
Subject: [PATCH 9/13] BC: privvmpages accounting (hooks)

Posted by [dev](#) on Thu, 09 Nov 2006 16:57:24 GMT

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

BC privvmpages accounting hooks in generic code.

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

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

```
fs/binfmt_elf.c      |  5 +--
include/asm-alpha/mman.h |  1
include/asm-generic/mman.h |  1
include/asm-mips/mman.h |  1
include/asm-parisc/mman.h |  1
include/asm-xtensa/mman.h |  1
kernel/fork.c        | 11 ++++++
mm/mlock.c           | 16 ++++++++
mm/mmap.c            | 74 ++++++++-----
mm/mprotect.c        | 18 ++++++++
mm/mremap.c          | 22 ++++++++
mm/shmem.c           | 23 ++++++++
12 files changed, 153 insertions(+), 21 deletions(-)
```

--- ./fs/binfmt_elf.c.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300

+++ ./fs/binfmt_elf.c 2006-11-03 17:50:04.000000000 +0300

@@ -363,7 +363,7 @@ static unsigned long load_elf_interp(str

```
    epnt = elf_phdata;
    for (i = 0; i < interp_elf_ex->e_phnum; i++, epnt++) {
        if (epnt->p_type == PT_LOAD) {
-       int elf_type = MAP_PRIVATE | MAP_DENYWRITE;
+       int elf_type = MAP_PRIVATE|MAP_DENYWRITE|MAP_EXECPRIO;
        int elf_prot = 0;
        unsigned long vaddr = 0;
        unsigned long k, map_addr;
```

@@ -849,7 +849,8 @@ static int load_elf_binary(struct linux_

```
    if (elf_ppnt->p_flags & PF_X)
        elf_prot |= PROT_EXEC;

-   elf_flags = MAP_PRIVATE | MAP_DENYWRITE | MAP_EXECUTABLE;
+   elf_flags = MAP_PRIVATE | MAP_DENYWRITE | MAP_EXECUTABLE |
+   MAP_EXECPRIO;
```

```
    vaddr = elf_ppnt->p_vaddr;
    if (loc->elf_ex.e_type == ET_EXEC || load_addr_set) {
--- ./include/asm-alpha/mman.h.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./include/asm-alpha/mman.h 2006-11-03 17:50:04.000000000 +0300
```

```

@@ -14,6 +14,7 @@
#define MAP_TYPE 0x0f /* Mask for type of mapping (OSF/1 is _wrong_) */
#define MAP_FIXED 0x100 /* Interpret addr exactly */
#define MAP_ANONYMOUS 0x10 /* don't use a file */
+#define MAP_EXECPRIO 0x20 /* charge with BC_LIMIT severity */

/* not used by linux, but here to make sure we don't clash with OSF/1 defines */
#define _MAP_HASSEMAPHORE 0x0200
--- ./include/asm-generic/mman.h.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./include/asm-generic/mman.h 2006-11-03 17:50:04.000000000 +0300
@@ -19,6 +19,7 @@
#define MAP_TYPE 0x0f /* Mask for type of mapping */
#define MAP_FIXED 0x10 /* Interpret addr exactly */
#define MAP_ANONYMOUS 0x20 /* don't use a file */
+#define MAP_EXECPRIO 0x2000 /* charge with BC_LIMIT severity */

#define MS_ASYNC 1 /* sync memory asynchronously */
#define MS_INVALIDATE 2 /* invalidate the caches */
--- ./include/asm-mips/mman.h.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./include/asm-mips/mman.h 2006-11-03 17:50:04.000000000 +0300
@@ -46,6 +46,7 @@
#define MAP_LOCKED 0x8000 /* pages are locked */
#define MAP_POPULATE 0x10000 /* populate (prefault) pagetables */
#define MAP_NONBLOCK 0x20000 /* do not block on IO */
+#define MAP_EXECPRIO 0x40000 /* charge with BC_LIMIT severity */

/*
 * Flags for msync
--- ./include/asm-parisc/mman.h.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./include/asm-parisc/mman.h 2006-11-03 17:50:04.000000000 +0300
@@ -22,6 +22,7 @@
#define MAP_GROWSDOWN 0x8000 /* stack-like segment */
#define MAP_POPULATE 0x10000 /* populate (prefault) pagetables */
#define MAP_NONBLOCK 0x20000 /* do not block on IO */
+#define MAP_EXECPRIO 0x40000 /* charge with BC_LIMIT severity */

#define MS_SYNC 1 /* synchronous memory sync */
#define MS_ASYNC 2 /* sync memory asynchronously */
--- ./include/asm-xtensa/mman.h.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./include/asm-xtensa/mman.h 2006-11-03 17:50:04.000000000 +0300
@@ -53,6 +53,7 @@
#define MAP_LOCKED 0x8000 /* pages are locked */
#define MAP_POPULATE 0x10000 /* populate (prefault) pagetables */
#define MAP_NONBLOCK 0x20000 /* do not block on IO */
+#define MAP_EXECPRIO 0x40000 /* charge with BC_LIMIT severity */

/*
 * Flags for msync

