

NAME

socket - create an endpoint for communication

SYNOPSIS

```
cc [ _f_l_a_g... ] _f_      i_l_e... - lsocket -lnsl [ _l_  i_b_r_a_r_y... ]  
#include <sys/types.h>  
#include <sys/socket.h>
```

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

DESCRIPTION

socket() creates an endpoint for communication and returns a descriptor.

The `_d_o_m_a_i_n` parameter specifies a communications domain within which communication will take place; this selects the protocol family which should be used. The protocol family generally is the same as the address family for the addresses supplied in later operations on the socket. These families are defined in the include file `<sys/socket.h>`. There must be an entry in the `netconfig(4)` file for at least each protocol family and type required. If `_p_r_o_t_o_c_o_l` has been specified, but no exact match for the tuple family, type, protocol is found, then the first entry containing the specified family and type with zero for protocol will be used. The currently understood formats are:

PF_UNIX

UNIX system internal protocols

PF_INET

Internet Protocol Version 4 (IPv4)

PF_INET6

Internet Protocol Version 6 (IPv6)

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

SOCK_STREAM

SOCK_DGRAM

SOCK_RAW

SOCK_SEQPACKET

SOCK_RDM

A `SOCK_STREAM` type provides sequenced, reliable, two-way connection-based byte streams. An out-of-band data transmission mechanism may be supported. A `SOCK_DGRAM` socket supports datagrams (connectionless, unreliable messages of a fixed (typically small) maximum length). A `SOCK_SEQPACKET` socket may provide a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer may be required to read an entire packet with each read system call. This facility is

protocol specific, and presently not implemented for any protocol family. SOCK_RAW sockets provide access to internal network interfaces. The types SOCK_RAW, which is available only to the superuser, and SOCK_RDM, for which no implementation currently exists, are not described here.

`_p_r_o_t_o_c_o_l` specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family. However, multiple protocols may exist, in which case a particular protocol must be specified in this manner. The protocol number to use is particular to the "communication domain" in which communication is to take place. If a protocol is specified by the caller, then it will be packaged into a socket level option request and sent to the underlying protocol layers.

Sockets of type SOCK_STREAM are full-duplex byte streams, similar to pipes. A stream socket must be in a `_c_o_n_n_e_c_t_e_d` state before any data may be sent or received on it. A connection to another socket is created with a `connect(3SOCKET)` call. Once connected, data may be transferred using `read(2)` and `write(2)` calls or some variant of the `send(3SOCKET)` and `recv(3SOCKET)` calls. When a session has been completed, a `close(2)` may be performed. Out-of-band data may also be transmitted as described on the `send(3SOCKET)` manual page and received as described on the `recv(3SOCKET)` manual page.

The communications protocols used to implement a SOCK_STREAM insure that data is not lost or duplicated. If a piece of data for which the peer protocol has buffer space cannot be successfully transmitted within a reasonable length of time, then the connection is considered broken and calls will indicate an error with -1 returns and with ETIMEDOUT as the specific code in the global variable `errno`. The protocols optionally keep sockets "warm" by forcing transmissions roughly every minute in the absence of other activity. An error is then indicated if no response can be elicited on an otherwise idle connection for an extended period (for instance 5 minutes). A SIGPIPE signal is raised if a process sends on a broken stream; this causes naive processes, which do not handle the signal, to exit.

SOCK_SEQPACKET sockets employ the same system calls as SOCK_STREAM sockets. The only difference is that `read(2)` calls will return only the amount of data requested, and any remaining in the arriving packet will be discarded.

SOCK_DGRAM and SOCK_RAW sockets allow datagrams to be sent to correspondents named in `sendto(3SOCKET)` calls. Datagrams are generally received with `recvfrom(3SOCKET)`, which returns the next datagram with its return address.

An `fcntl(2)` call can be used to specify a process group to receive a SIGURG signal when the out-of-band data arrives. It may also enable non-blocking I/O and asynchronous notifi-

cation of I/O events with SIGIO signals.

The operation of sockets is controlled by socket level `_o_p_t_i_o_n_s`. These options are defined in the file `<sys/socket.h>`. `setsockopt(3SOCKET)` and `getsockopt(3SOCKET)` are used to set and get options, respectively.

