Computer Networks Lab
Week-2
Jul 30, 2021

Venkata Naga Sai Ram Nomula
RA1911033010021
L2 - SWE

AIM:
To discuss some of the basic functions used for socket program

1.man socket
NAME: Socket – create an endpoint for communication.
SYNOPSIS:
#include<sys/types.h>
#include<sys/socket.h>
int socket(int domain,int type,int protocol);
DESCRIPTION:

```
DESCRIPTION
       socket()  creates  an  endpoint for communication and returns a file descriptor that refers to that endpoint.  The file descriptor returned by a suc-
       cessful 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  communication.  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
```

TYPES:

```
The socket has the indicated type, which specifies the communication semantics.  Currently defined types are:

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

SOCK_DGRAM      Supports datagrams (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; a consumer is
                required to read an entire packet with each input system call.

SOCK_RAW        Provides raw network protocol access.

SOCK_RDM        Provides a reliable datagram layer that does not guarantee ordering.

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

Some socket types may not be implemented by all protocol families.

Since Linux 2.6.27, the type argument serves a second purpose: in addition to specifying a socket type, it may include the bitwise OR of any  of  the
following values, to modify the behavior of socket():

SOCK_NONBLOCK   Set  the  O_NONBLOCK file status flag on the new open file description.  Using this flag saves extra calls to fcntl(2) to achieve the
                same result.

SOCK_CLOEXEC    Set the close-on-exec (FD_CLOEXEC) flag on the new file descriptor.  See the description of the O_CLOEXEC flag in open(2) for reasons
                why this may be useful.
```

2.SOCK_STREAM:

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

3.SOCK_DGRAM:

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

4.SOCK_SEQPACKET:

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

5.SOCK_RAW:

- Provides raw network protocol access.

6.SOCK_RDM:

- Provides a reliable datagram layer that doesn't guarantee ordering.

7.SOCK_PACKET:

- Obsolete and shouldn't be used in new programs.

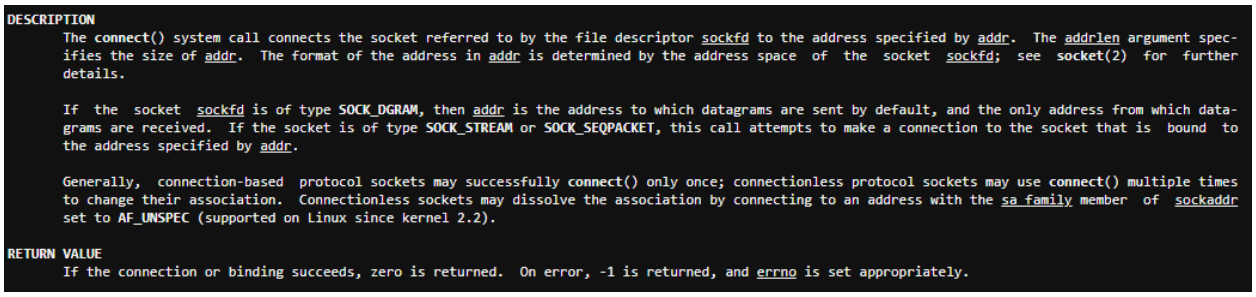
8.man connect

NAME: connect – initiate a connection on a socket.

SYNOPSIS:

#include<sys/types.h>
#include<sys/socket.h>
int connect(int sockfd,const (struct sockaddr*)serv_addr,socklen_t addrlen);

DESCRIPTION:



```
DESCRIPTION
       The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr.  The addrlen argument spec-
       ifies the size of addr.  The format of the address in addr is determined by the address space  of  the  socket  sockfd;  see  socket(2)  for  further
       details.

       If  the  socket  sockfd is of type SOCK_DGRAM, then addr is the address to which datagrams are sent by default, and the only address from which data-
       grams are received.  If the socket is of type SOCK_STREAM or SOCK_SEQPACKET, this call attempts to make a connection to the socket that is  bound  to
       the address specified by addr.

       Generally,  connection-based  protocol sockets may successfully connect() only once; connectionless protocol sockets may use connect() multiple times
       to change their association.  Connectionless sockets may dissolve the association by connecting to an address with the sa_family member  of  sockaddr
       set to AF_UNSPEC (supported on Linux since kernel 2.2).

RETURN VALUE
       If the connection or binding succeeds, zero is returned.  On error, -1 is returned, and errno is set appropriately.
```

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 is listening on the remote address.

