NAME

close_range − close all file descriptors in a given range

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

```
#define _GNU_SOURCE         /* See feature_test_macros(7) */
#include <unistd.h>

#include <linux/close_range.h> /* Definition of CLOSE_RANGE_* constants */

int close_range(unsigned int first, unsigned int last, int flags);
```

DESCRIPTION

The **close_range**() system call closes all open file descriptors from *first* to *last* (included).

Errors closing a given file descriptor are currently ignored.

flags is a bit mask containing 0 or more of the following:

CLOSE_RANGE_CLOEXEC (since Linux 5.11)
: Set the close-on-exec flag on the specified file descriptors, rather than immediately closing them.

CLOSE_RANGE_UNSHARE
: Unshare the specified file descriptors from any other processes before closing them, avoiding races with other threads sharing the file descriptor table.

RETURN VALUE

On success, **close_range**() returns 0. On error, −1 is returned and *errno* is set to indicate the error.

ERRORS

EINVAL
: *flags* is not valid, or *first* is greater than *last*.

The following can occur with **CLOSE_RANGE_UNSHARE** (when constructing the new descriptor table):

EMFILE
: The number of open file descriptors exceeds the limit specified in */proc/sys/fs/nr_open* (see **proc**(5)). This error can occur in situations where that limit was lowered before a call to **close_range**() where the **CLOSE_RANGE_UNSHARE** flag is specified.

ENOMEM
: Insufficient kernel memory was available.

STANDARDS

None.

HISTORY

FreeBSD. Linux 5.9, glibc 2.34.

NOTES

Closing all open file descriptors

To avoid blindly closing file descriptors in the range of possible file descriptors, this is sometimes implemented (on Linux) by listing open file descriptors in */proc/self/fd/* and calling **close**(2) on each one. **close_range**() can take care of this without requiring */proc* and within a single system call, which provides significant performance benefits.

Closing file descriptors before exec

File descriptors can be closed safely using

```
/* we don't want anything past stderr here */
close_range(3, ~0U, CLOSE_RANGE_UNSHARE);
execve(....);
```

CLOSE_RANGE_UNSHARE is conceptually equivalent to

```
unshare(CLONE_FILES);
close_range(first, last, 0);
```

but can be more efficient: if the unshared range extends past the current maximum number of file descriptors allocated in the caller's file descriptor table (the common case when *last* is ~0U), the kernel will unshare a new file descriptor table for the caller up to *first*, copying as few file descriptors as possible. This avoids subsequent **close**(2) calls entirely; the whole operation is complete once the table is unshared.

Closing files on *exec*

This is particularly useful in cases where multiple pre-**exec** setup steps risk conflicting with each other. For example, setting up a **seccomp**(2) profile can conflict with a **close_range**() call: if the file descriptors are closed before the **seccomp**(2) profile is set up, the profile setup can't use them itself, or control their closure; if the file descriptors are closed afterwards, the seccomp profile can't block the **close_range**() call or any fallbacks. Using**CLOSE_RANGE_CLOEXEC** a voids this: the descriptors can be marked before the **seccomp**(2) profile is set up, and the profile can control access to **close_range**() without affecting the calling process.

EXAMPLES

The program shown below opens the files named in its command-line arguments, displays the list of files that it has opened (by iterating through the entries in */proc/PID/fd*), uses **close_range**() to close all file descriptors greater than or equal to 3, and then once more displays the process's list of open files. The following example demonstrates the use of the program:

```
$ touch /tmp/a /tmp/b /tmp/c;
$ ./a.out /tmp/a /tmp/b /tmp/c;
/tmp/a opened as FD 3
/tmp/b opened as FD 4
/tmp/c opened as FD 5
/proc/self/fd/0 ==> /dev/pts/1
/proc/self/fd/1 ==> /dev/pts/1
/proc/self/fd/2 ==> /dev/pts/1
/proc/self/fd/3 ==> /tmp/a
/proc/self/fd/4 ==> /tmp/b
/proc/self/fd/5 ==> /tmp/c
/proc/self/fd/6 ==> /proc/9005/fd
========= About to call close_range() =======
/proc/self/fd/0 ==> /dev/pts/1
/proc/self/fd/1 ==> /dev/pts/1
/proc/self/fd/2 ==> /dev/pts/1
/proc/self/fd/3 ==> /proc/9005/fd
```

Note that the lines showing the pathname */proc/9005/fd* result from the calls to **opendir**(3).

Program source

```
#define _GNU_SOURCE
#include <dirent.h>
#include <fcntl.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

/* Show the contents of the symbolic links in /proc/self/fd */

static void
```

```
show_fds(void)
{
    DIR            *dirp;
    char           path[PATH_MAX], target[PATH_MAX];
    ssize_t        len;
    struct dirent  *dp;

    dirp = opendir("/proc/self/fd");
    if (dirp  == NULL) {
        perror("opendir");
        exit(EXIT_FAILURE);
    }

    for (;;) {
        dp = readdir(dirp);
        if (dp == NULL)
            break;

        if (dp->d_type == DT_LNK) {
            snprintf(path, sizeof(path), "/proc/self/fd/%s",
                     dp->d_name);

            len = readlink(path, target, sizeof(target));
            printf("%s ==> %.*s\n", path, (int) len, target);
        }
    }

    closedir(dirp);
}

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

    for (size_t j = 1; j < argc; j++) {
        fd = open(argv[j], O_RDONLY);
        if (fd == -1) {
            perror(argv[j]);
            exit(EXIT_FAILURE);
        }
        printf("%s opened as FD %d\n", argv[j], fd);
    }

    show_fds();

    printf("========= About to call close_range() =======\n");

    if (close_range(3, ~0U, 0) == -1) {
        perror("close_range");
        exit(EXIT_FAILURE);
    }

    show_fds();
```

```
            exit(EXIT_FAILURE);
        }
```

SEE ALSO

close(2)