Why does the precise point at which I get a stack overflow exception change from run to run?

devblogs.microsoft.com/oldnewthing/20211216-00

December 16, 2021

Raymond Chen

Consider this program:

```
#include <stdio.h>

int maxdepth = 0;

int f()
{
  ++maxdepth;
  return f();
}

int main()
{
  __try {
    f();
  } __except (GetExceptionCode() == STATUS_STACK_OVERFLOW ?
              EXCEPTION_EXECUTE_HANDLER :
              EXCEPTION_CONTINUE_SEARCH) {
    printf("Overflow at %u\n", maxdepth);
  }
}
```

Make sure to compile this program with optimizations disabled to ensure that the recursive call occupies stack and doesn't get tail-call-optimized.

When I run this program multiple times, the results are inconsistent.

```
Overflow at 4882
Overflow at 4877
Overflow at 4879
Overflow at 4884
Overflow at 4877
Overflow at 4883
Overflow at 4882
Overflow at 4881
Overflow at 4879
Overflow at 4882
```

This program is completely deterministic. Why does the overflow happen at different stack depths? In other words, why does the size of the stack vary?

The variation in the stack size is thanks to Address Space Layout Randomization, or ASLR. The system places the initial stack pointer at a random location in the last page of the stack, thereby randomizing the low-order bits and reducing stack predictability.

It also has a secondary benefit of reducing collisions in caches that are indexed by virtual address. For example, the internal tables used by `WaitOnAddress` use the address of a stack-allocated object as a key in a hash table to keep track of all the threads that are waiting on an address. Randomizing the initial stack location means that if you have multiple threads running the same code, their stack-allocated objects won't have similar addresses.

Today, we looked at where the stack starts. Next time, we'll look at where the stack ends.

Raymond Chen

Follow