9.man accept

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 *addrlen);
#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 socket's address family (see socket(2) and the respective protocol man pages). When addr is NULL, nothing is filled in; in this case, addrlen is not used, and should also be NULL.

The addrlen argument is a value-result argument: the caller must initialize it to contain the size (in bytes) of the structure pointed to by addr; on return it will contain the actual size of the peer address.

The returned address is truncated if the buffer provided is too small; in this case, addrlen will return a value greater than was supplied to the call.

If no pending connections are present on the queue, and the socket is not marked as non blocking, accept() blocks the caller until a connection is present.

If the socket is marked non-blocking and no pending connections are present on the queue, accept() fails with the error EAGAIN or EWOULD‐BLOCK**.**

In order to be notified of incoming connections on a socket, you can use select(2), poll(2), or epoll(7). A readable event will be delivered when a new

connection is attempted and you may then call accept() to get a socket for that connection. Alternatively, you can set the socket to deliver SIGIO when activity occurs on a socket; see socket(7) for details.

If flags is 0, then accept4() is the same as accept(). The following values can be bitwise ORed in flags to obtain different behavior:

SOCK_NONBLOCK Set the O_NONBLOCK file status flag on the new open file description. Using this flag saves extra calls to fcntl(2) to achieve the same result.

SOCK_CLOEXEC Set the close-on-exec (FD_CLOEXEC) flag on the new file descriptor. See the description of the O_CLOEXEC flag in open(2) for reasons why this may be useful.

RETURN VALUE

On success, these system calls return a nonnegative integer that is a file descriptor for the accepted socket. On error, -1 is returned, and errno is set appropriately.

Error handling

Linux accept() (and accept4()) passes already-pending network errors on the new socket as an error code from accept(). This behavior differs from other BSD socket implementations. For reliable operation the application should detect the network errors defined for the protocol after accept() and treat them like EAGAIN by retrying. In the case of TCP/IP, these are ENETDOWN, EPROTO, ENOPROTOOPT, EHOSTDOWN, ENONET, EHOSTUNREACH, EOPNOTSUPP, and ENETUNREACH.

10.man send

NAME: send, sendto, sendmsg - send a message on 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 (so that the intended recipient is known). The only difference between send() and

write(2) is the presence of flags. With a zero flags argument, send() is equivalent to write(2). Also, the following call
send(sockfd, buf, len, flags); is equivalent to
sendto(sockfd, buf, len, flags, NULL, 0);

The argument sockfd is the file descriptor of the sending socket.

If sendto() is used on a connection-mode (SOCK_STREAM, SOCK_SEQPACKET) socket, the arguments dest_addr and addrlen are ignored (and the error EISCONN may be returned when they are not NULL and 0), and the error ENOTCONN is returned when the socket was not actually connected. Otherwise, the address of the target is given by dest_addr with addrlen specifying its size. For sendmsg(), the address of the target is given by msg.msg_name, with msg.msg_namelen specifying its size.

For send() and sendto(), the message is found in buf and has length len. For sendmsg(), the message is pointed to by the elements of the array msg.msg_iov. The sendmsg() call also allows sending ancillary data (also known as control information).

If the message is too long to pass atomically through the underlying protocol, the error EMSGSIZE is returned, and the message is not transmitted.

No indication of failure to deliver is implicit in a send(). Locally detected errors are indicated by a return value of -1.

When the message does not fit into the send buffer of the socket, send() normally blocks, unless the socket has been placed in nonblocking I/O mode.

