

**NAME**

getgroups, setgroups – get/set list of supplementary group IDs

**SYNOPSIS**

```
#include <sys/types.h>
```

```
#include <unistd.h>
```

```
int getgroups(int size, gid_t list[]);
```

```
#include <grp.h>
```

```
int setgroups(size_t size, const gid_t *list);
```

Feature Test Macro Requirements for glibc (see **feature\_test\_macros(7)**):

```
setgroups():
```

Since glibc 2.19:

```
_DEFAULT_SOURCE
```

Glibc 2.19 and earlier:

```
_BSD_SOURCE
```

**DESCRIPTION**

**getgroups()** returns the supplementary group IDs of the calling process in *list*. The argument *size* should be set to the maximum number of items that can be stored in the buffer pointed to by *list*. If the calling process is a member of more than *size* supplementary groups, then an error results.

It is unspecified whether the effective group ID of the calling process is included in the returned list. (Thus, an application should also call **getegid(2)** and add or remove the resulting value.)

If *size* is zero, *list* is not modified, but the total number of supplementary group IDs for the process is returned. This allows the caller to determine the size of a dynamically allocated *list* to be used in a further call to **getgroups()**.

**setgroups()** sets the supplementary group IDs for the calling process. Appropriate privileges are required (see the description of the **EPERM** error, below). The *size* argument specifies the number of supplementary group IDs in the buffer pointed to by *list*. A process can drop all of its supplementary groups with the call:

```
setgroups(0, NULL);
```

**RETURN VALUE**

On success, **getgroups()** returns the number of supplementary group IDs. On error,  $-1$  is returned, and *errno* is set appropriately.

On success, **setgroups()** returns 0. On error,  $-1$  is returned, and *errno* is set appropriately.

**ERRORS****EFAULT**

*list* has an invalid address.

**getgroups()** can additionally fail with the following error:

**EINVAL**

*size* is less than the number of supplementary group IDs, but is not zero.

**setgroups()** can additionally fail with the following errors:

**EINVAL**

*size* is greater than **NGROUPS\_MAX** (32 before Linux 2.6.4; 65536 since Linux 2.6.4).

**ENOMEM**

Out of memory.

**EPERM**

The calling process has insufficient privilege (the caller does not have the **CAP\_SETGID** capability in the user namespace in which it resides).

**EPERM** (since Linux 3.19)

The use of **setgroups()** is denied in this user namespace. See the description of */proc/[pid]/setgroups* in **user\_namespaces(7)**.

## CONFORMING TO

**getgroups()**: SVr4, 4.3BSD, POSIX.1-2001, POSIX.1-2008.

**setgroups()**: SVr4, 4.3BSD. Since **setgroups()** requires privilege, it is not covered by POSIX.1.

## NOTES

A process can have up to **NGROUPS\_MAX** supplementary group IDs in addition to the effective group ID. The constant **NGROUPS\_MAX** is defined in *<limits.h>*. The set of supplementary group IDs is inherited from the parent process, and preserved across an **execve(2)**.

The maximum number of supplementary group IDs can be found at run time using **sysconf(3)**:

```
long ngroups_max;
ngroups_max = sysconf(_SC_NGROUPS_MAX);
```

The maximum return value of **getgroups()** cannot be larger than one more than this value. Since Linux 2.6.4, the maximum number of supplementary group IDs is also exposed via the Linux-specific read-only file, */proc/sys/kernel/ngroups\_max*.

The original Linux **getgroups()** system call supported only 16-bit group IDs. Subsequently, Linux 2.4 added **getgroups32()**, supporting 32-bit IDs. The glibc **getgroups()** wrapper function transparently deals with the variation across kernel versions.

## C library/kernel differences

At the kernel level, user IDs and group IDs are a per-thread attribute. However, POSIX requires that all threads in a process share the same credentials. The NPTL threading implementation handles the POSIX requirements by providing wrapper functions for the various system calls that change process UIDs and GIDs. These wrapper functions (including the one for **setgroups()**) employ a signal-based technique to ensure that when one thread changes credentials, all of the other threads in the process also change their credentials. For details, see **nptl(7)**.

## SEE ALSO

**getgid(2)**, **setgid(2)**, **getgrouplist(3)**, **group\_member(3)**, **initgroups(3)**, **capabilities(7)**, **credentials(7)**

## COLOPHON

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/>.