System Calls fork(2)

NAME

fork, fork1 - create a new process

SYNOPSIS

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

pid_t fork(void);

pid_t fork1(void);
```

DESCRIPTION

The fork() and fork1() functions create 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
- task ID and project ID
- processor bindings (see processor_bind(2))
- processor set bindings (see pset_bind(2))

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(3RT) are not inherited by the child process.
- No asynchronous input or asynchronous output operations are inherited by the child.
- Any preferred hardware address tranlsation sizes (see memcntl(2)) are inherited by the child.

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

Solaris Threads

In applications that use the Solaris threads API rather than the POSIX threads API (applications linked with -lthread but not -lpthread),fork() duplicates in the child process all threads (see thr_create(3THR)) and LWPs in the parent process. The fork1() function duplicates only the calling thread (LWP) in the child process.

POSIX Threads

In applications that use the POSIX threads API rather than the Solaris threads API (applications linked with -lpthread, whether or not linked with -lthread), a 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 a Solaris threads application calls fork1() or a POSIX threads application calls fork(), and the child does more than simply call exec(), there is a possibility of deadlock occurring in the child. The application should use pthread_atfork(3C) to ensure safety with respect to this deadlock. Should there be any outstanding mutexes throughout the process, the application should call pthread_atfork() to wait for and acquire those mutexes prior to calling fork() or fork1(). See "MT-Level of Libraries" on the attributes(5) manual page.

RETURN VALUES

Upon successful completion, fork() and fork1() return 0 to the child process and return the process ID of the child process to the parent process. Otherwise, (pid_t)-1 is returned to the parent process, no child process is created, and errno is set to indicate the error.

ERRORS

The fork() function will fail if:

EAGAIN

The system-imposed limit on the total number of processes under execution by a single user has been exceeded; or the total amount of system memory available is temporarily insufficient to duplicate this process.

ENOMEM

There is not enough swap space.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	fork() is Async-Signal-Safe

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(3C), signal(3C), system(3C), thr_create(3THR) timer_create(3RT), attributes(5), standards(5)

NOTES

An applications should call _exit() rather than exit(3C) if it cannot execve(), since exit() will flush and close standard I/O channels and thereby corrupt the parent process's standard I/O data structures. Using exit(3C) will flush buffered data twice. See exit(2).

The thread (or LWP) in the child that calls fork1() must not depend on any resources 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 multithreaded process, fork() or fork1() can cause blocking system calls to be interrupted and return with an EINTR error.

The fork() and fork1() functions suspend all threads in the process before proceeding. Threads that are executing in the kernel and are in an uninterruptible wait cannot be suspended immediately and therefore cause a delay before fork() and fork1() can complete. During this delay, since all other threads will have already been suspended, the process will appear "hung."

SunOS 5.9 Last change: 23 Jul 2001