Subject: [PATCH 5/6] BC: kernel memory accounting (core)
Posted by dev on Wed, 23 Aug 2006 11:04:41 GMT
View Forum Message <> Reply to Message

Introduce BC_KMEMSIZE resource which accounts kernel
objects allocated by task's request.

Reference to BC is kept on struct page or slab object.
For slabs each struct slab contains a set of pointers
corresponding objects are charged to.

Allocation charge rules:
1. Pages - if allocation is performed with __GFP_BC flag - page
   is charged to current's exec_bc.
2. Slabs - kmem_cache may be created with SLAB_BC flag - in this
   case each allocation is charged. Caches used by kmalloc are
   created with SLAB_BC | SLAB_BC_NOCHARGE flags. In this case
   only __GFP_BC allocations are charged.

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

---

 include/linux/gfp.h | 8 ++-
 include/linux/mm.h | 6 ++
 include/linux/slab.h | 4 +
 include/linux/vmalloc.h | 1
 include/bc/beancounter.h | 4 +
 include/bc/kmem.h | 33 ++++++++++++
 kernel/bc/Makefile | 1
 kernel/bc/beancounter.c | 3 +
 kernel/bc/kmem.c | 89 ++
 mm/mempool.c | 2
 mm/page_alloc.c | 11 ++++
 mm/slab.c | 121 ++---------
 mm/vmalloc.c | 6 ++
 13 files changed, 264 insertions(+), 25 deletions(-)

--- ./include/linux/gfp.h.bckmem 2006-07-28 18:43:52.000000000 +0400
+++ ./include/linux/gfp.h 2006-07-31 16:32:22.000000000 +0400
@@ -46,8 +46,10 @@ struct vm_area_struct;
 #define __GFP_NOMEMALLOC ((__force gfp_t)0x10000u) /* Don't use emergency reserves */
 #define __GFP_HARDWALL   ((__force gfp_t)0x20000u) /* Enforce hardwall cpuset memory
allocs */
 #define __GFP_THISNODE ((__force gfp_t)0x40000u)/* No fallback, no policies */
+#define __GFP_BC ((__force gfp_t)0x80000u) /* Charge allocation with BC */
+#define __GFP_BC_LIMIT ((__force gfp_t)0x100000u) /* Charge against BC limit */

```
-#define __GFP_BITS_SHIFT 20 /* Room for 20 __GFP_FOO bits */
+#define __GFP_BITS_SHIFT 21 /* Room for 21 __GFP_FOO bits */
 #define __GFP_BITS_MASK ((__force gfp_t)((1 << __GFP_BITS_SHIFT) - 1))

 /* if you forget to add the bitmask here kernel will crash, period */
@@ -54,7 +56,8 @@ struct vm_area_struct;
 #define GFP_LEVEL_MASK (__GFP_WAIT|__GFP_HIGH|__GFP_IO|__GFP_FS| \
    __GFP_COLD|__GFP_NOWARN|__GFP_REPEAT| \
    __GFP_NOFAIL|__GFP_NORETRY|__GFP_NO_GROW|__GFP_COMP| \
-   __GFP_NOMEMALLOC|__GFP_HARDWALL|__GFP_THISNODE)
+   __GFP_NOMEMALLOC|__GFP_HARDWALL|__GFP_THISNODE| \
+   __GFP_BC|__GFP_BC_LIMIT)

 /* This equals 0, but use constants in case they ever change */
 #define GFP_NOWAIT (GFP_ATOMIC & ~__GFP_HIGH)
@@ -63,6 +66,7 @@ struct vm_area_struct;
 #define GFP_NOIO (__GFP_WAIT)
 #define GFP_NOFS (__GFP_WAIT | __GFP_IO)
 #define GFP_KERNEL (__GFP_WAIT | __GFP_IO | __GFP_FS)
+#define GFP_KERNEL_BC (__GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_BC)
 #define GFP_USER (__GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_HARDWALL)
 #define GFP_HIGHUSER (__GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_HARDWALL | \
     __GFP_HIGHMEM)
--- ./include/linux/mm.h.bckmem 2006-07-28 18:43:52.000000000 +0400
+++ ./include/linux/mm.h 2006-07-31 17:50:29.000000000 +0400
@@ -267,8 +267,14 @@ struct page {
