Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by yamamoto on Thu, 27 Mar 2008 06:28:34 GMT
View Forum Message <> Reply to Message

hi,

i tried to reproduce the large swap cache issue, but no luck.
can you provide a little more detailed instruction?

the following is a new version of the patch.
changes from the previous:
 - fix a bug; update accounting for mremap properly.
 - some comments and assertions.

YAMAMOTO Takashi

Signed-off-by: YAMAMOTO Takashi <yamamoto@valinux.co.jp>

```
--- linux-2.6.25-rc3-mm1/init/Kconfig.BACKUP 2008-03-05 15:45:50.000000000 +0900
+++ linux-2.6.25-rc3-mm1/init/Kconfig 2008-03-12 11:52:48.000000000 +0900
@@ -379,6 +379,12 @@ 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_SWAP_RES_CTLR
+ bool "Swap Resource Controller for Control Groups"
+ depends on CGROUPS && RESOURCE_COUNTERS
+ help
+   XXX TBD
+
 config SYSFS_DEPRECATED
  bool "Create deprecated sysfs files"
  depends on SYSFS
--- linux-2.6.25-rc3-mm1/mm/swapfile.c.BACKUP 2008-03-05 15:45:52.000000000 +0900
+++ linux-2.6.25-rc3-mm1/mm/swapfile.c 2008-03-14 17:25:40.000000000 +0900
@@ -28,6 +28,7 @@
 #include <linux/capability.h>
 #include <linux/syscalls.h>
 #include <linux/memcontrol.h>
+#include <linux/swapcontrol.h>

 #include <asm/pgtable.h>
 #include <asm/tlbflush.h>
@@ -526,6 +527,7 @@ static int unuse_pte(struct vm_area_stru
  }
```

```
  inc_mm_counter(vma->vm_mm, anon_rss);
+ swap_cgroup_uncharge(pmd_page(*pmd));
  get_page(page);
  set_pte_at(vma->vm_mm, addr, pte,
      pte_mkold(mk_pte(page, vma->vm_page_prot)));
--- linux-2.6.25-rc3-mm1/mm/Makefile.BACKUP 2008-03-05 15:45:51.000000000 +0900
+++ linux-2.6.25-rc3-mm1/mm/Makefile 2008-03-12 11:53:31.000000000 +0900
@@ -33,4 +33,5 @@ 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_SWAP_RES_CTLR) += swapcontrol.o

--- linux-2.6.25-rc3-mm1/mm/rmap.c.BACKUP 2008-03-05 15:45:52.000000000 +0900
+++ linux-2.6.25-rc3-mm1/mm/rmap.c 2008-03-17 07:45:16.000000000 +0900
@@ -49,6 +49,7 @@
 #include <linux/module.h>
 #include <linux/kallsyms.h>
 #include <linux/memcontrol.h>
+#include <linux/swapcontrol.h>

 #include <asm/tlbflush.h>

@@ -237,8 +238,9 @@ unsigned long page_address_in_vma(struct
  *
  * On success returns with pte mapped and locked.
  */
-pte_t *page_check_address(struct page *page, struct mm_struct *mm,
-     unsigned long address, spinlock_t **ptlp)
+pte_t *page_check_address1(struct page *page, struct mm_struct *mm,
+     unsigned long address, spinlock_t **ptlp,
+     struct page **ptpp)
 {
  pgd_t *pgd;
  pud_t *pud;
@@ -269,12 +271,21 @@ pte_t *page_check_address(struct page *p
  spin_lock(ptl);
  if (pte_present(*pte) && page_to_pfn(page) == pte_pfn(*pte)) {
   *ptlp = ptl;
+  if (ptpp != NULL) {
+   *ptpp = pmd_page(*(pmd));
+  }
   return pte;
  }
  pte_unmap_unlock(pte, ptl);
  return NULL;
 }
```

```
+pte_t *page_check_address(struct page *page, struct mm_struct *mm,
+     unsigned long address, spinlock_t **ptlp)
+{
+ return page_check_address1(page, mm, address, ptlp, NULL);
+}
+
 /*
  * Subfunctions of page_referenced: page_referenced_one called
  * repeatedly from either page_referenced_anon or page_referenced_file.
