Subject: [PATCH 1/6 mm] swapoff: scan ptes preemptibly
Posted by Hugh Dickins on Fri, 09 Nov 2007 07:08:48 GMT
View Forum Message <> Reply to Message

Provided that CONFIG_HIGHPTE is not set, unuse_pte_range can reduce latency
in swapoff by scanning the page table preemptibly: so long as unuse_pte is
careful to recheck that entry under pte lock.

(To tell the truth, this patch was not inspired by any cries for lower
latency here: rather, this restructuring permits a future memory controller
patch to allocate with GFP_KERNEL in unuse_pte, where before it could not.
But it would be wrong to tuck this change away inside a memcgroup patch.)

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

This patch could go anywhere in the mm series before the memory-controller
patches: I suggest just after swapin-fix-valid-swaphandles-defect.patch
Subsequent patches N/6 go in different places amongst the memory-controller
patches: please see accompanying suggestions.

 mm/swapfile.c | 38 ++++++++++++++++++++++++++++++++------
 1 file changed, 31 insertions(+), 7 deletions(-)

```
--- patch0/mm/swapfile.c 2007-11-07 19:41:45.000000000 +0000
+++ patch1/mm/swapfile.c 2007-11-08 12:34:12.000000000 +0000
@@ -506,9 +506,19 @@ 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, pmd_t *pmd,
   unsigned long addr, swp_entry_t entry, struct page *page)
 {
+ spinlock_t *ptl;
+ pte_t *pte;
+ int found = 1;
+
+ pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
+ if (unlikely(!pte_same(*pte, swp_entry_to_pte(entry)))) {
+  found = 0;
+  goto out;
+ }
+
  inc_mm_counter(vma->vm_mm, anon_rss);
  get_page(page);
  set_pte_at(vma->vm_mm, addr, pte,
@@ -520,6 +530,9 @@ static void unuse_pte(struct vm_area_str
   * immediately swapped out again after swapon.
```

```
   */
  activate_page(page);
+out:
+ pte_unmap_unlock(pte, ptl);
+ return found;
 }

 static int unuse_pte_range(struct vm_area_struct *vma, pmd_t *pmd,
@@ -528,22 +541,33 @@ static int unuse_pte_range(struct vm_are
 {
  pte_t swp_pte = swp_entry_to_pte(entry);
  pte_t *pte;
- spinlock_t *ptl;
  int found = 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))) {
-   unuse_pte(vma, pte++, addr, entry, page);
-   found = 1;
-   break;
+   pte_unmap(pte);
+   found = unuse_pte(vma, pmd, addr, entry, page);
+   if (found)
+    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 found;
 }
```

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