
Subject: [PATCH 7/13] BC: kmemsize accounting (hooks)

Posted by [dev](#) on Thu, 09 Nov 2006 16:54:37 GMT

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

Mark some kmem caches with SLAB_BC and some allocations with __GFP_BC to cause charging/limiting of appropriate kernel resources.

Signed-off-by: Pavel Emelianov <xemul@sw.ru>

Signed-off-by: Kirill Korotaev <dev@sw.ru>

```
arch/i386/kernel/ldt.c      |  4 ++--
arch/i386/mm/init.c         |  4 ++--
arch/i386/mm/pgtable.c      |  6 +++++-
drivers/char/tty_io.c       | 10 +++++-----
fs/file.c                   |  6 +++--
fs/locks.c                  |  2 +-
fs/namespace.c              |  3 ++-
fs/select.c                 |  7 +++++-
include/asm-i386/thread_info.h |  4 ++--
include/asm-ia64/pgalloc.h  | 24 ++++++-----
include/asm-x86_64/pgalloc.h | 12 ++++++----
include/asm-x86_64/thread_info.h |  5 +++-
ipc/msgutil.c               |  4 ++--
ipc/sem.c                   |  7 +++++-
ipc/util.c                  |  8 +++++-
kernel/fork.c               | 15 ++++++-----
kernel/posix-timers.c       |  3 ++-
kernel/signal.c             |  2 +-
kernel/user.c               |  2 +-
mm/mempool.c                |  2 ++
mm/page_alloc.c             | 11 ++++++-----
mm/slab.c                   | 30 ++++++-----
22 files changed, 115 insertions(+), 56 deletions(-)
```

--- ./arch/i386/kernel/ldt.c.bckmem 2006-11-09 11:29:09.000000000 +0300

+++ ./arch/i386/kernel/ldt.c 2006-11-09 11:33:27.000000000 +0300

```
@@ -39,9 +39,9 @@ static int alloc_ldt(mm_context_t *pc, i
    oldsize = pc->size;
    mincount = (mincount+511)&(~511);
    if (mincount*LDT_ENTRY_SIZE > PAGE_SIZE)
-   newldt = vmalloc(mincount*LDT_ENTRY_SIZE);
+   newldt = vmalloc_bc(mincount*LDT_ENTRY_SIZE);
    else
-   newldt = kmalloc(mincount*LDT_ENTRY_SIZE, GFP_KERNEL);
+   newldt = kmalloc(mincount*LDT_ENTRY_SIZE, GFP_KERNEL_BC);
```

```

if (!newldt)
    return -ENOMEM;
--- ./arch/i386/mm/init.c.bckmem 2006-11-09 11:18:45.000000000 +0300
+++ ./arch/i386/mm/init.c 2006-11-09 11:33:27.000000000 +0300
@@ -708,7 +708,7 @@ void __init pgtable_cache_init(void)
    pmd_cache = kmem_cache_create("pmd",
        PTRS_PER_PMD*sizeof(pmd_t),
        PTRS_PER_PMD*sizeof(pmd_t),
-    0,
+    SLAB_BC,
        pmd_ctor,
        NULL);
    if (!pmd_cache)
@@ -717,7 +717,7 @@ void __init pgtable_cache_init(void)
    pgd_cache = kmem_cache_create("pgd",
        PTRS_PER_PGD*sizeof(pgd_t),
        PTRS_PER_PGD*sizeof(pgd_t),
-    0,
+    SLAB_BC,
        pgd_ctor,
        PTRS_PER_PMD == 1 ? pgd_dtor : NULL);
    if (!pgd_cache)
--- ./arch/i386/mm/pgtable.c.bckmem 2006-11-09 11:18:45.000000000 +0300
+++ ./arch/i386/mm/pgtable.c 2006-11-09 11:33:27.000000000 +0300
@@ -186,9 +186,11 @@ struct page *pte_alloc_one(struct mm_str
    struct page *pte;

#ifdef CONFIG_HIGHPTE
-    pte = alloc_pages(GFP_KERNEL|__GFP_HIGHMEM|__GFP_REPEAT|__GFP_ZERO, 0);
+    pte = alloc_pages(GFP_KERNEL|__GFP_HIGHMEM|__GFP_REPEAT|__GFP_ZERO |
+    + __GFP_BC | __GFP_BC_LIMIT, 0);
#else
-    pte = alloc_pages(GFP_KERNEL|__GFP_REPEAT|__GFP_ZERO, 0);
+    pte = alloc_pages(GFP_KERNEL|__GFP_REPEAT|__GFP_ZERO|
+    + __GFP_BC | __GFP_BC_LIMIT, 0);
#endif
    return pte;
}
--- ./drivers/char/tty_io.c.bckmem 2006-11-09 11:29:10.000000000 +0300
+++ ./drivers/char/tty_io.c 2006-11-09 11:33:27.000000000 +0300
@@ -167,7 +167,7 @@ static void release_mem(struct tty_struc

static struct tty_struct *alloc_tty_struct(void)
{
-    return kzalloc(sizeof(struct tty_struct), GFP_KERNEL);
+    return kzalloc(sizeof(struct tty_struct), GFP_KERNEL_BC);
}