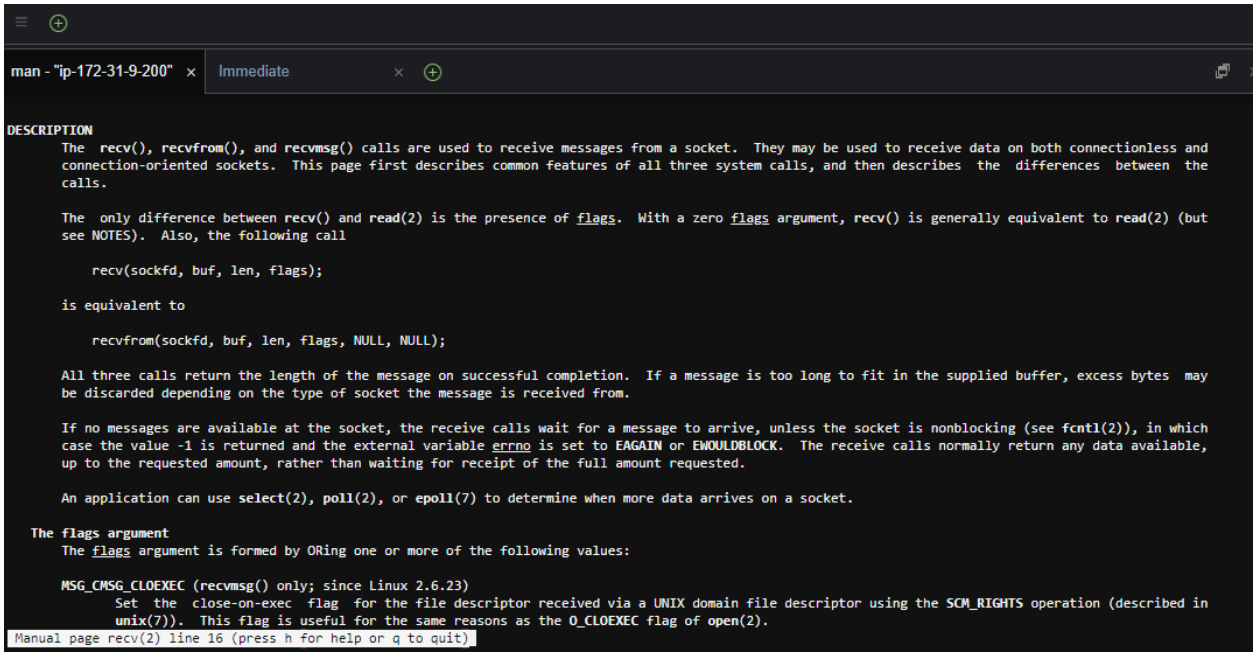
In nonblocking mode it would fail with the error EAGAIN or EWOULDBLOCK in this case. The select(2) call may be used to determine when it is possible to send more data.

11. 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 sockfd, void *buf, size_t len, int flags);
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen);
ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);
```



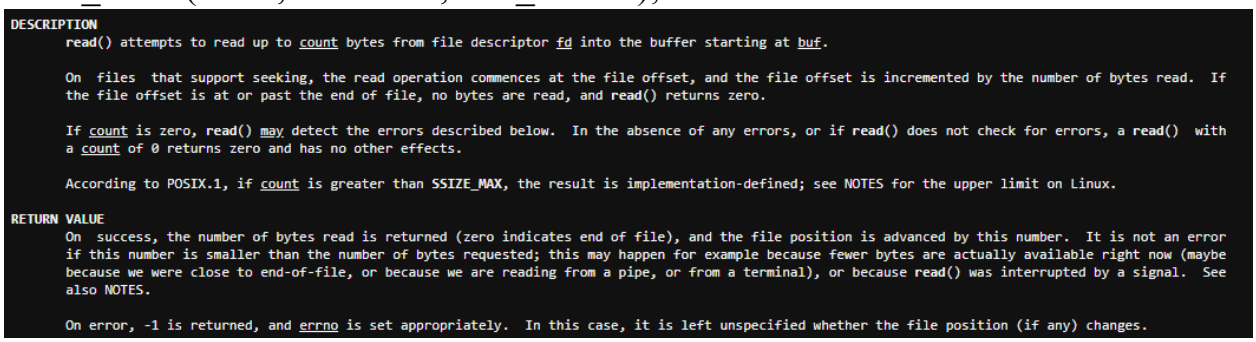
12. man read

NAME: read - read from a file descriptor

SYNOPSIS:

#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);



13. man write

NAME: write- send a message to another user.

SYNOPSIS:

write user [tty]

```
DESCRIPTION
     The write utility allows you to communicate with other users, by copying lines from your terminal to theirs.

     When you run the write command, the user you are writing to gets 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.

     When you are done, type an end-of-file or interrupt character.  The other user will see the message 'EOF' indicating that the conversation is over.

     You can prevent people (other than the super-user) from writing to you with the mesg(1) command.

