

[PATCH] Add notification of page becoming writable to VMA ops [try #3]

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-11/9716.html>

From: David Howells (dhowells_at_redhat.com)

Date: 11/30/05

To: torvalds@osdl.org, Andrew Morton <akpm@osdl.org>

Date: Wed, 30 Nov 2005 13:58:36 +0000

The attached patch adds a new VMA operation to notify a filesystem or other driver about the MMU generating a fault because userspace attempted to write to a page mapped through a read-only PTE.

This facility permits the filesystem or driver to:

- (*) Implement storage allocation/reservation on attempted write, and so to deal with problems such as ENOSPC more gracefully (perhaps by generating SIGBUS).
- (*) Delay making the page writable until the contents have been written to a backing cache. This is useful for NFS/AFS when using FS-Cache/CacheFS. It permits the filesystem to have some guarantee about the state of the cache.

Updated to 2.6.14-git14.

Signed-Off-By: David Howells <dhowells@redhat.com>

```
warthog>diffstat -p1 page-mkwrite-2614git14.diff
```

```
include/linux/mm.h |      4 ++
mm/memory.c        |     95 ++++++-----
mm/mmap.c          |     12 +++++-
mm/mprotect.c      |     11 +++++-
```

```
4 files changed, 94 insertions(+), 28 deletions(-)
```

```
diff -uNrp linux-2.6.14-git14/include/linux/mm.h linux-2.6.14-git14-pagenotify/include/linux/mm.h
```

```
--- linux-2.6.14-git14/include/linux/mm.h      2005-11-30 13:01:27.000000000 +0000
```

```
+++ linux-2.6.14-git14-pagenotify/include/linux/mm.h  2005-11-30 13:02:48.000000000 +0000
```

```
@@ -196,6 +196,10 @@ struct vm_operations_struct {
```

```
    void (*close)(struct vm_area_struct * area);
```

```
    struct page * (*nopcode)(struct vm_area_struct * area, unsigned long address, int *type);
```

```
    int (*populate)(struct vm_area_struct * area, unsigned long address, unsigned long len, p
```

```
+
```

```
+/ * notification that a previously read-only page is about to become
```

```
+/ * writable, if an error is returned it will cause a SIGBUS */
```

```
+/ int (*page_mkwrite)(struct vm_area_struct *vma, struct page *page);
```

```
+#ifdef CONFIG_NUMA
```

Linux-Kernel: [PATCH] Add notification of page becoming writable to VMA ops [try #3]

```
int (*set_policy)(struct vm_area_struct *vma, struct mempolicy *new);
struct mempolicy *(*get_policy)(struct vm_area_struct *vma,
diff -uNrp linux-2.6.14-git14/mm/memory.c linux-2.6.14-git14-pagenotify/mm/memory.c
--- linux-2.6.14-git14/mm/memory.c      2005-11-30 13:01:29.000000000 +0000
+++ linux-2.6.14-git14-pagenotify/mm/memory.c  2005-11-30 13:15:50.000000000 +0000
@@ -1253,7 +1253,7 @@ static int do_wp_page(struct mm_struct *
    struct page *old_page, *new_page;
    unsigned long pfn = pte_pfn(orig_pte);
    pte_t entry;
-   int ret = VM_FAULT_MINOR;
+   int reuse, ret = VM_FAULT_MINOR;

    BUG_ON(vma->vm_flags & VM_RESERVED);

@@ -1267,19 +1267,49 @@ static int do_wp_page(struct mm_struct *
    }
    old_page = pfn_to_page(pfn);

-   if (PageAnon(old_page) && !TestSetPageLocked(old_page)) {
-       int reuse = can_share_swap_page(old_page);
-       unlock_page(old_page);
-       if (reuse) {
-           flush_cache_page(vma, address, pfn);
-           entry = pte_mkyoung(orig_pte);
-           entry = maybe_mkwrite(pte_mkdirty(entry), vma);
-           ptep_set_access_flags(vma, address, page_table, entry, 1);
-           update_mmu_cache(vma, address, entry);
-           lazy_mmu_prot_update(entry);
-           ret |= VM_FAULT_WRITE;
-           goto unlock;
+   if (unlikely(vma->vm_flags & VM_SHARED)) {
+       if (vma->vm_ops && vma->vm_ops->page_mkwrite) {
+           /*
+            * Notify the page owner without the lock held,
+            * so they can sleep if they want to.
+            */
+           page_cache_get(old_page);
+           pte_unmap_unlock(page_table, ptl);
+
+           if (vma->vm_ops->page_mkwrite(vma, old_page) < 0)
+               goto unwritable_page;
+
+           page_cache_release(old_page);
+
+           /*
+            * Since we dropped the lock we need to revalidate
+            * the PTE as someone else may have changed it.  If
+            * they did, we just return, as we can count on the
+            * MMU to tell us if they didn't also make it writable.
+            */
+           page_table = pte_offset_map_lock(mm, pmd, address,
+                                           &ptl);
+           if (!pte_same(*page_table, orig_pte))
+               goto unlock;
+       }
+
+       reuse = 1;
+   } else if (PageAnon(old_page) && !TestSetPageLocked(old_page)) {
+       reuse = can_share_swap_page(old_page);
+       unlock_page(old_page);
+   } else {
+       reuse = 0;
```

