

---

Subject: [RFC][PATCH 1/2] add user namespace [try #2]  
Posted by [Cedric Le Goater](#) on Mon, 28 Aug 2006 14:56:44 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

This patch adds the user namespace.

Basically, it allows a process to unshare its user\_struct table, resetting at the same time its own user\_struct and all the associated accounting.

A new root user (uid == 0) is added to the user namespace upon creation. Such root users have full privileges and it seems that theses privileges should be controlled through some means (process capabilities ?)

Changes [try #2]

- removed struct user\_namespace\* argument from find\_user()
- added a root\_user per user namespace

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>  
Cc: Andrew Morton <akpm@osdl.org>  
Cc: Kirill Korotaev <dev@openvz.org>  
Cc: Eric W. Biederman <ebiederm@xmission.com>  
Cc: Herbert Poetzl <herbert@13thfloor.at>  
Cc: Serge E. Hallyn <serue@us.ibm.com>  
Cc: Dave Hansen <haveblue@us.ibm.com>

---  
include/linux/init\_task.h | 2  
include/linux/nsproxy.h | 2  
include/linux/sched.h | 4 +  
include/linux/user.h | 46 ++++++  
init/Kconfig | 8 ++  
kernel/fork.c | 2  
kernel/nsproxy.c | 15 ++++  
kernel/sys.c | 5 +  
kernel/user.c | 133  
+++++-----  
9 files changed, 203 insertions(+), 14 deletions(-)

Index: 2.6.18-rc4-mm3/kernel/user.c

=====

--- 2.6.18-rc4-mm3.orig/kernel/user.c  
+++ 2.6.18-rc4-mm3/kernel/user.c  
@@ -14,20 +14,29 @@  
#include <linux/bitops.h>  
#include <linux/key.h>



```

+ if (ns) {
+   int n;
+   struct user_struct *new_user;
+
+   kref_init(&ns->kref);
+
+   for(n = 0; n < UIDHASH_SZ; ++n)
+     INIT_LIST_HEAD(ns->uidhash_table + n);
+
+   /* Insert new root user. */
+   ns->root_user = alloc_uid(ns, 0);
+   if (!ns->root_user) {
+     kfree(ns);
+     return NULL;
+   }
+
+   /* Reset current->user with a new one */
+   new_user = alloc_uid(ns, current->uid);
+   if (!new_user) {
+     kfree(ns);
+     return NULL;
+   }
+
+   switch_uid(new_user);
+ }
+ return ns;
+}
+
+/*
+ * unshare the current process' user namespace.
+ */
+int unshare_user_ns(unsigned long unshare_flags,
+   struct user_namespace **new_user)
+{
+   if (unshare_flags & CLONE_NEWUSER) {
+     if (!capable(CAP_SYS_ADMIN))
+       return -EPERM;
+
+     *new_user = clone_user_ns(current->nsproxy->user_ns);
+     if (!*new_user)
+       return -ENOMEM;
+   }
+
+   return 0;
+}
+
+/*
+ * Copy task tsk's user namespace, or clone it if flags specifies

```

```

+ * CLONE_NEWUSER. In latter case, changes to the user namespace of
+ * this process won't be seen by parent, and vice versa.
+ */
+int copy_user_ns(int flags, struct task_struct *tsk)
+{
+ struct user_namespace *old_ns = tsk->nsproxy->user_ns;
+ struct user_namespace *new_ns;
+ int err = 0;
+
+ if (!old_ns)
+ return 0;
+
+ get_user_ns(old_ns);
+
+ if (!(flags & CLONE_NEWUSER))
+ return 0;
+
+ if (!capable(CAP_SYS_ADMIN)) {
+ err = -EPERM;
+ goto out;
+ }
+
+ new_ns = clone_user_ns(old_ns);
+ if (!new_ns) {
+ err = -ENOMEM;
+ goto out;
+ }
+ tsk->nsproxy->user_ns = new_ns;
+
+out:
+ put_user_ns(old_ns);
+ return err;
+}
+
+void free_user_ns(struct kref *kref)
+{
+ struct user_namespace *ns;
+
+ ns = container_of(kref, struct user_namespace, kref);
+ kfree(ns);
+}
+
+/* CONFIG_USER_NS */
+
+/*
+ * Locate the user_struct for the passed UID. If found, take a ref on it.
+ * The
+ * caller must undo that ref with free_uid().

