Subject: [PATCH 1/4] Block I/O tracking
Posted by Hirokazu Takahashi on Tue, 18 Mar 2008 09:25:08 GMT
View Forum Message <> Reply to Message

Hi,

This patch splits the cgroup memory subsystem into two parts.
One is for tracking which cgroup which pages and 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>

--- linux-2.6.25-rc5.pagecgroup/include/linux/memcontrol.h 2008-03-17 21:45:16.000000000 +0900
+++ linux-2.6.25-rc5-mm1/include/linux/memcontrol.h 2008-03-18 11:58:13.000000000 +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);
+}

 extern void mm_init_cgroup(struct mm_struct *mm, struct task_struct *p);
 extern void mm_free_cgroup(struct mm_struct *mm);
@@ -38,41 +87,11 @@ 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);
-
```

```
-#define mm_match_cgroup(mm, cgroup) \
- ((cgroup) == rcu_dereference((mm)->mem_cgroup))
-
 extern int mem_cgroup_prepare_migration(struct page *page);
 extern void mem_cgroup_end_migration(struct page *page);
 extern void mem_cgroup_page_migration(struct page *page, struct page *newpage);

-/*
- * 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 */
+#else /* CONFIG_CGROUP_PAGE */
 static inline void mm_init_cgroup(struct mm_struct *mm,
      struct task_struct *p)
 {
@@ -107,33 +126,69 @@ 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_page_migration(struct page *page, struct page *newpage)
 {
- return 1;
 }
+#endif /* CONFIG_CGROUP_PAGE */

-static inline int mem_cgroup_prepare_migration(struct page *page)
+
+#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);
+
+#define mm_match_cgroup(mm, cgroup) \
+ ((cgroup) == rcu_dereference((mm)->mem_cgroup))
+
+/*
+ * 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_page_migration(struct page *page, struct page *newpage)
+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)
--- linux-2.6.25-rc5.pagecgroup/include/linux/mm_types.h 2008-03-17 21:45:16.000000000 +0900
+++ linux-2.6.25-rc5-mm1/include/linux/mm_types.h 2008-03-18 11:53:35.000000000 +0900
@@ -88,7 +88,7 @@ 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
--- linux-2.6.25-rc5.pagecgroup/init/Kconfig 2008-03-17 21:45:16.000000000 +0900
+++ linux-2.6.25-rc5-mm1/init/Kconfig 2008-03-18 11:53:35.000000000 +0900
@@ -379,6 +379,10 @@ config CGROUP_MEM_RES_CTLR
    Only enable when you're ok with these trade offs and really
    sure you need the memory resource controller.

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

--- linux-2.6.25-rc5.pagecgroup/mm/Makefile 2008-03-17 21:45:16.000000000 +0900
+++ linux-2.6.25-rc5-mm1/mm/Makefile 2008-03-18 11:53:35.000000000 +0900
@@ -32,5 +32,5 @@ 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

--- linux-2.6.25-rc5.pagecgroup/mm/memcontrol.c 2008-03-17 21:45:16.000000000 +0900
```

```
+++ linux-2.6.25-rc5-mm1/mm/memcontrol.c 2008-03-18 12:05:25.000000000 +0900
@@ -33,9 +33,15 @@

 #include <asm/uaccess.h>

+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
 struct cgroup_subsys mem_cgroup_subsys;
 static const int MEM_CGROUP_RECLAIM_RETRIES = 5;

+static inline int mem_cgroup_disabled(void)
+{
+ return mem_cgroup_subsys.disabled;
+}
+
 /*
  * Statistics for memory cgroup.
  */
@@ -139,34 +145,6 @@ 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);
@@ -177,11 +155,6 @@ 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()
  */
@@ -242,51 +215,6 @@ static struct mem_cgroup *mem_cgroup_fro
     struct mem_cgroup, css);
 }

-void mm_init_cgroup(struct mm_struct *mm, struct task_struct *p)
-{
- struct mem_cgroup *mem;
-
- mem = mem_cgroup_from_task(p);
- css_get(&mem->css);
- mm->mem_cgroup = mem;
-}
-
-void mm_free_cgroup(struct mm_struct *mm)
-{
- css_put(&mm->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 page_cgroup *pc)
 {
  int from = pc->flags & PAGE_CGROUP_FLAG_ACTIVE;
@@ -519,253 +447,6 @@ 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 *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 = kzalloc(sizeof(struct page_cgroup), 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 (!mm)
-  mm = &init_mm;
-
- rcu_read_lock();
- mem = rcu_dereference(mm->mem_cgroup);
- /*
-  * For every charge from the cgroup, increment reference count
-  */
- css_get(&mem->css);
- rcu_read_unlock();
-
- 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;
-  }
-  congestion_wait(WRITE, HZ/10);
- }
-
- pc->ref_cnt = 1;
- pc->mem_cgroup = mem;
- pc->page = page;
- pc->flags = PAGE_CGROUP_FLAG_ACTIVE;
- if (ctype == MEM_CGROUP_CHARGE_TYPE_CACHE)
-  pc->flags |= PAGE_CGROUP_FLAG_CACHE;
-
- 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);
-  kfree(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(pc);
- spin_unlock_irqrestore(&mz->lru_lock, flags);
-
- unlock_page_cgroup(page);
-done:
- return 0;
-out:
- css_put(&mem->css);
- kfree(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);
-}
-
-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);
-}
-
-/*
- * 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(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);
-
-  kfree(pc);
-  return;
- }
-
-unlock:
- unlock_page_cgroup(page);
-}
-
-/*
- * Returns non-zero if a page (under migration) has valid page_cgroup member.
- * Refcnt of page_cgroup is incremented.
- */
-int mem_cgroup_prepare_migration(struct page *page)
-{
- struct page_cgroup *pc;
-
- if (mem_cgroup_subsys.disabled)
-  return 0;
-
- lock_page_cgroup(page);
- pc = page_get_page_cgroup(page);
- if (pc)
-  pc->ref_cnt++;
- unlock_page_cgroup(page);
- return pc != NULL;
-}
-
-void mem_cgroup_end_migration(struct page *page)
-{
- mem_cgroup_uncharge_page(page);
-}
-
-/*
- * We know both *page* and *newpage* are now not-on-LRU and PG_locked.
- * And no race with uncharge() routines because page_cgroup for *page*
- * has extra one reference by mem_cgroup_prepare_migration.
- */
-void mem_cgroup_page_migration(struct page *page, struct page *newpage)
-{
- struct page_cgroup *pc;
```

