NAME

CRYPTO_THREADID_set_callback, CRYPTO_THREADID_get_callback, CRYPTO_THREADID_current, CRYPTO_THREADID_cmp, CRYPTO_THREADID_cpy, CRYPTO_THREADID_hash, CRYPTO_set_locking_callback, CRYPTO_num_locks, CRYPTO_set_dynlock_create_callback, CRYPTO_set_dynlock_lock_callback, CRYPTO_set_dynlock_destroy_callback, CRYPTO_get_new_dynlockid, CRYPTO_destroy_dynlockid, CRYPTO_lock - OpenSSL thread support

SYNOPSIS

```
#include <openssl/crypto.h>

/* Don't use this structure directly. */
typedef struct crypto_threadid_st
{
void *ptr;
unsigned long val;
} CRYPTO_THREADID;
/* Only use CRYPTO_THREADID_set_[numeric|pointer]() within callbacks */
void CRYPTO_THREADID_set_numeric(CRYPTO_THREADID *id, unsigned long val);
void CRYPTO_THREADID_set_pointer(CRYPTO_THREADID *id, void *ptr);
int CRYPTO_THREADID_set_callback(void (*threadid_func)(CRYPTO_THREADID *));
void (*CRYPTO_THREADID_get_callback(void))(CRYPTO_THREADID *);
void CRYPTO_THREADID_current(CRYPTO_THREADID *id);
int CRYPTO_THREADID_cmp(const CRYPTO_THREADID *a,
const CRYPTO_THREADID *b);
void CRYPTO_THREADID_cpy(CRYPTO_THREADID *dest,
const CRYPTO_THREADID *src);
unsigned long CRYPTO_THREADID_hash(const CRYPTO_THREADID *id);

int CRYPTO_num_locks(void);

/* struct CRYPTO_dynlock_value needs to be defined by the user */
struct CRYPTO_dynlock_value;

void CRYPTO_set_dynlock_create_callback(struct CRYPTO_dynlock_value *
(*dyn_create_function)(char *file, int line));
void CRYPTO_set_dynlock_lock_callback(void (*dyn_lock_function)
(int mode, struct CRYPTO_dynlock_value *l,
const char *file, int line));
void CRYPTO_set_dynlock_destroy_callback(void (*dyn_destroy_function)
(struct CRYPTO_dynlock_value *l, const char *file, int line));

int CRYPTO_get_new_dynlockid(void);

void CRYPTO_destroy_dynlockid(int i);

void CRYPTO_lock(int mode, int n, const char *file, int line);

#define CRYPTO_w_lock(type) \
CRYPTO_lock(CRYPTO_LOCK|CRYPTO_WRITE,type,__FILE__,__LINE__)
#define CRYPTO_w_unlock(type) \
CRYPTO_lock(CRYPTO_UNLOCK|CRYPTO_WRITE,type,__FILE__,__LINE__)
#define CRYPTO_r_lock(type) \
CRYPTO_lock(CRYPTO_LOCK|CRYPTO_READ,type,__FILE__,__LINE__)
#define CRYPTO_r_unlock(type) \
```

```
CRYPTO_lock(CRYPTO_UNLOCK|CRYPTO_READ,type,__FILE__,__LINE__)
#define CRYPTO_add(addr,amount,type) \
CRYPTO_add_lock(addr,amount,type,__FILE__,__LINE__)
```

DESCRIPTION

OpenSSL can safely be used in multi-threaded applications provided that at least two callback functions are set, locking_function and threadid_func.

locking_function(int mode, int n, const char *file, int line) is needed to perform locking on shared data structures. (Note that OpenSSL uses a number of global data structures that will be implicitly shared whenever multiple threads use OpenSSL.) Multi-threaded applications will crash at random if it is not set.

locking_function() must be able to handle up to *CRYPTO_num_locks()* different mutex locks. It sets the **n**-th lock if **mode** & **CRYPTO_LOCK**, and releases it otherwise.

file and **line** are the file number of the function setting the lock. They can be useful for debugging.

threadid_func(CRYPTO_THREADID *id) is needed to record the currently-executing thread's identifier into **id**. The implementation of this callback should not fill in **id** directly, but should use *CRYPTO_THREADID_set_numeric()* if thread IDs are numeric, or *CRYPTO_THREADID_set_pointer()* if they are pointer-based. If the application does not register such a callback using *CRYPTO_THREADID_set_callback()*, then a default implementation is used - on Windows and BeOS this uses the system's default thread identifying APIs, and on all other platforms it uses the address of **errno**. The latter is satisfactory for thread-safety if and only if the platform has a thread-local error number facility.

