NEW EDITION

NETWORKING LABORATORY

(V semester of B.Tech)

As per the curricullam and syllabus

of

Bharath Institute of Higher Education & Research

SOFTWARE ENGINEERING LABORATORY

PREPARED BY
Mr.D.Bhaskar

Bharath

INSTITUTE OF HIGHER EDUCATION AND RESEARCH

(Declared as Deemed-to-be University under section 3 of UGC Act, 1956)

(Vide Notification No. F.9-5/2000 - U.3, Ministry of Human Resource Development, Govt. of India, dated 4th July 2002)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LAB MANUAL

SUBJECTNAME:NETWORKING lABORATORY

SUBJECT CODE:BCS5L1

Regulation R2015

(2016-2017)

BCS5L1	NETWORKING LABORATORY	L	T	P	C
	Total Contact Hours - 30	0	0	3	2
	Prerequisite –Computer Networks				
	Lab Manual Designed by – Dept. of Computer Science and Engineering.				

OBJECTIVES

The main Objective of this course is student know about Networking Concepts and Protocols.

COURSE OUTCOMES (COs)

CO1	Develop knowledge to implement client server applications.
CO2	Develop skills in UNIX socket programming.
CO3	Develop skills to use simulation tools.
CO4	Analyze the performance of network protocols.
CO5	Analyze the network traffic.
CO6	Establish a Connection using TCP/IP Protocol.

MAPPING BETWEEN COURSE OUTCOMES & PROGRAM OUTCOMES (3/2/1 INDICATES STRENGTH OF CORRELATION) 3- High, 2- Medium, 1-Low

COs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1			3	3	2	1	2	2	2	1	2	S	2	3	
CO2			3	3		2		2			2		2	3	
CO3	2		2	3				2				2	2	3	
CO4			3	3	3		3		2		2		2	3	
CO5			2	2		2		3		2			2	3	
CO6	1		3	3								2	2	3	

Category	Professional Core (PC)
Approval	37th Meeting of Academic Council, May 2015

LIST OF EXPERIMENTS:

1. Implement a simple TCP client-server where in a server acts as a time and date server.
2. Print a client address at server end.
3. Write a program in which a process is made to handle posix signals.
4. Program an echo UDP Server.
5. Create a daemon.
6. Create a simple Chat Program.
7. Create a simple out of band Sending and Receiving program.
8. Program to capture each packet and examine its checksum field.

BCS5L1 NETWORKING LABORATORY

LIST OF EXPERIMENTS

1. Implement a simple TCP client-server where in a server acts as a time and date server.
2. Print a client address at server end.
3. Write a program in which a process is made to handle posix signals.
4. Program an echo UDP Server.
5. Create a daemon.
6. Create a simple Chat Program.
7. Create a simple out of band Sending and Receiving program.
8. Program to capture each packet and examine its checksum field.

CONTENT

EX NO 1: IMPLEMENT A SIMPLE TCP CLIENT-SERVER WHERE IN A SERVER ACTS AS A TIME AND DATE SERVER.

AIM:

To write a program to implement TCP Server in C

ALGORITHM:

TCP SERVER

1. using create(), Create TCP socket.
2. using bind(), Bind the socket to server address.
3. using listen(), put the server socket in a passive mode, where it waits for the client to approach the server to make a connection
4. using accept(), At this point, connection is established between client and server, and they are ready to transfer data.
5. Go back to Step 3

TCP CLIENT

1. Create TCP socket.
2. connect newly created client socket to 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;
    // infinite loop for chat
    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);
        n = 0;
        // copy server message in the buffer
        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;
        }
    }
}

// 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 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);
}
t 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));
```

TCPCLIENT

