

## Re: push pop ebp

**Source:** <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2004-07/0495.html>

---

**From:** Beth (*BethStone21\_at\_hotmail.NOSPICEDHAM.com*)

**Date:** 07/12/04

Date: Mon, 12 Jul 2004 00:10:55 GMT

Alex McDonald wrote:

> *Beth wrote:*

> > *Alex McDonald wrote:*

> > > *SEH expects to find ESP pointing to a valid stack, nowhere else.*

[ snip ]

> > *Although, are you completely sure about this?*

[ snip ]

> *I have spent days of fruitless debugging trying to work out why the stack is*

> *required for the chain of pointers created, and can't work out why.*

Yes, that was my line of thinking here...that there shouldn't really be a reason "why?" and that not using the stack seems like it should be possible...because it's just a "chain of pointers" and, under "flat mode", a pointer can point anywhere in the 4GB address space...

> *I've*

> *written SEH code that uses other than the stack, and the response to errors*

> *is complete shutdown of the app, immediately and without any intermediate*

> *exception handlers being fired. That makes debugging difficult, and the*

> *dubuggers I've been using won't survive the type of termination that seems*

> *to happen -- the entire process tree ad the address space just disappears,*

> *without even a puff of smoke. No errors in the event log either. All tried*

> *under Win2K.*

Oh dear; That's rather drastic, isn't it? Yeah, I can well imagine that it makes debugging "difficult", albeit "impossible"...

Well, like you, I don't see any particular reason why it should be having that effect...indeed, the entire process tree and address space simply vanishing? That surely can't be right under any circumstances an application should face...which strongly suggests "Windows bug" of some kind...

Most likely Microsoft have simply "automatically presumed" that everyone will use the stack for such things (especially as the SEH is typically generated automatically by a compiler when, under C, say, it encounters the "try" / "catch" keywords and it is easiest and most flexible for a HLL that uses the typical "stack frames" to mere tack on SEH into that whole scheme on the stack)...and this "presumption" has gotten propogated into the subsequent coding and testing of SEH...

You know, the idea of the "chain of pointers" NOT being on the stack never occurred to their programmers who've, therefore, gone and coded something that's implicitly stack-dependent...even though, from all I can see of what was intended, there actually shouldn't be any reason that it must be stack-based in itself...

By all accounts, I think you've happened across an "implementation bug" there...for sure, there should never be a situation faced that's so drastic as all the processes and address space vanishing without any log error or anything at all...that's just got to be a bug on their side of things, for sure...there is no good reason for an application to ever face this situation, whatever it may or may not have done...

And, as this is Microsoft, then the prospect of a "oops, didn't think of that" bug on their end of things being the cause is...well, do the words "buffer overrun" mean anything to anyone? It certainly should to Microsoft because, at times, it feels as if ALL buffers in Windows have got "buffer overrun" problems...indeed, that's what comes from using C and programmers that are perhaps just a little too "fresh-faced" and maybe lacking a touch of "low-level" perspective...

It's a common elementary mistake to make in C: Calling an open-ended "strcpy" rather than "strncpy" and then wondering why things are going wrong...

Indeed, Pascal-style strings are safer from such simple "oops, I did think of that!" mistakes, as well as often being faster to process...NULL-terminated is more a "convenience" for easier programming than actually being a good idea in the long run...but, indeed, once they start coding in C, it's natural to also make the OS use NULL-terminated throughout...and far too simple to call an "unsafe" routine...and presume "sure, there'll always be a NULL at the end" / "sure, the buffer will never be

exceeded" or something similar...along comes a virus coder who's maliciously looking for where you've made such dumb mistakes and presumptions...and then – BANG! – out comes "Windows quickfix patch #564738" to cure the problem a month after a virus has wreaked havoc on machines everywhere...

An excellent example of "prevention rather than cure"; If it was a "company policy" – with a "poster" stuck up above the machines saying "don't forget buffer overrun!" (and other typical "common mistakes" listed as a "checklist" to mentally tick off (US: check off) as they program away ;) to constantly remind them – then this would have likely been done right the first time around...no viruses exploiting it, no need for "quickfix patch #564738", etc., etc....a classic example of "more haste, less speed", in my view...a "gentle reminder" to their coders to look for such things and perhaps a "policy" or two that certain things should always be written in a particular "approved" way...would add on practically nothing extra to the development in the grand scheme of things – work on getting their attitude right that they simply never type in any such bugs in the first place because these kinds of bugs are always in the back of their minds – and, thus, avoids all the expensive of having to "cure" it all later (plus, though Microsoft don't themselves pay this price – except in "bad PR" – the cost of any damage a virus does exploiting it is included there)...

