Subject: [PATCH 3/4] BIO tracking take2
Posted by Hirokazu Takahashi on Tue, 20 May 2008 12:03:24 GMT
View Forum Message <> Reply to Message

Hi,

This patch implements the bio cgroup on the memory cgroup.

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

```
diff -dupr linux-2.6.26-rc2.cg2/block/blk-ioc.c linux-2.6.26-rc2/block/blk-ioc.c
--- linux-2.6.26-rc2.cg2/block/blk-ioc.c 2008-05-19 13:51:22.000000000 +0900
+++ linux-2.6.26-rc2/block/blk-ioc.c 2008-05-19 18:40:10.000000000 +0900
@@ -84,24 +84,28 @@ void exit_io_context(void)
  }
 }

+void init_io_context(struct io_context *ioc)
+{
+ atomic_set(&ioc->refcount, 1);
+ atomic_set(&ioc->nr_tasks, 1);
+ spin_lock_init(&ioc->lock);
+ ioc->ioprio_changed = 0;
+ ioc->ioprio = 0;
+ ioc->last_waited = jiffies; /* doesn't matter... */
+ ioc->nr_batch_requests = 0; /* because this is 0 */
+ ioc->aic = NULL;
+ INIT_RADIX_TREE(&ioc->radix_root, GFP_ATOMIC | __GFP_HIGH);
+ INIT_HLIST_HEAD(&ioc->cic_list);
+ ioc->ioc_data = NULL;
+}
+
 struct io_context *alloc_io_context(gfp_t gfp_flags, int node)
 {
  struct io_context *ret;

  ret = kmem_cache_alloc_node(iocontext_cachep, gfp_flags, node);
- if (ret) {
-  atomic_set(&ret->refcount, 1);
-  atomic_set(&ret->nr_tasks, 1);
-  spin_lock_init(&ret->lock);
-  ret->ioprio_changed = 0;
-  ret->ioprio = 0;
-  ret->last_waited = jiffies; /* doesn't matter... */
-  ret->nr_batch_requests = 0; /* because this is 0 */
-  ret->aic = NULL;
-  INIT_RADIX_TREE(&ret->radix_root, GFP_ATOMIC | __GFP_HIGH);
```

```
-  INIT_HLIST_HEAD(&ret->cic_list);
-  ret->ioc_data = NULL;
- }
+ if (ret)
+  init_io_context(ret);

  return ret;
 }
diff -dupr linux-2.6.26-rc2.cg2/include/linux/biocontrol.h linux-2.6.26-rc2/include/linux/biocontrol.h
--- linux-2.6.26-rc2.cg2/include/linux/biocontrol.h 2008-05-19 13:51:21.000000000 +0900
+++ linux-2.6.26-rc2/include/linux/biocontrol.h 2008-05-19 18:40:10.000000000 +0900
@@ -0,0 +1,160 @@
+#include <linux/cgroup.h>
+#include <linux/mm.h>
+#include <linux/memcontrol.h>
+
+#ifndef _LINUX_BIOCONTROL_H
+#define _LINUX_BIOCONTROL_H
+
+#ifdef CONFIG_CGROUP_BIO
+
+struct io_context;
+struct block_device;
+
+struct bio_cgroup {
+ struct cgroup_subsys_state css;
+ int id;
+ struct io_context *io_context; /* default io_context */
+/* struct radix_tree_root io_context_root; per device io_context */
+ spinlock_t  page_list_lock;
+ struct list_head page_list;
+};
+
+static inline int bio_cgroup_disabled(void)
+{
+ return bio_cgroup_subsys.disabled;
+}
+
+static inline struct bio_cgroup *bio_cgroup_from_task(struct task_struct *p)
+{
+ return container_of(task_subsys_state(p, bio_cgroup_subsys_id),
+     struct bio_cgroup, css);
+}
+
+static inline void __bio_cgroup_add_page(struct page_cgroup *pc)
+{
+ struct bio_cgroup *biog = pc->bio_cgroup;
+ list_add(&pc->blist, &biog->page_list);
```

```
+}
+
+static inline void bio_cgroup_add_page(struct page_cgroup *pc)
+{
+ struct bio_cgroup *biog = pc->bio_cgroup;
+ unsigned long flags;
+ spin_lock_irqsave(&biog->page_list_lock, flags);
+ __bio_cgroup_add_page(pc);
+ spin_unlock_irqrestore(&biog->page_list_lock, flags);
+}
+
+static inline void __bio_cgroup_remove_page(struct page_cgroup *pc)
+{
+ list_del_init(&pc->blist);
+}
+
