Subject: [PATCH 0/5] Kernel memory accounting container (v5)
Posted by Pavel Emelianov on Tue, 25 Sep 2007 14:16:33 GMT
View Forum Message <> Reply to Message

```
Changes since v.4:
* make SLAB_NOTIFY caches mark pages as SlabDebug. That
  makes the interesting paths simpler (thanks to Christoph);
* the change above caused appropriate changes in "turn
  notifications on" path - all available pages must become
  SlabDebug and page's freelists must be flushed;
* added two more events - "on" and "off" to make kmalloc
  caches disabling more gracefully;
* turning notifications "off" is marked as "TODO". Right
  now it's hard w/o massive rework of slub.c in respect to
  full slabs handling.

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".

===============================================================
```

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 the kmem one.

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 control groups 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 marking of all the slabs in the cache as not-debug, but
due to full-pages in slub are not stored in any lists (usually)
this is impossible to do so, however this is in todo list.

Thanks,
Pavel

Subject: [PATCH 1/5] Add notification about some major slab events
Posted by Pavel Emelianov on Tue, 25 Sep 2007 14:18:00 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/include/linux/slub_def.h b/include/linux/slub_def.h
index 40801e7..8cfd9ff 100644
--- a/include/linux/slub_def.h
+++ b/include/linux/slub_def.h
@@ -200,4 +203,22 @@ static __always_inline void *kmalloc_nod
 }
 #endif
```

```
+struct slub_notify_struct {
+ struct kmem_cache *cachep;
+ void *objp;
+ gfp_t gfp;
+};
+
+enum {
+ SLUB_ON,
+ SLUB_OFF,
+ 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 31d04a3..e066a0e 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -1040,6 +1040,43 @@ static inline unsigned long kmem_cache_f
 }
 #define slub_debug 0
 #endif
+
+/*
+ * notifiers
+ */
+
+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)
+{
+}
+
+int __attribute__ ((weak)) slub_on_notify(struct kmem_cache *cachep)
+{
+ return 0;
+}
+
+int __attribute__ ((weak)) slub_off_notify(struct kmem_cache *cachep)
+{
+ return 0;
+}
+
 /*
  * Slab allocation and freeing
  */
@@ -1063,7 +1184,11 @@ 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 +1196,11 @@ 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,
@@ -1103,7 +1233,7 @@ static struct page *new_slab(struct kmem
  page->slab = s;
  page->flags |= 1 << PG_slab;
  if (s->flags & (SLAB_DEBUG_FREE | SLAB_RED_ZONE | SLAB_POISON |
-   SLAB_STORE_USER | SLAB_TRACE))
+   SLAB_STORE_USER | SLAB_TRACE | SLAB_NOTIFY))
   SetSlabDebug(page);

  start = page_address(page);
@@ -1158,6 +1288,9 @@ 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
@@ -1548,9 +1681,16 @@ debug:
  if (!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;
 }
@@ -1659,6 +1799,10 @@ slab_empty:
 debug:
  if (!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 Tue, 25 Sep 2007 14:19:05 GMT

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
+ 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/mm/slub.c b/mm/slub.c
index 31d04a3..e066a0e 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -1040,6 +1040,89 @@ static inline unsigned long kmem_cache_f
  * notifiers
  */

+#ifdef CONFIG_SLUB_NOTIFY
```

```
+static struct srcu_notifier_head slub_nb;
+
+static noinline
+int __slub_notify_fail(int cmd_alloc, int cmd_free, struct kmem_cache *cachep,
+  void *obj, gfp_t gfp)
+{
+ int ret, called = 0;
+ 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 && called)
+  __srcu_notifier_call_chain(&slub_nb, cmd_free, &arg,
+    called - 1, NULL);
+
+ return ret;
+}
+
+static noinline
+void __slub_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_notify_fail(SLUB_ALLOC, SLUB_FREE, cachep, obj, gfp);
+}
+
+static inline
+void slub_free_notify(struct kmem_cache *cachep, void *obj)
+{
+ __slub_notify(SLUB_FREE, cachep, obj);
+}
+
+static inline
+int slub_newpage_notify(struct kmem_cache *cachep, struct page *pg, gfp_t gfp)
+{
+ return __slub_notify_fail(SLUB_NEWPAGE, SLUB_FREEPAGE, cachep, pg, gfp);
+}
+
+static inline
+void slub_freepage_notify(struct kmem_cache *cachep, struct page *pg)
+{
+ __slub_notify(SLUB_FREEPAGE, cachep, pg);
+}
+
+static inline
+int slub_on_notify(struct kmem_cache *cachep)
+{
+ return __slub_notify_fail(SLUB_ON, SLUB_OFF, cachep, NULL, GFP_KERNEL);
+}
+
+static inline
+int slub_off_notify(struct kmem_cache *cachep)
+{
+ return __slub_notify_fail(SLUB_OFF, SLUB_ON, cachep, NULL, GFP_KERNEL);
+}
+#else
 int __attribute__ ((weak)) slub_alloc_notify(struct kmem_cache *cachep,
   void *obj, gfp_t gfp)
 {
@@ -1040,6 +1150,7 @@ static inline unsigned long kmem_cache_f
 {
  return 0;
 }
+#endif

 /*
  * Slab allocation and freeing
@@ -2785,8 +2937,9 @@ 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 Tue, 25 Sep 2007 14:22:51 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.

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. One can solve this by disabling of cache merging,
i.e. booting with slub_nomerge option to the kernel.

Turning the notifications "on" causes all te subsequent
allocations use the slow allocation path, so all the
per-cpu caches are flushed and all the partial pages
are marked as SlabDebug.

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 tell them
thay should not produce notifications any longer.

To make "off" possible we need to rework the slub.c in
respect to full slabs handling (this is currently
available with SLUB_DEBUG only). This is in TODO.

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

```
diff --git a/mm/slub.c b/mm/slub.c
index 31d04a3..e066a0e 100644
```

```
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -3779,6 +3932,60 @@ 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)
+{
+ int err;
+ int n;
+
+ if ((buf[0] == '1') && !(s->flags & SLAB_NOTIFY)) {
+  err = slub_on_notify(s);
+  if (err < 0)
+   return err;
+
+  s->flags |= SLAB_NOTIFY;
+
+  flush_all(s);
+  for_each_node_state(n, N_NORMAL_MEMORY) {
+   struct kmem_cache_node *node;
+   struct page *pg;
+
+   node = get_node(s, n);
+   spin_lock_irq(&node->list_lock);
+   list_for_each_entry(pg, &node->partial, lru)
+    SetSlabDebug(pg);
+   spin_unlock_irq(&node->list_lock);
+  }
+  return length;
+ }
+
+ if ((buf[0] == '0') && (s->flags & SLAB_NOTIFY)) {
+  /*
+   * TODO: make the notifications-off work
+   */
+  if (any_slab_objects(s))
+   return -EBUSY;
+
+  s->flags &= ~SLAB_NOTIFY;
+  err = slub_off_notify(s);
+  if (err < 0) {
```

```
+   s->flags |= SLAB_NOTIFY;
+   return err;
+  }
+
+  return length;
+ }
+
+ return -EINVAL;
+}
+
+SLAB_ATTR(cache_notify);
+
 static struct attribute * slab_attrs[] = {
  &slab_size_attr.attr,
  &object_size_attr.attr,
@@ -3809,6 +4016,7 @@ static struct attribute * slab_attrs[] =
 #ifdef CONFIG_NUMA
  &defrag_ratio_attr.attr,
 #endif
+ &cache_notify_attr.attr,
  NULL
 };
```

Subject: [PATCH 4/5] Setup the control group
Posted by Pavel Emelianov on Tue, 25 Sep 2007 14:24:30 GMT

View Forum Message <> Reply to Message

Attach the controller to the control groups. 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/cgroup_subsys.h b/include/linux/cgroup_subsys.h
index d822977..bd0f493 100644
--- a/include/linux/cgroup_subsys.h
+++ b/include/linux/cgroup_subsys.h
@@ -36,3 +36,9 @@ SUBSYS(mem_cgroup)
 #endif

 /* */
+
+#ifdef CONFIG_CGROUP_KMEM
```

```
+SUBSYS(kmem)
+#endif
+
+/* */
diff --git a/init/Kconfig b/init/Kconfig
index 684ccfb..e9acc29 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -353,6 +353,12 @@ config CGROUP_MEM_CONT
    Provides a memory controller that manages both page cache and
    RSS memory.

