Running Emulated Drivers

Wayne Meretsky

March 9, 1993

Summary

This document describes the problems of running emulated device drivers in the V1 software system. Also discussed are proposed solutions to some of those problems and the costs associated with those solutions.

Included is a brief description of hardware support that must be included in any native expansion slot card (PSI, PCI, etc). whose purpose is to allow NuBus cards to be plugged into the system. With this provision, a NuKernel based system should be able to remain compatible with all but the most bizzare device drivers and applications.

What software cannot be emulated?

In a NuKernel based system, software that is invoked as a result of a hardware interrupt, synchronously to the interruption, and at interrupt level, cannot be emulated. For the purpose of this discussion this classification of software is termed *interrupt-level software.*

Note that in System 7, there is a tremendous amount of interrupt-level software. Applications , for example, that make asynchronous I/O requests have their completion routines run synchronously to the hardware interrupt that signifies the completion of the I/O request. These applications, therefore, contain interrupt level software. In the V1 system, however, NuKernel Software Interrupts are used to get the execution of

completion routines out of interrupt level.

In a V1 system, interrupt-level software includes the following:

- ADB device drivers may be interrupt-level software. This depends upon the amount re-work done to the ADB manager. Techniques for removing ADB device drivers from interrupt level have been demonstrated by the NuKernel team.
- Certain system software managers such as the VBL Manager, Time Manager, SCSI Manager, etc. These managers all interact with some hardware at interrupt level. However, in a V1 system, these are all ported by the NuKernel team and execute only native instructions. Therefore, unless there is a need to support third party patches to these managers, they represent no problem. Because of changes to the hardware it is doubtful that these patches would work properly even if they were emulated.
- Some DRVRs (Serial, Sound, etc.) have hardware interrupt handlers. The TNT team in MSD will be providing native versions of these drivers for V1. Therefore, unless there is a need to support third party M68000 drivers for these devices, they represent no problem. Because of changes to the hardware in TNT it is doubtful that these third-party drivers will work even if they are emulated.

- Most NuBus cards have hardware interrupt handlers. These are the most troublesome because the drivers are contained in ROM on the NuBus cards and cannot be easily replaced. This document outlines a strategy that will be capable of supporting most, but not all, NuBus cards.
- Applications that have interrupt handlers. This includes certain strange applications that chose to avoid the Device Manager model for dealing with devices. These applications will not work in the V1 system.

What problems are associated with interrupt software?

Two major problems are associated with emulating interrupt-level software in V1: conforming to the NuKernel restrictions on interrupt level execution and conforming to the synchronization requirements of emulated software. Each is complex and they are not related.

NuKernel Restrictions on Interrupt Level Execution

Emulation of interruptions in the V1 system requires that either the V1 environment for interrupt level execution be adhered to by the emulator and the emulated software or that the interrupt level environment be altered. The V1 rules for interrupt level execution are the NuKernel rules for interrupt level execution and require that:

- No page faults occur
- Only limited kernel services are utilized

To comply with the no-page-fault rule would require that at least the entire emulator, the interrupt-level software that is being emulated (including everything it ever calls), and all state that is accessed be locked into physical memory. The only reliable way

to achieve this is to wire down the entire system heap as is done in System 7. This places a tremendous constraint upon the number of physical page frames available to the rest of the system. Many studies have shown that decreasing the number of physical page frames causes a non-linear increase in the page fault rate and a corresponding degradation in system performance. In particular, the ATG study entitled The Effects of Limiting Swappable Memory on System 7 Virtual Memory Performance by Eric Traut concludes that the impact in that system is cubic!

Going down a different path, one that allows page faults at interrupt time, is an interesting exercise. The problem with page faults at interrupt time is centered around the SCSI managers interrupt driven nature. Interrupts must be processed to make forward progress on page faults themselves. Cyclone deals with this problem by detecting the lack of forward progress and reverting to a polling scheme until the I/O for the page fault is complete. The Cyclone scheme cannot be made to work in a NuKernel system because the deadlock is not apparent. NuKernel has no I/O busy wait loops. Although it may be possible to detect the deadlock conditions by spreading various checks throughout the system, the resulting system is (as demonstrated by the Cyclone schedule) difficult to understand and maintain. Further, handling page faults at interrupt time requires use of many kernel services (sending messages, etc.) that are not presently callable at interrupt level. This provides a segue into that aspect of the problem.