RETURN VALUES

A -1 is returned if an error occurs. Otherwise the return value is a descriptor referencing the socket.

ERRORS

The `socket()` call fails if:

EACCES

Permission to create a socket of the specified type and/or protocol is denied.

EMFILE

The per-process descriptor table is full.

ENOMEM

Insufficient user memory is available.

ENOSR There were insufficient STREAMS resources available to complete the operation.

EPROTONOSUPPORT

The protocol type or the specified protocol is not supported within this domain.

NAME

`bind` - bind a name to a socket

SYNOPSIS

```
cc [ _f_l_a_g ... ] _f_      i_l_e ... - lsocket -lnsl [ _l_  i_b_r_a_r_y ... ]  
#include <sys/types.h>  
#include <sys/socket.h>
```

```
int bind(int s, const struct sockaddr *name, int namelen);
```

DESCRIPTION

`bind()` assigns a name to an unnamed socket. When a socket is created with `socket(3SOCKET)`, it exists in a name space (address family) but has no name assigned. `bind()` requests that the name pointed to by `_n_a_m_e` be assigned to the socket.

RETURN VALUES

If the bind is successful, 0 is returned. A return value of -1 indicates an error, which is further specified in the global `errno`.

ERRORS

The `bind()` call will fail if:

EACCES

The requested address is protected and the current user has inadequate permission to access it.

EADDRINUSE

The specified address is already in use.

EADDRNOTAVAIL

The specified address is not available on the local machine.

EBADF `_s` is not a valid descriptor.

EINVAL

`_n_a_m_e_l_e_n` is not the size of a valid address for the specified address family.

EINVAL

The socket is already bound to an address.

ENOSR There were insufficient STREAMS resources for the operation to complete.

ENOTSOCK

`_s` is a descriptor for a file, not a socket.

The following errors are specific to binding names in the UNIX domain:

EACCES

Search permission is denied for a component of the path prefix of the pathname in `_n_a_m_e`.

EIO An I/O error occurred while making the directory entry or allocating the inode.

EISDIR

A null pathname was specified.

ELOOP Too many symbolic links were encountered in translating the pathname in `_n_a_m_e`.

ENOENT

A component of the path prefix of the pathname in `_n_a_m_e` does not exist.

ENOTDIR

A component of the path prefix of the pathname in `_n_a_m_e` is not a directory.

EROFS The inode would reside on a read-only file system.

NAME

listen - network listener daemon

SYNOPSIS

```
/usr/lib/saf/listen [ -m _ d _ e _ v _ s _ t _ e _ m ] _ n _ e _ t _ s _ p _ e _ c
```

DESCRIPTION

The listen process ``listens" to a network for service requests, accepts requests when they arrive, and invokes servers in response to those service requests. The network listener process may be used with any connection-oriented network (more precisely, with any connection-oriented transport provider) that conforms to the Transport Layer Interface (TLI) Specification.

The listener internally generates a pathname for the minor device for each connection; it is this pathname that is used in the utmpx entry for a service, if one is created. By default, this pathname is the concatenation of the prefix /dev/_ n _ e _ t _ s _ p _ e _ c with the decimal representation of the minor device number. In either case, the representation of the minor device number will be at least two digits (for example, 05 or 27), or longer when it is necessary to accommodate minor device numbers larger than 99.

SERVER INVOCATION

When a connection indication is received, the listener creates a new transport endpoint and accepts the connection on that endpoint. Before giving the file descriptor for this new connection to the server, any designated STREAMS modules are pushed and the configuration script is executed, (if one exists). This file descriptor is appropriate for use with either TLI (see t_sync(3NSL)) or the sockets interface library.

By default, a new instance of the server is invoked for each connection. When the server is invoked, file descriptor 0 refers to the transport endpoint, and is open for reading and writing. File descriptors 1 and 2 are copies of file descriptor 0; no other file descriptors are open. The service is invoked with the user and group IDs of the user name under which the service was registered with the listener, and with the current directory set to the HOME directory of that user.