+config CGROUP_KMEM
+ bool "Kernel memory controller for containers"
+ depends on CGROUPS && 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/kmemcontrol.c b/mm/kmemcontrol.c
new file mode 100644
index 0000000..5698c0f
--- /dev/null
+++ b/mm/kmemcontrol.c
@@ -0,0 +1,119 @@
+/*
+ * 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/cgroup.h>
+#include <linux/res_counter.h>
+#include <linux/err.h>
```

```
+
+struct kmem_container {
+ struct cgroup_subsys_state css;
+ struct res_counter res;
+};
+
+static inline struct kmem_container *css_to_kmem(struct cgroup_subsys_state *ss)
+{
+ return container_of(ss, struct kmem_container, css);
+}
+
+static inline struct kmem_container *cgroup_to_kmem(struct cgroup *cg)
+{
+ return css_to_kmem(cgroup_subsys_state(cg, kmem_subsys_id));
+}
+
+static inline struct kmem_container *task_kmem_container(struct task_struct *t)
+{
+ return css_to_kmem(task_subsys_state(t, kmem_subsys_id));
+}
+
+/*
+ * cgroups interface
+ */
+
+static struct kmem_container init_kmem_container;
+
+static struct cgroup_subsys_state *kmem_create(struct cgroup_subsys *ss,
+  struct cgroup *cgroup)
+{
+ struct kmem_container *mem;
+
+ if (unlikely((cgroup->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 cgroup_subsys *ss, struct cgroup *cgroup)
+{
+ kfree(cgroup_to_kmem(cgroup));
```

```
+}
+
+static ssize_t kmem_container_read(struct cgroup *cg, struct cftype *cft,
+  struct file *file, char __user *userbuf, size_t nbytes,
+  loff_t *ppos)
+{
+ return res_counter_read(&cgroup_to_kmem(cg)->res,
+   cft->private, userbuf, nbytes, ppos, NULL);
+}
+
+static ssize_t kmem_container_write(struct cgroup *cg, struct cftype *cft,
+  struct file *file, const char __user *userbuf,
+  size_t nbytes, loff_t *ppos)
+{
+ return res_counter_write(&cgroup_to_kmem(cg)->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 cgroup_subsys *ss, struct cgroup *cnt)
+{
+ return cgroup_add_files(cnt, ss, kmem_files, ARRAY_SIZE(kmem_files));
+}
+
+struct cgroup_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 Tue, 25 Sep 2007 14:26:18 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).

This controller disables the kmalloc caches accounting, since
we cannot just track all the kmalloc calls, but we need to
explicitly specify which kmalloc do we want to account for
with (e.g.) additional GFP flag.

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

```
diff --git a/include/linux/mm_types.h b/include/linux/mm_types.h
index f83bd17..8e1ef21 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_CGROUP_MEM_CONT
- unsigned long page_cgroup;
+  unsigned long page_cgroup;
+#endif
+#ifdef CONFIG_CGROUP_KMEM
+  struct kmem_container **cgroups;
 #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..8cfd9ff 100644
--- a/include/linux/slub_def.h
+++ b/include/linux/slub_def.h
@@ -69,6 +69,9 @@ struct kmem_cache {
 #endif
 };

+int slab_index(void *p, struct kmem_cache *s, void *addr);
+int is_kmalloc_cache(struct kmem_cache *s);
+
 /*
  * Kmalloc subsystem.
  */
diff --git a/mm/Makefile b/mm/Makefile
index 81232a1..f7ec4b7 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_CGROUP_MEM_CONT) += memcontrol.o
+obj-$(CONFIG_CGROUP_KMEM) += kmemcontrol.o

diff --git a/mm/kmemcontrol.c b/mm/kmemcontrol.c
new file mode 100644
index 0000000..5698c0f
--- /dev/null
+++ b/mm/kmemcontrol.c
@@ -1,6 +1,9 @@
  * 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>
@@ -120,3 +129,144 @@
  .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->cgroups = ptr;
+ return 0;
+}
+
+void slub_freepage_notify(struct kmem_cache *s, struct page *pg)
+{
+ struct kmem_container **ptr;
+
+ ptr = pg->cgroups;
+ if (ptr == NULL)
+  return;
+
+ kfree(ptr);
+ pg->cgroups = 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->cgroups;
+ if (unlikely(obj_container == NULL)) {
+  /*
+   * turned on after some objects were allocated
+   */
+  if (slub_newpage_notify(s, pg, GFP_ATOMIC) < 0)
+   goto err;
+
+  obj_container = pg->cgroups;
+ }
+
+ 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->cgroups;
+ 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);
+}
+
+int slub_on_notify(struct kmem_cache *cachep)
+{
+ return (is_kmalloc_cache(cachep) ? -EINVAL : 0);
+}
+
+int slub_off_notify(struct kmem_cache *cachep)
+{
+ return 0;
+}
+
+#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;
+ case SLUB_ON:
+  ret = slub_on_notify(ns->cachep);
+  break;
+ case SLUB_OFF:
+  ret = slub_off_notify(ns->cachep);
+  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 31d04a3..e066a0e 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 +2504,14 @@ EXPORT_SYMBOL(kmem_cache_destroy);
 struct kmem_cache kmalloc_caches[PAGE_SHIFT] __cacheline_aligned;
 EXPORT_SYMBOL(kmalloc_caches);

+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
```

Subject: Re: [PATCH 1/5] Add notification about some major slab events
Posted by Christoph Lameter on Tue, 25 Sep 2007 21:47:00 GMT

View Forum Message <> Reply to Message

On Tue, 25 Sep 2007, Pavel Emelyanov wrote:

```
> @@ -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 */
>
```

You need to add SLAB_NOTIFY to the SLUB_NEVER_MERGE group. That way you can solve the merging issue.

Otherwise this looks okay now.

Subject: Re: [PATCH 3/5] Switch caches notification dynamically
Posted by Christoph Lameter on Tue, 25 Sep 2007 21:48:59 GMT

View Forum Message <> Reply to Message

On Tue, 25 Sep 2007, Pavel Emelyanov wrote:

```
> +  for_each_node_state(n, N_NORMAL_MEMORY) {
> +   struct kmem_cache_node *node;
> +   struct page *pg;
> +
> +   node = get_node(s, n);
> +   spin_lock_irq(&node->list_lock);
> +   list_for_each_entry(pg, &node->partial, lru)
> +    SetSlabDebug(pg);
> +   spin_unlock_irq(&node->list_lock);
```

Is it necessary to mark all the existing slabs with SLAB_DEBUG? Would it
not be sufficient to just mark the new ones coming up? The above operation
could be a bit expensive.

Subject: Re: [PATCH 1/5] Add notification about some major slab events
Posted by Pavel Emelianov on Wed, 26 Sep 2007 09:37:31 GMT

View Forum Message <> Reply to Message

Christoph Lameter wrote:
> On Tue, 25 Sep 2007, Pavel Emelyanov wrote:
>

```
>> @@ -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 */
>>
```

>
> You need to add SLAB_NOTIFY to the SLUB_NEVER_MERGE group. That way you
> can solve the merging issue.

True, but we mark the slubs as notifyable at runtime, after they
are merged. However, once someone decides to make his slab notifyable
from the very beginning this makes sense, thanks.

> Otherwise this looks okay now.

Oh, great! Does this mean I can put Acked-by when sending this to Andrew?

Thanks,
Pavel

Subject: Re: [PATCH 3/5] Switch caches notification dynamically
Posted by Pavel Emelianov on Wed, 26 Sep 2007 09:39:21 GMT
View Forum Message <> Reply to Message

Christoph Lameter wrote:
> On Tue, 25 Sep 2007, Pavel Emelyanov wrote:
>
```
>> +  for_each_node_state(n, N_NORMAL_MEMORY) {
>> +   struct kmem_cache_node *node;
>> +   struct page *pg;
>> +
>> +   node = get_node(s, n);
>> +   spin_lock_irq(&node->list_lock);
>> +   list_for_each_entry(pg, &node->partial, lru)
>> +    SetSlabDebug(pg);
>> +   spin_unlock_irq(&node->list_lock);
```
>
>
> Is it necessary to mark all the existing slabs with SLAB_DEBUG? Would it

Yup. Otherwise we can never receive a single event e.g. if we make
alloc/free in a loop, or similar, so that new slabs simply are not
created.

