NAME

inet6_opt_init, **inet6_opt_append**, **inet6_opt_finish**, **inet6_opt_set_val**, **inet6_opt_next**, **inet6_opt_find**, **inet6_opt_get_val** - IPv6 Hop-by-Hop and Destination Options manipulation

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <netinet/in.h>

int
inet6_opt_init(*void *extbuf, socklen_t extlen*);

int
inet6_opt_append(*void *extbuf, socklen_t extlen, int offset, uint8_t type, socklen_t len, uint8_t align, void **databufp*);

int
inet6_opt_finish(*void *extbuf, socklen_t extlen, int offset*);

int
inet6_opt_set_val(*void *databuf, int offset, void *val, socklen_t vallen*);

int
inet6_opt_next(*void *extbuf, socklen_t extlen, int offset, uint8_t *typep, socklen_t *lenp, void **databufp*);

int
inet6_opt_find(*void *extbuf, socklen_t extlen, int offset, uint8_t type, socklen_t *lenp, void **databufp*);

int
inet6_opt_get_val(*void *databuf, int offset, void *val, socklen_t vallen*);

DESCRIPTION

Building and parsing the Hop-by-Hop and Destination options is complicated. The advanced sockets API defines a set of functions to help applications create and manipulate Hop-by-Hop and Destination options. This man page describes the functions specified in IETF Draft RFC 3542. These functions use the formatting rules specified in Appendix B in RFC 2460, i.e., that the largest field is placed last in the option. The function prototypes for these functions are all contained in the *<netinet/in.h>* header file.

inet6_opt_init

The **inet6_opt_init**() function returns the number of bytes needed for an empty extension header, one without any options. If the *extbuf* argument points to a valid section of memory then the **inet6_opt_init**() function also initializes the extension header's length field. When attempting to initialize an extension buffer passed in the *extbuf* argument, *extlen* must be a positive multiple of 8 or else the function fails and returns -1 to the caller.

inet6_opt_append

The **inet6_opt_append**() function can perform two different jobs. When a valid *extbuf* argument is supplied it appends an option to the extension buffer and returns the updated total length as well as a pointer to the newly created option in *databufp*. If the value of *extbuf* is NULL then the **inet6_opt_append**() function only reports what the total length would be if the option were actually appended. The *len* and *align* arguments specify the length of the option and the required data alignment which must be used when appending the option. The *offset* argument should be the length returned by the **inet6_opt_init**() function or a previous call to **inet6_opt_append**().

The *type* argument is the 8-bit option type.

After **inet6_opt_append**() has been called, the application can use the buffer pointed to by *databufp* directly, or use **inet6_opt_set_val**() to specify the data to be contained in the option.

Option types of 0 and 1 are reserved for the Pad1 and PadN options. All other values from 2 through 255 may be used by applications.

The length of the option data is contained in an 8-bit value and so may contain any value from 0 through 255.

The *align* parameter must have a value of 1, 2, 4, or 8 and cannot exceed the value of *len*. The alignment values represent no alignment, 16 bit, 32 bit and 64 bit alignments, respectively.

inet6_opt_finish

The **inet6_opt_finish**() function calculates the final padding necessary to make the extension header a multiple of 8 bytes, as required by the IPv6 extension header specification, and returns the extension header's updated total length. The *offset* argument should be the length returned by **inet6_opt_init**() or **inet6_opt_append**(). When *extbuf* is not NULL the function also sets up the appropriate padding bytes by inserting a Pad1 or PadN option of the proper length.

If the extension header is too small to contain the proper padding then an error of -1 is returned to the caller.

inet6_opt_set_val

The **inet6_opt_set_val**() function inserts data items of various sizes into the data portion of the option. The *databuf* argument is a pointer to memory that was returned by the **inet6_opt_append**() call and the *offset* argument specifies where the option should be placed in the data buffer. The *val* argument points to an area of memory containing the data to be inserted into the extension header, and the *vallen* argument indicates how much data to copy.

The caller should ensure that each field is aligned on its natural boundaries as described in Appendix B of RFC 2460.

The function returns the offset for the next field which is calculated as *offset* + *vallen* and is used when composing options with multiple fields.

inet6_opt_next

The **inet6_opt_next**() function parses received extension headers. The *extbuf* and *extlen* arguments specify the location and length of the extension header being parsed. The *offset* argument should either be zero, for the first option, or the length value returned by a previous call to **inet6_opt_next**() or **inet6_opt_find**(). The return value specifies the position where to continue scanning the extension buffer. The option is returned in the arguments *typep*, *lenp*, and *databufp*, which point to the 8-bit option type, the 8-bit option length and the option data, respectively. This function does not return any PAD1 or PADN options. When an error occurs or there are no more options, the return value is -1.

inet6_opt_find

The **inet6_opt_find**() function searches the extension buffer for a particular option type, passed in through the *type* argument. If the option is found then the *lenp* and *databufp* arguments are updated to point to the option’s length and data, respectively. The *extbuf* and *extlen* arguments must point to a valid extension buffer and give its length. The *offset* argument can be used to search from a location anywhere in the extension header.

inet6_opt_get_val

The **inet6_opt_get_val**() function extracts data items of various sizes in the data portion of the option. The *databuf* is a pointer returned by the **inet6_opt_next**() or **inet6_opt_find**() functions. The *val* argument points to where the data will be extracted. The *offset* argument specifies from where in the data portion of the option the value should be extracted; the first byte of option data is specified by an offset of zero.

It is expected that each field is aligned on its natural boundaries as described in Appendix B of RFC 2460.

The function returns the offset for the next field by calculating *offset* + *vallen* which can be used when

extracting option content with multiple fields. Robust receivers must verify alignment before calling this function.

RETURN VALUES

All the functions return -1 on an error.

EXAMPLES

RFC 3542 gives comprehensive examples in Section 22.

KAME also provides examples in the *advapitest* directory of its kit.

SEE ALSO

W. Stevens, M. Thomas, E. Nordmark, and T. Jinmei, *Advanced Sockets API for IPv6*, RFC 3542, October 2002.

S. Deering and R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, RFC 2460, December 1998.

STANDARDS

The functions are documented in "Advanced Sockets API for IPv6" (RFC 3542).

HISTORY

The implementation first appeared in KAME advanced networking kit.