

**NAME**

getaddrinfo, getnameinfo, freeaddrinfo, gai\_strerror – translate between node name and address

**SYNOPSIS**

```
cc flag ... file ... -lsocket -lnsl library ...
```

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

```
#include <netdb.h>
```

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

```
int getnameinfo(const struct sockaddr *sa, socklen_t salen, char *host,
               size_t hostlen, char *serv, size_t servlen, int flags);
```

```
void freeaddrinfo(struct addrinfo *ai);
```

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

**DESCRIPTION**

These functions perform translations from node name to address and from address to node name in a protocol-independent manner.

The **getaddrinfo()** function performs the node name to address translation. The *nodename* and *servname* arguments are pointers to null-terminated strings or *NULL*. One or both of these arguments must be a non-null pointer. In the normal client scenario, both the *nodename* and *servname* are specified. In the normal server scenario, only the *servname* is specified.

A non-null *nodename* string can be a node name or a numeric host address string. The *nodename* can also be an IPv6 zone-id in the form:

```
<address>%<zone-id>
```

The address is the literal IPv6 link-local address or host name of the destination. The zone-id is the interface ID of the IPv6 link used to send the packet. The zone-id can either be a numeric value, indicating a literal zone value, or an interface name such as **hme0**.

A non-null *servname* string can be either a service name or a decimal port number.

The caller can optionally pass an **addrinfo** structure, pointed to by the *hints* argument, to provide hints concerning the type of socket that the caller supports.

The **addrinfo** structure is defined as:

```
struct addrinfo {
int      ai_flags;      /* AI_PASSIVE, AI_CANONNAME,
                        AI_NUMERICHOST, AI_NUMERICSERV
                        AI_V4MAPPED, AI_ALL, AI_ADDRCONFIG */
int      ai_family;    /* PF_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 nodename */
struct sockaddr *ai_addr; /* binary address */
struct addrinfo *ai_next; /* next structure in linked list */
};
```

In this *hints* structure, all members other than **ai\_flags**, **ai\_family**, **ai\_socktype**, and **ai\_protocol** must be 0 or a null pointer. A value of **PF\_UNSPEC** for **ai\_family** indicates that the caller will accept any protocol family. A value of 0 for **ai\_socktype** indicates that the caller will accept any socket type. A value of 0 for **ai\_protocol** indicates that the caller will accept any protocol. For example, if the caller handles only TCP and not UDP, then the **ai\_socktype** member of the *hints* structure should be set to **SOCK\_STREAM** when **getaddrinfo()** is called. If the caller handles only IPv4 and not IPv6, then the **ai\_family** member of the *hints* structure should be set to **PF\_INET** when **getaddrinfo()** is called. If the third argument to **getaddrinfo()** is a null pointer, it is as if the caller had filled in an **addrinfo** structure initialized to 0 with **ai\_family** set to **PF\_UNSPEC**.

Upon success, a pointer to a linked list of one or more **addrinfo** structures is returned through the final argument. The caller can process each **addrinfo** structure in this list by following the **ai\_next** pointer, until a null pointer is encountered. In each returned **addrinfo** structure the three members **ai\_family**, **ai\_socktype**, and **ai\_protocol** are the corresponding arguments for a call to the **socket(3SOCKET)** function. In each **addrinfo** structure the **ai\_addr** member points to a filled-in socket address structure whose length is specified by the **ai\_addrlen** member.

If the **AI\_PASSIVE** bit is set in the **ai\_flags** member of the *hints* structure, the caller plans to use the returned socket address structure in a call to **bind(3SOCKET)**. In this case, if the *nodename* argument is a null pointer, 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 in the **ai\_flags** member of the *hints* structure, then the returned socket address structure will be ready for a call to **connect(3SOCKET)** (for a connection-oriented protocol) or either **connect(3SOCKET)**, **sendto(3SOCKET)**, or **sendmsg(3SOCKET)** (for a connectionless protocol). If the *nodename* argument is a null pointer, the IP address portion of the socket address structure will be set to the loopback address.

If the **AI\_CANONNAME** bit is set in the **ai\_flags** member of the *hints* structure, then upon successful return the **ai\_canonname** member of the first **addrinfo** structure in the linked list will point to a null-terminated string containing the canonical name of the specified *nodename*.

If the **AI\_NUMERICHOST** bit is set in the **ai\_flags** member of the *hints* structure, then a non-null *nodename* string must be a numeric host address string. Otherwise an error of **EAI\_NONAME** is returned. This flag prevents any type of name resolution service (such as DNS) from being called.

If the **AI\_NUMERICSERV** flag is specified, then a non-null servname string supplied shall be a numeric port string. Otherwise, an **[EAI\_NONAME]** error is returned. This flag prevents any type of name resolution service (for example, NIS+) from being invoked.