@@ -710,13 +721,14 @@ static int try_to_unmap_one(struct page
  pte_t *pte;
  pte_t pteval;
  spinlock_t *ptl;
+ struct page *ptp;
  int ret = SWAP_AGAIN;

  address = vma_address(page, vma);
  if (address == -EFAULT)
   goto out;

- pte = page_check_address(page, mm, address, &ptl);
+ pte = page_check_address1(page, mm, address, &ptl, &ptp);
  if (!pte)
   goto out;

@@ -731,6 +743,12 @@ static int try_to_unmap_one(struct page
   goto out_unmap;
  }

+ if (!migration && PageSwapCache(page) && swap_cgroup_charge(ptp, mm)) {
+  /* XXX should make the caller free the swap slot? */
+  ret = SWAP_FAIL;
+  goto out_unmap;
+ }
+
  /* Nuke the page table entry. */
  flush_cache_page(vma, address, page_to_pfn(page));
  pteval = ptep_clear_flush(vma, address, pte);
--- linux-2.6.25-rc3-mm1/mm/memory.c.BACKUP 2008-03-05 15:45:52.000000000 +0900
+++ linux-2.6.25-rc3-mm1/mm/memory.c 2008-03-14 18:54:21.000000000 +0900
@@ -51,6 +51,7 @@
 #include <linux/init.h>
 #include <linux/writeback.h>
 #include <linux/memcontrol.h>
+#include <linux/swapcontrol.h>

 #include <asm/pgalloc.h>
 #include <asm/uaccess.h>
```

```
@@ -431,10 +432,10 @@ struct page *vm_normal_page(struct vm_ar
  * covered by this vma.
  */

-static inline void
+static inline int
 copy_one_pte(struct mm_struct *dst_mm, struct mm_struct *src_mm,
   pte_t *dst_pte, pte_t *src_pte, struct vm_area_struct *vma,
-  unsigned long addr, int *rss)
+  unsigned long addr, int *rss, struct page *dst_ptp)
 {
  unsigned long vm_flags = vma->vm_flags;
  pte_t pte = *src_pte;
@@ -445,6 +446,11 @@ copy_one_pte(struct mm_struct *dst_mm, s
   if (!pte_file(pte)) {
    swp_entry_t entry = pte_to_swp_entry(pte);

+   if (!is_write_migration_entry(entry) &&
+       swap_cgroup_charge(dst_ptp, dst_mm)) {
+    return -ENOMEM;
+   }
+
    swap_duplicate(entry);
    /* make sure dst_mm is on swapoff's mmlist. */
    if (unlikely(list_empty(&dst_mm->mmlist))) {
@@ -494,6 +500,7 @@ copy_one_pte(struct mm_struct *dst_mm, s

 out_set_pte:
  set_pte_at(dst_mm, addr, dst_pte, pte);
+ return 0;
 }

 static int copy_pte_range(struct mm_struct *dst_mm, struct mm_struct *src_mm,
@@ -504,6 +511,8 @@ static int copy_pte_range(struct mm_stru
  spinlock_t *src_ptl, *dst_ptl;
  int progress = 0;
  int rss[2];
+ struct page *dst_ptp;
+ int error = 0;

 again:
  rss[1] = rss[0] = 0;
@@ -515,6 +524,7 @@ again:
  spin_lock_nested(src_ptl, SINGLE_DEPTH_NESTING);
  arch_enter_lazy_mmu_mode();

+ dst_ptp = pmd_page(*(dst_pmd));
  do {
```

```
   /*
    * We are holding two locks at this point - either of them
@@ -530,7 +540,11 @@ again:
    progress++;
    continue;
   }
-  copy_one_pte(dst_mm, src_mm, dst_pte, src_pte, vma, addr, rss);
