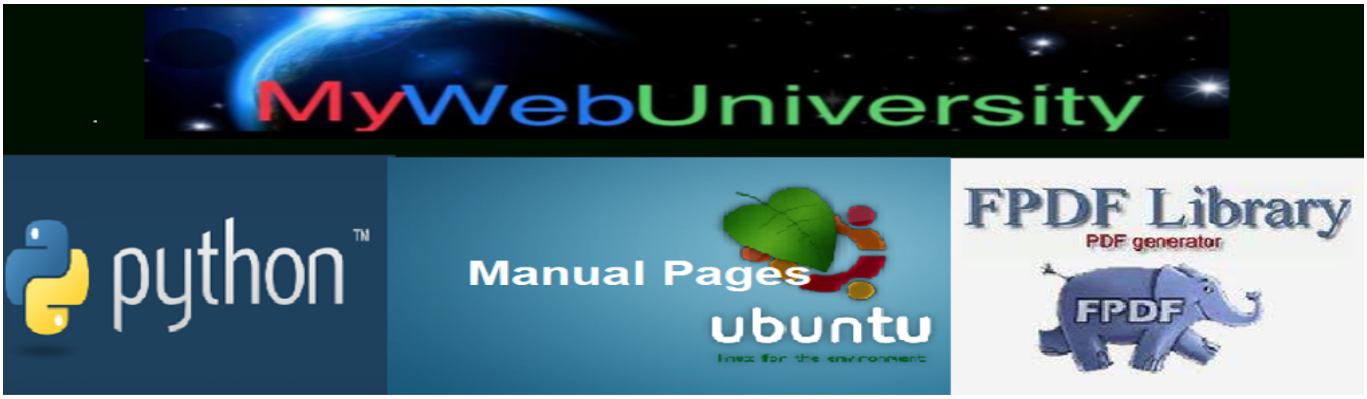

Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'gai_strerror.3'

$ man gai_strerror.3

GETADDRINFO(3) Linux Programmer's Manual GETADDRINFO(3)

