NAME

libcurl-thread - libcurl thread safety

Multi-threading with libcurl

libcurl is thread safe but has no internal thread synchronization. You may have to provide your own locking should you meet any of the thread safety exceptions below.

Handles

You must **never** share the same handle in multiple threads. You can pass the handles around among threads, but you must never use a single handle from more than one thread at any given time.

Shared objects

You can share certain data between multiple handles by using the share interface but you must provide your own locking and set *curl_share_setopt(3)* CURLSHOPT_LOCKFUNC and CURLSHOPT_UNLOCKFUNC.

Note that some items are specifically documented as not thread-safe in the share API (the connection pool and HSTS cache for example).

TLS

All current TLS libraries libcurl supports are thread-safe.

OpenSSL

OpenSSL 1.1.0+ can be safely used in multi-threaded applications provided that support for the underlying OS threading API is built-in. For older versions of OpenSSL, the user must set mutex callbacks.

libcurl may not be able to fully clean up after multi-threaded OpenSSL depending on how OpenSSL was built and loaded as a library. It is possible in some rare circumstances a memory leak could occur unless you implement your own OpenSSL thread cleanup.

For example, on Windows if both libcurl and OpenSSL are linked statically to a DLL or application then OpenSSL may leak memory unless the DLL or application calls OPENSSL_thread_stop() before each thread terminates. If OpenSSL is built as a DLL then it does this cleanup automatically and there is no leak. If libcurl is built as a DLL and OpenSSL is linked statically to it then libcurl does this cleanup automatically and there is no leak (added in libcurl 8.8.0).

Please review the OpenSSL documentation for a full list of circumstances: https://docs.openssl.org/3.0/man3/OPENSSL_init_crypto/#notes

Signals

Signals are used for timing out name resolves (during DNS lookup) - when built without using either the c-ares or threaded resolver backends. On systems that have a signal concept.

When using multiple threads you should set the *CURLOPT_NOSIGNAL(3)* option to 1L for all handles. Everything works fine except that timeouts cannot be honored during DNS lookups - which you can work around by building libcurl with c-ares or threaded-resolver support. c-ares is a library that provides asynchronous name resolves. On some platforms, libcurl simply cannot function properly multi-threaded unless the *CURLOPT_NOSIGNAL(3)* option is set.

When *CURLOPT_NOSIGNAL(3)* is set to 1L, your application needs to deal with the risk of a SIGPIPE (that at least the OpenSSL backend can trigger). Note that setting *CURLOPT_NOSIGNAL(3)* to 0L does not work in a threaded situation as there is a race condition where libcurl risks restoring the former signal handler while another thread should still ignore it.

Name resolving

The **gethostbyname** or **getaddrinfo** and other name resolving system calls used by libcurl are provided by your operating system and must be thread safe. It is important that libcurl can find and use thread safe versions of these and other system calls, as otherwise it cannot function fully thread safe. Some operating systems are known to have faulty thread implementations. We have previously received problem reports on *BSD (at least in the past, they may be working fine these days). Some operating systems that are known to have solid and working thread support are Linux, Solaris and Windows.

curl_global_* functions

These functions are thread-safe since libcurl 7.84.0 if *curl_version_info(3)* has the **CURL_VERSION_THREADSAFE** feature bit set (most platforms).

If these functions are not thread-safe and you are using libcurl with multiple threads it is especially important that before use you call *curl_global_init(3)* or *curl_global_init_mem(3)* to explicitly initialize the library and its dependents, rather than rely on the "lazy" fail-safe initialization that takes place the first time *curl_easy_init(3)* is called. For an in-depth explanation refer to *libcurl(3)* section **GLOBAL CONSTANTS**.

Memory functions

These functions, provided either by your operating system or your own replacements, must be thread safe. You can use *curl_global_init_mem(3)* to set your own replacement memory functions.

Non-safe functions

CURLOPT_DNS_USE_GLOBAL_CACHE(3) is not thread-safe.

curl_version_info(3) is not thread-safe before libcurl initialization.

SEE ALSO

libcurl-security(3)