Ruby master - Bug #18893
Don't redefine memcpy(3)
07/02/2022 11:02 PM - alx (Alejandro Colomar)

Status: Closed
Priority: Normal
Assignee: shyouhei (Shyouhei Urabe)
Target version: ruby -v:
ruby 3.0.4p208 (2022-04-12 revision 3fa771dedd) [x86_64-linux-gnu]

Description
It is Undefined Behavior, by any standard ever issued.

See what I have in my system right now:
alx@asus5775:/usr/include$ grepc memcpy
./string.h:43:
extern void *memcpy (void *__restrict __dest, const void *__restrict __src,
    size_t __n) __THROW __nonnull ((1, 2));

./x86_64-linux-gnu/ruby-3.0.0/rb_mjit_min_header-3.0.4.h:1520:
extern void *memcpy (void *__restrict __dest, const void *__restrict __src,
    size_t __n) __attribute__ ((__nothrow__, __leaf__)) __attribute__ ((__nonnull__ (1, 2)));

./x86_64-linux-gnu/ruby-3.0.0/rb_mjit_min_header-3.0.4.h:1670:
extern __inline __attribute__((__always_inline__)) __attribute__((__gnu_inline__)) __attribute__ ((__artificial__)) void *
    __attribute__((__nothrow__, __leaf__)) memcpy (void *__restrict __dest, const void *__restrict __src, size_t __len)
    {
        return __builtin_memcpy_chk (__dest, __src, __len,
                        __builtin_object_size (__dest, 0));
    }

./ruby-3.0.0/ruby/internal/memory.h:278:
define memcpy ruby_nonempty_memcpy

./x86_64-linux-gnu/ruby-3.0.0/rb_mjit_min_header-3.0.4.h:22679:
define memcpy ruby_nonempty_memcpy

$ grepc ruby_nonempty_memcpy
./ruby-3.0.0/ruby/internal/memory.h:266:
static inline void *
ruby_nonempty_memcpy(void *dest, const void *src, size_t n)
{
    if (n) {
        return memcpy(dest, src, n);
    }
    else {
        return dest;
    }
}

./x86_64-linux-gnu/ruby-3.0.0/rb_mjit_min_header-3.0.4.h:5673:
__attribute__((__nonnull__ (1)))
__attribute__((__returns_nonnull__))
static inline void *
ruby_nonempty_memcpy(void *dest, const void *src, size_t n)
{
    if (n) {
        return memcpy(dest, src, n);
    } else {
        return dest;
    }
}

Some code that I maintain includes some ruby headers, which end up defining memcpy(3) to that thing. Then, my code calls
memcpy(3) from a function, which happens to be inline (yes, inline, not static inline, which is a horrible thing), and I get an error from
the compiler, for using a static function within a non-static inline function.

So, I'd like you to please remove that definition from public headers, and refrain from redefining any ISO C functions.
If not, please define it to something not broken (and yes, static inline is broken, as it produces duplicated code when not inlined; you
could use [[gnu::always_inline]] if you want to keep static, but don't). Just C99 inline would be reasonable. See also:
https://www.greenend.org.uk/rjk/tech/inline.html

Thanks,
Alex

ALX (Alejandro Colomar) wrote:

alx@asus5775:/usr/include$ grep memcpy
./string.h:43:
extern void *memcpy (void *__restrict __dest, const void *__restrict __src,
size_t __n) __THROW __nonnull ((1, 2));

This __nonnull ((1, 2)) is the cancer. We have to work around (since we do memcpy(somewhere, 0, 0)). The attribute is AFAIK never any part of the
standard. As your OS is already doing something nasty I see no reason for us only to be overly doctrinarian here.

If you really want to use system-provided memcpy you can do this:

#include <ruby/ruby.h>
#undef memcpy

Because "redefinition of symbols reserved by the standard is an undefined behaviour" as you say, this HAS to be safe I guess.

