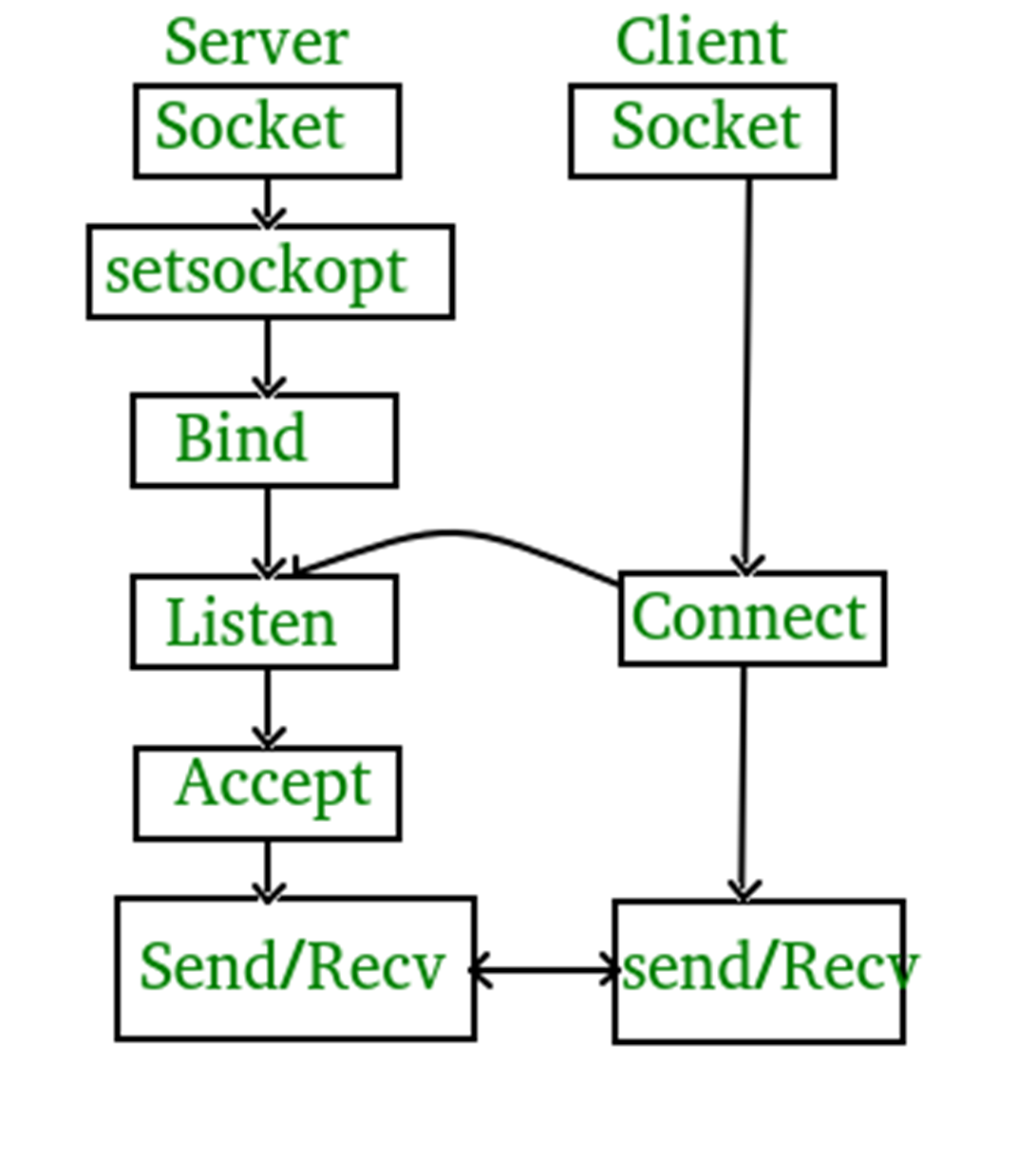
Client Server Report



**For server------**

The steps involved in establishing a socket on the server side are as follows:

1->Create a socket with the socket() function call

2->Bind the socket to an address using the bind() system call. For a server socket on the Internet, an address consists of a port number on the host machine.

(in this we are assigning these).

3->Listen for connections with the listen() system call

4->Accept a connection with the accept() system call. This call typically blocks until a client connects with the server.

5->Send and receive messages from clients.

6->close.

so let's see in the detail.

**\*\*\*Header files which we will use in our program.**

#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <iostream>

This header file contains declarations used in most input and output and is typically included in all CPP programs.

#include <sys/types.h>

This header file contains definitions of a number of data types used in system calls. These types are used in the next two include files.

#include <sys/socket.h>

The header file socket.h includes a number of definitions of structures needed for sockets.

#include <netinet/in.h>

The header file in.h contains constants and structures needed for internet domain addresses.

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

{

int sockfd, newsockfd, portno, clilen;

}

**sockfd and newsockfd**

These two variables store the values returned by the socket system call and the accept system call.

portno stores the port number on which the server accepts connections.

clilen stores the size of the address of the client. This is needed for the accept system call.

char buffer[256];

The server reads characters from the socket connection into this buffer.

**struct sockaddr\_in serv\_addr, cli\_addr;**

A sockaddr\_in is a structure containing an internet address. This structure is defined in netinet/in.h.

