NAME

pivot_root − change the root mount

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

#include <sys/syscall.h> /* Definition of **SYS_*** constants */
#include <unistd.h>

int syscall(SYS_pivot_root, const char ****new_root*, const char ****put_old***);**

Note: glibc provides no wrapper for **pivot_root**(), necessitating the use of **syscall**(2).

DESCRIPTION

pivot_root() changes the root mount in the mount namespace of the calling process. More precisely, it moves the root mount to the directory *put_old* and makes *new_root* the new root mount. The calling process must have the **CAP_SYS_ADMIN** capability in the user namespace that owns the caller's mount namespace.

pivot_root() changes the root directory and the current working directory of each process or thread in the same mount namespace to *new_root* if they point to the old root directory. (See also NOTES.) On the other hand, **pivot_root**() does not change the caller's current working directory (unless it is on the old root directory), and thus it should be followed by a **chdir("/")** call.

The following restrictions apply:

- *new_root* and *put_old* must be directories.
- *new_root* and *put_old* must not be on the same mount as the current root.
- *put_old* must be at or underneath *new_root*; that is, adding some nonnegative number of "/.." suffixes to the pathname pointed to by *put_old* must yield the same directory as *new_root*.
- *new_root* must be a path to a mount point, but can't be "/". A path that is not already a mount point can be converted into one by bind mounting the path onto itself.
- The propagation type of the parent mount of *new_root* and the parent mount of the current root directory must not be **MS_SHARED**; similarly, if *put_old* is an existing mount point, its propagation type must not be **MS_SHARED**. These restrictions ensure that**pi vot_root**() never propagates any changes to another mount namespace.
- The current root directory must be a mount point.

RETURN VALUE

On success, zero is returned. On error, −1 is returned, and *errno* is set to indicate the error.

ERRORS

pivot_root() may fail with any of the same errors as **stat**(2). Additionally, it may fail with the following errors:

EBUSY
new_root or *put_old* is on the current root mount. (This error covers the pathological case where *new_root* is "/".)

EINVAL
new_root is not a mount point.

EINVAL
put_old is not at or underneath *new_root*.

EINVAL
The current root directory is not a mount point (because of an earlier **chroot**(2)).

EINVAL
The current root is on the rootfs (initial ramfs) mount; see NOTES.

EINVAL
Either the mount point at *new_root*, or the parent mount of that mount point, has propagation type **MS_SHARED**.

EINVAL
put_old is a mount point and has the propagation type **MS_SHARED**.

ENOTDIR
new_root or *put_old* is not a directory.

EPERM
The calling process does not have the **CAP_SYS_ADMIN** capability.

VERSIONS
pivot_root() was introduced in Linux 2.3.41.

STANDARDS
pivot_root() is Linux-specific and hence is not portable.

NOTES
A command-line interface for this system call is provided by **pivot_root**(8).

pivot_root() allows the caller to switch to a new root filesystem while at the same time placing the old root mount at a location under *new_root* from where it can subsequently be unmounted. (The fact that it moves all processes that have a root directory or current working directory on the old root directory to the new root frees the old root directory of users, allowing the old root mount to be unmounted more easily.)

One use of **pivot_root**() is during system startup, when the system mounts a temporary root filesystem (e.g., an **initrd**(4)), then mounts the real root filesystem, and eventually turns the latter into the root directory of all relevant processes and threads. A modern use is to set up a root filesystem during the creation of a container.

The fact that **pivot_root**() modifies process root and current working directories in the manner noted in DESCRIPTION is necessary in order to prevent kernel threads from keeping the old root mount busy with their root and current working directories, even if they never access the filesystem in any way.

The rootfs (initial ramfs) cannot be **pivot_root**()ed. The recommended method of changing the root filesystem in this case is to delete everything in rootfs, overmount rootfs with the new root, attach *stdin*/*stdout*/*stderr* to the new */dev/console*, and exec the new **init**(1). Helper programs for this process exist; see **switch_root**(8).

pivot_root(".", ".")
new_root and *put_old* may be the same directory. In particular, the following sequence allows a pivot-root operation without needing to create and remove a temporary directory:

```
chdir(new_root);
pivot_root(".", ".");
umount2(".", MNT_DETACH);
```

This sequence succeeds because the **pivot_root**() call stacks the old root mount point on top of the new root mount point at /. At that point, the calling process's root directory and current working directory refer to the new root mount point (*new_root*). During the subsequent**umount**() call, resolution of"." starts with *new_root* and then moves up the list of mounts stacked at /, with the result that old root mount point is unmounted.

