KWN and ESK's Puzzle Page: Mutation Malarkey

This thread has been locked. Questions are automatically locked after two months of inactivity, or sooner if deemed necessary by a moderator.

2.6k

This is my love letter to the KON & BAL's Puzzle Pages of yore. If you weren't around to experience *develop* magazine — and let's face it, that's most of you! — a quick 'net search should yield the necessary backstory.

And no, I haven't included scores. That was *never* the point.

Share and Enjoy
—
Quinn "The Eskimo!"
Apple Developer Relations, Developer Technical Support, Core OS/Hardware
`let myEmail = "eskimo" + "1" + "@apple.com"`

KWN & ESK's Puzzle Page: Mutation Malarkey

See if you can solve this programming puzzle, presented in the form of a dialogue between Quinn (KWN) and Eskimo (ESK). The dialog gives clues to help you. Keep guessing until you're done; ~~your score is the number to the left of the clue that gave you the correct answer~~. Even if you never run into the particular problems being solved here, you'll learn some valuable debugging techniques that will help you solve your own programming conundrums. And you'll also learn interesting Macintosh trivia.

KWN I'm helping a developer with a really tricky bug.

ESK So, business as usual then. What's happening?

KWN They have a preferences pane (`.prefPane`) that won't load on 10.15. When you open it in System Preferences, you get this alert:

```
Preferences Error

Could not load MyPrefsPane preference pane.

                                    [[OK]]
```

ESK Dude, that's so not OK!

KWN Yeah, but we're supposed to debug this, not make snarky comments.

ESK Right, focus! So, what does the crash report say?

KWN There is no crash report.

ESK Tricky. What about the system log?

KWN That's more helpful. On triggering the problem I see an entry like this:

```
type: default
time: 2020-01-29 12:09:30.066173 +0000
process: legacyLoader
subsystem: –
category: <Unknown>
message: Preference pane loadMainView exception:*** Collection <__NSDictionaryM: 0x60000176d1e0> was mutated while being enumerated.
```

ESK That looks like an unhandled language exception. Someone is enumerating a dictionary using NSFastEnumeration, and the dictionary was mutated during that process.

KWN Right. But why didn't this trigger a crash report?

ESK Ah, I think I know that one. This is on the Mac, right? And this is about a pref pane, meaning it's UI code, so the code is probably based on AppKit. AppKit has a default exception handler that catches all language exceptions raised by code run from its run loop. It logs the exception and then swallows it, allowing the app to continue running.

KWN Who thought that was a good idea!?!

ESK What were you telling me about snarky comments?

KWN Yeah, right, focus. So, the next step is to get a backtrace to see where this dictionary is being used. But this is a pref pane, so we can't just run it from Xcode. We have to debug the system process that's loading the pref pane. How can we do that?

ESK Well, the log entry says the message is coming from a process called `legacyLoader`. Try attaching to that with LLDB.

KWN That fails due to System Integrity Protection (SIP):

```
% lldb
(lldb) process attach -n legacyLoader
error: attach failed: cannot attach to process due to System Integrity Protection
```

ESK What happens if you apply the universal 'try harder' modifier, `sudo`?

KWN Same thing.

ESK OK, it looks like you'll actually have to disable SIP. There's instructions for that in the System Integrity Protection Guide. You're on a 'victim' machine, right? Disabling SIP on a production machine is a bad idea.

KWN Dude, I'm working in a VM [virtual machine]!

ESK Smart. So, what do you see if you disable SIP?

KWN I can attach. Yay! But I still can't get a backtrace. Here's what I see:

1. I run System Preferences. Before I click on MyPrefsPane, I can't attach because `legacyLoader` isn't running.
2. I then click on MyPrefsPane. This causes `legacyLoader` to run but I can't catch the problem in action because it happens before I can attach.
3. Once I'm attached, I tried clicking on MyPrefsPane again. This doesn't reproduce the problem. It seems like System Preferences caches the fact that this prefs pane is 'dead'.

ESK Hmmm. `legacyLoader` is actually an XPC Service, right?

KWN Yep. It's embedded inside the PreferencePanes (private) framework.

ESK You might be able to use `launchctl attach` to attach with the debugger as the service starts. Check out the `launchctl` man page for the details.