```
#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 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);
}
```

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 acccept 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...

RESULT:

Thus the implementation of a simple TCP client-server where in a server acts as a time and date is executed and output verified successfully

Ex No : 2 PRINT A CLIENT ADDRESS AT SERVER END

AIM:

To write a program in c to print the client address at the server.

ALGORITHM:

Server side Algorithm

STEP 1: Start the program.
STEP 2: Declare the variables and structure for the socket.
STEP 3: The socket is binded at the specified port.
STEP 4: Using the object the port and address are declared.
STEP 5: The listen and accept functions are executed.
STEP 6: If the binding is successful then the client address is printed .
STEP 7: Execute the client program.
STEP 8: close the socket function.

Client side algorithm

STEP 1: Start the program.
STEP 2: Declare the variables and structure.
STEP 3: Socket is created and connect function is executed.
STEP 4: If the connection is successful then server sends the message.
STEP 5: Get the server IP address from the client.
STEP 6: Print the message that is read from the client.
STEP 6: Stop the program
de In C Network Programming
STEP 7: Execute the client program.
STEP 8: close the socket function.

Client side algorithm
STEP 1:
Server programming source code in c Print the client address at the server

```
#include<netinet/in.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>
main()
{
struct sockaddr_in sa;
struct sockadd_in cli;
int socketfd, conntfd;
int len;
char str[100];
time_t tick;
socketfd=socket(AF_INET,SOCK_STREAM,0);
if(socketfd<0)
{
print("\n ERROR IN SOCKET\n");
exit(0);
{
else
printf("\n SOCKET OPENED\n");
bzero(&sa, sizeof(sa));
sa.sin_family=AF_INET;
sa.sin_port=htons(5600);
```

```
if(bind(socketfd,(struct sockaddr*)&sa, sizeof(sa))<0)
{
printf("\nERROR FINDING FAILED\n");
exit(0);
}
else
printf("\nBINDED SUCCESSFYLLY\n);
listen(socketfd,50);
for(;;)
{
len=sizeoof(cli);
conntfd=accept(socketfd,(struct sockaddr*)&cli,&len);
printf("\nACCEPTED\n);
printf("\nREQUEST FROM%sPORT%d\n",inet-ntop(AF_INET,&cli.
sin_addr,str,sizeof(str)),htons(cli.sin_port));
tick=time(NULL);
snprintf(str,sizeof(str),"%s",ctime(&tick));
write (conntfd,str, sizeof(str));
}
}
```

Client Program Side Source code programming How to Print the client address at the server

```
#include<netinet/in.h>
#include<sys/socket.h>
main()
{
struct sockaddr_in sa;
struct sockaddr_in cli;
int n,socketfd;
int len;
char buff[100]
socketfd=socket(AF_INET,SOCK_STREAM,0);
if(socketfd<0)
{
printf("\ERROR IN SOCKET\n");
exit(0);
}
else
printf("\n SOCKET IS OPENED\N");
bzero(&sa,sizeof(sa));
sa.sin_family=AF_INET;
sa.sin_port=htons(5600);
if(connect(socketfd,(struct sockaddr*)&sa,sizeof(sa))<0)
{
print("ERROR IN CONNECTION\n");
exit(0);
}
else
printf("\nCONNECTED SUCCESSFULLY\n");
if(n=read(socketfd,buff,sizeof(buff))<0)
{
printf("\nERROR IN READING\n");
exit(0);
}
else{
print("\nMESSAGE READ%s",buff);
```

```
buff[n]='\0';
printf("%s",buff);
}
}
```

Output CS1305 Network Lab

CLIENT side status when socket open

Socket is opened

Connected successfully

Message read Wed Feb 711:52:58 2007

SERVER side status after connect with client and print client address

Socket is opened

Binded successfully

Accepted

Request from 127.0.0.1 port 33584

RESULT :

Thus the program to print the client address at the server is executed and verified.

PROGRAM TO HANDLE POSIX SIGNALS.

AIM:

To write a program to handle posix Signals

ALGORITHM:

1. JVM invoke the main().
2. main() pushed onto call stack, before invoking methodA().
3. methodA() pushed onto call stack, before invoking methodB().
4. methodB() pushed onto call stack, before invoking methodC().
5. methodC() completes.
6. methodB() popped out from call stack and completes.
7. methodA() popped out from the call stack and completes.
8. main() popped out from the call stack and completes. Program exits.

PROGRAM:

