Subject: [PATCH 0/4] Proposed slab patches as basis for memcg
Posted by Glauber Costa on Thu, 14 Jun 2012 12:17:20 GMT
View Forum Message <> Reply to Message

Hi,

These four patches are sat in my tree for kmem memcg work.
All of them are preparation patches that touch the allocators
to make them more consistent, allowing me to later use them
from common code.

In this current form, they are supposed to be applied after
Cristoph's series. They are not, however, dependent on it.

Glauber Costa (4):
  slab: rename gfpflags to allocflags
  provide a common place for initcall processing in kmem_cache
  slab: move FULL state transition to an initcall
  make CFLGS_OFF_SLAB visible for all slabs

 include/linux/slab.h | 2 ++
 include/linux/slab_def.h | 2 +-
 mm/slab.c | 40 ++++++++++++++++++++++----------------
 mm/slab.h | 1 +
 mm/slab_common.c | 5 +++++
 mm/slob.c | 5 +++++
 mm/slub.c | 4 +---
 7 files changed, 34 insertions(+), 25 deletions(-)

--
1.7.10.2

Subject: [PATCH 1/4] slab: rename gfpflags to allocflags
Posted by Glauber Costa on Thu, 14 Jun 2012 12:17:21 GMT
View Forum Message <> Reply to Message

A consistent name with slub saves us an acessor function.
In both caches, this field represents the same thing. We would
like to use it from the mem_cgroup code.

Signed-off-by: Glauber Costa <glommer@parallels.com>
Acked-by: Christoph Lameter <cl@linux.com>
CC: Pekka Enberg <penberg@cs.helsinki.fi>
---
 include/linux/slab_def.h | 2 +-
 mm/slab.c | 10 +++++-----
 2 files changed, 6 insertions(+), 6 deletions(-)

```
diff --git a/include/linux/slab_def.h b/include/linux/slab_def.h
index 1d93f27..0c634fa 100644
--- a/include/linux/slab_def.h
+++ b/include/linux/slab_def.h
@@ -39,7 +39,7 @@ struct kmem_cache {
  unsigned int gfporder;

  /* force GFP flags, e.g. GFP_DMA */
- gfp_t gfpflags;
+ gfp_t allocflags;

  size_t colour;   /* cache colouring range */
  unsigned int colour_off; /* colour offset */
diff --git a/mm/slab.c b/mm/slab.c
index 2476ad4..020605f 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -1746,7 +1746,7 @@ static void *kmem_getpages(struct kmem_cache *cachep, gfp_t flags,
int nodeid)
  flags |= __GFP_COMP;
 #endif

- flags |= cachep->gfpflags;
+ flags |= cachep->allocflags;
  if (cachep->flags & SLAB_RECLAIM_ACCOUNT)
   flags |= __GFP_RECLAIMABLE;

@@ -2338,9 +2338,9 @@ int __kmem_cache_create(struct kmem_cache *cachep)
  cachep->colour = left_over / cachep->colour_off;
  cachep->slab_size = slab_size;
  cachep->flags = flags;
- cachep->gfpflags = 0;
+ cachep->allocflags = 0;
  if (CONFIG_ZONE_DMA_FLAG && (flags & SLAB_CACHE_DMA))
-  cachep->gfpflags |= GFP_DMA;
+  cachep->allocflags |= GFP_DMA;
  cachep->size = size;
  cachep->reciprocal_buffer_size = reciprocal_value(size);

@@ -2653,9 +2653,9 @@ static void kmem_flagcheck(struct kmem_cache *cachep, gfp_t flags)
 {
  if (CONFIG_ZONE_DMA_FLAG) {
   if (flags & GFP_DMA)
-   BUG_ON(!(cachep->gfpflags & GFP_DMA));
+   BUG_ON(!(cachep->allocflags & GFP_DMA));
   else
-   BUG_ON(cachep->gfpflags & GFP_DMA);
```

```
+   BUG_ON(cachep->allocflags & GFP_DMA);
 }
}

--
1.7.10.2
```

Subject: [PATCH 2/4] provide a common place for initcall processing in kmem_cache
Posted by Glauber Costa on Thu, 14 Jun 2012 12:17:22 GMT
View Forum Message <> Reply to Message

Both SLAB and SLUB depend on some initialization to happen when the
system is already booted, with all subsystems working. This is done
by issuing an initcall that does the final initialization.