+  error = copy_one_pte(dst_mm, src_mm, dst_pte, src_pte, vma,
+      addr, rss, dst_ptp);
+  if (error) {
+   break;
+  }
   progress += 8;
  } while (dst_pte++, src_pte++, addr += PAGE_SIZE, addr != end);

@@ -540,9 +554,9 @@ again:
  add_mm_rss(dst_mm, rss[0], rss[1]);
  pte_unmap_unlock(dst_pte - 1, dst_ptl);
  cond_resched();
- if (addr != end)
+ if (addr != end && error == 0)
   goto again;
- return 0;
+ return error;
 }

 static inline int copy_pmd_range(struct mm_struct *dst_mm, struct mm_struct *src_mm,
@@ -697,8 +711,12 @@ static unsigned long zap_pte_range(struc
    */
   if (unlikely(details))
    continue;
-  if (!pte_file(ptent))
+  if (!pte_file(ptent)) {
+   if (!is_migration_entry(pte_to_swp_entry(ptent))) {
+    swap_cgroup_uncharge(pmd_page(*pmd));
+   }
    free_swap_and_cache(pte_to_swp_entry(ptent));
+  }
   pte_clear_not_present_full(mm, addr, pte, tlb->fullmm);
  } while (pte++, addr += PAGE_SIZE, (addr != end && *zap_work > 0));

@@ -2076,6 +2094,7 @@ static int do_swap_page(struct mm_struct
  /* The page isn't present yet, go ahead with the fault. */

  inc_mm_counter(mm, anon_rss);
+ swap_cgroup_uncharge(pmd_page(*pmd));
  pte = mk_pte(page, vma->vm_page_prot);
  if (write_access && can_share_swap_page(page)) {
```

```
   pte = maybe_mkwrite(pte_mkdirty(pte), vma);
--- linux-2.6.25-rc3-mm1/mm/mremap.c.BACKUP 2008-02-25 06:25:54.000000000 +0900
+++ linux-2.6.25-rc3-mm1/mm/mremap.c 2008-03-26 12:02:17.000000000 +0900
@@ -13,11 +13,13 @@
 #include <linux/shm.h>
 #include <linux/mman.h>
 #include <linux/swap.h>
+#include <linux/swapops.h>
 #include <linux/capability.h>
 #include <linux/fs.h>
 #include <linux/highmem.h>
 #include <linux/security.h>
 #include <linux/syscalls.h>
+#include <linux/swapcontrol.h>

 #include <asm/uaccess.h>
 #include <asm/cacheflush.h>
@@ -74,6 +76,8 @@ static void move_ptes(struct vm_area_str
  struct mm_struct *mm = vma->vm_mm;
  pte_t *old_pte, *new_pte, pte;
  spinlock_t *old_ptl, *new_ptl;
+ struct page *old_ptp = pmd_page(*old_pmd);
+ struct page *new_ptp = pmd_page(*new_pmd);

  if (vma->vm_file) {
   /*
@@ -106,6 +110,10 @@ static void move_ptes(struct vm_area_str
    continue;
   pte = ptep_clear_flush(vma, old_addr, old_pte);
   pte = move_pte(pte, new_vma->vm_page_prot, old_addr, new_addr);
+  if (!pte_present(pte) && !pte_file(pte) &&
+      !is_migration_entry(pte_to_swp_entry(pte))) {
+   swap_cgroup_remap_charge(old_ptp, new_ptp, mm);
+  }
   set_pte_at(mm, new_addr, new_pte, pte);
  }

--- linux-2.6.25-rc3-mm1/mm/swapcontrol.c.BACKUP 2008-03-12 12:08:30.000000000 +0900
+++ linux-2.6.25-rc3-mm1/mm/swapcontrol.c 2008-03-27 14:39:18.000000000 +0900
@@ -0,0 +1,335 @@
+
+/*
+ * swapcontrol.c COPYRIGHT FUJITSU LIMITED 2008