> not be sufficient to just mark the new ones coming up? The above operation
> could be a bit expensive.

Oh, that's OK from my POV - switching slabs mode was never supposed to
be a fast path.

Thanks,
Pavel

Subject: Re: [PATCH 3/5] Switch caches notification dynamically
Posted by Christoph Lameter on Wed, 26 Sep 2007 17:30:18 GMT
View Forum Message <> Reply to Message

On Wed, 26 Sep 2007, Pavel Emelyanov wrote:

> > Is it necessary to mark all the existing slabs with SLAB_DEBUG? Would it
>
> Yup. Otherwise we can never receive a single event e.g. if we make
> alloc/free in a loop, or similar, so that new slabs simply are not
> created.

Right but on the other hand: If objects in these slabs are freed then your
counters get decremented and may go negative.

Subject: Re: [PATCH 1/5] Add notification about some major slab events
Posted by Christoph Lameter on Wed, 26 Sep 2007 17:31:08 GMT
View Forum Message <> Reply to Message

On Wed, 26 Sep 2007, Pavel Emelyanov wrote:

> True, but we mark the slubs as notifyable at runtime, after they
> are merged. However, once someone decides to make his slab notifyable
> from the very beginning this makes sense, thanks.

This also makes sense if a device driver later creates a new slab. Adding
the bit will switch off additional merges for that slab.

> Oh, great! Does this mean I can put Acked-by when sending this to Andrew?

Yes.

Subject: Re: [PATCH 3/5] Switch caches notification dynamically
Posted by Pavel Emelianov on Thu, 27 Sep 2007 08:24:09 GMT
View Forum Message <> Reply to Message

Christoph Lameter wrote:
> On Wed, 26 Sep 2007, Pavel Emelyanov wrote:
>
>>> Is it necessary to mark all the existing slabs with SLAB_DEBUG? Would it
>> Yup. Otherwise we can never receive a single event e.g. if we make
>> alloc/free in a loop, or similar, so that new slabs simply are not
>> created.
>
> Right but on the other hand: If objects in these slabs are freed then your
> counters get decremented and may go negative.

Nope :) I store a pointer on each accounted object, so if I have
a notification about freeing of not-accounted object I will not
decrement my counter ;)

I think it's too hard (if possible at all) to handle at generic level
whether we have notified this particular object allocation to some
listener or not, so I'd better leave this for the listener to handle
such a case.

Thanks,
Pavel

Subject: Re: [PATCH 1/5] Add notification about some major slab events

Posted by Pavel Emelianov on Thu, 27 Sep 2007 08:25:26 GMT
View Forum Message <> Reply to Message

Christoph Lameter wrote:
> On Wed, 26 Sep 2007, Pavel Emelyanov wrote:
>
>> True, but we mark the slubs as notifyable at runtime, after they
>> are merged. However, once someone decides to make his slab notifyable
>> from the very beginning this makes sense, thanks.
>
> This also makes sense if a device driver later creates a new slab. Adding
> the bit will switch off additional merges for that slab.

True. I will fix it. Thanks.

>> Oh, great! Does this mean I can put Acked-by when sending this to Andrew?
>
> Yes.

Great! Thanks a lot!

Pavel

Subject: Re: [PATCH 1/5] Add notification about some major slab events
Posted by Balbir Singh on Mon, 01 Oct 2007 11:55:39 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> 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.
>

Please check that weak objects work across platforms. Please see
http://www.ussg.iu.edu/hypermail/linux/kernel/0706.0/0593.html

> 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/include/linux/slub_def.h b/include/linux/slub_def.h
> index 40801e7..8cfd9ff 100644
> --- a/include/linux/slub_def.h
> +++ b/include/linux/slub_def.h
> @@ -200,4 +203,22 @@ static __always_inline void *kmalloc_nod
>  }
>  #endif
>
> +struct slub_notify_struct {
> + struct kmem_cache *cachep;
> + void *objp;
> + gfp_t gfp;
> +};
> +
> +enum {
> + SLUB_ON,
> + SLUB_OFF,
> + 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 31d04a3..e066a0e 100644
> --- a/mm/slub.c
> +++ b/mm/slub.c
> @@ -1040,6 +1040,43 @@ static inline unsigned long kmem_cache_f
>  }
>  #define slub_debug 0
>  #endif
> +
> +/*
> + * notifiers
> + */
> +
> +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)
> +{
> +}
> +
> +int __attribute__ ((weak)) slub_on_notify(struct kmem_cache *cachep)
> +{
> + return 0;
> +}
> +
> +int __attribute__ ((weak)) slub_off_notify(struct kmem_cache *cachep)
> +{
> + return 0;
> +}
> +
>  /*
>   * Slab allocation and freeing
>   */
> @@ -1063,7 +1184,11 @@ 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 +1196,11 @@ static struct page *allocate_slab(struct
>    pages);
>
>   return page;
> +
> +out_free:
> + __free_pages(page, s->order);
```

allocate_slab fails if sub_newpage_notify() fails? Sounds a bit
harsh, hard to review since the definition of the above function
is not known.

```
> +out:
> + return NULL;
>  }
>
>  static void setup_object(struct kmem_cache *s, struct page *page,
> @@ -1103,7 +1233,7 @@ static struct page *new_slab(struct kmem
>   page->slab = s;
>   page->flags |= 1 << PG_slab;
>   if (s->flags & (SLAB_DEBUG_FREE | SLAB_RED_ZONE | SLAB_POISON |
> -   SLAB_STORE_USER | SLAB_TRACE))
> +   SLAB_STORE_USER | SLAB_TRACE | SLAB_NOTIFY))
>    SetSlabDebug(page);
>
>   start = page_address(page);
> @@ -1158,6 +1288,9 @@ 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
> @@ -1548,9 +1681,16 @@ debug:
>   if (!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;
>  }
> @@ -1659,6 +1799,10 @@ slab_empty:
>  debug:
>   if (!free_debug_processing(s, page, x, addr))
>    goto out_unlock;
> +
> + if (s->flags & SLAB_NOTIFY)
> +  slub_free_notify(s, x);
> +
>   goto checks_ok;
>  }
>
>
```

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

Subject: Re: [PATCH 1/5] Add notification about some major slab events
Posted by Pavel Emelianov on Mon, 01 Oct 2007 12:13:43 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelyanov wrote:
>> 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.
>>
>
> Please check that weak objects work across platforms. Please see
> http://www.ussg.iu.edu/hypermail/linux/kernel/0706.0/0593.html

OK.

>> 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/include/linux/slub_def.h b/include/linux/slub_def.h
>> index 40801e7..8cfd9ff 100644
>> --- a/include/linux/slub_def.h
>> +++ b/include/linux/slub_def.h
>> @@ -200,4 +203,22 @@ static __always_inline void *kmalloc_nod
>>  }
>>  #endif
>>
>> +struct slub_notify_struct {
>> + struct kmem_cache *cachep;
>> + void *objp;
>> + gfp_t gfp;
>> +};
>> +
>> +enum {
>> + SLUB_ON,
>> + SLUB_OFF,
>> + 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 31d04a3..e066a0e 100644
>> --- a/mm/slub.c
>> +++ b/mm/slub.c
>> @@ -1040,6 +1040,43 @@ static inline unsigned long kmem_cache_f
>>  }
>>  #define slub_debug 0
>>  #endif
>> +
>> +/*
>> + * notifiers
>> + */
>> +
>> +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)
>> +{
>> +}
>> +
>> +int __attribute__ ((weak)) slub_on_notify(struct kmem_cache *cachep)
>> +{
>> + return 0;
>> +}
>> +
>> +int __attribute__ ((weak)) slub_off_notify(struct kmem_cache *cachep)
>> +{
>> + return 0;
>> +}
>> +
>>  /*
>>   * Slab allocation and freeing
>>   */
>> @@ -1063,7 +1184,11 @@ 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 +1196,11 @@ static struct page *allocate_slab(struct
>>    pages);
>>
>>   return page;
>> +
>> +out_free:
>> + __free_pages(page, s->order);
```

>
> allocate_slab fails if sub_newpage_notify() fails? Sounds a bit

Yup. If we try to allocate some meta info for this page and fail,
what shall we do? Fail only!

> harsh, hard to review since the definition of the above function
> is not known.

