
Subject: [PATCH 11/13] BC: physpages accounting (hooks)

Posted by [dev](#) on Thu, 09 Nov 2006 17:01:28 GMT

[View Forum Message](#) <> [Reply to Message](#)

Introduce calls to BC code over the kernel to add
accounting of physical pages.

Signed-Off-By: Pavel Emelianov <xemul@sw.ru>

Signed-Off-By: Kirill Korotaev <dev@sw.ru>

```
fs/exec.c      | 15 +++-
include/linux/rmap.h | 9 +-
mm/fremap.c    | 11 ++
mm/memory.c    | 52 ++++++++--
mm/migrate.c   | 13 +-
mm/rmap.c      | 34 ++++++--
mm/swapfile.c  | 15 +++-
mm/vmscan.c    | 190 ++++++++
8 files changed, 310 insertions(+), 29 deletions(-)
```

--- ./fs/exec.c.ve10 2006-11-07 11:56:20.000000000 +0300

+++ ./fs/exec.c 2006-11-07 11:57:23.000000000 +0300

@@ -51,6 +51,8 @@

#include <linux/cn_proc.h>

#include <linux/audit.h>

+#include <bc/rsspages.h>

+

#include <asm/uaccess.h>

#include <asm/mmu_context.h>

@@ -309,27 +311,34 @@ void install_arg_page(struct vm_area_str

struct mm_struct *mm = vma->vm_mm;

pte_t *pte;

spinlock_t *ptl;

+ struct page_beancounter *pb;

if (unlikely(anon_vma_prepare(vma)))

goto out;

+ if (bc_rsspage_prepare(page, vma, &pb))

+ goto out;

+

flush_dcache_page(page);

pte = get_locked_pte(mm, address, &ptl);

if (!pte)

```

- goto out;
+ goto out_unch;
  if (!pte_none(*pte)) {
    pte_unmap_unlock(pte, ptl);
- goto out;
+ goto out_unch;
  }
  inc_mm_counter(mm, anon_rss);
  lru_cache_add_active(page);
  set_pte_at(mm, address, pte, pte_mkdirty(pte_mkwwrite(mk_pte(
    page, vma->vm_page_prot))));
- page_add_new_anon_rmap(page, vma, address);
+ page_add_new_anon_rmap(page, vma, address, pb);
  pte_unmap_unlock(pte, ptl);

  /* no need for flush_tlb */
  return;
+
+out_unch:
+ bc_rsspage_release(pb);
out:
  __free_page(page);
  force_sig(SIGKILL, current);
--- ./include/linux/rmap.h.ve10 2006-11-07 11:56:20.000000000 +0300
+++ ./include/linux/rmap.h 2006-11-07 11:57:23.000000000 +0300
@@ -69,9 +69,12 @@ void __anon_vma_link(struct vm_area_stru
/*
 * rmap interfaces called when adding or removing pte of page
 */
-void page_add_anon_rmap(struct page *, struct vm_area_struct *, unsigned long);
-void page_add_new_anon_rmap(struct page *, struct vm_area_struct *, unsigned long);
-void page_add_file_rmap(struct page *);
+struct page_beancounter;
+void page_add_anon_rmap(struct page *, struct vm_area_struct *, unsigned long,
+ struct page_beancounter *);
+void page_add_new_anon_rmap(struct page *, struct vm_area_struct *,
+ unsigned long, struct page_beancounter *);
+void page_add_file_rmap(struct page *, struct page_beancounter *);
void page_remove_rmap(struct page *);

/**
--- ./mm/fremap.c.ve10 2006-11-07 11:56:20.000000000 +0300
+++ ./mm/fremap.c 2006-11-07 11:57:23.000000000 +0300
@@ -16,6 +16,8 @@
#include <linux/module.h>
#include <linux/syscalls.h>

+#include <bc/rsspages.h>

