Subject: [PATCH 1/4] BIO tracking take2
Posted by Hirokazu Takahashi on Tue, 20 May 2008 12:02:00 GMT
View Forum Message <> Reply to Message

Hi,

This patch splits the cgroup memory subsystem into two parts.
One is for tracking pages to find out the owners. The other is
for controlling how much amount of memory should be assigned to
each cgroup.

With this patch, you can use the page tracking mechanism even if
the memory subsystem is off.

Signed-off-by: Hirokazu Takahashi <taka@valinux.co.jp>

```
diff -udpr linux-2.6.26-rc2.cg0/include/linux/memcontrol.h
linux-2.6.26-rc2/include/linux/memcontrol.h
--- linux-2.6.26-rc2.cg0/include/linux/memcontrol.h 2008-05-16 19:03:11.000000000 +0900
+++ linux-2.6.26-rc2/include/linux/memcontrol.h 2008-05-16 19:49:51.000000000 +0900
@@ -20,12 +20,61 @@
 #ifndef _LINUX_MEMCONTROL_H
 #define _LINUX_MEMCONTROL_H

+#include <linux/rcupdate.h>
+#include <linux/mm.h>
+#include <linux/smp.h>
+#include <linux/bit_spinlock.h>
+
 struct mem_cgroup;
 struct page_cgroup;
 struct page;
 struct mm_struct;

+#ifdef CONFIG_CGROUP_PAGE
+/*
+ * We use the lower bit of the page->page_cgroup pointer as a bit spin
+ * lock.  We need to ensure that page->page_cgroup is at least two
+ * byte aligned (based on comments from Nick Piggin).  But since
+ * bit_spin_lock doesn't actually set that lock bit in a non-debug
+ * uniprocessor kernel, we should avoid setting it here too.
+ */
+#define PAGE_CGROUP_LOCK_BIT  0x0
+#if defined(CONFIG_SMP) || defined(CONFIG_DEBUG_SPINLOCK)
+#define PAGE_CGROUP_LOCK  (1 << PAGE_CGROUP_LOCK_BIT)
+#else
+#define PAGE_CGROUP_LOCK 0x0
```

```
+#endif
+
+/*
+ * A page_cgroup page is associated with every page descriptor. The
+ * page_cgroup helps us identify information about the cgroup
+ */
+struct page_cgroup {
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ struct list_head lru;  /* per cgroup LRU list */
+ struct mem_cgroup *mem_cgroup;
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ struct page *page;
+ int ref_cnt;   /* cached, mapped, migrating */
+ int flags;
+};
+#define PAGE_CGROUP_FLAG_CACHE (0x1) /* charged as cache */
+#define PAGE_CGROUP_FLAG_ACTIVE (0x2) /* page is active in this cgroup */
+
+static inline void lock_page_cgroup(struct page *page)
+{
+ bit_spin_lock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
+}
+
+static inline int try_lock_page_cgroup(struct page *page)
+{
+ return bit_spin_trylock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
+}
+
+static inline void unlock_page_cgroup(struct page *page)
+{
+ bit_spin_unlock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
+}

 #define page_reset_bad_cgroup(page) ((page)->page_cgroup = 0)

@@ -35,44 +84,15 @@ extern int mem_cgroup_charge(struct page
 extern int mem_cgroup_cache_charge(struct page *page, struct mm_struct *mm,
      gfp_t gfp_mask);
 extern void mem_cgroup_uncharge_page(struct page *page);
-extern void mem_cgroup_move_lists(struct page *page, bool active);
-extern unsigned long mem_cgroup_isolate_pages(unsigned long nr_to_scan,
-     struct list_head *dst,
-     unsigned long *scanned, int order,
-     int mode, struct zone *z,
-     struct mem_cgroup *mem_cont,
-     int active);
-extern void mem_cgroup_out_of_memory(struct mem_cgroup *mem, gfp_t gfp_mask);
-int task_in_mem_cgroup(struct task_struct *task, const struct mem_cgroup *mem);
```

```
-
-extern struct mem_cgroup *mem_cgroup_from_task(struct task_struct *p);
-
-#define mm_match_cgroup(mm, cgroup) \
- ((cgroup) == mem_cgroup_from_task((mm)->owner))

 extern int
 mem_cgroup_prepare_migration(struct page *page, struct page *newpage);
 extern void mem_cgroup_end_migration(struct page *page);
 extern int mem_cgroup_getref(struct page *page);

-/*
- * For memory reclaim.
