Subject: [RFC][PATCH] another swap controller for cgroup
Posted by yamamoto on Mon, 17 Mar 2008 02:04:07 GMT
View Forum Message <> Reply to Message

hi,

the following is another swap controller, which was designed and
implemented independently from nishimura-san's one.

some random differences from nishimura-san's one:
- counts and limits the number of ptes with swap entries instead of
  on-disk swap slots.
- no swapon-time memory allocation.
- anonymous objects (shmem) are not accounted.
- precise wrt moving tasks between cgroups.

this patch contains some unrelated small fixes which i've posted separately:
- exe_file fput botch fix
- cgroup_rmdir EBUSY fix

any comments?

YAMAMOTO Takashi

```
--- 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/swapcontrol.c.BACKUP 2008-03-12 12:08:30.000000000 +0900
+++ linux-2.6.25-rc3-mm1/mm/swapcontrol.c 2008-03-17 08:27:53.000000000 +0900
@@ -0,0 +1,298 @@
+
+/*
+ * 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))
+
+/*
+ * called with page table locked.
+ */
+int
+swap_cgroup_charge(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.
```

```
+   */
+  rmb();
+  scg = mm->swap_cgroup;
+  BUG_ON(scg == NULL);
+  ptp->ptp_swap_cgroup = scg;
+ }
+ 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;
+
+ if (scg == NULL) {
+  return;
+ }
+ res_counter_uncharge(&scg->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)
+{
+
+ BUG_ON(mm->swap_cgroup == NULL);
+ css_put(&mm->swap_cgroup->scg_css);
+ mm->swap_cgroup = NULL;
+}
+
```

```
+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.
+  */
+ 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 2008-03-14 15:32:36.000000000 +0900
@@ -0,0 +1,50 @@
+
+/*
+ * 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_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;
```

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Balbir Singh on Mon, 17 Mar 2008 05:11:51 GMT
View Forum Message <> Reply to Message

YAMAMOTO Takashi wrote:
> hi,
>
> the following is another swap controller, which was designed and
> implemented independently from nishimura-san's one.
>
> some random differences from nishimura-san's one:
> - counts and limits the number of ptes with swap entries instead of
>   on-disk swap slots.
> - no swapon-time memory allocation.
> - anonymous objects (shmem) are not accounted.
> - precise wrt moving tasks between cgroups.
>
> this patch contains some unrelated small fixes which i've posted separately:
> - exe_file fput botch fix
> - cgroup_rmdir EBUSY fix
>
> any comments?
>

Hi, YAMAMOTO-San,

Thanks for the patch. I'll review and test it. I'll get back soon

Balbir

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Daisuke Nishimura on Mon, 17 Mar 2008 08:15:16 GMT
View Forum Message <> Reply to Message

Hi, Yamamoto-san.

I'm reviewing and testing your patch now.

I think your implementation is better because:
- the group to be charged is determined correctly
  at the point of swapout, without fixing the behavior
  of move_task of memcg.
  (I think the behavior of move_task of memcg should be
  fixed anyway.)
- the group to be uncharged is remembered in page struct
  of pmd, so there is no need to add array of pointers
  to swap_info_struct.

> - anonymous objects (shmem) are not accounted.
IMHO, shmem should be accounted.
I agree it's difficult in your implementation,
but are you going to support it?

Thanks,
Daisuke Nishimura.

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by yamamoto on Mon, 17 Mar 2008 08:50:03 GMT
View Forum Message <> Reply to Message

> > - anonymous objects (shmem) are not accounted.

> IMHO, shmem should be accounted.
> I agree it's difficult in your implementation,
> but are you going to support it?

it should be trivial to track how much swap an anonymous object is using.
i'm not sure how it should be associated with cgroups, tho.

YAMAMOTO Takashi

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Daisuke Nishimura on Mon, 24 Mar 2008 12:10:14 GMT
View Forum Message <> Reply to Message

Daisuke Nishimura wrote:
> Hi, Yamamoto-san.
>
> I'm reviewing and testing your patch now.
>

In building kernel infinitely(in a cgroup of
memory.limit=64M and swap.limit=128M, with swappiness=100),
almost all of the swap (1GB) is consumed as swap cache
after a day or so.
As a result, processes are occasionally OOM-killed even when
the swap.usage of the group doesn't exceed the limit.

I don't know why the swap cache uses up swap space.
I will test whether a similar issue happens without your patch.
Do you have any thoughts?

BTW, I think that it would be better, in the sence of
isolating memory resource, if there is a framework
to limit the usage of swap cache.

Thanks,
Daisuke Nishimura.

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Balbir Singh on Mon, 24 Mar 2008 12:22:08 GMT
View Forum Message <> Reply to Message

Daisuke Nishimura wrote:
> Daisuke Nishimura wrote:
>> Hi, Yamamoto-san.
>>
>> I'm reviewing and testing your patch now.
>>
>
> In building kernel infinitely(in a cgroup of
> memory.limit=64M and swap.limit=128M, with swappiness=100),
> almost all of the swap (1GB) is consumed as swap cache
> after a day or so.
> As a result, processes are occasionally OOM-killed even when
> the swap.usage of the group doesn't exceed the limit.
>
> I don't know why the swap cache uses up swap space.
> I will test whether a similar issue happens without your patch.
> Do you have any thoughts?
>
>
> BTW, I think that it would be better, in the sence of
> isolating memory resource, if there is a framework
> to limit the usage of swap cache.

We had this earlier, but dropped it later due to issues related to swap readahead and assigning the pages to the correct cgroup.

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by yamamoto on Tue, 25 Mar 2008 03:10:39 GMT
View Forum Message <> Reply to Message

hi,

> Daisuke Nishimura wrote:

> > Hi, Yamamoto-san.
> >
> > I'm reviewing and testing your patch now.
> >
>
> In building kernel infinitely(in a cgroup of
> memory.limit=64M and swap.limit=128M, with swappiness=100),
> almost all of the swap (1GB) is consumed as swap cache
> after a day or so.
> As a result, processes are occasionally OOM-killed even when
> the swap.usage of the group doesn't exceed the limit.
>
> I don't know why the swap cache uses up swap space.
> I will test whether a similar issue happens without your patch.
> Do you have any thoughts?

my patch tends to yield more swap cache because it makes try_to_unmap
fail and shrink_page_list leaves swap cache in that case.
i'm not sure how it causes 1GB swap cache, tho.

YAMAMOTO Takashi

>
> BTW, I think that it would be better, in the sence of
> isolating memory resource, if there is a framework
> to limit the usage of swap cache.
>
>
> Thanks,
> Daisuke Nishimura.

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

---

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Daisuke Nishimura on Tue, 25 Mar 2008 04:35:53 GMT
View Forum Message <> Reply to Message

YAMAMOTO Takashi wrote:
> hi,
>
>> Daisuke Nishimura wrote:
>>> Hi, Yamamoto-san.
>>>
>>> I'm reviewing and testing your patch now.
>>>

>> In building kernel infinitely(in a cgroup of
>> memory.limit=64M and swap.limit=128M, with swappiness=100),
>> almost all of the swap (1GB) is consumed as swap cache
>> after a day or so.
>> As a result, processes are occasionally OOM-killed even when
>> the swap.usage of the group doesn't exceed the limit.
>>
>> I don't know why the swap cache uses up swap space.
>> I will test whether a similar issue happens without your patch.
>> Do you have any thoughts?
>
> my patch tends to yield more swap cache because it makes try_to_unmap
> fail and shrink_page_list leaves swap cache in that case.
> i'm not sure how it causes 1GB swap cache, tho.
>

Agree.

I suspected that the cause of this problem was the behavior
of shrink_page_list as you said, so I thought one of Rik's
split-lru patchset:

  http://lkml.org/lkml/2008/3/4/492
  [patch 04/20] free swap space on swap-in/activation

would reduce the usage of swap cache to half of the total swap.
But it didn't help, so I think there may be some other causes.

Thanks,
Daisuke Nishimura.

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Daisuke Nishimura on Tue, 25 Mar 2008 06:46:28 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Daisuke Nishimura wrote:
>> Daisuke Nishimura wrote:
>>> Hi, Yamamoto-san.
>>>
>>> I'm reviewing and testing your patch now.
>>>

>> In building kernel infinitely(in a cgroup of
>> memory.limit=64M and swap.limit=128M, with swappiness=100),
>> almost all of the swap (1GB) is consumed as swap cache
>> after a day or so.
>> As a result, processes are occasionally OOM-killed even when
>> the swap.usage of the group doesn't exceed the limit.
>>
>> I don't know why the swap cache uses up swap space.
>> I will test whether a similar issue happens without your patch.
>> Do you have any thoughts?
>>
>>
>> BTW, I think that it would be better, in the sence of
>> isolating memory resource, if there is a framework
>> to limit the usage of swap cache.
>
> We had this earlier, but dropped it later due to issues related to swap
> readahead and assigning the pages to the correct cgroup.
>
Yes, I know.

In my swap subsystem posted before, I charge swap entries
and remember to which cgroup each swap entries is charged
in an array of pointers. So swap caches is charged as swap
not memory, and swap usage including swap cache can be accounted.

There may be better solution, and one of the issue of
my implementation is swap_cgroup_chage() returns error
before reclaiming swap entries which are only used
by swap caches.
I'm considering this issue now.

Thanks,
Daisuke Nishimura.

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by yamamoto on Tue, 25 Mar 2008 08:57:23 GMT
View Forum Message <> Reply to Message

> YAMAMOTO Takashi wrote:
> > hi,

> >
> >> Daisuke Nishimura wrote:
> >>> Hi, Yamamoto-san.
> >>>
> >>> I'm reviewing and testing your patch now.
> >>>
> >> In building kernel infinitely(in a cgroup of
> >> memory.limit=64M and swap.limit=128M, with swappiness=100),
> >> almost all of the swap (1GB) is consumed as swap cache
> >> after a day or so.
> >> As a result, processes are occasionally OOM-killed even when
> >> the swap.usage of the group doesn't exceed the limit.
> >>
> >> I don't know why the swap cache uses up swap space.
> >> I will test whether a similar issue happens without your patch.
> >> Do you have any thoughts?
> >
> > my patch tends to yield more swap cache because it makes try_to_unmap
> > fail and shrink_page_list leaves swap cache in that case.
> > i'm not sure how it causes 1GB swap cache, tho.
> >
>
> Agree.
>
> I suspected that the cause of this problem was the behavior
> of shrink_page_list as you said, so I thought one of Rik's
> split-lru patchset:
>
> http://lkml.org/lkml/2008/3/4/492
> [patch 04/20] free swap space on swap-in/activation
>
> would reduce the usage of swap cache to half of the total swap.
> But it didn't help, so I think there may be some other causes.

do you mean you tested with the patch in the url?
i don't think remove_exclusive_swap_page works for us
because our page has more references than it expects.
ie. ptes, cache, isolate_page

(unless your tree has another change for remove_exclusive_swap_page.
i haven't checked other patches in the patchset.)

YAMAMOTO Takashi

>
>
> Thanks,
> Daisuke Nishimura.

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Daisuke Nishimura on Tue, 25 Mar 2008 12:35:44 GMT
View Forum Message <> Reply to Message

YAMAMOTO Takashi wrote:
>> YAMAMOTO Takashi wrote:
>>> hi,
>>>
>>>> Daisuke Nishimura wrote:
>>>>> Hi, Yamamoto-san.
>>>>>
>>>>> I'm reviewing and testing your patch now.
>>>>>
>>>> In building kernel infinitely(in a cgroup of
>>>> memory.limit=64M and swap.limit=128M, with swappiness=100),
>>>> almost all of the swap (1GB) is consumed as swap cache
>>>> after a day or so.
>>>> As a result, processes are occasionally OOM-killed even when
>>>> the swap.usage of the group doesn't exceed the limit.
>>>>
>>>> I don't know why the swap cache uses up swap space.
>>>> I will test whether a similar issue happens without your patch.
>>>> Do you have any thoughts?
>>> my patch tends to yield more swap cache because it makes try_to_unmap
>>> fail and shrink_page_list leaves swap cache in that case.
>>> i'm not sure how it causes 1GB swap cache, tho.
>>>
>> Agree.
>>
>> I suspected that the cause of this problem was the behavior
>> of shrink_page_list as you said, so I thought one of Rik's
>> split-lru patchset:
>>
>> http://lkml.org/lkml/2008/3/4/492
>> [patch 04/20] free swap space on swap-in/activation
>>
>> would reduce the usage of swap cache to half of the total swap.
>> But it didn't help, so I think there may be some other causes.
>
> do you mean you tested with the patch in the url?
> i don't think remove_exclusive_swap_page works for us
> because our page has more references than it expects.

> ie. ptes, cache, isolate_page
>
umm... you are right.

I missed the fact that isolate_page increases page count.

Thanks,
Daisuke Nishimura.

