
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 deterministic 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, pti);
+
+ 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 *newcgr,
+ struct cgroup *oldcgr, struct task_struct *t)
+{
+ struct swap_cgroup *oldmmscg;
+ struct swap_cgroup *oldtaskscg;
+ struct swap_cgroup *newscg;
+ struct mm_struct *mm;
+
+ BUG_ON(oldcgr == NULL);

```

```

+ BUG_ON(newcgroup == NULL);
+ BUG_ON(cgroup_to_css(oldcgroup) == NULL);
+ BUG_ON(cgroup_to_css(newcgroup) == NULL);
+ BUG_ON(oldcgroup == newcgroup);
+
+ if (!thread_group_leader(t))
+ return;
+ mm = get_task_mm(t);
+ if (mm == NULL)
+ return;
+ oldtaskscgroup = cgroup_to_scgroup(oldcgroup);
+ newscgroup = cgroup_to_scgroup(newcgroup);
+ BUG_ON(oldtaskscgroup == newscgroup);
+ /*
+  * note that a task and its mm can belong to different cgroups
+  * with the current implementation.
+  */
+ oldmmsscgroup = mm->swap_cgroup;
+ if (oldmmsscgroup != newscgroup) {
+ css_get(&newscgroup->scgroup_css);
+ mm->swap_cgroup = newscgroup;
+ /*
+  * 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, oldmmsscgroup, newscgroup);
+ css_put(&oldmmsscgroup->scgroup_css);
+ }
+ mmput(mm);
+}
+
+void
+swap_cgroup_init_mm(struct mm_struct *mm, struct task_struct *t)
+{
+ struct swap_cgroup *scgroup;
+ 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);
+ scgroup = cgroup_to_scgroup(cg);
+ BUG_ON(scgroup == NULL);
+ css_get(&scgroup->scgroup_css);
+ mm->swap_cgroup = scgroup;
+}

```



```

+
+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_mkdirty(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 = iggrab(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

