Subject: [PATCH 25/33] memory controller memory accounting v7
Posted by Paul Menage on Mon, 17 Sep 2007 21:03:32 GMT
View Forum Message <> Reply to Message

From: Balbir Singh <balbir@linux.vnet.ibm.com>
(container->cgroup renaming by Paul Menage <menage@google.com>)

Add the accounting hooks. The accounting is carried out for RSS and Page
Cache (unmapped) pages. There is now a common limit and accounting for both.
The RSS accounting is accounted at page_add_*_rmap() and page_remove_rmap()
time. Page cache is accounted at add_to_page_cache(),
__delete_from_page_cache(). Swap cache is also accounted for.

Each page's page_cgroup is protected with the last bit of the
page_cgroup pointer, this makes handling of race conditions involving
simultaneous mappings of a page easier. A reference count is kept in the
page_cgroup to deal with cases where a page might be unmapped from the RSS
of all tasks, but still lives in the page cache.

Credits go to Vaidyanathan Srinivasan for helping with reference counting work
of the page cgroup. Almost all of the page cache accounting code has help
from Vaidyanathan Srinivasan.

Signed-off-by: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
Signed-off-by: Paul Menage <menage@google.com>

```
 include/linux/memcontrol.h |   20 ++++
 mm/filemap.c               |   12 ++
 mm/memcontrol.c            |  166 +++++++++++++++++++++++++++++++++++++++++-
 mm/memory.c                |   43 ++++++++-
 mm/migrate.c               |    6 +
 mm/page_alloc.c            |    3
 mm/rmap.c                  |   17 +++
 mm/swap_state.c            |   12 ++
 mm/swapfile.c              |   41 +++++---
 9 files changed, 293 insertions(+), 27 deletions(-)
```

```
diff -puN include/linux/memcontrol.h~memory-controller-memory-accounting-v7
include/linux/memcontrol.h
--- a/include/linux/memcontrol.h~memory-controller-memory-accounting-v7
+++ a/include/linux/memcontrol.h
@@ -30,6 +30,13 @@ extern void mm_free_cgroup(struct mm_
 extern void page_assign_page_cgroup(struct page *page,
     struct page_cgroup *pc);
 extern struct page_cgroup *page_get_page_cgroup(struct page *page);
+extern int mem_cgroup_charge(struct page *page, struct mm_struct *mm);
```

```
+extern void mem_cgroup_uncharge(struct page_cgroup *pc);
+
+static inline void mem_cgroup_uncharge_page(struct page *page)
+{
+ mem_cgroup_uncharge(page_get_page_cgroup(page));
+}

 #else /* CONFIG_CGROUP_MEM_CONT */
 static inline void mm_init_cgroup(struct mm_struct *mm,
@@ -51,6 +58,19 @@ static inline struct page_cgroup *pag
  return NULL;
 }

+static inline int mem_cgroup_charge(struct page *page, struct mm_struct *mm)
+{
+ return 0;
+}
+
+static inline void mem_cgroup_uncharge(struct page_cgroup *pc)
+{
+}
+
+static inline void mem_cgroup_uncharge_page(struct page *page)
+{
+}
+
 #endif /* CONFIG_CGROUP_MEM_CONT */

 #endif /* _LINUX_MEMCONTROL_H */
diff -puN mm/filemap.c~memory-controller-memory-accounting-v7 mm/filemap.c
--- a/mm/filemap.c~memory-controller-memory-accounting-v7
+++ a/mm/filemap.c
@@ -31,6 +31,7 @@
 #include <linux/syscalls.h>
 #include <linux/cpuset.h>
 #include <linux/hardirq.h> /* for BUG_ON(!in_atomic()) only */
+#include <linux/memcontrol.h>
 #include "internal.h"

 /*
@@ -116,6 +117,7 @@ void __remove_from_page_cache(struct pag
 {
  struct address_space *mapping = page->mapping;

+ mem_cgroup_uncharge_page(page);
  radix_tree_delete(&mapping->page_tree, page->index);
  page->mapping = NULL;
  mapping->nrpages--;
```

```
@@ -441,6 +443,11 @@ int add_to_page_cache(struct page *page,
