T E C H N O T E 1084

Running CFM-68K Code at Interrupt Time: Is Your Code at Risk?

CONTENTS

Under 680x0 systems running older versions of the CFM-68K Runtime Enabler, CFM-68K code that ran at interrupt time could cause a system crash. That problem has been fixed in the 4.0 release of CFM-68K. This document describes that problem and details the remaining limitations with CFM-68K code and interrupts. The reader is assumed to be familiar with the chapters on the Mixed Mode Manager and the Code Fragment Manager as documented in Inside Macintosh: PowerPC System Software. Most applications will not need modification. Users will simply install the new extension and reboot to take advantage of the fix. Threaded applications that don't use the Apple Thread Manager will need modification. Threaded application writers should read this document for instructions on how to modify their code so that their application is protected from this problem.

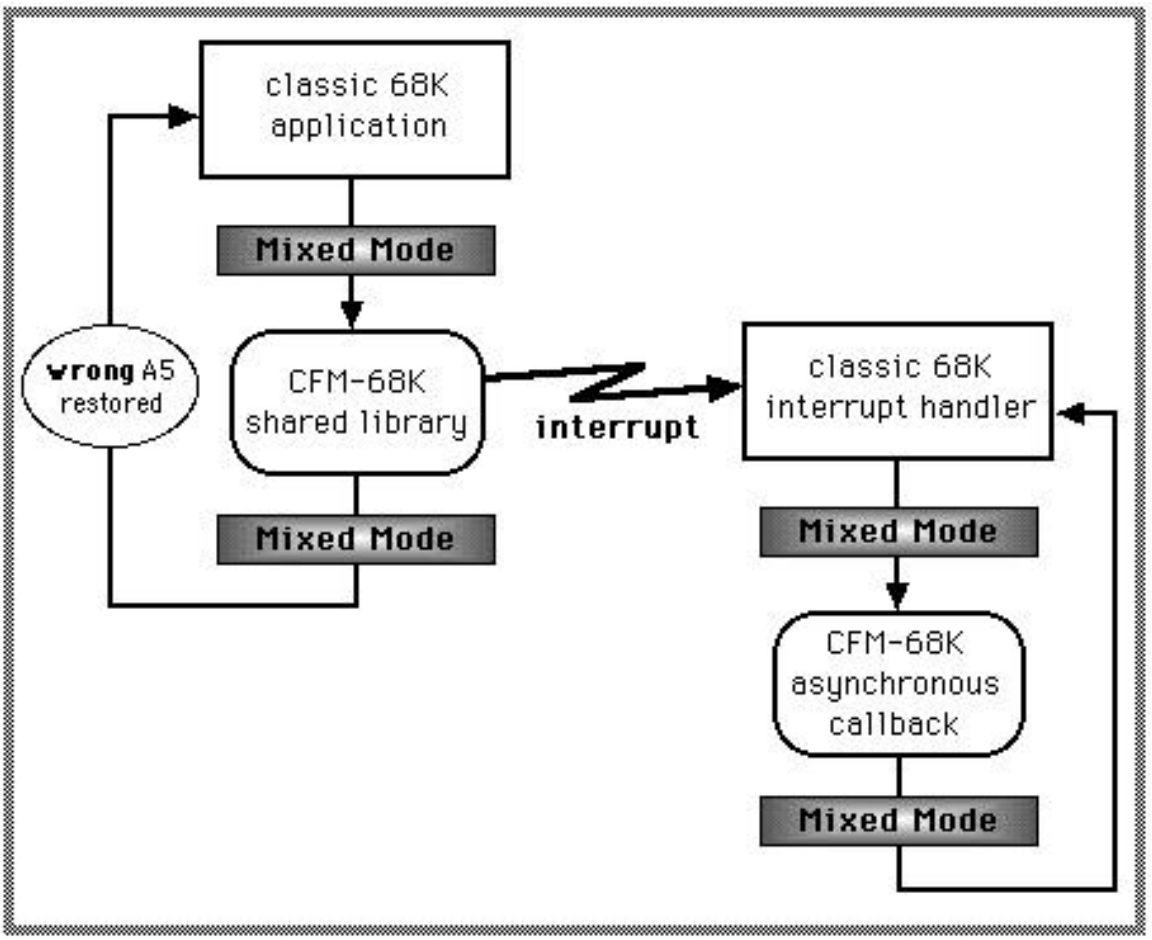
Defining the Interrupt Problem

Under systems running older versions of the CFM-68K Runtime Enabler (prior to version 4.0), CFM-68K code that executes at interrupt time could trigger a system crash. This limitation on CFM-68K code existed because on 680x0 machines, CFM-68K Mixed Mode is not invoked by interrupts. Mixed Mode must be invoked every time a transition is made between the "classic-68K" and the "CFM-68K" runtime worlds.

