Proposed POSIX threads permit object

Mr. Niall Douglas <http://www.nedproductions.biz/>

July 2012

My name is Niall Douglas and I am currently the ISO JTC1 SC22 convenor for the Republic of Ireland. The POSIX threads permit objects proposed by this document came from internal deliberations by WG14 during the preparation of the C11 standard – I am highly indebted to those on the committee who gave so freely of their time and thoughts. My thanks in particular are due to Hans Boehm without whose detailed feedback this proposal would look completely different. My thanks are also due to John Benito for his seemingly never tiring efforts on the behalf of C-ish programmers everywhere.

# What is being proposed?

Two asynchronous job completion notification objects for POSIX threads a.k.a. a “permit object”. The first type is the simpler and permits a single thread to pass per notification. The second type is slightly more complex and can permit multiple threads to pass per notification.

# Is there an existing implementation?

There is a reference implementation written in C11 at <https://github.com/ned14/ISO_POSIX_standards_stuff/tree/master/pthreads%20Notifer%20Object>

It contains support for Microsoft Windows 7 and POSIX. It has been tested on Microsoft Visual Studio 2010, GCC v4.6 and clang v3.2.

It contains a full set of unit tests written using CATCH. The simple permit object costs 48/0/142 CPU cycles for grant/revoke/wait uncontended and 359/4/372 cycles when contended between two threads. The more complex permit object costs a maximum of 102/0/137 cycles uncontended. These results are for an Intel Core 2 processor.

# Are permit objects existing standard practice?

The proposed permit objects very closely mirror Java’s fundamental permit object.

C++11’s futures and promises implementation could use a permit when one thread notifies another thread that it may proceed.

# Why is it necessary that a permit object be added to POSIX threads?

There are many occasions in threaded programming when a third party library goes off and does something asynchronous in the background. In the meantime, the foreground thread may do other tasks, occasionally polling a notification object to see if the background job has completed, or indeed if it runs out of foreground things to do, it may simply sleep until the completion of the background job or jobs. Put simply, the foreground threads polls or waits for *permission* to continue.

**The problem is that naive programmers think a wait condition is suitable for this purpose. This is highly incorrect due to the problem of spurious and lost wakeups inherent to wait conditions. Despite the documentation saying this, wait conditions are frequently proposed as the “correct” solution in many “expert advice” internet sites including stackflow.com among others. The present lack of asynchronous notification objects in POSIX leads to too many “roll your own” implementations which are too frequently subtly broken. This leads to unreliability in threaded programming.**

A completely safe notification object can be built from atomics and wait conditions – indeed, what is proposed is entirely built this way. The problem is rather one of **standardisation** on a safe, efficient, and well tested implementation.