NAME

rfork - manipulate process resources

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <unistd.h>

pid_t
rfork(*int flags*);

DESCRIPTION

Forking, vforking or rforking are the only ways new processes are created. The *flags* argument to **rfork**() selects which resources of the invoking process (parent) are shared by the new process (child) or initialized to their default values. The resources include the open file descriptor table (which, when shared, permits processes to open and close files for other processes), and open files. The *flags* argument is either RFSPAWN or the logical OR of some subset of:

RFPROC	If set a new process is created; otherwise changes affect the current process.
RFNOWAIT	If set, the child process will be dissociated from the parent. Upon exit the child will not leave a status for the parent to collect. See wait(2).
RFFDG	If set, the invoker’s file descriptor table (see intro(2)) is copied; otherwise the two processes share a single table.
RFCFDG	If set, the new process starts with a clean file descriptor table. Is mutually exclusive with RFFDG.
RFTHREAD	If set, the new process shares file descriptor to process leaders table with its parent. Only applies when neither RFFDG nor RFCFDG are set.
RFMEM	If set, the kernel will force sharing of the entire address space, typically by sharing the hardware page table directly. The child will thus inherit and share all the segments the parent process owns, whether they are normally shareable or not. The stack segment is not split (both the parent and child return on the same stack) and thus **rfork**() with the RFMEM flag may not generally be called directly from high level languages including C. May be set only with RFPROC. A helper function is provided to assist with this problem and will cause the new process to run on the

provided stack. See rfork_thread(3) for information. Note that a lot of code will not run correctly in such an environment.

RFSIGSHARE	If set, the kernel will force sharing the sigacts structure between the child and the parent.
RFTSIGZMB	If set, the kernel will deliver a specified signal to the parent upon the child exit, instead of default SIGCHLD. The signal number signum is specified by oring the RFTSIGFLAGS(signum) expression into *flags*. Specifying signal number 0 disables signal delivery upon the child exit.
RFLINUXTHPN	If set, the kernel will deliver SIGUSR1 instead of SIGCHLD upon thread exit for the child. This is intended to mimic certain Linux clone behaviour.

File descriptors in a shared file descriptor table are kept open until either they are explicitly closed or all processes sharing the table exit.

If RFSPAWN is passed, **rfork** will use vfork(2) semantics but reset all signal actions in the child to default. This flag is used by the posix_spawn(3) implementation in libc.

If RFPROC is set, the value returned in the parent process is the process id of the child process; the value returned in the child is zero. Without RFPROC, the return value is zero. Process id's range from 1 to the maximum integer (*int*) value. The **rfork**() system call will sleep, if necessary, until required process resources are available.

The **fork**() system call can be implemented as a call to **rfork**(*RFFDG | RFPROC*) but is not for backwards compatibility.

RETURN VALUES

Upon successful completion, **rfork**() returns a value of 0 to the child process and returns the process ID of the child process to the parent process. Otherwise, a value of -1 is returned to the parent process, no child process is created, and the global variable *errno* is set to indicate the error.

ERRORS

The **rfork**() system call will fail and no child process will be created if:

[EAGAIN]	The system-imposed limit on the total number of processes under execution would be exceeded. The limit is given by the sysctl(3) MIB variable KERN_MAXPROC. (The limit is actually ten less than this except for the super user).

[EAGAIN]	The user is not the super user, and the system-imposed limit on the total number of processes under execution by a single user would be exceeded. The limit is given by the sysctl(3) MIB variable KERN_MAXPROCPERUID.
[EAGAIN]	The user is not the super user, and the soft resource limit corresponding to the *resource* argument RLIMIT_NOFILE would be exceeded (see getrlimit(2)).
[EINVAL]	Both the RFFDG and the RFCFDG flags were specified.
[EINVAL]	Any flags not listed above were specified.
[EINVAL]	An invalid signal number was specified.
[ENOMEM]	There is insufficient swap space for the new process.

SEE ALSO

fork(2), intro(2), minherit(2), vfork(2), pthread_create(3), rfork_thread(3)

HISTORY

The **rfork**() function first appeared in Plan9.