Mixed Mode plays the same role on the 68K as it does on the PowerPC. It serves as a runtime translation engine to manage transitions between the "classic" runtime world and the "native" (CFM) runtime world. This includes saving and restoring the A5 register on each transition. Any time a transition takes place between these two worlds, Mixed Mode must be invoked.

Interrupts are a special situation with respect to Mixed Mode transitions. On the PowerPC, there is a tight integration of the interrupt system and Mixed Mode by an emulator. Native PowerPC code may run at interrupt time because the emulator performs the necessary Mixed Mode transitions. There is no need for an emulator on 68K machines. As such, mode switches don't take place between CFM-68K code that gets interrupted and a classic 68K interrupt handler that later gets invoked. CFM-68K code that runs at interrupt time could cause the A5 register to become invalid for classic code that executes after the interrupt. All subsequent A5-relative references (e.g., global data references) also become invalid.

The following is an illustration of the sequence of events that could trigger this problem:



In this example, a classic 68K application calls a routine in a CFM-68K shared library. In order to do so, it must go through Mixed Mode. The CFM-68K code installs an asynchronous callback routine (also written in CFM-68K code). The CFM-68K code gets interrupted and the interrupt handler calls the user's callback routine.

The problem in this example is that a transition is made from CFM-68K to classic code (via the interrupt) without going through Mixed Mode. In certain situations, the wrong value of A5 could get propagated back to the classic 68K application, resulting in a crash.

The Solution

The 4.0 version of CFM-68K fixes the most common problems with CFM-68K code running at interrupt time. The behavior of Mixed Mode has been modified; it now saves additional information into the switch frame on each transition. However, some limitations still exist. Please read the Limitations section in this document for more information.

What Now Works

The 4.0 version of the CFM-68K Runtime Enabler will allow the following to be safely implemented (either wholly or partially) in CFM-68K code, all of which were previously unsupported and could cause a system crash.

1. I/O completion routines
2. AppleTalk callback routines
3. VBL tasks
4. Deferred tasks
5. Pre-emptively scheduled applications using Apple's Thread Manager

For each of these five examples, this includes cases where:

a. CFM-68K code is wrapped in a routine descriptor and passed to the system as one of the above interrupt time routines (e.g., as an I/O completion routine),

b. CFM-68K code is called from within an interrupt handler,

c. Classic 68K code that wraps a CFM-68K implementation and is called from an interrupt handler/routine.

Note:
Cooperatively-scheduled threaded applications have always been supported with CFM-68K.

What Won't Work

There are several limitations that were in the previous versions of CFM-68K and remain limitations in the 4.0 version of CFM-68K. These limitations are discussed in depth in the Limitations section in this document. In short, they include:

1. Interrupt-safe A-Traps may not be implemented in CFM-68K code.
2. Processor exception handlers may not be implemented in CFM-68K code.
3. CFM-68K code cannot be called from a pre-emptive thread whose thread scheduler has not been modified to call the new Mixed Mode API (discussed below).

Change Description

The 4.0 version of the CFM-68K Runtime Enabler provides a new version of Mixed Mode that allows most interrupt code to be implemented in CFM-68K code.

With the exception of applications that use a non-Apple pre-emptive thread scheduler, no applications will need modification. Customers will simply install a new CFM-68K Runtime Enabler in their Extensions folder to take advantage of the fix.

The CFM-68K Runtime Enabler contains CFM-68K, Mixed Mode and several shared libraries. With this new release, we've added the Apple Thread Manager and `NuThreadsLib.slb` to the enabler file. This enabler is supported on 68K machines running Mac OS 7.1 through 7.6.x with at least a 68020 processor.

Note:
Installing the CFM-68K Runtime Enabler on PowerPC systems won't break anything. The CFM-68K INIT simply won't load on a PowerPC machine.

Below is a brief summary of the changes to the enabler components included in this release.

CFM-68K

All of the bug fixes that were included in the PowerPC version of the Code Fragment Manager for the System 7.5.5 Update are included in this release. Refer to Technote 1069 for details of the changes to CFM.

Mixed Mode

Mixed Mode now saves additional state information on the switch frame for a mode transition. Two new API routines have been added for pre-emptive thread switchers to save and restore Mixed Mode state. These API are only callable from classic 68K clients. There will not be a CFM-68K interface to these routines. The new Mixed Mode and Gestalt headers that include these changes will be available on E.T.O. #23.

