Subject: [RFC][PATCH 1/1]a new optional function for task assignment to cgroup
Posted by Kazunaga Ikeno on Wed, 05 Mar 2008 05:41:38 GMT
View Forum Message <> Reply to Message

From: Kazunaga Ikeno <k-ikeno@ak.jp.nec.com>

This patch provides the function leads a task, corresponding to specified user-id,
to a specific cgroup directory.

Signed-off-by: Kazunaga Ikeno <k-ikeno@ak.jp.nec.com>

Index: linux-2.6.25-rc3-mm1/kernel/sys.c
==========
======
--- linux-2.6.25-rc3-mm1.orig/kernel/sys.c 2008-03-04 21:43:11.000000000 +0900
+++ linux-2.6.25-rc3-mm1/kernel/sys.c 2008-03-04 21:55:34.000000000 +0900
@@ -33,6 +33,7 @@
 #include <linux/task_io_accounting_ops.h>
 #include <linux/seccomp.h>
 #include <linux/cpu.h>
+#include <linux/cgroup.h>

 #include <linux/compat.h>
 #include <linux/syscalls.h>
@@ -644,6 +645,10 @@ asmlinkage long sys_setreuid(uid_t ruid,
  key_fsuid_changed(current);
  proc_id_connector(current, PROC_EVENT_UID);

+ retval = cgroup_attach_by_uid(current);
+ if (retval)
+ return retval;
+
  return security_task_post_setuid(old_ruid, old_euid, old_suid, LSM_SETID_RE);
 }

@@ -691,6 +696,10 @@ asmlinkage long sys_setuid(uid_t uid)
  key_fsuid_changed(current);
  proc_id_connector(current, PROC_EVENT_UID);

+ retval = cgroup_attach_by_uid(current);
+ if (retval)
+ return retval;
+
  return security_task_post_setuid(old_ruid, old_euid, old_suid, LSM_SETID_ID);
 }

