NAME

SSL_shutdown - shut down a TLS/SSL connection

SYNOPSIS

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

int SSL_shutdown(SSL *ssl);
```

DESCRIPTION

SSL_shutdown() shuts down an active TLS/SSL connection. It sends the ‘‘close notify’’ shutdown alert to the peer.

NOTES

SSL_shutdown() tries to send the ‘‘close notify’’ shutdown alert to the peer. Whether the operation succeeds or not, the SSL_SENT_SHUTDOWN flag is set and a currently open session is considered closed and good and will be kept in the session cache for further reuse.

Note that *SSL_shutdown()* must not be called if a previous fatal error has occurred on a connection i.e. if *SSL_get_error()* has returned SSL_ERROR_SYSCALL or SSL_ERROR_SSL.

The shutdown procedure consists of 2 steps: the sending of the ‘‘close notify’’ shutdown alert and the reception of the peer’s ‘‘close notify’’ shutdown alert. According to the TLS standard, it is acceptable for an application to only send its shutdown alert and then close the underlying connection without waiting for the peer’s response (this way resources can be saved, as the process can already terminate or serve another connection). When the underlying connection shall be used for more communications, the complete shutdown procedure (bidirectional ‘‘close notify’’ alerts) must be performed, so that the peers stay synchronized.

SSL_shutdown() supports both uni- and bidirectional shutdown by its 2 step behaviour.

When the application is the first party to send the ‘‘close notify’’ alert, *SSL_shutdown()* will only send the alert and then set the SSL_SENT_SHUTDOWN flag (so that the session is considered good and will be kept in cache). *SSL_shutdown()* will then return with 0. If a unidirectional shutdown is enough (the underlying connection shall be closed anyway), this first call to *SSL_shutdown()* is sufficient. In order to complete the bidirectional shutdown handshake, *SSL_shutdown()* must be called again. The second call will make *SSL_shutdown()* wait for the peer’s ‘‘close notify’’ shutdown alert. On success, the second call to *SSL_shutdown()* will return with 1.

SSL_set_shutdown(3) call." 4

If the peer already sent the ‘‘close notify’’ alert **and** it was already processed implicitly inside another function (*SSL_read*(3)), the SSL_RECEIVED_SHUTDOWN flag is set. *SSL_shutdown()* will send the ‘‘close notify’’ alert, set the SSL_SENT_SHUTDOWN flag and will immediately return with 1. Whether SSL_RECEIVED_SHUTDOWN is already set can be checked using the *SSL_get_shutdown()* (see also

SSL_set_shutdown(3) call." 4

It is therefore recommended, to check the return value of *SSL_shutdown()* and call *SSL_shutdown()* again, if the bidirectional shutdown is not yet complete (return value of the first call is 0).

The behaviour of *SSL_shutdown()* additionally depends on the underlying BIO.

If the underlying BIO is **blocking**, *SSL_shutdown()* will only return once the handshake step has been finished or an error occurred.

If the underlying BIO is **non-blocking**, *SSL_shutdown()* will also return when the underlying BIO could not satisfy the needs of *SSL_shutdown()* to continue the handshake. In this case a call to *SSL_get_error()* with the return value of *SSL_shutdown()* will yield **SSL_ERROR_WANT_READ** or **SSL_ERROR_WANT_WRITE**. The calling process then must repeat the call after taking appropriate action to satisfy the needs of *SSL_shutdown()*. The action depends on the underlying BIO. When using a non-blocking socket, nothing is to be done, but *select()* can be used to check for the required condition. When using a buffering BIO, like a BIO pair, data must be written into or retrieved out of the BIO before being able to continue.

SSL_shutdown() can be modified to only set the connection to ‘‘shutdown’’ state but not actually send the ''close notify'' alert messages, see *SSL_CTX_set_quiet_shutdown(3)*. When ''quiet shutdown'' is enabled, *SSL_shutdown()* will always succeed and return 1.

RETURN VALUES

The following return values can occur:

0 The shutdown is not yet finished. Call *SSL_shutdown()* for a second time, if a bidirectional shutdown shall be performed. The output of *SSL_get_error(3)* may be misleading, as an erroneous SSL_ERROR_SYSCALL may be flagged even though no error occurred.

1 The shutdown was successfully completed. The ''close notify'' alert was sent and the peer's ''close notify'' alert was received.

<0 The shutdown was not successful because a fatal error occurred either at the protocol level or a connection failure occurred. It can also occur if action is need to continue the operation for non-blocking BIOs. Call *SSL_get_error(3)* with the return value **ret** to find out the reason.

SEE ALSO

SSL_get_error(3), *SSL_connect(3)*, *SSL_accept(3)*, *SSL_set_shutdown(3)*, *SSL_CTX_set_quiet_shutdown(3)*, *SSL_clear(3)*, *SSL_free(3)*, *ssl(3)*, *bio(3)*

COPYRIGHT