Two Page Summary of 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, WG14 convenor, 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 extremely simple and lightweight and permits a single thread to pass per notification. The second type is more complex and can permit multiple threads to pass per notification, as well as allowing infinite composition (a.k.a. select()).

# 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. It is a particularly lightweight implementation which uses no dynamic memory (except for fd association) which makes it suitable for bootstrap environments.

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. In fact, it is hoped that composability will be added to C++11’s futures and promises via this new permit object (see Intended consequences from adoption).

# 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 for pthread\_cond 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 standardised, safe 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. Furthermore, there is a problem of interoperability – how can third party libraries interoperate easily when each rolls its own asynchronous notification object.

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.

# Concerns about implementation and adoption

During pre-C11 discussions amongst interested members of the WG14 committee, there was a general consensus that standardisation of asynchronous notification objects was a worthwhile idea. There was no consensus on what form this should take, and the enclosed design is one of many considered. It was therefore suggested that post-C11 the proposal should be forwarded to the Austin Working Group for consideration of the wider impact of adoption, including:

* What effect will these permits have on the implementation of kernel wait queues? Linux, FreeBSD and DragonflyBSD all take dimorphic approaches to kernel wait queues, and other POSIX implementations even more so. The proposed permit could be greatly optimised through explicit kernel support, but what considerations of these should this permit design incorporate? The AWG should be a good forum for vendors to advise on consequences for their platforms.
* What effect will these permits have on NUMA hardware designs? The current implementation loops atomic operations which does not scale well with NUMA nodes. Are there future hardware features which could help e.g. memory transactions which this permit design should consider?

# Intended consequences from adoption

It is hoped to mirror any added POSIX threads permit object in future revisions of the C programming language standard.

It is hoped that C++’s futures and promises implementation can be revised to support composition using this object, while providing a C compatible API.

It is hoped that a POSIX threads permit object would be the first step in standardising how kernels asynchronously notify userspace code in a portable fashion.