Determining approximately how much stack space is available, part 1

devblogs.microsoft.com/oldnewthing/20200609-00

June 9, 2020

Raymond Chen

Recall that the original purpose of creating a temporary fiber was to ensure that a minimum amount of stack was available for the function to perform its operations. But it would be nice if we could bypass all the fiber machinery if the existing stack is large enough.

So how can you figure out if the existing stack is large enough?

One way is to use `_alloca` and catch the stack overflow. You need to put this in a separate function because the goal of the `_alloca` is not to use the allocated memory directly, but rather to ensure that enough memory is available. You want to free the `_alloca`-allocated memory immediately, which means returning from the function immediately.

```
__declspec(noinline)
bool is_stack_available(size_t amount)
{
  __try {
    _alloca(amount);
    return true;
  } __except (
    GetExceptionCode() == EXCEPTION_STACK_OVERFLOW
              ? EXCEPTION_EXECUTE_HANDLER
              : EXCEPTION_CONTINUE_SEARCH) {
    _resetstkoflw();
    return false;
  }
}
```

If there is at least `amount` of stack remaining,[1] then the `_alloca` will succeed, and the function returns `true`. The act of returning frees the memory. This is why it's important that the function be `noinline`: We need to make sure the function actually returns.

If there is insufficient stack, then an `EXCEPTION_STACK_OVERFLOW` structured exception is raised. If that happens, then we handle the exception by calling `_resetstkoflw`[2] to re-arm the guard page, and then return `false` to let the caller know that the allocation failed.

This technique has the advantage of relying on the C runtime library itself to do the overflow detection. This defers the work of keeping things in sync with the implementation to the implementation, which is a good thing for maintenance.

On the other hand, the `_alloca` function actually allocates the memory, converting the guard pages into real committed pages. If your function doesn't always consume all of the reserved space, the memory is nevertheless committed to your process and considered recently-accessed, which can force other pages out of your process's working set.

Next time, we'll look at another way to estimate the amount of available stack space.

[1] Note that the Itanium has two stacks, so this test probes only for remaining space on the data stack. There is no obvious way to probe for remaining space in the register backing store.

[2] Somebody must have been billing by the character when that function name was chosen.

Raymond Chen

Follow