+static inline void bio_cgroup_remove_page(struct page_cgroup *pc)
+{
+ struct bio_cgroup *biog = pc->bio_cgroup;
+ unsigned long flags;
+ spin_lock_irqsave(&biog->page_list_lock, flags);
+ __bio_cgroup_remove_page(pc);
+ spin_unlock_irqrestore(&biog->page_list_lock, flags);
+}
+
+static inline void get_bio_cgroup(struct bio_cgroup *biog)
+{
+ css_get(&biog->css);
+}
+
+static inline void put_bio_cgroup(struct bio_cgroup *biog)
+{
+ css_put(&biog->css);
+}
+
+static inline void set_bio_cgroup(struct page_cgroup *pc,
+     struct bio_cgroup *biog)
+{
+ pc->bio_cgroup = biog;
+}
+
+static inline void clear_bio_cgroup(struct page_cgroup *pc)
+{
+ struct bio_cgroup *biog = pc->bio_cgroup;
+ pc->bio_cgroup = NULL;
+ put_bio_cgroup(biog);
+}
+
```

```
+static inline struct bio_cgroup *get_bio_page_cgroup(struct page_cgroup *pc)
+{
+ struct bio_cgroup *biog = pc->bio_cgroup;
+ css_get(&biog->css);
+ return biog;
+}
+
+/* This sould be called in an RCU-protected section. */
+static inline struct bio_cgroup *mm_get_bio_cgroup(struct mm_struct *mm)
+{
+ struct bio_cgroup *biog;
+ biog = bio_cgroup_from_task(rcu_dereference(mm->owner));
+ get_bio_cgroup(biog);
+ return biog;
+}
+
+//extern int get_bio_cgroup_id(struct page *page);
+extern struct io_context *get_bio_cgroup_iocontext(struct bio *bio);
+
+#else /* CONFIG_CGROUP_BIO */
+
+struct bio_cgroup;
+
+static inline int bio_cgroup_disabled(void)
+{
+       return 1;
+}
+
+static inline void bio_cgroup_add_page(struct page_cgroup *pc)
+{
+}
+
+static inline void bio_cgroup_remove_page(struct page_cgroup *pc)
+{
+}
+
+static inline void get_bio_cgroup(struct bio_cgroup *biog)
+{
+}
+
+static inline void put_bio_cgroup(struct bio_cgroup *biog)
+{
+}
+
+static inline void set_bio_cgroup(struct page_cgroup *pc,
+     struct bio_cgroup *biog)
+{
+}
```

```
+
+static inline void clear_bio_cgroup(struct page_cgroup *pc)
+{
+}
+
+static inline struct bio_cgroup *get_bio_page_cgroup(struct page_cgroup *pc)
+{
+ return NULL;
+}
+
+static inline struct bio_cgroup *mm_get_bio_cgroup(struct mm_struct *mm)
+{
+ return NULL;
+}
+
+static inline int get_bio_cgroup_id(struct page *page)
+{
+ return 0;
+}
+
+static inline struct io_context *get_bio_cgroup_iocontext(struct bio *bio)
+{
+ return NULL;
+}
+
+#endif /* CONFIG_CGROUP_BIO */
+
+#endif /* _LINUX_BIOCONTROL_H */
diff -dupr linux-2.6.26-rc2.cg2/include/linux/blkdev.h linux-2.6.26-rc2/include/linux/blkdev.h
--- linux-2.6.26-rc2.cg2/include/linux/blkdev.h 2008-05-19 13:51:22.000000000 +0900
+++ linux-2.6.26-rc2/include/linux/blkdev.h 2008-05-19 18:40:10.000000000 +0900
@@ -38,6 +38,7 @@ int put_io_context(struct io_context *io
 void exit_io_context(void);
 struct io_context *get_io_context(gfp_t gfp_flags, int node);
 struct io_context *alloc_io_context(gfp_t gfp_flags, int node);
+void init_io_context(struct io_context *ioc);
 void copy_io_context(struct io_context **pdst, struct io_context **psrc);

 struct request;
diff -dupr linux-2.6.26-rc2.cg2/include/linux/cgroup_subsys.h
linux-2.6.26-rc2/include/linux/cgroup_subsys.h
--- linux-2.6.26-rc2.cg2/include/linux/cgroup_subsys.h 2008-05-19 13:51:21.000000000 +0900
+++ linux-2.6.26-rc2/include/linux/cgroup_subsys.h 2008-05-19 18:40:10.000000000 +0900
