Introduction To Socket Programming:

1. Socket Programming: -

In the 1980s, the US government's Advanced Research Projects Agency (ARPA) provided funds to the University of California at Berkeley to implement TCP/IP protocols under the UNIX operating system. During this project, a group of Berkeley researchers developed an application program interface (API) for TCP/IP network communications called the socket interface. The socket interface is an API TCP/IP networks i.e. it defines a variety of software functions or routines for the development of applications for TCP/IP networks. The socket interface designers originally built their interface into the UNIX operating system. However, the other operating systems, environments, such as Microsoft Windows, implement the socket interface as software libraries. However, regardless of the environment in which we program, the program code will look much the same.

Socket programming can be done in any language, which supports network communication, but java is preferred because it is platform independent, it has exception mechanisms for robust handling of common problems that occur during I/O and networking operations and its threading facilities provide a way to easily implement powerful servers One of java's major strong suits as a programming language for client-server application development is its wide range of network support. Java has this advantage because it was developed with the Internet in mind. Another advantage of java is that it provides security. The result is that we have a lot of options in regard to network programming in Java. Java performs all of its network communication through sockets.

2. Linux / Unix System: -

Linux is a family of free and open-source software operating systems based on the Linux kernel, an operating system kernel first released on September 17, 1991 by Linus Torvalds. Linux is typically packaged in a Linux distribution.

Distributions include the Linux kernel and supporting system software and libraries, many of which are provided by the GNU Project. Many Linux distributions use the word "Linux" in their name, but the

Free Software Foundation uses the name GNU/Linux to emphasize the importance of GNU software, causing some controversy.

Popular Linux distributions include Debian, Fedora, and Ubuntu. Commercial distributions include Red Hat Enterprise Linux and SUSE Linux Enterprise Server. Desktop Linux distributions include a windowing system such as X11 or Wayland, and a desktop environment such as GNOME or KDE Plasma. Distributions intended for servers may omit graphics altogether, and include a solution stack such as LAMP. Because Linux is freely redistributable, anyone may create a distribution for any purpose.

The Unix operating system was conceived and implemented in 1969, at AT&T's Bell Laboratories in the United States by Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna. First released in 1971, Unix was written entirely in assembly language, as was common practice at the time. Later, in a key pioneering approach in 1973, it was rewritten in the C programming language by Dennis Ritchie. The availability of a high-level language implementation of Unix made its porting to different computer platforms easier.

Due to an earlier antitrust case forbidding it from entering the computer business, AT&T was required to license the operating system's source code to anyone who asked. As a result, Unix grew quickly and became widely adopted by academic institutions and businesses. In 1984, AT&T divested itself of Bell Labs; freed of the legal obligation requiring free licensing, Bell Labs began selling Unix as a proprietary product, where users were not legally allowed to modify Unix. The GNU Project, started in 1983 by Richard Stallman, had the goal of creating a "complete Unix-compatible software system" composed entirely of free software. Work began in 1984. Later, in 1985, Stallman started the Free Software Foundation and wrote the GNU General Public License (GNU GPL) in 1989. By the early 1990s, many of the programs required in an operating system (such as libraries, compilers, text editors, a Unix shell, and a windowing system) were completed, although low-level elements such as device drivers, daemons, and the kernel, called GNU/Hurd, were stalled and incomplete.

Linus Torvalds has stated that if the GNU kernel had been available at the time (1991), he would not have decided to write his own.

3. C Language: -

C is a general-purpose, imperative computer programming language, supporting structured programming, lexical variable scope and recursion, while a static type system prevents many unintended operations. By design, C provides constructs that map efficiently to typical machine instructions, and it has therefore found lasting use in applications that were previously coded in assembly language. Such applications include operating systems, as well as various application software for computers ranging from supercomputers to embedded systems.

