

# [patch 1/2] infrastructure to debug (dynamic) objects

---

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2008-03/msg00122.html>

---

- *From:* Thomas Gleixner <[tglx@xxxxxxxxxxxxx](mailto:tglx@xxxxxxxxxxxxx)>
  - *Date:* Sat, 01 Mar 2008 10:24:56 -0000
- 

We can see an ever repeating problem pattern with objects of any kind in the kernel:

- 1) free of active objects
- 2) reinitialization of active objects

Both problems can be hard to debug because the crash happens at a point where we have no chance to decode the root cause anymore. One problem spot are kernel timers, where the detection of the problem often happens in interrupt context and usually causes the machine to panic.

While working on a timer related bug report I had to hack in specialized code into the timer subsystem to get a reasonable hint for the root cause. This debug hack was fine for temporary use, but far from a mergeable solution due to the intrusiveness into the timer code and the lack of the ability to detect and report the root cause instantly and at the same time keeping the system operational so that we have a decent chance to get hold of the debug information without special debugging aids like serial consoles.

The problems described above are not restricted to timers, but timers tend to expose it usually in a full system crash. Other objects are less explosive, but the symptoms caused by such mistakes can be even harder to debug.

Instead of creating specialized debugging code for the timer subsystem a generic infrastructure is created which allows developers to verify their code and provides an easy to enable debug facility for users in case of trouble.

The debugobjects core code keeps track of operations on static and dynamic objects by inserting them into a hashed list and sanity checking them on object operations and provides additional checks whenever kernel memory is freed.

The tracked object operations are:  
– initializing an object

[patch 1/2] infrastructure to debug (dynamic) objects

- adding an object to a subsystem list
- deleting an object from a subsystem list

Each operation is sanity checked before the operation is executed and the subsystem specific code can provide a fixup function which prevents the damage of the operation. When the sanity check triggers a warning message and a stack trace is printed.

The list of operations can be extended if the need arises. For now it's limited to the requirements of the first user (timers).

The core code enqueues the objects into hash buckets. The hash index is generated from the address of the object to simplify the lookup for the check on k/vfree. Each bucket has it's own spinlock to avoid contention on a global lock.

The debug code can be compiled in without being active. The runtime overhead is minimal and could be optimized by asm alternatives. A kernel command line option enables the debugging code.

Signed-off-by: Thomas Gleixner <tglx@xxxxxxxxxxxxx>

```

---
Documentation/kernel-parameters.txt | 2
include/linux/debugobjects.h | 47 ++++++
include/linux/mm.h | 1
init/main.c | 2
lib/Kconfig.debug | 16 ++
lib/Makefile | 1
lib/debugobjects.c | 256 ++++++++++++++++++++++++++++++++++++++
mm/page_alloc.c | 9 -
mm/slab.c | 2
mm/slub.c | 1
mm/vmalloc.c | 1
11 files changed, 336 insertions(+), 2 deletions(-)

```

Index: linux-2.6/Documentation/kernel-parameters.txt

```

=====
--- linux-2.6.orig/Documentation/kernel-parameters.txt
+++ linux-2.6/Documentation/kernel-parameters.txt
@@ -554,6 +554,8 @@ and is between 256 and 4096 characters.
1 will print _a lot_ more information - normally
only useful to kernel developers.

```

```

+ debug_objects [KNL] Enable object debugging
+
decnet.addr= [HW,NET]
Format: <area>[,<node>]
See also Documentation/networking/decnet.txt.
Index: linux-2.6/include/linux/debugobjects.h

