Object-based Reverse Mapping

Dave McCracken
IBM
dmccr@us.ibm.com

Abstract

Physical to virtual translation of user addresses (reverse mapping) has long been sought after to improve the pageout algorithms of the VM. An implementation was added to 2.6 that uses back pointers from each page to its mapping (pte chains). While pte chains do work, they add significant spaceoverhead and significant time overhead during page mapping/unmapping and fork/exit.

I will describe an alternative method of reverse mapping based on the object each page belongs to. I will discuss the partial implementation I did last year as well as the work done by Hugh Dickins and Andrea Arcangelli to complete it. I will describe the current implementations, their relative strengths and weaknesses, and what plans if any there are for solutions to the remaining issues.

1 Introduction

Up through version 2.4, the Linux® kernel had no mechanism for translating physical addresses to user virtual addresses, commonly called reverse mapping, or rmap. This meant it was not possible for the memory management subsystem to point to a physical page and remove all its mappings. There was a mechanism that walked through each process's mappings and selected pages to unmap. Only after all a page's mappings were removed could it be selected for pageout.

Many in the memory management community considered this very inefficient. Page aging and removal could be made much more efficient if the page could be directly unmapped when it was ready to be removed. Some form of rmap was clearly needed for this to work.

2 PTE Chains

Rik van Riel implemented an rmap mechanism that added a chain of pointers to each page back to all its mappings, commonly called `pte_chains`. It works by adding a linked list to the control structure for each physical page (struct page) which points to all the page table entries that map that page. His code was accepted into mainline early in the 2.5 development cycle.

Once this rmap implementation was in place the page aging and removal algorithm was changed to use it, streamlining the code and allowing better tuning.

One negative to the `pte_chain` implementation was a significant performance cost to `fork`, `exec`, and `exit`. The cost to these functions was related to the amount of memory mapped to the process, but was close to an order of magnitude worse.

A second cost was space. In its original form `pte_chains` cost two pointers per mapping. An optimization eliminated the extra structure for singly-mapped pages and another optimiza-

tion added multiple pointers per list entry, but the space taken by the `pte_chain` structures was still significant.

3 A Brief History of Object-based Rmap

Processes do not really map memory one page at a time. They map a range of data from an offset within some object (usually a file) to a range of addresses. The virtual addresses of all pages within that range can be calculated from their offset in that object and the base mapping address of the range.

The kernel has the information to do object-based reverse mapping for files. Each `struct page` for a file has an offset and a pointer to a `struct address_space`, which is the base anchor for all memory associated with a file. Every time a range of data from that file is mapped to a process, a `vm_area_struct` or `vma` is created. The `vma` contains the virtual address of the mapping and the base offset within the file. It is then added to a linked list of all `vma`s in the `address_space` for that file.

The remaining problem in the kernel is anonymous memory. Blocks of anonymous memory have `vma`s but these `vma`s are not connected to any common object that can be used for reverse mapping.

3.1 Partial Object-based Rmap

Given this information, last year I did a sample implementation of object-based rmap for files, but left the `pte_chain` implementation in place for anonymous memory. It works by following the pointer in the `struct page` to the `struct address_space`, then walking the linked list of `vma`s to find all that contain the page. A simple calculation then determines the virtual address of that page and a page table walk finds the page table entry.

This implementation recovers the performance of `fork`, `exec`, and `exit` and eliminates the space penalty used by `pte_chain` structures. It introduces a performance penalty when it walks the linked list of `vma`s, but this is incurred by the page aging code instead of the application code. It could still be significant, however, since it rises linearly with the number of times any part of the file is mapped while with `pte_chains` the cost rises linearly with the number of times that page is mapped.

3.2 First Cut at Full Object-based Rmap

Hugh Dickins took my implemenation and extended it to handle anonymous mappings, eliminating `pte_chains` entirely. He did this by creating an `anonmm` object for each process that all anonymous pages belong to. All `anonmm` structures are linked together by `fork`. A new `anonmm` structure is allocated on `exec`. The offset stored in `struct page` is the virtual address of the page, while the object pointer points to an `anonmm` that the page is mapped in.

Finding all mappings of a page is simple. The pointer in `struct page` is followed to the `anonmm` chain, which is then walked looking for mappings of that page at the virtual address specified in the offset.

Hugh's initial patch ignored the problem of shared anonymous pages that were remapped by an `mremap` call. The problem with `mremap` is that it allows an anonymous page to be at different virtual addresses in different processes, but there is only one offset for the page.