ALX - 07/04/2022 11:05 AM - alx (Alejandro Colomar)

Actually, ISO C specifies (indirectly) that memcpy(dest, NULL, 0) is Undefined Behavior.
See: https://stackoverflow.com/questions/5243012/is-it-guaranteed-to-be-safe-to-perform-memcpy0-0-0

ISO C doesn't specify the [[gnu::nonnull]] attribute (at least not yet), but that attribute makes sure that you don't invoke UB, so I don't think it's wrong here.

However, I can see reasons for calling memcpy(dest, NULL, 0) (e.g., not having to check always before calling memcpy(3)),
and glibc (and all libcs that I know of) implement memcpy(dest, NULL, 0) as a no-op (see below for the glibc implementation), so it should be a safe
call, as far as libc is concerned,
so maybe ISO C could add a guarantee that it is a safe call.

So, I see reasons to redefine memcpy(3) on your side, to remove the warning. But I think you could do it in a better way, that doesn't break users:

a - You could use ruby_nonempty_memcpy() directly, and don't redefine memcpy(3).
b - Or you could define ruby_nonempty_memcpy() as a macro, instead of a static inline, so that your memcpy(3) wrapper can be called everywhere.
c - Or you could define ruby_nonempty_memcpy() as a C99 inline function, adding an extern prototype in a .c file, and exporting the function.
d - Or use [[gnu::always_inline]], to make sure it's inlined (but we would still have a problem when a pointer to the function is taken), so (c) is better.

I think (c) will be the simplest for everyone.

About using #undef, I'm not sure it's legal to do that. It's likely to work, but I don't think users should have to do this anyway.
Thanks,
Alex

alx@asus5775:/src/gnu/glibc$ grep MEMCPY string/memcpy.c
string/memcpy.c:27:
void *
MEMCPY (void *dstpp, const void *srcpp, size_t len)
{
    unsigned long int dstp = (long int) dstpp;
    unsigned long int srcp = (long int) srcpp;

    /* Copy from the beginning to the end. */
    /* If there not too few bytes to copy, use word copy. */
    if (len >= OP_T_THRES)
    {
        /* Copy just a few bytes to make DSTP aligned. */
        len -= (-dstp) % OPSIZ;
        BYTE_COPY_FWD (dstp, srcp, (-dstp) % OPSIZ);

        /* Copy whole pages from SRCP to DSTP by virtual address manipulation,
        as much as possible. */
        PAGE_COPY_FWD_MAYBE (dstp, srcp, len, len);

        /* Copy from SRCP to DSTP taking advantage of the known alignment of
        DSTP. Number of bytes remaining is put in the third argument,
        i.e. in LEN. This number may vary from machine to machine. */
        WORD_COPY_FWD (dstp, srcp, len, len);

        /* Fall out and copy the tail. */
    }

    /* There are just a few bytes to copy. Use byte memory operations. */
    BYTE_COPY_FWD (dstp, srcp, len);

    return dstpp;
}

string/memcpy.c:24:
# define MEMCPY memcpy

alx@asus5775:/src/gnu/glibc$ grepc BYTE_COPY_FWD
./sysdeps/generic/memcopy.h:76:
#define BYTE_COPY_FWD(dst_bp, src_bp, nbytes)
  do
  { size_t __nbytes = (nbytes);
    while (__nbytes > 0)
    {
      byte __x = ((byte *) src_bp)[0];
      src_bp += 1;
      __nbytes -= 1;
      ((byte *) dst_bp)[0] = __x;
      dst_bp += 1;
    }
  } while (0)

./sysdeps/i386/memcopy.h:25:
#define BYTE_COPY_FWD(dst_bp, src_bp, nbytes)
  do
  { int __d0; 
    asm volatile("cld\n"
      "cln"
      /* Copy bytes. */
      