What do you mean by "not known"? I is declared above :)

```
>> +out:
>> + return NULL;
>>  }
>>
>>  static void setup_object(struct kmem_cache *s, struct page *page,
>> @@ -1103,7 +1233,7 @@ static struct page *new_slab(struct kmem
>>   page->slab = s;
>>   page->flags |= 1 << PG_slab;
>>   if (s->flags & (SLAB_DEBUG_FREE | SLAB_RED_ZONE | SLAB_POISON |
>> -   SLAB_STORE_USER | SLAB_TRACE))
>> +   SLAB_STORE_USER | SLAB_TRACE | SLAB_NOTIFY))
>>    SetSlabDebug(page);
>>
>>   start = page_address(page);
>> @@ -1158,6 +1288,9 @@ 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
>> @@ -1548,9 +1681,16 @@ debug:
>>   if (!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;
>>  }
>> @@ -1659,6 +1799,10 @@ slab_empty:
>>  debug:
>>   if (!free_debug_processing(s, page, x, addr))
>>    goto out_unlock;
>> +
>> + if (s->flags & SLAB_NOTIFY)
>> +  slub_free_notify(s, x);
>> +
>>   goto checks_ok;
>>  }
>>
>>
>
>
```

Subject: Re: [PATCH 1/5] Add notification about some major slab events
Posted by Balbir Singh on Mon, 01 Oct 2007 12:32:50 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
>> harsh, hard to review since the definition of the above function
>> is not known.
>
> What do you mean by "not known"? I is declared above :)
>

I meant, we only know the stub weak functions. Does slub_*_notify
fail only on fatal errors? If not, then the failure might be
unfair.

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

Subject: Re: [PATCH 1/5] Add notification about some major slab events
Posted by Pavel Emelianov on Mon, 01 Oct 2007 12:57:18 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:

> Pavel Emelyanov wrote:
>>> harsh, hard to review since the definition of the above function
>>> is not known.
>> What do you mean by "not known"? I is declared above :)
>>
>
> I meant, we only know the stub weak functions. Does slub_*_notify
> fail only on fatal errors? If not, then the failure might be

It depends on the listener of the event when to fail. The slub
notification mechanism is not the accounting at all it just
sends notifications.

> unfair.
>

Subject: Re: [PATCH 1/5] Add notification about some major slab events
Posted by Balbir Singh on Mon, 01 Oct 2007 13:03:27 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> Balbir Singh wrote:
>> Pavel Emelyanov wrote:
>>>> harsh, hard to review since the definition of the above function
>>>> is not known.
>>> What do you mean by "not known"? I is declared above :)
>>>
>> I meant, we only know the stub weak functions. Does slub_*_notify
>> fail only on fatal errors? If not, then the failure might be
>
> It depends on the listener of the event when to fail. The slub
> notification mechanism is not the accounting at all it just
> sends notifications.
>

OK. My concern is that a listener can fail allocation of a slub
object.

>> unfair.
>>
>

--
 Warm Regards,
 Balbir Singh
 Linux Technology Center

IBM, ISTL

Subject: Re: [PATCH 2/5] Generic notifiers for SLUB events
Posted by Balbir Singh on Mon, 01 Oct 2007 13:05:07 GMT
View Forum Message <> Reply to Message

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?

```
> + 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?

```
> +#endif
>   printk(KERN_INFO "SLUB: Genslabs=%d, HWalign=%d, Order=%d-%d, MinObjects=%d,"
>    " CPUs=%d, Nodes=%d\n",
>    caches, cache_line_size(),
>
```

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

Subject: Re: [PATCH 3/5] Switch caches notification dynamically
Posted by Balbir Singh on Mon, 01 Oct 2007 13:15:20 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> The /sys/slab/<name>/cache_notify attribute controls
> whether the cache <name> is to be accounted or not.
>
> 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. One can solve this by disabling of cache merging,
> i.e. booting with slub_nomerge option to the kernel.
>
> Turning the notifications "on" causes all te subsequent
> allocations use the slow allocation path, so all the
> per-cpu caches are flushed and all the partial pages
> are marked as SlabDebug.
>

Do we know of the overhead of slub notifications? It would
be nice to know and probably add it to the help text in
Kconfig.

> ---

```
>
> diff --git a/mm/slub.c b/mm/slub.c
> index 31d04a3..e066a0e 100644
> --- a/mm/slub.c
> +++ b/mm/slub.c
> @@ -3779,6 +3932,60 @@ 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)
> +{
> + int err;
> + int n;
> +
> + if ((buf[0] == '1') && !(s->flags & SLAB_NOTIFY)) {
```

Won't this code break if I pass 10 as input in buf?

```
> +  err = slub_on_notify(s);
> +  if (err < 0)
> +   return err;
> +
> +  s->flags |= SLAB_NOTIFY;
> +
> +  flush_all(s);
```

Sounds like an expensive operation, can we add a comment
to indicate the same. Also, in the documentation, could
we warn the user that turning notifications on/off
can be expensive?

```
> +  for_each_node_state(n, N_NORMAL_MEMORY) {
> +   struct kmem_cache_node *node;
> +   struct page *pg;
> +
> +   node = get_node(s, n);
> +   spin_lock_irq(&node->list_lock);
> +   list_for_each_entry(pg, &node->partial, lru)
> +    SetSlabDebug(pg);
> +   spin_unlock_irq(&node->list_lock);
> +  }
> +  return length;
> + }
> +
> + if ((buf[0] == '0') && (s->flags & SLAB_NOTIFY)) {
```

What happens if buf is 01?

```
> +  /*
> +   * TODO: make the notifications-off work
> +   */
> +  if (any_slab_objects(s))
> +   return -EBUSY;
> +
> +  s->flags &= ~SLAB_NOTIFY;
> +  err = slub_off_notify(s);
> +  if (err < 0) {
> +   s->flags |= SLAB_NOTIFY;
> +   return err;
> +  }
> +
> +  return length;
> + }
> +
> + return -EINVAL;
> +}
> +
> +SLAB_ATTR(cache_notify);
> +
>  static struct attribute * slab_attrs[] = {
>   &slab_size_attr.attr,
>   &object_size_attr.attr,
> @@ -3809,6 +4016,7 @@ static struct attribute * slab_attrs[] =
>  #ifdef CONFIG_NUMA
>   &defrag_ratio_attr.attr,
>  #endif
> + &cache_notify_attr.attr,
>   NULL
>  };
>
>
```

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

Subject: Re: [PATCH 3/5] Switch caches notification dynamically
Posted by Pavel Emelianov on Mon, 01 Oct 2007 13:19:39 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelyanov wrote:
>> The /sys/slab/<name>/cache_notify attribute controls
>> whether the cache <name> is to be accounted or not.
>>
>> 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. One can solve this by disabling of cache merging,
>> i.e. booting with slub_nomerge option to the kernel.
>>
>> Turning the notifications "on" causes all te subsequent
>> allocations use the slow allocation path, so all the
>> per-cpu caches are flushed and all the partial pages
>> are marked as SlabDebug.
>>
>
> Do we know of the overhead of slub notifications? It would
> be nice to know and probably add it to the help text in
> Kconfig.

When the notification is off the overhead is 0. When they are
on it depends on whether we use generic or direct notifications.

Roughly:

The generic notification costs us up to 10% of performance loss.
The direct notification costs us really small, providing the
listener is noop. If the listener is doing something - the performance
loss can be higher, but how much - depends on the listener :)