  unsigned int gfp_mask;
  unsigned long trace[8];
 #endif
+#ifdef CONFIG_BEANCOUNTERS
+ union {
+  struct beancounter *page_bc;
+ } bc;
+#endif
 };

+#define page_bc(page)   ((page)->bc.page_bc)
 #define page_private(page)  ((page)->private)
 #define set_page_private(page, v) ((page)->private = (v))

--- ./include/linux/slab.h.bckmem 2006-07-10 12:39:19.000000000 +0400
+++ ./include/linux/slab.h 2006-07-31 17:02:08.000000000 +0400
@@ -46,6 +46,8 @@ typedef struct kmem_cache kmem_cache_t;
 #define SLAB_PANIC  0x00040000UL /* panic if kmem_cache_create() fails */
 #define SLAB_DESTROY_BY_RCU 0x00080000UL /* defer freeing pages to RCU */
 #define SLAB_MEM_SPREAD  0x00100000UL /* Spread some memory over cpuset */
+#define SLAB_BC  0x00200000UL /* Account with BC */
```

```
+#define SLAB_BC_NOCHARGE 0x00400000UL /* Explicit accounting */

 /* flags passed to a constructor func */
 #define SLAB_CTOR_CONSTRUCTOR 0x001UL  /* if not set, then deconstructor */
@@ -265,6 +267,8 @@ extern kmem_cache_t *bio_cachep;

 extern atomic_t slab_reclaim_pages;

+struct beancounter;
+struct beancounter **kmem_cache_bcp(kmem_cache_t *cachep, void *obj);
 #endif /* __KERNEL__ */

 #endif /* _LINUX_SLAB_H */
--- ./include/linux/vmalloc.h.bckmem 2006-07-17 17:01:12.000000000 +0400
+++ ./include/linux/vmalloc.h 2006-08-01 13:21:59.000000000 +0400
@@ -36,6 +36,7 @@ struct vm_struct {
  * Highlevel APIs for driver use
  */
 extern void *vmalloc(unsigned long size);
+extern void *vmalloc_bc(unsigned long size);
 extern void *vmalloc_user(unsigned long size);
 extern void *vmalloc_node(unsigned long size, int node);
 extern void *vmalloc_exec(unsigned long size);
--- ./include/bc/beancounter.h.bckmem 2006-07-28 18:43:52.000000000 +0400
+++ ./include/bc/beancounter.h 2006-08-03 16:03:01.000000000 +0400
@@ -14,7 +14,9 @@
  * Resource list.
  */

-#define BC_RESOURCES 0
+#define BC_KMEMSIZE 0
+
+#define BC_RESOURCES 1

 struct resource_parm {
  /*
--- ./include/bc/kmem.h.bckmem 2006-07-28 18:43:52.000000000 +0400
+++ ./include/bc/kmem.h 2006-07-31 17:37:05.000000000 +0400
@@ -0,0 +1,33 @@
+/*
+ *  include/bc/kmem.h
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#ifndef __BC_KMEM_H_
+#define __BC_KMEM_H_
```

```
+
+#include <linux/config.h>
+
+/*
+ * BC_KMEMSIZE accounting
+ */
+
+struct mm_struct;
+struct page;
+struct beancounter;
+
+#ifdef CONFIG_BEANCOUNTERS
+int  bc_page_charge(struct page *page, int order, gfp_t flags);
+void bc_page_uncharge(struct page *page, int order);
+
+int bc_slab_charge(kmem_cache_t *cachep, void *obj, gfp_t flags);
+void bc_slab_uncharge(kmem_cache_t *cachep, void *obj);
+#else
+#define bc_page_charge(pg, o, mask) (0)
+#define bc_page_uncharge(pg, o)  do { } while (0)
+#define bc_slab_charge(cachep, o, f) (0)
+#define bc_slab_uncharge(cachep, o) do { } while (0)
+#endif
+#endif /* __BC_SLAB_H_ */
--- ./kernel/bc/Makefile.bcsys 2006-07-28 14:08:37.000000000 +0400
+++ ./kernel/bc/Makefile 2006-08-01 11:08:39.000000000 +0400
@@ -7,3 +7,4 @@
 obj-$(CONFIG_BEANCOUNTERS) += beancounter.o
 obj-$(CONFIG_BEANCOUNTERS) += misc.o
 obj-$(CONFIG_BEANCOUNTERS) += sys.o