```

```

+
#include <asm/mmu_context.h>
#include <asm/cacheflush.h>
#include <asm/tlbflush.h>
@@ -57,6 +59,10 @@ int install_page(struct mm_struct *mm, s
    pte_t *pte;
    pte_t pte_val;
    spinlock_t *ptl;
+ struct page_beancounter *pb;
+
+ if (bc_rsspage_prepare(page, vma, &pb))
+ goto out_nocharge;

    pte = get_locked_pte(mm, addr, &ptl);
    if (!pte)
@@ -81,13 +87,16 @@ int install_page(struct mm_struct *mm, s
    flush_icache_page(vma, page);
    pte_val = mk_pte(page, prot);
    set_pte_at(mm, addr, pte, pte_val);
- page_add_file_rmap(page);
+ page_add_file_rmap(page, pb);
    update_mmu_cache(vma, addr, pte_val);
    lazy_mmu_prot_update(pte_val);
    err = 0;
unlock:
    pte_unmap_unlock(pte, ptl);
out:
+ if (err != 0)
+ bc_rsspage_release(pb);
+out_nocharge:
    return err;
}
EXPORT_SYMBOL(install_page);
--- ./mm/memory.c.ve10 2006-11-07 11:56:20.000000000 +0300
+++ ./mm/memory.c 2006-11-07 11:57:23.000000000 +0300
@@ -57,6 +57,8 @@
#include <asm/tlbflush.h>
#include <asm/pgtable.h>

+#include <bc/rsspapes.h>
+
#include <linux/swapops.h>
#include <linux/elf.h>

@@ -1119,7 +1121,7 @@ static int zeromap_pte_range(struct mm_s
    struct page *page = ZERO_PAGE(addr);
    pte_t zero_pte = pte_wrprotect(mk_pte(page, prot));
    page_cache_get(page);

```

```

- page_add_file_rmap(page);
+ page_add_file_rmap(page, NULL);
  inc_mm_counter(mm, file_rss);
  BUG_ON(!pte_none(*pte));
  set_pte_at(mm, addr, pte, zero_pte);
@@ -1224,7 +1226,7 @@ static int insert_page(struct mm_struct
/* Ok, finally just insert the thing.. */
get_page(page);
inc_mm_counter(mm, file_rss);
- page_add_file_rmap(page);
+ page_add_file_rmap(page, NULL);
  set_pte_at(mm, addr, pte, mk_pte(page, prot));

  retval = 0;
@@ -1485,6 +1487,7 @@ static int do_wp_page(struct mm_struct *
  pte_t entry;
  int reuse = 0, ret = VM_FAULT_MINOR;
  struct page *dirty_page = NULL;
+ struct page_becounter *pb;

  old_page = vm_normal_page(vma, address, orig_pte);
  if (!old_page)
@@ -1570,6 +1573,9 @@ gotten:
  cow_user_page(new_page, old_page, address);
}

+ if (bc_rsspage_prepare(new_page, vma, &pb))
+ goto oom;
+
/*
 * Re-check the pte - we dropped the lock
 */
@@ -1597,12 +1603,14 @@ gotten:
  set_pte_at(mm, address, page_table, entry);
  update_mmu_cache(vma, address, entry);
  lru_cache_add_active(new_page);
- page_add_new_anon_rmap(new_page, vma, address);
+ page_add_new_anon_rmap(new_page, vma, address, pb);

  /* Free the old page.. */
  new_page = old_page;
  ret |= VM_FAULT_WRITE;
- }
+ } else
+ bc_rsspage_release(pb);
+
  if (new_page)
    page_cache_release(new_page);

