EVDNS(3) BSD Library Functions Manual EVDNS(3)

NAME top

evdns_init evdns_shutdown evdns_err_to_string evdns_nameserver_add evdns_count_nameservers evdns_clear_nameservers_and_suspend evdns_resume evdns_nameserver_ip_add evdns_resolve_ipv4 evdns_resolve_reverse evdns_resolv_conf_parse evdns_config_windows_nameservers evdns_search_clear evdns_search_add evdns_search_ndots_set evdns_set_log_fn — asynchronous functions for DNS resolution.

SYNOPSIS top

```
#include <sys/time.h>
#include <event.h>
#include <evdns.h>

int
evdns_init();

void
evdns_shutdown(int fail_requests);

const char *
evdns_err_to_string(int err);

int
evdns_nameserver_add(unsigned long int address);

int
evdns_count_nameservers();
```

```
int
evdns_clear_nameservers_and_suspend();

int
evdns_resume();

int
evdns_nameserver_ip_add(const(char, *ip_as_string););

int
evdns_resolve_ipv4(const char *name, int flags,
    evdns_callback_type callback, void *ptr);

int
evdns_resolve_reverse(struct in_addr *in, int flags,
    evdns_callback_type callback, void *ptr);

int
evdns_resolv_conf_parse(int flags, const char *);

void
evdns_search_clear();

void
evdns_search_add(const char *domain);

void
evdns_search_ndots_set(const int ndots);

void
evdns_set_log_fn(evdns_debug_log_fn_type fn);

int
evdns_config_windows_nameservers();
```

DESCRIPTION top

Welcome, gentle reader

Async DNS lookups are really a whole lot harder than they should be,

mostly stemming from the fact that the libc resolver has never been very good at them. Before you use this library you should see if libc can do the job for you with the modern async call getaddrinfo_a (see http://www.imperialviolet.org/page25.html#e498). Otherwise, please continue.

This code is based on libevent and you must call event_init before any of the APIs in this file. You must also seed the OpenSSL random source if you are using OpenSSL for ids (see below).

This library is designed to be included and shipped with your source code. You statically link with it. You should also test for the existence of strtok_r and define HAVE_STRTOK_R if you have it.

The DNS protocol requires a good source of id numbers and these numbers should be unpredictable for spoofing reasons. There are three methods for generating them here and you must define exactly one of them. In increasing order of preference:

DNS_USE_GETTIMEOFDAY_FOR_ID	Using the bottom 16 bits of the usec result from gettimeofday. This is a pretty poor solution but should work anywhere.
DNS_USE_CPU_CLOCK_FOR_ID	Using the bottom 16 bits of the nsec result from the CPU's time counter. This is better, but may not work everywhere. Requires POSIX realtime support and you'll need to link against -lrt on glibc systems at least.
DNS_USE_OPENSSL_FOR_ID	Uses the OpenSSL RAND_bytes call to generate the data. You must have seeded the pool before making any calls to this library.

The library keeps track of the state of nameservers and will avoid them when they go down. Otherwise it will round robin between them.

Quick start guide:

```
  #include "evdns.h"
  void callback(int result, char type, int count, int ttl,       void
```

```
*addresses, void *arg);
  evdns_resolv_conf_parse(DNS_OPTIONS_ALL, "/etc/resolv.conf");
  evdns_resolve("www.hostname.com", 0, callback, NULL);
```

When the lookup is complete the callback function is called. The first argument will be one of the DNS_ERR_* defines in evdns.h. Hopefully it will be DNS_ERR_NONE, in which case type will be DNS_IPv4_A, count will be the number of IP addresses, ttl is the time which the data can be cached for (in seconds), addresses will point to an array of uint32_t's and arg will be whatever you passed to evdns_resolve.

Searching:

In order for this library to be a good replacement for glibc's resolver it supports searching. This involves setting a list of default domains, in which names will be queried for. The number of dots in the query name determines the order in which this list is used.

Searching appears to be a single lookup from the point of view of the API, although many DNS queries may be generated from a single call to evdns_resolve. Searching can also drastically slow down the resolution of names.

To disable searching:

1. Never set it up. If you never call evdns_resolv_conf_parse,() evdns_init,() or evdns_search_add() then no searching will occur.
2. If you do call evdns_resolv_conf_parse() then don't pass DNS_OPTION_SEARCH (or DNS_OPTIONS_ALL, which implies it).
3. When calling evdns_resolve,() pass the DNS_QUERY_NO_SEARCH flag.

The order of searches depends on the number of dots in the name. If the number is greater than the ndots setting then the names is first tried globally. Otherwise each search domain is appended in turn.

The ndots setting can either be set from a resolv.conf, or by calling evdns_search_ndots_set.

