## NAME

CMSG_ALIGN, CMSG_SPACE, CMSG_NXTHDR, CMSG_FIRSTHDR - access ancillary data

## SYNOPSIS

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

**struct cmsghdr *CMSG_FIRSTHDR(struct msghdr *_msgh_);**
**struct cmsghdr *CMSG_NXTHDR(struct msghdr *_msgh_, struct cmsghdr *_cmsg_);**
**size_t CMSG_ALIGN(size_t _length_);**
**size_t CMSG_SPACE(size_t _length_);**
**size_t CMSG_LEN(size_t _length_);**
**unsigned char *CMSG_DATA(struct cmsghdr *_cmsg_);**

```
struct cmsghdr {
socklen_t cmsg_len; /* data byte count, including header */
int cmsg_level; /* originating protocol */
int cmsg_type; /* protocol-specific type */
/* followed by unsigned char cmsg_data[]; */
};
```

## 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 *struct cmsghdr* structures with appended data. This sequence should be accessed using only the macros described in this manual page and never directly. 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).

**CMSG_FIRSTHDR**() returns a pointer to the first *cmsghdr* in the ancillary data buffer associated with the passed *msghdr*.

**CMSG_NXTHDR**() returns the next valid *cmsghdr* after the passed *cmsghdr*. It returns NULL when there isn't enough space left in the buffer.

**CMSG_ALIGN**(), given a length, returns it including the required alignment. This is a constant expression.

**CMSG_SPACE**() returns the number of bytes an ancillary element with payload of the passed data length occupies. This is a constant expression.

**CMSG_DATA**() returns a pointer to the data portion of a *cmsghdr*.

**CMSG_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**CMSG_FIRSTHDR**() on the *msghdr* to get the first control message and **CMSG_NXTHDR**() to get all subsequent ones. In each control message, initialize *cmsg_len* (with **CMSG_LEN**()), the other *cmsghdr* header fields, and the data portion using **CMSG_DATA**(). Finally, the *msg_controllen* field of the *msghdr* should be set to the sum of the **CMSG_SPACE**() of the length of all control messages in the buffer. For more information on the *msghdr*, see recvmsg(2).

When the control message buffer is too short to store all messages, the **MSG_CTRUNC** flag is set in the *msg_flags* member of the *msghdr*.

## 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. **CMSG_ALIGN**() is a Linux extension.

## NOTES

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

In Linux, **CMSG_LEN**(), **CMSG_DATA**(), and **CMSG_ALIGN**() are constant expressions (assuming their argument is constant); this could 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 = CMSG_FIRSTHDR(&msgh); cmsg != NULL;
cmsg = CMSG_NXTHDR(&msgh,cmsg)) {
if (cmsg->cmsg_level == IPPROTO_IP
&& cmsg->cmsg_type == IP_TTL) {
ttlptr = (int *) CMSG_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 buf[CMSG_SPACE(sizeof myfds)]; /* ancillary data buffer */
int *fdptr;

msg.msg_control = buf;
msg.msg_controllen = sizeof buf;
cmsg = CMSG_FIRSTHDR(&msg);
cmsg->cmsg_level = SOL_SOCKET;
cmsg->cmsg_type = SCM_RIGHTS;
cmsg->cmsg_len = CMSG_LEN(sizeof(int) * NUM_FD);
/* Initialize the payload: */
fdptr = (int *) CMSG_DATA(cmsg);
memcpy(fdptr, myfds, NUM_FD * sizeof(int));
/* Sum of the length of all control messages in the buffer: */
msg.msg_controllen = cmsg->cmsg_len;
```

## SEE ALSO

recvmsg(2), sendmsg(2)

RFC 2292

## COLOPHON

This page is part of release 3.74 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 http://www.kernel.org/doc/man-pages/.