
Subject: [PATCH 6/13] BC: kernel memory (core)
Posted by [dev](#) on Tue, 05 Sep 2006 15:21:47 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/bc/beancounter.h | 4 +
include/bc/kmem.h        | 46 ++++++
include/linux/gfp.h      | 8 +-
include/linux/mm.h       | 4 +
include/linux/slab.h     | 4 +
include/linux/vmalloc.h | 1
kernel/bc/Makefile       | 1
kernel/bc/beancounter.c | 3 +
kernel/bc/kmem.c         | 85 ++++++
mm/mempool.c            | 2
mm/page_alloc.c         | 11 +++++
mm/slab.c               | 121 ++++++
mm/vmalloc.c            | 6 ++
13 files changed, 271 insertions(+), 25 deletions(-)
```

--- ./include/bc/beancounter.h.bckmemcore 2006-09-05 12:54:17.000000000 +0400

+++ ./include/bc/beancounter.h 2006-09-05 12:54:40.000000000 +0400

@@ -12,7 +12,9 @@

* Resource list.

*/

-#define BC_RESOURCES 0

+#define BC_KMEMSIZE 0

+

```
+ #define BC_RESOURCES 1
```

```
struct bc_resource_parm {
    unsigned long barrier; /* A barrier over which resource allocations
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/kmem.h 2006-09-05 12:54:40.000000000 +0400
@@ -0,0 +1,46 @@
+/*
+ * include/bc/kmem.h
+ *
+ * Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+ #ifndef __BC_KMEM_H_
+ #define __BC_KMEM_H_
+
+ /*
+ * BC_KMEMSIZE accounting
+ */
+
+ struct mm_struct;
+ struct page;
+ struct beancounter;
+
+ #ifdef CONFIG_BEANCOUNTERS
+ int __must_check bc_page_charge(struct page *page, int order, gfp_t flags);
+ void bc_page_uncharge(struct page *page, int order);
+
+ int __must_check bc_slab_charge(kmem_cache_t *cachep, void *obj, gfp_t flags);
+ void bc_slab_uncharge(kmem_cache_t *cachep, void *obj);
+ #else
+ static inline int __must_check bc_page_charge(struct page *page,
+ int order, gfp_t flags)
+ {
+     return 0;
+ }
+
+ static inline void bc_page_uncharge(struct page *page, int order)
+ {
+ }
+
+ static inline int __must_check bc_slab_charge(kmem_cache_t *cachep,
+ void *obj, gfp_t flags)
+ {
+     return 0;
+ }
+
+ 
```

```
+static inline void bc_slab_uncharge(kmem_cache_t *cachep, void *obj)
+{
+}
+##endif
+##endif /* __BC_SLAB_H_ */
--- ./include/linux/gfp.h.bckmemcore 2006-09-05 12:53:55.000000000 +0400
+++ ./include/linux/gfp.h 2006-09-05 12:54:40.000000000 +0400
@@ -46,15 +46,18 @@ 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 */
#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.bckmemcore 2006-09-05 12:53:55.000000000 +0400
+++ ./include/linux/mm.h 2006-09-05 12:55:28.000000000 +0400
@@ -274,8 +274,12 @@ struct page {
    unsigned int gfp_mask;
    unsigned long trace[8];
+##endif
+#ifdef CONFIG_BEANCOUNTERS
+ struct beancounter *page_bc;
+##endif
};
```

```

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

--- ./include/linux/slab.h.bckmemcore 2006-09-05 12:53:59.000000000 +0400
+++ ./include/linux/slab.h 2006-09-05 12:54:40.000000000 +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 */
@@ -291,6 +293,8 @@ extern kmem_cache_t *fs_cachep;
extern kmem_cache_t *sighand_cachep;
extern kmem_cache_t *bio_cachep;

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

#endif /* _LINUX_SLAB_H */
--- ./include/linux/vmalloc.h.bckmemcore 2006-09-05 12:53:59.000000000 +0400
+++ ./include/linux/vmalloc.h 2006-09-05 12:54:40.000000000 +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);
--- ./kernel/bc/Makefile.bckmemcore 2006-09-05 12:54:24.000000000 +0400
+++ ./kernel/bc/Makefile 2006-09-05 12:54:50.000000000 +0400
@@ -7,3 +7,4 @@
obj-y += beancounter.o
obj-y += misc.o
obj-y += sys.o
+obj-y += kmem.o
--- ./kernel/bc/beancounter.c.bckmemcore 2006-09-05 12:54:21.000000000 +0400
+++ ./kernel/bc/beancounter.c 2006-09-05 12:55:13.000000000 +0400
@@ -20,6 +20,7 @@ static void init_beancounter_struct(stru
struct beancounter init_bc;

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

```

```

};

#define BC_HASH_BITS 8
@@ -230,6 +231,8 @@ 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;
}
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/kmem.c 2006-09-05 12:54:40.000000000 +0400
@@ -0,0 +1,85 @@
+/*
+ * 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();
+
+    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_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.bckmemcore 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/mempool.c 2006-09-05 12:54:40.000000000 +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.bckmemcore 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/page_alloc.c 2006-09-05 12:54:40.000000000 +0400
@@ -40,6 +40,8 @@
#include <linux/sort.h>
#include <linux/pfn.h>

+#include <bc/kmem.h>
+
#include <asm/tlbflush.h>
#include <asm/div64.h>
#include "internal.h"
@@ -516,6 +518,8 @@ 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);
@@ -799,6 +803,8 @@ 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];
@@ -1188,6 +1194,11 @@ 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.bckmemcore 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/slab.c 2006-09-05 12:54:40.000000000 +0400
@@ -108,6 +108,8 @@
#include <linux/mutex.h>
#include <linux/rtmutex.h>

+#include <bc/kmem.h>
+
#include <asm/uaccess.h>
#include <asm/cacheflush.h>
#include <asm/tlbflush.h>
@@ -175,11 +177,13 @@
    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

@@ -793,9 +797,33 @@ static struct kmem_cache *kmem_find_gene
    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);

```



```

- 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;
@@ -1412,7 +1441,8 @@ 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) {
@@ -1420,7 +1450,8 @@ 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);
}

@@ -1438,7 +1469,8 @@ 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);
}

@@ -1941,7 +1973,8 @@ static size_t calculate_slab_order(struct
    * 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;
@@ -2249,8 +2282,8 @@ kmem_cache_create (const char *name, siz
    cachep = NULL;
    goto oops;

```



```

+ #endif
+
+ /*
+  * Get the memory for a slab management obj.
+  * For a slab cache when the slab descriptor is off-slab, slab descriptors
@@ -2529,7 +2584,8 @@ static struct slab *alloc_slabmgmt(struct
+ 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 {
@@ -2540,14 +2596,14 @@ static struct slab *alloc_slabmgmt(struct
+     slabp->colour_off = 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)
{
@@ -2725,7 +2781,7 @@ 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;

@@ -3073,6 +3129,6 @@ 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)
+{
@@ -3086,6 +3155,12 @@ 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;
+}
@@ -3283,6 +3358,8 @@ 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.bckmemcore 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/vmalloc.c 2006-09-05 12:54:40.000000000 +0400
@@ -520,6 +520,12 @@ 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
```
