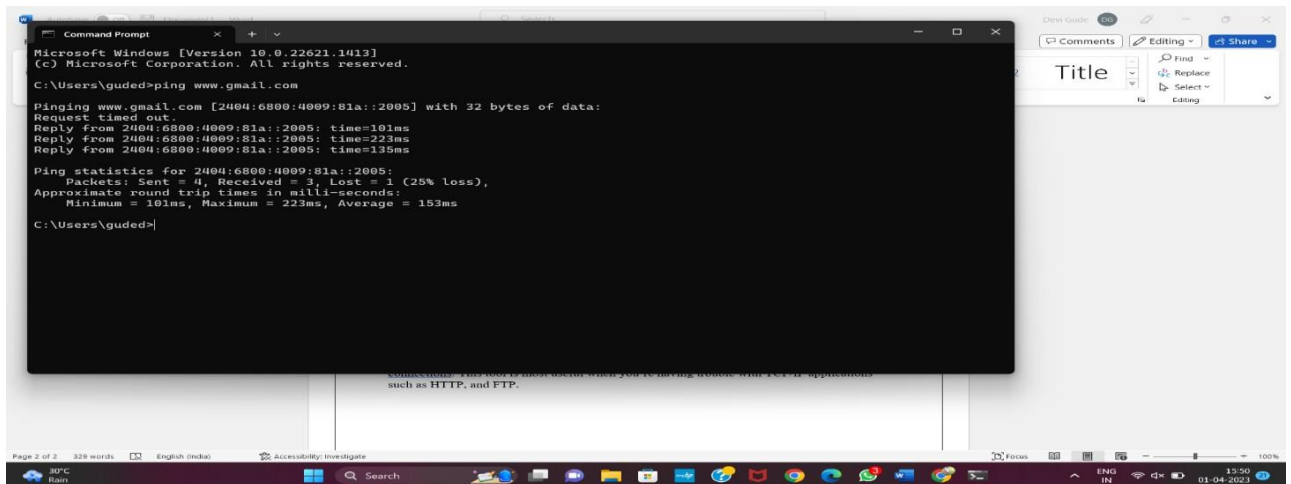
S.NO	Experiment	Page. No
1	Study of basic network commands and Network configuration commands. 1.Ping 2.Tracert/Traceroute 3.Ipconfig/ifconfig 4.Hostname 5.Nslookup 6.Netstat	3-5
2	Construct Detecting error using CRC-CCITT.	6-9
3	Implementing of Bit Stuffing.	10-12
4	Implementation of Character Stuffing.	13-14
5	Implementation of stop and wait protocol.	15-16
6	Implementation of Dijkstra's algorithm.	17-19
7	Implementation Distance vector algorithm.	20-26
8	Implementation of Congestion control using leaking bucket algorithms.	27-28
9	Implementation using Socket TCP both client and server programs.	29-34
10	Implementation using Socket UDP both client and server programs.	35-39

Task 1 : Study of basic network commands and Network configuration commands.

- 1.Pin 2.Tracert/Traceroute 3.Ipconfig/ifconfig 4.Hostname
5.Nslookup 6.Netstat

1.Ping: ping is the most basic TCP/IP command ,and it's the same as placing a phone call to your best friend. You pick up your telephone and dial a number, expecting your best friend to reply with “Hello” on the other end. Computers make phone calls to each other over a network by using a ping command. The Ping commands main purpose is to place a phone call to another computer on the network, and request an answer. Ping has 2 options it can us to place a phone call to another computer on the network. It can use the computers name or IP address.

Output:



```
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\Users\guded>ping www.gmail.com

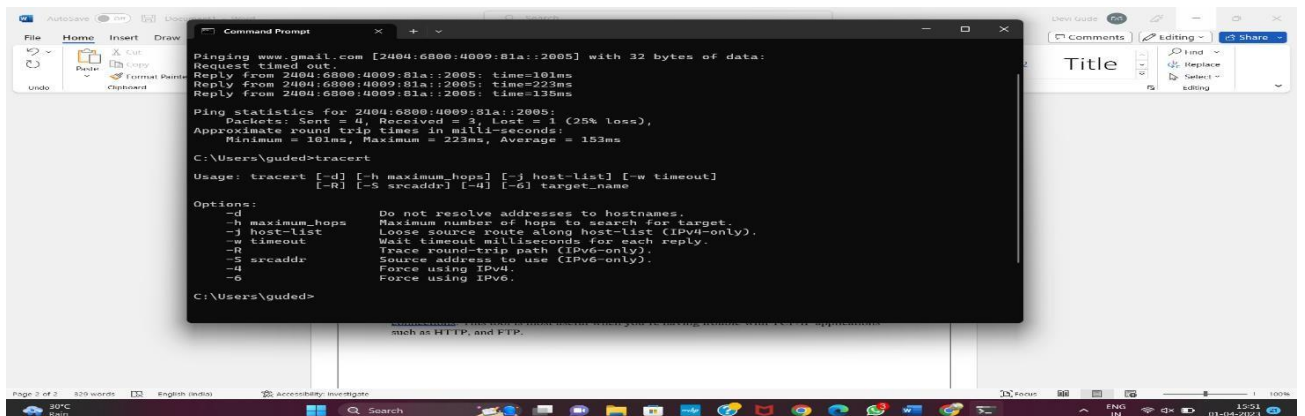
Pinging www.gmail.com [2404:6800:4009:81a::2005] with 32 bytes of data:
Request timed out.
Reply from 2404:6800:4009:81a::2005: time=101ms
Reply from 2404:6800:4009:81a::2005: time=223ms
Reply from 2404:6800:4009:81a::2005: time=135ms

Ping statistics for 2404:6800:4009:81a::2005:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 101ms, Maximum = 223ms, Average = 153ms

C:\Users\guded>
```

2.Tracert/Traceroute: The tracert command displays a list of all the routers that a packet has to go through to get from the computer where tracert is run to any other computer on the internet.