@@ -43,6 +43,12 @@ SUBSYS(mem_cgroup)

 /* */

+#ifdef CONFIG_CGROUP_BIO
```

```
+SUBSYS(bio_cgroup)
+#endif
+
+/* */
+
 #ifdef CONFIG_CGROUP_DEVICE
 SUBSYS(devices)
 #endif
diff -dupr linux-2.6.26-rc2.cg2/include/linux/iocontext.h linux-2.6.26-rc2/include/linux/iocontext.h
--- linux-2.6.26-rc2.cg2/include/linux/iocontext.h 2008-05-19 13:51:22.000000000 +0900
+++ linux-2.6.26-rc2/include/linux/iocontext.h 2008-05-19 18:40:10.000000000 +0900
@@ -83,6 +83,8 @@ struct io_context {
  struct radix_tree_root radix_root;
  struct hlist_head cic_list;
  void *ioc_data;
+
+ int id;  /* cgroup ID */
 };

 static inline struct io_context *ioc_task_link(struct io_context *ioc)
diff -dupr linux-2.6.26-rc2.cg2/include/linux/memcontrol.h
linux-2.6.26-rc2/include/linux/memcontrol.h
--- linux-2.6.26-rc2.cg2/include/linux/memcontrol.h 2008-05-19 13:51:21.000000000 +0900
+++ linux-2.6.26-rc2/include/linux/memcontrol.h 2008-05-19 18:40:10.000000000 +0900
@@ -54,6 +54,10 @@ struct page_cgroup {
  struct list_head lru;  /* per cgroup LRU list */
  struct mem_cgroup *mem_cgroup;
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+#ifdef  CONFIG_CGROUP_BIO
+ struct list_head blist;  /* for bio_cgroup page list */
+ struct bio_cgroup *bio_cgroup;
+#endif
  struct page *page;
  int ref_cnt;   /* cached, mapped, migrating */
  int flags;
diff -dupr linux-2.6.26-rc2.cg2/init/Kconfig linux-2.6.26-rc2/init/Kconfig
--- linux-2.6.26-rc2.cg2/init/Kconfig 2008-05-19 13:51:22.000000000 +0900
+++ linux-2.6.26-rc2/init/Kconfig 2008-05-19 18:40:10.000000000 +0900
@@ -407,9 +407,20 @@ 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_BIO
+ bool "Block I/O cgroup subsystem"
+ depends on CGROUPS
+ select MM_OWNER
+ help
+   Provides a Resource Controller which enables to track the onwner
```

```
+   of every Block I/O.
+   The information this subsystem provides can be used from any
+   kind of module such as dm-ioband device mapper modules or
+   the cfq-scheduler.
+
 config CGROUP_PAGE
   def_bool y
- depends on CGROUP_MEM_RES_CTLR
+ depends on CGROUP_MEM_RES_CTLR || CGROUP_BIO

 config SYSFS_DEPRECATED
   bool
diff -dupr linux-2.6.26-rc2.cg2/mm/biocontrol.c linux-2.6.26-rc2/mm/biocontrol.c
--- linux-2.6.26-rc2.cg2/mm/biocontrol.c 2008-05-19 13:51:22.000000000 +0900
+++ linux-2.6.26-rc2/mm/biocontrol.c 2008-05-19 20:51:01.000000000 +0900
@@ -0,0 +1,233 @@
+/* biocontrol.c - Block I/O Controller
+ *
+ * Copyright IBM Corporation, 2007
+ * Author Balbir Singh <balbir@linux.vnet.ibm.com>
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
+ * Copyright VA Linux Systems Japan, 2008
+ * Author Hirokazu Takahashi <taka@valinux.co.jp>
+ *
+ * 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/module.h>
+#include <linux/cgroup.h>
+#include <linux/mm.h>
+#include <linux/blkdev.h>
+#include <linux/smp.h>
+#include <linux/bit_spinlock.h>
+#include <linux/idr.h>
+#include <linux/err.h>
+#include <linux/biocontrol.h>
+
```

```
+/* return corresponding bio_cgroup object of a cgroup */
+static inline struct bio_cgroup *cgroup_bio(struct cgroup *cgrp)
+{
+ return container_of(cgroup_subsys_state(cgrp, bio_cgroup_subsys_id),
+       struct bio_cgroup, css);
+}
+
+static struct idr bio_cgroup_id;
