Socket Programming Interview Questions And Answers Guide.

Global Guideline.

https://globalguideline.com/

Socket Programming Job Interview Preparation Guide.

Question # 1

What Is Socket?

Answer:-

A socket is one end-point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes--Socket and ServerSocket--that implement the client side of the connection and the server side of the connection, respectively.

Read More Answers.

Question # 2

How does the race condition occur?

Answer:-

It occurs when two or more processes are reading or writing some shared data and the final result depends on who runs precisely when.

Read More Answers.

Question # 3

What is multiprogramming?

Answer:-

Multiprogramming is a rapid switching of the CPU back and forth between processes.

Read More Answers.

Question # 4

Name the seven layers of the OSI Model and describe them briefly?

Answer:-

Physical Layer - covers the physical interface between devices and the rules by which bits are passed from one to another.
Data Link Layer - attempts o make the physical link reliable and provides the means to activate, maintain, and deactivate the link.
Network Layer - provides for the transfer of information between end systems across
some sort communications network.
Transport Layer - provides a mechanism for the exchange of data between end system.
Session Layer - provides the mechanism for controlling the dialogue between applications in end systems.
Presentation Layer - defines the format of the data to be exchanged between applications and offers application programs a set of data transformation services.
Application Layer - provides a means for application programs to access the OSI environment.

Read More Answers.

Question # 5

What is the difference between TCP and UDP?

Answer:-

TCP and UDP are both transport-level protocols. TCP is designed to provide reliable
communication across a variety of reliable and unreliable networks and internets.
UDP provides a connectionless service for application-level procedures. Thus, UDP is basically an unreliable service; delivery and duplicate protection are not guareented.

Read More Answers.

Question # 6

What does a socket consists of?

Answer:-

The combination of an IP address and a port number is called a socket.

Read More Answers.

Question # 7

What is a JavaBean?

Answer:-

JavaBeans are reusable software components written in the Java programming language, designed to be manipulated visually by a software develpoment environment, like JBuilder or VisualAge for Java. They are similar to Microsoft's ActiveX components, but designed to be platform-neutral, running anywhere there is a Java Virtual Machine (JVM).

Read More Answers.

Question # 8

What are the seven layers(OSI model) of networking?

Answer:-

1.Physical,
2.Data Link,
3.Network,
4.Transport,
5.Session,
6.Presentation and
7.Application Layers.

Read More Answers.

Question # 9

What are some advantages and disadvantages of Java Sockets?

Answer:-

Advantages of Java Sockets:
Sockets are flexible and sufficient. Efficient socket based programming can be easily implemented for general communications.
Sockets cause low network traffic. Unlike HTML forms and CGI scripts that generate and transfer whole web pages for each new request, Java applets can send only necessary updated information.
Disadvantages of Java Sockets:
Security restrictions are sometimes overbearing because a Java applet running in a Web browser is only able to establish connections to the machine where it came from, and to nowhere else on the network
Despite all of the useful and helpful Java features, Socket based communications allows only to send packets of raw data between applications. Both the client-side and server-side have to provide mechanisms to make the data useful in any way.
Since the data formats and protocols remain application specific, the re-use of socket based implementations is limited.

Read More Answers.

Question # 10

What is the difference between a NULL pointer and a void pointer?

Answer:-

A NULL pointer is a pointer of any type whose value is zero. A void pointer is a pointer to an object of an unknown type, and is guaranteed to have enough bits to hold a pointer to any object. A void pointer is not guaranteed to have enough bits to point to a function (though in general practice it does).

Read More Answers.

Question # 11

What is encapsulation technique?

Answer:-

Hiding data within the class and making it available only through the methods. This technique is used to protect your class against accidental changes to fields, which might leave the class in an inconsistent state.

Read More Answers.

Question # 12

Socket Programming Interview Questions

Answer:-

1. User(s) are complaining of delays when using the network. What would you do?
2. What are some of the problems associated with operating a switched LAN?
3. Name some of the ways of combining TCP/IP traffic and SNA traffic over the same link.
4. What sort of cabling is suitable for Fast Ethernet protocols?
5. What is a Class D IP address?
6. Why do I sometimes lose a server's address when using more than one server?
7. What is Firewall?
8. How do I monitor the activity of sockets?
9. How would I put my socket in non-blocking mode?
10. What are RAW sockets?
11. What is the role of TCP protocol and IP protocol.
12. What is UDP?
13. How can I make my server a daemon?
14. How should I choose a port number for my server?
15. Layers in TCP/IP

16. How can I be sure that a UDP message is received?
17. How to get IP header of a UDP message
18. Writing UDP/SOCK_DGRAM applications
19. How many bytes in an IPX network address?
20. What is the difference between MUTEX and Semaphore?
21. What is priority inversion?
22. Different Solutions to dining philosophers problem.
23. What is a message queue?
24. Questions on Shared Memory.
25. What is DHCP?
26. Working of ping, telnet, gopher.
27. Can I connect two computers to internet using same line ?
28. Working of TCP and SSL Handshake
29. How P2P softwares work?
30. Setting up TOMCAT web service
31. Port numbers for FTP, HTTP, telnet, POP, finger
32. Difference - Passive FTP, Active FTP
33. Maximum Transmission Unit (MTU) what is it?
34. Security threats due to use of CGI
35. What is "spoofing"
36. Where could you find Apache server web log
37. Find web visitors by country
38. What is Virtual Private Network (VPN) and how does it work?
39. How does routing work?

Read More Answers.

Question # 13

How do I open a socket?

Answer:-

If you are programming a client, then you would open a socket like this:
Socket MyClient;
MyClient = new Socket("Machine name", PortNumber);
Where Machine name is the machine you are trying to open a connection to, and PortNumber is the port (a number) on which the server you are trying to connect to is running. When selecting a port number, you should note that port numbers between 0 and 1,023 are reserved for privileged users (that is, super user or root). These port numbers are reserved for standard services, such as email, FTP, and HTTP. When selecting a port number for your server, select one that is greater than 1,023!
In the example above, we didn't make use of exception handling, however, it is a good idea to handle exceptions. (From now on, all our code will handle exceptions!)
The above can be written as:

```
Socket MyClient;
   try {
 MyClient = new Socket("Machine name", PortNumber);
   }
   catch (IOException e) {
      System.out.println(e);
   }
```

If you are programming a server,
then this is how you open a socket:

```
ServerSocket MyService;
   try {
MyServerice = new ServerSocket(PortNumber);
      }
      catch (IOException e) {
        System.out.println(e);
      }
```

When implementing a server you also need
to create a socket object from the ServerSocket
in order to listen for and accept connections
from clients.

```
Socket clientSocket = null;
   try {
     serviceSocket = MyService.accept();
      }
   catch (IOException e) {
     System.out.println(e);
   }
```

Read More Answers.

Question # 14

How do I create an input stream?

Answer:-

On the client side, you can use the DataInputStream class to create an input stream to receive response from the server:

```
DataInputStream input;
try {
input = new DataInputStream(MyClient.getInputStream());
}
catch (IOException e) {
System.out.println(e);
```

}
The class DataInputStream allows you to read lines of text and Java primitive data types in a portable way. It has methods such as read, readChar, readInt, readDouble, and readLine,. Use whichever function you think suits your needs depending on the type of data that you receive from the server.
On the server side, you can use DataInputStream to receive input from the client:

```
DataInputStream input;
try {
input = new DataInputStream(serviceSocket.getInputStream());
}
catch (IOException e) {
System.out.println(e);
}
```

Read More Answers.

Question # 15

How do I create an output stream?

Answer:-

On the client side, you can create an output stream to send information to the server socket using the class PrintStream or DataOutputStream of java.io:

```
PrintStream output;
try {
output = new PrintStream(MyClient.getOutputStream());
}
catch (IOException e) {
System.out.println(e);
}
```

The class PrintStream has methods for displaying textual representation of Java primitive data types. Its Write and println methods are important here. Also, you may want to use the DataOutputStream:

```
DataOutputStream output;
try {
output = new DataOutputStream(MyClient.getOutputStream());
}
catch (IOException e) {
System.out.println(e);
}
```

The class DataOutputStream allows you to write Java primitive data types; many of its methods write a single Java primitive type to the output stream. The method writeBytes is a useful one.
On the server side, you can use the class PrintStream to send information to the client.

```
PrintStream output;
try {
output = new PrintStream(serviceSocket.getOutputStream());
}
catch (IOException e) {
System.out.println(e);
}
```

Note: You can use the class DataOutputStream as mentioned above.

Read More Answers.

Question # 16

How do I close sockets?

Answer:-

You should always close the output and input stream before you close the socket.
On the client side:

```
try {
output.close();
input.close();
MyClient.close();
}
catch (IOException e) {
System.out.println(e);
}
```

On the server side:

```
try {
output.close();
input.close();
serviceSocket.close();
MyService.close();
}
catch (IOException e) {
System.out.println(e);
}
```

Read More Answers.

Question # 17

Explain simple mail transfer protocol?

