**Aim :** To Study the Advance socket System Calls.

**Socket**

A **socket** is one endpoint of a **two way** communication link between two programs running on the network. The socket mechanism provides a means of inter-process communication (IPC) by establishing named contact points between which the communication take place.

A socket connecting to the network is created at each end of the communication. Each socket has a specific address. This address is composed of an IP address and a port number.

Socket are generally employed in client server applications. The server creates a socket, attaches it to a network port addresses then waits for the client to contact it. The client creates a socket and then attempts to connect to the server socket. When the connection is established, transfer of data takes place.

**Types of Socket**

1. **Stream Sockets:-**

Delivery in a networked environment is guaranteed. If you send through the stream socket three items "A, B, C", they will arrive in the same order − "A, B, C". These sockets use TCP (Transmission Control Protocol) for data transmission. If delivery is impossible, the sender receives an error indicator. Data records do not have any boundaries.

1. **Datagram Sockets:-**

Delivery in a networked environment is not guaranteed. They're connectionless because you don't need to have an open connection as in Stream Sockets − you build a packet with the destination information and send it out. They use UDP (User Datagram Protocol).

**Advance Socket System Calls**

1. **getsockopt() - get the socket options :**

The *getsockopt*() function manipulates options associated with a socket.The *getsockopt*() function shall retrieve the value for the option specified by the *option\_name* argument for the socket specified by the *socket* argument.

If the size of the option value is greater than *option\_len*, the value stored in the object pointed to by the *option\_value* argument shall be silently truncated. Otherwise, the object pointed to by the *option\_len* argument shall be modified to indicate the actual length of the value.

***Syntax:-***

#include <sys/types.h>

#include <sys/socket.h>

int getsockopt(int s, int level, int optname, void \*optval, int \*optlen);

***Example:-***

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <stdio.h>

main()

{

int optlen, gs, socktype, s;

/\* Create a datagram socket. \*/

s = socket(AF\_INET, SOCK\_DGRAM, 0);

if (s == -1) {

perror("Socket not created");

return EXIT\_FAILURE;

}

/\* Ask for the socket type. \*/

optlen = sizeof(socktype);

gs = getsockopt (s, SOL\_SOCKET, SO\_TYPE, &socktype, &optlen);

if (gs == -1) {

perror("getsockopt failed");

return EXIT\_FAILURE;

}

/\* Print socket type. \*/

switch (socktype)

{

case SOCK\_STREAM:

puts("Stream socket.\n");

break;

case SOCK\_DGRAM:

puts("Datagram socket.\n");

break;

case SOCK\_RAW:

puts("Raw socket.\n");

break;

default:

puts("Unknown socket type.\n");

break;

}

return EXIT\_SUCCESS;

}

1. **setsockopt() - set the socket options :**

The *setsockopt*() function shall set the option specified by the *option\_name* argument, at the protocol level specified by the *level* argument, to the value pointed to by the *option\_value* argument for the socket associated with the file descriptor specified by the *socket* argument.

The *setsockopt*() function provides an application program with the means to control socket behavior. An application program can use *setsockopt*() to allocate buffer space, control timeouts, or permit socket data broadcasts.  Upon successful completion, *setsockopt*() shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

***Syntax:-***

#include <sys/types.h>

#include <sys/socket.h>

int setsockopt(int s, int level, int optname, const void \*optval, int optlen);

***Example:-***

#include<stdio.h>  
#include<stdlib.h>  
#include<errno.h>  
#include<string.h>  
#include<sys/types.h>  
#include<sys/socket.h>  
#include<netinet/in.h>  
#include<netinet/tcp.h>  
  
