The Alpha AXP, part 17: Reconstructing a call stack

devblogs.microsoft.com/oldnewthing/20170830-00

August 30, 2017

Raymond Chen

I'm going to wrap up the formal part of the series by applying the information we've been learning over the past several ~~days~~ weeks: We're going to reconstruct a broken stack.

Suppose you have a debug session where all the `k` command says is

```
Callee-SP Return-RA  Call Site
 0235d380 777b63e4 : contoso!OnSysColorChange+0x60
```

That's it. We'll have to unwind the stack manually. We need to keep track of the stack pointer depth and watch where the compiler stashed the return address. We can do this by going forward or backward. I'll demonstrate both.

Let's go backward:

```
kd> u .-60
contoso!OnSysColorChange:
777b6380 : 23deffe0 lda    sp,-20(sp)
777b6384 : b53e0000 stq    s0,0(sp)
777b6388 : b55e0008 stq    s1,8(sp)
777b638c : b75e0010 stq    ra,10(sp)
```

From the function prologue, we see that it allocates `0x20` bytes of stack and puts the return address at offset `0x10`. So we can pull the first return address right away:

```
kd> dd @sp+10 L1
0235d390  7773b4e8
```

Since our local stack frame is `0x20` bytes, that means that the caller's stack begins at `@sp + 20`. Let's look at the caller.

```
kd> u 7773b4e8
contoso!WndProc+1f58:
7773b4e8 : a20b0040 ldl    a0,40(s2)
7773b4ec : 47ef0411 bis    zero,fp,a1
7773b4f0 : 47ed0412 bis    zero,s4,a2
7773b4f4 : 47ec0413 bis    zero,s3,a3
7773b4f8 : e61ff989 beq    a0,0000000077739b20 contoso!WndProc+590
7773b4fc : d3407858 bsr    ra,0000000077759660 contoso!ForwardMessage
7773b500 : 47ff0400 bis    zero,zero,v0
7773b504 : c3e00122 br     zero,000000007773b990 contoso!WndProc+2400
```

Referring to the source code reveals that the last two lines are a `return 0`, so that last jump goes to the function epilogue. This time, we're debugging forward.

```
kd> u 7773b990
contoso!WndProc+2400:
7773b990 : a75e0048 ldq    ra,48(sp)
7773b994 : a57e0020 ldq    s2,20(sp)
7773b998 : a59e0028 ldq    s3,28(sp)
7773b99c : a5be0030 ldq    s4,30(sp)
7773b9a0 : a5de0038 ldq    s5,38(sp)
7773b9a4 : a53e0010 ldq    s0,10(sp)
7773b9a8 : a55e0018 ldq    s1,18(sp)
7773b9ac : a5fe0040 ldq    fp,40(sp)
7773b9b0 : 63ff0000 trapb
7773b9b4 : 23de01b0 lda    sp,1b0(sp)
7773b9b8 : 6bfa8001 ret    zero,(ra),1  contoso!OnSysColorChange+60
```

(The disassembler is "helpfully" resolving the `(ra)` to `contoso!OnSysColorChange+60`, based on the current value in the *ra* register. It's not correct because the *ra* register will certainly change between the current execution point and the `ret`, but we'll give the debugger a nice pat on the head for trying.)

By studying the epilogue, we see that the function keeps its return address at offset `0x48`. Since we already had an adjustment of `0x20` from `WndProc`, the combined offset from `@sp` is

```
kd> dd @sp+20+48 L1
0235d3e8  77c9a028
```

And now we just repeat this procedure until we get the full stack trace or we get bored.

```
kd> u 77c9a028
user32!CallWindowProcAorW+1d8:
77c9a028 : b01e0040 stl    v0,40(sp)
77c9a02c : 43e00000 addl   zero,v0,v0
77c9a030 : a75e0030 ldq    ra,30(sp)
77c9a034 : a53e0008 ldq    s0,8(sp)
77c9a038 : a55e0010 ldq    s1,10(sp)
77c9a03c : a57e0018 ldq    s2,18(sp)
77c9a040 : a59e0020 ldq    s3,20(sp)
77c9a044 : a5be0028 ldq    s4,28(sp)
77c9a048 : 23de0060 lda    sp,60(sp)
77c9a04c : 6bfa8001 ret    zero,(ra),1  contoso!OnSysColorChange+60
kd> dd @sp+20+1b0+30 l1
0235d580  77cb64c0
kd> u 77cb64c0
user32!CallWindowProcW+10:
77cb64c0 : a75e0000 ldq    ra,0(sp)
77cb64c4 : 23de0010 lda    sp,10(sp)
77cb64c8 : 6bfa8001 ret    zero,(ra),1  contoso!OnSysColorChange+60
kd> dd @sp+20+1b0+60+0 l1
0235d5b0  777a7c04
```

So we have successfully reconstructed this call stack:

```
contoso!OnSysColorChange+0x60
contoso!WndProc+1f58
user32!CallWindowProcAorW+1d8
user32!CallWindowProcW+10
```

Lather, rinse, repeat. (In this particular case, I needed to go back around 20 stack frames in order to find out why the `WM_ SYSCOLORCHANGE` message was coming in at such a strange time.)

That concludes our rather lengthy whirlwind tour of the Alpha AXP processor. Maybe you found it interesting, maybe not, but there you have it.

Next up is the MIPS R4000. But I don't do it right now, since you're probably all tired of this CPU architecture stuff. I'll wait a while and then spring it on you when you least expect it.

Raymond Chen

Follow