NAME

getaddrinfo, **freeaddrinfo** - socket address structure to host and service name

SYNOPSIS

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

int
getaddrinfo(*const char *hostname, const char *servname, const struct addrinfo *hints,*
*struct addrinfo **res*);

void
freeaddrinfo(*struct addrinfo *ai*);

DESCRIPTION

The **getaddrinfo**() function is used to get a list of addresses and port numbers for host *hostname* and service *servname*. It is a replacement for and provides more flexibility than the gethostbyname(3) and getservbyname(3) functions.

The *hostname* and *servname* arguments are either pointers to NUL-terminated strings or the null pointer. An acceptable value for *hostname* is either a valid host name or a numeric host address string consisting of a dotted decimal IPv4 address, an IPv6 address, or a UNIX-domain address. The *servname* is either a decimal port number or a service name listed in services(5). At least one of *hostname* and *servname* must be non-null.

hints is an optional pointer to a struct addrinfo, as defined by <*netdb.h*>:

```
struct addrinfo {
        int     ai_flags;       /* AI_PASSIVE, AI_CANONNAME, .. */
        int     ai_family;      /* AF_xxx */
        int     ai_socktype;    /* SOCK_xxx */
        int     ai_protocol;    /* 0 or IPPROTO_xxx for IPv4 and IPv6 */
        socklen_t ai_addrlen;   /* length of ai_addr */
        char    *ai_canonname;  /* canonical name for hostname */
        struct  sockaddr *ai_addr;      /* binary address */
        struct  addrinfo *ai_next;      /* next structure in linked list */
};
```

This structure can be used to provide hints concerning the type of socket that the caller supports or

wishes to use. The caller can supply the following structure elements in *hints*:

ai_family The address family that should be used. When *ai_family* is set to AF_UNSPEC, it means the caller will accept any address family supported by the operating system.

ai_socktype Denotes the type of socket that is wanted: SOCK_STREAM, SOCK_DGRAM, SOCK_SEQPACKET, or SOCK_RAW. When *ai_socktype* is zero the caller will accept any socket type.

ai_protocol Indicates which transport protocol is desired, IPPROTO_UDP, IPPROTO_TCP, IPPROTO_SCTP, or IPPROTO_UDPLITE. If *ai_protocol* is zero the caller will accept any protocol.

ai_flags The *ai_flags* field to which the *hints* parameter points shall be set to zero or be the bitwise-inclusive OR of one or more of the values AI_ADDRCONFIG, AI_ALL, AI_CANONNAME, AI_NUMERICHOST, AI_NUMERICSERV, AI_PASSIVE and AI_V4MAPPED. For a UNIX-domain address, *ai_flags* is ignored.

AI_ADDRCONFIG If the AI_ADDRCONFIG bit is set, IPv4 addresses shall be returned only if an IPv4 address is configured on the local system, and IPv6 addresses shall be returned only if an IPv6 address is configured on the local system.

AI_ALL If the AI_ALL flag is used with the AI_V4MAPPED flag, then **getaddrinfo**() shall return all matching IPv6 and IPv4 addresses.

For example, when using the DNS, queries are made for both AAAA records and A records, and **getaddrinfo**() returns the combined results of both queries. Any IPv4 addresses found are returned as IPv4-mapped IPv6 addresses.

The AI_ALL flag without the AI_V4MAPPED flag is ignored.

AI_CANONNAME If the AI_CANONNAME bit is set, a successful call to **getaddrinfo**() will return a NUL-terminated string containing the canonical name of the specified hostname in the *ai_canonname* element of the first addrinfo structure returned.

AI_NUMERICHOST If the AI_NUMERICHOST bit is set, it indicates that

hostname should be treated as a numeric string defining an IPv4 or IPv6 address and no name resolution should be attempted.

AI_NUMERICSERV — If the AI_NUMERICSERV bit is set, then a non-null *servname* string supplied shall be a numeric port string. Otherwise, an EAI_NONAME error shall be returned. This bit shall prevent any type of name resolution service (for example, NIS+) from being invoked.

AI_PASSIVE — If the AI_PASSIVE bit is set it indicates that the returned socket address structure is intended for use in a call to bind(2). In this case, if the *hostname* argument is the null pointer, then the IP address portion of the socket address structure will be set to INADDR_ANY for an IPv4 address or IN6ADDR_ANY_INIT for an IPv6 address.

If the AI_PASSIVE bit is not set, the returned socket address structure will be ready for use in a call to connect(2) for a connection-oriented protocol or connect(2), sendto(2), or sendmsg(2) if a connectionless protocol was chosen. The IP address portion of the socket address structure will be set to the loopback address if *hostname* is the null pointer and AI_PASSIVE is not set.

AI_V4MAPPED — If the AI_V4MAPPED flag is specified along with an ai_family of AF_INET6, then **getaddrinfo**() shall return IPv4-mapped IPv6 addresses on finding no matching IPv6 addresses (*ai_addrlen* shall be 16).

For example, when using the DNS, if no AAAA records are found then a query is made for A records and any found are returned as IPv4-mapped IPv6 addresses.

The AI_V4MAPPED flag shall be ignored unless *ai_family* equals AF_INET6.

All other elements of the addrinfo structure passed via *hints* must be zero or the null pointer.

