Subject: [PATCH 0/5] Kernel memory accounting container (v4)
Posted by Pavel Emelianov on Fri, 21 Sep 2007 09:14:15 GMT
View Forum Message <> Reply to Message

Long time ago we decided to start memory control with the
user memory container. Now this container in -mm tree and
I think we can start with (at least discussion of) the
kmem one.

Changes since v.3:
* moved alloc/free notification into slow path and make
  "notify-able" caches walk this path always;
* introduced some optimization for the case, when there's
  only one listener for SLUB events (saves more that 10%
  of performance);
* ported on 2.6.23-rc6-mm1 tree.

Changes since v.2:
* introduced generic notifiers for slub. right now there
  are only events, needed by accounting, but this set can
  be extended in the future;
* moved the controller core into separate file, so that
  its extension and/or porting on slAb will look more
  logical;
* fixed this message :).

Changes since v.1:
* fixed Paul's comment about subsystem registration;
* return ERR_PTR from ->create callback, not NULL;
* make container-to-object assignment in rcu-safe section;
* make turning accounting on and off with "1" and "0".

==========

First of all - why do we need this kind of control. The major
"pros" is that kernel memory control protects the system
from DoS attacks by processes that live in container. As our
experience shows many exploits simply do not work in the
container with limited kernel memory.

I can split the kernel memory container into 4 parts:

1. kmalloc-ed objects control
2. vmalloc-ed objects control
3. buddy allocated pages control
4. kmem_cache_alloc-ed objects control

the control of first tree types of objects has one peculiarity:

one need to explicitly point out which allocations he wants to
account and this becomes not-configurable and is to be discussed.

On the other hands such objects as anon_vma-s, file-s, sighangds,
vfsmounts, etc are created by user request always and should
always be accounted. Fortunately they are allocated from their
own caches and thus the whole kmem cache can be accountable.

This is exactly what this patchset does - it adds the ability
to account for the total size of kmem-cache-allocated objects
from specified kmem caches.

This is based on the SLUB allocator, Paul's containers and the
resource counters I made for RSS controller and which are in
-mm tree already.

To play with it, one need to mount the container file system
with -o kmem and then mark some caches as accountable via
/sys/slab/<cache_name>/cache_notify.

As I have already told kmalloc caches cannot be accounted easily
so turning the accounting on for them will fail with -EINVAL.

Turning the accounting off is possible only if the cache has
no objects. This is done so because turning accounting off
implies unaccounting of all the objects in the cache, but due
to full-pages in slub are not stored in any lists (usually)
this is impossible to do so, however I'm open for discussion
of how to make this work.

Thanks,
Pavel

Subject: [PATCH 1/4] Add notification about some major slab events
Posted by Pavel Emelianov on Fri, 21 Sep 2007 09:17:17 GMT
View Forum Message <> Reply to Message

According to Christoph, there are already multiple people who
want to control slab allocations and track memory for various
reasons. So this is an introduction of such a hooks.

Currently, functions that are to call the notifiers are empty
and marked as "weak". Thus, if there's only _one_ listener
to these events, it can happily link with the vmlinux and
handle the events with more than 10% of performance saved.

The events tracked are:

1. allocation of an object;
2. freeing of an object;
3. allocation of a new page for objects;
4. freeing this page.

More events can be added on demand.

The kmem cache marked with SLAB_NOTIFY flag will cause all the
events above to generate notifications. By default no caches
come with this flag.

The events are generated on slow paths only and as soon as the
cache is marked as SLAB_NOTIFY, it will always use them for
allocation.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/include/linux/slab.h b/include/linux/slab.h
index f3a8eec..68d8e65 100644
--- a/include/linux/slab.h
+++ b/include/linux/slab.h
@@ -28,6 +28,7 @@
 #define SLAB_DESTROY_BY_RCU 0x00080000UL /* Defer freeing slabs to RCU */
 #define SLAB_MEM_SPREAD  0x00100000UL /* Spread some memory over cpuset */
 #define SLAB_TRACE  0x00200000UL /* Trace allocations and frees */
+#define SLAB_NOTIFY  0x00400000UL /* Notify major events */

 /* The following flags affect the page allocator grouping pages by mobility */
 #define SLAB_RECLAIM_ACCOUNT 0x00020000UL  /* Objects are reclaimable */
diff --git a/mm/slub.c b/mm/slub.c
index ac4f157..b5af598 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -1040,6 +1040,29 @@ static inline unsigned long kmem_cache_f
 }
 #define slub_debug 0
 #endif
+
+int __attribute__ ((weak))
+slub_alloc_notify(struct kmem_cache *cachep, void *obj, gfp_t gfp)
+{
+ return 0;
+}
+
+void __attribute__ ((weak))
+slub_free_notify(struct kmem_cache *cachep, void *obj)
```