```

```

=====
--- /dev/null

```

[patch 1/2] infrastructure to debug (dynamic) objects

```
+++ linux-2.6/include/linux/debugobjects.h
@@ -0,0 +1,47 @@
+#ifndef _LINUX_DEBUGOBJECTS_H
+#define _LINUX_DEBUGOBJECTS_H
+
+#include <linux/list.h>
+#include <linux/spinlock.h>
+
+enum debug_obj_op {
+ ODEBUG_INIT,
+ ODEBUG_ADD,
+ ODEBUG_DEL,
+ ODEBUG_FREE,
+ ODEBUG_TEST,
+};
+
+enum {
+ ODEBUG_TYPE_UNKNOWN,
+ ODEBUG_MAX_TYPES
+};
+
+struct debug_obj {
+ struct list_head list;
+ unsigned int type;
+};
+
+#ifdef CONFIG_DEBUG_OBJECT_OPS
+extern void debug_object_op(struct debug_obj *obj, enum debug_obj_op op);
+extern void debug_objects_init(void);
+
+#else
+
+static inline void debug_object_op(struct debug_obj *obj, enum debug_obj_op op)
+{
+}
+
+static inline void debug_objects_init(void) { }
+
+#endif
+
+#ifdef CONFIG_DEBUG_OBJECT_FREE
+extern void debug_check_no_obj_freed(const void *address, unsigned long size);
+#else
+static inline void
+debug_check_no_obj_freed(const void *address, unsigned long size) { }
+#endif
+
+#endif
Index: linux-2.6/include/linux/mm.h
```

---

[patch 1/2] infrastructure to debug (dynamic) objects

```
--- linux-2.6.orig/include/linux/mm.h
+++ linux-2.6/include/linux/mm.h
@@ -11,6 +11,7 @@
#include <linux/rbtree.h>
#include <linux/prio_tree.h>
#include <linux/debug_locks.h>
+#include <linux/debugobjects.h>
#include <linux/mm_types.h>
```

struct mempolicy;  
Index: linux-2.6/init/main.c

```
=====
--- linux-2.6.orig/init/main.c
+++ linux-2.6/init/main.c
@@ -52,6 +52,7 @@
#include <linux/unwind.h>
#include <linux/buffer_head.h>
#include <linux/debug_locks.h>
+#include <linux/debugobjects.h>
#include <linux/lockdep.h>
#include <linux/pid_namespace.h>
#include <linux/device.h>
@@ -523,6 +524,7 @@ asmlinkage void __init start_kernel(void
*/
unwind_init();
lockdep_init();
+ debug_objects_init();
cgroup_init_early();
```

local\_irq\_disable();  
Index: linux-2.6/lib/Kconfig.debug

```
=====
--- linux-2.6.orig/lib/Kconfig.debug
+++ linux-2.6/lib/Kconfig.debug
@@ -183,6 +183,22 @@ config TIMER_STATS
(it defaults to deactivated on bootup and will only be activated
if some application like powertop activates it explicitly).
```

```
+config DEBUG_OBJECT_OPS
+ bool "Debug object operations"
+ depends on DEBUG_KERNEL
+ help
+ If you say Y here, additional code will be inserted into the
+ kernel to validate operations on various objects.
+
+config DEBUG_OBJECT_FREE
+ bool "Debug objects in freed memory"
+ depends on DEBUG_OBJECT_OPS
+ help
+ This enables checks whether a k/v free operation frees an area
+ which contains an object which has not been deactivated
```

[patch 1/2] infrastructure to debug (dynamic) objects

+ properly. This can make kmalloc/kfree-intensive workloads  
+ much slower.

+

config DEBUG\_SLAB

bool "Debug slab memory allocations"

depends on DEBUG\_KERNEL && SLAB

Index: linux-2.6/lib/Makefile

----- linux-2.6.orig/lib/Makefile

+++ linux-2.6/lib/Makefile

@@ -36,6 +36,7 @@ obj-\$(CONFIG\_LOCK\_KERNEL) += kernel\_lock

obj-\$(CONFIG\_PLIST) += plist.o

obj-\$(CONFIG\_DEBUG\_PREEMPT) += smp\_processor\_id.o

obj-\$(CONFIG\_DEBUG\_LIST) += list\_debug.o

+obj-\$(CONFIG\_DEBUG\_OBJECT\_OPS) += debugobjects.o

ifneq (\$(CONFIG\_HAVE\_DEC\_LOCK),y)

lib-y += dec\_and\_lock.o

Index: linux-2.6/lib/debugobjects.c

----- /dev/null

+++ linux-2.6/lib/debugobjects.c

@@ -0,0 +1,256 @@

+/\*

+ \* Generic infrastructure for debugging of objects.

