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 #22
Topic: pthread_setspecific **Relevant Clauses:** 17.1.2.2, page 166 D10, lines 192-195

This paragraph states that calling pthread_setspecific() from within a destructor function may result in lost storage or infinite loops. This can be a real problem. POSIX.1c has just stated that it is impossible to use thread-specific data in an application. A portable and correctly behaving application cannot rely on calling pthread_setspecific() from within a destructor function. What makes thread-specific data impossible to use is the fact that in pthread_key_create() it is stated that when a thread terminates, any non-NULL TSD values which have destructors will have the destructor called for them.

The problem here is that the destructor function is called in a loop (potentially forever -- so a portable application has to assume forever) until the key value is NULL. However, according to pthread_setspecifc() a portable and correctly behaving application has no way possible to change the key value to a NULL value. Portable application have to rely on "worst-case" guaranteed behavior.

According to the definitions of pthread_setspecific() and pthread_key_create() an application cannot reliably make use of thread-specific data with destructor functions or its thread will end up in an infinite loop. Could you please clarify the intended behavior of these functions? Obviously there must be something missing here or this looping behavior for destructor functions would not have been added since an application must assume it results in an infinite loop.

Interpretation Response
This is a duplicate. See Interpretation #3, part 6.

Rationale for Interpretation
None.