Subject: [PATCH 5/13] BC: configfs interface
Posted by dev on Thu, 09 Nov 2006 16:52:24 GMT
View Forum Message <> Reply to Message

configfs interface to beancounters.
It may be done as module in case configfs itself is a module.

Usage example:

Prepare configfs interface
~~~~~~~~~~~~~~~~~~~~~~~~~~
```
# mount -t configfs none /cfg
# ls /cfg
  beancounters
# ls /cfg/beancounters/
  0  description
# cat /cfg/beancounters/description
  Beancounters controll resource usage of task groups
```

Look at beancounters subsystem structure
~~~~~~
```
# ls /cfg/beancounters/0/
    id  kmemsize  numfiles  numtasks  physpages  privvmpages  tasks  threads
# ls /cfg/beancounters/0/numfiles
    barrier  failcnt  held  limit  maxheld  minheld
# cat /cfg/beancounters/0/numfiles/*
    2147483647
    0
    163
    2147483647
    345
    0
```

Create a new beancounter and move task into it
~~~~~~
```
# mkdir /cfg/beancounters/1
# echo -n $$ > /cfg/beancounters/1/tasks
# cat /proc/$$/status | grep Bcid
Bcid:   1
```

now you can see bc 1 usages etc.

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

```
 configfs.c |  391 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```
~~~~~~

```
+
 1 files changed, 391 insertions(+)

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/configfs.c 2006-11-03 17:13:25.000000000 +0300
@@ -0,0 +1,391 @@
+/*
+ * kernel/bc/configfs.c
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ * Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
+#include <linux/configfs.h>
+#include <linux/module.h>
+
+#include <bc/beancounter.h>
+#include <bc/task.h>
+
+/*
+ * Declarations
+ */
+
+enum bc_attr_indices {
+ BC_ATTR_ID,
+ BC_ATTR_TASKS,
+ BC_ATTR_THREADS,
+ BC_RES_HELD,
+ BC_RES_MAXHELD,
+ BC_RES_MINHELD,
+ BC_RES_BARRIER,
+ BC_RES_LIMIT,
+ BC_RES_FAILCNT,
+};
+
+struct bc_configfs_attribute {
+ int idx;
+ struct configfs_attribute attr;
+};
+
+#define attr_to_bc(attr) container_of(attr, \
+  struct bc_configfs_attribute, attr)
+
+struct bc_resource_group {
+ int res;
+ struct config_group group;
```

```
+};
+
+#define item_to_res(i) container_of(i, \
+  struct bc_resource_group, group.cg_item)
+
+struct bc_group {
+ struct beancounter *bc;
+ struct config_group group;
+};
+
+#define item_to_bc(i) container_of(i, \
+  struct bc_group, group.cg_item)
+
+#define declare_bc_attr(i, name, mode)    \
+ static struct bc_configfs_attribute bc_attr_##name = { \
+  .idx = i,     \
+  .attr = {     \
+    .ca_owner = THIS_MODULE,  \
+    .ca_name = #name,   \
+    .ca_mode = mode,   \
+  },      \
+ }
+
+declare_bc_attr(BC_ATTR_ID, id, S_IRUGO);
+declare_bc_attr(BC_ATTR_TASKS, tasks, S_IWUSR);
+declare_bc_attr(BC_ATTR_THREADS, threads, S_IWUSR);
+declare_bc_attr(BC_RES_HELD, held, S_IRUGO);
+declare_bc_attr(BC_RES_BARRIER, barrier, S_IRUGO | S_IWUSR);
+declare_bc_attr(BC_RES_LIMIT, limit, S_IRUGO | S_IWUSR);
+declare_bc_attr(BC_RES_MAXHELD, maxheld, S_IRUGO);
+declare_bc_attr(BC_RES_MINHELD, minheld, S_IRUGO);
+declare_bc_attr(BC_RES_FAILCNT, failcnt, S_IRUGO);
+
+static struct configfs_attribute *bc_attributes[] = {
+ &bc_attr_id.attr,
+ &bc_attr_tasks.attr,
+ &bc_attr_threads.attr,
+ NULL,
+};
+
+static struct configfs_attribute *resource_attributes[] = {
+ &bc_attr_held.attr,
+ &bc_attr_barrier.attr,
+ &bc_attr_limit.attr,
+ &bc_attr_maxheld.attr,
+ &bc_attr_minheld.attr,
+ &bc_attr_failcnt.attr,
+ NULL,
```