```

```

    if (old_page)
@@ -1979,6 +1987,7 @@ static int do_swap_page(struct mm_struct
    swp_entry_t entry;
    pte_t pte;
    int ret = VM_FAULT_MINOR;
+ struct page_becounter *pb;

    if (!pte_unmap_same(mm, pmd, page_table, orig_pte))
        goto out;
@@ -2011,6 +2020,11 @@ static int do_swap_page(struct mm_struct
    count_vm_event(PGMAJFAULT);
}

+ if (bc_rsspage_prepare(page, vma, &pb)) {
+     ret = VM_FAULT_OOM;
+     goto out;
+ }
+
    delayacct_clear_flag(DELAYACCT_PF_SWAPIN);
    mark_page_accessed(page);
    lock_page(page);
@@ -2024,6 +2038,7 @@ static int do_swap_page(struct mm_struct

    if (unlikely(!PageUptodate(page))) {
        ret = VM_FAULT_SIGBUS;
+     bc_rsspage_release(pb);
        goto out_nomap;
    }

@@ -2038,7 +2053,7 @@ static int do_swap_page(struct mm_struct

    flush_icache_page(vma, page);
    set_pte_at(mm, address, page_table, pte);
- page_add_anon_rmap(page, vma, address);
+ page_add_anon_rmap(page, vma, address, pb);

    swap_free(entry);
    if (vm_swap_full())
@@ -2060,6 +2075,7 @@ unlock:
out:
    return ret;
out_nomap:
+ bc_rsspage_release(pb);
    pte_unmap_unlock(page_table, ptl);
    unlock_page(page);
    page_cache_release(page);
@@ -2078,6 +2094,7 @@ static int do_anonymous_page(struct mm_s
    struct page *page;

```

```

spinlock_t *ptl;
pte_t entry;
+ struct page_beancounter *pb;

if (write_access) {
    /* Allocate our own private page. */
@@ -2089,15 +2106,19 @@ static int do_anonymous_page(struct mm_s
    if (!page)
        goto oom;

+ if (bc_rsspage_prepare(page, vma, &pb))
+ goto oom_release;
+
    entry = mk_pte(page, vma->vm_page_prot);
    entry = maybe_mkdirty(pte_mkdirty(entry), vma);

    page_table = pte_offset_map_lock(mm, pmd, address, &ptl);
    if (!pte_none(*page_table))
- goto release;
+ goto release_pc;
+
    inc_mm_counter(mm, anon_rss);
    lru_cache_add_active(page);
- page_add_new_anon_rmap(page, vma, address);
+ page_add_new_anon_rmap(page, vma, address, pb);
} else {
    /* Map the ZERO_PAGE - vm_page_prot is readonly */
    page = ZERO_PAGE(address);
@@ -2109,7 +2130,7 @@ static int do_anonymous_page(struct mm_s
    if (!pte_none(*page_table))
        goto release;
    inc_mm_counter(mm, file_rss);
- page_add_file_rmap(page);
+ page_add_file_rmap(page, NULL);
}

    set_pte_at(mm, address, page_table, entry);
@@ -2120,9 +2141,14 @@ static int do_anonymous_page(struct mm_s
unlock:
    pte_unmap_unlock(page_table, ptl);
    return VM_FAULT_MINOR;
+release_pc:
+ bc_rsspage_release(pb);
release:
    page_cache_release(page);
    goto unlock;
+
+oom_release:

```

```

+ page_cache_release(page);
oom:
    return VM_FAULT_OOM;
}
@@ -2152,6 +2178,7 @@ static int do_no_page(struct mm_struct *
    int ret = VM_FAULT_MINOR;
    int anon = 0;
    struct page *dirty_page = NULL;
+ struct page_beancounter *pb;

    pte_unmap(page_table);
    BUG_ON(vma->vm_flags & VM_PFNMAP);
@@ -2209,6 +2236,9 @@ retry:
    }
}

+ if (bc_rsspage_prepare(new_page, vma, &pb))
+ goto oom;
+
    page_table = pte_offset_map_lock(mm, pmd, address, &ptl);
    /*
     * For a file-backed vma, someone could have truncated or otherwise
@@ -2217,6 +2247,7 @@ retry:
    */
    if (mapping && unlikely(sequence != mapping->truncate_count)) {
        pte_unmap_unlock(page_table, ptl);
+ bc_rsspage_release(pb);
        page_cache_release(new_page);
        cond_resched();
        sequence = mapping->truncate_count;
@@ -2244,10 +2275,10 @@ retry:
    if (anon) {
        inc_mm_counter(mm, anon_rss);
        lru_cache_add_active(new_page);
- page_add_new_anon_rmap(new_page, vma, address);
+ page_add_new_anon_rmap(new_page, vma, address, pb);
    } else {
        inc_mm_counter(mm, file_rss);
- page_add_file_rmap(new_page);
+ page_add_file_rmap(new_page, pb);
        if (write_access) {
            dirty_page = new_page;
            get_page(dirty_page);
@@ -2255,6 +2286,7 @@ retry:
    }
} else {
    /* One of our sibling threads was faster, back out. */
+ bc_rsspage_release(pb);