```
>
>> ---
>>
>> diff --git a/mm/slub.c b/mm/slub.c
>> index 31d04a3..e066a0e 100644
>> --- a/mm/slub.c
>> +++ b/mm/slub.c
>> @@ -3779,6 +3932,60 @@ 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)
>> +{
>> + int err;
>> + int n;
>> +
>> + if ((buf[0] == '1') && !(s->flags & SLAB_NOTIFY)) {
>
> Won't this code break if I pass 10 as input in buf?
>
>> +  err = slub_on_notify(s);
>> +  if (err < 0)
>> +   return err;
>> +
>> +  s->flags |= SLAB_NOTIFY;
>> +
>> +  flush_all(s);
>
> Sounds like an expensive operation, can we add a comment
> to indicate the same. Also, in the documentation, could
> we warn the user that turning notifications on/off
> can be expensive?
>
>> +  for_each_node_state(n, N_NORMAL_MEMORY) {
>> +   struct kmem_cache_node *node;
>> +   struct page *pg;
>> +
>> +   node = get_node(s, n);
>> +   spin_lock_irq(&node->list_lock);
>> +   list_for_each_entry(pg, &node->partial, lru)
>> +    SetSlabDebug(pg);
>> +   spin_unlock_irq(&node->list_lock);
>> +  }
>> +  return length;
>> + }
>> +
>> + if ((buf[0] == '0') && (s->flags & SLAB_NOTIFY)) {
>
> What happens if buf is 01?
>
>> +  /*
>> +   * TODO: make the notifications-off work
>> +   */
>> +  if (any_slab_objects(s))
>> +   return -EBUSY;
>> +
>> +  s->flags &= ~SLAB_NOTIFY;
>> +  err = slub_off_notify(s);
>> +  if (err < 0) {
>> +   s->flags |= SLAB_NOTIFY;
>> +   return err;
>> +  }
>> +
>> +  return length;
>> + }
>> +
>> + return -EINVAL;
>> +}
>> +
>> +SLAB_ATTR(cache_notify);
>> +
>>  static struct attribute * slab_attrs[] = {
>>   &slab_size_attr.attr,
>>   &object_size_attr.attr,
>> @@ -3809,6 +4016,7 @@ static struct attribute * slab_attrs[] =
>>  #ifdef CONFIG_NUMA
>>   &defrag_ratio_attr.attr,
>>  #endif
>> + &cache_notify_attr.attr,
>>   NULL
>>  };
>>
>>
>
>
```

Subject: Re: [PATCH 3/5] Switch caches notification dynamically
Posted by Pavel Emelianov on Mon, 01 Oct 2007 13:21:58 GMT
View Forum Message <> Reply to Message

```
>> diff --git a/mm/slub.c b/mm/slub.c
>> index 31d04a3..e066a0e 100644
>> --- a/mm/slub.c
>> +++ b/mm/slub.c
>> @@ -3779,6 +3932,60 @@ 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)
>> +{
>> + int err;
>> + int n;
>> +
>> + if ((buf[0] == '1') && !(s->flags & SLAB_NOTIFY)) {
>
> Won't this code break if I pass 10 as input in buf?
```

I don't care about this :) And no /sys/slab/<xxx>/ file does.
It's the problem of a user if he pass some shit and doesn't
see the notifications turned on/off ;)

```
>> +  err = slub_on_notify(s);
>> +  if (err < 0)
>> +   return err;
>> +
>> +  s->flags |= SLAB_NOTIFY;
>> +
>> +  flush_all(s);
>
> Sounds like an expensive operation, can we add a comment
> to indicate the same. Also, in the documentation, could
> we warn the user that turning notifications on/off
> can be expensive?
```

Sure.

```
>> +  for_each_node_state(n, N_NORMAL_MEMORY) {
>> +   struct kmem_cache_node *node;
>> +   struct page *pg;
>> +
>> +   node = get_node(s, n);
>> +   spin_lock_irq(&node->list_lock);
>> +   list_for_each_entry(pg, &node->partial, lru)
>> +    SetSlabDebug(pg);
>> +   spin_unlock_irq(&node->list_lock);
>> +  }
>> +  return length;
>> + }
>> +
>> + if ((buf[0] == '0') && (s->flags & SLAB_NOTIFY)) {
>
> What happens if buf is 01?
```

```
>
>> +  /*
>> +   * TODO: make the notifications-off work
>> +   */
>> +  if (any_slab_objects(s))
>> +   return -EBUSY;
>> +
>> +  s->flags &= ~SLAB_NOTIFY;
>> +  err = slub_off_notify(s);
>> +  if (err < 0) {
>> +   s->flags |= SLAB_NOTIFY;
>> +   return err;
>> +  }
>> +
>> +  return length;
>> + }
>> +
>> + return -EINVAL;
>> +}
>> +
>> +SLAB_ATTR(cache_notify);
>> +
>>  static struct attribute * slab_attrs[] = {
>>   &slab_size_attr.attr,
>>   &object_size_attr.attr,
>> @@ -3809,6 +4016,7 @@ static struct attribute * slab_attrs[] =
>>  #ifdef CONFIG_NUMA
>>   &defrag_ratio_attr.attr,
>>  #endif
>> + &cache_notify_attr.attr,
>>   NULL
>>  };
>>
>>
>
>
```

Subject: Re: [PATCH 3/5] Switch caches notification dynamically
Posted by Balbir Singh on Mon, 01 Oct 2007 13:38:35 GMT
View Forum Message <> Reply to Message

```
>>> + if ((buf[0] == '1') && !(s->flags & SLAB_NOTIFY)) {
>> Won't this code break if I pass 10 as input in buf?
>
> I don't care about this :) And no /sys/slab/<xxx>/ file does.
> It's the problem of a user if he pass some shit and doesn't
> see the notifications turned on/off ;)
```

>

Well, we don't want confused users? How does a user know what
is a valid value to pass? Are you saying that we don't need
to care about user interface, if so I don't agree.

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

Subject: Re: [PATCH 3/5] Switch caches notification dynamically
Posted by Pavel Emelianov on Mon, 01 Oct 2007 13:45:46 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
>>>> + if ((buf[0] == '1') && !(s->flags & SLAB_NOTIFY)) {
>>> Won't this code break if I pass 10 as input in buf?
>> I don't care about this :) And no /sys/slab/<xxx>/ file does.
>> It's the problem of a user if he pass some shit and doesn't
>> see the notifications turned on/off ;)
>>
>
> Well, we don't want confused users? How does a user know what
> is a valid value to pass? Are you saying that we don't need

The rules are simple - 1 to turn them on and 0 to turn them off
and nothing more. These values user sees when he "cat"s this file.

> to care about user interface, if so I don't agree.

That's the convention used in the /sys/slab/<x>/ files - the
1 means on the 0 means 0, all the others are not guaranteed :)

Thanks,
Pavel

Subject: Re: [PATCH 4/5] Setup the control group
Posted by Balbir Singh on Mon, 01 Oct 2007 13:48:28 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> Attach the controller to the control groups. 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.
>

That makes it easier for me to review it:)

```
> +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,
> + },
> +};
> +
```

Could we make the user interface similar to the one in the memory
controller please! It would make it easier for users to configure
and control both.

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

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

Subject: Re: [PATCH 4/5] Setup the control group
Posted by Pavel Emelianov on Mon, 01 Oct 2007 13:51:46 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelyanov wrote:
>> Attach the controller to the control groups. 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.
>>
>
> That makes it easier for me to review it:)

:)

```
>> +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,
>> + },
>> +};
>> +
```
>
> Could we make the user interface similar to the one in the memory
> controller please! It would make it easier for users to configure
> and control both.

Do you mean by merging the similar code or by making them look like

the same? If the second, than can you point at least one difference
except for "usage" vs "usage_in_bytes" (and similar) naming?

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

Subject: Re: [PATCH 5/5] Account for the slub objects
Posted by Balbir Singh on Mon, 01 Oct 2007 14:07:27 GMT

View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> 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).
>
> This controller disables the kmalloc caches accounting, since
> we cannot just track all the kmalloc calls, but we need to
> explicitly specify which kmalloc do we want to account for
> with (e.g.) additional GFP flag.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---

```
>
> diff --git a/include/linux/mm_types.h b/include/linux/mm_types.h
> index f83bd17..8e1ef21 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_CGROUP_MEM_CONT
> - unsigned long page_cgroup;
> +  unsigned long page_cgroup;
> +#endif
> +#ifdef CONFIG_CGROUP_KMEM
> +  struct kmem_container **cgroups;
>  #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..8cfd9ff 100644
> --- a/include/linux/slub_def.h
> +++ b/include/linux/slub_def.h
> @@ -69,6 +69,9 @@ struct kmem_cache {
>  #endif
>  };
>
> +int slab_index(void *p, struct kmem_cache *s, void *addr);
> +int is_kmalloc_cache(struct kmem_cache *s);
> +
>  /*
>   * Kmalloc subsystem.
>   */
> diff --git a/mm/Makefile b/mm/Makefile
> index 81232a1..f7ec4b7 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_CGROUP_MEM_CONT) += memcontrol.o
> +obj-$(CONFIG_CGROUP_KMEM) += kmemcontrol.o
>
> diff --git a/mm/kmemcontrol.c b/mm/kmemcontrol.c
> new file mode 100644
> index 0000000..5698c0f
> --- /dev/null
> +++ b/mm/kmemcontrol.c
> @@ -1,6 +1,9 @@
>   * 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>
> @@ -120,3 +129,144 @@
>   .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->cgroups = ptr;
> + return 0;
> +}
> +
> +void slub_freepage_notify(struct kmem_cache *s, struct page *pg)
> +{
> + struct kmem_container **ptr;
> +
> + ptr = pg->cgroups;
> + if (ptr == NULL)
> +  return;
> +
> + kfree(ptr);
> + pg->cgroups = 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->cgroups;
> + if (unlikely(obj_container == NULL)) {
> +  /*
> +   * turned on after some objects were allocated
> +   */
> +  if (slub_newpage_notify(s, pg, GFP_ATOMIC) < 0)
> +   goto err;
> +
> +  obj_container = pg->cgroups;
> + }
> +
> + rcu_read_lock();
> + cnt = task_kmem_container(current);
> + if (res_counter_charge(&cnt->res, s->size))
```

Is s->size measure in pages or bytes? I suspect it is bytes.