Alternatively, a service may be registered so that the listener will pass connections to a standing server process through a FIFO or a named STREAM, instead of invoking the server anew for each connection. In this case, the connection is passed in the form of a file descriptor that refers to the new transport endpoint. Before the file descriptor is sent to the server, the listener interprets any

NAME

connect - initiate a connection on a socket

SYNOPSIS

```
cc [ _ f _ l _ a _ g ... ] _ f _ i _ l _ e ... - lsocket -lnsl [ _ l _ i _ b _ r _ a _ r _ y ... ]
```

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

```
int connect(int s, const struct sockaddr *name, int
namelen);
```

DESCRIPTION

The parameter `s` is a socket. If it is of type `SOCK_DGRAM`, `connect()` specifies the peer with which the socket is to be associated; this address is the address to which datagrams are to be sent if a receiver is not explicitly designated; it is the only address from which datagrams are to be received. If the socket `s` is of type `SOCK_STREAM`, `connect()` attempts to make a connection to another socket. The other socket is specified by `name`. `name` is an address in the communication space of the socket. Each communication space interprets the `name` parameter in its own way. If `s` is not bound, then it will be bound to an address selected by the underlying transport provider. Generally, stream sockets may successfully `connect()` only once; datagram sockets may use `connect()` multiple times to change their association. Datagram sockets may dissolve the association by connecting to a null address.

RETURN VALUES

If the connection or binding succeeds, 0 is returned. Otherwise, -1 is returned and sets `errno` to indicate the error.

ERRORS

The call fails if:

EACCES

Search permission is denied for a component of the path prefix of the pathname in `name`.

EADDRINUSE

The address is already in use.

EADDRNOTAVAIL

The specified address is not available on the remote machine.

EAFNOSUPPORT

Addresses in the specified address family cannot be used with this socket.

EALREADY

The socket is non-blocking and a previous connection attempt has not yet been completed.

`EBADF` `s` is not a valid descriptor.

ECONNREFUSED

The attempt to connect was forcefully rejected. The

calling program should close(2) the socket descriptor, and issue another socket(3SOCKET) call to obtain a new descriptor before attempting another connect() call.

EINPROGRESS

The socket is non-blocking and the connection cannot be completed immediately. It is possible to select(3C) for completion by selecting the socket for writing. However, this is only possible if the socket STREAMS module is the topmost module on the protocol stack with a write service procedure. This will be the normal case.

EINTR The connection attempt was interrupted before any data arrived by the delivery of a signal.

EINVAL

`_n_a_m_e_l_e_n` is not the size of a valid address for the specified address family.

EIO An I/O error occurred while reading from or writing to the file system.

EISCONN

The socket is already connected.

ELOOP Too many symbolic links were encountered in translating the pathname in `_n_a_m_e`.

ENETUNREACH

The network is not reachable from this host.

ENOENT

A component of the path prefix of the pathname in `_n_a_m_e` does not exist.

ENOENT

The socket referred to by the pathname in `_n_a_m_e` does not exist.

ENOSR There were insufficient STREAMS resources available to complete the operation.

ENXIO The server exited before the connection was complete.

ETIMEDOUT

Connection establishment timed out without establishing a connection.

EWouldBlock

The socket is marked as non-blocking, and the requested operation would block.

The following errors are specific to connecting names in the UNIX domain. These errors may not apply in future versions of the UNIX IPC domain.

ENOTDIR

A component of the path prefix of the pathname in `_n_a_m_e` is not a directory.

ENOTSOCK

`_s` is not a socket.

ENOTSOCK

`_n_a_m_e` is not a socket.

EPROTOTYPE

The file referred to by `_n_a_m_e` is a socket of a type other than type `_s` (for example, `_s` is a `SOCK_DGRAM` socket, while `_n_a_m_e` refers to a `SOCK_STREAM` socket).

NAME

`accept`, `reject` - accept or reject print requests

SYNOPSIS

```
accept _d_e_s_t _i_n_a_t _i_o_n...
```

```
reject [-r_r_e_a_s_o_n] _d_e_s_t _i_n_a_t _i_o_n...
```

DESCRIPTION

`accept` allows the queueing of print requests for the named destinations.

`reject` prevents queueing of print requests for the named destinations.

Use `lpstat -a` to check if destinations are accepting or rejecting print requests.