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);
```

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

getaddrinfo(), freeaddrinfo(), gai_strerror():

Since glibc 2.22: _POSIX_C_SOURCE >= 200112L

Glibc 2.21 and earlier: _POSIX_C_SOURCE

DESCRIPTION

Given node and service, which identify an Internet host and a service, getaddrinfo() re? turns one or more addrinfo structures, each of which contains an Internet address that can be specified in a call to bind(2) or connect(2). The getaddrinfo() function combines the functionality provided by the gethostbyname(3) and getservbyname(3) functions into a sin? gle interface, but unlike the latter functions, getaddrinfo() is reentrant and allows pro? grams to eliminate IPv4-versus-IPv6 dependencies.

The addrinfo structure used by getaddrinfo() contains the following fields:

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

The hints argument points to an addrinfo structure that specifies criteria for selecting the socket address structures returned in the list pointed to by res. If hints is not NULL it points to an addrinfo structure whose ai_family, ai_socktype, and ai_protocol specify criteria that limit the set of socket addresses returned by getaddrinfo(), as fol? lows:

ai_family

This field specifies the desired address family for the returned addresses. Valid values for this field include AF_INET and AF_INET6. The value AF_UNSPEC indicates that getaddrinfo() should return socket addresses for any address family (either IPv4 or IPv6, for example) that can be used with node and service.

ai_socktype

This field specifies the preferred socket type, for example SOCK_STREAM or SOCK_DGRAM. Specifying 0 in this field indicates that socket addresses of any type can be returned by getaddrinfo().

ai_protocol

This field specifies the protocol for the returned socket addresses. Specifying 0 in this field indicates that socket addresses with any protocol can be returned by getaddrinfo().

ai_flags

This field specifies additional options, described below. Multiple flags are spec? ified by bitwise OR-ing them together.

All the other fields in the structure pointed to by hints must contain either 0 or a null

pointer, as appropriate.

Specifying hints as NULL is equivalent to setting ai_socktype and ai_protocol to 0; ai_family to AF_UNSPEC; and ai_flags to (AI_V4MAPPED | AI_ADDRCONFIG). (POSIX specifies different defaults for ai_flags; see NOTES.) node specifies either a numerical network address (for IPv4, numbers-and-dots notation as supported by inet_aton(3); for IPv6, hexa? decimal string format as supported by inet_pton(3)), or a network hostname, whose network addresses are looked up and resolved. If hints.ai_flags contains the AI_NUMERICHOST flag, then node must be a numerical network address. The AI_NUMERICHOST flag suppresses any po? tentially lengthy network host address lookups.

If the AI_PASSIVE flag is specified in hints.ai_flags, and node is NULL, then the returned socket addresses will be suitable for bind(2)ing a socket that will accept(2) connections. The returned socket address will contain the "wildcard address" (INADDR_ANY for IPv4 ad? dresses, IN6ADDR_ANY_INIT for IPv6 address). The wildcard address is used by applications (typically servers) that intend to accept connections on any of the host's network ad? dresses. If node is not NULL, then the AI_PASSIVE flag is ignored.

If the AI_PASSIVE flag is not set in hints.ai_flags, then the returned socket addresses will be suitable for use with connect(2), sendto(2), or sendmsg(2). If node is NULL, then the network address will be set to the loopback interface address (INADDR_LOOPBACK for IPv4 addresses, IN6ADDR_LOOPBACK_INIT for IPv6 address); this is used by applications that intend to communicate with peers running on the same host.

service sets the port in each returned address structure. If this argument is a service name (see services(5)), it is translated to the corresponding port number. This argument can also be specified as a decimal number, which is simply converted to binary. If ser? vice is NULL, then the port number of the returned socket addresses will be left unini? tialized. 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 in? hibit the invocation of a name resolution service in cases where it is known not to be re? quired.

Either node or service, but not both, may be NULL.

The getaddrinfo() function allocates and initializes a linked list of addrinfo structures, one for each network address that matches node and service, subject to any restrictions imposed by hints, and returns a pointer to the start of the list in res. The items in the linked list are linked by the ai_next field.

There are several reasons why the linked list may have more than one addrinfo structure, including: the network host is multihomed, accessible over multiple protocols (e.g., both AF_INET and AF_INET6); or the same service is available from multiple socket types (one SOCK_STREAM address and another SOCK_DGRAM address, for example). Normally, the applica? tion should try using the addresses in the order in which they are returned. The sorting function used within getaddrinfo() is defined in RFC 3484; the order can be tweaked for a particular system by editing /etc/gai.conf (available since glibc 2.5).

If hints.ai_flags includes the AI_CANONNAME flag, then the ai_canonname field of the first of the addrinfo structures in the returned list is set to point to the official name of the host.

The remaining fields of each returned addrinfo structure are initialized as follows:

* The ai_family, ai_socktype, and ai_protocol fields return the socket creation parameters (i.e., these fields have the same meaning as the corresponding arguments of socket(2)). For example, ai_family might return AF_INET or AF_INET6; ai_socktype might return SOCK_DGRAM or SOCK_STREAM; and ai_protocol returns the protocol for the socket.
* A pointer to the socket address is placed in the ai_addr field, and the length of the socket address, in bytes, is placed in the ai_addrlen field.

If hints.ai_flags includes the AI_ADDRCONFIG flag, then IPv4 addresses are returned in the list pointed to by res only if the local system has at least one IPv4 address configured, and IPv6 addresses are returned only if the local system has at least one IPv6 address configured. The loopback address is not considered for this case as valid as a configured address. This flag is useful on, for example, IPv4-only systems, to ensure that getad? drinfo() does not return IPv6 socket addresses that would always fail in connect(2) or bind(2).

If hints.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 ad? dresses in the list pointed to by res. If both AI_V4MAPPED and AI_ALL are specified in hints.ai_flags, then return both IPv6 and IPv4-mapped IPv6 addresses in the list pointed to by res. AI_ALL is ignored if AI_V4MAPPED is not also specified.