+obj-$(CONFIG_BEANCOUNTERS) += kmem.o
--- ./kernel/bc/beancounter.c.bckmem 2006-07-28 18:43:52.000000000 +0400
+++ ./kernel/bc/beancounter.c 2006-08-03 16:14:17.000000000 +0400
@@ -19,6 +19,7 @@ static void init_beancounter_struct(stru
 struct beancounter init_bc;

 const char *bc_rnames[] = {
+ "kmemsize", /* 0 */
 };

 #define bc_hash_fun(x) ((((x) >> 8) ^ (x)) & (BC_HASH_SIZE - 1))
@@ -348,6 +378,8 @@ static void init_beancounter_syslimits(s
 {
  int k;

+ bc->bc_parms[BC_KMEMSIZE].limit = 32 * 1024 * 1024;
+
  for (k = 0; k < BC_RESOURCES; k++)
```

```
  bc->bc_parms[k].barrier = bc->bc_parms[k].limit;
 }
--- ./kernel/bc/kmem.c.bckmem 2006-07-31 16:32:22.000000000 +0400
+++ ./kernel/bc/kmem.c 2006-07-31 17:51:27.000000000 +0400
@@ -0,0 +1,89 @@
+/*
+ *  kernel/bc/kmem.c
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#include <linux/sched.h>
+#include <linux/gfp.h>
+#include <linux/slab.h>
+#include <linux/mm.h>
+
+#include <bc/beancounter.h>
+#include <bc/kmem.h>
+#include <bc/task.h>
+
+/*
+ * Slab accounting
+ */
+
+int bc_slab_charge(kmem_cache_t *cachep, void *objp, gfp_t flags)
+{
+ unsigned int size;
+ struct beancounter *bc, **slab_bcp;
+
+ bc = get_exec_bc();
+ if (bc == NULL)
+  return 0;
+
+ size = kmem_cache_size(cachep);
+ if (bc_charge(bc, BC_KMEMSIZE, size,
+    (flags & __GFP_BC_LIMIT ? BC_LIMIT : BC_BARRIER)))
+  return -ENOMEM;
+
+ slab_bcp = kmem_cache_bcp(cachep, objp);
+ *slab_bcp = get_beancounter(bc);
+ return 0;
+}
+
+void bc_slab_uncharge(kmem_cache_t *cachep, void *objp)
+{
+ unsigned int size;
+ struct beancounter *bc, **slab_bcp;
```

```
+
+ slab_bcp = kmem_cache_bcp(cachep, objp);
+ if (*slab_bcp == NULL)
+  return;
+
+ bc = *slab_bcp;
+ size = kmem_cache_size(cachep);
+ bc_uncharge(bc, BC_KMEMSIZE, size);
+ put_beancounter(bc);
+ *slab_bcp = NULL;
+}
+
+/*
+ * Pages accounting
+ */
+
+int bc_page_charge(struct page *page, int order, gfp_t flags)
+{
+ struct beancounter *bc;
+
+ BUG_ON(page_bc(page) != NULL);
+
+ bc = get_exec_bc();
+ if (bc == NULL)
+  return 0;
+
+ if (bc_charge(bc, BC_KMEMSIZE, PAGE_SIZE << order,
+   (flags & __GFP_BC_LIMIT ? BC_LIMIT : BC_BARRIER)))
+  return -ENOMEM;
+
+ page_bc(page) = get_beancounter(bc);
+ return 0;
+}
+
+void bc_page_uncharge(struct page *page, int order)
+{
+ struct beancounter *bc;
+
+ bc = page_bc(page);
+ if (bc == NULL)
+  return;
+
+ bc_uncharge(bc, BC_KMEMSIZE, PAGE_SIZE << order);
+ put_beancounter(bc);
+ page_bc(page) = NULL;
+}
--- ./mm/mempool.c.bckmem 2006-04-21 11:59:36.000000000 +0400
+++ ./mm/mempool.c 2006-08-01 13:25:26.000000000 +0400
```

```
@@ -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 2006-07-28 18:43:52.000000000 +0400
+++ ./mm/page_alloc.c 2006-07-31 19:52:08.000000000 +0400
@@ -38,6 +38,8 @@
 #include <linux/mempolicy.h>
 #include <linux/stop_machine.h>

+#include <bc/kmem.h>
+
 #include <asm/tlbflush.h>
 #include <asm/div64.h>
 #include "internal.h"
@@ -453,6 +455,8 @@ static void __free_pages_ok(struct page
  if (reserved)
   return;

+ bc_page_uncharge(page, order);
+
  kernel_map_pages(page, 1 << order, 0);
  local_irq_save(flags);
   __count_vm_events(PGFREE, 1 << order);
@@ -724,6 +728,8 @@ static void fastcall free_hot_cold_page(