```
+{
+}
+
+int __attribute__ ((weak))
+slub_newpage_notify(struct kmem_cache *cachep, struct page *pg, gfp_t gfp)
+{
+ return 0;
+}
+
+void __attribute__ ((weak))
+slub_freepage_notify(struct kmem_cache *cachep, struct page *pg)
+{
+}
+
 /*
  * Slab allocation and freeing
  */
@@ -1063,7 +1162,11 @@ static struct page *allocate_slab(struct
   page = alloc_pages_node(node, flags, s->order);

  if (!page)
-  return NULL;
+  goto out;
+
+ if ((s->flags & SLAB_NOTIFY) &&
+   slub_newpage_notify(s, page, flags) < 0)
+  goto out_free;

  mod_zone_page_state(page_zone(page),
   (s->flags & SLAB_RECLAIM_ACCOUNT) ?
@@ -1071,6 +1174,11 @@ static struct page *allocate_slab(struct
   pages);

  return page;
+
+out_free:
+ __free_pages(page, s->order);
+out:
+ return NULL;
 }

 static void setup_object(struct kmem_cache *s, struct page *page,
@@ -1158,6 +1266,9 @@ static void rcu_free_slab(struct rcu_hea

 static void free_slab(struct kmem_cache *s, struct page *page)
 {
+ if (s->flags & SLAB_NOTIFY)
+  slub_freepage_notify(s, page);
```

```
+
  if (unlikely(s->flags & SLAB_DESTROY_BY_RCU)) {
   /*
    * RCU free overloads the RCU head over the LRU
@@ -1486,7 +1597,7 @@ load_freelist:
  object = c->page->freelist;
  if (unlikely(!object))
   goto another_slab;
- if (unlikely(SlabDebug(c->page)))
+ if (unlikely(SlabDebug(c->page)) || (s->flags & SLAB_NOTIFY))
   goto debug;

  object = c->page->freelist;
@@ -1545,12 +1656,20 @@ new_slab:
  return NULL;
 debug:
  object = c->page->freelist;
- if (!alloc_debug_processing(s, c->page, object, addr))
+ if (SlabDebug(c->page) &&
+    !alloc_debug_processing(s, c->page, object, addr))
   goto another_slab;

+ if ((s->flags & SLAB_NOTIFY) &&
+    slub_alloc_notify(s, object, gfpflags) < 0) {
+  object = NULL;
+  goto out;
+ }
+
  c->page->inuse++;
  c->page->freelist = object[c->offset];
  c->node = -1;
+out:
  slab_unlock(c->page);
  return object;
 }
@@ -1620,7 +1739,7 @@ static void __slab_free(struct kmem_cach

  slab_lock(page);

- if (unlikely(SlabDebug(page)))
+ if (unlikely(SlabDebug(page)) || (s->flags & SLAB_NOTIFY))
   goto debug;
 checks_ok:
  prior = object[offset] = page->freelist;
@@ -1657,8 +1776,12 @@ slab_empty:
  return;