      "mvnsb":
      "=d" (dst_bp), "=S" (src_bp), "=c" (__d0):
      "*0" (dst_bp), "1" (src_bp), "Z" (nbytes):
      "memory");
  } while (0)
#define BYTE_COPY_FWD(dst_bp, src_bp, nbytes) \   
{ \   
  size_t __nbytes = (nbytes); \   
  if (__nbytes & 1) \   
  { \   
      ((byte *) dst_bp)[0] = ((byte *) src_bp)[0]; \   
      src_bp += 1; \   
      dst_bp += 1; \   
      __nbytes -= 1; \   
  } \   
  while (__nbytes > 0) \   
  { \   
      byte __x = ((byte *) src_bp)[0]; \   
      byte __y = ((byte *) src_bp)[1]; \   
      src_bp += 2; \   
      __nbytes -= 2; \   
      ((byte *) dst_bp)[0] = __x; \   
      ((byte *) dst_bp)[1] = __y; \   
      dst_bp += 2; \   
  } \   
} while (0)

#3 - 07/04/2022 11:12 AM - alx (Alejandro Colomar)

Also, your ruby_nonempty_memcpy() could be simplified to call memcpy(3) directly, without any ifs, since memcpy(dest, NULL, 0) is implemented as a no-op, so you could simplify the code a little bit (remove one run-time check for every call). This would only be possible if you implement ruby_nonempty_memcpy() as an inline (or static inline) function, of course.

#4 - 07/05/2022 01:00 AM - shyouhei (Shyouhei Urabe)

- Status changed from Rejected to Open
- Assignee set to shyouhei (Shyouhei Urabe)

alx (Alejandro Colomar) wrote in #note-2:

Actually, ISO C specifies (indirectly) that memcpy(dest, NULL, 0) is Undefined Behavior. See: https://stackoverflow.com/questions/5243012/is-it-guaranteed-to-be-safe-to-perform-memcpy0-0-0

ISO C doesn't specify the [[gnu::nonnull]] attribute (at least not yet), but that attribute makes sure that you don't invoke UB, so I don't think it's wrong here.

Oh, didn't know this. Thank you.

However, I can see reasons for calling memcpy(dest, NULL, 0) (e.g., not having to check always before calling memcpy(3)), and glibc (and all libcs that I know of) implement memcpy(dest, NULL, 0) as a no-op (see below for the glibc implementation), so it should be a safe call, as far as libc is concerned, so maybe ISO C could add a guarantee that it is a safe call.

So, I see reasons to redefine memcpy(3) on your side, to remove the warning. But I think you could do it in a better way, that doesn't break users:

a - You could use ruby_nonempty_memcpy() directly, and don't redefine memcpy(3).

b - Or you could define ruby_nonempty_memcpy() as a macro, instead of a static inline, so that your memcpy(3) wrapper can be called everywhere.

c - Or you could define ruby_nonempty_memcpy() as a C99 inline function, adding an extern prototype in a .c file, and exporting the function.

I think (c) will be the simplest for everyone.

About using #undef, I'm not sure it's legal to do that. It's likely to work, but I don't think users should have to do this anyway.

Thanks,

Alex

Now I'm persuaded. It is us who is doing something nasty. Let me handle this.
Do we need to force the function to be inlined?

diff --git a/include/ruby/internal/memory.h b/include/ruby/internal/memory.h
index aa3464465da..0c9fa17ee92 100644
--- a/include/ruby/internal/memory.h
+++ b/include/ruby/internal/memory.h
@@ -649,6 +649,7 @@ RBIMPL_SYMBOL_EXPORT_BEGIN()
    RBIMPL_ATTR_NONNULL((1))
    RBIMPL_ATTR_RETURNS_NONNULL()
+RBIMPL_ATTR_FORCEINLINE()

    /* At least since 2004, glibc's <string.h> annotates memcpy to be
       *attribute__((__nonnull__(1, 2))). However it is safe to pass NULL to the
       * source pointer, if n is 0. Let's wrap memcpy. */
@@ -664,7 +665,7 @@ ruby_nonempty_memcpy(void *dest, const void *src, size_t n)
 }
 RBIMPL_SYMBOL_EXPORT_END()
#undef memcpy
+#define memcpy(dest, src, n) ruby_nonempty_memcpy(dest, src, n)
#endif
#endif /* RBIMPL_MEMORY_H */

You can, but it's not the best solution.