  if (free_pages_check(page))
   return;

+ bc_page_uncharge(page, 0);
+
  kernel_map_pages(page, 1, 0);

  pcp = &zone_pcp(zone, get_cpu())->pcp[cold];
@@ -1056,6 +1062,11 @@ 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 2006-07-17 17:01:12.000000000 +0400
+++ ./mm/slab.c 2006-08-01 13:24:06.000000000 +0400
@@ -109,6 +109,8 @@
 #include <linux/mutex.h>
 #include <linux/rtmutex.h>

+#include <bc/kmem.h>
+
 #include <asm/uaccess.h>
 #include <asm/cacheflush.h>
 #include <asm/tlbflush.h>
@@ -176,11 +178,13 @@
     SLAB_CACHE_DMA | \
     SLAB_MUST_HWCACHE_ALIGN | SLAB_STORE_USER | \
     SLAB_RECLAIM_ACCOUNT | SLAB_PANIC | \
+    SLAB_BC | SLAB_BC_NOCHARGE | \
     SLAB_DESTROY_BY_RCU | SLAB_MEM_SPREAD)
 #else
 # define CREATE_MASK (SLAB_HWCACHE_ALIGN | \
     SLAB_CACHE_DMA | SLAB_MUST_HWCACHE_ALIGN | \
     SLAB_RECLAIM_ACCOUNT | SLAB_PANIC | \
+    SLAB_BC | SLAB_BC_NOCHARGE | \
     SLAB_DESTROY_BY_RCU | SLAB_MEM_SPREAD)
 #endif

@@ -774,9 +778,33 @@ struct kmem_cache *kmem_find_general_cac
  return __find_general_cachep(size, gfpflags);
 }

-static size_t slab_mgmt_size(size_t nr_objs, size_t align)
+static size_t slab_mgmt_size_raw(size_t nr_objs)
 {
- return ALIGN(sizeof(struct slab)+nr_objs*sizeof(kmem_bufctl_t), align);
+ return sizeof(struct slab) + nr_objs * sizeof(kmem_bufctl_t);
+}
+
+#ifdef CONFIG_BEANCOUNTERS
+#define BC_EXTRASIZE sizeof(struct beancounter *)
+static inline size_t slab_mgmt_size_noalign(int flags, size_t nr_objs)
```

```
+{
+ size_t size;
+
+ size = slab_mgmt_size_raw(nr_objs);
+ if (flags & SLAB_BC)
+  size = ALIGN(size, BC_EXTRASIZE) + nr_objs * BC_EXTRASIZE;
+ return size;
+}
+#else
+#define BC_EXTRASIZE 0
+static inline size_t slab_mgmt_size_noalign(int flags, size_t nr_objs)
+{
+ return slab_mgmt_size_raw(nr_objs);
+}
+#endif
+
+static inline size_t slab_mgmt_size(int flags, size_t nr_objs, size_t align)
+{
+ return ALIGN(slab_mgmt_size_noalign(flags, nr_objs), align);
 }

 /*
@@ -821,20 +849,21 @@ static void cache_estimate(unsigned long
    * into account.
    */
   nr_objs = (slab_size - sizeof(struct slab)) /
-     (buffer_size + sizeof(kmem_bufctl_t));
+     (buffer_size + sizeof(kmem_bufctl_t) +
+     (flags & SLAB_BC ? BC_EXTRASIZE : 0));

   /*
    * This calculated number will be either the right
    * amount, or one greater than what we want.
    */
-  if (slab_mgmt_size(nr_objs, align) + nr_objs*buffer_size
+  if (slab_mgmt_size(flags, nr_objs, align) + nr_objs*buffer_size
          > slab_size)
    nr_objs--;

   if (nr_objs > SLAB_LIMIT)
    nr_objs = SLAB_LIMIT;

-  mgmt_size = slab_mgmt_size(nr_objs, align);
+  mgmt_size = slab_mgmt_size(flags, nr_objs, align);
  }
  *num = nr_objs;
  *left_over = slab_size - nr_objs*buffer_size - mgmt_size;
@@ -1393,7 +1422,8 @@ 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) {
@@ -1401,7 +1431,8 @@ 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);
  }

@@ -1419,7 +1450,8 @@ 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);
   }

@@ -1869,7 +1901,8 @@ static size_t calculate_slab_order(struc
     * looping condition in cache_grow().