 debug:
```

```
- if (!free_debug_processing(s, page, x, addr))
+ if (SlabDebug(page) && !free_debug_processing(s, page, x, addr))
   goto out_unlock;
+
+ if (s->flags & SLAB_NOTIFY)
+  slub_free_notify(s, x);
+
  goto checks_ok;
 }
```

Subject: [PATCH 2/5] Generic notifiers for SLUB events
Posted by Pavel Emelianov on Fri, 21 Sep 2007 09:19:58 GMT
View Forum Message <> Reply to Message

If the user wants more than one listener for SLUB event,
it can register them all as notifier_block-s so all of
them will be notified.

This costs us about 10% of performance loss, in comparison
with static linking.

The selected method of notification is srcu notifier blocks.
This is selected because the "call" path, i.e. when the
notification is done, is lockless and at the same time the
handlers can sleep. Neither regular nor atomic notifiers
provide such facilities.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/init/Kconfig b/init/Kconfig
index 0bb211a..326fd55 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -593,6 +599,16 @@ config SLUB_DEBUG
    SLUB sysfs support. /sys/slab will not exist and there will be
    no support for cache validation etc.

+config SLUB_NOTIFY
+ default y
+ bool "Enable SLUB events generic notification"
+ depends on SLUB
+ help
+   When Y, this option enables generic notifications of some major
+   slub events. However, if you do know that there will be the
+   only listener for them, you may say N here, so that callbacks
```

```
+   will be called directly.
+
 choice
  prompt "Choose SLAB allocator"
  default SLUB
diff --git a/include/linux/slub_def.h b/include/linux/slub_def.h
index 40801e7..fd9a3d4 100644
--- a/include/linux/slub_def.h
+++ b/include/linux/slub_def.h
@@ -200,4 +202,20 @@ static __always_inline void *kmalloc_nod
 }
 #endif

+struct slub_notify_struct {
+ struct kmem_cache *cachep;
+ void *objp;
+ gfp_t gfp;
+};
+
+enum {
+ SLUB_ALLOC,
+ SLUB_FREE,
+ SLUB_NEWPAGE,
+ SLUB_FREEPAGE,
+};
+
+int slub_register_notifier(struct notifier_block *nb);
+void slub_unregister_notifier(struct notifier_block *nb);
+
 #endif /* _LINUX_SLUB_DEF_H */
diff --git a/mm/slub.c b/mm/slub.c
index ac4f157..b5af598 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -1040,6 +1040,81 @@ static inline unsigned long kmem_cache_f
 #define slub_debug 0
 #endif

+/*
+ * notifiers
+ */
+
+#ifdef CONFIG_SLUB_NOTIFY
+static struct srcu_notifier_head slub_nb;
+
+static noinline
+int __slub_alloc_notify(int cmd_alloc, int cmd_free, struct kmem_cache *cachep,
+  void *obj, gfp_t gfp)
```

```
+{
+ int ret, called;
+ struct slub_notify_struct arg;
+
+ arg.cachep = cachep;
+ arg.objp = obj;
+ arg.gfp = gfp;
+
+ ret = __srcu_notifier_call_chain(&slub_nb, cmd_alloc, &arg,
+   -1, &called);
+ ret = notifier_to_errno(ret);
+
+ if (ret < 0)
+  __srcu_notifier_call_chain(&slub_nb, cmd_free, &arg,
+    called, NULL);
+
+ return ret;
+}
+
+static noinline
+void __slub_free_notify(int cmd, struct kmem_cache *cachep, void *obj)
+{
+ struct slub_notify_struct arg;
+
+ arg.cachep = cachep;
+ arg.objp = obj;
+ arg.gfp = 0;
+
+ srcu_notifier_call_chain(&slub_nb, cmd, &arg);
+}
+
+int slub_register_notifier(struct notifier_block *nb)
+{
+ return srcu_notifier_chain_register(&slub_nb, nb);
+}
+
+void slub_unregister_notifier(struct notifier_block *nb)
+{
+ srcu_notifier_chain_unregister(&slub_nb, nb);
+}
+
+static inline
+int slub_alloc_notify(struct kmem_cache *cachep, void *obj, gfp_t gfp)
+{
+ return __slub_alloc_notify(SLUB_ALLOC, SLUB_FREE, cachep, obj, gfp);
+}
+
+static inline
```

```
+void slub_free_notify(struct kmem_cache *cachep, void *obj)
+{
+ __slub_free_notify(SLUB_FREE, cachep, obj);
+}
+
+static inline
+int slub_newpage_notify(struct kmem_cache *cachep, struct page *pg, gfp_t gfp)
+{
+ return __slub_alloc_notify(SLUB_NEWPAGE, SLUB_FREEPAGE, cachep, pg, gfp);
+}
+
+static inline
+void slub_freepage_notify(struct kmem_cache *cachep, struct page *pg)
+{
+ __slub_free_notify(SLUB_FREEPAGE, cachep, pg);
+}
+#else
 int __attribute__ ((weak))
 slub_alloc_notify(struct kmem_cache *cachep, void *obj, gfp_t gfp)
 {
@@ -1060,6 +1040,7 @@ static inline unsigned long kmem_cache_f
 slub_freepage_notify(struct kmem_cache *cachep, struct page *pg)
 {
 }
+#endif

 /*
  * Slab allocation and freeing
@@ -2785,8 +2916,9 @@ void __init kmem_cache_init(void)
 #else
  kmem_size = sizeof(struct kmem_cache);
 #endif
-
-
+#ifdef CONFIG_SLUB_NOTIFY
+ srcu_init_notifier_head(&slub_nb);
+#endif
  printk(KERN_INFO "SLUB: Genslabs=%d, HWalign=%d, Order=%d-%d, MinObjects=%d,"
   " CPUs=%d, Nodes=%d\n",
   caches, cache_line_size(),
```

Subject: [PATCH 3/5] Switch caches notification dynamically
Posted by Pavel Emelianov on Fri, 21 Sep 2007 09:22:03 GMT
View Forum Message <> Reply to Message

The /sys/slab/<name>/cache_notify attribute controls
whether the cache <name> is to be accounted or not.