```

```

    page_cache_release(new_page);
    goto unlock;
}
--- ./mm/migrate.c.ve10 2006-11-07 11:56:20.000000000 +0300
+++ ./mm/migrate.c 2006-11-07 11:57:23.000000000 +0300
@@ -134,6 +134,7 @@ static void remove_migration_pte(struct
    pte_t *ptep, pte;
    spinlock_t *ptl;
    unsigned long addr = page_address_in_vma(new, vma);
+ struct page_beancounter *pb;

    if (addr == -EFAULT)
        return;
@@ -157,6 +158,11 @@ static void remove_migration_pte(struct
    return;
}

+ if (bc_rsspage_prepare(new, vma, &pb)) {
+     pte_unmap(ptep);
+     return;
+ }
+
    ptl = pte_lockptr(mm, pmd);
    spin_lock(ptl);
    pte = *ptep;
@@ -175,16 +181,19 @@ static void remove_migration_pte(struct
    set_pte_at(mm, addr, ptep, pte);

    if (PageAnon(new))
-     page_add_anon_rmap(new, vma, addr);
+     page_add_anon_rmap(new, vma, addr, pb);
    else
-     page_add_file_rmap(new);
+     page_add_file_rmap(new, pb);

    /* No need to invalidate - it was non-present before */
    update_mmu_cache(vma, addr, pte);
    lazy_mmu_prot_update(pte);
+ pte_unmap_unlock(ptep, ptl);
+ return;

out:
    pte_unmap_unlock(ptep, ptl);
+ bc_rsspage_release(pb);
}

/*
--- ./mm/rmap.c.ve10 2006-11-07 11:56:20.000000000 +0300

```



```

+++ ./mm/rmap.c 2006-11-07 11:57:51.000000000 +0300
@@ -48,6 +48,8 @@
#include <linux/rcupdate.h>
#include <linux/module.h>

+#include <bc/rsspages.h>
+
#include <asm/tlbflush.h>

struct kmem_cache *anon_vma_cache;
@@ -173,7 +175,8 @@ static void anon_vma_ctor(void *data, st
void __init anon_vma_init(void)
{
    anon_vma_cache = kmem_cache_create("anon_vma", sizeof(struct anon_vma),
- 0, SLAB_DESTROY_BY_RCU|SLAB_PANIC, anon_vma_ctor, NULL);
+ 0, SLAB_DESTROY_BY_RCU|SLAB_PANIC|SLAB_BC,
+ anon_vma_ctor, NULL);
}

/*
@@ -522,14 +525,19 @@ static void __page_set_anon_rmap(struct
 * @page: the page to add the mapping to
 * @vma: the vm area in which the mapping is added
 * @address: the user virtual address mapped
+ * @pb: the page beancounter to charge page with
 *
 * The caller needs to hold the pte lock.
 */
void page_add_anon_rmap(struct page *page,
- struct vm_area_struct *vma, unsigned long address)
+ struct vm_area_struct *vma, unsigned long address,
+ struct page_beancounter *pb)
{
- if (atomic_inc_and_test(&page->_mapcount))
+ if (atomic_inc_and_test(&page->_mapcount)) {
+ bc_rsspage_charge(pb);
    __page_set_anon_rmap(page, vma, address);
+ } else
+ bc_rsspage_release(pb);
    /* else checking page index and mapping is racy */
}

@@ -538,27 +546,35 @@ void page_add_anon_rmap(struct page *pag
 * @page: the page to add the mapping to
 * @vma: the vm area in which the mapping is added
 * @address: the user virtual address mapped
+ * @pb: the page beancounter to charge page with
 *