KWN Dude, that's some scary `launchd` force lightning. Can you think of an easier way?

ESK Come to think of it, I can! Try this:

1. Use Xcode to create your own trivial pref pane from the macOS > Preferences Pane template.
2. Install that on your victim machine.
3. Start it. This will get the `legacyLoader` XPC Service running.
4. Attach to it with LLDB.
5. Then click on MyPrefsPane. It loads in the same instance of `legacyLoader`, so you can debug its startup sequence.

KWN Sneaky! And hey, it worked. So how do I trap this exception?

ESK Set an LLDB exception breakpoint like so:

```
(lldb) br set -E objc
```

KWN Cool beans! I now have a backtrace for the code that threw the exception:

```
(lldb) bt
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
  * … #0:  … libobjc.A.dylib`objc_exception_throw
    … #1:  … CoreFoundation`__NSFastEnumerationMutationHandler + 159
    … #2:  … MyPrefsPane`-[MPPDefaultsController initWithFile:]… at MPPDefaultsController.m:47:4
    … #3:  … MyPrefsPane`-[MyPrefsPane initWithBundle:]… at MyPrefsPanePref.m:33:14
    … #4:  … PreferencePanes`-[NSPrefRemoteViewService loadView] + 1076
    … #5:  … AppKit`-[NSViewController _loadViewIfRequired] + 72
    … #6:  … AppKit`-[NSViewController view] + 23
    …
    … #48: … PreferencePanes`PreferencePaneMain + 179
    … #49: … legacyLoader`main + 40
    … #50: … libdyld.dylib`start + 1
    … #51: … libdyld.dylib`start + 1
```

ESK 51 frames! That's quite a backtrace. Still, the exception seems to be coming directly from the developer's code. Check out frame 2, which is the immediate cause of the exception. What does that code look like? Specifically, line 47 of `MPPDefaultsController.m`?

KWN Fortunately, the developer sent me the source code for a test project that reproduces the problem, so that's easy to answer. `MPPDefaultsController` is a model-level class that manages preferences. The *Defaults* in the name is like the *Defaults* in `NSUserDefaults`.

It looks like this:

```
14 - (id)initWithFile:(NSString *)file
15 {
16   if (self = [super init])
17   {
18     _file = [file retain];
…
36       _values = [[NSMutableDictionary dictionaryWithContentsOfFile:_file] retain];
…
43
44     // Add ourself as an observer for each value in the dictionary
45     if (_values != nil)
46     {
47       for (id key in _values)
48       {
49         [self addObserver:self
50              forKeyPath:[NSString stringWithFormat:@"values.%@", key]
51               options:NSKeyValueObservingOptionNew
52               context:nil];
53       }
54     }
55   }
56
57     return self;
58 }
```

Here `_file` and `_values` are both ivars [instance variables]. The code sets up `_file` based on the incoming path. It reads the dictionary at that path and sets up key-value observing [KVO] on each of the keys.

ESK Hey hey, manual retain and release [MRR]. Kickin' it old school!

KWN Yeah, but there's nothing intrinsically wrong with MRR, right?

ESK Right!

It's hard to see how `_values` could have been mutated by that `for` loop. The code inside the `for` loop never modifies the dictionary, it only ever reads it. Any chance that multiple threads are in play?

KWN Unlikely. I did a backtrace of all the threads like so:

```
(lldb) bt all
…
```

This list is remarkably short, and none of the threads seem to be related to this code.

ESK Hmmm, perhaps there's some other environmental thing in play. If you extract that code to a standalone test app, does it still crash?

KWN Yes! I'm kinda surprised by that. I would've given good odds on this being something specific to the pref pane environment, but it reproduces in a vanilla Mac app.

ESK And that, young padawan, is why debugging is both an art and a science. Always test your assumptions!

KWN OK, so somehow the KVO is triggering the `for` loops mutation check. Isn't there some tricky way of causing `-addObserver:forKeyPath:options:context:` to run the observer once when you register the observation? Maybe the observer is mutating the dictionary?

ESK That'd be `NSKeyValueObservingOptionInitial`, and it's not in play here. But if you want to rule out stuff like that, start stripping down the class.

