International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 8, October - 2012

Sockets and Socket Address Structure

Mohit Mittal[1]
[1]*Assistant Professor,*
Anand College of Engineering & Management, Kapurthala.

Tarun Bhalla[2]
[2]*Assistant Professor,*
Anand College of Engineering & Management, Kapurthala.

Abstract:-*In this paper specifies the concept of socket and socket address structures. In this, we have discussed how communication has been performed between two hosts and discussed the role of the sockets. Sockets address of IPv4 and IPv6 is defined. The socket function call is also included which consists of various main functions like socket, connect, listen, bind, accept and close. It consists of two types of servers which uses socket function.*

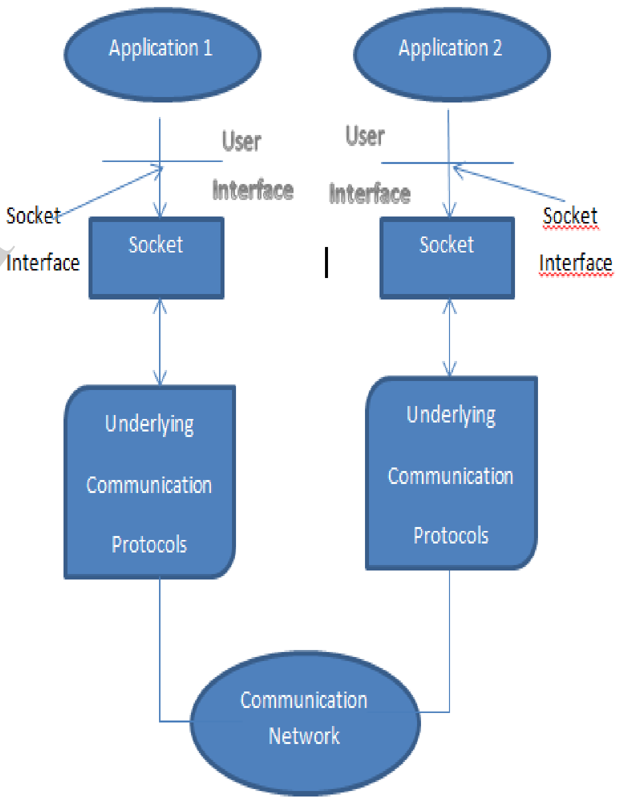
Keywords: Introduction, socket and its address, socket function calls, TCP socket call, and servers.

I. Introduction: Java's socket model is derived from BSD (UNIX) sockets, introduced in the early 1980s for inter-process communication using IP, the Internet Protocol. The Internet Protocol breaks all communications into *packets*, finite-sized chunks of data which are separately and individually routed from source to destination. IP allows routers, bridges, etc. to drop packets--there is no delivery guarantee. Packet size is limited by the IP protocol to 65535 bytes. Of this, a minimum of 20 bytes is needed for the IP packet header, so there is a maximum of 65515 bytes available for user data in each packet.

Sockets are a means of using IP to communicate between machines, so sockets are one major feature that allows Java to interoperate with legacy systems by simply talking to existing servers using their pre-defined protocol. [1]

2. API: The application interface is the interface available to the programmer for using the communication protocols. The API is depends to the OS the programming language.
We discuss the socket API. With sockets, the network connection can be used as a file. Network I/O is, however, more complicated than file I/O because:

- Asymmetric. The connection requires the program to know which process it is, the client or the server.
- A network connection that is connection-oriented is somewhat like opening a file. A connectionless protocol doesn't have anything like an open.
- A network application needs additional information to maintain protections, for example, of the other process.
- There are more parameters required to specify network connection than the file Input/Output. The parameters have different formats for different protocols.
- The network interface must support different protocols. These protocols may use different-size variable for addresses and other fields. [3]



