# OSF* Real-Time Mach:

# Making the Mach Kernel Preemptible†

8 October 1993

Dan Swartzendruber

Open Software Foundation
Cambridge Research Institute
1 Cambridge Center
Cambridge, MA 02142 USA

## Introduction

As currently implemented, the Mach kernel is not preemptible. What this means is that if a higher priority thread is made runnable, either by the lower priority thread currently executing, or by an interrupt of some sort, the higher priority thread will not be given control of the CPU until the lower priority thread returns from the kernel to user mode. A naive approach would be to switch to the higher priority thread as soon as the lower priority thread has released its last spinlock. However, experiments with an instrumented Mach kernel indicate not only that spinlocks can be held to a nesting depth as great as 5 in some situations; but that due to certain algorithmic peculiarities, the total elapsed time during which one or more spinlocks is held can be arbitrarily long, making any sort of predictable scheduling problematic.

## Preemption methodology

In the current Mach kernel, when a thread is made runnable that has a higher priority than the thread currently executing, an AST (Asynchronous Software Trap) is set for that CPU. When the thread is preparing to return to user mode, this AST is noticed and the scheduler is entered to context switch to the new thread. The preemptible Mach kernel attempts to reduce the time between when the higher priority thread is made runnable and when it is actually scheduled. It does this by checking for a scheduling AST anytime the kernel is returning from an interrupt or trap, and the current thread does not hold any spinlocks. Latency is further improved by modifying the code which unlocks spinlocks to detect that a scheduling AST has been requested, and to invoke the Mach



†This research was supported in part by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Material Command (AFMC).

scheduler. Note that it is not desirable to preempt in all situations, as timesharing threads (as well as higher priority threads in the OSF/1 server) do not have any compelling need to reduce their scheduling latency. Currently only threads whose priority is in the hard real-time range are eligible for preemptive scheduling.

**Locking methodology**

Spinlocks in the Mach kernel fall into one of two categories: those that can be accessed from an interrupt handler, and therefore require *spl* protection, and those that are used only to guarantee mutual exclusion in an SMP scenario. Most spinlocks fall into the second category. If spinlocks in the second category are replaced by some type of blocking lock, the preemption rule can be changed to allow preemption while holding any number of such locks, since there is no risk of deadlock.

A new type of lock intermediate in complexity between reader/writer locks and simple spinlocks was implemented. Their runtime overhead is only marginally higher than that of simple spinlocks. This new lock type is semantically identical to a simple spinlock, with the exception that failure to acquire such a lock results in the thread blocking instead of spinning. These locks will be referred to as mutexes. When a mutex is unlocked, if there are any waiting threads, one of these threads will be made runnable.

**Performance implications**

In the simple uniprocessor environment, Mach spinlocks in the current Mach do not generate any code due to conditional compilation options. Obviously, the mutexes previously described need to be present regardless of the kernel build configuration, so it is expected that the preemptible kernel will suffer some degree of performance degradation. This was measured by running the AIM-III suite with 40 users on a 50-Mhz 486. This test was executed 5 times and the mean elapsed time was calculated. Using this methodology, the preemptible Mach kernel was found to be approximately 1.4% slower than an unmodified Mach kernel.

**Improvements in scheduling latency**

Scheduling latency was measured with a program that repeatedly sleeps using one of the new clocks/ timers interfaces, which accepts an absolute wall-clock time, and upon regaining control of the CPU, determines how long it took to do so. This program was executed simultaneously with the previously described AIM-III suite. In both cases, the great majority of events were in the 0-499 microsecond range. Most of the remaining events were in the 500-999 range, with a few blips in the 1000-1499 range. Most of the improvement due to preemption was in the 500-999 range, reducing the number of events by approximately 50%.

## Current Status

Although the basic Mach kernel is preemptible, there are still some scheduling anomalies. This represents the current state of the project rather than inherent problems. The following problems still exist:

- The i386/486 device drivers have not yet been modified to be preemptible. This code was written originally to run on uniprocessors only. The effort to convert these drivers is significant and has not yet been started. These drivers operate in a compatibility mode wherein the kernel will not preempt a thread that is in an unconverted device driver.
- The scheduler code is overly complex. This problem has not yet been fully analyzed.
- Some Mach subsystems hold spinlocks for an excessively long time. In order to reduce latency periods, certain subsystems will have to be modified to reduce the duration of the critical sections in their code.

## Future Work

The next phase of development will focus on reducing the latency inherent in the basic Mach kernel. More comprehensive analysis and profiling will be used to identify critical regions where spinlocks are held for excessively long periods. In addition, minor optimizations to the new locking code should provide some performance improvement.

## References

[Travos92] Travostino, F., <u>Mach 3 Locking Protocol,</u> OSF Research Institute Technical Report, Forthcoming.