**Experiment No:1**

**Aim:-** To learn handling and configuration of networking hardware like RJ45 connector, CAT-6 cable, crimping tool, etc.

**Apparatus Required:**- RJ-45 connector, Crimping Tool, CAT-6 Cable

**Method:-** Using a crimping tool: Strip the cable back 1 inch (25 mm) from the end. Insert the cable

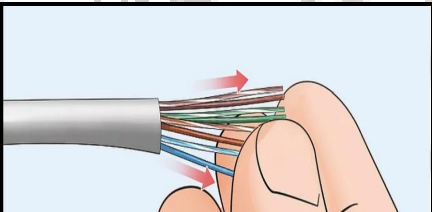
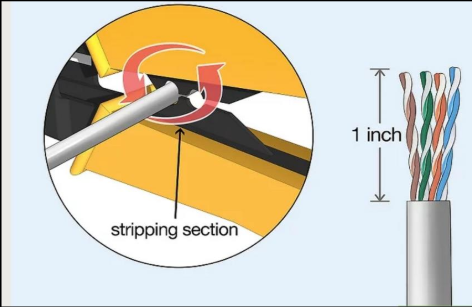
into the stripper section of the tool and squeeze it tight. Then, rotate the crimping tool around

the cable in a smooth and even motion to create a clean cut. Keep the tool clamped and pull

away towards the end of the wire to remove the sheathing.

➢ The stripping section is a round hole near the handle of the tool.

➢ The sheathing should come off cleanly, leaving the wires exposed



2. Untwist and straighten the wires inside of the cable. Inside of the cable you’ll see a

bunch of smaller wires twisted together. Separate the twisted wires and straighten them out so

they’re easier to sort into the right order.

➢ Cut off the small plastic wire separator or core so it’s out of the way.

➢ Don’t cut off or remove any of the wires or you won’t be able to crimp them into the

connector.

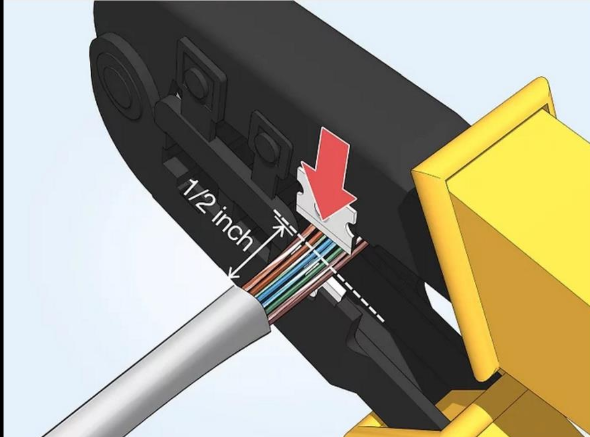
3. Arrange the wires into the right order. Use your fingers to put the wires in the correct

order so they can be properly crimped. The proper sequence is as follows from left to right:

Orange/White, Orange, Green/White, Blue, Blue/White, Green, Brown/White, Brown.

➢ There are 8 wires in total that need to be arranged in the right sequence.

➢ Note that the wires labeled Orange/White or Brown/White indicate the small wires that have

4. Cut the wires into an even line 1 ⁄2 inch (13 mm) from sheathing. Hold the wires with your thumb and index finger to keep them in order. Then, use the cutting section of the crimping tool to cut them into an even line.

➢ The cutting section of the tool will resemble wire cutters.