```

```

@@ -94,9 +208,10 @@ struct user_struct *find_user(uid_t uid)
{
    struct user_struct *ret;
    unsigned long flags;
+ struct user_namespace *ns = current->nsproxy->user_ns;

    spin_lock_irqsave(&uidhash_lock, flags);
- ret = uid_hash_find(uid, uidhashentry(uid));
+ ret = uid_hash_find(uid, uidhashentry(ns, uid));
    spin_unlock_irqrestore(&uidhash_lock, flags);
    return ret;
}
@@ -120,9 +235,9 @@ void free_uid(struct user_struct *up)
}
}

-struct user_struct * alloc_uid(uid_t uid)
+struct user_struct * alloc_uid(struct user_namespace *ns, uid_t uid)
{
- struct list_head *hashent = uidhashentry(uid);
+ struct list_head *hashent = uidhashentry(ns, uid);
    struct user_struct *up;

    spin_lock_irq(&uidhash_lock);
@@ -200,11 +315,11 @@ static int __init uid_cache_init(void)
    0, SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);

    for(n = 0; n < UIDHASH_SZ; ++n)
- INIT_LIST_HEAD(uidhash_table + n);
+ INIT_LIST_HEAD(init_user_ns.uidhash_table + n);

    /* Insert the root user immediately (init already runs as root) */
    spin_lock_irq(&uidhash_lock);
- uid_hash_insert(&root_user, uidhashentry(0));
+ uid_hash_insert(&root_user, uidhashentry(&init_user_ns, 0));
    spin_unlock_irq(&uidhash_lock);

    return 0;
Index: 2.6.18-rc4-mm3/include/linux/nsproxy.h
=====
--- 2.6.18-rc4-mm3.orig/include/linux/nsproxy.h
+++ 2.6.18-rc4-mm3/include/linux/nsproxy.h
@@ -7,6 +7,7 @@
    struct namespace;
    struct uts_namespace;
    struct ipc_namespace;
+struct user_namespace;

```

```

/*
 * A structure to contain pointers to all per-process
@@ -25,6 +26,7 @@ struct nsproxy {
    spinlock_t nslock;
    struct uts_namespace *uts_ns;
    struct ipc_namespace *ipc_ns;
+ struct user_namespace *user_ns;
    struct namespace *namespace;
};
extern struct nsproxy init_nsproxy;
Index: 2.6.18-rc4-mm3/include/linux/user.h
=====
--- 2.6.18-rc4-mm3.orig/include/linux/user.h
+++ 2.6.18-rc4-mm3/include/linux/user.h
@@ -1 +1,47 @@
+#ifndef _LINUX_USER_H
+#define _LINUX_USER_H
+
+ #include <asm/user.h>
+
+#define UIDHASH_BITS (CONFIG_BASE_SMALL ? 3 : 8)
+#define UIDHASH_SZ (1 << UIDHASH_BITS)
+
+struct user_namespace {
+ struct kref kref;
+ struct list_head uidhash_table[UIDHASH_SZ];
+ struct user_struct *root_user;
+};
+
+extern struct user_namespace init_user_ns;
+
+static inline void get_user_ns(struct user_namespace *ns)
+{
+ kref_get(&ns->kref);
+}
+
+#ifdef CONFIG_USER_NS
+extern int unshare_user_ns(unsigned long unshare_flags,
+ struct user_namespace **new_user);
+extern int copy_user_ns(int flags, struct task_struct *tsk);
+extern void free_user_ns(struct kref *kref);
+
+static inline void put_user_ns(struct user_namespace *ns)
+{
+ kref_put(&ns->kref, free_user_ns);
+}
+#else
+static inline int unshare_user_ns(unsigned long unshare_flags,

```

```
+ struct user_namespace **new_user)
+{
+ return -EINVAL;
+}
+static inline int copy_user_ns(int flags, struct task_struct *tsk)
+{
+ return 0;
+}
+static inline void put_user_ns(struct user_namespace *ns)
+{
+}
+}
+
+/* CONFIG_USER_NS */
+
+/* _LINUX_USER_H */
Index: 2.6.18-rc4-mm3/kernel/nsproxy.c
```

```

=====
--- 2.6.18-rc4-mm3.orig/kernel/nsproxy.c
+++ 2.6.18-rc4-mm3/kernel/nsproxy.c
@@ -19,6 +19,7 @@
#include <linux/init_task.h>
#include <linux/namespace.h>
#include <linux/utsname.h>
+#include <linux/user.h>

```

```
struct nsproxy init_nsproxy = INIT_NS_PROXY(init_nsproxy);
```

```
return ns;
@@ -88,7 +91,8 @@ int copy_namespaces(int flags, struct ta
```

```
get_nsproxy(old_ns);

- if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC)))
+ if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
+ CLONE_NEWUSER)))
    return 0;
```

```
new_ns = clone_namespaces(old_ns);
@@ -111,10 +115,17 @@ int copy_namespaces(int flags, struct ta
if (err)
    goto out_ipc;
```

```

+ err = copy_user_ns(flags, tsk);
+ if (err)
+ goto out_user;
+
out:
    put_nsproxy(old_ns);
    return err;

+out_user:
+ if (new_ns->ipc_ns)
+ put_ipc_ns(new_ns->ipc_ns);
out_ipc:
    if (new_ns->uts_ns)
        put_uts_ns(new_ns->uts_ns);
@@ -135,5 +146,7 @@ void free_nsproxy(struct nsproxy *ns)
    put_uts_ns(ns->uts_ns);
    if (ns->ipc_ns)
        put_ipc_ns(ns->ipc_ns);
+ if (ns->user_ns)
+ put_user_ns(ns->user_ns);
    kfree(ns);
}