C was originally developed at Bell Labs by Dennis Ritchie, between 1972 and 1973. It was created to make utilities running on Unix. Later, it was applied to re-implementing the kernel of the Unix operating system. During the 1980s, C gradually gained popularity. Nowadays, it is one of the most widely used programming languages, with C compilers from various vendors available for the majority of existing computer architectures and operating systems. C has been standardized by the American National Standards Institute (ANSI) since 1989 (see ANSI C) and subsequently by the International Organization for Standardization (ISO).

C is an imperative procedural language. It was designed to be compiled using a relatively straightforward compiler, to provide low-level access to memory, to provide language constructs that map efficiently to machine instructions, and to require minimal runtime support. Despite its low-level capabilities, the language was designed to encourage cross-platform programming. A standards-compliant C program that is written with portability in mind can be compiled for a wide variety of computer platforms and operating systems with few changes to its source code; the language has become available on various platforms, from embedded microcontrollers to supercomputers.

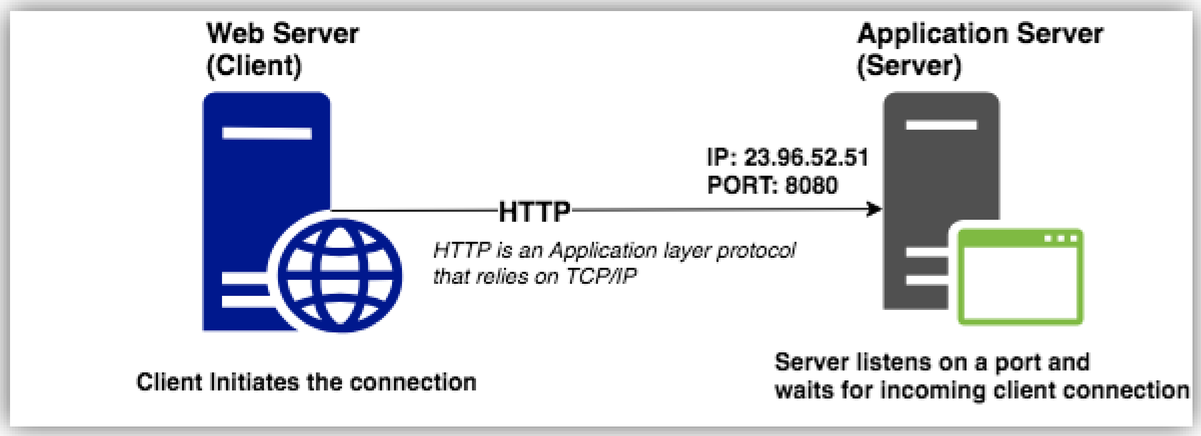
Introduction To Project: Port 8080

Port 8080 is commonly used as proxy and caching port. It is also above the service port range. Port 8080 also can run a Web server as a non-root user. Other assignments for port 8080 include Apache Tomcat, an M2MLogger and a Web GUI. If port 8080 is used in a Web address, it requires a default port override to be able to connect to port 8080 in place of the typical port 80.

It is a window or point of access to a specific place. You can redirect them in a router, take something sent to one port and alter the path to another. POP3 email programs, like Outlook or Thunderbird, send and receive email through specific ports.... 110 and 995 for receiving email, ports 25, 2525, and 443 for sending email, and ports 143 and 993 for connecting to IMAP servers. VNC servers typically accept connections on 5900, while pushing the java client on port 5800.

Your web browser works on port 80. Port 8080 is typically used for a personally hosted web server, when the ISP restricts this type of usage for non-commercial customers. If you were going to host your own website from your computer, you would prefer to be able to do so on port 80, since this would mean that anyone connecting to your computer wouldn't have to add a port number to the end of the WWW address you paid for. They could just connect to it, or to your specific IP address, and they'd have the website visible in their browser, while being served from your desktop or laptop.

Some ISPs want to avoid people paying for a cheaper home connection, but using it for commercial webservice. So, they restrict access on port 80. To get around this, you can use whatever port you like. You could use port 12345 if you wanted to. Port 8080 is the just the default second choice for a webserver.