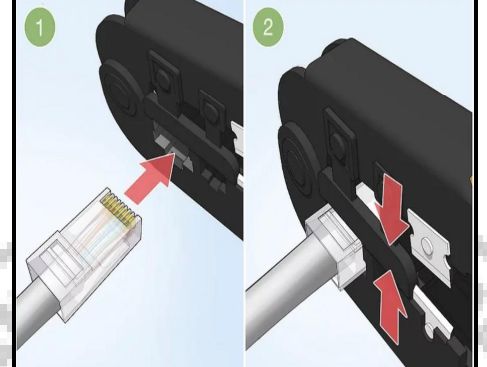
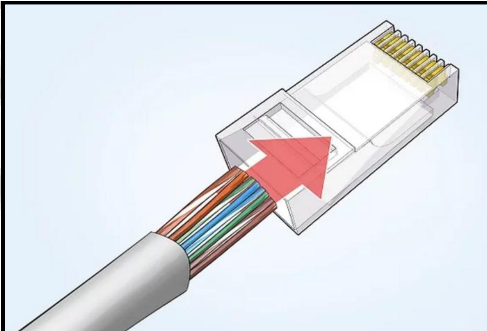
➢ The wires must be in an even line to be crimped into the RJ-45 connector properly. If you cut them in an uneven line, move further down the wires and cut them again.

5. Insert the wires into the RJ-45 connector. Hold the RJ-45 connector so the clip is on the underside and the small metal pins are facing up. Insert the cable into the connector so that each of the small wires fits into the small grooves in the connector.

➢ The sheathing of the cable should fit just inside of the connector so it’s past the base.

➢ If any of the small wires bend or don’t fit into a groove correctly, take the cable out and straighten the wires with your fingers before trying again.

➢ The wires must be inserted in the correct order and each wire must fit into a groove before you crimp the connector.

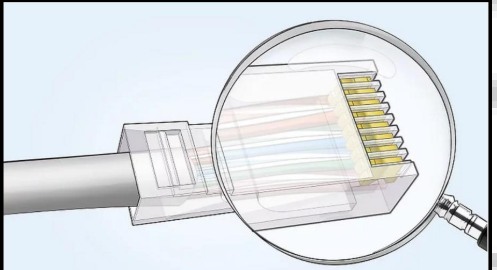


6. Stick the connector into the crimping part of the tool and squeeze twice. Insert the connector in the crimping section of the tool until it can’t fit any further. Squeeze the handles to crimp the connector and secure the wires. Release the handles, then squeeze the tool again to make sure all of the pins are pushed down.

➢ The crimping tool pushes small pins in the grooves down onto the wires to hold and connect them to the RJ-45 connector.

7. Remove the cable from the tool and check that all of the pins are down. Take the connector out of the tool and look at the pins to see that they’re all pushed down in an even line. Lightly tug at the connector to make sure it’s attached to the cable.

➢ If any of the pins aren’t pushed down, put the wire back into the crimping tool and crimp it again.



**Result-** Configuration of networking hardware like RJ-45 connector, CAT-6 cable, crimping tool, etc. is learned.

**Experiment No:2**

**Aim:-**Configuration of router,hub,switch etc(using real devices or simulator).

**Description:-**

1. Repeater:Functioning at Physical Layer.A repeater is an electronic device that receives a signal and retransmits it at a higher level and/or higher power, or onto the other side of an obstruction, so that the signal can cover longer distances. Repeater have two ports ,so cannot be use to connect for more than two devices

2. Hub: An Ethernet hub, active hub, network hub, repeater hub, hub or concentrator is a device for connecting multiple twisted pair or fiber optic Ethernet devices together and making them act as a single network segment. Hubs work at the physical layer (layer 1) of the OSI model. The device is a form of multiport repeater. Repeater hubs also participate in collision detection, forwarding a jam signal to all ports if it detects a collision.

3. Switch:A network switch or switching hub is a computer networking device that connects network segments.The term commonly refers to a network bridge that processes and routes data at the data link layer (layer 2) of the OSI model. Switches that additionally process data at the network layer (layer 3 and above) are often referred to as Layer 3 switches or multilayer switches.

4. Bridge: A network bridge connects multiple network segments at the data link layer (Layer 2) of the OSI model. In Ethernet networks, the term bridge formally means a device that behaves according to the IEEE 802.1D standard. A bridge and switch are very much alike; a switch being a bridge with numerous ports.

5. Router: A router is an electronic device that interconnects two or more computer networks, and selectively interchanges packets of data between them. Each data packet contains address information that a router can use to determine if the source and destination are on the same network, or if the data packet must be transferred from one network to another. Where multiple routers are used in a large collection of interconnected networks, the routers exchange information about target system addresses, so that each router can build up a table showing the preferred paths between any two systems on the interconnected networks.