     If the user you want to write to is logged in on more than one terminal, you can specify which terminal to write to by specifying the terminal name as
     the second operand to the write command.  Alternatively, you can let write select one of the terminals - it will pick the one with the shortest idle
     time.  This is so that if the user is logged in at work and also dialed up from home, the message will go to the right place.

     The traditional protocol for writing to someone is that the string '-o', either at the end of a line or on a line by itself, means that it is the other
     person's turn to talk.  The string 'oo' means that the person believes the conversation to be over.
```

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 specifies 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 families.  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:

           struct sockaddr {
               sa_family_t sa_family;
               char        sa_data[14];
           }
```

15. Ifconfig

```
Kkottilingam:~/environment $ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:77:df:f3:6f  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 172.31.0.209  netmask 255.255.240.0  broadcast 172.31.15.255
        inet6 fe80::415:5aff:fed3:84b7  prefixlen 64  scopeid 0x20<link>
        ether 06:15:5a:d3:84:b7  txqueuelen 1000  (Ethernet)
        RX packets 332986  bytes 226674424 (226.6 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 200803  bytes 40046711 (40.0 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 532  bytes 72104 (72.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 532  bytes 72104 (72.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

16. man htons/ man htonl

```
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 Internet, is Most Significant Byte
       first.
```

17. man gethostname

```
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 < 500)
           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 hostname 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 terminating null byte.)
```

18. man gethostbyname

```
DESCRIPTION
       The gethostbyname*(), gethostbyaddr*(), herror(), and hstrerror() functions are obsolete.  Applications should  use  getaddrinfo(3),  getnameinfo(3),
       and gai_strerror(3) instead.

       The gethostbyname() function returns a structure of type hostent for the given host name.  Here name is either a hostname or an IPv4 address in stan-
       dard dot notation (as for inet_addr(3)).  If name is an IPv4 address, no lookup is performed and gethostbyname() simply copies name into  the  h_name
       field  and its struct in_addr equivalent into the h_addr_list[0] field of the returned hostent structure.  If name doesn't end in a dot and the envi-
       ronment variable HOSTALIASES is set, the alias file pointed to by HOSTALIASES will first be searched for name (see hostname(7) for the file  format).
       The current domain and its parents are searched unless name ends in a dot.

       The  gethostbyaddr() function returns a structure of type hostent for the given host address addr of length len and address type type.  Valid address
       types are AF_INET and AF_INET6.  The host address argument is a pointer to a struct of a type depending on the address type,  for  example  a  struct
       in_addr * (probably obtained via a call to inet_addr(3)) for address type AF_INET.

       The  sethostent()  function  specifies,  if stayopen is true (1), that a connected TCP socket should be used for the name server queries and that the
       connection should remain open during successive queries.  Otherwise, name server queries will use UDP datagrams.

       The endhostent() function ends the use of a TCP connection for name server queries.

       The (obsolete) herror() function prints the error message associated with the current value of h_errno on stderr.

       The (obsolete) hstrerror() function takes an error number (typically h_errno) and returns the corresponding message string.

       The domain name queries carried out by gethostbyname() and gethostbyaddr() rely on the Name Service Switch (nsswitch.conf(5)) configured sources or a
       local  name  server  (named(8)).  The default action is to query the Name Service Switch (nsswitch.conf(5)) configured sources, failing that, a local
       name server (named(8)).
```

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

NAME
       gethostbyname,  gethostbyaddr,  sethostent,  gethostent,  endhostent,  h_errno, herror, hstrerror, gethostbyaddr_r, gethostbyname2, 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);
```

```
Kkottilingam:~/environment $ man socket
Kkottilingam:~/environment $ man connect
Kkottilingam:~/environment $ man accept
Kkottilingam:~/environment $ man send
Kkottilingam:~/environment $ man recv
Kkottilingam:~/environment $ man read
Kkottilingam:~/environment $ man write
Kkottilingam:~/environment $ man bind
Kkottilingam:~/environment $ man htons
Kkottilingam:~/environment $ man gethostname
Kkottilingam:~/environment $ man gethostbyname
Kkottilingam:~/environment $ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        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<link>
        ether 06:7b:6f:64:1e:5d  txqueuelen 1000  (Ethernet)
        RX packets 424871  bytes 257936958 (257.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 286400  bytes 82621503 (82.6 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
```