```

```

static void tty_buffer_free_all(struct tty_struct *);
@@ -1932,7 +1932,7 @@ static int init_dev(struct tty_driver *d

    if (!*tp_loc) {
        tp = (struct ktermios *) kmalloc(sizeof(struct ktermios),
-    GFP_KERNEL);
+    GFP_KERNEL_BC);
        if (!tp)
            goto free_mem_out;
        *tp = driver->init_termios;
@@ -1940,7 +1940,7 @@ static int init_dev(struct tty_driver *d

    if (!*ltp_loc) {
        ltp = (struct ktermios *) kmalloc(sizeof(struct ktermios),
-    GFP_KERNEL);
+    GFP_KERNEL_BC);
        if (!ltp)
            goto free_mem_out;
        memset(ltp, 0, sizeof(struct ktermios));
@@ -1965,7 +1965,7 @@ static int init_dev(struct tty_driver *d

    if (!*o_tp_loc) {
        o_tp = (struct ktermios *)
-    kmalloc(sizeof(struct ktermios), GFP_KERNEL);
+    kmalloc(sizeof(struct ktermios), GFP_KERNEL_BC);
        if (!o_tp)
            goto free_mem_out;
        *o_tp = driver->other->init_termios;
@@ -1973,7 +1973,7 @@ static int init_dev(struct tty_driver *d

    if (!*o_ltp_loc) {
        o_ltp = (struct ktermios *)
-    kmalloc(sizeof(struct ktermios), GFP_KERNEL);
+    kmalloc(sizeof(struct ktermios), GFP_KERNEL_BC);
        if (!o_ltp)
            goto free_mem_out;
        memset(o_ltp, 0, sizeof(struct ktermios));
--- ./fs/file.c.bckmem 2006-11-09 11:29:11.000000000 +0300
+++ ./fs/file.c 2006-11-09 11:33:27.000000000 +0300
@@ -35,9 +35,9 @@ static DEFINE_PER_CPU(struct fdtable_def
static inline void * alloc_fdmem(unsigned int size)
{
    if (size <= PAGE_SIZE)
-    return kmalloc(size, GFP_KERNEL);
+    return kmalloc(size, GFP_KERNEL_BC);
    else
-    return vmalloc(size);

