1. Create straight cable, cross cable and console cable using crimping tools. Describe the procedure with proper diagram.

Answer: The color codes for the straight and cross cables are as depicted in the table below:

Pin ID	Side A	Side B
1	Orange-White	Orange-White
2	Orange	Orange
3	Green-White	Green-White
4	Blue	Blue
5	Blue-White	Blue-White
6	Green	Green
7	Brown-White	Brown-White
8	Brown	Brown

Straight Cable Configuration

Pin ID	Side A	Side B
1	Orange-White	Green -White
2	Orange	Green
3	Green-White	Orange -White
4	Blue	Brown-White
5	Blue-White	Brown
6	Green	Orange
7	Brown-White	Blue
8	Brown	Blue-White

Cross Cable Configuration

The following steps need to be followed carefully while configuring the cables:

i. Take the crimping tool and using the cutting edge of the tool remove the upper layer of insulation generally grey in colour. The cut should be made carefully keeping in mind not to damage the cable internally.

ii. Now, take out the coloured parts of the wire from the groups and using that same cutting edge remove the plastic insulation, also remove the fibrous part that may be present inside.

iii. Straighten the cables that you just removed from the groove, and make sure that they are arranged as per the colour code given the table above.

iv. This process needs to be done at both the ends of the cable. for a straight cable that colour coated both the ends will be same, and for a cross cable the colour code at the two ends would be different as shown in the table.

v. Take the RJ45 connector, and insert the cable into the pins of the RJ45 such that the ends of the cable touch the pinheads in the RJ45. The same thing needs to be done at both ends.

vi. Once the arrangement is done, take the crimping tool and fix the cable permanently along with the RJ45 connector.

2. Discuss the following network commands:

a. ping [including special purpose commands, like ping -b, ping -i, ping -c]
b. ifconfig [including special purpose commands]
c. traceroute
d. finger

Answer: ping: The ping command is usually used as a simple way to verify that a computer can communicate over the network with another computer or network device. The ping command operates by sending *Internet Control Message Protocol (ICMP) Echo Request* messages to the destination computer and waiting for a response.

i. ping –b: It allows to ping to a broadcast address
ii. ping –i: This option sets the Time to Live (TTL) value, the maximum of which is 255.
iii. ping –c: It can set the number of ECHO_REQUEST packets to be sent

ifconfig: It is a short term for Interface Configuration. It is used for displaying current network configuration information, setting up an ip address, netmask or broadcast address to a network interface.

- **i. ifconfig**: It shows all the active interfaces details. The ifconfig command also used to check the assigned IP address of an server.
- **ii. ifconfig -a**: It will display information of all active or inactive network interfaces on server.
- **iii. ifconfig eth0 up**: It activates the network interface eth0.
- **iv. ifconfig eth0 down**: It deactivates the network interface eth0.
- **v. ifconfig eth0 hw ether AA:BB:CC:DD:EE:FF:** It is used to manually change the MAC Address of the eth0 network interface.
- **vi. ifconfig arp:** This is an option specific to broadcast networks such as Ethernets or packet radio. It enables the use of the Address Resolution Protocol (ARP) to detect the physical addresses of hosts attached to the network. For broadcast networks, it is on by default. If ARP is disabled, ifconfig displays the NOARP flag.
- **vii. ifconfig netmask *mask***: This option assigns a subnet mask to be used by the interface.

traceroute: The traceroute command traces the network path of Internet routers that packets take as they are forwarded from your computer to a destination address. The "length" of the network connection is indicated by the number of Internet routers in the traceroute path. Traceroutes can be useful to diagnose slow network connections. For example, if you can usually reach an Internet site but it is slow today, then a traceroute to that site should show you one or more hops with either long times or marked with "*" indicating the time was really long.

finger: The /usr/bin/finger command displays information about the users currently logged in to a host. The format of the output varies with the options for the information presented. The default format includes the ***Login name***, ***Full user name***, ***Terminal name***, ***Write status*** (an * (asterisk) before the terminal name indicates that write permission is denied)

3. Write down the steps to configure an IP address of a Linux machine.

Answer: An IP address, short for Internet Protocol address, is an identifying number for a piece of network hardware. Having an IP address allows a device to communicate with other devices over an IP-based network like the internet.
An IP address provides an identity to a networked device. Similar to a home or business address supplying that specific physical location with an identifiable address, devices on a network are differentiated from one another through IP addresses.

