Why does strcpy trigger a segmentation fault with global variables?

Asked 16 years, 3 months ago Modified 13 years, 1 month ago Viewed 11k times

7

So I've got some C code:

```
#include <stdio.h>
#include <string.h>

/* putting one of the "char*"s here causes a segfault */
void main() {
  char* path = "/temp";
  char* temp;
  strcpy(temp, path);
}
```

This compiles, runs, and behaves as it looks. However, if one or both of the character pointers is declared as global variable, strcpy results in a segmentation fault. Why does this happen? Evidently there's an error in my understanding of scope.

c scope pointers segmentation-fault character

Share

Improve this question

Follow

edited Nov 10, 2011 at 17:32

asked Sep 23, 2008 at 18:35

codemonkey

1,009 ● 2 ● 10 ● 16

Since I don't think it's actually going to solve the problem I'll just comment that strncpy is highly recommended over strcpy. – Josh Gagnon Sep 23, 2008 at 18:36

Josh Gagnon: Actually, strncpy doesn't place a null terminator when the input string's length is >= the buffer. strcpy is perfectly safe if you know the buffer is big enough. Otherwise, use `snprintf(buffer, buffer_len, "%s", src)` , as snprintf always puts a null terminator (just make sure buffer_len > 0). – Joey Adams Aug 5, 2011 at 6:40

@Josh: I prefer `strlcpy` . Unfortunately glibc doesn't support it so I don't get much chance to use it. I suppose I could always roll my own implementation and add it to my personal header library of null-checking malloc and file-related functions, but it still annoys me that many versions of Unix have it while Linux generally doesn't. – JAB Aug 5, 2011 at 15:42

In fact that almost screwed me over last semester of college, as I was testing a program for the fourth and final homework grade for one of my classes (which I had already missed one of the homeworks due to thinking it was due three days after it actually was) remotely and accidentally logged on to the Unix server rather than the Linux one without noticing, and compiled and ran it just fine there, but come exam time I get an e-mail from one of the class's

TAs saying "your program won't run or compile on the [Linux] test box". – JAB Aug 5, 2011 at 15:45

(Luckily I had included lines for both `strlcat` and `strncat` [similar difference to that between `strlcpy` and `strncpy` in regards to how they work and what implements them] versions of the operation that was being done, so it was just a matter of commenting out one line and uncommenting the line after.) ...Didn't expect that to take so many comment boxes. – JAB Aug 5, 2011 at 15:47

8 Answers

Sorted by: Highest score (default)

16

As other posters mentioned, the root of the problem is that temp is uninitialized. When declared as an automatic variable on the stack it will contain whatever garbage happens to be in that memory location. Apparently for the compiler+CPU+OS you are running, the garbage at that location is a valid pointer. The strcpy "succeeds" in that it does not segfault, but really it copied a string to some arbitrary location elsewhere in memory. This kind of memory corruption problem strikes fear into the hearts of C programmers everywhere as it is extraordinarily difficult to debug.

When you move the temp variable declaration to global scope, it is placed in the BSS section and automatically zeroed. Attempts to dereference *temp then result in a segfault.

When you move *path to global scope, then *temp moves up one location on the stack. The garbage at that location is apparently not a valid pointer, and so dereferencing *temp results in a segfault.

Share Improve this answer Follow

answered Sep 23, 2008 at 18:43

DGentry

16.3k ● 8 ● 53 ● 66

As expected, swapping the order of the variable declarations makes the program segfault. Thanks! – codemonkey Sep 23, 2008 at 19:58

9

The temp variable doesn't point to any storage (memory) and it is uninitialized.

if temp is declared as `char temp[32];` then the code would work no matter where it is declared. However, there are other problems with declaring temp with a fixed size like that, but that is a question for another day.

Now, why does it crash when declared globally and not locally. Luck...

When declared locally, the value of temp is coming from what ever value might be on the stack at that time. It is luck that it points to an address that doesn't cause a crash.

However, it is trashing memory used by someone else.

When declared globally, on most processors these variables will be stored in data segments that will use demand zero pages. Thus `char *temp` appears as if it was declared `char *temp=0`.