Answer:-

Write an SMTP (simple mail transfer protocol) client -- one so simple that we have all the data encapsulated within the program. You may change the code around to suit your needs. An interesting modification would be to change it so that you accept the data from the command-line argument and also get the input (the body of the

message) from standard input. Try to modify it so that it behaves the same as the mail program that comes with Unix.

```
import java.io.*;
import java.net.*;
public class smtpClient {
public static void main(String[] args) {
// declaration section:
// smtpClient: our client socket
// os: output stream
// is: input stream
Socket smtpSocket = null;
DataOutputStream os = null;
DataInputStream is = null;
// Initialization section:
// Try to open a socket on port 25
// Try to open input and output streams
try {
smtpSocket = new Socket("hostname", 25);
os = new DataOutputStream(smtpSocket.getOutputStream());
is = new DataInputStream(smtpSocket.getInputStream());
} catch (UnknownHostException e) {
System.err.println("Don't know about host: hostname");
} catch (IOException e) {
System.err.println("Couldn't get I/O for the connection to: hostname");
}
// If everything has been initialized then we want to write some data
// to the socket we have opened a connection to on port 25
if (smtpSocket != null && os != null && is != null) {
try {
// The capital string before each colon has a special meaning to SMTP
// you may want to read the SMTP specification, RFC1822/3
os.writeBytes("HELOn");
os.writeBytes("MAIL From: k3is@fundy.csd.unbsj.can");
os.writeBytes("RCPT To: k3is@fundy.csd.unbsj.can");
os.writeBytes("DATAn");
os.writeBytes("From: k3is@fundy.csd.unbsj.can");
os.writeBytes("Subject: testingn");
os.writeBytes("Hi theren"); // message body
os.writeBytes("n.n");
os.writeBytes("QUIT");
// keep on reading from/to the socket till we receive the "Ok" from SMTP,
// once we received that then we want to break.
String responseLine;
while ((responseLine = is.readLine()) != null) {
System.out.println("Server: " + responseLine);
if (responseLine.indexOf("Ok") != -1) {
break;
}
}
// clean up:
// close the output stream
// close the input stream
// close the socket
os.close();
is.close();
smtpSocket.close();
} catch (UnknownHostException e) {
System.err.println("Trying to connect to unknown host: " + e);
} catch (IOException e) {
System.err.println("IOException: " + e);
}
}
}
}
```

When programming a client, you must follow these four steps:

* Open a socket.
* Open an input and output stream to the socket.
* Read from and write to the socket according to the server's protocol.
* Clean up.

These steps are pretty much the same for all clients. The only step that varies is step three, since it depends on the server you are talking to.

Read More Answers.

Question # 18

Explain simple Echo server?

Answer:-

2. Echo server

Write a server. This server is very similar to the echo server running on port 7. Basically, the echo server receives text from the client and then sends that exact text back to the client. This is just about the simplest server you can write. Note that this server handles only one client. Try to modify it to handle multiple clients using threads.

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

```
public class echo3 {
public static void main(String args[]) {
// declaration section:
// declare a server socket and a client socket for the server
// declare an input and an output stream
ServerSocket echoServer = null;
String line;
DataInputStream is;
PrintStream os;
Socket clientSocket = null;
// Try to open a server socket on port 9999
// Note that we can't choose a port less than 1023 if we are not
// privileged users (root)
try {
echoServer = new ServerSocket(9999);
}
catch (IOException e) {
System.out.println(e);
}
// Create a socket object from the ServerSocket to listen and accept
// connections.
// Open input and output streams
try {
clientSocket = echoServer.accept();
is = new DataInputStream(clientSocket.getInputStream());
os = new PrintStream(clientSocket.getOutputStream());
// As long as we receive data, echo that data back to the client.
while (true) {
line = is.readLine();
os.println(line);
}
}
catch (IOException e) {
System.out.println(e);
}
}
}
```

Read More Answers.

Question # 19

What is Socket Programming?

Answer:-

Sockets are a generalized networking capability first introduced in 4.1cBSD and subsequently refined into their current form with 4.2BSD. The sockets feature is available with most current UNIX system releases. (Transport Layer Interface (TLI) is the System V alternative). Sockets allow communication between two different processes on the same or different machines. Internet protocols are used by default for communication between machines; other protocols such as DECnet can be used if they are available.
To a programmer a socket looks and behaves much like a low level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes. The differences between sockets and normal file descriptors occurs in the creation of a socket and through a variety of special operations to control a socket. These operations are different between sockets and normal file descriptors because of the additional complexity in establishing network connections when compared with normal disk access.
For most operations using sockets, the roles of client and server must be assigned. A server is a process which does some function on request from a client. As will be seen in this discussion, the roles are not symmetric and cannot be reversed without some effort.
This description of the use of sockets progresses in three stages:
The use of sockets in a connectionless or datagram mode between client and server processes on the same host. In this situation, the client does not explicitly establish a connection with the server. The client, of course, must know the server's address. The server, in turn, simply waits for a message to show up. The client's address is one of the parameters of the message receive request and is used by the server for response.
The use of sockets in a connected mode between client and server on the same host. In this case, the roles of client and server are further reinforced by the way in which the socket is established and used. This model is often referred to as a connection-oriented client-server model.
The use of sockets in a connected mode between client and server on different hosts. This is the network extension of Stage 2, above.
The connectionless or datagram mode between client and server on different hosts is not explicitly discussed here. Its use can be inferred from the presentations made in Stages 1 and 3.

Read More Answers.

Question # 20

What this function socketpair() does?

Answer:-

Socket Creation Using socketpair()
#include <sys/types.h>
#include <sys/socket.h>
int socketpair(int af, int type, int protocol, int sv[2])
socketpair() results in the creation of two connected sockets. sv[] is the array where the file descriptors for the sockets are returned. Each descriptor in sv[] is associated with one end of the communications link. Each descriptor can be used for both input and output. This means that full two-way communication between a parent process and one child process is possible.
Normally, one descriptor is reserved for use by a parent process and the other descriptor is used by a child process. The parent process closes the descriptor used by the child process. Conversely, the child process closes the descriptor used by the parent process. fork() is still required to pass one of the sockets to a child.
af represents the domain or address family to which the socket belongs. type is the type of socket to create.
Domains refer to the area where the communicating processes exist. Commonly used domains include:
* AF_UNIX for communication between processes on one system;

* AF_INET for communication between processes on the same or different systems using the DARPA standard protocols (IP/UDP/TCP).
Socket type refers to the "style" of communication. The two most commonly used values include:
* SOCK_STREAM: A stream of data with no record boundaries. Delivery in a networked environment is guaranteed; if delivery is impossible, the sender receives an error indicator.
* SOCK_DGRAM: A stream of records, each of a given size. Delivery in a networked environment is not guaranteed.
A protocol value of 0 is very common. This permits the system to choose the first protocol which is permitted with the pair of values specified for family and type.
Sample Code

```
#define DATA1   "test string 1"
#define DATA2   "test string 2"
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <errno.h>
main()
{
  int sockets[2], child;
  char buf[1024];
  /* Get the socket pair */
if (socketpair(AF_UNIX, SOCK_STREAM,
    0, sockets) < 0) {
 printf("error %d on socketpairn", errno);
     exit(1);
  }
  /* create child process */
  if ((child = fork()) == -1) {
    printf("fork error %dn", errno);
    exit(1);
  }
  if (child != 0) { /* this is the parent */
    /* close child's end of socket */
    close(sockets[0]);
    /* read message from child */
 if (read(sockets[1], buf, sizeof(buf)) < 0) {
 printf("error %d reading socketn", errno);
     exit(1);
    }
    printf("-->%sn", buf);
 /* write message to child */
 if (write(sockets[1],
 DATA1, sizeof(DATA1)) < 0) {
 printf("error %d writing socketn", errno);
     exit(1);
    }
    /* finished */
    close(sockets[1]);
  } else {  /* the child */
    /* close parent's end of socket */
    close(sockets[1]);
    /* send message to parent */
    if (write(sockets[0], DATA2,
     sizeof(DATA1)) < 0) {
 printf("error %d writing socketn", errno);
     exit(1);
    }
    /* get message from parent */
 if (read(sockets[0],
 buf, sizeof(buf)) < 0) {
printf("error %d reading socketn", errno);
     exit(1);
    }
    printf("-->%sn", buf);
    /* finished */
    close(sockets[0]);
  }
}
```

Read More Answers.

Question # 21

What this function socket() does?

Answer:-

Socket Creation Using socket()

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

socket() is very similar to socketpair() except that only one socket is created instead of two. This is most commonly used if the process you wish to communicate with is not a child process. The af, type, and protocol fields are used just as in the socketpair() system call.
On success, a file descriptor to the socket is returned. On failure, -1 is returned and errno describes the problem.

Read More Answers.

Question # 22

What this function bind() does?

Answer:-

Giving a Socket a Name - bind()
#include <sys/types.h>
#include <sys/socket.h>
int bind(int s, struct sockaddr *name, int namelen)
Recall that, using socketpair(), sockets could only be shared between parent and child processes or children of the same parent. With a name attached to the socket, any process on the system can describe (and use) it.
In a call to bind(), s is the file descriptor for the socket, obtained from the call to socket(). name is a pointer to a structure of type sockaddr. If the address family is AF_UNIX (as specified when the socket is created), the structure is defined as follows:

```
struct sockaddr {
 u_short sa_family;
 char    sa_data[14];
 };
```

name.sa_family should be AF_UNIX. name.sa_data should contain up to 14 bytes of a file name which will be assigned to the socket. namelen gives the actual length of name, that is, the length of the initialized contents of the data structure.
A value of 0 is return on success. On failure, -1 is returned with errno describing the error.
Example:

```
struct sockaddr name;
int s;
name.sa_family = AF_UNIX;
strcpy(name.sa_data, "/tmp/sock");
if((s = socket(AF_UNIX, SOCK_STREAM, 0) < 0)
   {
printf("socket create failure %dn", errno);
       exit(0);
   }
if (bind(s, &name, strlen(name.sa_data) +
   sizeof(name.sa_family)) < 0)
      printf("bind failure %dn", errno);
```

Read More Answers.

Question # 23

What this function connect() does?

Answer:-

Specifying a Remote Socket - connect()
#include <sys/types.h>
#include <sys/socket.h>
int connect(int s, struct sockaddr *name, int namelen)
The bind() call only allows specification of a local address. To specify the remote side of an address connection the connect() call is used. In the call to connect, s is the file descriptor for the socket. name is a pointer to a structure of type sockaddr:

```
struct sockaddr {
 u_short sa_family;
 char    sa_data[14];
 };
```

As with the bind() system call, name.sa_family should be AF_UNIX. name.sa_data should contain up to 14 bytes of a file name which will be assigned to the socket. namelen gives the actual length of name. A return value of 0 indicates success, while a value of -1 indicates failure with errno describing the error.
A sample code fragment:

```
struct sockaddr name;
name.sa_family = AF_UNIX;
strcpy(name.sa_data, "/tmp/sock");
if (connect(s, &name, strlen
(name.sa_data) +
sizeof(name.sa_family)) < 0) {
printf("connect failure %dn", errno);
}
```

Read More Answers.

Question # 24

What this function sendto() does?