```
public class MethodCallStackDemo {
  public static void main(String[] args) {
    System.out.println("Enter main()");
    methodA();
    System.out.println("Exit main()");
  }

  public static void methodA() {
    System.out.println("Enter methodA()");
    methodB();
    System.out.println("Exit methodA()");
  }

  public static void methodB() {
    System.out.println("Enter methodB()");
    methodC();
    System.out.println("Exit methodB()");
  }

  public static void methodC() {
    System.out.println("Enter methodC()");
    System.out.println("Exit methodC()");
  }
}
```

OUTPUT:

```
Enter main()
Enter methodA()
Enter methodB()
Enter methodC()
Exit methodC()
Exit methodB()
Exit methodA()
```

Exit main()
Exception_MethodCallStack.png

RESULT:
Thus the program to write a program to handle posix Signals is executed and the output is verified.

Ex No : 4 PROGRAM AN ECHO UDP SERVER

AIM:

To write a program in c echo UDP Server.

ALGORITHM:

UDP Server :

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

UDP Client

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

UDP SERVER

```
// Server 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 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/resuslt

    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

```
// 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

```
$ ./server

Client : Hello from client

Hello message sent.

$ ./client

Hello message sent.

Server : Hello from server
```

RESULT

Thus the program for echo UDP server is executed and verified successfully
|

Ex No : 4 Create a daemon

AIM:

To write a program in c echo UDP Server.

ALGORITHM:

Create a normal process (Parent process)
Create a child process from within the above parent process
The process hierarchy at this stage looks like : TERMINAL -> PARENT PROCESS -> CHILD PROCESS
Terminate the the parent process.
The child process now becomes orphan and is taken over by the init process.
Call setsid() function to run the process in new session and have a new group.
After the above step we can say that now this process becomes a daemon process without having a controlling terminal.
Change the working directory of the daemon process to root and close stdin, stdout and stderr file descriptors.

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
int main(int argc, char* argv[])
{
FILE *fp= NULL;
pid_t process_id = 0;
pid_t sid = 0;
// Create child process
process_id = fork();
// Indication of fork() failure
if (process_id < 0)
{
printf("fork failed!\n");
// Return failure in exit status
exit(1);
}
// PARENT PROCESS. Need to kill it.
if (process_id > 0)
{
printf("process_id of child process %d \n", process_id);
// return success in exit status
exit(0);
}
//unmask the file mode
umask(0);
//set new session
sid = setsid();
```

```
if(sid < 0)

{
// Return failure
exit(1);
}
// Change the current working directory to root.
chdir("/");
// Close stdin. stdout and stderr
close(STDIN_FILENO);
close(STDOUT_FILENO);
close(STDERR_FILENO);
// Open a log file in write mode.
fp = fopen ("Log.txt", "w+");
while (1)
{
//Dont block context switches, let the process sleep for some time
sleep(1);
fprintf(fp, "Logging info...\n");
fflush(fp);
// Implement and call some function that does core work for this daemon.
}
fclose(fp);
return (0);
}
```

OUTPUT

```
$ gcc -Wall deamon.c -o deamon
$ sudo ./deamon
process_id of child process 2936
```

Just observe that the control immediately came back to the terminal ie the daemon is now not associated to any terminal.

```
$
$ tail -f /Log.txt
Logging info...
Logging info...
Logging info...
Logging info...
Logging info...
Logging info...
Logging info...
Logging info...
Logging info...
Logging info...
```


RESULT

Thus the program for echo UDP server is executed and verified successfully

EXNO: 6 CHAT APPLICATION

AIM:

To write a network program for implementing chat using TCP socket.

ALGORITHM:

1: Client sends the request to server

2: Server runs the request and the connection with the client that has requested the server.

3: The client sends the message to server.

4: The server process it and it is displayed (i.e.,) replier by sending the message to clientalso displayed.

5: Stop the program.