6. Gate Way: In a communications network, a network node equipped for interfacing with another network that uses different protocols. A gateway may contain devices such as protocol translators, impedance matching devices, rate converters, fault isolators, or signal translators as necessary to provide system interoperability. It also requires the establishment of mutually acceptable administrative procedures between both networks. A protocol translation/mapping gateway interconnects networks with different network protocol technologies by performing the required protocol conversions.

**Result-** Configuration of router, hub, switch is performed .

**Experiment No:3**

**Aim:-** Running and using services/commands like ping, trace route, nslookup, arp, telnet, ftp,

etc.

**Syntax/Code:-**

PINGVersion: To get ping version installed on your system.

sudo ping -v

Using PING: ping www.google.com

To stop pinging we should use ctrl+c otherwise it will keep on sending packets.

Controlling the number of pings:Earlier we did not define the number of packets to send to the server/host by using -c option we can do so.

ping -c 5 [www.google.com](http://www.google.com)

Controlling the size of packets send: Ealier a default sized packets were sent to a host but we can send light and heavy packet by using -s option.

ping -s 4- -c 5 [www.google.com](http://www.google.com)

Trace route-

Click on start.

Click in the search box.

Then type cmd (you may need to type command in Windows 95/98/ME).

Once you have your Terminal box open, just type in the following but be sure to replace example.com with your domain name:

tracert example.com

It is also possible to run a traceroute using access domain or IP:

tracert rcbi-b1hp.accessdomain.com

tracert 64.13.192.208

You should see a response similar to the following:

Tracing route to example.com [64.13.192.208]

over a maximum of 30 hops:

1 <1 ms <1 ms <1 ms 72.10.62.1

2 <1 ms <1 ms <1 ms 10.101.248.1

3 1 ms <1 ms 1 ms 10.104.65.161

4 1 ms 5 ms 1 ms 10.104.0.1

5 2 ms 2 ms 3 ms 10.0.10.33

6 5 ms 3 ms 2 ms example.com [64.13.192.208]

Nslookup:-

$ nslookup redhat.com

Server: 192.168.19.2

Address: 192.168.19.2#53

Non-authoritative answer:

Name: redhat.com

Address: 209.132.183.181

Arp-

C:>arp -a 192.168.168.22

Interface: 192.168.168.21 --- 0x10004

Internet Address Physical Address Type

192.168.168.22 00-60-08-39-e5-a1 dynamic

C:>arp -a

Interface: 192.168.168.21 --- 0x10004

Internet Address Physical Address Type

192.168.168.9 00-02-e3-16-e4-5d dynamic

192.168.168.10 00-50-04-17-66-90 dynamic

192.168.168.22 00-60-08-39-e5-a1 dynamic

192.168.168.254 00-40-10-18-42-49 dynamic

telnet-

Select the Windows key and the R key.

In the Run box type CMD. Select OK. Type Telnet 13531.

Note: Do not include the <> when entering the IP Address.

If you see a blank cursor then the connection is fine. You can close the command prompt window.

If you get the message that 'telnet' is not recognized as an internal or external command, operable program or batch file. you will want to enable Telnet. See Additional Information on how to Enable telnet.

If you get an error or are unable to telnet to the server please contact your Network Administrator.

Example: If the server's ip address is 192.168.0.100, then:

Open command prompt on the workstation computer

Type: telnet 192.168.0.100 13531

ftp-

$ ftp IP/hostname

Or

$ ftp

ftp> open IP/hostname

**Result-**Services/commands like ping, trace route, nslookup, arp, telnet, ftp, etc. are run

**Experiment No:4**

**Aim:-** Network packet analysis using tools like Wireshark,tcpdump,cisco paper tracer,NetSim,OMNet++,NS2,NS3 etc.

**Description/Process/Commands:-**

WireShark:-  
1. Download and Install Wireshark:

Go to the Wireshark website to download the version that is compatible with your operating system.

