NASM Subroutine Call Segmentation Fault

Asked 8 years ago Modified 8 years ago Viewed 950 times

0

I'm working on a school project involving NASM, and while the language makes some kind of sense to me I always end up having a problem that makes no sense.

The program I'm writing involves 1 command line argument, which must be a string of 0s, 1s, and/or 2s. If it is not, an error message is displayed and the program ends.

If there are no errors, the "suffixes" of the string are displayed in order.

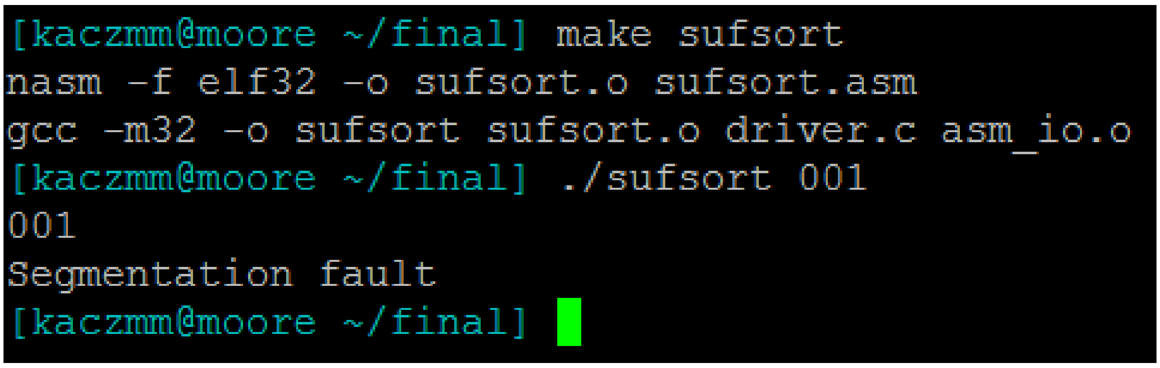
Example:

```
"./sufsort 00100102" should produce

sorted suffixes:
00100102
00102
0100102
0102
02
100102
102
2
```

Right now, I'm getting a segmentation fault when I try to call my subroutine `sufcmp`

```
%include "asm_io.inc"

section .data
    arg_error_msg: db "Error, incorrect number of
arguments. Only 2 arguments are allowed.", 0
    bad_char_msg: db "Error, invalid character
used. Only 0, 1, or 2 are allowed.", 0
    bad_length_msg: db "Error, invalid input
length. Length must be between 1 and 30.", 0
    sorted_msg: db "sorted suffixes:", 0
    ; y is an array of suffix indeces, which are
sorted later by the main method
    y: dd 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
25, 26, 27, 28, 29, 30

section .bss
    argc: resd 1        ; number of command line
arguments
    X: resb 31          ; array copy of the input
string
    N: resd 1           ; length of the input
string

section .text
    global asm_main

sufcmp:                         ; sufcmp(String Z,
int i, int j)
    enter 0, 0
    pusha

    mov edx, dword [ebp+16]     ; edx = String Z
    mov esi, dword [ebp+12]     ; esi = int i
    mov edi, dword [ebp+8]      ; edi = int j

    CMP_LOOP:
        cmp byte [edx+esi], byte 0          ; if
```

And I am getting this



```
[kaczmm@moore ~/final] make sufsort
nasm -f elf32 -o sufsort.o sufsort.asm
gcc -m32 -o sufsort sufsort.o driver.c asm_io.o
[kaczmm@moore ~/final] ./sufsort 001
001
Segmentation fault
[kaczmm@moore ~/final]
```

I can't find anything that would cause a segmentation fault, since I don't manipulate the stack in any way besides pushing the function arguments and restoring esp after the subroutine returns.

assembly x86 nasm

Share

Improve this question

Follow

edited Dec 18, 2016 at 4:37

Ped7g

16.6k ● 4 ● 31 ● 67

asked Dec 18, 2016 at 2:03

Matt

5 ● 2

2 Segmentation fault doesn't mean that there is necessarily a problem with the stack. It means that some pointer somewhere is invalid. I noticed that you are using `pusha` in 32-bit code, which is not doing what you think. (You probably want \`pushad‘.) Step through the code in a debugger to find the bad pointer. – Raymond Chen Dec 18, 2016 at 4:46

@RaymondChen: `pushad` is alias of `pusha`, the opcode is identical. The CPU will do that one by whichever mode it is ran in. But he definitely needs debugger anyway, also the invalid pointer note is spot on. – Ped7g Dec 18, 2016 at 5:36

Use a debugger (like gdb) to see which instruction faults (and what the register values are at that point). See stackoverflow.com/tags/x86/info, especially the bottom where there are gdb tips – Peter Cordes Dec 18, 2016 at 9:43

A `pusha` in 32-bit code is encoded with a 16-bit override prefix. But maybe nasm autoconverts ousha to pushad. (But then how would you say that you want a true 16-bit pusha?) – Raymond Chen Dec 18, 2016 at 18:59

1 Answer

Sorted by: Highest score (default)

0

Well, you will need debugger, as there are several problems in your code, and it's a bit too large for me to run it in head accurately (like 100% guarding stack/etc), so only few things I see:

In `CHAR_CHECK:` loop test the length during loop, so you don't overwrite `.bss` memory when somebody gives you too long string. You can move the length check right under `CHAR_LOOP:`, when `edi` is out of bounds, stop looping.

