Subject: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by akpm on Wed, 09 May 2007 02:45:35 GMT
View Forum Message <> Reply to Message

The patch titled
    Merge sys_clone()/sys_unshare() nsproxy and namespace handling
has been removed from the -mm tree. Its filename was
    merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch

This patch was dropped because it was merged into mainline or a subsystem tree

---
Subject: Merge sys_clone()/sys_unshare() nsproxy and namespace handling
From: Badari Pulavarty <pbadari@us.ibm.com>

sys_clone() and sys_unshare() both makes copies of nsproxy and its associated
namespaces. But they have different code paths.

This patch merges all the nsproxy and its associated namespace copy/clone
handling (as much as possible). Posted on container list earlier for
feedback.

- Create a new nsproxy and its associated namespaces and pass it back to
  caller to attach it to right process.

- Changed all copy_*_ns() routines to return a new copy of namespace
  instead of attaching it to task->nsproxy.

- Moved the CAP_SYS_ADMIN checks out of copy_*_ns() routines.

- Removed unnessary !ns checks from copy_*_ns() and added BUG_ON()
  just incase.

- Get rid of all individual unshare_*_ns() routines and make use of
  copy_*_ns() instead.

[akpm@osdl.org: cleanups, warning fix]
[clg@fr.ibm.com: remove dup_namespaces() declaration]
[serue@us.ibm.com: fix CONFIG_IPC_NS=n, clone(CLONE_NEWIPC) retval]
[akpm@linux-foundation.org: fix build with CONFIG_SYSVIPC=n]
Signed-off-by: Badari Pulavarty <pbadari@us.ibm.com>
Signed-off-by: Serge Hallyn <serue@us.ibm.com>
Cc: Cedric Le Goater <clg@fr.ibm.com>
Cc: "Eric W. Biederman" <ebiederm@xmission.com>
Cc: <containers@lists.osdl.org>
Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>

```
Cc: Oleg Nesterov <oleg@tv-sign.ru>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>
---

 fs/namespace.c               |   30 +-----
 include/linux/ipc.h          |   11 +-
 include/linux/mnt_namespace.h |    5 -
 include/linux/nsproxy.h      |    3
 include/linux/pid_namespace.h |    2
 include/linux/utsname.h      |   19 ----
 ipc/util.c                   |   53 ++----------
 kernel/fork.c                |   85 +------------------
 kernel/nsproxy.c             |  139 ++++++++++++++++++++++++++++++++-----------
 kernel/pid.c                 |   11 --
 kernel/utsname.c             |   41 ---------
 11 files changed, 131 insertions(+), 268 deletions(-)

diff -puN fs/namespace.c~merge-sys_clone-sys_unshare-nsproxy-and-namespace
fs/namespace.c
--- a/fs/namespace.c~merge-sys_clone-sys_unshare-nsproxy-and-namespace
+++ a/fs/namespace.c
@@ -1441,10 +1441,9 @@ dput_out:
  * Allocate a new namespace structure and populate it with contents
  * copied from the namespace of the passed in task structure.
  */
-struct mnt_namespace *dup_mnt_ns(struct task_struct *tsk,
+static struct mnt_namespace *dup_mnt_ns(struct mnt_namespace *mnt_ns,
   struct fs_struct *fs)
 {
- struct mnt_namespace *mnt_ns = tsk->nsproxy->mnt_ns;
  struct mnt_namespace *new_ns;
  struct vfsmount *rootmnt = NULL, *pwdmnt = NULL, *altrootmnt = NULL;
  struct vfsmount *p, *q;
@@ -1509,36 +1508,21 @@ struct mnt_namespace *dup_mnt_ns(struct
  return new_ns;
 }

-int copy_mnt_ns(int flags, struct task_struct *tsk)
+struct mnt_namespace *copy_mnt_ns(int flags, struct mnt_namespace *ns,
+  struct fs_struct *new_fs)
 {
- struct mnt_namespace *ns = tsk->nsproxy->mnt_ns;
  struct mnt_namespace *new_ns;
- int err = 0;
-
- if (!ns)
-  return 0;
```

```
+ BUG_ON(!ns);
  get_mnt_ns(ns);

  if (!(flags & CLONE_NEWNS))
-  return 0;
+  return ns;

- if (!capable(CAP_SYS_ADMIN)) {
-  err = -EPERM;
-  goto out;
- }
+ new_ns = dup_mnt_ns(ns, new_fs);

- new_ns = dup_mnt_ns(tsk, tsk->fs);
- if (!new_ns) {
-  err = -ENOMEM;
-  goto out;
- }
-
- tsk->nsproxy->mnt_ns = new_ns;
-
-out:
  put_mnt_ns(ns);
- return err;
+ return new_ns;
 }

 asmlinkage long sys_mount(char __user * dev_name, char __user * dir_name,
diff -puN include/linux/ipc.h~merge-sys_clone-sys_unshare-nsproxy-and-namespace
include/linux/ipc.h
--- a/include/linux/ipc.h~merge-sys_clone-sys_unshare-nsproxy-and-namespace
+++ a/include/linux/ipc.h
@@ -92,16 +92,19 @@ extern struct ipc_namespace init_ipc_ns;

 #ifdef CONFIG_SYSVIPC
 #define INIT_IPC_NS(ns)  .ns  = &init_ipc_ns,
-extern int copy_ipcs(unsigned long flags, struct task_struct *tsk);
+extern struct ipc_namespace *copy_ipcs(unsigned long flags,
+       struct ipc_namespace *ns);
 #else
 #define INIT_IPC_NS(ns)
-static inline int copy_ipcs(unsigned long flags, struct task_struct *tsk)
-{ return 0; }
+static inline struct ipc_namespace *copy_ipcs(unsigned long flags,
+       struct ipc_namespace *ns)
+{
+ return ns;
+}
```

```
 #endif

 #ifdef CONFIG_IPC_NS
 extern void free_ipc_ns(struct kref *kref);
-extern int unshare_ipcs(unsigned long flags, struct ipc_namespace **ns);
 #endif

 static inline struct ipc_namespace *get_ipc_ns(struct ipc_namespace *ns)
diff -puN
include/linux/mnt_namespace.h~merge-sys_clone-sys_unshare-nsproxy-and-namespace
include/linux/mnt_namespace.h
--- a/include/linux/mnt_namespace.h~merge-sys_clone-sys_unshare-nsproxy-and-namespace
+++ a/include/linux/mnt_namespace.h
@@ -14,10 +14,9 @@ struct mnt_namespace {
  int event;
 };

-extern int copy_mnt_ns(int, struct task_struct *);
-extern void __put_mnt_ns(struct mnt_namespace *ns);
-extern struct mnt_namespace *dup_mnt_ns(struct task_struct *,
+extern struct mnt_namespace *copy_mnt_ns(int, struct mnt_namespace *,
   struct fs_struct *);
+extern void __put_mnt_ns(struct mnt_namespace *ns);

 static inline void put_mnt_ns(struct mnt_namespace *ns)
 {
diff -puN include/linux/nsproxy.h~merge-sys_clone-sys_unshare-nsproxy-and-namespace
include/linux/nsproxy.h
--- a/include/linux/nsproxy.h~merge-sys_clone-sys_unshare-nsproxy-and-namespace
+++ a/include/linux/nsproxy.h
@@ -31,10 +31,11 @@ struct nsproxy {
 };
 extern struct nsproxy init_nsproxy;

-struct nsproxy *dup_namespaces(struct nsproxy *orig);
 int copy_namespaces(int flags, struct task_struct *tsk);
 void get_task_namespaces(struct task_struct *tsk);
 void free_nsproxy(struct nsproxy *ns);
+int unshare_nsproxy_namespaces(unsigned long, struct nsproxy **,
+ struct fs_struct *);

 static inline void put_nsproxy(struct nsproxy *ns)
 {
diff -puN include/linux/pid_namespace.h~merge-sys_clone-sys_unshare-nsproxy-and-namespace
include/linux/pid_namespace.h
--- a/include/linux/pid_namespace.h~merge-sys_clone-sys_unshare-nsproxy-and-namespace
+++ a/include/linux/pid_namespace.h
@@ -29,7 +29,7 @@ static inline void get_pid_ns(struct pid
```

```
  kref_get(&ns->kref);
 }

-extern int copy_pid_ns(int flags, struct task_struct *tsk);
+extern struct pid_namespace *copy_pid_ns(int flags, struct pid_namespace *ns);
 extern void free_pid_ns(struct kref *kref);

 static inline void put_pid_ns(struct pid_namespace *ns)
diff -puN include/linux/utsname.h~merge-sys_clone-sys_unshare-nsproxy-and-namespace
include/linux/utsname.h
--- a/include/linux/utsname.h~merge-sys_clone-sys_unshare-nsproxy-and-namespace
+++ a/include/linux/utsname.h
@@ -49,9 +49,7 @@ static inline void get_uts_ns(struct uts
 }

 #ifdef CONFIG_UTS_NS
-extern int unshare_utsname(unsigned long unshare_flags,
-    struct uts_namespace **new_uts);
-extern int copy_utsname(int flags, struct task_struct *tsk);
+extern struct uts_namespace *copy_utsname(int flags, struct uts_namespace *ns);
 extern void free_uts_ns(struct kref *kref);

 static inline void put_uts_ns(struct uts_namespace *ns)
@@ -59,21 +57,12 @@ static inline void put_uts_ns(struct uts
  kref_put(&ns->kref, free_uts_ns);
 }
 #else
-static inline int unshare_utsname(unsigned long unshare_flags,
-   struct uts_namespace **new_uts)
+static inline struct uts_namespace *copy_utsname(int flags,
+       struct uts_namespace *ns)
 {
- if (unshare_flags & CLONE_NEWUTS)
-  return -EINVAL;
-
- return 0;
+ return ns;
 }

-static inline int copy_utsname(int flags, struct task_struct *tsk)
-{
- if (flags & CLONE_NEWUTS)
-  return -EINVAL;
- return 0;
-}
 static inline void put_uts_ns(struct uts_namespace *ns)
 {
 }
```

```
diff -puN ipc/util.c~merge-sys_clone-sys_unshare-nsproxy-and-namespace ipc/util.c
--- a/ipc/util.c~merge-sys_clone-sys_unshare-nsproxy-and-namespace
+++ a/ipc/util.c
@@ -85,53 +85,20 @@ err_mem:
  return ERR_PTR(err);
 }

-int unshare_ipcs(unsigned long unshare_flags, struct ipc_namespace **new_ipc)
+struct ipc_namespace *copy_ipcs(unsigned long flags, struct ipc_namespace *ns)
 {
- struct ipc_namespace *new;
-
- if (unshare_flags & CLONE_NEWIPC) {
-  if (!capable(CAP_SYS_ADMIN))
-   return -EPERM;
-
-  new = clone_ipc_ns(current->nsproxy->ipc_ns);
-  if (IS_ERR(new))
-   return PTR_ERR(new);
-
-  *new_ipc = new;
- }
-
- return 0;
-}
-
-int copy_ipcs(unsigned long flags, struct task_struct *tsk)
-{
- struct ipc_namespace *old_ns = tsk->nsproxy->ipc_ns;
  struct ipc_namespace *new_ns;
- int err = 0;

- if (!old_ns)
-  return 0;
-
- get_ipc_ns(old_ns);
+ BUG_ON(!ns);
+ get_ipc_ns(ns);

  if (!(flags & CLONE_NEWIPC))
-  return 0;
-
- if (!capable(CAP_SYS_ADMIN)) {
-  err = -EPERM;
-  goto out;
- }
+  return ns;
```

```
- new_ns = clone_ipc_ns(old_ns);
- if (!new_ns) {
-  err = -ENOMEM;
-  goto out;
- }
+ new_ns = clone_ipc_ns(ns);

- tsk->nsproxy->ipc_ns = new_ns;
-out:
- put_ipc_ns(old_ns);
- return err;
+ put_ipc_ns(ns);
+ return new_ns;
 }

 void free_ipc_ns(struct kref *kref)
@@ -145,11 +112,11 @@ void free_ipc_ns(struct kref *kref)
  kfree(ns);
 }
 #else
-int copy_ipcs(unsigned long flags, struct task_struct *tsk)
+struct ipc_namespace *copy_ipcs(unsigned long flags, struct ipc_namespace *ns)
 {
  if (flags & CLONE_NEWIPC)
-  return -EINVAL;
- return 0;
+  return ERR_PTR(-EINVAL);
+ return ns;
 }
 #endif

diff -puN kernel/fork.c~merge-sys_clone-sys_unshare-nsproxy-and-namespace kernel/fork.c
--- a/kernel/fork.c~merge-sys_clone-sys_unshare-nsproxy-and-namespace
+++ a/kernel/fork.c
@@ -1516,26 +1516,6 @@ static int unshare_fs(unsigned long unsh
 }

 /*
- * Unshare the mnt_namespace structure if it is being shared
- */
-static int unshare_mnt_namespace(unsigned long unshare_flags,
-  struct mnt_namespace **new_nsp, struct fs_struct *new_fs)
-{
- struct mnt_namespace *ns = current->nsproxy->mnt_ns;
-
- if ((unshare_flags & CLONE_NEWNS) && ns) {