```

```

* Same as page_add_anon_rmap but must only be called on *new* pages.
* This means the inc-and-test can be bypassed.
*/
void page_add_new_anon_rmap(struct page *page,
- struct vm_area_struct *vma, unsigned long address)
+ struct vm_area_struct *vma, unsigned long address,
+ struct page_beancounter *pb)
{
    atomic_set(&page->_mapcount, 0); /* elevate count by 1 (starts at -1) */
+ bc_rsspage_charge(pb);
    __page_set_anon_rmap(page, vma, address);
}

/**
 * page_add_file_rmap - add pte mapping to a file page
- * @page: the page to add the mapping to
+ * @page: the page to add the mapping to
+ * @pb: the page beancounter to charge page with
 *
 * The caller needs to hold the pte lock.
 */
-void page_add_file_rmap(struct page *page)
+void page_add_file_rmap(struct page *page, struct page_beancounter *pb)
{
- if (atomic_inc_and_test(&page->_mapcount))
+ if (atomic_inc_and_test(&page->_mapcount)) {
+ if (pb)
+ bc_rsspage_charge(pb);
    __inc_zone_page_state(page, NR_FILE_MAPPED);
+ } else if (pb)
+ bc_rsspage_release(pb);
}

/**
@@ -569,6 +585,9 @@ void page_add_file_rmap(struct page *pag
 */
void page_remove_rmap(struct page *page)
{
+ struct page_beancounter *pb;
+
+ pb = page_pb(page);
    if (atomic_add_negative(-1, &page->_mapcount)) {
        if (unlikely(page_mapcount(page) < 0)) {
            printk (KERN_EMERG "Eeek! page_mapcount(page) went negative! (%d)\n",
page_mapcount(page));
@@ -578,6 +597,7 @@ void page_remove_rmap(struct page *page)
    BUG();
}

```

```

+ bc_rsspage_uncharge(pb);
/*
 * It would be tidy to reset the PageAnon mapping here,
 * but that might overwrite a racing page_add_anon_rmap
--- ./mm/swapfile.c 2006-11-07 11:56:20.000000000 +0300
+++ ./mm/swapfile.c 2006-11-07 11:57:23.000000000 +0300
@@ -28,6 +28,8 @@
#include <linux/capability.h>
#include <linux/syscalls.h>

+#include <bc/rsspapes.h>
+
#include <asm/pgtable.h>
#include <asm/tlbflush.h>
#include <linux/swapops.h>
@@ -501,13 +503,14 @@ unsigned int count_swap_pages(int type,
 * force COW, vm_page_prot omits write permission from any private vma.
 */
static void unuse_pte(struct vm_area_struct *vma, pte_t *pte,
- unsigned long addr, swp_entry_t entry, struct page *page)
+ unsigned long addr, swp_entry_t entry, struct page *page,
+ struct page_beancounter *pb)
{
    inc_mm_counter(vma->vm_mm, anon_rss);
    get_page(page);
    set_pte_at(vma->vm_mm, addr, pte,
        pte_mkold(mk_pte(page, vma->vm_page_prot)));
- page_add_anon_rmap(page, vma, addr);
+ page_add_anon_rmap(page, vma, addr, pb);
    swap_free(entry);
/*
 * Move the page to the active list so it is not
@@ -524,6 +527,10 @@ static int unuse_pte_range(struct vm_are
    pte_t *pte;
    spinlock_t *ptl;
    int found = 0;
+ struct page_beancounter *pb;
+
+ if (bc_rsspage_prepare(page, vma, &pb))
+ return 0;

    pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
    do {
@@ -532,12 +539,14 @@ static int unuse_pte_range(struct vm_are
 * Test inline before going to call unuse_pte.
 */
    if (unlikely(pte_same(*pte, swp_pte))) {