Also add the null character before storing `N` (swap those two `mov` lines), as `N` is stored right after `X` in memory, so with 31 (?) long input string you will overwrite `N` to `0` (this particularly is not exploitable, but the copy of long string may be).

`jl/jg` used in length check, but length is unsigned, so `jb/ja` would made more sense to me (not a bug, signed test `>=1 && <= 30` will fail at the same time as unsigned one, just doesn't feel right if you have programming OCD).

good/bad char test - you can make it a bit shorter by doing only two tests `('0' <= char && char <= '2')`, as `['0', '1', '2']` are values `[48, 49, 50]`.

And now more serious stuff follows.

In I/J loop you don't reset J, so logic of your inner loop will be flawed.

`push dword [X]` I don't think this does what you think it does. The address of string is `X`, `[X]` is content of memory (chars of string). (this will make the `sufcmp` code to segfault early, when it will try to access "address" `'0010'`, which is not legal.

In the swap, for example `mov edx, dword [y + edi]` ... you increment `edi` by 1, but `Y` is defined as array of dwords, so everywhere the indexing should be `edi*4`.

`cmp esi, dword [N-1] ; if i = N-1` uhm, nope, it will compare `esi` with value at address `N-1`, so if `[N]` contains 16 and ahead of it is single zero byte, the `cmp` will compare `esi` with value `4096` (memory at N-1 is: `00 10 00 00 00`, so `[N] == 0x00000010` and `[N-1] == 0x00001000`).

`mov eax, dword [X]  ; move address of X to eax` - no, `lea` would do what the comment says. `mov` will fetch content of at address `X`.

`add eax, [y + esi]` - again using +-1 indexing with `dword` array.

And you forget to call print_string, only new line is called.

You can rewrite that part as:

```
mov eax,[y + esi*4]   ; eax = Y[i]
lea eax,[X + eax]     ; eax = address X + Y[i]
```

And, as I'm cruel and tired, I kept the my biggest worry for last note.

I don't think this will work at all. Your bubble sort is iterating over original `X` string (well, it's not, but once you fix the argument issues with correct addresses, it will).

Every time. So you keep shuffling content of `Y` array according to original string in every iteration.

The most important part of my answer is the first sentence. You absolutely need debugger. If you felt like the language made some sense to you up till now, your source doesn't prove that. Actually I can see a glimpses of understanding, so you are basically right, but you would have to be total prodigy whizz kid to be able to pull this without debugger within reasonable time. I would grade you only as above-average, maybe good, but far away from prodigious premises.

If you still want to go without debugger, change technique. Don't write so much of code without compiling + running it. Do it by much much much smaller steps, and keep displaying all sort of things, to be sure your new 3 lines of code do what they should. For example if you would create empty stub for `sufcmp` just printing the string from pointer, it would segfault right after trying to access the string.

That would maybe give you better hint, than when almost final app code is segfaulting, so instead of hunting problem on recent 10 lines you have 50+ to reason about.

EDIT: algorithm suggestion:

Unless you really must use bubble sort, I would avoid that, and do the brute-force dumb "count" variant of sort.

```
i:[0,N): count[i] = countif(j:[0,N): X[j] < X[i])
i:[0,N): j:[0,N): if (i == count[j]) print X[j]
```

I hope you will be able to decipher it... it means that I would calculate for every suffix how many suffixes are "smaller" lexicographically, ie. full O(N^2) loopy loop (which is in reality N^3, because comparing strings is another O(N) ... but who cares with N=30, even N^5 would be bearable).

Then to print suffixes in correct order you simply search the `count` array again and again, first time for `0` smaller-count (that's the smallest one), then for `1`, ... etc. Till you print all of them.

Actually you may loop through all suffixes, calculate how many are smaller, and put index of that suffix into `sorted[smaller_count]`, so for printing you will just loop through `sorted` array from 0 to N-1, no searching involved.

Share Improve this answer Follow

edited Dec 18, 2016 at 5:58

answered Dec 18, 2016 at 5:32

Ped7g

16.6k ● 4 ● 31 ● 67

Of course I didn't debug your code (it's not minimal reproducible example), so I may be wrong in some assumptions, what is bug and what will not work, maybe your algorithm is correct and I just somehow don't get it – Ped7g Dec 18, 2016 at 5:33

Thank you so much! Sorry I'm so incompetent with this stuff, but like I said, assembly make some sense to me but not a lot. The reason I'm not using a debugger is because I don't know how; I never learned how to use one in my course. Thanks to your advice and a little more fiddling, it works! – Matt Dec 18, 2016 at 18:43

@Matt: nah, you are doing actually quite OK. Somewhat missed that overall algorithm, it was flawed, right? But generally you show some principle-level understanding what you are doing. My list of "problems" is somewhat too pedantic plus you need years of experience to "see" these, especially on your own source. Normally you simply do the problem, and fix it afterwards during debugging + refactoring, when it's important to switch mind into "reading alien source, not mine". About debugger: either drop the course, or learn one on your own. `gdb` works for any binary, not just asm (investment). – Ped7g Dec 18, 2016 at 18:56