-  if (!capable(CAP_SYS_ADMIN))
-   return -EPERM;
```

```
-
-  *new_nsp = dup_mnt_ns(current, new_fs ? new_fs : current->fs);
-  if (!*new_nsp)
-   return -ENOMEM;
- }
-
- return 0;
-}
-
-/*
  * Unsharing of sighand is not supported yet
  */
 static int unshare_sighand(unsigned long unshare_flags, struct sighand_struct **new_sighp)
@@ -1593,16 +1573,6 @@ static int unshare_semundo(unsigned long
  return 0;
 }

-#ifndef CONFIG_IPC_NS
-static inline int unshare_ipcs(unsigned long flags, struct ipc_namespace **ns)
-{
- if (flags & CLONE_NEWIPC)
-  return -EINVAL;
-
- return 0;
-}
-#endif
-
 /*
  * unshare allows a process to 'unshare' part of the process
  * context which was originally shared using clone.  copy_*
@@ -1615,14 +1585,11 @@ asmlinkage long sys_unshare(unsigned lon
 {
  int err = 0;
  struct fs_struct *fs, *new_fs = NULL;
- struct mnt_namespace *ns, *new_ns = NULL;
  struct sighand_struct *new_sigh = NULL;
  struct mm_struct *mm, *new_mm = NULL, *active_mm = NULL;
  struct files_struct *fd, *new_fd = NULL;
  struct sem_undo_list *new_ulist = NULL;
  struct nsproxy *new_nsproxy = NULL, *old_nsproxy = NULL;
- struct uts_namespace *uts, *new_uts = NULL;
- struct ipc_namespace *ipc, *new_ipc = NULL;

  check_unshare_flags(&unshare_flags);

@@ -1637,36 +1604,24 @@ asmlinkage long sys_unshare(unsigned lon
   goto bad_unshare_out;
  if ((err = unshare_fs(unshare_flags, &new_fs)))
```

```
   goto bad_unshare_cleanup_thread;
- if ((err = unshare_mnt_namespace(unshare_flags, &new_ns, new_fs)))
-  goto bad_unshare_cleanup_fs;
  if ((err = unshare_sighand(unshare_flags, &new_sigh)))
-  goto bad_unshare_cleanup_ns;
+  goto bad_unshare_cleanup_fs;
  if ((err = unshare_vm(unshare_flags, &new_mm)))
   goto bad_unshare_cleanup_sigh;
  if ((err = unshare_fd(unshare_flags, &new_fd)))
   goto bad_unshare_cleanup_vm;
  if ((err = unshare_semundo(unshare_flags, &new_ulist)))
   goto bad_unshare_cleanup_fd;
- if ((err = unshare_utsname(unshare_flags, &new_uts)))
+ if ((err = unshare_nsproxy_namespaces(unshare_flags, &new_nsproxy,
+   new_fs)))
   goto bad_unshare_cleanup_semundo;
- if ((err = unshare_ipcs(unshare_flags, &new_ipc)))
-  goto bad_unshare_cleanup_uts;

- if (new_ns || new_uts || new_ipc) {
-  old_nsproxy = current->nsproxy;
-  new_nsproxy = dup_namespaces(old_nsproxy);
-  if (!new_nsproxy) {
-   err = -ENOMEM;
-   goto bad_unshare_cleanup_ipc;
-  }
- }
-
- if (new_fs || new_ns || new_mm || new_fd || new_ulist ||
-    new_uts || new_ipc) {
+ if (new_fs ||  new_mm || new_fd || new_ulist || new_nsproxy) {

   task_lock(current);

   if (new_nsproxy) {
+   old_nsproxy = current->nsproxy;
    current->nsproxy = new_nsproxy;
    new_nsproxy = old_nsproxy;
   }
@@ -1677,12 +1632,6 @@ asmlinkage long sys_unshare(unsigned lon
    new_fs = fs;
   }

-  if (new_ns) {
-   ns = current->nsproxy->mnt_ns;
-   current->nsproxy->mnt_ns = new_ns;
-   new_ns = ns;
-  }
```

```
-
   if (new_mm) {
    mm = current->mm;
    active_mm = current->active_mm;
@@ -1698,32 +1647,12 @@ asmlinkage long sys_unshare(unsigned lon
    new_fd = fd;
   }

-  if (new_uts) {
-   uts = current->nsproxy->uts_ns;
-   current->nsproxy->uts_ns = new_uts;
-   new_uts = uts;
-  }
-
-  if (new_ipc) {
-   ipc = current->nsproxy->ipc_ns;
-   current->nsproxy->ipc_ns = new_ipc;
-   new_ipc = ipc;
-  }
-
   task_unlock(current);
  }

  if (new_nsproxy)
   put_nsproxy(new_nsproxy);

-bad_unshare_cleanup_ipc:
- if (new_ipc)
-  put_ipc_ns(new_ipc);
-
-bad_unshare_cleanup_uts:
- if (new_uts)
-  put_uts_ns(new_uts);
-
 bad_unshare_cleanup_semundo:
 bad_unshare_cleanup_fd:
  if (new_fd)
@@ -1738,10 +1667,6 @@ bad_unshare_cleanup_sigh:
    if (atomic_dec_and_test(&new_sigh->count))
     kmem_cache_free(sighand_cachep, new_sigh);

-bad_unshare_cleanup_ns:
- if (new_ns)
-  put_mnt_ns(new_ns);
-
 bad_unshare_cleanup_fs:
  if (new_fs)
   put_fs_struct(new_fs);
```

```
diff -puN kernel/nsproxy.c~merge-sys_clone-sys_unshare-nsproxy-and-namespace
kernel/nsproxy.c
--- a/kernel/nsproxy.c~merge-sys_clone-sys_unshare-nsproxy-and-namespace
+++ a/kernel/nsproxy.c
@@ -38,10 +38,8 @@ void get_task_namespaces(struct task_str

 /*
  * creates a copy of "orig" with refcount 1.
- * This does not grab references to the contained namespaces,
- * so that needs to be done by dup_namespaces.
  */
-static inline struct nsproxy *clone_namespaces(struct nsproxy *orig)
+static inline struct nsproxy *clone_nsproxy(struct nsproxy *orig)
 {
  struct nsproxy *ns;

@@ -52,26 +50,49 @@ static inline struct nsproxy *clone_name
 }

 /*
- * copies the nsproxy, setting refcount to 1, and grabbing a
- * reference to all contained namespaces.  Called from
- * sys_unshare()
+ * Create new nsproxy and all of its the associated namespaces.
+ * Return the newly created nsproxy.  Do not attach this to the task,
+ * leave it to the caller to do proper locking and attach it to task.
  */
-struct nsproxy *dup_namespaces(struct nsproxy *orig)
+static struct nsproxy *create_new_namespaces(int flags, struct task_struct *tsk,
+   struct fs_struct *new_fs)
 {
- struct nsproxy *ns = clone_namespaces(orig);
+ struct nsproxy *new_nsp;

- if (ns) {
-  if (ns->mnt_ns)
-   get_mnt_ns(ns->mnt_ns);
-  if (ns->uts_ns)
-   get_uts_ns(ns->uts_ns);
-  if (ns->ipc_ns)
-   get_ipc_ns(ns->ipc_ns);
-  if (ns->pid_ns)
-   get_pid_ns(ns->pid_ns);
- }
+ new_nsp = clone_nsproxy(tsk->nsproxy);
+ if (!new_nsp)
+  return ERR_PTR(-ENOMEM);
```

```
- return ns;
+ new_nsp->mnt_ns = copy_mnt_ns(flags, tsk->nsproxy->mnt_ns, new_fs);
+ if (IS_ERR(new_nsp->mnt_ns))
+  goto out_ns;
+
+ new_nsp->uts_ns = copy_utsname(flags, tsk->nsproxy->uts_ns);
+ if (IS_ERR(new_nsp->uts_ns))
+  goto out_uts;
+
+ new_nsp->ipc_ns = copy_ipcs(flags, tsk->nsproxy->ipc_ns);
+ if (IS_ERR(new_nsp->ipc_ns))
+  goto out_ipc;
+
+ new_nsp->pid_ns = copy_pid_ns(flags, tsk->nsproxy->pid_ns);
+ if (IS_ERR(new_nsp->pid_ns))
+  goto out_pid;
+
+ return new_nsp;
+
+out_pid:
+ if (new_nsp->ipc_ns)
+  put_ipc_ns(new_nsp->ipc_ns);
+out_ipc:
+ if (new_nsp->uts_ns)
+  put_uts_ns(new_nsp->uts_ns);
+out_uts:
+ if (new_nsp->mnt_ns)
+  put_mnt_ns(new_nsp->mnt_ns);
+out_ns:
+ kfree(new_nsp);
+ return ERR_PTR(-ENOMEM);
 }

 /*
@@ -92,47 +113,21 @@ int copy_namespaces(int flags, struct ta
  if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC)))
   return 0;

- new_ns = clone_namespaces(old_ns);
- if (!new_ns) {
-  err = -ENOMEM;
+ if (!capable(CAP_SYS_ADMIN)) {
+  err = -EPERM;
   goto out;
  }

- tsk->nsproxy = new_ns;
-
```

```
- err = copy_mnt_ns(flags, tsk);
- if (err)
-  goto out_ns;
-
- err = copy_utsname(flags, tsk);
- if (err)
-  goto out_uts;
-
- err = copy_ipcs(flags, tsk);
- if (err)
-  goto out_ipc;
-
- err = copy_pid_ns(flags, tsk);
- if (err)
-  goto out_pid;
+ new_ns = create_new_namespaces(flags, tsk, tsk->fs);
+ if (IS_ERR(new_ns)) {
+  err = PTR_ERR(new_ns);
+  goto out;
+ }

+ tsk->nsproxy = new_ns;
 out:
  put_nsproxy(old_ns);
  return err;
-
-out_pid:
- 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);
-out_uts:
- if (new_ns->mnt_ns)
-  put_mnt_ns(new_ns->mnt_ns);
-out_ns:
- tsk->nsproxy = old_ns;
- kfree(new_ns);
- goto out;
 }

 void free_nsproxy(struct nsproxy *ns)
@@ -147,3 +142,41 @@ void free_nsproxy(struct nsproxy *ns)
   put_pid_ns(ns->pid_ns);
  kfree(ns);
 }
+
+/*
```

```
+ * Called from unshare. Unshare all the namespaces part of nsproxy.
+ * On sucess, returns the new nsproxy and a reference to old nsproxy
+ * to make sure it stays around.
+ */
+int unshare_nsproxy_namespaces(unsigned long unshare_flags,
+  struct nsproxy **new_nsp, struct fs_struct *new_fs)
+{
+ struct nsproxy *old_ns = current->nsproxy;
+ int err = 0;
+
+ if (!(unshare_flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC)))
+  return 0;
+
+#ifndef CONFIG_IPC_NS
+ if (unshare_flags & CLONE_NEWIPC)
+  return -EINVAL;
+#endif
+
+#ifndef CONFIG_UTS_NS
+ if (unshare_flags & CLONE_NEWUTS)
+  return -EINVAL;
+#endif
+
+ if (!capable(CAP_SYS_ADMIN))
+  return -EPERM;
+
+ get_nsproxy(old_ns);
+
+ *new_nsp = create_new_namespaces(unshare_flags, current,
+    new_fs ? new_fs : current->fs);
+ if (IS_ERR(*new_nsp)) {
+  err = PTR_ERR(*new_nsp);
+  put_nsproxy(old_ns);
+ }
+ return err;
+}
diff -puN kernel/pid.c~merge-sys_clone-sys_unshare-nsproxy-and-namespace kernel/pid.c
--- a/kernel/pid.c~merge-sys_clone-sys_unshare-nsproxy-and-namespace
+++ a/kernel/pid.c
@@ -360,16 +360,11 @@ struct pid *find_ge_pid(int nr)
 }
 EXPORT_SYMBOL_GPL(find_get_pid);

-int copy_pid_ns(int flags, struct task_struct *tsk)
+struct pid_namespace *copy_pid_ns(int flags, struct pid_namespace *old_ns)
 {
- struct pid_namespace *old_ns = tsk->nsproxy->pid_ns;
- int err = 0;
```

```
-
- if (!old_ns)
-  return 0;
-
+ BUG_ON(!old_ns);
  get_pid_ns(old_ns);
- return err;
+ return old_ns;
 }

 void free_pid_ns(struct kref *kref)
diff -puN kernel/utsname.c~merge-sys_clone-sys_unshare-nsproxy-and-namespace
kernel/utsname.c
--- a/kernel/utsname.c~merge-sys_clone-sys_unshare-nsproxy-and-namespace
+++ a/kernel/utsname.c
@@ -32,58 +32,25 @@ static struct uts_namespace *clone_uts_n
 }

 /*
- * unshare the current process' utsname namespace.
- * called only in sys_unshare()
- */
-int unshare_utsname(unsigned long unshare_flags, struct uts_namespace **new_uts)
-{
- if (unshare_flags & CLONE_NEWUTS) {
-  if (!capable(CAP_SYS_ADMIN))
-   return -EPERM;
-
-  *new_uts = clone_uts_ns(current->nsproxy->uts_ns);
-  if (!*new_uts)
-   return -ENOMEM;
- }
-
- return 0;
-}
-
-/*
  * Copy task tsk's utsname namespace, or clone it if flags
  * specifies CLONE_NEWUTS.  In latter case, changes to the
  * utsname of this process won't be seen by parent, and vice
  * versa.
  */
-int copy_utsname(int flags, struct task_struct *tsk)
+struct uts_namespace *copy_utsname(int flags, struct uts_namespace *old_ns)
 {
- struct uts_namespace *old_ns = tsk->nsproxy->uts_ns;
  struct uts_namespace *new_ns;
- int err = 0;
```

```
-
- if (!old_ns)
-  return 0;

+ BUG_ON(!old_ns);
  get_uts_ns(old_ns);

  if (!(flags & CLONE_NEWUTS))
-  return 0;
-
- if (!capable(CAP_SYS_ADMIN)) {
-  err = -EPERM;
-  goto out;
- }
+  return old_ns;

  new_ns = clone_uts_ns(old_ns);
- if (!new_ns) {
-  err = -ENOMEM;
-  goto out;
- }
- tsk->nsproxy->uts_ns = new_ns;

-out:
  put_uts_ns(old_ns);
- return err;
+ return new_ns;
 }

 void free_uts_ns(struct kref *kref)
```

