IEEE Standards Interpretations for IEEE Std 1003.1c™-1995 IEEE Standard for Information Technology--Portable Operating System Interface (POSIX(R)) - System Application Program Interface (API) Amendment 2: Threads Extension (C Language)

Interpretation Request #37
Topic: pthread_atfork **Relevant Clauses:** 3.1.1.2 and 3.3.1.3

This is an interesting and potentially nasty inconsistency in the standard, which someone just pointed out to me as a result of reviewing the UNIX98 spec. Clause 3.3.1.3 (Signal Actions), paragraph (3)(f), states that fork is async-signal safe. That is, it may be called from an asynchronous signal catching function. Very traditional, and more or less intentional. I also believe we didn't completely think through all the implications.

1003.1c-1995 added pthread_atfork, which runs USER CODE as part of fork. The specific intent of atfork handlers is to protect mutexes and shared data across the fork. That means they are nearly useless unless they lock mutexes. Mutexes cannot be locked within a signal catching function. Therefore... either 1) atfork handlers cannot be run when fork is called from a signal catching function, or, 2) fork cannot be called from a signal catching function when there are atfork handlers. And, because atfork handlers can be used "anywhere", in independent libraries, this essentially means that fork cannot be called from a signal catching function. Note, further, that 3.1.1.2, paragraph (16), specifically states that when fork is called in a multithreaded process, "the child process may only execute async-signal safe operations until such time as one of the exec functions is called".

This statement holds regardless of whether fork is called from an async signal catching function, rendering the entire concept of atfork handlers essentially useless (or at least, non-portable). Perhaps this is a good idea, and atfork handlers should be removed from the standard. Perhaps there should be a requirement in the standard that implementations shall provide POSIX and ANSI C functions that use atfork and/or equivalent mech-

anisms as necessary to insure that a child process can continue running its copy of the parent program instead of just exec-ing. Such a requirement, however, MUST exclude forks from async signal handlers, as the function of atfork handlers cannot be accomplished asynchronously.

If we keep atfork handlers (it still seems like a good idea, basically), and if we allow fork to be called from an async signal catching function (this is ancient tradition, and breaking it would probably not be a good idea), then either

1) atfork handlers are IGNORED when fork is called from a signal handler. (This requires implementations to keep track of when the thread is "at signal level", which may be easy but is not at all common practice.)
2) fork FAILS when called from a signal handler when there are atfork (or "atfork-like") handlers installed. I'm not sure this is much different from removing fork from the async-signal safe list, unless there are requirements placed upon libpthread, libc, and so forth, to require no cleanup on fork (which is not reasonable). And, of course, even if "the implementation" doesn't require cleanup around fork, the caller of fork cannot know whether some other library in the process has declared an atfork handler. The result is, at best, unpredictable.
3) fork detects that it is run from a signal handler, and that there are multiple threads, and atfork handlers, and fails; but will succeed (and skip atfork handlers) if there is only one thread.
4) Require that implementations do not deliver asynchronous signals to any thread running program code. The implementation could, for example, create a "hidden" dedicated thread that loops on sigsuspend(), and have all asynchronous signals delivered to that thread. Because it would never lock mutexes, there would be no self-deadlock problem from asynchronous execution of atfork handlers. This, however, would also require that a thread can lock, and potentially block upon (!) a mutex within an asynchronous signal handler -- it may be possible or even easy for some implementations, but it is not a reasonable requirement upon all implementations.

I believe that as the standard is currently written, 3.1.1.2, paragraph (16) normatively states that atfork handlers are of no use, and a child process can call only async-signal safe functions until exec. There is no normative text describing what atfork handlers may be used for, or what a programmer may expect of them, only that pthread_atfork provides 3 user routines that are called in the proper places. One may reasonably infer from 3.1.1.2 that atfork handlers may call only async-signal safe functions, and that, regardless of atfork handlers, the child may still do nothing other than call async-signal safe functions until exec.

A reasonable conclusion is that a libc and libpthread (and any others) need not provide protection for their own internal state across a fork, because that state will not be impacted in the parent and cannot be used in the child, in any case. This was not the intent of the working group. I would like to see the wording repaired so that, given proper use of atfork handlers, the child CAN continue "normally", even to using thread synchroniza-

tion objects. The implementation must also be required to protect itself across a fork, for this to occur. But it should NOT be required to protect itself against a fork from a signal handler, because this may be impossible to accomplish.

Interpretation Response
The standard is clear that conforming applications: 1) shall only call the set of functions defined on page 78 lines 843-863 from signal handling routines 2) if fork is called from a signal handling routine, then the 3 routines defined by pthread_atfork shall only use the set of functions defined on page 78 lines 843-863. If a signal handling routine calls a function not in the list of async safe functions, then, as stated on page 78 line 872, the behavior is undefined an conforming implementations shall handle any behaviors that result. The Interpretation Committee believes that, while clear, 1) and 2) could use a little more explanation and is recommending to the sponsor that the following explanatory additions be made: Pg 78 line 864 "Additionally, invocations of the fork handlers set by pthread_atfork from a fork called from a signal handler are required to be async safe." and Pg 64, line 297 or so.. "If the fork handlers use functions async unsafe then the behavior is undefined if fork is called from signal handlers." The committee does not believe that these in anyway change the requirements of the standard Additionally, while the standard is clear in what it says it appears that the facilities provided are not quite what the working and balloting group intended. The rationale is very specific that the handlers should be designed so they **can't** be used when fork is called in a signal handler which under normal programming practices meant that they should 'acquire' the mutexes, but the standard does not require that mutex operations be async safe. This is being referred to the sponsor for consideration.

Rationale for Interpretation
None.