There are private IP addresses, public IP addresses, static IP addresses, and dynamic IP addresses. private IP addresses are used "inside" a network, like the one you probably run at home. These types of IP addresses are used to provide a way for your devices to communicate with your router and all the other devices in your private network. Private IP addresses can be set manually or assigned automatically by your router.

Public IP addresses are used on the "outside" of your network and are assigned by your ISP. It's the main address that your home or business network uses to communicate with the rest of the networked devices around the world (i.e. the internet).

The following steps need to be performed to configure the ip address:

i. Open terminal (in case of linux OS)
ii. Enter the command ifconfig eth0 192.168.31.1 netmask 255.255.255.0
iii. This would set the address of the eth0 network interface as 192.168.31.1

4. Configure Telnet server.

Answer: Telnet is a protocol that allows you to connect to remote computers (called hosts) over a TCP/IP network (such as the Internet). Using telnet client software on your computer, you can make a connection to a telnet server (i.e., the remote host). Once your telnet client establishes a connection to the remote host, your client becomes a virtual terminal, allowing you to communicate with the remote host from your computer.

Telnet is most likely to be used by program developers and anyone who has a need to use specific applications or data located at a particular host computer. Telnet is not a secure communication protocol because it does not use any security mechanism and transfers the data over network/internet in a plain-text form including the passwords and so any one can sniff the packets to get that important information.

The following are the steps to configure a Telnet server:

Step 1: go to /etc/xinetd.d (cd /etc/xinetd.d)

Step 2: open the xinetd.d file

Step 3: change *disable = yes* to *disable = no*

Step 4: Now restart the xinetd service using the command *service xinetd restart*

Step 5: Now type telnet *IPAddressServer* from client

5. Configure FTP server. What is anonymous FTP?

Answer: The File Transfer Protocol (FTP) is used as one of the most common means of copying files between servers over the Internet. Most web based download sites use the built in FTP capabilities of web browsers and therefore most server oriented operating systems usually include an FTP server application as part of the software suite. FTP relies on a pair of TCP ports to get the job done. It operates in two connection channels:
FTP Control Channel, TCP Port 21: All commands you send and the ftp server's responses to those commands will go over the control connection, but any data sent back (such as "ls" directory lists or actual file data in either direction) will go over the data connection.

FTP Data Channel, TCP Port 20: This port is used for all subsequent data transfers between the client and server.

The following are the steps to configure a FTP server in RedHat Linux 9:

Step 1: go to /etc/vsftpd/vsftpd.conf (cd /etc/vsftpd/vsftpd.conf)

Step 2: open the vsftpd.conf file

Step 3: Disable anonymous FTP by changing anonymous_enable=NO to anonymous_enable=YES

Step 4: Enable individual logins by uncommenting the following line local_enable=YES

Step 5: Start the vsftp server using the command *service vsftpd start*

6. Configure DHCP server.

Answer: DHCP (Dynamic Host Configuration Protocol) is a network management protocol used to dynamically assign an Internet Protocol (IP) address to any device, or node, on a network so they can communicate using IP. DHCP automates and centrally manages these configurations rather than requiring network administrators to manually assign IP addresses to all network devices.

DHCP runs at the application layer of the Transmission Control Protocol/IP (TCP/IP) protocol stack to dynamically assign IP addresses to DHCP clients and to allocate TCP/IP configuration information to DHCP clients. This includes subnet mask information, default gateway IP addresses and domain name system (DNS) addresses.