SERVER PROGRAM:

```
import java.io.*; import
java.net.*; public class
chatserver
{
public static void main(String args[])throws Exception
{
DataInputStream din=null;
DataOutputStream dout=null;
Socket c=null;
ServerSocket m=null;
DataInputStream stdin=new DataInputStream(System.in);try
{
m=new ServerSocket(68);
c=m.accept();
din=new DataInputStream(c.getInputStream()); dout=new
DataOutputStream(c.getOutputStream());
}
catch(Exception e)
{
}
while(c!=null)
{
String m2; System.out.println("Server");
while(true)
{
String m1=din.readLine();
```

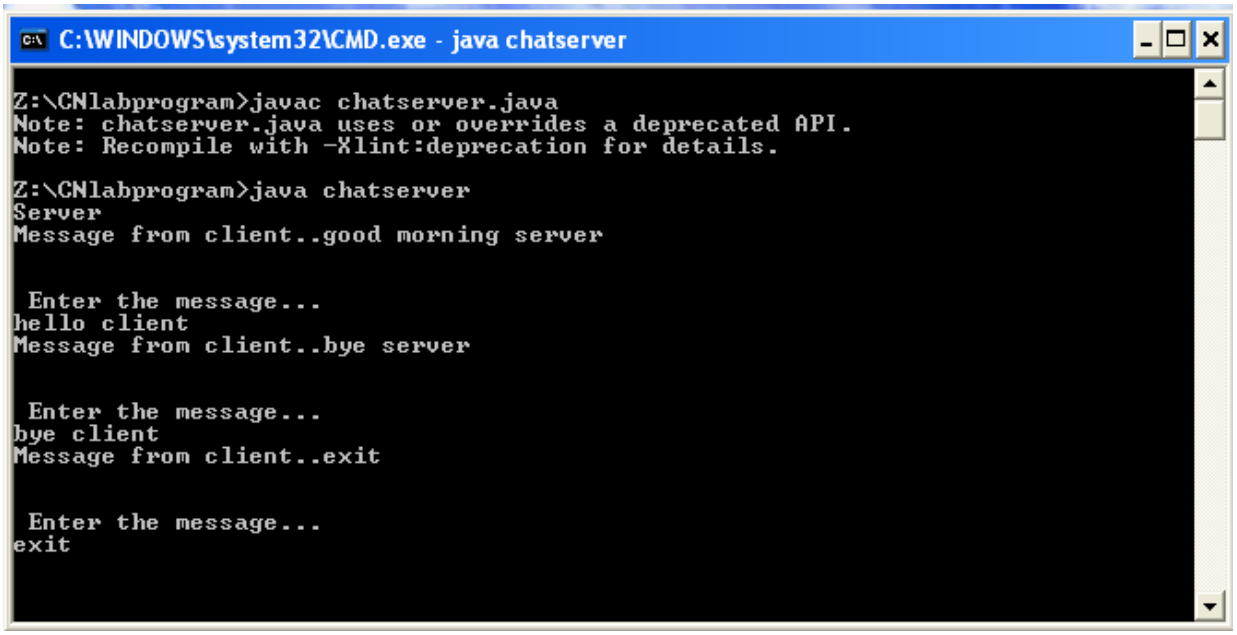
```
System.out.println("Message from client.."+m1);
System.out.println("\n\n Enter the message...");
m2=stdin.readLine();
dout.writeBytes(""+m2);dout.writeBytes("\n");
}}
din.close();
dout.close();
c.close();
m.close();
}}
```

CLIENT PROGRAM:

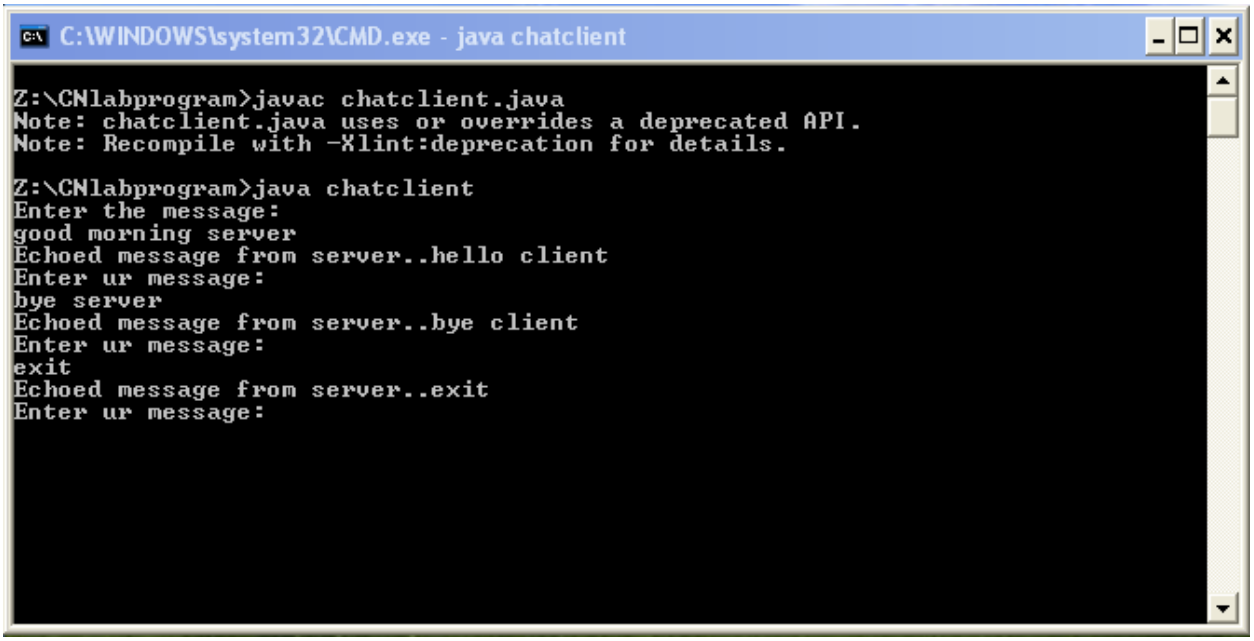
```
import java.io.*; import
java.net.*; public class
chatclient
{
public static void main(String args[])throws Exception
{
Socket c=null; DataInputStream
uin=null; DataInputStream
din=null; DataOutputStream
dout=null;try
{
c=new Socket("localhost",68); uin=new
DataInputStream(System.in);
din=new DataInputStream(c.getInputStream()); dout=new
DataOutputStream(c.getOutputStream());
}
catch(Exception e)
{
}
if(c!=null)
{
String inp;
System.out.println("Enter the message:");
while((inp=uin.readLine())!=null)
{
dout.writeBytes(""+inp);dout.writeBytes("\n");
System.out.println("Echoed message from server.."+din.readLine());
System.out.println("Enter ur message:");
}}
din.close();
dout.close();
c.close();
}}
```

OUTPUT:

SERVER:



CLIENT:



RESULT:

Thus the program for CHAT APPLICATION was executed and output is verifiedsuccessfully.

EXNO: 7 SIMPLE OUT OF BAND SENDING AND RECEIVING PROGRAM.

AIM:

To write a JAVA program to perform file transfer using TCP sockets

ALGORITHM:

SERVER:

1. Create a server socket
2. Create a new file
3. Get the input from the file using FileInputStream
4. Copy the contents of the file to the FileOutputStream
5. Close the socket
6. Stop the program

CLIENT:

1. Create a socket
2. Get the input from the socket using InputStream
3. Copy the contents of the file to the FileOutputStream
4. Print the contents of the file
5. Close the socket
6. Stop the program

PROGRAM

FILE CLIENT

```
import java.io.*; import java.net.*; import java.util.*; class Clientfile
{
public static void main(String args[])
{
try
{
BufferedReader in=new BufferedReader(new InputStreamReader(System.in)); Socket clsct=new Socket("127.0.0.1",139);
```

```
DataInputStream din=new DataInputStream(clsct.getInputStream()); DataOutputStream dout=new DataOutputStream(clsct.getOutputStream()); System.out.println("Enter the file name:");
String str=in.readLine(); dout.writeBytes(str+'\n'); System.out.println("Enter the new file name:"); String str2=in.readLine();
String str1,ss;
FileWriter f=new FileWriter(str2);
char buffer[]; while(true)
{
str1=din.readLine(); if(str1.equals("-1")) break; System.out.println(str1); buffer=new char[str1.length()];
str1.getChars(0,str1.length(),buffer,0); f.write(buffer);
}
f.close(); clsct.close();
}
catch (Exception e)
{
System.out.println(e);
}}}
```