int main ()  
{  
        int  x1, x2, x3, x4  ;  
        if ( ( x1 = socket ( AF\_INET , SOCK\_STREAM , 0 ) ) < 0 )  
        {  
                [perror](https://www.opengroup.org/onlinepubs/009695399/functions/perror.html) ( " Checking the device : " ) ;  
                [exit](https://www.opengroup.org/onlinepubs/009695399/functions/exit.html) ( 0 ) ;  
        }  
        x4 = sizeof ( x2 ) ;  
        if ( getsockopt ( x1 , IPPROTO\_TCP , TCP\_MAXSEG , ( char\* ) &x2 , &x4 ) < 0 )  
        {  
              [perror](https://www.opengroup.org/onlinepubs/009695399/functions/perror.html) ( " Error occurred due to the function failure : " ) ;  
                [exit](https://www.opengroup.org/onlinepubs/009695399/functions/exit.html) ( 0 ) ;  
        }  
        [printf](https://www.opengroup.org/onlinepubs/009695399/functions/printf.html) ( " \n The probablistic value of x2 is : = %d " , x2 ) ;  
        x3 = 12324 ;  
        if ( setsockopt ( x1 , SOL\_SOCKET , SO\_SNDBUF , ( char\* ) &x3 , sizeof ( x3 ) ) < 0 )  
        {  
                [perror](https://www.opengroup.org/onlinepubs/009695399/functions/perror.html) ( " The chances of failure to respond " ) ;  
                [exit](https://www.opengroup.org/onlinepubs/009695399/functions/exit.html) ( 0 ) ;  
        }  
        x4 = sizeof ( x3 ) ;  
        if ( getsockopt ( x1 , SOL\_SOCKET , SO\_SNDBUF , ( char\* ) &x3 , &x4 ) < 0 )  
        {  
                [perror](https://www.opengroup.org/onlinepubs/009695399/functions/perror.html) ( " Function does not respond properly : " ) ;  
                [exit](https://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(0);  
        }  
        [printf](https://www.opengroup.org/onlinepubs/009695399/functions/printf.html) ( " \nThe  buffer value is = %d \n " , x3 ) ;  
        return 0 ;  
}

1. **getsockname() - get the socket name:-**

The *getsockname*() function shall retrieve the locally-bound name of the specified socket, store this address in the **sockaddr** structure pointed to by the *address* argument, and store the length of this address in the object pointed to by the *address\_len* argument.

If the actual length of the address is greater than the length of the supplied **sockaddr** structure, the stored address shall be truncated. If the socket has not been bound to a local name, the value stored in the object pointed to by *address* is unspecified.

***Syntax:-***

#include <sys/types.h>

#include <sys/socket.h>

int getsockname(int s, void \*addr, int \*addrlen);

***Example:-***

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

f()

{

int s;

struct sockaddr\_in sa;

int sa\_len;

.

.

/\* We must put the length in a variable. \*/

sa\_len = sizeof(sa);

/\* Ask getsockname to fill in this socket's local \*/

/\* address. \*/

if (getsockname(s, &sa, &sa\_len) == -1) {

perror("getsockname() failed");

return -1;

}

/\* Print it. The IP address is often zero beacuase \*/

/\* sockets are seldom bound to a specific local \*/

/\* interface. \*/

printf("Local IP address is: %s\n", inet\_ntoa(sa.sin\_add r));

printf("Local port is: %d\n", (int) ntohs(sa.sin\_port));

.

.

}

1. **getpeername() - get the name of the peer socket :**

The *getpeername*() function shall retrieve the peer address of the specified socket, store this address in the **sockaddr** structure pointed to by the *address* argument, and store the length of this address in the object pointed to by the *address\_len* argument.

If the actual length of the address is greater than the length of the supplied **sockaddr** structure, the stored address shall be truncated. If the protocol permits connections by unbound clients, and the peer is not bound, then the value stored in the object pointed to by *address* is unspecified.

Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error.

***Syntax :-***

#include <sys/types.h>

#include <sys/socket.h>

int getpeername(int s, void \*addr, int \*addrlen);

***Example :-***

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdio.h>

f()

{

int s;

struct sockaddr\_in peer;

int peer\_len;

.

.

.