After some initial discussion among the community, both Hugh and I moved on to other things.

3.3 A Second Cut at Full Object-based Rmap

In February of this year Andrea Arcangeli began to investigate what could be done about the problems of `pte_chains`. He took my partial object-based rmap patch and implemented his own solution for anonymous memory, called `anon_vma`.

The basic mechanism of `anon_vma` is the addition of an `anon_vma` structure linked to each `vma` that has anonymous pages. The `anon_vma` structure has a linked list of all `vma`s that map that anonymous range. The pointer in `struct page` points to the `anon_vma` and the index is the offset into the current mapping.

An advantage of Andrea's `anon_vma` structure is that it solves the mremap problem that the `anonmm` structure did not. Since the offset stored in each page is relative to the base of the `vma` that maps it, the region can be remapped without changing the offset. However, since `vma`s can be merged, it is not an an absolutely painless solution.

4 Advancements All Around

In response to Andrea's patch, Hugh resumed work on his `anonmm` patch. Prompted by a discussion among the community and an approach suggested by Linus, Hugh implemented a simple scheme for handling the remap case. For each page, if there is only one reference, that page can simply have its offset changed. If the page is shared, a copy is forced and the new unshared page is mapped at the new address. Since all anonymous pages are already copy-on-write, it is likely that the page would be written to eventually and the copy taken. It is possible that some read-only pages might be duplicated, but to date there is no evidence that any code actually remaps shared read-only anonymous pages.

5 The `vma` List Problem

All these implementations still include the original implementation for file pages, including the need to walk the linked list of `vma`s attached to the `address_space` structure. This has been identified as a possible performance issue for massively mapped files, though few if any real-life examples have been found. A few optimizations have been tried, including sorting the list by start address and making a two level list based on start and end address. Both these solutions share the problem that adding or modifying a `vma` is fairly expensive and holds the associated lock for a long time.

A recent contribution by Rajesh Venkatasubramanian is the use of a `prio_tree`, which is similar to a radix tree but supports sorting objects by both start and end addresses. It adds some complexity to the `vma` list but greatly reduces the potential performance impact of a large number of mappings.

6 The `remap_file_pages` Problem

While object-based rmap appears relatively simple, there is one new feature that greatly complicates the problem. This feature is `remap_file_pages`.

The `remap_file_pages` system call was introduced during the 2.5 development cycle. It works on a range of shared memory mapped from a file, and allows an application to change the memory range to map a different offset within that file. This is done without modifying the `vma` describing the mapping. This means the offsets specified within the `vma` are now

wrong. Since the `address_space` pointer and offset within the `page` structure are intact, the page can still be mapped back to its place in the file, but it is no longer possible to use this information to find its virtual mappings. The `vma` is called a `nonlinear vma` and is put on a special list within the `address_space`.

Andrea and Hugh have provided two different solutions to the problem of what to do when a nonlinear page is called to be unmapped. Andrea's solution is the more draconian in that it walks the list of nonlinear `vma`s and unmaps all pages in them until the page in question has no more mappings. Hugh's solution only unmaps a fixed number of nonlinear pages and makes no attempt to unmap the actual page passed in.

7 Release Status

As of the date this was written, Hugh has been submitting incremental rmap changes to Andrew Morton for the -mm tree over the past couple of months. The early submissions were primarily cleanup, but later patches included first my partial object-based rmap implementation followed by his `anonmm` implementation, which completely removed the `pte_chain` code.

Hugh has just submitted a final set of patches to Andrew that removes his `anonmm` implementation and replaces it with Andrea's `anon_vma` implementation.

The general expectation among the VM developer community is that once this code has been adequately tested in the -mm tree that it will replace the existing `pte_chain` implementation in mainline 2.6.

Legal Statement

This paper represents the views of the author and does not necessarily represent the view of IBM.

Proceedings of the Linux Symposium

Volume Two

July 21st–24th, 2004
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
Stephanie Donovan, *Linux Symposium*
C. Craig Ross, *Linux Symposium*

Review Committee

Jes Sorensen, *Wild Open Source, Inc.*
Matt Domsch, *Dell*
Gerrit Huizenga, *IBM*
Matthew Wilcox, *Hewlett-Packard*
Dirk Hohndel, *Intel*
Val Henson, *Sun Microsystems*
Jamal Hadi Salimi, *Znyx*
Andrew Hutton, *Steamballoon, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*