This patch moves that to slab_common.c, while creating an empty
placeholder for the SLOB.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Christoph Lameter <cl@linux.com>
CC: Pekka Enberg <penberg@cs.helsinki.fi>
CC: David Rientjes <rientjes@google.com>

```
 mm/slab.c        |    5 ++---
 mm/slab.h        |    1 +
 mm/slab_common.c |    5 +++++
 mm/slob.c        |    5 +++++
 mm/slub.c        |    4 +---
 5 files changed, 14 insertions(+), 6 deletions(-)

diff --git a/mm/slab.c b/mm/slab.c
index 020605f..e174e50 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -853,7 +853,7 @@ static void __cpuinit start_cpu_timer(int cpu)
  struct delayed_work *reap_work = &per_cpu(slab_reap_work, cpu);

  /*
-  * When this gets called from do_initcalls via cpucache_init(),
+  * When this gets called from do_initcalls via __kmem_cache_initcall(),
   * init_workqueues() has already run, so keventd will be setup
   * at that time.
   */
@@ -1666,7 +1666,7 @@ void __init kmem_cache_init_late(void)
   */
 }
```

```
-static int __init cpucache_init(void)
+int __init __kmem_cache_initcall(void)
 {
  int cpu;

@@ -1677,7 +1677,6 @@ static int __init cpucache_init(void)
   start_cpu_timer(cpu);
  return 0;
 }
-__initcall(cpucache_init);

 static noinline void
 slab_out_of_memory(struct kmem_cache *cachep, gfp_t gfpflags, int nodeid)
diff --git a/mm/slab.h b/mm/slab.h
index b44a8cc..19e17c7 100644
--- a/mm/slab.h
+++ b/mm/slab.h
@@ -37,6 +37,7 @@ unsigned long calculate_alignment(unsigned long flags,

 /* Functions provided by the slab allocators */
 int __kmem_cache_create(struct kmem_cache *s);
+int __kmem_cache_initcall(void);

 #ifdef CONFIG_SLUB
 struct kmem_cache *__kmem_cache_alias(const char *name, size_t size,
diff --git a/mm/slab_common.c b/mm/slab_common.c
index b5def07..15db694ac 100644
--- a/mm/slab_common.c
+++ b/mm/slab_common.c
@@ -203,3 +203,8 @@ int slab_is_available(void)
  return slab_state >= UP;
 }

+static int __init kmem_cache_initcall(void)
+{
+ return __kmem_cache_initcall();
+}
+__initcall(kmem_cache_initcall);
diff --git a/mm/slob.c b/mm/slob.c
index 507589d..61b1845 100644
--- a/mm/slob.c
+++ b/mm/slob.c
@@ -610,3 +610,8 @@ void __init kmem_cache_init(void)
 void __init kmem_cache_init_late(void)
 {
 }
+
```

```
+int __init kmem_cache_initcall(void)
+{
+ return 0;
+}
diff --git a/mm/slub.c b/mm/slub.c
index 7fc3499..d9d4d5a 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -5309,7 +5309,7 @@ static int sysfs_slab_alias(struct kmem_cache *s, const char *name)
  return 0;
 }

-static int __init slab_sysfs_init(void)
+int __init __kmem_cache_initcall(void)
 {
  struct kmem_cache *s;
  int err;
@@ -5347,8 +5347,6 @@ static int __init slab_sysfs_init(void)
  resiliency_test();
  return 0;
 }
-
-__initcall(slab_sysfs_init);
 #endif /* CONFIG_SYSFS */

 /*
--
1.7.10.2
```

