NAME

BIO_sendmmsg, BIO_recvmmsg, BIO_dgram_set_local_addr_enable, BIO_dgram_get_local_addr_enable, BIO_dgram_get_local_addr_cap, BIO_err_is_non_fatal − send and receive multiple datagrams in a single call

SYNOPSIS

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

typedef struct bio_msg_st {
    void *data;
    size_t data_len;
    BIO_ADDR *peer, *local;
    uint64_t flags;
} BIO_MSG;

int BIO_sendmmsg(BIO *b, BIO_MSG *msg,
                 size_t stride, size_t num_msg, uint64_t flags,
                 size_t *msgs_processed);
int BIO_recvmmsg(BIO *b, BIO_MSG *msg,
                 size_t stride, size_t num_msg, uint64_t flags,
                 size_t *msgs_processed);

int BIO_dgram_set_local_addr_enable(BIO *b, int enable);
int BIO_dgram_get_local_addr_enable(BIO *b, int *enable);
int BIO_dgram_get_local_addr_cap(BIO *b);
int BIO_err_is_non_fatal(unsigned int errcode);
```

DESCRIPTION

BIO_sendmmsg() and **BIO_recvmmsg()** functions can be used to send and receive multiple messages in a single call to a BIO. They are analogous to **sendmmsg** (2) and **recvmmsg** (2) on operating systems which provide those functions.

The **BIO_MSG** structure provides a subset of the functionality of the **struct msghdr** structure defined by POSIX. These functions accept an array of **BIO_MSG** structures. On any particular invocation, these functions may process all of the passed structures, some of them, or none of them. This is indicated by the value stored in **msgs_processed*, which expresses the number of messages processed.

The caller should set the *data* member of a **BIO_MSG** to a buffer containing the data to send, or to be filled with a received message. *data_len* should be set to the size of the buffer in bytes. If the given **BIO_MSG** is processed (in other words, if the integer returned by the function is greater than or equal to that **BIO_MSG**'s array index), *data_len* will be modified to specify the actual amount of data sent or received.

The *flags* field of a **BIO_MSG** provides input per−message flags to the invocation. If the invocation processes that **BIO_MSG**, the *flags* field is written with output per−message flags, or zero if no such flags are applicable.

Currently, no input or output per−message flags are defined and this field should be set to zero before calling **BIO_sendmmsg()** or **BIO_recvmmsg()**.

The *flags* argument to **BIO_sendmmsg()** and **BIO_recvmmsg()** provides global flags which affect the entire invocation. No global flags are currently defined and this argument should be set to zero.

When these functions are used to send and receive datagrams, the *peer* field of a **BIO_MSG** allows the destination address of sent datagrams to be specified on a per−datagram basis, and the source address of received datagrams to be determined. The *peer* field should be set to point to a **BIO_ADDR**, which will be read by **BIO_sendmmsg()** and used as the destination address for sent datagrams, and written by **BIO_recvmmsg()** with the source address of received datagrams.

Similarly, the *local* field of a **BIO_MSG** allows the source address of sent datagrams to be specified on a per−datagram basis, and the destination address of received datagrams to be determined. Unlike *peer*,

support for *local* must be explicitly enabled on a **BIO** before it can be used; see **BIO_dgram_set_local_addr_enable()**. If *local* is non−NULL in a **BIO_MSG** and support for *local* has not been enabled, processing of that **BIO_MSG** fails.

peer and *local* should be set to NULL if they are not required. Support for *local* may not be available on all platforms; on these platforms, these functions always fail if *local* is non−NULL.

If *local* is specified and local address support is enabled, but the operating system does not report a local address for a specific received message, the **BIO_ADDR** it points to will be cleared (address family set to `AF_UNSPEC`). This is known to happen on Windows when a packet is received which was sent by the local system, regardless of whether the packet's destination address was the loopback address or the IP address of a local non−loopback interface. This is also known to happen on macOS in some circumstances, such as for packets sent before local address support was enabled for a receiving socket. These are OS−specific limitations. As such, users of this API using local address support should expect to sometimes receive a cleared local **BIO_ADDR** instead of the correct value.

The *stride* argument must be set to `sizeof(BIO_MSG)`. This argument facilitates backwards compatibility if fields are added to **BIO_MSG**. Callers must zero−initialize **BIO_MSG**.

num_msg should be sent to the maximum number of messages to send or receive, which is also the length of the array pointed to by *msg*.

msgs_processed must be non−NULL and points to an integer written with the number of messages successfully processed; see the RETURN VALUES section for further discussion.