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

---

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;
```

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Daisuke Nishimura on Fri, 28 Mar 2008 09:00:33 GMT
View Forum Message <> Reply to Message

YAMAMOTO Takashi wrote:
> hi,
>
> i tried to reproduce the large swap cache issue, but no luck.
> can you provide a little more detailed instruction?
>
This issue also happens on generic 2.6.25-rc3-mm1
(with limitting only memory), so I think this issue is not
related to your patch.
I'm investigating this issue too.

Below is my environment and how to reproduce.

- System
  full virtualized xen guest based on RHEL5.1(x86_64).
    CPU: 2
    memory: 2GB
    swap: 1GB
  A config of the running kernel(2.6.25-rc3-mm1 with your patch)
  is attached.

- how to reproduce
  - change swappines to 100

    echo 100 >/proc/sys/vm/swappiness

  - mount cgroup fs

    # mount -t cgroup -o memory,swap none /cgroup

  - make cgroup for test

    # mkdir /cgroup/02
    # echo -n 64M >/cgroup/02/memory.limit_in_bytes
    # echo -n `expr 128 \* 1024 \* 1024` >/cgroup/02/swap.limit_in_bytes

  - run test

echo $$ >/cgropu/02/tasks
while true; do make clean; make -j2; done

In other terminals, I run some monitoring processes, top,
"tail -f /var/log/messages", and displaying *.usage_in_bytes
every seconds.

Thanks,
Daisuke Nishimura.

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

---

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by yamamoto on Tue, 08 Apr 2008 03:29:37 GMT
View Forum Message <> Reply to Message

> YAMAMOTO Takashi wrote:
> > hi,
> >
> > i tried to reproduce the large swap cache issue, but no luck.
> > can you provide a little more detailed instruction?
> >
> This issue also happens on generic 2.6.25-rc3-mm1
> (with limitting only memory), so I think this issue is not
> related to your patch.
> I'm investigating this issue too.
>
> Below is my environment and how to reproduce.
>
> - System
> full virtualized xen guest based on RHEL5.1(x86_64).
> CPU: 2
> memory: 2GB
> swap: 1GB
> A config of the running kernel(2.6.25-rc3-mm1 with your patch)
> is attached.
>
> - how to reproduce
> - change swappines to 100
>
> echo 100 >/proc/sys/vm/swappiness

>
> - mount cgroup fs
>
>   # mount -t cgroup -o memory,swap none /cgroup
>
> - make cgroup for test
>
>   # mkdir /cgroup/02
>   # echo -n 64M >/cgroup/02/memory.limit_in_bytes
>   # echo -n `expr 128 \* 1024 \* 1024` >/cgroup/02/swap.limit_in_bytes
>
> - run test
>
>   # echo $$ >/cgropu/02/tasks
>   # while true; do make clean; make -j2; done
>
> In other terminals, I run some monitoring processes, top,
> "tail -f /var/log/messages", and displaying *.usage_in_bytes
> every seconds.
>
>
> Thanks,
> Daisuke Nishimura.

what i tried was essentially same.
for me, once vm_swap_full() got true, swap cache stopped growing as expected.

 http://people.valinux.co.jp/~yamamoto/swap.png

it was taken by running
 while :;do swapon -s|tail -1;sleep 1;done > foo
in an unlimited cgroup, and then plotted by gnuplot.
 plot "foo" u 4

as my system has 1GB swap configured, the vm_swap_full() threshold is
around 500MB.

YAMAMOTO Takashi

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

---

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by yamamoto on Thu, 10 Apr 2008 07:40:04 GMT
View Forum Message <> Reply to Message

the following is a new version of the patch.
changes from the previous:
 - fix a BUG_ON in swap_cgroup_attach and add a comment about it.

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 2008-04-01 12:27:51.000000000 +0900
@@ -0,0 +1,342 @@
+
+/*
+ * 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 *oldmmscg;
+ struct swap_cgroup *oldtaskscg;
+ 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;
+ }
+ oldtaskscg = cg_to_scg(oldcg);
+ newscg = cg_to_scg(newcg);
+ BUG_ON(oldtaskscg == newscg);
+ /*
+  * note that a task and its mm can belong to different cgroups
+  * with the current implementation.
+  */
+ oldmmscg = mm->swap_cgroup;
+ if (oldmmscg != newscg) {
+  css_get(&newscg->scg_css);
+  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, oldmmscg, newscg);
+  css_put(&oldmmscg->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;
```

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

---

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by yamamoto on Tue, 29 Apr 2008 22:50:47 GMT
View Forum Message <> Reply to Message

> > > - anonymous objects (shmem) are not accounted.
> > IMHO, shmem should be accounted.
> > I agree it's difficult in your implementation,
> > but are you going to support it?
>
> it should be trivial to track how much swap an anonymous object is using.
> i'm not sure how it should be associated with cgroups, tho.
>
> YAMAMOTO Takashi

i implemented shmem swap accounting. see below.

YAMAMOTO Takashi

the following is another swap controller, which was designed and
implemented independently from nishimura-san's one.

some random differences from nishimura-san's one:
- counts and limits the number of ptes with swap entries instead of
  on-disk swap slots. (except swap slots used by anonymous objects,
  which are counted differently. see below.)
- no swapon-time memory allocation.
- precise wrt moving tasks between cgroups.
- [NEW] anonymous objects (shmem) are associated to a cgroup when they are
  created and the number of on-disk swap slots used by them are counted for
  the cgroup. the association is persistent except cgroup removal,
  in which case, its associated objects are moved to init_mm's cgroup.