+ *
+ * Author: yamamoto@valinux.co.jp
+ */
+
+#include <linux/err.h>
```

```
+#include <linux/cgroup.h>
+#include <linux/hugetlb.h>
+#include <linux/res_counter.h>
+#include <linux/pagemap.h>
+#include <linux/slab.h>
+#include <linux/swap.h>
+#include <linux/swapcontrol.h>
+#include <linux/swapops.h>
+
+struct swap_cgroup {
+ struct cgroup_subsys_state scg_css;
+ struct res_counter scg_counter;
+};
+
+#define css_to_scg(css) container_of((css), struct swap_cgroup, scg_css)
+#define cg_to_css(cg) cgroup_subsys_state((cg), swap_cgroup_subsys_id)
+#define cg_to_scg(cg) css_to_scg(cg_to_css(cg))
+
+static struct swap_cgroup *
+swap_cgroup_prepare_ptp(struct page *ptp, struct mm_struct *mm)
+{
+ struct swap_cgroup *scg = ptp->ptp_swap_cgroup;
+
+ BUG_ON(mm == NULL);
+ BUG_ON(mm->swap_cgroup == NULL);
+ if (scg == NULL) {
+  /*
+   * see swap_cgroup_attach.
+   */
+  smp_rmb();
+  scg = mm->swap_cgroup;
+  BUG_ON(scg == NULL);
+  ptp->ptp_swap_cgroup = scg;
+ }
+
+ return scg;
+}
+
+/*
+ * called with page table locked.
+ */
+int
+swap_cgroup_charge(struct page *ptp, struct mm_struct *mm)
+{
+ struct swap_cgroup * const scg = swap_cgroup_prepare_ptp(ptp, mm);
+
+ return res_counter_charge(&scg->scg_counter, PAGE_CACHE_SIZE);
+}
```

```
+
+/*
+ * called with page table locked.
+ */
+void
+swap_cgroup_uncharge(struct page *ptp)
+{
+ struct swap_cgroup * const scg = ptp->ptp_swap_cgroup;
+
+ BUG_ON(scg == NULL);
+ res_counter_uncharge(&scg->scg_counter, PAGE_CACHE_SIZE);
+}
+
+/*
+ * called with both page tables locked.
+ */
+void
+swap_cgroup_remap_charge(struct page *oldptp, struct page *newptp,
+    struct mm_struct *mm)
+{
+ struct swap_cgroup * const oldscg = oldptp->ptp_swap_cgroup;
+ struct swap_cgroup * const newscg = swap_cgroup_prepare_ptp(newptp, mm);
+
+ BUG_ON(oldscg == NULL);
+ BUG_ON(newscg == NULL);
+ if (oldscg == newscg) {
+  return;
+ }
+
+ /*
+  * swap_cgroup_attach is in progress.
+  */
+
+ res_counter_charge_force(&newscg->scg_counter, PAGE_CACHE_SIZE);
+ res_counter_uncharge(&oldscg->scg_counter, PAGE_CACHE_SIZE);
+}
+
+void
+swap_cgroup_init_mm(struct mm_struct *mm, struct task_struct *t)
+{
+ struct swap_cgroup *scg;
+ struct cgroup *cg;
+
+ /* mm->swap_cgroup is not NULL in the case of dup_mm */
+ cg = task_cgroup(t, swap_cgroup_subsys_id);
+ BUG_ON(cg == NULL);
+ scg = cg_to_scg(cg);
+ BUG_ON(scg == NULL);
```

```
+ css_get(&scg->scg_css);
+ mm->swap_cgroup = scg;
+}
+
+void
+swap_cgroup_exit_mm(struct mm_struct *mm)
+{
+ struct swap_cgroup * const scg = mm->swap_cgroup;
+
+ /*
+  * note that swap_cgroup_attach does get_task_mm which
+  * prevents us from being called.  we can assume mm->swap_cgroup
+  * is stable.