- */
-extern int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem);
-extern long mem_cgroup_reclaim_imbalance(struct mem_cgroup *mem);
-
-extern int mem_cgroup_get_reclaim_priority(struct mem_cgroup *mem);
-extern void mem_cgroup_note_reclaim_priority(struct mem_cgroup *mem,
-       int priority);
-extern void mem_cgroup_record_reclaim_priority(struct mem_cgroup *mem,
-       int priority);
-
-extern long mem_cgroup_calc_reclaim_active(struct mem_cgroup *mem,
-    struct zone *zone, int priority);
-extern long mem_cgroup_calc_reclaim_inactive(struct mem_cgroup *mem,
-    struct zone *zone, int priority);
+extern void page_cgroup_init(void);

-#else /* CONFIG_CGROUP_MEM_RES_CTLR */
+#else /* CONFIG_CGROUP_PAGE */
 static inline void page_reset_bad_cgroup(struct page *page)
 {
 }
@@ -98,33 +118,70 @@ static inline void mem_cgroup_uncharge_p
 {
 }

-static inline void mem_cgroup_move_lists(struct page *page, bool active)
+static inline int mem_cgroup_prepare_migration(struct page *page)
 {
+ return 0;
 }

-static inline int mm_match_cgroup(struct mm_struct *mm, struct mem_cgroup *mem)
+static inline void mem_cgroup_end_migration(struct page *page)
 {
- return 1;
```

```
 }

-static inline int task_in_mem_cgroup(struct task_struct *task,
-        const struct mem_cgroup *mem)
+static inline void mem_cgroup_getref(struct page *page)
 {
- return 1;
 }
+#endif /* CONFIG_CGROUP_PAGE */

-static inline int
-mem_cgroup_prepare_migration(struct page *page, struct page *newpage)
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+
+extern void mem_cgroup_move_lists(struct page *page, bool active);
+extern unsigned long mem_cgroup_isolate_pages(unsigned long nr_to_scan,
+     struct list_head *dst,
+     unsigned long *scanned, int order,
+     int mode, struct zone *z,
+     struct mem_cgroup *mem_cont,
+     int active);
+extern void mem_cgroup_out_of_memory(struct mem_cgroup *mem, gfp_t gfp_mask);
+int task_in_mem_cgroup(struct task_struct *task, const struct mem_cgroup *mem);
+
+extern struct mem_cgroup *mem_cgroup_from_task(struct task_struct *p);
+
+#define mm_match_cgroup(mm, cgroup) \
+ ((cgroup) == mem_cgroup_from_task((mm)->owner))
+
+/*
+ * For memory reclaim.
+ */
+extern int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem);
+extern long mem_cgroup_reclaim_imbalance(struct mem_cgroup *mem);
+
+extern int mem_cgroup_get_reclaim_priority(struct mem_cgroup *mem);
+extern void mem_cgroup_note_reclaim_priority(struct mem_cgroup *mem,
+       int priority);
+extern void mem_cgroup_record_reclaim_priority(struct mem_cgroup *mem,
+       int priority);
+
+extern long mem_cgroup_calc_reclaim_active(struct mem_cgroup *mem,
+    struct zone *zone, int priority);
+extern long mem_cgroup_calc_reclaim_inactive(struct mem_cgroup *mem,
+    struct zone *zone, int priority);
+
+#else /* CONFIG_CGROUP_MEM_RES_CTLR */
```

```
+
+static inline void mem_cgroup_move_lists(struct page *page, bool active)
 {
- return 0;
 }

-static inline void mem_cgroup_end_migration(struct page *page)
+static inline int mm_match_cgroup(struct mm_struct *mm, struct mem_cgroup *mem)
 {
