COMPUTER NETWORKS

EXP 2

Basic Functions Used for Socket Programming

JULY 30, 2021

ROEHIT RANGANATHAN
RA1911033010017 | L2

Aim:

To Discuss Some OF The Basic Functions Used For Socket Programming

Description:

1. Man socket

```
SOCKET(2)            Linux Programmer's Manual            SOCKET(2)

NAME
       socket - create an endpoint for communication

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

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

DESCRIPTION
       socket() creates an endpoint for communication and returns a
       file descriptor that refers to that endpoint. The file
       descriptor returned by a successful call will be the lowest-
       numbered file descriptor not currently open for the process.

       The domain argument specifies a communication domain; this
       selects the protocol family which will be used for communi-
       cation. These families are defined in <sys/socket.h>. The
       currently understood formats include:

       Name                Purpose                          Man page
       AF_UNIX, AF_LOCAL   Local communication              unix(7)
       AF_INET             IPv4 Internet protocols          ip(7)
       AF_INET6            IPv6 Internet protocols          ipv6(7)
       AF_IPX              IPX - Novell protocols
       AF_NETLINK          Kernel user interface device     netlink(7
)
       AF_X25              ITU-T X.25 / ISO-8208 protocol   x25(7)
       AF_AX25             Amateur radio AX.25 protocol
       AF_ATMPVC           Access to raw ATM PVCs
       AF_APPLETALK        AppleTalk                        ddp(7)
       AF_PACKET           Low level packet interface       packet(7)
       AF_ALG              Interface to kernel crypto API
```

2. SOCK_STREAM

```
SOCK_STREAM     Provides sequenced, reliable, two-way, connection-based byte streams.  An out-of-band data transmission mechanism may be supported.
```

3. SOCK_DGRAM

```
       SOCK_DGRAM      Supports datagrams (connectionless, unreliable
 messages of a fixed maximum length).
```

4. SOCK_SEQPACKET

```
       SOCK_SEQPACKET  Provides  a  sequenced,  reliable,  two-way  c
onnection-based  data transmission path for datagrams of fixed maximu
m length; a consumer is required to read an
                       entire packet with each input system call.
```

5. SOCK_RAW

```
  SOCK_RAW        Provides raw network protocol access.
```

6. SOCK_RDM

```
       SOCK_RDM        Provides a reliable datagram layer that does n
ot guarantee ordering.
```

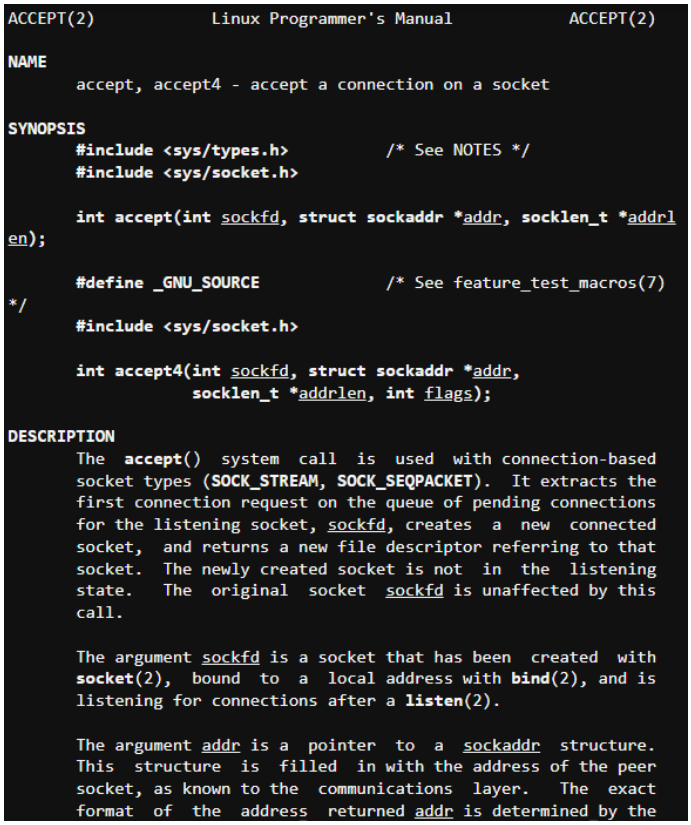
7. SOCK_PACKET

```
  SOCK_PACKET     Obsolete and should not be used in new programs; see packet(7).
```

