

[PATCH] common implementation of iterative div/mod

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2008-05/msg04172.html>

- *From:* Jeremy Fitzhardinge <jeremy@xxxxxxxx>
- *Date:* Thu, 08 May 2008 16:16:41 +0100

We have a few instances of the open-coded iterative div/mod loop, used when we don't expect the dividend to be much bigger than the divisor. Unfortunately modern gcc's have the tendency to strength "reduce" this into a full mod operation, which isn't necessarily any faster, and even if it were, doesn't exist if gcc implements it in libgcc.

The workaround is to put a dummy asm statement in the loop to prevent gcc from performing the transformation.

This patch creates a single implementation of this loop, and uses it to replace the open-coded versions I know about.

Signed-off-by: Jeremy Fitzhardinge <jeremy.fitzhardinge@xxxxxxxx>
 Cc: Ingo Molnar <mingo@xxxxxxx>
 Cc: Thomas Gleixner <tglx@xxxxxxxxxxxxxx>
 Cc: Andrew Morton <akpm@xxxxxxxxxxxxxxxxxxxxxx>
 Cc: john stultz <johnstul@xxxxxxxxxx>
 Cc: Segher Boessenkool <segher@xxxxxxxxxxxxxxxxxxxxxx>
 Cc: Christian Kujau <lists@xxxxxxxxxxxxxxxxxx>
 Cc: Robert Hancock <hancockr@xxxxxxx>

```
---
arch/x86/xen/time.c | 13 +++-----
include/linux/math64.h | 2 ++
include/linux/time.h | 11 +------
lib/div64.c | 23 ++++++
4 files changed, 30 insertions(+), 19 deletions(-)
```

```
=====
--- a/arch/x86/xen/time.c
+++ b/arch/x86/xen/time.c
@@ -12,6 +12,7 @@
#include <linux/clocksource.h>
#include <linux/clockchips.h>
#include <linux/kernel_stat.h>
+#include <linux/math64.h>

#include <asm/xen/hypervisor.h>
#include <asm/xen/hypercall.h>
```

[PATCH] common implementation of iterative div/mod

```
@@ -150,11 +151,7 @@ static void do_stolen_accounting(void)
if (stolen < 0)
stolen = 0;

- ticks = 0;
- while (stolen >= NS_PER_TICK) {
- ticks++;
- stolen -= NS_PER_TICK;
- }
+ ticks = iter_div_u64_rem(stolen, NS_PER_TICK, &stolen);
__get_cpu_var(residual_stolen) = stolen;
account_steal_time(NULL, ticks);
```

```
@@ -166,11 +163,7 @@ static void do_stolen_accounting(void)
if (blocked < 0)
blocked = 0;

- ticks = 0;
- while (blocked >= NS_PER_TICK) {
- ticks++;
- blocked -= NS_PER_TICK;
- }
+ ticks = iter_div_u64_rem(blocked, NS_PER_TICK, &blocked);
__get_cpu_var(residual_blocked) = blocked;
account_steal_time(idle_task(smp_processor_id()), ticks);
}
```

```
=====
--- a/include/linux/math64.h
+++ b/include/linux/math64.h
@@ -81,4 +81,6 @@ static inline s64 div_s64(s64 dividend, }
#endif

+unsigned iter_div_u64_rem(u64 dividend, u32 divisor, u64 *remainder);
+
#endif /* _LINUX_MATH64_H */
```

```
=====
--- a/include/linux/time.h
+++ b/include/linux/time.h
@@ -2,6 +2,7 @@
#define _LINUX_TIME_H

#include <linux/types.h>
#include <linux/math64.h>

#ifdef __KERNEL__
# include <linux/cache.h>
@@ -172,15 +173,7 @@ extern struct timeval ns_to_timeval(const
*/
static inline void timespec_add_ns(struct timespec *a, u64 ns)
{
- ns += a->tv_nsec;
```

[PATCH] common implementation of iterative div/mod

```
- while(unlikely(ns >= NSEC_PER_SEC)) {
- /* The following asm() prevents the compiler from
- * optimising this loop into a modulo operation. */
- asm("" : "+r"(ns));
-
- ns -= NSEC_PER_SEC;
- a->tv_sec++;
- }
+ a->tv_sec += iter_div_u64_rem(a->tv_nsec + ns, NSEC_PER_SEC, &ns);
a->tv_nsec = ns;
}
#endif /* __KERNEL__ */

=====
--- a/lib/div64.c
+++ b/lib/div64.c
@@ -98,3 +98,26 @@ EXPORT_SYMBOL(div64_u64);
#endif

#endif /* BITS_PER_LONG == 32 */
+
+ /*
+ * Iterative div/mod for use when dividend is not expected to be much
+ * bigger than divisor.
+ */
+ unsigned iter_div_u64_rem(u64 dividend, u32 divisor, u64 *remainder)
+ {
+   unsigned ret = 0;
+
+   while(dividend >= divisor) {
+     /* The following asm() prevents the compiler from
+     * optimising this loop into a modulo operation. */
+     asm("" : "+rm"(dividend));
+
+     dividend -= divisor;
+     ret++;
+   }
+
+   *remainder = dividend;
+
+   return ret;
+ }
+ EXPORT_SYMBOL(iter_div_u64_rem);

--
```

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@xxxxxxxxxxxxxxxxxxx
More majordomo info at <http://vger.kernel.org/majordomo-info.html>
Please read the FAQ at <http://www.tux.org/lkml/>