@@ -739,6 +748,10 @@ asmlinkage long sys_setresuid(uid_t ruid

```
  key_fsuid_changed(current);
  proc_id_connector(current, PROC_EVENT_UID);

+ retval = cgroup_attach_by_uid(current);
+ if (retval)
+  return retval;
+
  return security_task_post_setuid(old_ruid, old_euid, old_suid, LSM_SETID_RES);
 }


Index: linux-2.6.25-rc3-mm1/include/linux/cgroup.h
===================================================================
======
--- linux-2.6.25-rc3-mm1.orig/include/linux/cgroup.h 2008-03-04 21:43:10.000000000 +0900
+++ linux-2.6.25-rc3-mm1/include/linux/cgroup.h 2008-03-04 21:57:32.000000000 +0900
@@ -30,6 +30,7 @@ extern void cgroup_unlock(void);
 extern void cgroup_fork(struct task_struct *p);
 extern void cgroup_fork_callbacks(struct task_struct *p);
 extern void cgroup_post_fork(struct task_struct *p);
+extern int cgroup_attach_by_uid(struct task_struct *p);
 extern void cgroup_exit(struct task_struct *p, int run_callbacks);
 extern int cgroupstats_build(struct cgroupstats *stats,
     struct dentry *dentry);
@@ -356,6 +357,7 @@ static inline void cgroup_init_smp(void)
 static inline void cgroup_fork(struct task_struct *p) {}
 static inline void cgroup_fork_callbacks(struct task_struct *p) {}
 static inline void cgroup_post_fork(struct task_struct *p) {}
+static inline int cgroup_attach_by_uid(struct task_struct *p) {}
 static inline void cgroup_exit(struct task_struct *p, int callbacks) {}

 static inline void cgroup_lock(void) {}

Index: linux-2.6.25-rc3-mm1/kernel/cgroup.c
===================================================================
======
--- linux-2.6.25-rc3-mm1.orig/kernel/cgroup.c 2008-03-04 21:43:11.000000000 +0900
+++ linux-2.6.25-rc3-mm1/kernel/cgroup.c 2008-03-04 22:02:42.000000000 +0900
@@ -94,6 +94,9 @@ struct cgroupfs_root {
   * notification. We ensure that it's always a valid
   * NUL-terminated string */
  char release_agent_path[PATH_MAX];
+
+ /* A list for the option that leads a task to cgroup */
+ struct list_head leadopt_list;
 };
```

```
@@ -177,6 +180,15 @@ struct cg_cgroup_link {
  struct css_set *cg;
 };

+/* Link structure for the option that leads a task to cgroup */
+struct leadopt_entry {
+ struct list_head leadopt_link;
+ /* user-id of task to lead */
+ uid_t uid;
+ /* pathname of a cgroup directory */
+ char *pathname;
+};
+
 /* The default css_set - used by init and its children prior to any
  * hierarchies being mounted. It contains a pointer to the root state
  * for each subsystem. Also used to anchor the list of css_sets. Not
@@ -854,6 +866,7 @@ static void init_cgroup_root(struct cgro
  struct cgroup *cgrp = &root->top_cgroup;
  INIT_LIST_HEAD(&root->subsys_list);
  INIT_LIST_HEAD(&root->root_list);
+ INIT_LIST_HEAD(&root->leadopt_list);
  root->number_of_cgroups = 1;
  cgrp->root = root;
  cgrp->top_cgroup = cgrp;
@@ -1282,6 +1295,166 @@ static int attach_task_by_pid(struct cgr
  return ret;
 }

+/*
+ * Lead a task, corresponding to specified user-id, to a specific
+ * cgroup directory.
+ */
+int cgroup_attach_by_uid(struct task_struct *tsk)
+{
+ struct cgroupfs_root *root;
+ struct list_head *l;
+ struct nameidata nd;
+ int ret = 0;
+ char *cgpath;
+
+ mutex_lock(&cgroup_mutex);
+ for_each_root(root) {
+  /* Skip this hierarchy if it has no active subsystems */
+  if (!root->actual_subsys_bits)
+   continue;
+  cgpath = 0;
+  l = root->leadopt_list.next;
+  while (l != &root->leadopt_list) {
```

```
+   struct leadopt_entry *link =
+    list_entry(l,
+     struct leadopt_entry, leadopt_link);
+   if (tsk->suid == link->uid)
+    cgpath = link->pathname;
+   else if (!cgpath && (tsk->euid == link->uid))
+    cgpath = link->pathname;
+   l = l->next;
+  }
+  if (cgpath) {
+   ret = path_lookup(cgpath, LOOKUP_DIRECTORY, &nd);
+   if (ret)
+    goto err0;
+   if (nd.path.dentry->d_fsdata == NULL) {
+    ret = -EPERM;
+    goto err1;
+   }
+   ret = cgroup_attach_task(
+    (struct cgroup *)(nd.path.dentry->d_fsdata),
+         tsk);
+   if (ret)
+    goto err1;
+   path_put(&nd.path);
+  }
+ }
+err0:
+ mutex_unlock(&cgroup_mutex);
+ return ret;
+
+err1:
+ path_put(&nd.path);
+ mutex_unlock(&cgroup_mutex);