+ \*

+ \* Started by Thomas Gleixner

+ \*

+ \* Copyright (C) 2008, Thomas Gleixner <tglx@xxxxxxxxxxxxxx>

+ \*

+ \*

+ \* For licencing details see kernel-base/COPYING

+ \*/

+

+#include <linux/debugfs.h>

+#include <linux/debugobjects.h>

+#include <linux/seq\_file.h>

+

+#define ODEBUG\_HASH\_SIZE 4096

+#define ODEBUG\_HASH\_MASK (ODEBUG\_HASH\_SIZE - 1)

+

+struct odebug {

+ struct list\_head list;

+ spinlock\_t lock;

+};

+

+static struct odebug obj\_hash[ODEBUG\_HASH\_SIZE];

+

+static int debug\_objects\_selftest \_\_read\_mostly;

+static int debug\_objects\_maxchain \_\_read\_mostly;

+static int debug\_objects\_fixups \_\_read\_mostly;

[patch 1/2] infrastructure to debug (dynamic) objects

[patch 1/2] infrastructure to debug (dynamic) objects

```
+static int debug_objects_enabled __read_mostly;
+
+static int __init enable_object_debug(char *str)
+{
+ debug_objects_enabled = 1;
+ return 0;
+}
+
+early_param("debug_objects", enable_object_debug);
+
+/*
+ * We use the pfn of the address for the hash. That way we can check
+ * for freed objects simply by checking the affected bucket.
+ */
+static struct odebug *object_get_hash(unsigned long addr)
+{
+ unsigned long hash = (addr >> PAGE_SHIFT) & ODEBUG_HASH_MASK;
+
+ return &obj_hash[hash];
+}
+
+static const void * const debug_fixup[ODEBUG_MAX_TYPES] = {
+};
+
+static const char * const obj_types[ODEBUG_MAX_TYPES] = {
+
+ [ODEBUG_TYPE_UNKNOWN] = "unknown type",
+};
+
+static void debug_print_object(struct debug_obj *obj, char *msg)
+{
+ static int once;
+
+ if (obj->type >= ODEBUG_MAX_TYPES)
+ obj->type = ODEBUG_TYPE_UNKNOWN;
+
+ if (!once && !debug_objects_selftest) {
+ once = 1;
+ printk(KERN_ERR "ODEBUG: %s: %p %s\n", msg, obj,
+ obj_types[obj->type]);
+ WARN_ON(1);
+ }
+}
+
+static int debug_object_fixup(struct debug_obj *obj, enum debug_obj_op op)
+{
+ int (*fixup_fn)(struct debug_obj *obj, enum debug_obj_op op);
+
+ /*
+ * Try to repair the damage, so we have a better chance to get
+ * useful debug output.
+ */
+}
```

[patch 1/2] infrastructure to debug (dynamic) objects

```
+ */
+ fixup_fn = debug_fixup[obj->type];
+ if (!fixup_fn)
+ return -1;
+
+ return fixup_fn(obj, op);
+}
+
+void __debug_object_op(struct debug_obj *obj, enum debug_obj_op op)
+{
+ struct odebug *od = object_get_hash((unsigned long) obj);
+ struct debug_obj *dobj;
+ unsigned long flags;
+ int fixup = 0, cnt = 0;
+
+ spin_lock_irqsave(&od->lock, flags);
+
+ list_for_each_entry(dobj, &od->list, list) {
+ cnt++;
+ if (dobj == obj)
+ break;
+ }
+
+ if (cnt > debug_objects_maxchain)
+ debug_objects_maxchain = cnt;
+
+ /* New object ? */
+ if (dobj != obj) {
+ switch (op) {
+ case ODEB_DEBUG_ADD:
+ list_add(&obj->list, &od->list);
+ break;
+ case ODEB_DEBUG_DEL:
+ debug_print_object(obj, "delete inactive object");
+ fixup = 1;
+ break;
+ default:
+ break;
+ }
+ } else {
+ switch (op) {
+ case ODEB_DEBUG_INIT:
+ debug_print_object(obj, "init active object");
+ fixup = 1;
+ break;
+ case ODEB_DEBUG_ADD:
+ debug_print_object(obj, "add active object");
+ fixup = 1;
+ break;
+ case ODEB_DEBUG_DEL:
+ list_del(&obj->list);
```