+static DEFINE_SPINLOCK(bio_cgroup_idr_lock);
+
+static struct cgroup_subsys_state *
+bio_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cgrp)
+{
+ struct bio_cgroup *biog;
+ struct io_context *ioc;
+ int error;
+
+ if (!cgrp->parent) {
+  static struct bio_cgroup default_bio_cgroup;
+  static struct io_context default_bio_io_context;
+
+  biog = &default_bio_cgroup;
+  ioc = &default_bio_io_context;
+  init_io_context(ioc);
+
+  idr_init(&bio_cgroup_id);
+  biog->id = 0;
+
+  page_cgroup_init();
+ } else {
+  biog = kzalloc(sizeof(*biog), GFP_KERNEL);
+  ioc = alloc_io_context(GFP_KERNEL, -1);
+  if (!ioc || !biog) {
+   error = -ENOMEM;
+   goto out;
+  }
+retry:
+  if (unlikely(!idr_pre_get(&bio_cgroup_id, GFP_KERNEL))) {
+   error = -EAGAIN;
+   goto out;
+  }
+  spin_lock_irq(&bio_cgroup_idr_lock);
+  error = idr_get_new_above(&bio_cgroup_id, (void *)biog, 1, &biog->id);
+  spin_unlock_irq(&bio_cgroup_idr_lock);
+  if (error == -EAGAIN)
+   goto retry;
+  else if (error)
+   goto out;
```

```
+ }
+
+ ioc->id = biog->id;
+ biog->io_context = ioc;
+
+ INIT_LIST_HEAD(&biog->page_list);
+ spin_lock_init(&biog->page_list_lock);
+
+ /* Bind the cgroup to bio_cgroup object we just created */
+ biog->css.cgroup = cgrp;
+
+ return &biog->css;
+out:
+ if (ioc)
+  put_io_context(ioc);
+ if (biog)
+  kfree(biog);
+ return ERR_PTR(error);
+}
+
+#define FORCE_UNCHARGE_BATCH (128)
+static void bio_cgroup_force_empty(struct bio_cgroup *biog)
+{
+ struct page_cgroup *pc;
+ struct page *page;
+ int count = FORCE_UNCHARGE_BATCH;
+ struct list_head *list = &biog->page_list;
+ unsigned long flags;
+
+ spin_lock_irqsave(&biog->page_list_lock, flags);
+ while (!list_empty(list)) {
+  pc = list_entry(list->prev, struct page_cgroup, blist);
+  page = pc->page;
+  get_page(page);
+  spin_unlock_irqrestore(&biog->page_list_lock, flags);
+  mem_cgroup_uncharge_page(page);
+  put_page(page);
+  if (--count <= 0) {
+   count = FORCE_UNCHARGE_BATCH;
+   cond_resched();
+  }
+  spin_lock_irqsave(&biog->page_list_lock, flags);
+ }
+ spin_unlock_irqrestore(&biog->page_list_lock, flags);
+ return;
+}
+
+static void bio_cgroup_pre_destroy(struct cgroup_subsys *ss, struct cgroup *cgrp)
```

```
+{
+ struct bio_cgroup *biog = cgroup_bio(cgrp);
+ bio_cgroup_force_empty(biog);
+}
+
+static void bio_cgroup_destroy(struct cgroup_subsys *ss, struct cgroup *cgrp)
+{
+ struct bio_cgroup *biog = cgroup_bio(cgrp);
+
+ put_io_context(biog->io_context);
+
+       spin_lock_irq(&bio_cgroup_idr_lock);
+       idr_remove(&bio_cgroup_id, biog->id);
+       spin_unlock_irq(&bio_cgroup_idr_lock);
+
+ kfree(biog);
+}
+
+struct bio_cgroup *find_bio_cgroup(int id)
+{
+ struct bio_cgroup *biog;
+       spin_lock_irq(&bio_cgroup_idr_lock);
+ biog = (struct bio_cgroup *)
+       idr_find(&bio_cgroup_id, id);
+       spin_unlock_irq(&bio_cgroup_idr_lock);
+ get_bio_cgroup(biog);
+ return biog;
+}
+
+struct io_context *get_bio_cgroup_iocontext(struct bio *bio)
+{
+ struct io_context *ioc;
+ struct page_cgroup *pc;
+ struct bio_cgroup *biog;
+ struct page *page = bio_iovec_idx(bio, 0)->bv_page;
+
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ if (pc)
+  biog = pc->bio_cgroup;
+ else
+  biog = bio_cgroup_from_task(rcu_dereference(init_mm.owner));