+ return 1;
 }

-static inline void mem_cgroup_getref(struct page *page)
+static inline int task_in_mem_cgroup(struct task_struct *task,
+         const struct mem_cgroup *mem)
 {
+ return 1;
 }

 static inline int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem)
@@ -163,7 +220,7 @@ static inline long mem_cgroup_calc_recla
 {
  return 0;
 }
-#endif /* CONFIG_CGROUP_MEM_CONT */
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */

 #endif /* _LINUX_MEMCONTROL_H */

diff -udpr linux-2.6.26-rc2.cg0/include/linux/mm_types.h linux-2.6.26-rc2/include/linux/mm_types.h
--- linux-2.6.26-rc2.cg0/include/linux/mm_types.h 2008-05-16 19:03:11.000000000 +0900
+++ linux-2.6.26-rc2/include/linux/mm_types.h 2008-05-16 19:03:43.000000000 +0900
@@ -91,7 +91,7 @@ struct page {
  void *virtual;   /* Kernel virtual address (NULL if
         not kmapped, ie. highmem) */
 #endif /* WANT_PAGE_VIRTUAL */
-#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+#ifdef CONFIG_CGROUP_PAGE
  unsigned long page_cgroup;
 #endif
 #ifdef CONFIG_PAGE_OWNER
diff -udpr linux-2.6.26-rc2.cg0/init/Kconfig linux-2.6.26-rc2/init/Kconfig
--- linux-2.6.26-rc2.cg0/init/Kconfig 2008-05-16 19:03:11.000000000 +0900
+++ linux-2.6.26-rc2/init/Kconfig 2008-05-16 19:03:43.000000000 +0900
@@ -407,6 +407,10 @@ config CGROUP_MEM_RES_CTLR
    This config option also selects MM_OWNER config option, which
    could in turn add some fork/exit overhead.
```

```
+config CGROUP_PAGE
+ def_bool y
+ depends on CGROUP_MEM_RES_CTLR
+
 config SYSFS_DEPRECATED
  bool

diff -udpr linux-2.6.26-rc2.cg0/mm/Makefile linux-2.6.26-rc2/mm/Makefile
--- linux-2.6.26-rc2.cg0/mm/Makefile 2008-05-16 19:03:11.000000000 +0900
+++ linux-2.6.26-rc2/mm/Makefile 2008-05-16 19:03:43.000000000 +0900
@@ -33,5 +33,5 @@ obj-$(CONFIG_FS_XIP) += filemap_xip.o
 obj-$(CONFIG_MIGRATION) += migrate.o
 obj-$(CONFIG_SMP) += allocpercpu.o
 obj-$(CONFIG_QUICKLIST) += quicklist.o
-obj-$(CONFIG_CGROUP_MEM_RES_CTLR) += memcontrol.o
+obj-$(CONFIG_CGROUP_PAGE) += memcontrol.o

diff -udpr linux-2.6.26-rc2.cg0/mm/memcontrol.c linux-2.6.26-rc2/mm/memcontrol.c
--- linux-2.6.26-rc2.cg0/mm/memcontrol.c 2008-05-16 19:03:11.000000000 +0900
+++ linux-2.6.26-rc2/mm/memcontrol.c 2008-05-19 11:39:43.000000000 +0900
@@ -35,10 +35,17 @@

 #include <asm/uaccess.h>

-struct cgroup_subsys mem_cgroup_subsys __read_mostly;
 static struct kmem_cache *page_cgroup_cache __read_mostly;
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+struct cgroup_subsys mem_cgroup_subsys __read_mostly;
 #define MEM_CGROUP_RECLAIM_RETRIES 5

+static inline int mem_cgroup_disabled(void)
+{
+ return mem_cgroup_subsys.disabled;
+}
+
 /*
  * Statistics for memory cgroup.
  */
@@ -144,34 +151,6 @@ struct mem_cgroup {
 };
 static struct mem_cgroup init_mem_cgroup;

-/*
- * We use the lower bit of the page->page_cgroup pointer as a bit spin
- * lock.  We need to ensure that page->page_cgroup is at least two
- * byte aligned (based on comments from Nick Piggin).  But since
- * bit_spin_lock doesn't actually set that lock bit in a non-debug
```

```
- * uniprocessor kernel, we should avoid setting it here too.