_

Patches currently in -mm which might be from pbadari@us.ibm.com are

origin.patch

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Herbert Poetzl on Sat, 16 Jun 2007 19:17:42 GMT
View Forum Message <> Reply to Message

On Tue, May 08, 2007 at 07:45:35PM -0700, akpm@linux-foundation.org wrote:

>
> The patch titled
>   Merge sys_clone()/sys_unshare() nsproxy and namespace handling
> has been removed from the -mm tree. Its filename was
>   merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch
>
> This patch was dropped because it was merged into mainline or a subsystem tree
>
> --
> Subject: Merge sys_clone()/sys_unshare() nsproxy and namespace handling
> From: Badari Pulavarty <pbadari@us.ibm.com>
>
> sys_clone() and sys_unshare() both makes copies of nsproxy and its associated
> namespaces. But they have different code paths.
>
> This patch merges all the nsproxy and its associated namespace copy/clone
> handling (as much as possible). Posted on container list earlier for
> feedback.
>
>
> - Create a new nsproxy and its associated namespaces and pass it back to
>   caller to attach it to right process.
>
> - Changed all copy_*_ns() routines to return a new copy of namespace
>   instead of attaching it to task->nsproxy.
>
> - Moved the CAP_SYS_ADMIN checks out of copy_*_ns() routines.
>
> - Removed unnessary !ns checks from copy_*_ns() and added BUG_ON()
>   just incase.
>
> - Get rid of all individual unshare_*_ns() routines and make use of
>   copy_*_ns() instead.
>

.. [zapped] ...

> + * Called from unshare. Unshare all the namespaces part of nsproxy.
> + * On sucess, returns the new nsproxy and a reference to old nsproxy
> + * to make sure it stays around.
> + */
> +int unshare_nsproxy_namespaces(unsigned long unshare_flags,
> + struct nsproxy **new_nsp, struct fs_struct *new_fs)
> +{

this makes sys_unshare leak and nsproxy (reference)

can be tested with the following command sequence:

vcmd -nu ^17 -- vcmd -nu ^17 -- sleep 10

(and some nsproxy accounting/debugging as used in
Linux-VServer)

we probably want to drop the reference to the old
nsproxy in sys_unshare() but I do not see a good reason
to take the reference in the first place (at least not
with the code in mainline 2.6.22-rc4)

HTH,
Herbert

PS: vcmd can be found here:
http://vserver.13thfloor.at/Experimental/TOOLS/vcmd-0.09.tar.bz2

... [more zapped] ...

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Oleg Nesterov on Sun, 17 Jun 2007 14:38:30 GMT
View Forum Message <> Reply to Message

On 06/16, Herbert Poetzl wrote:
>
> On Tue, May 08, 2007 at 07:45:35PM -0700, akpm@linux-foundation.org wrote:
> >
> > The patch titled
> >   Merge sys_clone()/sys_unshare() nsproxy and namespace handling
> > has been removed from the -mm tree. Its filename was
> >   merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch
> >
> > This patch was dropped because it was merged into mainline or a subsystem tree
> >
>
> .. [zapped] ...
>
> > + * Called from unshare. Unshare all the namespaces part of nsproxy.
> > + * On sucess, returns the new nsproxy and a reference to old nsproxy
> > + * to make sure it stays around.
> > + */
> > +int unshare_nsproxy_namespaces(unsigned long unshare_flags,
> > + struct nsproxy **new_nsp, struct fs_struct *new_fs)

> > +{
>
> this makes sys_unshare leak and nsproxy (reference)
>
> can be tested with the following command sequence:
> vcmd -nu ^17 -- vcmd -nu ^17 -- sleep 10

I know almost nothing about this stuff, could you please explain in brief
what this command does and how do you detect a leak?

