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.

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 two steps: sending of the close_notify shutdown alert, and reception of the peer's close_notify shutdown alert. The order of those two steps depends on the application.

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. This should only be done when it is known that the other side will not send more data, otherwise there is a risk of a truncation attack.

When a client only writes and never reads from the connection, and the server has sent a session ticket to establish a session, the client might not be able to resume the session because it did not received and process the session ticket from the server. In case the application wants to be able to resume the session, it is recommended to do a complete shutdown procedure (bidirectional close_notify alerts).

When the underlying connection shall be used for more communications, the complete shutdown procedure must be performed, so that the peers stay synchronized.

SSL_shutdown() only closes the write direction. It is not possible to call **SSL_write()** after calling **SSL_shutdown()**. The read direction is closed by the peer.

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 **nonblocking**, **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 nonblocking 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.

After **SSL_shutdown()** returned 0, it is possible to call **SSL_shutdown()** again to wait for the peer's close_notify alert. **SSL_shutdown()** will return 1 in that case. However, it is recommended to wait for it using **SSL_read()** instead.

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. Note that this is not standard compliant behaviour. It should only be done when the peer has a way to make sure all data has been received and doesn't wait for the close_notify alert message, otherwise an unexpected EOF will be reported.

There are implementations that do not send the required close_notify alert. If there is a need to communicate with such an implementation, and it's clear that all data has been received, do not wait for the peer's close_notify alert. Waiting for the close_notify alert when the peer just closes the connection will result in an error being generated. The error can be ignored using the **SSL_OP_IGNORE_UNEXPECTED_EOF**. For more information see **SSL_CTX_set_options**(3).

First to close the connection

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 the cache). If successful, **SSL_shutdown()** will return 0.

If a unidirectional shutdown is enough (the underlying connection shall be closed anyway), this first successful call to **SSL_shutdown()** is sufficient.

In order to complete the bidirectional shutdown handshake, the peer needs to send back a close_notify alert. The SSL_RECEIVED_SHUTDOWN flag will be set after receiving and processing it.

The peer is still allowed to send data after receiving the close_notify event. When it is done sending

data, it will send the close_notify alert. **SSL_read()** should be called until all data is received. **SSL_read()** will indicate the end of the peer data by returning <= 0 and **SSL_get_error()** returning SSL_ERROR_ZERO_RETURN.

Peer closes the connection

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_read()** will return <= 0 in that case, and **SSL_get_error()** will return SSL_ERROR_ZERO_RETURN. **SSL_shutdown()** will send the close_notify alert, set the SSL_SENT_SHUTDOWN flag. If successful, **SSL_shutdown()** will return 1.

Whether SSL_RECEIVED_SHUTDOWN is already set can be checked using the **SSL_get_shutdown()** (see also **SSL_set_shutdown**(3) call.

RETURN VALUES

The following return values can occur:

0 The shutdown is not yet finished: the close_notify was sent but the peer did not send it back yet. Call **SSL_read()** to do a bidirectional shutdown.

Unlike most other function, returning 0 does not indicate an error. **SSL_get_error**(3) should not get called, it may misleadingly indicate an error 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. Call **SSL_get_error**(3) with the return value **ret** to find out the reason. It can occur if an action is needed to continue the operation for nonblocking BIOs.

It can also occur when not all data was read using **SSL_read()**.

SEE ALSO

SSL_get_error(3), **SSL_connect**(3), **SSL_accept**(3), **SSL_set_shutdown**(3), **SSL_CTX_set_quiet_shutdown**(3), **SSL_CTX_set_options**(3) **SSL_clear**(3), **SSL_free**(3), **ssl**(7), **bio**(7)

COPYRIGHT