+ ioc = biog->io_context; /* default io_context for this cgroup */
+ atomic_inc(&ioc->refcount);
+ unlock_page_cgroup(page);
+ return ioc;
+}
+EXPORT_SYMBOL(get_bio_cgroup_iocontext);
```

```
+
+static u64 bio_id_read(struct cgroup *cgrp, struct cftype *cft)
+{
+ struct bio_cgroup *biog = cgroup_bio(cgrp);
+
+ return (u64) biog->id;
+}
+
+
+static struct cftype bio_files[] = {
+ {
+  .name = "id",
+  .read_u64 = bio_id_read,
+ },
+};
+
+static int bio_cgroup_populate(struct cgroup_subsys *ss, struct cgroup *cont)
+{
+ if (bio_cgroup_disabled())
+  return 0;
+ return cgroup_add_files(cont, ss, bio_files, ARRAY_SIZE(bio_files));
+}
+
+static void bio_cgroup_move_task(struct cgroup_subsys *ss,
+    struct cgroup *cont,
+    struct cgroup *old_cont,
+    struct task_struct *p)
+{
+ struct mm_struct *mm;
+ struct bio_cgroup *biog, *old_biog;
+
+ if (bio_cgroup_disabled())
+  return;
+
+ mm = get_task_mm(p);
+ if (mm == NULL)
+  return;
+
+ biog = cgroup_bio(cont);
+ old_biog = cgroup_bio(old_cont);
+
+ mmput(mm);
+ return;
+}
+
+
+struct cgroup_subsys bio_cgroup_subsys = {
+ .name           = "bio",
```

```
+ .subsys_id      = bio_cgroup_subsys_id,
+ .create         = bio_cgroup_create,
+ .destroy        = bio_cgroup_destroy,
+ .pre_destroy = bio_cgroup_pre_destroy,
+// .can_attach     = bio_cgroup_can_attach,
+ .populate       = bio_cgroup_populate,
+ .attach  = bio_cgroup_move_task,
+ .early_init = 0,
+};
diff -dupr linux-2.6.26-rc2.cg2/mm/Makefile linux-2.6.26-rc2/mm/Makefile
--- linux-2.6.26-rc2.cg2/mm/Makefile 2008-05-19 13:51:22.000000000 +0900
+++ linux-2.6.26-rc2/mm/Makefile 2008-05-19 18:40:10.000000000 +0900
@@ -34,4 +34,5 @@ obj-$(CONFIG_MIGRATION) += migrate.o
 obj-$(CONFIG_SMP) += allocpercpu.o
 obj-$(CONFIG_QUICKLIST) += quicklist.o
 obj-$(CONFIG_CGROUP_PAGE) += memcontrol.o
+obj-$(CONFIG_CGROUP_BIO) += biocontrol.o

diff -dupr linux-2.6.26-rc2.cg2/mm/memcontrol.c linux-2.6.26-rc2/mm/memcontrol.c
--- linux-2.6.26-rc2.cg2/mm/memcontrol.c 2008-05-19 13:51:22.000000000 +0900
+++ linux-2.6.26-rc2/mm/memcontrol.c 2008-05-19 18:40:10.000000000 +0900
@@ -20,6 +20,7 @@
 #include <linux/res_counter.h>
 #include <linux/memcontrol.h>
 #include <linux/cgroup.h>
+#include <linux/biocontrol.h>
 #include <linux/mm.h>
 #include <linux/smp.h>
 #include <linux/page-flags.h>
@@ -978,12 +979,13 @@ enum charge_type {
  */
 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 *memcg, struct bio_cgroup *biocg)
 {
  struct page_cgroup *pc;
  struct mem_cgroup *mem;
+ struct bio_cgroup *biog;

- if (mem_cgroup_disabled())
+ if (mem_cgroup_disabled() && bio_cgroup_disabled())
   return 0;

  /*
@@ -1020,23 +1022,19 @@ retry:
   * 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 = mm_get_mem_cgroup(mm);
-  rcu_read_unlock();
- } else {
-  mem = memcg;
-  get_mem_cgroup(mem);
- }
+ if (!mm)
+  mm = &init_mm;
+ rcu_read_lock();
+ mem = memcg ? memcg : mm_get_mem_cgroup(mm);
+ biog = biocg ? biocg : mm_get_bio_cgroup(mm);
+ rcu_read_unlock();

  if (mem_cgroup_try_to_allocate(mem, gfp_mask) < 0)
   goto out;

  pc->ref_cnt = 1;
  set_mem_cgroup(pc, mem);
+ set_bio_cgroup(pc, biog);
  pc->page = page;
  /*
   * If a page is accounted as a page cache, insert to inactive list.