Web Server (Client)
Application Server (Server)
IP: 23.96.52.51
PORT: 8080
HTTP
HTTP is an Application layer protocol that relies on TCP/IP
Client Initiates the connection
Server listens on a port and waits for incoming client connection

Modules Used In Project

1. #include <sys/types.h>: -

The <sys/types.h> header shall include definitions for at least the following types:

blkcnt_t:

Used for file block counts.

blksize_t:

Used for block sizes.

clock_t:

Used for system times in clock ticks or CLOCKS_PER_SEC; see <time.h>.

clockid_t:

Used for clock ID type in the clock and timer functions.

dev_t:

Used for device IDs.

fsblkcnt_t:

Used for file system block counts.

fsfilcnt_t:

Used for file system file counts.

gid_t:

Used for group IDs.

id_t:

Used as a general identifier; can be used to contain at least a pid_t, uid_t, or gid_t.

ino_t:

Used for file serial numbers.

key_t:

Used for XSI interprocess communication.

mode_t:

Used for some file attributes.

nlink_t:

Used for link counts.

off_t:

Used for file sizes.

pid_t:

Used for process IDs and process group IDs.

pthread_attr_t:

Used to identify a thread attribute object.

pthread_barrier_t:

Used to identify a barrier.

pthread_barrierattr_t:

[BAR] ⌦ Used to define a barrier attributes object. ⌫

pthread_cond_t:

Used for condition variables.

pthread_condattr_t:

Used to identify a condition attribute object.

2. #include <sys/socket.h>:-

<sys/socket.h> makes available a type, socklen_t, which is an unsigned opaque integral type of length of at least 32 bits. To forestall portability problems, it is recommended that applications should not use values larger than 2^{32} - 1.

The <sys/socket.h> header defines the unsigned integral type sa_family_t.
The <sys/socket.h> header defines the sockaddr structure that includes at least the following members:

• sa_family_t	sa_family	address family
• char	sa_data[]	socket address (variable-length data)

The <sys/socket.h> header defines the msghdr structure that includes at least the following members:

• void	*msg_name	optional address
• socklen_t	msg_namelen	size of address
• struct iovec	*msg_iov	scatter/gather array
• int	msg_iovlen	members in msg_iov
• void	*msg_control	ancillary data, see below
• socklen_t	msg_controllen	ancillary data buffer len
• int	msg_flags	flags on received message

The <sys/socket.h> header defines the cmsghdr structure that includes at least the following members:

• socklen_t	cmsg_len	data byte count, including the cmsghdr
• int	cmsg_level	originating protocol
• int	cmsg_type	protocol-specific type

3. #include <netinet/in.h>:-

The <netinet/in.h> header shall define the following types:
in_port_t:
    Equivalent to the type uint16_t as defined in <inttypes.h> .
in_addr_t:
    Equivalent to the type uint32_t as defined in <inttypes.h> .

The sa_family_t type shall be defined as described in <sys/socket.h>. The uint8_t and uint32_t type shall be defined as described in <inttypes.h>. Inclusion of the <netinet/in.h> header may also make visible all symbols from <inttypes.h> and <sys/socket.h>. The <netinet/in.h> header shall define the in_addr structure that includes at least the following member: in_addr_t s_addr

The <netinet/in.h> header shall define the sockaddr_in structure that includes at least the following members:

• sa_family_t	sin_family	AF_INET.
• in_port_t	sin_port	Port number.
• struct in_addr	sin_addr	IP address.

The sin_port and sin_addr members shall be in network byte order.
The sockaddr_in structure is used to store addresses for the Internet address family. Values of this type shall be cast by applications to struct sockaddr for use with socket functions.

[IP6] ⌦ The <netinet/in.h> header shall define the in6_addr structure that contains at least the following member: uint8_t s6_addr[16]

This array is used to contain a 128-bit IPv6 address, stored in network byte order.
The <netinet/in.h> header shall define the sockaddr_in6 structure that includes at least the following members:

sa_family_t	sin6_family	AF_INET6.
in_port_t	sin6_port	Port number.
uint32_t	sin6_flowinfo	IPv6 traffic class and flow information.
struct in6_addr	sin6_addr	IPv6 address.
uint32_t	sin6_scope_id	Set of interfaces for a scope.

