Pranit Tandon

RA1911028010014 (CSE – H2)

## 

**CN LAB Exercise- 2**

## STUDY OF BASIC FUNCTIONS OF SOCKET PROGRAMMING

### AIM:

To discuss some of the basic functions used for socket programming.

## man socket

### NAME:

Socket – create an endpoint for communication.

### DESCRIPTION:

* + Socket creates an endpoint for communication and returns a descriptor.
  + The domain parameter specifies a common domain this selects the protocol family which will be used for communication.
  + These families are defined in <sys/socket.h>.

### FORMAT:

|  |  |
| --- | --- |
| **NAME** | **PURPOSE** |
| PF\_UNIX, PF\_LOCAL | Local Communication. |
| PF\_INET | IPV4 Internet Protocols. |
| PF\_IPX | IPX-Novell Protocols. |
| PF\_APPLETALK | Apple Talk. |

* + The socket has the indicated type, which specifies the communication semantics.

### TYPES:

1. **SOCK\_STREAM:**
   * Provides sequenced, reliable, two-way, connection based byte streams.
   * An out-of-band data transmission mechanism, may be supported.

### SOCK\_DGRAM:

* + Supports datagram (connectionless, unreliable messages of a fixed maximum length).

### SOCK\_SEQPACKET:

* + Provides a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length.

### SOCK\_RAW:

* + Provides raw network protocol access.

### SOCK\_RDM:

* + Provides a reliable datagram layer that doesn’t guarantee ordering.

### SOCK\_PACKET:

* + Obsolete and shouldn’t be used in new programs.

## man connect

### NAME:

connect – initiate a connection on a socket.

### DESCRIPTION:

* + The file descriptor sockfd must refer to a socket.
  + If the socket is of type SOCK\_DGRAM then the serv\_addr address is the address to which datagrams are sent by default and the only addr from which datagrams are received.
  + If the socket is of type SOCK\_STREAM or SOCK\_SEQPACKET, this call attempts to make a connection to another socket.

### RETURN VALUE:

* + If the connection or binding succeeds, zero is returned.
  + On error, -1 is returned, and error number is set appropriately.

**ERRORS:**

|  |  |
| --- | --- |
| EBADF | Not a valid Index. |
| EFAULT | The socket structure address is outside the user’s address space. |
| ENOTSOCK | Not associated with a socket. |
| EISCONN | Socket is already connected. |
| ECONNREFUSED | No one listening on the remote address. |

## man accept

### NAME:

accept/reject job is sent to a destination.

### SYNOPSIS:

accept destination(s)

reject[-t] [-h server] [-r reason] destination(s)

### DESCRIPTION:

* accept instructs the printing system to accept print jobs to the specified destination.
* The –r option sets the reason for rejecting print jobs.
* The –e option forces encryption when connecting to the server.

## man send

### NAME:

send, sendto, sendmsg - send a message from a socket.

### SYNOPSIS:

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

ssize\_t send(int s, const void \*buf, size\_t len, int flags);

ssize\_t sendto(int s, const void \*buf, size\_t len, int flags, const struct sock\_addr\*to, socklen\_t tolen); ssize\_t sendmsg(int s, const struct msghdr \*msg, int flags);

### DESCRIPTION:

* The system calls send, sendto and sendmsg are used to transmit a message to another socket.
* The send call may be used only when the socket is in a connected state.
* The only difference between send and write is the presence of flags.
* The parameter is the file descriptor of the sending socket.

## man recv

### NAME:

recv, recvfrom, recvmsg – receive a message from a socket.

### SYNOPSIS:

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

ssize\_t recv(int s, void \*buf, size\_t len, int flags);

ssize\_t recvfrom(int s, void \*buf, size\_t len, int flags, struct sockaddr \*from, socklen\_t\* from len); ssize\_t recvmsg(int s, struct msghdr \*msg, int flags);

### DESCRIPTION:

* The recvfrom and recvmsg calls are used to receive messages from a socket, and may be used to recv data on a socket whether or not it is connection oriented.
* If from is not NULL, and the underlying protocol provides the src addr , this src addr is filled in.
* The recv call is normally used only on a connection socket and is identical to recvfrom with a NULL from parameter.

## man read

### NAME:

read, readonly, return

## man write

### NAME:

write- send a message to another user.

### SYNOPSIS:

write user[ttyname]

### DESCRIPTION:

* + write allows you to communicate with other users, by copying lines from terminal to ………
  + When you run the write and the user you are writing to get a message of the form: Message from yourname @yourhost on yourtty at hh:mm:…
  + Any further lines you enter will be copied to the specified user’s terminal.
  + If the other user wants to reply they must run write as well.

## ifconfig

### NAME:

ifconfig- configure a network interface.

### SYNOPSIS:

ifconfig[interface]

ifconfig interface[aftype] options | address……

### DESCRIPTION:

* ifconfig is used to configure the kernel resident network interfaces.
* It is used at boot time to setup interfaces as necessary.
* After that, it is usually only needed when debugging or when system tuning is needed.
* If no arguments are given, ifconfig displays the status of the currently active interfaces

## man bind

### SYNOPSIS:

bind[-m keymap] [-lp sv psv]

## man htons/ man htonl

### NAME:

htonl, htons, ntohl, ntohs - convert values between host and network byte order.

