Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'fork.3p' command

$ man fork.3p

FORK(3P) POSIX Programmer's Manual FORK(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fork ? create a new process

SYNOPSIS

#include <unistd.h>

pid_t fork(void);

DESCRIPTION

The fork() function shall create a new process. The new process (child process) shall be an exact copy of the calling process (parent process) except as detailed below:

* The child process shall have a unique process ID.
* The child process ID also shall not match any active process group ID.
* The child process shall have a different parent process ID, which shall be the process ID of the calling process.
* The child process shall have its own copy of the parent's file de? scriptors. Each of the child's file descriptors shall refer to the same open file description with the corresponding file descriptor

of the parent.

* The child process shall have its own copy of the parent's open di? rectory streams. Each open directory stream in the child process may share directory stream positioning with the corresponding di? rectory stream of the parent.
* The child process shall have its own copy of the parent's message catalog descriptors.
* The child process values of tms_utime, tms_stime, tms_cutime, and tms_cstime shall be set to 0.
* The time left until an alarm clock signal shall be reset to zero, and the alarm, if any, shall be canceled; see alarm().
* All semadj values shall be cleared.
* File locks set by the parent process shall not be inherited by the child process.
* The set of signals pending for the child process shall be initial? ized to the empty set.
* Interval timers shall be reset in the child process.
* Any semaphores that are open in the parent process shall also be open in the child process.
* The child process shall not inherit any address space memory locks established by the parent process via calls to mlockall() or mlock().
* Memory mappings created in the parent shall be retained in the child process. MAP_PRIVATE mappings inherited from the parent shall also be MAP_PRIVATE mappings in the child, and any modifications to the data in these mappings made by the parent prior to calling fork() shall be visible to the child. Any modifications to the data in MAP_PRIVATE mappings made by the parent after fork() returns shall be visible only to the parent. Modifications to the data in MAP_PRIVATE mappings made by the child shall be visible only to the child.
* For the SCHED_FIFO and SCHED_RR scheduling policies, the child process shall inherit the policy and priority settings of the par?

ent process during a fork() function. For other scheduling poli?
cies, the policy and priority settings on fork() are implementa?
tion-defined.

* Per-process timers created by the parent shall not be inherited by the child process.

* The child process shall have its own copy of the message queue de?
scriptors of the parent. Each of the message descriptors of the child shall refer to the same open message queue description as the corresponding message descriptor of the parent.

* No asynchronous input or asynchronous output operations shall be inherited by the child process. Any use of asynchronous control blocks created by the parent produces undefined behavior.

* A process shall be created with a single thread. If a multi-threaded process calls fork(), the new process shall contain a replica of the calling thread and its entire address space, possi?
bly including the states of mutexes and other resources. Conse?
quently, to avoid errors, the child process may only execute async-signal-safe operations until such time as one of the exec functions is called.

  When the application calls fork() from a signal handler and any of the fork handlers registered by pthread_atfork() calls a function that is not async-signal-safe, the behavior is undefined.

* If the Trace option and the Trace Inherit option are both sup?
ported:

  If the calling process was being traced in a trace stream that had its inheritance policy set to POSIX_TRACE_INHERITED, the child process shall be traced into that trace stream, and the child process shall inherit the parent's mapping of trace event names to trace event type identifiers. If the trace stream in which the calling process was being traced had its inheritance policy set to POSIX_TRACE_CLOSE_FOR_CHILD, the child process shall not be traced into that trace stream. The inheritance policy is set by a call to the posix_trace_attr_setinherited() function.

* If the Trace option is supported, but the Trace Inherit option is not supported:
  The child process shall not be traced into any of the trace streams of its parent process.
* If the Trace option is supported, the child process of a trace con? troller process shall not control the trace streams controlled by its parent process.
* The initial value of the CPU-time clock of the child process shall be set to zero.
* The initial value of the CPU-time clock of the single thread of the child process shall be set to zero.

All other process characteristics defined by POSIX.1?2008 shall be the same in the parent and child processes. The inheritance of process characteristics not defined by POSIX.1?2008 is unspecified by POSIX.1?2008.

After fork(), both the parent and the child processes shall be capable of executing independently before either one terminates.

RETURN VALUE

Upon successful completion, fork() shall return 0 to the child process and shall return the process ID of the child process to the parent process. Both processes shall continue to execute from the fork() func? tion. Otherwise, -1 shall be returned to the parent process, no child process shall be created, and errno shall be set to indicate the error.

ERRORS

The fork() function shall fail if:

EAGAIN The system lacked the necessary resources to create another process, or the system-imposed limit on the total number of pro? cesses under execution system-wide or by a single user {CHILD_MAX} would be exceeded.

The fork() function may fail if:

ENOMEM Insufficient storage space is available.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Many historical implementations have timing windows where a signal sent to a process group (for example, an interactive SIGINT) just prior to or during execution of fork() is delivered to the parent following the fork() but not to the child because the fork() code clears the child's set of pending signals. This volume of POSIX.1?2017 does not require, or even permit, this behavior. However, it is pragmatic to expect that problems of this nature may continue to exist in implementations that appear to conform to this volume of POSIX.1?2017 and pass available verification suites. This behavior is only a consequence of the imple? mentation failing to make the interval between signal generation and delivery totally invisible. From the application's perspective, a fork() call should appear atomic. A signal that is generated prior to the fork() should be delivered prior to the fork(). A signal sent to the process group after the fork() should be delivered to both parent and child. The implementation may actually initialize internal data structures corresponding to the child's set of pending signals to in? clude signals sent to the process group during the fork(). Since the fork() call can be considered as atomic from the application's perspec? tive, the set would be initialized as empty and such signals would have arrived after the fork(); see also <signal.h>.