+  */
+
+ BUG_ON(scg == NULL);
+ mm->swap_cgroup = NULL; /* it isn't necessary.  just being tidy. */
+ css_put(&scg->scg_css);
+}
+
+static u64
+swap_cgroup_read_u64(struct cgroup *cg, struct cftype *cft)
+{
+
+ return res_counter_read_u64(&cg_to_scg(cg)->scg_counter, cft->private);
+}
+
+static int
+swap_cgroup_write_u64(struct cgroup *cg, struct cftype *cft, u64 val)
+{
+ struct res_counter *counter = &cg_to_scg(cg)->scg_counter;
+ unsigned long flags;
+
+ /* XXX res_counter_write_u64 */
+ BUG_ON(cft->private != RES_LIMIT);
+ spin_lock_irqsave(&counter->lock, flags);
+ counter->limit = val;
+ spin_unlock_irqrestore(&counter->lock, flags);
+ return 0;
+}
+
+static const struct cftype files[] = {
+ {
+  .name = "usage_in_bytes",
+  .private = RES_USAGE,
+  .read_u64 = swap_cgroup_read_u64,
+ },
+ {
```

```
+  .name = "failcnt",
+  .private = RES_FAILCNT,
+  .read_u64 = swap_cgroup_read_u64,
+ },
+ {
+  .name = "limit_in_bytes",
+  .private = RES_LIMIT,
+  .read_u64 = swap_cgroup_read_u64,
+  .write_u64 = swap_cgroup_write_u64,
+ },
+};
+
+static struct cgroup_subsys_state *
+swap_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cg)
+{
+ struct swap_cgroup *scg;
+
+ scg = kzalloc(sizeof(*scg), GFP_KERNEL);
+ if (scg == NULL) {
+  return ERR_PTR(-ENOMEM);
+ }
+ res_counter_init(&scg->scg_counter);
+ if (unlikely(cg->parent == NULL)) {
+  init_mm.swap_cgroup = scg;
+ }
+ return &scg->scg_css;
+}
+
+static void
+swap_cgroup_destroy(struct cgroup_subsys *ss, struct cgroup *cg)
+{
+
+ kfree(cg_to_scg(cg));
+}
+
+static int
+swap_cgroup_populate(struct cgroup_subsys *ss, struct cgroup *cg)
+{
+
+ return cgroup_add_files(cg, ss, files, ARRAY_SIZE(files));
+}
+
+struct swap_cgroup_attach_mm_cb_args {
+ struct vm_area_struct *vma;
+ struct swap_cgroup *oldscg;
+ struct swap_cgroup *newscg;
+};
+
```

```
+static int
+swap_cgroup_attach_mm_cb(pmd_t *pmd, unsigned long startva, unsigned long endva,
+    void *private)
+{
+ const struct swap_cgroup_attach_mm_cb_args * const args = private;
+ struct vm_area_struct * const vma = args->vma;
+ struct swap_cgroup * const oldscg = args->oldscg;
+ struct swap_cgroup * const newscg = args->newscg;
+ struct page *ptp;
+ spinlock_t *ptl;
+ const pte_t *startpte;
+ const pte_t *pte;
+ unsigned long va;
+ int swslots;
+ int bytes;
+
+ BUG_ON((startva & ~PMD_MASK) != 0);
+ BUG_ON((endva & ~PMD_MASK) != 0);
+
+ startpte = pte_offset_map_lock(vma->vm_mm, pmd, startva, &ptl);
+ ptp = pmd_page(*pmd);
+ if (ptp->ptp_swap_cgroup == NULL || ptp->ptp_swap_cgroup == newscg) {
+  goto out;
+ }
+ BUG_ON(ptp->ptp_swap_cgroup != oldscg);
+
+ /*
+  * count the number of swap entries in this page table page.