Linux-Kernel: [PATCH] Add notification of page becoming writable to VMA ops [try #3]

```

+     }
+
+     if (reuse) {
+         flush_cache_page(vma, address, pfn);
+         entry = pte_mkyoung(orig_pte);
+         entry = maybe_mkwrite(pte_mkdirty(entry), vma);
+         ptep_set_access_flags(vma, address, page_table, entry, 1);
+         update_mmu_cache(vma, address, entry);
+         lazy_mmu_prot_update(entry);
+         ret |= VM_FAULT_WRITE;
+         goto unlock;
+     }
+
+     /*
@@ -1332,6 +1362,10 @@ unlock:
    oom:
        page_cache_release(old_page);
        return VM_FAULT_OOM;
+
+unwritable_page:
+    page_cache_release(old_page);
+    return VM_FAULT_SIGBUS;
    }
+
+    /*
@@ -1853,18 +1887,31 @@ retry:
    /*
    * Should we do an early C-O-W break?
    */
-    if (write_access && !(vma->vm_flags & VM_SHARED)) {
-        struct page *page;
+    if (write_access) {
+        if (!(vma->vm_flags & VM_SHARED)) {
+            struct page *page;
+
+            if (unlikely(anon_vma_prepare(vma)))
+                goto oom;
+            page = alloc_page_vma(GFP_HIGHUSER, vma, address);
+            if (!page)
+                goto oom;
+            copy_user_highpage(page, new_page, address);
+            page_cache_release(new_page);
+            new_page = page;
+            anon = 1;
+
+            if (unlikely(anon_vma_prepare(vma)))
+                goto oom;
+            page = alloc_page_vma(GFP_HIGHUSER, vma, address);
+            if (!page)
+                goto oom;
+            copy_user_highpage(page, new_page, address);
+            page_cache_release(new_page);
+            new_page = page;
+            anon = 1;
+
+        } else {
+            /* if the page will be shareable, see if the backing
+            * address space wants to know that the page is about
+            * to become writable */
+            if (vma->vm_ops->page_mkwrite &&
+                vma->vm_ops->page_mkwrite(vma, new_page) < 0
+            ) {
+                page_cache_release(new_page);

```


Linux-Kernel: [PATCH] Add notification of page becoming writable to VMA ops [try #3]

```
@@ -168,6 +167,14 @@ mprotect_fixup(struct vm_area_struct *vm
    }
```

success:

```
+     /* Don't make the VMA automatically writable if it's shared, but the
+      * backer wishes to know when pages are first written to */
+     mask = VM_READ|VM_WRITE|VM_EXEC|VM_SHARED;
+     if (vma->vm_ops && vma->vm_ops->page_mkwrite)
+         mask &= ~VM_SHARED;
+
+     newprot = protection_map[newflags & mask];
+
+     /*
+      * vm_flags and vm_page_prot are protected by the mmap_sem
+      * held in write mode.
```

-
To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at <http://vger.kernel.org/majordomo-info.html>
Please read the FAQ at <http://www.tux.org/lkml/>