Document: N1687

C TSS Destructor Specification

Owen Shepherd

Public Domain C Library Project

The full revision history of this document may be found at https://e43oss.atlassian.net/wiki/display/PDCLIB/C+TSS+Destructor+Specification. Editing of that page is restricted, but comments are welcomed

Table of Contents

Introduction

The final release of the C11 international standard leaves many aspects regarding thread local storage unspecified. Specifically, the following aspects are unspecified:

- If or when destructors for thread specific storage ("tss") objects are invoked
- The ordering (or lack thereof) of destructor invocation
- The identity of the thread invoking the TSS destructors
- The number of times TSS destructors may be invoked (and the meaning of the `TSS_DTOR_ITERATIONS` constant)
- The behavior of TSS destructor invocation in the face of parallel modifications

This ambiguity leaves the utility of the TSS feature in a fully conforming application (i.e. one not relying on additional assertions by the implementation) greatly reduced. In particular, it prevents the usage of the thread specific storage feature for reliable resource cleanup

This proposal is submitted in relation to specification defect reports DR 416 (by the same author as this proposal) and DR 424. This proposal will look at existing implementations of thread specific storage and their behavior, and will then propose alterations to the C11 standard .

Existing implementations and specifications

This proposal will look at the specifications of thread specific storage and related mechanisms under two common platforms (POSIX.1 2008 and Microsoft Windows) and additionally at the defined behaviour of the C++11 international standard.

POSIX.1 2008

POSIX.1 implements thread specific storage under the POSIX Threads ("pthreads") API. It is implemented in terms of the type `pthread_key_t`, which mirrors `tss_t`, and four functions, which exactly mirror those provided by the C11 standard:

C11 Function	POSIX.1 Function
tss_create	pthread_key_create
tss_get	pthread_getspecific
tss_set	pthread_setspecific
tss_delete	pthread_key_delete

In addition, POSIX.1 defines the constant `PTHREAD_DESTRUCTOR_ITERATIONS` which has a description which presumably matches the intent of the C11 specification's `TSS_DTOR_ITERATIONS` constant.

POSIX.1 defines that, at thread exit time

- For each key which was created with a destructor, the value associated with the key will be set to `NULL` and the key's destructor will be invoked with the value that the key had immediately prior to being set to `NULL`
- The ordering of destructor calls for distinct keys is undefined
- If after invoking the destructor for each key created with one there remain keys with destructors which have values which are non-NULL, the process will be repeated up to `PTHREAD_DESTRUCTOR_ITERATIONS` times.
- POSIX.1 leaves undefined the behaviour of invoking `pthread_exit` (the function which exits a thread, analogously to C11's `thrd_exit`) from within a destructor

POSIX.1 defines that `pthread_key_delete` and `exit` do **not** cause destructor invocations.

POSIX.1 leaves undefined whether destructors for keys created or destroyed concurrently with a thread running destructors (in another thread, or from a destructor running on the thread executing destructors) will alter the set of destructors run by said thread.

Microsoft Windows

Thread Specific Storage on Microsoft Windows is implemented in terms of four functions:

```
// Common Win32 API types, defined for those unfamiliar:
typedef uint32_t DWORD;
typedef void *LPVOID;

// Win32 TLS functions
DWORD TlsAlloc(void);
BOOL TlsFree(DWORD dwTlsIndex);
BOOL TlsSetValue(DWORD dwTlsIndex, LPVOID lpTlsValue);
LPVOID TlsGetValue(DWORD dwTlsIndex);
```

The function `TlsAlloc` is used in order to allocate a new thread specific storage "index.", Note that the Windows API does not directly have any concept of a thread local storage destructor. The `TlsFree` function is used to deallocate a thread specific storage index. The `TlsSetValue` and `TlsGetValue` functions, aside from the differences in the types involved and method of indicating errors, behave in the same manner as the `tss_set` and `tss_get` functions in the C11 standard.

Windows does not directly provide destructor support for thread specific storage objects, but does provide mechanisms by which they may be implemented:

- A dynamic link library ("DLL") may provide an entry point, conventionally called `DllMain`, which receives notifications on various events, including thread startup and termination
- On recent versions of Windows (5.1/XP, released 2001) and above, an executable may request to have similar notifications delivered to one or more functions by placing a structure pointing to them in a table placed in a specifically located executable segment

Either method may be used to implement thread local storage destructors (the former is used by, for example, the pthreads-win32 library to implement the POSIX.1 threading library on top of Windows).