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

--- linux-2.6.25-rc8-mm2/mm/swapcontrol.c.BACKUP 2008-04-21 12:06:44.000000000 +0900
+++ linux-2.6.25-rc8-mm2/mm/swapcontrol.c 2008-04-28 14:16:03.000000000 +0900
@@ -0,0 +1,447 @@
+
+/*
+ * 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>
+#include <linux/shmem_fs.h>
+
+struct swap_cgroup {
+ struct cgroup_subsys_state scg_css;
+ struct res_counter scg_counter;
+ struct list_head scg_shmem_list;

```
+};
+
+#define task_to_css(task) task_subsys_state((task), swap_cgroup_subsys_id)
+#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 DEFINE_MUTEX(swap_cgroup_shmem_lock);
+
+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;
+ }
+ INIT_LIST_HEAD(&scg->scg_shmem_list);
+ return &scg->scg_css;
+}
+
+static void
+swap_cgroup_destroy(struct cgroup_subsys *ss, struct cgroup *cg)
+{
+ struct swap_cgroup *oldscg = cg_to_scg(cg);
+ struct swap_cgroup *newscg;
+ struct list_head *pos;
+ struct list_head *next;
+
+ /*
+  * move our anonymous objects to init_mm's group.
+  */
+ newscg = init_mm.swap_cgroup;
+ BUG_ON(newscg == NULL);
+ BUG_ON(newscg == oldscg);
+
+ mutex_lock(&swap_cgroup_shmem_lock);
+ list_for_each_safe(pos, next, &oldscg->scg_shmem_list) {
+  struct shmem_inode_info * const info =
+      list_entry(pos, struct shmem_inode_info, swap_cgroup_list);
+  long bytes;
+
+  spin_lock(&info->lock);
+  BUG_ON(info->swap_cgroup != oldscg);
+  bytes = info->swapped << PAGE_SHIFT;
+  info->swap_cgroup = newscg;
+  res_counter_uncharge(&oldscg->scg_counter, bytes);
+  res_counter_charge_force(&newscg->scg_counter, bytes);
+  spin_unlock(&info->lock);
```

```
+  list_del_init(&info->swap_cgroup_list);
+  list_add(&info->swap_cgroup_list, &newscg->scg_shmem_list);
+ }
+ mutex_unlock(&swap_cgroup_shmem_lock);
+
+ BUG_ON(res_counter_read_u64(&oldscg->scg_counter, RES_USAGE) != 0);
+ kfree(oldscg);
+}
+
+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 *oldmmscg;
+ struct swap_cgroup *oldtaskscg;
+ 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;
+ }
+ oldtaskscg = cg_to_scg(oldcg);
+ newscg = cg_to_scg(newcg);
+ BUG_ON(oldtaskscg == newscg);
+ /*
+  * note that a task and its mm can belong to different cgroups
+  * with the current implementation.
+  */
+ oldmmscg = mm->swap_cgroup;
+ if (oldmmscg != newscg) {
+  css_get(&newscg->scg_css);
+  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, oldmmscg, newscg);
+  css_put(&oldmmscg->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,
+};
+
+/*
+ * swap-backed objects (shmem)
+ */
+
+void
+swap_cgroup_shmem_init(struct shmem_inode_info *info)
+{
+ struct swap_cgroup *scg;
+
+ BUG_ON(info->swapped != 0);
+
+ INIT_LIST_HEAD(&info->swap_cgroup_list);
+
+ rcu_read_lock();
+ scg = css_to_scg(task_to_css(current));
+ css_get(&scg->scg_css);
+ rcu_read_unlock();
+
+ mutex_lock(&swap_cgroup_shmem_lock);
+ info->swap_cgroup = scg;
+ list_add(&info->swap_cgroup_list, &scg->scg_shmem_list);
+ mutex_unlock(&swap_cgroup_shmem_lock);
+
+ css_put(&scg->scg_css);
+}
+
+void
```

```
+swap_cgroup_shmem_destroy(struct shmem_inode_info *info)
+{
+ struct swap_cgroup *scg;
+
+ BUG_ON(scg == NULL);
+ BUG_ON(list_empty(&info->swap_cgroup_list));
+ BUG_ON(info->swapped != 0);
+
+ mutex_lock(&swap_cgroup_shmem_lock);
+ scg = info->swap_cgroup;
+ list_del_init(&info->swap_cgroup_list);
+ mutex_unlock(&swap_cgroup_shmem_lock);
+}
+
+/*
+ * => called with info->lock held
+ */
+int
+swap_cgroup_shmem_charge(struct shmem_inode_info *info, long delta)
+{
+ struct swap_cgroup *scg = info->swap_cgroup;
+
+ BUG_ON(scg == NULL);
+ BUG_ON(list_empty(&info->swap_cgroup_list));
+
+ return res_counter_charge(&scg->scg_counter, delta << PAGE_CACHE_SHIFT);
+}
+
+/*
+ * => called with info->lock held
+ */
+void
+swap_cgroup_shmem_uncharge(struct shmem_inode_info *info, long delta)
+{
+ struct swap_cgroup *scg = info->swap_cgroup;
+
+ BUG_ON(scg == NULL);
+ BUG_ON(list_empty(&info->swap_cgroup_list));
+
+ res_counter_uncharge(&scg->scg_counter, delta << PAGE_CACHE_SHIFT);
+}
--- linux-2.6.25-rc8-mm2/mm/memory.c.BACKUP 2008-04-21 10:04:08.000000000 +0900
+++ linux-2.6.25-rc8-mm2/mm/memory.c 2008-04-21 12:06:44.000000000 +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>
@@ -467,10 +468,10 @@ out:
  * 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;
@@ -481,6 +482,11 @@ 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))) {
@@ -530,6 +536,7 @@ 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,
@@ -540,6 +547,8 @@ 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;
@@ -551,6 +560,7 @@ 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
@@ -566,7 +576,11 @@ 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);

@@ -576,9 +590,9 @@ 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,
@@ -733,8 +747,12 @@ 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));

@@ -2155,6 +2173,7 @@ 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-rc8-mm2/mm/Makefile.BACKUP 2008-04-21 10:04:08.000000000 +0900
+++ linux-2.6.25-rc8-mm2/mm/Makefile 2008-04-21 12:06:44.000000000 +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-rc8-mm2/mm/rmap.c.BACKUP 2008-04-21 10:04:09.000000000 +0900
+++ linux-2.6.25-rc8-mm2/mm/rmap.c 2008-04-21 12:06:44.000000000 +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>

@@ -225,8 +226,9 @@ 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;
@@ -257,12 +259,21 @@ 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.
@@ -700,13 +711,14 @@ 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;

@@ -721,6 +733,12 @@ 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-rc8-mm2/mm/swapfile.c.BACKUP 2008-04-21 10:04:09.000000000 +0900
+++ linux-2.6.25-rc8-mm2/mm/swapfile.c 2008-04-21 12:06:44.000000000 +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-rc8-mm2/mm/shmem.c.BACKUP 2008-04-21 10:04:09.000000000 +0900
+++ linux-2.6.25-rc8-mm2/mm/shmem.c 2008-04-28 08:12:54.000000000 +0900
@@ -50,6 +50,7 @@
 #include <linux/migrate.h>
 #include <linux/highmem.h>
 #include <linux/seq_file.h>
+#include <linux/swapcontrol.h>

 #include <asm/uaccess.h>
 #include <asm/div64.h>
@@ -360,16 +361,27 @@ static swp_entry_t *shmem_swp_entry(stru
  return shmem_swp_map(subdir) + offset;
 }

-static void shmem_swp_set(struct shmem_inode_info *info, swp_entry_t *entry, unsigned long
value)
+static int shmem_swp_set(struct shmem_inode_info *info, swp_entry_t *entry, unsigned long
value)
 {
  long incdec = value? 1: -1;

+ if (incdec > 0) {
+  int error;
+
+  error = swap_cgroup_shmem_charge(info, incdec);
+  if (error) {
+   return error;
+  }
+ } else {
+  swap_cgroup_shmem_uncharge(info, -incdec);
+ }
  entry->val = value;
  info->swapped += incdec;
  if ((unsigned long)(entry - info->i_direct) >= SHMEM_NR_DIRECT) {
   struct page *page = kmap_atomic_to_page(entry);
   set_page_private(page, page_private(page) + incdec);
  }
+ return 0;
 }

 /**
```

```
@@ -727,6 +739,7 @@ done2:
  spin_lock(&info->lock);
  info->flags &= ~SHMEM_TRUNCATE;
  info->swapped -= nr_swaps_freed;
+ swap_cgroup_shmem_uncharge(info, nr_swaps_freed);
  if (nr_pages_to_free)
   shmem_free_blocks(inode, nr_pages_to_free);
  shmem_recalc_inode(inode);
@@ -1042,9 +1055,16 @@ static int shmem_writepage(struct page *
  shmem_recalc_inode(inode);

  if (swap.val && add_to_swap_cache(page, swap, GFP_ATOMIC) == 0) {
-  remove_from_page_cache(page);
-  shmem_swp_set(info, entry, swap.val);
+  int error;
+
+  error = shmem_swp_set(info, entry, swap.val);
   shmem_swp_unmap(entry);
+  if (error != 0) {
+   delete_from_swap_cache(page); /* does swap_free */
+   spin_unlock(&info->lock);
+   goto redirty;
+  }
+  remove_from_page_cache(page);
   if (list_empty(&info->swaplist))
    inode = igrab(inode);
   else
@@ -1522,6 +1542,7 @@ shmem_get_inode(struct super_block *sb,
    inode->i_fop = &shmem_file_operations;
    mpol_shared_policy_init(&info->policy,
        shmem_get_sbmpol(sbinfo));
+   swap_cgroup_shmem_init(info);
    break;
   case S_IFDIR:
    inc_nlink(inode);
@@ -2325,6 +2346,7 @@ static void shmem_destroy_inode(struct i
  if ((inode->i_mode & S_IFMT) == S_IFREG) {
   /* only struct inode is valid if it's an inline symlink */
   mpol_free_shared_policy(&SHMEM_I(inode)->policy);
+  swap_cgroup_shmem_destroy(SHMEM_I(inode));
  }
  shmem_acl_destroy_inode(inode);
  kmem_cache_free(shmem_inode_cachep, SHMEM_I(inode));
--- linux-2.6.25-rc8-mm2/mm/mremap.c.BACKUP 2008-04-02 04:44:26.000000000 +0900
+++ linux-2.6.25-rc8-mm2/mm/mremap.c 2008-04-21 12:06:44.000000000 +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-rc8-mm2/include/linux/res_counter.h.BACKUP 2008-04-21 10:03:58.000000000
+0900
+++ linux-2.6.25-rc8-mm2/include/linux/res_counter.h 2008-04-21 12:06:50.000000000 +0900
@@ -97,6 +97,7 @@ 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-rc8-mm2/include/linux/swapcontrol.h.BACKUP 2008-04-21 12:06:45.000000000
+0900
+++ linux-2.6.25-rc8-mm2/include/linux/swapcontrol.h 2008-04-24 15:16:30.000000000 +0900
@@ -0,0 +1,86 @@
+
+/*
```

```
+ * swapcontrol.h COPYRIGHT FUJITSU LIMITED 2008
+ *
+ * Author: yamamoto@valinux.co.jp
+ */
+
+struct task_struct;
+struct mm_struct;
+struct page;
+struct shmem_inode_info;
+
+#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 *);
+
+void swap_cgroup_shmem_init(struct shmem_inode_info *);
+void swap_cgroup_shmem_destroy(struct shmem_inode_info *);
+int swap_cgroup_shmem_charge(struct shmem_inode_info *, long);
+void swap_cgroup_shmem_uncharge(struct shmem_inode_info *, long);
+
+#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 */
+}
+
+static inline void
+swap_cgroup_shmem_init(struct shmem_inode_info *info)
+{
+
+ /* nothing */
+}
+
+static inline void
+swap_cgroup_shmem_destroy(struct shmem_inode_info *info)
+{
+
+ /* nothing */
+}
+
+static inline int
+swap_cgroup_shmem_charge(struct shmem_inode_info *info, long delta)
+{
+
+ /* nothing */
+ return 0;
+}
+
+static inline void
+swap_cgroup_shmem_uncharge(struct shmem_inode_info *info, long delta)
+{
+
+ /* nothing */
+}
+
+#endif /* defined(CONFIG_CGROUP_SWAP_RES_CTLR) */
--- linux-2.6.25-rc8-mm2/include/linux/shmem_fs.h.BACKUP 2008-04-21 10:03:59.000000000
+0900
+++ linux-2.6.25-rc8-mm2/include/linux/shmem_fs.h 2008-04-23 18:07:39.000000000 +0900
@@ -4,6 +4,8 @@
 #include <linux/swap.h>
 #include <linux/mempolicy.h>

+struct swap_cgroup;
+
 /* inode in-kernel data */

 #define SHMEM_NR_DIRECT 16
```

```
@@ -23,6 +25,10 @@ struct shmem_inode_info {
  struct posix_acl *i_acl;
  struct posix_acl *i_default_acl;
 #endif
+#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
+ struct swap_cgroup *swap_cgroup;
+ struct list_head swap_cgroup_list;
+#endif
 };

 struct shmem_sb_info {
--- linux-2.6.25-rc8-mm2/include/linux/cgroup_subsys.h.BACKUP 2008-04-21
10:03:52.000000000 +0900
+++ linux-2.6.25-rc8-mm2/include/linux/cgroup_subsys.h 2008-04-21 12:06:51.000000000 +0900
@@ -43,6 +43,12 @@ SUBSYS(mem_cgroup)

 /* */

+#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
+SUBSYS(swap_cgroup)
+#endif
+
+/* */
+
 #ifdef CONFIG_CGROUP_DEVICE
 SUBSYS(devices)
 #endif
--- linux-2.6.25-rc8-mm2/include/linux/mm_types.h.BACKUP 2008-04-21 10:03:56.000000000
+0900
+++ linux-2.6.25-rc8-mm2/include/linux/mm_types.h 2008-04-21 12:06:51.000000000 +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;
@@ -73,6 +74,7 @@ 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 !
@@ -233,6 +235,9 @@ 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-rc8-mm2/include/linux/mm.h.BACKUP 2008-04-21 10:03:56.000000000 +0900
+++ linux-2.6.25-rc8-mm2/include/linux/mm.h 2008-04-21 12:06:51.000000000 +0900
@@ -926,6 +926,7 @@ 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-rc8-mm2/kernel/res_counter.c.BACKUP 2008-04-21 10:04:06.000000000 +0900
+++ linux-2.6.25-rc8-mm2/kernel/res_counter.c 2008-04-21 12:06:51.000000000 +0900
@@ -44,6 +44,15 @@ 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-rc8-mm2/kernel/fork.c.BACKUP 2008-04-21 10:04:04.000000000 +0900
+++ linux-2.6.25-rc8-mm2/kernel/fork.c 2008-04-21 12:17:53.000000000 +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;
  }
@@ -417,6 +419,7 @@ void mmput(struct mm_struct *mm)
   }
   put_swap_token(mm);
   mm_free_cgroup(mm);
+  swap_cgroup_exit_mm(mm);
   mmdrop(mm);
  }
 }
--- linux-2.6.25-rc8-mm2/init/Kconfig.BACKUP 2008-04-21 10:04:04.000000000 +0900
+++ linux-2.6.25-rc8-mm2/init/Kconfig 2008-04-21 12:06:44.000000000 +0900
@@ -386,6 +386,12 @@ 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
```

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Daisuke Nishimura on Wed, 30 Apr 2008 04:09:44 GMT
View Forum Message <> Reply to Message

YAMAMOTO Takashi wrote:
>>>> - anonymous objects (shmem) are not accounted.
>>> IMHO, shmem should be accounted.
>>> I agree it's difficult in your implementation,
>>> but are you going to support it?
>> it should be trivial to track how much swap an anonymous object is using.
>> i'm not sure how it should be associated with cgroups, tho.
>>
>> YAMAMOTO Takashi

>
> i implemented shmem swap accounting. see below.
>
> YAMAMOTO Takashi
>
Hi, YAMAMOTO san.

Thank you for implementing this feature.
I will review it.

BTW, I'm just trying to make my swapcontroller patch
that is rebased on recent kernel and implemented
as part of memory controller.
I'm going to submit it by the middle of May.

Thanks,
Daisuke Nishimura.

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

---

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Balbir Singh on Mon, 05 May 2008 06:11:42 GMT
View Forum Message <> Reply to Message

* YAMAMOTO Takashi <yamamoto@valinux.co.jp> [2008-04-30 07:50:47]:

> > > > - anonymous objects (shmem) are not accounted.
> > > IMHO, shmem should be accounted.
> > > I agree it's difficult in your implementation,
> > > but are you going to support it?
> >
> > it should be trivial to track how much swap an anonymous object is using.
> > i'm not sure how it should be associated with cgroups, tho.
> >
> > YAMAMOTO Takashi
>
> i implemented shmem swap accounting. see below.
>
> YAMAMOTO Takashi
>
>
> the following is another swap controller, which was designed and
> implemented independently from nishimura-san's one.
>

Hi, Thanks for the patches and your patience. I've just applied your
patches on top of 2.6.25-mm1 (it had a minor reject, that I've fixed).
I am building and testing the patches along with KAMEZAWA-San's low
overhead patches.

> some random differences from nishimura-san's one:
> - counts and limits the number of ptes with swap entries instead of
>   on-disk swap slots. (except swap slots used by anonymous objects,
>   which are counted differently. see below.)
> - no swapon-time memory allocation.
> - precise wrt moving tasks between cgroups.
> - [NEW] anonymous objects (shmem) are associated to a cgroup when they are
>   created and the number of on-disk swap slots used by them are counted for
>   the cgroup. the association is persistent except cgroup removal,
>   in which case, its associated objects are moved to init_mm's cgroup.
>
>
> Signed-off-by: YAMAMOTO Takashi <yamamoto@valinux.co.jp>
> ---
>
> --- linux-2.6.25-rc8-mm2/mm/swapcontrol.c.BACKUP 2008-04-21 12:06:44.000000000 +0900
> +++ linux-2.6.25-rc8-mm2/mm/swapcontrol.c 2008-04-28 14:16:03.000000000 +0900
> @@ -0,0 +1,447 @@
> +
> +/*
> + * swapcontrol.c COPYRIGHT FUJITSU LIMITED 2008
> + *
> + * Author: yamamoto@valinux.co.jp
> + */
> +
> +#include <linux/err.h>
> +#include <linux/cgroup.h>
> +#include <linux/hugetlb.h>

My powerpc build fails, we need to move hugetlb.h down to the bottom

> +#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>
> +#include <linux/shmem_fs.h>
> +
> +struct swap_cgroup {
> + struct cgroup_subsys_state scg_css;

Can't we call this just css. Since the structure is swap_cgroup it
already has the required namespace required to distinguish it from
other css's. Please see page 4 of "The practice of programming", be
consistent. The same comment applies to all members below.

> + struct res_counter scg_counter;
> + struct list_head scg_shmem_list;
> +};
> +
> +#define task_to_css(task) task_subsys_state((task), swap_cgroup_subsys_id)
> +#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))

Aren't static inline better than macros? I would suggest moving to
them.

> +
> +static DEFINE_MUTEX(swap_cgroup_shmem_lock);
> +
> +static struct swap_cgroup *
> +swap_cgroup_prepare_ptp(struct page *ptp, struct mm_struct *mm)
> +{
> + struct swap_cgroup *scg = ptp->ptp_swap_cgroup;
> +

Is this routine safe w.r.t. concurrent operations, modifications to
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;

With the mm->owner patches, we need not maintain a separate
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.
> + */

Some more comments here would help.

> +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);

So, we force the counter to go over limit?

> + 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;

With mm->owner changes, this routine can be simplified.

> +}
> +
> +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. */

If it's not required, I'd say remove it.

> + 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;
> +}
> +

We need to write actual numbers here? Can't we keep the write
interface consistent with the memory controller?

> +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) {

Extra braces, not required if there is just one statement following if

> + return ERR_PTR(-ENOMEM);
> + }
> + res_counter_init(&scg->scg_counter);
> + if (unlikely(cg->parent == NULL)) {

Ditto

> + init_mm.swap_cgroup = scg;
> + }
> + INIT_LIST_HEAD(&scg->scg_shmem_list);
> + return &scg->scg_css;
> +}
> +
> +static void
> +swap_cgroup_destroy(struct cgroup_subsys *ss, struct cgroup *cg)
> +{
> + struct swap_cgroup *oldscg = cg_to_scg(cg);
> + struct swap_cgroup *newscg;
> + struct list_head *pos;
> + struct list_head *next;
> +
> + /*
> + * move our anonymous objects to init_mm's group.
> + */

Is this good design, should be allow a swap_cgroup to be destroyed,
even though pages are allocated to it? Is moving to init_mm (root
cgroup) a good idea? Ideally with support for hierarchies, if we ever
do move things, it should be to the parent cgroup.