Subject: [PATCH 3/4] slab: move FULL state transition to an initcall
Posted by Glauber Costa on Thu, 14 Jun 2012 12:17:23 GMT
View Forum Message <> Reply to Message

During kmem_cache_init_late(), we transition to the LATE state,
and after some more work, to the FULL state, its last state

This is quite different from slub, that will only transition to
its last state (previously SYSFS), in a (late)initcall, after a lot
more of the kernel is ready.

This means that in slab, we have no way to taking actions dependent
on the initialization of other pieces of the kernel that are supposed
to start way after kmem_init_late(), such as cgroups initialization.

To achieve more consistency in this behavior, that patch only
transitions to the UP state in kmem_init_late. In my analysis,
setup_cpu_cache() should be happy to test for >= UP, instead of

== FULL. It also has passed some tests I've made.

We then only mark FULL state after the reap timers are in place,
meaning that no further setup is expected.

Signed-off-by: Glauber Costa <glommer@parallels.com>
Acked-by: Christoph Lameter <cl@linux.com>
CC: Pekka Enberg <penberg@cs.helsinki.fi>
CC: David Rientjes <rientjes@google.com>
---
 mm/slab.c | 8 ++++----
 1 file changed, 4 insertions(+), 4 deletions(-)

diff --git a/mm/slab.c b/mm/slab.c
index e174e50..2d5fe28 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -1643,9 +1643,6 @@ void __init kmem_cache_init_late(void)
    BUG();
  mutex_unlock(&slab_mutex);

- /* Done! */
- slab_state = FULL;
-
  /*
   * Register a cpu startup notifier callback that initializes
   * cpu_cache_get for all new cpus
@@ -1675,6 +1672,9 @@ int __init __kmem_cache_initcall(void)
   */
  for_each_online_cpu(cpu)
   start_cpu_timer(cpu);
+
+ /* Done! */
+ slab_state = FULL;
  return 0;
 }

@@ -2120,7 +2120,7 @@ static size_t calculate_slab_order(struct kmem_cache *cachep,

 static int __init_refok setup_cpu_cache(struct kmem_cache *cachep, gfp_t gfp)
 {
- if (slab_state == FULL)
+ if (slab_state >= UP)
   return enable_cpucache(cachep, gfp);

  if (slab_state == DOWN) {
--
1.7.10.2

Subject: [PATCH 4/4] make CFLGS_OFF_SLAB visible for all slabs
Posted by Glauber Costa on Thu, 14 Jun 2012 12:17:24 GMT
View Forum Message <> Reply to Message

Since we're now moving towards a unified slab allocator interface,
make CFLGS_OFF_SLAB visible to all allocators, even though SLAB keeps
being its only users. Also, make the name consistent with the other
flags, that start with SLAB_xx.

