## NAME

**clearenv**, **getenv**, **putenv**, **secure_getenv**, **setenv**, **unsetenv** - environment variable functions

## LIBRARY

Standard C Library (libc, -lc)

## SYNOPSIS

**#include <stdlib.h>**

*int*
**clearenv**(*void*);

*char **
**getenv**(*const char *name*);

*char **
**secure_getenv**(*const char *name*);

*int*
**setenv**(*const char *name*, *const char *value*, *int overwrite*);

*int*
**putenv**(*char *string*);

*int*
**unsetenv**(*const char *name*);

## DESCRIPTION

These functions set, unset and fetch environment variables from the host *environment list*.

The **clearenv**() function clears all environment variables. New variables can be added using **setenv**() and **putenv**().

The **getenv**() function obtains the current value of the environment variable, *name*. The application should not modify the string pointed to by the **getenv**() function.

The **secure_getenv**() returns *NULL* when the environment cannot be trusted, otherwise it acts like **getenv**(). The environment currently is not trusted when issetugid(3) returns a non-zero value, but other conditions may be added in the future.

The **setenv**() function inserts or resets the environment variable *name* in the current environment list. If the variable *name* does not exist in the list, it is inserted with the given *value*. If the variable does exist, the argument *overwrite* is tested; if *overwrite* is zero, the variable is not reset, otherwise it is reset to the given *value*.

The **putenv**() function takes an argument of the form ‘‘name=value’’ and puts it directly into the current environment, so altering the argument shall change the environment. If the variable *name* does not exist in the list, it is inserted with the given *value*. If the variable *name* does exist, it is reset to the given *value*.

The **unsetenv**() function deletes all instances of the variable name pointed to by *name* from the list.

If corruption (e.g., a name without a value) is detected while making a copy of environ for internal usage, then **setenv**(), **unsetenv**() and **putenv**() will output a warning to stderr about the issue, drop the corrupt entry and complete the task without error.

## RETURN VALUES

The **getenv**() function returns the value of the environment variable as a NUL-terminated string. If the variable *name* is not in the current environment, NULL is returned.

The **secure_getenv**() function returns NULL if the process is in "secure execution," otherwise it will call **getenv**().

The **clearenv**(), **setenv**(), **putenv**(), and **unsetenv**() functions return the value 0 if successful; otherwise the value -1 is returned and the global variable *errno* is set to indicate the error.

## ERRORS

[EINVAL] The function **getenv**(), **setenv**() or **unsetenv**() failed because the *name* is a NULL pointer, points to an empty string, or points to a string containing an "=" character.

The function **putenv**() failed because *string* is a NULL pointer, *string* is without an "=" character or "=" is the first character in *string*. This does not follow the POSIX specification.

[ENOMEM] The function **setenv**(), **unsetenv**() or **putenv**() failed because they were unable to allocate memory for the environment.

## SEE ALSO

csh(1), sh(1), execve(2), environ(7)

## STANDARDS

The **getenv**() function conforms to ISO/IEC 9899:1990 ("ISO C90"). The **setenv**(), **putenv**() and **unsetenv**() functions conforms to IEEE Std 1003.1-2001 ("POSIX.1"). The **secure_getenv**() function is expected to be glibc-compatible.

## HISTORY

The functions **setenv**() and **unsetenv**() appeared in Version 7 AT&T UNIX. The **putenv**() function appeared in 4.3BSD-Reno.

Until FreeBSD 7.0, **putenv**() would make a copy of *string* and insert it into the environment using **setenv**(). This was changed to use *string* as the memory location of the ‘‘name=value’’ pair to follow the POSIX specification.

The **clearenv**() and **secure_getenv**() functions were added in FreeBSD 14.

## BUGS

Successive calls to **setenv**() that assign a larger-sized *value* than any previous value to the same *name* will result in a memory leak. The FreeBSD semantics for this function (namely, that the contents of *value* are copied and that old values remain accessible indefinitely) make this bug unavoidable. Future versions may eliminate one or both of these semantic guarantees in order to fix the bug.