```

```

- unuse_pte(vma, pte++, addr, entry, page);
+ unuse_pte(vma, pte++, addr, entry, page, pb);
  found = 1;
  break;
}
} while (pte++, addr += PAGE_SIZE, addr != end);
pte_unmap_unlock(pte - 1, ptl);
+ if (!found)
+ bc_rsspage_release(pb);
  return found;
}

--- ./mm/vmscan.c.ve10 2006-11-07 11:56:20.000000000 +0300
+++ ./mm/vmscan.c 2006-11-07 11:57:23.000000000 +0300
@@ -39,6 +39,8 @@
#include <linux/kthread.h>
#include <linux/freezer.h>

+#include <bc/rsspages.h>
+
#include <asm/tlbflush.h>
#include <asm/div64.h>

@@ -1103,6 +1105,194 @@ out:
  return ret;
}

+#ifdef CONFIG_BEANCOUNTERS
+/*
+ * These are bc's inactive and active pages shrinkers.
+ * These work like shrink_inactive_list() and shrink_active_list()
+ *
+ * Two main differences is that bc_isolate_pages() is used to isolate
+ * pages, and that reclaim_mapped is considered to be 1 as hitting BC
+ * limit implies we have to shrink_mapped_pages
+ */
+static unsigned long bc_shrink_pages_inactive(unsigned long max_scan,
+ struct beancounter *bc, struct scan_control *sc)
+{
+ LIST_HEAD(page_list);
+ unsigned long nr_scanned = 0;
+ unsigned long nr_reclaimed = 0;
+
+ do {
+ struct page *page;
+ unsigned long nr_taken;
+ unsigned long nr_scan;
+ struct zone *z;

```

```

+
+ nr_taken = bc_isolate_pages(sc->swap_cluster_max, bc,
+   &page_list, 0, &nr_scan);
+
+ nr_scanned += nr_scan;
+ nr_reclaimed += shrink_page_list(&page_list, sc);
+ if (nr_taken == 0)
+   goto done;
+
+ while (!list_empty(&page_list)) {
+   page = lru_to_page(&page_list);
+   z = page_zone(page);
+
+   spin_lock_irq(&z->lru_lock);
+   VM_BUG_ON(PageLRU(page));
+   SetPageLRU(page);
+   list_del(&page->lru);
+   if (PageActive(page))
+     add_page_to_active_list(z, page);
+   else
+     add_page_to_inactive_list(z, page);
+   spin_unlock_irq(&z->lru_lock);
+
+   put_page(page);
+ }
+ } while (nr_scanned < max_scan);
+done:
+ return nr_reclaimed;
+}
+
+static void bc_shrink_pages_active(unsigned long nr_pages,
+ struct beancounter *bc, struct scan_control *sc)
+{
+ LIST_HEAD(l_hold);
+ LIST_HEAD(l_inactive);
+ LIST_HEAD(l_active);
+ struct page *page;
+ unsigned long nr_scanned;
+ unsigned long nr_deactivated = 0;
+ struct zone *z;
+
+ bc_isolate_pages(nr_pages, bc, &l_hold, 1, &nr_scanned);
+
+ while (!list_empty(&l_hold)) {
+   cond_resched();
+   page = lru_to_page(&l_hold);
+   list_del(&page->lru);
+   if (page_mapped(page)) {