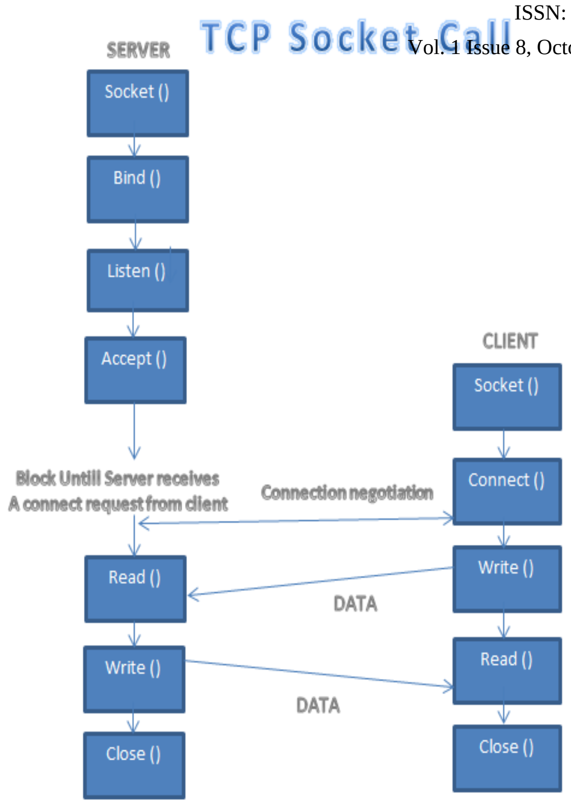
"Figure 1: Socket Interface." [2]

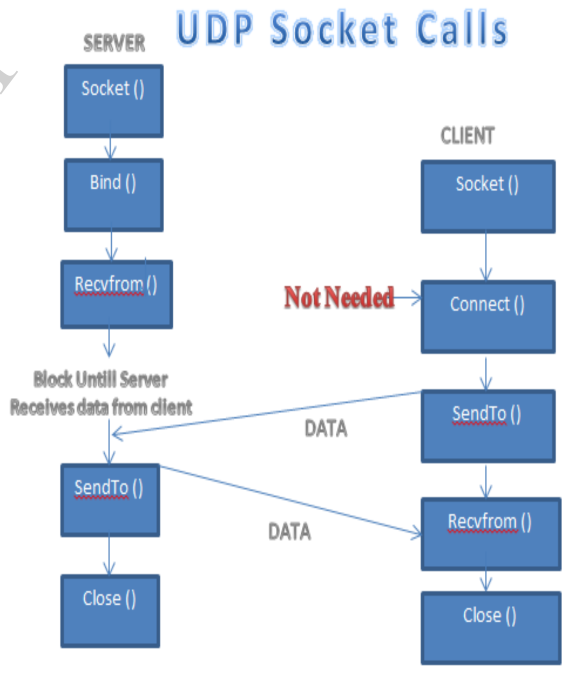
3. Socket Address Structure:A Character Recognition deal with the problem of reading offline handwritten character i.e. at some point in time (in mins, sec, hrs.) after it has been written. However recognition of unconstrained handwritten text can be very difficult because characters cannot be reliably isolated especially when the text is cursive handwriting. [2]

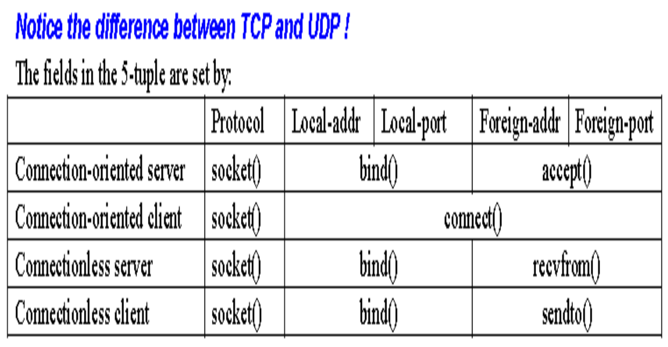
```
/* Generic Socket Address Structure, length=16*/
<sys/socket.h>
Structsocketaddr
 {
unit8_t              sa_len;
sa_family_tsa_family;          /*   address   family:
AF_XXX value */

char  sa_data[14]   /*up to 14 types of   protocol-
                     specific      address */
}; [3]

/* Ipv4 Socket Address Structure, length=16*/
<netinet/in.h>
structin_addr
{
in_addr_ts_addr;               /* 32-bit IPv4 address,
network   byte ordered */
};

structsockaddr_in
{
unit8_t        sin_len;            /* length of structure
(16 byte) */
sa_family_tsin_family;       /*AF_INET*/
in_port_tsin_port;            /* 16-bit TCP or UDP
port number, network byte ordered */

structin_addrsin_addr;     /*32-bit   Ipv4   address,
network byte  ordered */
charsin_zero[8];        /*  unused  –  initialize  to  all
zeroes */
};

/* Ipv6 Socket Address Structure, length=24*/
<netinet/in.h>
struct  in6_addr {
unit8_t             s6_addr[16];        /*  128-bit  Ipv6
address, network byte ordered */
};

#define SIN6_LEN           /* required for compile-
time tests */

struct sockaddr_in6
{
unit8_t     sin6_len;        /* length of this structure
(24byte) */
sa_family_t      sin6_family;        /*AF_INET6*/
in_port_t     sin6_port;              /* 16-bit TCP or
UDP port number, network byte ordered */
}; [3]
```