The sin6_port and sin6_addr members shall be in network byte order.

4. #include<netdb.h>:-

The <netdb.h> header may make available the type in_port_t and the type in_addr_t as defined in the description of <netinet/in.h>. The <netdb.h> header defines the hostent structure that includes at least the following members:

char *h_name:
    Official name of the host.
char **h_aliases:
    A pointer to an array of pointers to alternative host names, terminated by a null pointer.
int   h_addrtype:
    Address type.
int   h_length:
    The length, in bytes, of the address.
char **h_addr_list:
    A pointer to an array of pointers to network addresses for the host, terminated by a null pointer.

The <netdb.h> header defines the netent structure that includes at least the following members:
char *n_name:
    Official, fully-qualified (including the domain) name of the host.
char **n_aliases:
    A pointer to an array of pointers to alternative network names, terminated by a null pointer.
int   n_addrtype:
    The address type of the network.
uint32_t n_net:
    The network number, in host byte order.
The uint_32_t type is made available by inclusion of <inttypes.h>.

The <netdb.h> header defines the protoent structure that includes at least the following members:
char *p_name:
    Official name of the protocol.
char **p_aliases:
    A pointer to an array of pointers to alternative protocol names, terminated by a null pointer.

int p_proto:

The protocol number.

The <netdb.h> header defines the following macros for use as error values for gethostbyaddr() and gethostbyname():

HOST_NOT_FOUND
NO_DATA
NO_RECOVERY
TRY_AGAIN

The following are declared as functions, and may also be defined as macros:

```
void  endhostent(void);
void  endnetent(void);
void  endprotoent(void);
void  endservent(void);
struct hostent  *gethostbyaddr(const void *addr, size_t len, int type);
struct hostent  *gethostbyname(const char *name);
struct hostent  *gethostent(void);
struct netent   *getnetbyaddr(uint32_t net, int type);
struct netent   *getnetbyname(const char *name);
struct netent   *getnetent(void);
struct protoent *getprotobyname(const char *name);
struct protoent *getprotobynumber(int proto);
struct protoent *getprotoent(void);
struct servent  *getservbyname(const char *name, const char *proto);
struct servent  *getservbyport(int port, const char *proto);
struct servent  *getservent(void);
void  sethostent(int stayopen);
void  setnetent(int stayopen);
void  setprotoent(int stayopen);
void  setservent(int stayopen);
```

Source Code

1. Server.c : -

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>


void error(char *msg)
{
	perror(msg);
	exit(1);
}


int main(int argc, char *argv[])
{
	int sockfd,newsockfd,portno,clilen;
	char buffer[256];
	struct sockaddr_in serv_addr,cli_addr;
	int n;

	if(argc<2)
	{
		fprintf(stderr,"\nERROR! Port number is not provided.");
		exit(1);
	}

	sockfd=socket(AF_INET,SOCK_STREAM,0);

	if(sockfd<0)
	{
		error("\nERROR! Opening socket.");
	}

	bzero((char *) &serv_addr, sizeof(serv_addr));

	portno=atoi(argv[1]);

	serv_addr.sin_family=AF_INET;
	serv_addr.sin_addr.s_addr=INADDR_ANY;
	serv_addr.sin_port=htons(portno);
	if(bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr))<0)
	{
		error("\nERROR! No binding.");
	}

	listen(sockfd,5);
	clilen=sizeof(cli_addr);

	newsockfd=accept(sockfd,(struct sockaddr *) &cli_addr, &clilen);

	if(newsockfd<0)
	{
		error("\nERROR! No accept.");
	}

	bzero(buffer,256);
	n=read(newsockfd,buffer,255);

	if(n<0)
	{
		error("\nERROR! Reading from socket.");
	}

	printf("\nMessage Received :--  %s",buffer);
	n=write(newsockfd,"Message successfully received.",18);