```

```

--- ./kernel/fork.c.bcvmpagesk 2006-11-03 17:48:46.000000000 +0300
+++ ./kernel/fork.c 2006-11-03 17:50:04.000000000 +0300
@@ -50,6 +50,7 @@ static inline int dup_mmap(struct mm_str
#include <linux/random.h>

#include <bc/task.h>
+#include <bc/vmpages.h>

#include <asm/pgtable.h>
#include <asm/pgalloc.h>
@@ -257,6 +257,9 @@ static inline int dup_mmap(struct mm_str
    tmp->vm_flags &= ~VM_LOCKED;
    tmp->vm_mm = mm;
    tmp->vm_next = NULL;
+ vma_set_bc(tmp);
+ if (bc_vma_charge(tmp))
+ goto fail_charge;
    anon_vma_link(tmp);
    file = tmp->vm_file;
    if (file) {
@@ -299,6 +302,10 @@ out:
    flush_tlb_mm(oldmm);
    up_write(&oldmm->mmap_sem);
    return retval;
+
+fail_charge:
+ mpol_free(pol);
+ vma_release_bc(tmp);
fail_nomem_policy:
    kmem_cache_free(vm_area_cachep, tmp);
fail_nomem:
@@ -349,6 +356,7 @@ static struct mm_struct * mm_init(struct
    mm->cached_hole_size = ~0UL;

    if (likely(!mm_alloc_pgd(mm))) {
+ mm_init_beancounter(mm);
    mm->def_flags = 0;
    return mm;
    }
@@ -379,6 +387,7 @@ struct mm_struct * mm_alloc(void)
void fastcall __mmdrop(struct mm_struct *mm)
{
    BUG_ON(mm == &init_mm);
+ mm_free_beancounter(mm);
    mm_free_pgd(mm);
    destroy_context(mm);
    free_mm(mm);
@@ -526,6 +535,7 @@ fail_nocontext:

```

```

    * If init_new_context() failed, we cannot use mmpu() to free the mm
    * because it calls destroy_context()
    */
+ mm_free_beancounter(mm);
  mm_free_pgdp(mm);
  free_mm(mm);
  return NULL;
--- ./mm/mlock.c.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./mm/mlock.c 2006-11-03 17:50:04.000000000 +0300
@@ -11,6 +11,7 @@
#include <linux/mempolicy.h>
#include <linux/syscalls.h>

+#include <bc/vmpages.h>

static int mlock_fixup(struct vm_area_struct *vma, struct vm_area_struct **prev,
    unsigned long start, unsigned long end, unsigned int newflags)
@@ -19,12 +20,21 @@ static int mlock_fixup(struct vm_area_struct
    pgoff_t pgoff;
    int pages;
    int ret = 0;
-
+ int bc_recharge;
+
+ bc_recharge = BC_NOCHARGE;
+ if (newflags == vma->vm_flags) {
+     *prev = vma;
+     goto out;
+ }

+ bc_recharge = bc_need_memory_recharge(vma, vma->vm_file, newflags);
+ if (bc_recharge == BC_CHARGE) {
+     ret = __bc_memory_charge(mm, end - start, BC_BARRIER);
+     if (ret < 0)
+         goto out;
+ }
+
    pgoff = vma->vm_pgoff + ((start - vma->vm_start) >> PAGE_SHIFT);
    *prev = vma_merge(mm, *prev, start, end, newflags, vma->anon_vma,
        vma->vm_file, pgoff, vma_policy(vma));
@@ -48,6 +58,8 @@ static int mlock_fixup(struct vm_area_struct
}

success:
+ if (bc_recharge == BC_UNCHARGE)
+     __bc_memory_uncharge(mm, end - start);
/*
    * vm_flags is protected by the mmap_sem held in write mode.