> (and some nsproxy accounting/debugging as used in
> Linux-VServer)
>
> we probably want to drop the reference to the old
> nsproxy in sys_unshare() but I do not see a good reason
> to take the reference in the first place (at least not
> with the code in mainline 2.6.22-rc4)

At first glance, sys_unshare() drops the reference to the old nsproxy,

```
 old_nsproxy = current->nsproxy;
 current->nsproxy = new_nsproxy;
 new_nsproxy = old_nsproxy;

 ...

 if (new_nsproxy)
  put_nsproxy(new_nsproxy);
```

However, nsproxy's code is full of strange unneeded get/put calls, for
example:

```
 struct uts_namespace *copy_utsname(int flags, struct uts_namespace *old_ns)
 {
  struct uts_namespace *new_ns;

  BUG_ON(!old_ns);
  get_uts_ns(old_ns);

  if (!(flags & CLONE_NEWUTS))
   return old_ns;

  new_ns = clone_uts_ns(old_ns);

  put_uts_ns(old_ns);
  return new_ns;
 }
```

I think it would be better to do

```
 struct uts_namespace *copy_utsname(int flags, struct uts_namespace *old_ns)
 {
  struct uts_namespace *new_ns;

  BUG_ON(!old_ns);

  if (!(flags & CLONE_NEWUTS)) {
   get_uts_ns(old_ns);
   return old_ns;
  }

  new_ns = clone_uts_ns(old_ns);
  return new_ns;
 }
```

Not only this looks better (imho), this is more robust.

Let's look at copy_namespaces(), it does the same "get_xxx() in advance", but
-EPERM forgets to do put_nsproxy(), so we definitely have a leak in copy_process().

So, if the command above does clone() which fails, perhaps this can explain the
problem.

Oleg.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Oleg Nesterov on Sun, 17 Jun 2007 14:49:15 GMT
View Forum Message <> Reply to Message

On 06/17, Oleg Nesterov wrote:
>
> Let's look at copy_namespaces(), it does the same "get_xxx() in advance", but
> -EPERM forgets to do put_nsproxy(), so we definitely have a leak in copy_process().

Ugh, I am sorry, EPERM does put_nsproxy(). Still I can't understand why
copy_namespaces() does get_nsproxy() unconditionally.

Oleg.

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Oleg Nesterov on Sun, 17 Jun 2007 16:30:04 GMT
View Forum Message <> Reply to Message

On 06/17, Oleg Nesterov wrote:
>
> However, nsproxy's code is full of strange unneeded get/put calls, for
> example:
>
> struct uts_namespace *copy_utsname(int flags, struct uts_namespace *old_ns)
> {
>   struct uts_namespace *new_ns;
>
>   BUG_ON(!old_ns);
>   get_uts_ns(old_ns);
>
>   if (!(flags & CLONE_NEWUTS))
>   return old_ns;
>
>   new_ns = clone_uts_ns(old_ns);
>
>   put_uts_ns(old_ns);
>   return new_ns;
> }

Perhaps I missed something again, but this looks wrong to me.

copy_utsname() assumes that old_ns != NULL. OK, it should not.

However, clone_uts_ns() returns NULL if kmalloc() fails.
create_new_namespaces() checks IS_ERR(new_ns), but IS_ERR(NULL) = false.
So the next copy_namespaces/unshare_nsproxy_namespaces will oops ?

The same for all ->xxx_ns fields.

Oleg.

Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Herbert Poetzl on Sun, 17 Jun 2007 17:09:42 GMT
View Forum Message <> Reply to Message

On Sun, Jun 17, 2007 at 06:38:30PM +0400, Oleg Nesterov wrote:
> On 06/16, Herbert Poetzl wrote:
> >
> > On Tue, May 08, 2007 at 07:45:35PM -0700, akpm@linux-foundation.org wrote:
> > >
> > > The patch titled
> > >   Merge sys_clone()/sys_unshare() nsproxy and namespace handling
> > > has been removed from the -mm tree. Its filename was
> > >   merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch
> > >
> > > This patch was dropped because it was merged into mainline or a subsystem tree
> > >
> >
> > .. [zapped] ...
> >
> > > + * Called from unshare. Unshare all the namespaces part of nsproxy.
> > > + * On sucess, returns the new nsproxy and a reference to old nsproxy
> > > + * to make sure it stays around.
> > > + */
> > > +int unshare_nsproxy_namespaces(unsigned long unshare_flags,
> > > + struct nsproxy **new_nsp, struct fs_struct *new_fs)
> > > +{
> >
> > this makes sys_unshare leak and nsproxy (reference)
> >
> > can be tested with the following command sequence:
> >   vcmd -nu ^17 -- vcmd -nu ^17 -- sleep 10
>
> I know almost nothing about this stuff, could you please explain in
> brief what this command does ...

yeah, sure, it basically calls sys_unshare() with
bit 17 (CLONE_NEWNS) set then invokes the chained
command, so we get a sleep which is in a separate
namespace, unshared from a namespace != the main
one ...

> ... and how do you detect a leak?

> > (and some nsproxy accounting/debugging as used in

> > Linux-VServer)

on Linux-VServer,we have accounting for those
proxies (and several other namespace related stuff)
because we already suspected leakage and reference
bugs in this area some time ago ... btw, I also
suggested to put a similar functionality in mainline
for the time being, but it was ignored, as usual ...

> > we probably want to drop the reference to the old
> > nsproxy in sys_unshare() but I do not see a good reason
> > to take the reference in the first place (at least not
> > with the code in mainline 2.6.22-rc4)
>
> At first glance, sys_unshare() drops the reference to
> the old nsproxy,

okay, the 'current' task has an nsproxy, and keeps
a reference to that (let's assume it is the only
task using this nsproxy, then the count will be 1)

unshare_nsproxy_namespaces() now does get_nsproxy()
which makes the count=2, then it creates a new
nsproxy (which will get count=1), and returns ...

>   old_nsproxy = current->nsproxy;
>   current->nsproxy = new_nsproxy;
>   new_nsproxy = old_nsproxy;

sys_unshare, now replaces the current->nsproxy with
the new one, which will have the correct count=1,
and puts the old nsproxy (which has count=2), and
thus the nsproxy will not get released, although
it isn't referenced/used anymore ...

> if (new_nsproxy)
>   put_nsproxy(new_nsproxy);
>
>
> However, nsproxy's code is full of strange unneeded get/put
> calls, for example:

yep, I totally agree, it is quite a mess, and if
not handled carefully, you end up leaking proxies :)

best,
Herbert

> struct uts_namespace *copy_utsname(int flags, struct uts_namespace *old_ns)
> {
>   struct uts_namespace *new_ns;
>
>   BUG_ON(!old_ns);
>   get_uts_ns(old_ns);
>
>   if (!(flags & CLONE_NEWUTS))
>   return old_ns;
>
>   new_ns = clone_uts_ns(old_ns);
>
>   put_uts_ns(old_ns);
>   return new_ns;
> }
>
> I think it would be better to do
>
> struct uts_namespace *copy_utsname(int flags, struct uts_namespace *old_ns)
> {
>   struct uts_namespace *new_ns;
>
>   BUG_ON(!old_ns);
>
>   if (!(flags & CLONE_NEWUTS)) {
>   get_uts_ns(old_ns);
>   return old_ns;
>   }
>
>   new_ns = clone_uts_ns(old_ns);
>   return new_ns;
> }
>
> Not only this looks better (imho), this is more robust.
>
> Let's look at copy_namespaces(), it does the same
> "get_xxx() in advance", but -EPERM forgets to do
> put_nsproxy(), so we definitely have a leak in copy_process().
>
> So, if the command above does clone() which fails, perhaps
> this can explain the problem.
>
> Oleg.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Oleg Nesterov on Sun, 17 Jun 2007 18:28:22 GMT
View Forum Message <> Reply to Message

On 06/17, Herbert Poetzl wrote:
>
> On Sun, Jun 17, 2007 at 06:38:30PM +0400, Oleg Nesterov wrote:
> >
> > At first glance, sys_unshare() drops the reference to
> > the old nsproxy,
>
> okay, the 'current' task has an nsproxy, and keeps
> a reference to that (let's assume it is the only
> task using this nsproxy, then the count will be 1)
>
> unshare_nsproxy_namespaces() now does get_nsproxy()
> which makes the count=2, then it creates a new
> nsproxy (which will get count=1), and returns ...

Ah yes, stupid me.

Looks like we should just remove get/put old_ns from unshare_nsproxy_namespaces()
as you suggested.

Thanks!

Oleg.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Cedric Le Goater on Mon, 18 Jun 2007 11:51:20 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 06/17, Oleg Nesterov wrote:
>> Let's look at copy_namespaces(), it does the same "get_xxx() in advance", but
>> -EPERM forgets to do put_nsproxy(), so we definitely have a leak in copy_process().
>
> Ugh, I am sorry, EPERM does put_nsproxy(). Still I can't understand why
> copy_namespaces() does get_nsproxy() unconditionally.

well, if you're cloning a new task and not unsharing some of the namespaces
you still want to increase the refcount on the nsproxy bc a new task is now
referencing it. nop ?

C.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Cedric Le Goater on Mon, 18 Jun 2007 12:02:19 GMT
View Forum Message <> Reply to Message

> on Linux-VServer,we have accounting for those
> proxies (and several other namespace related stuff)
> because we already suspected leakage and reference
> bugs in this area some time ago ... btw, I also
> suggested to put a similar functionality in mainline
> for the time being, but it was ignored, as usual ...

something like a kmem_cache ? and we are not ignoring vserver ! :)

Cheers,

C.

Add a kmem_cache to manage nsproxy objects.

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
---
 kernel/nsproxy.c | 21 ++++++++++++++++++----
 1 file changed, 17 insertions(+), 4 deletions(-)

Index: 2.6.22-rc4-mm2/kernel/nsproxy.c
===
--- 2.6.22-rc4-mm2.orig/kernel/nsproxy.c
+++ 2.6.22-rc4-mm2/kernel/nsproxy.c
@@ -21,6 +21,8 @@
 #include <linux/utsname.h>
 #include <linux/pid_namespace.h>

+static struct kmem_cache *nsproxy_cachep;
+
 struct nsproxy init_nsproxy = INIT_NSPROXY(init_nsproxy);

```
 static inline void get_nsproxy(struct nsproxy *ns)
@@ -43,9 +45,11 @@ static inline struct nsproxy *clone_nspr
 {
  struct nsproxy *ns;

- ns = kmemdup(orig, sizeof(struct nsproxy), GFP_KERNEL);
- if (ns)
+ ns = kmem_cache_alloc(nsproxy_cachep, GFP_KERNEL);
+ if (ns) {
+  memcpy(ns, orig, sizeof(struct nsproxy));
   atomic_set(&ns->count, 1);
+ }
  return ns;
 }

@@ -109,7 +113,7 @@ out_uts:
  if (new_nsp->mnt_ns)
   put_mnt_ns(new_nsp->mnt_ns);
 out_ns:
- kfree(new_nsp);
+ kmem_cache_free(nsproxy_cachep, new_nsp);
  return ERR_PTR(err);
 }

@@ -160,7 +164,7 @@ void free_nsproxy(struct nsproxy *ns)
   put_pid_ns(ns->pid_ns);
  if (ns->user_ns)
   put_user_ns(ns->user_ns);
- kfree(ns);
+ kmem_cache_free(nsproxy_cachep, ns);
 }

 /*
@@ -191,3 +195,12 @@ int unshare_nsproxy_namespaces(unsigned
  }
  return err;
 }
+
+static int __init nsproxy_cache_init(void)
+{
+ nsproxy_cachep = kmem_cache_create("nsproxy", sizeof(struct nsproxy),
+       0, SLAB_PANIC, NULL, NULL);
+ return 0;
+}
+
+module_init(nsproxy_cache_init);
```

_____

Containers mailing list

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Oleg Nesterov on Mon, 18 Jun 2007 12:21:48 GMT
View Forum Message <> Reply to Message

On 06/18, Cedric Le Goater wrote:
>
> Oleg Nesterov wrote:
> > On 06/17, Oleg Nesterov wrote:
> >> Let's look at copy_namespaces(), it does the same "get_xxx() in advance", but
> >> -EPERM forgets to do put_nsproxy(), so we definitely have a leak in copy_process().
> >
> > Ugh, I am sorry, EPERM does put_nsproxy(). Still I can't understand why
> > copy_namespaces() does get_nsproxy() unconditionally.
>
> well, if you're cloning a new task and not unsharing some of the namespaces
> you still want to increase the refcount on the nsproxy bc a new task is now
> referencing it. nop ?