`accept` and `request` must be run on the print server; they have no meaning to a client system.

OPTIONS

The following options are supported for `reject`.

`-r_r_e_a_s_o_n`

Assigns a reason for rejection of print requests for

`_d_e_s_t _i_n_a_t _i_o_n`. Enclose `_r_e_a_s_o_n` in quotes if it contains

blanks. `_r_e_a_s_o_n` is reported by `lpstat -a`. By default,

`_r_e_a_s_o_n` is unknown reason for existing destinations, and new printer for destinations added to the system but not yet accepting requests.

OPERANDS

The following operands are supported.

```
_d_e_s_t _i_n_a_t _i_o_n
```

The name of the destination accepting or rejecting print requests. Destination specifies the name of a printer or class of printers (see `lpadmin(1M)`).

Specify `_d_e_s_t_` `i_n_a_t_` `i_o_n` using atomic name. See `printers.conf(4)` for information regarding the naming conventions for atomic names.

EXIT STATUS

The following exit values are returned:

0 Successful completion.

`_n_o_n_z_e_r_o`

An error occurred.

Networking Services Library Functions `gethostbyname(3NSL)`

NAME

`gethostbyname`, `gethostbyname_r`, `gethostbyaddr`,
`gethostbyaddr_r`, `gethostent`, `gethostent_r`, `sethostent`,
`endhostent` - get network host entry

SYNOPSIS

```
cc [ _f_l_a_g... ] _f_ i_l_e... - lns1 [ _l_ i_b_r_a_r_y... ]  
#include <netdb.h>
```

```
struct hostent *gethostbyname(const char *name);
```

```
struct hostent *gethostbyname_r(const char *name, struct  
hostent *result, char *buffer, intbuflen, int *h_errnop);
```

```
struct hostent *gethostbyaddr(const char *addr, int len, int  
type);
```

```
struct hostent *gethostbyaddr_r(const char *addr, int  
length, int type, struct hostent *result, char *buffer, int  
buflen, int *h_errnop);
```

```
struct hostent *gethostent(void);
```

```
struct hostent *gethostent_r(struct hostent *result, char  
*buffer, int buflen, int *h_errnop);
```

```
int sethostent(int stayopen);
```

```
int endhostent(void);
```

DESCRIPTION

These functions are used to obtain entries describing hosts.

An entry may come from any of the sources for hosts specified in the `/etc/nsswitch.conf` file. See `nsswitch.conf(4)`. Please take note that these functions have been superseded by the newer functions, `getipnodebyname(3SOCKET)`, `getipnodebyaddr(3SOCKET)`, and `getaddrinfo(3SOCKET)`. The newer functions provide greater portability to applications when multithreading is done or technologies such as IPv6 are used. For example, the functions described below cannot be used with applications targeted to work with IPv6.

`gethostbyname()` searches for information for a host with the hostname specified by the character-string parameter `_n_a_m_e`.

`gethostbyaddr()` searches for information for a host with a given host address. The parameter type specifies the family of the address. This should be one of the address families defined in `<sys/socket.h>`. The parameter `_a_d_d_r` must be a pointer to a buffer containing the address. The address is given in a form specific to the address family. See the NOTES section below for more information. Also see the EXAMPLES section below on how to convert a ``." separated Internet IP address notation into the `_a_d_d_r` parameter. The parameter `_l_e_n` specifies the length of the buffer indicated by `_a_d_d_r`.

All addresses are returned in network order. In order to interpret the addresses, `byteorder(3SOCKET)` must be used for byte order conversion.

The functions `sethostent()`, `gethostent()`, and `endhostent()` are used to enumerate host entries from the database.

`sethostent()` sets (or resets) the enumeration to the beginning of the set of host entries. This function should be called before the first call to `gethostent()`. Calls to `gethostbyname()` and `gethostbyaddr()` leave the enumeration position in an indeterminate state. If the `_s_t_a_y_o_p_e_n` flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to `endhostent()`.

Successive calls to `gethostent()` return either successive entries or `_N_U_L`, indicating the end of the enumeration.

`endhostent()` may be called to indicate that the caller expects to do no further host entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more host retrieval functions after calling `endhostent()`.