4. Procedure of Socket Programming

In order to communicate between two processes, the two processes must provide the formation used by ICP/IP (or UDP/IP) to exchange data. This information is the 5-tupe: {protocol, local-addr, local-process, foreign-addr and foreign-process}.
Several network systems calls are used to specify this information and use the socket. [3]






Notice the difference between TCP and UDP !

The fields in the 5-tuple are set by:

	Protocol	Local-addr	Local-port	Foreign-addr	Foreign-port
Connection-oriented server	socket()	bind()		accept()	
Connection-oriented client	socket()	connect()			
Connectionless server	socket()	bind()		recvfrom()	
Connectionless client	socket()	bind()		sendto()	

“Figure 2: Difference Between TCP and UDP” [3]

5. SOCKET FUNCTION CALLING

#include <sys/types.h>

#include <sys/socket.h>

Socket Function

int socket (int family, int type, int protocol);

Family: specifies the protocol family {AF_INET for TCP/IP}

Type: indicates communications semantics

SOCK_STREAM stream socket TCP

SOCK_DGRAM datagram socket UDP

SOCK_RAW raw socket

Protocol: set to 0 except for raw sockets

Returns on success: socket descriptor {a small nonnegative integer}

On error: -1

if ((sd= socket (AF_INET, SOCK_STREAM, 0)) < 0)

err_sys("socket call error");

Connect Function

intconnect(int**sockfd**,conststructsockaddr***servaddr**, socklen_t**addrlen**);

sockfd: a socket descriptor returned by the socket function.

***servaddr:** a pointer to a socket address structure

addrlen: the size of the socket address structure

The socket address structure must contain the ***IP address*** and the ***port number*** for the connection wanted. In TCP **connect** initiates a three-way handshake. **Connect** returns only when the connection is established or when an error occurs.

Returns on success: 0
on error: -1

Example:
if (connect (sd, (structsockaddr*) &servaddr, sizeof(servaddr)) != 0)

err_sys("connectcall error");

6. TCP SOCKET CALLS:

Bind Function

int**bind** (int**sockfd**, conststructsockaddr* **myaddr**, socklen_t**addrlen**);

Bind assigns a local protocol address to a socket.

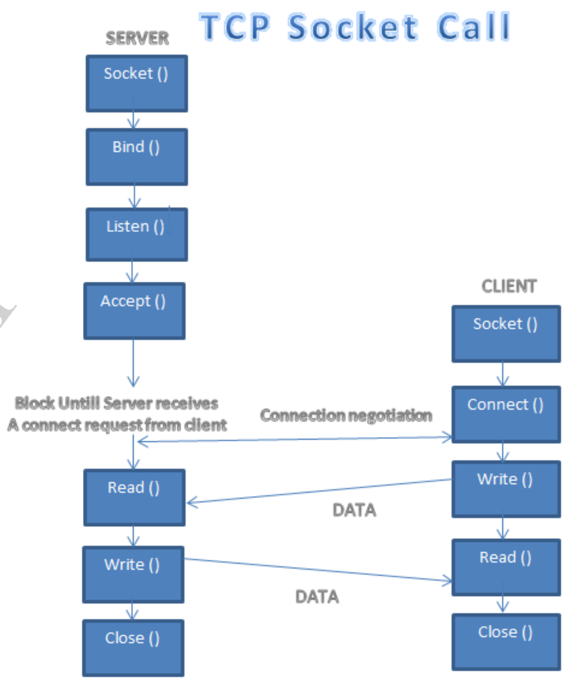
Protocol address: a 32 bit IPv4 address and a 16 bit TCP or UDP port number.

sockfd: a socket descriptor returned by the socket function.
***myaddr:** a pointer to a protocol-specific address.[2]
addrlen: the size of the socket address structure.

Servers bind their "well-known port" when they start.

