NAME

SSL_write_ex, SSL_write, SSL_sendfile - write bytes to a TLS/SSL connection

SYNOPSIS

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

ossl_ssize_t SSL_sendfile(SSL *s, int fd, off_t offset, size_t size, int flags);
int SSL_write_ex(SSL *s, const void *buf, size_t num, size_t *written);
int SSL_write(SSL *ssl, const void *buf, int num);
```

DESCRIPTION

SSL_write_ex() and **SSL_write()** write **num** bytes from the buffer **buf** into the specified **ssl** connection. On success **SSL_write_ex()** will store the number of bytes written in ***written**.

SSL_sendfile() writes **size** bytes from offset **offset** in the file descriptor **fd** to the specified SSL connection **s**. This function provides efficient zero-copy semantics. **SSL_sendfile()** is available only when Kernel TLS is enabled, which can be checked by calling **BIO_get_ktls_send()**. It is provided here to allow users to maintain the same interface. The meaning of **flags** is platform dependent. Currently, under Linux it is ignored.

NOTES

In the paragraphs below a "write function" is defined as one of either **SSL_write_ex()**, or **SSL_write()**.

If necessary, a write function will negotiate a TLS/SSL session, if not already explicitly performed by **SSL_connect**(3) or **SSL_accept**(3). If the peer requests a re-negotiation, it will be performed transparently during the write function operation. The behaviour of the write functions depends on the underlying BIO.

For the transparent negotiation to succeed, the **ssl** must have been initialized to client or server mode. This is being done by calling **SSL_set_connect_state**(3) or **SSL_set_accept_state()** before the first call to a write function.

If the underlying BIO is **blocking**, the write functions will only return, once the write operation has been finished or an error occurred.

If the underlying BIO is **nonblocking** the write functions will also return when the underlying BIO could not satisfy the needs of the function to continue the operation. In this case a call to **SSL_get_error**(3) with the return value of the write function will yield **SSL_ERROR_WANT_READ** or **SSL_ERROR_WANT_WRITE**. As at any time a re-negotiation is possible, a call to a write function can also cause read operations! The calling process then must repeat the call after taking appropriate

action to satisfy the needs of the write function. 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.

The write functions will only return with success when the complete contents of **buf** of length **num** has been written. This default behaviour can be changed with the SSL_MODE_ENABLE_PARTIAL_WRITE option of **SSL_CTX_set_mode**(3). When this flag is set the write functions will also return with success when a partial write has been successfully completed. In this case the write function operation is considered completed. The bytes are sent and a new write call with a new buffer (with the already sent bytes removed) must be started. A partial write is performed with the size of a message block, which is 16kB.

WARNINGS

When a write function call has to be repeated because **SSL_get_error**(3) returned **SSL_ERROR_WANT_READ** or **SSL_ERROR_WANT_WRITE**, it must be repeated with the same arguments. The data that was passed might have been partially processed. When **SSL_MODE_ACCEPT_MOVING_WRITE_BUFFER** was set using **SSL_CTX_set_mode**(3) the pointer can be different, but the data and length should still be the same.

You should not call **SSL_write()** with num=0, it will return an error. **SSL_write_ex()** can be called with num=0, but will not send application data to the peer.

RETURN VALUES

SSL_write_ex() will return 1 for success or 0 for failure. Success means that all requested application data bytes have been written to the SSL connection or, if SSL_MODE_ENABLE_PARTIAL_WRITE is in use, at least 1 application data byte has been written to the SSL connection. Failure means that not all the requested bytes have been written yet (if SSL_MODE_ENABLE_PARTIAL_WRITE is not in use) or no bytes could be written to the SSL connection (if SSL_MODE_ENABLE_PARTIAL_WRITE is in use). Failures can be retryable (e.g. the network write buffer has temporarily filled up) or non-retryable (e.g. a fatal network error). In the event of a failure call **SSL_get_error**(3) to find out the reason which indicates whether the call is retryable or not.

For **SSL_write()** the following return values can occur:

> 0 The write operation was successful, the return value is the number of bytes actually written to the TLS/SSL connection.

<= 0
The write operation was not successful, because either the connection was closed, an error

occurred or action must be taken by the calling process. Call **SSL_get_error()** with the return value **ret** to find out the reason.

Old documentation indicated a difference between 0 and -1, and that -1 was retryable. You should instead call **SSL_get_error()** to find out if it's retryable.

For **SSL_sendfile()**, the following return values can occur:

>= 0
The write operation was successful, the return value is the number of bytes of the file written to the TLS/SSL connection. The return value can be less than **size** for a partial write.

< 0 The write operation was not successful, because either the connection was closed, an error occurred or action must be taken by the calling process. Call **SSL_get_error()** with the return value to find out the reason.

SEE ALSO

SSL_get_error(3), **SSL_read_ex**(3), **SSL_read**(3) **SSL_CTX_set_mode**(3), **SSL_CTX_new**(3), **SSL_connect**(3), **SSL_accept**(3) **SSL_set_connect_state**(3), **BIO_ctrl**(3), **ssl**(7), **bio**(7)

HISTORY

The **SSL_write_ex()** function was added in OpenSSL 1.1.1. The **SSL_sendfile()** function was added in OpenSSL 3.0.

COPYRIGHT