FILE SERVER

```
import java.io.*; import java.net.*; import java.util.*; class Serverfile
{
public static void main(String args[])
{
try
{
ServerSocket obj=new ServerSocket(139); while(true)
{
Socket obj1=obj.accept();
DataInputStream din=new DataInputStream(obj1.getInputStream()); DataOutputStream dout=new DataOutputStream(obj1.getOutputStream()); String str=din.readLine();
FileReader f=new FileReader(str); BufferedReader b=new BufferedReader(f); String s;
```

```
while((s=b.readLine())!=null)
{
System.out.println(s); dout.writeBytes(s+'\n');
}
f.close(); dout.writeBytes("-1\n");
}}
catch(Exception e)
{ System.out.println(e);}
}}
```

OUTPUT:

```
C:\Program Files\Java\jdk1.5.0\bin>javac Clientfile.java
Note: Clientfile.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Program Files\Java\jdk1.5.0\bin>java Clientfile
Enter the file name:
Clientfile.java
Enter the new file name:
```

```
C:\Program Files\Java\jdk1.5.0\bin>javac Serverfile.java
Note: Serverfile.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Program Files\Java\jdk1.5.0\bin>java Serverfile
import java.io.*;
import java.net.*;
import java.util.*;
class Clientfile
{
public static void main(String args[])
{
try
{
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
Socket clsct=new Socket("127.0.0.1",139);
DataInputStream din=new DataInputStream(clsct.getInputStream());
DataOutputStream dout=new DataOutputStream(clsct.getOutputStream());
System.out.println("Enter the file name:");
String str=in.readLine();
dout.writeBytes(str+'\n');
System.out.println("Enter the new file name:");
String str2=in.readLine();
String str1,ss;
FileWriter f=new FileWriter(str2);
char buffer[];
while(true)
{
str1=din.readLine();
if(str1.equals("-1")) break;
System.out.println(str1);
buffer=new char[str1.length()];
str1.getChars(0,str1.length(),buffer,0);
f.write(buffer);
}
f.close();
clsct.close();
}
catch (Exception e)
{
System.out.println(e);
}
}
}
```

RESULT:

Thus the program for FILE TRANSFER was executed and the output is verified successfully.

EX NO:8 PROGRAM TO CAPTURE EACH PACKET AND EXAMINE ITS CHECKSUM FIELD.

AIM: To write a program in java to capture each packet and examine its checksum field.

ALGORITHM:

At the Sender Side :

1)First, ask for the length of data to send, in order to ascertain the number of segments.
2)Then perform a ones complement of each data being entered simultaneously adding them. This means the sum would not be required to be complemented again.
3)Then send the data along with the computed checksum to the server.
4)Then report the successful transference of the message or otherwise depending on the feedback received from the server.

At the Receiver Side :

1)The receiver waits for data to arrive from the sender.
2)Once the data along with checksum is received from the sender, the receiver complements what is received and simultaneously keeps on adding them.
3)Finally, the receiver complements the above sum, and checks whether the result is a zero or not and reports the same to the sender. A zero would indicate a successful data transfer and anything else would indicate an error in the data that is received.