Reentrant Interfaces

The functions `gethostbyname()`, `gethostbyaddr()`, and `gethostent()` use static storage that is reused in each call, making these functions unsafe for use in multi-threaded applications.

The functions `gethostbyname_r()`, `gethostbyaddr_r()`, and `gethostent_r()` provide reentrant interfaces for these operations.

Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the ```_r"` suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multi-threaded applications.

Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter `_r_e_s_u_l_t` must be a pointer to a struct hostent structure allocated by the caller. On successful completion, the function returns the host entry in this structure. The parameter `_b_u_f_f_e_r` must be a pointer to a buffer supplied by the caller. This buffer is used as storage space for the host data. All of the pointers within the returned struct hostent `_r_e_s_u_l_t` point to data stored within this buffer. See RETURN VALUES. The buffer must be large enough to hold all of the data associated with the host entry. The parameter `_b_u_f_l_e_n` should give the size in bytes of the buffer indicated by `_b_u_f_f_e_r`. The parameter `_h_e_r_r_n_o_p` should be a pointer to an integer. An integer error status value is stored there on certain error conditions. See ERRORS.

For enumeration in multi-threaded applications, the position within the enumeration is a process-wide property shared by all threads. `sethostent()` may be used in a multi-threaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `gethostent_r()`, the threads will enumerate disjoint subsets of the host database.

Like their non-reentrant counterparts, `gethostbyname_r()` and `gethostbyaddr_r()` leave the enumeration position in an indeterminate state.

RETURN VALUES

Host entries are represented by the struct hostent structure defined in `<netdb.h>`:

```
struct hostent {
    char *h_name; /* canonical name of host */
    char **h_aliases; /* alias list */
    int h_addrtype; /* host address type */
    int h_length; /* length of address */
    char **h_addr_list; /* list of addresses */
};
```

See the EXAMPLES section below for information about how to retrieve a ```."` separated Internet IP address string from

the `_h__a__d__d__r__l__i__s__t` field of `struct hostent`.

The functions `gethostbyname()`, `gethostbyname_r()`, `gethostbyaddr()`, and `gethostbyaddr_r()` each return a pointer to a `struct hostent` if they successfully locate the requested entry; otherwise they return `_N_U_L_ L`.

The functions `gethostent()` and `gethostent_r()` each return a pointer to a `struct hostent` if they successfully enumerate an entry; otherwise they return `_N_U_L_ L`, indicating the end of the enumeration.

The functions `gethostbyname()`, `gethostbyaddr()`, and `gethostent()` use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions `gethostbyname_r()`, `gethostbyaddr_r()`, and `gethostent_r()` is not `_N_U_L_ L`, it is always equal to the `_r__e__s__u__l__t` pointer that was supplied by the caller.

The functions `sethostent()` and `endhostent()` return 0 on success.

ERRORS

The reentrant functions `gethostbyname_r()`, `gethostbyaddr_r()`, and `gethostent_r()` will return `_N_U_L_ L` and set `_e__r__r__n__o` to `ERANGE` if the length of the buffer supplied by caller is not large enough to store the result. See `Intro(2)` for the proper usage and interpretation of `errno` in multithreaded applications.

The reentrant functions `gethostbyname_r()` and `gethostbyaddr_r()` set the integer pointed to by `_h__e__r__r__n__o__p` to one of these values in case of error.

On failures, the non-reentrant functions `gethostbyname()` and `gethostbyaddr()` set a global integer `_h__e__r__r__n__o` to indicate one of these error codes (defined in `<netdb.h>`): `HOST_NOT_FOUND`, `TRY_AGAIN`, `NO_RECOVERY`, `NO_DATA`, and `NO_ADDRESS`.

Note however that if a resolver is provided with a malformed address, or if any other error occurs before `gethostbyname()` is resolved, then `gethostbyname()` returns an internal error with a value of `-1`.

`gethostbyname()` will set `_h__e__r__r__n__o` to `NETDB_INTERNAL` when it returns a `_N_U_L_ L` value.

