Can an x64 function repurpose parameter home space as general scratch space?

devblogs.microsoft.com/oldnewthing/20130830-00

August 30, 2013

Raymond Chen

We saw some time ago that the x64 calling convention in Windows reserves space for the register parameters on the stack, in case the called function wants to spill them. But can the called function use the memory for other purposes, too?

You sort of already know the answer to this question. Consider this function:

```
void testfunction(int a)
{
 a = 42;
}
```

How would a naïve compiler generate code for this function?

```
testfunction:
    sub rsp, 8 ;; realign the stack
    ;; spill all register parameters into home locations
    mov [rsp+0x10], rcx
    mov [rsp+0x18], rdx
    mov [rsp+0x20], r8
    mov [rsp+0x28], r9
    ;; a = 42
    mov [rsp+0x10], 42
    ;; return
    add rsp, 8 ;; clean up local frame
    ret
```

Observe that after spilling the register parameters into their home locations onto the stack, the function modified the local variable, which updated the value in the home location.

Since a function can arbitrarily modify a parameter, you can see that a function is therefore allowed to arbitrarily modify a parameter's home location. At which point you can see that an optimizing compiler might choose an arbitrary value completely unrelated to the parameter.

Our test function has only one parameter. What about the other three home registers?

> The caller is responsible for allocating space for parameters to the callee, and must always allocate sufficient space for the 4 register parameters, even if the callee doesn't have that many parameters.

A function can therefore treat those 32 bytes as *bonus free play*. The rationale behind those 32 bytes is that it gives you a place to spill your inbound register parameters so that they will be adjacent to the stack-based parameters. (We saw how the naïve compiler took advantage of this by not trying to be clever in its function prologue and simply spilling all register parameters whether it needs them or not.)

Nevertheless, you are free to use them for whatever purpose you like, and if you're looking at heavily-optimized code, you'll probably find that the compiler found all sorts of clever things it can do with them. For example, a common trick is to use them to save the nonvolatile registers that the function locally uses to hold the corresponding parameter!

(Did this article look familiar? Turns out I covered this article a few years ago, but I'm senile and accidentally repeated a topic. And since I put so much effort into writing it, I'm going to make you suffer through it, even though it's a repeat. Hey, television programs repeat during the summer.)

Raymond Chen

Follow