Output:



```
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\Users\guded>tracert

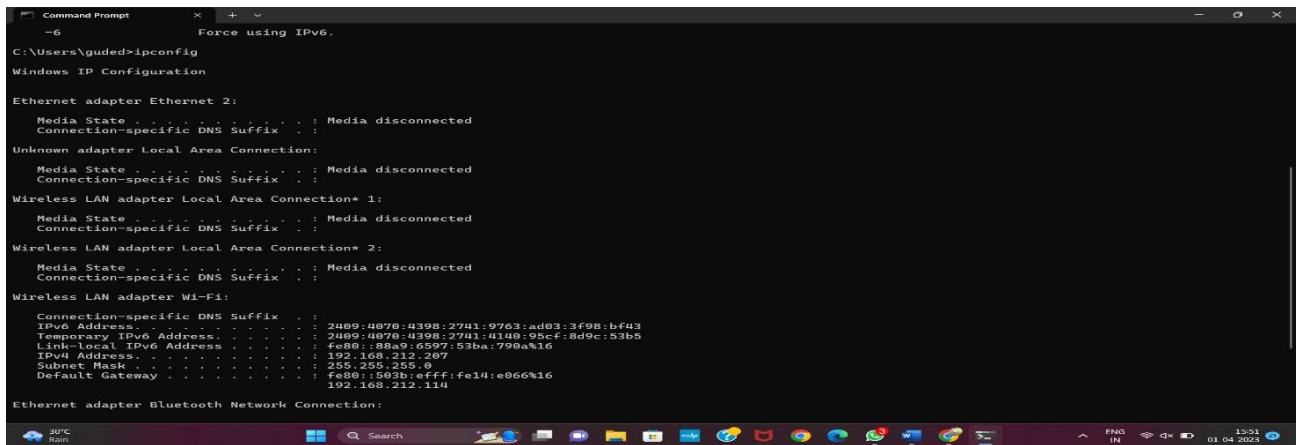
Usage: tracert [-d] [-h maximum_hops] [-j host-list] [-w timeout]
           [-R] [-S srcaddr] [-4] [-6] target_name

Options:
    -d          Do not resolve addresses to hostnames.
    -h maximum_hops  Maximum number of hops to search for target.
    -j host-list  Loose source route along host-list (IPv4-only).
    -w timeout   Wait timeout milliseconds for each reply.
    -R          Trace round-trip path (IPv6-only).
    -S srcaddr   Source address to use (IPv6-only).
    -4          Force using IPv4.
    -6          Force using IPv6.

C:\Users\guded>
```

3. Ipconfig/ifconfig: The ipconfig command displays information about the host computer TCP/IP configuration.

Output:



```
Command Prompt
-6 Force using IPv6.

C:\Users\guled>ipconfig

Windows IP Configuration

Ethernet adapter Ethernet 2:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 
    . . . . .

Unknown adapter Local Area Connection:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 
    . . . . .

Wireless LAN adapter Local Area Connection* 1:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 
    . . . . .

Wireless LAN adapter Local Area Connection* 2:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 
    . . . . .

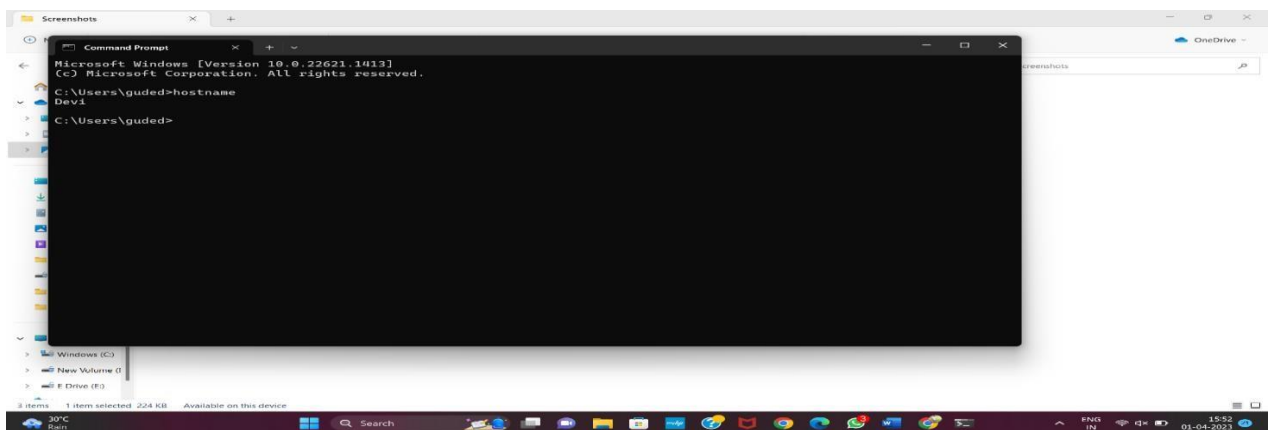
Wireless LAN adapter Wi-Fi:

    Connection-specific DNS Suffix  . : 
    IPv6 Address. . . . . : 2409:4070:4398:2741:9763:ad03:3f98:b443
    Temporary IPv6 Address. . . . . : 2409:4070:4398:2741:4140:95cf:8d9c:53b5
    Link-Local IPv6 Address . . . . . : fe80:18ba9:6597:53ba:798a%16
    IPv4 Address. . . . . : 192.168.212.207
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : fe80:1503b:iefff:fe14:e066%16
    . . . . . : 192.168.212.114

Ethernet adapter Bluetooth Network Connection:
```

4. Hostname: This is the simplest of all TCP/IP commands. It simply displays the name of your computer.

Output:



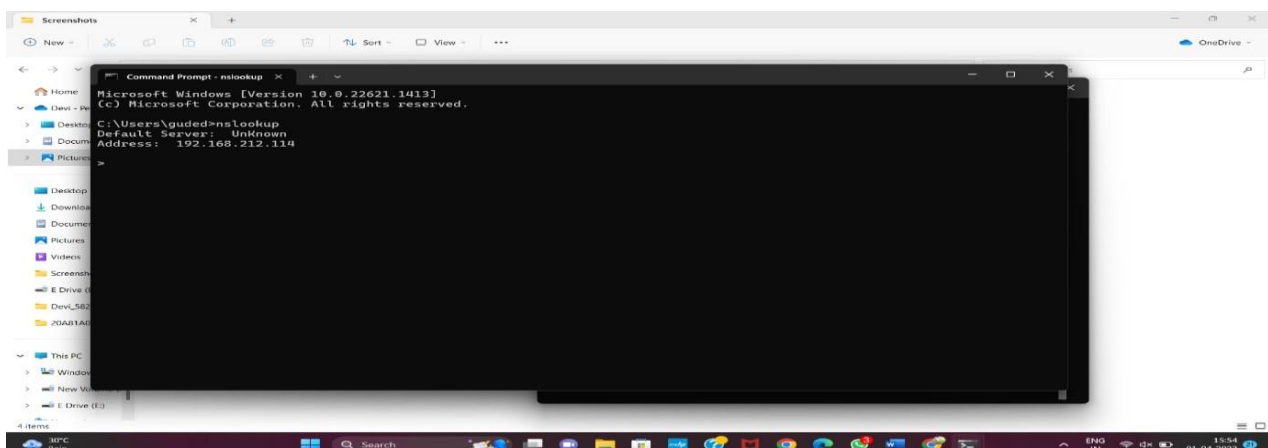
```
Screenshots
Command Prompt
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\Users\guled>hostname
Dev1

C:\Users\guled>
```

5. Nslookup: Nslookup is used for diagnosing DNS problems. If you can access a resource by specifying an IP address but not its DNS you have a DNS problem.

Output:



```
Screenshots
Command Prompt - nslookup
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\Users\guled>nslookup
Default Server: Unknown
Address: 192.168.212.114
```

6.Netstat: Netstat displays a variety of statistics about a computers active TCP/IP connections. This tool is most useful when you're having trouble with TCP/IP applications such as HTTP, and FTP.

Output:

```
Command Prompt - netstat
Microsoft Windows [Version 10.0.22621.1413]
(c) Microsoft Corporation. All rights reserved.

C:\Users\guded>netstat

Active Connections

Proto Local Address           Foreign Address         State
TCP    127.0.0.1:58076          Dev1:58076              ESTABLISHED
TCP    127.0.0.1:58078          Dev1:58076              ESTABLISHED
TCP    192.168.212.207:49411    20.198.119.84:https      ESTABLISHED TCP
:50541 ec2-65-2-109-57:https    ESTABLISHED TCP    192.168.212.207:50544 ec2-65-2-109-57:https    ESTABLISHED TCP    192.168.212.207:50908 20.50.201.201:ht
tps ESTABLISHED TCP    192.168.212.207:50917 20.50.201.201:https    ESTABLISHED TCP    192.168.212.207:50920 1drv:https          ESTABLISHED TCP
192.168.212.207:50925 52.109.56.86:https      TIME_WAIT
TCP    192.168.212.207:50926    20.42.72.131:https      ESTABLISHED TCP    192.168.212.207:50929 13.107.18.254:https   ESTABLISHED TCP    192.168.212.207
:50930 52.123.128.254:https    ESTABLISHED TCP    192.168.212.207:50932 52.123.128.254:https   ESTABLISHED
TCP    192.168.212.207:50933    204.79.197.222:https    ESTABLISHED
```

Task2:Construct Detecting error using CRC-CCITTDescription.

The Data Link Layer is the second layer in the OSI model, above the Physical Layer, which ensures that the error free data is transferred between the adjacent nodes in the network. It breaks the datagram passed down by above layers and converts them into frames ready for transfer. This is called Framing. It provides two main functionalities

Reliable data transfer service between two peer network layers

Flow Control mechanism, which regulates the flow of frames such that data congestion is not there at slow receivers due to fast senders.

There are two basic strategies for dealing with errors. One way is to include enough redundant information (extra bits are introduced into the data stream at the transmitter on a regular and logical basis) along with each block of data sent to enable the receiver to deduce what the transmitted character must have been. The other way is to include only enough redundancy to allow the receiver to deduce that error has occurred, but not which error has occurred and the receiver asks for a retransmission. The former strategy uses Error-Correcting Codes and latter uses Error-detecting Codes.

CRC method can detect a single burst of length n , since only one bit per column will be changed, a burst of length $n+1$ will pass undetected, if the first bit is inverted, the last bit is inverted and all other bits are correct. If the block is badly garbled by a long burst or by multiple shorter burst, the probability that any of the n columns will have the correct parity that is 0.5. so the probability of a bad block being expected when it should not be $2 \text{ power}(-n)$. This scheme some times known as Cyclic Redundancy Code Algorithm for computing checksum:

Let 'r' be the degree of $G(x)$. Append 'r' to the lower end of the frame so that it contains $(m + r)$ bits.

Divide $M(x)$ by $G(x)$ using MOD-2 division.

Subtract the remainder from $M(x)$ using MOD-2 subtraction.

The result is the check summed frame to be transmitted.

Eg: frame = 1101011011

$G(x) = x^4 + x + 1 = 10011$

degree = 4

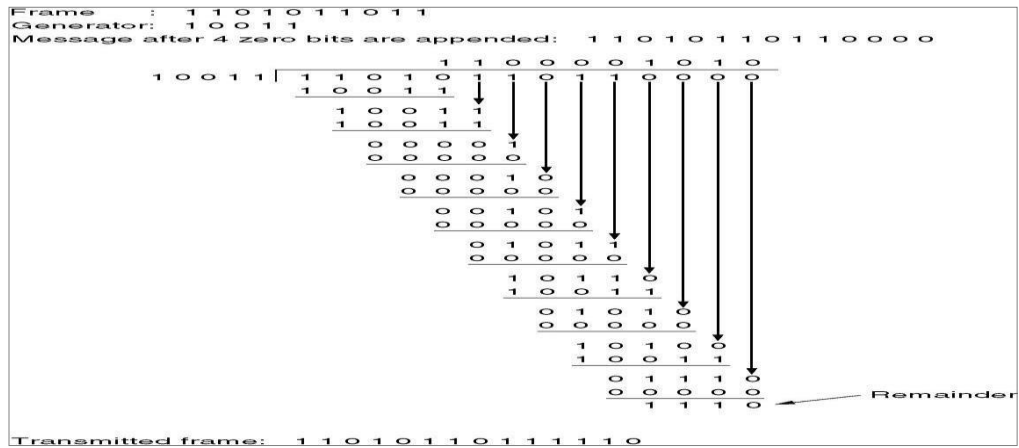
Therefore, frame = 1101011011 + 0000

$M(x) = 11010110110000$

Commonly used divisor polynomials are:

