Underspecified Aspects of Threads in C

David Keaton, Jens Gustedt

Underspecified Aspects of Threads in C

David Keaton **Jens Gustedt**

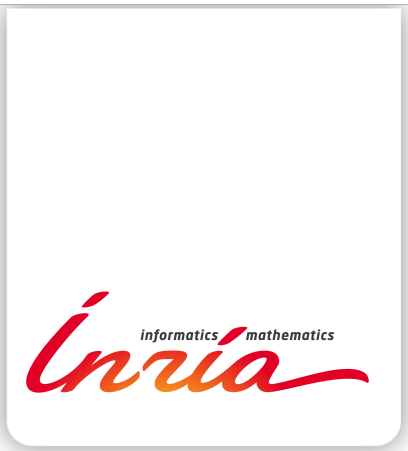
TECHNICAL REPORT

N° 470

November 2015

Project-Team Camus

ISSN 0249-0803 ISRN INRIA/RT--470--FR+ENG



Underspecified Aspects of Threads in C

David Keaton[*] Jens Gustedt[†‡*]

Project-Team Camus

Technical Report n° 470 — November 2015 — 6 pages

Abstract: To provide a standard method for using threads in C, a threads interface had been added to C11. It unifies various existing application programming interfaces (APIs) so that software developers could write portable multithreading code. C11 threads were deliberately defined with the least constraints such that they could be implemented as a standard veneer over native threads of a given platform. However, in some cases too much was left out of the specification. This document lists such cases and tracks the proposed solutions by the C standards committee.

Key-words: C standard, threads

[*] CERT, USA
[†] Inria, France
[‡] Université de Strasbourg, ICube, France

RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Aspects sous-specifiés des processus légers en C

Résumé : Pour fournir une méthode standardisée d'utilisation de processus légers à partir de C, une interface de processus légers a été ajouté à C11. Pour permettre aux développeurs de logiciel d'écrire du code multi-processus de façon portable, elle unifie plusieurs interfaces de programmation. Les processus légers C11 ont délibérément étés spécifiés avec le moins de contraintes possibles pour permettre de les implanter en tant que revêtement standardisé au dessus des processus légers d'une plate-forme donnée. Néanmoins, dans quelques cas trop a été omis de la spécification. Ce document liste de tels cas et trace les solutions proposés par le comité de standardisation.

Mots-clés : standard C, processus légers

1 Introduction

Today, even many embedded devices contain multicore processors. As squeezing additional performance out of each core becomes more and more difficult, new devices will continue to adopt multicore solutions, and the number of cores per processor will increase as well. To keep up with the evolving hardware, software developers have no choice but to use multithreading. An underspecified thread API can lead developers to make incorrect assumptions, or to be unsure of the expected behavior. This can cause deadlock, dangling pointers, and other conditions that can make code insecure.

When this occurs in ordinary equipment, such as devices purchased by a consumer, it can lead to the devices being used as part of a botnet to attack infrastructure. When it occurs in defense-critical systems, it can cause even more severe security problems.

Threads were recently added to the C programming language. The purpose was to provide a standard method for using threads in C, unifying various existing application programming interfaces (APIs) so that software developers could write portable multithreading code. C threads were defined such that they could be implemented on top of any of the existing thread implementations, providing a standard veneer over native threads.

To implement this standard veneer, C threads needed to be flexible enough to accommodate all the idiosyncrasies of different underlying thread packages. This was accomplished by specifying various aspects of C threads more loosely than other thread packages, so that the behavior of existing APIs would be considered valid. This is a helpful approach because it enables the development of a portable API, which was not available before. However, there are some cases where too much was left out of the specification.

In the present work, the C standard as presented in C (2011) was studied to identify underspecified aspects of C threads. The section below indicates the clauses of the C standard that are affected and relates it to existing *defect reports* (DR, summarized in Section 4) and other documents of the C committee (in the bibliography).

2 Items in the C Standard

5.1.2.4 Multi-threaded executions and data races

Introducing a concept of a *blocked thread* may help clarify some of the other items. If so, ISO vocabulary standard (Vocab (2015)) may be helpful.

7.26.2.1 The call_once function

This function is used when a collection of threads all execute the same code, and there is some initialization that must be performed by just one of those threads before any of the threads can proceed.

Paragraph 2 states that completion of an effective call "synchronizes with all subsequent calls," implying that the side effects of the first call will be visible to all calls initiated after the end of the first call. This is insufficient, because the completion of the first call must synchronize with all calls that occur after the beginning of the first call as well. Otherwise, some threads will proceed with their work without the side effects of the first call being visible.