Historical notes
For many years, this manual page carried the following text:

> **pivot_root**() may or may not change the current root and the current working directory of any processes or threads which use the old root directory. The caller of**pi vot_root**() must ensure that processes with root or current working directory at the old root operate correctly in either case. An

easy way to ensure this is to change their root and current working directory to *new_root* before invoking **pivot_root**().

This text, written before the system call implementation was even finalized in the kernel, was probably intended to warn users at that time that the implementation might change before final release. However, the behavior stated in DESCRIPTION has remained consistent since this system call was first implemented and will not change now.

EXAMPLES

The program below demonstrates the use of **pivot_root**() inside a mount namespace that is created using **clone**(2). After pivoting to the root directory named in the program's first command-line argument, the child created by **clone**(2) then executes the program named in the remaining command-line arguments.

We demonstrate the program by creating a directory that will serve as the new root filesystem and placing a copy of the (statically linked) **busybox**(1) executable in that directory.

```
$ mkdir /tmp/rootfs
$ ls -id /tmp/rootfs    # Show inode number of new root directory
319459 /tmp/rootfs
$ cp $(which busybox) /tmp/rootfs
$ PS1='bbsh$ ' sudo ./pivot_root_demo /tmp/rootfs /busybox sh
bbsh$ PATH=/
bbsh$ busybox ln busybox ln
bbsh$ ln busybox echo
bbsh$ ln busybox ls
bbsh$ ls
busybox  echo     ln       ls
bbsh$ ls -id /          # Compare with inode number above
319459 /
bbsh$ echo 'hello world'
hello world
```

Program source

```
/* pivot_root_demo.c */

#define _GNU_SOURCE
#include <err.h>
#include <limits.h>
#include <sched.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/mount.h>
#include <sys/stat.h>
#include <sys/syscall.h>
#include <sys/wait.h>
#include <unistd.h>

static int
pivot_root(const char *new_root, const char *put_old)
{
    return syscall(SYS_pivot_root, new_root, put_old);
}

#define STACK_SIZE (1024 * 1024)
```

```
static int              /* Startup function for cloned child */
child(void *arg)
{
    char        path[PATH_MAX];
    char        **args = arg;
    char        *new_root = args[0];
    const char  *put_old = "/oldrootfs";

    /* Ensure that 'new_root' and its parent mount don't have
       shared propagation (which would cause pivot_root() to
       return an error), and prevent propagation of mount
       events to the initial mount namespace. */

    if (mount(NULL, "/", NULL, MS_REC | MS_PRIVATE, NULL) == -1)
        err(EXIT_FAILURE, "mount-MS_PRIVATE");

    /* Ensure that 'new_root' is a mount point. */

    if (mount(new_root, new_root, NULL, MS_BIND, NULL) == -1)
        err(EXIT_FAILURE, "mount-MS_BIND");

    /* Create directory to which old root will be pivoted. */

    snprintf(path, sizeof(path), "%s/%s", new_root, put_old);
    if (mkdir(path, 0777) == -1)
        err(EXIT_FAILURE, "mkdir");

    /* And pivot the root filesystem. */

    if (pivot_root(new_root, path) == -1)
        err(EXIT_FAILURE, "pivot_root");

    /* Switch the current working directory to "/". */

    if (chdir("/") == -1)
        err(EXIT_FAILURE, "chdir");

    /* Unmount old root and remove mount point. */

    if (umount2(put_old, MNT_DETACH) == -1)
        perror("umount2");
    if (rmdir(put_old) == -1)
        perror("rmdir");

    /* Execute the command specified in argv[1]... */

    execv(args[1], &args[1]);
    err(EXIT_FAILURE, "execv");
}

int
main(int argc, char *argv[])
{
    char *stack;
```

```
        /* Create a child process in a new mount namespace. */

        stack = mmap(NULL, STACK_SIZE, PROT_READ | PROT_WRITE,
                     MAP_PRIVATE | MAP_ANONYMOUS | MAP_STACK, -1, 0);
        if (stack == MAP_FAILED)
            err(EXIT_FAILURE, "mmap");

        if (clone(child, stack + STACK_SIZE,
                  CLONE_NEWNS | SIGCHLD, &argv[1]) == -1)
            err(EXIT_FAILURE, "clone");

        /* Parent falls through to here; wait for child. */

        if (wait(NULL) == -1)
            err(EXIT_FAILURE, "wait");

        exit(EXIT_SUCCESS);
    }
```

SEE ALSO

chdir(2), **chroot**(2), **mount**(2), **stat**(2), **initrd**(4), **mount_namespaces**(7), **pivot_root**(8), **switch_root**(8)