Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'getenv.3p' command

$ man getenv.3p

GETENV(3P) POSIX Programmer's Manual GETENV(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getenv ? get value of an environment variable

SYNOPSIS

#include <stdlib.h>

char *getenv(const char *name);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1?2017 defers to the ISO C standard.

The getenv() function shall search the environment of the calling process (see the Base Definitions volume of POSIX.1?2017, Chapter 8, Environment Variables) for the environment variable name if it exists and return a pointer to the value of the environment variable. If the specified environment variable cannot be found, a null pointer shall be returned. The application shall ensure that it does not modify the string pointed to by the getenv() function.

The returned string pointer might be invalidated or the string content might be overwritten by a subsequent call to getenv(), setenv(), un? setenv(), or (if supported) putenv() but they shall not be affected by a call to any other function in this volume of POSIX.1?2017.

The returned string pointer might also be invalidated if the calling thread is terminated.

The getenv() function need not be thread-safe.

RETURN VALUE

Upon successful completion, getenv() shall return a pointer to a string containing the value for the specified name. If the specified name cannot be found in the environment of the calling process, a null pointer shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Getting the Value of an Environment Variable

The following example gets the value of the HOME environment variable.

```
#include <stdlib.h>
...
const char *name = "HOME";
char *value;
value = getenv(name);
```

APPLICATION USAGE

None.

RATIONALE

The clearenv() function was considered but rejected. The putenv() func? tion has now been included for alignment with the Single UNIX Specifi? cation.

The getenv() function is inherently not thread-safe because it returns a value pointing to static data.

Conforming applications are required not to directly modify the point?

ers to which environ points, but to use only the setenv(), unsetenv(), and putenv() functions, or assignment to environ itself, to manipulate the process environment. This constraint allows the implementation to properly manage the memory it allocates. This enables the implementa? tion to free any space it has allocated to strings (and perhaps the pointers to them) stored in environ when unsetenv() is called. A C run? time start-up procedure (that which invokes main() and perhaps initial? izes environ) can also initialize a flag indicating that none of the environment has yet been copied to allocated storage, or that the sepa? rate table has not yet been initialized. If the application switches to a complete new environment by assigning a new value to environ, this can be detected by getenv(), setenv(), unsetenv(), or putenv() and the implementation can at that point reinitialize based on the new environ? ment. (This may include copying the environment strings into a new ar? ray and assigning environ to point to it.)

In fact, for higher performance of getenv(), implementations that do not provide putenv() could also maintain a separate copy of the envi? ronment in a data structure that could be searched much more quickly (such as an indexed hash table, or a binary tree), and update both it and the linear list at environ when setenv() or unsetenv() is invoked. On implementations that do provide putenv(), such a copy might still be worthwhile but would need to allow for the fact that applications can directly modify the content of environment strings added with putenv(). For example, if an environment string found by searching the copy is one that was added using putenv(), the implementation would need to check that the string in environ still has the same name (and value, if the copy includes values), and whenever searching the copy produces no match the implementation would then need to search each environment string in environ that was added using putenv() in case any of them have changed their names and now match. Thus, each use of putenv() to add to the environment would reduce the speed advantage of having the copy.

Performance of getenv() can be important for applications which have

large numbers of environment variables. Typically, applications like this use the environment as a resource database of user-configurable parameters. The fact that these variables are in the user's shell en? vironment usually means that any other program that uses environment variables (such as ls, which attempts to use COLUMNS), or really almost any utility (LANG, LC_ALL, and so on) is similarly slowed down by the linear search through the variables.

An implementation that maintains separate data structures, or even one that manages the memory it consumes, is not currently required as it was thought it would reduce consensus among implementors who do not want to change their historical implementations.

FUTURE DIRECTIONS

A future version may add one or more functions to access and modify the environment in a thread-safe manner.

SEE ALSO

exec, putenv(), setenv(), unsetenv()

The Base Definitions volume of POSIX.1?2017, Chapter 8, Environment Variables, <stdlib.h>

COPYRIGHT