Yes, but copy_namespaces() does get_nsproxy() unconditionally, and then it does
put_nsproxy() if not unsharing, which is not good imho.

IOW, I think the patch below makes the code a bit better. copy_namespaces()
doesn't need put_nsproxy() at all.

Oleg.

```
--- t/kernel/nsproxy.c~ 2007-06-18 16:10:53.000000000 +0400
+++ t/kernel/nsproxy.c 2007-06-18 16:13:02.000000000 +0400
@@ -103,31 +103,24 @@ int copy_namespaces(int flags, struct ta
 {
  struct nsproxy *old_ns = tsk->nsproxy;
  struct nsproxy *new_ns;
- int err = 0;

  if (!old_ns)
   return 0;

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

```
-
- if (!capable(CAP_SYS_ADMIN)) {
-  err = -EPERM;
-  goto out;
  }

+ if (!capable(CAP_SYS_ADMIN))
+  return -EPERM;
+
  new_ns = create_new_namespaces(flags, tsk, tsk->fs);
- if (IS_ERR(new_ns)) {
-  err = PTR_ERR(new_ns);
-  goto out;
- }
+ if (IS_ERR(new_ns))
+  return PTR_ERR(new_ns);

  tsk->nsproxy = new_ns;
-out:
- put_nsproxy(old_ns);
- return err;
+ return 0;
 }

 void free_nsproxy(struct nsproxy *ns)
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Cedric Le Goater on Mon, 18 Jun 2007 12:25:28 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 06/18, Cedric Le Goater wrote:
>> Oleg Nesterov wrote:
>>> On 06/17, Oleg Nesterov wrote:
>>>> Let's look at copy_namespaces(), it does the same "get_xxx() in advance", but
>>>> -EPERM forgets to do put_nsproxy(), so we definitely have a leak in copy_process().
>>> Ugh, I am sorry, EPERM does put_nsproxy(). Still I can't understand why
>>> copy_namespaces() does get_nsproxy() unconditionally.
>> well, if you're cloning a new task and not unsharing some of the namespaces
>> you still want to increase the refcount on the nsproxy bc a new task is now
>> referencing it. nop ?

>
> Yes, but copy_namespaces() does get_nsproxy() unconditionally, and then it does
> put_nsproxy() if not unsharing, which is not good imho.
>
> IOW, I think the patch below makes the code a bit better. copy_namespaces()
> doesn't need put_nsproxy() at all.

Indeed it does and also :

nsproxy.c | 23 ++++++++----------
1 file changed, 8 insertions(+), 15 deletions(-)

Thanks,

C.

> Oleg.
>
> --- t/kernel/nsproxy.c~ 2007-06-18 16:10:53.000000000 +0400
> +++ t/kernel/nsproxy.c 2007-06-18 16:13:02.000000000 +0400
> @@ -103,31 +103,24 @@ int copy_namespaces(int flags, struct ta
> {
> struct nsproxy *old_ns = tsk->nsproxy;
> struct nsproxy *new_ns;
> - int err = 0;
>
> if (!old_ns)
> return 0;
>
> - get_nsproxy(old_ns);
> -
> - if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC)))
> + if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC))) {
> + get_nsproxy(old_ns);
> return 0;
> -
> - if (!capable(CAP_SYS_ADMIN)) {
> - err = -EPERM;
> - goto out;
> }
>
> + if (!capable(CAP_SYS_ADMIN))
> + return -EPERM;
> +
> new_ns = create_new_namespaces(flags, tsk, tsk->fs);
> - if (IS_ERR(new_ns)) {
> - err = PTR_ERR(new_ns);
> - goto out;

> - }
> + if (IS_ERR(new_ns))
> + return PTR_ERR(new_ns);
>
>   tsk->nsproxy = new_ns;
> -out:
> - put_nsproxy(old_ns);
> - return err;
> + return 0;
> }
>
> void free_nsproxy(struct nsproxy *ns)
>
>

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Cedric Le Goater on Mon, 18 Jun 2007 12:37:25 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Sun, Jun 17, 2007 at 06:38:30PM +0400, Oleg Nesterov wrote:
>> On 06/16, Herbert Poetzl wrote:
>>> On Tue, May 08, 2007 at 07:45:35PM -0700, akpm@linux-foundation.org wrote:
>>>> The patch titled
>>>>   Merge sys_clone()/sys_unshare() nsproxy and namespace handling
>>>> has been removed from the -mm tree. Its filename was
>>>>   merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch
>>>>
>>>> This patch was dropped because it was merged into mainline or a subsystem tree
>>>>
>>> .. [zapped] ...
>>>
>>>> + * Called from unshare. Unshare all the namespaces part of nsproxy.
>>>> + * On sucess, returns the new nsproxy and a reference to old nsproxy
>>>> + * to make sure it stays around.
>>>> + */
>>>> +int unshare_nsproxy_namespaces(unsigned long unshare_flags,
>>>> + struct nsproxy **new_nsp, struct fs_struct *new_fs)
>>>> +{
>>> this makes sys_unshare leak and nsproxy (reference)
>>>

>>> can be tested with the following command sequence:
>>>   vcmd -nu ^17 -- vcmd -nu ^17 -- sleep 10
>> I know almost nothing about this stuff, could you please explain in
>> brief what this command does ...
>
> yeah, sure, it basically calls sys_unshare() with
> bit 17 (CLONE_NEWNS) set then invokes the chained
> command, so we get a sleep which is in a separate
> namespace, unshared from a namespace != the main
> one ...
>
>> ... and how do you detect a leak?
>
>>> (and some nsproxy accounting/debugging as used in
>>> Linux-VServer)
>
> on Linux-VServer,we have accounting for those
> proxies (and several other namespace related stuff)
> because we already suspected leakage and reference
> bugs in this area some time ago ... btw, I also
> suggested to put a similar functionality in mainline
> for the time being, but it was ignored, as usual ...
>
>>> we probably want to drop the reference to the old
>>> nsproxy in sys_unshare() but I do not see a good reason
>>> to take the reference in the first place (at least not
>>> with the code in mainline 2.6.22-rc4)
>> At first glance, sys_unshare() drops the reference to
>> the old nsproxy,
>
> okay, the 'current' task has an nsproxy, and keeps
> a reference to that (let's assume it is the only
> task using this nsproxy, then the count will be 1)
>
> unshare_nsproxy_namespaces() now does get_nsproxy()
> which makes the count=2, then it creates a new
> nsproxy (which will get count=1), and returns ...
>
>>   old_nsproxy = current->nsproxy;
>>   current->nsproxy = new_nsproxy;
>>   new_nsproxy = old_nsproxy;
>
> sys_unshare, now replaces the current->nsproxy with
> the new one, which will have the correct count=1,
> and puts the old nsproxy (which has count=2), and
> thus the nsproxy will not get released, although
> it isn't referenced/used anymore ...

Herbert,

Could you give a try to the patch i've sent previously and this one
which removes an extra get_nsproxy() ? It fixes the leak for me. I've
run the ltp tests we have on namespace unsharing and i could see the
no leaks in /proc/slabinfo.

Badari,

That extra get_nsproxy() seemed a superfluous remain from the 2.6.20.
Do you see any issues with it ?

If we're all happy with these fixes, i'll send them on lkml@ for review.
They might deserve to be in 2.6.22.

Thanks,

C.

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
---
 kernel/nsproxy.c | 7 +------
 1 file changed, 1 insertion(+), 6 deletions(-)

Index: 2.6.22-rc4-mm2/kernel/nsproxy.c
===
--- 2.6.22-rc4-mm2.orig/kernel/nsproxy.c
+++ 2.6.22-rc4-mm2/kernel/nsproxy.c
@@ -175,7 +175,6 @@ void free_nsproxy(struct nsproxy *ns)
 int unshare_nsproxy_namespaces(unsigned long unshare_flags,
 	struct nsproxy **new_nsp, struct fs_struct *new_fs)
 {
-	struct nsproxy *old_ns = current->nsproxy;
 	int err = 0;

 	if (!(unshare_flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
@@ -185,14 +184,10 @@ int unshare_nsproxy_namespaces(unsigned
 	if (!capable(CAP_SYS_ADMIN))
 		return -EPERM;