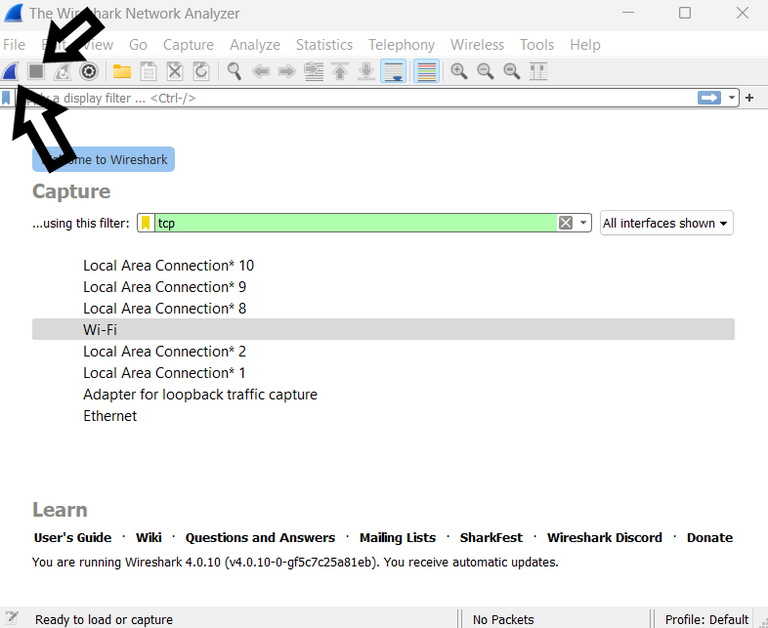
Adhere to the website’s installation instructions.

2. Begin Packet Capturing:

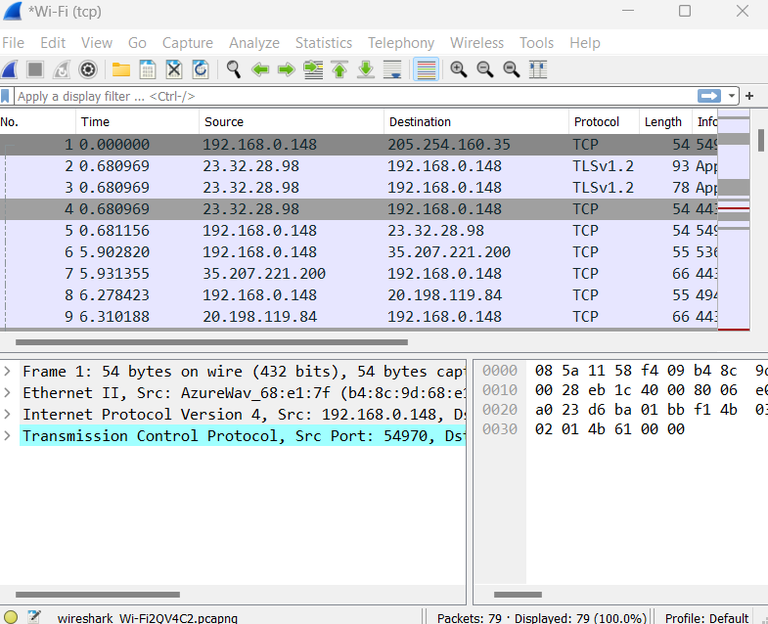
Open Wireshark and choose the network interface from which to begin collecting packets.

The display filter in Wireshark. The display filter in Wireshark’s default configuration is a bar that sits right above the column display. Here is where we enter expressions to narrow down what we can see in a pcap file, be it Ethernet frames, IP packets, or TCP segments.

There are several local interfaces available; please choose one.



Press the Start button.



In essence, you are recording and intercepting data packets as they pass through a network interface when you capture packets.

Tcpdump:-

1. Launch a Terminal or Command Prompt:

On Unix-based systems, open a terminal window. As an administrator, run the Command Prompt on Windows.