```
> +  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->cgroups;
> + 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);
> +}
> +
```

Quick check, slub_free_notify() and slab_alloc_notify() are called
from serialized contexts, right?

```
> +int slub_on_notify(struct kmem_cache *cachep)
> +{
> + return (is_kmalloc_cache(cachep) ? -EINVAL : 0);
> +}
> +
```

I know you've mentioned several times in the comments that
kmalloc slab's cannot be accounted for, could you please
add a big comment here as well.

Also, looks like if I turn on notification for kmalloc
slab's, the listener fails, which will cause all
allocations to fail?

```
> +int slub_off_notify(struct kmem_cache *cachep)
> +{
> + return 0;
> +}
> +
> +#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;
> + case SLUB_ON:
> +  ret = slub_on_notify(ns->cachep);
> +  break;
> + case SLUB_OFF:
> +  ret = slub_off_notify(ns->cachep);
> +  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 31d04a3..e066a0e 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 +2504,14 @@ EXPORT_SYMBOL(kmem_cache_destroy);
>  struct kmem_cache kmalloc_caches[PAGE_SHIFT] __cacheline_aligned;
>  EXPORT_SYMBOL(kmalloc_caches);
>
> +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);
> +}
> +
```

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

Subject: Re: [PATCH 5/5] Account for the slub objects
Posted by Pavel Emelianov on Mon, 01 Oct 2007 14:10:11 GMT

View Forum Message <> Reply to Message

[snip]

```
>> +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->cgroups;
>> + if (unlikely(obj_container == NULL)) {
>> +  /*
>> +   * turned on after some objects were allocated
>> +   */
>> +  if (slub_newpage_notify(s, pg, GFP_ATOMIC) < 0)
>> +   goto err;
>> +
>> +  obj_container = pg->cgroups;
>> + }
>> +
>> + rcu_read_lock();
>> + cnt = task_kmem_container(current);
>> + if (res_counter_charge(&cnt->res, s->size))
>
> Is s->size measure in pages or bytes? I suspect it is bytes.
```

In bytes, of course :) the struct anon_vma (for example) is difficult
to account in pages :P

```
>> +  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->cgroups;
>> + 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);
>> +}
>> +
>
```

> Quick check, slub_free_notify() and slab_alloc_notify() are called
> from serialized contexts, right?

Yup.