  int error = radix_tree_preload(gfp_mask & ~__GFP_HIGHMEM);

  if (error == 0) {
+
+  error = mem_cgroup_charge(page, current->mm);
+  if (error)
+   goto out;
+
   write_lock_irq(&mapping->tree_lock);
   error = radix_tree_insert(&mapping->page_tree, offset, page);
   if (!error) {
@@ -450,10 +457,13 @@ int add_to_page_cache(struct page *page,
    page->index = offset;
    mapping->nrpages++;
    __inc_zone_page_state(page, NR_FILE_PAGES);
-  }
+  } else
+   mem_cgroup_uncharge_page(page);
+
   write_unlock_irq(&mapping->tree_lock);
   radix_tree_preload_end();
  }
+out:
  return error;
 }
 EXPORT_SYMBOL(add_to_page_cache);
diff -puN mm/memcontrol.c~memory-controller-memory-accounting-v7 mm/memcontrol.c
--- a/mm/memcontrol.c~memory-controller-memory-accounting-v7
+++ a/mm/memcontrol.c
@@ -21,6 +21,9 @@
 #include <linux/memcontrol.h>
 #include <linux/cgroup.h>
 #include <linux/mm.h>
+#include <linux/page-flags.h>
+#include <linux/bit_spinlock.h>
+#include <linux/rcupdate.h>

 struct cgroup_subsys mem_cgroup_subsys;

@@ -31,7 +34,9 @@ struct cgroup_subsys mem_cgroup_su
  * to help the administrator determine what knobs to tune.
  *
  * TODO: Add a water mark for the memory controller. Reclaim will begin when
- * we hit the water mark.
+ * we hit the water mark. May be even add a low water mark, such that
+ * no reclaim occurs from a cgroup at it's low water mark, this is
+ * a feature that will be implemented much later in the future.
```

```
 */
 struct mem_cgroup {
  struct cgroup_subsys_state css;
@@ -49,6 +54,14 @@ struct mem_cgroup {
 };

 /*
+ * We use the lower bit of the page->page_cgroup pointer as a bit spin
+ * lock. We need to ensure that page->page_cgroup is atleast two
+ * byte aligned (based on comments from Nick Piggin)
+ */
+#define PAGE_CGROUP_LOCK_BIT  0x0
+#define PAGE_CGROUP_LOCK    (1 << PAGE_CGROUP_LOCK_BIT)
+
+/*
  * A page_cgroup page is associated with every page descriptor. The
  * page_cgroup helps us identify information about the cgroup
  */
@@ -56,6 +69,8 @@ struct page_cgroup {
  struct list_head lru;  /* per cgroup LRU list */
  struct page *page;
  struct mem_cgroup *mem_cgroup;
+ atomic_t ref_cnt;  /* Helpful when pages move b/w  */
+     /* mapped and cached states     */
 };


@@ -88,14 +103,157 @@ void mm_free_cgroup(struct mm_struct
  css_put(&mm->mem_cgroup->css);
 }

+static inline int page_cgroup_locked(struct page *page)
+{
+ return bit_spin_is_locked(PAGE_CGROUP_LOCK_BIT,
+     &page->page_cgroup);
+}
+
 void page_assign_page_cgroup(struct page *page, struct page_cgroup *pc)
 {
- page->page_cgroup = (unsigned long)pc;
+ int locked;
+
+ /*
+  * While resetting the page_cgroup we might not hold the
+  * page_cgroup lock. free_hot_cold_page() is an example
+  * of such a scenario
+  */
+ if (pc)
```

```
+  VM_BUG_ON(!page_cgroup_locked(page));
+ locked = (page->page_cgroup & PAGE_CGROUP_LOCK);
+ page->page_cgroup = ((unsigned long)pc | locked);
 }

 struct page_cgroup *page_get_page_cgroup(struct page *page)
 {
- return page->page_cgroup;
+ return (struct page_cgroup *)
+  (page->page_cgroup & ~PAGE_CGROUP_LOCK);
+}
+
+void __always_inline lock_page_cgroup(struct page *page)
+{
+ bit_spin_lock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
+ VM_BUG_ON(!page_cgroup_locked(page));
+}
+
+void __always_inline unlock_page_cgroup(struct page *page)
+{
+ bit_spin_unlock(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
+}
+
+/*
+ * Charge the memory controller for page usage.