> + newscg = init_mm.swap_cgroup;
> + BUG_ON(newscg == NULL);
> + BUG_ON(newscg == oldscg);
> +
> + mutex_lock(&swap_cgroup_shmem_lock);
> + list_for_each_safe(pos, next, &oldscg->scg_shmem_list) {
> + struct shmem_inode_info * const info =
> +   list_entry(pos, struct shmem_inode_info, swap_cgroup_list);
> + long bytes;
> +
> + spin_lock(&info->lock);
> + BUG_ON(info->swap_cgroup != oldscg);
> + bytes = info->swapped << PAGE_SHIFT;

Shouldn't this be PAGE_CACHE_SHIFT just to be consistent

> + info->swap_cgroup = newscg;
> + res_counter_uncharge(&oldscg->scg_counter, bytes);

> + res_counter_charge_force(&newscg->scg_counter, bytes);

I don't like the excessive use of res_counter_charge_force(), it seems
like we might end up bypassing the controller all together. I would
rather fail the destroy operation if the charge fails.

> + spin_unlock(&info->lock);
> + list_del_init(&info->swap_cgroup_list);
> + list_add(&info->swap_cgroup_list, &newscg->scg_shmem_list);
> + }
> + mutex_unlock(&swap_cgroup_shmem_lock);
> +
> + BUG_ON(res_counter_read_u64(&oldscg->scg_counter, RES_USAGE) != 0);
> + kfree(oldscg);
> +}
> +
> +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;
> +};
> +

We need comments for the function below. I am afraid I fail to
understand it.

> +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);
```

But in the future, we could plan on making attach fail (I have a
requirement for it). Again, I don't like the _force operation

```
> + 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)

We need comments about the function, why do we attach an mm?

> +{
> + 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 *oldmmscg;
> + struct swap_cgroup *oldtaskscg;
> + 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)) {

Coding style, brances not needed

> + return;
> + }
> + mm = get_task_mm(t);
> + if (mm == NULL) {

ditto

> + return;
> + }
> + oldtaskscg = cg_to_scg(oldcg);
> + newscg = cg_to_scg(newcg);
> + BUG_ON(oldtaskscg == newscg);
> + /*
> + * note that a task and its mm can belong to different cgroups
> + * with the current implementation.
> + */
> + oldmmscg = mm->swap_cgroup;
> + if (oldmmscg != newscg) {
> + css_get(&newscg->scg_css);
> + 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, oldmmscg, newscg);
> + css_put(&oldmmscg->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,
> +};
> +
> +/*
> + * swap-backed objects (shmem)
> + */

> +
> +void
> +swap_cgroup_shmem_init(struct shmem_inode_info *info)
> +{
> + struct swap_cgroup *scg;
> +
> + BUG_ON(info->swapped != 0);
> +
> + INIT_LIST_HEAD(&info->swap_cgroup_list);
> +
> + rcu_read_lock();
> + scg = css_to_scg(task_to_css(current));
> + css_get(&scg->scg_css);
> + rcu_read_unlock();
> +
> + mutex_lock(&swap_cgroup_shmem_lock);
> + info->swap_cgroup = scg;
> + list_add(&info->swap_cgroup_list, &scg->scg_shmem_list);
> + mutex_unlock(&swap_cgroup_shmem_lock);
> +
> + css_put(&scg->scg_css);
> +}
> +
> +void
> +swap_cgroup_shmem_destroy(struct shmem_inode_info *info)
> +{
> + struct swap_cgroup *scg;
> +
> + BUG_ON(scg == NULL);

This is bogus, this check does not make any sense.

> + BUG_ON(list_empty(&info->swap_cgroup_list));
> + BUG_ON(info->swapped != 0);
> +
> + mutex_lock(&swap_cgroup_shmem_lock);
> + scg = info->swap_cgroup;
> + list_del_init(&info->swap_cgroup_list);
> + mutex_unlock(&swap_cgroup_shmem_lock);
> +}
> +
> +/*
> + * => called with info->lock held
> + */
> +int
> +swap_cgroup_shmem_charge(struct shmem_inode_info *info, long delta)
> +{
> + struct swap_cgroup *scg = info->swap_cgroup;

> +
> + BUG_ON(scg == NULL);
> + BUG_ON(list_empty(&info->swap_cgroup_list));
> +
> + return res_counter_charge(&scg->scg_counter, delta << PAGE_CACHE_SHIFT);
> +}
> +
> +/*
> + * => called with info->lock held
> + */
> +void
> +swap_cgroup_shmem_uncharge(struct shmem_inode_info *info, long delta)
> +{
> + struct swap_cgroup *scg = info->swap_cgroup;
> +
> + BUG_ON(scg == NULL);
> + BUG_ON(list_empty(&info->swap_cgroup_list));
> +
> + res_counter_uncharge(&scg->scg_counter, delta << PAGE_CACHE_SHIFT);
> +}
> --- linux-2.6.25-rc8-mm2/mm/memory.c.BACKUP 2008-04-21 10:04:08.000000000 +0900
> +++ linux-2.6.25-rc8-mm2/mm/memory.c 2008-04-21 12:06:44.000000000 +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>
> @@ -467,10 +468,10 @@ out:
>   * 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;
> @@ -481,6 +482,11 @@ 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))) {
> @@ -530,6 +536,7 @@ 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,
> @@ -540,6 +547,8 @@ 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;
> @@ -551,6 +560,7 @@ 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
> @@ -566,7 +576,11 @@ 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) {
```

Coding style, braces

```
> +   break;
> +  }
>    progress += 8;
>   } while (dst_pte++, src_pte++, addr += PAGE_SIZE, addr != end);
>
```

```
> @@ -576,9 +590,9 @@ 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,
> @@ -733,8 +747,12 @@ 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));
>
> @@ -2155,6 +2173,7 @@ 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-rc8-mm2/mm/Makefile.BACKUP 2008-04-21 10:04:08.000000000 +0900
> +++ linux-2.6.25-rc8-mm2/mm/Makefile 2008-04-21 12:06:44.000000000 +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-rc8-mm2/mm/rmap.c.BACKUP 2008-04-21 10:04:09.000000000 +0900
> +++ linux-2.6.25-rc8-mm2/mm/rmap.c 2008-04-21 12:06:44.000000000 +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>
>
> @@ -225,8 +226,9 @@ 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)

Why address1, I'd prefer a better name or even __page_check_address.

> {
>   pgd_t *pgd;
>   pud_t *pud;
> @@ -257,12 +259,21 @@ 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) {

Coding style

> +   *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.
> @@ -700,13 +711,14 @@ 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;
>
> @@ -721,6 +733,12 @@ 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? */

I suspect so, otherwise we could end up slots being marked as used
but not really used.

> + 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-rc8-mm2/mm/swapfile.c.BACKUP 2008-04-21 10:04:09.000000000 +0900
> +++ linux-2.6.25-rc8-mm2/mm/swapfile.c 2008-04-21 12:06:44.000000000 +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-rc8-mm2/mm/shmem.c.BACKUP 2008-04-21 10:04:09.000000000 +0900
> +++ linux-2.6.25-rc8-mm2/mm/shmem.c 2008-04-28 08:12:54.000000000 +0900
> @@ -50,6 +50,7 @@
> #include <linux/migrate.h>
> #include <linux/highmem.h>
> #include <linux/seq_file.h>
> +#include <linux/swapcontrol.h>
>
> #include <asm/uaccess.h>
> #include <asm/div64.h>
> @@ -360,16 +361,27 @@ static swp_entry_t *shmem_swp_entry(stru
>   return shmem_swp_map(subdir) + offset;
> }
>
> -static void shmem_swp_set(struct shmem_inode_info *info, swp_entry_t *entry, unsigned long
value)
> +static int shmem_swp_set(struct shmem_inode_info *info, swp_entry_t *entry, unsigned long
value)
> {
>   long incdec = value? 1: -1;
>
> + if (incdec > 0) {
> + int error;
> +
> + error = swap_cgroup_shmem_charge(info, incdec);
> + if (error) {
> +   return error;
> + }
> + } else {
> + swap_cgroup_shmem_uncharge(info, -incdec);
> + }
>   entry->val = value;
>   info->swapped += incdec;
>   if ((unsigned long)(entry - info->i_direct) >= SHMEM_NR_DIRECT) {
>   struct page *page = kmap_atomic_to_page(entry);
>   set_page_private(page, page_private(page) + incdec);
>   }
> + return 0;
> }
>
> /**
> @@ -727,6 +739,7 @@ done2:
>   spin_lock(&info->lock);
>   info->flags &= ~SHMEM_TRUNCATE;
>   info->swapped -= nr_swaps_freed;
> + swap_cgroup_shmem_uncharge(info, nr_swaps_freed);
>   if (nr_pages_to_free)
>   shmem_free_blocks(inode, nr_pages_to_free);

>   shmem_recalc_inode(inode);
> @@ -1042,9 +1055,16 @@ static int shmem_writepage(struct page *
>   shmem_recalc_inode(inode);
>
>   if (swap.val && add_to_swap_cache(page, swap, GFP_ATOMIC) == 0) {
> - remove_from_page_cache(page);
> - shmem_swp_set(info, entry, swap.val);
> + int error;
> +
> + error = shmem_swp_set(info, entry, swap.val);
>   shmem_swp_unmap(entry);
> + if (error != 0) {
> +   delete_from_swap_cache(page); /* does swap_free */
> +   spin_unlock(&info->lock);
> +   goto redirty;
> + }
> + remove_from_page_cache(page);
>   if (list_empty(&info->swaplist))
>   inode = igrab(inode);
>   else
> @@ -1522,6 +1542,7 @@ shmem_get_inode(struct super_block *sb,
>   inode->i_fop = &shmem_file_operations;
>   mpol_shared_policy_init(&info->policy,
>   shmem_get_sbmpol(sbinfo));
> +   swap_cgroup_shmem_init(info);
>   break;
>   case S_IFDIR:
>   inc_nlink(inode);
> @@ -2325,6 +2346,7 @@ static void shmem_destroy_inode(struct i
>   if ((inode->i_mode & S_IFMT) == S_IFREG) {
>   /* only struct inode is valid if it's an inline symlink */
>   mpol_free_shared_policy(&SHMEM_I(inode)->policy);
> + swap_cgroup_shmem_destroy(SHMEM_I(inode));
>   }
>   shmem_acl_destroy_inode(inode);
>   kmem_cache_free(shmem_inode_cachep, SHMEM_I(inode));
> --- linux-2.6.25-rc8-mm2/mm/mremap.c.BACKUP 2008-04-02 04:44:26.000000000 +0900
> +++ linux-2.6.25-rc8-mm2/mm/mremap.c 2008-04-21 12:06:44.000000000 +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-rc8-mm2/include/linux/res_counter.h.BACKUP 2008-04-21 10:03:58.000000000
+0900
> +++ linux-2.6.25-rc8-mm2/include/linux/res_counter.h 2008-04-21 12:06:50.000000000 +0900
> @@ -97,6 +97,7 @@ 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-rc8-mm2/include/linux/swapcontrol.h.BACKUP 2008-04-21 12:06:45.000000000
+0900
> +++ linux-2.6.25-rc8-mm2/include/linux/swapcontrol.h 2008-04-24 15:16:30.000000000 +0900
> @@ -0,0 +1,86 @@
> +
> +/*
> + * swapcontrol.h COPYRIGHT FUJITSU LIMITED 2008
> + *
> + * Author: yamamoto@valinux.co.jp
> + */
> +
> +struct task_struct;
> +struct mm_struct;
```