- */
-#define PAGE_CGROUP_LOCK_BIT  0x0
-#if defined(CONFIG_SMP) || defined(CONFIG_DEBUG_SPINLOCK)
-#define PAGE_CGROUP_LOCK  (1 << PAGE_CGROUP_LOCK_BIT)
-#else
-#define PAGE_CGROUP_LOCK 0x0
-#endif
-
-/*
- * A page_cgroup page is associated with every page descriptor. The
- * page_cgroup helps us identify information about the cgroup
- */
-struct page_cgroup {
- struct list_head lru;  /* per cgroup LRU list */
- struct page *page;
- struct mem_cgroup *mem_cgroup;
- int ref_cnt;   /* cached, mapped, migrating */
- int flags;
-};
-#define PAGE_CGROUP_FLAG_CACHE (0x1) /* charged as cache */
-#define PAGE_CGROUP_FLAG_ACTIVE (0x2) /* page is active in this cgroup */
-
 static int page_cgroup_nid(struct page_cgroup *pc)
 {
  return page_to_nid(pc->page);
@@ -182,11 +161,6 @@ static enum zone_type page_cgroup_zid(st
  return page_zonenum(pc->page);
 }

-enum charge_type {
- MEM_CGROUP_CHARGE_TYPE_CACHE = 0,
- MEM_CGROUP_CHARGE_TYPE_MAPPED,
-};
-
 /*
  * Always modified under lru lock. Then, not necessary to preempt_disable()
  */
@@ -254,37 +228,6 @@ struct mem_cgroup *mem_cgroup_from_task(
     struct mem_cgroup, css);
 }

-static inline int page_cgroup_locked(struct page *page)
-{
- return bit_spin_is_locked(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
-}
-
-static void page_assign_page_cgroup(struct page *page, struct page_cgroup *pc)
```

```
-{
- VM_BUG_ON(!page_cgroup_locked(page));
- page->page_cgroup = ((unsigned long)pc | PAGE_CGROUP_LOCK);
-}
-
-struct page_cgroup *page_get_page_cgroup(struct page *page)
-{
- return (struct page_cgroup *) (page->page_cgroup & ~PAGE_CGROUP_LOCK);
-}
-
-static void lock_page_cgroup(struct page *page)
-{
- bit_spin_lock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
-}
-
-static int try_lock_page_cgroup(struct page *page)
-{
- return bit_spin_trylock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
-}
-
-static void unlock_page_cgroup(struct page *page)
-{
- bit_spin_unlock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
-}
-
 static void __mem_cgroup_remove_list(struct mem_cgroup_per_zone *mz,
    struct page_cgroup *pc)
 {
@@ -518,252 +461,6 @@ unsigned long mem_cgroup_isolate_pages(u
 }

 /*
- * Charge the memory controller for page usage.
- * Return
- * 0 if the charge was successful
- * < 0 if the cgroup is over its limit
- */
-static int mem_cgroup_charge_common(struct page *page, struct mm_struct *mm,
-    gfp_t gfp_mask, enum charge_type ctype,
-    struct mem_cgroup *memcg)
-{
- struct mem_cgroup *mem;
- struct page_cgroup *pc;
- unsigned long flags;
- unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
- struct mem_cgroup_per_zone *mz;
-
- if (mem_cgroup_subsys.disabled)
```

```
-  return 0;
-
- /*
-  * Should page_cgroup's go to their own slab?
-  * One could optimize the performance of the charging routine
-  * by saving a bit in the page_flags and using it as a lock
-  * to see if the cgroup page already has a page_cgroup associated
-  * with it
-  */
-retry:
- lock_page_cgroup(page);
- pc = page_get_page_cgroup(page);
- /*
-  * The page_cgroup exists and
-  * the page has already been accounted.
-  */
- if (pc) {
-  VM_BUG_ON(pc->page != page);
-  VM_BUG_ON(pc->ref_cnt <= 0);
-
-  pc->ref_cnt++;
-  unlock_page_cgroup(page);
-  goto done;
- }
- unlock_page_cgroup(page);
-
- pc = kmem_cache_alloc(page_cgroup_cache, gfp_mask);
- if (pc == NULL)
-  goto err;
-
- /*
-  * We always charge the cgroup the mm_struct belongs to.
-  * The mm_struct's mem_cgroup changes on task migration if the
-  * thread group leader migrates. It's possible that mm is not
-  * set, if so charge the init_mm (happens for pagecache usage).
