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 'getcwd.3p' command

$ man getcwd.3p

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

getcwd ? get the pathname of the current working directory

SYNOPSIS

#include <unistd.h>

char *getcwd(char *buf, size_t size);

DESCRIPTION

The getcwd() function shall place an absolute pathname of the current working directory in the array pointed to by buf, and return buf. The pathname shall contain no components that are dot or dot-dot, or are symbolic links.

If there are multiple pathnames that getcwd() could place in the array pointed to by buf, one beginning with a single <slash> character and one or more beginning with two <slash> characters, then getcwd() shall place the pathname beginning with a single <slash> character in the ar? ray. The pathname shall not contain any unnecessary <slash> characters after the leading one or two <slash> characters.

The size argument is the size in bytes of the character array pointed

to by the buf argument. If buf is a null pointer, the behavior of getcwd() is unspecified.

RETURN VALUE

Upon successful completion, getcwd() shall return the buf argument. Otherwise, getcwd() shall return a null pointer and set errno to indi? cate the error. The contents of the array pointed to by buf are then undefined.

ERRORS

The getcwd() function shall fail if:

EINVAL The size argument is 0.

ERANGE The size argument is greater than 0, but is smaller than the length of the string +1.

The getcwd() function may fail if:

EACCES Search permission was denied for the current directory, or read or search permission was denied for a directory above the cur? rent directory in the file hierarchy.

ENOMEM Insufficient storage space is available.

The following sections are informative.

EXAMPLES

The following example uses {PATH_MAX} as the initial buffer size (un? less it is indeterminate or very large), and calls getcwd() with pro? gressively larger buffers until it does not give an [ERANGE] error.

```
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
...
long path_max;
size_t size;
char *buf;
char *ptr;
path_max = pathconf(".", _PC_PATH_MAX);
if (path_max == -1)
    size = 1024;
```

```
else if (path_max > 10240)
    size = 10240;
else
    size = path_max;
for (buf = ptr = NULL; ptr == NULL; size *= 2)
{
    if ((buf = realloc(buf, size)) == NULL)
    {
        ... handle error ...
    }
    ptr = getcwd(buf, size);
    if (ptr == NULL && errno != ERANGE)
    {
        ... handle error ...
    }
}
...
free (buf);
```

APPLICATION USAGE

If the pathname obtained from getcwd() is longer than {PATH_MAX} bytes, it could produce an [ENAMETOOLONG] error if passed to chdir(). There? fore, in order to return to that directory it may be necessary to break the pathname into sections shorter than {PATH_MAX} bytes and call chdir() on each section in turn (the first section being an absolute pathname and subsequent sections being relative pathnames). A simpler way to handle saving and restoring the working directory when it may be deeper than {PATH_MAX} bytes in the file hierarchy is to use a file de? scriptor and fchdir(), rather than getcwd() and chdir(). However, the two methods do have some differences. The fchdir() approach causes the program to restore a working directory even if it has been renamed in the meantime, whereas the chdir() approach restores to a directory with the same name as the original, even if the directories were renamed in the meantime. Since the fchdir() approach does not access parent direc?

tories, it can succeed when getcwd() would fail due to permissions problems. In applications conforming to earlier versions of this stan? dard, it was not possible to use the fchdir() approach when the working directory is searchable but not readable, as the only way to open a di? rectory was with O_RDONLY, whereas the getcwd() approach can succeed in this case.

RATIONALE

Having getcwd() take no arguments and instead use the malloc() function to produce space for the returned argument was considered. The advan? tage is that getcwd() knows how big the working directory pathname is and can allocate an appropriate amount of space. But the programmer would have to use the free() function to free the resulting object, or each use of getcwd() would further reduce the available memory. Fi? nally, getcwd() is taken from the SVID where it has the two arguments used in this volume of POSIX.1?2017.

The older function getwd() was rejected for use in this context because it had only a buffer argument and no size argument, and thus had no way to prevent overwriting the buffer, except to depend on the programmer to provide a large enough buffer.

On some implementations, if buf is a null pointer, getcwd() may obtain size bytes of memory using malloc(). In this case, the pointer re? turned by getcwd() may be used as the argument in a subsequent call to free(). Invoking getcwd() with buf as a null pointer is not recom? mended in conforming applications.

Earlier implementations of getcwd() sometimes generated pathnames like "../../../subdirname" internally, using them to explore the path of an? cestor directories back to the root. If one of these internal pathnames exceeded {PATH_MAX} in length, the implementation could fail with errno set to [ENAMETOOLONG]. This is no longer allowed.

If a program is operating in a directory where some (grand)parent di? rectory does not permit reading, getcwd() may fail, as in most imple? mentations it must read the directory to determine the name of the file. This can occur if search, but not read, permission is granted in

an intermediate directory, or if the program is placed in that direc?
tory by some more privileged process (for example, login). Including
the [EACCES] error condition makes the reporting of the error consis?
tent and warns the application developer that getcwd() can fail for
reasons beyond the control of the application developer or user. Some
implementations can avoid this occurrence (for example, by implementing
getcwd() using pwd, where pwd is a set-user-root process), thus the er?
ror was made optional. Since this volume of POSIX.1?2017 permits the
addition of other errors, this would be a common addition and yet one
that applications could not be expected to deal with without this addi?
tion.

FUTURE DIRECTIONS

None.

SEE ALSO

malloc()

The Base Definitions volume of POSIX.1?2017, <unistd.h>

COPYRIGHT