

## Re: A more structured approach

---

*Source:* <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2006-06/msg00122.html>

---

- *From:* Frank Kotler <[fbkotler@xxxxxxxxxxx](mailto:fbkotler@xxxxxxxxxxx)>
  - *Date:* Tue, 06 Jun 2006 10:11:12 -0400
- 

Chinlu wrote:

....

Yes sorry, I've been looking at doing this the most difficult way

Good. If you were looking for the "easy way out", we'd send ya to a VB newsgroup! :)

....

I couldn't do it any of the ways. I'm finding gas a bit strict, and quite raw and difficult to get to know.

In the "good old days", Gas had a reputation as being "only a back end to gcc" and "not fit for human consumption". It was true, I think, but Gas has improved a \*lot\* since then. The ".intel\_syntax" switch wasn't added for gcc's convenience! Somebody's thinking about us humans. I still find Gas fairly cryptic, but I haven't spent much time with it. You're fortunate to have choices... Nasm (which won't do 64-bit), Fasm, Yasm, HLA (also 32-bit only, currently)... I think you'll find a "first date" with any of 'em somewhat... awkward. Give Gas a little more time, and if it still isn't satisfying you by the third or fourth "date", move on to something else...

I've been trying the other way round, which is, I get the beginning of the first environment variable, and then start looking in reverse order till I find the first "/", then I'd have binary name, as well as current working directory, all in one go.

I'm not sure this will work as you intend. In dos, when we find the "program name" (after the environment variables, in another segment), it's the "full path name". In Linux, what I'm seeing is exactly what I've typed. Generally, just "myprog" (since I've got "." on my path – as long as I'm not root). Might be "./myprog" or "../myprog" or "/home/myname/programs/test/gas/myprog"... Appending "myprog" to the current working directory isn't always going to give the result we want.

Now that Sevag reminds me... we had a similar conversation, and he came up with the process of: finding our PID, looking in the "proc" directory for what we \*know\* is a link to "us", then using the "readlink" syscall to

## Re: A more structured approach

return our "true name" – including full path. This works very well... if that's what we need. If we've got "mylink", a link to "myprog", this will return the full path to "myprog", even if we started it with "mylink". We could then compare that with what is says in "argv[0]". Seems rather convoluted, just to see if we were started as a link or not. Doing "stat" on what we find in "argv[0]" would tell us if it's a symbolic link or not. If "myprog" is in one of the paths in the PATH environment variable, that won't work, we'd have to do something like the "which" command – append "myprog" to each of the paths in PATH, until we find it, then "stat" that...

We may need to re-specify exactly what we're trying to do here.

I've tried all the possible ways one could imagine.

Oh, ye of limited imagination... :)

Pure heuristic,  
none of them  
worked, from cmp\*, comps\*, using rep, using test, etc, etc.  
Surprisingly, I get  
some decent result with sub. I've uploaded the source:

[http://es.geocities.com/ucho\\_trabajo/asm/test.s.txt](http://es.geocities.com/ucho_trabajo/asm/test.s.txt)

Well, let's have a look, it isn't too long...

```
..section .data

# .equ slash, 0x2f
#slash: .ascii "/"
#slash: .ascix "/"
slash: .byte 0x2f

..section .bss
.lcomm cwd_len, 4 # used to store pwd's len
.lcomm bin_len, 4 # used to store binary's name lenght
.lcomm addr, 4 # aux, for testing
.section .text
..globl _start

_start:
nop # let gdb stop in here if needed
movl %esp, %ebp # save stack pointer, just in case

movl 4(%esp), %eax # give eax argv[0]'s value

xorl $1, %ebx # set ebx to one
```

This just toggles bit 0 – won't "set it to one" unless it's already zero – which it is, in this case (depending on

## Re: A more structured approach

kernel, I think!), so you're okay.

```
movl 8(%esp, %ebx, 4), %edi # move %ebx before any command line value
```

I'm not sure what this does. If we started with no command line parameters, this would point at an environment variable, If we started with one or more command line parameters, this might point to one of them, or to the zero that separates command line args from environment variables. It seems to work correctly anyway, but I'm damned if I understand why!

```
subl %eax, %edi # (command line length()+1 should be in %edi now
```

```
dec %edi # get rid of the null separator  
movl %edi, cwd_len # save cwd's real length
```

So far, so good...

```
movl (%esp, %ebx, 4), %ecx # now, move to to the end of argv[0]
```

This moves to the *\*beginning\** of argv[0] – same value as you've got in %eax. Maybe you want to add %edi to it, to get to the end?

```
find:  
sub $slash, %ecx # don't really know if this working as expected  
jg exit # exit if found
```