The second area of execution environment impact has to do with the limited use of kernel services at interrupt level. Given the lack of compliance with even the limited guidelines suggested for M68000 interrupt-level software, it is doubtful that limiting the use of kernel services at interrupt level is possible when

emulating those drivers. Lets review the reasons for limiting the kernel services available at interrupt time.

One of the primary design goals of NuKernel is to reduce the interrupt latency of the system. This goal is motivated by the increasing amount of near real-time processing that is being done on Macintosh systems. Typical examples of this are QuickTime, WorldPort, MIDI, etc. For all of these examples, high interrupt latency causes rapid degradation in performance. A quick investigation into why System 7 has such high interrupt latency shows that OS, ToolBox, and application software are forced to disable interrupts to achieve synchronization with asynchronous processing that is done either directly or indirectly as the result of an interruption. In effect, the critical sections in all software on Macintosh are protected by disabling hardware interrupts.

NuKernel does not disable interrupts to guarantee synchronous access to its critical sections. Instead it uses the secondary interrupt handling concept to serialize such accesses. This means that kernel services that require synchronization cannot be called from hardware interrupt handlers but can be called from secondary interrupt handlers. If kernel services that require synchronization are to be callable from hardware interrupt handlers then the synchronization techniques used by the kernel will have to disable interrupts for the duration of the critical section. This is completely contrary to our goals of improving responsiveness by reducing interrupt latency.

Synchronization Concerns

There are two of these: macro instruction boundary synchronization and interrupt synchronization.

M68000 processors only allow interruptions at macro instruction boundaries. Much M68000 code, especially interrupt-level software, depend upon this level of implicit synchronization for correct operation. PowerPC processor architecture similarly

allows interrupts only at macro instruction boundaries. Unfortunately, when running the emulator, the PowerPC interrupts can arrive in the middle of a sequence of PowerPC instructions that emulate a single M68000 macro instruction. If the interrupt is fielded by an emulated handler then, from the perspective of the emulated code, the interrupt is mid-instruction. Handling interrupts mid-instructions will break M68000 drivers.

The V0 solution to this problem is based upon specialized interrupt hardware and emulator/nanokernel interactions that conspire to delay interruptions until macro instruction boundaries. All interruptions in that system are fielded by M68000 handlers.

In a NuKernel system, it is not currently possible to determine if the emulator is running. The emulator is just native code that executes like any other ToolBox/application code. The fact that the emulator's purpose is to emulate is not apparent. Similarly, the kernel is not aware that the file systems purpose is to read and write files. If the kernel is to become cognizant of the emulator, the kernel must provide interfaces that are invoked on each mode switch. This level of integration with the emulator is not desirable in a NuKernel based system. The impact on interrupt latency and the requirements placed on hardware implied by supporting macro-instruction synchronization are not in line with the NuKernel design goals.

The second area of synchronization concern is that of interrupt synchronization. Drivers frequently synchronize with their interrupt handlers by disabling interrupts. When the drivers are emulated, this means that the emulation of M68000 instructions that disable M68000 interrupts must also disable real hardware interrupts. In a NuKernel system we had planned upon not

disabling interrupts when an application disabled M68000 interrupts. Rather, the effect of disabling M68000 interrupts would be only to disable software interrupts to the current task. This would allow applications to synchronize with their asynchronous code (completion routines, VBL tasks, Time Manager Tasks, etc.).

This approach allows application compatibility to be maintained without sacrificing the kernel's goals for interrupt latency.

What interrupt-level software needs to be emulated?

In System 7, there are many kinds of software that participate in the device control portion of the system. Motherboard device drivers (Serial, Sound, Floppy, etc.), Slot Drivers, ADB Drivers, SCSI Drivers are just a few. Each of these typically contain interrupt-level software. If NuKernel is to be an integral part of V1 on TNT, exactly what drivers need to be emulated?

The system provided drivers (Floppy, Sound, Serial, Keyboard, Mouse, etc.) are already scheduled to be ported and run as native NuKernel drivers. Although some applications may bypass those system-provided drivers and instead use either their own code or have DRVR resources integral to the application, these won't work even if they are emulated because the TNT hardware is sufficiently different from previous systems.