```
> +struct page;
> +struct shmem_inode_info;
> +
> +#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 *);
> +
> +void swap_cgroup_shmem_init(struct shmem_inode_info *);
> +void swap_cgroup_shmem_destroy(struct shmem_inode_info *);
> +int swap_cgroup_shmem_charge(struct shmem_inode_info *, long);
> +void swap_cgroup_shmem_uncharge(struct shmem_inode_info *, long);
> +
> +#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 */
> +}
> +
> +static inline void
```

> +swap_cgroup_shmem_init(struct shmem_inode_info *info)
> +{
> +
> + /* nothing */
> +}
> +
> +static inline void
> +swap_cgroup_shmem_destroy(struct shmem_inode_info *info)
> +{
> +
> + /* nothing */
> +}
> +
> +static inline int
> +swap_cgroup_shmem_charge(struct shmem_inode_info *info, long delta)
> +{
> +
> + /* nothing */
> + return 0;
> +}
> +
> +static inline void
> +swap_cgroup_shmem_uncharge(struct shmem_inode_info *info, long delta)
> +{
> +
> + /* nothing */
> +}
> +
> +#endif /* defined(CONFIG_CGROUP_SWAP_RES_CTLR) */
> --- linux-2.6.25-rc8-mm2/include/linux/shmem_fs.h.BACKUP 2008-04-21 10:03:59.000000000
+0900
> +++ linux-2.6.25-rc8-mm2/include/linux/shmem_fs.h 2008-04-23 18:07:39.000000000 +0900
> @@ -4,6 +4,8 @@
> #include <linux/swap.h>
> #include <linux/mempolicy.h>
>
> +struct swap_cgroup;
> +
> /* inode in-kernel data */
>
> #define SHMEM_NR_DIRECT 16
> @@ -23,6 +25,10 @@ struct shmem_inode_info {
> struct posix_acl *i_acl;
> struct posix_acl *i_default_acl;
> #endif
> +#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
> + struct swap_cgroup *swap_cgroup;
> + struct list_head swap_cgroup_list;

> +#endif
> };
>
> struct shmem_sb_info {
> --- linux-2.6.25-rc8-mm2/include/linux/cgroup_subsys.h.BACKUP 2008-04-21
10:03:52.000000000 +0900
> +++ linux-2.6.25-rc8-mm2/include/linux/cgroup_subsys.h 2008-04-21 12:06:51.000000000
+0900
> @@ -43,6 +43,12 @@ SUBSYS(mem_cgroup)
>
> /* */
>
> +#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
> +SUBSYS(swap_cgroup)
> +#endif
> +
> +/* */
> +
> #ifdef CONFIG_CGROUP_DEVICE
> SUBSYS(devices)
> #endif
> --- linux-2.6.25-rc8-mm2/include/linux/mm_types.h.BACKUP 2008-04-21 10:03:56.000000000
+0900
> +++ linux-2.6.25-rc8-mm2/include/linux/mm_types.h 2008-04-21 12:06:51.000000000 +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;
> @@ -73,6 +74,7 @@ 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 !
> @@ -233,6 +235,9 @@ 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-rc8-mm2/include/linux/mm.h.BACKUP 2008-04-21 10:03:56.000000000 +0900
> +++ linux-2.6.25-rc8-mm2/include/linux/mm.h 2008-04-21 12:06:51.000000000 +0900
> @@ -926,6 +926,7 @@ 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-rc8-mm2/kernel/res_counter.c.BACKUP 2008-04-21 10:04:06.000000000 +0900
> +++ linux-2.6.25-rc8-mm2/kernel/res_counter.c 2008-04-21 12:06:51.000000000 +0900
> @@ -44,6 +44,15 @@ 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-rc8-mm2/kernel/fork.c.BACKUP 2008-04-21 10:04:04.000000000 +0900
> +++ linux-2.6.25-rc8-mm2/kernel/fork.c 2008-04-21 12:17:53.000000000 +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;
>   }

> @@ -417,6 +419,7 @@ void mmput(struct mm_struct *mm)
> }
> put_swap_token(mm);
> mm_free_cgroup(mm);
> + swap_cgroup_exit_mm(mm);
> mmdrop(mm);
> }
> }
> --- linux-2.6.25-rc8-mm2/init/Kconfig.BACKUP 2008-04-21 10:04:04.000000000 +0900
> +++ linux-2.6.25-rc8-mm2/init/Kconfig 2008-04-21 12:06:44.000000000 +0900
> @@ -386,6 +386,12 @@ 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
>
>
> --
> To unsubscribe, send a message with 'unsubscribe linux-mm' in
> the body to majordomo@kvack.org. For more info on Linux MM,
> see: http://www.linux-mm.org/ .
> Don't email: email@kvack.org

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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

---

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by yamamoto on Wed, 07 May 2008 05:50:44 GMT
View Forum Message <> Reply to Message

hi,

> Hi, Thanks for the patches and your patience. I've just applied your

> patches on top of 2.6.25-mm1 (it had a minor reject, that I've fixed).
> I am building and testing the patches along with KAMEZAWA-San's low
> overhead patches.

thanks.

> > +#include <linux/err.h>
> > +#include <linux/cgroup.h>
> > +#include <linux/hugetlb.h>
>
> My powerpc build fails, we need to move hugetlb.h down to the bottom

what's the error message?

> > +struct swap_cgroup {
> > + struct cgroup_subsys_state scg_css;
>
> Can't we call this just css. Since the structure is swap_cgroup it
> already has the required namespace required to distinguish it from
> other css's. Please see page 4 of "The practice of programming", be
> consistent. The same comment applies to all members below.

i don't have the book.
i like this kind of prefixes as it's grep-friendly.

> > +#define task_to_css(task) task_subsys_state((task), swap_cgroup_subsys_id)
> > +#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))
>
> Aren't static inline better than macros? I would suggest moving to
> them.

sounds like a matter of preference.
either ok for me.

> > +static struct swap_cgroup *
> > +swap_cgroup_prepare_ptp(struct page *ptp, struct mm_struct *mm)
> > +{
> > + struct swap_cgroup *scg = ptp->ptp_swap_cgroup;
> > +
>
> Is this routine safe w.r.t. concurrent operations, modifications to
> ptp_swap_cgroup?

it's always accessed with the page table locked.

> > + BUG_ON(mm == NULL);

> > + BUG_ON(mm->swap_cgroup == NULL);
> > + if (scg == NULL) {
> > + /*
> > +   * see swap_cgroup_attach.
> > +   */
> > + smp_rmb();
> > + scg = mm->swap_cgroup;
>
> With the mm->owner patches, we need not maintain a separate
> mm->swap_cgroup.

i don't think the mm->owner patch, at least with the current form,
can replace it.

> > + /*
> > + * swap_cgroup_attach is in progress.
> > + */
> > +
> > + res_counter_charge_force(&newscg->scg_counter, PAGE_CACHE_SIZE);
>
> So, we force the counter to go over limit?

yes.

as newscg != oldscg here means the task is being moved between cgroups,
this instance of res_counter_charge_force should not matter much.

> > +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;
> > +}
> > +
>
> We need to write actual numbers here? Can't we keep the write
> interface consistent with the memory controller?

i'm not sure what you mean here. can you explain a bit more?
do you mean K, M, etc?

> > +static void
> > +swap_cgroup_destroy(struct cgroup_subsys *ss, struct cgroup *cg)
> > +{
> > + struct swap_cgroup *oldscg = cg_to_scg(cg);
> > + struct swap_cgroup *newscg;
> > + struct list_head *pos;
> > + struct list_head *next;
> > +
> > + /*
> > + * move our anonymous objects to init_mm's group.
> > + */
>
> Is this good design, should be allow a swap_cgroup to be destroyed,
> even though pages are allocated to it?

first of all, i'm not quite happy with this design. :)
having said that, what else can we do?
i tend to think that trying to swap-in these pages is too much effort
for little benefit.

> Is moving to init_mm (root
> cgroup) a good idea? Ideally with support for hierarchies, if we ever
> do move things, it should be to the parent cgroup.

i chose init_mm because there seemed to be no consensus about
cgroup hierarchy semantics.

> > + info->swap_cgroup = newscg;
> > + res_counter_uncharge(&oldscg->scg_counter, bytes);
> > + res_counter_charge_force(&newscg->scg_counter, bytes);
>
> I don't like the excessive use of res_counter_charge_force(), it seems
> like we might end up bypassing the controller all together. I would
> rather fail the destroy operation if the charge fails.

> > + 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);
>
> But in the future, we could plan on making attach fail (I have a
> requirement for it). Again, I don't like the _force operation

allowing these operations fail implies to have code to back out
partial operations. i'm afraid that it will be too complex.

> > +static void
> > +swap_cgroup_attach_mm(struct mm_struct *mm, struct swap_cgroup *oldscg,
> > +   struct swap_cgroup *newscg)
>
> We need comments about the function, why do we attach an mm?

instead of a task, you mean?
because we count the number of ptes which points to swap
and ptes belong to an mm, not a task.

YAMAMOTO Takashi

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

---

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Balbir Singh on Thu, 08 May 2008 15:43:50 GMT
View Forum Message <> Reply to Message

YAMAMOTO Takashi wrote:
> hi,
>
>> Hi, Thanks for the patches and your patience. I've just applied your
>> patches on top of 2.6.25-mm1 (it had a minor reject, that I've fixed).
>> I am building and testing the patches along with KAMEZAWA-San's low
>> overhead patches.
>
> thanks.
>
>>> +#include <linux/err.h>
>>> +#include <linux/cgroup.h>
>>> +#include <linux/hugetlb.h>
>> My powerpc build fails, we need to move hugetlb.h down to the bottom
>
> what's the error message?
>

It's unable to find the hugetlb call, I think is_hugetlb_vma() or so.

>>> +struct swap_cgroup {
>>> + struct cgroup_subsys_state scg_css;
>> Can't we call this just css. Since the structure is swap_cgroup it
>> already has the required namespace required to distinguish it from
>> other css's. Please see page 4 of "The practice of programming", be
>> consistent. The same comment applies to all members below.
>

> i don't have the book.
> i like this kind of prefixes as it's grep-friendly.
>
>>> +#define task_to_css(task) task_subsys_state((task), swap_cgroup_subsys_id)
>>> +#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))
>> Aren't static inline better than macros? I would suggest moving to
>> them.
>
> sounds like a matter of preference.
> either ok for me.
>

There are other advantages, like better type checking of the arguments. The compiler might even determine that it's better of making a function call instead of inlining it (rare, but possible).

>>> +static struct swap_cgroup *
>>> +swap_cgroup_prepare_ptp(struct page *ptp, struct mm_struct *mm)
>>> +{
>>> + struct swap_cgroup *scg = ptp->ptp_swap_cgroup;
>>> +
>> Is this routine safe w.r.t. concurrent operations, modifications to
>> ptp_swap_cgroup?
>
> it's always accessed with the page table locked.
>
>>> + BUG_ON(mm == NULL);
>>> + BUG_ON(mm->swap_cgroup == NULL);
>>> + if (scg == NULL) {
>>> + /*
>>> +   * see swap_cgroup_attach.
>>> +   */
>>> + smp_rmb();
>>> + scg = mm->swap_cgroup;
>> With the mm->owner patches, we need not maintain a separate
>> mm->swap_cgroup.
>
> i don't think the mm->owner patch, at least with the current form,
> can replace it.
>

Could you please mention what the limitations are? We could get those fixed or take another serious look at the mm->owner patches.

>>> + /*
>>> + * swap_cgroup_attach is in progress.

>>> + */
>>> +
>>> + res_counter_charge_force(&newscg->scg_counter, PAGE_CACHE_SIZE);
>> So, we force the counter to go over limit?
>
> yes.
>
> as newscg != oldscg here means the task is being moved between cgroups,
> this instance of res_counter_charge_force should not matter much.
>

Isn't it bad to force a group to go over it's limit due to migration?

>>> +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;
>>> +}
>>> +
>> We need to write actual numbers here? Can't we keep the write
>> interface consistent with the memory controller?
>
> i'm not sure what you mean here. can you explain a bit more?
> do you mean K, M, etc?
>

Yes, I mean the same format that memparse() uses.

>>> +static void
>>> +swap_cgroup_destroy(struct cgroup_subsys *ss, struct cgroup *cg)
>>> +{
>>> + struct swap_cgroup *oldscg = cg_to_scg(cg);
>>> + struct swap_cgroup *newscg;
>>> + struct list_head *pos;
>>> + struct list_head *next;
>>> +
>>> + /*
>>> + * move our anonymous objects to init_mm's group.
>>> + */
>> Is this good design, should be allow a swap_cgroup to be destroyed,

>> even though pages are allocated to it?
>
> first of all, i'm not quite happy with this design. :)
> having said that, what else can we do?
> i tend to think that trying to swap-in these pages is too much effort
> for little benefit.
>

Just fail the destroy operation, in this case.

>> Is moving to init_mm (root
>> cgroup) a good idea? Ideally with support for hierarchies, if we ever
>> do move things, it should be to the parent cgroup.
>
> i chose init_mm because there seemed to be no consensus about
> cgroup hierarchy semantics.
>

I would suggest that we fail deletion of a group for now. I have a set of
patches for the cgroup hierarchy semantics. I think the parent is the best place
to move it.

>>> + info->swap_cgroup = newscg;
>>> + res_counter_uncharge(&oldscg->scg_counter, bytes);
>>> + res_counter_charge_force(&newscg->scg_counter, bytes);
>> I don't like the excessive use of res_counter_charge_force(), it seems
>> like we might end up bypassing the controller all together. I would
>> rather fail the destroy operation if the charge fails.
>
>>> + 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);
>> But in the future, we could plan on making attach fail (I have a
>> requirement for it). Again, I don't like the _force operation
>
> allowing these operations fail implies to have code to back out
> partial operations. i'm afraid that it will be too complex.
>

OK, we need to find out a way to fix that then.

>>> +static void
>>> +swap_cgroup_attach_mm(struct mm_struct *mm, struct swap_cgroup *oldscg,
>>> +   struct swap_cgroup *newscg)
>> We need comments about the function, why do we attach an mm?

>
> instead of a task, you mean?
> because we count the number of ptes which points to swap
> and ptes belong to an mm, not a task.
>

OK

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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

---

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by yamamoto on Wed, 14 May 2008 03:21:25 GMT
View Forum Message <> Reply to Message

> >>> + BUG_ON(mm == NULL);
> >>> + BUG_ON(mm->swap_cgroup == NULL);
> >>> + if (scg == NULL) {
> >>> + /*
> >>> +   * see swap_cgroup_attach.
> >>> +   */
> >>> + smp_rmb();
> >>> + scg = mm->swap_cgroup;
> >> With the mm->owner patches, we need not maintain a separate
> >> mm->swap_cgroup.
> >
> > i don't think the mm->owner patch, at least with the current form,
> > can replace it.
> >
>
> Could you please mention what the limitations are? We could get those fixed or
> take another serious look at the mm->owner patches.

for example, its callback can't sleep.

> >>> + /*
> >>> + * swap_cgroup_attach is in progress.
> >>> + */
> >>> +
> >>> + res_counter_charge_force(&newscg->scg_counter, PAGE_CACHE_SIZE);

> >> So, we force the counter to go over limit?
> >
> > yes.
> >
> > as newscg != oldscg here means the task is being moved between cgroups,
> > this instance of res_counter_charge_force should not matter much.
> >
>
> Isn't it bad to force a group to go over it's limit due to migration?

we don't have many choices as far as ->attach can't fail.
although we can have racy checks in ->can_attach, i'm not happy with it.

> >> We need to write actual numbers here? Can't we keep the write
> >> interface consistent with the memory controller?
> >
> > i'm not sure what you mean here. can you explain a bit more?
> > do you mean K, M, etc?
> >
>
> Yes, I mean the same format that memparse() uses.

i'll take a look.

> >> Is moving to init_mm (root
> >> cgroup) a good idea? Ideally with support for hierarchies, if we ever
> >> do move things, it should be to the parent cgroup.
> >
> > i chose init_mm because there seemed to be no consensus about
> > cgroup hierarchy semantics.
> >
>
> I would suggest that we fail deletion of a group for now. I have a set of
> patches for the cgroup hierarchy semantics. I think the parent is the best place
> to move it.

ok.

> >>> + info->swap_cgroup = newscg;
