Subject: Re: [PATCH] mark read_crX() asm code as volatile
Posted by Nick Piggin on Tue, 02 Oct 2007 12:14:54 GMT
View Forum Message <> Reply to Message

On Wednesday 03 October 2007 04:27, Chuck Ebbert wrote:
> On 10/02/2007 11:28 AM, Arjan van de Ven wrote:
> > On Tue, 02 Oct 2007 18:08:32 +0400
> >
> > Kirill Korotaev <dev@openvz.org> wrote:
> >> Some gcc versions (I checked at least 4.1.1 from RHEL5 & 4.1.2 from
> >> gentoo) can generate incorrect code with read_crX()/write_crX()
> >> functions mix up, due to cached results of read_crX().
> >
> > I'm not so sure volatile is the right answer, as compared to giving the
> > asm more strict contraints....
> >
> > asm volatile tends to mean something else than "the result has
> > changed"....
>
> It means "don't eliminate this code if it's reachable" which should be
> just enough for this case. But it could still be reordered in some cases
> that could break, I think.
>
> This should work because the result gets used before reading again:
>
> read_cr3(a);
> write_cr3(a | 1);
> read_cr3(a);
>
> But this might be reordered so that b gets read before the write:
>
> read_cr3(a);
> write_cr3(a | 1);
> read_cr3(b);
>
> ?

I don't see how, as write_cr3 clobbers memory.

Subject: [PATCH] mark read_crX() asm code as volatile
Posted by Kirill Korotaev on Tue, 02 Oct 2007 14:04:42 GMT
View Forum Message <> Reply to Message

Some gcc versions (I checked at least 4.1.1 from RHEL5 & 4.1.2 from gentoo)
can generate incorrect code with read_crX()/write_crX() functions mix up,
due to cached results of read_crX().

The small app for x8664 below compiled with -O2 demonstrates this
(i686 does the same thing):

```
------------------ cut ----------------------
static inline unsigned long read_cr3(void)
{
        unsigned long cr3;
        asm("movq %%cr3,%0" : "=r" (cr3));
        return cr3;
}

static inline void write_cr3(unsigned long val)
{
        asm volatile("movq %0,%%cr3" :: "r" (val) : "memory");
}

void main()
{
        unsigned long c;
        c = read_cr3();
        write_cr3(c | 0x80);
        c = read_cr3();
        write_cr3(c | 0x100);
}
------------------ cut ----------------------

# objdump -dr tst
                ....
0000000000400430 <main>:
  400430:       0f 20 d8                mov    %cr3,%rax
  400433:       48 89 c2                mov    %rax,%rdx
  400436:       80 ca 80                or     $0x80,%dl
  400439:       0f 22 da                mov    %rdx,%cr3
  40043c:       80 cc 01                or     $0x1,%ah
  40043f:       0f 22 d8                mov    %rax,%cr3
  400442:       c3                      retq
                ....
```

As one can notice, cr3 value is incorrectly read only once,
and finally updated with only one bit set.

So better be on the safe side and mark all asm statements
in read_crX() functions as volatile which helps.
i686 already has most of these functions marked as volatile already.

I faced this bug myself in i686 arch code when did code
rearrangement in 2.6.18.

Signed-Off-By: Kirill Korotaev <dev@openvz.org>
Acked-By: Pavel Emelianov <xemul@openvz.org>

---

 asm-i386/system.h | 2 +-
 asm-x86_64/system.h | 8 ++++----
 2 files changed, 5 insertions(+), 5 deletions(-)

--- ./include/asm-i386/system.h.ve4321 2007-10-02 17:09:53.000000000 +0400
+++ ./include/asm-i386/system.h 2007-10-02 17:39:40.000000000 +0400
@@ -141,7 +141,7 @@ static inline unsigned long native_read_
 {
  unsigned long val;
  /* This could fault if %cr4 does not exist */
- asm("1: movl %%cr4, %0 \n"
+ asm volatile("1: movl %%cr4, %0 \n"
   "2:   \n"
   ".section __ex_table,\"a\" \n"
   ".long 1b,2b   \n"
--- ./include/asm-x86_64/system.h.ve4321 2007-09-18 12:42:19.000000000 +0400
+++ ./include/asm-x86_64/system.h 2007-10-02 17:40:17.000000000 +0400
@@ -85,7 +85,7 @@ static inline void write_cr0(unsigned lo
 static inline unsigned long read_cr2(void)
 {
  unsigned long cr2;
- asm("movq %%cr2,%0" : "=r" (cr2));
+ asm volatile("movq %%cr2,%0" : "=r" (cr2));
  return cr2;
 }