I suggest the following patches (if you like them, I'll send them to the mailing list):

commit f9da464637f6f705d334fa5d00da7a32f4624173 (HEAD -> master)
Author: Alejandro Colomar <alx.manpages@gmail.com>
Date: Tue Jul 5 10:50:13 2022 +0200

    Simplify memcpy() wrapper

    Although ISO C specifies that memcpy(dest, NULL, 0) is Undefined
    Behavior (hence the [[gnu::nonnull]] attribute in glibc), all
    known libc implementations produce a no-op for that call, so we
    don't need to add a useless branch for that case.

    Link: <https://stackoverflow.com/questions/5243012/is-it-guaranteed-to-be-safe-to-perform-memcpy0-0-0>

Signed-off-by: Alejandro Colomar <alx.manpages@gmail.com>

diff --git a/include/ruby/internal/memory.h b/include/ruby/internal/memory.h
index f044e61ef9..b03ec5dbfd 100644
--- a/include/ruby/internal/memory.h
+++ b/include/ruby/internal/memory.h
@@ -655,12 +655,7 @@ RBIMPL_ATTR_RETURNS_NONNULL()
 inline void *
    ruby_nonempty_memcpy(void *dest, const void *src, size_t n)
    {
-       if (n) {
-          return memcpy(dest, src, n);
-       }
-       else {
-          return dest;
-       }
+       return memcpy(dest, src, n);
    }
 RBIMPL_SYMBOL_EXPORT_END()
#undef memcpy

commit b5a09c246703908fc39c4835d2206eb7a080a4bc
Author: Alejandro Colomar <alx.manpages@gmail.com>
Date: Tue Jul 5 10:42:28 2022 +0200

    ruby_nonempty_memcpy(): Use C99 inline
'static' should only be ever used in source (.c) files, since its meaning is that "this function will be private to this translation unit", which forces the compiler to create a different standalone private function for every translation unit, in cases where a function is not inlined by the compiler.

This has triggered some warnings in code that includes our header, where an inline (C99 inline) function called (unknowingly) our memcpy() wrapper, which is ruby_nonempty_memcpy() (after the preprocessor). An inline function shouldn't call a static function (since the compiler expects that 'static' will only be used in source files, not header files), and therefore the warning.

This commit removes that warning, and also improves this function, by not emitting duplicate code.

Signed-off-by: Alejandro Colomar <alx.manpages@gmail.com>

diff --git a/include/ruby/internal/memory.h b/include/ruby/internal/memory.h
index aa346465d..f044e61ef9 100644
--- a/include/ruby/internal/memory.h
+++ b/include/ruby/internal/memory.h
@@ -652,7 +652,7 @@ RBIMPL_ATTR_RETURNS_NONNULL()
 /* At least since 2004, glibc's <string.h> annotates memcpy to be
 * __attribute__((__nonnull__(1, 2))). However it is safe to pass NULL to the
 * source pointer, if n is 0. Let's wrap memcpy. */
-static inline void *
+inline void *
ruby_nonempty_memcpy(void *dest, const void *src, size_t n)
{| if (n) |

diff --git a/memory.c b/memory.c
new file mode 100644
index 0000000000..bfa080c835
--- /dev/null
+++ b/memory.c
@@ -0,0 +1,14 @@
+/**********************************************************************+
+/memory.c - Memory
+/Copyright (C) 2022 Alejandro Colomar <alx.manpages@gmail.com>
+/+%include "internal/memory.h"
+/+%include <stddef.h>
+/+
+extern inline void *ruby_nonempty_memcpy(void *dest, const void *src, size_t n);

I didn't test them; it's just a draft. Feel free to test them if you want. I also didn't check your build system, maybe it needs to include the new file too.

#7 - 07/05/2022 08:55 AM - alx (Alejandro Colomar)

shyouhei (Shyouhei Urabe) wrote in #note-4:
alx (Alejandro Colomar) wrote in #note-2:

Actually, ISO C specifies (indirectly) that memcpy(dest, NULL, 0) is Undefined Behavior. See: https://stackoverflow.com/questions/5243012/is-it-guaranteed-to-be-safe-to-perform-memcpy0-0-0

ISO C doesn't specify the [[gnu::nonnull]] attribute (at least not yet), but that attribute makes sure that you don't invoke UB, so I don't think it's wrong here.