> >>> + res_counter_uncharge(&oldscg->scg_counter, bytes);
> >>> + res_counter_charge_force(&newscg->scg_counter, bytes);
> >> I don't like the excessive use of res_counter_charge_force(), it seems
> >> like we might end up bypassing the controller all together. I would
> >> rather fail the destroy operation if the charge fails.
> >
> >>> + 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);
> >> But in the future, we could plan on making attach fail (I have a
> >> requirement for it). Again, I don't like the _force operation
> >
> > allowing these operations fail implies to have code to back out
> > partial operations. i'm afraid that it will be too complex.
> >
>
> OK, we need to find out a way to fix that then.

note that a failure can affect other subsystems which belong to
the same hierarchy as well, and, even worse, a back-out attempt can also fail.
i'm afraid that we need to play some kind of transaction-commit game,
which can make subsystems too complex to implement properly.

YAMAMOTO Takashi

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Paul Menage on Wed, 14 May 2008 03:27:59 GMT
View Forum Message <> Reply to Message

On Tue, May 13, 2008 at 8:21 PM, YAMAMOTO Takashi
<yamamoto@valinux.co.jp> wrote:
>
> note that a failure can affect other subsystems which belong to
> the same hierarchy as well, and, even worse, a back-out attempt can also fail.
> i'm afraid that we need to play some kind of transaction-commit game,
> which can make subsystems too complex to implement properly.
>

I was considering something like that - every call to can_attach()
would be guaranteed to be followed by either a call to attach() or to
a new method called cancel_attach(). Then the subsystem would just
need to ensure that nothing could happen which would cause the attach
to become invalid between the two calls.

Or possibly, since for some subsystems that might involve holding a
spinlock, we should extend it to:

After a successful call to can_attach(), either abort_attach() or
commit_attach() will be called; these calls are not allowed to sleep,

and cgroup.c will not sleep between calls.. If commit_attach() is
called, it will be followed shortly by attach(), which is allowed to
sleep.

Paul

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Paul Menage on Wed, 14 May 2008 08:44:38 GMT
View Forum Message <> Reply to Message

On Tue, May 13, 2008 at 8:21 PM, YAMAMOTO Takashi
<yamamoto@valinux.co.jp> wrote:
> >
> > Could you please mention what the limitations are? We could get those fixed or
> > take another serious look at the mm->owner patches.
>
> for example, its callback can't sleep.
>

You need to be able to sleep in order to take mmap_sem, right?
Since I think that the other current user of the mm->owner callback
probably also needs mmap_sem, it might make sense to take mmap_sem in
mm_update_next_owner() prior to locking the old owner, and hold it
across the callback, which would presumably solve the problem.

> >
> > Isn't it bad to force a group to go over it's limit due to migration?
>
> we don't have many choices as far as ->attach can't fail.
> although we can have racy checks in ->can_attach, i'm not happy with it.

can_attach() isn't racy iff you ensure that a successful result from
can_attach() can't be invalidated by any code not holding
cgroup_mutex.

The existing user of can_attach() is cpusets, and the only way to make
an attachable cpuset non-attachable is to remove its last node or cpu.
The only code that can do this (update_nodemask, update_cpumask, and
common_cpu_mem_hotplug_unplug) all call cgroup_lock() to ensure that
this synchronization occurs.

Of course, having lots of datapath operations also take cgroup_mutex
would be really painful, so it's not practical to use for things that

can become non-attachable due to a process consuming some resources.
This is part of the reason that I started working on the lock-mode
patches that I sent out yesterday, in order to make finer-grained
locking simpler. I'm going to rework those to make the locking more
explicit, and I'll bear this use case in mind while I'm doing it.

A few comments on the patch:

- you're not really limiting swap usage, you're limiting swapped-out
address space. So it looks as though if a process has swapped out most
of its address space, and forks a child, the total "swap" charge for
the cgroup will double. Is that correct? If so, why is this better
than charging for actual swap usage?

- what will happen if someone creates non-NPTL threads, which share an
mm but not a thread group (so each of them is a thread group leader)?

- if you were to store a pointer in the page rather than the
swap_cgroup pointer, then (in combination with mm->owner) you wouldn't
need to do the rebinding to the new swap_cgroup when a process moves
to a different cgroup - you could instead keep a "swapped pte" count
in the mm, and just charge that to the new cgroup and uncharge it from
the old cgroup. You also wouldn't need to keep ref counts on the
swap_cgroup.

- ideally this wouldn't actually start charging until it was bound on
to a cgroups hierarchy, although I guess that the performance of this
is less important than something like the virtual address space
controller, since once we start swapping we can expect performance to
be bad anyway.

Paul

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by yamamoto on Thu, 15 May 2008 06:23:18 GMT
View Forum Message <> Reply to Message

> On Tue, May 13, 2008 at 8:21 PM, YAMAMOTO Takashi
> <yamamoto@valinux.co.jp> wrote:
> > >
> > > Could you please mention what the limitations are? We could get those fixed or
> > > take another serious look at the mm->owner patches.
> >

> > for example, its callback can't sleep.
> >
>
> You need to be able to sleep in order to take mmap_sem, right?

yes.
besides that, i prefer not to hold a spinlock when traversing PTEs
as it can take somewhat long.

> Of course, having lots of datapath operations also take cgroup_mutex
> would be really painful, so it's not practical to use for things that
> can become non-attachable due to a process consuming some resources.
> This is part of the reason that I started working on the lock-mode
> patches that I sent out yesterday, in order to make finer-grained
> locking simpler. I'm going to rework those to make the locking more
> explicit, and I'll bear this use case in mind while I'm doing it.

thanks.

> A few comments on the patch:
>
> - you're not really limiting swap usage, you're limiting swapped-out
> address space. So it looks as though if a process has swapped out most
> of its address space, and forks a child, the total "swap" charge for
> the cgroup will double. Is that correct?

yes.

> If so, why is this better
> than charging for actual swap usage?

its behaviour is more determinstic and it uses less memory.
(than nishimura-san's one, which charges for actual swap usage.)

> - what will happen if someone creates non-NPTL threads, which share an
> mm but not a thread group (so each of them is a thread group leader)?

a thread which is most recently assigned to a cgroup will "win".

> - if you were to store a pointer in the page rather than the

"a pointer"? a pointer to what?

> swap_cgroup pointer, then (in combination with mm->owner) you wouldn't
> need to do the rebinding to the new swap_cgroup when a process moves
> to a different cgroup - you could instead keep a "swapped pte" count
> in the mm, and just charge that to the new cgroup and uncharge it from
> the old cgroup. You also wouldn't need to keep ref counts on the

> swap_cgroup.

PTE walking in my patch is for locking, not for "rebinding".
ie. to deal with concurrent swap activities.
the fact that each page table pages have their own locks (pte_lockptr)
complicated it.

> - ideally this wouldn't actually start charging until it was bound on
> to a cgroups hierarchy, although I guess that the performance of this
> is less important than something like the virtual address space
> controller, since once we start swapping we can expect performance to
> be bad anyway.

i agree.

YAMAMOTO Takashi

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

---

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Paul Menage on Thu, 15 May 2008 07:19:39 GMT
View Forum Message <> Reply to Message

On Wed, May 14, 2008 at 11:23 PM, YAMAMOTO Takashi
<yamamoto@valinux.co.jp> wrote:
> > >
> >
> > You need to be able to sleep in order to take mmap_sem, right?
>
> yes.

OK, well in the thread on balbir's vm limit controller, we discussed
the idea of having mmap_sem held across calls to
mm_update_next_owner(), and hence the cgroup callbacks. Would that
help if mmap_sem were already held when you get the mm_owner_changed()
callback.

>
> > A few comments on the patch:
> >
> > - you're not really limiting swap usage, you're limiting swapped-out
> > address space. So it looks as though if a process has swapped out most
> > of its address space, and forks a child, the total "swap" charge for
> > the cgroup will double. Is that correct?
>

> yes.
>
>
> > If so, why is this better
> > than charging for actual swap usage?
>
> its behaviour is more determinstic and it uses less memory.
> (than nishimura-san's one, which charges for actual swap usage.)
>

Using less memory is good, but maybe not worth it if the result isn't so useful.

I'd say that it's less deterministic than nishimura-san's controller -
with his you just need to know how much swap is in use (which you can
tell by observing the app on a real system) but with yours you also
have to know whether there are any processes sharing anon pages (but
not mms).

Now it's true that if all the apps you need to run do an execve()
after forking, then the number of swap ptes really does track the
amount of swap space in use pretty accurately, since there's not going
to be any sharing of anon memory between mms. And it might be that
people decide that the reduced memory overhead justifies this
limitation. But I think it should be made explicit in the patch
description and documentation that this controller achieves its
reduced overhead at the cost of giving (IMO) bogus results on a rather
ancient but still perfectly legitimate class of Unix application. (The
apache httpd server used to work this way, for instance. It may still
but I've not looked at it in a while).

>
> > - what will happen if someone creates non-NPTL threads, which share an
> > mm but not a thread group (so each of them is a thread group leader)?
>
> a thread which is most recently assigned to a cgroup will "win".
>

Doesn't that risk triggering the BUG_ON(mm->swap_cgroup != oldscg) in
swap_cgroup_attach() ?

>
> > - if you were to store a pointer in the page rather than the
>
> "a pointer"? a pointer to what?

Oops, sorry - I meant to say "a pointer to the mm". So from there you
can get to mm->owner, and hence to mm->owner->cgroups[swap]

Paul

_____

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

---

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by yamamoto on Thu, 15 May 2008 08:56:06 GMT
View Forum Message <> Reply to Message

> > > If so, why is this better
> > > than charging for actual swap usage?
> >
> > its behaviour is more determinstic and it uses less memory.
> > (than nishimura-san's one, which charges for actual swap usage.)
> >
>
> Using less memory is good, but maybe not worth it if the result isn't so useful.
>
> I'd say that it's less deterministic than nishimura-san's controller -
> with his you just need to know how much swap is in use (which you can
> tell by observing the app on a real system) but with yours you also
> have to know whether there are any processes sharing anon pages (but
> not mms).

deterministic in the sense that, even when two or more processes
from different cgroups are sharing a page, both of them, rather than
only unlucky one, are always charged.

another related advantage is that it's possible to move charges
quite precisely when moving a task among cgroups.

> Now it's true that if all the apps you need to run do an execve()
> after forking, then the number of swap ptes really does track the
> amount of swap space in use pretty accurately, since there's not going
> to be any sharing of anon memory between mms. And it might be that
> people decide that the reduced memory overhead justifies this
> limitation. But I think it should be made explicit in the patch
> description and documentation that this controller achieves its
> reduced overhead at the cost of giving (IMO) bogus results on a rather
> ancient but still perfectly legitimate class of Unix application. (The
> apache httpd server used to work this way, for instance. It may still
> but I've not looked at it in a while).

fair enough.

> > > - what will happen if someone creates non-NPTL threads, which share an

> > > mm but not a thread group (so each of them is a thread group leader)?
> >
> > a thread which is most recently assigned to a cgroup will "win".
> >
>
> Doesn't that risk triggering the BUG_ON(mm->swap_cgroup != oldscg) in
> swap_cgroup_attach() ?

which version of the patch you are looking at?
the following is the latest copy.

YAMAMOTO Takashi

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

--- linux-2.6.25-rc8-mm2/mm/swapcontrol.c.BACKUP 2008-04-21 12:06:44.000000000 +0900
+++ linux-2.6.25-rc8-mm2/mm/swapcontrol.c 2008-05-15 13:48:00.000000000 +0900
@@ -0,0 +1,570 @@
+
+/*
+ * swapcontrol.c COPYRIGHT FUJITSU LIMITED 2008
+ *
+ * Author: yamamoto@valinux.co.jp
+ */
+
+/*
+ * there are two types of swap users.
+ * this controller handles both of them.
+ *
+ * - anonymous pages
+ * eg. user stacks, MAP_PRIVATE pages
+ *
+ * we track the number of PTEs with swap entries.
+ * it's precise wrt moving tasks between cgroups.
+ *
+ * - anonymous objects (aka "shmem")
+ * eg. tmpfs, sysvshm, MAP_SHARED anonymous mapping
+ *
+ * anonymous objects are associated to a cgroup when they are created
+ * and the number of on-disk swap slots used by them are counted
+ * for the cgroup. the association is persistent except cgroup removal,
+ * in which case, its associated objects are moved to init_mm's cgroup.
+ */
+
+#include <linux/err.h>
+#include <linux/cgroup.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>
+#include <linux/shmem_fs.h>
+#include <linux/hugetlb.h>
+
+struct swap_cgroup {
+ struct cgroup_subsys_state scg_css;
+ struct res_counter scg_counter;
+ struct list_head scg_shmem_list;
+};
+
+static struct cgroup_subsys_state *
+task_to_css(struct task_struct *task)
+{
+
+ return task_subsys_state(task, swap_cgroup_subsys_id);
+}
+
+static struct swap_cgroup *
+css_to_scg(struct cgroup_subsys_state *css)
+{
+
+ return container_of(css, struct swap_cgroup, scg_css);
+}
+
+static struct cgroup_subsys_state *
+cg_to_css(struct cgroup *cg)
+{
+
+ return cgroup_subsys_state(cg, swap_cgroup_subsys_id);
+}
+
+static struct swap_cgroup *
+cg_to_scg(struct cgroup *cg)
+{
+
+ return css_to_scg(cg_to_css(cg));
+}
+
+/* ================================================================== */
+/*
+ * anonymous pages
+ */
+
```

```
+/*
+ * lazily initialize ptp->ptp_swap_cgroup.
+ *
+ * called with page table locked.
+ */
+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);
+
+ /*
+  * scg == NULL here means that it's the first time we access
+  * this PTP.  in that case, initialize ptp->ptp_swap_cgroup
+  * from mm->swap_cgroup.
+  */
+ 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);
+}
+
+/*
+ * special version of swap_cgroup_charge/swap_cgroup_uncharge for mremap.
+ *
+ * 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);
+
+ /*
+  * normally we have nothing to do as these PTPs belong to the same mm.
+  */
+ if (oldscg == newscg)
+  return;
+
+ /*
+  * swap_cgroup_attach is in progress.
+  * it's an exceptional event.
+  *