```

```

+ return vmalloc_bc(size);
}

static inline void free_fdarr(struct fdtable *fdt)
@@ -148,7 +148,7 @@ static struct fdtable * alloc_fdtable(un
    if (nr > NR_OPEN)
        nr = NR_OPEN;

- fdt = kmalloc(sizeof(struct fdtable), GFP_KERNEL);
+ fdt = kmalloc(sizeof(struct fdtable), GFP_KERNEL_BC);
    if (!fdt)
        goto out;
    fdt->max_fds = nr;
--- ./fs/locks.c.bckmem 2006-11-09 11:29:11.000000000 +0300
+++ ./fs/locks.c 2006-11-09 11:33:27.000000000 +0300
@@ -2228,7 +2228,7 @@ EXPORT_SYMBOL(lock_may_write);
static int __init filelock_init(void)
{
    filelock_cache = kmem_cache_create("file_lock_cache",
- sizeof(struct file_lock), 0, SLAB_PANIC,
+ sizeof(struct file_lock), 0, SLAB_PANIC | SLAB_BC,
    init_once, NULL);
    return 0;
}
--- ./fs/namespace.c.bckmem 2006-11-09 11:29:11.000000000 +0300
+++ ./fs/namespace.c 2006-11-09 11:33:27.000000000 +0300
@@ -1813,7 +1813,8 @@ void __init mnt_init(unsigned long mempa
    init_rwsem(&namespace_sem);

    mnt_cache = kmem_cache_create("mnt_cache", sizeof(struct vfsmount),
- 0, SLAB_HWCACHE_ALIGN | SLAB_PANIC, NULL, NULL);
+ 0, SLAB_HWCACHE_ALIGN | SLAB_BC | SLAB_PANIC,
+ NULL, NULL);

    mount_hashtable = (struct list_head *)__get_free_page(GFP_ATOMIC);

--- ./fs/select.c.bckmem 2006-11-09 11:29:12.000000000 +0300
+++ ./fs/select.c 2006-11-09 11:33:27.000000000 +0300
@@ -103,7 +103,8 @@ static struct poll_table_entry *poll_get
    if (!table || POLL_TABLE_FULL(table)) {
        struct poll_table_page *new_table;

- new_table = (struct poll_table_page *)__get_free_page(GFP_KERNEL);
+ new_table = (struct poll_table_page *)
+ __get_free_page(GFP_KERNEL_BC);
        if (!new_table) {
            p->error = -ENOMEM;
            __set_current_state(TASK_RUNNING);

```

```

@@ -339,7 +340,7 @@ static int core_sys_select(int n, fd_set
    if (size > sizeof(stack_fds) / 6) {
        /* Not enough space in on-stack array; must use kmalloc */
        ret = -ENOMEM;
-   bits = kmalloc(6 * size, GFP_KERNEL);
+   bits = kmalloc(6 * size, GFP_KERNEL_BC);
        if (!bits)
            goto out_nofds;
    }
@@ -687,7 +688,7 @@ int do_sys_poll(struct pollfd __user *uf
    if (!stack_pp)
        stack_pp = pp = (struct poll_list *)stack_pps;
    else {
-   pp = kmalloc(size, GFP_KERNEL);
+   pp = kmalloc(size, GFP_KERNEL_BC);
        if (!pp)
            goto out_fds;
    }
--- ./include/asm-i386/thread_info.h.bckmem 2006-11-09 11:29:12.000000000 +0300
+++ ./include/asm-i386/thread_info.h 2006-11-09 11:33:27.000000000 +0300
@@ -99,13 +99,13 @@ static inline struct thread_info *current
({
    \
    struct thread_info *ret; \
    \
-   ret = kmalloc(THREAD_SIZE, GFP_KERNEL); \
+   ret = kmalloc(THREAD_SIZE, GFP_KERNEL_BC); \
    if (ret) \
        memset(ret, 0, THREAD_SIZE); \
    ret; \
})
#else
-#define alloc_thread_info(tsk) kmalloc(THREAD_SIZE, GFP_KERNEL)
+#define alloc_thread_info(tsk) kmalloc(THREAD_SIZE, GFP_KERNEL_BC)
#endif

#define free_thread_info(info) kfree(info)
--- ./include/asm-ia64/pgalloc.h.bckmem 2006-09-20 14:46:38.000000000 +0400
+++ ./include/asm-ia64/pgalloc.h 2006-11-09 11:33:27.000000000 +0300
@@ -19,6 +19,8 @@
#include <linux/page-flags.h>
#include <linux/threads.h>

+#include <bc/kmem.h>
+
#include <asm/mmu_context.h>

DECLARE_PER_CPU(unsigned long *, __pgtable_quicklist);
@@ -37,7 +39,7 @@ static inline long pgtable_quicklist_tot

