**Digidesign Developer Services -- Tech Note #21**

# Plug-ins:
# Important Threading Issue on OSX and RTAS Performance

---

**Applies to:** Pro Tools 7.0, Plug-In SDK 7.0 (Final Candidate or higher)
**Originally Posted:** Thursday, February 9, 2006.
**Last updated:** Thursday, February 9, 2006.
**Author**: Joel Kustka.

---

**KEYWORDS:** Multi-processor, Multithreading, Thread safety, OS X, RTAS

---

## SUMMARY:

A bug exists within the implementation of the pthreads library on OSX which allows ordinarily non-blocking threads to block. Severe performance problems can occur when most forms of OSX thread synchronization are used within the high-priority RTAS thread. ( i.e. from the RenderAudio/ProcessData callbacks ) This is because most OSX threading/synchronization layers are built off of pthreads. As a result, ***you needn't directly be using pthreads to encounter this problem.****

---

## SYMPTOMS:

Poor or spiky RTAS plug-in performance in Pro Tools 7.0, which is more pronounced at low hardware buffer sizes, or with multiple processors enabled for RTAS processing. This is caused by a priority inversion when the high-priority audio thread is blocked by a lower priority thread, causing an audio dropout and a subsequent DAE error message.

## PROBLEM:

The plug-in is calling an OSX API function that blocks the current thread from within their ProcessData/RenderAudio function. The API called may not be one that is expected to block, but which does so anyway. (Plug-ins should never call a function known to block from within ProcessData/RenderAudio). Apple has acknowledged a bug in the implementation of their "pthreads" library which can cause ordinarily non-blocking pthreads API calls to block. Since most other thread-synchronization APIs in OSX are built upon pthreads, the problem is quite pervasive. A plug-in may innocently be calling MPSignalSemaphore or pthread_cond_signal without realizing the severe impact this has on PT performance.

Spinlocks are a method of protecting access that can potentially block other threads. A spinlock function will attempt to grab the lock being sought for a certain number of tries (this is the "spinning"), and if the lock is or becomes available during this time, the function returns immediately. If the lock still isn't available after the allotted amount of time, the current thread is put to sleep. The scheduler is told to depress the thread's priority to zero for a certain period of time, allowing other threads to run - potentially including the thread which currently holds the spinlock. After the time limit expires, the process starts over again with more attempts to grab the lock. A single period of sleep can be as large as 1 ms, a very significant amount of time at low hardware buffer sizes.

Spinlocks are used quite frequently from user-mode code within OSX. One of their uses is to serialize access to the internal structures of pthreads synchronization objects (i.e. condition variables and mutexes). Each condition variable and mutex object has a spinlock which code must grab before any operations on the object are allowed. Unfortunately, this includes the operations of signaling a condition variable or unlocking a mutex, which one might not expect to allow the current thread to block. As OSX builds most of its threading/synchronization layers upon pthreads, a plug-in that calls MPSignalSemaphore or pthread_cond_signal from ProcessData/RenderAudio risks encountering priority inversion. If these calls are made from the high-priority RTAS thread, they could have a severe impact on PT performance. ***All user-level threads in OSX are based off of pthreads, but pthread synchronization objects should be avoided to help prevent priority inversion and poor plug-in performance.*** See reference "Apple TN2028" for more information on Apple threading architecture.

## SOLUTION:

We have found that the BSD semaphore APIs on OSX are immune to this problem. These APIs are not built from the pthread library and appear to protect the BSD semaphore objects with non-blocking synchronization methods. The APIs that are available are: sem_open, sem_close, sem_post, sem_wait, and sem_trywait. These functions allow code to synchronize execution between multiple threads with counted semaphores. These functions are documented in the man pages, and that same documentation is available by typing the function name at the Apple developer website, http://developer.apple.com. If you find yourself searching for examples of semaphore usage, bear in mind that they are based off of the FreeBSD semaphore APIs, which are very similar to the POSIX.4 versions. *As indicated in the man pages, sem_wait is a blocking call, and as such should not be called from ProcessData/RenderAudio.*

As an alternative to pthread synchronization, we strongly encourage RTAS plug-in developers to use the BSD sem_xxxx calls. Unlike pthread-based synchronization objects, they do not cause unstable RTAS performance, and are suitable for synchronizing execution of separate threads with the processing code of their ProcessData/RenderAudio function. BSD is only available as MachO function calls, developers will need to convert to MachO or use the appropriate transition mechanism if they are still building their plug-ins as CFM binaries. As noted on the Apple developer website, the CarbonLib SDK includes a number of samples that show how to create CFM-to-MachO glue code: http://developer.apple.com/sdk/

## SOURCES:

Apple TN2028: Threading Architectures - http://developer.apple.com/technotes/tn/tn2028.html#MacOSXThreading

Apple Multiprocessing Services Reference - http://developer.apple.com/documentation/Carbon/Reference/Multiprocessing_Services/index.html#//apple_ref/doc/uid/TP30000145

Apple Multithreading Programming Topics - http://developer.apple.com/documentation/Cocoa/Conceptual/Multithreading/index.html

BSD sem_open System Call: http://developer.apple.com/documentation/Darwin/Reference/ManPages/man2/sem_open.2.html

Apple Synchronization Primitives - http://developer.apple.com/documentation/Darwin/Conceptual/KernelProgramming/synchronization/chapter_15_section_2.html *

*An alternative to BSD semaphores is Mach semaphores. We didn't perform any testing with these primitives. They are based off of the Mach library and should serve as another viable option for thread synchronization.