+  * forcing charge here shouldn't matter much
+  * as the condition is likely transitional.
+  */
+
+ res_counter_charge_force(&newscg->scg_counter, PAGE_CACHE_SIZE);
+ res_counter_uncharge(&oldscg->scg_counter, PAGE_CACHE_SIZE);
+}
+
+struct swap_cgroup_attach_mm_cb_args {
+ struct vm_area_struct *vma;
+ struct swap_cgroup *oldscg;
+ struct swap_cgroup *newscg;
+};
+
+/*
+ * an mm_walk callback function.  used by swap_cgroup_attach_mm.
+ *
```

```
+ * investigate each PTEs in the range and adjust res_counters.
+ * note that the page table lock prevents concurrent attempts of
+ * charge/uncharge.
+ */
+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);
+ /*
+  * ptp->ptp_swap_cgroup == newscg here means this PTP is covered
+  * by another VMA as well.
+  */
+ 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,
+};
+
+/*
+ * walk VMAs to adjust res_counters.
+ */
+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 *oldmmscg;
+ struct swap_cgroup *oldtaskscg;
+ 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;
+ oldtaskscg = cg_to_scg(oldcg);
+ newscg = cg_to_scg(newcg);
+ BUG_ON(oldtaskscg == newscg);
+ /*
+  * note that a task and its mm can belong to different cgroups
+  * with the current implementation.
+  */
+ oldmmscg = mm->swap_cgroup;
+ if (oldmmscg != newscg) {
+  css_get(&newscg->scg_css);
+  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, oldmmscg, newscg);
+  css_put(&oldmmscg->scg_css);
+ }
+ mmput(mm);
+}
+
+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);
+}
+
+/* ============================================================ */
+/*
+ * anonymous objects (aka "shmem")
+ *
+ * locking rule:
+ * info->swap_cgroup_list and info->swap_cgroup can be read only
+ * if you have either of info->lock or swap_cgroup_shmem_lock.
+ * on the other hand, a writer needs both of these locks held.
+ * an exception: swap_cgroup_shmem_init and swap_cgroup_shmem_destroy doesn't
+ * need to hold info->lock because no other threads can reach the object
+ * without swap_cgroup_shmem_lock held.
+ */
+
+static DEFINE_MUTEX(swap_cgroup_shmem_lock);
+
+void
+swap_cgroup_shmem_init(struct shmem_inode_info *info)
+{
+ struct swap_cgroup *scg;
+
+ BUG_ON(info->swapped != 0);
+
+ INIT_LIST_HEAD(&info->swap_cgroup_list);
+
```

```
+ rcu_read_lock();
+ scg = css_to_scg(task_to_css(current));
+ css_get(&scg->scg_css);
+ rcu_read_unlock();
+
+ mutex_lock(&swap_cgroup_shmem_lock);
+ info->swap_cgroup = scg;
+ list_add(&info->swap_cgroup_list, &scg->scg_shmem_list);
+ mutex_unlock(&swap_cgroup_shmem_lock);
+
+#if 0 /* XXX see a comment in swap_cgroup_destroy_shmem */
+ css_put(&scg->scg_css);
+#endif
+}
+
+void
+swap_cgroup_shmem_destroy(struct shmem_inode_info *info)
+{
+ struct swap_cgroup *scg;
+
+ BUG_ON(list_empty(&info->swap_cgroup_list));
+ BUG_ON(info->swapped != 0);
+
+ mutex_lock(&swap_cgroup_shmem_lock);
+ scg = info->swap_cgroup;
+ list_del_init(&info->swap_cgroup_list);
+ mutex_unlock(&swap_cgroup_shmem_lock);
+
+ /* see a comment in swap_cgroup_destroy_shmem */
+ css_put(&scg->scg_css); /* XXX */
+}
+
+/*
+ * => called with info->lock held
+ */
+int
+swap_cgroup_shmem_charge(struct shmem_inode_info *info, long delta)
+{
+ struct swap_cgroup *scg = info->swap_cgroup;
+
+ BUG_ON(scg == NULL);
+ BUG_ON(list_empty(&info->swap_cgroup_list));
+
+ return res_counter_charge(&scg->scg_counter, delta << PAGE_CACHE_SHIFT);
+}
+
+/*
+ * => called with info->lock held
```

```
+ */
+void
+swap_cgroup_shmem_uncharge(struct shmem_inode_info *info, long delta)
+{
+ struct swap_cgroup *scg = info->swap_cgroup;
+
+ BUG_ON(scg == NULL);
+ BUG_ON(list_empty(&info->swap_cgroup_list));
+
+ res_counter_uncharge(&scg->scg_counter, delta << PAGE_CACHE_SHIFT);
+}
+
+static void
+swap_cgroup_destroy_shmem(struct swap_cgroup *scg)
+{
+ /*
+  * disabled as suggested by balbir.
+  *
+  * Message-ID: <48231FB6.7000206@linux.vnet.ibm.com>
+  * I would suggest that we fail deletion of a group for now.
+  * I have a set of patches for the cgroup hierarchy semantics.
+  * I think the parent is the best place to move it.
+  */
+#if 0
+ struct swap_cgroup *newscg;
+ struct list_head *pos;
+ struct list_head *next;
+
+ /*
+  * move our anonymous objects to init_mm's group.
+  *
+  * XXX
+  * possible alternatives:
+  * - make destroy fail.
+  * - move it to the parent cgroup.
+  * - make it orphan.
+  */
+ newscg = init_mm.swap_cgroup;
+ BUG_ON(newscg == NULL);
+ BUG_ON(newscg == scg);
+
+ mutex_lock(&swap_cgroup_shmem_lock);
+ list_for_each_safe(pos, next, &scg->scg_shmem_list) {
+  struct shmem_inode_info * const info =
+      list_entry(pos, struct shmem_inode_info, swap_cgroup_list);
+  long bytes;
+
+  spin_lock(&info->lock);
```

```
+  BUG_ON(info->swap_cgroup != scg);
+  bytes = info->swapped << PAGE_SHIFT;
+  info->swap_cgroup = newscg;
+  res_counter_uncharge(&scg->scg_counter, bytes);
+  res_counter_charge_force(&newscg->scg_counter, bytes);
+  spin_unlock(&info->lock);
+  list_del_init(&info->swap_cgroup_list);
+  list_add(&info->swap_cgroup_list, &newscg->scg_shmem_list);
+ }
+ mutex_unlock(&swap_cgroup_shmem_lock);
+#endif /* 0 */
+ BUG_ON(!list_empty(&scg->scg_shmem_list));
+}
+
+/* ================================================================== */
+/*
+ * common code and cgroup glue.
+ */
+
+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_strategy(char *buf, unsigned long long *resultp)
+{
+ unsigned long long val;
+ char *ep;
+
+ val = memparse(buf, &ep);
+ if (*ep)
+  return -EINVAL;
+ *resultp = roundup(val, PAGE_CACHE_SIZE);
+ return 0;
+}
+
+static ssize_t
+swap_cgroup_write(struct cgroup *cg, struct cftype *cft,
+    struct file *file, const char __user *buf, size_t nbytes, loff_t *ppos)
+{
+ struct res_counter *counter = &cg_to_scg(cg)->scg_counter;
+
+ BUG_ON(cft->private != RES_LIMIT);
+ return res_counter_write(counter, cft->private, buf, nbytes, ppos,
+     swap_cgroup_write_strategy);
```

```
+}
+
+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 = swap_cgroup_write,
+ },
+};
+
+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;
+ INIT_LIST_HEAD(&scg->scg_shmem_list);
+ return &scg->scg_css;
+}
+
+static void
+swap_cgroup_destroy(struct cgroup_subsys *ss, struct cgroup *cg)
+{
+ struct swap_cgroup *scg = cg_to_scg(cg);
+
+ swap_cgroup_destroy_shmem(scg);
+
+ BUG_ON(res_counter_read_u64(&scg->scg_counter, RES_USAGE) != 0);
+ kfree(scg);
+}
+
+static int
```

```
+swap_cgroup_populate(struct cgroup_subsys *ss, struct cgroup *cg)
+{
+
+ return cgroup_add_files(cg, ss, files, ARRAY_SIZE(files));
+}
+
+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-rc8-mm2/mm/memory.c.BACKUP 2008-04-21 10:04:08.000000000 +0900
+++ linux-2.6.25-rc8-mm2/mm/memory.c 2008-05-12 22:33:37.000000000 +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>
@@ -467,10 +468,10 @@ out:
  * 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;
@@ -481,6 +482,10 @@ 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))) {
```

```
@@ -530,6 +535,7 @@ 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,
@@ -540,6 +546,8 @@ 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;
@@ -551,6 +559,7 @@ 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
@@ -566,7 +575,10 @@ 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);

@@ -576,9 +588,9 @@ 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,
@@ -733,8 +745,11 @@ 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));

@@ -2155,6 +2170,7 @@ 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-rc8-mm2/mm/Makefile.BACKUP 2008-04-21 10:04:08.000000000 +0900
+++ linux-2.6.25-rc8-mm2/mm/Makefile 2008-04-21 12:06:44.000000000 +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-rc8-mm2/mm/rmap.c.BACKUP 2008-04-21 10:04:09.000000000 +0900
+++ linux-2.6.25-rc8-mm2/mm/rmap.c 2008-05-12 22:35:57.000000000 +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>

@@ -225,8 +226,9 @@ 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)
+static pte_t *__page_check_address(struct page *page, struct mm_struct *mm,
+      unsigned long address, spinlock_t **ptlp,
```

```
+     struct page **ptpp)
 {
  pgd_t *pgd;
  pud_t *pud;
@@ -257,12 +259,21 @@ 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_address(page, mm, address, ptlp, NULL);
+}
+
 /*
  * Subfunctions of page_referenced: page_referenced_one called
  * repeatedly from either page_referenced_anon or page_referenced_file.
@@ -700,13 +711,14 @@ 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_address(page, mm, address, &ptl, &ptp);
  if (!pte)
   goto out;

@@ -721,6 +733,12 @@ 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-rc8-mm2/mm/swapfile.c.BACKUP 2008-04-21 10:04:09.000000000 +0900
+++ linux-2.6.25-rc8-mm2/mm/swapfile.c 2008-04-21 12:06:44.000000000 +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-rc8-mm2/mm/shmem.c.BACKUP 2008-04-21 10:04:09.000000000 +0900
+++ linux-2.6.25-rc8-mm2/mm/shmem.c 2008-05-12 22:36:28.000000000 +0900
@@ -50,6 +50,7 @@
 #include <linux/migrate.h>
 #include <linux/highmem.h>
 #include <linux/seq_file.h>
+#include <linux/swapcontrol.h>

 #include <asm/uaccess.h>
 #include <asm/div64.h>
@@ -360,16 +361,25 @@ static swp_entry_t *shmem_swp_entry(stru
  return shmem_swp_map(subdir) + offset;
 }

-static void shmem_swp_set(struct shmem_inode_info *info, swp_entry_t *entry, unsigned long
value)
+static int shmem_swp_set(struct shmem_inode_info *info, swp_entry_t *entry, unsigned long
value)
 {
  long incdec = value? 1: -1;

+ if (incdec > 0) {
+  int error;
```

```
+
+  error = swap_cgroup_shmem_charge(info, incdec);
+  if (error)
+   return error;
+ } else
+  swap_cgroup_shmem_uncharge(info, -incdec);
  entry->val = value;
  info->swapped += incdec;
  if ((unsigned long)(entry - info->i_direct) >= SHMEM_NR_DIRECT) {
   struct page *page = kmap_atomic_to_page(entry);
   set_page_private(page, page_private(page) + incdec);
  }
+ return 0;
 }

 /**
@@ -727,6 +737,7 @@ done2:
  spin_lock(&info->lock);
  info->flags &= ~SHMEM_TRUNCATE;
  info->swapped -= nr_swaps_freed;
+ swap_cgroup_shmem_uncharge(info, nr_swaps_freed);
  if (nr_pages_to_free)
   shmem_free_blocks(inode, nr_pages_to_free);
  shmem_recalc_inode(inode);
@@ -1042,9 +1053,16 @@ static int shmem_writepage(struct page *
  shmem_recalc_inode(inode);

  if (swap.val && add_to_swap_cache(page, swap, GFP_ATOMIC) == 0) {
-  remove_from_page_cache(page);
-  shmem_swp_set(info, entry, swap.val);
+  int error;
+
+  error = shmem_swp_set(info, entry, swap.val);
   shmem_swp_unmap(entry);
+  if (error != 0) {
+   delete_from_swap_cache(page); /* does swap_free */
+   spin_unlock(&info->lock);
+   goto redirty;
+  }
+  remove_from_page_cache(page);
   if (list_empty(&info->swaplist))
    inode = igrab(inode);
   else
@@ -1522,6 +1540,7 @@ shmem_get_inode(struct super_block *sb,
    inode->i_fop = &shmem_file_operations;
    mpol_shared_policy_init(&info->policy,
        shmem_get_sbmpol(sbinfo));
+   swap_cgroup_shmem_init(info);
```

```
    break;
   case S_IFDIR:
    inc_nlink(inode);
@@ -2325,6 +2344,7 @@ static void shmem_destroy_inode(struct i
  if ((inode->i_mode & S_IFMT) == S_IFREG) {
   /* only struct inode is valid if it's an inline symlink */
   mpol_free_shared_policy(&SHMEM_I(inode)->policy);
+  swap_cgroup_shmem_destroy(SHMEM_I(inode));
  }
  shmem_acl_destroy_inode(inode);
  kmem_cache_free(shmem_inode_cachep, SHMEM_I(inode));
--- linux-2.6.25-rc8-mm2/mm/mremap.c.BACKUP 2008-04-02 04:44:26.000000000 +0900
+++ linux-2.6.25-rc8-mm2/mm/mremap.c 2008-05-12 22:35:12.000000000 +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 +110,9 @@ 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-rc8-mm2/include/linux/res_counter.h.BACKUP 2008-04-21 10:03:58.000000000
+0900
```

```
+++ linux-2.6.25-rc8-mm2/include/linux/res_counter.h 2008-04-21 12:06:50.000000000 +0900
@@ -97,6 +97,7 @@ 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-rc8-mm2/include/linux/swapcontrol.h.BACKUP 2008-04-21 12:06:45.000000000
+0900
+++ linux-2.6.25-rc8-mm2/include/linux/swapcontrol.h 2008-05-07 10:31:58.000000000 +0900
@@ -0,0 +1,94 @@
+
+/*
+ * swapcontrol.h COPYRIGHT FUJITSU LIMITED 2008
+ *
+ * Author: yamamoto@valinux.co.jp
+ */
+
+struct task_struct;
+struct mm_struct;
+struct page;
+struct shmem_inode_info;
+
+#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 *);
+
+void swap_cgroup_shmem_init(struct shmem_inode_info *);
+void swap_cgroup_shmem_destroy(struct shmem_inode_info *);
+int swap_cgroup_shmem_charge(struct shmem_inode_info *, long);
+void swap_cgroup_shmem_uncharge(struct shmem_inode_info *, long);
+
+#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 inline void
+swap_cgroup_remap_charge(struct page *oldptp, struct page *newptp,
+    struct mm_struct *mm)
+{
+
+ /* nothing */
+}
+
+static inline 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 */
+}
+
+static inline void
+swap_cgroup_shmem_init(struct shmem_inode_info *info)
+{
+
+ /* nothing */
+}
+
+static inline void
+swap_cgroup_shmem_destroy(struct shmem_inode_info *info)
+{
+
+ /* nothing */
+}
+
+static inline int
+swap_cgroup_shmem_charge(struct shmem_inode_info *info, long delta)
+{
+
+ /* nothing */
```

```
+ return 0;
+}
+
+static inline void
+swap_cgroup_shmem_uncharge(struct shmem_inode_info *info, long delta)
+{
+
+ /* nothing */
+}
+
+#endif /* defined(CONFIG_CGROUP_SWAP_RES_CTLR) */
--- linux-2.6.25-rc8-mm2/include/linux/shmem_fs.h.BACKUP 2008-04-21 10:03:59.000000000
+0900
+++ linux-2.6.25-rc8-mm2/include/linux/shmem_fs.h 2008-04-23 18:07:39.000000000 +0900
@@ -4,6 +4,8 @@
 #include <linux/swap.h>
 #include <linux/mempolicy.h>

+struct swap_cgroup;
+
 /* inode in-kernel data */

 #define SHMEM_NR_DIRECT 16
@@ -23,6 +25,10 @@ struct shmem_inode_info {
  struct posix_acl *i_acl;
  struct posix_acl *i_default_acl;
 #endif
+#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
+ struct swap_cgroup *swap_cgroup;
+ struct list_head swap_cgroup_list;
+#endif
 };

 struct shmem_sb_info {
--- linux-2.6.25-rc8-mm2/include/linux/cgroup_subsys.h.BACKUP 2008-04-21
10:03:52.000000000 +0900
+++ linux-2.6.25-rc8-mm2/include/linux/cgroup_subsys.h 2008-04-21 12:06:51.000000000 +0900
@@ -43,6 +43,12 @@ SUBSYS(mem_cgroup)

 /* */

+#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
+SUBSYS(swap_cgroup)
+#endif
+
+/* */
+
 #ifdef CONFIG_CGROUP_DEVICE
```

```
 SUBSYS(devices)
 #endif
--- linux-2.6.25-rc8-mm2/include/linux/mm_types.h.BACKUP 2008-04-21 10:03:56.000000000
+0900
+++ linux-2.6.25-rc8-mm2/include/linux/mm_types.h 2008-04-21 12:06:51.000000000 +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;
@@ -73,6 +74,7 @@ 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 !
@@ -233,6 +235,9 @@ 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-rc8-mm2/include/linux/mm.h.BACKUP 2008-04-21 10:03:56.000000000 +0900
+++ linux-2.6.25-rc8-mm2/include/linux/mm.h 2008-04-21 12:06:51.000000000 +0900
@@ -926,6 +926,7 @@ 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-rc8-mm2/kernel/res_counter.c.BACKUP 2008-04-21 10:04:06.000000000 +0900
+++ linux-2.6.25-rc8-mm2/kernel/res_counter.c 2008-04-21 12:06:51.000000000 +0900
@@ -44,6 +44,15 @@ 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-rc8-mm2/kernel/fork.c.BACKUP 2008-04-21 10:04:04.000000000 +0900
+++ linux-2.6.25-rc8-mm2/kernel/fork.c 2008-04-21 12:17:53.000000000 +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;
  }
@@ -417,6 +419,7 @@ void mmput(struct mm_struct *mm)
   }
   put_swap_token(mm);
   mm_free_cgroup(mm);
+  swap_cgroup_exit_mm(mm);
   mmdrop(mm);
  }
 }
--- linux-2.6.25-rc8-mm2/init/Kconfig.BACKUP 2008-04-21 10:04:04.000000000 +0900
+++ linux-2.6.25-rc8-mm2/init/Kconfig 2008-05-07 10:21:49.000000000 +0900
@@ -386,6 +386,12 @@ 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 && SWAP
+ help
+   Provides a resource controller to manage swap space.
```

