IEEE Standards Interpretation for IEEE Std 1003.1™-2001 IEEE Standard Standard for Information Technology -- Portable Operating System Interface (POSIX®)

Interpretation Request #67
Topic: pthread_exit(), funlockfile **Relevant Sections:** XSH pthread_exit() Page: 0 Line: 0

The flockfile functions (flockfile(), ftrylockfile(), and funlockfile()) can be used to lock and unlock FILE* objects. They can be used to specify a sequence of I/O statements. This locking of files is much like the locking of mutexes. For example, when a file is locked by a thread, only that locking thread can operate on the file. The issue comes about when the locking thread exits without unlocking its locked files. The spec doesn't discuss whether or not the locks should be released when the locking thread calls pthread_exit().

The only relevant statement within the standard is this one on the pthread_exit page:

"Thread termination does not release any application visible process resources, including, but not limited to, mutexes and file descriptors, nor does it perform any process-level cleanup actions, including, but not limited to, calling any atexit() routines that may exist."

We would like clarification if this applies to flockfile et al. That is is the application's responsibility to funlockfile() any locks before the thread exits? Or should the implementation clean up the locks on pthread_exit()?

Interpretation Response #67
The standard clearly states that is the applications responsibility to call funlockfile() before a thread exits.

Rationale for Interpretation
pthread_exit() is described in the standard and the equivalent operation to funlockfile() is not on the list of actions to perform and thus it is not allowed to do it.

The standard states on page 383 lines 12369-12370 "The funlockfile() function shall relinquish the ownership granted to the thread. The behavior is undefined if a thread other than the current owner calls the funlockfile() function."

It is the applications responsibility to call funlockfile before the thread exits.