@@ -1056,18 +1054,21 @@ retry:
    * page->cgroup, increment refcnt.... just retry is OK.
    */
   clear_mem_cgroup(pc);
+  clear_bio_cgroup(pc);
   kmem_cache_free(page_cgroup_cache, pc);
   goto retry;
  }
  page_assign_page_cgroup(page, pc);

  mem_cgroup_add_page(pc);
+ bio_cgroup_add_page(pc);

  unlock_page_cgroup(page);
 done:
  return 0;
 out:
  put_mem_cgroup(mem);
+ put_bio_cgroup(biog);
  kmem_cache_free(page_cgroup_cache, pc);
```

```
 err:
  return -ENOMEM;
@@ -1076,7 +1077,7 @@ err:
 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);
+     MEM_CGROUP_CHARGE_TYPE_MAPPED, NULL, NULL);
 }

 int mem_cgroup_cache_charge(struct page *page, struct mm_struct *mm,
@@ -1085,7 +1086,7 @@ int mem_cgroup_cache_charge(struct page
   if (!mm)
    mm = &init_mm;
   return mem_cgroup_charge_common(page, mm, gfp_mask,
-    MEM_CGROUP_CHARGE_TYPE_CACHE, NULL);
+     MEM_CGROUP_CHARGE_TYPE_CACHE, NULL, NULL);
 }

 int mem_cgroup_getref(struct page *page)
@@ -1111,7 +1112,7 @@ void mem_cgroup_uncharge_page(struct pag
 {
  struct page_cgroup *pc;

- if (mem_cgroup_disabled())
+ if (mem_cgroup_disabled() && bio_cgroup_disabled())
   return;

  /*
@@ -1127,11 +1128,13 @@ void mem_cgroup_uncharge_page(struct pag

  if (--(pc->ref_cnt) == 0) {
   mem_cgroup_remove_page(pc);
+  bio_cgroup_remove_page(pc);

   page_assign_page_cgroup(page, NULL);
   unlock_page_cgroup(page);

   clear_mem_cgroup(pc);
+  clear_bio_cgroup(pc);

   kmem_cache_free(page_cgroup_cache, pc);
   return;
@@ -1148,24 +1151,29 @@ int mem_cgroup_prepare_migration(struct
 {
  struct page_cgroup *pc;
  struct mem_cgroup *mem = NULL;
+ struct bio_cgroup *biog = NULL;
```

```
  enum charge_type ctype = MEM_CGROUP_CHARGE_TYPE_MAPPED;
  int ret = 0;

- if (mem_cgroup_disabled())
+ if (mem_cgroup_disabled() && bio_cgroup_disabled())
   return 0;

  lock_page_cgroup(page);
  pc = page_get_page_cgroup(page);
  if (pc) {
   mem = get_mem_page_cgroup(pc);
+  biog = get_bio_page_cgroup(pc);
   if (pc->flags & PAGE_CGROUP_FLAG_CACHE)
    ctype = MEM_CGROUP_CHARGE_TYPE_CACHE;
  }
  unlock_page_cgroup(page);
- if (mem) {
+ if (pc) {
   ret = mem_cgroup_charge_common(newpage, NULL, GFP_KERNEL,
-   ctype, mem);
-  put_mem_cgroup(mem);
+   ctype, mem, biog);
+  if (mem)
+   put_mem_cgroup(mem);
+  if (biog)
+   put_bio_cgroup(biog);
  }
  return ret;
 }
```

_____

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