Subject: [PATCH 7/7] BeanCounters over generic process containers
Posted by Paul Menage on Thu, 23 Nov 2006 12:08:55 GMT
View Forum Message <> Reply to Message

This patch implements the BeanCounter resource control abstraction
over generic process containers. It contains the beancounter core
code, plus the numfiles resource counter. It doesn't currently contain
any of the memory tracking code or the code for switching beancounter
context in interrupts.

Currently all the beancounters resource counters are lumped into a
single hierarchy; ideally it would be possible for each resource
counter to be a separate container subsystem, allowing them to be
connected to different hierarchies.

---
 fs/file_table.c | 11 +
 include/bc/beancounter.h | 192 ++++++++++++++++++++++
 include/bc/misc.h | 27 +++
 include/linux/fs.h | 3
 init/Kconfig | 4
 init/main.c | 3
 kernel/Makefile | 1
 kernel/bc/Kconfig | 17 ++
 kernel/bc/Makefile | 7
 kernel/bc/beancounter.c | 371 ++
 kernel/bc/misc.c | 56 +++++++
 11 files changed, 691 insertions(+), 1 deletion(-)

Index: container-2.6.19-rc6/init/Kconfig
 === =======
--- container-2.6.19-rc6.orig/init/Kconfig
+++ container-2.6.19-rc6/init/Kconfig
@@ -601,6 +601,10 @@ config STOP_MACHINE
   Need stop_machine() primitive.
 endmenu

+menu "Beancounters"
+source "kernel/bc/Kconfig"
+endmenu
+
 menu "Block layer"
 source "block/Kconfig"
 endmenu
Index: container-2.6.19-rc6/kernel/Makefile
 === =======
--- container-2.6.19-rc6.orig/kernel/Makefile
+++ container-2.6.19-rc6/kernel/Makefile

```
@@ -12,6 +12,7 @@ obj-y     = sched.o fork.o exec_domain.o

 obj-$(CONFIG_STACKTRACE) += stacktrace.o
 obj-y += time/
+obj-$(CONFIG_BEANCOUNTERS) += bc/
 obj-$(CONFIG_DEBUG_MUTEXES) += mutex-debug.o
 obj-$(CONFIG_LOCKDEP) += lockdep.o
 ifeq ($(CONFIG_PROC_FS),y)
Index: container-2.6.19-rc6/kernel/bc/Kconfig
 ===================================================================
--- /dev/null
+++ container-2.6.19-rc6/kernel/bc/Kconfig
@@ -0,0 +1,17 @@
+config BEANCOUNTERS
+ bool "Enable resource accounting/control"
+ default n
+ select CONTAINERS
+ help
+   When Y this option provides accounting and allows configuring
+   limits for user's consumption of exhaustible system resources.
+   The most important resource controlled by this patch is unswappable
+   memory (either mlock'ed or used by internal kernel structures and
+   buffers). The main goal of this patch is to protect processes
+   from running short of important resources because of accidental
+   misbehavior of processes or malicious activity aiming to ``kill''
+   the system. It's worth mentioning that resource limits configured
+   by setrlimit(2) do not give an acceptable level of protection
+   because they cover only a small fraction of resources and work on a
+   per-process basis.  Per-process accounting doesn't prevent malicious
+   users from spawning a lot of resource-consuming processes.
Index: container-2.6.19-rc6/kernel/bc/Makefile
 ===================================================================
--- /dev/null
+++ container-2.6.19-rc6/kernel/bc/Makefile
@@ -0,0 +1,7 @@
+#
+# kernel/bc/Makefile
+#
+# Copyright (C) 2006 OpenVZ SWsoft Inc.