CRC 12: $x^{12} + x^{11} + x^3 + x^2 + x + 1$ CRC 16 : $x^{16} + x^{15} + x^2 + 1$

CRC CCITT : $x^{16} + x^{12} + x^5$



Program:

```

#include<stdio.h>

#include <string.h>

#define N strlen(g)

char t[28],cs[28],g[28];

int a,e,c,b;

void xor()

{
  for(c=1;c<N;c++) cs[c]=((cs[c]==g[c])?'0':'1');
}

void crc()

{
  for(e=0;e<N;e++)cs[e]=t[e];
  do{
    if(cs[0]=='1')
      xor(); for(c=0;c<N-1;c++)

    cs[c]=cs[c+1];

    cs[c]=t[e++];
  }while(e<=a+N);

```

```

}

int main()
{
int flag=0; do{
printf("\n1.crc12\n2.crc16\ncrc ccit\n4.exit\n\nEnter your option.");
scanf("%d",&b);
switch(b)
{
case 1:strcpy(g,"1100000001111"); break;
case 2:strcpy(g,"11000000000000101"); break;
case 3:strcpy(g,"10001000000100001"); break;
case 4:return 0;
}
printf("\n enter data:");
scanf("%s",t);
printf("\n      \n");
printf("\n generating polynomial:%s",g);
a=strlen(t);
for(e=a;e<a+N-1;e++)
t[e]='0';
printf("\n      \n");
printf("mod-ified data is:%s",t);
printf("\n      \n"); crc();
printf("checksum is:%s",cs);
for(e=a;e<a+N-1;e++)
t[e]=cs[e-a];
printf("\n      \n");
printf("\n final codeword is : %s",t);
printf("\n      \n");
printf("\ntest error detection 0(yes) 1(no)?:");

```



```
scanf("%d",&e);
if(e==0)
{
do{
printf("\n\tenter the position where error is to be inserted:");
scanf("%d",&e);
}
```

```
while(e==0||e>a+N-1);
t[e-1]=(t[e-1]=='0')?'1':'0';
printf("\n\t\t\t\t\t");
printf("\n\t\t\t\t\tterroneous data:%s\n",t);
}
crc();
for(e=0;(e<N-1)&&(cs[e]!='1');e++);
```

```
if(e<N-1)
printf("error detected\n\n"); else
printf("\n no error detected \n\n");
printf("\n\t\t\t\t\t");
}while(flag!=1);
}
```

Output:

```
OnlineGDB beta
online compiler and debugger for c/c++
code, compile, run, debug, share.

IDE
My Projects
Classroom new
Learn Programming
Programming Questions
Jobs new
Sign Up
Login

GOT AN OPINION?
PLEASE ASK FOR FEEDBACK!
Have fun taking surveys
and get paid!

About • FAQ • Blog • Terms of Use • Contact Us • GDB
Tutorial • Credits • Privacy
© 2016 - 2023 GDB Online

input
1.crc12
2.crc16
crc.celt
4.exit

Enter your option.1
enter data:1001

generating polynomial:110000000111
mod-ified data is:1001000000000000
checksum is:000001011010

final codeword is : 1001000001011010

test error detection 0(yes) 1(no)?:0
enter the position where error is to be inserted:5

errroneous data:1001100001011010
error detected

1.crc12
2.crc16
crc.celt
4.exit

Enter your option.4
```

Task 3: Program for implementing the data link layer framing method for Bit Stuffing.

Description:

The Bit stuffing technique allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary number of bits per character. In this each frame begins and ends with a special bit pattern, 01111110 called a flag byte. When ever the sender's data link encounters five consecutive ones in the data. It automatically stuffs a '0' bit into the out going bit stream. This is called "bit stuffing".

Program:

```
#include<stdio.h>

void main()
{
    int i=0,len_ip=0,count;
    int len_op=0;
    int j=0,s=0,xxx=0,five=0;
    char ip[' '],op[' '],decode_op[' '];
    char pre_post[]={ '0','1','1','1','1','1','1','0' };

    printf("Enter input bitsequence: \n");
    scanf("%s",ip);
    for(i=0;ip[i]!=0;i++)
        len_ip++;
    /* Stuffing*/
    for(i=0;i<8;i++)
        op[i]=pre_post[i];
    count=i;
    do
    {
        if(ip[j]=='1')
            five++;
        else
            five=0;
        op[count]=ip[j];
```

```

        count++;
        j++;

        if(five==5)
        {
            op[count++]=0;
            five=0;
        }
    } while(j<len_ip);
    for(s=count,xxx=0;s<(count+8);s++,xxx++)
    {
        op[s]=pre_post[xxx];
    }
    printf("Output\n");
    printf(" ----- ");
    printf("\nStuffed BitSequence is:");
    op[s]=0;
    puts(op);
    /*Destuffing*/
    j=0;
    five=0;
    len_op=count+8;
    i=8;
    do
    {
        decode_op[j++]=op[i];
        if(op[i]=='1')
            five++;
        else
            five=0;
    }

```

```

        if(five==5)
        {
            i++;
            five=0;
        }

        len_op--;
        i++;
    } while((len_op-8)!=0);
    decode_op[len_ip]=0;
    printf("Destuffed BitSequence is:");
    puts(decode_op);
}

```

Output:

The screenshot shows the OnlineGDB online compiler interface. The code editor contains the C program from the previous block. The console output shows the following:

```

Enter input bitsequence:
01111111111110
Output
-----
Stuffed BitSequence is:011111100111110111101100111110
Destuffed BitSequence is:01111111111110
...Program finished with exit code 0
Press ENTER to exit console.

```

Task 4 : Program for implementing the data link layer framing method for Character Stuffing.

Description:

In character stuffing method, the frame is delimited by DLE STX and DLE ETX. It may easily happen that the characters for DLE STX or DLE ETX occur in the data, which will interface with the framing. One way to solve this problem is to have the sender's data link layer insert an ASCII DLE character just before each "accidental" DLE character in the data. The data link layer on the receiving end removes the DLE before the data is given to the network layer. This technique is called "Character Stuffing".