This will allow us to mask out this flag from common code, which will
of course have no effect in allocators not using it. It will also
avoid other allocators using this bit in the future by mistake.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Christoph Lameter <cl@linux.com>
CC: Pekka Enberg <penberg@cs.helsinki.fi>
CC: David Rientjes <rientjes@google.com>
---
 include/linux/slab.h | 2 ++
 mm/slab.c | 17 ++++++++---------
 2 files changed, 10 insertions(+), 9 deletions(-)

```
diff --git a/include/linux/slab.h b/include/linux/slab.h
index 3c2181a..62deb32 100644
--- a/include/linux/slab.h
+++ b/include/linux/slab.h
@@ -79,6 +79,8 @@
 /* The following flags affect the page allocator grouping pages by mobility */
 #define SLAB_RECLAIM_ACCOUNT 0x00020000UL  /* Objects are reclaimable */
 #define SLAB_TEMPORARY  SLAB_RECLAIM_ACCOUNT /* Objects are short-lived */
+
+#define SLAB_OFF_SLAB  0x80000000UL
 /*
  * ZERO_SIZE_PTR will be returned for zero sized kmalloc requests.
  *
diff --git a/mm/slab.c b/mm/slab.c
index 2d5fe28..c0cf297 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -358,8 +358,7 @@ static void kmem_list3_init(struct kmem_list3 *parent)
  MAKE_LIST((cachep), (&(ptr)->slabs_free), slabs_free, nodeid); \
  } while (0)

-#define CFLGS_OFF_SLAB  (0x80000000UL)
-#define OFF_SLAB(x) ((x)->flags & CFLGS_OFF_SLAB)
+#define OFF_SLAB(x) ((x)->flags & SLAB_OFF_SLAB)

 #define BATCHREFILL_LIMIT 16
 /*
```

```
@@ -744,7 +743,7 @@ static void cache_estimate(unsigned long gfporder, size_t buffer_size,
   * the slabs are all pages aligned, the objects will be at the
   * correct alignment when allocated.
   */
- if (flags & CFLGS_OFF_SLAB) {
+ if (flags & SLAB_OFF_SLAB) {
   mgmt_size = 0;
   nr_objs = slab_size / buffer_size;

@@ -2076,7 +2075,7 @@ static size_t calculate_slab_order(struct kmem_cache *cachep,
   if (!num)
    continue;

-  if (flags & CFLGS_OFF_SLAB) {
+  if (flags & SLAB_OFF_SLAB) {
    /*
     * Max number of objs-per-slab for caches which
     * use off-slab slabs. Needed to avoid a possible
@@ -2294,7 +2293,7 @@ int __kmem_cache_create(struct kmem_cache *cachep)
    * Size is large, assume best to place the slab management obj
    * off-slab (should allow better packing of objs).
    */
-  flags |= CFLGS_OFF_SLAB;
+  flags |= SLAB_OFF_SLAB;

  size = ALIGN(size, align);

@@ -2310,12 +2309,12 @@ int __kmem_cache_create(struct kmem_cache *cachep)
   * If the slab has been placed off-slab, and we have enough space then
   * move it on-slab. This is at the expense of any extra colouring.
   */
- if (flags & CFLGS_OFF_SLAB && left_over >= slab_size) {
-  flags &= ~CFLGS_OFF_SLAB;
+ if (flags & SLAB_OFF_SLAB && left_over >= slab_size) {
+  flags &= ~SLAB_OFF_SLAB;
   left_over -= slab_size;
  }

- if (flags & CFLGS_OFF_SLAB) {
+ if (flags & SLAB_OFF_SLAB) {
   /* really off slab. No need for manual alignment */
   slab_size =
       cachep->num * sizeof(kmem_bufctl_t) + sizeof(struct slab);
@@ -2343,7 +2342,7 @@ int __kmem_cache_create(struct kmem_cache *cachep)
  cachep->size = size;
  cachep->reciprocal_buffer_size = reciprocal_value(size);

- if (flags & CFLGS_OFF_SLAB) {
```

```
+ if (flags & SLAB_OFF_SLAB) {
   cachep->slabp_cache = kmem_find_general_cachep(slab_size, 0u);
   /*
    * This is a possibility for one of the malloc_sizes caches.
--
1.7.10.2
```

Subject: Re: [PATCH 4/4] make CFLGS_OFF_SLAB visible for all slabs
Posted by Christoph Lameter on Thu, 14 Jun 2012 15:19:55 GMT
View Forum Message <> Reply to Message

On Thu, 14 Jun 2012, Glauber Costa wrote:

> Since we're now moving towards a unified slab allocator interface,
> make CFLGS_OFF_SLAB visible to all allocators, even though SLAB keeps
> being its only users. Also, make the name consistent with the other
> flags, that start with SLAB_xx.