+#
+
+obj-y = beancounter.o misc.o
Index: container-2.6.19-rc6/include/bc/beancounter.h
 ===================================================================
--- /dev/null
+++ container-2.6.19-rc6/include/bc/beancounter.h
@@ -0,0 +1,192 @@
+/*
```

```
+ * include/bc/beancounter.h
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#ifndef __BEANCOUNTER_H__
+#define __BEANCOUNTER_H__
+
+#include <linux/container.h>
+
+enum {
+ BC_KMEMSIZE,
+ BC_PRIVVMPAGES,
+ BC_PHYSPAGES,
+ BC_NUMTASKS,
+ BC_NUMFILES,
+
+ BC_RESOURCES
+};
+
+struct bc_resource_parm {
+ unsigned long barrier;
+ unsigned long limit;
+ unsigned long held;
+ unsigned long minheld;
+ unsigned long maxheld;
+ unsigned long failcnt;
+
+};
+
+#ifdef __KERNEL__
+
+#include <linux/list.h>
+#include <linux/spinlock.h>
+#include <linux/init.h>
+#include <linux/configfs.h>
+#include <asm/atomic.h>
+
+#define BC_MAXVALUE ((unsigned long)LONG_MAX)
+
+enum bc_severity {
+ BC_BARRIER,
+ BC_LIMIT,
+ BC_FORCE,
+};
+
+struct beancounter;
```

```
+
+#ifdef CONFIG_BEANCOUNTERS
+
+enum bc_attr_index {
+ BC_RES_HELD,
+ BC_RES_MAXHELD,
+ BC_RES_MINHELD,
+ BC_RES_BARRIER,
+ BC_RES_LIMIT,
+ BC_RES_FAILCNT,
+
+ BC_ATTRS
+};
+
+struct bc_resource {
+ char *bcr_name;
+ int      res_id;
+
+ int (*bcr_init)(struct beancounter *bc, int res);
+ int (*bcr_change)(struct beancounter *bc,
+   unsigned long new_bar, unsigned long new_lim);
+ void (*bcr_barrier_hit)(struct beancounter *bc);
+ int (*bcr_limit_hit)(struct beancounter *bc, unsigned long val,
+   unsigned long flags);
+ void (*bcr_fini)(struct beancounter *bc);
+
+ /* container file handlers */
+ struct cftype cft_attrs[BC_ATTRS];
+};
+
+extern struct bc_resource *bc_resources[];
+extern struct container_subsys bc_subsys;
+
+struct beancounter {
+ struct container_subsys_state css;
+ spinlock_t  bc_lock;
+
+ struct bc_resource_parm bc_parms[BC_RESOURCES];
+};
+
+/* Update the beancounter for a container */
+static inline void set_container_bc(struct container *cont,
+        struct beancounter *bc)
+{
+ cont->subsys[bc_subsys.subsys_id] = &bc->css;
+}
+
+/* Retrieve the beancounter for a container */
```

```
+static inline struct beancounter *container_bc(struct container *cont)
+{
+ return container_of(container_subsys_state(cont, &bc_subsys),
+       struct beancounter, css);
+}
+
+/* Retrieve the beancounter for a task */
+static inline struct beancounter *task_bc(struct task_struct *task)
+{
+ return container_bc(task_container(task, &bc_subsys));
+}
+
+static inline void bc_adjust_maxheld(struct bc_resource_parm *parm)
+{
+ if (parm->maxheld < parm->held)
+  parm->maxheld = parm->held;
+}
+
+static inline void bc_adjust_minheld(struct bc_resource_parm *parm)
+{
+ if (parm->minheld > parm->held)
+  parm->minheld = parm->held;
+}
+
+static inline void bc_init_resource(struct bc_resource_parm *parm,
+  unsigned long bar, unsigned long lim)
+{
+ parm->barrier = bar;
+ parm->limit = lim;
+ parm->held = 0;
+ parm->minheld = 0;
+ parm->maxheld = 0;
+ parm->failcnt = 0;
+}
+
+int bc_change_param(struct beancounter *bc, int res,
+  unsigned long bar, unsigned long lim);
+
+int __must_check bc_charge_locked(struct beancounter *bc, int res_id,
+  unsigned long val, int strict, unsigned long flags);
+static inline int __must_check bc_charge(struct beancounter *bc, int res_id,
+  unsigned long val, int strict)
+{
+ int ret;
+ unsigned long flags;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ ret = bc_charge_locked(bc, res_id, val, strict, flags);