```

```

    return ql_size;
}

```

```

-static inline void *pgtable_quicklist_alloc(void)
+static inline void *pgtable_quicklist_alloc(int charge)
{
    unsigned long *ret = NULL;

```

```

@@ -45,13 +47,20 @@ static inline void *pgtable_quicklist_al

```

```

    ret = pgtable_quicklist;
    if (likely(ret != NULL)) {
+   if (charge && bc_page_charge(virt_to_page(ret),
+   0, __GFP_BC_LIMIT)) {
+   ret = NULL;
+   goto out;
+   }
    pgtable_quicklist = (unsigned long *)(*ret);
    ret[0] = 0;
    --pgtable_quicklist_size;
+out:
    preempt_enable();
    } else {
    preempt_enable();
-   ret = (unsigned long *)__get_free_page(GFP_KERNEL | __GFP_ZERO);
+   ret = (unsigned long *)__get_free_page(GFP_KERNEL |
+   __GFP_ZERO | __GFP_BC | __GFP_BC_LIMIT);
    }

```

```

    return ret;
@@ -69,6 +78,7 @@ static inline void pgtable_quicklist_fre
#endif

```

```

    preempt_disable();
+   bc_page_uncharge(virt_to_page(pgtable_entry), 0);
    *(unsigned long *)pgtable_entry = (unsigned long)pgtable_quicklist;
    pgtable_quicklist = (unsigned long *)pgtable_entry;
    ++pgtable_quicklist_size;

```

```

@@ -77,7 +87,7 @@ static inline void pgtable_quicklist_fre

```

```

static inline pgd_t *pgd_alloc(struct mm_struct *mm)
{
-   return pgtable_quicklist_alloc();
+   return pgtable_quicklist_alloc(1);
}

```

```

static inline void pgd_free(pgd_t *pgd)
@@ -94,7 +104,7 @@ pgd_populate(struct mm_struct *mm, pgd_t

```

```

static inline pud_t *pud_alloc_one(struct mm_struct *mm, unsigned long addr)
{
- return pgtable_quicklist_alloc();
+ return pgtable_quicklist_alloc(1);
}

static inline void pud_free(pud_t * pud)
@@ -112,7 +122,7 @@ pud_populate(struct mm_struct *mm, pud_t

static inline pmd_t *pmd_alloc_one(struct mm_struct *mm, unsigned long addr)
{
- return pgtable_quicklist_alloc();
+ return pgtable_quicklist_alloc(1);
}

static inline void pmd_free(pmd_t * pmd)
@@ -137,13 +147,13 @@ pmd_populate_kernel(struct mm_struct *mm
static inline struct page *pte_alloc_one(struct mm_struct *mm,
    unsigned long addr)
{
- return virt_to_page(pgtable_quicklist_alloc());
+ return virt_to_page(pgtable_quicklist_alloc(1));
}

static inline pte_t *pte_alloc_one_kernel(struct mm_struct *mm,
    unsigned long addr)
{
- return pgtable_quicklist_alloc();
+ return pgtable_quicklist_alloc(0);
}

static inline void pte_free(struct page *pte)
--- ./include/asm-x86_64/pgalloc.h.bckmem 2006-09-20 14:46:40.000000000 +0400
+++ ./include/asm-x86_64/pgalloc.h 2006-11-09 11:33:27.000000000 +0300
@@ -31,12 +31,14 @@ static inline void pmd_free(pmd_t *pmd)

static inline pmd_t *pmd_alloc_one (struct mm_struct *mm, unsigned long addr)
{
- return (pmd_t *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT);
+ return (pmd_t *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT|
+ __GFP_BC | __GFP_BC_LIMIT);
}

static inline pud_t *pud_alloc_one(struct mm_struct *mm, unsigned long addr)
{
- return (pud_t *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT);
+ return (pud_t *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT|