-	get_nsproxy(old_ns);
-
 	*new_nsp = create_new_namespaces(unshare_flags, current,
 			new_fs ? new_fs : current->fs);
-	if (IS_ERR(*new_nsp)) {
+	 if (IS_ERR(*new_nsp))
 		err = PTR_ERR(*new_nsp);

```
-               put_nsproxy(old_ns);
-       }
        return err;
 }
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Herbert Poetzl on Mon, 18 Jun 2007 15:38:41 GMT
View Forum Message <> Reply to Message

On Mon, Jun 18, 2007 at 02:37:25PM +0200, Cedric Le Goater wrote:
> Herbert Poetzl wrote:
> > On Sun, Jun 17, 2007 at 06:38:30PM +0400, Oleg Nesterov wrote:
> >> On 06/16, Herbert Poetzl wrote:
> >>> On Tue, May 08, 2007 at 07:45:35PM -0700, akpm@linux-foundation.org wrote:
> >>>> The patch titled
> >>>>   Merge sys_clone()/sys_unshare() nsproxy and namespace handling
> >>>> has been removed from the -mm tree. Its filename was
> >>>>   merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch
> >>>>
> >>>> This patch was dropped because it was merged into mainline or a subsystem tree
> >>>>
> >>> .. [zapped] ...
> >>>
> >>> + * Called from unshare. Unshare all the namespaces part of nsproxy.
> >>> + * On sucess, returns the new nsproxy and a reference to old nsproxy
> >>> + * to make sure it stays around.
> >>> + */
> >>> +int unshare_nsproxy_namespaces(unsigned long unshare_flags,
> >>> + struct nsproxy **new_nsp, struct fs_struct *new_fs)
> >>> +{
> >>> this makes sys_unshare leak and nsproxy (reference)
> >>>
> >>> can be tested with the following command sequence:
> >>>   vcmd -nu ^17 -- vcmd -nu ^17 -- sleep 10
> >> I know almost nothing about this stuff, could you please explain in
> >> brief what this command does ...
> >
> > yeah, sure, it basically calls sys_unshare() with
> > bit 17 (CLONE_NEWNS) set then invokes the chained
> > command, so we get a sleep which is in a separate
> > namespace, unshared from a namespace != the main

> > one ...
> >
> >> ... and how do you detect a leak?
> >
> >>> (and some nsproxy accounting/debugging as used in
> >>> Linux-VServer)
> >
> > on Linux-VServer,we have accounting for those
> > proxies (and several other namespace related stuff)
> > because we already suspected leakage and reference
> > bugs in this area some time ago ... btw, I also
> > suggested to put a similar functionality in mainline
> > for the time being, but it was ignored, as usual ...
> >
> >>> we probably want to drop the reference to the old
> >>> nsproxy in sys_unshare() but I do not see a good reason
> >>> to take the reference in the first place (at least not
> >>> with the code in mainline 2.6.22-rc4)
> >> At first glance, sys_unshare() drops the reference to
> >> the old nsproxy,
> >
> > okay, the 'current' task has an nsproxy, and keeps
> > a reference to that (let's assume it is the only
> > task using this nsproxy, then the count will be 1)
> >
> > unshare_nsproxy_namespaces() now does get_nsproxy()
> > which makes the count=2, then it creates a new
> > nsproxy (which will get count=1), and returns ...
> >
> >>   old_nsproxy = current->nsproxy;
> >>   current->nsproxy = new_nsproxy;
> >>   new_nsproxy = old_nsproxy;
> >
> > sys_unshare, now replaces the current->nsproxy with
> > the new one, which will have the correct count=1,
> > and puts the old nsproxy (which has count=2), and
> > thus the nsproxy will not get released, although
> > it isn't referenced/used anymore ...
>
>
> Herbert,
>
> Could you give a try to the patch i've sent previously
> and this one which removes an extra get_nsproxy() ?

will do so shortly ...

> It fixes the leak for me. I've run the ltp tests we

> have on namespace unsharing and i could see the no
> leaks in /proc/slabinfo.

> Badari,
>
> That extra get_nsproxy() seemed a superfluous remain
> from the 2.6.20.
> Do you see any issues with it ?
>
> If we're all happy with these fixes, i'll send them on
> lkml@ for review.

I'm not terribly happy with the current nsproxy
framework, although it improved somewhat ...

I'm still missing some mechanism to 'mix' two
proxies according to a flagmask (which is required
to enter a guest 'partially') ...

best,
Herbert

> They might deserve to be in 2.6.22.
>
> Thanks,
>
> C.
>
> Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
> ---
> kernel/nsproxy.c | 7 +------
> 1 file changed, 1 insertion(+), 6 deletions(-)
>
> Index: 2.6.22-rc4-mm2/kernel/nsproxy.c
> ===
> --- 2.6.22-rc4-mm2.orig/kernel/nsproxy.c
> +++ 2.6.22-rc4-mm2/kernel/nsproxy.c
> @@ -175,7 +175,6 @@ void free_nsproxy(struct nsproxy *ns)
> int unshare_nsproxy_namespaces(unsigned long unshare_flags,
> 		struct nsproxy **new_nsp, struct fs_struct *new_fs)
> {
> -	struct nsproxy *old_ns = current->nsproxy;
> 	int err = 0;
>
> 	if (!(unshare_flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
> @@ -185,14 +184,10 @@ int unshare_nsproxy_namespaces(unsigned
> 	if (!capable(CAP_SYS_ADMIN))
> 		return -EPERM;

>
> -	get_nsproxy(old_ns);
> -
>	*new_nsp = create_new_namespaces(unshare_flags, current,
>	new_fs ? new_fs : current->fs);
> -	if (IS_ERR(*new_nsp)) {
> +	 if (IS_ERR(*new_nsp))
>	err = PTR_ERR(*new_nsp);
> -	put_nsproxy(old_ns);
> -	}
>	return err;
> }

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Herbert Poetzl on Mon, 18 Jun 2007 15:41:08 GMT
View Forum Message <> Reply to Message

On Mon, Jun 18, 2007 at 02:02:19PM +0200, Cedric Le Goater wrote:
>
> > on Linux-VServer,we have accounting for those
> > proxies (and several other namespace related stuff)
> > because we already suspected leakage and reference
> > bugs in this area some time ago ... btw, I also
> > suggested to put a similar functionality in mainline
> > for the time being, but it was ignored, as usual ...
>
> something like a kmem_cache ?

maybe, but even simplest accounting of the
different 'objects' would be more then enough
for the test phase (or maybe as statistics :)

> and we are not ignoring vserver ! :)

good for them, our project is called Linux-VServer :)

best,
Herbert

> Cheers,
>
> C.

>
> Add a kmem_cache to manage nsproxy objects.
>
> Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
> ---
> kernel/nsproxy.c | 21 ++++++++++++++++++----
> 1 file changed, 17 insertions(+), 4 deletions(-)
>
> Index: 2.6.22-rc4-mm2/kernel/nsproxy.c
> ===
> --- 2.6.22-rc4-mm2.orig/kernel/nsproxy.c
> +++ 2.6.22-rc4-mm2/kernel/nsproxy.c
> @@ -21,6 +21,8 @@
> #include <linux/utsname.h>
> #include <linux/pid_namespace.h>
>
> +static struct kmem_cache *nsproxy_cachep;
> +
> struct nsproxy init_nsproxy = INIT_NSPROXY(init_nsproxy);
>
> static inline void get_nsproxy(struct nsproxy *ns)
> @@ -43,9 +45,11 @@ static inline struct nsproxy *clone_nspr
> {
>   struct nsproxy *ns;
>
> - ns = kmemdup(orig, sizeof(struct nsproxy), GFP_KERNEL);
> - if (ns)
> + ns = kmem_cache_alloc(nsproxy_cachep, GFP_KERNEL);
> + if (ns) {
> + memcpy(ns, orig, sizeof(struct nsproxy));
>   atomic_set(&ns->count, 1);
> + }
>   return ns;
> }
>
> @@ -109,7 +113,7 @@ out_uts:
>   if (new_nsp->mnt_ns)
>   put_mnt_ns(new_nsp->mnt_ns);
> out_ns:
> - kfree(new_nsp);
> + kmem_cache_free(nsproxy_cachep, new_nsp);
>   return ERR_PTR(err);
> }
>
> @@ -160,7 +164,7 @@ void free_nsproxy(struct nsproxy *ns)
>   put_pid_ns(ns->pid_ns);
>   if (ns->user_ns)
>   put_user_ns(ns->user_ns);

> - kfree(ns);
> + kmem_cache_free(nsproxy_cachep, ns);
> }
>
> /*
> @@ -191,3 +195,12 @@ int unshare_nsproxy_namespaces(unsigned
>   }
>   return err;
> }
> +
> +static int __init nsproxy_cache_init(void)
> +{
> + nsproxy_cachep = kmem_cache_create("nsproxy", sizeof(struct nsproxy),
> + 0, SLAB_PANIC, NULL, NULL);
> + return 0;
> +}
> +
> +module_init(nsproxy_cache_init);

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Herbert Poetzl on Mon, 18 Jun 2007 15:48:05 GMT
View Forum Message <> Reply to Message

On Mon, Jun 18, 2007 at 02:37:25PM +0200, Cedric Le Goater wrote:
> Herbert Poetzl wrote:
> > On Sun, Jun 17, 2007 at 06:38:30PM +0400, Oleg Nesterov wrote:
> >> On 06/16, Herbert Poetzl wrote:
> >>> On Tue, May 08, 2007 at 07:45:35PM -0700, akpm@linux-foundation.org wrote:
> >>>> The patch titled
> >>>> Merge sys_clone()/sys_unshare() nsproxy and namespace handling
> >>>> has been removed from the -mm tree. Its filename was
> >>>> merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch
> >>>>
> >>>> This patch was dropped because it was merged into mainline or a subsystem tree
> >>>>
> >>> .. [zapped] ...
> >>>
> >>> + * Called from unshare. Unshare all the namespaces part of nsproxy.
> >>> + * On sucess, returns the new nsproxy and a reference to old nsproxy
> >>> + * to make sure it stays around.
> >>> + */
> >>> +int unshare_nsproxy_namespaces(unsigned long unshare_flags,

> >>>> + struct nsproxy **new_nsp, struct fs_struct *new_fs)
> >>>> +{
> >>> this makes sys_unshare leak and nsproxy (reference)
> >>>
> >>> can be tested with the following command sequence:
> >>>   vcmd -nu ^17 -- vcmd -nu ^17 -- sleep 10
> >> I know almost nothing about this stuff, could you please explain in
> >> brief what this command does ...
> >
> > yeah, sure, it basically calls sys_unshare() with
> > bit 17 (CLONE_NEWNS) set then invokes the chained
> > command, so we get a sleep which is in a separate
> > namespace, unshared from a namespace != the main
> > one ...
> >
> >> ... and how do you detect a leak?
> >
> >>> (and some nsproxy accounting/debugging as used in
> >>> Linux-VServer)
> >
> > on Linux-VServer,we have accounting for those
> > proxies (and several other namespace related stuff)
> > because we already suspected leakage and reference
> > bugs in this area some time ago ... btw, I also
> > suggested to put a similar functionality in mainline
> > for the time being, but it was ignored, as usual ...
> >
> >>> we probably want to drop the reference to the old
> >>> nsproxy in sys_unshare() but I do not see a good reason
> >>> to take the reference in the first place (at least not
> >>> with the code in mainline 2.6.22-rc4)
> >> At first glance, sys_unshare() drops the reference to
> >> the old nsproxy,
> >
> > okay, the 'current' task has an nsproxy, and keeps
> > a reference to that (let's assume it is the only
> > task using this nsproxy, then the count will be 1)
> >
> > unshare_nsproxy_namespaces() now does get_nsproxy()
> > which makes the count=2, then it creates a new
> > nsproxy (which will get count=1), and returns ...
> >
> >>   old_nsproxy = current->nsproxy;
> >>   current->nsproxy = new_nsproxy;
> >>   new_nsproxy = old_nsproxy;
> >
> > sys_unshare, now replaces the current->nsproxy with
> > the new one, which will have the correct count=1,

> > and puts the old nsproxy (which has count=2), and
> > thus the nsproxy will not get released, although
> > it isn't referenced/used anymore ...
>
>
> Herbert,
>
> Could you give a try to the patch i've sent previously and this one
> which removes an extra get_nsproxy() ? It fixes the leak for me. I've
> run the ltp tests we have on namespace unsharing and i could see the
> no leaks in /proc/slabinfo.
>
> Badari,
>
> That extra get_nsproxy() seemed a superfluous remain from the 2.6.20.
> Do you see any issues with it ?
>
> If we're all happy with these fixes, i'll send them on lkml@ for review.
> They might deserve to be in 2.6.22.

the patch looks like it should fix the issue
(will test that soon) but it leaves the comment
unmodified, which is now wrong ...

 * Called from unshare. Unshare all the namespaces part of nsproxy.
 * On sucess, returns the new nsproxy and a reference to old nsproxy
 * to make sure it stays around.

best,
Herbert

> Thanks,
>
> C.
>
> Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
> ---
> kernel/nsproxy.c | 7 +------
> 1 file changed, 1 insertion(+), 6 deletions(-)
>
> Index: 2.6.22-rc4-mm2/kernel/nsproxy.c
> ===
> --- 2.6.22-rc4-mm2.orig/kernel/nsproxy.c
> +++ 2.6.22-rc4-mm2/kernel/nsproxy.c
> @@ -175,7 +175,6 @@ void free_nsproxy(struct nsproxy *ns)
> int unshare_nsproxy_namespaces(unsigned long unshare_flags,
> 		struct nsproxy **new_nsp, struct fs_struct *new_fs)
> {