```

```
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+ return ret;
+}
+
+void __must_check bc_uncharge_locked(struct beancounter *bc, int res_id,
+  unsigned long val);
+static inline void bc_uncharge(struct beancounter *bc, int res_id,
+  unsigned long val)
+{
+ unsigned long flags;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ bc_uncharge_locked(bc, res_id, val);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+}
+
+void __init bc_register_resource(int res_id, struct bc_resource *br);
+void __init bc_init_early(void);
+#else /* CONFIG_BEANCOUNTERS */
+static inline int __must_check bc_charge_locked(struct beancounter *bc, int res,
+  unsigned long val, int strict, unsigned long flags)
+{
+ return 0;
+}
+
+static inline int __must_check bc_charge(struct beancounter *bc, int res,
+  unsigned long val, int strict)
+{
+ return 0;
+}
+
+static inline void bc_uncharge_locked(struct beancounter *bc, int res,
+  unsigned long val)
+{
+}
+
+static inline void bc_uncharge(struct beancounter *bc, int res,
+  unsigned long val)
+{
+}
+
+static inline void bc_init_early(void)
+{
+}
+#endif /* CONFIG_BEANCOUNTERS */
+#endif /* __KERNEL__ */
+#endif
Index: container-2.6.19-rc6/init/main.c
```

```
 ===================================================== =======
--- container-2.6.19-rc6.orig/init/main.c
+++ container-2.6.19-rc6/init/main.c
@@ -52,6 +52,8 @@
 #include <linux/debug_locks.h>
 #include <linux/lockdep.h>

+#include <bc/beancounter.h>
+
 #include <asm/io.h>
 #include <asm/bugs.h>
 #include <asm/setup.h>
@@ -482,6 +484,7 @@ asmlinkage void __init start_kernel(void
  char * command_line;
  extern struct kernel_param __start___param[], __stop___param[];

+ bc_init_early();
  smp_setup_processor_id();

  /*
Index: container-2.6.19-rc6/kernel/bc/beancounter.c
 ===================================================== =======
--- /dev/null
+++ container-2.6.19-rc6/kernel/bc/beancounter.c
@@ -0,0 +1,371 @@
+/*
+ * kernel/bc/beancounter.c
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#include <linux/sched.h>
+#include <linux/list.h>
+#include <linux/hash.h>
+#include <linux/gfp.h>
+#include <linux/slab.h>
+#include <linux/module.h>
+#include <linux/fs.h>
+#include <linux/uaccess.h>
+
+#include <bc/beancounter.h>
+
+#define BC_HASH_BITS (8)
+#define BC_HASH_SIZE (1 << BC_HASH_BITS)
+
+static int bc_dummy_init(struct beancounter *bc, int i)
+{
```

```
+ bc_init_resource(&bc->bc_parms[i], BC_MAXVALUE, BC_MAXVALUE);
+ return 0;
+}
+
+static struct bc_resource bc_dummy_res = {
+ .bcr_name = "dummy",
+ .bcr_init = bc_dummy_init,
+};
+
+struct bc_resource *bc_resources[BC_RESOURCES] = {
+ [0 ... BC_RESOURCES - 1] = &bc_dummy_res,
+};
+
+struct beancounter init_bc;
+static kmem_cache_t *bc_cache;
+
+static int bc_create(struct container_subsys *ss,
+       struct container *cont)
+{
+ int i;
+ struct beancounter *new_bc;
+
+ if (!cont->parent) {
+  /* Early initialization for top container */
+  set_container_bc(cont, &init_bc);
+  init_bc.css.container = cont;
+  return 0;
+ }
+
+ new_bc = kmem_cache_alloc(bc_cache, GFP_KERNEL);
+ if (!new_bc)
+  return -ENOMEM;
+
+ spin_lock_init(&new_bc->bc_lock);
+
+ for (i = 0; i < BC_RESOURCES; i++) {
+  int err;
+  if ((err = bc_resources[i]->bcr_init(new_bc, i))) {
+   for (i--; i >= 0; i--)
+    if (bc_resources[i]->bcr_fini)
+     bc_resources[i]->bcr_fini(new_bc);