Answer:-

Sending to a Named Socket - sendto()
int sendto(int s, char *msg, int len, int flags, struct sockaddr *to, int tolen)
This function allows a message msg of length len to be sent on a socket with descriptor s to the socket named by to and tolen, where tolen is the actual length of to. flags will always be zero for our purposes. The number of characters sent is the return value of the function. On error, -1 is returned and errno describes the error.
An example:

```
struct sockaddr to_name;
to_name.sa_family = AF_UNIX;
strcpy(to_name.sa_data, "/tmp/sock");
if (sendto(s, buf, sizeof(buf),
 0, &to_name,
 strlen(to_name.sa_data) +
sizeof(to_name.sa_family)) < 0) {
 printf("send failuren");
 exit(1);
```

```
}
```

Read More Answers.

Question # 25

What this function recvfrom() does?

Answer:-

Receiving on a Named Socket - recvfrom()
#include <sys/types.h>
#include <sys/socket.h>
int recvfrom(int s, char *msg, int len, int flags,struct sockaddr *from, int *fromlen)
This function allows a message msg of maximum length len to be read from a socket with descriptor s from the socket named by from and fromlen, where fromlen is the actual length of from. The number of characters actually read from the socket is the return value of the function. On error, -1 is returned and errno describes the error. flags may be 0, or may specify MSG_PEEK to examine a message without actually receiving it from the queue.
If no message is available to be read, the process will suspend waiting for one unless the socket is set to nonblocking mode (via an ioctl call).
The system I/O call read() can also be used to read data from a socket.

Read More Answers.

Question # 26

How to disposing of a Socket?

Answer:-

Code Sample: How to disposing of a Socket

```
#include <stdio.h>
void close(int s).
/* The I/O call close() will close the socket
descriptor s just as it closes
any open file descriptor.
Example - sendto() and recvfrom()
*/
/* receiver */
#include <sys/types.h>
#include <sys/socket.h>
struct sockaddr myname;
struct sockaddr from_name;
char buf[80];
main()
{
  int   sock;
  int   fromlen, cnt;
  sock = socket(AF_UNIX, SOCK_DGRAM, 0);
  if (sock < 0) {
    printf("socket failure %dn", errno);
    exit(1);
  }
  myname.sa_family = AF_UNIX;
  strcpy(myname.sa_data, "/tmp/tsck");
if (bind(sock, &myname,
 strlen(myname.sa_data) +
       sizeof(name.sa_family)) < 0) {
    printf("bind failure %dn", errno);
    exit(1);
  }
  cnt = recvfrom(sock, buf, sizeof(buf),
    0, &from_name, &fromlen);
  if (cnt < 0) {
    printf("recvfrom failure %dn", errno);
    exit(1);
  }
  buf[cnt] = ' ';  /* assure null byte */
  from_name.sa_data[fromlen] = ' ';
  printf("'%s' received from %sn",
    buf, from_name.sa_data);
}
/* sender */
#include <sys/types.h>
#include <sys/socket.h>
char buf[80];
struct sockaddr to_name;
main()
{
  int   sock;
  sock = socket(AF_UNIX, SOCK_DGRAM, 0);
  if (sock < 0) {
    printf("socket failure %dn", errno);
    exit(1);
  }
  to_name.sa_family = AF_UNIX;
  strcpy(to_name.sa_data, "/tmp/tsck");
  strcpy(buf, "test data line");
```

```
 cnt = sendto(sock, buf, strlen(buf),
  0, &to_name,
    strlen(to_name.sa_data) +
    sizeof(to_name.sa_family));
  if (cnt < 0) {
    printf("sendto failure %dn", errno);
    exit(1);
  }
}
```

Read More Answers.

Question # 27

How Sockets can be used to write client-server applications using a connection-oriented client-server technique?

Answer:-

Sockets can be used to write client-server applications using a connection-oriented client-server technique. Some characteristics of this technique include:
* The server can handle multiple client requests for connection and service.
* The server responds to any one client's request independently of all other clients.
* A client knows how to establish a connection with the server.

The client-server connection, when established, remains in existence until either the client or the server explicitly breaks it, much like a trouble-free telephone call. The socket used for this connection is called a connection-oriented socket.
The socket type is specified as SOCK_STREAM. As a result, the process receiving a message processes that message by the following rules:
* The data transmitted has no boundaries.
* All bytes in a received message must be read before the next message can be processed.
* Bytes in a received message can be read in a loop program control structure since no data bytes are discarded.
* The server will usually fork() a child process upon establishment of a client connection.
* This child server process is designed to communicate with exactly one client process.
* The child server process performs the requested service for its connected client.
* The child server process terminates when the service request has been completed.

Functions listen() and accept() enable the server to listen for service requests. read() and write() may be used by client and server to send/receive messages; send() and recv() may also be used.

Read More Answers.

Question # 28

How to make a Socket a Listen-only Connection Endpoint - listen()?

Answer:-

```
/*
Code Sample: Make a Socket a Listen-only
Connection Endpoint - listen()
by GlobalGuideline.com
*/
#include <sys/types.h>
#include <sys/socket.h>
int listen(int s, int backlog)
/*
listen establishes the socket as a passive
endpoint of a connection.  It does not
suspend process execution.
*/
/*
No messages can be sent through this socket.
Incoming messages can be received.
*/
/*
s is the file descriptor associated with
the socket created using the socket() system
call. backlog is the size of the queue
of waiting requests while the server is busy
with a service request. The current
system-imposed maximum value is 5.
*/
/*
0 is returned on success, -1 on error
 with errno indicating the problem.
*/
Example:
#include <sys/types.h>
#include <sys/socket.h>
int sockfd; /* socket file descriptor */
if(listen(sockfd, 5) < 0)
   printf ("listen error %dn", errno);
```

Read More Answers.

Question # 29

Explain Connection Establishment by Server - accept()?

Answer:-

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

```
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *name,
     int *namelen)
```

The accept() call establishes a client-server connection on the server side. (The client requests the connection using the connect() system call.) The server must have created the socket using socket(), given the socket a name using bind(), and established a listen queue using listen().
sockfd is the socket file descriptor returned from the socket() system call. name is a pointer to a structure of type sockaddr as described above

```
struct sockaddr {
 u_short sa_family;
 char    sa_data[14];
};
```

Upon successful return from accept(), this structure will contain the protocol address of the client's socket. The data area pointed to by namelen should be initialized to the actual length of name. Upon successful return from accept, the data area pointed to by namelen will contain the actual length of the protocol address of the client's socket.
If successful, accept() creates a new socket of the same family, type, and protocol as sockfd. The file descriptor for this new socket is the return value of accept(). This new socket is used for all communications with the client.
If there is no client connection request waiting, accept() will block until a client request is queued.
accept() will fail mainly if sockfd is not a file descriptor for a socket or if the socket type is not SOCK_STREAM. In this case, accept() returns the value -1 and errno describes the problem.

Read More Answers.

Question # 30

Explain Data Transfer over Connected Sockets - send() and recv()?

Answer:-

Two additional data transfer library calls, namely send() and recv(), are available if the sockets are connected. They correspond very closely to the read() and write() functions used for I/O on ordinary file descriptors.

```
#include <sys/types.h>
#include <sys/socket.h>
int send(int sd, char *buf, int len, int flags)
int recv(int sd, char * buf, int len, int flags)
```

In both cases, sd is the socket descriptor. For send(), buf points to a buffer containing the data to be sent, len is the length of the data and flags will usually be 0. The return value is the number of bytes sent if successful. If not successful, -1 is returned and errno describes the error.
For recv(), buf points to a data area into which the received data is copied, len is the size of this data area in bytes, and flags is usually either 0 or set to MSG_PEEK if the received data is to be retained in the system after it is received. The return value is the number of bytes received if successful. If not successful, -1 is returned and errno describes the error.

Read More Answers.

Question # 31

Explain with a Connection-Oriented Example - listen(), accept()?

Answer:-

```
/* Generic program structure for establishing
connection-oriented client-server environment.
 */
/* server program */
#include <stdio.h>
#include <errno.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
struct sockaddr myname;
char buf[80];
main()
{
  int sock, new_sd, adrlen, cnt;
sock = socket(AF_UNIX, SOCK_STREAM, 0);
  if (sock < 0) {
printf("server socket failure %dn", errno);
    perror("server: ");
    exit(1);
  }
  myname.sa_family = AF_UNIX;
  strcpy(myname.sa_data, "/tmp/billb");
  adrlen = strlen(myname.sa_data) +
      sizeof(myname.sa_family);
unlink("/tmp/billb"); /* defensive programming */
  if (bind(sock, &myname, adrlen) < 0) {
    printf("server bind failure %dn", errno);
    perror("server: ");
    exit(1);
  }
  if (listen(sock, 5) < 0 {
  printf("server listen failure %dn", errno);
    perror("server: ");
    exit(1);
  }
  /* Ignore child process termination.   */
  signal (SIGCHLD, SIG_IGN);
/*  Place the server in an infinite loop, waiting
  on connection requests to come from clients.
```

```
  In practice, there would need to be a clean
  way to terminate this process, but for now it
 will simply stay resident until terminated by
 the starting terminal or the super-user.    */
  while (1) {
if (new_sd = accept(sock,
 &myname, &adrlen)) < 0 {
 printf("server accept failure %dn", errno);
      perror("server: ");
      exit(1);
    }
 /* Create child server process.  Parent does no
further processing -- it loops back to wait
  for another connection request.            */
printf("Socket address in server %d is %x, %sn",
      getpid(), myname.sa_data, myname.sa_data);
    if (fork() == 0) {  /* child process */
    close (sock);  /* child does not need it  */
      /*  . . . . .  */
      cnt = read(new_sd, buf, strlen(buf));
   printf ("Server with pid %d got message %sn",
         getpid(), buf);
      strcpy (buf, "Message to client");
      cnt = write(new_sd, buf, strlen(buf));
 printf("Socket address in server %d is %x, %sn",
        getpid(), myname.sa_data, myname.sa_data);
      /*  . . . . .  */
    close (new_sd); /* close prior to exiting  */
      exit(0);
  }   /* closing bracket for if (fork() ... )  */
  }  /* closing bracket for while (1) ... ) */
}  /* closing bracket for main procedure */
/* client program */
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
char buf[80];
struct sockaddr myname;
main()
{
  int   sock, adrlen, cnt;
  sock = socket(AF_UNIX, SOCK_STREAM, 0);
  if (sock < 0) {
 printf("client socket failure %dn", errno);
    perror("client: ");
    exit(1);
  }
  myname.sa_family = AF_UNIX;
  strcpy(myname.sa_data, "/tmp/billb");
  adrlen = strlen(myname.sa_data) +
      sizeof(myname.sa_family);
if (connect( sock, &myname, adrlen) < 0) {
    printf("client connect failure %dn", errno);
    perror("client: ");
    exit(1);
  }
  /*  . . . . .  */
  strcpy(buf, "Message sent to server");
  cnt = write(sock, buf, strlen(buf));
  cnt = read(sock, buf, strlen(buf));
  printf("Client with pid %d got message %sn",
     getpid(), buf);
printf("Socket address in server %d is %x, %sn",
     getpid(), myname.sa_data, myname.sa_data);
  /*  . . . . .  */
  exit(0);
}
```

Read More Answers.