If the **AI\_V4MAPPED** flag is specified along with an **ai\_family** of **AF\_INET6**, then **getaddrinfo()** returns IPv4-mapped IPv6 addresses on finding no matching IPv6 addresses (**ai\_addrlen** shall be 16). For example, if no AAAA records are found when using DNS, a query is made for A records. Any found records are returned as IPv4-mapped IPv6 addresses.

The **AI\_V4MAPPED** flag is ignored unless **ai\_family** equals **AF\_INET6**.

If the **AI\_ALL** flag is used with the **AI\_V4MAPPED** flag, then **getaddrinfo()** returns 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.

When **ai\_family** is not specified (**AF\_UNSPEC**), **AI\_V4MAPPED** and **AI\_ALL** flags are used only if **AF\_INET6** is supported.

If the **AI\_ADDRCONFIG** flag is specified, IPv4 addresses are returned only if an IPv4 address is configured on the local system, and IPv6 addresses are returned only if an IPv6 address is configured on the local system. For this case, the loopback address is not considered to be as valid as a configured address. For example, when using the DNS, a query for AAAA records should occur only if the node

has at least one IPv6 address configured (other than IPv6 loopback) and a query for A records should occur only if the node has at least one IPv4 address configured (other than the IPv4 loopback).

All of the information returned by **getaddrinfo()** is dynamically allocated: the **addrinfo** structures as well as the socket address structures and canonical node name strings pointed to by the **addrinfo** structures. The **freeaddrinfo()** function is called to return this information to the system the function. For **freeaddrinfo()**, the **addrinfo** structure pointed to by the *ai* argument is freed, along with any dynamic storage pointed to by the structure. This operation is repeated until a null **ai\_next** pointer is encountered.

To aid applications in printing error messages based on the **EAI\_\*** codes returned by **getaddrinfo()**, the **gai\_strerror()** is defined. The argument is one of the **EAI\_\*** values defined below and the return value points to a string describing the error. If the argument is not one of the **EAI\_\*** values, the function still returns a pointer to a string whose contents indicate an unknown error.

The **getnameinfo()** function looks up an IP address and port number provided by the caller in the name service database and system-specific database, and returns text strings for both in buffers provided by the caller. The function indicates successful completion by a 0 return value; a non-zero return value indicates failure.

The first argument, *sa*, points to either a **sockaddr\_in** structure (for IPv4) or a **sockaddr\_in6** structure (for IPv6) that holds the IP address and port number. The *salen* argument gives the length of the **sockaddr\_in** or **sockaddr\_in6** structure.

The function returns the node name associated with the IP address in the buffer pointed to by the *host* argument.

The function can also return the IPv6 zone-id in the form:

```
<address>%<zone-id>
```

The caller provides the size of this buffer with the *hostlen* argument. The service name associated with the port number is returned in the buffer pointed to by *serv*, and the *servlen* argument gives the length of this buffer. The caller specifies not to return either string by providing a 0 value for the *hostlen* or *servlen* arguments. Otherwise, the caller must provide buffers large enough to hold the node name and the service name, including the terminating null characters.

To aid the application in allocating buffers for these two returned strings, the following constants are defined in **<netdb.h>**:

```
#define NI_MAXHOST 1025
#define NI_MAXSERV 32
```

The final argument is a flag that changes the default actions of this function. By default, the fully-qualified domain name (**FQDN**) for the host is looked up in the name service database and returned. If the flag bit **NI\_NOFQDN** is set, only the node name portion of the **FQDN** is returned for local hosts.

If the flag bit **NI\_NUMERICHOST** is set, or if the host's name cannot be located in the name service, the numeric form of the host's address is returned instead of its name, for example, by calling **inet\_ntop()** (see **inet(3SOCKET)**) instead of **getipnodebyname(3SOCKET)**. If the flag bit **NI\_NAMEREQD** is set, an error is returned if the host's name cannot be located in the name service database.

If the flag bit **NI\_NUMERICSERV** is set, the numeric form of the service address is returned (for example, its port number) instead of its name. The two **NI\_NUMERIC\*** flags are required to support the **-n** flag that many commands provide.

A fifth flag bit, **NI\_DGRAM**, specifies that the service is a datagram service, and causes **getservbyport(3SOCKET)** to be called with a second argument of **udp** instead of the default **tcp**. This is required for the few ports (for example, 512-514) that have different services for UDP and TCP.

These **NI\_\*** flags are defined in `<netdb.h>` along with the **AI\_\*** flags already defined for **getaddrinfo()**.