+   kmem_cache_free(bc_cache, new_bc);
+   return err;
+  }
+ }
+
+ new_bc->css.container = cont;
+ set_container_bc(cont, new_bc);
```

```
+ return 0;
+}
+
+static void bc_destroy(struct container_subsys *ss,
+         struct container *cont)
+{
+ struct beancounter *bc = container_bc(cont);
+ kmem_cache_free(bc_cache, bc);
+}
+
+int bc_charge_locked(struct beancounter *bc, int res, unsigned long val,
+  int strict, unsigned long flags)
+{
+ struct bc_resource_parm *parm;
+ unsigned long new_held;
+
+ BUG_ON(val > BC_MAXVALUE);
+
+ parm = &bc->bc_parms[res];
+ new_held = parm->held + val;
+
+ switch (strict) {
+ case BC_LIMIT:
+  if (new_held > parm->limit)
+   break;
+  /* fallthrough */
+ case BC_BARRIER:
+  if (new_held > parm->barrier) {
+   if (strict == BC_BARRIER)
+    break;
+   if (parm->held < parm->barrier &&
+     bc_resources[res]->bcr_barrier_hit)
+    bc_resources[res]->bcr_barrier_hit(bc);
+  }
+  /* fallthrough */
+ case BC_FORCE:
+  parm->held = new_held;
+  bc_adjust_maxheld(parm);
+  return 0;
+ default:
+  BUG();
+ }
+
+ if (bc_resources[res]->bcr_limit_hit)
+  return bc_resources[res]->bcr_limit_hit(bc, val, flags);
+
+ parm->failcnt++;
+ return -ENOMEM;
```

```
+}
+
+void bc_uncharge_locked(struct beancounter *bc, int res, unsigned long val)
+{
+ struct bc_resource_parm *parm;
+
+ BUG_ON(val > BC_MAXVALUE);
+
+ parm = &bc->bc_parms[res];
+ if (unlikely(val > parm->held)) {
+  printk(KERN_ERR "BC: Uncharging too much of %s: %lu vs %lu\n",
+    bc_resources[res]->bcr_name,
+    val, parm->held);
+  val = parm->held;
+ }
+
+ parm->held -= val;
+ bc_adjust_minheld(parm);
+}
+
+int bc_change_param(struct beancounter *bc, int res,
+      unsigned long bar, unsigned long lim)
+{
+ int ret;
+
+ ret = -EINVAL;
+ if (bar > lim)
+  goto out;
+ if (bar > BC_MAXVALUE || lim > BC_MAXVALUE)
+  goto out;
+
+ ret = 0;
+ spin_lock_irq(&bc->bc_lock);
+ if (bc_resources[res]->bcr_change) {
+  ret = bc_resources[res]->bcr_change(bc, bar, lim);
+  if (ret < 0)
+   goto out_unlock;
+ }
+
+ bc->bc_parms[res].barrier = bar;
+ bc->bc_parms[res].limit = lim;
+
+out_unlock:
+ spin_unlock_irq(&bc->bc_lock);
+out:
+ return ret;
+}
+
```

```
+static ssize_t bc_resource_show(struct container *cont, struct cftype *cft,
+    struct file *file, char __user *userbuf,
+    size_t nbytes, loff_t *ppos)
+{
+ struct beancounter *bc = container_bc(cont);
+ int idx = cft->private;
+ struct bc_resource *res = container_of(cft, struct bc_resource,
+            cft_attrs[idx]);
+
+ char *page;
+ ssize_t retval = 0;
+ char *s;
+
+ if (!(page = (char *)__get_free_page(GFP_KERNEL)))
+  return -ENOMEM;
+
+ s = page;
+
+ switch(idx) {
+ case BC_RES_HELD:
+  s += sprintf(page, "%lu\n", bc->bc_parms[res->res_id].held);
+  break;
+ case BC_RES_BARRIER:
+  s += sprintf(page, "%lu\n", bc->bc_parms[res->res_id].barrier);
+  break;
+ case BC_RES_LIMIT:
+  s += sprintf(page, "%lu\n", bc->bc_parms[res->res_id].limit);
+  break;
+ case BC_RES_MAXHELD:
+  s += sprintf(page, "%lu\n", bc->bc_parms[res->res_id].maxheld);
+  break;
+ case BC_RES_MINHELD:
+  s += sprintf(page, "%lu\n", bc->bc_parms[res->res_id].minheld);
+  break;
+ case BC_RES_FAILCNT:
+  s += sprintf(page, "%lu\n", bc->bc_parms[res->res_id].failcnt);
+  break;
+ default:
+  retval = -EINVAL;
+  goto out;
+  break;
+ }
+
+ retval = simple_read_from_buffer(userbuf, nbytes, ppos, page, s-page);
+ out:
+ free_page((unsigned long) page);
+ return retval;
+}
```

```
+
