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.

TLS

If you are accessing HTTPS or FTPS URLs in a multi-threaded manner, you are then of course using the underlying SSL library multi-threaded and those libs might have their own requirements on this issue. You may need to provide one or two functions to allow it to function properly:

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."

  https://www.openssl.org/docs/manmaster/crypto/threads.html#DESCRIPTION

  OpenSSL <= 1.0.2 the user must set callbacks.

  https://www.openssl.org/docs/man1.0.2/crypto/threads.html#DESCRIPTION

  https://curl.haxx.se/libcurl/c/opensslthreadlock.html

GnuTLS
: http://gnutls.org/manual/html_node/Thread-safety.html

NSS
: thread-safe already without anything required.

PolarSSL
: Required actions unknown.

yassl
: Required actions unknown.

axTLS
: Required actions unknown.

Secure-Transport
: The engine is used by libcurl in a way that is fully thread-safe.

WinSSL
: The engine is used by libcurl in a way that is fully thread-safe.

wolfSSL
: The engine is used by libcurl in a way that is fully thread-safe.

BoringSSL
: The engine is used by libcurl in a way that is fully thread-safe.

Other areas of caution

Signals
: Signals are used for timing out name resolves (during DNS lookup) - when built without using either the c-ares or threaded resolver backends. When using multiple threads you should set the *CURLOPT_NOSIGNAL(3)* option to 1L for all handles. Everything will or might work fine except that timeouts are not honored during the DNS lookup - which you can work around by building libcurl with c-ares support. c-ares is a library that provides asynchronous name resolves. On some

platforms, libcurl simply will not function properly multi-threaded unless this option is set.

Name resolving

gethostby* functions and other system calls. These functions, provided by your operating system, must be thread safe. It is very important that libcurl can find and use thread safe versions of these and other system calls, as otherwise it can't 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 not thread safe. If 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.