```

```

+ __GFP_BC | __GFP_BC_LIMIT);
}

static inline void pud_free (pud_t *pud)
@@ -74,7 +76,8 @@ static inline void pgd_list_del(pgd_t *p
static inline pgd_t *pgd_alloc(struct mm_struct *mm)
{
    unsigned boundary;
- pgd_t *pgd = (pgd_t *)__get_free_page(GFP_KERNEL|__GFP_REPEAT);
+ pgd_t *pgd = (pgd_t *)__get_free_page(GFP_KERNEL|__GFP_REPEAT|
+ __GFP_BC | __GFP_BC_LIMIT);
    if (!pgd)
        return NULL;
    pgd_list_add(pgd);
@@ -105,7 +108,8 @@ static inline pte_t *pte_alloc_one_kerne

static inline struct page *pte_alloc_one(struct mm_struct *mm, unsigned long address)
{
- void *p = (void *)__get_zeroed_page(GFP_KERNEL|__GFP_REPEAT);
+ void *p = (void *)__get_zeroed_page(GFP_KERNEL|__GFP_REPEAT|
+ __GFP_BC | __GFP_BC_LIMIT);
    if (!p)
        return NULL;
    return virt_to_page(p);
--- ./include/asm-x86_64/thread_info.h.bckmem 2006-11-09 11:19:09.000000000 +0300
+++ ./include/asm-x86_64/thread_info.h 2006-11-09 11:33:27.000000000 +0300
@@ -78,14 +78,15 @@ static inline struct thread_info *stack_
    ({
        \
        struct thread_info *ret; \
        \
- ret = ((struct thread_info *)__get_free_pages(GFP_KERNEL,THREAD_ORDER)); \
+ ret = ((struct thread_info *)__get_free_pages(GFP_KERNEL_BC, \
+ THREAD_ORDER)); \
    if (ret) \
        memset(ret, 0, THREAD_SIZE); \
    ret; \
    })
#else
#define alloc_thread_info(tsk) \
- ((struct thread_info *)__get_free_pages(GFP_KERNEL,THREAD_ORDER))
+ ((struct thread_info *)__get_free_pages(GFP_KERNEL_BC,THREAD_ORDER))
#endif

#define free_thread_info(ti) free_pages((unsigned long) (ti), THREAD_ORDER)
--- ./ipc/msgutil.c.bckmem 2006-11-09 11:19:09.000000000 +0300
+++ ./ipc/msgutil.c 2006-11-09 11:33:27.000000000 +0300
@@ -36,7 +36,7 @@ struct msg_msg *load_msg(const void __us
    if (alen > DATALEN_MSG)

```

```

alen = DATALEN_MSG;

- msg = (struct msg_msg *)kmalloc(sizeof(*msg) + alen, GFP_KERNEL);
+ msg = (struct msg_msg *)kmalloc(sizeof(*msg) + alen, GFP_KERNEL_BC);
  if (msg == NULL)
    return ERR_PTR(-ENOMEM);

@@ -57,7 +57,7 @@ struct msg_msg *load_msg(const void __us
  if (alen > DATALEN_SEG)
    alen = DATALEN_SEG;
  seg = (struct msg_msgseg *)kmalloc(sizeof(*seg) + alen,
-   GFP_KERNEL);
+   GFP_KERNEL_BC);
  if (seg == NULL) {
    err = -ENOMEM;
    goto out_err;
--- ./ipc/sem.c.bckmem 2006-11-09 11:29:12.000000000 +0300
+++ ./ipc/sem.c 2006-11-09 11:33:27.000000000 +0300
@@ -1009,7 +1009,7 @@ static inline int get_undo_list(struct s

  undo_list = current->sysvsem.undo_list;
  if (!undo_list) {
-   undo_list = kzalloc(sizeof(*undo_list), GFP_KERNEL);
+   undo_list = kzalloc(sizeof(*undo_list), GFP_KERNEL_BC);
    if (undo_list == NULL)
      return -ENOMEM;
    spin_lock_init(&undo_list->lock);
@@ -1072,7 +1072,8 @@ static struct sem_undo *find_undo(struct
  ipc_rcu_getref(sma);
  sem_unlock(sma);

-   new = (struct sem_undo *) kmalloc(sizeof(struct sem_undo) + sizeof(short)*nsems,
GFP_KERNEL);
+   new = (struct sem_undo *) kmalloc(sizeof(struct sem_undo) +
+   sizeof(short)*nsems, GFP_KERNEL_BC);
  if (!new) {
    ipc_lock_by_ptr(&sma->sem_perm);
    ipc_rcu_putref(sma);
@@ -1133,7 +1134,7 @@ asmlinkage long sys_sem timedop(int semid
  if (nsops > ns->sc_semopm)
    return -E2BIG;
  if (nsops > SEMOPM_FAST) {
-   sops = kmalloc(sizeof(*sops)*nsops, GFP_KERNEL);
+   sops = kmalloc(sizeof(*sops)*nsops, GFP_KERNEL_BC);
    if (sops == NULL)
      return -ENOMEM;
  }
--- ./ipc/util.c.bckmem 2006-11-09 11:22:58.000000000 +0300

