NAME

CRYPTO_THREAD_run_once, CRYPTO_THREAD_lock_new, CRYPTO_THREAD_read_lock, CRYPTO_THREAD_write_lock, CRYPTO_THREAD_unlock, CRYPTO_THREAD_lock_free, CRYPTO_atomic_add, CRYPTO_atomic_or, CRYPTO_atomic_load - OpenSSL thread support

SYNOPSIS

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

CRYPTO_ONCE CRYPTO_ONCE_STATIC_INIT;
int CRYPTO_THREAD_run_once(CRYPTO_ONCE *once, void (*init)(void));

CRYPTO_RWLOCK *CRYPTO_THREAD_lock_new(void);
int CRYPTO_THREAD_read_lock(CRYPTO_RWLOCK *lock);
int CRYPTO_THREAD_write_lock(CRYPTO_RWLOCK *lock);
int CRYPTO_THREAD_unlock(CRYPTO_RWLOCK *lock);
void CRYPTO_THREAD_lock_free(CRYPTO_RWLOCK *lock);

int CRYPTO_atomic_add(int *val, int amount, int *ret, CRYPTO_RWLOCK *lock);
int CRYPTO_atomic_or(uint64_t *val, uint64_t op, uint64_t *ret,
                     CRYPTO_RWLOCK *lock);
int CRYPTO_atomic_load(uint64_t *val, uint64_t *ret, CRYPTO_RWLOCK *lock);
```

DESCRIPTION

OpenSSL can be safely used in multi-threaded applications provided that support for the underlying OS threading API is built-in. Currently, OpenSSL supports the pthread and Windows APIs. OpenSSL can also be built without any multi-threading support, for example on platforms that don't provide any threading support or that provide a threading API that is not yet supported by OpenSSL.

The following multi-threading function are provided:

- **CRYPTO_THREAD_run_once()** can be used to perform one-time initialization. The *once* argument must be a pointer to a static object of type **CRYPTO_ONCE** that was statically initialized to the value **CRYPTO_ONCE_STATIC_INIT**. The *init* argument is a pointer to a function that performs the desired exactly once initialization. In particular, this can be used to allocate locks in a thread-safe manner, which can then be used with the locking functions below.

- **CRYPTO_THREAD_lock_new()** allocates, initializes and returns a new read/write lock.

- **CRYPTO_THREAD_read_lock()** locks the provided *lock* for reading.

- **CRYPTO_THREAD_write_lock()** locks the provided *lock* for writing.

- **CRYPTO_THREAD_unlock()** unlocks the previously locked *lock*.

- **CRYPTO_THREAD_lock_free()** frees the provided *lock*.

- **CRYPTO_atomic_add()** atomically adds *amount* to **val* and returns the result of the operation in **ret*. *lock* will be locked, unless atomic operations are supported on the specific platform. Because of this, if a variable is modified by **CRYPTO_atomic_add()** then **CRYPTO_atomic_add()** must be the only way that the variable is modified. If atomic operations are not supported and *lock* is NULL, then the function will fail.

- **CRYPTO_atomic_or()** performs an atomic bitwise or of *op* and **val* and stores the result back in **val*. It also returns the result of the operation in **ret*. *lock* will be locked, unless atomic operations are supported on the specific platform. Because of this, if a variable is modified by **CRYPTO_atomic_or()** or read by **CRYPTO_atomic_load()** then **CRYPTO_atomic_or()** must be the only way that the variable is modified. If atomic operations are not supported and *lock* is NULL, then the function will fail.

- **CRYPTO_atomic_load()** atomically loads the contents of **val* into **ret*. *lock* will be locked, unless atomic operations are supported on the specific platform. Because of this, if a variable is modified by **CRYPTO_atomic_or()** or read by **CRYPTO_atomic_load()** then **CRYPTO_atomic_load()** must be the only way that the variable is read. If atomic operations are not supported and *lock* is NULL, then the function will fail.

RETURN VALUES

CRYPTO_THREAD_run_once() returns 1 on success, or 0 on error.

CRYPTO_THREAD_lock_new() returns the allocated lock, or NULL on error.

CRYPTO_THREAD_lock_free() returns no value.

The other functions return 1 on success, or 0 on error.

NOTES

On Windows platforms the CRYPTO_THREAD_* types and functions in the <*openssl/crypto.h*> header are dependent on some of the types customarily made available by including <*windows.h*>. The application developer is likely to require control over when the latter is included, commonly as one of the first included headers. Therefore, it is defined as an application developer's responsibility to include <*windows.h*> prior to <*openssl/crypto.h*> where use of CRYPTO_THREAD_* types and functions is required.

EXAMPLES

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

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

This example safely initializes and uses a lock.

```
 #ifdef _WIN32
 # include <windows.h>
 #endif
 #include <openssl/crypto.h>

 static CRYPTO_ONCE once = CRYPTO_ONCE_STATIC_INIT;
 static CRYPTO_RWLOCK *lock;

 static void myinit(void)
 {
     lock = CRYPTO_THREAD_lock_new();
 }

 static int mylock(void)
 {
     if (!CRYPTO_THREAD_run_once(&once, void init) || lock == NULL)
         return 0;
     return CRYPTO_THREAD_write_lock(lock);
 }

 static int myunlock(void)
 {
     return CRYPTO_THREAD_unlock(lock);
 }

 int serialized(void)
 {
     int ret = 0;

     if (mylock()) {
         /* Your code here, do not return without releasing the lock! */
         ret = ... ;
     }
     myunlock();
     return ret;
 }
```

Finalization of locks is an advanced topic, not covered in this example. This can only be done at process exit or when a dynamically loaded library is no longer in use and is unloaded. The simplest solution is to just "leak" the lock in applications and not repeatedly load/unload shared libraries that allocate locks.

SEE ALSO

crypto(7), **openssl-threads**(7).

COPYRIGHT