```

* It's okay if try_to_unmap_one unmaps a page just after we
@@ -67,6 +79,8 @@ success:

```
vma->vm_mm->locked_vm -= pages;
out:
+ if (ret < 0 && bc_recharge == BC_CHARGE)
+ __bc_memory_uncharge(mm, end - start);
+ if (ret == -ENOMEM)
+   ret = -EAGAIN;
+ return ret;
--- ./mm/mmap.c.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./mm/mmap.c 2006-11-03 17:50:04.000000000 +0300
@@ -26,6 +26,8 @@
#include <linux/mempolicy.h>
#include <linux/rmap.h>

+#include <bc/vmpages.h>
+
#include <asm/uaccess.h>
#include <asm/cacheflush.h>
#include <asm/tlb.h>
@@ -37,6 +39,7 @@
static void unmap_region(struct mm_struct *mm,
    struct vm_area_struct *vma, struct vm_area_struct *prev,
    unsigned long start, unsigned long end);
+static unsigned long __do_brk(unsigned long addr, unsigned long len, int prio);

/*
 * WARNING: the debugging will use recursive algorithms so never enable this
@@ -224,6 +227,8 @@ static struct vm_area_struct *remove_vma
    struct vm_area_struct *next = vma->vm_next;

    might_sleep();
+
+ bc_vma_uncharge(vma);
+ if (vma->vm_ops && vma->vm_ops->close)
+   vma->vm_ops->close(vma);
+ if (vma->vm_file)
@@ -271,7 +276,7 @@ asmlinkage unsigned long sys_brk(unsigned
    goto out;

    /* Ok, looks good - let it rip. */
- if (do_brk(oldbrk, newbrk-oldbrk) != oldbrk)
+ if (__do_brk(oldbrk, newbrk-oldbrk, BC_BARRIER) != oldbrk)
    goto out;
set_brk:
    mm->brk = brk;
@@ -620,6 +625,7 @@ again: remove_next = 1 + (end > next->
```

```

    fput(file);
    mm->map_count--;
    mpol_free(vma_policy(next));
+ vma_release_bc(next);
    kmem_cache_free(vm_area_cache, next);
    /*
     * In mprotect's case 6 (see comments on vma_merge),
@@ -761,15 +767,17 @@ struct vm_area_struct *vma_merge(struct
    */
    if (prev && prev->vm_end == addr &&
        mpol_equal(vma_policy(prev), policy) &&
+ bc_equal(mm->mm_bc, prev->vma_bc) &&
        can_vma_merge_after(prev, vm_flags,
- anon_vma, file, pgoff)) {
+ anon_vma, file, pgoff)) {
    /*
     * OK, it can. Can we now merge in the successor as well?
    */
    if (next && end == next->vm_start &&
        mpol_equal(policy, vma_policy(next)) &&
+ bc_equal(mm->mm_bc, next->vma_bc) &&
        can_vma_merge_before(next, vm_flags,
- anon_vma, file, pgoff+pglen) &&
+ anon_vma, file, pgoff + pglen) &&
        is_mergeable_anon_vma(prev->anon_vma,
            next->anon_vma)) {
        /* cases 1, 6 */
@@ -786,8 +794,9 @@ struct vm_area_struct *vma_merge(struct
    */
    if (next && end == next->vm_start &&
        mpol_equal(policy, vma_policy(next)) &&
+ bc_equal(mm->mm_bc, next->vma_bc) &&
        can_vma_merge_before(next, vm_flags,
- anon_vma, file, pgoff+pglen)) {
+ anon_vma, file, pgoff + pglen)) {
        if (prev && addr < prev->vm_end) /* case 4 */
            vma_adjust(prev, prev->vm_start,
                addr, prev->vm_pgoff, NULL);
@@ -828,6 +837,7 @@ struct anon_vma *find_mergeable_anon_vma

    if (near->anon_vma && vma->vm_end == near->vm_start &&
        mpol_equal(vma_policy(vma), vma_policy(near)) &&
+ bc_equal(vma->vma_bc, near->vma_bc) &&
        can_vma_merge_before(near, vm_flags,
            NULL, vma->vm_file, vma->vm_pgoff +
                ((vma->vm_end - vma->vm_start) >> PAGE_SHIFT)))
@@ -849,6 +859,7 @@ try_prev:

```

```

    if (near->anon_vma && near->vm_end == vma->vm_start &&
        mpol_equal(vma_policy(near), vma_policy(vma)) &&
+   bc_equal(vma->vma_bc, near->vma_bc) &&
        can_vma_merge_after(near, vm_flags,
            NULL, vma->vm_file, vma->vm_pgoff))
        return near->anon_vma;
@@ -1054,6 +1065,10 @@ munmap_back:
    }
}

+ if (bc_memory_charge(mm, len, file, vm_flags,
+   flags & MAP_EXECPRIO ? BC_LIMIT : BC_BARRIER))
+   goto charge_error;
+
/*
 * Can we just expand an old private anonymous mapping?
 * The VM_SHARED test is necessary because shmem_zero_setup
@@ -1082,6 +1097,7 @@ munmap_back:
    vma->vm_page_prot = protection_map[vm_flags &
        (VM_READ|VM_WRITE|VM_EXEC|VM_SHARED)];
    vma->vm_pgoff = pgoff;
+ vma_set_bc(vma);

    if (file) {
        error = -EINVAL;
@@ -1138,6 +1154,7 @@ munmap_back:
        fput(file);
    }
    mpol_free(vma_policy(vma));
+ vma_release_bc(vma);
    kmem_cache_free(vm_area_cachep, vma);
}
out:
@@ -1167,6 +1184,8 @@ unmap_and_free_vma:
free_vma:
    kmem_cache_free(vm_area_cachep, vma);
unacct_error:
+ bc_memory_uncharge(mm, len, file, vm_flags);
+charge_error:
    if (charged)
        vm_unacct_memory(charged);
    return error;
@@ -1496,12 +1515,16 @@ static int acct_stack_growth(struct vm_a
    return -ENOMEM;
}

+ if (bc_memory_charge(mm, grow << PAGE_SHIFT,
+   vma->vm_file, vma->vm_flags, BC_LIMIT))

```

```

+ goto out_charge;
+
/*
 * Overcommit.. This must be the final test, as it will
 * update security statistics.
 */
if (security_vm_enough_memory(grow))
- return -ENOMEM;
+ goto out_sec;

/* Ok, everything looks good - let it rip */
mm->total_vm += grow;
@@ -1509,6 +1532,11 @@ static int acct_stack_growth(struct vm_a
mm->locked_vm += grow;
vm_stat_account(mm, vma->vm_flags, vma->vm_file, grow);
return 0;
+
+out_sec:
+ bc_memory_uncharge(mm, grow << PAGE_SHIFT, vma->vm_file, vma->vm_flags);
+out_charge:
+ return -ENOMEM;
}

#if defined(CONFIG_STACK_GROWSUP) || defined(CONFIG_IA64)
@@ -1742,6 +1770,7 @@ int split_vma(struct mm_struct * mm, str

/* most fields are the same, copy all, and then fixup */
*new = *vma;
+ vma_copy_bc(new);

if (new_below)
new->vm_end = addr;
@@ -1752,6 +1781,7 @@ int split_vma(struct mm_struct * mm, str

pol = mpol_copy(vma_policy(vma));
if (IS_ERR(pol)) {
+ vma_release_bc(new);
kmem_cache_free(vm_area_cachep, new);
return PTR_ERR(pol);
}
@@ -1864,7 +1894,8 @@ static inline void verify_mm_writelocked
* anonymous maps. eventually we may be able to do some
* brk-specific accounting here.
*/
-unsigned long do_brk(unsigned long addr, unsigned long len)
+static unsigned long __do_brk(unsigned long addr, unsigned long len,
+ int bc_prio)
{