[patch 1/2] infrastructure to debug (dynamic) objects

```
+ break;
+ default:
+ break;
+ }
+ }
+ spin_unlock_irqrestore(&od->lock, flags);
+
+ if (!fixup)
+ return;
+
+ /* try to fixup the wreckage */
+ if (!debug_object_fixup(obj, op)) {
+ debug_objects_fixups++;
+ spin_lock_irqsave(&od->lock, flags);
+ switch (op) {
+ case ODEBUG_ADD:
+ list_add(&obj->list, &od->list);
+ break;
+ default:
+ break;
+ }
+ spin_unlock_irqrestore(&od->lock, flags);
+ }
+ }
+
+/**
+ * debug_object_op - check object operation
+ * @obj: object to check
+ * @op: operation
+ */
+void debug_object_op(struct debug_obj *obj, enum debug_obj_op op)
+{
+ if (debug_objects_enabled)
+ __debug_object_op(obj, op);
+ }
+
+#ifdef CONFIG_DEBUG_OBJECT_FREE
+static void __debug_check_no_obj_freed(const void *address, unsigned long size)
+{
+ unsigned long flags, oaddr, saddr, eaddr, paddr;
+ struct debug_obj *dobj, *tmp;
+ struct odebug *od;
+ unsigned int cnt = 0, pages = PAGE_ALIGN(size) >> PAGE_SHIFT;
+
+ saddr = (unsigned long) address;
+ eaddr = saddr + size;
+ paddr = saddr & PAGE_MASK;
+
+ for (; pages > 0; pages--, paddr += PAGE_SIZE) {
+ od = object_get_hash(paddr);
+ }
```

[patch 1/2] infrastructure to debug (dynamic) objects

```
+ spin_lock_irqsave(&od->lock, flags);
+ list_for_each_entry_safe(dobj, tmp, &od->list, list) {
+ cnt++;
+ oaddr = (unsigned long) dobj;
+ if (oaddr < saddr || oaddr >= eaddr)
+ continue;
+ debug_print_object(dobj, "active object freed");
+ spin_unlock_irqrestore(&od->lock, flags);
+ debug_object_fixup(dobj, ODEBUG_FREE);
+ spin_lock_irqsave(&od->lock, flags);
+ }
+ spin_unlock_irqrestore(&od->lock, flags);
+ if (cnt > debug_objects_maxchain)
+ debug_objects_maxchain = cnt;
+ }
+ }
+
+void debug_check_no_obj_freed(const void *address, unsigned long size)
+{
+ if (debug_objects_enabled)
+ __debug_check_no_obj_freed(address, size);
+ }
+ #endif
+
+ #ifdef CONFIG_DEBUG_FS
+
+ static int debug_objects_show(struct seq_file *m, void *v)
+ {
+ seq_printf(m, "max_chain :%d\n", debug_objects_maxchain);
+ seq_printf(m, "fixups :%d\n", debug_objects_fixups);
+ return 0;
+ }
+
+ static int debug_objects_open(struct inode *inode, struct file *filp)
+ {
+ return single_open(filp, debug_objects_show, NULL);
+ }
+
+ static const struct file_operations debug_objects_fops = {
+ .open = debug_objects_open,
+ .read = seq_read,
+ .llseek = seq_lseek,
+ .release = single_release,
+ };
+
+ static void __init debug_objects_init_debugfs(void)
+ {
+ debugfs_create_file("debug_objects_info", 0600, NULL, NULL,
+ &debug_objects_fops);
+ }
+
+ 
```