-  */
- if (!memcg) {
-  if (!mm)
-   mm = &init_mm;
-
-  rcu_read_lock();
-  mem = mem_cgroup_from_task(rcu_dereference(mm->owner));
-  /*
-   * For every charge from the cgroup, increment reference count
-   */
-  css_get(&mem->css);
-  rcu_read_unlock();
- } else {
```

```
-  mem = memcg;
-  css_get(&memcg->css);
- }
-
- while (res_counter_charge(&mem->res, PAGE_SIZE)) {
-  if (!(gfp_mask & __GFP_WAIT))
-   goto out;
-
-  if (try_to_free_mem_cgroup_pages(mem, gfp_mask))
-   continue;
-
-  /*
-   * try_to_free_mem_cgroup_pages() might not give us a full
-   * picture of reclaim. Some pages are reclaimed and might be
-   * moved to swap cache or just unmapped from the cgroup.
-   * Check the limit again to see if the reclaim reduced the
-   * current usage of the cgroup before giving up
-   */
-  if (res_counter_check_under_limit(&mem->res))
-   continue;
-
-  if (!nr_retries--) {
-   mem_cgroup_out_of_memory(mem, gfp_mask);
-   goto out;
-  }
- }
-
- pc->ref_cnt = 1;
- pc->mem_cgroup = mem;
- pc->page = page;
- /*
-  * If a page is accounted as a page cache, insert to inactive list.
-  * If anon, insert to active list.
-  */
- if (ctype == MEM_CGROUP_CHARGE_TYPE_CACHE)
-  pc->flags = PAGE_CGROUP_FLAG_CACHE;
- else
-  pc->flags = PAGE_CGROUP_FLAG_ACTIVE;
-
- lock_page_cgroup(page);
- if (page_get_page_cgroup(page)) {
-  unlock_page_cgroup(page);
-  /*
-   * Another charge has been added to this page already.
-   * We take lock_page_cgroup(page) again and read
-   * page->cgroup, increment refcnt.... just retry is OK.
-   */
-  res_counter_uncharge(&mem->res, PAGE_SIZE);
```

```
-  css_put(&mem->css);
-  kmem_cache_free(page_cgroup_cache, pc);
-  goto retry;
- }
- page_assign_page_cgroup(page, pc);
-
- mz = page_cgroup_zoneinfo(pc);
- spin_lock_irqsave(&mz->lru_lock, flags);
- __mem_cgroup_add_list(mz, pc);
- spin_unlock_irqrestore(&mz->lru_lock, flags);
-
- unlock_page_cgroup(page);
-done:
- return 0;
-out:
- css_put(&mem->css);
- kmem_cache_free(page_cgroup_cache, pc);
-err:
- return -ENOMEM;
-}
-
-int mem_cgroup_charge(struct page *page, struct mm_struct *mm, gfp_t gfp_mask)
-{
- return mem_cgroup_charge_common(page, mm, gfp_mask,
-    MEM_CGROUP_CHARGE_TYPE_MAPPED, NULL);
-}
-
-int mem_cgroup_cache_charge(struct page *page, struct mm_struct *mm,
-    gfp_t gfp_mask)
-{
- if (!mm)
-  mm = &init_mm;
- return mem_cgroup_charge_common(page, mm, gfp_mask,
-    MEM_CGROUP_CHARGE_TYPE_CACHE, NULL);
-}
-
-int mem_cgroup_getref(struct page *page)
-{
- struct page_cgroup *pc;
-
- if (mem_cgroup_subsys.disabled)
-  return 0;
-
- lock_page_cgroup(page);
- pc = page_get_page_cgroup(page);
- VM_BUG_ON(!pc);
- pc->ref_cnt++;
- unlock_page_cgroup(page);
```

```
- return 0;
-}
-
-/*
- * Uncharging is always a welcome operation, we never complain, simply
- * uncharge.