2. Begin Packet Capturing:

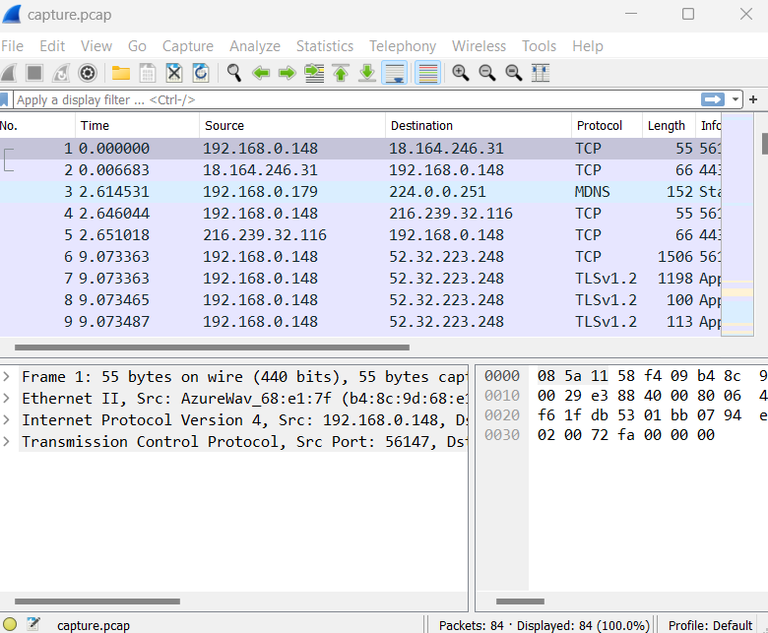
In the first case, run dumpcap -i <interface>-w<output\_file>, where <interface> is the network interface that you choose to start capturing from.

Example:

dumpcap -i Wi-Fi -w capture.pcap

Press Ctrl + C to stop capturing.

You can also launch capture.pcap from the same directory to view captured packets.



Result- Network packet analysis using tools like Wireshark, tcpdump , Cisco Packet Tracer, NetSim, OMNeT++, NS2, NS3, etc is done and understood.

**Experiment No:5**

**Aim:-**Socket programming using UDP and TCP (e.g., simple DNS, data & time client/server, echo client/server, iterative & concurrent servers) and raw sockets.

**Code:-**

// 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

int main() {

int sockfd;

char buffer[MAXLINE];

char \*hello = "Hello from client";

struct sockaddr\_in servaddr;

if ( (sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0 ) {

perror("socket creation failed"); exit(EXIT\_FAILURE); }

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

servaddr.sin\_family = AF\_INET;

servaddr.sin\_port = htons(PORT);

servaddr.sin\_addr.s\_addr = INADDR\_ANY;

int n, len;

endto(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; }

// 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

int main() {

int sockfd;

char buffer[MAXLINE];

char \*hello = "Hello from server";

struct sockaddr\_in servaddr, cliaddr;

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));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = INADDR\_ANY;

servaddr.sin\_port = htons(PORT);

if ( bind(sockfd, (const struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0 ) {

perror("bind failed"); exit(EXIT\_FAILURE); }

int len, n;

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, en);

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

return 0; }

TCP- TCP works with the Internet Protocol (IP), which defines how computers send packets of data to each other.

#include<netdb.h>

#include<stdlib.h>

#include<string.h>

#include<sys/socket.h>

#include<stdio.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;

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");

func(sockfd);

close(sockfd);

}

#include<stdio.h>

#include<stdlib.h>

#include<netdb.h>

#include<string.h>

#include<sys/types.h>

#include<sys/socket.h>

#include<netinet/in.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;

while ((buff[n++] = getchar()) != '\n') ;

write(sockfd, buff, sizeof(buff));

if (strncmp("exit", buff, 4) == 0) {

printf("Server Exit...\n"); break; } } }

int main() {

int sockfd, connfd, len;

struct sockaddr\_in servaddr, cli;

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));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

servaddr.sin\_port = htons(PORT);

if ((bind(sockfd, (SA\*)&servaddr, sizeof(servaddr))) != 0) {

printf("socket bind failed...\n"); exit(0);