/\* We must put the length in a variable. \*/

peer\_len = sizeof(peer);

/\* Ask getpeername to fill in peer's socket address. \*/

if (getpeername(s, &peer, &peer\_len) == -1) {

perror("getpeername() failed");

return -1;

}

/\* Print it. The IP address is often zero because \*/

/\* sockets are seldom bound to a specific local \*/

/\* interface. \*/

printf("Peer's IP address is: %s\n", inet\_ntoa(peer.sin\_addr));

printf("Peer's port is: %d\n", (int) ntohs(peer.sin\_port));

.

.

.

}

1. **readv()-Reads Data from a Socket Descriptor into an Array of Buffers :**

**readv** reads data from socket or file descriptor **s** into the **iovcnt** buffers specified by the **iov** array. As with the **read** call, the socket must have been previously associated with a remote address via the **connect** system call. If there are no data, **readv** blocks the caller unless the socket is in non-blocking mode.

The **iovec** structure is defined in **<sys/uio.h>** . Each **iovec** entry specifies the base address and length of an area in memory in which the data should be placed. **readv** completely fills one area before proceeding to the next area.

Although **readv** is primarily used with sockets, it can also be used to read any file that can be accessed by the **read** function. 

If **readv** succeeds, it returns the number of bytes read into the buffer. If **readv** returns a **0** , the end-of-file has been reached. If **readv** fails, it returns a **-1** . It is common for **readv** to return a value less than the total number of bytes in the buffers. This is not an error.

***Syntax :-***

#include <sys/types.h>

#include <sys/uio.h>

#include <fcntl.h>

int readv(int s, struct iovec \*iov, int iovcnt);

***Example:-***

#include <sys/types.h>

#include <sys/uio.h>

#include <unistd.h>

...

ssize\_t bytes\_read;

int fd;

char buf0[20];

char buf1[30];

char buf2[40];

int iovcnt;

struct iovec iov[3];

iov[0].iov\_base = buf0;

iov[0].iov\_len = sizeof(buf0);

iov[1].iov\_base = buf1;

iov[1].iov\_len = sizeof(buf1);

iov[2].iov\_base = buf2;

iov[2].iov\_len = sizeof(buf2);

...

iovcnt = sizeof(iov) / sizeof(struct iovec);

bytes\_read = readv(fd, iov, iovcnt);

...

1. **Writev()-****Writes Data from an Array of Buffers to a Socket :**

**writev** writes data to connected socket descriptor **s** from the **iovcnt** buffers specified by the **iov** array. As with the **write** call, the socket must have been previously associated with a remote address via the **connect** system call. If there are no data, **writev** blocks the caller unless the socket is in non-blocking mode.

The **iovec** structure is defined in the **<sys/uio.h>** header file. Each **iovec** entry specifies the base address and length of an area in memory from which the data should be taken. **writev** completely sends one area before proceeding to the next area.**writev** is an atomic operation. For datagram sockets, each **writev** call causes one datagram to be sent.

Although **writev** is primarily used with sockets, it can also be used to write any file that can be accessed by the **write** function. If **writev** succeeds, it returns the number of bytes written. If **writev** returns a **0** , the end-of-file has been reached. If **writev** fails, it returns a **-1** .

***Syntax:-***

#include <sys/types.h>

#include <sys/uio.h>

#include <fcntl.h>

int writev(int s, struct iovec \*iov, int iovcnt);

***Example :-***

#include <sys/types.h>

#include <sys/uio.h>

#include <unistd.h>

...

ssize\_t bytes\_written;

int fd;

char \*buf0 = "short string\n";

char \*buf1 = "This is a longer string\n";

char \*buf2 = "This is the longest string in this example\n";

int iovcnt;

struct iovec iov[3];

iov[0].iov\_base = buf0;

iov[0].iov\_len = strlen(buf0);

iov[1].iov\_base = buf1;

iov[1].iov\_len = strlen(buf1);

iov[2].iov\_base = buf2;

iov[2].iov\_len = strlen(buf2);

...

iovcnt = sizeof(iov) / sizeof(struct iovec);

bytes\_written = writev(fd, iov, iovcnt);

...