NAME

getaddrinfo, freeaddrinfo, gai_strerror – network address and service translation

SYNOPSIS

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

int getaddrinfo(const char *node, const char *service,
        const struct addrinfo *hints,
        struct addrinfo **res);

void freeaddrinfo(struct addrinfo *res);

const char *gai_strerror(int errcode);
```

DESCRIPTION

The **getaddrinfo**(3) function combines the functionality provided by the **getipnodebyname**(3), **getipnodebyaddr**(3), **getservbyname**(3), and **getservbyport**(3) functions into a single interface. The thread-safe **getaddrinfo**(3) function creates one or more socket address structures that can be used by the **bind**(2) and **connect**(2) system calls to create a client or a server socket.

The **getaddrinfo**(3) function is not limited to creating IPv4 socket address structures; IPv6 socket address structures can be created if IPv6 support is available. These socket address structures can be used directly by **bind**(2) or **connect**(2), to prepare a client or a server socket.

The *addrinfo* structure used by this function contains the following members:

```
struct addrinfo {
  int     ai_flags;
  int     ai_family;
  int     ai_socktype;
  int     ai_protocol;
  size_t  ai_addrlen;
  struct sockaddr *ai_addr;
  char    *ai_canonname;
  struct addrinfo *ai_next;
};
```

getaddrinfo(3) sets *res* to point to a dynamically-allocated linked list of *addrinfo* structures, linked by the *ai_next* member. There are several reasons why the linked list may have more than one *addrinfo* structure, including: if the network host is multi-homed; or if the same service is available from multiple socket protocols (one **SOCK_STREAM** address and another **SOCK_DGRAM** address, for example).

The members *ai_family*, *ai_socktype*, and *ai_protocol* have the same meaning as the corresponding parameters in the **socket**(2) system call. The **getaddrinfo**(3) function returns socket addresses in either IPv4 or IPv6 address family, (*ai_family* will be set to either **AF_INET** or **AF_INET6**).

The *hints* parameter specifies the preferred socket type, or protocol. A NULL *hints* specifies that any network address or protocol is acceptable. If this parameter is not NULL it points to an *addrinfo* structure whose *ai_family*, *ai_socktype*, and *ai_protocol* members specify the preferred socket type. **AF_UNSPEC** in *ai_family* specifies any protocol family (either IPv4 or IPv6, for example). 0 in *ai_socktype* or *ai_protocol* specifies that any socket type or protocol is acceptable as well. The *ai_flags* member specifies additional options, defined below. Multiple flags are specified by logically OR-ing them together. All the other members in the *hints* parameter must contain either 0, or a null pointer.

The *node* or *service* parameter, but not both, may be NULL. *node* specifies either a numerical network address (dotted-decimal format for IPv4, hexadecimal format for IPv6) or a network hostname, whose network addresses are looked up and resolved. If *hints.ai_flags* contains the **AI_NUMERICHOST** flag then the *node* parameter must be a numerical network address. The **AI_NUMERICHOST** flag suppresses any potentially lengthy network host address lookups.

The **getaddrinfo**(3) function creates a linked list of *addrinfo* structures, one for each network address subject to any restrictions imposed by the *hints* parameter. The *ai_canonname* field of the first of these *addrinfo* structures is set to point to the official name of the host, if *hints.ai_flags* includes the **AI_CANONNAME** flag. *ai_family*, *ai_socktype*, and *ai_protocol* specify the socket creation parameters. A pointer to the socket address is placed in the *ai_addr* member, and the length of the socket address, in bytes, is placed in the *ai_addrlen* member.

If *node* is NULL, the network address in each socket structure is initialized according to the **AI_PASSIVE** flag, which is set in *hints.ai_flags*. The network address in each socket structure will be left unspecified if **AI_PASSIVE** flag is set. This is used by server applications, which intend to accept client connections on any network address. The network address will be set to the loopback interface address if the **AI_PASSIVE** flag is not set. This is used by client applications, which intend to connect to a server running on the same network host.

If *hints.ai_flags* includes the **AI_ADDRCONFIG** flag, then IPv4 addresses are returned in the list pointed to by *result* only if the local system has at least one IPv4 address configured, and IPv6 addresses are only returned if the local system has at least one IPv6 address configured.

If *hint.ai_flags* specifies the **AI_V4MAPPED** flag, and *hints.ai_family* was specified as **AF_INET6**, and no matching IPv6 addresses could be found, then return IPv4-mapped IPv6 addresses in the list pointed to by *result*. If both **AI_V4MAPPED** and **AI_ALL** are specified in *hints.ai_family*, then return both IPv6 and IPv4-mapped IPv6 addresses in the list pointed to by *result*. **AI_ALL** is ignored if **AI_V4MAPPED** is not also specified.

service sets the port number in the network address of each socket structure. If *service* is NULL the port number will be left uninitialized. If **AI_NUMERICSERV** is specified in *hints.ai_flags* and *service* is not NULL, then *service* must point to a string containing a numeric port number. This flag is used to inhibit the invocation of a name resolution service in cases where it is known not to be required.

The **freeaddrinfo**(3) function frees the memory that was allocated for the dynamically allocated linked list *res*.

RETURN VALUE

getaddrinfo(3) returns 0 if it succeeds, or one of the following non-zero error codes:

EAI_ADDRFAMILY
The specified network host does not have any network addresses in the requested address family.

EAI_AGAIN
The name server returned a temporary failure indication. Try again later.

EAI_BADFLAGS
ai_flags contains invalid flags.

EAI_FAIL
The name server returned a permanent failure indication.

EAI_FAMILY
The requested address family is not supported at all.

EAI_MEMORY
Out of memory.

EAI_NODATA
The specified network host exists, but does not have any network addresses defined.

EAI_NONAME
The *node* or *service* is not known; or both *node* and *service* are NULL; or **AI_NUMERICSERV** was specified in *hints.ai_flags* and *service* was not a numeric port-number string.

EAI_SERVICE
The requested service is not available for the requested socket type. It may be available through another socket type.

EAI_SOCKTYPE
The requested socket type is not supported at all.

EAI_SYSTEM
Other system error, check *errno* for details.

The **gai_strerror**(3) function translates these error codes to a human readable string, suitable for error reporting.

CONFORMING TO

POSIX.1-2001. The **getaddrinfo**() function is documented in RFC 2553.

NOTES

AI_ADDRCONFIG, **AI_ALL**, and **AI_V4MAPPED** are available since glibc 2.3.3. **AI_NUMERICSERV** is available since glibc 2.3.4.

SEE ALSO

getipnodebyaddr(3), **getipnodebyname**(3)