PROGRAM:

```
package checksum_sender;

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

public class Checksum_Sender
{
   // Setting maximum data length
   private int MAX = 100;

   // initialize socket and I/O streams
   private Socket socket = null;
```

```
private ServerSocket servsock = null;
private DataInputStream dis = null;
private DataOutputStream dos = null;

public Checksum_Sender(int port) throws IOException
{
   servsock = new ServerSocket(port);

   // Used to block until a client connects to the server
   socket = servsock.accept();

   dis = new DataInputStream(socket.getInputStream());
   dos = new DataOutputStream(socket.getOutputStream());

   while (true)
   {
      int i, l, sum = 0, nob;
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter data length");
      l = sc.nextInt();

      // Array to hold the data being entered
      int data[] = new int[MAX];

      // Array to hold the complement of each data
      int c_data[] = new int[MAX];

      System.out.println("Enter data to send");

      for (i = 0; i < l; i++)
      {
         data[i] = sc.nextInt();

         // Complementing the entered data
         // Here we find the number of bits required to represent
         // the data, like say 8 requires 1000, i.e 4 bits
         nob = (int)(Math.floor(Math.log(data[i]) / Math.log(2))) + 1;

         // Here we do a XOR of the data with the number 2^n -1,
         // where n is the nob calculated in previous step
```

```
            c_data[i] = ((1 << nob) - 1) ^ data[i];

            // Adding the complemented data and storing in sum
            sum += c_data[i];
        }

        // The sum(i.e checksum) is also sent along with the data
        data[i] = sum;
        l += 1;

        System.out.println("Checksum Calculated is : " + sum);
        System.out.println("Data being sent along with Checkum.....");

        // Sends the data length to receiver
        dos.writeInt(l);

        // Sends the data one by one to receiver
        for (int j = 0; j < l; j++)
            dos.writeInt(data[j]);

        // Displaying appropriate message depending on feedback received
        if (dis.readUTF().equals("success"))
        {
            System.out.println("Thanks for the feedback!! Message received
                                Successfully!");
            break;
        }


        else if (dis.readUTF().equals("failure"))

        {

            System.out.println("Message was not received successfully!");

            break;

        }

    }
```

```
        // Closing all connections
        dis.close();
        dos.close();
        socket.close();
    }

    // Driver Method
    public static void main(String args[]) throws IOException
    {
        Checksum_Sender cs = new Checksum_Sender(45678);
    }
}

// Java code for Checksum_Receiver
package checksum_sender;

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

public class Checksum_Receiver {

    // Initialize socket and I/O streams
```

```
private Socket s = null;
private DataInputStream dis = null;
private DataOutputStream dos = null;

// Constructor to put ip address and port
public Checksum_Receiver(InetAddress ip,int port)throws IOException
{

   // Opens a socket for connection
   s = new Socket(ip,port);

   dis = new DataInputStream(s.getInputStream());
   dos = new DataOutputStream(s.getOutputStream());

   while (true)
   {   Scanner sc = new Scanner(System.in);
      int i, l, nob, sum = 0, chk_sum;

      // Reads the data length sent by sender
      l = dis.readInt();

      // Initializes the arrays based on data length received
      int c_data[] = new int[l];
      int data[] = new int[l];
```

```
System.out.println("Data received (alond with checksum) is");

for(i = 0; i< data.length; i++)
{
   // Reading the data being sent one by one
   data[i] = dis.readInt();
   System.out.println(data[i]);

   // Complementing the data being received
   nob = (int)(Math.floor(Math.log(data[i]) / Math.log(2))) + 1;
   c_data[i] = ((1 << nob) - 1) ^ data[i];

   // Adding the complemented data
   sum += c_data[i];
}
System.out.println("Sum(in ones complement) is : "+sum);

// Complementing the sum
nob = (int)(Math.floor(Math.log(sum) / Math.log(2))) + 1;
sum = ((1 << nob) - 1) ^ sum;
System.out.println("Calculated Checksum is : "+sum);

// Checking whether final result is 0 or something else
```

```
        // and sending feedback accordingly
        if(sum == 0)
        {
            dos.writeUTF("success");
            break;
        }
        else
        {
            dos.writeUTF("failure");
            break;
        }
    }

    // Closing all connections
    dis.close();
    dos.close();
    s.close();
}

// Driver Method
public static void main(String args[])throws IOException
{
    // Getting ip address on which the receiver is running
    // Here, it is "localhost"
```

```
        InetAddress ip = InetAddress.getLocalHost();

        Checksum_Receiver cr = new Checksum_Receiver(ip,5000);

    }

}
```

i. Output :

```
C:\Users\Anshul>java Checksum_Receiver
Data received (alond with checksum) is
67
43
0
22
90
Sum(in ones complement) is : 127
Calculated Checksum is : 0
```

```
C:\Users\Anshul>java Checksum_Sender
Enter data length
4
Enter data to send
67
43
0
22
Checksum Calculated is : 90
Data being sent along with Checkum.....
Thanks for the feedback!! Message received Successfully!
```

RESULT: Thus the program in java to capture each packet and examine its checksum field is executed and the output is verified.