```
>> +int slub_on_notify(struct kmem_cache *cachep)
>> +{
>> + return (is_kmalloc_cache(cachep) ? -EINVAL : 0);
>> +}
```

>> +
>
> I know you've mentioned several times in the comments that
> kmalloc slab's cannot be accounted for, could you please
> add a big comment here as well.

This is useless comment - I will implement the kmalloc accounting
soon, so it will just go away.

> Also, looks like if I turn on notification for kmalloc

You cannot, but let's go on with this assumption :)

> slab's, the listener fails, which will cause all
> allocations to fail?

Nope, just this cache will not generate events and that's it.
____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 0/5] Kernel memory accounting container (v5)
Posted by Balbir Singh on Mon, 01 Oct 2007 14:12:59 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> Changes since v.4:
> * make SLAB_NOTIFY caches mark pages as SlabDebug. That
>   makes the interesting paths simpler (thanks to Christoph);
> * the change above caused appropriate changes in "turn
>   notifications on" path - all available pages must become
>   SlabDebug and page's freelists must be flushed;
> * added two more events - "on" and "off" to make kmalloc
>   caches disabling more gracefully;
> * turning notifications "off" is marked as "TODO". Right
>   now it's hard w/o massive rework of slub.c in respect to
>   full slabs handling.
>
> 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".
>

Hi, Pavel,

Overall, the patches look good, except for the comments
noted in the other patch postings. I am going to test
these patches and see how they hold out.

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

Subject: Re: [PATCH 3/5] Switch caches notification dynamically
Posted by Balbir Singh on Mon, 01 Oct 2007 14:14:40 GMT

View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> Balbir Singh wrote:
>>>>> + if ((buf[0] == '1') && !(s->flags & SLAB_NOTIFY)) {
>>>> Won't this code break if I pass 10 as input in buf?
>>> I don't care about this :) And no /sys/slab/<xxx>/ file does.
>>> It's the problem of a user if he pass some shit and doesn't
>>> see the notifications turned on/off ;)
>>>
>> Well, we don't want confused users? How does a user know what
>> is a valid value to pass? Are you saying that we don't need
>
> The rules are simple - 1 to turn them on and 0 to turn them off
> and nothing more. These values user sees when he "cat"s this file.
>

I think correct error reporting is critical, if the user sets
the value to 01 and that ends up disabling slab notifications,
the handling is wrong.

>> to care about user interface, if so I don't agree.
>
> That's the convention used in the /sys/slab/<x>/ files - the
> 1 means on the 0 means 0, all the others are not guaranteed :)
>

Is this documented somewhere or is this interpreted from looking
at the code of other file handlers?

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

Subject: Re: [PATCH 4/5] Setup the control group
Posted by Balbir Singh on Mon, 01 Oct 2007 14:16:26 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
>> Could we make the user interface similar to the one in the memory
>> controller please! It would make it easier for users to configure
>> and control both.
>
> Do you mean by merging the similar code or by making them look like
> the same? If the second, than can you point at least one difference
> except for "usage" vs "usage_in_bytes" (and similar) naming?
>

I meant similar naming, it makes it is easier if the naming convention
is the same.

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

Subject: Re: [PATCH 4/5] Setup the control group
Posted by Pavel Emelianov on Mon, 01 Oct 2007 14:17:13 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelyanov wrote:
>>> Could we make the user interface similar to the one in the memory
>>> controller please! It would make it easier for users to configure
>>> and control both.
>> Do you mean by merging the similar code or by making them look like
>> the same? If the second, than can you point at least one difference
>> except for "usage" vs "usage_in_bytes" (and similar) naming?
>>
>
> I meant similar naming, it makes it is easier if the naming convention
> is the same.

Ok. It looks like the naming in the memcontrol.c has changed after I
copied it from there :)

BTW, after Andrew accept these patches I think I will spend some time
thinking about how to consolidate the res_counter files.

Thanks,
Pavel

Subject: Re: [PATCH 4/5] Setup the control group
Posted by Balbir Singh on Mon, 01 Oct 2007 14:21:42 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> Balbir Singh wrote:
>> Pavel Emelyanov wrote:
>>>> Could we make the user interface similar to the one in the memory
>>>> controller please! It would make it easier for users to configure
>>>> and control both.
>>> Do you mean by merging the similar code or by making them look like
>>> the same? If the second, than can you point at least one difference
>>> except for "usage" vs "usage_in_bytes" (and similar) naming?
>>>
>> I meant similar naming, it makes it is easier if the naming convention
>> is the same.
>
> Ok. It looks like the naming in the memcontrol.c has changed after I
> copied it from there :)
>
> BTW, after Andrew accept these patches I think I will spend some time

> thinking about how to consolidate the res_counter files.
>

Yes, we need to do that. Paul Menage wanted to pull in resource counters
into cgroups. Also, I'd like to be able to share res_counters between
container groups or at-least support a hierarchy. We could discuss this
at that point and/or send out an RFC to get more comments.

> Thanks,
> Pavel

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

Subject: Re: [PATCH 4/5] Setup the control group
Posted by Pavel Emelianov on Mon, 01 Oct 2007 14:27:27 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelyanov wrote:
>> Balbir Singh wrote:
>>> Pavel Emelyanov wrote:
>>>>> Could we make the user interface similar to the one in the memory
>>>>> controller please! It would make it easier for users to configure
>>>>> and control both.
>>>> Do you mean by merging the similar code or by making them look like
>>>> the same? If the second, than can you point at least one difference
>>>> except for "usage" vs "usage_in_bytes" (and similar) naming?
>>>>
>>> I meant similar naming, it makes it is easier if the naming convention
>>> is the same.
>> Ok. It looks like the naming in the memcontrol.c has changed after I
>> copied it from there :)
>>
>> BTW, after Andrew accept these patches I think I will spend some time
>> thinking about how to consolidate the res_counter files.
>>
>
> Yes, we need to do that. Paul Menage wanted to pull in resource counters

No, please, no :) Let's have them splitted apart. I think that the
resource counter abstraction can be used in other places than the
control groups...

> into cgroups. Also, I'd like to be able to share res_counters between
> container groups or at-least support a hierarchy. We could discuss this
> at that point and/or send out an RFC to get more comments.
>
>> Thanks,
>> Pavel
>
>

Subject: Re: [PATCH 0/5] Kernel memory accounting container (v5)
Posted by Pavel Emelianov on Mon, 01 Oct 2007 15:43:44 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelyanov wrote:
>> Changes since v.4:
>> * make SLAB_NOTIFY caches mark pages as SlabDebug. That
>>   makes the interesting paths simpler (thanks to Christoph);
>> * the change above caused appropriate changes in "turn
>>   notifications on" path - all available pages must become
>>   SlabDebug and page's freelists must be flushed;
>> * added two more events - "on" and "off" to make kmalloc
>>   caches disabling more gracefully;
>> * turning notifications "off" is marked as "TODO". Right
>>   now it's hard w/o massive rework of slub.c in respect to
>>   full slabs handling.
>>
>> 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".
>>
>
> Hi, Pavel,
>
> Overall, the patches look good, except for the comments
> noted in the other patch postings. I am going to test
> these patches and see how they hold out.
>

OK, thanks. It's already time to go home now, so I will send
them to Andrew tomorrow. Maybe you'll find some more BUGs by
that time :)

BTW, Acked-by-s from different teams make Andrew more willing
to accept the patches ;)

Thanks,
Pavel

Subject: Re: [PATCH 3/5] Switch caches notification dynamically
Posted by Pavel Emelianov on Mon, 01 Oct 2007 15:45:39 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelyanov wrote:
>> Balbir Singh wrote:
>>>>>> + if ((buf[0] == '1') && !(s->flags & SLAB_NOTIFY)) {
>>>>> Won't this code break if I pass 10 as input in buf?
>>>> I don't care about this :) And no /sys/slab/<xxx>/ file does.
>>>> It's the problem of a user if he pass some shit and doesn't
>>>> see the notifications turned on/off ;)
>>>>
>>> Well, we don't want confused users? How does a user know what
>>> is a valid value to pass? Are you saying that we don't need
>> The rules are simple - 1 to turn them on and 0 to turn them off
>> and nothing more. These values user sees when he "cat"s this file.
>>
>
> I think correct error reporting is critical, if the user sets
> the value to 01 and that ends up disabling slab notifications,
> the handling is wrong.

This is true for all the /sys/slab/<x>/ files...

>>> to care about user interface, if so I don't agree.
>> That's the convention used in the /sys/slab/<x>/ files - the
>> 1 means on the 0 means 0, all the others are not guaranteed :)
>>
>
> Is this documented somewhere or is this interpreted from looking
> at the code of other file handlers?

I didn't notice this in any Documentation. Maybe Christoph can
clarify this question.

Subject: Re: Re: [PATCH 4/5] Setup the control group
Posted by Paul Menage on Mon, 01 Oct 2007 15:50:04 GMT
View Forum Message <> Reply to Message

On 10/1/07, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>
> Yes, we need to do that. Paul Menage wanted to pull in resource counters
> into cgroups.

No, that's not what I wanted - I just wanted to try to cut down the
boilerplate required to use resource counters. Ideally it would be as
simple as including a call to some function such as
"add_res_counter(...)" which would deal with setting up all the cgroup
control files, provide handlers, etc. This also helps to keep all the
APIs consistent.

See the patch that I sent last week entitled "Simplify memory
controller and resource counter I/O" for a first step in that
direction.

> Also, I'd like to be able to share res_counters between
> container groups or at-least support a hierarchy.

A hierarchy sounds like a good idea.

For the sharing, there are two approaches - any subsystem that wants
sharing has to implement it itself, or else implement sharing at the
control groups level and make it (mostly) transparent to the
subsystems. I think I prefer the latter and have been playing with
some ways to do this.

Paul

Subject: Re: Re: [PATCH 4/5] Setup the control group
Posted by Paul Menage on Mon, 01 Oct 2007 15:52:20 GMT
View Forum Message <> Reply to Message

On 10/1/07, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
> Pavel Emelyanov wrote:
> >> Could we make the user interface similar to the one in the memory
> >> controller please! It would make it easier for users to configure
> >> and control both.
> >
> > Do you mean by merging the similar code or by making them look like
> > the same? If the second, than can you point at least one difference
> > except for "usage" vs "usage_in_bytes" (and similar) naming?
> >
>
> I meant similar naming, it makes it is easier if the naming convention
> is the same.
>

I agree with this, but I'd rather see standardization on "limit/usage"
rather than "limit_in_bytes/usage_in_bytes". Then the same API can be
used for subsystems that use resource counters for things that can't
be measured in bytes, e.g. number of tasks.

Paul

Subject: Re: Re: [PATCH 4/5] Setup the control group
Posted by Balbir Singh on Mon, 01 Oct 2007 15:53:54 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On 10/1/07, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>> Yes, we need to do that. Paul Menage wanted to pull in resource counters