Share Improve this answer Follow

answered Sep 23, 2008 at 18:39

Torlack
4,475 ● 1 ● 25 ● 24

8

You forgot to allocate and initialize temp:

```
temp = (char *)malloc(TEMP_SIZE);
```

Just make sure TEMP_SIZE is big enough. You can also calculate this at run-time, then make sure the size is enough (should be at least strlen(path))

Share

Improve this answer

Follow

edited Sep 23, 2008 at 18:49

answered Sep 23, 2008 at 18:36

terminus
14.4k ● 8 ● 37 ● 38

There's no need to initialize the memory at temp -- strcpy will take care of this, even if it's just setting the initial null term. – Serafina Brocious Sep 23, 2008 at 18:38

In addition, it needs to be at least strlen(path)+1 to fit the null term or Bad Things T(M) will result. – Serafina Brocious Sep 23, 2008 at 18:39

He means you don't need the "temp[0] = 0" line, since strcpy() will add the NULL terminator for you. – John Millikin Sep 23, 2008 at 18:45

strcpy doesn't care what the destination string points to as long as it is large enough to contain the result. The temp[0] = 0 is pointless and even if the source string is empty, will still be set by strcpy. If you want to clear the whole string, you have to use memset (or something else) – Torlack Sep 23, 2008 at 18:47

Sorry, misunderstood 'strcpy' for 'strcat'. I fixed my answer – terminus Sep 23, 2008 at 18:49

3

As mentioned above, you forgot to allocate space for temp. I prefer `strdup` to `malloc+strcpy`. It does what you want to do.

Share Improve this answer Follow

answered Sep 23, 2008 at 18:43

Sridhar Iyer
2,830 ● 1 ● 22 ● 28

No - this doesn't work regardless of the variables - it just looks like it did because you got (un)lucky. You need to allocate space to store the contents of the string, rather than leave the variable uninitialised.

Uninitialised variables on the stack are going to be pointing at pretty much random locations of memory. If these addresses happen to be valid, your code will trample all over whatever was there, but you won't get an error (but may get nasty memory corruption related bugs elsewhere in your code).

Globals consistently fail because they usually get set to specific patterns that point to unmapped memory. Attempting to dereference these gives you an segfault immediately (which is better - leaving it to later makes the bug very hard to track down).

Share Improve this answer Follow

answered Sep 23, 2008 at 18:43

Brian

119k • 29 • 110 • 114

I'd like to rewrite first Adam's fragment as

```
// Make temp a static array of 256 chars
char temp[256];
strncpy(temp, sizeof(temp), path);
temp[sizeof(temp)-1] = '\0';
```

That way you:

```
1. don't have magic numbers laced through the code, and
2. you guarantee that your string is null terminated.
```

The second point is at the loss of the last char of your source string if it is >=256 characters long.

Share Improve this answer Follow

answered Sep 23, 2008 at 18:57

Rob Wells

37k • 13 • 84 • 147

When using wchar_t, this doesn't work. Using sizeof only works with chars. I know, this is a question about chars, but I run into a lot of bugs where people use sizeof with wchar_t.
– Torlack Sep 23, 2008 at 19:25

1

The important part to note:
destination string dest must be large enough to receive the copy.
In your situation temp has no memory allocated to copy into.

Copied from the man page of strcpy:

```
DESCRIPTION
   The  strcpy()  function  copies the string pointed to by src (including
   the terminating '\0' character) to the array pointed to by  dest.   The
   strings  may not overlap, and the destination string dest must be large
   enough to receive the copy.
```

Share Improve this answer Follow

answered Sep 23, 2008 at 18:39

Loki Astari
264k ● 86 ● 342 ● 571

1

You're invoking undefined behavior, since you're not initializing the `temp` variable. It points to a random location in memory, so your program *may* work, but most likely it will segfault. You need to have your destination string be an array, or have it point to dynamic memory:

```
// Make temp a static array of 256 chars
char temp[256];
strncpy(temp, 256, path);

// Or, use dynamic memory
char *temp = (char *)malloc(256);
strncpy(temp, 256, path);
```

Also, use `strncpy()` instead of `strcpy()` to avoid buffer overruns.

Share Improve this answer Follow

answered Sep 23, 2008 at 18:39

Adam Rosenfield
399k ● 101 ● 522 ● 597

Adam, is it a good idea to have all those magic numbers floating around? ;-) – Rob Wells Sep 23, 2008 at 18:43

that still has bugs since if the source string is of 256 length, then the strings don't have a null termination. – Torlack Sep 23, 2008 at 18:45

No need to cast the pointer returned by malloc -- malloc returns a void pointer and thus no cast is needed. In addition, casting it could hurt you because it could hide problems with malloc not being properly defined (as the implicit definition of malloc is it returning an int) at this line of code. – Daniel Papasian Jun 4, 2009 at 15:11

@Daniel Papasian: the cast is not necessary in C, but it is necessary in C++, and I usually like to make my C code as compatible with C++ as possible. The use of implicitly defined functions without declaring them is deprecated in C, and they should not be used in new C code. I always compile with -Wall, which includes -Wimplicit-function-declaration.
– Adam Rosenfield Jun 4, 2009 at 15:28