#### RETURN VALUES

For **getaddrinfo()**, if the query is successful, a pointer to a linked list of one or more **addrinfo** structures is returned by the fourth argument and the function returns **0**. The order of the addresses returned in the fourth argument is discussed in the ADDRESS ORDERING section. If the query fails, a non-zero error code will be returned. For **getnameinfo()**, if successful, the strings *hostname* and *service* are copied into *host* and *serv*, respectively. If unsuccessful, zero values for either *hostlen* or *servlen* will suppress the associated lookup; in this case no data is copied into the applicable buffer. If **gai\_strerror()** is successful, a pointer to a string containing an error message appropriate for the **EAI\_\*** errors is returned. If *errcode* is not one of the **EAI\_\*** values, a pointer to a string indicating an unknown error is returned.

#### Address Ordering

AF\_INET6 addresses returned by the fourth argument of **getaddrinfo()** are ordered according to the algorithm described in *RFC 3484, Default Address Selection for Internet Protocol version 6 (IPv6)*. The addresses are ordered using a list of pair-wise comparison rules which are applied in order. If a rule determines that one address is better than another, the remaining rules are irrelevant to the comparison of those two addresses. If two addresses are equivalent according to one rule, the remaining rules act as a tie-breaker. The address ordering list of pair-wise comparison rules follow below:

```
tab() box; lw(2.75i) |lw(2.75i) lw(2.75i) |lw(2.75i) Avoid unusable destinations.T{ Prefer a destination
that is reachable through the IP routing table. T} _ Prefer matching scope.T{ Prefer a destination whose
scope is equal to the scope of its source address. See inet6(7P) for the definition of scope used by this
rule. T} _ Avoid link-local source.T{ Avoid selecting a link-local source address when the destination
address is not a link-local address. T} _ Avoid deprecated addresses.T{ Prefer a destination that is not
deprecated (IFF_DEPRECATED). T} _ T{ Prefer matching label. This rule uses labels that are
obtained through the IPv6 default address selection policy table. See ipaddrsel(1M) for a description of
the default contents of the table and how the table is configured. T}T{ Prefer a destination whose label
is equal to the label of its source address. T} _ T{ Prefer higher precedence. This rule uses precedence
values that are obtained through the IPv6 default address selection policy table. See ipaddrsel(1M) for a
description of the default contents of the table and how the table is configured. T}T{ Prefer the destina-
tion whose precedence is higher than the other destination. T} _ Prefer native transport.T{ Prefer a des-
tination if the interface that is used for sending packets to that destination is not an IP over IP tunnel.
T} _ T{ Prefer smaller scope. See inet6(7P) for the definition of this rule. T}T{ Prefer the destination
whose scope is smaller than the other destination. T} _ Use longest matching prefix.T{ When the two
destinations belong to the same address family, prefer the destination that has the longer matching prefix
with its source address. T}
```

#### ERRORS

The following names are the error values returned by **getaddrinfo()** and are defined in `<netdb.h>`:

**EAI\_ADDRFAMILY** Address family for nodename is not supported.

**EAI\_AGAIN** Temporary failure in name resolution has occurred .

<b>EAI_BADFLAGS</b>	Invalid value specified for <b>ai_flags</b> .
<b>EAI_FAIL</b>	Non-recoverable failure in name resolution has occurred.
<b>EAI_FAMILY</b>	The <b>ai_family</b> is not supported.
<b>EAI_MEMORY</b>	Memory allocation failure has occurred.
<b>EAI_NODATA</b>	No address is associated with <i>nodename</i> .
<b>EAI_NONAME</b>	Neither <i>nodename</i> nor <i>servname</i> is provided or known.
<b>EAI_SERVICE</b>	The <i>servname</i> is not supported for <b>ai_socktype</b> .
<b>EAI_SOCKTYPE</b>	The <b>ai_socktype</b> is not supported.
<b>EAI_OVERFLOW</b>	Argument buffer has overflowed.
<b>EAI_SYSTEM</b>	System error was returned in <i>errno</i> .

**FILES**

<b>/etc/inet/hosts</b>	local database that associates names of nodes with IP addresses
<b>/etc/netconfig</b>	network configuration database
<b>/etc/nsswitch.conf</b>	configuration file for the name service switch

**ATTRIBUTES**

See **attributes(5)** for description of the following attributes:

```
tab() box; cw(2.75i) |cw(2.75i) lw(2.75i) |lw(2.75i) ATTRIBUTE TYPEATTRIBUTE VALUE _ MT-
LevelMT-Safe with exceptions
```

**SEE ALSO**

**ipaddrsel(1M)**, **gethostbyname(3NSL)**, **getipnodebyname(3SOCKET)**, **htonl(3SOCKET)**,  
**inet(3SOCKET)**, **netdb.h(3HEAD)**, **socket(3SOCKET)**, **hosts(4)**, **nsswitch.conf(4)**, **inet6(7P)**

Draves, R. *RFC 3484, Default Address Selection for Internet Protocol version 6 (IPv6)*. Network Working Group. February 2003.