Technote 1063

Inside Macintosh: Processes, Time Manager Addenda

By Eric Simenel
Revised by Quinn "The Eskimo!"
Apple Worldwide Developer Technical Support

CONTENTS

This technote discusses a number of Time Manager issues that are not covered in the Time Manager chapter of *Inside Macintosh: Processes*.

This Note is intended for all developers who want to do time measurement using the Time Manager routines.

Some Basic Time Manager Rules

When programming with the Time Manager, it is important to observe the following rules.

- For each Time Manager task that you insert (using `InsTime` or `InsXTime`), you must remove the same Time Manager task (using `RmvTime`) once and only once.
- While your Time Manager task is inserted (that is, between `InsTime` and `RmvTime`), you can prime the task multiple times. For example, the sequence `InsTime`, `PrimeTime`, fire, `PrimeTime`, fire, `PrimeTime`, fire, `RmvTime` is perfectly legal. Some developers always insert, prime, and remove their Time Manager tasks. While legal, this is unnecessarily inefficient.
- If you remove a Time Manager task (using `RmvTime`), you must insert it again (using `InsTime` or `InsXTime`) before priming it.
- If you have primed a Time Manager task, you must not prime it again until it fires. This is explicitly called out in Inside Macintosh: Processes, page 3-11. If you need to cancel a Time Manager task, simply remove it using `RmvTime`. If you need to reschedule a Time Manager task, remove it, then reinsert it, then prime it again.
- Don't use `InsXTime` unless you want drift-free timing. A common mistake is to install a drift-free Time Manager task (using `InsXTime`), prime it for 1 second, let it fire, and then, 5 minutes later, prime it again for another 1 second. The task fires immediately because it was installed as a drift-free task. To avoid this behavior, simply use the original `InsTime` call.

This rules are not new; they are all explicitly or implicitly described in Inside Macintosh: Processes. However, recent systems now enforce these rules more strictly. If you break these rules, you may see one of the following symptoms.

- The system crashes with a `dsVMDeferredFuncTableFull` (112) system error.
- The system freezes because low-memory is trashed (pre-System 7.5.5).
- The system bus errors because of a fatal page fault.
- `PrimeTime` returns `qErr` (-1) when you attempt to prime a task that isn't installed.
- Timer Manager tasks not firing at the right time.

Note:
"Timer.h" defines the various Time Managers routines as void functions, which prevents you getting the error result from `PrimeTime` [2389936]. This is a shame because the assembly language definition of this routine has always included an error result in register `D0`. If necessary, you can access this error result using custom Mixed Mode Manager glue.

Back to top

Setting Up tmReserved

On page 3-8 of *Inside Macintosh: Processes*, it clearly states that both `tmWakeUp` and `tmReserved` should be set to 0 prior to the first call to `InsXTime` when using the extended Time Manager:

```
theTMTask.tmWakeUp = 0;
theTMTask.tmReserved = 0;
InsXTime((QElemPtr)&theTMTask);
PrimeTime((QElemPtr)&theTMTask, 2000);
```

If you do want to do some time measurement, then you have to call `RmvTime` to get the current value of `tmCount`, which leads later to a new call to `InsXTime`, and a call to `PrimeTime` with a 0 delay which has a special meaning in that case. Although it appears, after much reading, rather clear that you leave the current value of `tmWakeUp` untouched in the `TMTask` structure, you can't be sure what to do about the value of `tmReserved`.

The truth is that prior to October, 1992 (System Software 7.1), you didn't care, but it's more of a concern now, since Apple slightly modified the behavior of the Time Manager to deal with performance issues.

If you leave `tmReserved` untouched, then, after 127 calls to the following code:

```
RmvTime((QElemPtr)&theTMTask);
remaining = theTMTask.tmCount;
InsXTime((QElemPtr)&theTMTask);
PrimeTime((QElemPtr)&theTMTask, 0);
```

for some good but can't-be-disclosed reason, your extended time task is converted into a non-extended time task which, being waked up with a 0 delay `PrimeTime` (which has no special meaning for a non-extended time task), will suddenly be called and called again--more frequently than it should be.

So, if you perform that kind of time measurement, be sure to write instead:

```
RmvTime((QElemPtr)&theTMTask);
remaining = theTMTask.tmCount;
theTMTask.tmReserved = 0;
InsXTime((QElemPtr)&theTMTask);
PrimeTime((QElemPtr)&theTMTask, 0);
```

Since the Time Manager, prior to System Software 7.1, doesn't care about `tmReserved`, then you can set `tmReserved` to 0 before each call to `InsXTime` without checking the system version. You still have, of course, to ensure that the Time Manager you're using is the extended one (the response to `gestaltTimeMgrVersion` is `gestaltExtendedTimeMgr` (3) or greater).

Back to top

About tmWakeUp

The following sentence, also on page 3-8 in Inside Macintosh: Processes, is incorrect: "The `tmWakeUp` field contains the time at which the Time Manager task specified by `tmAddr` was last executed (or 0 if it has not yet been executed)." It should say: "The `tmWakeUp` field contains the time at which the Time Manager task specified by `tmAddr` is scheduled to be executed (or 0 if it has not yet been executed)."