The freeaddrinfo() function frees the memory that was allocated for the dynamically allo? cated linked list res.

Extensions to getaddrinfo() for Internationalized Domain Names

Starting with glibc 2.3.4, getaddrinfo() has been extended to selectively allow the incom?

ing and outgoing hostnames to be transparently converted to and from the Internationalized Domain Name (IDN) format (see RFC 3490, Internationalizing Domain Names in Applications (IDNA)). Four new flags are defined:

AI_IDN If this flag is specified, then the node name given in node is converted to IDN format if necessary. The source encoding is that of the current locale.

If the input name contains non-ASCII characters, then the IDN encoding is used. Those parts of the node name (delimited by dots) that contain non-ASCII characters are encoded using ASCII Compatible Encoding (ACE) before being passed to the name resolution functions.

AI_CANONIDN

After a successful name lookup, and if the AI_CANONNAME flag was specified, getad? drinfo() will return the canonical name of the node corresponding to the addrinfo structure value passed back. The return value is an exact copy of the value re? turned by the name resolution function.

If the name is encoded using ACE, then it will contain the xn-- prefix for one or more components of the name. To convert these components into a readable form the AI_CANONIDN flag can be passed in addition to AI_CANONNAME. The resulting string is encoded using the current locale's encoding.

AI_IDN_ALLOW_UNASSIGNED, AI_IDN_USE_STD3_ASCII_RULES

Setting these flags will enable the IDNA_ALLOW_UNASSIGNED (allow unassigned Unicode code points) and IDNA_USE_STD3_ASCII_RULES (check output to make sure it is a STD3 conforming hostname) flags respectively to be used in the IDNA handling.

RETURN VALUE

getaddrinfo() returns 0 if it succeeds, or one of the following nonzero error codes:

EAI_ADDRFAMILY

The specified network host does not have any network addresses in the requested ad? dress family.

EAI_AGAIN

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

EAI_BADFLAGS

hints.ai_flags contains invalid flags; or, hints.ai_flags included AI_CANONNAME and name was NULL.

EAI_FAIL

The name server returned a permanent failure indication.

EAI_FAMILY

The requested address family is not supported.

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_NUMERIC?
SERV 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. For example, this error could occur if ser?
vice was "shell" (a service available only on stream sockets), and either
hints.ai_protocol was IPPROTO_UDP, or hints.ai_socktype was SOCK_DGRAM; or the er?
ror could occur if service was not NULL, and hints.ai_socktype was SOCK_RAW (a
socket type that does not support the concept of services).

EAI_SOCKTYPE

The requested socket type is not supported. This could occur, for example, if
hints.ai_socktype and hints.ai_protocol are inconsistent (e.g., SOCK_DGRAM and IP?
PROTO_TCP, respectively).

EAI_SYSTEM

Other system error, check errno for details.

The gai_strerror() function translates these error codes to a human readable string, suit?
able for error reporting.

FILES

/etc/gai.conf

ATTRIBUTES

For an explanation of the terms used in this section, see attributes(7).

??

?Interface ? Attribute ? Value ?

??

?getaddrinfo() ? Thread safety ? MT-Safe env locale ?

??

?freeaddrinfo(), ? Thread safety ? MT-Safe ?

?gai_strerror() ? ? ?

??

CONFORMING TO

POSIX.1-2001, POSIX.1-2008. The getaddrinfo() function is documented in RFC 2553.

NOTES

getaddrinfo() supports the address%scope-id notation for specifying the IPv6 scope-ID.

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

According to POSIX.1, specifying hints as NULL should cause ai_flags to be assumed as 0.

The GNU C library instead assumes a value of (AI_V4MAPPED | AI_ADDRCONFIG) for this case, since this value is considered an improvement on the specification.

EXAMPLES

The following programs demonstrate the use of getaddrinfo(), gai_strerror(), freead? drinfo(), and getnameinfo(3). The programs are an echo server and client for UDP data? grams.

Server program

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>
#define BUF_SIZE 500
int
main(int argc, char *argv[])
{
    struct addrinfo hints;
    struct addrinfo *result, *rp;
    int sfd, s;
```

```
struct sockaddr_storage peer_addr;
socklen_t peer_addr_len;
ssize_t nread;
char buf[BUF_SIZE];
if (argc != 2) {
    fprintf(stderr, "Usage: %s port\n", argv[0]);
    exit(EXIT_FAILURE);
}
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC;    /* Allow IPv4 or IPv6 */
hints.ai_socktype = SOCK_DGRAM; /* Datagram socket */
hints.ai_flags = AI_PASSIVE;    /* For wildcard IP address */
hints.ai_protocol = 0;          /* Any protocol */
hints.ai_canonname = NULL;
hints.ai_addr = NULL;
hints.ai_next = NULL;
s = getaddrinfo(NULL, argv[1], &hints, &result);
if (s != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
    exit(EXIT_FAILURE);
}
/* getaddrinfo() returns a list of address structures.
   Try each address until we successfully bind(2).
   If socket(2) (or bind(2)) fails, we (close the socket
   and) try the next address. */
for (rp = result; rp != NULL; rp = rp->ai_next) {
    sfd = socket(rp->ai_family, rp->ai_socktype,
            rp->ai_protocol);
    if (sfd == -1)
        continue;
    if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
        break;                  /* Success */
    close(sfd);
```

```
        }
        freeaddrinfo(result);           /* No longer needed */
        if (rp == NULL) {               /* No address succeeded */
            fprintf(stderr, "Could not bind\n");
            exit(EXIT_FAILURE);
        }
        /* Read datagrams and echo them back to sender */
        for (;;) {
            peer_addr_len = sizeof(peer_addr);
            nread = recvfrom(sfd, buf, BUF_SIZE, 0,
                    (struct sockaddr *) &peer_addr, &peer_addr_len);
            if (nread == -1)
                continue;               /* Ignore failed request */
            char host[NI_MAXHOST], service[NI_MAXSERV];
            s = getnameinfo((struct sockaddr *) &peer_addr,
                            peer_addr_len, host, NI_MAXHOST,
                            service, NI_MAXSERV, NI_NUMERICSERV);
            if (s == 0)
                printf("Received %zd bytes from %s:%s\n",
                        nread, host, service);
            else
                fprintf(stderr, "getnameinfo: %s\n", gai_strerror(s));
            if (sendto(sfd, buf, nread, 0,
                        (struct sockaddr *) &peer_addr,
                        peer_addr_len) != nread)
                fprintf(stderr, "Error sending response\n");
        }
    }
```