+  */
+ swslots = 0;
+ for (va = startva, pte = startpte; va != endva;
+      pte++, va += PAGE_SIZE) {
+  const pte_t pt_entry = *pte;
+
+  if (pte_present(pt_entry)) {
+   continue;
+  }
+  if (pte_none(pt_entry)) {
+   continue;
+  }
+  if (pte_file(pt_entry)) {
+   continue;
+  }
+  if (is_migration_entry(pte_to_swp_entry(pt_entry))) {
+   continue;
+  }
+  swslots++;
+ }
```

```
+
+ bytes = swslots * PAGE_CACHE_SIZE;
+ res_counter_uncharge(&oldscg->scg_counter, bytes);
+ /*
+  * XXX ignore newscg's limit because cgroup ->attach method can't fail.
+  */
+ res_counter_charge_force(&newscg->scg_counter, bytes);
+ ptp->ptp_swap_cgroup = newscg;
+out:
+ pte_unmap_unlock(startpte, ptl);
+
+ return 0;
+}
+
+static const struct mm_walk swap_cgroup_attach_mm_walk = {
+ .pmd_entry = swap_cgroup_attach_mm_cb,
+};
+
+static void
+swap_cgroup_attach_mm(struct mm_struct *mm, struct swap_cgroup *oldscg,
+    struct swap_cgroup *newscg)
+{
+ struct swap_cgroup_attach_mm_cb_args args;
+ struct vm_area_struct *vma;
+
+ args.oldscg = oldscg;
+ args.newscg = newscg;
+ down_read(&mm->mmap_sem);
+ for (vma = mm->mmap; vma; vma = vma->vm_next) {
+  if (is_vm_hugetlb_page(vma)) {
+   continue;
+  }
+  args.vma = vma;
+  walk_page_range(mm, vma->vm_start & PMD_MASK,
+      (vma->vm_end + PMD_SIZE - 1) & PMD_MASK,
+      &swap_cgroup_attach_mm_walk, &args);
+ }
+ up_read(&mm->mmap_sem);
+}
+
+static void
+swap_cgroup_attach(struct cgroup_subsys *ss, struct cgroup *newcg,
+    struct cgroup *oldcg, struct task_struct *t)
+{
+ struct swap_cgroup *oldscg;
+ struct swap_cgroup *newscg;
+ struct mm_struct *mm;
+
```

```
+ BUG_ON(oldcg == NULL);
+ BUG_ON(newcg == NULL);
+ BUG_ON(cg_to_css(oldcg) == NULL);
+ BUG_ON(cg_to_css(newcg) == NULL);
+ BUG_ON(oldcg == newcg);
+
+ if (!thread_group_leader(t)) {
+  return;
+ }
+ mm = get_task_mm(t);
+ if (mm == NULL) {
+  return;
+ }
+ oldscg = cg_to_scg(oldcg);
+ newscg = cg_to_scg(newcg);
+ BUG_ON(oldscg == newscg);
+ css_get(&newscg->scg_css);
+ BUG_ON(mm->swap_cgroup != oldscg);
+ mm->swap_cgroup = newscg;
+ /*
+  * issue a barrier to ensure that swap_cgroup_charge will
+  * see the new mm->swap_cgroup.  it might be redundant given locking
+  * activities around, but this is not a performance critical path
+  * anyway.
