Subject: [PATCH 6/13] BC: kmemsize accounting (core)
Posted by dev on Thu, 09 Nov 2006 16:53:38 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/kmem.h | 48 ++++++++++++++++++++
 include/linux/gfp.h | 8 ++-
 include/linux/mm.h | 1
 include/linux/mm_types.h | 3 +
 include/linux/slab.h | 4 +
 include/linux/vmalloc.h | 1
 kernel/bc/kmem.c | 112 ++
 mm/slab.c | 91 ++++++++++++++++++++++++++++++++--------
 mm/vmalloc.c | 6 ++
 9 files changed, 253 insertions(+), 21 deletions(-)

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/kmem.h 2006-11-03 15:48:26.000000000 +0300
@@ -0,0 +1,48 @@
+/*
+ * include/bc/kmem.h
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#ifndef __BC_KMEM_H_
+#define __BC_KMEM_H_
+

```
+/*
+ * BC_KMEMSIZE accounting
+ */
+
+#include <linux/slab.h>
+#include <linux/gfp.h>
+
+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.bc 2006-11-03 15:35:28.000000000 +0300
+++ ./include/linux/gfp.h 2006-11-03 15:48:26.000000000 +0300
@@ -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.bc 2006-11-03 15:35:28.000000000 +0300
+++ ./include/linux/mm.h 2006-11-03 15:48:26.000000000 +0300
@@ -219,6 +222,7 @@ struct vm_operations_struct {
 struct mmu_gather;
 struct inode;

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

--- ./include/linux/mm_types.h.bc 2006-11-03 15:35:28.000000000 +0300
+++ ./include/linux/mm_types.h 2006-11-03 15:48:26.000000000 +0300
@@ -62,6 +62,9 @@ struct page {
  void *virtual;   /* Kernel virtual address (NULL if
        not kmapped, ie. highmem) */
 #endif /* WANT_PAGE_VIRTUAL */
+#ifdef CONFIG_BEANCOUNTERS
+ struct beancounter *bc;
+#endif
 #ifdef CONFIG_PAGE_OWNER
  int order;
  unsigned int gfp_mask;
--- ./include/linux/slab.h.bc 2006-11-03 15:35:28.000000000 +0300
+++ ./include/linux/slab.h 2006-11-03 15:48:26.000000000 +0300
@@ -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 */
@@ -305,6 +307,8 @@ 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.bc 2006-11-03 15:35:28.000000000 +0300
+++ ./include/linux/vmalloc.h 2006-11-03 15:48:26.000000000 +0300
@@ -37,6 +37,7 @@ 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);
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/kmem.c 2006-11-03 15:48:26.000000000 +0300
@@ -0,0 +1,112 @@
+/*
+ * 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>
+
+#define BC_KMEMSIZE_BARRIER (64 * 1024)
+#define BC_KMEMSIZE_LIMIT (64 * 1024)
+
```

```
+/*
+ * 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);
+ BUG_ON(*slab_bcp != NULL);
+ *slab_bcp = bc_get(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);
+ bc_put(bc);
+ *slab_bcp = NULL;
+}
+
+/*
+ * Pages accounting
+ */
+
+int bc_page_charge(struct page *page, int order, gfp_t flags)
+{
+ struct beancounter *bc;
+
+ bc = get_exec_bc();
+
```

```
+ if (bc_charge(bc, BC_KMEMSIZE, PAGE_SIZE << order,
+    (flags & __GFP_BC_LIMIT ? BC_LIMIT : BC_BARRIER)))
+  return -ENOMEM;
+
+ BUG_ON(page_bc(page) != NULL);
+ page_bc(page) = bc_get(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);
+ bc_put(bc);
+ page_bc(page) = NULL;
+}
+
+/*
+ * Generic resource info
+ */
+
+static int bc_kmem_init(struct beancounter *bc, int res)
+{
+ bc_init_resource(&bc->bc_parms[BC_KMEMSIZE],
+    BC_KMEMSIZE_BARRIER, BC_KMEMSIZE_LIMIT);
+ return 0;
+}
+
+struct bc_resource bc_kmem_resource = {
+ .bcr_name  = "kmemsize",
+ .bcr_init  = bc_kmem_init,
+};
+
+static int __init bc_kmem_init_resource(void)
+{
+ bc_register_resource(BC_KMEMSIZE, &bc_kmem_resource);
+ return 0;
+}
+
+__initcall(bc_kmem_init_resource);
--- ./mm/slab.c.bc 2006-11-03 15:35:28.000000000 +0300
+++ ./mm/slab.c 2006-11-03 15:48:26.000000000 +0300
@@ -109,6 +109,8 @@
```

```
 #include <linux/rtmutex.h>
 #include <linux/uaccess.h>

+#include <bc/kmem.h>
+
 #include <asm/cacheflush.h>
 #include <asm/tlbflush.h>
 #include <asm/page.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

@@ -796,9 +800,33 @@ 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);
+ 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);
 }

 /*
@@ -843,20 +871,21 @@ 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;
@@ -1989,7 +2021,8 @@ 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;
@@ -2291,8 +2324,8 @@ 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
@@ -2303,11 +2336,9 @@ 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. */
@@ -2548,6 +2579,30 @@ void kmem_cache_destroy(struct kmem_cach
 }
 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
@@ -2568,7 +2623,8 @@ 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 {
@@ -2579,14 +2635,14 @@ 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)
 {
@@ -2766,7 +2822,7 @@ 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;

--- ./mm/vmalloc.c.bc 2006-11-03 15:35:28.000000000 +0300
+++ ./mm/vmalloc.c 2006-11-03 15:48:26.000000000 +0300
@@ -517,6 +517,12 @@ 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 zeroed virtually contiguous memory for userspace
  * @size: allocation size
```