DHCP is a client-server protocol in which servers manage a pool of unique IP addresses, as well as information about client configuration parameters, and assign addresses out of those address pools. DHCP-enabled clients send a request to the DHCP server whenever they connect to a network.

Clients configured with DHCP broadcast a request to the DHCP server and request network configuration information for the local network to which they're attached. A client typically broadcasts a query for this information immediately after booting up. The DHCP server responds to the client request by providing IP configuration information previously specified by a network administrator. This includes a specific IP address as well as for the time period, also called a lease, for which the allocation is valid. When refreshing an assignment, a DHCP client requests the same parameters, but the DHCP server may assign a new IP address based on policies set by administrators.

The following are the steps to configure a DHCP server:

Step 1: Assign a static IP address to the server

Step 2: Open the dhcpd.conf file at etc/dhcp/dhcpd.conf.

Step 3: Declare a subnet as per the network configuration made in step 1.

Step 4: Now start the dhcp server by using the command *service dhcpd start*

7. Configure DNS.

Answer: The Domain Name System (DNS) is the phonebook of the Internet. Humans access information online through domain names, like nytimes.com or espn.com. Web browsers interact through Internet Protocol (IP) addresses. DNS translates domain names to IP addresses so browsers can load Internet resources.

Each device connected to the Internet has a unique IP address which other machines use to find the device. DNS servers eliminate the need for humans to memorize IP addresses such as 192.168.1.1 (in IPv4), or more complex newer alphanumeric IP addresses such as 2400:cb00:2048:1::c629:d7a2 (in IPv6).

The process of DNS resolution involves converting a hostname (such as www.example.com) into a computer-friendly IP address (such as 192.168.1.1). An IP address is given to each device on the Internet, and that address is necessary to find the appropriate Internet device - like a street address is used to find a particular home. When a user wants to load a webpage, a translation must occur between what a user types into their web browser (example.com) and the machine-friendly address necessary to locate the example.com webpage.

There are 4 DNS servers involved in loading a webpage:

DNS recursor - The recursor can be thought of as a librarian who is asked to go find a particular book somewhere in a library. The DNS recursor is a server designed to receive queries from client machines through applications such as web browsers. Typically, the recursor is then responsible for making additional requests in order to satisfy the client's DNS query.

Root nameserver - The root server is the first step in translating (resolving) human readable host names into IP addresses. It can be thought of like an index in a library that points to different racks of books - typically it serves as a reference to other more specific locations.

TLD nameserver - The top level domain server (TLD) can be thought of as a specific rack of books in a library. This nameserver is the next step in the search for a specific IP address, and it hosts the last portion of a hostname (In example.com, the TLD server is "com").

Authoritative nameserver - This final nameserver can be thought of as a dictionary on a rack of books, in which a specific name can be translated into its definition. The authoritative name server is the last stop in the nameserver query. If the authoritative name server has access to the requested record, it will return the IP address for the requested hostname back to the DNS Recursor (the librarian) that made the initial request.

The entire hostname with its domain such as server.example.com is called a fully qualified domain name (FQDN). The right-most part of the FQDN such as .com or .net is called the top level domain, with the remaining parts of the FQDN, which are separated by periods, being sub-domains. These sub-domains are used to divide FQDNs into zones, with the DNS information for each zone being maintained by at least one authoritative name server.