### SYNOPSIS:

#include<netinet/in.h>

uint32\_t htonl(uint32\_t hostlong); uint16\_t htons(uint32\_t hostshort);

uint32\_t ntohl(uint32\_t netlong); uint16\_t ntohs(uint16\_t netshort);

### DESCRIPTION:

* The htonl() function converts the unsigned integer hostlong from host byte order to network byte order.
* The htons() converts the unsigned short integer hostshort from host byte order to network byte order.
* The ntohl() converts the unsigned integer netlong from network byte order to host byte order.

## man gethostname

### NAME:

gethostname, sethostname- get/set host name.

### SYNOPSIS:

#include<unistd.h>

int gethostname(char \*name,size\_t len); int sethostname(const char \*name,siz

### RESULT:

Thus the basic functions used for Socket Programming was studied successfully.

## listen function

The *listen* function is called only by a TCP server and it performs two actions −

* The listen function converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket.
* The second argument to this function specifies the maximum number of connections the kernel should queue for this socket.

# Parameters:

* **sockfd** − It is a socket descriptor returned by the socket function.
* **backlog** − It is the number of allowed connections.

1. **sendto Function**

The *sendto* function is used to send data over UNCONNECTED datagram sockets. Its signature is as follows −

int sendto(int sockfd, const void \*msg, int len, unsigned int flags, const struct sockaddr \*to, int tolen);

This call returns the number of bytes sent, otherwise it returns -1 on error.

# Parameters

* **sockfd** − It is a socket descriptor returned by the socket function.
* **msg** − It is a pointer to the data you want to send.
* **len** − It is the length of the data you want to send (in bytes).
* **flags** − It is set to 0.
* **to** − It is a pointer to struct sockaddr for the host where data has to be sent.
* **tolen** − It is set it to sizeof(struct sockaddr).

1. **recvfrom Function**

The *recvfrom* function is used to receive data from UNCONNECTED datagram sockets.

int recvfrom(int sockfd, void \*buf, int len, unsigned int flags struct sockaddr \*from, int \*fromlen);

This call returns the number of bytes read into the buffer, otherwise it returns -1 on error.

Parameters

* **sockfd** − It is a socket descriptor returned by the socket function.
* **buf** − It is the buffer to read the information into.
* **len** − It is the maximum length of the buffer.
* **flags** − It is set to 0.
* **from** − It is a pointer to struct sockaddr for the host where data has to be read.
* **fromlen** − It is set it to sizeof(struct sockaddr).

## close Function

The *close* function is used to close the communication between the client and the server. Its syntax is as follows −

int close( int sockfd );

This call returns 0 on success, otherwise it returns -1 on error.

# Parameters

* **sockfd** − It is a socket descriptor returned by the socket function.

1. **shutdown Function**

The *shutdown* function is used to gracefully close the communication between the client and the server. This function gives more control in comparison to the *close* function. Given below is the syntax of *shutdown* −

int shutdown(int sockfd, int how);

This call returns 0 on success, otherwise it returns -1 on error.

# Parameters

* **sockfd** − It is a socket descriptor returned by the socket function.
* **how** − Put one of the numbers −
  + **0** − indicates that receiving is not allowed,
  + **1** − indicates that sending is not allowed, and
  + **2** − indicates that both sending and receiving are not allowed. When *how* is set to 2, it's the same thing as close().

1. **select Function**

The *select* function indicates which of the specified file descriptors is ready for reading, ready for writing, or has an error condition pending.

When an application calls *recv or recvfrom*, it is blocked until data arrives for that socket. An application could be doing other useful processing while the incoming data stream is empty. Another situation is when an application receives data from multiple sockets.

Calling *recv or recvfrom* on a socket that has no data in its input queue prevents immediate reception of data from other sockets. The select function call solves this problem by allowing the program to poll all the socket handles to see if they are available for non-blocking reading and writing operations.

Given below is the syntax of *select* −

int select(int nfds, fd\_set \*readfds, fd\_set \*writefds, fd\_set

\*errorfds, struct timeval \*timeout);

This call returns 0 on success, otherwise it returns -1 on error.

# Parameters

* **nfds** − It specifies the range of file descriptors to be tested. The select() function tests file descriptors in the range of 0 to nfds-1
* **readfds** − It points to an object of type *fd\_set* that on input, specifies the file descriptors to be checked for being ready to read, and on output, indicates which file descriptors are ready to read. It can be NULL to indicate an empty set.
* **writefds** − It points to an object of type *fd\_set* that on input, specifies the file descriptors to be checked for being ready to write, and on output, indicates which file descriptors are ready to write. It can be NULL to indicate an empty set.
* **exceptfds** − It points to an object of type *fd\_set* that on input, specifies the file descriptors to be checked for error conditions pending, and on output indicates, which file descriptors have error conditions pending. It can be NULL to indicate an empty set.
* **timeout** − It points to a timeval struct that specifies how long the select call should poll the descriptors for an available I/O operation. If the timeout value is 0, then select will return immediately.