```

```

+ if ((total_swap_pages == 0 && PageAnon(page)) ||
+     page_referenced(page, 0)) {
+     list_add(&page->lru, &l_active);
+     continue;
+ }
+ }
+ nr_deactivated++;
+ list_add(&page->lru, &l_inactive);
+ }
+
+ while (!list_empty(&l_inactive)) {
+     page = lru_to_page(&l_inactive);
+     z = page_zone(page);
+
+     spin_lock_irq(&z->lru_lock);
+     VM_BUG_ON(PageLRU(page));
+     SetPageLRU(page);
+     VM_BUG_ON(!PageActive(page));
+     ClearPageActive(page);
+
+     list_move(&page->lru, &z->inactive_list);
+     z->nr_inactive++;
+     spin_unlock_irq(&z->lru_lock);
+
+     put_page(page);
+ }
+
+ while (!list_empty(&l_active)) {
+     page = lru_to_page(&l_active);
+     z = page_zone(page);
+
+     spin_lock_irq(&z->lru_lock);
+     VM_BUG_ON(PageLRU(page));
+     SetPageLRU(page);
+     VM_BUG_ON(!PageActive(page));
+     list_move(&page->lru, &z->active_list);
+     z->nr_active++;
+     spin_unlock_irq(&z->lru_lock);
+
+     put_page(page);
+ }
+}
+
+/*
+ * This is a reworked shrink_zone() routine - it scans active pages firsts,
+ * then inactive and returns the number of pages reclaimed
+ */
+static unsigned long bc_shrink_pages(int priority, struct beancounter *bc,

```

```

+ struct scan_control *sc)
+{
+ unsigned long nr_pages;
+ unsigned long nr_to_scan;
+ unsigned long nr_reclaimed = 0;
+
+ nr_pages = (bc_nr_physpages(bc) >> priority) + 1;
+ if (nr_pages < sc->swap_cluster_max)
+ nr_pages = 0;
+
+ while (nr_pages) {
+ nr_to_scan = min(nr_pages, (unsigned long)sc->swap_cluster_max);
+ nr_pages -= nr_to_scan;
+ bc_shrink_pages_active(nr_to_scan, bc, sc);
+ }
+
+ nr_pages = (bc_nr_physpages(bc) >> priority) + 1;
+ if (nr_pages < sc->swap_cluster_max)
+ nr_pages = 0;
+
+ while (nr_pages) {
+ nr_to_scan = min(nr_pages, (unsigned long)sc->swap_cluster_max);
+ nr_pages -= nr_to_scan;
+ nr_reclaimed += bc_shrink_pages_inactive(nr_to_scan, bc, sc);
+ }
+
+ throttle_vm_writeout();
+ return nr_reclaimed;
+}
+
+/*
+ * This functions works like try_to_free_pages() - it tries
+ * to shrink bc's pages with increasing priority
+ */
+unsigned long bc_try_to_free_pages(struct beancounter *bc)
+{
+ int priority;
+ int ret = 0;
+ unsigned long total_scanned = 0;
+ unsigned long nr_reclaimed = 0;
+ struct scan_control sc = {
+ .gfp_mask = GFP_KERNEL,
+ .may_writpage = !laptop_mode,
+ .swap_cluster_max = SWAP_CLUSTER_MAX,
+ .may_swap = 1,
+ .swappiness = vm_swappiness,
+ };
+
+

```

```

+ for (priority = DEF_PRIORITY; priority >= 0; priority--) {
+   sc.nr_scanned = 0;
+   nr_reclaimed += bc_shrink_pages(priority, bc, &sc);
+   total_scanned += sc.nr_scanned;
+   if (nr_reclaimed >= sc.swap_cluster_max) {
+     ret = 1;
+     goto out;
+   }
+
+   if (total_scanned > sc.swap_cluster_max +
+       sc.swap_cluster_max / 2) {
+     wakeup_pdflush(laptop_mode ? 0 : total_scanned);
+     sc.may_writepage = 1;
+   }
+
+   if (sc.nr_scanned && priority < DEF_PRIORITY - 2)
+     congestion_wait(WRITE, HZ/10);
+ }
+out:
+ return ret;
+}
+
+/*
+ * For kswapd, balance_pgdat() will work across all this node's zones until
+ * they are all at pages_high.

```