8. man connect

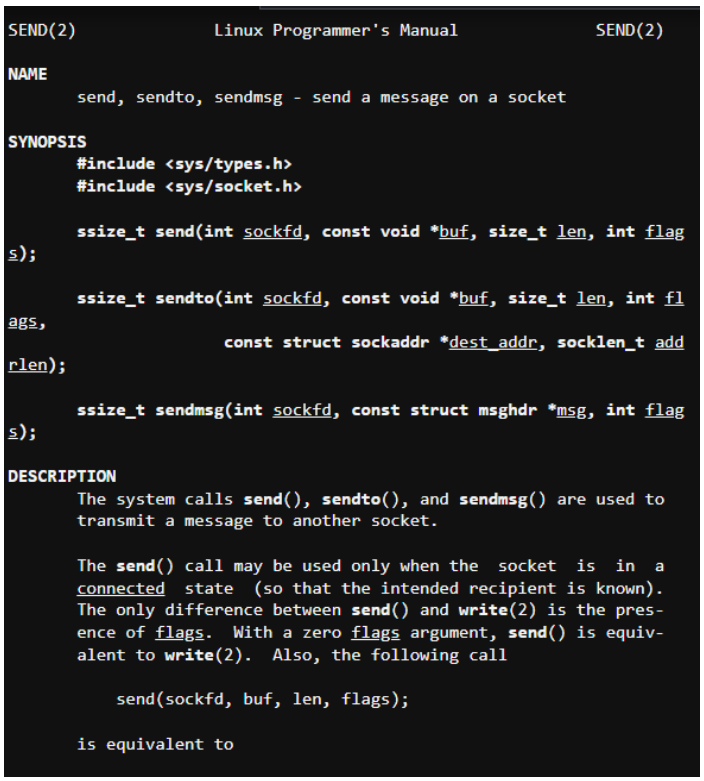
```
CONNECT(2)          Linux Programmer's Manual          CONNECT(2)

NAME
       connect - initiate a connection on a socket

SYNOPSIS
 Manual page connect(2) line 1 (press h for help or q to quit)
```

9. man accept



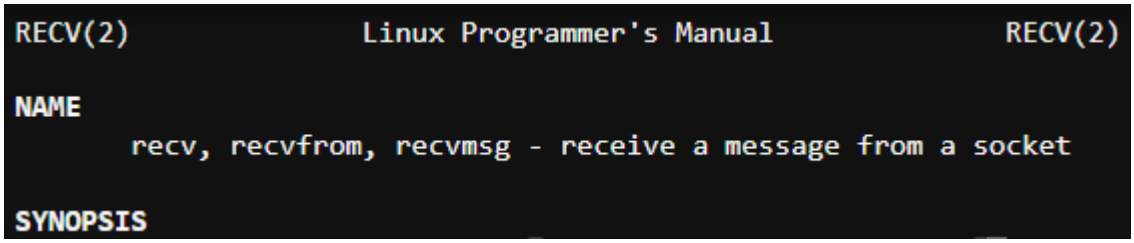
```
ACCEPT(2)            Linux Programmer's Manual            ACCEPT(2)

NAME
       accept, accept4 - accept a connection on a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int accept(int sockfd, struct sockaddr *addr, socklen_t *addrl
en);

       #define _GNU_SOURCE             /* See feature_test_macros(7)
*/
       #include <sys/socket.h>

       int accept4(int sockfd, struct sockaddr *addr,
                   socklen_t *addrlen, int flags);

DESCRIPTION
       The  accept()  system  call  is  used  with connection-based
       socket types (SOCK_STREAM, SOCK_SEQPACKET).  It extracts the
       first connection request on the queue of pending connections
       for the listening socket, sockfd, creates  a  new  connected
       socket,  and returns a new file descriptor referring to that
       socket.  The newly created socket is not  in  the  listening
       state.   The  original  socket  sockfd is unaffected by this
       call.

       The argument sockfd is a socket that has been  created  with
       socket(2),  bound  to  a  local address with bind(2), and is
       listening for connections after a listen(2).

       The argument addr is a  pointer  to  a  sockaddr  structure.
       This  structure  is  filled  in with the address of the peer
       socket, as known to the  communications  layer.   The  exact
       format  of  the  address  returned addr is determined by the
```