+  */
+ smp_wmb();
+ swap_cgroup_attach_mm(mm, oldscg, newscg);
+ css_put(&oldscg->scg_css);
+ mmput(mm);
+}
+
+struct cgroup_subsys swap_cgroup_subsys = {
+ .name = "swap",
+ .subsys_id = swap_cgroup_subsys_id,
+ .create = swap_cgroup_create,
+ .destroy = swap_cgroup_destroy,
+ .populate = swap_cgroup_populate,
+ .attach = swap_cgroup_attach,
+};
--- linux-2.6.25-rc3-mm1/include/linux/swapcontrol.h.BACKUP 2008-03-13 07:18:00.000000000
+0900
+++ linux-2.6.25-rc3-mm1/include/linux/swapcontrol.h 2008-03-26 12:03:09.000000000 +0900
@@ -0,0 +1,51 @@
+
+/*
+ * swapcontrol.h COPYRIGHT FUJITSU LIMITED 2008
+ *
+ * Author: yamamoto@valinux.co.jp
```

```
+ */
+
+struct task_struct;
+struct mm_struct;
+struct page;
+
+#if defined(CONFIG_CGROUP_SWAP_RES_CTLR)
+
+int swap_cgroup_charge(struct page *, struct mm_struct *);
+void swap_cgroup_uncharge(struct page *);
+void swap_cgroup_remap_charge(struct page *, struct page *, struct mm_struct *);
+void swap_cgroup_init_mm(struct mm_struct *, struct task_struct *);
+void swap_cgroup_exit_mm(struct mm_struct *);
+
+#else /* defined(CONFIG_CGROUP_SWAP_RES_CTLR) */
+
+static inline int
+swap_cgroup_charge(struct page *ptp, struct mm_struct *mm)
+{
+
+ /* nothing */
+ return 0;
+}
+
+static inline void
+swap_cgroup_uncharge(struct page *ptp)
+{
+
+ /* nothing */
+}
+
+static void
+swap_cgroup_init_mm(struct mm_struct *mm, struct task_struct *t)
+{
+
+ /* nothing */
+}
+
+static inline void
+swap_cgroup_exit_mm(struct mm_struct *mm)
+{
+
+ /* nothing */
+}
+
+#endif /* defined(CONFIG_CGROUP_SWAP_RES_CTLR) */
--- linux-2.6.25-rc3-mm1/include/linux/res_counter.h.BACKUP 2008-03-05 15:45:48.000000000
+0900
```

```
+++ linux-2.6.25-rc3-mm1/include/linux/res_counter.h 2008-03-17 08:03:16.000000000 +0900
@@ -90,6 +90,7 @@ void res_counter_init(struct res_counter

 int res_counter_charge_locked(struct res_counter *counter, unsigned long val);
 int res_counter_charge(struct res_counter *counter, unsigned long val);
+void res_counter_charge_force(struct res_counter *counter, unsigned long val);

 /*
  * uncharge - tell that some portion of the resource is released
--- linux-2.6.25-rc3-mm1/include/linux/cgroup_subsys.h.BACKUP 2008-03-05
15:45:46.000000000 +0900
+++ linux-2.6.25-rc3-mm1/include/linux/cgroup_subsys.h 2008-03-12 12:01:32.000000000 +0900
@@ -42,3 +42,9 @@ SUBSYS(mem_cgroup)
 #endif

 /* */
+
+#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
+SUBSYS(swap_cgroup)
+#endif
+
+/* */
--- linux-2.6.25-rc3-mm1/include/linux/mm.h.BACKUP 2008-03-05 15:45:47.000000000 +0900
+++ linux-2.6.25-rc3-mm1/include/linux/mm.h 2008-03-17 07:28:05.000000000 +0900
@@ -888,6 +888,7 @@ static inline pmd_t *pmd_alloc(struct mm

 static inline void pgtable_page_ctor(struct page *page)
 {
+ page->ptp_swap_cgroup = NULL;
  pte_lock_init(page);
  inc_zone_page_state(page, NR_PAGETABLE);
 }
--- linux-2.6.25-rc3-mm1/include/linux/mm_types.h.BACKUP 2008-03-05 15:45:47.000000000
+0900
+++ linux-2.6.25-rc3-mm1/include/linux/mm_types.h 2008-03-17 07:24:47.000000000 +0900
@@ -19,6 +19,7 @@
 #define AT_VECTOR_SIZE (2*(AT_VECTOR_SIZE_ARCH + AT_VECTOR_SIZE_BASE + 1))

 struct address_space;
+struct swap_cgroup;

 #if NR_CPUS >= CONFIG_SPLIT_PTLOCK_CPUS
 typedef atomic_long_t mm_counter_t;
@@ -70,6 +71,7 @@ struct page {
  union {
   pgoff_t index;  /* Our offset within mapping. */
   void *freelist;  /* SLUB: freelist req. slab lock */
+  struct swap_cgroup *ptp_swap_cgroup; /* PTP: swap cgroup */
```

```
 };
 struct list_head lru;  /* Pageout list, eg. active_list
      * protected by zone->lru_lock !
