NAME

cap_clear, cap_clear_flag, cap_get_flag, cap_set_flag, cap_fill_flag, cap_fill, cap_compare, cap_max_bits − capability data object manipulation

SYNOPSIS

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

int cap_clear(cap_t cap_p);
int cap_clear_flag(cap_t cap_p, cap_flag_t flag);
int cap_get_flag(cap_t cap_p, cap_value_t cap,
        cap_flag_t flag, cap_flag_value_t *value_p);
int cap_set_flag(cap_t cap_p, cap_flag_t flag, int ncap,
        const cap_value_t *caps, cap_flag_value_t value);
int cap_fill_flag(cap_t cap_p, cap_flag_t to,
         const cap_t ref, cap_flag_t from);
int cap_fill(cap_t cap_p, cap_flag_t to, cap_flag_t from);
int cap_compare(cap_t cap_a, cap_t cap_b);
cap_value_t cap_max_bits();
```

Link with −*lcap*.

DESCRIPTION

These functions work on a capability state held in working storage. A *cap_t* holds information about the capabilities in each of the three flags, Permitted, Inheritable, and Effective. Each capability in a set may be clear (disabled, 0) or set (enabled, 1).

These functions work with the following data types:

cap_value_t identifies a capability, such as **CAP_CHOWN**.

cap_flag_t identifies one of the three flags associated with a capability (i.e., it identifies one of the three capability dimensions). Valid values for this type are **CAP_EFFECTIVE**, **CAP_INHERITABLE** or **CAP_PERMITTED**.

cap_flag_value_t identifies the setting of a particular capability flag (i.e, the value of a capability in a set). Valid values for this type are **CAP_CLEAR** (0) or **CAP_SET** (1).

cap_clear() initializes the capability state in working storage identified by *cap_p* so that all capability flags are cleared.

cap_clear_flag() clears all of the capabilities of the specified capability flag, *flag*.

cap_get_flag() obtains the current value of the capability flag, *flag*, of the capability, *cap*, from the capability state identified by *cap_p* and places it in the location pointed to by *value_p*.

cap_set_flag() sets the flag, *flag*, of each capability in the array *caps* in the capability state identified by *cap_p* to *value*. The argument, *ncap*, is used to specify the number of capabilities in the array, *caps*.

cap_fill_flag() fills the to flag of one capability set, with the values in the from flag of a reference capability set.

cap_fill() fills the to flag values by copying all of the from flag values.

cap_compare() compares two full capability sets and, in the spirit of **memcmp**(), returns zero if the two capability sets are identical. A positive return *value* indicates there is a difference between them. The returned *value* carries further information about the **cap_flag_t** *flag* differences. Specifically, the macro **CAP_DIFFERS** (*value*, *flag*) evaluates to non-zero if the returned *value* differs in its *flag* components.

cap_max_bits() returns the number of capability values known to the running kernel. This may differ from libcap's list known at compilation time. Unnamed, at compilation time, capabilites can be referred to numerically and libcap will handle them appropriately. Note, the running kernel wins and it gets to define what "all" capabilities means.

RETURN VALUE

cap_clear(), **cap_clear_flag**(), **cap_get_flag**() **cap_set_flag**() and **cap_compare**() return zero on success, and −1 on failure. Other return values for **cap_compare**() are described above. The function **cap_max_bits**() returns a numeric value of type **cap_value_t** that is one larger than the largest actual value known to the running kernel.

On failure, *errno* is set to **EINVAL**, indicating that one of the arguments is invalid.

CONFORMING TO

These functions are mostly as per specified in the withdrawn POSIX.1e draft specification. The following are Linux extensions: **cap_fill**(), **cap_fill_flag**(), **cap_clear_flag**(), **cap_compare**() and **cap_max_bits**().

SEE ALSO

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