} else printf("Socket successfully binded..\n");

if((listen(sockfd, 5)) != 0) {

printf("Listen failed...\n"); exit(0);

} else printf("Server listening..\n");

len = sizeof(cli);

connfd = accept(sockfd, (SA\*)&cli, &len);

if (connfd < 0) { printf("server acccept failed...\n"); exit(0);

} else printf("server acccept the client...\n");

func(connfd);

close the socket

close(sockfd);

}

**Result-** Socket programming using UDP and TCP is done.

**Experiment No:6**

**Aim:-**Programming using Raw Sockets.

**Code:-** Using raw sockets involves direct manipulation of the IP packets at the network layer, which requires special privileges and is typically used for low-level network tasks such as packet sniffing, network scanning, or crafting custom network protocols. Below is a basic example of how to create a raw socket and send a custom ICMP packet (ping) using C:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/ip.h>

#include <netinet/ip\_icmp.h>

#include <arpa/inet.h>

#define PACKET\_SIZE 64

unsigned short checksum(unsigned short \*ptr, int nbytes) {

unsigned long sum;

unsigned short oddbyte;

unsigned short answer;

sum = 0;

while (nbytes > 1) {

sum += \*ptr++;

nbytes -= 2;

}

if (nbytes == 1) {

oddbyte = 0;

\*((unsigned char \*)&oddbyte) = \*(unsigned char \*)ptr;

sum += oddbyte;

}

sum = (sum >> 16) + (sum & 0xFFFF);

sum += (sum >> 16);

answer = ~sum;

return answer;

}

int main(int argc, char \*argv[]) {

int sockfd;

char packet[PACKET\_SIZE];

struct sockaddr\_in dest;

struct iphdr \*ip\_header;

struct icmphdr \*icmp\_header;

int data\_size = PACKET\_SIZE - sizeof(struct iphdr) - sizeof(struct icmphdr);

if (argc != 2) {

fprintf(stderr, "Usage: %s <destination\_ip>\n", argv[0]);

exit(EXIT\_FAILURE);

}// Create raw socket

sockfd = socket(AF\_INET, SOCK\_RAW, IPPROTO\_ICMP);

if (sockfd < 0) {

perror("socket() failed");

exit(EXIT\_FAILURE);

}// Fill in the destination address structure

dest.sin\_family = AF\_INET;

dest.sin\_addr.s\_addr = inet\_addr(argv[1]);

memset(packet, 0, PACKET\_SIZE); // Prepare packet

ip\_header = (struct iphdr \*)packet;

icmp\_header = (struct icmphdr \*)(packet + sizeof(struct iphdr));

ip\_header->ihl = 5;

ip\_header->version = 4;

ip\_header->tos = 0;

ip\_header->tot\_len = htons(PACKET\_SIZE);

ip\_header->id = htons(0);

ip\_header->frag\_off = 0;

ip\_header->ttl = 255;

ip\_header->protocol = IPPROTO\_ICMP;

ip\_header->saddr = inet\_addr("0.0.0.0");

ip\_header->daddr = dest.sin\_addr.s\_addr;

ip\_header->check = 0;

icmp\_header->type = ICMP\_ECHO;

icmp\_header->code = 0;

icmp\_header->un.echo.id = 0;

icmp\_header->un.echo.sequence = 0;

icmp\_header->checksum = 0;

memset(packet + sizeof(struct iphdr) + sizeof(struct icmphdr), 'a', data\_size); // Fill in data

icmp\_header->checksum = checksum((unsigned short \*)icmp\_header, sizeof(struct icmphdr) + data\_size); // Calculate checksum

if (sendto(sockfd, packet, PACKET\_SIZE, 0, (struct sockaddr \*)&dest, sizeof(dest)) < 0) {

perror("sendto() failed");

exit(EXIT\_FAILURE);

}

printf("Packet sent successfully.\n");

close(sockfd);

return 0;

}

This code creates a raw socket using socket(AF\_INET, SOCK\_RAW, IPPROTO\_ICMP) and constructs an ICMP echo request packet. It then sends this packet to the specified destination IP address.