For the reasons described before kmalloc caches cannot
be turned on.

By default no caches are accountable. Simply make
  # echo -n 1 > /sys/slab/<name>cache_notify
to turn notification of this cache on.

If we turn accounting on on some cache and this cache
is merged with some other, this "other" will be notified
as well. We can solve this by disabling of cache merging.

Turning the notification off is possible only when this
cache is empty. The reason for this is that the pages,
that are full of objects are not linked in any list, so
we wouldn't be able to walk these pages and notify others
that these objects are no longer tracked.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/mm/slub.c b/mm/slub.c
index ac4f157..b5af598 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -3775,6 +3917,36 @@ static ssize_t defrag_ratio_store(struct
 SLAB_ATTR(defrag_ratio);
 #endif

+static ssize_t cache_notify_show(struct kmem_cache *s, char *buf)
+{
+ return sprintf(buf, "%d\n", !!(s->flags & SLAB_NOTIFY));
+}
+
+static ssize_t cache_notify_store(struct kmem_cache *s,
+  const char *buf, size_t length)
+{
+ if (buf[0] == '1') {
+  s->flags |= SLAB_NOTIFY;
+  return length;
+ }
+
+ if (buf[0] == '0') {
+  if (any_slab_objects(s))
+   /*
+    * we cannot turn this off because of the
+    * full slabs cannot be found in this case
```

```
+    */
+   return -EBUSY;
+
+  s->flags &= ~SLAB_NOTIFY;
+  return length;
+ }
+
+ return -EINVAL;
+}
+
+SLAB_ATTR(cache_notify);
+
 static struct attribute * slab_attrs[] = {
  &slab_size_attr.attr,
  &object_size_attr.attr,
@@ -3805,6 +3984,7 @@ static struct attribute * slab_attrs[] =
 #ifdef CONFIG_NUMA
  &defrag_ratio_attr.attr,
 #endif
+ &cache_notify_attr.attr,
  NULL
 };
```

Subject: [PATCH 4/5] Setup the container
Posted by Pavel Emelianov on Fri, 21 Sep 2007 09:23:05 GMT
View Forum Message <> Reply to Message

Attach the controller to the containers. This will work
with the SLUB allocator only. However, if we need I can
port this on SLAB (and maybe SLOB ;)).

This setup is simple and stupid.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/include/linux/container_subsys.h b/include/linux/container_subsys.h
index 81d11c2..9dd90d9 100644
--- a/include/linux/container_subsys.h
+++ b/include/linux/container_subsys.h
@@ -36,3 +36,9 @@ SUBSYS(mem_container)
 #endif

 /* */
+
+#ifdef CONFIG_CONTAINER_KMEM
```

```
+SUBSYS(kmem)
+#endif
+
+/* */
diff --git a/init/Kconfig b/init/Kconfig
index 0bb211a..326fd55 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -353,6 +353,12 @@ config CONTAINER_MEM_CONT
    Provides a memory controller that manages both page cache and
    RSS memory.

+config CONTAINER_KMEM
+ bool "Kernel memory controller for containers"
+ depends on CONTAINERS && RESOURCE_COUNTERS && SLUB
+ help
+   Provides a kernel memory usage control for containers
+
 config PROC_PID_CPUSET
  bool "Include legacy /proc/<pid>/cpuset file"
  depends on CPUSETS
diff --git a/mm/Makefile b/mm/Makefile
index 6237dd6..1cb7e6d 100644
--- a/mm/Makefile
+++ b/mm/Makefile
@@ -31,4 +31,5 @@ obj-$(CONFIG_MIGRATION) += migrate.o
 obj-$(CONFIG_SMP) += allocpercpu.o
 obj-$(CONFIG_QUICKLIST) += quicklist.o
 obj-$(CONFIG_CONTAINER_MEM_CONT) += memcontrol.o
+obj-$(CONFIG_CONTAINER_KMEM) += kmemcontrol.o

diff --git a/mm/kmemcontrol.c b/mm/kmemcontrol.c
new file mode 100644
index 0000000..a2c45bb
--- /dev/null
+++ b/mm/kmemcontrol.c
@@ -0,0 +1,123 @@
+/*
+ * kmemcontrol.c - Kernel Memory Controller
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ * Author: Pavel Emelyanov <xemul@openvz.org>
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
```

```
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
+ * GNU General Public License for more details.
+ */
+
+#include <linux/mm.h>
+#include <linux/container.h>
+#include <linux/res_counter.h>
+#include <linux/err.h>
+
+struct kmem_container {
+ struct container_subsys_state css;
+ struct res_counter res;
+};
+
+static inline
+struct kmem_container *css_to_kmem(struct container_subsys_state *css)
+{
+ return container_of(css, struct kmem_container, css);
+}
+
+static inline
+struct kmem_container *container_to_kmem(struct container *cont)
+{
+ return css_to_kmem(container_subsys_state(cont, kmem_subsys_id));
+}
+
+static inline
+struct kmem_container *task_kmem_container(struct task_struct *tsk)
+{
+ return css_to_kmem(task_subsys_state(tsk, kmem_subsys_id));
+}
+
+/*
+ * containers interface
+ */
+
+static struct kmem_container init_kmem_container;
+
+static struct container_subsys_state *kmem_create(struct container_subsys *ss,
+  struct container *container)
+{
+ struct kmem_container *mem;
+
+ if (unlikely((container->parent) == NULL))
+  mem = &init_kmem_container;
+ else
```

```
+  mem = kzalloc(sizeof(struct kmem_container), GFP_KERNEL);
+
+ if (mem == NULL)
+  return ERR_PTR(-ENOMEM);
+
+ res_counter_init(&mem->res);
+ return &mem->css;
+
+}
+
+static void kmem_destroy(struct container_subsys *ss,
+  struct container *container)
+{
+ kfree(container_to_kmem(container));
+}
+
+static ssize_t kmem_container_read(struct container *cont, struct cftype *cft,
+  struct file *file, char __user *userbuf, size_t nbytes,
+  loff_t *ppos)
+{
+ return res_counter_read(&container_to_kmem(cont)->res,
+   cft->private, userbuf, nbytes, ppos, NULL);
+}
+
+static ssize_t kmem_container_write(struct container *cont, struct cftype *cft,
+  struct file *file, const char __user *userbuf,
+  size_t nbytes, loff_t *ppos)
+{
+ return res_counter_write(&container_to_kmem(cont)->res,
+   cft->private, userbuf, nbytes, ppos, NULL);
+}
+
+static struct cftype kmem_files[] = {
+ {
+  .name = "usage",
+  .private = RES_USAGE,
+  .read = kmem_container_read,
+ },
+ {
+  .name = "limit",
+  .private = RES_LIMIT,
+  .write = kmem_container_write,
+  .read = kmem_container_read,
+ },
+ {
+  .name = "failcnt",
+  .private = RES_FAILCNT,
+  .read = kmem_container_read,
```

```
+ },
+};
+
+static int kmem_populate(struct container_subsys *ss, struct container *cnt)
+{
+ return container_add_files(cnt, ss, kmem_files, ARRAY_SIZE(kmem_files));
+}
+
+struct container_subsys kmem_subsys = {
+ .name = "kmem",
+ .create = kmem_create,
+ .destroy  = kmem_destroy,
+ .populate = kmem_populate,
+ .subsys_id = kmem_subsys_id,
+ .early_init = 1,
+};
```