If *hints* is the null pointer, **getaddrinfo**() behaves as if the caller provided a struct addrinfo with *ai_family*

set to AF_UNSPEC and all other elements set to zero or NULL.

After a successful call to **getaddrinfo**(), **res* is a pointer to a linked list of one or more addrinfo structures. The list can be traversed by following the *ai_next* pointer in each addrinfo structure until a null pointer is encountered. Each returned addrinfo structure contains three members that are suitable for a call to socket(2): *ai_family*, *ai_socktype*, and *ai_protocol*. For each addrinfo structure in the list, the *ai_addr* member points to a filled-in socket address structure of length *ai_addrlen*.

This implementation of **getaddrinfo**() allows numeric IPv6 address notation with scope identifier, as documented in chapter 11 of RFC 4007. By appending the percent character and scope identifier to addresses, one can fill the sin6_scope_id field for addresses. This would make management of scoped addresses easier and allows cut-and-paste input of scoped addresses.

At this moment the code supports only link-local addresses with the format. The scope identifier is hardcoded to the name of the hardware interface associated with the link (such as ne0). An example is "fe80::1%ne0", which means "fe80::1 on the link associated with the ne0 interface".

The current implementation assumes a one-to-one relationship between the interface and link, which is not necessarily true from the specification.

All of the information returned by **getaddrinfo**() is dynamically allocated: the addrinfo structures themselves as well as the socket address structures and the canonical host name strings included in the addrinfo structures.

Memory allocated for the dynamically allocated structures created by a successful call to **getaddrinfo**() is released by the **freeaddrinfo**() function. The *ai* pointer should be a addrinfo structure created by a call to **getaddrinfo**().

IMPLEMENTATION NOTES

The behavior of freeaddrinfo(NULL) is left unspecified by both Version 4 of the Single UNIX Specification ("SUSv4") and RFC 3493. The current implementation ignores a NULL argument for compatibility with programs that rely on the implementation details of other operating systems.

RETURN VALUES

getaddrinfo() returns zero on success or one of the error codes listed in gai_strerror(3) if an error occurs.

EXAMPLES

The following code tries to connect to "www.kame.net" service "http" via a stream socket. It loops through all the addresses available, regardless of address family. If the destination resolves to an IPv4 address, it will use an AF_INET socket. Similarly, if it resolves to IPv6, an AF_INET6 socket is used.

Observe that there is no hardcoded reference to a particular address family. The code works even if **getaddrinfo**() returns addresses that are not IPv4/v6.

```
struct addrinfo hints, *res, *res0;
int error;
int s;
const char *cause = NULL;

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
error = getaddrinfo("www.kame.net", "http", &hints, &res0);
if (error) {
        errx(1, "%s", gai_strerror(error));
        /* NOTREACHED */
}
s = -1;
for (res = res0; res; res = res->ai_next) {
        s = socket(res->ai_family, res->ai_socktype,
           res->ai_protocol);
        if (s < 0) {
                cause = "socket";
                continue;
        }

        if (connect(s, res->ai_addr, res->ai_addrlen) < 0) {
                cause = "connect";
                close(s);
                s = -1;
                continue;
        }

        break;  /* okay we got one */
}
if (s < 0) {
        err(1, "%s", cause);
        /* NOTREACHED */
}
freeaddrinfo(res0);
```

The following example tries to open a wildcard listening socket onto service "http", for all the address families available.

```
struct addrinfo hints, *res, *res0;
int error;
int s[MAXSOCK];
int nsock;
const char *cause = NULL;

memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
hints.ai_flags = AI_PASSIVE;
error = getaddrinfo(NULL, "http", &hints, &res0);
if (error) {
        errx(1, "%s", gai_strerror(error));
        /* NOTREACHED */
}
nsock = 0;
for (res = res0; res && nsock < MAXSOCK; res = res->ai_next) {
        s[nsock] = socket(res->ai_family, res->ai_socktype,
           res->ai_protocol);
        if (s[nsock] < 0) {
                cause = "socket";
                continue;
        }

        if (bind(s[nsock], res->ai_addr, res->ai_addrlen) < 0) {
                cause = "bind";
                close(s[nsock]);
                continue;
        }
        (void) listen(s[nsock], 5);

        nsock++;
}
if (nsock == 0) {
        err(1, "%s", cause);
        /* NOTREACHED */
}
```

```
freeaddrinfo(res0);
```

SEE ALSO

bind(2), connect(2), send(2), socket(2), gai_strerror(3), gethostbyname(3), getnameinfo(3), getservbyname(3), resolver(3), inet(4), inet6(4), unix(4), hosts(5), resolv.conf(5), services(5), hostname(7)

R. Gilligan, S. Thomson, J. Bound, J. McCann, and W. Stevens, *Basic Socket Interface Extensions for IPv6*, RFC 3493, February 2003.

S. Deering, B. Haberman, T. Jinmei, E. Nordmark, and B. Zill, *IPv6 Scoped Address Architecture*, RFC 4007, March 2005.

Craig Metz, "Protocol Independence Using the Sockets API", *Proceedings of the freenix track: 2000 USENIX annual technical conference*, June 2000.

STANDARDS

The **getaddrinfo**() function is defined by the IEEE Std 1003.1-2004 ("POSIX.1") specification and documented in RFC 3493, "Basic Socket Interface Extensions for IPv6".