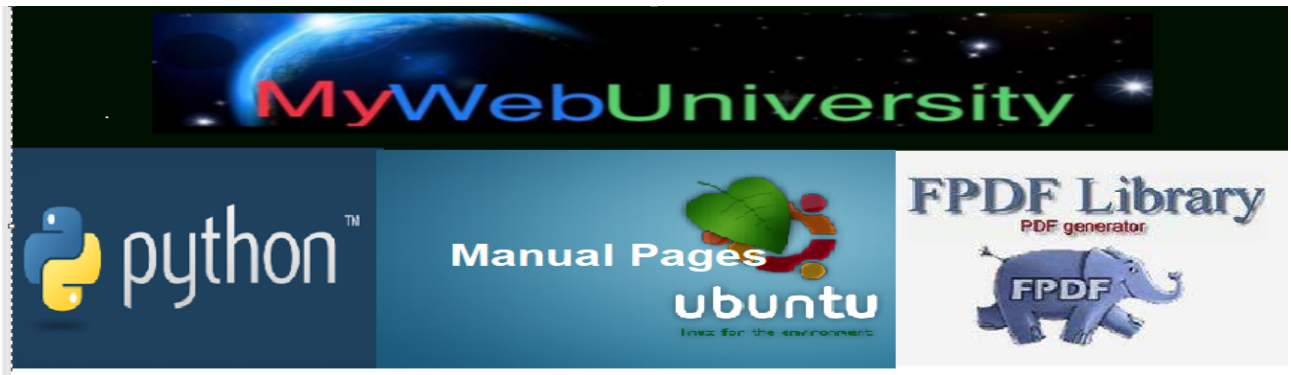

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'setgroups32.2'

$ man setgroups32.2

getgroups(2) System Calls Manual getgroups(2)

NAME

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

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <unistd.h>
int getgroups(int size, gid_t list[_Nullable size]);
#include <grp.h>
int setgroups(size_t size, const gid_t list[size]);
```

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 argu? ment 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 re?

sulting 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 al? located list to be used in a further call to getgroups().

setgroups() sets the supplementary group IDs for the calling process. Appropriate privi? leges are required (see the description of the EPERM error, below). The size argument spec? ifies 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, (gid_t [0]){});
```

RETURN VALUE

On success, getgroups() returns the number of supplementary group IDs. On error, -1 is re? turned, and errno is set to indicate the error.

On success, setgroups() returns 0. On error, -1 is returned, and errno is set to indicate the error.

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_SET? GID 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).

VERSIONS

C library/kernel differences

At the kernel level, user IDs and group IDs are a per-thread attribute. However, POSIX re? quires that all threads in a process share the same credentials. The NPTL threading imple? mentation handles the POSIX requirements by providing wrapper functions for the various sys? tem 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 creden?

tials, all of the other threads in the process also change their credentials. For details, see nptl(7).

STANDARDS

getgroups()

POSIX.1-2024.

setgroups()

None.

HISTORY

getgroups()

4.3BSD, SVr4, POSIX.1-1988.

setgroups()

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

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.

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.

EXAMPLES

```
#include <err.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
```

```
#define MALLOC(n, T)  ((T *) reallocarray(NULL, n, sizeof(T)))
static gid_t *agetgroups(size_t *ngids);
int
main(void)
{
    gid_t   *gids;
    size_t  n;
    gids = agetgroups(&n);
    if (gids == NULL)
        err(EXIT_FAILURE, "agetgroups");
    if (n != 0) {
        printf("%jd", (intmax_t) gids[0]);
        for (size_t i = 1; i < n; i++)
            printf(" %jd", (intmax_t) gids[i]);
    }
    puts("");
    free(gids);
    exit(EXIT_SUCCESS);
}
static gid_t *
agetgroups(size_t *ngids)
{
    int    n;
    gid_t  *gids;
    n = getgroups(0, NULL);
    if (n == -1)
        return NULL;
    gids = MALLOC(n, gid_t);
    if (gids == NULL)
        return NULL;
    n = getgroups(n, gids);
    if (n == -1) {
        free(gids);
```

```
            return NULL;
        }
        *ngids = n;
        return gids;
    }
```

SEE ALSO

getgid(2), setgid(2), getgrouplist(3), group_member(3), initgroups(3), capabilities(7), cre?
dentials(7)