KWN Good idea. I removed *all* of the code from the `MPPDefaultsController` class, except this init method, and it still crashes. If I remove the `-addObserver:forKeyPath:options:context:` call inside the loop, the crash goes away. It's definitely that KVO that's the problem.

Also, I can avoid the problem entirely by making a snapshot of the dictionary's keys. Changing line 47 to this:

```
47       for (id key in _values.allKeys)
```

fixes the crash in both the standalone test app and the developer's pref pane.

ESK Excellent. So we can go home right? Mission accomplished.

KWN and **ESK** No!

KWN I *really* want to know why this is failing. A wise man once told me "Bugs that mysteriously disappear can mysteriously re-appear."

ESK Fair 'nuff. To learn more we have to dig deeper, much deeper. Let's start by looking at how Objective-C `for` loops work. Under the covers the compiler translates the `for` loop into something like this:

1. Allocate a buffer.
2. Call -countByEnumeratingWithState:objects:count: to populate that buffer.
3. If that returns no results, deallocate the buffer and we're done.
4. Process each object in the buffer.
5. Go back to step 2.

The key thing to notice here is the `state` parameter to that method. This is a pointer to an NSFastEnumerationState structure. One member of that structure is mutationsPtr. The dictionary sets this up to point to a value that changes when the dictionary is mutated. The `for` loop monitors that value and, if it detects a change, calls `__NSFastEnumerationMutationHandler` to raise this exception.

KWN Ah, I see where you're going here. If we can monitor that value for changes, we can see who's responsible for the mutation.

ESK Right. And the weapon of choice here is watchpoints!

KWN Please enlighten me, Jedi Master!

ESK Of course. Use Xcode to set a breakpoint on line 47 and run until we hit that breakpoint. Now disassemble the code from there.

KWN OK, here's what I see:

```
(lldb) disas -s $pc -c 17
Test`-[MPPDefaultsController initWithFile:]:
->  0x100000e9c <+236>: leaq   -0xf8(%rbp), %rax
    0x100000ea3 <+243>: movq   %rax, %rcx
    0x100000ea6 <+246>: movq   %rcx, %rdi
    0x100000ea9 <+249>: movl   $0x40, %edx
    0x100000eae <+254>: movq   %rax, -0x100(%rbp)
    0x100000eb5 <+261>: callq  0x100001136               ; symbol stub for: memset
    0x100000eba <+266>: movq   -0x90(%rbp), %rax
    0x100000ec1 <+273>: movq   0x8(%rax), %rax
    0x100000ec5 <+277>: movq   0x1d64(%rip), %rsi        ; "countByEnumeratingWithState:objects:count:"
    0x100000ecc <+284>: movq   %rax, %rcx
    0x100000ecf <+287>: movq   %rcx, %rdi
    0x100000ed2 <+290>: movq   -0x100(%rbp), %rdx
    0x100000ed9 <+297>: leaq   -0x88(%rbp), %rcx
    0x100000ee0 <+304>: movl   $0x10, %r8d
    0x100000ee6 <+310>: movq   %rax, -0x108(%rbp)
    0x100000eed <+317>: callq  *0x111d(%rip)             ; (void *)0x00007fff62ec7000: objc_msgSend
    0x100000ef3 <+323>: cmpq   $0x0, %rax
```

ESK Note the instruction at +277, which loads the `countByEnumeratingWithState:objects:count:` selector into the RSI register. That's a strong indication that the code is about to call `-countByEnumeratingWithState:objects:count:`, and we see the associated `objc_msgSend` call at +317. Set a breakpoint at that call and continue.

KWN OK, here we go:

```
(lldb) breakpoint set -a 0x100000eed
Breakpoint 3: where = Test`-[MPPDefaultsController initWithFile:] + 317 at MPPDefaultsController.m:13:9, address = 0x0000000100000eed
(lldb) continue
Process 34801 resuming
```

Dude! We stopped at +273, not +317. W-T-F!

ESK Calm down young apprentice, there's a logical explanation for this. When we used Xcode to set a breakpoint on line 47, it actually sets two breakpoints, one outside the `for` loop and one inside. Previously we stopped outside the `for` loop, and now we've stopped inside. If you continue, you'll stop at our breakpoint.

KWN *phew* Much better. Now what?