```
+};
+
+static ssize_t bc_show(struct config_item *item,
+  struct configfs_attribute *attr, char *page)
+{
+ struct beancounter *bc;
+
+ bc = item_to_bc(item)->bc;
+
+ switch (attr_to_bc(attr)->idx) {
+ case BC_ATTR_ID:
+  return sprintf(page, "%d\n", bc->bc_id);
+ }
+
+ return 0;
+}
+
+static ssize_t bc_store(struct config_item *item,
+  struct configfs_attribute *attr,
+  const char *page, size_t size)
+{
+ struct beancounter *bc;
+ char *end;
+ int ret;
+
+ ret = simple_strtoul(page, &end, 10);
+ if (*end != '\0')
+  return -EINVAL;
+
+ bc = item_to_bc(item)->bc;
+
+ switch (attr_to_bc(attr)->idx) {
+ case BC_ATTR_TASKS:
+  ret = bc_task_move(ret, bc, 1);
+  if (ret == 0)
+   ret = size;
+  break;
+ case BC_ATTR_THREADS:
+  ret = bc_task_move(ret, bc, 0);
+  if (ret == 0)
+   ret = size;
+  break;
+ };
+
+ return ret;
+}
+
+static ssize_t resource_show(struct config_item *item,
```

```
+  struct configfs_attribute *attr, char *page)
+{
+ struct beancounter *bc;
+ struct bc_resource_group *grp;
+
+ bc = item_to_bc(item->ci_parent)->bc;
+ grp = item_to_res(item);
+
+ switch (attr_to_bc(attr)->idx) {
+ case BC_RES_HELD:
+  return sprintf(page, "%lu\n", bc->bc_parms[grp->res].held);
+ case BC_RES_BARRIER:
+  return sprintf(page, "%lu\n", bc->bc_parms[grp->res].barrier);
+ case BC_RES_LIMIT:
+  return sprintf(page, "%lu\n", bc->bc_parms[grp->res].limit);
+ case BC_RES_MAXHELD:
+  return sprintf(page, "%lu\n", bc->bc_parms[grp->res].maxheld);
+ case BC_RES_MINHELD:
+  return sprintf(page, "%lu\n", bc->bc_parms[grp->res].minheld);
+ case BC_RES_FAILCNT:
+  return sprintf(page, "%lu\n", bc->bc_parms[grp->res].failcnt);
+ }
+
+ return 0;
+}
+
+static ssize_t resource_store(struct config_item *item,
+  struct configfs_attribute *attr,
+  const char *page, size_t size)
+{
+ struct beancounter *bc;
+ struct bc_resource_group *grp;
+ unsigned long val;
+ char *end;
+ int ret;
+
+ bc = item_to_bc(item->ci_parent)->bc;
+ grp = item_to_res(item);
+
+ ret = -EINVAL;
+ val = simple_strtoul(page, &end, 10);
+ if (*end != '\0')
+  goto out;
+
+ switch (attr_to_bc(attr)->idx) {
+ case BC_RES_BARRIER:
+  ret = bc_change_param(bc, grp->res,
+    val, bc->bc_parms[grp->res].limit);
```

```
+  if (ret == 0)
+   ret = size;
+  break;
+ case BC_RES_LIMIT:
+  ret = bc_change_param(bc, grp->res,
+    bc->bc_parms[grp->res].barrier, val);
+  if (ret == 0)
+   ret = size;
+  break;
+ }
+out:
+ return ret;
+}
+
+static void bc_release(struct config_item *item)
+{
+ kfree(to_config_group(item)->default_groups);
+ bc_put(item_to_bc(item)->bc);
+}
+
+static void resource_release(struct config_item *item)
+{
+ kfree(to_config_group(item));
+}
+
+static struct configfs_item_operations bc_item_ops = {
+ .show_attribute = bc_show,
+ .store_attribute = bc_store,
+ .release = bc_release,
+};
+
+static struct config_item_type bc_item_type = {
+ .ct_item_ops = &bc_item_ops,
+ .ct_attrs = bc_attributes,
+ .ct_owner = THIS_MODULE,
+};
+
+static struct configfs_item_operations resource_item_ops = {
+ .show_attribute = resource_show,
+ .store_attribute = resource_store,
+ .release = resource_release,
+};
+
+static struct config_item_type resource_item_type = {
+ .ct_item_ops = &resource_item_ops,
+ .ct_attrs = resource_attributes,
+ .ct_owner = THIS_MODULE,
+};
```