Subject: [PATCH 5/5] Account for the slub objects
Posted by Pavel Emelianov on Fri, 21 Sep 2007 09:24:20 GMT
View Forum Message <> Reply to Message

The struct page gets an extra pointer (just like it has with
the RSS controller) and this pointer points to the array of
the kmem_container pointers - one for each object stored on
that page itself.

Thus the i'th object on the page is accounted to the container
pointed by the i'th pointer on that array and when the object
is freed we unaccount its size to this particular container,
not the container current task belongs to.

This is done so, because the context objects are freed is most
often not the same as the one this objects was allocated in
(due to RCU and reference counters).

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/include/linux/mm_types.h b/include/linux/mm_types.h
index 48df4b4..1a41901 100644
--- a/include/linux/mm_types.h
+++ b/include/linux/mm_types.h
@@ -83,9 +83,14 @@ struct page {
  void *virtual;   /* Kernel virtual address (NULL if
        not kmapped, ie. highmem) */
 #endif /* WANT_PAGE_VIRTUAL */
```

```
+ union {
 #ifdef CONFIG_CONTAINER_MEM_CONT
- unsigned long page_container;
+  unsigned long page_container;
+#endif
+#ifdef CONFIG_CONTAINER_KMEM
+  struct kmem_container **containers;
 #endif
+ };
 #ifdef CONFIG_PAGE_OWNER
  int order;
  unsigned int gfp_mask;
diff --git a/include/linux/slub_def.h b/include/linux/slub_def.h
index 40801e7..fd9a3d4 100644
--- a/include/linux/slub_def.h
+++ b/include/linux/slub_def.h
@@ -69,6 +69,8 @@ struct kmem_cache {
 #endif
 };