"Prevention rather than cure" is always the best way...don't put the bugs in there in the first place! And typical software these days has become "big and complex" enough that "oh, just do it and if there's a problem we'll come back to it" doesn't really work anymore...the odds of hacking out some small 2KB program without making any "show stoppers" in the olden Golden Days was arguably a "reasonable" chance to take...you were likely to get through it "first go" without making any great mistakes...but the odds of coding something the size and complexity of a modern OS with the same basic attitude? Nah, the odds are firmly stacked against "hacking it out first time" without errors...the underlying attitude to development in some places really needs to be brought "up to date"...especially because the audience has changed over the time too...back when it was only hobbyists and enthusiasts, there was implicitly more tolerance to things being more complicated and that a program needed to be used in a particular way as a "work-around" to a problem...this kind of thing really doesn't work "mass-market" these days...

Anyway, I've got started on Microsoft again...best stop here or I'll go on for another ten pages (and you know I'm not exaggerating in the slightest there ;)...time to get back to the actual topic...

> *The code I am writing is for a Forth; Forth is a two stack VM, and the stacks' contents just can't have any old rubbish pushed or popped off them (such as SEH frames).*

Yeah, exactly the kind of thing which displays the foolishness of making "presumptions" that every application will always be happy with doing things in the "C way"...

This is a perfectly legitimate example of where, yes, you can't have any old rubbish lying around on the stack...in my case, that got me thinking about the idea of SEH without putting it onto the stack (not that I implemented it yet but I'd "pre-planned" the idea in my mind...again, it would be interesting to find out how many people do and don't "think ahead" like that...but that's really the difference between "prevention" and "cure" to just exercise a touch of "foresight" ;)...but, well, more a case of not wanting "any old rubbish" on the stack rather than, as you've got here, a situation where you really can't have any old rubbish because of how Forth is "stack-based" throughout...

Ah, I've not touched Forth for ages...an odd but excellent language...if Intel's "reversed" operand order seems odd to someone, then Forth will drive them bananas...you "write things backwards" constantly because it's all "stack-based" throughout...and it's got to be one of the main candidates in any "explosion in an alphabet spaghetti factory" programming language contest...indeed, write it vertically rather than horizontally using a green font and you've got an "instant impression" of the Matrix ;)...

> *The solution was to reserve the first 16K or so of the ESP stack for Windows stacky type stuff; the SEH frames live there.*

Mind you, isn't that implicitly putting a 16KB limitation on things? Of course, if Windows is broken that you can't do anything else, then you can't do anything else...but, well, it really kind of "sucks" doesn't it, that just because Microsoft couldn't program it properly, such "hacks" and limitations get added onto your program...when, really, if Microsoft had done their job properly, you should have been able to do this elegantly: No "reserving" any memory (potentially wasteful...yes, I don't care if it's 1GB, 1MB, 1KB or 1 byte...a waste is a waste is a waste...generally, not even a byte should really go AWOL, UNLESS it's serving a good purpose...so, yes, "padding for alignment" is not a "waste" in the same way because you're actually getting something out of it...although, of course, if we order the variables in a particular way, is there

always a need to lose that byte or two? The point here is not anything to do with "what's 10KB to modern machines?" but a case of "if you're being this wasteful without thinking, is your mind really in the correct mode and at the right wavelength to be writing the best code"...it's not the bytes themselves that are necessarily important to be saved, you see...I'd say the point is that your brain is in "do the best job I can" mode...if you're not even thinking "good code" then how can you expect to write it? Indeed, this is perhaps another point to add onto the "ASM vs. HLL" debate...it's not necessarily that the bytes or clock cycles saved are always that important but that your attitude and your thinking are constantly thinking "is this really the best way to do it?" all the time...it's also simply the "frame of mind" it puts a coder into ;)...no need for "limitations" to be added...a case of allocating what's needed when needed (and the only implicit "limitation" being the inescapable "out of memory" error ;)...

A nice "work-around" but it's a shame that Microsoft have caused the difficulty to be "worked around" in the first place...

- > *EBP is*
- > *used as the second stack pointer, and SEH code provides a similar exception*
- > *based mechanism for providing memory to this second stack, much in the way*
- > *Windows provides it for the ESP stack.*

Ah, a clever solution...but, again, it's a pity that you had to come up with any "solution" in the first place, as there really shouldn't have been a "problem" to need a "solution"...now we've got EBP "used up" too, as well as maintaining two stacks (although, I scrub out my earlier "limitation" thing, as it seems you're saying you've worked out a way around that too ;)...

Sorry, I'm feeling all annoyed at Microsoft on your behalf now...I like the "solution": two stacks...it's the "problem" which shouldn't have been there mandating this "solution" in the first place that's crap! ;)

- > *As you noted, bitter experience. If anyone knows the answer to this, I'd be*
- > *delighted to know.*

Not at the moment; But I'll probably look into this at some point (not immediately, as I'm not doing anything with SEH at the present moment)...especially now you've told me that there's a problem that I wouldn't have expected to be there...if I discover anything interesting anywhere about this, I'll post up here to let you and others know...

alt.lang.asm: Re: push pop ebp

Beth :)