     */
    offslab_limit = size - sizeof(struct slab);
-   offslab_limit /= sizeof(kmem_bufctl_t);
+   offslab_limit /= (sizeof(kmem_bufctl_t) +
+     (flags & SLAB_BC ? BC_EXTRASIZE : 0));

    if (num > offslab_limit)
    break;
@@ -2170,8 +2203,8 @@ kmem_cache_create (const char *name, siz
   cachep = NULL;
   goto oops;
  }
- slab_size = ALIGN(cachep->num * sizeof(kmem_bufctl_t)
-     + sizeof(struct slab), align);
+
+ slab_size = slab_mgmt_size(flags, cachep->num, align);
```

```
  /*
   * If the slab has been placed off-slab, and we have enough space then
@@ -2182,11 +2215,9 @@ kmem_cache_create (const char *name, siz
   left_over -= slab_size;
  }

- if (flags & CFLGS_OFF_SLAB) {
+ if (flags & CFLGS_OFF_SLAB)
   /* really off slab. No need for manual alignment */
-  slab_size =
-      cachep->num * sizeof(kmem_bufctl_t) + sizeof(struct slab);
- }
+  slab_size = slab_mgmt_size_noalign(flags, cachep->num);

  cachep->colour_off = cache_line_size();
  /* Offset must be a multiple of the alignment. */
@@ -2435,6 +2466,30 @@ int kmem_cache_destroy(struct kmem_cache
 }
 EXPORT_SYMBOL(kmem_cache_destroy);

+static inline kmem_bufctl_t *slab_bufctl(struct slab *slabp)
+{
+ return (kmem_bufctl_t *) (slabp + 1);
+}
+
+#ifdef CONFIG_BEANCOUNTERS
+static inline struct beancounter **slab_bc_ptrs(kmem_cache_t *cachep,
+  struct slab *slabp)
+{
+ return (struct beancounter **) ALIGN((unsigned long)
+   (slab_bufctl(slabp) + cachep->num), BC_EXTRASIZE);
+}
+
+struct beancounter **kmem_cache_bcp(kmem_cache_t *cachep, void *objp)
+{
+ struct slab *slabp;
+ struct beancounter **bcs;
+
+ slabp = virt_to_slab(objp);
+ bcs = slab_bc_ptrs(cachep, slabp);
+ return bcs + obj_to_index(cachep, slabp, objp);
+}
+#endif
+
 /*
  * Get the memory for a slab management obj.
  * For a slab cache when the slab descriptor is off-slab, slab descriptors
@@ -2445,7 +2500,8 @@ static struct slab *alloc_slabmgmt(struc
```

```
  if (OFF_SLAB(cachep)) {
   /* Slab management obj is off-slab. */
   slabp = kmem_cache_alloc_node(cachep->slabp_cache,
-           local_flags, nodeid);
+           local_flags & (~__GFP_BC),
+           nodeid);
   if (!slabp)
    return NULL;
  } else {
@@ -2456,14 +2512,14 @@ static struct slab *alloc_slabmgmt(struc
  slabp->colouroff = colour_off;
  slabp->s_mem = objp + colour_off;
  slabp->nodeid = nodeid;
+#ifdef CONFIG_BEANCOUNTERS
+ if (cachep->flags & SLAB_BC)
+  memset(slab_bc_ptrs(cachep, slabp), 0,
+    cachep->num * BC_EXTRASIZE);
+#endif
  return slabp;
 }

-static inline kmem_bufctl_t *slab_bufctl(struct slab *slabp)
-{
- return (kmem_bufctl_t *) (slabp + 1);
-}
-
 static void cache_init_objs(struct kmem_cache *cachep,
        struct slab *slabp, unsigned long ctor_flags)
 {
@@ -2641,7 +2697,7 @@ static int cache_grow(struct kmem_cache
   * Get mem for the objs.  Attempt to allocate a physical page from
   * 'nodeid'.
   */
- objp = kmem_getpages(cachep, flags, nodeid);
+ objp = kmem_getpages(cachep, flags & (~__GFP_BC), nodeid);
  if (!objp)
   goto failed;

@@ -2989,6 +3045,19 @@ 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)
 {
@@ -3002,6 +3071,12 @@ 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;
 }
@@ -3193,6 +3266,8 @@ 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))
--- ./mm/vmalloc.c.bckmem 2006-07-17 17:01:12.000000000 +0400
+++ ./mm/vmalloc.c 2006-08-01 15:27:24.000000000 +0400
@@ -518,6 +518,12 @@ void *vmalloc(unsigned long size)
 }
 EXPORT_SYMBOL(vmalloc);

+void *vmalloc_bc(unsigned long size)
+{
+ return __vmalloc(size, GFP_KERNEL_BC | __GFP_HIGHMEM, PAGE_KERNEL);
+}
+EXPORT_SYMBOL(vmalloc_bc);
+
 /**
  * vmalloc_user  -  allocate virtually contiguous memory which has
  *      been zeroed so it can be mapped to userspace without
```