+int slab_index(void *p, struct kmem_cache *s, void *addr);
+
 /*
  * Kmalloc subsystem.
  */
diff --git a/mm/kmemcontrol.c b/mm/kmemcontrol.c
new file mode 100644
index 0000000..a2c45bb
--- /dev/null
+++ b/mm/kmemcontrol.c
@@ -13,6 +13,9 @@
  * but WITHOUT ANY WARRANTY; without even the implied warranty of
  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  * GNU General Public License for more details.
+ *
+ * Changelog:
+ *   2007 Pavel Emelyanov : Add slub accounting
  */

 #include <linux/mm.h>
@@ -121,3 +124,128 @@
  .subsys_id = kmem_subsys_id,
  .early_init = 1,
 };
+
+/*
+ * slub accounting
+ */
```

```
+
+int slub_newpage_notify(struct kmem_cache *s, struct page *pg, gfp_t flags)
+{
+ struct kmem_container **ptr;
+
+ ptr = kzalloc(s->objects * sizeof(struct kmem_container *), flags);
+ if (ptr == NULL)
+  return -ENOMEM;
+
+ pg->containers = ptr;
+ return 0;
+}
+
+void slub_freepage_notify(struct kmem_cache *s, struct page *pg)
+{
+ struct kmem_container **ptr;
+
+ ptr = pg->containers;
+ if (ptr == NULL)
+  return;
+
+ kfree(ptr);
+ pg->containers = NULL;
+}
+
+int slub_alloc_notify(struct kmem_cache *s, void *obj, gfp_t gfp)
+{
+ struct page *pg;
+ struct kmem_container *cnt;
+ struct kmem_container **obj_container;
+
+ pg = virt_to_head_page(obj);
+ obj_container = pg->containers;
+ if (unlikely(obj_container == NULL)) {
+  /*
+   * turned on after some objects were allocated
+   */
+  if (slub_newpage_notify(s, pg, gfp) < 0)
+   goto err;
+
+  obj_container = pg->containers;
+ }
+
+ rcu_read_lock();
+ cnt = task_kmem_container(current);
+ if (res_counter_charge(&cnt->res, s->size))
+  goto err_locked;
+
```

```
+ css_get(&cnt->css);
+ rcu_read_unlock();
+ obj_container[slab_index(obj, s, page_address(pg))] = cnt;
+ return 0;
+
+err_locked:
+ rcu_read_unlock();
+err:
+ return -ENOMEM;
+}
+
+void slub_free_notify(struct kmem_cache *s, void *obj)
+{
+ struct page *pg;
+ struct kmem_container *cnt;
+ struct kmem_container **obj_container;
+
+ pg = virt_to_head_page(obj);
+ obj_container = pg->containers;
+ if (obj_container == NULL)
+  return;
+
+ obj_container += slab_index(obj, s, page_address(pg));
+ cnt = *obj_container;
+ if (cnt == NULL)
+  return;
+
+ res_counter_uncharge(&cnt->res, s->size);
+ *obj_container = NULL;
+ css_put(&cnt->css);
+}
+
+#ifdef CONFIG_SLUB_NOTIFY
+static int kmem_notify(struct notifier_block *nb, unsigned long cmd, void *arg)
+{
+ int ret;
+ struct slub_notify_struct *ns;
+
+ ns = (struct slub_notify_struct *)arg;
+
+ switch (cmd) {
+ case SLUB_ALLOC:
+  ret = slub_alloc_notify(ns->cachep, ns->objp, ns->gfp);
+  break;
+ case SLUB_FREE:
+  ret = 0;
+  slub_free_notify(ns->cachep, ns->objp);
+  break;
```

```
+ case SLUB_NEWPAGE:
+  ret = slub_newpage_notify(ns->cachep, ns->objp, ns->gfp);
+  break;
+ case SLUB_FREEPAGE:
+  ret = 0;
+  slub_freepage_notify(ns->cachep, ns->objp);
+  break;
+ default:
+  return NOTIFY_DONE;
+ }
+
+ return (ret < 0) ? notifier_from_errno(ret) : NOTIFY_OK;
+}
+
+static struct notifier_block kmem_block = {
+ .notifier_call = kmem_notify,
+};
+
+static int kmem_subsys_register(void)
+{
+ return slub_register_notifier(&kmem_block);
+}
+
+__initcall(kmem_subsys_register);
+#endif
diff --git a/mm/slub.c b/mm/slub.c
index ac4f157..b5af598 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -327,7 +327,7 @@ static inline void set_freepointer(struc
  for (__p = (__free); __p; __p = get_freepointer((__s), __p))