```

```

+++ ./ipc/util.c 2006-11-09 11:33:27.000000000 +0300
@@ -406,9 +406,9 @@ void* ipc_alloc(int size)
{
    void* out;
    if(size > PAGE_SIZE)
-   out = vmalloc(size);
+   out = vmalloc_bc(size);
    else
-   out = kmalloc(size, GFP_KERNEL);
+   out = kmalloc(size, GFP_KERNEL_BC);
    return out;
}

@@ -491,14 +491,14 @@ void* ipc_rcu_alloc(int size)
    * workqueue if necessary (for vmalloc).
    */
    if (rcu_use_vmalloc(size)) {
-   out = vmalloc(HDRLEN_VMALLOC + size);
+   out = vmalloc_bc(HDRLEN_VMALLOC + size);
        if (out) {
            out += HDRLEN_VMALLOC;
            container_of(out, struct ipc_rcu_hdr, data)->is_vmalloc = 1;
            container_of(out, struct ipc_rcu_hdr, data)->refcount = 1;
        }
    } else {
-   out = kmalloc(HDRLEN_KMALLOC + size, GFP_KERNEL);
+   out = kmalloc(HDRLEN_KMALLOC + size, GFP_KERNEL_BC);
        if (out) {
            out += HDRLEN_KMALLOC;
            container_of(out, struct ipc_rcu_hdr, data)->is_vmalloc = 0;
--- ./kernel/fork.c.bckmem 2006-11-09 11:32:24.000000000 +0300
+++ ./kernel/fork.c 2006-11-09 11:33:27.000000000 +0300
@@ -143,7 +143,7 @@ void __init fork_init(unsigned long memp
/* create a slab on which task_structs can be allocated */
task_struct_cachep =
    kmem_cache_create("task_struct", sizeof(struct task_struct),
-   ARCH_MIN_TASKALIGN, SLAB_PANIC, NULL, NULL);
+   ARCH_MIN_TASKALIGN, SLAB_PANIC | SLAB_BC, NULL, NULL);
#ifdef
/*
@@ -1441,23 +1441,24 @@ void __init proc_caches_init(void)
{
    sighand_cachep = kmem_cache_create("sighand_cache",
        sizeof(struct sighand_struct), 0,
-   SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_DESTROY_BY_RCU,
+   SLAB_HWCACHE_ALIGN | SLAB_PANIC | \
+   SLAB_DESTROY_BY_RCU | SLAB_BC,

```

```

    sighand_ctor, NULL);
    signal_cachep = kmem_cache_create("signal_cache",
        sizeof(struct signal_struct), 0,
-   SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+   SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_BC, NULL, NULL);
    files_cachep = kmem_cache_create("files_cache",
        sizeof(struct files_struct), 0,
-   SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+   SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_BC, NULL, NULL);
    fs_cachep = kmem_cache_create("fs_cache",
        sizeof(struct fs_struct), 0,
-   SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+   SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_BC, NULL, NULL);
    vm_area_cachep = kmem_cache_create("vm_area_struct",
        sizeof(struct vm_area_struct), 0,
-   SLAB_PANIC, NULL, NULL);
+   SLAB_PANIC|SLAB_BC, NULL, NULL);
    mm_cachep = kmem_cache_create("mm_struct",
        sizeof(struct mm_struct), ARCH_MIN_MMSTRUCT_ALIGN,
-   SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+   SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_BC, NULL, NULL);
}