Program:

```
#include<stdio.h>

#include<string.h>

void main()
{
    char source[ ],char_stuff[ ],char_destuff[ ];
    int i=0,j=0,k=0;
    printf("Enter plain text");
    gets(source);
    char_stuff[0]='d', stuff[1]='l', stuff[2]='e', stuff[3]='s', stuff[4]='t', stuff[5]='x';
    j=6;
    for(k=0;k<strlen(source);)
    {
        if(source[k]=='d'&&source[k+1]=='l'&&source[k+2]=='e')
        {
            char_stuff[j++]='d';char_stuff[j++]='l';char_stuff[j++]='e';
            char_stuff[j++]='d';char_stuff[j++]='l';char_stuff[j++]='e';
            k+=3;
        }
        else{
            char_stuff[j++]=source[k++];
        }
        char_stuff[j++]='d';char_stuff[j++]='l';char_stuff[j++]='e';
        char_stuff[j++]='s';char_stuff[j++]='t';char_stuff[j++]='x';
    }
```

```

char_stuff[j]=0;

printf("After character stuffing:");

puts(char_stuff);

j=0;

for(i=6;i<(strlen(char_stuff)-6);)

{

    if(char_stuff[i]=='d'&&char_stuff[i+1]=='l'&&char_stuff[i+2]=='e')

    {

        char_destuff[j++]='d'; char_destuff[j++]='l'; char_destuff[j++]='e';

        i=i+6;

    }

else

    char_destuff[j++]=char_stuff[i++];

}

char_destuff[j]=0;

printf("\nAfter character de-stuffing:");

puts(char_destuff);

}

```

Output:

The screenshot shows an online C compiler interface. The code editor contains the following C program:

```

1 #include<stdio.h>
2 #include<string.h>
3
4 void main()
5 {
6     char source[' '],char_stuff[' '],char_destuff[' '];
7     int i=0,j=0,k=0;
8     printf("Enter plain text");
9     gets(source);
10    char_stuff[i]=source[i];

```

The output window shows the following execution results:

```

main.c: in function 'main':
main.c:10:9: warning: 'gets' is deprecated [-Wdeprecated-declarations]
10 |     gets(source);
    |     ^~~~~
In file included from main.c:1:
/usr/include/stdio.h:605:14: note: declared here
605 | extern char *gets(char *__s) __wur __attribute_deprecated__;
    |
    |
/usr/bin/ld: /tmp/ccogthj7.o: in function 'main':
main.c:(.text+0x59): warning: the 'gets' function is dangerous and should not be used.
Enter plain text:Devil
After character stuffing:Devil\x00\x00\x00\x00\x00\x00
After character de-stuffing:Devil
... Program finished with exit code 0
Press ENTER to exit console.

```

Task 5 : Implementation of stop and wait protocol.

Description:

The sender must never transmit a new frame until the previous one has been fetched from physical layer, then the new one overwrites the old one. The general solution to this problem is to have the receiver send a little dummy frame to the sender which, in effect, gives the sender permission to transmit the next frame. Protocol in which the receiver sends a dummy frame back to the sender which in effect gives the sender permission to transmit the next frame. Protocols in which the sender sends one frame and then waits for acknowledgement before proceeding are called stop and wait protocols.

1. Start with the window size of 1 from the transmitting (Source) node
2. After transmission of a frame the transmitting (Source) node waits for a reply (Acknowledgement) from the receiving (Destination) node.
3. If the transmitted frame reaches the receiver (Destination) without error, the receiver (Destination) transmits a Positive Acknowledgement.
4. If the transmitted frame reaches the receiver (Destination) with error, the receiver (Destination) does not transmit acknowledgement.
5. If the transmitter receives a positive acknowledgement it transmits the next frame if any. Else if the transmission timer expires, it retransmits the same frame again.
6. If the transmitted acknowledgement reaches the Transmitter (Destination) without error, the Transmitter (Destination) transmits the next frame if any.
7. If the transmitted frame reaches the Transmitter (Destination) with error, the Transmitter (Destination) retransmits the same frame.
8. This concept of the Transmitting (Source) node waiting after transmission for a reply from the receiver is known as STOP and WAIT.

Program:

```
#include<stdio.h>

int main()
{
    int framesize, sent=0,ack,i;
    printf("enter number of frames\n");
    scanf("%d",&framesize);
    while(1)
    {

        for(i=0;i<framesize;i++)
```

```

{
    printf("frame %d has been transmitted.\n",sent);

    sent++;

    if(sent==framesize)

        break;
}

printf("\n please enter the last acknowledgement received.\n");
scanf("%d",&ack);
if(ack>=framesize)

    break;

else

    sent=ack;
}

return 0;
}

```

Output:

The screenshot shows the OnlineGDB interface. The code editor on the left contains the C program. The output window on the right shows the following text:

```

enter number of frames
5
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.

please enter the last acknowledgement received.
5

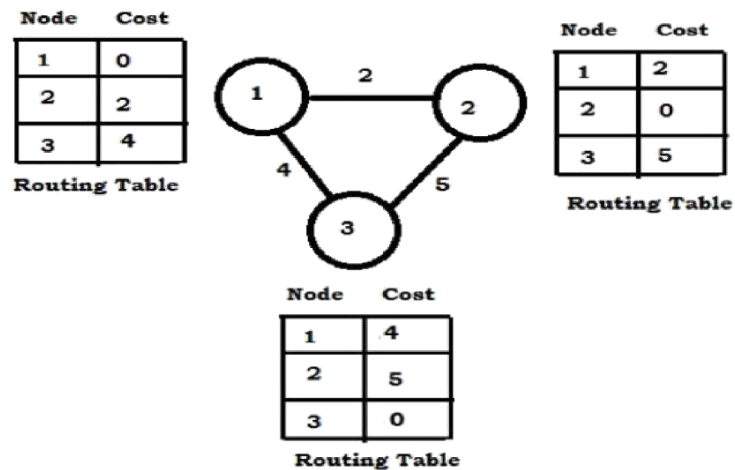
...Program finished with exit code 0
Press ENTER to exit console.

```


Task 6: Implementation Distance vector algorithm.

Description:

Distance Vector Routing Algorithms calculate a best route to reach a destination based solely on distance. E.g. RIP. RIP calculates the reachability based on hop count. It's different from link state algorithms which consider some other factors like bandwidth and other metrics to reach a destination. Distance vector routing algorithms are not preferable for complex networks and take longer to converge.



