



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

\$ man gai_cancel.3

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

NAME

getaddrinfo_a, gai_suspend, gai_error, gai_cancel - asynchronous net?
work address and service translation

SYNOPSIS

```
#define _GNU_SOURCE      /* See feature_test_macros(7) */

#include <netdb.h>

int getaddrinfo_a(int mode, struct gaicb *list[],
                  int nitems, struct sigevent *sevp);

int gai_suspend(const struct gaicb * const list[], int nitems,
               const struct timespec *timeout);

int gai_error(struct gaicb *req);

int gai_cancel(struct gaicb *req);

Link with -lanl.
```

DESCRIPTION

The getaddrinfo_a() function performs the same task as getaddrinfo(3), but allows multiple name look-ups to be performed asynchronously, with optional notification on completion of look-up operations.

The mode argument has one of the following values:

GAI_WAIT

Perform the look-ups synchronously. The call blocks until the look-ups have completed.

GAI_NOWAIT

Perform the look-ups asynchronously. The call returns immediately, and the requests are resolved in the background. See the discussion of the sevp argument below.

The array list specifies the look-up requests to process. The nitems argument specifies the number of elements in list. The requested look-up operations are started in parallel. NULL elements in list are ignored. Each request is described by a gaicb structure, defined as follows:

```
struct gaicb {
    const char      *ar_name;
    const char      *ar_service;
    const struct addrinfo *ar_request;
    struct addrinfo  *ar_result;
};
```

The elements of this structure correspond to the arguments of getaddrinfo(3). Thus, ar_name corresponds to the node argument and ar_service to the service argument, identifying an Internet host and a service. The ar_request element corresponds to the hints argument, specifying the criteria for selecting the returned socket address structures. Finally, ar_result corresponds to the res argument; you do not need to initialize this element, it will be automatically set when the request is resolved. The addrinfo structure referenced by the last two elements is described in getaddrinfo(3).

When mode is specified as GAI_NOWAIT, notifications about resolved requests can be obtained by employing the sigevent structure pointed to by the sevp argument. For the definition and general details of this structure, see sigevent(7). The sevp->sigev_notify field can have the following values:

SIGEV_NONE

Don't provide any notification.

SIGEV_SIGNAL

When a look-up completes, generate the signal `sigev_signo` for the process. See `sigevent(7)` for general details. The `si_code` field of the `siginfo_t` structure will be set to `SI_ASYNCNL`.

SIGEV_THREAD

When a look-up completes, invoke `sigev_notify_function` as if it were the start function of a new thread. See `sigevent(7)` for details.

For `SIGEV_SIGNAL` and `SIGEV_THREAD`, it may be useful to point `sevp->sigev_value.sival_ptr` to list.

The `gai_suspend()` function suspends execution of the calling thread, waiting for the completion of one or more requests in the array list.

The `nitems` argument specifies the size of the array list. The call blocks until one of the following occurs:

- * One or more of the operations in list completes.
- * The call is interrupted by a signal that is caught.
- * The time interval specified in `timeout` elapses. This argument specifies a timeout in seconds plus nanoseconds (see `nanosleep(2)` for details of the `timespec` structure). If `timeout` is `NULL`, then the call blocks indefinitely (until one of the events above occurs).

No explicit indication of which request was completed is given; you must determine which request(s) have completed by iterating with `gai_error()` over the list of requests.

The `gai_error()` function returns the status of the request `req`: either `EAI_INPROGRESS` if the request was not completed yet, 0 if it was handled successfully, or an error code if the request could not be resolved.

The `gai_cancel()` function cancels the request `req`. If the request has been canceled successfully, the error status of the request will be set to `EAI_CANCELED` and normal asynchronous notification will be performed.

The request cannot be canceled if it is currently being processed; in

that case, it will be handled as if `gai_cancel()` has never been called.

If `req` is `NULL`, an attempt is made to cancel all outstanding requests that the process has made.

RETURN VALUE

The `getaddrinfo_a()` function returns 0 if all of the requests have been enqueued successfully, or one of the following nonzero error codes:

EAI_AGAIN

The resources necessary to enqueue the look-up requests were not available. The application may check the error status of each request to determine which ones failed.

EAI_MEMORY

Out of memory.

EAI_SYSTEM

mode is invalid.

