NAME

openssl-threads - Overview of thread safety in OpenSSL

DESCRIPTION

In this man page, we use the term **thread-safe** to indicate that an object or function can be used by multiple threads at the same time.

OpenSSL can be built with or without threads support. The most important use of this support is so that OpenSSL itself can use a single consistent API, as shown in "EXAMPLES" in **CRYPTO_THREAD_run_once**(3). Multi-platform applications can also use this API.

In particular, being configured for threads support does not imply that all OpenSSL objects are thread-safe. To emphasize: *most objects are not safe for simultaneous use*. Exceptions to this should be documented on the specific manual pages, and some general high-level guidance is given here.

One major use of the OpenSSL thread API is to implement reference counting. Many objects within OpenSSL are reference-counted, so resources are not released, until the last reference is removed. References are often increased automatically (such as when an **X509** certificate object is added into an **X509_STORE** trust store). There is often an ***object*_up_ref**() function that can be used to increase the reference count. Failure to match ***object*_up_ref**() calls with the right number of ***object*_free**() calls is a common source of memory leaks when a program exits.

Many objects have set and get API's to set attributes in the object. A "set0" passes ownership from the caller to the object and a "get0" returns a pointer but the attribute ownership remains with the object and a reference to it is returned. A "set1" or "get1" function does not change the ownership, but instead updates the attribute's reference count so that the object is shared between the caller and the object; the caller must free the returned attribute when finished. Functions that involve attributes that have reference counts themselves, but are named with just "set" or "get" are historical; and the documentation must state how the references are handled. Get methods are often thread-safe as long as the ownership requirements are met and shared objects are not modified. Set methods, or modifying shared objects, are generally not thread-safe as discussed below.

Objects are thread-safe as long as the API's being invoked don't modify the object; in this case the parameter is usually marked in the API as "const". Not all parameters are marked this way. Note that a "const" declaration does not mean immutable; for example **X509_cmp**(3) takes pointers to "const" objects, but the implementation uses a C cast to remove that so it can lock objects, generate and cache a DER encoding, and so on.

Another instance of thread-safety is when updates to an object's internal state, such as cached values, are done with locks. One example of this is the reference counting API's described above.

In all cases, however, it is generally not safe for one thread to mutate an object, such as setting elements of a private or public key, while another thread is using that object, such as verifying a signature.

The same API's can usually be used simultaneously on different objects without interference. For example, two threads can calculate a signature using two different **EVP_PKEY_CTX** objects.

For implicit global state or singletons, thread-safety depends on the facility. The **CRYPTO_secure_malloc**(3) and related API's have their own lock, while **CRYPTO_malloc**(3) assumes the underlying platform allocation will do any necessary locking. Some API's, such as **NCONF_load**(3) and related, or **OBJ_create**(3) do no locking at all; this can be considered a bug.

A separate, although related, issue is modifying "factory" objects when other objects have been created from that. For example, an **SSL_CTX** object created by **SSL_CTX_new**(3) is used to create per-connection **SSL** objects by calling **SSL_new**(3). In this specific case, and probably for factory methods in general, it is not safe to modify the factory object after it has been used to create other objects.

SEE ALSO

CRYPTO_THREAD_run_once(3), local system threads documentation.

BUGS

This page is admittedly very incomplete.

COPYRIGHT