

Re: hugepage: Strict page reservation for hugepage inodes

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2006-02/msg09875.html>

- *From:* David Gibson <david@xxxxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Tue, 28 Feb 2006 20:14:40 +1100
-

On Tue, Feb 28, 2006 at 04:53:49PM +0800, Zhang, Yanmin wrote:
[snip]

```
+ if (vma->vm_flags & VM_MAYSHARE) {
+
+ /* idx = radix tree index, i.e. offset into file in
+ * HPAGE_SIZE units */
+ idx = ((addr - vma->vm_start) >> HPAGE_SHIFT)
+ + (vma->vm_pgoff >> (HPAGE_SHIFT -
+ PAGE_SHIFT));
+
+ /* The hugetlbf's specific inode info stores the number
+ * of "guaranteed available" (huge) pages. That is,
+ * the first 'prereserved_hpages' pages of the inode
+ * are either already instantiated, or have been
+ * pre-reserved (by hugetlb_reserve_for_inode()). Here
+ * we're in the process of instantiating the page, so
+ * we use this to determine whether to draw from the
+ * pre-reserved pool or the truly free pool. */
+ if (idx < HUGETLBFS_I(inode)->prereserved_hpages)
+ use_reserve = 1;
+ }
+
+ if (!use_reserve) {
+ if (free_huge_pages <= reserved_huge_pages)
+ goto fail;
+ } else {
+ BUG_ON(reserved_huge_pages == 0);
+ reserved_huge_pages--;
```

[YM] Consider this scenario of multi-thread:

One process has 2 threads. The process mmmaps a hugetlb area with 1 huge page and there is a free huge page. Later on, the 2 threads fault on the huge page at the same time. The second thread would fail, and WARN_ON check is triggered, then the second thread is killed by function hugetlb_no_page.

Re: hugepage: Strict page reservation for hugepage inodes

That's why this patch **must** go after my other patch which serializes the allocation→instantiation path.

```
    }
+
+ page = dequeue_huge_page(vma, addr);
+ if (!page)
+ goto fail;
+
spin_unlock(&hugetlb_lock);
set_page_count(page, 1);
return page;
+
+ fail:
+ WARN_ON(use_reserve); /* reserved allocations shouldn't
fail */
+ spin_unlock(&hugetlb_lock);
+ return NULL;
+}
+
+/* hugetlb_extend_reservation()
+ *
+ * Ensure that at least 'atleast' hugepages are, and will
remain,
+ * available to instantiate the first 'atleast' pages of the given
+ * inode. If the inode doesn't already have this many pages
reserved
+ * or instantiated, set aside some hugepages in the reserved
pool to
+ * satisfy later faults (or fail now if there aren't enough,
rather
+ * than getting the SIGBUS later).
+ */
+int hugetlb_extend_reservation(struct hugetlbf_inode_info
*info,
+ unsigned long atleast)
+{
+ struct inode *inode = &info->vfs_inode;
+ struct address_space *mapping = inode->i_mapping;
+ unsigned long idx;
+ unsigned long change_in_reserve = 0;
+ struct page *page;
+ int ret = 0;
+
+ spin_lock(&hugetlb_lock);
+ read_lock_irq(&inode->i_mapping->tree_lock);
+
+ if (info->prereserved_hpages >= atleast)
+ goto out;
```

Re: hugepage: Strict page reservation for hugepage inodes

```
+
+ /* prereserved_hpages stores the number of pages already
+ * guaranteed (reserved or instantiated) for this inode.
+ * Count how many extra pages we need to reserve. */
+ for (idx = info->prereserved_hpages; idx < atleast; idx++)
+ {
+ page = radix_tree_lookup(&mapping->page_tree, idx);
+ if (!page)
+ /* Pages which are already instantiated don't
+ * need to be reserved */
+ change_in_reserve++;
+ }
```

[YM] Why always to go through the page cache? prereserved_hpages and reserved_huge_pages are protected by hugetlb_lock.

Erm.. sorry, I don't see how that helps. We need to go through the page cache to see which pages have already been instantiated, and so don't need to be reserved.

--

David Gibson | I'll have my music baroque, and my code
david AT gibson.dropbear.id.au | minimalist, thank you. NOT _the__other_
| _way__around_!
<http://www.ozlabs.org/~dgibson>

-

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