The `gai_suspend()` function returns 0 if at least one of the listed requests has been completed. Otherwise, it returns one of the following nonzero error codes:

EAI_AGAIN

The given timeout expired before any of the requests could be completed.

EAI_ALLDONE

There were no actual requests given to the function.

EAI_INTR

A signal has interrupted the function. Note that this interruption might have been caused by signal notification of some completed look-up request.

The `gai_error()` function can return `EAI_INPROGRESS` for an unfinished look-up request, 0 for a successfully completed look-up (as described above), one of the error codes that could be returned by `getaddrinfo(3)`, or the error code `EAI_CANCELED` if the request has been canceled explicitly before it could be finished.

The `gai_cancel()` function can return one of these values:

EAI_CANCELED

The request has been canceled successfully.

EAI_NOTCANCELED

The request has not been canceled.

EAI_ALLDONE

The request has already completed.

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

ATTRIBUTES

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

??

?Interface ? Attribute ? Value ?

??

?getaddrinfo_a(), gai_suspend(), ? Thread safety ? MT-Safe ?

?gai_error(), gai_cancel() ? ? ?

??

CONFORMING TO

These functions are GNU extensions; they first appeared in glibc in version 2.2.3.

NOTES

The interface of `getaddrinfo_a()` was modeled after the `lio_listio(3)` interface.

EXAMPLES

Two examples are provided: a simple example that resolves several requests in parallel synchronously, and a complex example showing some of the asynchronous capabilities.

Synchronous example

The program below simply resolves several hostnames in parallel, giving a speed-up compared to resolving the hostnames sequentially using `getaddrinfo(3)`. The program might be used like this:

```
$ ./a.out ftp.us.kernel.org enoent.linuxfoundation.org gnu.cz
```

```
ftp.us.kernel.org: 128.30.2.36
```

```
enoent.linuxfoundation.org: Name or service not known
```

gnu.cz: 87.236.197.13

Here is the program source code

```
#define _GNU_SOURCE

#include <netdb.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int

main(int argc, char *argv[])

{

    int ret;

    struct gaicb *reqs[argc - 1];

    char host[NI_MAXHOST];

    struct addrinfo *res;

    if (argc < 2) {

        fprintf(stderr, "Usage: %s HOST...\n", argv[0]);

        exit(EXIT_FAILURE);

    }

    for (int i = 0; i < argc - 1; i++) {

        reqs[i] = malloc(sizeof(*reqs[0]));

        if (reqs[i] == NULL) {

            perror("malloc");

            exit(EXIT_FAILURE);

        }

        memset(reqs[i], 0, sizeof(*reqs[0]));

        reqs[i]->ar_name = argv[i + 1];

    }

    ret = getaddrinfo_a(GAI_WAIT, reqs, argc - 1, NULL);

    if (ret != 0) {

        fprintf(stderr, "getaddrinfo_a() failed: %s\n",

            gai_strerror(ret));

        exit(EXIT_FAILURE);

    }
```

```

for (int i = 0; i < argc - 1; i++) {
    printf("%s: ", reqs[i]->ar_name);
    ret = gai_error(reqs[i]);
    if (ret == 0) {
        res = reqs[i]->ar_result;
        ret = getnameinfo(res->ai_addr, res->ai_addrlen,
            host, sizeof(host),
            NULL, 0, NI_NUMERICHOST);
        if (ret != 0) {
            fprintf(stderr, "getnameinfo() failed: %s\n",
                gai_strerror(ret));
            exit(EXIT_FAILURE);
        }
        puts(host);
    } else {
        puts(gai_strerror(ret));
    }
}
exit(EXIT_SUCCESS);
}

```

Asynchronous example

This example shows a simple interactive `getaddrinfo_a()` front-end. The notification facility is not demonstrated.

An example session might look like this:

```

$ ./a.out
> a ftp.us.kernel.org enoent.linuxfoundation.org gnu.cz
> c 2
[2] gnu.cz: Request not canceled
> w 0 1
[00] ftp.us.kernel.org: Finished
> l
[00] ftp.us.kernel.org: 216.165.129.139
[01] enoent.linuxfoundation.org: Processing request in progress