 /* Determine object index from a given position */
-static inline int slab_index(void *p, struct kmem_cache *s, void *addr)
+inline int slab_index(void *p, struct kmem_cache *s, void *addr)
 {
  return (p - addr) / s->size;
 }
@@ -2360,6 +2483,14 @@ EXPORT_SYMBOL(kmem_cache_destroy);
 struct kmem_cache kmalloc_caches[PAGE_SHIFT] __cacheline_aligned;
 EXPORT_SYMBOL(kmalloc_caches);

+static inline int is_kmalloc_cache(struct kmem_cache *s)
+{
+ int km_idx;
+
+ km_idx = s - kmalloc_caches;
+ return km_idx >= 0 && km_idx < ARRAY_SIZE(kmalloc_caches);
```

```
+}
+
 #ifdef CONFIG_ZONE_DMA
 static struct kmem_cache *kmalloc_caches_dma[PAGE_SHIFT];
 #endif
@@ -2811,6 +2943,16 @@ static int slab_unmergeable(struct kmem_
  if (s->refcount < 0)
   return 1;

+#ifdef CONFIG_CONTAINER_KMEM
+ /*
+  * many caches that can be accountable are usually merged with
+  * kmalloc caches, which are disabled for accounting for a while
+  */
+
+ if (is_kmalloc_cache(s))
+  return 1;
+#endif
+
  return 0;
 }

