Subject: [PATCH 4/6 mm] memcgroup: reinstate swapoff mod
Posted by Hugh Dickins on Fri, 09 Nov 2007 07:12:03 GMT
View Forum Message <> Reply to Message

This patch reinstates the "swapoff: scan ptes preemptibly" mod we started
with: in due course it should be rendered down into the earlier patches,
leaving us with a more straightforward mem_cgroup_charge mod to unuse_pte,
allocating with GFP_KERNEL while holding no spinlock and no atomic kmap.

Signed-off-by: Hugh Dickins <hugh@veritas.com>

Insert just after memory-controller-make-charging-gfp-mask-aware.patch or
you may prefer to insert 4-6 all together before memory-cgroup-enhancements

 mm/swapfile.c | 42 ++++++++++++++++++++++++++++++++++++++--------
 1 file changed, 34 insertions(+), 8 deletions(-)

```
--- patch3/mm/swapfile.c 2007-11-08 15:48:08.000000000 +0000
+++ patch4/mm/swapfile.c 2007-11-08 15:55:12.000000000 +0000
@@ -507,11 +507,23 @@ 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 int unuse_pte(struct vm_area_struct *vma, pte_t *pte,
+static int unuse_pte(struct vm_area_struct *vma, pmd_t *pmd,
   unsigned long addr, swp_entry_t entry, struct page *page)
 {
+ spinlock_t *ptl;
+ pte_t *pte;
+ int ret = 1;
+
  if (mem_cgroup_charge(page, vma->vm_mm, GFP_KERNEL))
-  return -ENOMEM;
+  ret = -ENOMEM;
+
+ pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
+ if (unlikely(!pte_same(*pte, swp_entry_to_pte(entry)))) {
+  if (ret > 0)
+   mem_cgroup_uncharge_page(page);
+  ret = 0;
+  goto out;
+ }

  inc_mm_counter(vma->vm_mm, anon_rss);
  get_page(page);
@@ -524,7 +536,9 @@ static int unuse_pte(struct vm_area_stru
   * immediately swapped out again after swapon.
   */
```

```
  activate_page(page);
- return 1;
+out:
+ pte_unmap_unlock(pte, ptl);
+ return ret;
 }

 static int unuse_pte_range(struct vm_area_struct *vma, pmd_t *pmd,
@@ -533,21 +547,33 @@ static int unuse_pte_range(struct vm_are
 {
  pte_t swp_pte = swp_entry_to_pte(entry);
  pte_t *pte;
- spinlock_t *ptl;
  int ret = 0;

- pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
+ /*
+  * We don't actually need pte lock while scanning for swp_pte: since
+  * we hold page lock and mmap_sem, swp_pte cannot be inserted into the
+  * page table while we're scanning; though it could get zapped, and on
+  * some architectures (e.g. x86_32 with PAE) we might catch a glimpse
+  * of unmatched parts which look like swp_pte, so unuse_pte must
+  * recheck under pte lock.  Scanning without pte lock lets it be
+  * preemptible whenever CONFIG_PREEMPT but not CONFIG_HIGHPTE.
+  */
+ pte = pte_offset_map(pmd, addr);
  do {
   /*
    * swapoff spends a _lot_ of time in this loop!
    * Test inline before going to call unuse_pte.
    */
   if (unlikely(pte_same(*pte, swp_pte))) {
-   ret = unuse_pte(vma, pte++, addr, entry, page);
-   break;
+   pte_unmap(pte);
+   ret = unuse_pte(vma, pmd, addr, entry, page);
+   if (ret)
+    goto out;
+   pte = pte_offset_map(pmd, addr);
   }
  } while (pte++, addr += PAGE_SIZE, addr != end);
- pte_unmap_unlock(pte - 1, ptl);
+ pte_unmap(pte - 1);
+out:
  return ret;
 }
```

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