NAME

vfork − create a child process and block parent

SYNOPSIS

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

pid_t vfork(void);
```

Feature Test Macro Requirements for glibc (see **feature_test_macros**(7)):

vfork(): _BSD_SOURCE || _XOPEN_SOURCE >= 500

DESCRIPTION

Standard Description

(From POSIX.1) The **vfork**() function has the same effect as **fork**(2), except that the behavior is undefined if the process created by **vfork**() either modifies any data other than a variable of type *pid_t* used to store the return value from **vfork**(), or returns from the function in which **vfork**() was called, or calls any other function before successfully calling **_exit**(2) or one of the **exec**(3) family of functions.

Linux Description

vfork(), just like **fork**(2), creates a child process of the calling process. For details and return value and errors, see **fork**(2).

vfork() is a special case of **clone**(2). It is used to create new processes without copying the page tables of the parent process. It may be useful in performance-sensitive applications where a child will be created which then immediately issues an **execve**(2).

vfork() differs from **fork**(2) in that the parent is suspended until the child terminates (either normally, by calling **_exit**(2), or abnormally, after delivery of a fatal signal), or it makes a call to **execve**(2). Until that point, the child shares all memory with its parent, including the stack. The child must not return from the current function or call **exit**(3), but may call **_exit**(2).

Signal handlers are inherited, but not shared. Signals to the parent arrive after the child releases the parent's memory (i.e., after the child terminates or calls **execve**(2)).

Historic Description

Under Linux, **fork**(2) is implemented using copy-on-write pages, so the only penalty incurred by **fork**(2) is the time and memory required to duplicate the parent's page tables, and to create a unique task structure for the child. However, in the bad old days a **fork**(2) would require making a complete copy of the caller's data space, often needlessly, since usually immediately afterwards an **exec**(3) is done. Thus, for greater efficiency, BSD introduced the **vfork**() system call, which did not fully copy the address space of the parent process, but borrowed the parent's memory and thread of control until a call to **execve**(2) or an exit occurred. The parent process was suspended while the child was using its resources. The use of **vfork**() was tricky: for example, not modifying data in the parent process depended on knowing which variables are held in a register.

CONFORMING TO

4.3BSD, POSIX.1-2001. POSIX.1-2008 removes the specification of **vfork**(). The requirements put on **vfork**() by the standards are weaker than those put on **fork**(2), so an implementation where the two are synonymous is compliant. In particular, the programmer cannot rely on the parent remaining blocked until the child either terminates or calls **execve**(2), and cannot rely on any specific behavior with respect to shared memory.

NOTES

Linux Notes

Fork handlers established using **pthread_atfork**(3) are not called when a multithreaded program employing the NPTL threading library calls **vfork**(). Fork handlers are called in this case in a program using the LinuxThreads threading library. (See **pthreads**(7) for a description of Linux threading libraries.)

History

The **vfork**() system call appeared in 3.0BSD. In 4.4BSD it was made synonymous to **fork**(2) but NetBSD introduced it again, cf. http://www.netbsd.org/Documentation/kernel/vfork.html . In Linux, it has been equivalent to **fork**(2) until 2.2.0-pre6 or so. Since 2.2.0-pre9 (on i386, somewhat later on other architectures) it is an independent system call. Support was added in glibc 2.0.112.

BUGS

It is rather unfortunate that Linux revived this specter from the past. The BSD man page states: "This system call will be eliminated when proper system sharing mechanisms are implemented. Users should not depend on the memory sharing semantics of **vfork**() as it will, in that case, be made synonymous to **fork**(2)."

Details of the signal handling are obscure and differ between systems. The BSD man page states: "To avoid a possible deadlock situation, processes that are children in the middle of a **vfork**() are never sent **SIGTTOU** or **SIGTTIN** signals; rather, output or *ioctl*s are allowed and input attempts result in an end-of-file indication."

SEE ALSO

clone(2), **execve**(2), **fork**(2), **unshare**(2), **wait**(2)

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/.