NAME

fork − create a child process

SYNOPSIS

#include <unistd.h>

pid_t fork(void);

DESCRIPTION

fork() creates a new process by duplicating the calling process. The new process, referred to as the *child*, is an exact duplicate of the calling process, referred to as the *parent*, except for the following points:

* The child has its own unique process ID, and this PID does not match the ID of any existing process group (**setpgid**(2)).

* The child's parent process ID is the same as the parent's process ID.

* The child does not inherit its parent's memory locks (**mlock**(2), **mlockall**(2)).

* Process resource utilizations (**getrusage**(2)) and CPU time counters (**times**(2)) are reset to zero in the child.

* The child's set of pending signals is initially empty (**sigpending**(2)).

* The child does not inherit semaphore adjustments from its parent (**semop**(2)).

* The child does not inherit record locks from its parent (**fcntl**(2)).

* The child does not inherit timers from its parent (**setitimer**(2), **alarm**(2), **timer_create**(2)).

* The child does not inherit outstanding asynchronous I/O operations from its parent (**aio_read**(3), **aio_write**(3)), nor does it inherit any asynchronous I/O contexts from its parent (**see**io_setup**(2)).**

The process attributes in the preceding list are all specified in POSIX.1-2001. The parent and child also differ with respect to the following Linux-specific process attributes:

* The child does not inherit directory change notifications (dnotify) from its parent (see the description of **F_NOTIFY** in **fcntl**(2)).

* The **prctl**(2) **PR_SET_PDEATHSIG** setting is reset so that the child does not receive a signal when its parent terminates.

* Memory mappings that have been marked with the **madvise**(2) **MADV_DONTFORK** flag are not inherited across a **fork**().

* The termination signal of the child is always **SIGCHLD** (see **clone**(2)).

Note the following further points:

* The child process is created with a single thread — the one that called **fork**(). The entire virtual address space of the parent is replicated in the child, including the states of mutexes, condition variables, and other pthreads objects; the use of **pthread_atfork**(3) may be helpful for dealing with problems that this can cause.

* The child inherits copies of the parent's set of open file descriptors. Each file descriptor in the child refers to the same open file description (see **open**(2)) as the corresponding file descriptor in the parent. This means that the two descriptors share open file status flags, current file offset, and signal-driven I/O attributes (see the description of **F_SETOWN** and **F_SETSIG** in **fcntl**(2)).

* The child inherits copies of the parent's set of open message queue descriptors (see **mq_overview**(7)). Each descriptor in the child refers to the same open message queue description as the corresponding descriptor in the parent. This means that the two descriptors share the same flags (*mq_flags*).

* The child inherits copies of the parent's set of open directory streams (see **opendir**(3)). POSIX.1-2001 says that the corresponding directory streams in the parent and child *may* share the directory stream positioning; on Linux/glibc they do not.

RETURN VALUE

On success, the PID of the child process is returned in the parent, and 0 is returned in the child. On failure, −1 is returned in the parent, no child process is created, and *errno* is set appropriately.

ERRORS

EAGAIN
fork() cannot allocate sufficient memory to copy the parent's page tables and allocate a task structure for the child.

EAGAIN
It was not possible to create a new process because the caller's **RLIMIT_NPROC** resource limit was encountered. To exceed this limit, the process must have either the **CAP_SYS_ADMIN** or the **CAP_SYS_RESOURCE** capability.

ENOMEM
fork() failed to allocate the necessary kernel structures because memory is tight.

CONFORMING TO

SVr4, 4.3BSD, POSIX.1-2001.

NOTES

Under Linux, **fork**() is implemented using copy-on-write pages, so the only penalty that it incurs is the time and memory required to duplicate the parent's page tables, and to create a unique task structure for the child.

Since version 2.3.3, rather than invoking the kernel's **fork**() system call, the glibc **fork**() wrapper that is provided as part of the NPTL threading implementation invokes **clone**(2) with flags that provide the same effect as the traditional system call. The glibc wrapper invokes any fork handlers that have been established using **pthread_atfork**(3).

EXAMPLE

See **pipe**(2) and **wait**(2).

SEE ALSO

clone(2), **execve**(2), **setrlimit**(2), **unshare**(2), **vfork**(2), **wait**(2), **daemon**(3), **capabilities**(7), **credentials**(7)

COLOPHON

This page is part of release 3.22 of the Linux *man-pages* project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/.