+static ssize_t bc_resource_store(struct container *cont, struct cftype *cft,
+     struct file *file,
+     const char __user *userbuf,
+     size_t nbytes, loff_t *ppos)
+{
+ struct beancounter *bc = container_bc(cont);
+ int idx = cft->private;
+ struct bc_resource *res = container_of(cft, struct bc_resource,
+           cft_attrs[idx]);
+
+ char buffer[64];
+ unsigned long val;
+ char *end;
+ int ret = 0;
+
+ if (nbytes >= sizeof(buffer))
+  return -E2BIG;
+
+ if (copy_from_user(buffer, userbuf, nbytes))
+  return -EFAULT;
+
+ buffer[nbytes] = 0;
+
+ container_manage_lock();
+
+ if (container_is_removed(cont)) {
+  ret = -ENODEV;
+  goto out_unlock;
+ }
+
+ ret = -EINVAL;
+ val = simple_strtoul(buffer, &end, 10);
+ if (*end != '\0')
+  goto out_unlock;
+
+ switch (idx) {
+ case BC_RES_BARRIER:
+  ret = bc_change_param(bc, res->res_id,
+         val, bc->bc_parms[res->res_id].limit);
+  break;
+ case BC_RES_LIMIT:
+  ret = bc_change_param(bc, res->res_id,
+         bc->bc_parms[res->res_id].barrier, val);
+  break;
+ }
+
+ if (ret == 0)
```

```
+  ret = nbytes;
+
+ out_unlock:
+ container_manage_unlock();
+
+ return ret;
+}
+
+
+
+void __init bc_register_resource(int res_id, struct bc_resource *br)
+{
+ int attr;
+ BUG_ON(bc_resources[res_id] != &bc_dummy_res);
+ BUG_ON(res_id >= BC_RESOURCES);
+
+ bc_resources[res_id] = br;
+ br->res_id = res_id;
+
+ for (attr = 0; attr < BC_ATTRS; attr++) {
+
+  /* Construct a file handler for each attribute of this
+   * resource; the name is of the form
+   * "bc.<resname>.<attrname>" */
+
+  struct cftype *cft = &br->cft_attrs[attr];
+  const char *attrname;
+  cft->private = attr;
+  strcpy(cft->name, "bc.");
+  strcat(cft->name, br->bcr_name);
+  strcat(cft->name, ".");
+  switch (attr) {
+  case BC_RES_HELD:
+   attrname = "held"; break;
+  case BC_RES_BARRIER:
+   attrname = "barrier"; break;
+  case BC_RES_LIMIT:
+   attrname = "limit"; break;
+  case BC_RES_MAXHELD:
+   attrname = "maxheld"; break;
+  case BC_RES_MINHELD:
+   attrname = "minheld"; break;
+  case BC_RES_FAILCNT:
+   attrname = "failcnt"; break;
+  default:
+   BUG();
+  }
+  strcat(cft->name, attrname);
```

```
+  cft->read = bc_resource_show;
+  cft->write = bc_resource_store;
+ }
+}
+
+void __init bc_init_early(void)
+{
+ int i;
+
+ spin_lock_init(&init_bc.bc_lock);
+
+ for (i = 0; i < BC_RESOURCES; i++) {
+  init_bc.bc_parms[i].barrier = BC_MAXVALUE;
+  init_bc.bc_parms[i].limit = BC_MAXVALUE;
+
+ }
+
+ if (container_register_subsys(&bc_subsys) < 0)
+  panic("Couldn't register beancounter subsystem");
+
+}
+
+int __init bc_init_late(void)
+{
+ bc_cache = kmem_cache_create("beancounters",
+   sizeof(struct beancounter), 0,
+   SLAB_HWCACHE_ALIGN | SLAB_PANIC, NULL, NULL);
+ return 0;
+}
+
+__initcall(bc_init_late);
+
+static int bc_populate(struct container_subsys *ss, struct container *cont)
+{
+ int err;
+ int attr, res;
+ for (res = 0; res < BC_RESOURCES; res++) {
+  struct bc_resource *bcr = bc_resources[res];
+
+  for (attr = 0; attr < BC_ATTRS; attr++) {
+   struct cftype *cft = &bcr->cft_attrs[attr];
+   if (!cft->name[0]) continue;
+   err = container_add_file(cont, cft);
+   if (err < 0) return err;
+  }
+ }
+ return 0;
+}
```

```
+
+struct container_subsys bc_subsys = {
+ .name = "bc",
+ .create = bc_create,
+ .destroy = bc_destroy,
+ .populate = bc_populate,
+ .subsys_id = -1,
+};
+
+EXPORT_SYMBOL(bc_resources);
+EXPORT_SYMBOL(init_bc);
+EXPORT_SYMBOL(bc_change_param);