```



```

    struct mm_struct * mm = current->mm;
    struct vm_area_struct * vma, * prev;
@@ -1924,7 +1955,10 @@ unsigned long do_brk(unsigned long addr,
    return -ENOMEM;

    if (security_vm_enough_memory(len >> PAGE_SHIFT))
- return -ENOMEM;
+ goto err_sec;
+
+ if (bc_memory_charge(mm, len, NULL, flags, bc_prio))
+ goto err_ch;

    /* Can we just expand an old private anonymous mapping? */
    if (vma_merge(mm, prev, addr, addr + len, flags,
@@ -1935,10 +1969,8 @@ unsigned long do_brk(unsigned long addr,
        * create a vma struct for an anonymous mapping
        */
    vma = kmem_cache_zalloc(vm_area_cachep, GFP_KERNEL);
- if (!vma) {
- vm_unacct_memory(len >> PAGE_SHIFT);
- return -ENOMEM;
- }
+ if (!vma)
+ goto err_alloc;

    vma->vm_mm = mm;
    vma->vm_start = addr;
@@ -1947,6 +1979,7 @@ unsigned long do_brk(unsigned long addr,
    vma->vm_flags = flags;
    vma->vm_page_prot = protection_map[flags &
        (VM_READ|VM_WRITE|VM_EXEC|VM_SHARED)];
+ vma_set_bc(vma);
    vma_link(mm, vma, prev, rb_link, rb_parent);
out:
    mm->total_vm += len >> PAGE_SHIFT;
@@ -1955,8 +1988,19 @@ out:
    make_pages_present(addr, addr + len);
    }
    return addr;
+
+err_alloc:
+ bc_memory_uncharge(mm, len, NULL, flags);
+err_ch:
+ vm_unacct_memory(len >> PAGE_SHIFT);
+err_sec:
+ return -ENOMEM;
    }

```

```

+unsigned long do_brk(unsigned long addr, unsigned long len)
+{
+ return __do_brk(addr, len, BC_LIMIT);
+}
EXPORT_SYMBOL(do_brk);

/* Release all mmap. */
@@ -2018,6 +2062,12 @@ int insert_vm_struct(struct mm_struct *
    if ((vma->vm_flags & VM_ACCOUNT) &&
        security_vm_enough_memory(vma_pages(vma)))
        return -ENOMEM;
+
+ if (bc_vma_charge(vma)) {
+ if (vma->vm_flags & VM_ACCOUNT)
+ vm_unacct_memory(vma_pages(vma));
+ return -ENOMEM;
+ }
    vma_link(mm, vma, prev, rb_link, rb_parent);
    return 0;
}
@@ -2057,8 +2107,10 @@ struct vm_area_struct *copy_vma(struct v
    new_vma = kmem_cache_alloc(vm_area_cachep, SLAB_KERNEL);
    if (new_vma) {
        *new_vma = *vma;
+ vma_copy_bc(new_vma);
        pol = mpol_copy(vma_policy(vma));
        if (IS_ERR(pol)) {
+ vma_release_bc(new_vma);
            kmem_cache_free(vm_area_cachep, new_vma);
            return NULL;
        }
--- ./mm/mprotect.c.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./mm/mprotect.c 2006-11-03 17:50:04.000000000 +0300
@@ -22,6 +22,7 @@
#include <linux/module.h>
#include <linux/swap.h>
#include <linux/swapops.h>
+#include <bc/vmpages.h>
#include <asm/uaccess.h>
#include <asm/pgtable.h>
#include <asm/cache flush.h>
@@ -140,12 +141,20 @@ mprotect_fixup(struct vm_area_struct *vm
    pgoff_t pgoff;
    int error;
    int dirty_accountable = 0;
+ int bc_recharge;

    if (newflags == oldflags) {

```

```

    *pprev = vma;
    return 0;
}

+ bc_recharge = bc_need_memory_recharge(vma, vma->vm_file, newflags);
+ if (bc_recharge == BC_CHARGE) {
+   error = __bc_memory_charge(mm, end - start, BC_BARRIER);
+   if (error < 0)
+     goto fail_charge;
+ }
+
/*
 * If we make a private mapping writable we increase our commit;
 * but (without finer accounting) cannot reduce our commit if we
@@ -157,8 +166,9 @@ mprotect_fixup(struct vm_area_struct *vm
if (newflags & VM_WRITE) {
if (!(oldflags & (VM_ACCOUNT|VM_WRITE|VM_SHARED))) {
charged = nrpages;
+   error = -ENOMEM;
+   if (security_vm_enough_memory(charged))
-   return -ENOMEM;
+   goto fail_acct;
+   newflags |= VM_ACCOUNT;
}
}
@@ -189,6 +199,8 @@ mprotect_fixup(struct vm_area_struct *vm
}

success:
+ if (bc_recharge == BC_UNCHARGE)
+   __bc_memory_uncharge(mm, end - start);
/*
 * vm_flags and vm_page_prot are protected by the mmap_sem
 * held in write mode.
@@ -212,6 +224,10 @@ success:

fail:
vm_unacct_memory(charged);
+fail_acct:
+ if (bc_recharge == BC_CHARGE)
+   __bc_memory_uncharge(mm, end - start);
+fail_charge:
return error;
}
/*
--- ./mm/mremap.c.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./mm/mremap.c 2006-11-03 17:50:04.000000000 +0300
@@ -19,6 +19,8 @@