+ * Return
+ * 0 if the charge was successful
+ * < 0 if the cgroup is over its limit
+ */
+int mem_cgroup_charge(struct page *page, struct mm_struct *mm)
+{
+ struct mem_cgroup *mem;
+ struct page_cgroup *pc, *race_pc;
+
+ /*
+  * Should page_cgroup's go to their own slab?
+  * One could optimize the performance of the charging routine
+  * by saving a bit in the page_flags and using it as a lock
+  * to see if the cgroup page already has a page_cgroup associated
+  * with it
+  */
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ /*
+  * The page_cgroup exists and the page has already been accounted
+  */
+ if (pc) {
+  atomic_inc(&pc->ref_cnt);
```

```
+  goto done;
+ }
+
+ unlock_page_cgroup(page);
+
+ pc = kzalloc(sizeof(struct page_cgroup), GFP_KERNEL);
+ if (pc == NULL)
+  goto err;
+
+ rcu_read_lock();
+ /*
+  * We always charge the cgroup the mm_struct belongs to
+  * the mm_struct's mem_cgroup changes on task migration if the
+  * thread group leader migrates. It's possible that mm is not
+  * set, if so charge the init_mm (happens for pagecache usage).
+  */
+ if (!mm)
+  mm = &init_mm;
+
+ mem = rcu_dereference(mm->mem_cgroup);
+ /*
+  * For every charge from the cgroup, increment reference
+  * count
+  */
+ css_get(&mem->css);
+ rcu_read_unlock();
+
+ /*
+  * If we created the page_cgroup, we should free it on exceeding
+  * the cgroup limit.
+  */
+ if (res_counter_charge(&mem->res, 1)) {
+  css_put(&mem->css);
+  goto free_pc;
+ }
+
+ lock_page_cgroup(page);
+ /*
+  * Check if somebody else beat us to allocating the page_cgroup
+  */
+ race_pc = page_get_page_cgroup(page);
+ if (race_pc) {
+  kfree(pc);
+  pc = race_pc;
+  atomic_inc(&pc->ref_cnt);
+  res_counter_uncharge(&mem->res, 1);
+  css_put(&mem->css);
+  goto done;
```

```
+ }
+
+ atomic_set(&pc->ref_cnt, 1);
+ pc->mem_cgroup = mem;
+ pc->page = page;
+ page_assign_page_cgroup(page, pc);
+
+done:
+ unlock_page_cgroup(page);
+ return 0;
+free_pc:
+ kfree(pc);
+ return -ENOMEM;
+err:
+ unlock_page_cgroup(page);
+ return -ENOMEM;
+}
+
+/*
+ * Uncharging is always a welcome operation, we never complain, simply
+ * uncharge.