Once *threadid_func()* is registered, or if the built-in default implementation is to be used;

- *CRYPTO_THREADID_current()* records the currently-executing thread ID into the given **id** object.
- *CRYPTO_THREADID_cmp()* compares two thread IDs (returning zero for equality, ie. the same semantics as *memcmp()*).
- *CRYPTO_THREADID_cpy()* duplicates a thread ID value,
- *CRYPTO_THREADID_hash()* returns a numeric value usable as a hash-table key. This is usually the exact numeric or pointer-based thread ID used internally, however this also handles the unusual case where pointers are larger than 'long' variables and the platform's thread IDs are pointer-based - in this case, mixing is done to attempt to produce a unique numeric value even though it is not as wide as the platform's true thread IDs.

Additionally, OpenSSL supports dynamic locks, and sometimes, some parts of OpenSSL need it for better performance. To enable this, the following is required:

- Three additional callback function, dyn_create_function, dyn_lock_function and dyn_destroy_function.
- A structure defined with the data that each lock needs to handle.

struct CRYPTO_dynlock_value has to be defined to contain whatever structure is needed to handle locks.

dyn_create_function(const char *file, int line) is needed to create a lock. Multi-threaded applications might crash at random if it is not set.

dyn_lock_function(int mode, CRYPTO_dynlock *l, const char *file, int line) is needed to perform locking off dynamic lock numbered n. Multi-threaded applications might crash at random if it is not set.

dyn_destroy_function(CRYPTO_dynlock *l, const char *file, int line) is needed to destroy the lock l. Multi-threaded applications might crash at random if it is not set.

CRYPTO_get_new_dynlockid() is used to create locks. It will call dyn_create_function for the actual creation.

CRYPTO_destroy_dynlockid() is used to destroy locks. It will call dyn_destroy_function for the actual destruction.

CRYPTO_lock() is used to lock and unlock the locks. mode is a bitfield describing what should be done

with the lock. n is the number of the lock as returned from *CRYPTO_get_new_dynlockid()*. mode can be combined from the following values. These values are pairwise exclusive, with undefined behaviour if misused (for example, CRYPTO_READ and CRYPTO_WRITE should not be used together):

```
CRYPTO_LOCK 0x01
CRYPTO_UNLOCK 0x02
CRYPTO_READ 0x04
CRYPTO_WRITE 0x08
```

RETURN VALUES

CRYPTO_num_locks() returns the required number of locks.

CRYPTO_get_new_dynlockid() returns the index to the newly created lock.

The other functions return no values.

NOTES

You can find out if OpenSSL was configured with thread support:

```
#define OPENSSL_THREAD_DEFINES
#include <openssl/opensslconf.h>
#if defined(OPENSSL_THREADS)
// thread support enabled
#else
// no thread support
#endif
```

Also, dynamic locks are currently not used internally by OpenSSL, but may do so in the future.

EXAMPLES

crypto/threads/mttest.c shows examples of the callback functions on Solaris, Irix and Win32.

HISTORY

CRYPTO_set_locking_callback() is available in all versions of SSLeay and OpenSSL. *CRYPTO_num_locks()* was added in OpenSSL 0.9.4. All functions dealing with dynamic locks were added in OpenSSL 0.9.5b-dev. **CRYPTO_THREADID** and associated functions were introduced in OpenSSL 1.0.0 to replace (actually, deprecate) the previous *CRYPTO_set_id_callback()*, *CRYPTO_get_id_callback()*, and *CRYPTO_thread_id()* functions which assumed thread IDs to always be represented by 'unsigned long'.

SEE ALSO

crypto(3)