[patch 1/2] infrastructure to debug (dynamic) objects

```
+#else
+static inline void debug_objects_init_debugfs(void) { }
+#endif
+
+void __init debug_objects_init(void)
+{
+ int i;
+
+ for (i = 0; i < ODEBUG_HASH_SIZE; i++) {
+ INIT_LIST_HEAD(&obj_hash[i].list);
+ spin_lock_init(&obj_hash[i].lock);
+ }
+}
+
+int __init debug_objects_do_selftest(void)
+{
+ if (!debug_objects_enabled)
+ return 0;
+
+ debug_objects_init_debugfs();
+ printk(KERN_INFO "ODEBUG: Selftest pass\n");
+ return 0;
+}
+__initcall(debug_objects_do_selftest);
```

Index: linux-2.6/mm/page\_alloc.c

```
-----
--- linux-2.6.orig/mm/page_alloc.c
+++ linux-2.6/mm/page_alloc.c
@@ -524,8 +524,11 @@ static void __free_pages_ok(struct page
if (reserved)
return;

- if (!PageHighMem(page))
+ if (!PageHighMem(page)) {
debug_check_no_locks_freed(page_address(page),PAGE_SIZE<<order);
+ debug_check_no_obj_freed(page_address(page),
+ PAGE_SIZE << order);
+ }
arch_free_page(page, order);
kernel_map_pages(page, 1 << order, 0);

@@ -986,8 +989,10 @@ static void free_hot_cold_page(struct pa
if (free_pages_check(page))
return;

- if (!PageHighMem(page))
+ if (!PageHighMem(page)) {
debug_check_no_locks_freed(page_address(page), PAGE_SIZE);
+ debug_check_no_obj_freed(page_address(page), PAGE_SIZE);
+ }
VM_BUG_ON(page_get_page_cgroup(page));
```

[patch 1/2] infrastructure to debug (dynamic) objects

```
arch_free_page(page, 0);
kernel_map_pages(page, 1, 0);
Index: linux-2.6/mm/slab.c
```

```
=====
--- linux-2.6.orig/mm/slab.c
+++ linux-2.6/mm/slab.c
@@ -3766,6 +3766,7 @@ void kmem_cache_free(struct kmem_cache *
```

```
local_irq_save(flags);
debug_check_no_locks_freed(objp, obj_size(cachep));
+ debug_check_no_obj_freed(objp, obj_size(cachep));
__cache_free(cachep, objp);
local_irq_restore(flags);
}
@@ -3791,6 +3792,7 @@ void kfree(const void *objp)
kfree_debugcheck(objp);
c = virt_to_cache(objp);
debug_check_no_locks_freed(objp, obj_size(c));
+ debug_check_no_obj_freed(objp, obj_size(c));
__cache_free(c, (void *)objp);
local_irq_restore(flags);
}
Index: linux-2.6/mm/slub.c
```

```
=====
--- linux-2.6.orig/mm/slub.c
+++ linux-2.6/mm/slub.c
@@ -1725,6 +1725,7 @@ static __always_inline void slab_free(st
```

```
local_irq_save(flags);
debug_check_no_locks_freed(object, s->objsize);
+ debug_check_no_obj_freed(object, s->objsize);
c = get_cpu_slab(s, smp_processor_id());
if (likely(page == c->page && c->node >= 0)) {
object[c->offset] = c->freelist;
Index: linux-2.6/mm/vmalloc.c
```

```
=====
--- linux-2.6.orig/mm/vmalloc.c
+++ linux-2.6/mm/vmalloc.c
@@ -383,6 +383,7 @@ static void __vunmap(const void *addr, i
}
```

```
debug_check_no_locks_freed(addr, area->size);
+ debug_check_no_obj_freed(addr, area->size);
```

```
if (deallocate_pages) {
int i;
```

```
---
```

```
---
```

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in

[patch 1/2] infrastructure to debug (dynamic) objects

the body of a message to majordomo@xxxxxxxxxxxxxxxx

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>