```
> -       struct nsproxy *old_ns = current->nsproxy;
>         int err = 0;
>
>         if (!(unshare_flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
> @@ -185,14 +184,10 @@ int unshare_nsproxy_namespaces(unsigned
>         if (!capable(CAP_SYS_ADMIN))
>                 return -EPERM;
>
> -       get_nsproxy(old_ns);
> -
>         *new_nsp = create_new_namespaces(unshare_flags, current,
>                                 new_fs ? new_fs : current->fs);
> -       if (IS_ERR(*new_nsp)) {
> +       if (IS_ERR(*new_nsp))
>                 err = PTR_ERR(*new_nsp);
> -               put_nsproxy(old_ns);
> -       }
>         return err;
>  }
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Cedric Le Goater on Mon, 18 Jun 2007 16:44:57 GMT
View Forum Message <> Reply to Message

[...]

>> If we're all happy with these fixes, i'll send them on lkml@ for review.
>> They might deserve to be in 2.6.22.
>
> the patch looks like it should fix the issue
> (will test that soon) but it leaves the comment
> unmodified, which is now wrong ...
>
> * Called from unshare. Unshare all the namespaces part of nsproxy.
> * On sucess, returns the new nsproxy and a reference to old nsproxy
> * to make sure it stays around.

right. I will fix that.

Thanks,

C.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Cedric Le Goater on Mon, 18 Jun 2007 16:54:33 GMT
View Forum Message <> Reply to Message

[...]

>> It fixes the leak for me. I've run the ltp tests we
>> have on namespace unsharing and i could see the no
>> leaks in /proc/slabinfo.
>
>> Badari,
>>
>> That extra get_nsproxy() seemed a superfluous remain
>> from the 2.6.20.
>> Do you see any issues with it ?
>>
>> If we're all happy with these fixes, i'll send them on
>> lkml@ for review.
>
> I'm not terribly happy with the current nsproxy
> framework, although it improved somewhat ...
>
> I'm still missing some mechanism to 'mix' two
> proxies according to a flagmask (which is required
> to enter a guest 'partially') ...

We have that bind_ns() syscall that does that. I sent it last year
but it didn't have much success. We still use and there may be room
for improvement to make it altruistically useful.

Here's the patch on a 2.6.21-mm2. It it's of any interest, I can
refresh it on the latest -mm.

Thanks,

C.

The following patch defines the new bind_ns syscall specific to
nsproxy and namespaces, which allows a process to bind :

 1 - its nsproxy to some identifier

2 - to another nsproxy using an identifier or -pid

Here's a sample user space program to use it.

```
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <libgen.h>
#include <sys/syscall.h>

#ifndef HAVE_UNSHARE

#if __i386__
#    define __NR_unshare 310
#elif __x86_64__
#    define __NR_unshare 272
#elif __ia64__
#    define __NR_unshare 1296
#elif __s390x__
#    define __NR_unshare 303
#elif __powerpc__
#    define __NR_unshare 282
#else
#    error "Architecture not supported"
#endif

#endif /* HAVE_UNSHARE */

#if __i386__
#    define __NR_bind_ns  326
#elif __ia64__
#    define __NR_bind_ns 1305
#elif __powerpc__
#    define __NR_bind_ns 304
#elif __s390x__
#    define __NR_bind_ns 314
#elif __x86_64__
#    define __NR_bind_ns 284
#else
#    error "Architecture not supported"
#endif

#ifndef CLONE_NEWUTS
#define CLONE_NEWUTS  0x04000000
#endif
```

```
#ifndef CLONE_NEWIPC
#define CLONE_NEWIPC  0x08000000
#endif

#ifndef CLONE_NEWUSER
#define CLONE_NEWUSER  0x10000000
#endif

#ifndef CLONE_NEWNET2
#define CLONE_NEWNET2  0x20000000
#endif

#ifndef CLONE_NEWNET3
#define CLONE_NEWNET3  0x40000000
#endif

#ifndef CLONE_NEWPID
#define CLONE_NEWPID  0x80000000
#endif


static inline _syscall1 (int,  unshare, unsigned long, flags)
static inline _syscall2 (int,  bind_ns, int, id, unsigned long, flags)

static const char* procname;

static void usage(const char *name)
{
 printf("usage: %s [-h] [-I id] [-muiUnNp] [command [arg ...]]\n", name);
 printf("\n");
 printf("  -h  this message\n");
 printf("\n");
 printf("  -I <id> bind process to nsproxy <id>\n");
 printf("  -m  mount namespace\n");
 printf("  -u  utsname namespace\n");
 printf("  -i  ipc namespace\n");
 printf("  -U  user namespace\n");
 printf("  -n  net namespace level 2\n");
 printf("  -N  net namespace level 3\n");
 printf("  -p  pid namespace\n");
 printf("\n");
 printf("(C) Copyright IBM Corp. 2006\n");
 printf("\n");
 exit(1);
}

int main(int argc, char *argv[])
```

```
{
 int c;
 unsigned long flags = 0;
 int id = -1;

 procname = basename(argv[0]);

 while ((c = getopt(argc, argv, "+muiUnNphI:")) != EOF) {
  switch (c) {
  case 'I': if (optarg)
    id = atoi(optarg); break;

  case 'm': flags |= CLONE_NEWNS;  break;
  case 'u': flags |= CLONE_NEWUTS;  break;
  case 'i': flags |= CLONE_NEWIPC;  break;
  case 'U': flags |= CLONE_NEWUSER; break;
  case 'n': flags |= CLONE_NEWNET2;  break;
  case 'N': flags |= CLONE_NEWNET3;  break;
  case 'p': flags |= CLONE_NEWPID;  break;
  case 'h':
  default:
   usage(procname);
  }
 };

 argv = &argv[optind];
 argc = argc - optind;

 if (!strcmp(procname, "unsharens")) {
  if (unshare(flags) == -1) {
   perror("unshare");
   return 1;
  }
 }

 if (bind_ns(id, flags) == -1) {
  perror("bind_ns");
  return 1;
 }

 if (argc) {
  execve(argv[0], argv, __environ);
  perror("execve");
  return 1;
 }

 return 0;
}
```

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>

---
 arch/i386/kernel/syscall_table.S | 1
 arch/ia64/kernel/entry.S | 1
 arch/s390/kernel/compat_wrapper.S | 6
 arch/s390/kernel/syscalls.S | 1
 arch/x86_64/ia32/ia32entry.S | 1
 include/asm-i386/unistd.h | 3
 include/asm-ia64/unistd.h | 3
 include/asm-powerpc/systbl.h | 1
 include/asm-powerpc/unistd.h | 3
 include/asm-s390/unistd.h | 3
 include/asm-x86_64/unistd.h | 2
 include/linux/nsproxy.h | 10 -
 include/linux/sched.h | 2
 include/linux/syscalls.h | 2
 kernel/nsproxy.c | 264 ++++++++++++++++++++++++++++++++++++++-
 kernel/sys_ni.c | 2
 16 files changed, 290 insertions(+), 15 deletions(-)

Index: 2.6.21-mm2/include/linux/syscalls.h
===
--- 2.6.21-mm2.orig/include/linux/syscalls.h
+++ 2.6.21-mm2/include/linux/syscalls.h
@@ -609,6 +609,8 @@ asmlinkage long sys_timerfd(int ufd, int
        const struct itimerspec __user *utmr);
 asmlinkage long sys_eventfd(unsigned int count);

+asmlinkage long sys_bind_ns(int id, unsigned long unshare_flags);
+
 int kernel_execve(const char *filename, char *const argv[], char *const envp[]);

 asmlinkage long sys_revokeat(int dfd, const char __user *filename);
Index: 2.6.21-mm2/kernel/nsproxy.c
===
--- 2.6.21-mm2.orig/kernel/nsproxy.c
+++ 2.6.21-mm2/kernel/nsproxy.c
@@ -22,7 +22,11 @@

 static struct kmem_cache *nsproxy_cachep;