+ */
+void mem_cgroup_uncharge(struct page_cgroup *pc)
+{
+ struct mem_cgroup *mem;
+ struct page *page;
+
+ if (!pc)
+  return;
+
+ if (atomic_dec_and_test(&pc->ref_cnt)) {
+  page = pc->page;
+  lock_page_cgroup(page);
+  mem = pc->mem_cgroup;
+  css_put(&mem->css);
+  page_assign_page_cgroup(page, NULL);
+  unlock_page_cgroup(page);
+  res_counter_uncharge(&mem->res, 1);
+  kfree(pc);
+ }
 }

 static ssize_t mem_cgroup_read(struct cgroup *cont, struct cftype *cft,
@@ -150,6 +308,8 @@ mem_cgroup_create(struct cgroup_su
   return NULL;

  res_counter_init(&mem->res);
+ INIT_LIST_HEAD(&mem->active_list);
```

```
+ INIT_LIST_HEAD(&mem->inactive_list);
  return &mem->css;
 }

diff -puN mm/memory.c~memory-controller-memory-accounting-v7 mm/memory.c
--- a/mm/memory.c~memory-controller-memory-accounting-v7
+++ a/mm/memory.c
@@ -50,6 +50,7 @@
 #include <linux/delayacct.h>
 #include <linux/init.h>
 #include <linux/writeback.h>
+#include <linux/memcontrol.h>

 #include <asm/pgalloc.h>
 #include <asm/uaccess.h>
@@ -1136,14 +1137,18 @@ static int insert_page(struct mm_struct
  pte_t *pte;
  spinlock_t *ptl;

+ retval = mem_cgroup_charge(page, mm);
+ if (retval)
+  goto out;
+
  retval = -EINVAL;
  if (PageAnon(page))
-  goto out;
+  goto out_uncharge;
  retval = -ENOMEM;
  flush_dcache_page(page);
  pte = get_locked_pte(mm, addr, &ptl);
  if (!pte)
-  goto out;
+  goto out_uncharge;
  retval = -EBUSY;
  if (!pte_none(*pte))
   goto out_unlock;
@@ -1155,8 +1160,11 @@ static int insert_page(struct mm_struct
  set_pte_at(mm, addr, pte, mk_pte(page, prot));

  retval = 0;
+ return retval;
 out_unlock:
  pte_unmap_unlock(pte, ptl);
+out_uncharge:
+ mem_cgroup_uncharge_page(page);
 out:
  return retval;
 }
```

```
@@ -1629,6 +1637,9 @@ gotten:
   goto oom;
  cow_user_page(new_page, old_page, address, vma);

+ if (mem_cgroup_charge(new_page, mm))
+  goto oom_free_new;
+
  /*
   * Re-check the pte - we dropped the lock
   */
@@ -1660,7 +1671,9 @@ gotten:
   /* Free the old page.. */
   new_page = old_page;
   ret |= VM_FAULT_WRITE;
- }
+ } else
+  mem_cgroup_uncharge_page(new_page);
+
  if (new_page)
   page_cache_release(new_page);
  if (old_page)
@@ -1681,6 +1694,8 @@ unlock:
   put_page(dirty_page);
  }
  return ret;
+oom_free_new:
+ __free_page(new_page);
 oom:
  if (old_page)
   page_cache_release(old_page);
@@ -2085,6 +2100,11 @@ static int do_swap_page(struct mm_struct
  }

  delayacct_clear_flag(DELAYACCT_PF_SWAPIN);
+ if (mem_cgroup_charge(page, mm)) {
+  ret = VM_FAULT_OOM;
+  goto out;
+ }
+
  mark_page_accessed(page);
  lock_page(page);

@@ -2121,8 +2141,10 @@ static int do_swap_page(struct mm_struct
  if (write_access) {
   /* XXX: We could OR the do_wp_page code with this one? */
   if (do_wp_page(mm, vma, address,
-    page_table, pmd, ptl, pte) & VM_FAULT_OOM)
+    page_table, pmd, ptl, pte) & VM_FAULT_OOM) {
```

```
+   mem_cgroup_uncharge_page(page);
    ret = VM_FAULT_OOM;
+  }
   goto out;
  }

@@ -2133,6 +2155,7 @@ unlock:
 out:
  return ret;
 out_nomap:
+ mem_cgroup_uncharge_page(page);