What is the significance of knowledge about internal slab structures (such
as the CFGLFS_OFF_SLAB) outside of the allocators?

Subject: Re: [PATCH 4/4] make CFLGS_OFF_SLAB visible for all slabs
Posted by Glauber Costa on Thu, 14 Jun 2012 16:01:31 GMT
View Forum Message <> Reply to Message

On 06/14/2012 07:19 PM, Christoph Lameter wrote:
> On Thu, 14 Jun 2012, Glauber Costa wrote:
>
>> Since we're now moving towards a unified slab allocator interface,
>> make CFLGS_OFF_SLAB visible to all allocators, even though SLAB keeps
>> being its only users. Also, make the name consistent with the other
>> flags, that start with SLAB_xx.
>
> What is the significance of knowledge about internal slab structures (such
> as the CFGLFS_OFF_SLAB) outside of the allocators?

I want to mask that out in kmem-specific slab creation. Since I am
copying the original flags, and that flag is embedded in the slab saved
flags, it will be carried to the new slab if I don't mask it out.

Alternatively to this, I can tweak slab.c to always mask out this at the
beginning of cache creation, if you so prefer.

Subject: Re: [PATCH 4/4] make CFLGS_OFF_SLAB visible for all slabs
Posted by Christoph Lameter on Thu, 14 Jun 2012 17:29:02 GMT
View Forum Message <> Reply to Message

On Thu, 14 Jun 2012, Glauber Costa wrote:

> I want to mask that out in kmem-specific slab creation. Since I am copying the
> original flags, and that flag is embedded in the slab saved flags, it will be
> carried to the new slab if I don't mask it out.

I thought you intercepted slab creation? You can copy the flags at that
point.

Subject: Re: [PATCH 0/4] Proposed slab patches as basis for memcg
Posted by Pekka Enberg on Mon, 02 Jul 2012 10:52:38 GMT
View Forum Message <> Reply to Message

On Thu, 14 Jun 2012, Glauber Costa wrote:
> These four patches are sat in my tree for kmem memcg work.
> All of them are preparation patches that touch the allocators
> to make them more consistent, allowing me to later use them
> from common code.
>
> In this current form, they are supposed to be applied after
> Cristoph's series. They are not, however, dependent on it.
>
> Glauber Costa (4):
> slab: rename gfpflags to allocflags
> provide a common place for initcall processing in kmem_cache
> slab: move FULL state transition to an initcall
> make CFLGS_OFF_SLAB visible for all slabs
>
> include/linux/slab.h | 2 ++
> include/linux/slab_def.h | 2 +-
> mm/slab.c | 40 ++++++++++++++++++++----------------
> mm/slab.h | 1 +
> mm/slab_common.c | 5 +++++
> mm/slob.c | 5 +++++
> mm/slub.c | 4 +---
> 7 files changed, 34 insertions(+), 25 deletions(-)

I applied the first patch. Rest of them don't apply on top of slab/next
branch because it's missing some of the key patches from Christoph's
series. Did anyone fix them up while I was offline?

Pekka

Subject: Re: [PATCH 4/4] make CFLGS_OFF_SLAB visible for all slabs
Posted by Glauber Costa on Mon, 02 Jul 2012 10:57:01 GMT
View Forum Message <> Reply to Message

On 06/14/2012 07:19 PM, Christoph Lameter wrote:
> On Thu, 14 Jun 2012, Glauber Costa wrote:
>
>> Since we're now moving towards a unified slab allocator interface,
>> make CFLGS_OFF_SLAB visible to all allocators, even though SLAB keeps
>> being its only users. Also, make the name consistent with the other
>> flags, that start with SLAB_xx.
>
> What is the significance of knowledge about internal slab structures (such
> as the CFGLFS_OFF_SLAB) outside of the allocators?
>
Pekka, please note this comment when you are scanning through the series
(which you seem to be doing now).

This one is better left off for now.