EXAMPLES

Example 1: Using `gethostbyname()`

Here is a sample program that gets the canonical name, aliases, and ```.` separated Internet IP addresses for a given ```.` separated IP address:

```

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

```

Networking Services Library Functions gethostbyname(3NSL)

```

main(int argc, const char **argv)
{
    ulong_t addr;
    struct hostent *hp;
    char **p;
    if (argc != 2) {
        (void) printf("usage: %s IP-address\n", argv[0]);
        exit (1);
    }
    if ((int)(addr = inet_addr(argv[1])) == -1) {
        (void) printf("IP-address must be of the form a.b.c.d\n");
        exit (2);
    }
    hp = gethostbyaddr((char *)&addr, sizeof (addr), AF_INET);
    if (hp == NULL) {
        (void) printf("host information for %s not found\n", argv[1]);
        exit (3);
    }
    for (p = hp->h_addr_list; *p != 0; p++) {
        struct in_addr in;
        char **q;
        (void) memcpy(&in.s_addr, *p, sizeof (in.s_addr));
        (void) printf("%s\t%s", inet_ntoa(in), hp->h_name);
        for (q = hp->h_aliases; *q != 0; q++)
            (void) printf(" %s", *q);
        (void) putchar('\n');
    }
    exit (0);
}

```

Note that the above sample program is unsafe for use in multithreaded applications.

FILES

/etc/hosts

/etc/netconfig

/etc/nsswitch.conf

WARNINGS

The reentrant interfaces `gethostbyname_r()`, `gethostbyaddr_r()`, and `gethostent_r()` are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

NOTES

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

In order to ensure that they all return consistent results, `gethostbyname()`, `gethostbyname_r()`, and `netdir_getbyname()` are implemented in terms of the same internal library function. This function obtains the system-wide source lookup policy based on the `inet` family entries in `netconfig(4)` and the `hosts:` entry in `nsswitch.conf(4)`. Similarly, `gethostbyaddr()`, `gethostbyaddr_r()`, and `netdir_getbyaddr()` are implemented in terms of the same internal library function. If the `inet` family entries in `netconfig(4)` have a ```-"` in the last column for `nametoaddr` libraries, then the entry for `hosts` in `nsswitch.conf` will be used; otherwise the `name-toaddr` libraries in that column will be used, and `nsswitch.conf` will not be consulted.

There is no analogue of `gethostent()` and `gethostent_r()` in the `netdir` functions, so these enumeration functions go straight to the `hosts` entry in `nsswitch.conf`. Thus enumeration may return results from a different source than that used by `gethostbyname()`, `gethostbyname_r()`, `gethostbyaddr()`, and `gethostbyaddr_r()`.

All the functions that return a `struct hostent` must always return the `_c_a_n_o_n_ i_c_a_l_n_a_m_e` in the `_h__n_a_m_e` field.

This name, by

definition, is the well-known and official `hostname` shared between all aliases and all addresses. The underlying source that satisfies the request determines the mapping of the input name or address into the set of names and addresses in `hostent`. Different sources might do that in different ways. If there is more than one alias and more than one address in `hostent`, no pairing is implied between them.

The system will strive to put the addresses on the same subnet as that of the caller first.

When compiling multi-threaded applications, see `Intro(3)`,

`_N_o_t_e_s_ O_n_ M_u_l_t_ i_t_h_r_e_a_d`
`_A_p_ p_l_ i_c_a_t_ i_o_n_s`, for information about the
use of the `_REENTRANT` flag.

Use of the enumeration interfaces `gethostent()` and `gethostent_r()` is discouraged; enumeration may not be supported for all database sources. The semantics of enumeration are discussed further in `nsswitch.conf(4)`.

The current implementations of these functions only return or accept addresses for the Internet address family (type `AF_INET`).

The form for an address of type `AF_INET` is a struct `in_addr` defined in `<netinet/in.h>`. The functions described in `inet(3SOCKET)`, and illustrated in the `EXAMPLES` section above, are helpful in constructing and manipulating addresses in this form.