@@ -3775,6 +3917,13 @@ static ssize_t defrag_ratio_store(struct
   const char *buf, size_t length)
 {
  if (buf[0] == '1') {
+#ifdef CONFIG_CONTAINER_KMEM
+  if (is_kmalloc_cache(s))
+   /*
+    * cannot just make these caches accountable
+    */
+   return -EINVAL;
+#endif
   s->flags |= SLAB_NOTIFY;
   return length;
  }
```

Subject: Re: [PATCH 0/5] Kernel memory accounting container (v4)
Posted by Balbir Singh on Fri, 21 Sep 2007 12:23:06 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> Long time ago we decided to start memory control with the
> user memory container. Now this container in -mm tree and
> I think we can start with (at least discussion of) the
> kmem one.
>

Hi, Pavel,

Yes, I remember the entire discussion. Also in the TODO
list is an mlock() controller and a virtual memory
controller. Just wanted to ensure that they are still
in the TODO list.

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

Subject: Re: [PATCH 1/4] Add notification about some major slab events
Posted by Christoph Lameter on Mon, 24 Sep 2007 21:05:07 GMT
View Forum Message <> Reply to Message

On Fri, 21 Sep 2007, Pavel Emelyanov wrote:

> @@ -1486,7 +1597,7 @@ load_freelist:
> object = c->page->freelist;
> if (unlikely(!object))
> goto another_slab;
> - if (unlikely(SlabDebug(c->page)))
> + if (unlikely(SlabDebug(c->page)) || (s->flags & SLAB_NOTIFY))
> goto debug;

There is no need to check s->flags & SLAB_NOTIFY here. Just make sure that
SlabDebug() is always set on allocated slabs. Set SlabDebug when a slab
page is allocated is sufficient.

> debug:
> object = c->page->freelist;
> - if (!alloc_debug_processing(s, c->page, object, addr))
> + if (SlabDebug(c->page) &&
> + !alloc_debug_processing(s, c->page, object, addr))
> goto another_slab;
>

SlabDebug will always be set and does not need to be tested.

> + if ((s->flags & SLAB_NOTIFY) &&
> + slub_alloc_notify(s, object, gfpflags) < 0) {
> + object = NULL;
> + goto out;
> + }

Looks like the above piece is the only addition we need in __slab_alloc().

> @@ -1620,7 +1739,7 @@ static void __slab_free(struct kmem_cach
>
> slab_lock(page);
>
> - if (unlikely(SlabDebug(page)))
> + if (unlikely(SlabDebug(page)) || (s->flags & SLAB_NOTIFY))
> goto debug;

Not needed either.

> checks_ok:
> prior = object[offset] = page->freelist;
> @@ -1657,8 +1776,12 @@ slab_empty:
> return;
>
> debug:
> - if (!free_debug_processing(s, page, x, addr))
> + if (SlabDebug(page) && !free_debug_processing(s, page, x, addr))
> goto out_unlock;

Ditto.

> +
> + if (s->flags & SLAB_NOTIFY)
> + slub_free_notify(s, x);
> +

Yup we need this for __slab_free.

Subject: Re: [PATCH 3/5] Switch caches notification dynamically
Posted by Christoph Lameter on Mon, 24 Sep 2007 21:07:53 GMT
View Forum Message <> Reply to Message

On Fri, 21 Sep 2007, Pavel Emelyanov wrote:

> The /sys/slab/<name>/cache_notify attribute controls
> whether the cache <name> is to be accounted or not.
>
> For the reasons described before kmalloc caches cannot
> be turned on.

It looks like the patch is forbidding to turn the notification off? On is
okay even if there are already objects present? Full slabs may exist that
cannot be accounted for now. I guess that is okay because we only want

to track new allocations?

Subject: Re: [PATCH 3/5] Switch caches notification dynamically
Posted by Pavel Emelianov on Tue, 25 Sep 2007 07:46:44 GMT
View Forum Message <> Reply to Message

Christoph Lameter wrote:
> On Fri, 21 Sep 2007, Pavel Emelyanov wrote:
>
>> The /sys/slab/<name>/cache_notify attribute controls
>> whether the cache <name> is to be accounted or not.
>>
>> For the reasons described before kmalloc caches cannot
>> be turned on.
>
> It looks like the patch is forbidding to turn the notification off? On is
> okay even if there are already objects present?

Yup. If we "on" the notifications we will just receive new events.

> Full slabs may exist that cannot be accounted for now. I guess that is
> okay because we only want to track new allocations?

If we turn notifications off with full slabs in memory we won't be
able to (in case of kmem accounting) release the page's metadata that
the slub_newpage event attached to them.

We can make this by tracking all the full slabs, and at the moment of
"off" walk them and generate slab_free and slab_release_page events,
but does it worth doing it ever?

Thanks,
Pavel

Subject: Re: [PATCH 2/5] Generic notifiers for SLUB events
Posted by Pavel Emelianov on Mon, 01 Oct 2007 13:07:44 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelyanov wrote:
>> If the user wants more than one listener for SLUB event,
>> it can register them all as notifier_block-s so all of
>> them will be notified.
>>

>> This costs us about 10% of performance loss, in comparison
>> with static linking.
>>
>> The selected method of notification is srcu notifier blocks.
>> This is selected because the "call" path, i.e. when the
>> notification is done, is lockless and at the same time the
>> handlers can sleep. Neither regular nor atomic notifiers
>> provide such facilities.
>>
>> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>>
>> ---
>>
>> diff --git a/init/Kconfig b/init/Kconfig
>> index 684ccfb..e9acc29 100644
>> --- a/init/Kconfig
>> +++ b/init/Kconfig
>> @@ -593,6 +599,16 @@ config SLUB_DEBUG
>>   SLUB sysfs support. /sys/slab will not exist and there will be
>>   no support for cache validation etc.
>>
>> +config SLUB_NOTIFY
>> + default y
>
> Should the default be on? Shouldn't it depend on KMEM?

Well... I think that is should be N by default and has
noting to do with the KMEM :) Thanks for noticing.

>> + bool "Enable SLUB events generic notification"
>> + depends on SLUB
>> + help
>> +   When Y, this option enables generic notifications of some major
>> +   slub events. However, if you do know that there will be the
>> +   only listener for them, you may say N here, so that callbacks
>> +   will be called directly.
>> +
>> -
>> -
>> +#ifdef CONFIG_SLUB_NOTIFY
>> + srcu_init_notifier_head(&slub_nb);
>
> Can we get rid of the #ifdef CONFIG_SLUB_NOTIFY?

I don't think this is really useful in the __init code :)

>> +#endif
>>   printk(KERN_INFO "SLUB: Genslabs=%d, HWalign=%d, Order=%d-%d, MinObjects=%d,"

>> " CPUs=%d, Nodes=%d\n",
>> caches, cache_line_size(),
>>
>
>

Subject: Re: [PATCH 2/5] Generic notifiers for SLUB events
Posted by Christoph Lameter on Mon, 01 Oct 2007 20:39:09 GMT
View Forum Message <> Reply to Message

On Mon, 1 Oct 2007, Pavel Emelyanov wrote:

> > Should the default be on? Shouldn't it depend on KMEM?
>
> Well... I think that is should be N by default and has
> noting to do with the KMEM :) Thanks for noticing.

Right.

> >> +#ifdef CONFIG_SLUB_NOTIFY
> >> + srcu_init_notifier_head(&slub_nb);
> >
> > Can we get rid of the #ifdef CONFIG_SLUB_NOTIFY?
>
> I don't think this is really useful in the __init code :)

You could define some macros that avoid the #ifdefs?