	if(n<0)
	{
		error("\nERROR! Writing to socket.");
	}

	return 0;
}
```

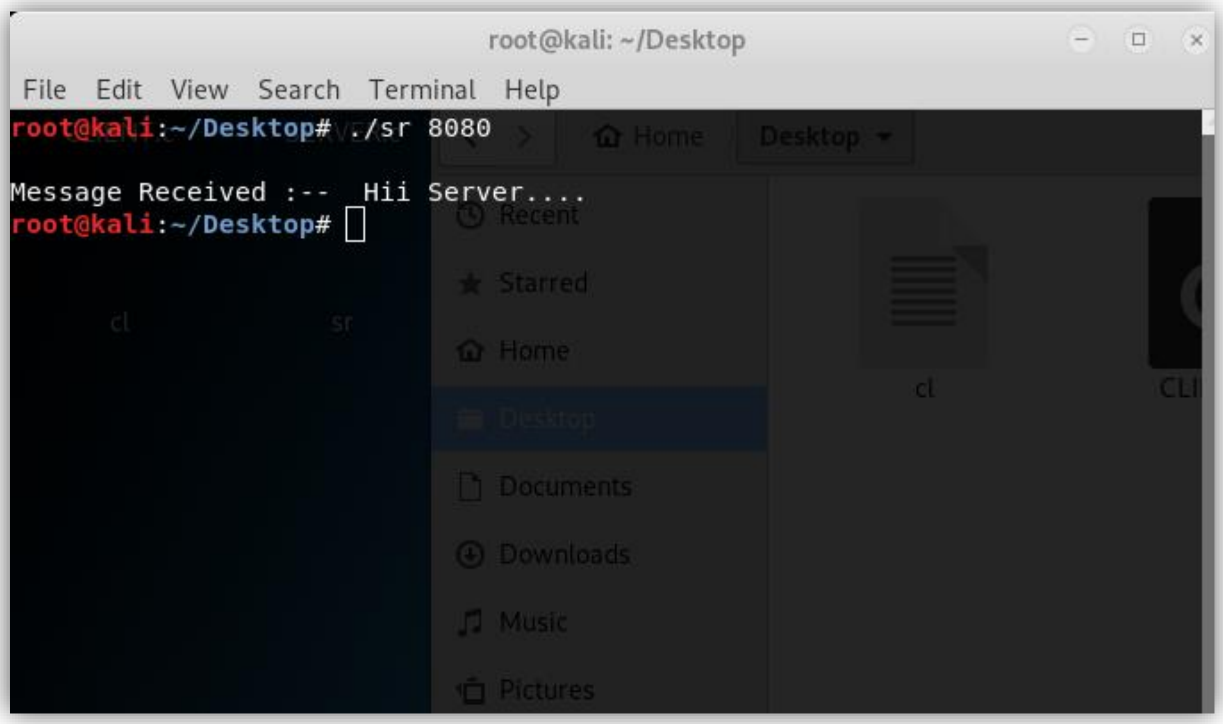
2. Client.c : -

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<netdb.h>


void error(char *msg)
```

```
{
        perror(msg);
        exit(0);
}


int main(int argc, char *argv[])
{
        int sockfd,portno,n;
        struct sockaddr_in serv_addr;
        struct hostent *server;

        char buffer[256];

        if(argc<3)
        {
                fprintf(stderr,"\nUsage %s host port.",argv[0]);
                exit(0);
        }

        portno=atoi(argv[2]);

        sockfd=socket(AF_INET,SOCK_STREAM,0);

        if(sockfd<0)
        {
                error("\nERROR! Opening socket.");
        }

        server=gethostbyname(argv[1]);

        if(server==NULL)
        {
                fprintf(stderr, "\nERROR! No such host found.");
                exit(0);
        }

        bzero((char *) &serv_addr, sizeof(serv_addr));
        serv_addr.sin_family=AF_INET;
        bcopy((char *)server->h_addr,(char *)&serv_addr.sin_addr.s_addr,server->h_length);
        serv_addr.sin_port=htons(portno);

        if(connect(sockfd,&serv_addr,sizeof(serv_addr))<0)
        {
                error("\nERROR! Connection error.");
        }

        printf("\nEnter the message to be send :--  ");
```