Sockets Library Functions `getservbyname(3SOCKET)`

NAME

`getservbyname`, `getservbyname_r`, `getservbyport`,
`getservbyport_r`, `getservent`, `getservent_r`, `setservent`,
`endservent` - get service entry

SYNOPSIS

```
cc [ _f_l_a_g ... ] _f_ i_l_e ... - lsocket -lnsl [ _l_ i_b_r_a_r_y ... ]  
#include <netdb.h>
```

```
struct servent *getservbyname(const char *name, const char  
*proto);
```

```
struct servent *getservbyname_r(const char *name, const char  
*proto, struct servent *result, char *buffer, int buflen);
```

```
struct servent *getservbyport(int port, const char *proto);
```

```
struct servent *getservbyport_r(int port, const char *proto,  
struct servent *result, char *buffer, int buflen);
```

```
struct servent *getservent(void);
```

```
struct servent *getservent_r(struct servent *result, char  
*buffer, int buflen);
```

```
int setservent(int stayopen);
```

```
int endservent(void);
```

DESCRIPTION

These functions are used to obtain entries for Internet services. An entry may come from any of the sources for ser-

vices specified in the `/etc/nsswitch.conf` file. See `nsswitch.conf(4)`.

`getservbyname()` and `getservbyport()` sequentially search from the beginning of the file until a matching protocol name or port number is found, or until end-of-file is encountered. If a protocol name is also supplied (non-`_N_U_L_`), searches must also match the protocol.

`getservbyname()` searches for an entry with the Internet service name specified by the parameter `_n_a_m_e`.

`getservbyport()` searches for an entry with the Internet port number `_p_o_r_t`.

All addresses are returned in network order. In order to interpret the addresses, `byteorder(3SOCKET)`

must be used for byte order conversion. The string `_p_r_o_t_o` is used by both `getservbyname()` and `getservbyport()` to restrict

the search to entries with the specified protocol. If `_p_r_o_t_o` is `_N_U_L_`, entries with any protocol may be returned.

The functions `setservent()`, `getservent()`, and `endservent()` are used to enumerate entries from the services database.

`setservent()` sets (or resets) the enumeration to the beginning of the set of service entries. This function should be called before the first call to `getservent()`. Calls to the functions `getservbyname()` and `getservbyport()` leave the enumeration position in an indeterminate state. If the `_s_t_a_y_o_p_e_n` flag is non-zero, the system may keep allocated resources such as open file descriptors until a subsequent call to `endservent()`.

`getservent()` reads the next line of the file, opening the file if necessary. `getservent()` opens and rewinds the file. If the `_s_t_a_y_o_p_e_n` flag is non-zero, the net data base will not be closed after each call to `getservent()` (either directly, or indirectly through one of the other "getserv" calls).

Successive calls to `getservent()` return either successive entries or `_N_U_L_`, indicating the end of the enumeration.

`endservent()` closes the file. `endservent()` may be called to indicate that the caller expects to do no further service entry retrieval operations; the system may then deallocate resources it was using. It is still allowed, but possibly less efficient, for the process to call more service entry retrieval functions after calling `endservent()`.

Reentrant Interfaces

The functions `getservbyname()`, `getservbyport()`, and `getservent()` use static storage that is re-used in each call, making these functions unsafe for use in multithreaded applications.

The functions `getservbyname_r()`, `getservbyport_r()`, and `getservent_r()` provide reentrant interfaces for these operations.

Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the `"_r"` suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multithreaded applications.

Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter `_r_e_s_u_l_t` must be a pointer

to a struct `servent` structure allocated by the caller. On successful completion, the function returns the service entry in this structure. The parameter `_b_u_f_f_e_r` must be a pointer to a buffer supplied by the caller. This buffer is used as storage space for the service entry data. All of the pointers within the returned struct `servent_r_e_s_u_l_t` point to data stored within this buffer. See the RETURN VALUES section of this man page. The buffer must be large enough to hold all of the data associated with the service entry. The parameter `_b_u_f_l_e_n` should give the size in bytes of the buffer indicated by `_b_u_f_f_e_r`.

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. `setservent()` may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to `getservent_r()`, the threads will enumerate disjoint subsets of the service database.

Like their non-reentrant counterparts, `getservbyname_r()` and `getservbyport_r()` leave the enumeration position in an indeterminate state.