Question # 32

What is the IP Address?

Answer:-

The IP host address, or more commonly just IP address, is used to identify hosts connected to the Internet. IP stands for Internet Protocol and refers to the Internet Layer of the overall network architecture of the Internet. An IP address is a 32-bit quantity interpreted as 4 8-bit numbers or octets. Each IP address uniquely identifies the participating user network, the host on the network, and the class of the user network.
An IP address is usually written in a dotted-decimal notation of the form N1.N2.N3.N4, where each Ni is a decimal number between 0 and 255 decimal (00 through ff hexadecimal). Addresses are controlled and assigned by the Internet Network Information Center at SRI International. There are currently 5 defined classes of networks, called A, B, C, D and E; classes A, B and C are available to users. Each interface on a host system connected to the Internet must have its own IP address.

Read More Answers.

Question # 33

What is Network Host Names?

Answer:-

Using IP addresses to access hosts on a network is fine for the IP software. Most people are more comfortable with names, and procedures for both proper name construction and translation of these names into IP addresses has been in existence for some time. The most commonly used is the Domain Name System (DNS), occasionally but inaccurately referred to as the Domain Name Service. Naming in DNS is done hierarchically, that is, in a tree-structure format, much like the UNIX file system naming. The top two levels are controlled by the Internet Network Information Center (NIC) at SRI International.
At the top of the domain are two-letter country designators and three-letter (usually) general category designators within the USA. Some examples are:

* fr -- France
* gov -- government
* nz -- New Zealand
* com -- commercial business
* us -- USA
* edu -- educational institution
* uk -- United Kingdom
* mil -- military

The next level usually identifies the institution. For example:

* ibm -- IBM Corporation
* utdallas -- UT-D
* ti -- Texas Instruments (TI)
* nasa -- NASA

DNS and other software help in maintenance of these naming conventions and in the translation of host names to IP addresses and vice versa.

Read More Answers.

Question # 34

What is the /etc/hosts File?

Answer:-

The correspondence between host names and IP addresses is maintained in a file called hosts in the (top-level) directory /etc. On most systems, any user can read this file. (A word of caution: on many systems, printing this file may be injurious to local paper supply!!)
Entries in this file look like the following:
127.0.0.1 localhost
192.217.44.208 snoopy beagle hound metlife
153.110.34.18 bugs wabbit wascal
153.110.34.19 elmer
153.110.34.20 sam
Note that more than one name may be associated with a given IP address. This file is used when converting from IP address to host name and vice versa.

Read More Answers.

Question # 35

How to Selecte a Service Port?

Answer:-

The protocol specifications of the protocols used in the AF_INET domain require specification of a port. The port number is used to determine which process on the remote machine to talk to.
Certain network services have been assigned Well Known Ports. The port assignments to network services can be found in the file /etc/services. Selection of a Well Known Port involves searching this file and is done with the following functions:

```
#include <netdb.h>
struct servent *getservbyname(char *name,char *proto)
struct servent *getservbyport(int port,char *proto)
```

The two options for proto in each call are tcp for stream oriented communications, and udp for datagram oriented communications. port is the (known) port number when the service name is requested, while name is the character string containing the service name when the port number is requested.
The return value for each function is a pointer to a structure with the following form:

```
struct    servent {
char *s_name;      /* official name of service */
char **s_aliases;  /* alias service name list */
long s_port;       /* port service resides at */
char *s_proto;     /* protocol to use */
    };
```

If a program does not need to communicate with a Well Known Port it is possible to choose an unused port for use by a program.

Read More Answers.

Question # 36

How to Put a Host Program Address Together?

Answer:-

Once a host address and port number are known then the complete process address must be put together in a form that may be used by the system calls already covered. The structures set up to allow this follow:

```
#include <netinet/in.h>
/*
 * Internet address
 (a structure for historical reasons)
 */
struct in_addr {
    union {
struct { u_char s_b1,s_b2,s_b3,s_b4; } S_un_b;
  struct { u_short s_w1,s_w2; } S_un_w;
```

```
        u_long S_addr;
    } S_un;
#define s_addr  S_un.S_addr
/* can be used for most tcp & ip code */
};
/*
 * Socket address, internet style.
 */
struct sockaddr_in {
      short   sin_family;
      u_short sin_port;
      struct  in_addr sin_addr;
      char    sin_zero[8];
};
```

Filling in the fields for sockaddr_in will produce an Internet version of a socket address.

Read More Answers.

Question # 37

How to finding a Machine Address?

Answer:-

Finding an address that can be used to connect to a remote machine is done with either of the following commands:

```
#include <netdb.h>
struct hostent *gethostbyname(char *name)
struct hostent *gethostbyaddr(char *addr,
    int len, int type)
```

name contains the host name for which the IP address is needed. addr points to a structure of type in_addr and len is the size in bytes of this structure. In this discussion type is always AF_INET since the discussion is limited to use of IP addresses on the Internet.
Both calls return a pointer to a host entry structure.
This structure has the following form:

```
struct hostent {
char   *h_name; /*official name of host*/
char   **h_aliases;  /* alias list */
int    h_addrtype;   /* address type */
int    h_length;     /* length of address */
char   **h_addr_list;
/* list of addresses from name server */
 #define  h_addr h_addr_list[0]
/* address for backward compatibility */
 };
```

Read More Answers.

Question # 38

Explain Routines for converting data between a hosts internal representation and Network Byte Order?

Answer:-

Routines for converting data between a host's internal representation and Network Byte Order are:

```
#include <sys/types.h>
#include <netinet/in.h>
u_long htonl(u_long hostlong);
u_short htons(u_short hostshort);
u_long ntohl(u_long netlong);
u_short ntohs(u_short netshort);
```

These functions are macros and result in the insertion of conversion source code into the calling program. On little-endian machines the code will change the values around to network byte order. On big-endian machines no code is inserted since none is needed; the functions are defined as null.

Read More Answers.

Question # 39

Explain Network Address Conversion Routines?

Answer:-

Network Address Conversion Routines

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
    /* in_addr structure */
unsigned long inet_addr(char *ptr)
char *inet_ntoa(struct in_addr inaddr)
```

inet_addr() converts a character string in dotted-decimal notation to a 32-bit Internet address. The return value is not consistent, unfortunately. The correct return should be a pointer to a structure of type in_addr but many systems, following an older convention, return only the internal representation of the dotted-decimal notation. The man pages will clarify the situation for the host system on which the function is used.
inet_ntoa() expects a structure of type in_addr as a parameter (note that the structure itself is passed, not a pointer) and returns a pointer to a character string containing the dotted-decimal representation of the Internet address.

Read More Answers.

Question # 40

Socket Programming in C using TCP with Code?

Answer:-

To compile :
* linux : gcc -Wall -o foo foo.c
* solaris :gcc -Wall -o foo foo.c -lsocket -lnsl
1. TCP
* TCP server : simple TCP server
that prints received messages.
source : tcpServer.c
usage : ./tcpServer
* TCP client : simple TCP client
that sends data to server.
source : tcpClient.c
usage :./tcpClient server data1...dataN

```
/********** tcpserver.c **********/
by : prasad
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h>/*close*/
#define SUCCESS 0
#define ERROR    1
#define END_LINE 0x0
#define SERVER_PORT 1500
#define MAX_MSG 100
/* function readline */
int read_line();
int main (int argc, char *argv[])
{

   int sd, newSd, cliLen;
struct sockaddr_in cliAddr, servAddr;
   char line[MAX_MSG];
   /* create socket */
sd = socket(AF_INET, SOCK_STREAM, 0);
    if(sd<0) {
      perror("cannot open socket ");
      return ERROR;
   }

/* bind server port */
   servAddr.sin_family = AF_INET;
servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
   servAddr.sin_port = htons(SERVER_PORT);

if(bind(sd, (struct sockaddr *) &servAddr,
sizeof(servAddr))<0) {
      perror("cannot bind port ");
      return ERROR;
   }
   listen(sd,5);

   while(1) {
printf("%s: waiting for data on port TCP %un"
,argv[0],SERVER_PORT);
      cliLen = sizeof(cliAddr);
newSd = accept(sd, (struct sockaddr *)
   &cliAddr, &cliLen);
      if(newSd<0) {
 perror("cannot accept connection ");
        return ERROR;
      }

      /* init line */
      memset(line,0x0,MAX_MSG);

      /* receive segments */
while(read_line(newSd,line)!=ERROR) {

printf("%s: received from %s:TCP%d :
  %sn", argv[0],
       inet_ntoa(cliAddr.sin_addr),
       ntohs(cliAddr.sin_port), line);
         /* init line */
         memset(line,0x0,MAX_MSG);

