



Rocky Enterprise Linux 9.2 Manual Pages on command 'MSG_LEN.3'

C:\>man MSG_LEN.3

MSG(3) Linux Programmer's Manual MSG(3)

NAME

MSG_ALIGN, MSG_SPACE, MSG_NXTHDR, MSG_FIRSTHDR - access ancillary data

SYNOPSIS

```
#include <sys/socket.h>

struct cmsghdr *MSG_FIRSTHDR(struct msghdr *msg);
struct cmsghdr *MSG_NXTHDR(struct msghdr *msg ,
                             struct cmsghdr *cmsg);

size_t MSG_ALIGN(size_t length);
size_t MSG_SPACE(size_t length);
size_t MSG_LEN(size_t length);
unsigned char *MSG_DATA(struct cmsghdr *cmsg);
```

DESCRIPTION

These macros are used to create and access control messages (also called ancillary data) that are not a part of the socket payload. This control information may include the interface the packet was received on, various rarely used header fields, an extended error description, a set of file descriptors, or UNIX credentials. For instance, control messages can be used to send additional header fields such as IP options. Ancillary data is sent by calling `sendmsg(2)` and received by calling `recvmsg(2)`. See their manual pages for more information.

Ancillary data is a sequence of `cmsghdr` structures with appended data. See the specific protocol man pages for the available control message types. The maximum

ancillary buffer size allowed per socket can be set using `/proc/sys/net/core/optmem_max`; see `socket(7)`.

The `cmsghdr` structure is defined as follows:

```
struct cmsghdr {
    size_t msg_len; /* Data byte count, including header
                    (type is socklen_t in POSIX) */
    int    msg_level; /* Originating protocol */
    int    msg_type; /* Protocol-specific type */
    /* followed by
     unsigned char msg_data[]; */
};
```

The sequence of `cmsghdr` structures should never be accessed directly. Instead, use only the following macros:

- * `MSG_FIRSTHDR()` returns a pointer to the first `cmsghdr` in the ancillary data buffer associated with the passed `msg_hdr`. It returns `NULL` if there isn't enough space for a `cmsghdr` in the buffer.
- * `MSG_NXTHDR()` returns the next valid `cmsghdr` after the passed `cmsghdr`. It returns `NULL` when there isn't enough space left in the buffer.

When initializing a buffer that will contain a series of `cmsghdr` structures (e.g., to be sent with `sendmsg(2)`), that buffer should first be zero-initialized to ensure the correct operation of `MSG_NXTHDR()`.

- * `MSG_ALIGN()`, given a length, returns it including the required alignment. This is a constant expression.
- * `MSG_SPACE()` returns the number of bytes an ancillary element with payload of the passed data length occupies. This is a constant expression.
- * `MSG_DATA()` returns a pointer to the data portion of a `cmsghdr`. The pointer returned cannot be assumed to be suitably aligned for accessing arbitrary payload data types. Applications should not cast it to a pointer type matching the payload, but should instead use `memcpy(3)` to copy data to or from a suitably declared object.
- * `MSG_LEN()` returns the value to store in the `msg_len` member of the `cmsghdr` structure, taking into account any necessary alignment. It takes the data length as an argument. This is a constant expression.

To create ancillary data, first initialize the `msg_control` member of the `msg_hdr` with the length of the control message buffer. Use `MSG_FIRSTHDR()` on the `msg_hdr` to get the first control message and `MSG_NXTHDR()` to get all subsequent ones. In each control message, initialize `msg_len` (with `MSG_LEN()`), the other `msg_hdr` header fields, and the data portion using `MSG_DATA()`. Finally, the `msg_control` field of the `msg_hdr` should be set to the sum of the `MSG_SPACE()` of the length of all control messages in the buffer. For more information on the `msg_hdr`, see `recvmsg(2)`.

CONFORMING TO

This ancillary data model conforms to the POSIX.1g draft, 4.4BSD-Lite, the IPv6 advanced API described in RFC 2292 and SUSv2. `MSG_ALIGN()` is a Linux extension.

NOTES

For portability, ancillary data should be accessed using only the macros described here. `MSG_ALIGN()` is a Linux extension and should not be used in portable programs.

In Linux, `MSG_LEN()`, `MSG_DATA()`, and `MSG_ALIGN()` are constant expressions (assuming their argument is constant), meaning that these values can be used to declare the size of global variables. This may not be portable, however.

EXAMPLE

This code looks for the `IP_TTL` option in a received ancillary buffer:

```
struct msg_hdr msgh;
struct cmsghdr *cmsg;
int received_ttl;
/* Receive auxiliary data in msgh */
for (cmsg = MSG_FIRSTHDR(&msgh); cmsg != NULL;
     cmsg = MSG_NXTHDR(&msgh, cmsg)) {
    if (cmsg->cmsg_level == IPPROTO_IP
        && cmsg->cmsg_type == IP_TTL) {
        memcpy(&received_ttl, MSG_DATA(cmsg), sizeof(int));
        break;
    }
}
if (cmsg == NULL) {
```

```

    /* Error: IP_TTL not enabled or small buffer or I/O error */
}

```

The code below passes an array of file descriptors over a UNIX domain socket using

SCM_RIGHTS:

```

struct msghdr msg = { 0 };
struct cmsghdr *cmsg;
int myfds[NUM_FD]; /* Contains the file descriptors to pass */
char iobuf[1];
struct iovec io = {
    .iov_base = iobuf,
    .iov_len = sizeof(iobuf)
};
union {
    /* Ancillary data buffer, wrapped in a union
       in order to ensure it is suitably aligned */
    char buf[CMSG_SPACE(sizeof(myfds))];
    struct cmsghdr align;
} u;
msg.msg_iov = &io;
msg.msg_iovlen = 1;
msg.msg_control = u.buf;
msg.msg_controllen = sizeof(u.buf);
cmsg = CMSG_FIRSTHDR(&msg);
cmsg->cmsg_level = SOL_SOCKET;
cmsg->cmsg_type = SCM_RIGHTS;
cmsg->cmsg_len = CMSG_LEN(sizeof(int) * NUM_FD);
memcpy(CMSG_DATA(cmsg), myfds, NUM_FD * sizeof(int));

```

SEE ALSO

recvmsg(2), sendmsg(2)

RFC 2292

COLOPHON

This page is part of release 5.05 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at <https://www.kernel.org/doc/man-pages/>.