  pte_unmap_unlock(page_table, ptl);
  unlock_page(page);
  page_cache_release(page);
@@ -2161,6 +2184,9 @@ static int do_anonymous_page(struct mm_s
  if (!page)
   goto oom;

+  if (mem_cgroup_charge(page, mm))
+   goto oom_free_page;
+
  entry = mk_pte(page, vma->vm_page_prot);
  entry = maybe_mkwrite(pte_mkdirty(entry), vma);

@@ -2178,8 +2204,11 @@ unlock:
  pte_unmap_unlock(page_table, ptl);
  return 0;
 release:
+ mem_cgroup_uncharge_page(page);
  page_cache_release(page);
  goto unlock;
+oom_free_page:
+ __free_page(page);
 oom:
  return VM_FAULT_OOM;
 }
@@ -2290,6 +2319,11 @@ static int __do_fault(struct mm_struct *

  }

+ if (mem_cgroup_charge(page, mm)) {
+  ret = VM_FAULT_OOM;
+  goto out;
+ }
+
  page_table = pte_offset_map_lock(mm, pmd, address, &ptl);

  /*
```

```
@@ -2325,6 +2359,7 @@ static int __do_fault(struct mm_struct *
   /* no need to invalidate: a not-present page won't be cached */
   update_mmu_cache(vma, address, entry);
  } else {
+  mem_cgroup_uncharge_page(page);
   if (anon)
    page_cache_release(page);
   else
diff -puN mm/migrate.c~memory-controller-memory-accounting-v7 mm/migrate.c
--- a/mm/migrate.c~memory-controller-memory-accounting-v7
+++ a/mm/migrate.c
@@ -29,6 +29,7 @@
 #include <linux/mempolicy.h>
 #include <linux/vmalloc.h>
 #include <linux/security.h>
+#include <linux/memcontrol.h>

 #include "internal.h"

@@ -157,6 +158,11 @@ static void remove_migration_pte(struct
    return;
   }

+ if (mem_cgroup_charge(new, mm)) {
+  pte_unmap(ptep);
+  return;
+ }
+
   ptl = pte_lockptr(mm, pmd);
   spin_lock(ptl);
  pte = *ptep;
diff -puN mm/page_alloc.c~memory-controller-memory-accounting-v7 mm/page_alloc.c
--- a/mm/page_alloc.c~memory-controller-memory-accounting-v7
+++ a/mm/page_alloc.c
@@ -42,6 +42,7 @@
 #include <linux/backing-dev.h>
 #include <linux/fault-inject.h>
 #include <linux/page-isolation.h>
+#include <linux/memcontrol.h>

 #include <asm/tlbflush.h>
 #include <asm/div64.h>
@@ -1019,6 +1020,7 @@ static void fastcall free_hot_cold_page(

  if (!PageHighMem(page))
   debug_check_no_locks_freed(page_address(page), PAGE_SIZE);
+ page_assign_page_cgroup(page, NULL);
  arch_free_page(page, 0);
```

```
  kernel_map_pages(page, 1, 0);

@@ -2561,6 +2563,7 @@ void __meminit memmap_init_zone(unsigned
   set_page_links(page, zone, nid, pfn);
   init_page_count(page);
   reset_page_mapcount(page);
+  page_assign_page_cgroup(page, NULL);
   SetPageReserved(page);

   /*
diff -puN mm/rmap.c~memory-controller-memory-accounting-v7 mm/rmap.c
--- a/mm/rmap.c~memory-controller-memory-accounting-v7
+++ a/mm/rmap.c
@@ -48,6 +48,7 @@
 #include <linux/rcupdate.h>
 #include <linux/module.h>
 #include <linux/kallsyms.h>
+#include <linux/memcontrol.h>

 #include <asm/tlbflush.h>

@@ -550,8 +551,14 @@ void page_add_anon_rmap(struct page *pag
  VM_BUG_ON(address < vma->vm_start || address >= vma->vm_end);
  if (atomic_inc_and_test(&page->_mapcount))
   __page_set_anon_rmap(page, vma, address);
- else
+ else {
   __page_check_anon_rmap(page, vma, address);
+  /*
+   * We unconditionally charged during prepare, we uncharge here