The two new API's are `SaveMixedModeState` and `RestoreMixedModeState`. Whenever the thread switcher saves and restores machine state for a thread switch, it needs to call these API's to save the Mixed Mode state and restore it when it switches in the new thread. `RestoreMixedModeState` must be called with the same `MixedModeStateRecord` returned by `SaveMixedModeState`. All thread switchers should be modified to prevent the A5 crash from ever occurring. When CFM-68K code is invoked from a pre-emptive thread and the switcher hasn't been modified to call these new API's, the A5 crash could occur.

SaveMixedModeState

Saves Mixed Mode state information in a buffer allocated by the caller.

```
OSErr SaveMixedModeState ( MixedModeStateRecord *stateStorage,
                           UInt32 stateVersion);
```

stateStorage

On exit, the MixedModeStateRecord buffer has been initialized by Mixed Mode. This buffer needs to be saved for a subsequent call to RestoreMixedModeState. The buffer contents are reserved by Mixed Mode and should not be changed or interpreted in any way by the caller as the contents may change in future versions of Mixed Mode.

The MixedModeStateRecord is defined as follows:

```
struct MixedModeStateRecord {
        UInt32  state1;
        UInt32  state2;
        UInt32  state3;
        UInt32  state4;
};
typedef struct MixedModeStateRecord MixedModeStateRecord;
```

stateVersion

The version of the MixedModeStateRecord. You should use the constant `kCurrentMixedModeStateRecord`, currently defined as 1:

```
enum {
        kCurrentMixedModeStateRecord = 1
};
```

Result Codes
`noErr 0` No Error

RestoreMixedModeState

Restores Mixed Mode state information.

```
OSErr RestoreMixedModeState ( MixedModeStateRecord *stateStorage,
                              UInt32 stateVersion);
```

stateStorage

A pointer to a `MixedModeStateRecord` buffer that was originally returned by a call to `SaveMixedModeState`.

stateVersion

The version of the `MixedModeStateRecord`. You should use the constant `kCurrentMixedModeStateRecord`.

Result Codes
`noErr 0` No Error

To check the availability of these routines, use the new Mixed Mode gestalt attribute `gestaltMixedModeCFM68KHasState`. This bit is defined in a new version of the Gestalt header available on E.T.O. #23. The Mixed Mode attribute bit settings are now defined as:

```
enum {
        gestaltMixedModeAttr    =       'mixd',
        gestaltMixedModePowerPC =       0,
        gestaltPowerPCAware     =       0,
        gestaltMixedModeCFM68K  =       1,
        gestaltMixedModeCFM68KHasTrap   =       2,
        gestaltMixedModeCFM68KHasState  =       3
};
```

You can also use the `gestaltMixedModeCFM68KHasState` attribute to determine if the user's system supports this newer version of Mixed Mode.

Apple Thread Manager

The Apple Thread Manager has been added to the 4.0 release of the CFM-68K Runtime Enabler. This version overrides any version of the Apple Thread Manager available before Mac OS 8. It calls the new Mixed Mode API to save and restore Mixed Mode state around a pre-emptive thread switch.

`NuThreadsLib.slb`

Version 2.1.1 of the CFM-68K ThreadsLib shared library is now included in the CFM-68K Runtime Enabler.

StdCLib

StdCLib and IntEnv version 3.4 are included in this release.

Pre-emptive Thread Switchers

Unmodified non-Apple pre-emptive thread switchers could still trigger the A5 corruption problem if any CFM-68K code is called from a pre-emptive thread. As such, we have called out and distinguished the term "pre-emptive safe". Without CFM-68K, pre-emptive safe is synonymous with interrupt-safe. Interrupt-safe is a familiar concept to many Apple developers and interrupt task guidelines are documented in Inside Macintosh: Processes. Routines that are interrupt-safe are callable at interrupt time and therefore are callable from pre-emptive threads since these threads are triggered by interrupts.

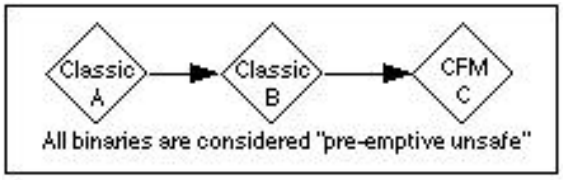
For CFM-68K code, pre-emptive safe and interrupt-safe are no longer synonymous. All CFM-68K code is inherently pre-emptive unsafe because it could be invoked by an unmodified thread switcher which would trigger the A5 bug. The new definition for the term pre-emptive safe is any software that doesn't contain or call any CFM-68K code and is otherwise deemed interrupt-safe. Conversely, code that is otherwise deemed interrupt-safe, but does contain or call CFM-68K code is not pre-emptive safe code, a.k.a. pre-emptive unsafe code.

