NAME

CMSG_ALIGN, CMSG_SPACE, CMSG_NXTHDR, CMSG_FIRSTHDR − access ancillary data

LIBRARY

Standard C library (*libc*, *−lc*)

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*);**

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:

CMSG_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.

CMSG_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 **CMSG_NXTHDR**().

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*. 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.

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).

VERSIONS

For portability, ancillary data should be accessed using only the macros described here.

In Linux, **CMSG_LEN**(), **CMSG_DATA**(), and **CMSG_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.

STANDARDS

CMSG_FIRSTHDR()
CMSG_NXTHDR()
CMSG_DATA()
: POSIX.1-2008.

CMSG_SPACE()
CMSG_LEN()
CMSG_ALIGN()
: Linux.

HISTORY

This ancillary data model conforms to the POSIX.1g draft, 4.4BSD-Lite, the IPv6 advanced API described in RFC 2292 and SUSv2.

CMSG_SPACE() and **CMSG_LEN**() will be included in the next POSIX release (Issue 8).

EXAMPLES

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

```
struct msghdr msgh;
struct cmsghdr *cmsg;
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) {
        memcpy(&receive_ttl, CMSG_DATA(cmsg), sizeof(received_ttl));
        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(myfds));
memcpy(CMSG_DATA(cmsg), myfds, sizeof(myfds));
```

For a complete code example that shows passing of file descriptors over a UNIX domain socket, see **seccomp_unotify**(2).

SEE ALSO

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

RFC 2292