Subject: [PATCH 4/5] Setup the control group
Posted by Pavel Emelianov on Tue, 25 Sep 2007 14:24:30 GMT
View Forum Message <> Reply to Message

Attach the controller to the control groups. This will work
with the SLUB allocator only. However, if we need I can
port this on SLAB (and maybe SLOB ;)).

This setup is simple and stupid.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
index d822977..bd0f493 100644
--- a/include/linux/cgroup_subsys.h
+++ b/include/linux/cgroup_subsys.h
@@ -36,3 +36,9 @@ SUBSYS(mem_cgroup)
 #endif

 /* */
+
+#ifdef CONFIG_CGROUP_KMEM
+SUBSYS(kmem)
+#endif
+
+/* */
diff --git a/init/Kconfig b/init/Kconfig
index 684ccfb..e9acc29 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -353,6 +353,12 @@ config CGROUP_MEM_CONT
    Provides a memory controller that manages both page cache and
    RSS memory.

+config CGROUP_KMEM
+ bool "Kernel memory controller for containers"
+ depends on CGROUPS && RESOURCE_COUNTERS && SLUB
+ help
+   Provides a kernel memory usage control for containers
+
 config PROC_PID_CPUSET
  bool "Include legacy /proc/<pid>/cpuset file"
  depends on CPUSETS
diff --git a/mm/kmemcontrol.c b/mm/kmemcontrol.c
new file mode 100644
index 0000000..5698c0f
```

```
--- /dev/null
+++ b/mm/kmemcontrol.c
@@ -0,0 +1,119 @@
+/*
+ * kmemcontrol.c - Kernel Memory Controller
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ * Author: Pavel Emelyanov <xemul@openvz.org>
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
+ * GNU General Public License for more details.
+ */
+
+#include <linux/mm.h>
+#include <linux/cgroup.h>
+#include <linux/res_counter.h>
+#include <linux/err.h>
+
+struct kmem_container {
+ struct cgroup_subsys_state css;
+ struct res_counter res;
+};
+
+static inline struct kmem_container *css_to_kmem(struct cgroup_subsys_state *ss)
+{
+ return container_of(ss, struct kmem_container, css);
+}
+
+static inline struct kmem_container *cgroup_to_kmem(struct cgroup *cg)
+{
+ return css_to_kmem(cgroup_subsys_state(cg, kmem_subsys_id));
+}
+
+static inline struct kmem_container *task_kmem_container(struct task_struct *t)
+{
+ return css_to_kmem(task_subsys_state(t, kmem_subsys_id));
+}
+
+/*
+ * cgroups interface
+ */
```

```
+
+static struct kmem_container init_kmem_container;
+
+static struct cgroup_subsys_state *kmem_create(struct cgroup_subsys *ss,
+  struct cgroup *cgroup)
+{
+ struct kmem_container *mem;
+
+ if (unlikely((cgroup->parent) == NULL))
+  mem = &init_kmem_container;
+ else
+  mem = kzalloc(sizeof(struct kmem_container), GFP_KERNEL);
+
+ if (mem == NULL)
+  return ERR_PTR(-ENOMEM);
+
+ res_counter_init(&mem->res);
+ return &mem->css;
+
+}
+
+static void kmem_destroy(struct cgroup_subsys *ss, struct cgroup *cgroup)
+{
+ kfree(cgroup_to_kmem(cgroup));
+}
+
+static ssize_t kmem_container_read(struct cgroup *cg, struct cftype *cft,
+  struct file *file, char __user *userbuf, size_t nbytes,
+  loff_t *ppos)
+{
+ return res_counter_read(&cgroup_to_kmem(cg)->res,
+   cft->private, userbuf, nbytes, ppos, NULL);
+}
+
+static ssize_t kmem_container_write(struct cgroup *cg, struct cftype *cft,
+  struct file *file, const char __user *userbuf,
+  size_t nbytes, loff_t *ppos)
+{
+ return res_counter_write(&cgroup_to_kmem(cg)->res,
+   cft->private, userbuf, nbytes, ppos, NULL);
+}
+
+static struct cftype kmem_files[] = {
+ {
+  .name = "usage",
+  .private = RES_USAGE,
+  .read = kmem_container_read,
+ },
```

```
+ {
+  .name = "limit",
+  .private = RES_LIMIT,
+  .write = kmem_container_write,
+  .read = kmem_container_read,
+ },
+ {
+  .name = "failcnt",
+  .private = RES_FAILCNT,
+  .read = kmem_container_read,
+ },
+};
+
+static int kmem_populate(struct cgroup_subsys *ss, struct cgroup *cnt)
+{
+ return cgroup_add_files(cnt, ss, kmem_files, ARRAY_SIZE(kmem_files));
+}
+
+struct cgroup_subsys kmem_subsys = {
+ .name = "kmem",
+ .create = kmem_create,
+ .destroy  = kmem_destroy,
+ .populate = kmem_populate,
+ .subsys_id = kmem_subsys_id,
+ .early_init = 1,
+};
```