```

Index: 2.6.18-rc4-mm3/include/linux/sched.h

```

=====
--- 2.6.18-rc4-mm3.orig/include/linux/sched.h
+++ 2.6.18-rc4-mm3/include/linux/sched.h
@@ -26,6 +26,7 @@
#define CLONE_STOPPED 0x02000000 /* Start in stopped state */
#define CLONE_NEWUTS 0x04000000 /* New utsname group? */
#define CLONE_NEWIPC 0x08000000 /* New ipcs */
+#define CLONE_NEWUSER 0x10000000 /* New user */

/*
 * Scheduling policies
@@ -242,6 +243,7 @@ extern signed long schedule_timeout_unin
asmlinkage void schedule(void);

struct nsproxy;
+struct user_namespace;

/* Maximum number of active map areas.. This is a random (large) number */
#define DEFAULT_MAX_MAP_COUNT 65536
@@ -1249,7 +1251,7 @@ extern void set_special_pids(pid_t sessi
extern void __set_special_pids(pid_t session, pid_t pgrp);

/* per-UID process charging. */
-extern struct user_struct * alloc_uid(uid_t);

```



```
+extern struct user_struct * alloc_uid(struct user_namespace *, uid_t);
static inline struct user_struct *get_uid(struct user_struct *u)
{
    atomic_inc(&u->__count);
```

Index: 2.6.18-rc4-mm3/init/Kconfig

```
=====
--- 2.6.18-rc4-mm3.orig/init/Kconfig
+++ 2.6.18-rc4-mm3/init/Kconfig
@@ -250,6 +250,14 @@ config UTS_NS
     vservers, to use uts namespaces to provide different
     uts info for different servers. If unsure, say N.
```

```
+config USER_NS
+ bool "User Namespaces"
+ default n
+ help
+ Support user namespaces. This allows containers, i.e.
+ vservers, to use user namespaces to provide different
+ user info for different servers. If unsure, say N.
+
```

```
config AUDIT
    bool "Auditing support"
    depends on NET
```

Index: 2.6.18-rc4-mm3/include/linux/init\_task.h

```
=====
--- 2.6.18-rc4-mm3.orig/include/linux/init_task.h
+++ 2.6.18-rc4-mm3/include/linux/init_task.h
@@ -7,6 +7,7 @@
#include <linux/utsname.h>
#include <linux/lockdep.h>
#include <linux/ipc.h>
+#include <linux/user.h>

#define INIT_FDTABLE \
{ \
@@ -77,6 +78,7 @@ extern struct nsproxy init_nsproxy;
    .uts_ns = &init_uts_ns, \
    .namespace = NULL, \
    INIT_IPC_NS(ipc_ns) \
+ .user_ns = &init_user_ns, \
}
```

```
#define INIT_SIGHAND(sighand) { \
```

Index: 2.6.18-rc4-mm3/kernel/sys.c

```
=====
--- 2.6.18-rc4-mm3.orig/kernel/sys.c
+++ 2.6.18-rc4-mm3/kernel/sys.c
@@ -33,6 +33,7 @@
```

```

#include <linux/compat.h>
#include <linux/syscalls.h>
#include <linux/kprobes.h>
+#include <linux/user.h>

#include <asm/uaccess.h>
#include <asm/io.h>
@@ -1010,13 +1011,13 @@ static int set_user(uid_t new_ruid, int
{
    struct user_struct *new_user;

- new_user = alloc_uid(new_ruid);
+ new_user = alloc_uid(current->nsproxy->user_ns, new_ruid);
    if (!new_user)
        return -EAGAIN;

    if (atomic_read(&new_user->processes) >=
        current->signal->rlim[RLIMIT_NPROC].rlim_cur &&
- new_user != &root_user) {
+ new_user != current->nsproxy->user_ns->root_user) {
        free_uid(new_user);
        return -EAGAIN;
    }

```

Index: 2.6.18-rc4-mm3/kernel/fork.c

```

=====
--- 2.6.18-rc4-mm3.orig/kernel/fork.c
+++ 2.6.18-rc4-mm3/kernel/fork.c
@@ -991,7 +991,7 @@ static struct task_struct *copy_process(
    if (atomic_read(&p->user->processes) >=
        p->signal->rlim[RLIMIT_NPROC].rlim_cur) {
        if (!capable(CAP_SYS_ADMIN) && !capable(CAP_SYS_RESOURCE) &&
- p->user != &root_user)
+ p->user != current->nsproxy->user_ns->root_user)
            goto bad_fork_free;
    }

```

---

Containers mailing list  
Containers@lists.osdl.org  
<https://lists.osdl.org/mailman/listinfo/containers>

---