+ return ret;
+}
+
+static int cgroup_leadopt_file_write(struct cgroupfs_root *root, char *buffer)
+{
+ struct leadopt_entry *link;
+ struct list_head *l;
+ int ret = 1;
+ int erase = 1;
+ char *options, *value, *dirto = NULL;
+ uid_t uid;
+
+ /* parse a command line */
+ while ((options = strsep(&buffer, " ,:")) != NULL) {
+  if (*options == (char)NULL) {
+   continue;
```

```
+  } else if (strcmp(options, "uid") == 0) {
+   while ((value = strsep(&buffer, " ,:")) != NULL) {
+    if (*value == (char)NULL)
+     continue;
+    if (sscanf(value, "%d", &uid) != 1)
+     return -EIO;
+    break;
+   }
+  } else if (strcmp(options, "leadto") == 0) {
+   while ((dirto = strsep(&buffer, " ,:")) != NULL) {
+    if (*dirto == (char)NULL)
+     continue;
+    erase = 0;
+    break;
+   }
+  }
+ }
+
+ /* Update a list for the option that leads a task to cgroup */
+ l = root->leadopt_list.next;
+ while (l != &root->leadopt_list) {
+  link = list_entry(l, struct leadopt_entry, leadopt_link);
+  if (link->uid == uid) {
+   list_del(&link->leadopt_link);
+   kfree(link->pathname);
+   kfree(link);
+   if (!erase) {
+    link = kmalloc(sizeof(*link), GFP_KERNEL);
+    if (link == NULL) {
+     ret = -ENOMEM;
+     goto err0;
+    }
+    link->uid = uid;
+    /* +1 for nul-terminator */
+    link->pathname = kmalloc((strlen(dirto)+1),
+        GFP_KERNEL);
+    strcpy(link->pathname, dirto);
+    list_add(&link->leadopt_link,
+       &root->leadopt_list);
+   }
+   break;
+  }
+  l = l->next;
+ }
+ if (!erase && (l == &root->leadopt_list)) {
+  link = kmalloc(sizeof(*link), GFP_KERNEL);
+  if (link == NULL) {
+   ret = -ENOMEM;
```

```
+   goto err0;
+  }
+  link->uid = uid;
+  /* +1 for nul-terminator */
+  link->pathname = kmalloc((strlen(dirto)+1), GFP_KERNEL);
+  strcpy(link->pathname, dirto);
+  list_add(&link->leadopt_link, &root->leadopt_list);
+ }
+
+err0:
+ return ret;
+}
+
+static ssize_t cgroup_leadopt_read(struct cgroup *cgrp,
+        struct cftype *cft,
+        struct file *file, char __user *buf,
+        size_t nbytes, loff_t *ppos)
+{
+ char *page;
+ ssize_t retval = 0;
+ struct cgroupfs_root *root = cgrp->root;
+ struct list_head *l;
+ int cnt = 0;
+
+ page = (char *)__get_free_page(GFP_KERNEL);
+ if (!page)
+  return -ENOMEM;
+
+ /* truncate a buffer beyond page size */
+ l = root->leadopt_list.next;
+ while (l != &root->leadopt_list) {
+  struct leadopt_entry *link =
+   list_entry(l, struct leadopt_entry, leadopt_link);
+  cnt += snprintf(page + cnt, max((int)PAGE_SIZE - cnt, 0),
+      "uid:%4d\tleadto:%s\n",
+      link->uid, link->pathname);
+  l = l->next;
+ }
+ retval = simple_read_from_buffer(buf, nbytes, ppos, page, strlen(page));
+
+ free_page((unsigned long)page);
+ return retval;
+}
+
 /* The various types of files and directories in a cgroup file system */
 enum cgroup_filetype {
  FILE_ROOT,
@@ -1289,6 +1462,7 @@ enum cgroup_filetype {
```

```
  FILE_TASKLIST,
  FILE_NOTIFY_ON_RELEASE,
  FILE_RELEASE_AGENT,
+ FILE_LEAD_OPTION,
 };

 static ssize_t cgroup_write_u64(struct cgroup *cgrp, struct cftype *cft,
@@ -1372,6 +1546,12 @@ static ssize_t cgroup_common_file_write(
   BUILD_BUG_ON(sizeof(cgrp->root->release_agent_path) < PATH_MAX);
   strcpy(cgrp->root->release_agent_path, buffer);
   break;
+ case FILE_LEAD_OPTION:
+  if (cgroup_leadopt_file_write(cgrp->root, buffer) != 1) {
+   retval = -EINVAL;
+   goto out2;
+  }
+  break;
  default:
   retval = -EINVAL;
   goto out2;
@@ -2189,6 +2369,13 @@ static struct cftype cft_release_agent =
  .private = FILE_RELEASE_AGENT,
 };

+static struct cftype cft_lead_option = {
+ .name = "lead_option",
+ .read = cgroup_leadopt_read,
+ .write = cgroup_common_file_write,
+ .private = FILE_LEAD_OPTION,
+};
+
 static int cgroup_populate_dir(struct cgroup *cgrp)
 {
  int err;
@@ -2206,6 +2393,11 @@ static int cgroup_populate_dir(struct cg
    return err;
  }

+ if (cgrp == cgrp->top_cgroup) {
+  if ((err = cgroup_add_file(cgrp, NULL, &cft_lead_option)) < 0)
+   return err;
+ }
+
  for_each_subsys(cgrp->root, ss) {
   if (ss->populate && (err = ss->populate(ss, cgrp)) < 0)
    return err;
--
-
```