>> into cgroups.
>
> No, that's not what I wanted - I just wanted to try to cut down the
> boilerplate required to use resource counters. Ideally it would be as
> simple as including a call to some function such as
> "add_res_counter(...)" which would deal with setting up all the cgroup
> control files, provide handlers, etc. This also helps to keep all the
> APIs consistent.
>
> See the patch that I sent last week entitled "Simplify memory

> controller and resource counter I/O" for a first step in that
> direction.
>

OK. I'll take a closer look

>> Also, I'd like to be able to share res_counters between
>> container groups or at-least support a hierarchy.
>
> A hierarchy sounds like a good idea.
>
> For the sharing, there are two approaches - any subsystem that wants
> sharing has to implement it itself, or else implement sharing at the
> control groups level and make it (mostly) transparent to the
> subsystems. I think I prefer the latter and have been playing with
> some ways to do this.
>

Excellent, I prefer the later as well, but it would mean overheads
for controllers not using the hierarchy. If we can come up with
a design such that parents<->children can effectively share resources,
track them and do so recursively, that would be really nice.

> Paul

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

Subject: Re: Re: [PATCH 4/5] Setup the control group
Posted by Paul Menage on Mon, 01 Oct 2007 16:04:12 GMT
View Forum Message <> Reply to Message

On 10/1/07, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>
> Excellent, I prefer the later as well, but it would mean overheads
> for controllers not using the hierarchy.

I don't think it would have to with the ideas I've been thinking about
- each task would still have a set of pointers to subsystems which
could be dereferenced just as quickly. The complexity comes in trying
to map a task to its actual cgroup object in a given hierarchy - this
would involve a bit more work on the part of the cgroup framework, but
wouldn't be a fast path operation.

See my mail last week titled "Thoughts on virtualizing task containers".

> a design such that parents<->children can effectively share resources,
> track them and do so recursively, that would be really nice.

I think the recursive tracking would probably need to be supplied by
the subsystem rather than by the framework. But there's no reason that
multiple subsystems couldn't re-use the same hierarchy code via e.g.
resource counters. So when you initialize a resource counter you'd
tell it about its parent resource counter, and it would handle the
recursion automatically in charge/uncharge.

Paul

Subject: Re: [PATCH 0/5] Kernel memory accounting container (v5)
Posted by Paul Menage on Mon, 01 Oct 2007 16:32:29 GMT

View Forum Message <> Reply to Message

Hi Pavel,

One question about the general design of this - have you tested an
approach where rather than tagging each object within the cache with
the cgroup that allocated it, you instead have (inside the cache code)
a separate cache structure for each cgroup? So the space overheads
would go from having a per-object overhead (one pointer per object?)
to having a "wastage" overhead (on average half a slab per cgroup).
And the time overhead would be the time required to lookup the
relevant cache for a cgroup at the start of the allocation operation,
and the relevant cache for an object (from its struct page) at
deallocation, rather then the time required to update the per-object
housekeeping pointer.

Each cache would need to be assigned a unique ID, used as an index
into a per-cgroup lookup table of localized caches. (This could almost
be regarded as a form of kmem_cache namespace).

It seems to me that this alternative approach would be a lower memory
overhead for people who have the kernel memory controller compiled in
but aren't using it, or are only using a few groups.

Paul

On 9/25/07, Pavel Emelyanov <xemul@openvz.org> wrote:
> Changes since v.4:
> * make SLAB_NOTIFY caches mark pages as SlabDebug. That
>   makes the interesting paths simpler (thanks to Christoph);

> * the change above caused appropriate changes in "turn
>   notifications on" path - all available pages must become
>   SlabDebug and page's freelists must be flushed;
> * added two more events - "on" and "off" to make kmalloc
>   caches disabling more gracefully;
> * turning notifications "off" is marked as "TODO". Right
>   now it's hard w/o massive rework of slub.c in respect to
>   full slabs handling.
>
> 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".
>
> ==
>
> 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 the kmem one.
>
> 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 control groups 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 marking of all the slabs in the cache as not-debug, but
> due to full-pages in slub are not stored in any lists (usually)
> this is impossible to do so, however this is in todo list.
>
> Thanks,
> Pavel
>

Subject: Re: [PATCH 3/5] Switch caches notification dynamically
Posted by Christoph Lameter on Mon, 01 Oct 2007 20:39:54 GMT

View Forum Message <> Reply to Message

On Mon, 1 Oct 2007, Balbir Singh wrote:

> Is this documented somewhere or is this interpreted from looking
> at the code of other file handlers?

Documentation/vm/slub.txt

Subject: Re: [PATCH 5/5] Account for the slub objects
Posted by Christoph Lameter on Mon, 01 Oct 2007 20:41:47 GMT
View Forum Message <> Reply to Message

On Mon, 1 Oct 2007, Pavel Emelyanov wrote:

> >> +
> >
> > Quick check, slub_free_notify() and slab_alloc_notify() are called
> > from serialized contexts, right?
>
> Yup.

How is it serialized?

Subject: Re: [PATCH 5/5] Account for the slub objects
Posted by Pavel Emelianov on Tue, 02 Oct 2007 12:44:08 GMT
View Forum Message <> Reply to Message

Christoph Lameter wrote:
> On Mon, 1 Oct 2007, Pavel Emelyanov wrote:
>
>>>> +
>>> Quick check, slub_free_notify() and slab_alloc_notify() are called
>>> from serialized contexts, right?
>> Yup.
>
> How is it serialized?

They are booth called from __slab_alloc()/__slab_free() from under
the slab_lock(page).

Thanks,

Pavel

____

Subject: Re: [PATCH 0/5] Kernel memory accounting container (v5)
Posted by Pavel Emelianov on Tue, 02 Oct 2007 12:51:10 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> Hi Pavel,
>
> One question about the general design of this - have you tested an
> approach where rather than tagging each object within the cache with
> the cgroup that allocated it, you instead have (inside the cache code)
> a separate cache structure for each cgroup? So the space overheads
> would go from having a per-object overhead (one pointer per object?)
> to having a "wastage" overhead (on average half a slab per cgroup).
> And the time overhead would be the time required to lookup the
> relevant cache for a cgroup at the start of the allocation operation,
> and the relevant cache for an object (from its struct page) at
> deallocation, rather then the time required to update the per-object
> housekeeping pointer.

Such a lookup would require a hastable or something similar. We already
have such a bad experience (with OpenVZ RSS fractions accounting for
example). Hash lookups imply the CPU caches screwup and hurt the performance.
See also the comment below.

> Each cache would need to be assigned a unique ID, used as an index
> into a per-cgroup lookup table of localized caches. (This could almost
> be regarded as a form of kmem_cache namespace).
>
> It seems to me that this alternative approach would be a lower memory
> overhead for people who have the kernel memory controller compiled in
> but aren't using it, or are only using a few groups.

I thought the same some time ago and tried to make a per-beancounter kmem
caches. The result was awful - the memory waste was much larger than in the
case of pointer-per-object approach. Let alone the performance questions -
each kmalloc required a synchronized hash table lookup that was too bad.

If you insist I can try to repeat the experiment, but I'm afraid the result
would be the same.

> Paul
>

Subject: Re: [PATCH 5/5] Account for the slub objects
Posted by Christoph Lameter on Tue, 02 Oct 2007 18:04:38 GMT
View Forum Message <> Reply to Message

On Tue, 2 Oct 2007, Pavel Emelyanov wrote:

> Christoph Lameter wrote:
> > On Mon, 1 Oct 2007, Pavel Emelyanov wrote:
> >
> >>>> +
> >>> Quick check, slub_free_notify() and slab_alloc_notify() are called
> >>> from serialized contexts, right?
> >> Yup.
> >
> > How is it serialized?
>
> They are booth called from __slab_alloc()/__slab_free() from under
> the slab_lock(page).

This means they are serialized per slab. Which means you can guarantee
that multiple of these callbacks are not done at the same time for the
same object. Is that what you need?

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH 5/5] Account for the slub objects
Posted by Pavel Emelianov on Wed, 03 Oct 2007 07:29:13 GMT
View Forum Message <> Reply to Message

Christoph Lameter wrote:
> On Tue, 2 Oct 2007, Pavel Emelyanov wrote:
>
>> Christoph Lameter wrote:
>>> On Mon, 1 Oct 2007, Pavel Emelyanov wrote:
>>>
>>>>>> +
>>>>> Quick check, slub_free_notify() and slab_alloc_notify() are called
>>>>> from serialized contexts, right?
>>>> Yup.

>>> How is it serialized?
>> They are booth called from __slab_alloc()/__slab_free() from under
>> the slab_lock(page).
>
> This means they are serialized per slab. Which means you can guarantee
> that multiple of these callbacks are not done at the same time for the
> same object. Is that what you need?

Yes I know it :) But I do not rely on this lock inside the callbacks. What
I need is to notify each new object only once, but this doesn't matter for
the callbacks whether there exists some lock or not. In other words - this
is just a coincidence that these callbacks are called from under this lock,
this was not done deliberately. Fortunately, the rollback in case when the
callbacks return an error is done easily under this lock.

Thanks,
Pavel

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 0/5] Kernel memory accounting container (v5)
Posted by Paul Menage on Fri, 05 Oct 2007 07:11:30 GMT
View Forum Message <> Reply to Message

On 10/2/07, Pavel Emelyanov <xemul@openvz.org> wrote:
>
> Such a lookup would require a hastable or something similar. We already
> have such a bad experience (with OpenVZ RSS fractions accounting for
> example). Hash lookups imply the CPU caches screwup and hurt the performance.
> See also the comment below.

I think you could do it with an array lookup if you assigned an index
to each cache as it was created, and used that as an offset into a
per-cgroup array.

>
> I thought the same some time ago and tried to make a per-beancounter kmem
> caches. The result was awful - the memory waste was much larger than in the
> case of pointer-per-object approach.

OK, fair enough.

Was this with a large number of bean counters? I imagine that with a
small number, the waste might be rather more reasonable.

Paul

Subject: Re: [PATCH 0/5] Kernel memory accounting container (v5)
Posted by Pavel Emelianov on Fri, 05 Oct 2007 13:17:48 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On 10/2/07, Pavel Emelyanov <xemul@openvz.org> wrote:
>> Such a lookup would require a hastable or something similar. We already
>> have such a bad experience (with OpenVZ RSS fractions accounting for
>> example). Hash lookups imply the CPU caches screwup and hurt the performance.
>> See also the comment below.
>
> I think you could do it with an array lookup if you assigned an index
> to each cache as it was created, and used that as an offset into a
> per-cgroup array.
>
>> I thought the same some time ago and tried to make a per-beancounter kmem
>> caches. The result was awful - the memory waste was much larger than in the
>> case of pointer-per-object approach.
>
> OK, fair enough.
>
> Was this with a large number of bean counters? I imagine that with a
> small number, the waste might be rather more reasonable.

Yup. I do not remember the exact number, but this model didn't scale
well enough in respect to the number of beancounters.

> Paul
>