Oh, didn't know this. Thank you.

[..]

Now I'm persuaded. It is us who is doing something nasty. Let me handle this.
I'm happy to help!

#8 - 07/06/2022 12:26 AM - shyouhei (Shyouhei Urabe)

This is my take:

From 505d00311d82fdbed80688e9db792c3b54b8ee27 Mon Sep 17 00:00:00 2001
From: =?UTF-8?q?=E5=8D=9C=E9=83=A8=E6=98=8C=E5=B9=B3?= <shyouhei@ruby-lang.org>
Date: Wed, 6 Jul 2022 09:18:28 +0900
Subject: [PATCH] do not define our own version of memcpy

The (sole) use of memcpy in our public header is now replaced to
directly call ruby_nonempty_memcpy, and the previous definition of
memcpy is now internal-only.
---
include/ruby/internal/memory.h | 6 +++++
internal.h | 3 +++
2 files changed, 4 insertions(+), 5 deletions(-)
diff --git a/include/ruby/internal/memory.h b/include/ruby/internal/memory.h
index aa3464445d..0f59262a91 100644
--- a/include/ruby/internal/memory.h
+++ b/include/ruby/internal/memory.h
@@ -363,7 +363,7 @@ typedef uint128_t DSIZE_T;
 * @return `p1`.
 * @post First `n` elements of `p2` are copied into `p1`.
 */
-#define MEMCPY(p1,p2,type,n) memcpy((p1), (p2), rbimpl_size_mul_or_raise(sizeof(type), (n)))
+#define MEMCPY(p1,p2,type,n) ruby_nonempty_memcpy((p1), (p2), rbimpl_size_mul_or_raise(sizeof(type), (n)))
/**
 * Handy macro to call memmove.
 @@ -644,7 +644,6 @@ rb_alloc_tmp_buffer2(volatile VALUE *store, long count, size_t elsize)
 return rb_alloc_tmp_buffer_with_count(store, total_size, cnt);
 }
-#if ! defined(__MINGW32__) && ! defined(__DOXYGEN__)
-RBIMPL_SYMBOL_EXPORT_BEGIN()
-RBIMPL_ATTR_NOALIAS()
-RBIMPL_ATTR_NONNULL((1))
-#endif
@@ -663,8 +662,5 @@ ruby_nonempty_memcpy(void *dest, const void *src, size_t n)
 }
}
RBIMPL_SYMBOL_EXPORT_END()
-#undef memcpy
-#define memcpy ruby_nonempty_memcpy
-#endif
-}
+/ * RBIMPL_MEMORY_H */
diff --git a/internal.h b/internal.h
index 00a8295295..7e1abcbab8 100644
--- a/internal.h
+++ b/internal.h
@@ -106,4 +106,7 @@ RUBY_SYMBOL_EXPORT_END
#define RBOOL(v) ((v) ? Qtrue : Qfalse)
#define RBignum_Type_P(x) RB_TYPE_P((x), T_BIGNUM)
+ifdefined __MINGW32__
+#define memcpy ruby_nonempty_memcpy
+#endif
+}
+}
+}
+/ * RUBY_INTERNAL_H */
-2.17.1

#9 - 07/06/2022 10:32 PM - alx (Alejandro Colomar)

shyouhei (Shyouhei Urabe) wrote in #note-8:

This is my take:

From 505d00311d82fdbed80688e9db792c3b54b8ee27 Mon Sep 17 00:00:00 2001
The (sole) use of memcpy in our public header is now replaced to
directly call ruby_nonempty_memcpy, and the previous definition of
memcpy is now internal-only.

Acked-by: Alejandro Colomar alx.manpages@gmail.com

#10 - 07/11/2022 07:24 AM - shyouhei (Shyouhei Urabe)
- Status changed from Open to Closed

This is fixed and will be included in the next release. Thank you!