Program:

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

main()
{
    int i,j,x,y,n,d,delay=1000;
    int edge[50][50],c[50],cost[50][50];
    char ch;
    printf("\nEnter the No. of nodes in graph: ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        for(j=i;j<n;j++)
        {
            fflush(stdin);
            printf("\nIs there any edge from %d to %d? ",i+1,j+1);
```

```

scanf("%d",&d);

if(d)
{
    edge[i][j]=1;
}
else
{
    edge[i][j]=0;
}
}

printf("\nWhich Routing table do u want to find the cost of  which destination? ");
scanf("%d%d",&x,&y);
for(i=0;i<=n;i++)
{
    if((edge[i][x-1])==1||(edge[x-1][i])==1)
    {
        printf("\nEnter the cost of %d node to its neighbour %d:",x,i+1);
        scanf("%d",&c[i]);
        printf("\nEnter the Routing table entry to %d to %d: ",i+1,y);
        scanf("%d",&cost[i][y-1]);
        if(delay>(c[i]+cost[i][y-1]))
        {
            d=i+1;
            delay=c[i]+cost[i][y-1];
        }
    }
}

printf("\nEstimated cost from node %d to %d is %d via the node is %d",x,j,delay,d);

```

}

Output:

The screenshot shows the OnlineGDB online compiler interface. The browser address bar displays 'onlinegdb.com/online_c_compiler'. The interface includes a sidebar with navigation links like 'IDE', 'My Projects', 'Classroom', 'Learn Programming', 'Programming Questions', 'Jobs', 'Sign Up', and 'Login'. The main editor area shows a C program for a graph routing algorithm. The program includes a warning: 'main.c:3:1: warning: return type defaults to 'int' [-Wimplicit-int]'. The code defines a graph with 3 nodes and asks the user to input edges and their costs. It then calculates the estimated cost from node 1 to node 3 via node 2, resulting in a cost of 4. The program finishes with exit code 0.

```
main.c
main.c:3:1: warning: return type defaults to 'int' [-Wimplicit-int]
3 | main()
  | ~~~~

Enter the No.of nodes in graph: 3

Is there are any edge from 1 to 1? 0
Is there are any edge from 1 to 2? 1
Is there are any edge from 1 to 3? 1
Is there are any edge from 2 to 2? 0
Is there are any edge from 2 to 3? 1
Is there are any edge from 3 to 3? 0

Which Routing table do u want to find the cost of which destination? 1 3

Enter the cost of 1 node to its neighbour 2:2
Enter the Routing table entry to 2 to 3: 5
Enter the cost of 1 node to its neighbour 3:4
Enter the Routing table entry to 3 to 3: 0

Estimated cost from node 1 to 3 is 4 via the node is 3

...Program finished with exit code 0
Press ENTER to exit console.
```

Task 7: Implementation of Dijkstra's algorithm.

Description:

It is a static routing algorithm. It is used to build a graph of the subnet, with each node of graph representing a router and each arc of the graph representing a communication line. To choose a route between a given pair of routers, the algorithm just finds the shortest path between them on the graph. Different ways of measuring the path length is the number of Hops, Geographical distance in kmts, Mean Queuing delay, Transmission delay, Functions of distance, Bandwidth, Average traffic, communication cost etc.,

This algorithm is given in Fig. 5-8. The global variables n and $dist$ describe the graph and are initialized before `shortest_path` is called. The only difference between the program and the algorithm described above is that in Fig. 5-8, we compute the shortest path starting at the terminal node, t , rather than at the source node, s . Since the shortest path from t to s in an undirected graph is the same as the shortest path from s to t , it does not matter at which end we begin (unless there are several shortest paths, in which case reversing the search might discover a different one). The reason for searching backward is that each node is labeled with its predecessor rather than its successor. When the final path is copied into the output variable, `path`, the path is thus reversed. By reversing the search, the two effects cancel, and the answer is produced in the correct order.

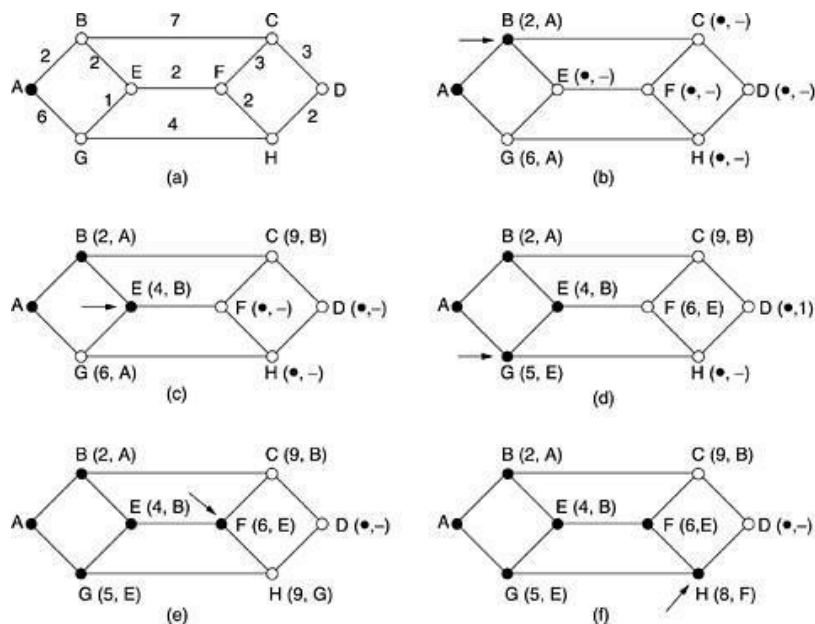


Figure: The first five steps used in computing the shortest path from A to D. The arrows indicate the working. Finally, Destination 'D' is relabeled as D(10, H).

The path is (D-H-F-E-B-A) as follows:

$$\begin{aligned}
 D(10, H) &= H(8, F) \\
 &= F(6, E) \\
 &= E(4, B) \\
 &= B(2, A) \\
 &= A
 \end{aligned}$$

Program:

```
#include<stdio.h>
#include<conio.h>
#define MAX_NODES 50
#define INFINITY 1000
int n;
int shortest_path(int,int,int a[]);
char name[MAX_NODES];
int dist[MAX_NODES][MAX_NODES];
void main()
{
    int i,j,l;
    char sour,dest,c=65;
    int path[MAX_NODES];
    clrscr();
    printf("Enter the no. of nodes:");
    scanf("%d",&n);
    printf("\nEnter the distances between each & every node\n");
    printf("if there is no path then enter 0 as its distance\n\n");
    printf("\n\t");
    for(i=0;i<n;i++)
    {
        name[i]=c++;
        printf("%c\t",name[i]);
    }
    printf("\n\n");
    for(i=0;i<n;i++)
    {
        printf("%c\t",name[i]);
        for(j=0;j<n;j++)
```

```

        {
            scanf("%d",&dist[i][j]);

        }
    }
A:
    printf("\n Enter the source name:");
    sour=getch();
    getch();
    for(i=0;i<n;i++)
    {
        if(name[i]==sour)
        {
            break;
        }
    }
    if(i==n)
    {
        printf("\n NO node with the given name\n");
        goto A;
    }
B:
    printf("\n\n Enter the distination name:");
    dest=getch();
    getch();
    for(j=0;j<n;j++)
    {
        if(name[j]==dest)
        {
            break;

```

```

        }
    }
    if(j==n)
    {
        printf("\n NO node with given destination name\n");

        goto B;
    }
    l=shortest_path(i,j,path);
    printf("The shortest path is ");
    for(i=(l-1);i>0;i--)
    {
        printf("%c->",name[path[i]]);
    }
    printf("%c",name[path[i]]);
    getch();
}

int shortest_path(int s,int d,int p[])
{
    struct state
    {
        int predecessor;
        int length;
        int label;
    }state[MAX_NODES];
    int i,k,min,m;
    for(i=0;i<n;i++)
    {
        state[i].predecessor=-1;
        state[i].length=INFINITY;
    }

```

```

        state[i].label=0;
    }
    state[s].length=0;
    state[s].label=1;
    k=s;
    do
    {
        for(i=0;i<n;i++)

        {
            if(dist[k][i]!=0 && state[i].label==0)
            {
                if(state[k].length+dist[k][i]<state[i].length)
                {
                    state[i].predecessor=k;
                    state[i].length=state[k].length+dist[k][i];
                }
            }
        }
        k=0;
        min=INFINITY;
        for(i=0;i<n;i++)
        {
            if(state[i].label==0 && state[i].length<min)
            {
                min=state[i].length;
                k=i;
            }
        }
        state[k].label=1;
    }

```



```

    }while(k!=d);

    i=0;

    k=d;

    do
    {

        p[i++]=k;

        k=state[k].predecessor;

    }while(k>=0);

    printf("\n\n The shortest distance %d \n\n",state[d].length);

    return(i);

}

```

Output:

```

venky@localhost:~$ vi dijk.c
[venky@localhost ~]$ cc dijk.c
[venky@localhost ~]$ ./a.out

enter the number of nodes:3

enter the cost matrix:
0 2 1
2 0 3
1 3 0

enter the source matrix:1

shortest path :
1->2,cost 2
1->3,cost 1
[venky@localhost ~]$

```

Task 8: : Implementation of Congestion control using leaky bucket algorithms.

Description:

The congesting control algorithms are basically divided into two groups: open loop and closed loop. Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made. Open loop algorithms are further divided into ones that act at source versus ones that act at the destination.

In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithm, the source deduces the existence of congestion by making local observation, such as the time needed for acknowledgment to come back.

The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. For subnets that use virtual circuits internally, these methods can be used at the network layer.

Another open loop method to help manage congestion is forcing the packet to be transmitted at a more predictable rate. This approach to congestion management is widely used in ATM networks and is called traffic shaping.

The other method is the leaky bucket algorithm. Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more process are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulated by the host operating system. In fact it is nothing other than a single server queuing system with constant service time.

The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flow of packet onto the network, smoothing out bursts and greatly reducing the chances of congestion

The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flow of packet onto the network, smoothing out bursts and greatly reducing the chances of congestion.

Program:

```
#include<stdio.h>

#include<stdlib.h>

#define bucketSize 512

void bktInput(int a,int b) {
    if(a>bucketSize)
        printf("\n\t\tBucket overflow");
    else {
        sleep(2);
```

```

while(a>b){

    printf("\n\t\t bytes outputted."); a-=b;
    sleep(2);
}

if (a>0)
    printf("\n\t\tLast %d bytes sent\t",a);
    printf("\n\t\tBucket output successful");
}
}

void main() {
    int op,i,pktSize;
    rand();

    printf("Enter output rate : ");
    scanf("%d",&op); for(i=1;i<=5;i++)
    {
        sleep(2);

        pktSize=rand();

        printf("\nPacket no:%d \tPacket size:%d ",i,pktSize);

        bktInput(pktSize,op);
    }
}

```

Output:

```

main.c
main.c:18:12: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
8 |     sleep(2);
  |     ^~~~~
Enter output rate : 100
Packet no:1      Packet size:846930886
Bucket overflow
Packet no:2      Packet size:1481692777
Bucket overflow
Packet no:3      Packet size:1714636915
Bucket overflow
Packet no:4      Packet size:1957747793
Bucket overflow
Packet no:5      Packet size:1424238335
Bucket overflow
...Program finished with exit code 0
Press ENTER to exit console.

```

Task 9: Implementation using Socket TCP both client and server.

Description:

What is socket programming?

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

Stages for server

Socket creation:

```
int sockfd = socket(domain, type, protocol)
```

sockfd: socket descriptor, an integer (like a file-handle)

domain: integer, communication domain e.g., AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)

type: communication type

SOCK_STREAM: TCP(reliable, connection oriented)SOCK_DGRAM: UDP(unreliable, connectionless)

protocol: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)

Setsockopt:

```
int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen);
```

This helps in manipulating options for the socket referred by the filedescriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: “address already in use”.

Bind:

```
int bind(int sockfd, const struct sockaddr *addr,  
socklen_t addrlen);
```

After creation of the socket, bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR_ANY tospecify the IP address.

Listen:

```
int listen(int sockfd, int backlog);
```

It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the clientmay receive an error with an indication of ECONNREFUSED.

Accept:

```
int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t
*addrlen);
```

It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, connection is established between client and server, and they are ready to transfer data.

Stages for Client

Socket connection: Exactly same as that of server's socket creation

```
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.