+
 config SYSFS_DEPRECATED
  bool

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

---

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Daisuke Nishimura on Thu, 15 May 2008 12:01:53 GMT
View Forum Message <> Reply to Message

On 2008/05/15 17:56 +0900, YAMAMOTO Takashi wrote:
>>> > If so, why is this better
>>> > than charging for actual swap usage?
>>>
>>> its behaviour is more determinstic and it uses less memory.
>>> (than nishimura-san's one, which charges for actual swap usage.)
>>>

Consuming more memory cannot be helped for my controller...

>> Using less memory is good, but maybe not worth it if the result isn't so useful.
>>
>> I'd say that it's less deterministic than nishimura-san's controller -
>> with his you just need to know how much swap is in use (which you can
>> tell by observing the app on a real system) but with yours you also
>> have to know whether there are any processes sharing anon pages (but
>> not mms).
>
> deterministic in the sense that, even when two or more processes
> from different cgroups are sharing a page, both of them, rather than
> only unlucky one, are always charged.
>

I'm not sure whether this behavior itself is good or bad,
but I think it's not good idea to make memory controller,
which charges only one process for a shared page,
and swap controller behave differently.
I think it will be confusing for users. At least,
I would feel it strange.

> another related advantage is that it's possible to move charges
> quite precisely when moving a task among cgroups.
>

Moving charges is one of future todo of my controller.
But, as you say, it won't be so precise as yours.

Thanks,
Daisuke Nishimura.

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

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by yamamoto on Mon, 19 May 2008 04:14:57 GMT
View Forum Message <> Reply to Message

> > deterministic in the sense that, even when two or more processes
> > from different cgroups are sharing a page, both of them, rather than
> > only unlucky one, are always charged.
> >
>
> I'm not sure whether this behavior itself is good or bad,
> but I think it's not good idea to make memory controller,
> which charges only one process for a shared page,
> and swap controller behave differently.
> I think it will be confusing for users. At least,
> I would feel it strange.

i agree that yours can be better integrated with the memory controller.

unlike yours, mine was designed to be independent from
the memory controller as far as possible.
(i don't want to complicate the memory controller.)

YAMAMOTO Takashi

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

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by yamamoto on Thu, 22 May 2008 04:46:56 GMT
View Forum Message <> Reply to Message

> BTW, I'm just trying to make my swapcontroller patch
> that is rebased on recent kernel and implemented
> as part of memory controller.
> I'm going to submit it by the middle of May.

what's the status of this?

YAMAMOTO Takashi

Subject: Re: [RFC][PATCH] another swap controller for cgroup
Posted by Daisuke Nishimura on Thu, 22 May 2008 04:54:04 GMT
View Forum Message <> Reply to Message

On 2008/05/22 13:46 +0900, YAMAMOTO Takashi wrote:
>> BTW, I'm just trying to make my swapcontroller patch
>> that is rebased on recent kernel and implemented
>> as part of memory controller.
>> I'm going to submit it by the middle of May.
>
> what's the status of this?
>
> YAMAMOTO Takashi
>

Sorry.

I will post it very soon.
Actually, I've been writing mails :-)

Daisuke Nishimura.