@@ -97,7 +97,7 @@ static inline void write_cr2(unsigned lo
 static inline unsigned long read_cr3(void)
 {
  unsigned long cr3;
- asm("movq %%cr3,%0" : "=r" (cr3));
+ asm volatile("movq %%cr3,%0" : "=r" (cr3));
  return cr3;
 }

@@ -109,7 +109,7 @@ static inline void write_cr3(unsigned lo
 static inline unsigned long read_cr4(void)
 {
  unsigned long cr4;
- asm("movq %%cr4,%0" : "=r" (cr4));
+ asm volatile("movq %%cr4,%0" : "=r" (cr4));
  return cr4;

```
 }

@@ -121,7 +121,7 @@ static inline void write_cr4(unsigned lo
 static inline unsigned long read_cr8(void)
 {
  unsigned long cr8;
- asm("movq %%cr8,%0" : "=r" (cr8));
+ asm volatile("movq %%cr8,%0" : "=r" (cr8));
  return cr8;
 }
```

Subject: Re: [PATCH] mark read_crX() asm code as volatile
Posted by Andi Kleen on Tue, 02 Oct 2007 14:17:44 GMT
View Forum Message <> Reply to Message

On Tuesday 02 October 2007 16:08:32 Kirill Korotaev wrote:
> Some gcc versions (I checked at least 4.1.1 from RHEL5 & 4.1.2 from gentoo)
> can generate incorrect code with read_crX()/write_crX() functions mix up,
> due to cached results of read_crX().

added thanks

-Andi

Subject: Re: [PATCH] mark read_crX() asm code as volatile
Posted by Arjan van de Ven on Tue, 02 Oct 2007 15:28:56 GMT
View Forum Message <> Reply to Message

On Tue, 02 Oct 2007 18:08:32 +0400
Kirill Korotaev <dev@openvz.org> wrote:

> Some gcc versions (I checked at least 4.1.1 from RHEL5 & 4.1.2 from
> gentoo) can generate incorrect code with read_crX()/write_crX()
> functions mix up, due to cached results of read_crX().
>

I'm not so sure volatile is the right answer, as compared to giving the
asm more strict contraints....

asm volatile tends to mean something else than "the result has
changed"....

Subject: Re: [PATCH] mark read_crX() asm code as volatile

Posted by Nick Piggin on Tue, 02 Oct 2007 15:49:24 GMT
View Forum Message <> Reply to Message

On Wednesday 03 October 2007 16:18, H. Peter Anvin wrote:
> Nick Piggin wrote:
> >> This should work because the result gets used before reading again:
> >>
> >> read_cr3(a);
> >> write_cr3(a | 1);
> >> read_cr3(a);
> >>
> >> But this might be reordered so that b gets read before the write:
> >>
> >> read_cr3(a);
> >> write_cr3(a | 1);
> >> read_cr3(b);
> >>
> >> ?
> >
> > I don't see how, as write_cr3 clobbers memory.
>
> Because read_cr3() doesn't depend on memory, and b could be stored in a
> register.

How does the compiler know it doesn't depend on memory?

How do you say it depends on memory? You really need something
as heavy as volatile?

Subject: Re: [PATCH] mark read_crX() asm code as volatile
Posted by Chuck Ebbert on Tue, 02 Oct 2007 18:27:37 GMT
View Forum Message <> Reply to Message

On 10/02/2007 11:28 AM, Arjan van de Ven wrote:
> On Tue, 02 Oct 2007 18:08:32 +0400
> Kirill Korotaev <dev@openvz.org> wrote:
>
>> Some gcc versions (I checked at least 4.1.1 from RHEL5 & 4.1.2 from
>> gentoo) can generate incorrect code with read_crX()/write_crX()
>> functions mix up, due to cached results of read_crX().
>>
>
> I'm not so sure volatile is the right answer, as compared to giving the
> asm more strict contraints....
>
> asm volatile tends to mean something else than "the result has
> changed"....

It means "don't eliminate this code if it's reachable" which should be
just enough for this case. But it could still be reordered in some cases
that could break, I think.

This should work because the result gets used before reading again:

read_cr3(a);
write_cr3(a | 1);
read_cr3(a);

But this might be reordered so that b gets read before the write:

read_cr3(a);
write_cr3(a | 1);
read_cr3(b);

?