      } /* while(read_line) */

   } /* while (1) */
}
```

```
/* WARNING WARNING WARNING WARNING WARNING
WARNING WARNING */
/* this function is experimental..
I don't know yet if it works */
/* correctly or not. Use Steven's
readline() function to have */
/* something robust.       */
/* WARNING WARNING WARNING
WARNING WARNING WARNING WARNING */
/* rcv_line is my function readline().
Data is read from the socket when */
/* needed, but not byte after bytes.
All the received data is read.
 */
/* This means only one call to recv(),
instead of one call for
    */
/* each received byte.
    */
/* You can set END_CHAR to whatever
means endofline for you. (0x0A is n)*/
/* read_lin returns the number of
bytes returned in line_to_return
    */
int read_line(int newSd, char *line_to_return)
{

   static int rcv_ptr=0;
   static char rcv_msg[MAX_MSG];
   static int n;
   int offset;
   offset=0;
   while(1) {
      if(rcv_ptr==0) {
         /* read data from socket */
 memset(rcv_msg,0x0,MAX_MSG);
  /* init buffer */
 n = recv(newSd, rcv_msg, MAX_MSG, 0);
  /* wait for data */
         if (n<0) {
perror(" cannot receive data ");
return ERROR;
         } else if (n==0) {
printf(" connection closed by clientn");
close(newSd);
return ERROR;
         }
      }

      /* if new data read on socket */
      /* OR */
      /* if another line is still in buffer */
/* copy line into 'line_to_return' */
 while(*(rcv_msg+rcv_ptr)!=
END_LINE && rcv_ptr<n) {
memcpy(line_to_return+offset,rcv_msg+rcv_ptr,1);
         offset++;
         rcv_ptr++;
      }

/*end of line + end of buffer =<returnline*/
      if(rcv_ptr==n-1) {
         /* set last byte to END_LINE */
         *(line_to_return+offset)=END_LINE;
         rcv_ptr=0;
         return ++offset;
      }

/* end of line but still some data in
buffer =>return line */
      if(rcv_ptr <n-1) {
 /* set last byte to END_LINE */
  *(line_to_return+offset)=END_LINE;
         rcv_ptr++;
         return ++offset;
      }
/* end of buffer but line is not ended =>*/
/* wait for more data to arrive on socket */
      if(rcv_ptr == n) {
         rcv_ptr = 0;
      }

   } /* while */
```

```
}
/******* tcpclient.c *******/
/* tcpClient.c */
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <unistd.h> /*close*/
#define SERVER_PORT 1500
#define MAX_MSG 100
int main (int argc, char *argv[]) {
   int sd, rc, i;
struct sockaddr_in localAddr, servAddr;
   struct hostent *h;

   if(argc < 3) {
 printf("usage: %s <server> <data1>
  <data2>  ... <dataN>n",argv[0]);
      exit(1);
   }
   h = gethostbyname(argv[1]);
   if(h==NULL) {
      printf("%s: unknown host '%s'n"
        ,argv[0],argv[1]);
      exit(1);
   }
servAddr.sin_family = h-<h_addrtype;
memcpy((char *) &servAddr.sin_addr.s_addr,
h-<h_addr_list[0],h-<h_length);
   servAddr.sin_port = htons(SERVER_PORT);
   /* create socket */
   sd = socket(AF_INET, SOCK_STREAM, 0);
   if(sd<0) {
      perror("cannot open socket ");
      exit(1);
   }
   /* bind any port number */
   localAddr.sin_family = AF_INET;
localAddr.sin_addr.s_addr =
     htonl(INADDR_ANY);
   localAddr.sin_port = htons(0);

   rc = bind(sd, (struct sockaddr *) &
      localAddr, sizeof(localAddr));
   if(rc<0) {
printf("%s: cannot bind port TCP %un",
  argv[0],SERVER_PORT);
      perror("error ");
      exit(1);
   }
   /* connect to server */
rc = connect(sd, (struct sockaddr *)
   &servAddr, sizeof(servAddr));
   if(rc<0) {
      perror("cannot connect ");
      exit(1);
   }
   for(i=2;i<argc;i++) {

rc = send(sd, argv[i], strlen(argv[i])
   + 1, 0);

      if(rc<0) {
         perror("cannot send data ");
         close(sd);
         exit(1);

      }
printf("%s: data%u sent (%s)n",
   argv[0],i-1,argv[i]);

   }
return 0;

}
```

Read More Answers.

Question # 41

RMI vs. Sockets and Object Serialization

Answer:-

RMI vs. Sockets and Object Serialization
The Remote Method Invocation (RMI) is a Java system that can be used to easily develop distributed object-based applications. RMI, which makes extensive use of object serialization, can be expressed by the following formula:
RMI = Sockets + Object Serialization + Some Utilities
The utilities are the rmi registry and the compiler to generate stubs and skeletons.
If you are familiar with RMI, you would know that developing distributed object-based applications in RMI is much simpler than using sockets. So why bother with sockets and object serialization then?
The advantages of RMI in comparison with sockets are:
* Simplicity: RMI is much easier to work with than sockets
* No protocol design: unlike sockets, when working with RMI there is no need to worry about designing a protocol between the client and server -- a process that is error-prone.
The simplicity of RMI, however, comes at the expense of the network. There is a communication overhead involved when using RMI and that is due to the RMI registry and client stubs or proxies that make remote invocations transparent. For each RMI remote object there is a need for a proxy, which slows the performance down.
(by)Qusay H. Mahmoud

Read More Answers.

Question # 42

Explain How to transport your own custom objects?

Answer:-

In this example, write an array multiplier server. The client sends two objects, each representing an array; the server receives the objects, unpack them by invoking a method and multiplies the arrays together and sends the output array (as an object) to the client. The client unpacks the array by invoking a method and prints the new array.
Start by making the class, whose objects will be transportable over sockets, serializable by implementing the Serializable interface as shown in Code Sample
Code Sample : SerializedObject.java

```
import java.io.*;
import java.util.*;
public class SerializedObject
implements Serializable {
  private int array[] = null;
  public SerializedObject() {
  }
public void setArray(int array[]) {
    this.array = array;
  }
  public int[] getArray() {
    return array;
  }
}
```

The next step is to develop the client. In this example, the client creates two instances of SerializedObject and writes them to the output stream (to the server), as shown from the source code in Code Sample.
Code Sample: ArrayClient.java

```
import java.io.*;
import java.net.*;
public class ArrayClient {
  public static void main(String argv[])
   {
     ObjectOutputStream oos = null;
     ObjectInputStream ois = null;
     // two arrays
  int dataset1[] = {3, 3, 3, 3, 3, 3, 3};
  int dataset2[] = {5, 5, 5, 5, 5, 5, 5};
     try {
       // open a socket connection
Socket socket = new Socket("YourMachineNameORipAddress",
 4000);
       // open I/O streams for objects
oos = new ObjectOutputStream(socket.getOutputStream());
ois = new ObjectInputStream(socket.getInputStream());
       // create two serialized objects
SerializedObject so1 = new SerializedObject();
SerializedObject so2 = new SerializedObject();
       SerializedObject result = null;
       int outArray[] = new int[7];
       so1.setArray(dataset1);
       so2.setArray(dataset2);
       // write the objects to the server
       oos.writeObject(so1);
       oos.writeObject(so2);
       oos.flush();
       // read an object from the server
result = (SerializedObject) ois.readObject();
       outArray = result.getArray();
       System.out.print("The new array is: ");
// after unpacking the array, iterate through it
       for(int i=0;i<outArray.length;i++) {
         System.out.print(outArray[i] + "   ");
       }
       oos.close();
```

```
      ois.close();
    } catch(Exception e) {
      System.out.println(e.getMessage());
    }
  }
}
```

Read More Answers.

Question # 43

Explain How does the client receives the object and prints the date?

Answer:-

The client, DateClient, does not have to send any messages to the DateServer once a connection has been established. It simply receives a Date object that represents the current day and time of the remote machine. The client receives the object and prints the date as shown in Code Sample.
Code Sample: DateClient.java

```
import java.io.*;
import java.net.*;
import java.util.*;
public class DateClient {
public static void main(String argv[]) {
ObjectOutputStream oos = null;
ObjectInputStream ois = null;
Socket socket = null;
Date date = null;
try {
// open a socket connection
socket = new Socket("yourMachineNameORipAddress", 3000);
// open I/O streams for objects
oos = new ObjectOutputStream(socket.getOutputStream());
ois = new ObjectInputStream(socket.getInputStream());
// read an object from the server
date = (Date) ois.readObject();
System.out.print("The date is: " + date);
oos.close();
ois.close();
} catch(Exception e) {
System.out.println(e.getMessage());
    }
  }
}
```

To run this example, the first step is to replace the bold line in DateClient with the machine name or IP address where the DateServer will run. If both, the DateServer and DateClient, will run on the same machine then you can use "localhost" or "127.0.0.1" as the machine name. The next step is to compile the source files DateServer.java and DateClient.java. Then run the DateServer in one window (if you are working under Windows) or in the background (if you are working under UNIX) and run the DateClient. The client should print the current date and time of the remote machine.

Read More Answers.

Question # 44

Explain A multi-threaded DateServe that listens on port 3000 and waits for requests from clients?

Answer:-

Here A multi-threaded DateServer that listens on port 3000 and waits for requests from clients. Whenever there is a request, the server replies by sending a Date object (over sockets) to the client as shown in Code Sample.
Code Sample: DateServer.java

```
import java.io.*;
import java.net.*;
import java.util.*;
public class DateServer
 extends Thread {
  private ServerSocket dateServer;

  public static void main
  (String argv[]) throws Exception {
    new DateServer();
  }
public DateServer() throws Exception {
    dateServer = new ServerSocket(3000);
    System.out.println
    ("Server listening on port 3000.");
    this.start();
  }
  public void run() {
    while(true) {
      try {
System.out.println("Waiting for connections.");
Socket client = dateServer.accept();
System.out.println("Accepted a connection
 from: "+ client.getInetAddress());
       Connect c = new Connect(client);
      } catch(Exception e) {}
    }
```