ADB device drivers are far and few between. Most third party ADB devices (light pens, track balls, keyboards, etc.) simply use the system supplied keyboard and mouse drivers. Those that have their own drivers would break unless the a new, native, driver is provided or the old driver is emulated. These drivers do not seem to be a significant concern and it is hard to see how they could justify the impact of emulating driver code in a NuKernel based system. The ADB manager changes proposed by the NuKernel team would allow existing ADB device drivers to be run emulated but not at interrupt-level. These changes have been prototyped by the NuKernel team and are used in the existing M68000 NuKernel system.

SCSI device drivers never execute at interrupt-level in System 7 because the SCSI manager is not interrupt driver in that system.

The NuKernel SCSI manager is interrupt driven. However, the interface to that SCSI manager that provides compatibility with System 7 SCSI drivers does not cause those drivers to be executed at interrupt-level. Therefore, emulating SCSI drivers are not a problem in the V1 system. Of course, NuKernel requires that SCSI disk drivers conform to the new SCSI API and driver API if they are to be used as swapping devices by the NuKernel VM system. NuKernel will most likely include a "universal" disk driver that will be used if the device -resident SCSI driver for a hard disk is not a native NuKernel driver. If this universal driver does not support the disk then that disk cannot be used for paging.

This leaves only NuBus slot drivers. Neither PDM nor TNT have NuBus slots and it is unlikely that future PowerPC machines will include direct support for NuBus. NuBus becomes a problem if adapters are created that allow NuBus cards to be plugged into the expansion slots of PowerPC machines. The NuBus problem is compounded by the fact that most NuBus cards have M68000 drivers in the card's declaration ROM.

NuBus cards can be supported three different ways. Two of these are obvious: a new native driver can be required or the declaration driver in the ROM can be emulated. The former approach is the most desirable from a kernel perspective. These drivers would perform much more efficiently and would eliminate the problems of emulating the drivers.

A third approach to the NuBus dilemma is possible. If the hardware adapter that allows NuBus cards to be plugged into the expansion slots of PowerPC machines contains a small amount of extra hardware, then the entire slot device driver can be run within the context of a NuKernel task thereby removing the

requirement that parts of the driver be emulated at interrupt time. This approach removes the entire NuBus card driver's interrupt-level execution requirements in a compatible fashion.

The reason that extra hardware is required is a result of the interrupt structure of slot devices. In the NuBus world, the card's interrupt handler must be run in order to clear the interrupt request. If the request is not cleared, the interrupt remains pending. If the adapter hardware included the ability to mask the interrupt from the slot device, then there is no requirement to run the handler at interrupt time. The expansion slot interrupt handler could simply mask the NuBus card's interrupt request and then queue a software interrupt to a high priority task that would invoke the card's driver's interrupt handler. After the NuBus card's handler had been run, the expansion slot driver would unmask the interrupt source so that future interrupts could occur. This would be transparent to the slot device and its driver.

This approach could be made to work only if the increase in latency between slot interrupt request and slot interrupt handling implied by scheduling a task is acceptable. For many slot devices (frame buffers, Ethernet cards, etc.) it should be fine. For some esoteric, custom cards that perform data acquisition or MIDI processing it may introduce unacceptable latency. It could be argued that drivers for those types of cards would need to be rewritten in any case because the performance of emulated code is either not acceptable or highly undesirable.

Conclusions

The costs of emulating M68000 interrupt time code, particularly that found in device drivers, is extremely high. It impacts the overall performance of the system in both space and time. It also adds prohibitive interrupt latency restrictions to the entire system.

Because the majority of the device drivers in a given system are provided by Apple and will be native in PowerPC systems the

only justifications for emulation must be NuBus or oddball applications. Hopefully the oddball applications can be eliminated without much discussion. Most NuBus drivers can be dealt with in the fashion described. Those remaining would be broken and Apple will probably receive criticism for this failure to achieve 100% compatibility.

The NuKernel strategy provides an infinitely greater degree of support for third party hardware than Microsoft provides in their transition to NT or DEC provided in their transition to Alpha.

Given the limited number of drivers that could require emulated interrupt level execution it seems that a cost/benefit analysis can not possibly justify total compatibility with existing M68000 drivers in a NuKernel system. This is before any consideration is given to the fact that these drivers are not viable in a multi-address space or SMP system.

If NuKernel is to provide the basis for system software in the future, now is the time to present the few incompatibilities necessary to make that forward progress.