A caveat which must be noted with either of the above methods of implementing thread specific storage destructors is that in both cases the notifications are delivered while the system holds a lock on an internal mutex (The "Loader lock", which is also taken internally during calls to certain functions exposed by the system)

C++2011

C++2011 introduces the language keyword `thread_local`, aligned to the `thread_local` macro introduced by `<threads.h>` and `_Thread_local` keyword in `C11`, which introduces an object of thread local storage duration. C++ objects have both constructors and destructors, and therefore must be allocated and constructed before first use in a thread, and destroyed and deallocated at thread exit.

C++ defines that thread local storage destructors are called

- As a result of calling `exit` for objects associated with the thread that invoked `exit`, prior to commencing invocation of functions registered with `atexit`
- Upon return from `main`, for the objects associated with the process' initial thread (following the rule that an application which returns from `main` shall behave as if `exit` was invoked with the value returned)
- Upon thread exit (in unspecified order)

It is noted that the requirement that destructors be called from `exit` differs from that of POSIX.1. It is also noted that C++ does not need to invoke destructors multiple times because of the nature of C++ objects.

Proposed Behavior

The proposed behavior is to align the C specification behavior with POSIX.1

Implementation Considerations

The behavior of the thread specific storage primitives defined in the POSIX.1 specification are thought to be possible to implement on all platforms which support threads. On platforms which do not implement thread specific storage destructors natively, or which do so in a manner incompatible with the mechanisms defined by POSIX.1 can be handled either by

- Implementation of the invocation of thread specific storage destructors using a platform dependent mechanism, as done by `pthreads-win32`, referenced above
- Implementation of the invocation of thread specific storage destructors entirely within the C library

An example implementation would be for the C library to invoke destructors manually from within the `thrd_exit` function. This would be sufficient to support fully conforming C programs, though may not be useful for applications where some components may not use the C library threading primitives.

Proposed Technical Corrigendum

This proposed corrigendum is a lightly edited version of that originally proposed in DR 416:

After 7.26.5.1p2, add

> Returning from `func` shall have the same behaviour as invoking `thrd_exit` with the returned value

Change 7.26.5.5 part 2 from

> The `thrd_exit` function terminates execution of the calling thread and sets its result code to `res`.

to

> For every thread specific storage key which was created with a non-NULL destructor and for which the value is non-NULL, `thrd_exit` shall set the value associated with the key to NULL and then invoke the destructor with its previous value. The order in which destructors are invoked is unspecified.
>
> If after this process there remain keys with both non-NULL destructors and values, the implementation shall repeat this process up to `TSS_DTOR_ITERATIONS` times.
>
> Following this, the `thrd_exit` function terminates execution of the calling thread and sets its result code to `res`.

After 7.26.6.1p2, add

> The value NULL shall be associated with the newly created key in all existing threads. Upon thread creation, the value associated with all keys shall be initialized to NULL
>
> Note that destructors associated with thread specific storage are not invoked at process exit.
>
> It is undefined to call `tss_create` from within a destructor invocation.
>
> It is undefined if calls to `tss_set, tss_get` or `tss_delete` for a storage are valid on a thread if the `tss_create` call which allocated it completed after the thread commenced executing destructors.

To 7.26.6.2p2, append

> If `tss_delete` is called while another thread is executing destructors, whether this will affect the number of invocations of the destructor associated with `key` on that thread is unspecified. If the thread from which `tss_delete` is invoked is executing destructors, then no further invocations of the destructor associated with `key` will occur on said thread.
>
> Calling `tss_delete` will not result in the invocation of any destructors.

After 7.26.6.4p2, add

> This action will not invoke the destructor associated with the key on the value being replaced.

(This additionally clarifies whether or not a destructor will be invoked for a storage created after a thread has already begun executing destructors: because `tss_set` is an undefined operation, a value may never be associated with the storage and therefore the destructor may never be invoked)

Changes since DR 416:

- Revert change to 2.26.5.5 part 3 to language in the existing international standard
- State that invoking `tss_create` from within a destructor invocation is undefined
- State that invoking `tss_set`, `tss_get` or `tss_delete` on a storage created after a thread has begun executing destructors from within that thread is undefined

Change History

Version	Date	Comment
Current Version (v. 3)	**Mar 19, 2013 15:41**	**Owen Shepherd**: Revision for submission
v. 2	Mar 05, 2013 23:17	**Owen Shepherd**: Include link to page in header. Modify page labels
v. 1	Mar 05, 2013 23:11	**Owen Shepherd**

Page Properties

Standard	ISO C11
Resolution	Open
Status	Submitted
WG Documents	