```
+
+static int bc_init_default_groups(struct config_group *group)
+{
+ int i;
+ struct bc_resource_group *def_group;
+
+ group->default_groups = kmalloc((BC_RESOURCES + 1) *
+   sizeof(struct config_group *), GFP_KERNEL);
+ if (group->default_groups == NULL)
+  goto out;
+
+ for (i = 0; i < BC_RESOURCES; i++) {
+  def_group = kzalloc(sizeof(struct bc_resource_group),
+    GFP_KERNEL);
+  if (def_group == NULL)
+   goto out_free;
+
+  def_group->res = i;
+  config_group_init_type_name(&def_group->group,
+    bc_resources[i]->bcr_name, &resource_item_type);
+  group->default_groups[i] = &def_group->group;
+ }
+
+ group->default_groups[i] = NULL;
+ return 0;
+
+out_free:
+ for (i--; i >= 0; i--)
+  config_group_put(group->default_groups[i]);
+
+ kfree(group->default_groups);
+out:
+ return -ENOMEM;
+}
+
+static struct config_group *bc_new_group(struct beancounter *bc,
+  const char *name)
+{
+ struct bc_group *bc_group;
+
+ bc_group = kzalloc(sizeof(struct bc_group), GFP_KERNEL);
+ if (bc_group == NULL)
+  goto out;
+
+ config_group_init_type_name(&bc_group->group, name, &bc_item_type);
+ if (bc_init_default_groups(&bc_group->group))
+  goto out_free;
+
```

```
+ bc_group->bc = bc;
+ return &bc_group->group;
+
+out_free:
+ config_group_put(&bc_group->group);
+out:
+ return NULL;
+}
+
+static struct config_group *bc_make_group(struct config_group *group,
+  const char *name)
+{
+ int id;
+ char *end;
+ struct beancounter *bc;
+ struct config_group *grp;
+
+ id = simple_strtoul(name, &end, 10);
+ if (*end != '\0')
+  goto out;
+
+ bc = bc_findcreate(id, BC_ALLOC);
+ if (bc == NULL)
+  goto out;
+
+ grp = bc_new_group(bc, name);
+ if (grp == NULL)
+  goto out_put;
+
+ return grp;
+
+out_put:
+ bc_put(bc);
+out:
+ return NULL;
+}
+
+/* subsystem description */
+
+static struct configfs_group_operations bc_group_group_ops = {
+ .make_group = bc_make_group,
+};
+
+static ssize_t bc_group_attr_show(struct config_item *item,
+  struct configfs_attribute *attr,
+  char *page)
+{
+ return sprintf(page, "Beancounters "
```

```
+   "controll resource usage of task groups\n");
+}
+
+static struct configfs_item_operations bc_group_item_ops = {
+ .show_attribute = bc_group_attr_show,
+};
+
+static struct configfs_attribute bc_attr_description = {
+ .ca_owner = THIS_MODULE,
+ .ca_name = "description",
+ .ca_mode = S_IRUGO,
+};
+
+static struct configfs_attribute *bc_group_attributes[] = {
+ &bc_attr_description,
+ NULL,
+};
+
+static struct config_item_type bc_group_type = {
+ .ct_item_ops = &bc_group_item_ops,
+ .ct_group_ops = &bc_group_group_ops,
+ .ct_attrs = bc_group_attributes,
+ .ct_owner = THIS_MODULE,
+};
+
+struct configfs_subsystem bc_subsystem = {
+ .su_group = {
+  .cg_item = {
+   .ci_namebuf = "beancounters",
+   .ci_type = &bc_group_type,
+  },
+ },
+};
+
+int __init bc_init_configfs(void)
+{
+ static struct config_group *subsys_default_groups[2];
+ struct config_group *bc_group;
+
+ bc_group = bc_new_group(&init_bc, "0");
+ if (bc_group == NULL)
+  return -ENOMEM;
+
+ subsys_default_groups[0] = bc_group;
+ subsys_default_groups[1] = NULL;
+ bc_subsystem.su_group.default_groups = subsys_default_groups;
+
+ config_group_init(&bc_subsystem.su_group);
```

```
+ init_MUTEX(&bc_subsystem.su_sem);
+
+ return configfs_register_subsystem(&bc_subsystem);
+}
+
+static void __exit bc_fini_configfs(void)
+{
+ configfs_unregister_subsystem(&bc_subsystem);
+}
+
+module_init(bc_init_configfs);
+module_exit(bc_fini_configfs);
+MODULE_LICENSE("GPL");
```