```

```

--- ./kernel/posix-timers.c.bckmem 2006-11-09 11:29:12.000000000 +0300
+++ ./kernel/posix-timers.c 2006-11-09 11:33:27.000000000 +0300
@@ -242,7 +242,8 @@ static __init int init_posix_timers(void
    register_posix_clock(CLOCK_MONOTONIC, &clock_monotonic);

```

```

    posix_timers_cache = kmem_cache_create("posix_timers_cache",
-   sizeof (struct k_itimer), 0, 0, NULL, NULL);
+   sizeof (struct k_itimer), 0, SLAB_BC,
+   NULL, NULL);
    idr_init(&posix_timers_id);
    return 0;
}

```

```

--- ./kernel/signal.c.bckmem 2006-11-09 11:29:12.000000000 +0300
+++ ./kernel/signal.c 2006-11-09 11:33:27.000000000 +0300
@@ -2764,5 +2764,5 @@ void __init signals_init(void)
    kmem_cache_create("sigqueue",
        sizeof(struct sigqueue),
        __alignof__(struct sigqueue),
-   SLAB_PANIC, NULL, NULL);
+   SLAB_PANIC | SLAB_BC, NULL, NULL);
}

```

```

--- ./kernel/user.c.bckmem 2006-11-09 11:29:56.000000000 +0300
+++ ./kernel/user.c 2006-11-09 11:33:27.000000000 +0300
@@ -205,7 +205,7 @@ static int __init uid_cache_init(void)

```

```

int n;

uid_cachep = kmem_cache_create("uid_cache", sizeof(struct user_struct),
- 0, SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+ 0, SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_BC, NULL, NULL);

for(n = 0; n < UIDHASH_SZ; ++n)
    INIT_LIST_HEAD(uidhash_table + n);
--- ./mm/mempool.c.bckmem 2006-09-20 14:46:41.000000000 +0400
+++ ./mm/mempool.c 2006-11-09 11:33:27.000000000 +0300
@@ -119,6 +119,7 @@ int mempool_resize(mempool_t *pool, int
    unsigned long flags;

    BUG_ON(new_min_nr <= 0);
+ gfp_mask &= ~__GFP_BC;

    spin_lock_irqsave(&pool->lock, flags);
    if (new_min_nr <= pool->min_nr) {
@@ -212,6 +213,7 @@ void * mempool_alloc(mempool_t *pool, gf
    gfp_mask |= __GFP_NOMEMALLOC; /* don't allocate emergency reserves */
    gfp_mask |= __GFP_NORETRY; /* don't loop in __alloc_pages */
    gfp_mask |= __GFP_NOWARN; /* failures are OK */
+ gfp_mask &= ~__GFP_BC; /* do not charge */

    gfp_temp = gfp_mask & ~(__GFP_WAIT|__GFP_IO);

--- ./mm/page_alloc.c.bckmem 2006-11-09 11:29:12.000000000 +0300
+++ ./mm/page_alloc.c 2006-11-09 11:33:27.000000000 +0300
@@ -41,6 +41,8 @@
#include <linux/pfn.h>
#include <linux/backing-dev.h>

+#include <bc/kmem.h>
+
#include <asm/tlbflush.h>
#include <asm/div64.h>
#include "internal.h"
@@ -513,6 +515,8 @@ static void __free_pages_ok(struct page
    if (reserved)
        return;

+ bc_page_uncharge(page, order);
+
    if (!PageHighMem(page))
        debug_check_no_locks_freed(page_address(page), PAGE_SIZE<<order);
    arch_free_page(page, order);
@@ -800,6 +804,8 @@ static void fastcall free_hot_cold_page(
    if (free_pages_check(page))