The steps to configure a DNS server are:

Step 1: Go to the location /etc using the command *cd /etc*

Step 2: Open the file *named.conf*
Step 3: Enter the following in the file after *zone localhost* block ends.

```
zone "yourDNS.com" IN {
        type master;
        file "yourDNS.com.zone";
        allow-update { none; };
```

};

Step 4: Now go to var/named by using the command *cd var/named* and create a file *yourDNS.com.zone*

Step 5: Open that file and enter the following:

```
$TTL 86400
$ORIGIN yourDNS.com.
@       IN      SOA     ns1. yourDNS.com.
admin. yourDNS.com. (
                                2004042601      ; serial
                                21600           ; refresh
                                3600            ; retry
                                604800          ; expires
                                86400 )         ; minimum
        IN      NS              ns1. yourDNS.com.
        IN      MX      10      mail. yourDNS.com.
        IN      A               192.168.1.200
ns1     IN      A               192.168.1.200
www     IN      A               192.168.1.200
ftp     IN      A               192.168.1.200
mail    IN      A               192.168.1.200
```

Step 6: Now reload the zone using the command */etc/init.d/named reload*

Step 7: Now we test the domain using nslookup as below.

```
nslookup
server 127.0.0.1
yourDNS.com
www.yourDNS.com
mail.yourDNS.com
```

8. Implement a program that can retrieve the name of a host.

```
import java.net.*;

class ShowHostName{
      public static void main(String[] args) {
            try {
             InetAddress host = InetAddress.getLocalHost();
             String hostName = host.getHostName();
             System.out.println("Current IP address : " + host);
             System.out.println("Current Hostname : " + hostName);

        }catch (Exception e) {

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

9. Write socket program to show the time and date of a server machine.

Server side:

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

class DisplayDateServer{
	public static void main(String[] args) {
		try{
			ServerSocket ss = new ServerSocket(3456);
			Socket s = ss.accept();
			Date d = new Date();
			PrintStream out = new PrintStream(s.getOutputStream());

			out.println(d);
			s.close();
			ss.close();
		}catch(Exception e){
			e.printStackTrace();
		}
	}
}
```

Client side:

```
import java.net.*;
import java.io.*;

class DisplayDateClient{
	public static void main(String[] args) {
		try{
			Socket s = new Socket("localhost", 3456);
			BufferedReader br = new BufferedReader(new
			InputStreamReader(s.getInputStream()));

			PrintStream out = new PrintStream(s.getOutputStream());

			System.out.println("Waiting for server...");

			String str = br.readLine();
			System.out.println("Server Time: " + str);

			s.close();
		}catch(Exception e){
			e.printStackTrace();
		}
	}
}
```

10. Write a socket program to send a message from client machine to server machine and reply back the same message from server to client.

Server Side:

```
import java.net.*;
import java.io.*;

class ReplyMessageServer{
	public static void main(String[] args) {
		try{
			ServerSocket ss = new ServerSocket(1111);
			Socket s = ss.accept();
			BufferedReader br = new BufferedReader(new
			InputStreamReader(s.getInputStream()));
			PrintStream out = new PrintStream(s.getOutputStream());

			System.out.println("Waiting for server...");
			String str = br.readLine();

			System.out.println("Message recieved. Sending to client");
			out.println(str);

			s.close();
			ss.close();
		}catch(Exception e){
			e.printStackTrace();
		}
	}
}
```

Client Side:

```
import java.net.*;
import java.io.*;

class ReplyMessageClient{
	public static void main(String[] args) {
		try{
			Socket s = new Socket("localhost", 1111);
			BufferedReader br = new BufferedReader(new
			InputStreamReader(s.getInputStream()));
			BufferedReader in = new BufferedReader(new
			InputStreamReader(System.in));
			PrintStream out = new PrintStream(s.getOutputStream());

			System.out.print("Enter your message: ");
			String str = in.readLine();
			out.println(str);

			System.out.println("Waiting for server...");
			str = br.readLine();
			System.out.println("Message recieved from server: " + str);

			s.close();
		}catch(Exception e){
			e.printStackTrace();
		}
	}
}
```

11. Write a socket program to send a message in lower case from client machine to server machine and the server will convert the message into uppercase.