@@ -230,6 +232,9 @@ struct mm_struct {
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR
  struct mem_cgroup *mem_cgroup;
 #endif
+#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
+ struct swap_cgroup *swap_cgroup;
+#endif

 #ifdef CONFIG_PROC_FS
  /* store ref to file /proc/<pid>/exe symlink points to */
--- linux-2.6.25-rc3-mm1/kernel/res_counter.c.BACKUP 2008-03-05 15:45:50.000000000 +0900
+++ linux-2.6.25-rc3-mm1/kernel/res_counter.c 2008-03-17 08:03:59.000000000 +0900
@@ -41,6 +41,15 @@ int res_counter_charge(struct res_counte
  return ret;
 }

+void res_counter_charge_force(struct res_counter *counter, unsigned long val)
+{
+ unsigned long flags;
+
+ spin_lock_irqsave(&counter->lock, flags);
+ counter->usage += val;
+ spin_unlock_irqrestore(&counter->lock, flags);
+}
+
 void res_counter_uncharge_locked(struct res_counter *counter, unsigned long val)
 {
  if (WARN_ON(counter->usage < val))
--- linux-2.6.25-rc3-mm1/kernel/fork.c.BACKUP 2008-03-05 15:45:50.000000000 +0900
+++ linux-2.6.25-rc3-mm1/kernel/fork.c 2008-03-17 09:17:39.000000000 +0900
@@ -41,6 +41,7 @@
 #include <linux/mount.h>
 #include <linux/audit.h>
 #include <linux/memcontrol.h>
+#include <linux/swapcontrol.h>
 #include <linux/profile.h>
 #include <linux/rmap.h>
 #include <linux/acct.h>
@@ -361,6 +362,7 @@ static struct mm_struct * mm_init(struct
  mm_init_cgroup(mm, p);

  if (likely(!mm_alloc_pgd(mm))) {
+  swap_cgroup_init_mm(mm, p);
   mm->def_flags = 0;
   return mm;
```

```
 }
@@ -394,7 +396,6 @@ void __mmdrop(struct mm_struct *mm)
 {
  BUG_ON(mm == &init_mm);
  mm_free_pgd(mm);
- mm_free_cgroup(mm);
  destroy_context(mm);
  free_mm(mm);
 }
@@ -417,6 +418,8 @@ void mmput(struct mm_struct *mm)
    spin_unlock(&mmlist_lock);
   }
   put_swap_token(mm);
+  mm_free_cgroup(mm);
+  swap_cgroup_exit_mm(mm);
   mmdrop(mm);
  }
 }
@@ -523,11 +526,12 @@ static struct mm_struct *dup_mm(struct t
  if (init_new_context(tsk, mm))
   goto fail_nocontext;

+ dup_mm_exe_file(oldmm, mm);
+
  err = dup_mmap(mm, oldmm);
  if (err)
   goto free_pt;

- dup_mm_exe_file(oldmm, mm);
  mm->hiwater_rss = get_mm_rss(mm);
  mm->hiwater_vm = mm->total_vm;
```