Subject: Linux .brc handling bug
Posted by zsolt on Fri, 10 Nov 2006 13:13:44 GMT
View Forum Message <> Reply to Message

I get "Segmentation fault" when trying to access a binary resource.
I gess that binary resources are compiling into code area and operating system does not allow access to it or the pointer points to a wrong address.

I tried reference/brc as well and it has this problem also.
Debugger stops at memcpy int the next method:

```
dword MemStream::_Get(void *data, dword size) {
 if(size > (dword)(uintptr_t)(rdlim - ptr))
  size = rdlim - ptr;
 memcpy(data, ptr, size);
 ptr += size;
 return size;
}
```

Subject: Re: Linux .brc handling bug
Posted by mirek on Sat, 11 Nov 2006 12:42:06 GMT
View Forum Message <> Reply to Message

Strange, just tested, reference/brc seems to work in ubuntu/64.

Platform?

Mirek

Subject: Re: Linux .brc handling bug
Posted by zsolt on Sat, 11 Nov 2006 14:02:35 GMT
View Forum Message <> Reply to Message

Ubuntu 606 32 bit.
I tried with different gcc versions (3.3, 3.4 and 4.0) but the result is the same.
gets segfault in memcpy()

Output is:

```
brc length: 99
cpp length: 1166
all count:  2
Segmentation fault
```

I will try it on my desktop machine with a newly installed U++ soon. Maybe my hacked TheIDE is the source of the problem.

Subject: Re: Linux .brc handling bug
Posted by zsolt on Sat, 11 Nov 2006 14:36:58 GMT
View Forum Message <> Reply to Message

I have tried upp-605 (downloaded binary from sf) and the problem is the same: reference/brc segfaults.

Ubuntu 606 32 bit, Intel PIV, 32 bit.

Subject: Re: Linux .brc handling bug
Posted by zsolt on Tue, 19 Dec 2006 13:13:03 GMT
View Forum Message <> Reply to Message

Can somebody help me in finding my bug with .brc handling?
I created a small app, loading and displaying main.cpp as brc.
It is working on Windows, but not on Linux (Ubuntu 606, 32 bit).
I have attached the testcase sources.

File Attachments
1) BrcTest.zip, downloaded 367 times

Subject: Re: Linux .brc handling bug
Posted by Balage on Tue, 19 Dec 2006 14:37:38 GMT
View Forum Message <> Reply to Message

Another example, that pointers and arrays ARE NOT THE SAME THING!

Modify the macro BINARY in Defs.h to:

```
#define BINARY(i, f) \
extern "C" char i[]; \
extern "C" int COMBINE(i, _length);
```

Subject: Re: Linux .brc handling bug
Posted by zsolt on Tue, 19 Dec 2006 17:41:16 GMT
View Forum Message <> Reply to Message

Thanks, you showd me, where is the problem, but I changed my source instead of the upp header.

The correct line would be:String s((const char*)&std_tmpl_main, std_tmpl_main_length);

The very "good" feature of msc is that it masks the problems and solves them silently.

Subject: Re: Linux .brc handling bug
Posted by Balage on Tue, 19 Dec 2006 18:23:35 GMT
View Forum Message <> Reply to Message

And that works? It shouldn't
Taking the address of a char* results in char**, which you convert to const char*. It's only blind luck or coincidence I suppose why that doesn't crash big

EDIT: After reading the C FAQ, the reason why that doesn't crash, is that you take the address of a variable which holds a pointer, namely the address of std_tmpl_main[0].

This part of the library is awfully messy.

I've found someone else who also had this problem:
http://cprogrammers.blogspot.com/2006/10/c-programming-diffe rent-signature.html

EDIT: I took a look at the C FAQ, and the exact problem is mentioned there (a VERY GOOD read): http://c-faq.com/aryptr/index.html

So I recommend to DON'T use the above code, but instead, use what produced the segfault, and fix the library.

Subject: Re: Linux .brc handling bug
Posted by zsolt on Tue, 19 Dec 2006 19:56:00 GMT
View Forum Message <> Reply to Message

Thanks for your help, now I undestand it better. I don't feel myself a C coder, so I didn't assume that the bug is the library.
Maybe BINARY_ARRAY and BINARY_MASK has the same problem.
I will ask Mirek about this whole thing.

Subject: Re: Linux .brc handling bug
Posted by mirek on Tue, 19 Dec 2006 20:26:08 GMT
View Forum Message <> Reply to Message

I am afraid it can be more tricky... (because in Win32 the .obj is generated directly from data). brc is not my creation, I will rather contact the author

Subject: Re: Linux .brc handling bug
Posted by Balage on Tue, 19 Dec 2006 21:36:29 GMT

View Forum Message <> Reply to Message

That would be a good thing.

Generating the .obj directly? Why? The linux solution (if the externs are used correctly) works just as good on windows.

Subject: Re: Linux .brc handling bug
Posted by mirek on Tue, 19 Dec 2006 22:36:23 GMT
View Forum Message <> Reply to Message

Balage wrote on Tue, 19 December 2006 16:36That would be a good thing.

Generating the .obj directly? Why? The linux solution (if the externs are used correctly) works just as good on windows.

Because we can? Well, Tom's COFF library is in anyway (because of .dlls), so using it for this was natural things to do at the time.

Mirek

Subject: Re: Linux .brc handling bug
Posted by Balage on Tue, 19 Dec 2006 23:06:14 GMT
View Forum Message <> Reply to Message

BINARY_MASK is also dangerous.

```
#define BINARY_MASK(i, m) \
extern "C" byte *i[]; \
extern "C" int COMBINE(i, _length)[]; \
extern "C" int COMBINE(i, _count); \
extern "C" char *COMBINE(i, _files)[];
```

The xxx_files part is generated like this:

```
char *xxx_files[] = {
 "File1.cpp",
 "File2.cpp",
};
```

That's an array of pointers, pointing to string literals. And string literals are const. So trying to modify xxx_files[i][n] also segfaults.

The solution is to either:
- Modify type to this: const char* xxx_files[]
(This also means to modify BINARY_MASK macro)

Or:
- Generate an array for each filename, then add those to xxx_files, like this:

```
static char xxx_file_1[] = "File1.cpp";

static char xxx_file_2[] = "File2.cpp";

char *xxx_files[] = {
 xxx_file_1,
 xxx_file_2,
};
```

Anyway, I also think that byte should be replaced with char, as that's what is used for the definition.

The current fix:

```
#define BINARY(i, f) \
extern "C" char i[]; \
extern "C" int COMBINE(i, _length);

#define BINARY_ARRAY(i, x, f) \
extern "C" char *i[]; \
extern "C" int COMBINE(i, _length)[]; \
extern "C" int COMBINE(i, _count);

#define BINARY_MASK(i, m) \
extern "C" char *i[]; \
extern "C" int COMBINE(i, _length)[]; \
extern "C" int COMBINE(i, _count); \
extern "C" const char *COMBINE(i, _files)[];
```

This makes types correct, so you don't accidentally segfault.