```
  }
}
class Connect extends Thread {
  private Socket client = null;
  private ObjectInputStream ois = null;
  private ObjectOutputStream oos = null;

  public Connect() {}
  public Connect(Socket clientSocket) {
    client = clientSocket;
    try {
     ois = new ObjectInputStream
     (client.getInputStream());
     oos = new ObjectOutputStream
     (client.getOutputStream());
    } catch(Exception e1) {
       try {
         client.close();
       }catch(Exception e) {
 System.out.println(e.getMessage());
       }
       return;
    }
    this.start();
  }

  public void run() {
     try {
       oos.writeObject(new Date());
       oos.flush();
// close streams and connections
       ois.close();
       oos.close();
       client.close();
     } catch(Exception e) {}
  }
}
```

Read More Answers.

Question # 45

How to write a class that reads the objects that have been saved, and invokes a method?

Answer:-

Write a class that reads the objects that have been saved, and invokes a method as shown in Code Sample. Again, as with writeObject, the readObject method can be called any number of times to read any number of objects from the input stream.
Code Sample: ReadInfo.java

```
import java.io.*;
import java.util.Date;
public class ReadInfo {
public static void main(String argv[]) throws Exception {
FileInputStream fis = new FileInputStream("name.out");
ObjectInputStream ois = new ObjectInputStream(fis);
// read the objects from the input stream (the file name.out)
UserInfo user1 = (UserInfo) ois.readObject();
UserInfo user2 = (UserInfo) ois.readObject();
// invoke a method on the constructed object
user1.printInfo();
user2.printInfo();
ois.close();
fis.close();
  }
}
```

To try out this example, compile the source files: UserInfo.java, SaveInfo.java, and ReadInfo.java. Run SaveInfo, then ReadInfo, and you would see some output similar to this:

```
The name is: Java Duke
The name is: Java Blue
```

Read More Answers.

Question # 46

How to create a class that creates a instance of the UserInfo class and writes the object to an output stream?

Answer:-

Create a class that creates a instance of the UserInfo class and writes the object to an output stream as shown in Code Sample. The output stream in this example is a file called "name.out". The important thing to note from Code Sample is that the writeObject method can be called any number of times to write any number of objects to the output stream.
Code Sample: SaveInfo.java

```
import java.io.*;
import java.util.Date;
public class SaveInfo {
public static void main
```

```
(String argv[]) throws Exception {
    FileOutputStream fos =
    new FileOutputStream("name.out");
    ObjectOutputStream oos =
    new ObjectOutputStream(fos);
    // create two objects
UserInfo user1 = new UserInfo("Java Duke");
UserInfo user2 = new UserInfo("Java Blue");
// write the objects to the output stream
    oos.writeObject(user1);
    oos.writeObject(user2);
    oos.flush();
    oos.close();
    fos.close();
  }
}
```

Read More Answers.

Question # 47

How to serialize a custom class?

Answer:-

In this example, we create a custom class, UserInfo which is shown in Code Sample. To make it serializable, it implements the Serializable interface.
Code Sample 3: UserInfo.java

```
mport java.io.*;
import java.util.*;
public class
UserInfo implements Serializable {
   String name = null;
   public UserInfo(String name) {
      this.name = name;
   }
   public void printInfo() {
System.out.println("The name is: "+name);
   }
}
```

Read More Answers.

Question # 48

Shows how to read a serialized object and print its information?

Answer:-

Code Sample: ReadDate.java

```
import java.io.*;
import java.util.Date;
public class ReadDate {
public static void main
(String argv[]) throws Exception
 {
    FileInputStream fis =
     new FileInputStream("date.out");

    ObjectInputStream ois =
     new ObjectInputStream(fis);

    Date date = (Date) ois.readObject();

System.out.println("The date is: "+date);
    ois.close();
    fis.close();
  }
}
```

In the example above we have worked with an instance of the Date class, which is an existing serialized Java class. The question that may come to mind is: are all existing Java class serialized? The answer is: No. Either because they don't need to be, or it doesn't make sense to serialize some classes. To find out if a class is serializable, use the tool serialver that comes with the JDK. You can either use it from the command line as follows:

```
c:> serialver java.util.Date
java.util.Date: static final long serialVersionUID = 7523967970034938905L;
```

(In this example, we are testing if the Date class is serializable. The output here means that the Date class is serializable and it print its version unique identifier.)
Or, alternatively, you can use the GUI-based serialver tool using the command:

```
c:> serialver -show
```

This command pops up a window, where you can write the name of the class (including its path) that you want to check.

Read More Answers.

Question # 49

Shows how to save a Date object to a file?

Answer:-

Code Sample 1: SaveDate.java

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

```
import java.util.Date;
public class SaveDate {
public static void main(
String argv[]) throws Exception {
  FileOutputStream fos =
new FileOutputStream("date.out");
ObjectOutputStream oos =
 new ObjectOutputStream(fos);

    Date date = new Date();
    oos.writeObject(date);
    oos.flush();
    oos.close();
    fos.close();
  }
}
```

Read More Answers.

Question # 50

Overview of Object Serialization

Answer:-

Overview of Object Serialization
Object serialization is a mechanism that is useful in any program that wants to save the state of objects to a file and later read those objects to reconstruct the state of the program, or to send an object over the network using sockets. Serializing a class can be easily done simply by having the class implement the java.io.Serializable interface. This interface is a marker interface. In other words, it does not have any methods that need to be implemented by the class implementing it. It is mainly used to inform the Java virtual machine (JVM) that you want the object to be serialized.
There are two main classes that are used for reading and writing objects to streams: ObjectOutputStream and ObjectInputStream. The ObjectOutputStream provides the writeObject method for writing an object to an output stream, and the ObjectInputStream provides the readObject method for reading the object from an input stream. It is important to note that the objects used with these methods must be serialized. That is, their classes must implement the Serializable interface.

Read More Answers.

Question # 51

Explain How do sockets work?

Answer:-

A socket has a typical flow of events. In a connection-oriented client-to-server model, the socket on the server process waits for requests from a client. To do this, the server first establishes (binds) an address that clients can use to find the server. When the address is established, the server waits for clients to request a service. The server performs the client's request and sends the reply back to the client. The two endpoints establish a connection, and bring the client and server together. The client-to-server data exchange takes place when a client connects to the server through a socket.
1. The socket() function creates an endpoint for communications and returns a socket descriptor that represents the endpoint.
2. When an application has a socket descriptor, it can bind a unique name to the socket. Servers must bind a name to be accessible from the network.
3. The listen() function indicates a willingness to accept client connection requests. When a listen() is issued for a socket, that socket cannot actively initiate connection requests. The listen() API is issued after a socket is allocated with a socket() function and the bind() function binds a name to the socket. A listen() function must be issued before an accept() function is issued.
4. The client application uses a connect() function on a stream socket to establish a connection to the server.
5. Servers use stream sockets to accept a client connection request with the accept() function. The server must issue the bind() and listen() functions successfully before it can issue an accept() function to accept an incoming connection request.
6. When a connection is established between stream sockets (between client and server), you can use any of the socket API data transfer functions. Clients and servers have many data transfer functions from which to choose, such as send(), recv(), read(), write(), and others.
7. When a server or client wants to cease operations, it issues a close() function to release any system resources acquired by the socket.

Read More Answers.

Question # 52

Explain what is a socket?

Answer:-

A socket is a communications connection point (endpoint) that you can name and address in a network. The processes that use a socket can reside on the same system or on different systems on different networks. Sockets are useful for both stand-alone and network applications.
Sockets commonly are used for client/server interaction. Typical system configuration places the server on one machine, with the clients on other machines. The clients connect to the server, exchange information, and then disconnect.
Socket characteristics:
. A socket is represented by an integer. That integer is called a socket descriptor.
. A socket exists as long as the process maintains an open link to the socket.
. You can name a socket and use it to communicate with other sockets in a communication domain.
. Sockets perform the communication when the server accepts connections from them, or when it exchanges messages with them.
.You can create sockets in pairs.
The connection that a socket provides can be connection-oriented or connectionless.
Connection-oriented communication implies that a connection is established, and a dialog between the programs will follow. The program that provides the service (the server program) establishes the connection. It assigns itself a name that identifies where to obtain that service. The client of the service (the client program) must request the service of the server program. The client does this by connecting to the distinct name that the server program has designated. It is similar to dialing a telephone number (an identifier) and making a connection with another party that is offering a service (for example, a plumber). When the receiver of the call (the server) answers the telephone, the connection is established. The plumber verifies that you have reached the correct party, and the connection remains active as long as both parties require it.
Connectionless communication implies that no connection is established over which a dialog or data transfer can take place. Instead, the server program designates a name that identifies where to reach it (much like a post office box). By sending a letter to a post office box, you cannot be absolutely sure the letter is received. You may have to send another letter to reestablish communication.

Read More Answers.

Question # 53

What are basic functions of network programming interface?

Answer:-

Most operating systems provide precompiled programs that communicate across a network. Common examples into the TCP/IP world are web clients(browsers) and web servers, and the FTP and TELNET clients and servers.
A socket is an endpoint used by a process for bi-directional communication with a socket associated with another process. Sockets, introduced in Berkeley Unix, are a basic mechanism for IPC on a computer system, or on different computer systems connected by local or wide area networks(resource 2). To understand some structs into this subject is necessary a deeper knowledge about the operating system and his networking protocols. This subject can be used as either beginners programmers or as a reference for experienced programmers.
The Socket Function
Most network applications can be divided into two pieces: a client and a server.
Creating a socket
#include <sys/types.h>
#include <sys/socket.h>
When you create a socket there are three main parameters that you have to specify:
* the domain
* the type
* the protocol
int socket(int domain, int type, int protocol);
The Domain parameter specifies a communications domain within which communication will take place, in the example the domain parameter was AF_INET, that specify the ARPA Internet Protocols The Type parameter specifies the semantics of communication, in the mini chat used the Stream socket type(SOCK_STREAM), because it offers a bi-directional, reliable, two-way connection based byte stream(resource 2). Finally the protocol type, since used a Stream Socket type, must use a protocol that provide a connection-oriented protocol, like IP, decide to use IP in our protocol Type, and we saw in /etc/protocols the number of ip, 0. So the function now is:
s = socket(AF_INET , SOCK_STREAM , 0)
where 's' is the file descriptor returned by the socket function.
Since the mini chat is divided in two parts it will be divided the explanation in the server, the client and the both, showing the basic differences between them, as see next.
The Mini-chat Server structure
Binding a socket to a port and waiting for the connections
Like all services in a Network TCP/IP based, the sockets are always associated with a port, like Telnet is associated to Port 23, FTP to 21... In the Server, to do the same thing, bind some port to be prepared to listening for connections (that is the basic difference between Client and Server), Listing 2. Bind is used to specify for a socket the protocol port number where it will be waiting for messages.
So there is a question, which port could be new service? Since the system pre-defined a lot of ports between 1 and 7000 (/etc/services) choose the port number 15000.
The function of bind is:
int bind(int s, struct sockaddr *addr, int addrlen)
The struct necessary to make socket works is the struct sockaddr_in address; and the follow lines to say to system the information about the socket.
The type of socket
address.sin_family = AF_INET /* use a internet domain */
The IP used
address.sin_addr.s_addr = INADDR_ANY /*use a specific IP of host*/
The port used
address.sin_port = htons(15000); /* use a specific port number */
And finally bind our port to the socket
bind(create_socket , (struct sockaddr *)&address,sizeof(address));
Now another important phase, prepare a socket to accept messages from clients, the listen function is used on the server in the case of connection oriented communication and also the maximum number of pending connections(resource 3).
listen (create_socket, MAXNUMBER)
where MAXNUMER in the case is 3. And to finish, tell the server to accept a connection, using the accept() function. Accept is used with connection based sockets such as streams.
accept(create_socket,(struct sockaddr *)&address,&addrlen);
As see in Listing 2 The parameters are the socket descriptor of the master socket (create_socket), followed by a sockeaddr_in structure and the size of the structure.(resource 3)
The Mini-chat Client structure
Maybe the biggest difference is that client needs a Connect() function. The connect operation is used on the client side to identify and, possibly, start the connection to the server. The connect syntax is
connect(create_socket,(struct sockaddr *)&address,sizeof(address)) ;
The common structure
A common structure between Client and the Server is the use of the struct hostent as seeing in Listing 1 and 2. The use of the Send and Recv functions are another common codes.
The Send() function is used to send the buffer to the server
send(new_socket,buffer,bufsize,0);
and the Recv() function is used to receive the buffer from the server, look that it is used both in server and client.
recv(new_socket,buffer,bufsize,0);
Since the software of the TCP/IP protocol is inside the operating system, the exactly interface between an application and the TCP/IP protocols depends of the details of the operating system(resource 4). In the case, examine the UNIX BSD socket interface because Linux follow this. The Mini-chat developed here is nothing more than a explain model of a client/server application using sockets in Linux and should be used like a introduction of how easy is to develop applications using sockets. After understand this you can easily start to think about IPC (interprocess Communication), fork, threads(resource 5) and much more. The basic steps to make it work is:
1. Run the server
2. Run the client with the address of the server
By Pedro Paulo Ferreira Bueno and Antonio Pires de Castro Junior

Read More Answers.

Question # 54

What is Whois Client?

Answer:-

Whois Client: This is whois, a very simple and generic whois client. This client, unlike the classic whois client, does not check for supported flags at the client side,

except for -h (whois host) and -p (whois port).
The whois(1) client makes a TCP/IP connection
to the server and conducts the kind of protocol
that you would type if you where to make
a connection by hand:

```
   [7:30am julian] whois reggers
   There were 1 matches on your request.
            Full Name: Quinton, Reg
           Department: Info Tech Svcs
                 Room: NSC 214
                Phone: 679-2111x(6026)
            Index Key: 481800
        Machine Address: reggers@julian.uuu.com
     Directory Addresses: reg.quinton@uuu.com
                        : r.quinton@uuu.com
                        : reggers@uuu.com
                        : quinton@uuu.com
   For more information try 'whois help'.
```

The client sends the command "reggers", the server sends back the answer and the client displays the answer received to the user. When the server is finished the connection is closed.
Sample code: whois(1) client

```
sub tcpopen {
 use Socket;               # need socket interface
 my($server, $service) = @_;# args to this function
 my($proto, $port, $iaddr); # local variables
 my($handle)="$server::$service";
  # localized obscure handle
die("550:Cannot getprotobyname('tcp')rn")
      unless ($proto = getprotobyname('tcp'));
die("550:Cannot getservbyname($service)rn")
unless ($port = getservbyname($service, 'tcp'));
die("550:Cannot gethostbyname($server)rn")
     unless ($iaddr = gethostbyname($server));
die("550:Cannot create socketrn")
unless socket($handle, PF_INET, SOCK_STREAM, $proto);
die("550:Cannot connect($service://$server)rn")
unless connect($handle, sockaddr_in($port, $iaddr));
      # unbuffered I/O to that service
 select($handle); $| = 1; select(STDOUT); $| = 1;
     return($handle);
   }
```

Read More Answers.

Question # 55

How to run the WHOIS Daemon?

Answer:-

You can run the whois daemon (on the server) to
see what it does:

```
[3:27pm julian] echo reggers | /usr/lib/whois/whoisd
   There were 1 matches on your request.
            Full Name: Quinton, Reg
           Department: Info Tech Svcs
                 Room: NSC 214
                Phone: 679-2111x(6026)
            Index Key: 481800
        Machine Address: reggers@julian.uuu.com
     Directory Addresses: reg.quinton@uuu.com
                        : r.quinton@uuu.com
                        : reggers@uuu.com
                        : quinton@uuu.com
   For more information try 'whois help'.
```

The program is command driven -- you give a command (or query string) on stdin, it produces results on stdout, and exits.
Connecting to the Server:
You can make a telnet(1) connection to
the whois service on the server.

```
   [3:47pm julian] telnet julian whois
Trying 129.100.2.12 ... Connected to julian.
   Escape character is '^]'.
   reggers .... my command input
   There were 1 matches on your request.
            Full Name: Quinton, Reg
           Department: Info Tech Svcs
                 Room: NSC 214
                Phone: 679-2111x(6026)
            Index Key: 481800
        Machine Address: reggers@julian.uuu.com
     Directory Addresses: reg.quinton@uuu.com
                        : r.quinton@uuu.com
                        : reggers@uuu.com
                        : quinton@uuu.com
```

For more information try 'whois help'.
Connection closed by foreign host.

Read More Answers.

Question # 56

What is Whois Daemon?

Answer:-

Simple WHOIS Daemon, provides the standard Internet whois directory service It is much simpler to setup and administrator than the other whois daemons available, such as RIPE whois or the original SRC whois. This is because it uses a flat-text file, /etc/swhoisd.conf, instead of a complex database. This whois server is recommended only for small databases (preferably under 100 records and no more than 1000).

Read More Answers.

Question # 57

What is Inetd Services?

Answer:-

Not all services are started at boot time by running a server application. Eg. you won't usually see a process running for the finger service like you do for the smtp service. Many are handled by the InterNet Daemon inetd(1M). This is a generic service configured by the file inetd.conf(4).
[2:35pm julian] page /etc/inetd.conf
$Author: reggers $
$Date: 1997/05/02 20:17:16 $
#
Internet server configuration database
ftp stream tcp nowait root /usr/etc/ftpd ftpd
telnet stream tcp nowait root /usr/etc/telnetd telnetd
shell stream tcp nowait root /usr/etc/rshd rshd
login stream tcp nowait root /usr/etc/rlogind rlogind
exec stream tcp nowait root /usr/etc/rexecd rexecd
uucpd stream tcp nowait root /usr/etc/uucpd uucpd
finger stream tcp nowait nobody /usr/etc/fingerd fingerd
etc...
whois stream tcp nowait nobody /usr/lib/whois/whoisd whoisd
etc...
Inetd Comments:
For each service listed in /etc/inetd.conf the inetd(1M) process, and that is a process is started at boot time, executes the socket(3N), bind(3N), listen(3N) and accept(3N) calls as discussed above. Inetd also handles many of the daemon issues (signal handling, set process group and controlling tty) which we've studiously avoided.
The inetd(1M) process spawns the appropriate server application (with fork(2) and exec(2)) when a client connects to the service port. The daemon continues to listen for further connections to the service while the spawned child process handles the request which just came in.
The server application (ie. the child spawned by inetd(1M)) is started with stdin and stdout connected to the remote host.port of the client process which made the connection. Any input/output by the server appliation on stdin/stdout are sent/received by the client application. You have Inter Process Communication (or IPC)!
This means, any application written to use stdin/stdout can be a server application. Writing a server application should therefore be fairly simple.

Read More Answers.

Question # 58

How do we get a process bound behind a port?

Answer:-

Server Bind:
A Server uses bind(3N) to establish the local host.port assignment -- ie. so it is the process behind that port. That's really only required for servers -- applications which accept(3N) connections to provide a service.
struct servent *sp;
struct sockaddr_in sin;
if ((sp=getservbyname(service,"tcp")) == NULL) then error...
sin.sin_family=AF_INET;
sin.sin_port=sp->s_port;
sin.sin_addr.s_addr=htonl(INADDR_ANY);
if ((s=socket(AF_INET,SOCK_STREAM,0)) < 0) then error...
if (bind(s, &sin, sizeof(sin)) < 0) then error...
htonl(3N) converts a long to the right sequence (given different byte ordering on different machines). The IP address INADDR_ANY means all interfaces. You could, if you wanted, provide a service only on some interfaces -- eg. if you only provided the service on the loopback interface (127.0.0.1) then the service would only be available to clients on the same system.
What this code fragment does is specify a local interface and port (into the sin structure). The process is bound to that port -- it's now the process behind the local port.
Client applications usually aren't concerned about the local host.port assignment (the connect(3N) does a bind o some random but unused local port on the right interface). But rcp(1) and related programs (like rlogin(1) and rsh(1)) do connect from reserved port numbers.
For example, the version of tcpopen.c used in the Passwdd/Passwd -- An authentication Daemon/Client. There's an instance where a client application connects from a reserved port.
Listen and Accept:
To accept connections, a socket is created with socket(3N), it's bound to a service port with bind(3N), a queue for incoming connections is specified with listen(3N) and then the connections are accepted with accept(3N) as in this fragment:
struct servent *sp;
struct sockaddr_in sin,from;
if ((sp=getservbyname(service,"tcp")) == NULL)
then error...
sin.sin_family=etc...
if ((s=socket(AF_INET,SOCK_STREAM,0)) < 0)
then error...