Client program

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

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#define BUF_SIZE 500
int
main(int argc, char *argv[])
{
    struct addrinfo hints;
    struct addrinfo *result, *rp;
    int sfd, s;
    size_t len;
    ssize_t nread;
    char buf[BUF_SIZE];
    if (argc < 3) {
        fprintf(stderr, "Usage: %s host port msg...\n", argv[0]);
        exit(EXIT_FAILURE);
    }
    /* Obtain address(es) matching host/port */
    memset(&hints, 0, sizeof(hints));
    hints.ai_family = AF_UNSPEC;    /* Allow IPv4 or IPv6 */
    hints.ai_socktype = SOCK_DGRAM; /* Datagram socket */
    hints.ai_flags = 0;
    hints.ai_protocol = 0;          /* Any protocol */
    s = getaddrinfo(argv[1], argv[2], &hints, &result);
    if (s != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
        exit(EXIT_FAILURE);
    }
    /* getaddrinfo() returns a list of address structures.
       Try each address until we successfully connect(2).
       If socket(2) (or connect(2)) fails, we (close the socket
       and) try the next address. */
    for (rp = result; rp != NULL; rp = rp->ai_next) {
```

```
    sfd = socket(rp->ai_family, rp->ai_socktype,
                 rp->ai_protocol);
    if (sfd == -1)
        continue;
    if (connect(sfd, rp->ai_addr, rp->ai_addrlen) != -1)
        break;                  /* Success */
    close(sfd);
}
freeaddrinfo(result);           /* No longer needed */
if (rp == NULL) {               /* No address succeeded */
    fprintf(stderr, "Could not connect\n");
    exit(EXIT_FAILURE);
}
/* Send remaining command-line arguments as separate
   datagrams, and read responses from server */
for (int j = 3; j < argc; j++) {
    len = strlen(argv[j]) + 1;
            /* +1 for terminating null byte */
    if (len > BUF_SIZE) {
        fprintf(stderr,
                "Ignoring long message in argument %d\n", j);
        continue;
    }
    if (write(sfd, argv[j], len) != len) {
        fprintf(stderr, "partial/failed write\n");
        exit(EXIT_FAILURE);
    }
    nread = read(sfd, buf, BUF_SIZE);
    if (nread == -1) {
        perror("read");
        exit(EXIT_FAILURE);
    }
    printf("Received %zd bytes: %s\n", nread, buf);
```

```
        }
        exit(EXIT_SUCCESS);
    }
```

SEE ALSO

getaddrinfo_a(3), gethostbyname(3), getnameinfo(3), inet(3), gai.conf(5), hostname(7), ip(7)

COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man-pages/.