ESK We need to find the `state` parameter and the `mutationsPtr` field within that. Remember this is an Objective-C message send, so there are two hidden parameters, the object and the selector. That means the `state` parameter is the third parameter, which you can access using the LLDB convenience value `$argc3`. Dump that.

KWN Here's what I see:

```
(lldb) memory read -f x -c 3 -s 8 $arg3
0x7ffeefbfdb48: 0x0000000000000000 0x0000000000000000
0x7ffeefbfdb58: 0x0000000000000000
```

ESK Right. This is as expected. The compiler has cleared the `NSFastEnumerationState` value prior to the first call. Now step over the `objc_msgSend` call and dump the state again.

KWN OK, here's what I got:

```
(lldb) thread step-inst-over
(lldb) memory read -f x -c 3 -s 8 0x7ffeefbfdb48
0x7ffeefbfdb48: 0x0000000000000025 0x00007ffeefbfdbb8
0x7ffeefbfdb58: 0x0000600000258030
```

ESK Very clever! You remembered to use the actual address (0x7ffeefbfdb48) rather than `$arg3`, because the register represented by `$arg3` (RDX) is not preserved by the function call.

The third word in this structure is the `mutationsPtr` member. Set a watchpoint on that and continue again.

KWN [furiously enters `apropos watchpoint` into LLDB to work out the syntax] OK, here we go:

```
(lldb) watchpoint set expression -w write -s 8 -- (void*)0x0000600000258030
Watchpoint created: Watchpoint 1: addr = 0x600000258030 size = 8 state = enabled type = w
    new value: 0x1c00003900000073
(lldb) continue
Process 34987 resuming

Watchpoint 1 hit:
old value: 0x1c00003900000073
new value: 0x1e00003900000074
```

ESK Now look at the backtrace.

KWN And here it is:

```
(lldb) bt
* thread #1, queue = 'com.apple.main-thread', stop reason = watchpoint 1
  * …0… CoreFoundation`-[__NSDictionaryM setObservationInfo:] + 228
    …1… Foundation`_NSKeyValueReplaceObservationInfoForObject + 194
    …2… Foundation`-[NSObject(NSKeyValueObserverRegistration) _addObserver:forProperty:options:context:] + 403
    …3… Foundation`-[NSObject(NSKeyValueObserverRegistration) addObserver:forKeyPath:options:context:] + 93
    …4… Foundation`-[NSKeyValueNestedProperty object:didAddObservance:recurse:] + 250
    …5… Foundation`-[NSObject(NSKeyValueObserverRegistration) _addObserver:forProperty:options:context:] + 431
    …6… Foundation`-[NSObject(NSKeyValueObserverRegistration) addObserver:forKeyPath:options:context:] + 93
    …7… Test`-[MPPDefaultsController initWithFile:]… at MPPDefaultsController.m:14:13
```

It kinda makes sense that KVO is calling -setObservationInfo:, but I don't understand why that's considered a mutation.

ESK As a wise man once told *me*, "Use the source, Luke!"

KWN But Foundation isn't part of the Darwin open source.

ESK Oh, yeah, right. In that case, "Use a disassembly, Luke!" Let's disassemble `-[__NSDictionaryM setObservationInfo:]` and see if it offers any clues.

KWN I've got nothing.

ESK Let me take a look. Oh yeah, there we go! Check out the instruction at +144:

```
(lldb) disas -f
CoreFoundation`-[__NSDictionaryM setObservationInfo:]:
    …
    0x7fff2cc5b90f <+144>: callq  0x7fff2cbb39d5            ; _cow_mutate_slow
    …
->  0x7fff2cc5b963 <+228>: leaq   -0x40(%rbp), %rdi
```

In this context, *COW* stands for *copy on write*. A few releases back, Foundation adopted this technique for its collection types (see Foundation Release Notes for macOS 10.13. This means that multiple independent copies of `NSMutableDictionary` can share the same underlying state. Foundation only needs to copy that state if one of the dictionaries changes.

One fallout from this design is that a seemingly-benign change, `-setObservationInfo:`, must be treated as a mutation. This mutation is what's being detected by the `for` loop, and ultimately causing the pref pane to not load.

KWN Nasty.

ESK Yeah.

Debugging

Asked 1 year ago by eskimo

Reply to this question