```

[02] gnu.cz: 87.236.197.13

> |

[00] ftp.us.kernel.org: 216.165.129.139

[01] enoent.linuxfoundation.org: Name or service not known

[02] gnu.cz: 87.236.197.13

The program source is as follows:

```
#define _GNU_SOURCE

#include <netdb.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

static struct gaicb **reqs = NULL;

static int nreqs = 0;

static char *

getcmd(void)

{

    static char buf[256];

    fputs("> ", stdout); fflush(stdout);

    if (fgets(buf, sizeof(buf), stdin) == NULL)

        return NULL;

    if (buf[strlen(buf) - 1] == '\n')

        buf[strlen(buf) - 1] = 0;

    return buf;

}

/* Add requests for specified hostnames */

static void

add_requests(void)

{

    int nreqs_base = nreqs;

    char *host;

    int ret;

    while ((host = strtok(NULL, " ")) {

        nreqs++;
```



```

    reqs = realloc(reqs, sizeof(reqs[0]) * nreqs);

    reqs[nreqs - 1] = calloc(1, sizeof(*reqs[0]));

    reqs[nreqs - 1]->ar_name = strdup(host);
}

/* Queue nreqs_base..nreqs requests. */
ret = getaddrinfo_a(GAI_NOWAIT, &reqs[nreqs_base],
                    nreqs - nreqs_base, NULL);

if (ret) {
    fprintf(stderr, "getaddrinfo_a() failed: %s\n",
            gai_strerror(ret));
    exit(EXIT_FAILURE);
}
}

/* Wait until at least one of specified requests completes */
static void
wait_requests(void)
{
    char *id;
    int ret, n;

    struct gaicb const **wait_reqs = calloc(nreqs, sizeof(*wait_reqs));

    /* NULL elements are ignored by gai_suspend(). */
    while ((id = strtok(NULL, " ")) != NULL) {
        n = atoi(id);
        if (n >= nreqs) {
            printf("Bad request number: %s\n", id);
            return;
        }
        wait_reqs[n] = reqs[n];
    }

    ret = gai_suspend(wait_reqs, nreqs, NULL);

    if (ret) {
        printf("gai_suspend(): %s\n", gai_strerror(ret));
        return;
    }
}

```

```

    }

    for (int i = 0; i < nreqs; i++) {
        if (wait_reqs[i] == NULL)
            continue;

        ret = gai_error(reqs[i]);
        if (ret == EAI_INPROGRESS)
            continue;

        printf("[%02d] %s: %s\n", i, reqs[i]->ar_name,
            ret == 0 ? "Finished" : gai_strerror(ret));
    }
}

/* Cancel specified requests */

static void
cancel_requests(void)
{
    char *id;
    int ret, n;
    while ((id = strtok(NULL, " ")) != NULL) {
        n = atoi(id);
        if (n >= nreqs) {
            printf("Bad request number: %s\n", id);
            return;
        }
        ret = gai_cancel(reqs[n]);
        printf("[%s] %s: %s\n", id, reqs[atoi(id)]->ar_name,
            gai_strerror(ret));
    }
}

/* List all requests */

static void
list_requests(void)
{
    int ret;

```

```

char host[NI_MAXHOST];

struct addrinfo *res;

for (int i = 0; i < nreqs; i++) {
    printf("[%02d] %s: ", i, reqs[i]->ar_name);

    ret = gai_error(reqs[i]);

    if (!ret) {
        res = reqs[i]->ar_result;

        ret = getnameinfo(res->ai_addr, res->ai_addrlen,

            host, sizeof(host),

            NULL, 0, NI_NUMERICHOST);

        if (ret) {
            fprintf(stderr, "getnameinfo() failed: %s\n",
                gai_strerror(ret));

            exit(EXIT_FAILURE);
        }

        puts(host);
    } else {
        puts(gai_strerror(ret));
    }
}

int
main(int argc, char *argv[])
{
    char *cmdline;

    char *cmd;

    while ((cmdline = getcmd()) != NULL) {
        cmd = strtok(cmdline, " ");

        if (cmd == NULL) {
            list_requests();
        } else {
            switch (cmd[0]) {
                case 'a':

```

```

        add_requests();

        break;

    case 'w':

        wait_requests();

        break;

    case 'c':

        cancel_requests();

        break;

    case 'l':

        list_requests();

        break;

    default:

        fprintf(stderr, "Bad command: %c\n", cmd[0]);

        break;

    }

}

}

exit(EXIT_SUCCESS);

}

```

SEE ALSO

getaddrinfo(3), inet(3), lio_listio(3), hostname(7), ip(7), sigevent(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/>.

GNU 2020-11-01 GETADDRINFO_A(3)