Note:
Pre-emptive unsafe code can be called by an application that uses Apple's latest Thread Manager or a modified non-Apple thread scheduler. Therefore, "pre-emptive unsafe" could be qualified as "pre-emptive at risk".

Developers of new code who declare their code as interrupt-safe must document the pre-emptive safeness of their code. Code that is not interrupt-safe is also not pre-emptive safe; therefore, it is unnecessary to declare code as pre-emptive unsafe if it is already interrupt-unsafe. This is only necessary for code, such as plug-ins, that are provided to other developers and which could potentially be called from a pre-emptive thread. End users need not know about "pre-emptive safety".

Since all CFM-68K code is inherently pre-emptive unsafe, then so is any classic service that internally uses CFM-68K code or internally uses some other classic code that itself is pre-emptive unsafe code.

The following diagram is an example of pre-emptive unsafe code. All CFM-68K code is inherently pre-emptive unsafe, as defined above. In addition, classic code which in turn invokes CFM-68K code is itself pre-emptive unsafe, even if otherwise considered interrupt-safe.



This new version of CFM-68K provides a means for pre-emptively threaded applications to call any interrupt-safe routine, even those that are otherwise deemed pre-emptive unsafe. A new Mixed Mode API is provided to save and restore the Mixed Mode state. Pre-emptive applications should be modified to call these services when switching threads to properly maintain the Mixed Mode state for each thread. Therefore, pre-emptively threaded applications should be separated into two categories: those that can and those that cannot call pre-emptive unsafe routines.

Applications that can call pre-emptive unsafe code include:

1. Applications using the newer version of the Apple Thread Manager that has been modified to call the new Mixed Mode API.
2. Applications using a non-Apple thread scheduler that has been modified to call the new Mixed Mode API.

Applications that cannot call pre-emptive unsafe code include:

1. Applications using an older version of the Apple Thread Manager that hasn't been modified to call the new Mixed Mode API.
2. Applications using a non-Apple thread scheduler that has not been modified to call the new Mixed Mode API.

Limitations

The 68K Macintosh operating system lacks the emulator present on the PowerPC. As a consequence, CFM-68K is subject to limitations which don't exist in the PowerPC version of CFM. This release of CFM-68K removes most of the limitations which existed in the previous version of CFM-68K. However, there are some remaining limitations which cannot be eliminated without a major CFM-68K architectural change or the implementation of a nanokernel on 680x0 machines.

The following limitations remain in the 4.0 version of CFM-68K and represent a difference between the 68K and PowerPC. Therefore, if you are porting PowerPC code to CFM-68K code, you should take the necessary precautions to ensure that your code obeys these restrictions.

1. No interrupt safe ATraps, nor any other services currently defined as interrupt safe, can be implemented in CFM-68K code, since when they were defined, interrupt safe also meant pre-emptive safe.

2. CFM-68K shared library developers whose code is interrupt-safe must document their code as being pre-emptive unsafe.

3. Classic 68K code providers (object files, stand-alone code resources, etc.) who declare their code as interrupt-safe and invoke any CFM-68K code must declare their code as being pre-emptive unsafe.

4. Non-Apple pre-emptive thread switching:

    - must be modified to use the Apple Thread Manager, or
    - must be modified to use the new Mixed Mode API routines to save and restore Mixed Mode state at the same point that machine state (e.g., the register set) is saved and restored for the thread switch., or
    - can only support threads that are pre-emptive safe.

5. No processor exception handlers can be implemented in CFM-68K code. Examples include bus error handlers, illegal instruction handlers, divide by zero handlers, etc.

Incompatibilities

This version of CFM-68K is known to be incompatible with early versions of AOL, Cyberdog and The Debugger.

AOL 3.0

Do not use 68K versions of AOL 3.0 prior to the Preview 6 release. Early beta releases contain crashing bugs unrelated to CFM-68K.

Cyberdog

Do not use Cyberdog versions 1.2, 1.2.1 or 2.0 Alpha with this extension. There is an incompatibility between these versions of Cyberdog and the 4.0 version of the CFM-68K Runtime Enabler. This problem is fixed in the 2.0 Beta version of Cyberdog.

The Debugger

Early versions of The Debugger will cause CFM-68K to crash when an application quits. Be sure to obtain an update dated after March 3, 1997.

Downloadables

Acrobat version of this Note (K)

Download the New Version of CFM-68K

- http://www.macos.apple.com/macos/cfm-68k.html
- Classic to CFM Project (324K)

Further References

- Technote 1077 - Calling CFM Code From Classic 68K Code, or There and Back Again, A Mixed Mode Magic Adventure

To contact us, please use the Contact Us page.
Updated: 25-March-97