RETURN VALUES

Service entries are represented by the struct `servent` structure defined in `<netdb.h>`:

```
struct servent {
    char *s_name;          /* official name of service */
    char **s_aliases;     /* alias list */
    int s_port;           /* port service resides at */
    char *s_proto;        /* protocol to use */
};
```

The members of this structure are:

- `s_name`
The official name of the service.
- `s_aliases`
A zero terminated list of alternate names for the service.
- `s_port`
The port number at which the service resides. Port numbers are returned in network byte order.
- `s_proto`
The name of the protocol to use when contacting the service

The functions `getservbyname()`, `getservbyname_r()`, `getservbyport()`, and `getservbyport_r()` each return a pointer to a struct `servent` if they successfully locate the requested entry; otherwise they return `_N_U_L_` L.

The functions `getservent()` and `getservent_r()` each return a pointer to a struct `servent` if they successfully enumerate an entry; otherwise they return `_N_U_L_` L, indicating the end of the enumeration.

The functions `getservbyname()`, `getservbyport()`, and `getservent()` use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions `getservbyname_r()`, `getservbyport_r()`, and `getservent_r()` is non-null, it is always equal to the `_r_e_s_u_l_t` pointer that was supplied by the caller.

ERRORS

The reentrant functions `getservbyname_r()`, `getservbyport_r()` and `getservent_r()` will return `_N_U_L_` L and set `errno` to `ERANGE` if the length of the buffer supplied by caller is not large enough to store the result. See `intro(2)` for the proper usage and interpretation of `errno` in multithreaded applications.

FILES

- `/etc/services`
Internet network services
- `/etc/netconfig`
network configuration file

/etc/nsswitch.conf
configuration file for the name-service switch

WARNINGS

The reentrant interfaces `getservbyname_r()`, `getservbyport_r()`, and `getservent_r()` are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

NOTES

The functions that return `struct servent` return the least significant 16-bits of the `_s_p_o_r_t` field in `_n_e_t_w_o_r_k_b_y_t_e_o_r_d_e_r`. `getservbyport()` and `getservbyport_r()` also expect the input parameter `_p_o_r_t` in the `_n_e_t_w_o_r_k_b_y_t_e_o_r_d_e_r`. See `htons(3SOCKET)` for more details on converting between host and network byte orders.

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

In order to ensure that they all return consistent results, `getservbyname()`, `getservbyname_r()`, and `netdir_getbyname()` are implemented in terms of the same internal library function. This function obtains the system-wide source lookup policy based on the `inet` family entries in `netconfig(4)` and the `services:entry` in `nsswitch.conf(4)`. Similarly, `getservbyport()`, `getservbyport_r()`, and `netdir_getbyaddr()` are implemented in terms of the same internal library function. If the `inet` family entries in `netconfig(4)` have a ```-` in the last column for `nametoaddr` libraries, then the entry for `services` in `nsswitch.conf` will be used; otherwise the `nametoaddr` libraries in that column will be used, and `nsswitch.conf` will not be consulted.

There is no analogue of `getservent()` and `getservent_r()` in the `netdir` functions, so these enumeration functions go straight to the `services` entry in `nsswitch.conf`. Thus enumeration may return results from a different source than that used by `getservbyname()`, `getservbyname_r()`, `getservbyport()`, and `getservbyport_r()`.

When compiling multithreaded applications, see `intro(3)`, `_N_o_t_e_s_O_n_M_u_l_t_i_t_h_r_e_a_d_A_p_p_l_i_c_a_t_i_o_n_s`, for information about the use of the `_REENTRANT` flag.

Use of the enumeration interfaces `getservent()` and `getservent_r()` is discouraged; enumeration may not be supported for all database sources. The semantics of

enumeration are discussed further in `nsswitch.conf(4)`.

Sockets Library Functions `byteorder(3SOCKET)`

NAME

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

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
#include <inttypes.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

These routines convert 16 and 32 bit quantities between network byte order and host byte order. On some architectures these routines are defined as NULL macros in the include file <netinet/in.h>. On other architectures, if their host byte order is different from network byte order, these routines are functional.

These routines are most often used in conjunction with Internet addresses and ports as returned by `gethostent()` and `getservent()`. See `gethostbyname(3NSL)` and `getservbyname(3SOCKET)`.