+   * This takes care of balancing the reference counts
+   */
+  mem_cgroup_uncharge_page(page);
+ }
 }

 /*
@@ -582,6 +589,12 @@ void page_add_file_rmap(struct page *pag
 {
  if (atomic_inc_and_test(&page->_mapcount))
   __inc_zone_page_state(page, NR_FILE_MAPPED);
+ else
+  /*
+   * We unconditionally charged during prepare, we uncharge here
+   * This takes care of balancing the reference counts
+   */
+  mem_cgroup_uncharge_page(page);
 }
```

```
 #ifdef CONFIG_DEBUG_VM
@@ -642,6 +655,8 @@ void page_remove_rmap(struct page *page,
    page_clear_dirty(page);
    set_page_dirty(page);
   }
+  mem_cgroup_uncharge_page(page);
+
   __dec_zone_page_state(page,
     PageAnon(page) ? NR_ANON_PAGES : NR_FILE_MAPPED);
  }
diff -puN mm/swap_state.c~memory-controller-memory-accounting-v7 mm/swap_state.c
--- a/mm/swap_state.c~memory-controller-memory-accounting-v7
+++ a/mm/swap_state.c
@@ -16,6 +16,7 @@
 #include <linux/backing-dev.h>
 #include <linux/pagevec.h>
 #include <linux/migrate.h>
+#include <linux/memcontrol.h>

 #include <asm/pgtable.h>

@@ -79,6 +80,11 @@ static int __add_to_swap_cache(struct pa
  BUG_ON(PagePrivate(page));
  error = radix_tree_preload(gfp_mask);
  if (!error) {
+
+  error = mem_cgroup_charge(page, current->mm);
+  if (error)
+   goto out;
+
   write_lock_irq(&swapper_space.tree_lock);
   error = radix_tree_insert(&swapper_space.page_tree,
       entry.val, page);
@@ -88,10 +94,13 @@ static int __add_to_swap_cache(struct pa
    set_page_private(page, entry.val);
    total_swapcache_pages++;
    __inc_zone_page_state(page, NR_FILE_PAGES);
-  }
+  } else
+   mem_cgroup_uncharge_page(page);
+
   write_unlock_irq(&swapper_space.tree_lock);
   radix_tree_preload_end();
  }
+out:
  return error;
 }
```

```
@@ -131,6 +140,7 @@ void __delete_from_swap_cache(struct pag
  BUG_ON(PageWriteback(page));
  BUG_ON(PagePrivate(page));

+ mem_cgroup_uncharge_page(page);
  radix_tree_delete(&swapper_space.page_tree, page_private(page));
  set_page_private(page, 0);
  ClearPageSwapCache(page);
diff -puN mm/swapfile.c~memory-controller-memory-accounting-v7 mm/swapfile.c
--- a/mm/swapfile.c~memory-controller-memory-accounting-v7
+++ a/mm/swapfile.c
@@ -27,6 +27,7 @@
 #include <linux/mutex.h>
 #include <linux/capability.h>
 #include <linux/syscalls.h>
+#include <linux/memcontrol.h>

 #include <asm/pgtable.h>
 #include <asm/tlbflush.h>
@@ -506,9 +507,12 @@ unsigned int count_swap_pages(int type,
  * just let do_wp_page work it out if a write is requested later - to
  * force COW, vm_page_prot omits write permission from any private vma.
  */
-static void unuse_pte(struct vm_area_struct *vma, pte_t *pte,
+static int unuse_pte(struct vm_area_struct *vma, pte_t *pte,
   unsigned long addr, swp_entry_t entry, struct page *page)
 {
+ if (mem_cgroup_charge(page, vma->vm_mm))
+  return -ENOMEM;
+
  inc_mm_counter(vma->vm_mm, anon_rss);
  get_page(page);
  set_pte_at(vma->vm_mm, addr, pte,
@@ -520,6 +524,7 @@ static void unuse_pte(struct vm_area_str
   * immediately swapped out again after swapon.