```
if (bind(s, &sin, sizeof(sin)) < 0)
 then error...
if (listen(s,QUELEN) < 0) then error...
for (;;) {
 if ((g=accept(f,&from,&len)) < 0)
  then error...
        if (!fork()) {
          child handles request...
            ...and exits
          exit(0);
        }
 close(g);   /* parent releases file */
      }
```

This is the programming schema used by utilities like sendmail(1M) and others -- they create their socket and listen for connections. When connections are made, the process forks off a child to handle that service request and the parent process continues to listen for and accept further service requests.

Read More Answers.

Question # 59

What is Socket Addressing?

Answer:-

A Socket Address is a host.port pair (communication is between host.port pairs -- one on the server, the other on the client). We know how to determine host numbers and service numbers so we're well on our way to filling out a structure were we specify those numbers. The structure is sockaddr_in, which has the address family is AF_INET as in this fragment:

```
   int     tcpopen(host,service)
   char    *service, *host;
   {   int     unit;
       struct  sockaddr_in     sin;
       struct  servent         *sp;
       struct  hostent         *hp;
            etc...
if ((sp=getservbyname(service,"tcp"))
 == NULL) then error...
if ((hp=gethostbyname(host)) == NULL)
then error...
       bzero((char *)&sin, sizeof(sin));
       sin.sin_family=AF_INET;
bcopy(hp->h_addr,(char *)&sin.sin_addr,
  hp->h_length);
       sin.sin_port=sp->s_port;
            etc...
```

The code fragment is filling in the IP address type AF_INET, port number and IP address in the Socket Address structure -- the address of the remote host.port where we want to connect to find a service.
There's a generic Socket Address structure, a sockaddr, used for communication in arbitrary domains. It has an address family field and an address (or data) field:

```
/* from: /usr/include/sys/socket.h */

struct sockaddr {
 u_short sa_family; /*address family */
 char  sa_data[14];/*max 14 byte addr*/
   };
```

The sockaddr_in structure is for
Internet Socket Addresses
address family AF_INET). An instance
of the generic socket address.

```
/* from: /usr/include/netinet/in.h */
struct sockaddr_in {
  short   sin_family;   /* AF_INET   */
  u_short sin_port;     /* service port */
  struct  in_addr sin_addr; /*host number */
  char    sin_zero[8];  /* not used  */
   };
```

The family defines the interpretation of the data. In other domains addressing will be different -- services in the UNIX domain are names (eg. /dev/printer). In the sockaddr_in structure we've got fields to specify a port and a host IP number (and 8 octets that aren't used at all!). That structure specifies one end of an IPC connection. Creating that structure and filling in the right numbers has been pretty easy so far.

Read More Answers.

Question # 60

What is File Descriptors and Sockets?

Answer:-

File Descriptors:
File Descriptors are the fundamental I/O object. You read(2) and write(2) to file descriptors.

```
int cc, fd, nbytes;
char *buf;
cc = read(fd, buf, nbytes);
cc = write(fd, buf, nbytes)
```

The read attempts to read nbytes of data from the object referenced by the file descriptor fd into the buffer pointed to by buf. The write does a write to the file descriptor from the buffer. Unix I/O is a byte stream.
File descriptors are numbers used for I/O. Usually the result of open(2) and creat(2) calls.

All Unix applications run with stdin as file descriptor 0, stdout as file descriptor 1, and stderr as file descriptior 3. But stdin is a FILE (see stdio(3S)) not a file descriptor. If you want a stdio FILE on a file descriptor use fdopen(3S).
Sockets:
A Socket is a Unix file descriptor created by the socket(3N) call -- you don't open(2) or creat(2) a socket. By way of comparison pipe(2) creates file descriptors too -- you might be familiar with pipes which predate sockets in the development of the Unix system.

```
int s, domain, type, protocol;
s = socket(domain, type, protocol);
etc...
cc = read(s, buf, nbytes);
```

The domain parameter specifies a communications domain (or address family). For IP use AF_INET but note that socket.h lists all sorts of address families. This is to inform the system how an address should be understood -- on different networks, like AF_DECnet, addressing may be longer than the four octets of an IP number. We're only concerned with IP and the AF_INET address family.
The type parameter specifies the semantics of communication (sometimes know as a specification of quality of services). For TCP/IP use SOCK_STREAM (for UDP/IP use SOCK_DGRAM). Note that any address family might support those service types. See socket.h for a list of service types that might be supported.
A SOCK_STREAM is a sequenced, reliable, two-way connection based byte stream. If a data cannot be successfully transmitted within a reasonable length of time the connection is considered broken and I/O calls will indicate an error.
The protocol specifies a particular protocol to be used with the socket -- for TCP/IP use 0. Actually there's another programmers interface getprotobyname(3N) that provides translates protocol names to numbers. It's an interface to the data found in /etc/protocols -- compare with the translation of service names to port numbers discussed above.

Read More Answers.

Question # 61

What is Client Connect?

Answer:-

A client application creates a socket(3N)
and then issues a connect(3N)
to a service specified
in a sockaddr_in structure:

```
   int     tcpopen(host,service)
   char    *service, *host;
   {   int     unit;
       struct  sockaddr_in     sin;
       struct  servent         *sp;
       struct  hostent         *hp;
if ((sp=getservbyname(service,"tcp")) == NULL)
   then error...
 if ((hp=gethostbyname(host)) == NULL)
   then Ierror...
       bzero((char *)&sin, sizeof(sin))
            etc...
if ((unit=socket(AF_INET,SOCK_STREAM,0)) < 0)
  then error...
 if (connect(unit,&sin,sizeof(sin)) <0)
  then error...
       return(unit);
   }
```

The result returned is a file descriptor which is connected to a server process. A communications channel on which one can conduct an application specific protocol.
Client Communication:
Having connected a socket to a server to establish a file descriptor communication is with the usual Unix I/O calls. You have Inter Process Communication (or IPC) to a server.
Many programmers turn file descriptors into stdio(3S) streams so they can use fputs, fgets, fprintf, etc. -- use fdopen(3S).

```
   main(argc,argv)
   int     argc;
   char    *argv[];
   {
       int     unit,i;
       char    buf[BUFSIZ];
       FILE    *sockin,*sockout;
if ((unit=tcpopen(WHOHOST,WHOPORT))
            < 0) then error...
       sockin=fdopen(unit,"r");
       sockout=fdopen(unit,"w");
            etc...
       fprintf(sockout,"%sn",argv[i]);
            etc...
 while (fgets(buf,BUFSIZ,sockin)) etc...
```

Stdio Buffers:
Stdio streams have powerful manipulation tools (eg. fscanf is amazing). But beware, streams are buffered! This means a well placed fflush(3S) is often required to flush a buffer to the peer.

```
 fprintf(sockout,"%sn",argv[i]);
 fflush(sockout);
 while (fgets(buf,BUFSIZ,sockin)) etc...
```

Many client/server protocols are client driven -- the client sends a command and expects an answer. The server won't see the command if the client doesn't flush the output. Likewise, the client won't see the answer if the server doesn't flush it's output.
Watch out for client and server blocking -- both waiting for input from the other.

Read More Answers.

Question # 62

What is Server Applications?

Answer:-

Server Applications:
A system offers a service by having an application running that is listening at the service port and willing to accept a connection from a client. If there is no application listening at the service port then the machine doesn't offer that service.
The SMTP service is provided by an application listening on port 25. On Unix systems this is usually the sendmail(1M) application which is started at boot time.
[2:20pm julian] ps -agx | grep sendmail
419 ? SW 0:03 /usr/lib/sendmail -bd -q15m
18438 ? IW 0:01 /usr/lib/sendmail -bd -q15m
[2:28pm julian] netstat -a | grep smtp
tcp 0 0 julian.3155 acad3.alask.smtp SYN_SENT
tcp 0 0 *.smtp *.* LISTEN
In the example we have a process listening to the smtp port (for inbound mail) and another process talking to the smtp port on acad3.alaska.edu (ie. sending mail to that system).

Read More Answers.

Computer Programming Most Popular Interview Topics.

1 : PHP Frequently Asked Interview Questions and Answers Guide.

2 : C++ Programming Frequently Asked Interview Questions and Answers Guide.

3 : C Programming Frequently Asked Interview Questions and Answers Guide.

4 : Software engineering Frequently Asked Interview Questions and Answers Guide.

5 : Cobol Frequently Asked Interview Questions and Answers Guide.

6 : Visual Basic (VB) Frequently Asked Interview Questions and Answers Guide.

7 : Perl Programming Frequently Asked Interview Questions and Answers Guide.

8 : VBA Frequently Asked Interview Questions and Answers Guide.

9 : OOP Frequently Asked Interview Questions and Answers Guide.

10 : Python Frequently Asked Interview Questions and Answers Guide.

About Global Guideline.

Global Guideline is a platform to develop your own skills with thousands of job interview questions and web tutorials for fresher's and experienced candidates. These interview questions and web tutorials will help you strengthen your technical skills, prepare for the interviews and quickly revise the concepts. Global Guideline invite you to unlock your potentials with thousands of **Interview Questions with Answers** and much more. Learn the most common technologies at Global Guideline. We will help you to explore the resources of the World Wide Web and develop your own skills from the basics to the advanced. Here you will learn anything quite easily and you will really enjoy while learning. Global Guideline will help you to become a professional and Expert, well prepared for the future.

* This PDF was generated from https://GlobalGuideline.com at **November 29th, 2023**

* If any answer or question is incorrect or inappropriate or you have correct answer or you found any problem in this document then don't hesitate feel free and e-mail us we will fix it.

You can follow us on FaceBook for latest Jobs, Updates and other interviews material.
www.facebook.com/InterviewQuestionsAnswers

Follow us on Twitter for latest Jobs and interview preparation guides
https://twitter.com/InterviewGuide

Best Of Luck.

Global Guideline Team
https://GlobalGuideline.com
Info@globalguideline.com