Here is the definition:

struct sockaddr\_in

{

short sin\_family; ( AF\_INET)

u\_short sin\_port;

struct in\_addr sin\_addr;

char sin\_zero[8]; ( zero )

};

An in\_addr structure, defined in the same header file, contains only one field, an unsigned long called s\_addr.

The variable serv\_addr will contain the address of the server, and cli\_addr will contain the address of the client which connects to the server.

**#creating socket------**

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

if (sockfd < 0)

error("ERROR opening socket");

The socket() system call creates a new socket. It takes three arguments. The first is the address domain of the socket.

The second argument is the type of socket.

The third argument is the protocol.It will choose TCP for stream sockets and UDP for datagram sockets.If the socket call fails, it returns -1.

**portno = atoi(argv[1]);**

The port number on which the server will listen for connections is passed in as an argument, and this statement uses the atoi() function to

convert this from a string of digits to an integer.

**serv\_addr.sin\_family = AF\_INET;**

The variable serv\_addr is a structure of type struct sockaddr\_in. This structure has four fields. The first field is short sin\_family,

which contains a code for the address family. It should always be set to the symbolic constant AF\_INET.

**serv\_addr.sin\_port = htons(portno);**

The second field of serv\_addr is unsigned short sin\_port, which contain the port number. However, instead

of simply copying the port number to this field, it is necessary to convert this to network byte order using the function htons()

which converts a port number in host byte order to a port number in network byte order.

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

The third field of sockaddr\_in is a structure of type struct in\_addr which contains only a single field unsigned long s\_addr. This field

contains the IP address of the host. For server code, this will always be the IP address of the machine on which the server is running, and

there is a symbolic constant INADDR\_ANY which gets this address.

**#Binding socket.**

if (bind(sockfd, (struct sockaddr \*) &serv\_addr,sizeof(serv\_addr)) < 0)

error("ERROR on binding");

The bind() system call binds a socket to an address, in this case the address of the current host and port number on which the server

will run. It takes three arguments, the socket file descriptor, the address to which it is bound, and the size of the address to which it is bound. The second argument is a pointer to a structure of type sockaddr, but what is passed in is a structure of type sockaddr\_in, and

so this must be cast to the correct type. This can fail for a number of reasons, the most obvious being that this socket is already in use on this machine. The bind() manual has more information.

**listen(sockfd,5);**

The listen system call allows the process to listen on the socket for connections. The first argument is the socket file descriptor, and the second is the size of the backlog queue, i.e., the number of connections that can be waiting while the process is handling a particular connection. This should be set to 5, the maximum size permitted by most systems. If the first argument is a valid socket, this call cannot fail, and so the code doesn't check for errors. The listen() man page has more information.

**clilen = sizeof(cli\_addr);**

newsockfd = accept(sockfd, (struct sockaddr \*) &cli\_addr, &clilen);

if (newsockfd < 0)

error("ERROR on accept");

**#accept**

The accept() system call causes the process to block until a client connects to the server. Thus, it wakes up the process when a connection from a client has been successfully established. It returns a new file descriptor, and all communication on this connection should be done using the new file descriptor. The second argument is a reference pointer to the address of the client on the other end of the connection, and the third argument is the size of this structure. The accept() man page has more information. bzero(buffer,256);

n = read(newsockfd,buffer,255);

**if (n < 0) error("ERROR reading from socket");**

printf("Here is the message: %s",buffer);

Note that we would only get to this point after a client has successfully connected to our server. This code initializes the buffer using the bzero() function, and then reads from the socket. Note that the read call uses the new file descriptor, the one returned by accept(), not the original file descriptor returned by socket(). Note also that the read() will block until there is something for it to read in the socket, i.e. after the client has executed a write().

It will read either the total number of characters in the socket or 255, whichever is less, and return the number of characters read. The read() man page has more information.

n = write(newsockfd,"I got your message",18);

**if (n < 0) error("ERROR writing to socket");**

Once a connection has been established, both ends can both read and write to the connection. Naturally, everything written by the client will be read by the server, and everything written by the server will be read by the client.

At the end we will close our socket like on a phone call we hang up after our work.

When we talk the word Client, it means to talk of a person or an organization using a particular service. Similarly in the digital world a Client is a computer (Host) i.e. capable of receiving information or using a particular service from the service providers (Servers).

Firstly we have to include the header file #include<sys/socket.h>.This file contains the sockaddr structure.The sockaddr structure is used to define a socket address which is used in bind(), connect(), getsockname() functions . this file also definesockaddr\_storage structure.

Then we include the header file #include<unistd.h> .unistd.h is the name of the header file that provides access to the POSIX operating system API. It is defined by the POSIX.1 standard, the base of the Single Unix Specification, and should therefore be available in any POSIX-compliant operating system and compiler. For instance, this includes Unix and Unix-like operating systems, such as GNU variants, distributions of Linux.

Then we include the header file #include<arpa/inet.h>.Thisheader file contains definitions for internet operations.

Socket -

Sockets are commonly used for client and server interaction. 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.

Connect -

The functionconnect() attempts to establish a connection between two sockets. For datagram sockets, the connect() call specifies the peer for a socket. The socket parameter is the socket used to originate the connection request. The connect() call performs two tasks when called for a stream socket.

The function connect work as-

A client initiates the communication by connecting to a server. The client sends requests to the server, and the server sends replies back. Finally, the client disconnects. A server might handle connections from many clients simultaneously, and clients might also connect to multiple servers.