- */
-void mem_cgroup_uncharge_page(struct page *page)
-{
- struct page_cgroup *pc;
- struct mem_cgroup *mem;
- struct mem_cgroup_per_zone *mz;
- unsigned long flags;
-
- if (mem_cgroup_subsys.disabled)
-  return;
-
- /*
-  * Check if our page_cgroup is valid
-  */
- lock_page_cgroup(page);
- pc = page_get_page_cgroup(page);
- if (!pc)
-  goto unlock;
-
- VM_BUG_ON(pc->page != page);
- VM_BUG_ON(pc->ref_cnt <= 0);
-
- if (--(pc->ref_cnt) == 0) {
-  mz = page_cgroup_zoneinfo(pc);
-  spin_lock_irqsave(&mz->lru_lock, flags);
-  __mem_cgroup_remove_list(mz, pc);
-  spin_unlock_irqrestore(&mz->lru_lock, flags);
-
-  page_assign_page_cgroup(page, NULL);
-  unlock_page_cgroup(page);
-
-  mem = pc->mem_cgroup;
-  res_counter_uncharge(&mem->res, PAGE_SIZE);
-  css_put(&mem->css);
-
-  kmem_cache_free(page_cgroup_cache, pc);
-  return;
- }
-
-unlock:
- unlock_page_cgroup(page);
-}
```

```
-
-/*
- * Before starting migration, account against new page.
- */
-int mem_cgroup_prepare_migration(struct page *page, struct page *newpage)
-{
- struct page_cgroup *pc;
- struct mem_cgroup *mem = NULL;
- enum charge_type ctype = MEM_CGROUP_CHARGE_TYPE_MAPPED;
- int ret = 0;
-
- if (mem_cgroup_subsys.disabled)
-  return 0;
-
- lock_page_cgroup(page);
- pc = page_get_page_cgroup(page);
- if (pc) {
-  mem = pc->mem_cgroup;
-  css_get(&mem->css);
-  if (pc->flags & PAGE_CGROUP_FLAG_CACHE)
-   ctype = MEM_CGROUP_CHARGE_TYPE_CACHE;
- }
- unlock_page_cgroup(page);
- if (mem) {
-  ret = mem_cgroup_charge_common(newpage, NULL, GFP_KERNEL,
-   ctype, mem);
-  css_put(&mem->css);
- }
- return ret;
-}
-
-/* remove redundant charge */
-void mem_cgroup_end_migration(struct page *newpage)
-{
- mem_cgroup_uncharge_page(newpage);
-}
-
-/*
  * This routine traverse page_cgroup in given list and drop them all.
  * This routine ignores page_cgroup->ref_cnt.
  * *And* this routine doesn't reclaim page itself, just removes page_cgroup.
@@ -817,7 +514,7 @@ static int mem_cgroup_force_empty(struct
  int ret = -EBUSY;
  int node, zid;

- if (mem_cgroup_subsys.disabled)
+ if (mem_cgroup_disabled())
   return 0;
```

```
  css_get(&mem->css);
@@ -1033,7 +730,7 @@ mem_cgroup_create(struct cgroup_subsys *

  if (unlikely((cont->parent) == NULL)) {
   mem = &init_mem_cgroup;
-  page_cgroup_cache = KMEM_CACHE(page_cgroup, SLAB_PANIC);
+  page_cgroup_init();
  } else {
   mem = mem_cgroup_alloc();
   if (!mem)
@@ -1077,7 +774,7 @@ static void mem_cgroup_destroy(struct cg
 static int mem_cgroup_populate(struct cgroup_subsys *ss,
     struct cgroup *cont)
 {
- if (mem_cgroup_subsys.disabled)
+ if (mem_cgroup_disabled())
   return 0;
  return cgroup_add_files(cont, ss, mem_cgroup_files,
      ARRAY_SIZE(mem_cgroup_files));
@@ -1091,7 +788,7 @@ static void mem_cgroup_move_task(struct
  struct mm_struct *mm;
  struct mem_cgroup *mem, *old_mem;

- if (mem_cgroup_subsys.disabled)
+ if (mem_cgroup_disabled())
   return;

  mm = get_task_mm(p);
@@ -1125,3 +822,310 @@ struct cgroup_subsys mem_cgroup_subsys =
  .attach = mem_cgroup_move_task,
  .early_init = 0,
 };
+
+#else /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+static inline int mem_cgroup_disabled(void)
+{
+ return 1;
+}
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+static inline int page_cgroup_locked(struct page *page)
+{
+ return bit_spin_is_locked(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
+}
+
+static void page_assign_page_cgroup(struct page *page, struct page_cgroup *pc)
```

```
+{
+ VM_BUG_ON(!page_cgroup_locked(page));
+ page->page_cgroup = ((unsigned long)pc | PAGE_CGROUP_LOCK);
+}
+
+struct page_cgroup *page_get_page_cgroup(struct page *page)
+{
+ return (struct page_cgroup *) (page->page_cgroup & ~PAGE_CGROUP_LOCK);
+}
+
+enum charge_type {
+ MEM_CGROUP_CHARGE_TYPE_CACHE = 0,
+ MEM_CGROUP_CHARGE_TYPE_MAPPED,
+};
+
+/*
+ * Charge the memory controller for page usage.