Server Side

```
import java.net.*;
import java.io.*;

class lowerToUpperServer{
	public static void main(String[] args) {
		try{
			ServerSocket ss = new ServerSocket(1111);
			Socket s = ss.accept();
			BufferedReader br = new BufferedReader(new
			InputStreamReader(s.getInputStream()));
			PrintStream out = new PrintStream(s.getOutputStream());

			System.out.println("Waiting for clients message...");
			String str = br.readLine();

			System.out.println("Message Recieved. Converting case...");

			out.println(str.toUpperCase());
			System.out.println("Done.");

			s.close();
			ss.close();

		}catch(Exception e){
			e.printStackTrace();
		}
	}
}
```

Client Side

```
import java.net.*;
import java.io.*;

class lowerToUpperClient{
	public static void main(String[] args) {
		try{
			Socket s = new Socket("localhost", 1111);
			BufferedReader br = new BufferedReader(new
			InputStreamReader(s.getInputStream()));

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

			PrintStream out = new PrintStream(s.getOutputStream());

			System.out.print("Enter the message: ");

			String str = in.readLine();
			out.println(str);

			System.out.println("Waiting for the server...");
			str = br.readLine();
			System.out.println("Sever responded. Message Received: " + str);

			s.close();
		}catch(Exception e){
			e.printStackTrace();
		}
	}
}
```

12. Write a socket program which will take numbers as input from client machine and send it to server machine where calculation would be done and send the result to the client machine (Iterative Server).

Server Side:

```
import java.io.*;
import java.net.*;

class calculationServer{
        public static void main(String[] args) {
                try{
                        ServerSocket ss = new ServerSocket(1111);
                        Socket s = ss.accept();
                        BufferedReader br = new BufferedReader(new
                        InputStreamReader(s.getInputStream()));

                        PrintStream out = new PrintStream(s.getOutputStream());

                        System.out.println("Waiting for client...");
                        String str = br.readLine();

                        System.out.println("Message Recieved. Calculating.");

                        String[] strArr = str.split(" ");

                        int sum = 0;
                        for(int i = 0; i<strArr.length;i++){
                                sum = sum + Integer.parseInt(strArr[i]);
                        }

                        System.out.println("Sending summation to client...");
                        out.println(sum);

                        s.close();
                        ss.close();
                }catch(Exception e){
                        e.printStackTrace();
                }
        }
}
```

Client Side:

```
import java.io.*;
import java.net.*;

class calculationClient{
        public static void main(String[] args) {
                try{
                        Socket s = new Socket("localhost", 1111);
                        BufferedReader br = new BufferedReader(new
                        InputStreamReader(s.getInputStream()));

                        BufferedReader in = new BufferedReader(new
                        InputStreamReader(System.in));
                        PrintStream out = new PrintStream(s.getOutputStream());

                        System.out.print("Enter the number seperated by space: ");
                        String str = in.readLine();

                        System.out.println("Sending to server.");
                        out.println(str);

                        str = br.readLine();
```

```
                        System.out.println("Server responded. Summation: " + str);

                        s.close();
                }catch(Exception e){
                        e.printStackTrace();
                }
        }
}
```

13. Write a socket program that will receive multicast message from client computer and display the output.

Server Side:

```
import java.io.*;
import java.net.*;
public class Server {
        public static void main(String args[]) {
                try {
                        InetAddress group = InetAddress.getByName("224.0.0.1");
                        MulticastSocket s = new MulticastSocket(1234);
                        s.joinGroup(group);
                        System.out.println("Group Joined. Waiting for message...");
                        byte[] buffer = new byte[100];

                        DatagramPacket packet = new DatagramPacket(buffer,
                        buffer.length);
                        s.receive(packet);
                        System.out.println(new String(buffer));
                        s.close();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

Client Side:

```
import java.io.*;
import java.net.*;

class Client{
        public static void main(String[] args) {
                try{
                        InetAddress group = InetAddress.getByName("224.0.0.1");
                        MulticastSocket s = new MulticastSocket(1234);