For example, with ndots set to 1 (the default) and a search domain list of ["myhome.net"]:

```
Query: www
Order: www.myhome.net, www.

Query: www.abc
Order: www.abc., www.abc.myhome.net
```

API reference top

`int evdns_init()`
: Initializes support for non-blocking name resolution by calling `evdns_resolv_conf_parse()` on UNIX and `evdns_config_windows_nameservers()` on Windows.

`int evdns_nameserver_add(unsigned long int address)`
: Add a nameserver. The address should be an IP address in network byte order. The type of address is chosen so that it matches in_addr.s_addr. Returns non-zero on error.

`int evdns_nameserver_ip_add(const char *ip_as_string)`
: This wraps the above function by parsing a string as an IP address and adds it as a nameserver. Returns non-zero on error

`int evdns_resolve(const char *name, int flags, evdns_callback_type callback, void *ptr)`
: Resolve a name. The name parameter should be a DNS name. The flags parameter should be 0, or DNS_QUERY_NO_SEARCH which disables searching for this query. (see defn of searching above).

  The callback argument is a function which is called when this query completes and ptr is an argument which is passed to that callback function.

  Returns non-zero on error

`void evdns_search_clear()`
: Clears the list of search domains

`void evdns_search_add(const char *domain)`
: Add a domain to the list of search domains

```
void evdns_search_ndots_set(int ndots)
        Set the number of dots which, when found in a name, causes the
        first query to be without any search domain.

int evdns_count_nameservers(void)
        Return the number of configured nameservers (not necessarily
        the number of running nameservers).  This is useful for dou-
        ble-checking whether our calls to the various nameserver con-
        figuration functions have been successful.

int evdns_clear_nameservers_and_suspend(void)
        Remove all currently configured nameservers, and suspend all
        pending resolves.  Resolves will not necessarily be re-
        attempted until evdns_resume() is called.

int evdns_resume(void)
        Re-attempt resolves left in limbo after an earlier call to
        evdns_clear_nameservers_and_suspend().

int evdns_config_windows_nameservers(void)
        Attempt to configure a set of nameservers based on platform
        settings on a win32 host.  Preferentially tries to use GetNet-
        workParams; if that fails, looks in the registry.  Returns 0
        on success, nonzero on failure.

int evdns_resolv_conf_parse(int flags, const char *filename)
        Parse a resolv.conf like file from the given filename.

        See the man page for resolv.conf for the format of this file.
        The flags argument determines what information is parsed from
        this file:
              DNS_OPTION_SEARCH         domain, search and ndots
                                        options
              DNS_OPTION_NAMESERVERS    nameserver lines
              DNS_OPTION_MISC           timeout and attempts options
              DNS_OPTIONS_ALL           all of the above

        The following directives are not parsed from the file:
          sortlist, rotate, no-check-names, inet6, debug
```

Returns non-zero on error:

0	no errors
1	failed to open file
2	failed to stat file
3	file too large
4	out of memory
5	short read from file

Internals: top

Requests are kept in two queues. The first is the inflight queue. In this queue requests have an allocated transaction id and nameserver. They will soon be transmitted if they haven't already been.

The second is the waiting queue. The size of the inflight ring is limited and all other requests wait in waiting queue for space. This bounds the number of concurrent requests so that we don't flood the nameserver. Several algorithms require a full walk of the inflight queue and so bounding its size keeps thing going nicely under huge (many thousands of requests) loads.

If a nameserver loses too many requests it is considered down and we try not to use it. After a while we send a probe to that nameserver (a lookup for google.com) and, if it replies, we consider it working again. If the nameserver fails a probe we wait longer to try again with the next probe.

SEE ALSO top

event(3), gethostbyname(3), resolv.conf(5)

HISTORY top

The evdns API was developed by Adam Langley on top of the libevent API. The code was integrate into Tor by Nick Mathewson and finally put into libevent itself by Niels Provos.

AUTHORS top

The evdns API and code was written by Adam Langley with significant contributions by Nick Mathewson.

BUGS top

This documentation is neither complete nor authoritative. If you are in doubt about the usage of this API then check the source code to find out how it works, write up the missing piece of documentation and send it to me for inclusion in this man page.

COLOPHON top

This page is part of the libevent (an event notification library) project. Information about the project can be found at http://libevent.org/. If you have a bug report for this manual page,

see ⟨http://sourceforge.net/p/levent/bugs/⟩. This page was obtained

from the project's upstream Git repository

⟨https://github.com/libevent/libevent.git⟩ on 2018-10-29. (At that

time, the date of the most recent commit that was found in the reposi - tory was 2018-10-28.) If you discover any rendering problems in this HTML version of the page, or you believe there is a better or more up- to-date source for the page, or you have corrections or improvements to the information in this COLOPHON (which is not part of the original manual page), send a mail to man-pages@man7.org

HTML rendering created 2018-10-29 by Michael Kerrisk, author of The Linux Programming Interface, maintainer of the Linux man-pages project.

For details of in-depth Linux/UNIX system programming training courses that I teach, look here.

Hosting by jambit GmbH.