**Result:-** Hence we do programming using raw sockets.

**Experiment No:7**

**Aim:-** Programming using remote procedure call.

**Code:-**

Program 1 for RPC (Interface):

import java.rmi.\*;

public interface greater extends Remote {

public String getresult(int first,int second) throws RemoteException;

}

RPC Program for client:

import java.io.\*;

import java.rmi.\*;

public class great{

public static void main(String args[]) throws IOException{

String result;

int n=0,n1=0;

try{

greater f=(greater)Naming.lookup("greaterser");

try{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

System.out.println("\n\n\t Rpc program for clientside");

System.out.println("\n\t Enter the two numbers");

System.out.println("\n\t the First number is :");

n=Integer.parseInt(br.readLine());

System.out.println("\n\t the Second number is:");

n1=Integer.parseInt(br.readLine());

}

catch(IOException e){}

result=f.getresult(n,n1);

if(result.equals("Equal"))

System.out.println("\n\n\t Both nos are equal");

else

System.out.println("\n\n\t The greater no is"+result);

}

catch(Exception e)

{

System.out.println("Exception from client side:"+e);

} } }

RPC Program for server:

import java.io.\*;

import java.rmi.\*;

import java.rmi.server.\*;

import java.util.\*;

public class greaterser extends UnicastRemoteObject implements greater{

public greaterser()throws RemoteException{}

public String getresult(int first,int second)throws RemoteException{

if(first<second)

return(Integer.toString(second));

else{

if(second<first)

return(Integer.toString(first));

else

return("EQUAL");

} }

public static void main(String args[]){

try{

greaterser f=new greaterser();

Naming.rebind("greaterser",f);

System.out.println("\n\n\t Server is ready");

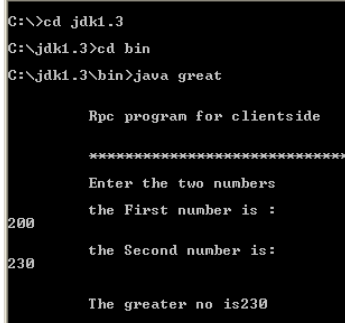
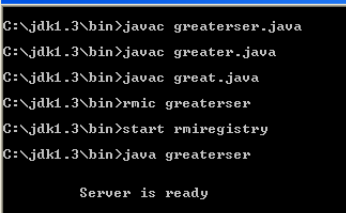
}catch(Exception e){

System.out.println("error from ss");

}}

}

**Result:-**

****

**Experiment No:8**

**Aim:-**Write a code simulating ping and traceroute commands.

**Code:-**PING:

Variation 1)

int main(){

const char \*hostName = [@"stackoverflow.com"

cStringUsingEncoding:NSASCIIStringEncoding];

SCNetworkConnectionFlags flags = 0;

if (SCNetworkCheckReachabilityByName(hostName, &flags) && flags > 0) {

NSLog(@"Host is reachable: %d", flags);

}else { NSLog(@"Host is unreachable"); }}

Variation 2)

bool success = false;

const char \*host\_name = [@"stackoverflow.com"

cStringUsingEncoding:NSASCIIStringEncoding];

SCNetworkReachabilityRef reachability =

SCNetworkReachabilityCreateWithName(NULL,

host\_name);

SCNetworkReachabilityFlags flags;

success = SCNetworkReachabilityGetFlags(reachability, &flags);

bool isAvailable = success && (flags & kSCNetworkFlagsReachable) &&

!(flags & kSCNetworkFlagsConnectionRequired);

if (isAvailable) {

NSLog(@"Host is reachable: %d", flags);

}else{

NSLog(@"Host is unreachable");

}

**Experiment No:9**

**Aim:-**Create a sockets for HTTP for web page upload and download.

**Code:-**

#include<stdio.h>

#include<stdlib.h>

#include<netdb.h>

#include<netinet/in.h>

#include<arpa/inet.h>

#include<sys/types.h>

#include<sys/socket.h>

#define size 100

int main(int argc, char \*argv[]){

struct sockaddr\_in sock;

struct hostent \*hp,port;

char \*req,\*hostname,\*cp;

