NAME   top

       chroot - change root directory

SYNOPSIS   top

       #include <unistd.h>

       int chroot(const char *path);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       chroot():
           Since glibc 2.2.2:
               _XOPEN_SOURCE && ! (_POSIX_C_SOURCE >= 200112L)
                   || /* Since glibc 2.20: */ _DEFAULT_SOURCE
                   || /* Glibc versions <= 2.19: */ _BSD_SOURCE
           Before glibc 2.2.2: none

DESCRIPTION   top

       chroot() changes the root directory of the calling process to that
       specified in path. This directory will be used for pathnames
       beginning with /. The root directory is inherited by all children of
       the calling process.

       Only a privileged process (Linux: one with the CAP_SYS_CHROOT
       capability in its user namespace) may call chroot().

       This call changes an ingredient in the pathname resolution process
       and does nothing else. In particular, it is not intended to be used
       for any kind of security purpose, neither to fully sandbox a process
       nor to restrict filesystem system calls. In the past, chroot() has
       been used by daemons to restrict themselves prior to passing paths
       supplied by untrusted users to system calls such as open(2).
       However, if a folder is moved out of the chroot directory, an
       attacker can exploit that to get out of the chroot directory as well.
       The easiest way to do that is to chdir(2) to the to-be-moved
       directory, wait for it to be moved out, then open a path like
       ../../../etc/passwd.

       A slightly trickier variation also works under some circumstances if
       chdir(2) is not permitted. If a daemon allows a "chroot directory"
       to be specified, that usually means that if you want to prevent
       remote users from accessing files outside the chroot directory, you
       must ensure that folders are never moved out of it.

       This call does not change the current working directory, so that
       after the call '.' can be outside the tree rooted at '/'. In
       particular, the superuser can escape from a "chroot jail" by doing:

           mkdir foo; chroot foo; cd ..

       This call does not close open file descriptors, and such file
       descriptors may allow access to files outside the chroot tree.

RETURN VALUE   top

       On success, zero is returned. On error, -1 is returned, and errno is
       set appropriately.

ERRORS   top

       Depending on the filesystem, other errors can be returned. The more
       general errors are listed below:

       EACCES Search permission is denied on a component of the path prefix.
              (See also path_resolution(7).)

       EFAULT path points outside your accessible address space.

       EIO   An I/O error occurred.

       ELOOP Too many symbolic links were encountered in resolving path.

       ENAMETOOLONG
              path is too long.

       ENOENT The file does not exist.

       ENOMEM Insufficient kernel memory was available.

       ENOTDIR
              A component of path is not a directory.

       EPERM The caller has insufficient privilege.

CONFORMING TO   top

       SVr4, 4.4BSD, SUSv2 (marked LEGACY). This function is not part of
       POSIX.1-2001.

NOTES   top

       A child process created via fork(2) inherits its parent's root
       directory. The root directory is left unchanged by execve(2).

       The magic symbolic link, /proc/[pid]/root, can be used to discover a
       process's root directory; see proc(5) for details.

       FreeBSD has a stronger jail() system call.

SEE ALSO   top

       chroot(1), chdir(2), pivot_root(2), path_resolution(7),
       switch_root(8)

COLOPHON   top

       This page is part of release 5.02 of the Linux man-pages project. A
       description of the project, information about reporting bugs, and the
       latest version of this page, can be found at
       https://www.kernel.org/doc/man-pages/.