```

```

return;

+ bc_page_uncharge(page, 0);
+
+ if (!PageHighMem(page))
+   debug_check_no_locks_freed(page_address(page), PAGE_SIZE);
+   arch_free_page(page, 0);
@@ -1337,6 +1343,11 @@ nopage:
+   show_mem();
+ }
+ got_pg:
+ if ((gfp_mask & __GFP_BC) &&
+   bc_page_charge(page, order, gfp_mask)) {
+   __free_pages(page, order);
+   page = NULL;
+ }
+ #ifdef CONFIG_PAGE_OWNER
+   if (page)
+     set_page_owner(page, order, gfp_mask);
--- ./mm/slab.c.bckmem 2006-11-09 11:33:17.000000000 +0300
+++ ./mm/slab.c 2006-11-09 11:33:27.000000000 +0300
@@ -1483,7 +1483,8 @@ void __init kmem_cache_init(void)
+   sizes[INDEX_AC].cs_cachep = kmem_cache_create(names[INDEX_AC].name,
+     sizes[INDEX_AC].cs_size,
+     ARCH_KMALLOC_MINALIGN,
-   ARCH_KMALLOC_FLAGS|SLAB_PANIC,
+   ARCH_KMALLOC_FLAGS | SLAB_BC |
+   SLAB_BC_NOCHARGE | SLAB_PANIC,
+   NULL, NULL);

+   if (INDEX_AC != INDEX_L3) {
@@ -1491,7 +1492,8 @@ void __init kmem_cache_init(void)
+   kmem_cache_create(names[INDEX_L3].name,
+     sizes[INDEX_L3].cs_size,
+     ARCH_KMALLOC_MINALIGN,
-   ARCH_KMALLOC_FLAGS|SLAB_PANIC,
+   ARCH_KMALLOC_FLAGS | SLAB_BC |
+   SLAB_BC_NOCHARGE | SLAB_PANIC,
+   NULL, NULL);
+ }

@@ -1509,7 +1511,8 @@ void __init kmem_cache_init(void)
+   sizes->cs_cachep = kmem_cache_create(names->name,
+     sizes->cs_size,
+     ARCH_KMALLOC_MINALIGN,
-   ARCH_KMALLOC_FLAGS|SLAB_PANIC,
+   ARCH_KMALLOC_FLAGS | SLAB_BC |
+   SLAB_BC_NOCHARGE | SLAB_PANIC,

```

```

    NULL, NULL);
}
if (CONFIG_ZONE_DMA_FLAG)
@@ -3168,6 +3171,19 @@ static inline void *__cache_alloc(stru
    return objp;
}

+static inline int bc_should_charge(kmem_cache_t *cachep, gfp_t flags)
+{
+ifdef CONFIG_BEANCOUNTERS
+ if (!(cachep->flags & SLAB_BC))
+ return 0;
+ if (flags & __GFP_BC)
+ return 1;
+ if (!(cachep->flags & SLAB_BC_NOCHARGE))
+ return 1;
+endif
+ return 0;
+}
+
static __always_inline void *__cache_alloc(struct kmem_cache *cachep,
    gfp_t flags, void *caller)
{
@@ -3193,6 +3209,12 @@ static __always_inline void *__cache_all
    local_irq_restore(save_flags);
    objp = cache_alloc_debugcheck_after(cachep, flags, objp,
        caller);
+
+ if (objp && bc_should_charge(cachep, flags))
+ if (bc_slab_charge(cachep, objp, flags)) {
+ kmem_cache_free(cachep, objp);
+ objp = NULL;
+ }
    prefetchw(objp);
    return objp;
}
@@ -3419,6 +3441,8 @@ static inline void __cache_free(struct k
    struct array_cache *ac = cpu_cache_get(cachep);

    check_irq_off();
+ if (cachep->flags & SLAB_BC)
+ bc_slab_uncharge(cachep, objp);
    objp = cache_free_debugcheck(cachep, objp, __builtin_return_address(0));

    if (cache_free_alien(cachep, objp))

```