Program:**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 sockfd)

{

    char buff[MAX];int n;

    for (;;) {

        bzero(buff, MAX);

        // read the message from client and copy it in buffer

        read(sockfd, buff, sizeof(buff));

        // print buffer which contains the client contents
```

```

    printf("From client: %s\t To client : ", buff);

    bzero(buff, MAX);

// copy server message in the buffer

    n = 0;
    while ((buff[n++] = getchar()) != '\n') ;
// and send that buffer to client
write(sockfd, buff, sizeof(buff));
// if msg contains "Exit" then server exit and chat ended.
    if (strncmp("exit", buff, 4) == 0) {
        printf("Server Exit...\n");break;
    }
}
}

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);
len = sizeof(cli);
// Accept the data packet from client and verification
connfd = accept(sockfd, (SA*)&cli, &len);
if (connfd < 0) {
printf("server acccept failed...\n");exit(0);
}
else
    printf("server acccept 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 ((strcmp(buff, "exit", 4)) == 0) {
printf("Client Exit...\n");break;
}
}
}
int main()
{
int sockfd, connfd;
struct sockaddr_in servaddr, cli;
```



```

// socket create and varification
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);
}

```

Output:

Compilation – Server side:

```
gcc server.c -o server (or) gcc server.c
```

```
./server (or) ./a.outClient side:
```

```
gcc client.c -o client
```

```
./client
```

Server side:

Socket successfully created..Socket successfully binded..Server listening..

server accept the client...From client: hi

To client : helloFrom client: exitTo client : exit Server Exit...

Client side:

Socket successfully created..connected to the server..

Enter the string : hiFrom Server : hello

Enter the string : exitFrom Server : exit Client Exit...

Task 10: Implementation using Socket UDP both client and server programs.

Description:

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

UDP Client/Server function calls

The entire process can be broken down into following steps :

UDP Server :

Create UDP socket.

Bind the socket to server address.

Wait until datagram packet arrives from client.

Process the datagram packet and send a reply to client.Go back to Step 3.

UDP Client :

Create UDP socket.

Send message to server.

Wait until response from server is recieved.

Process reply and go back to step 2,if necessary.

Close socket descriptor and exit.Necessary Functions :

int socket(int domain, int type, int protocol) Creates an unbound socket in the specified domain.Returns socket file descriptor.

Arguments :

domain – Specifies the communication

domain (AF_INET for IPv4/ AF_INET6 for IPv6)type – Type of socket to be created

(SOCK_STREAM for TCP / SOCK_DGRAM for UDP)

protocol – Protocol to be used by socket.

0 means use default protocol for the address family.

int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen)

Assigns address to the unbound socket.

Arguments :

sockfd – File descriptor of socket to be binded

addr – Structure in which address to be binded to is specifiedaddrlen – Size of addr structure

```
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags, const structsockaddr *dest_addr, socklen_t addrlen)
```

Send a message on the socket

Arguments :

sockfd – File descriptor of socket

buf – Application buffer containing the data to be sent

len – Size of buf application buffer

flags – Bitwise OR of flags to modify socket behaviour

dest_addr – Structure containing address of destination

addrlen – Size of dest_addr structure

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,  
struct sockaddr *src_addr, socklen_t *addrlen)
```

Receive a message from the socket.

Arguments :

sockfd – File descriptor of socket

buf – Application buffer in which to receive data len – Size of buf application buffer

flags – Bitwise OR of flags to modify socket behaviour

src_addr – Structure containing source address is returned

addrlen – Variable in which size of src_addr structure is returned

int close(int fd) Close a file descriptor**Arguments :**

fd – File descriptor

In the below code, exchange of one hello message between server and client is shown to demonstrate the model.

Program:

UDPSERVER.C

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

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

```
#include <unistd.h>
```

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

```
#include <sys/types.h>
```

```
#include <sys/socket.h>
```

```

#include <arpa/inet.h>
#include <netinet/in.h>
#define PORT 8080
#define MAXLINE 1024
// Driver code
int main() {
    int sockfd;
    char buffer[MAXLINE];
    char *hello = "Hello from server";
    struct sockaddr_in servaddr, cliaddr;

    // Creating socket file descriptor
    if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));

    // Filling server information
    servaddr.sin_family = AF_INET;

    // IPv4
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);

    // Bind the socket with the server address
    if ( bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0 )
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    int len, n;
    len = sizeof(cliaddr);
    // len is value/result

```

```

n = recvfrom(sockfd, (char *)buffer, MAXLINE, MSG_WAITALL, ( struct sockaddr *) &cliaddr,&len);
buffer[n] = '\0';
printf("Client : %s\n", buffer);
sendto(sockfd, (const char *)hello, strlen(hello), MSG_CONFIRM, (const struct sockaddr *) &cliaddr,len);
printf("Hello message sent.\n");
return 0;
}

```

UDP CLIENT.C

// Client side implementation of UDP client-server model

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#define PORT 8080
#define MAXLINE 1024
// Driver code
int main() {
int sockfd;
char buffer[MAXLINE];
char *hello = "Hello from client";
struct sockaddr_in servaddr;
// Creating socket file descriptor
if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
perror("socket creation failed");
exit(EXIT_FAILURE);
}

```

```

memset(&servaddr, 0, sizeof(servaddr));

// Filling server information

servaddr.sin_family = AF_INET; servaddr.sin_port = htons(PORT); servaddr.sin_addr.s_addr =
INADDR_ANY;

int n, len;

sendto(sockfd, (const char *)hello, strlen(hello), MSG_CONFIRM, (const struct sockaddr *) &servaddr,
sizeof(servaddr));

printf("Hello message sent.\n");

n = recvfrom(sockfd, (char *)buffer, MAXLINE, MSG_WAITALL, (struct sockaddr *) &servaddr, &len);
buffer[n] = '\0';

printf("Server : %s\n", buffer);

close(sockfd); return 0;

}

```

OUTPUT :

vi udpserver.c

gcc udpserver.c

./a.out

Client : Hello from clientHello message sent.

vi udpclient.c

gcc udpclient.c

./a.out

Hello message sent. Server : Hello from server