25.4.1 Environment Access

The value of an environment variable can be accessed with the `getenv` function. This is declared in the header file `stdlib.h`.

Libraries should use `secure_getenv` instead of `getenv`, so that they do not accidentally use untrusted environment variables. Modifications of environment variables are not allowed in multi-threaded programs. The `getenv` and `secure_getenv` functions can be safely used in multi-threaded programs.

Function: *char* * **getenv** *(const char *name)*

> Preliminary: | MT-Safe env | AS-Safe | AC-Safe | See POSIX Safety Concepts.
>
> This function returns a string that is the value of the environment variable *name*. You must not modify this string. In some non-Unix systems not using the GNU C Library, it might be overwritten by subsequent calls to `getenv` (but not by any other library function). If the environment variable *name* is not defined, the value is a null pointer.

Function: *char* * **secure_getenv** *(const char *name)*

> Preliminary: | MT-Safe env | AS-Safe | AC-Safe | See POSIX Safety Concepts.
>
> This function is similar to `getenv`, but it returns a null pointer if the environment is untrusted. This happens when the program file has SUID or SGID bits set. General-purpose libraries should always prefer this function over `getenv` to avoid vulnerabilities if the library is referenced from a SUID/SGID program.
>
> This function is a GNU extension.

Function: *int* **putenv** *(char *string)*

> Preliminary: | MT-Unsafe const:env | AS-Unsafe heap lock | AC-Unsafe corrupt lock mem | See POSIX Safety Concepts.
>
> The `putenv` function adds or removes definitions from the environment. If the *string* is of the form ‘`name=value`’, the definition is added to the environment. Otherwise, the *string* is interpreted as the name of an environment variable, and any definition for this variable in the environment is removed.
>
> If the function is successful it returns `0`. Otherwise the return value is nonzero and `errno` is set to indicate the error.

The difference to the `setenv` function is that the exact string given as the parameter *string* is put into the environment. If the user should change the string after the `putenv` call this will reflect automatically in the environment. This also requires that *string* not be an automatic variable whose scope is left before the variable is removed from the environment. The same applies of course to dynamically allocated variables which are freed later.

This function is part of the extended Unix interface. You should define *_XOPEN_SOURCE* before including any header.

Function: *int* **setenv** *(const char *name, const char *value, int replace)*

Preliminary: | MT-Unsafe const:env | AS-Unsafe heap lock | AC-Unsafe corrupt lock mem | See POSIX Safety Concepts.

The `setenv` function can be used to add a new definition to the environment. The entry with the name *name* is replaced by the value ‘*`name=value`*’. Please note that this is also true if *value* is the empty string. To do this a new string is created and the strings *name* and *value* are copied. A null pointer for the *value* parameter is illegal. If the environment already contains an entry with key *name* the *replace* parameter controls the action. If replace is zero, nothing happens. Otherwise the old entry is replaced by the new one.

Please note that you cannot remove an entry completely using this function.

If the function is successful it returns `0`. Otherwise the environment is unchanged and the return value is `-1` and `errno` is set.

This function was originally part of the BSD library but is now part of the Unix standard.

Function: *int* **unsetenv** *(const char *name)*

Preliminary: | MT-Unsafe const:env | AS-Unsafe lock | AC-Unsafe lock | See POSIX Safety Concepts.

Using this function one can remove an entry completely from the environment. If the environment contains an entry with the key *name* this whole entry is removed. A call to this function is equivalent to a call to `putenv` when the *value* part of the string is empty.

The function returns `-1` if *name* is a null pointer, points to an empty string, or points to a string containing a `=` character. It returns `0` if the call succeeded.

This function was originally part of the BSD library but is now part of the Unix standard. The BSD version had no return value, though.

There is one more function to modify the whole environment. This function is said to be used in the POSIX.9 (POSIX bindings for Fortran 77) and so one should expect it did made it into POSIX.1. But this never happened. But we still provide this function as a GNU extension to enable writing standard compliant Fortran environments.

Function: *int* **clearenv** *(void)*

Preliminary: | MT-Unsafe const:env | AS-Unsafe heap lock | AC-Unsafe lock mem | See POSIX Safety Concepts.

The `clearenv` function removes all entries from the environment. Using `putenv` and `setenv` new entries can be added again later.

If the function is successful it returns `0`. Otherwise the return value is nonzero.

You can deal directly with the underlying representation of environment objects to add more variables to the environment (for example, to communicate with another program you are about to execute; see Executing a File).

Variable: *char* ** **environ**

The environment is represented as an array of strings. Each string is of the format '*`name`*`=`*`value`*'. The order in which strings appear in the environment is not significant, but the same *name* must not appear more than once. The last element of the array is a null pointer.

This variable is declared in the header file `unistd.h`.

If you just want to get the value of an environment variable, use `getenv`.

Unix systems, and GNU systems, pass the initial value of `environ` as the third argument to `main`. See Program Arguments.

Next: Standard Environment Variables, Up: Environment Variables [Contents][Index]