10. man send

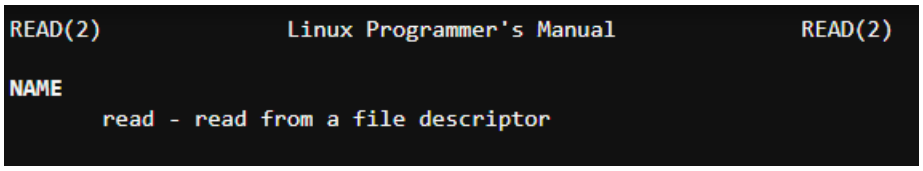


```
SEND(2)              Linux Programmer's Manual              SEND(2)

NAME
       send, sendto, sendmsg - send a message on a socket

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

       ssize_t send(int sockfd, const void *buf, size_t len, int flag
s);

       ssize_t sendto(int sockfd, const void *buf, size_t len, int fl
ags,
                      const struct sockaddr *dest_addr, socklen_t add
rlen);

       ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flag
s);

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  (so that the intended recipient is known).
       The only difference between send() and write(2) is the pres-
       ence of flags.  With a zero flags argument, send() is equiv-
       alent to write(2).  Also, the following call

           send(sockfd, buf, len, flags);

       is equivalent to
```

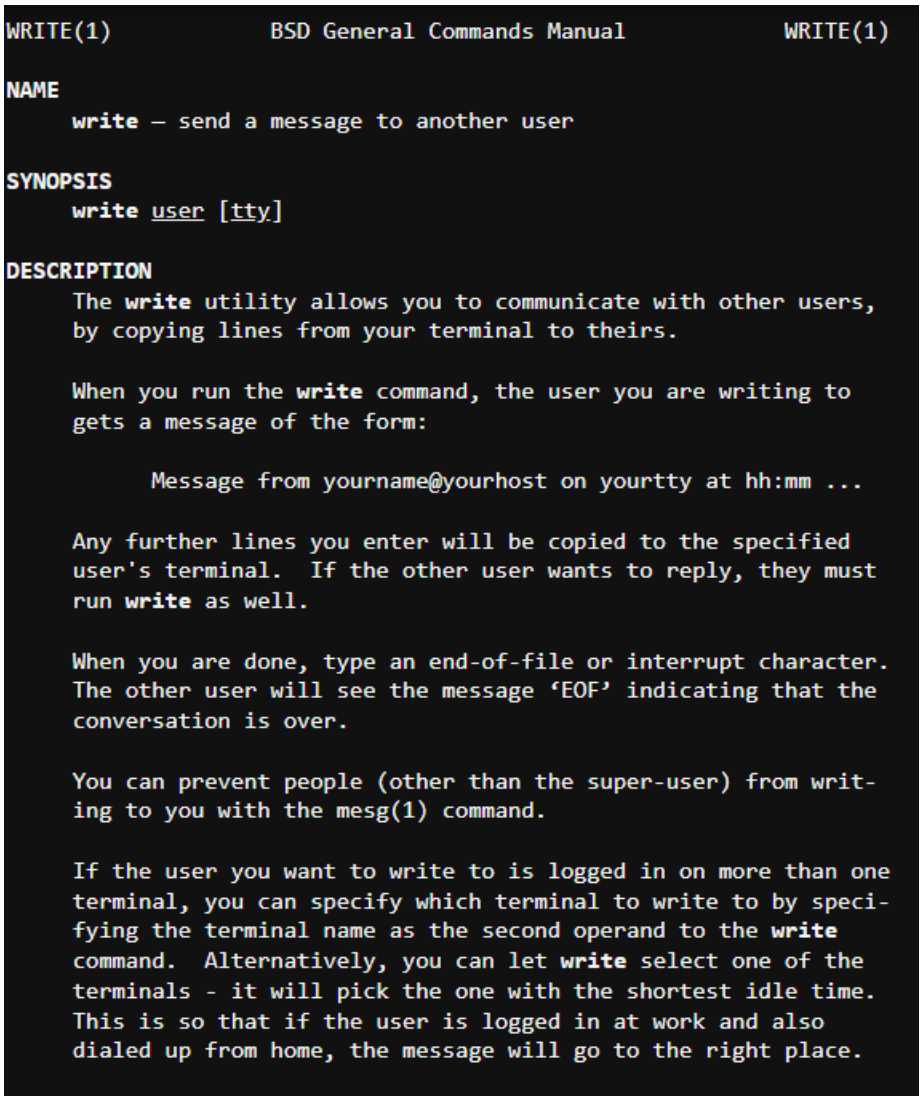
11. man recv



12. man read



13. man write



14. man bind

```
BIND(2)              Linux Programmer's Manual              BIND(2)

NAME
       bind - bind a name to a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int bind(int sockfd, const struct sockaddr *addr,
                socklen_t addrlen);

DESCRIPTION
       When a socket is created with socket(2), it exists in a name
       space (address family) but has no address assigned to it.
       bind() assigns the address specified by addr to the socket
       referred to by the file descriptor sockfd.  addrlen speci-
       fies the size, in bytes, of the address structure pointed to
       by addr.  Traditionally, this operation is called “assigning
       a name to a socket”.

       It is normally necessary to assign a local address using
       bind() before a SOCK_STREAM socket may receive connections
       (see accept(2)).

       The rules used in name binding vary between address fami-
       lies.  Consult the manual entries in Section 7 for detailed
       information.  For AF_INET, see ip(7); for AF_INET6, see
       ipv6(7); for AF_UNIX, see unix(7); for AF_APPLETALK, see
       ddp(7); for AF_PACKET, see packet(7); for AF_X25, see
       x25(7); and for AF_NETLINK, see netlink(7).

       The actual structure passed for the addr argument will
       depend on the address family.  The sockaddr structure is
       defined as something like:
```

15. ifconfig

```
Kkottilingam:~/environment $ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.25
5
        ether 02:42:ef:25:cc:24  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens5: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9001
        inet 172.31.9.200  netmask 255.255.240.0  broadcast 172.31.15
.255
        inet6 fe80::47b:6fff:fe64:1e5d  prefixlen 64  scopeid 0x20<li
nk>
        ether 06:7b:6f:64:1e:5d  txqueuelen 1000  (Ethernet)
        RX packets 332007  bytes 244600134 (244.6 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 195655  bytes 49685414 (49.6 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 666  bytes 95406 (95.4 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 666  bytes 95406 (95.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

16. man htons/man htonl

