NAME

cap_from_text, cap_to_text, cap_to_name, cap_from_name − capability state textual representation translation

SYNOPSIS

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

cap_t cap_from_text(const char *buf_p);
char *cap_to_text(cap_t caps, ssize_t *len_p);
int cap_from_name(const char *name, cap_value_t *cap_p);
char *cap_to_name(cap_value_t cap);
```

Link with −*lcap*.

DESCRIPTION

These functions translate a capability state between an internal representation and a textual one. The internal representation is managed by the capability functions in working storage. The textual representation is a structured, human-readable string suitable for display.

cap_from_text() allocates and initializes a capability state in working storage. It then sets the contents of this newly created capability state to the state represented by a human-readable, nul-terminated character string pointed to by *buf_p*. It returns a pointer to the newly created capability state. When the capability state in working storage is no longer required, the caller should free any releasable memory by calling **cap_free**() with *cap_t* as an argument. The function returns an error if it cannot parse the contents of the string pointed to by *buf_p* or does not recognize any *capability_name* or flag character as valid. The function also returns an error if any flag is both set and cleared within a single clause.

cap_to_text() converts the capability state in working storage identified by *caps* into a nul-terminated human-readable string. This function allocates any memory necessary to contain the string, and returns a pointer to the string. If the pointer *len_p* is not NULL, the function shall also return the full length of the string (not including the nul terminator) in the location pointed to by *len_p*. The capability state in working storage, identified by *caps*, is completely represented in the character string. When the capability state in working storage is no longer required, the caller should free any releasable memory by calling **cap_free**() with the returned string pointer as an argument.

cap_from_name() converts a text representation of a capability, such as "cap_chown", to its numerical representation (**CAP_CHOWN=0**), writing the decoded value into**cap_p*. If*cap_p* is NULL no result is written, but the return code of the function indicates whether or not the specified capability can be represented by the library.

cap_to_name() converts a capability index value, *cap*, to a libcap-allocated textual string. This string should be deallocated with **cap_free**().

TEXTUAL REPRESENTATION

The text format is described in the **cap_text_formats**(7) man page.

RETURN VALUE

cap_from_text(), **cap_to_text**() and **cap_to_name**() return a non-NULL value on success, and NULL on failure. **cap_from_name**() returns 0 for success, and −1 on failure (unknown capability).

On failure, *errno* is set to **EINVAL**, or **ENOMEM**.

CONFORMING TO

cap_from_text() and **cap_to_text**() are specified by the withdrawn POSIX.1e draft specification. **cap_from_name**() and **cap_to_name**() are Linux extensions.

EXAMPLE

The example program below demonstrates the use of **cap_from_text**() and **cap_to_text**(). The following shell session shows some example runs:

```
$ ./a.out "cap_chown=p cap_chown+e"
```

```
caps_to_text() returned "cap_chown=ep"
$ ./a.out "all=pe cap_chown−e cap_kill−pe"
caps_to_text() returned "=ep cap_chown−e cap_kill−ep"
```

The source code of the program is as follows:

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/capability.h>

#define handle_error(msg) \
   do { perror(msg); exit(EXIT_FAILURE); } while (0)

int
main(int argc, char *argv[])
{
   cap_t caps;
   char *txt_caps;

   if (argc != 2) {
      fprintf(stderr, "%s <textual−cap−set>\n", argv[0]);
      exit(EXIT_FAILURE);
   }

   caps = cap_from_text(argv[1]);
   if (caps == NULL)
      handle_error("cap_from_text");

   txt_caps = cap_to_text(caps, NULL);
   if (txt_caps == NULL)
      handle_error("cap_to_text");

   printf("caps_to_text() returned \"%s\"\n", txt_caps);

   if (cap_free(txt_caps) != 0 || cap_free(caps) != 0)
      handle_error("cap_free");

   exit(EXIT_SUCCESS);
}
```

SEE ALSO

libcap(3), **cap_clear**(3), **cap_copy_ext**(3), **cap_get_file**(3), **cap_get_proc**(3), **cap_init**(3), **cap_text_formats**(7), **capabilities**(7)