One approach that has been suggested to address the problem of signal inheritance across fork() is to add an [EINTR] error, which would be returned when a signal is detected during the call. While this is preferable to losing signals, it was not considered an optimal solu? tion. Although it is not recommended for this purpose, such an error would be an allowable extension for an implementation.

The [ENOMEM] error value is reserved for those implementations that de? tect and distinguish such a condition. This condition occurs when an implementation detects that there is not enough memory to create the

process. This is intended to be returned when [EAGAIN] is inappropriate because there can never be enough memory (either primary or secondary storage) to perform the operation. Since fork() duplicates an existing process, this must be a condition where there is sufficient memory for one such process, but not for two. Many historical implementations ac? tually return [ENOMEM] due to temporary lack of memory, a case that is not generally distinct from [EAGAIN] from the perspective of a conform? ing application.

Part of the reason for including the optional error [ENOMEM] is because the SVID specifies it and it should be reserved for the error condition specified there. The condition is not applicable on many implementa? tions.

IEEE Std 1003.1?1988 neglected to require concurrent execution of the parent and child of fork(). A system that single-threads processes was clearly not intended and is considered an unacceptable ``toy implemen? tation'' of this volume of POSIX.1?2017. The only objection antici? pated to the phrase ``executing independently'' is testability, but this assertion should be testable. Such tests require that both the parent and child can block on a detectable action of the other, such as a write to a pipe or a signal. An interactive exchange of such actions should be possible for the system to conform to the intent of this vol? ume of POSIX.1?2017.

The [EAGAIN] error exists to warn applications that such a condition might occur. Whether it occurs or not is not in any practical sense under the control of the application because the condition is usually a consequence of the user's use of the system, not of the application's code. Thus, no application can or should rely upon its occurrence under any circumstances, nor should the exact semantics of what concept of ``user'' is used be of concern to the application developer. Valida? tion writers should be cognizant of this limitation.

There are two reasons why POSIX programmers call fork(). One reason is to create a new thread of control within the same program (which was originally only possible in POSIX by creating a new process); the other

is to create a new process running a different program. In the latter case, the call to fork() is soon followed by a call to one of the exec functions.

The general problem with making fork() work in a multi-threaded world is what to do with all of the threads. There are two alternatives. One is to copy all of the threads into the new process. This causes the programmer or implementation to deal with threads that are suspended on system calls or that might be about to execute system calls that should not be executed in the new process. The other alternative is to copy only the thread that calls fork(). This creates the difficulty that the state of process-local resources is usually held in process memory. If a thread that is not calling fork() holds a resource, that resource is never released in the child process because the thread whose job it is to release the resource does not exist in the child process.

When a programmer is writing a multi-threaded program, the first de? scribed use of fork(), creating new threads in the same program, is provided by the pthread_create() function. The fork() function is thus used only to run new programs, and the effects of calling functions that require certain resources between the call to fork() and the call to an exec function are undefined.

The addition of the forkall() function to the standard was considered and rejected. The forkall() function lets all the threads in the parent be duplicated in the child. This essentially duplicates the state of the parent in the child. This allows threads in the child to continue processing and allows locks and the state to be preserved without ex? plicit pthread_atfork() code. The calling process has to ensure that the threads processing state that is shared between the parent and child (that is, file descriptors or MAP_SHARED memory) behaves properly after forkall(). For example, if a thread is reading a file descriptor in the parent when forkall() is called, then two threads (one in the parent and one in the child) are reading the file descriptor after the forkall(). If this is not desired behavior, the parent process has to synchronize with such threads before calling forkall().

While the fork() function is async-signal-safe, there is no way for an implementation to determine whether the fork handlers established by pthread_atfork() are async-signal-safe. The fork handlers may attempt to execute portions of the implementation that are not async-signal-safe, such as those that are protected by mutexes, leading to a dead?lock condition. It is therefore undefined for the fork handlers to ex?ecute functions that are not async-signal-safe when fork() is called from a signal handler.

When forkall() is called, threads, other than the calling thread, that are in functions that can return with an [EINTR] error may have those functions return [EINTR] if the implementation cannot ensure that the function behaves correctly in the parent and child. In particular, pthread_cond_wait() and pthread_cond_timedwait() need to return in or?der to ensure that the condition has not changed. These functions can be awakened by a spurious condition wakeup rather than returning [EINTR].

FUTURE DIRECTIONS

None.

SEE ALSO

alarm(), exec, fcntl(), posix_trace_attr_getinherited(), posix_trace_eventid_equal(), pthread_atfork(), semop(), signal(), times()

The Base Definitions volume of POSIX.1?2017, Section 4.12, Memory Syn?chronization, <sys_types.h>, <unistd.h>

COPYRIGHT

at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker?nel.org/doc/man-pages/reporting_bugs.html .