Now you've *\*really\** lost me! You're subtracting the address of "slash" from... whatever – an address on the stack. Something like:

```
0xBFFF???  
-0x0804???
```

This produces a meaningless (?) number, which decrements in a loop until "jg" is true, then we exit. Say what???

```
dec %ecx # decrement %ecx  
inc %ebx # increment %ebx
```

Comments don't give much information :)

```
cmp %eax, %ecx # bail out if cwd's len is traspased  
je exit
```

This is *\*never\** going to be true (until we "wrap around"), since we started with %eax=%ecx, and have decremented %ecx at least once. If you'd added %edi to %ecx first, it would work as intended (I think).

```
jmp find
```

```
exit:  
movl $1, %eax  
int $0x80
```

## Re: A more structured approach

I suspect what you tried that \*didn't\* work was a memory to memory compare "cmpb (%ecx), slash" or some such. That would do what we want, but there's no such instruction. We can compare contents of memory with a register, or with an immediate, but not with more memory...

```
movb slash, %dl
cmpb %dl, (%ecx)
```

Or, simpler perhaps (but doesn't use our "slash")

```
cmpb $/, (%ecx)
```

I went with the simpler form. I made a couple "corrections" and got something that "worked" – unless I gave it one command line arg, when it segfaulted! More than one is okay. This re–raises my suspicion that we're not finding the end of "argv[0]" in a reliable way. I "fixed" this version so it returns zero for "found" and 1 for "not found". I had to run your original code in ald to see what was happening, and that may have confused the issue. I'll post what I've got so far, but this isn't right, and needs more work!!!

....

[re:earlier program]

... Saving the initial %esp is a good  
idea... I'm not sure %ebp is the best place...

Well, I'm taking this from the programming grund up book, where do you suggest to save it then?

What I had in mind is something like...

```
..section .bss
```

```
..lcomm initial_esp, 4
```

....

```
..section .text
```

```
_start:
```

```
nop
```

```
movl %esp, initial_esp
```

...

Then, from anywhere in our program, no matter what we've pushed or how deeply nested a subroutine, we can calculate where to find our command line args, and environment variables – we don't need to do it "first". In this example, since that's about all we're doing, it wouldn't be that helpful.

Saving %esp to %ebp is a very common thing to do – to use it as a "stack frame pointer". The initial %ebp is always ("always"... big word) preserved, so once we've returned from a subroutine, %ebp is again set to our "initial %esp", but during the subroutine, it isn't. This doesn't \*have\* to be done like this, but that's the "usual" way. So I wanted to save initial %esp someplace less "volatile" than %ebp. No matter – we don't need this yet, if we need it at all...

## Re: A more structured approach

If you like "Sevag's method" better, we can work with that. PITA to translate it to Gas, but it can be done. Have you looked at HLA? Personally, I \*hate\* the syntax (maybe more than Gas), but if you like it, or are willing to get used to it, it's quite well suited to the "structured approach".

As I mentioned above, we may need to define just what in hell we're trying to do, before proceeding much farther...

Best,  
Frank

```
..section .data

# .equ slash, 0x2f
#slash: .ascii "/"
#slash: .ascix "/"
slash: .byte 0x2f

..section .bss
.lcomm cwd_len, 4 # used to store pwd's len
.lcomm bin_len, 4 # used to store binary's name length
.lcomm addr, 4 # aux, for testing
.section .text
..globl _start

_start:
nop # let gdb stop in here if needed
movl %esp, %ebp # save stack pointer, just in case

movl 4(%esp), %eax # give eax argv[0]'s value

xorl $1, %ebx # set ebx to one
movl 8(%esp, %ebx, 4), %edi # move %ebx before any command line value

subl %eax, %edi # (command line length\0)+1 should be in %edi now

dec %edi # get rid of the null separator
movl %edi, cwd_len # save cwd's real length

movl (%esp, %ebx, 4), %ecx # now, move to to the end of argv[0]
addl %edi, %ecx

find:
cmpb $',', (%ecx)

je found # exit if found

dec %ecx # decrement %ecx

cmp %eax, %ecx # bail out if cwd's len is traspassed
je not_found
```

Re: A more structured approach

## Re: A more structured approach

```
jmp find
```

```
found:
```

```
xorl %ebx, %ebx # return zero if found
```

```
jmp exit
```

```
not_found:
```

```
movl $1, %ebx # return error 1 if not found
```

```
exit:
```

```
movl $1, %eax
```

```
int $0x80
```

```
.
```