```
- struct mem_cgroup_per_zone *mz;
- unsigned long flags;
-
- lock_page_cgroup(page);
- pc = page_get_page_cgroup(page);
- if (!pc) {
-  unlock_page_cgroup(page);
-  return;
- }
-
- mz = page_cgroup_zoneinfo(pc);
- spin_lock_irqsave(&mz->lru_lock, flags);
- __mem_cgroup_remove_list(pc);
- spin_unlock_irqrestore(&mz->lru_lock, flags);
-
- page_assign_page_cgroup(page, NULL);
- unlock_page_cgroup(page);
-
- pc->page = newpage;
- lock_page_cgroup(newpage);
- page_assign_page_cgroup(newpage, pc);
-
- mz = page_cgroup_zoneinfo(pc);
- spin_lock_irqsave(&mz->lru_lock, flags);
- __mem_cgroup_add_list(pc);
- spin_unlock_irqrestore(&mz->lru_lock, flags);
-
- unlock_page_cgroup(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.
@@ -812,7 +493,7 @@ 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);
@@ -1034,7 +715,7 @@ 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));
@@ -1048,7 +729,7 @@ 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);
@@ -1086,3 +767,319 @@ 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 */
+
+void mm_init_cgroup(struct mm_struct *mm, struct task_struct *p)
+{
+ struct mem_cgroup *mem;
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ mem = mem_cgroup_from_task(p);
+ css_get(&mem->css);
+ mm->mem_cgroup = mem;
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+}
+
+void mm_free_cgroup(struct mm_struct *mm)
+{
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ css_put(&mm->mem_cgroup->css);
+#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 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 = kzalloc(sizeof(struct page_cgroup), 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 (!mm)
+  mm = &init_mm;
+
+ rcu_read_lock();
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ mem = rcu_dereference(mm->mem_cgroup);
+ /*
+  * For every charge from the cgroup, increment reference count
+  */
+ css_get(&mem->css);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+ rcu_read_unlock();
+
+#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;
+  }
+  congestion_wait(WRITE, HZ/10);
+ }
+#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;
+ pc->flags = PAGE_CGROUP_FLAG_ACTIVE;
+ if (ctype == MEM_CGROUP_CHARGE_TYPE_CACHE)
+  pc->flags |= PAGE_CGROUP_FLAG_CACHE;
+
+ 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 */
+  kfree(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(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 */
+ kfree(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);
+}
+
+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);
+}
+
+/*
+ * 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_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(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 */
+
+  kfree(pc);
+  return;
+ }
+
+unlock:
+ unlock_page_cgroup(page);
+}
+
+/*
+ * Returns non-zero if a page (under migration) has valid page_cgroup member.
+ * Refcnt of page_cgroup is incremented.
+ */
+int mem_cgroup_prepare_migration(struct page *page)
+{
+ struct page_cgroup *pc;
+
+ if (mem_cgroup_disabled())
+  return 0;
+
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ if (pc)
+  pc->ref_cnt++;
+ unlock_page_cgroup(page);
+ return pc != NULL;
+}
+
+void mem_cgroup_end_migration(struct page *page)
+{
```

```
+ mem_cgroup_uncharge_page(page);
+}
+
+/*
+ * We know both *page* and *newpage* are now not-on-LRU and PG_locked.
+ * And no race with uncharge() routines because page_cgroup for *page*
+ * has extra one reference by mem_cgroup_prepare_migration.
+ */
+void mem_cgroup_page_migration(struct page *page, struct page *newpage)
+{
+ struct page_cgroup *pc;
+ struct mem_cgroup_per_zone *mz;
+ unsigned long flags;
+
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ if (!pc) {
+  unlock_page_cgroup(page);
+  return;
+ }
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ mz = page_cgroup_zoneinfo(pc);
+ spin_lock_irqsave(&mz->lru_lock, flags);
+ __mem_cgroup_remove_list(pc);
+ spin_unlock_irqrestore(&mz->lru_lock, flags);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+ page_assign_page_cgroup(page, NULL);
+ unlock_page_cgroup(page);
+
+ pc->page = newpage;
+ lock_page_cgroup(newpage);
+ page_assign_page_cgroup(newpage, pc);
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+ mz = page_cgroup_zoneinfo(pc);
+ spin_lock_irqsave(&mz->lru_lock, flags);
+ __mem_cgroup_add_list(pc);
+ spin_unlock_irqrestore(&mz->lru_lock, flags);
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+ unlock_page_cgroup(newpage);
+}
+
```

Containers mailing list