```

```

#include <linux/security.h>
#include <linux/syscalls.h>

+#include <bc/vmpages.h>
+
#include <asm/uaccess.h>
#include <asm/cache flush.h>
#include <asm/tlbflush.h>
@@ -344,10 +346,16 @@ unsigned long do_mremap(unsigned long ad
    goto out;
}

+ if (bc_memory_charge(mm, new_len - old_len,
+   vma->vm_file, vma->vm_flags, BC_BARRIER)) {
+   ret = -ENOMEM;
+   goto out;
+ }
+
+ if (vma->vm_flags & VM_ACCOUNT) {
+   charged = (new_len - old_len) >> PAGE_SHIFT;
+   if (security_vm_enough_memory(charged))
-   goto out_nc;
+   goto out_bc;
+ }

/* old_len exactly to the end of the area..
@@ -374,7 +382,7 @@ unsigned long do_mremap(unsigned long ad
    addr + new_len);
}
ret = addr;
- goto out;
+ goto out_nc;
}
}

@@ -393,14 +401,18 @@ unsigned long do_mremap(unsigned long ad
    vma->vm_pgoff, map_flags);
ret = new_addr;
if (new_addr & ~PAGE_MASK)
- goto out;
+ goto out_nc;
}
ret = move_vma(vma, addr, old_len, new_len, new_addr);
}
-out:
+out_nc:
if (ret & ~PAGE_MASK)
    vm_unacct_memory(charged);

```

```

-out_nc:
+out_bc:
+ if (ret & ~PAGE_MASK)
+   bc_memory_uncharge(mm, new_len - old_len,
+     vma->vm_file, vma->vm_flags);
+out:
+   return ret;
+}

--- ./mm/shmem.c.bcvmpagesk 2006-11-03 17:46:10.000000000 +0300
+++ ./mm/shmem.c 2006-11-03 17:50:04.000000000 +0300
@@ -50,6 +50,8 @@
#include <linux/highmem.h>
#include <linux/backing-dev.h>

+#include <bc/vmpages.h>
+
#include <asm/uaccess.h>
#include <asm/div64.h>
#include <asm/pgtable.h>
@@ -372,7 +374,8 @@ static swp_entry_t *shmem_swp_alloc(stru
}

spin_unlock(&info->lock);
- page = shmem_dir_alloc(mapping_gfp_mask(inode->i_mapping) | __GFP_ZERO);
+ page = shmem_dir_alloc(mapping_gfp_mask(inode->i_mapping) | \
+   __GFP_ZERO | __GFP_BC);
if (page)
set_page_private(page, 0);
spin_lock(&info->lock);
@@ -2525,6 +2528,24 @@ int shmem_zero_setup(struct vm_area_stru

if (vma->vm_file)
fput(vma->vm_file);
+ else {
+   /*
+    * vma obtains file which was not on it when bc_memory_charge()
+    * was called in do_mmap_pgoff, so we must check whether or
+    * not to recharge private memory
+    */
+   switch (bc_need_memory_recharge(vma, file, vma->vm_flags)) {
+   case BC_UNCHARGE:
+     __bc_memory_uncharge(vma->vm_mm,
+       vma->vm_end - vma->vm_start);
+     break;
+   case BC_CHARGE:
+     if (__bc_memory_charge(vma->vm_mm,
+       vma->vm_end - vma->vm_start,

```

```
+    BC_BARRIER))
+    return -ENOMEM;
+ }
+ }
vma->vm_file = file;
vma->vm_ops = &shmem_vm_ops;
return 0;
```
