What is between ESP and EBP?

Asked 8 years, 8 months ago Active 5 months ago Viewed 92k times

38

24

Right, I'm sure this is implicitly answered many times, but I seem not to be able to quite get to it.

If you have a (x86) stack trace (say, looking at it in WinDbg), and you look at the registers, what does it mean for EBP and ESP values to be x bytes apart?

Links:

- https://stackoverflow.com/a/3699916/321013
- https://stackoverflow.com/a/2466587/321013
- https://stackoverflow.com/a/5738940/321013

To give an example of a recent stack trace I had:

```
0:016> k
ChildEBP RetAddr
1ac5ee8c 76b831bb ntdll!NtDelayExecution+0x15
1ac5eef4 76b83a8b KERNELBASE!SleepEx+0x65
1ac5ef04 0060e848 KERNELBASE!Sleep+0xf
1ac5ef10 76859d77 MyApp!application_crash::CommonUnhandledExceptionFilter+0x48
[...\applicationcrash.inc.cpp @ 47]
1ac5ef98 775a0df7 kernel32!UnhandledExceptionFilter+0x127
1ac5efa0 775a0cd4 ntdll!__RtlUserThreadStart+0x62
1ac5efb4 775a0b71 ntdll!_EH4_CallFilterFunc+0x12
1ac5efdc 77576ac9 ntdll!_except_handler4+0x8e
1ac5f000 77576a9b ntdll!ExecuteHandler2+0x26
1ac5f0b0 7754010f ntdll!ExecuteHandler+0x24
1ac5f0b0 6e8858bb ntdll!KiUserExceptionDispatcher+0xf
1ac5f400 74e68ed7 mfc80u!ATL::CSimpleStringT<wchar_t,1>::GetString
[f:\dd\vctools\vc7libs\ship\atlmfc\include\atlsimpstr.h @ 548]
1ac5fec0 6e8c818e msvcr80!_NLG_Return
[F:\dd\vctools\crt_bld\SELF_X86\crt\prebuild\eh\i386\lowhelpr.asm @ 73]
1ac5ff48 74e429bb mfc80u!_AfxThreadEntry+0xf2
[f:\dd\vctools\vc7libs\ship\atlmfc\src\mfc\thrdcore.cpp @ 109]
1ac5ff80 74e42a47 msvcr80!_callthreadstartex+0x1b
[f:\dd\vctools\crt_bld\self_x86\crt\src\threadex.c @ 348]
1ac5ff88 76833677 msvcr80!_threadstartex+0x66
[f:\dd\vctools\crt_bld\self_x86\crt\src\threadex.c @ 326]
1ac5ff94 77569f02 kernel32!BaseThreadInitThunk+0xe
1ac5ffd4 77569ed5 ntdll!__RtlUserThreadStart+0x70
1ac5ffec 00000000 ntdll!_RtlUserThreadStart+0x1b

0:016> r
eax=00000000 ebx=1ac5efc8 ecx=19850614 edx=00000000 esi=1ac5eed0 edi=00000000
eip=7754fd21 esp=1ac5ee8c ebp=1ac5eef4 iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010206
```

Values of ESP `1ac5ee8c` - EBP `1ac5eef4` = 104 bytes difference. So what's in there?

debugging visual-c++ x86 cpu-registers stack-frame

2 Answers

Active | Oldest | Votes

104

ESP is the current stack pointer. EBP is the base pointer for the current stack frame.

When you call a function, typically space is reserved on the stack for local variables. This space is usually referenced via EBP (all local variables and function parameters are a known constant offset from this register for the duration of the function call.) ESP, on the other hand, will change during the function call as other functions are called, or as temporary stack space is used for partial operation results.

Note that most compilers these days have an option to reference all local variables through ESP. This frees up EBP for use as a general purpose register.

In general, when you look at the disassembly code at the top of a function you'll see something like this:

```
push EBP
mov  EBP, ESP
sub  ESP, <some_number>
```

So EBP will point to the top of your stack for this frame, and ESP will point to the next available byte on the stack. (Stacks usually - but don't have to - grow down in memory.)

Share Improve this answer Follow

answered Feb 22 '13 at 9:29
user420442

Wrt. my example, I checked in the debugger for a call to Sleep: Neither `SleepEx` nor `NtDelayExecution` does a `push EBP` and especially `SleepEx` seems to be nontrivial. So in my example the EBP is still pointing to the `Sleep` stack location, with all the stuff from the two called functions also on the stack. – Martin Ba Feb 22 '13 at 9:53

Yes, it's possible most WinAPI functions were built with the compiler option that frees up EBP for general purpose use. Ie the compiler keeps track of how much it modifies ESP throughout the function and references local variables relative to ESP with different offsets as necessary. In this case, there is no way of knowing whether EBP-ESP accurately reflects the size of the current stack frame. Most probably it doesn't. – user420442 Feb 22 '13 at 12:31

what is `some_number` ? Not always be 0? – Joey.Z Aug 29 '17 at 10:58

@zoujyjs `some_number` depends on the amount of storage needed in the frame, and the amount needed depends on what is happening in the function. – Dan Barowy Oct 18 '17 at 17:03

1

Usually, this space is reserved for local variables that end up stored on the stack. At the start of the function, `ESP` is decremented by the appropriate value.

In your case, there are 104 bytes worth of locals in the function.

104 bytes for NtDelayExecution does seem amazing ... but then again, maybe the callstack shown is broken ... – Martin Ba Feb 22 '13 at 9:45

Well, 104 bytes is not that much, especially if `NtDelayExecution()` allocates a string buffer on the stack for some reason. – Frédéric Hamidi Feb 22 '13 at 9:47

1 I crosschecked. *Neither* `SleepEx` nor NtDelayExecution does a `push EBP` and especially `SleepEx` seems to be nontrivial. So in my example the EBP is still pointing to the `Sleep` stack location. – Martin Ba Feb 22 '13 at 9:52