

Accessing Global Variables on Apple Silicon

This thread has been locked by a moderator.



72

I was helping a developer with a gnarly issue today and, as part of that, I had to explain how Apple silicon code accesses global variables. I've done this a few times now, so I figured I might as well write it up for all.

If you have questions or comments, put them in a new thread and tag it with *Debugging* so that I see it.

Share and Enjoy

Quinn "The Eskimo!" @ Developer Technical Support @ Apple
let myEmail = "eskimo" + "1" + "@" + "apple.com"

Accessing Global Variables on Apple Silicon

Consider the `-[WKWebView navigationDelegate]` getter method. The code for that is [available in Darwin](#):

```
691 - (id <WKNavigationDelegate>)navigationDelegate
692 {
693     return _navigationState->navigationDelegate().autorelease();
694 }
```

This reads the `_navigationState` ivar, whose value is a C++ object pointer of type `NavigationState`, and calls the `navigationDelegate()` method on it. Finally, it calls the `autorelease()` method on the result.

Disassembling this on Intel you see this:

```
(lldb) disas -f
WebKit`-[WKWebView navigationDelegate]:
-> ... <+0>: pushq %rbp
... <+1>: movq %rsp, %rbp
... <+4>: pushq %rbx
... <+5>: pushq %rax
... <+6>: movq 0x875bd2(%rip), %rax ; WKWebView._navigationState
... <+13>: movq (%rdi,%rax), %rsi
... <+17>: leaq -0x10(%rbp), %rbx
... <+21>: movq %rbx, %rdi
... <+24>: callq 0x10adce41c ; WebKit::NavigationState::navigationDelegate()
... <+29>: movq (%rbx), %rdi
... <+32>: callq 0x10b37398e ; symbol stub for: CFMakeCollectable
... <+37>: movq %rax, %rdi
... <+40>: callq 0x10b37ac0c ; symbol stub for: objc_autorelease
... <+45>: addq $0x8, %rsp
... <+49>: popq %rbx
... <+50>: popq %rbp
... <+51>: retq
```

The code starting at +29 is the `autorelease()` stuff, so ignore that. Rather, focus on the code from +6 through to +24:

- At +6 it reads the `WKWebView._navigationState` global variable. The Objective-C runtime sets this up to be the offset from the start of the object to the `_navigationState` ivar.
- At +13 it reads the ivar itself.
- The remaining instructions set up the call to `navigationDelegate()`.

The instruction at +6 is a PC-relative read (`rip` is the PC). This is well supported on 64-bit Intel [1], so it's only one instruction.

Now consider this same disassembly on Apple silicon:

```
(lldb) disas -f
WebKit`-[WKWebView navigationDelegate]:
-> ... <+0>: sub sp, sp, #0x20
... <+4>: stp x29, x30, [sp, #0x10]
... <+8>: add x29, sp, #0x10
... <+12>: adrp x8, 2127
... <+16>: ldrsw x8, [x8, #0xc24]
... <+20>: ldr x0, [x0, x8]
... <+24>: add x8, sp, #0x8
... <+28>: bl 0x10523b620 ; WebKit::NavigationState::navigationDelegate()
... <+32>: ldr x0, [sp, #0x8]
... <+36>: bl 0x1057a9d8c ; symbol stub for: CFMakeCollectable
... <+40>: bl 0x1057b8270 ; symbol stub for: objc_autorelease
... <+44>: ldp x29, x30, [sp, #0x10]
... <+48>: add sp, sp, #0x20
... <+52>: ret
```

Again, the stuff from +32 onwards is uninteresting. The instructions of interest run from +12 to +28. Specifically, the two instructions at +12 and +16 represent a PC-relative read.

This requires two instructions because Apple silicon instructions are of a fixed width. There's not enough space in a 32-bit instruction to encode a large PC-relative offset. Rather, it has to be split across two instructions.

The most interesting instruction is the one at +12, `adrp`. I'm not sure what this mnemonic is officially, but I always think of it as *add relative to page*. The instruction:

- Takes an immediate value
- Shifts it left by 12 bits
- Adds it to the PC
- Masks off the bottom 12-bits
- Puts that in the target register

The instruction after the `adrp` can vary. In this case the goal is to load an `int` from a PC-relative address, so it's a load instruction, `ldrsw`. This specific syntax uses an immediate offset from a base register. This offset 'fills in' the bits 'missing' in the address calculated by the preceding `adrp` instruction.

Now, let's say that the `adrp` instruction was at PC 0x1051665c4. Here's how you calculate the value it generates:

```
(lldb) p / x ((0x1051665c4+(2127<<12))&~0xfff)
(long) $10 = 0x00000001059b5000
```

Now add the immediate from the `ldrsw` to calculate the address used by that instruction:

```
(lldb) p / a 0x00000001059b5000+0xc24
(long) $11 = 0x00000001059b5c24 WebKit`WKWebView._navigationState
```

Finally, load an `int` value from that address:

```
(lldb) p *(int *)0x00000001059b5c24
(int) $13 = 400
```

And so the value of the `WKWebView._navigationState` global variable, which is the offset of the `_navigationState` ivar within the `WKWebView` object, is 400. And on with the debugging!

[1] Notably, it was very poorly supported on 32-bit Intel, but fortunately we don't care about that any more.

Debugging

Apple Silicon

Reply

Posted 3 weeks ago by

eskimo

Add a Comment

This site contains user submitted content, comments and opinions and is for informational purposes only. Apple disclaims any and all liability for the acts, omissions and conduct of any third parties in connection with or related to your use of the site. All postings and use of the content on this site are subject to the [Apple Developer Forums Participation Agreement](#).

Developer
Forums

Platforms

iOS

iPadOS

macOS

tvOS

watchOS

Tools

Swift

SwiftUI

SF Symbols

Swift Playgrounds

TestFlight

Xcode

Xcode Cloud

Topics & Technologies

Accessibility

Accessories

App Extensions

App Store

Audio & Video

Augmented Reality

Business

Design

Distribution

Education

Fonts

Games

Health & Fitness

In-App Purchase

Localization

Maps & Location

Machine Learning

Security

Safari & Web

Resources

Documentation

Curriculum

Downloads

Forums

Videos

Support

Support Articles

Contact Us

Bug Reporting

System Status

Account

Apple Developer

App Store Connect

Certificates, IDs, & Profiles

Feedback Assistant

Programs

Apple Developer Program

Apple Developer Enterprise Program

App Store Small Business Program

MFi Program

News Partner Program

Video Partner Program

Security Bounty Program

Security Research Device Program

Events

App Accelerators

App Store Awards

Apple Design Awards

Apple Developer Academies

Entrepreneur Camp

Tech Talks

WWDC