                        String msg = "Hello, this is a multicast message.";

                        DatagramPacket packet = new DatagramPacket(msg.getBytes(),
                        msg.length(), group, 1234);

                        s.send(packet);
                        s.close();
                }catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

14. Write a socket program to implement data link layer Error Detection Mechanism (Cyclic Redundancy Check)

Sender:

```
import java.io.*;
import java.net.*;

public class CRCSender{
        public static void main(String args[]){
                try{
                        String msg, divisor, toSend = "";
                        Socket cs = new Socket("localhost", 1111);
                        PrintStream out = new PrintStream(cs.getOutputStream());

                        BufferedReader in = new BufferedReader(new
                        InputStreamReader(System.in));
                        System.out.print("Enter the message: ");
                        msg = in.readLine();
                        System.out.print("Enter the divisor: ");
                        divisor = in.readLine();

                        String[] s = msg.split("");
                        String[] s2 = divisor.split("");

                        int[] data = new int[msg.length()+divisor.length()-1];
                        int[] dataDivisor = new int[divisor.length()];

                        for(int i = 0; i<msg.length();i++)
                                data[i] = Integer.parseInt(s[i]);

                        for(int i = 0; i<divisor.length();i++)
                                dataDivisor[i] = Integer.parseInt(s2[i]);

                        System.out.println();

                        for(int i= 0;i<msg.length();i++){
                                if(data[i] == 1)
                                        for(int j=0;j<divisor.length();j++)
                                                data[i+j]^=dataDivisor[j];
                        }

                        System.out.println();

                        System.out.print("CRC: ");

                        for(int i = 0; i<msg.length()+divisor.length()-1;i++){
                                System.out.print(data[i]);
                        }

                        for(int i=0;i<msg.length();i++){
                                toSend = toSend + s[i];
                        }

                        for(int i=0;i<divisor.length()-1;i++){
                                toSend = toSend + data[msg.length()+i];
                        }

                        System.out.println();
                        System.out.println("Message with CRC: " + toSend);
                        out.println(toSend);
                        cs.close();

                }catch(Exception e){
                        e.printStackTrace();
                }
```

```
        }
}
```

Receiver:

```
import java.io.*;
import java.net.*;

public class CRCReceiver{
        public static void main(String args[]){
                try{
                        ServerSocket serverSocket = new ServerSocket(1111);
                        Socket clientSocket = serverSocket.accept();
                        BufferedReader in = new BufferedReader(new
                        InputStreamReader(System.in));

                        BufferedReader getClientMessage = new BufferedReader(new
                        InputStreamReader(clientSocket.getInputStream()));

                        PrintStream sendToClient = new
                        PrintStream(clientSocket.getOutputStream());

                        String msg = getClientMessage.readLine();
                        System.out.println("Sender send: " + msg);

                        System.out.print("Enter the divisor: ");
                        String divisor = in.readLine();
                        String[] s2 = divisor.split("");

                        int[] dataDivisor = new int[divisor.length()];

                        for(int i = 0; i<divisor.length();i++)
                                dataDivisor[i] = Integer.parseInt(s2[i]);

                        String[] s = msg.split("");

                        int[] data = new int[msg.length()];

                        for(int i = 0; i<msg.length();i++)
                                data[i] = Integer.parseInt(s[i]);

                        for(int i= 0;i<msg.length();i++){
                                if(data[i] == 1)
                                        for(int j=0;j<divisor.length();j++)
                                                data[i+j]^=dataDivisor[j];
                        }


                        for(int i = 0; i<msg.length();i++){
                                if(data[i] == 1){
                                        System.out.print("Error in data.");
                                        System.exit(0);
                                }
                        }

                        System.out.println("No error in message. Received
                        successfully.");

                        clientSocket.close();
                        serverSocket.close();
                }catch(Exception e){
                        e.printStackTrace();
                }
        }
}
```