```
BYTEORDER(3)          Linux Programmer's Manual          BYTEORDER(3)

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

SYNOPSIS
       #include <arpa/inet.h>

       uint32_t htonl(uint32_t hostlong);

       uint16_t htons(uint16_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()  function  converts  the unsigned short integer
       hostshort from host byte order to network byte order.

       The ntohl() function converts the unsigned  integer  netlong
       from network byte order to host byte order.

       The  ntohs()  function  converts  the unsigned short integer
       netshort from network byte order to host byte order.

       On the i386 the host byte order is  Least  Significant  Byte
       first, whereas the network byte order, as used on the Inter-
       net, is Most Significant Byte first.

ATTRIBUTES
       For an explanation of the terms used in  this  section,  see
       attributes(7).
```

17. man gethostname

```
GETHOSTNAME(2)        Linux Programmer's Manual        GETHOSTNAME(2)

NAME
       gethostname, sethostname - get/set hostname

SYNOPSIS
       #include <unistd.h>

       int gethostname(char *name, size_t len);
       int sethostname(const char *name, size_t len);

   Feature  Test  Macro  Requirements  for  glibc  (see  feature_test_macros(7)):

       gethostname():
           Since glibc 2.12: _BSD_SOURCE || _XOPEN_SOURCE >= 500
           || /* Since glibc 2.12: */ _POSIX_C_SOURCE >= 200112L
       sethostname():
           Since glibc 2.21:
               _DEFAULT_SOURCE
           In glibc 2.19 and 2.20:
               _DEFAULT_SOURCE || (_XOPEN_SOURCE && _XOPEN_SOURCE < 5
00)
           Up to and including glibc 2.19:
               _BSD_SOURCE || (_XOPEN_SOURCE && _XOPEN_SOURCE < 500)

DESCRIPTION
       These system calls are used to access or to change the host-
       name of the current processor.

       sethostname()  sets  the  hostname to the value given in the
       character array name.  The len argument specifies the number
       of  bytes in name.  (Thus, name does not require a terminat-
       ing null byte.)
```

18. gethostbyname

```
GETHOSTBYNAME(3)     Linux Programmer's Manual     GETHOSTBYNAME(3)

NAME
       gethostbyname,  gethostbyaddr,  sethostent, gethostent, end-
       hostent, h_errno, herror, hstrerror, gethostbyaddr_r,  geth-
       ostbyname2,  gethostbyname2_r, gethostbyname_r, gethostent_r
       - get network host entry

SYNOPSIS
       #include <netdb.h>
       extern int h_errno;

       struct hostent *gethostbyname(const char *name);

       #include <sys/socket.h>       /* for AF_INET */
       struct hostent *gethostbyaddr(const void *addr,
                                     socklen_t len, int type);

       void sethostent(int stayopen);

       void endhostent(void);

       void herror(const char *s);

       const char *hstrerror(int err);

       /* System V/POSIX extension */
       struct hostent *gethostent(void);

       /* GNU extensions */
       struct hostent *gethostbyname2(const char *name, int af);

       int gethostent_r(
               struct hostent *ret, char *buf, size_t buflen,
               struct hostent **result, int *h_errnop);
```