

**NAME**

CMMSG\_ALIGN, CMMSG\_SPACE, CMMSG\_NXTHDR, CMMSG\_FIRSTHDR – access ancillary data

**SYNOPSIS**

```
#include <sys/socket.h>
struct cmsghdr *CMMSG_FIRSTHDR(struct msghdr *msgh);
struct cmsghdr *CMMSG_NXTHDR(struct msghdr *msgh,
                             struct cmsghdr *cmsg);
size_t CMMSG_ALIGN(size_t length);
size_t CMMSG_SPACE(size_t length);
size_t CMMSG_LEN(size_t length);
unsigned char *CMMSG_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 cmsg_len;      /* Data byte count, including header
                           (type is socklen_t in POSIX) */
    int     cmsg_level;    /* Originating protocol */
    int     cmsg_type;     /* Protocol-specific type */
    /* followed by
       unsigned char cmsg_data[]; */
};
```

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

- \* **CMMSG\_FIRSTHDR()** returns a pointer to the first *cmsghdr* in the ancillary data buffer associated with the passed *msghdr*. It returns NULL if there isn't enough space for a *cmsghdr* in the buffer.
- \* **CMMSG\_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 **CMMSG\_NXTHDR()**.

- \* **CMMSG\_ALIGN()**, given a length, returns it including the required alignment. This is a constant expression.
- \* **CMMSG\_SPACE()** returns the number of bytes an ancillary element with payload of the passed data length occupies. This is a constant expression.
- \* **CMMSG\_DATA()** returns a pointer to the data portion of a *cmsghdr*.
- \* **CMMSG\_LEN()** returns the value to store in the *cmsg\_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\_controllen* member of the *msghdr* with the length of the control message buffer. Use **CMMSG\_FIRSTHDR()** on the *msghdr* to get the first control message and

**CMMSG\_NXTHDR()** to get all subsequent ones. In each control message, initialize *cmsg\_len* (with **CMMSG\_LEN()**), the other *cmsg\_hdr* header fields, and the data portion using **CMMSG\_DATA()**. Finally, the *msg\_controllen* field of the *msg\_hdr* should be set to the sum of the **CMMSG\_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. **CMMSG\_ALIGN()** is a Linux extension.

## NOTES

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

In Linux, **CMMSG\_LEN()**, **CMMSG\_DATA()**, and **CMMSG\_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 msghdr msgh;
struct cmsghdr *cmsg;
int *ttlptr;
int received_ttl;

/* Receive auxiliary data in msgh */

for (cmsg = CMMSG_FIRSTHDR(&msgh); cmsg != NULL;
     cmsg = CMMSG_NXTHDR(&msgh, cmsg)) {
    if (cmsg->cmsg_level == IPPROTO_IP
        && cmsg->cmsg_type == IP_TTL) {
        ttlptr = (int *) CMMSG_DATA(cmsg);
        received_ttl = *ttlptr;
        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[CMMSG_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 = CMMSG_FIRSTHDR(&msg);
cmsg->cmsg_level = SOL_SOCKET;
cmsg->cmsg_type = SCM_RIGHTS;
cmsg->cmsg_len = CMMSG_LEN(sizeof(int) * NUM_FD);
memcpy(CMMSG_DATA(cmsg), myfds, NUM_FD * sizeof(int));
```

**SEE ALSO**

**recvmsg(2), sendmsg(2)**

RFC 2292

**COLOPHON**

This page is part of release 5.02 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/>.