Index: container-2.6.19-rc6/include/bc/misc.h
 ===================================================================
--- /dev/null
+++ container-2.6.19-rc6/include/bc/misc.h
@@ -0,0 +1,27 @@
+/*
+ * include/bc/misc.h
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#ifndef __BC_MISC_H__
+#define __BC_MISC_H__
+
+struct file;
+
+#ifdef CONFIG_BEANCOUNTERS
+int __must_check bc_file_charge(struct file *);
+void bc_file_uncharge(struct file *);
+#else
+static inline int __must_check bc_file_charge(struct file *f)
+{
+ return 0;
+}
+
+static inline void bc_file_uncharge(struct file *f)
+{
+}
+#endif
+
+#endif
Index: container-2.6.19-rc6/kernel/bc/misc.c
 ===================================================================
--- /dev/null
+++ container-2.6.19-rc6/kernel/bc/misc.c
```

```
@@ -0,0 +1,56 @@
+
+#include <linux/fs.h>
+#include <bc/beancounter.h>
+
+int bc_file_charge(struct file *file)
+{
+ int sev;
+ struct beancounter *bc;
+
+ task_lock(current);
+ bc = task_bc(current);
+ css_get_current(&bc->css);
+ task_unlock(current);
+
+ sev = (capable(CAP_SYS_ADMIN) ? BC_LIMIT : BC_BARRIER);
+
+ if (bc_charge(bc, BC_NUMFILES, 1, sev)) {
+  css_put(&bc->css);
+  return -EMFILE;
+ }
+
+ file->f_bc = bc;
+ return 0;
+}
+
+void bc_file_uncharge(struct file *file)
+{
+ struct beancounter *bc;
+
+ bc = file->f_bc;
+ bc_uncharge(bc, BC_NUMFILES, 1);
+ css_put(&bc->css);
+}
+
+#define BC_NUMFILES_BARRIER 256
+#define BC_NUMFILES_LIMIT 512
+
+static int bc_files_init(struct beancounter *bc, int i)
+{
+ bc_init_resource(&bc->bc_parms[BC_NUMFILES],
+   BC_NUMFILES_BARRIER, BC_NUMFILES_LIMIT);
+ return 0;
+}
+
+static struct bc_resource bc_files_resource = {
+ .bcr_name = "numfiles",
+ .bcr_init = bc_files_init,
```

```
+};
+
+static int __init bc_misc_init_resource(void)
+{
+ bc_register_resource(BC_NUMFILES, &bc_files_resource);
+  return 0;
+}
+
+__initcall(bc_misc_init_resource);
Index: container-2.6.19-rc6/fs/file_table.c
 ===================================================================
--- container-2.6.19-rc6.orig/fs/file_table.c
+++ container-2.6.19-rc6/fs/file_table.c
@@ -22,6 +22,8 @@
 #include <linux/sysctl.h>
 #include <linux/percpu_counter.h>

+#include <bc/misc.h>
+
 #include <asm/atomic.h>

 /* sysctl tunables... */
@@ -43,6 +45,7 @@ static inline void file_free_rcu(struct
 static inline void file_free(struct file *f)
 {
  percpu_counter_dec(&nr_files);
+ bc_file_uncharge(f);
  call_rcu(&f->f_u.fu_rcuhead, file_free_rcu);
 }

@@ -107,8 +110,10 @@ struct file *get_empty_filp(void)
  if (f == NULL)
   goto fail;

- percpu_counter_inc(&nr_files);
  memset(f, 0, sizeof(*f));
+ if (bc_file_charge(f))
+  goto fail_charge;
+ percpu_counter_inc(&nr_files);
  if (security_file_alloc(f))
   goto fail_sec;

@@ -135,6 +140,10 @@ fail_sec:
  file_free(f);
 fail:
  return NULL;
+
+ fail_charge:
```

```
+ kmem_cache_free(filp_cachep, f);
+ return NULL;
 }

 EXPORT_SYMBOL(get_empty_filp);
Index: container-2.6.19-rc6/include/linux/fs.h
 ===================================================================== =======
--- container-2.6.19-rc6.orig/include/linux/fs.h
+++ container-2.6.19-rc6/include/linux/fs.h
@@ -750,6 +750,9 @@ struct file {
  spinlock_t  f_ep_lock;
 #endif /* #ifdef CONFIG_EPOLL */
  struct address_space *f_mapping;
+#ifdef CONFIG_BEANCOUNTERS
+ struct beancounter      *f_bc;
+#endif
 };
 extern spinlock_t files_lock;
 #define file_list_lock() spin_lock(&files_lock);

--
```