Subject: Re: [PATCH] mark read_crX() asm code as volatile
Posted by hpa on Tue, 02 Oct 2007 19:21:22 GMT
View Forum Message <> Reply to Message

Arjan van de Ven wrote:
> On Tue, 02 Oct 2007 18:08:32 +0400
> Kirill Korotaev <dev@openvz.org> wrote:
>
>> Some gcc versions (I checked at least 4.1.1 from RHEL5 & 4.1.2 from
>> gentoo) can generate incorrect code with read_crX()/write_crX()
>> functions mix up, due to cached results of read_crX().
>>
>
> I'm not so sure volatile is the right answer, as compared to giving the
> asm more strict contraints....
>
> asm volatile tends to mean something else than "the result has
> changed"....
>

One of the aspect of volatility is "the result will change in ways you
(gcc) just don't understand."

-hpa

Subject: Re: [PATCH] mark read_crX() asm code as volatile

Posted by hpa on Wed, 03 Oct 2007 06:18:22 GMT
View Forum Message <> Reply to Message

Nick Piggin wrote:
>>
>> This should work because the result gets used before reading again:
>>
>> read_cr3(a);
>> write_cr3(a | 1);
>> read_cr3(a);
>>
>> But this might be reordered so that b gets read before the write:
>>
>> read_cr3(a);
>> write_cr3(a | 1);
>> read_cr3(b);
>>
>> ?
>
> I don't see how, as write_cr3 clobbers memory.

Because read_cr3() doesn't depend on memory, and b could be stored in a
register.

-hpa

Subject: Re: [PATCH] mark read_crX() asm code as volatile
Posted by Kirill Korotaev on Wed, 03 Oct 2007 08:22:57 GMT
View Forum Message <> Reply to Message

Arjan,

I can experiment with any constraints if you suggest which one.

>From our experiments with gcc, it compares asm strings (sic!!!) to find matches
to be merged! Sigh...
Below are 2 programs which differ in one space in read_cr3_b() asm statement.
The first one compiles incorrectly, while 2nd one - correctly.

My personal feeling is that comparing asm strings is simply a "misfeature".

---------- cut ----------
```
static inline unsigned long read_cr3_a(void)
{
	unsigned long cr3;
	asm("movq %%cr3,%0" : "=r" (cr3));
	return cr3;
```

```
}

static inline unsigned long read_cr3_b(void)
{
	unsigned long cr3;
	asm("movq %%cr3,%0" : "=r" (cr3));
	return cr3;
}

static inline void write_cr3(unsigned long val)
{
	asm volatile("movq %0,%%cr3" :: "r" (val) : "memory");
}

void main()
{
	unsigned long c;
	c = read_cr3_a();
	write_cr3(c | 0x80);
	c = read_cr3_b();
	write_cr3(c | 0x100);
}
------------------------- cut ---------------------------


------------------------- cut ---------------------------
static inline unsigned long read_cr3_a(void)
{
	unsigned long cr3;
	asm("movq %%cr3,%0" : "=r" (cr3));
	return cr3;
}

static inline unsigned long read_cr3_b(void)
{
	unsigned long cr3;
	asm("movq  %%cr3,%0" : "=r" (cr3));
	return cr3;
}

static inline void write_cr3(unsigned long val)
{
	asm volatile("movq %0,%%cr3" :: "r" (val) : "memory");
}

void main()
{
	unsigned long c;
```

```
	c = read_cr3_a();
	write_cr3(c | 0x80);
	c = read_cr3_b();
	write_cr3(c | 0x100);
}
------------------------ cut --------------------------
```

Kirill

Arjan van de Ven wrote:
> On Tue, 02 Oct 2007 18:08:32 +0400
> Kirill Korotaev <dev@openvz.org> wrote:
>
>
>>Some gcc versions (I checked at least 4.1.1 from RHEL5 & 4.1.2 from
>>gentoo) can generate incorrect code with read_crX()/write_crX()
>>functions mix up, due to cached results of read_crX().
>>
>
>
> I'm not so sure volatile is the right answer, as compared to giving the
> asm more strict contraints....
>
> asm volatile tends to mean something else than "the result has
> changed"....
>
>

Subject: Re: [PATCH] mark read_crX() asm code as volatile
Posted by Andi Kleen on Wed, 03 Oct 2007 08:45:05 GMT
View Forum Message <> Reply to Message

>
> How does the compiler know it doesn't depend on memory?

When it has no m (or equivalent like g) constrained argument
and no memory clobber.

> How do you say it depends on memory?

You add any of the above.

> You really need something as heavy as volatile?

You could do a memory clobber, but it would be heavier than the volatile
because the memory clobber clobbers all cached variables. volatile essentially
just says "don't remove; has side effects". Normally gcc does that automatically

for something without outputs, but this one has.

Besides a CRx access does not actually clobber memory.

-Andi