WARNING:
Since the format of that field is undocumented and used internally by the Time Manager, developers are strongly discouraged anyway from performing any kind of calculation or comparison on the value of this field, since that format could change in the future.

Back to top

Undeferred Time Manager Tasks

This section describes an optimization that you might want to employ when using the Time Manager in the presence of virtual memory (VM). Most developers will not be interested in this; however, all users of the Time Manager should heed the following warning.

WARNING:
Because there is an **extremely remote** possibility that the memory you have allocated for your Time Manager task contains the special value listed below, if you want to ensure the behavior defined in **Inside Macintosh: Memory**, you should always clear the `qLink` field in the `TMTask` before installing it.

As described in **Inside Macintosh: Memory**, Time Manager tasks are automatically deferred by the Virtual Memory (VM) system to avoid fatal page faults. This was done for backward compatibility with existing applications that use the Time Manager, but it can seriously increase the latency between when the timer expires and when your Time Manager task executes.

For more information about interactions between the Time Manager and VM, see Technote 1094, "Virtual Memory Application Compatibility."

For example, if you schedule a Time Manager task to execute at time X and, at time (X - delta) some process takes a page fault, your Time Manager task will not be called until time (X + Y - delta), where Y is the time required to field a page fault. If the page fault causes the hard disk to seek, Y could be as great as the hard disk's average seek time, approximately 10 ms. If you are trying to use the Time Manager to measure time in **microseconds**, this could be a problem.

There is a way you can install Time Manager tasks so the callback is not deferred by VM; however, before using this technique, you should be aware of its dangers. Because VM does not defer these special Time Manager tasks, it is possible for them to fire when paging is not safe. To avoid fatal page faults, you must ensure:

- The `TMTask` record is held for the entire time the Time Manager task is installed. You can do that using the following code:

```
HoldMemory(&theTask, sizeof(TMTask));
```

- The code for the timer task and any data it references is held. If the code for your timer task is stored in a code resource, you can use the following snippet to make sure it is held. If your timer task code is not in a code resource, it's very difficult to ensure that it and its data are held.

```
// Ensure the code doesn't move in logical memory
HLock(ttaskCodeHandle);
// Ensure the code is held in physical memory and cannot be paged to disk
HoldMemory(*ttaskCodeHandle, GetResourceSizeOnDisk(ttaskCodeHandle));
```

- You timer task code only calls system routines that are guaranteed to meet the above requirement--this typically means only that routines that are known to be interrupt-safe.

WARNING:
If you fail to meet these requirements, you will cause a fatal page fault and crash the system.

If you call `InsTime` or `InsXTime` with the `qLink` field set to `$65616461`, the VM patch on the Time Manager will recognize your special requirements and execute your timer task as soon as it fires, regardless of whether paging is safe or not.

Back to top

The Microseconds Alternative

Another way to perform time measurement would be to use the `Microseconds` call, which is much easier to use and less likely to change in the future.

```
pascal void Microseconds(UnsignedWide *microseconds);
```

IMPORTANT:
Currently, even with the most recent system software (7.5.3 revision 2) on a PCI Power Macintosh, both the Time Manager calls and the `Microseconds` call are still in 68K code and thus are executed by the emulator. If you call them from PowerPC code, you'll get a switch from PowerPC code to the 68K emulator, so the values returned are incorrect by a few tenths of a microsecond. This means that you have to be careful when using either of them to do time measurement. If you do use `Microseconds`, then your time measurement is done by the difference of the 2 values returned by `Microseconds` before and after the code you measure, and since the latency induced by the switches is the same in both case, then your time measurement is correct. If you do use the Time Manager way of performing time measurement, however, the `tmCount` field may be off by a few tenths of a microsecond.

Back to top

Summary

The following points explain what you should and should not do in working with the Time Manager:

- Always follow the basic rules.
- Always set `tmReserved` to 0 before calling `InsXTime`.
- Set `tmWakeUp` to 0 before the first call to `InsXTime`, never look at it or modify it (except to set it to 0 in some cases, no other value is acceptable) afterwards.
- `tmCount` is only valid after a call to `RmvTime`.
- Always clear `qLink` before calling `InsXTime`.
- `Microseconds` might be a good alternate way.

Further References

- *Inside Macintosh: Processes*, Chapter 3, The Time Manager
- Denis G. Pelli's Web page

Back to top

Change History

- Originally written by Eric Simenel (Developer Technical Support).
- Updated by Quinn "The Eskimo!" (Apple Developer Technical Support) in March 1997 to include information about undeferred Time Manager tasks.
- Updated by Quinn "The Eskimo!" (Worldwide Developer Technical Support) in October 1999 to reiterate some basic Time Manager rules. Also made some minor clarifications and cosmetic changes.

Downloadables

 Acrobat version of this Note (49K).

Acknowledgments

Thanks to Brian Bechtel, Bob Bradley, Drew Colace, Jim Luther, Denis G. Pelli, Geoff Stahl, and Bob Wambaugh.

To contact us, please use the Contact Us page.
Updated: 11-October-1999

Technotes | Contents
Previous Technote | Next Technote