+ * Return
+ * 0 if the charge was successful
+ * < 0 if the cgroup is over its limit
+ */
+static int mem_cgroup_charge_common(struct page *page, struct mm_struct *mm,
+    gfp_t gfp_mask, enum charge_type ctype,
+    struct mem_cgroup *memcg)
+{
+ struct page_cgroup *pc;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ struct mem_cgroup *mem;
+ unsigned long flags;
+ unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
+ struct mem_cgroup_per_zone *mz;
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+ if (mem_cgroup_disabled())
+  return 0;
+
+ /*
+  * Should page_cgroup's go to their own slab?
+  * One could optimize the performance of the charging routine
+  * by saving a bit in the page_flags and using it as a lock
+  * to see if the cgroup page already has a page_cgroup associated
+  * with it
+  */
+retry:
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ /*
+  * The page_cgroup exists and
```

```
+  * the page has already been accounted.
+  */
+ if (pc) {
+  VM_BUG_ON(pc->page != page);
+  VM_BUG_ON(pc->ref_cnt <= 0);
+
+  pc->ref_cnt++;
+  unlock_page_cgroup(page);
+  goto done;
+ }
+ unlock_page_cgroup(page);
+
+ pc = kmem_cache_alloc(page_cgroup_cache, gfp_mask);
+ if (pc == NULL)
+  goto err;
+
+ /*
+  * We always charge the cgroup the mm_struct belongs to.
+  * The mm_struct's mem_cgroup changes on task migration if the
+  * thread group leader migrates. It's possible that mm is not
+  * set, if so charge the init_mm (happens for pagecache usage).
+  */
+ if (!memcg) {
+  if (!mm)
+   mm = &init_mm;
+
+  rcu_read_lock();
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+  mem = mem_cgroup_from_task(rcu_dereference(mm->owner));
+  /*
+   * For every charge from the cgroup, increment reference count
+   */
+  css_get(&mem->css);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+  rcu_read_unlock();
+ } else {
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+  mem = memcg;
+  css_get(&memcg->css);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ }
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ while (res_counter_charge(&mem->res, PAGE_SIZE)) {
+  if (!(gfp_mask & __GFP_WAIT))
+   goto out;
+
+  if (try_to_free_mem_cgroup_pages(mem, gfp_mask))
```

```
+   continue;
+
+  /*
+   * try_to_free_mem_cgroup_pages() might not give us a full
+   * picture of reclaim. Some pages are reclaimed and might be
+   * moved to swap cache or just unmapped from the cgroup.
+   * Check the limit again to see if the reclaim reduced the
+   * current usage of the cgroup before giving up
+   */
+  if (res_counter_check_under_limit(&mem->res))
+   continue;
+
+  if (!nr_retries--) {
+   mem_cgroup_out_of_memory(mem, gfp_mask);
+   goto out;
+  }
+ }
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+ pc->ref_cnt = 1;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ pc->mem_cgroup = mem;
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ pc->page = page;
+ /*
+  * If a page is accounted as a page cache, insert to inactive list.
+  * If anon, insert to active list.
+  */
+ if (ctype == MEM_CGROUP_CHARGE_TYPE_CACHE)
+  pc->flags = PAGE_CGROUP_FLAG_CACHE;
+ else
+  pc->flags = PAGE_CGROUP_FLAG_ACTIVE;
+
+ lock_page_cgroup(page);
+ if (page_get_page_cgroup(page)) {
+  unlock_page_cgroup(page);
+  /*
+   * Another charge has been added to this page already.
+   * We take lock_page_cgroup(page) again and read
+   * page->cgroup, increment refcnt.... just retry is OK.