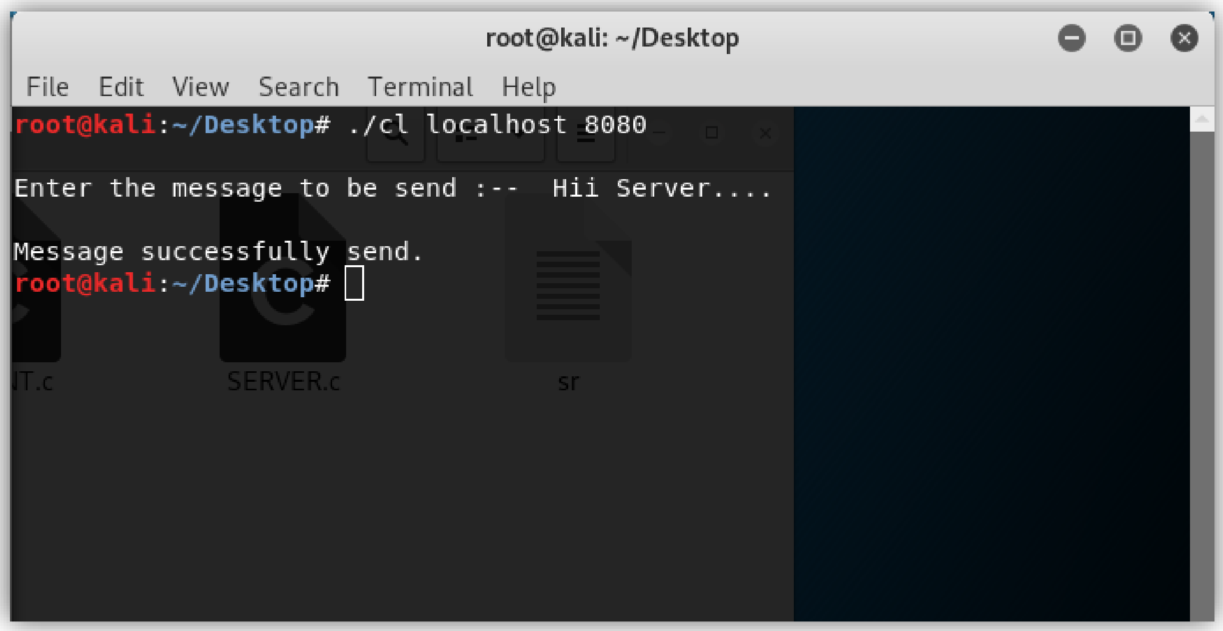
```
	bzero(buffer,256);
	fgets(buffer,255,stdin);
	n=write(sockfd,buffer,strlen(buffer));
	if(n<0)
	{
		error("\nERROR! Writing to socket.");
	}

	printf("\nMessage successfully send.\n");
	return 0;
}
```

Snapshots



```
root@kali:~/Desktop# ./sr 8080

Message Received :--  Hii Server....
root@kali:~/Desktop# 
```

Fig 1. Server Connectivity



```
root@kali:~/Desktop# ./cl localhost 8080

Enter the message to be send :--  Hii Server....

Message successfully send.
root@kali:~/Desktop# 
```

Fig. 2 Client Connection

System Requirements

1. Hardware Requirements:

Processor :	Intel i3 2.0 GHz and above
RAM :	512 MB and above
Hard disk :	80GB and above
Monitor :	CRT or LCD monitor
Keyboard :	Normal or Multimedia
Mouse :	Compatible mouse

2. Software Requirements:

Language :	C
Operation System :	Linux OS like Ubuntu

REFERENCES

1. https://en.wikipedia.org/wiki/Network_socket

2.. http://pubs.opengroup.org

3. “The Definitive Guide to Linux Network Programming” Book by “Kathryn Davis”.

4. “TCP/IP Sockets in C: Practical Guide for Programmers” Book by “Michael J. Donahoo”