   */
  activate_page(page);
+ return 1;
 }

 static int unuse_pte_range(struct vm_area_struct *vma, pmd_t *pmd,
@@ -529,7 +534,7 @@ static int unuse_pte_range(struct vm_are
  pte_t swp_pte = swp_entry_to_pte(entry);
  pte_t *pte;
  spinlock_t *ptl;
- int found = 0;
+ int ret = 0;
```

```
  pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
  do {
@@ -538,13 +543,12 @@ static int unuse_pte_range(struct vm_are
     * Test inline before going to call unuse_pte.
     */
    if (unlikely(pte_same(*pte, swp_pte))) {
-    unuse_pte(vma, pte++, addr, entry, page);
-    found = 1;
+    ret = unuse_pte(vma, pte++, addr, entry, page);
     break;
    }
  } while (pte++, addr += PAGE_SIZE, addr != end);
  pte_unmap_unlock(pte - 1, ptl);
- return found;
+ return ret;
 }

 static inline int unuse_pmd_range(struct vm_area_struct *vma, pud_t *pud,
@@ -553,14 +557,16 @@ static inline int unuse_pmd_range(struct
 {
  pmd_t *pmd;
  unsigned long next;
+ int ret;

  pmd = pmd_offset(pud, addr);
  do {
   next = pmd_addr_end(addr, end);
   if (pmd_none_or_clear_bad(pmd))
    continue;
-  if (unuse_pte_range(vma, pmd, addr, next, entry, page))
-   return 1;
+  ret = unuse_pte_range(vma, pmd, addr, next, entry, page);
+  if (ret)
+   return ret;
  } while (pmd++, addr = next, addr != end);
  return 0;
 }
@@ -571,14 +577,16 @@ static inline int unuse_pud_range(struct
 {
  pud_t *pud;
  unsigned long next;
+ int ret;

  pud = pud_offset(pgd, addr);
  do {
   next = pud_addr_end(addr, end);
   if (pud_none_or_clear_bad(pud))
```

```
    continue;
-  if (unuse_pmd_range(vma, pud, addr, next, entry, page))
-   return 1;
+  ret = unuse_pmd_range(vma, pud, addr, next, entry, page);
+  if (ret)
+   return ret;
  } while (pud++, addr = next, addr != end);
  return 0;
 }
@@ -588,6 +596,7 @@ static int unuse_vma(struct vm_area_stru
 {
  pgd_t *pgd;
  unsigned long addr, end, next;
+ int ret;

  if (page->mapping) {
   addr = page_address_in_vma(page, vma);
@@ -605,8 +614,9 @@ static int unuse_vma(struct vm_area_stru
   next = pgd_addr_end(addr, end);
   if (pgd_none_or_clear_bad(pgd))
    continue;
-  if (unuse_pud_range(vma, pgd, addr, next, entry, page))
-   return 1;
+  ret = unuse_pud_range(vma, pgd, addr, next, entry, page);
+  if (ret)
+   return ret;
  } while (pgd++, addr = next, addr != end);
  return 0;
 }
@@ -615,6 +625,7 @@ static int unuse_mm(struct mm_struct *mm
     swp_entry_t entry, struct page *page)
 {
  struct vm_area_struct *vma;
+ int ret = 0;

  if (!down_read_trylock(&mm->mmap_sem)) {
   /*
@@ -627,15 +638,11 @@ static int unuse_mm(struct mm_struct *mm
   lock_page(page);
  }
  for (vma = mm->mmap; vma; vma = vma->vm_next) {
-  if (vma->anon_vma && unuse_vma(vma, entry, page))
+  if (vma->anon_vma && (ret = unuse_vma(vma, entry, page)))
    break;
  }
  up_read(&mm->mmap_sem);
- /*
-  * Currently unuse_mm cannot fail, but leave error handling
```

- * at call sites for now, since we change it from time to time.
- */
- return 0;
+ return ret;
 }

 /*

—

--

_____

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