+   */
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+  res_counter_uncharge(&mem->res, PAGE_SIZE);
+  css_put(&mem->css);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+  kmem_cache_free(page_cgroup_cache, pc);
+  goto retry;
+ }
```

```
+ page_assign_page_cgroup(page, pc);
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ mz = page_cgroup_zoneinfo(pc);
+ spin_lock_irqsave(&mz->lru_lock, flags);
+ __mem_cgroup_add_list(mz, pc);
+ spin_unlock_irqrestore(&mz->lru_lock, flags);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+ unlock_page_cgroup(page);
+done:
+ return 0;
+out:
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ css_put(&mem->css);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ kmem_cache_free(page_cgroup_cache, pc);
+err:
+ return -ENOMEM;
+}
+
+int mem_cgroup_charge(struct page *page, struct mm_struct *mm, gfp_t gfp_mask)
+{
+ return mem_cgroup_charge_common(page, mm, gfp_mask,
+    MEM_CGROUP_CHARGE_TYPE_MAPPED, NULL);
+}
+
+int mem_cgroup_cache_charge(struct page *page, struct mm_struct *mm,
+    gfp_t gfp_mask)
+{
+ if (!mm)
+  mm = &init_mm;
+ return mem_cgroup_charge_common(page, mm, gfp_mask,
+    MEM_CGROUP_CHARGE_TYPE_CACHE, NULL);
+}
+
+int mem_cgroup_getref(struct page *page)
+{
+ struct page_cgroup *pc;
+
+ if (mem_cgroup_disabled())
+  return 0;
+
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ VM_BUG_ON(!pc);
+ pc->ref_cnt++;
+ unlock_page_cgroup(page);
```

```
+ return 0;
+}
+
+/*
+ * Uncharging is always a welcome operation, we never complain, simply
+ * uncharge.
+ */
+void mem_cgroup_uncharge_page(struct page *page)
+{
+ struct page_cgroup *pc;
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ struct mem_cgroup *mem;
+ struct mem_cgroup_per_zone *mz;
+ unsigned long flags;
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+ if (mem_cgroup_disabled())
+  return;
+
+ /*
+  * Check if our page_cgroup is valid
+  */
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ if (!pc)
+  goto unlock;
+
+ VM_BUG_ON(pc->page != page);
+ VM_BUG_ON(pc->ref_cnt <= 0);
+
+ if (--(pc->ref_cnt) == 0) {
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+  mz = page_cgroup_zoneinfo(pc);
+  spin_lock_irqsave(&mz->lru_lock, flags);
+  __mem_cgroup_remove_list(mz, pc);
+  spin_unlock_irqrestore(&mz->lru_lock, flags);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+  page_assign_page_cgroup(page, NULL);
+  unlock_page_cgroup(page);
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+  mem = pc->mem_cgroup;
+  res_counter_uncharge(&mem->res, PAGE_SIZE);
+  css_put(&mem->css);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+  kmem_cache_free(page_cgroup_cache, pc);
```

```
+  return;
+ }
+
+unlock:
+ unlock_page_cgroup(page);
+}
+
+/*
+ * Before starting migration, account against new page.
+ */
+int mem_cgroup_prepare_migration(struct page *page, struct page *newpage)
+{
+ struct page_cgroup *pc;
+ struct mem_cgroup *mem = NULL;
+ enum charge_type ctype = MEM_CGROUP_CHARGE_TYPE_MAPPED;
+ int ret = 0;
+
+ if (mem_cgroup_disabled())
+  return 0;
+
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ if (pc) {
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+  mem = pc->mem_cgroup;
+  css_get(&mem->css);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+  if (pc->flags & PAGE_CGROUP_FLAG_CACHE)
+   ctype = MEM_CGROUP_CHARGE_TYPE_CACHE;
+ }
+ unlock_page_cgroup(page);
+ if (mem) {
+  ret = mem_cgroup_charge_common(newpage, NULL, GFP_KERNEL,
+   ctype, mem);
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+  css_put(&mem->css);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ }
+ return ret;
+}
+
+/* remove redundant charge */
+void mem_cgroup_end_migration(struct page *newpage)
+{
+ mem_cgroup_uncharge_page(newpage);
+}
+
+void page_cgroup_init()
```

```
+{
+ if (!page_cgroup_cache)
+  page_cgroup_cache = KMEM_CACHE(page_cgroup, SLAB_PANIC);
+}
```

_____

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