New Technical Notes

Macintosh

Developer Support

MultiFinder and _SetGrowZone

Memory M.ME.MultiFinderandSetGrowZone

Written by: Andrew Shebanow June 1989

MultiFinder patches the `_SetGrowZone` trap, and this patch can cause your program to crash if you attempt to save and restore the grow zone procedure.

MultiFinder gives each application its own heap in which to run. Because it wants to do some fairly tricky memory management, MultiFinder installs its own grow zone procedure (`gzProc`) in the application heap, and patches `_SetGrowZone` to store your application's `gzProc` in a temporary variable inside of itself.

A problem arises when you want to allocate some memory **without** invoking the application's `gzProc`. This can be useful if you are writing a library of routines that does its own internal caching, and you do not want that cache to purge the application's reserved memory. Let's say that to do this, you write a pair of routines, `KillGZProc` and `RestoreGZProc`, which look like this:

```
#include <Memory.h>

GrowZoneProcPtr savedGZProc;

pascal void KillGZProc(void)
{
       THZ myZone;

       myZone = GetZone();
       /*     since there is no GetGrowZone trap, we have to pull it directly
              from the zone header (Ugh! Very gross!) */
       savedGZProc = myZone->gzProc;
       /* we don't want a grow zone proc */
       SetGrowZone( (GrowZoneProcPtr) nil);
}

pascal void RestoreGZProc(void)
{
       /* set to saved value */
       SetGrowZone(savedGZProc);
}
```

Now let's say that you bracket your call to `_NewHandle` with these two routines. When MultiFinder is active, you get the following:

- When the application starts, you set your `gzProc` to the routine `MyGZProc`. MultiFinder stores the procedure pointer inside of your application's MultiFinder data area.

- You call `KillGZProc`. The global variable `savedGZProc` now contains a pointer to **MultiFinder's** `gzProc`, which MultiFinder installed in your zone header before your application started.
- You do your memory allocation, and your `gzProc` (`MyGZProc`) doesn't get called, just as you intended.
- You call `RestoreGZProc`, which stores a pointer to MultiFinder's `gzProc` in your application's MultiFinder data area.
- The next time you do a memory allocation that causes the `gzProc` to be called, MultiFinder's `gzProc` will be called. One of the things this `gzProc` does is to see if there is a valid `gzProc` stored in your application's MultiFinder data area. If there is a valid `gzProc`, it gets called. But the `gzProc` in your application's MultiFinder data area is MultiFinder's `gzProc`, so we go into an infinite loop. Oops…

The only solution to work around this problem is to avoid reading the value of the `gzProc` out of the zone header, since it isn't valid when MultiFinder is active. (Reading the fields of the zone header is dangerous, compatibility wise as well.) Your application should only have one grow zone procedure, so you should change your `KillGZProc` and `RestoreGZProc` to restore your application's grow zone procedure directly. The corrected code would look like the following:

```
#include <Memory.h>

pascal long MyGrowZone(Size cbNeeded);

pascal void KillGZProc(void)
{
       /* we don't want a grow zone proc */
       SetGrowZone( (GrowZoneProcPtr) nil);
}

pascal void RestoreGZProc(void)
{
       /* set to my routine */
       SetGrowZone(MyGrowZone);
}
```

As you can see, the code is simpler, though not quite as flexible, but at least it won't throw your machine for a loop.

Further Reference:

- *Inside Macintosh*, Volume II, Memory Manager
- *Programmers Guide To MultiFinder* (APDA)
- Technical Note M.TB.Multifinder — MultiFinder Questions
- Technical Note M.OV.Multifinder — MultiFinder Revisited
- Technical Note M.OV.32BitClean — The Joy Of Being 32-Bit Clean