FILE \*local;

char buff[size];

int con,i,l,nrv,sd;

if(argc!=3){

printf(“\nUsage: %s<server>filename”,argv[0]);

exit(1);}

if(cp=strchr(argv[1],’/’)){

\*cp=’\0’;

l=strlen(argv[1]);

hostname=malloc(l+1);

strcpy(hostname,argv[1]);

\*cp=’/’;

l=strlen(cp);

req=malloc(l+1);

strcpy(req,cp);

}else{

hostname=argv[1];

req=”/”;

}

printf(“\nHost=%s\nReq= %s”,hostname,req);

sd=socket(AF\_INET,SOCK\_STREAM,0);

if(sd<0){

perror(“\nCannot open socket”);

exit(1);

}

bzero(&sock,sizeof(sock));

sock.sin\_family=AF\_INET;

con=inet\_pton(AF\_INET, argv[1], &sock);

sock.sin\_port=htons(80);

con=connect(sd,(struct sockaddr \*)&sock, sizeof(sock));

if(con<0){

perror(“\nConnection failed”);

exit(1);

}

sprintf(buff,”Get HTTP:%s//1.1\r\nHost: %s\r\nConnection: class\r\n\r”, req, hostname);

printf(“Buff=%s\n”, buff);

l=strlen(buff);

local=fopen(argv[2],”w”);

write(sd,buff,1);

do{

nrv=read(sd,buff,size);

if(nrv>0){

for(i=0;i<nrv;i++)

putc(buff[i],local);

}

else break;

}while(1);

close(sd);

fclose(local);

return 0;

}

**Experiment No:10**

**Aim:-** Implementation of Stop and Wait Protocol and Sliding Window Protocol.

**Code:-**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

int k,time,win=2,i2=0,frame=0,a[20],b[20],i,j,s,r,ack,c,d;

int send(int,int);

int receive();

int checsum(int \*);

main() {

int i1=0,j1=0,c1;

printf("Enter the frame size\n");

scanf("%d",&frame);

printf("Enter the window size\n");

scanf("%d",&win);

j1=win;

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

a[i]=rand();

}

k=1;

while(i1<frame) {

if((frame-i1)<win)

j1=frame-i1;

printf("\n\ntransmit the window no %d\n\n",k);

c1=send(i1,i1+j1);

ack=receive(i1,i1+j1,c1);

if (ack!=0) {

printf("\n\n1.Selective window\n");

printf("2.Go back N\n");

scanf("%d",&ack);

switch(ack){

case 1:

printf("\n\n\t Selective window \t\nEnter the faulty frame no\n");

scanf("%d",&i2);

printf("\n\n Retransmit the frame %d \n",i2);

send(i2,i2+1);

break;

case 2:

printf("\n\n\t Go back n\t\n\n");

printf("\nRetransmit the frames from %d to %d\n",i1,i1+j1);

send(i1,i1+j1);

break;

}

}

i1=i1+win;

k++;

}

}

int send(c,d) {

int t1;

for(i=c;i<d;i++) {

b[i]=a[i];

printf("frame %d is sent\n",i);

}

s=checsum(&a[c]);

return(s); }

int receive(c,d,c2)

int c2;

{

r=checsum(&b[c]);

if(c2==r) {

return(0);

}

else

return(1);

}

int checsum(int \*c){

int sum=0;

for(i=0;i<win;i++)

sum=sum^(\*c);

return sum;

}

**Output:**

Enter the frame size

50

Enter the window size

5

transmit the window no 1

frame 0 is sent

frame 1 is sent

frame 2 is sent

frame 3 is sent

frame 4 is sent

1.selective window

2.Go back N

1

selective window

enter the faculty frame no

15

retransmit the frames from 15

frame 15 is sent

transmit the window no 2

frame 5 is sent

frame 6 is sent

frame 7 is sent

frame 8 is sent

frame 9 is sent

1.selective window

2.Go back N