It is unclear what happens if the flag argument to call_once has not been initialized with ONCE_FLAG_INIT. This should probably be undefined behavior.

7.26.4 Mutex functions

Riegel (2014) asks about the state of a mutex if its locking thread exits. Ongoing DR 469 treats this problem.

7.26.4.5 The mtx_trylock function

Riegel and Boehm (2014) asks if mtx_trylock may fail spuriously, that is if it may fail without an apparent reason. Such a behavior is allowed by POSIX (2009). Ongoing DR 470 treats this problem.

7.26.5.1 The thrd_create function

This function creates a new thread that executes a specified function. The standard does not specify what happens when that specified function returns. Defect Report 416 (resolved but not yet published) remedies this by clarifying that such a return is equivalent to calling thrd_exit with the returned value.

7.26.5.5 The thrd_exit function

This function terminates the current thread. It is unclear what happens if it is called from within a thread-specific-storage destructor.

7.26.6.1 The tss_create function

This function creates thread-specific storage and assigns a destructor for it. The standard does not specify when the destructor is called. DR 416 remedies this by specifying that the destructor is called by thrd_exit.

7.26.6.2 The tss_delete function

This function releases thread-specific storage. In the standard, it is unclear what tss_delete will do with storage that was created after destructors began executing (including storage that is created within a destructor).

DR 416 attempted to remedy this by specifying that a call to tss_delete on such storage would result in undefined behavior. However, DR 416's statement of the condition is flawed. It specifies that for the behavior to be defined, the associated call to tss_create must complete before destructors began executing. This is necessary but not sufficient. Not only must the call to tss_create complete, but its side effects must be visible to the destructors as well.

7.26.6.3 The tss_get function

This function retrieves a value from thread-specific storage. In the standard, it is unclear what will happen if tss_get is called on storage that was created after destructors began executing (including storage that is created within a destructor).

DR 416 attempted to remedy this by specifying that a call to tss_get on such storage would result in undefined behavior. However, DR 416's statement of the condition is flawed. It specifies that for the behavior to be defined, the associated call to tss_create must complete before destructors began executing. This is necessary but not sufficient. Not only must the call to tss_create complete, but its side effects must be visible to the destructors as well.

7.26.6.4 The tss_set function

This function stores a value into thread-specific storage. In the standard, it is unclear what will happen if tss_set is called on storage that was created after destructors began executing (including storage that is created within a destructor).

DR 416 attempted to remedy this by specifying that a call to tss_set on such storage would result in undefined behavior. However, DR 416's statement of the condition is flawed. It specifies that for the behavior to be defined, the associated call to tss_create must complete before destructors began executing. This is necessary but not sufficient. Not only must the call to tss_create complete, but its side effects must be visible to the destructors as well.

K.3.6.1 Runtime-constraint handling

Annex K specifies a set of bounds checking functions to be used in place of less safe legacy functions. In case of a bounds violation, these functions call a constraint handler. However, Annex K was written before threads were added to the language, and the specification of constraint handlers assumes a single-threaded execution model. Constraint handlers need to be updated to take threads into account. For example, a library routine in one thread should be able to change its constraint handler without affecting application code in another thread. See O'Donell and

Sebor (2015a,b) for a general discussion about Annex K, and Sebor (2015b,a) for the problems with runtime constraint handlers.

3 Next steps

Tightening up the specification is tricky. It must be done in a way that still allows all the differing behaviors of the underlying thread packages. Therefore, to tighten up the C threads specification, each relevant operating system needs to be studied to make sure its needs are taken into account.

These areas of underspecification need to be evaluated against POSIX (2009) to ensure that any remedy does not make it impossible to implement C threads on top of POSIX threads.

The areas of underspecification also need to be evaluated against Windows and against a representative set of embedded operating systems to ensure that C threads remain implementable on top of those threading models.

Finally, once all the relevant operating systems have been taken into account, proposals to remedy the areas of underspecification can be written and submitted to the C committee.

4 Defect Reports related to the C11 thread interface

DR 405 *The mutex specification* Status closed, adds a TC that clarifies the total ordering of lock and unlock operations on a mutex.

DR 406 *Visible sequences of side effects are redundant.* Status open, only relevant in connection with the optional C11 atomics interface and other memory consistency models than memory_order_seq_cst.

DR 408 *Intra-thread synchronization.* Status closed, no TC.

DR 414 *Typos in 6.27 Threads <threads.h>.* Status closed, a TC with editorial changes.

