author	James Houghton <jthoughton@google.com>	2022-10-18 20:01:25 +0000
committer	Andrew Morton <akpm@linux-foundation.org>	2022-11-08 15:57:22 -0800
commit	8625147cafaa9ba74713d682f5185eb62cb2aedb (patch)	
tree	185342d7d161810419b46382581893ad93e46a22	
parent	120b116208a0877227fc82e3f0df81e7a3ed4ab1 (diff)	
download	linux-8625147cafaa9ba74713d682f5185eb62cb2aedb.tar.gz	

hugetlbfs: don't delete error page from pagecache

```
This change is very similar to the change that was made for shmem [1], and
it solves the same problem but for HugeTLBFS instead.

Currently, when poison is found in a HugeTLB page, the page is removed
from the page cache.  That means that attempting to map or read that
hugepage in the future will result in a new hugepage being allocated
instead of notifying the user that the page was poisoned.  As [1] states,
this is effectively memory corruption.

The fix is to leave the page in the page cache.  If the user attempts to
use a poisoned HugeTLB page with a syscall, the syscall will fail with
EIO, the same error code that shmem uses.  For attempts to map the page,
the thread will get a BUS_MCEERR_AR SIGBUS.

[1]: commit a76054266661 ("mm: shmem: don't truncate page if memory failure happens")

Link: https://lkml.kernel.org/r/20221018200125.848471-1-jthoughton@google.com
Signed-off-by: James Houghton <jthoughton@google.com>
Reviewed-by: Mike Kravetz <mike.kravetz@oracle.com>
Reviewed-by: Naoya Horiguchi <naoya.horiguchi@nec.com>
Tested-by: Naoya Horiguchi <naoya.horiguchi@nec.com>
Reviewed-by: Yang Shi <shy828301@gmail.com>
Cc: Axel Rasmussen <axelrasmussen@google.com>
Cc: James Houghton <jthoughton@google.com>
Cc: Miaohe Lin <linmiaohe@huawei.com>
Cc: Muchun Song <songmuchun@bytedance.com>
Cc: <stable@vger.kernel.org>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
```

Diffstat

-rw-r--r--	fs/hugetlbfs/inode.c	13
-rw-r--r--	mm/hugetlb.c	4
-rw-r--r--	mm/memory-failure.c	5

3 files changed, 14 insertions, 8 deletions

```
diff --git a/fs/hugetlbfs/inode.c b/fs/hugetlbfs/inode.c
index dd54f67e47fdf1..df7772335dc0e4 100644
--- a/fs/hugetlbfs/inode.c
+++ b/fs/hugetlbfs/inode.c
@@ -328,6 +328,12 @@ static ssize_t hugetlbfs_read_iter(struct kiocb *iocb, struct iov_iter *to)
                } else {
```

```
                        unlock_page(page);

+                       if (PageHWPoison(page)) {
+                               put_page(page);
+                               retval = -EIO;
+                               break;
+                       }
+
                        /*
                         * We have the page, copy it to user space buffer.
                         */
@@ -1111,13 +1117,6 @@ static int hugetlbfs_migrate_folio(struct address_space *mapping,
 static int hugetlbfs_error_remove_page(struct address_space *mapping,
                                struct page *page)
 {
-       struct inode *inode = mapping->host;
-       pgoff_t index = page->index;
-
-       hugetlb_delete_from_page_cache(page);
-       if (unlikely(hugetlb_unreserve_pages(inode, index, index + 1, 1)))
-               hugetlb_fix_reserve_counts(inode);
-
        return 0;
 }


diff --git a/mm/hugetlb.c b/mm/hugetlb.c
index 546df97c31e4cb..e48f8ef45b17ff 100644
--- a/mm/hugetlb.c
+++ b/mm/hugetlb.c
@@ -6111,6 +6111,10 @@ int hugetlb_mcopy_atomic_pte(struct mm_struct *dst_mm,

        ptl = huge_pte_lock(h, dst_mm, dst_pte);

+       ret = -EIO;
+       if (PageHWPoison(page))
+               goto out_release_unlock;
+
        /*
         * We allow to overwrite a pte marker: consider when both MISSING|WP
         * registered, we firstly wr-protect a none pte which has no page cache

diff --git a/mm/memory-failure.c b/mm/memory-failure.c
index 145bb561ddb3ad..bead6bccc7f28e 100644
--- a/mm/memory-failure.c
+++ b/mm/memory-failure.c
@@ -1080,6 +1080,7 @@ static int me_huge_page(struct page_state *ps, struct page *p)
        int res;
        struct page *hpage = compound_head(p);
        struct address_space *mapping;
+       bool extra_pins = false;

        if (!PageHuge(hpage))
                return MF_DELAYED;
@@ -1087,6 +1088,8 @@ static int me_huge_page(struct page_state *ps, struct page *p)
        mapping = page_mapping(hpage);
        if (mapping) {
                res = truncate_error_page(hpage, page_to_pfn(p), mapping);
+               /* The page is kept in page cache. */
+               extra_pins = true;
                unlock_page(hpage);
        } else {
```

```
                unlock_page(hpage);
@@ -1104,7 +1107,7 @@ static int me_huge_page(struct page_state *ps, struct page *p)
                }
        }

-       if (has_extra_refcount(ps, p, false))
+       if (has_extra_refcount(ps, p, extra_pins))
                res = MF_FAILED;

        return res;
```