Returns on success: 0
On error : -1

Example:
If (bind (sd, (structsockaddr *) &servaddr ,sizeof (servaddr)) != 0)
errsys("bind call error");



"Figure 3: TCP socket calls" [2]

Listen Function

int**listen** (int**sockfd**, int**backlog**);[2]

Listen is called only by a TCP server and performs two actions:

1. Converts an unconnected socket (sockfd) into a passive socket.
2. Specifies the maximum number of connections (backlog) that the kernel should queue for this socket.

Listen is normally called before the accept function.

Returns on success: 0
on error: -1
Example:

if (listen (sd, 2) != 0)

errsys(“listen call error”);

Accept Function

int**accept** (intsockfd , structsockaddr***cliaddr**, socklen_t***addrlen**);

Accept is called by the TCP server to return the next completed connection from the front of the completed connection queue.

sockfd: This is the same socket descriptor as in listencall.

***cliaddr:** used to return the protocol address of the connected peer process (i.e., the client process).[2]

***addrlen:** {this is a value-result argument}

Before the accept call:We set the integer value pointed to by *addrlento the size of the socket address structure pointed to by *cliaddr;

on return from the accept call: This integer value contains the actual number of bytes stored in the socket address structure.

Returns on success: a new socket descriptor
on error : -1

For **accept** the first argument **sockfd**is the listening socketand the returned value is the connected socket.

The server will have one connected socket for each client connection accepted.
When the server is finished with a client, the connected socket must be closed.

Example:

sfd= accept (sd, NULL, NULL);

if (sfd== -1) err_sys (“accept error”);[2]

Close Function

int**close** (int**sockfd**);

Close marks the socket as closed and returns to the process immediately.

sockfd: This socket descriptor is no longer useable.

***Note* –** TCP will try to send any data already queued to the other end before the normal connection termination sequence.

Returns on success: 0
on error : -1

Example:
close (sd); [2]

7. TWO TYPES OF SERVER

Concurrent server – forks a new process, so multiple clients can be handled at the same time.
Iterative server – the server processes one request before accepting the next.

Concurrent Server
```
listenfd = socket(…);
bind(listenfd,…);
listen(listenfd,…)
for ( ; ; ) {
connfd = accept(listenfd, …);
If (( pid = fork()) == 0) { /* child*/
close(listenfd);
/* process the request */
close(connfd);
exit(0);
}
close(connfd); /* parent*/
}[3]
```

Iterative Server
```
listenfd = socket(…);
bind(listenfd,…);
listen(listenfd,…)
for ( ; ; ) {
connfd = accept(listenfd, …);
/* process the request */
close(connfd);
}[3]
```

Client
```
sockfd = socket(…);
connect(sockfd, …)
/* process the request */
close(sockfd);[3]
```

CONCLUSION

In this paper ,we have concluded that the socket interface generally holds the communication between the user and the kernel. Also the two different applications can be interfaced through the communication network. In TCP socket calls, the client sends the request to the server and the server performs all the functions i.e. socket(), bind(), listen() and accept(). In concurrent servers, multiple clients can be handledat the same time, whereas in iterative server, the server processes only one request before accepting the next one. The Internet Protocol breaks all communications into packets, finite-sized chunks of data which are separately and individually routed from source to destination.

REFERENCES

[1] Elementary TCP Sockets UNIX Network Programming Vol. 1, Second Ed. Stevens Chapter 4

http://www.cs.usfca.edu/~parrt/doc/java/Sockets-notes.pdf

[2] UNIX Network Programming by W. Richard Stevens, Prentice Hall,

Englewood Cliffs, NJ, 1997.

http://web.cs.wpi.edu/~rek/Undergrad_Nets/B06/TCP_Sockets.pdf

[3] Unix Network Programming, W.R. Stevens, 1990,Prentice-Hall, Chapter 6.

[4] Unix Network Programming, W.R. Stevens, 1998,Prentice-Hall, Volume 1, Chapter 3-4.http://www.ece.eng.wayne.edu/~gchen/ece5650/lecture7.pdf

Mohit Mittal received his B.Tech and M.Tech degree in Computer Science from Guru Nanak Dev University, in 2011. He is working as Assistant Professor in Anand college of Engineering and Management, Kapurthala. His research areas include image processing, computer networks and Network Security.

Tarun Bhalla received his B.Tech degree in Computer Science from Punjab Technical University. He is currently working as a Assistant Professor in Anand College of Engineering and Management, Kapurthala. His research interest area includes Database, Network Security, Mobile Computing and adhoc network .