NAME

fork, fork1 – create a new process

SYNOPSIS

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

pid_t fork(void);

pid_t fork1(void);

MT-LEVEL

fork( ) is Async-Signal-Safe

DESCRIPTION

fork( ) and **fork1( )** cause creation of a new process. The new process (child process) is an exact copy of the calling process (parent process). The child process inherits the following attributes from the parent process:

- real user ID, real group ID, effective user ID, effective group ID
- environment
- open file descriptors
- close-on-exec flags (see **exec**(2))
- signal handling settings (that is, **SIG_DFL , SIG_IGN , SIG_HOLD ,** function address)
- supplementary group IDs
- set-user-ID mode bit
- set-group-ID mode bit
- profiling on/off status
- nice value (see **nice**(2))
- scheduler class (see **priocntl**(2))
- all attached shared memory segments (see **shmop**(2))
- process group ID -- memory mappings (see **mmap**(2))
- session ID (see **exit**(2))
- current working directory
- root directory
- file mode creation mask (see **umask**(2))
- resource limits (see **getrlimit**(2))
- controlling terminal
- saved user ID and group ID

Scheduling priority and any per-process scheduling parameters that are specific to a given scheduling class may or may not be inherited according to the policy of that particular class (see **priocntl**(2)). The child process differs from the parent process in the following ways:

- The child process has a unique process ID which does not match any active process group ID.
- The child process has a different parent process ID (that is, the process ID of the parent process).
- The child process has its own copy of the parent's file descriptors and directory streams. Each of the child's file descriptors shares a common file pointer with the corresponding file descriptor of the parent.

- Each shared memory segment remains attached and the value of **shm_nattach** is incremented by 1.
- All **semadj** values are cleared (see **semop**(2)).
- Process locks, text locks, data locks, and other memory locks are not inherited by the child (see **plock**(3C) and **memcntl**(2)).
- The child process's **tms** structure is cleared: **tms_utime**, **stime**, **cutime**, and **cstime** are set to 0 (see **times**(2)).
- The child processes resource utilizations are set to 0; see **getrlimit**(2). The **it_value** and **it_interval** values for the **ITIMER_REAL** timer are reset to 0; see **getitimer**(2).
- The set of signals pending for the child process is initialized to the empty set.
- Timers created by **timer_create**(3R) are not inherited by the child process.
- No asynchronous input or asynchronous output operations are inherited by the child.

Record locks set by the parent process are not inherited by the child process (see **fcntl**(2)).

MT fork( )

Solaris Threads

The following are the **fork( )** semantics in programs that use the Solaris threads API rather than the POSIX threads API (programs linked with –**lthread** but not –**lpthread**):

fork( ) duplicates all the threads (see **thr_create**(3T)) and LWPs in the parent process in the child process. **fork1( )** duplicates only the calling thread (LWP) in the child process.

POSIX Threads

The following are the **fork( )** semantics in programs that use the POSIX threads API rather than the Solaris threads API (programs linked with –**lpthread** but not –**lthread**):

The call to **fork( )** is like a call to **fork1( )**, which replicates only the calling thread. There is no call that forks a child with all threads and LWPs duplicated in the child.

Note that if a program is linked with both libraries (–**lthread** and –**lpthread**), the POSIX semantic of **fork( )** prevails.

Fork-safety

If **fork1( )** is called in a Solaris thread program or **fork( )** is called in a POSIX thread program, and the child does more than just call **exec( )**, there is a possibility of deadlocking in the child. To ensure that the application is safe with respect to this deadlock, it should use **pthread_atfork**(3T). Should there be any outstanding mutexes throughout the process, the application should call **pthread_atfork**(3T), to wait for and acquire those mutexes, prior to calling **fork( )**. (See **Intro**(3), "MT-Level of Libraries")

RETURN VALUES

Upon successful completion, **fork( )** and **fork1( )** 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 **(pid_t)**–**1** is returned to the parent process, no child process is created, and **errno** is set to indicate the error.

ERRORS

fork( ) fails and no child process are created if one or more of the following is true:

EAGAIN There are two conditions that will cause an EAGAIN error.

The system-imposed limit on the total number of processes under execution by a single user would be exceeded.

The total amount of system memory available is temporarily insufficient to duplicate this process.

ENOMEM There is not enough swap space.

SEE ALSO

alarm(2), **exec**(2), **exit**(2), **fcntl**(2), **getitimer**(2), **getrlimit**(2), **memcntl**(2), **mmap**(2), **nice**(2), **priocntl**(2), **ptrace**(2), **semop**(2), **shmop**(2), **times**(2), **umask**(2), **wait**(2), **exit**(3C), **plock**(3C), **pthread_atfork**(3T), **signal**(3C), **system**(3S), **thr_create**(3T)**, timer_create**(3R)

NOTES

Be careful to call **_exit( )** rather than **exit**(3C) if you cannot **execve( )**, since **exit**(3C) will flush and close standard I/O channels, and thereby corrupt the parent processes standard I/O data structures. Using **exit**(3C) will flush buffered data twice. See **exit**(2).

When calling **fork1( )** the thread (or LWP) in the child must not depend on any resources that are held by threads (or LWPs) that no longer exist in the child. In particular, locks held by these threads (or LWPs) will not be released.

In a multi-threaded process, **fork( )** or **fork1( )** can cause blocking system calls to be interrupted and return with an error of EINTR.