Unlike most BIO functions, these functions explicitly support multi−threaded use. Multiple concurrent writers and multiple concurrent readers of the same BIO are permitted in any combination. As such, these functions do not clear, set, or otherwise modify BIO retry flags. The return value must be used to determine whether an operation should be retried; see below.

The support for concurrent use extends to **BIO_sendmmsg()** and **BIO_recvmmsg()** only, and no other function may be called on a given BIO while any call to **BIO_sendmmsg()** or **BIO_recvmmsg()** is in progress, or vice versa.

BIO_dgram_set_local_addr_enable() and **BIO_dgram_get_local_addr_enable()** control whether local address support is enabled. To enable local address support, call **BIO_dgram_set_local_addr_enable()** with an argument of 1. The call will fail if local address support is not available for the platform. **BIO_dgram_get_local_addr_enable()** retrieves the value set by **BIO_dgram_set_local_addr_enable()**.

BIO_dgram_get_local_addr_cap() determines if the **BIO** is capable of supporting local addresses.

BIO_err_is_non_fatal() determines if a packed error code represents an error which is transient in nature.

NOTES

Some implementations of the **BIO_sendmmsg()** and **BIO_recvmmsg()** BIO methods might always process at most one message at a time, for example when OS−level functionality to transmit or receive multiple messages at a time is not available.

RETURN VALUES

On success, the functions **BIO_sendmmsg()** and **BIO_recvmmsg()** return 1 and write the number of messages successfully processed (which need not be nonzero) to *msgs_processed*. Where a positive value n is written to *msgs_processed*, all entries in the **BIO_MSG** array from 0 through n−1 inclusive have their *data_len* and *flags* fields updated with the results of the operation on that message. If the call was to **BIO_recvmmsg()** and the *peer* or *local* fields of that message are non−NULL, the **BIO_ADDR** structures they point to are written with the relevant address.

On failure, the functions **BIO_sendmmsg()** and **BIO_recvmmsg()** return 0 and write zero to *msgs_processed*. Thus *msgs_processed* is always written regardless of the outcome of the function call.

If **BIO_sendmmsg()** and **BIO_recvmmsg()** fail, they always raise an **ERR_LIB_BIO** error using **ERR_raise** (3). Any error may be raised, but the following in particular may be noted:

BIO_R_LOCAL_ADDR_NOT_AVAILABLE
The *local* field was set to a non−NULL value, but local address support is not available or not enabled on the BIO.

BIO_R_PEER_ADDR_NOT_AVAILABLE
The *peer* field was set to a non−NULL value, but peer address support is not available on the BIO.

BIO_R_UNSUPPORTED_METHOD
The **BIO_sendmmsg()** or **BIO_recvmmsg()** method is not supported on the BIO.

BIO_R_NON_FATAL
The call failed due to a transient, non−fatal error (for example, because the BIO is in nonblocking mode and the call would otherwise have blocked).

Implementations of this interface which do not make system calls and thereby pass through system error codes using **ERR_LIB_SYS** (for example, memory−based implementations) should issue this reason code to indicate a transient failure. However, users of this interface should not test for this reason code directly, as there are multiple possible packed error codes representing a transient failure; use **BIO_err_is_non_fatal()** instead (discussed below).

Socket errors
OS−level socket errors are reported using an error with library code **ERR_LIB_SYS**; for a packed error code **errcode** where `ERR_SYSTEM_ERROR(errcode) == 1`, the OS−level socket error code can be retrieved using `ERR_GET_REASON(errcode)`. The packed error code can be retrieved by calling **ERR_peek_last_error**(3) after the call to **BIO_sendmmsg()** or **BIO_recvmmsg()** returns 0.

Non−fatal errors
Whether an error is transient can be determined by passing the packed error code to **BIO_err_is_non_fatal()**. Callers should do this instead of testing the reason code directly, as there are many possible error codes which can indicate a transient error, many of which are system specific.

Third parties implementing custom BIOs supporting the **BIO_sendmmsg()** or **BIO_recvmmsg()** methods should note that it is a required part of the API contract that an error is always raised when either of these functions return 0.

BIO_dgram_set_local_addr_enable() returns 1 if local address support was successfully enabled or disabled and 0 otherwise.

BIO_dgram_get_local_addr_enable() returns 1 if the local address support enable flag was successfully retrieved.

BIO_dgram_get_local_addr_cap() returns 1 if the **BIO** can support local addresses.

BIO_err_is_non_fatal() returns 1 if the passed packed error code represents an error which is transient in nature.

HISTORY

These functions were added in OpenSSL 3.2.

COPYRIGHT