DR 416 *tss_t destruction unspecified.* Status closed, a TC elaborates the committee's intent for the exit of threads and the destruction feature for tss_t.

DR 424 *Underspecification of tss_t.* Status closed, solution integrated with DR 416.

DR 449 *What is the value of TSS_DTOR_ITERATIONS for implementations with no maximum?* Status closed, no TC.

DR 469 *Lock ownership vs. thread termination.* Status open, work to clarify the semantics of recursive mutexes is ongoing.

DR 470 *mtx_trylock should be allowed to fail spuriously.* Status open, discussion converges to a TC that allows mtx_trylock to fail spuriously.

References

Aaron Ballman. 2013a. *Implicit thrd_exit()*. Document n1750. ISO JTC1/SC22/WG14. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1750.htm

Aaron Ballman. 2013b. *Thread-specific storage destructor invocation*. Document n1751. ISO JTC1/SC22/WG14. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1751.htm

Marc Batty. 2011. *Late changes to the C++ memory model*. Document n1584. ISO JTC1/SC22/WG14. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1584.pdf

C. 2011. *ISO/IEC Information technology – Programming languages – C*. Vol. 9899:2011. ISO, Geneva, Switzerland. 3rd ed.

Blaine Garst. 2015a. *Proposal for DR 469*. Document n1907. ISO JTC1/SC22/WG14. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1907.htm

Blaine Garst. 2015b. *Proposal for DR 470*. Document n1922. ISO JTC1/SC22/WG14. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1922.pdf

Jens Gustedt. 2012a. *underspecification of tss_t*. Document n1651. ISO JTC1/SC22/WG14. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1651.htm

Jens Gustedt. 2012b. *underspecification of thread functions with void return*. Document n1654. ISO JTC1/SC22/WG14. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1654.htm

Carlos O'Donell and Martin Sebor. 2015a. *Field Experience With Annex K — Bounds Checking Interfaces*. Document n1967. ISO JTC1/SC22/WG14. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1967.htm

Carlos O'Donell and Martin Sebor. 2015b. *Updated Field Experience With Annex K — Bounds Checking Interfaces*. Document n1969. ISO JTC1/SC22/WG14. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1969.htm

Tom Plum. 2012. *Typos in 6.27 Threads <threads.h>*. Document n1614. ISO JTC1/SC22/WG14. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1614.htm

POSIX. 2009. *ISO/IEC/IEEE Information technology – Portable Operating Systems Interface (POSIX®) Base Specifications*. Vol. 9945:2009. ISO, Geneva, Switzerland. Issue 7.

Torvald Riegel. 2014. *Lock ownership vs. thread termination*. Document n1881. ISO JTC1/SC22/WG14. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1881.htm

Torvald Riegel and Hans Boehm. 2014. *mtx_trylock should be allowed to fail spuriously*. Document n1882. ISO JTC1/SC22/WG14. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1882.htm

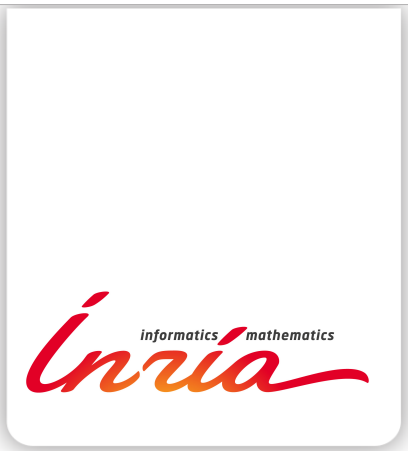
Martin Sebor. 2015a. *Proposal for thread-safe set_constraint_handler_s*. Document n1962. ISO JTC1/SC22/WG14. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1962.htm

Martin Sebor. 2015b. *thread safety of set_constraint_handler_s*. Document n1866. ISO JTC1/SC22/WG14. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1866.htm

Owen Shepherd. 2012. *tss_t destruction unspecified*. Document n1627. ISO JTC1/SC22/WG14. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1627.htm

Vocab. 2015. *ISO/IEC/IEEE Information technology – Vocabulary*. Vol. 2382:2015. ISO, Geneva, Switzerland.

Douglas Walls. 2013. *What is the value of TSS_DTOR_ITERATIONS for implementations with no maximum?* Document n1744. ISO JTC1/SC22/WG14. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1744.htm



RESEARCH CENTRE
NANCY – GRAND EST

615 rue du Jardin Botanique
CS20101
54603 Villers-lès-Nancy Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-0803