-#define NS_HASH_BITS   3 /* this might need some configuration */
+/*
+ * nsproxies are stored in a hash but a rbtree might be more
+ * appropriate.
+ */

```
+#define NS_HASH_BITS   3
 #define NS_HASH_SIZE  (1 << NS_HASH_BITS)
 #define NS_HASH_MASK  (NS_HASH_SIZE - 1)
 #define ns_hashfn(id)  (((id >> NS_HASH_BITS) + id) & NS_HASH_MASK)
@@ -63,6 +67,30 @@ static inline struct nsproxy *clone_nspr
 }

 /*
+ * copies the nsproxy, setting refcount to 1, and grabbing a
+ * reference to all contained namespaces.  Called from
+ * sys_unshare()
+ */
+static struct nsproxy *dup_namespaces(struct nsproxy *orig)
+{
+ struct nsproxy *ns = clone_nsproxy(orig);
+
+ if (ns) {
+  if (ns->mnt_ns)
+   get_mnt_ns(ns->mnt_ns);
+  if (ns->uts_ns)
+   get_uts_ns(ns->uts_ns);
+  if (ns->ipc_ns)
+   get_ipc_ns(ns->ipc_ns);
+  if (ns->pid_ns)
+   get_pid_ns(ns->pid_ns);
+  if (ns->user_ns)
+   get_user_ns(ns->user_ns);
+ }
+
+ return ns;
+}
+/*
  * Create new nsproxy and all of its the associated namespaces.
  * Return the newly created nsproxy.  Do not attach this to the task,
  * leave it to the caller to do proper locking and attach it to task.
@@ -123,7 +151,7 @@ int copy_namespaces(int flags, struct ta

  get_nsproxy(old_ns);

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

  if (!capable(CAP_SYS_ADMIN)) {
@@ -143,7 +171,7 @@ out:
  return err;
 }
```

```
-void free_nsproxy(struct nsproxy *ns)
+static void free_nsproxy(struct nsproxy *ns)
 {
  if (ns->mnt_ns)
   put_mnt_ns(ns->mnt_ns);
@@ -157,6 +185,29 @@ void free_nsproxy(struct nsproxy *ns)
 }

 /*
+ * put_nsproxy() is similar to free_uid() in kernel/user.c
+ *
+ * the lock can be taken from a tasklet context (task getting freed by
+ * RCU) which requires to be irq safe.
+ */
+void put_nsproxy(struct nsproxy *ns)
+{
+ unsigned long flags;
+
+ local_irq_save(flags);
+ if (atomic_dec_and_lock(&ns->count, &ns_hash_lock)) {
+  BUG_ON(!ns->id);
+  if (ns->id != -1)
+   hlist_del(&ns->ns_hash_node);
+  spin_unlock_irqrestore(&ns_hash_lock, flags);
+  free_nsproxy(ns);
+ } else {
+  local_irq_restore(flags);
+ }
+}
+EXPORT_SYMBOL_GPL(put_nsproxy);
+
++/*
  * Called from unshare. Unshare all the namespaces part of nsproxy.
  * On sucess, returns the new nsproxy and a reference to old nsproxy
  * to make sure it stays around.
@@ -211,6 +262,212 @@ static inline struct nsproxy *ns_hash_fi
  return NULL;
 }

+struct nsproxy *find_nsproxy_by_id(int id)
+{
+ struct nsproxy *ns;
+
+ if (id < 0)
+  return NULL;
+
+ spin_lock_irq(&ns_hash_lock);
+ ns = ns_hash_find(id);
```

```
+ spin_unlock_irq(&ns_hash_lock);
+
+  return ns;
+}
+
+EXPORT_SYMBOL_GPL(find_nsproxy_by_id);
+
+static int bind_ns(int id, struct nsproxy *ns)
+{
+  struct nsproxy *prev;
+ int ret = 0;
+
+  if (id < 0)
+   return -EINVAL;
+
+  spin_lock_irq(&ns_hash_lock);
+  prev = ns_hash_find(id);
+  if (!prev) {
+  ns->id = id;
+  hlist_add_head(&ns->ns_hash_node, ns_hash_head(ns->id));
+ }
+ spin_unlock_irq(&ns_hash_lock);
+
+ if (prev) {
+  ret = -EBUSY;
+  put_nsproxy(prev);
+ }
+  return ret;
+}
+
+static int switch_ns(int id, unsigned long flags)
+{
+ int err = 0;
+ struct nsproxy *ns = NULL, *old_ns = NULL, *new_ns = NULL;
+
+ if (flags & ~NS_ALL)
+  return -EINVAL;
+
+ /* Let 0 be a default value ? */
+ if (!flags)
+  flags = NS_ALL;
+
+ if (id < 0) {
+  struct task_struct *p;
+
+  err = -ESRCH;
+  read_lock(&tasklist_lock);
+  p = find_task_by_pid(-id);
```

```
+  if (p) {
+   task_lock(p);
+   get_nsproxy(p->nsproxy);
+   ns = p->nsproxy;
+   task_unlock(p);
+  }
+  read_unlock(&tasklist_lock);
+ } else {
+  err = -ENOENT;
+  spin_lock_irq(&ns_hash_lock);
+  ns = ns_hash_find(id);
+  spin_unlock_irq(&ns_hash_lock);
+ }
+
+ if (!ns)
+  goto out;
+
+ new_ns = ns;
+
+ /*
+  * clone current nsproxy and populate it with the namespaces
+  * chosen by flags.
+  */
+ if (flags != NS_ALL) {
+  new_ns = dup_namespaces(current->nsproxy);
+  if (!new_ns) {
+   err = -ENOMEM;
+   goto out_ns;
+  }
+
+  if (flags & CLONE_NEWNS) {
+   put_mnt_ns(new_ns->mnt_ns);
+   get_mnt_ns(ns->mnt_ns);
+   new_ns->mnt_ns = ns->mnt_ns;
+  }
+
+  if (flags & CLONE_NEWUTS) {
+   put_uts_ns(new_ns->uts_ns);
+   get_uts_ns(ns->uts_ns);
+   new_ns->uts_ns = ns->uts_ns;
+  }
+
+  if (flags & CLONE_NEWIPC) {
+   put_ipc_ns(new_ns->ipc_ns);
+   new_ns->ipc_ns = get_ipc_ns(ns->ipc_ns);
+  }
+
+  if (flags & CLONE_NEWUSER) {
```

```
+   put_user_ns(new_ns->user_ns);
+   get_user_ns(ns->user_ns);
+   new_ns->user_ns = ns->user_ns;
+  }
+
+ out_ns:
+  put_nsproxy(ns);
+ }
+
+ task_lock(current);
+ if (new_ns) {
+  old_ns = current->nsproxy;
+  current->nsproxy = new_ns;
+ }
+ task_unlock(current);
+
+ if (old_ns)
+  put_nsproxy(old_ns);
+
+ err = 0;
+out:
+ return err;
+}
+
+
+/*
+ * bind_ns - bind the nsproxy of a task to an id or bind a task to a
+ *           identified nsproxy
+ *
+ * @id: nsproxy identifier if positive or pid if negative
+ * @flags: identifies the namespaces to bind to
+ *
+ * bind_ns serves 2 purposes.
+ *
+ * The first is to bind the nsproxy of the current task to the
+ * identifier @id. If the identifier is already used, -EBUSY is
+ * returned. If the nsproxy is already bound, -EACCES is returned.
+ * flags is not used in that case.
+ *
+ * The second use is to bind the current task to a subset of
+ * namespaces of an identified nsproxy. If positive, @id is considered
+ * being an nsproxy identifier previously used to bind the nsproxy to
+ * @id. If negative, @id is the pid of a task which is another way to
+ * identify a nsproxy. Switching nsproxy is restricted to tasks within
+ * nsproxy 0, the default nsproxy. If unknown, -ENOENT is returned.
+ * @flags is used to bind the task to the selected namespaces.
+ *
+ * Both uses may return -EINVAL for invalid arguments and -EPERM for
```

```
+ * insufficient privileges.
+ *
+ * Returns 0 on success.
+ */
+asmlinkage long sys_bind_ns(int id, unsigned long flags)
+{
+ struct nsproxy *ns = current->nsproxy;
+ int ret = 0;
+
+ /*
+  * ns is being changed by switch_ns(), protect it
+  */
+ get_nsproxy(ns);
+
+ /*
+  * protect ns->id
+  */
+ spin_lock(&ns->nslock);
+ switch (ns->id) {
+ case -1:
+  /*
+   * only an unbound nsproxy can be bound to an id.
+   */
+  ret = bind_ns(id, ns);
+  break;
+
+ case 0:
+  if (!capable(CAP_SYS_ADMIN)) {
+   ret = -EPERM;
+   goto unlock;
+  }
+
+  /*
+   * only nsproxy 0 can switch nsproxy. if target id is
+   * 0, this is a nop.
+   */
+  if (id)
+   ret = switch_ns(id, flags);
+  break;
+
+ default:
+  /*
+   * current nsproxy is already bound. forbid any
+   * switch.
+   */
+  ret = -EACCES;
+ }
+unlock:
```

```
+ spin_unlock(&ns->nslock);
+ put_nsproxy(ns);
+ return ret;
+}
+
 static int __init nshash_init(void)
 {
  int i;
@@ -235,3 +492,4 @@ static int __init nsproxy_cache_init(voi
 }

 module_init(nsproxy_cache_init);
+
Index: 2.6.21-mm2/kernel/sys_ni.c
===================================================================
--- 2.6.21-mm2.orig/kernel/sys_ni.c
+++ 2.6.21-mm2/kernel/sys_ni.c
@@ -146,3 +146,5 @@ cond_syscall(sys_ioprio_get);
 cond_syscall(sys_signalfd);
 cond_syscall(sys_timerfd);
 cond_syscall(sys_eventfd);
+
+cond_syscall(sys_bind_ns);
Index: 2.6.21-mm2/arch/ia64/kernel/entry.S
===================================================================
--- 2.6.21-mm2.orig/arch/ia64/kernel/entry.S
+++ 2.6.21-mm2/arch/ia64/kernel/entry.S
@@ -1583,5 +1583,6 @@ sys_call_table:
  data8 sys_vmsplice
  data8 sys_ni_syscall   // reserved for move_pages
  data8 sys_getcpu
+ data8 sys_bind_ns

  .org sys_call_table + 8*NR_syscalls // guard against failures to increase NR_syscalls
Index: 2.6.21-mm2/arch/s390/kernel/compat_wrapper.S
===================================================================
--- 2.6.21-mm2.orig/arch/s390/kernel/compat_wrapper.S
+++ 2.6.21-mm2/arch/s390/kernel/compat_wrapper.S
@@ -1682,3 +1682,9 @@ compat_sys_utimes_wrapper:
  llgtr %r2,%r2   # char *
  llgtr %r3,%r3   # struct compat_timeval *
  jg compat_sys_utimes
+
+ .globl sys_bind_ns_wrapper
+sys_bind_ns_wrapper:
+ lgfr    %r2,%r2              # int
+ llgfr   %r3,%r3              # unsigned long
+ jg      sys_bind_ns
```

```
Index: 2.6.21-mm2/arch/s390/kernel/syscalls.S
===================================================================
--- 2.6.21-mm2.orig/arch/s390/kernel/syscalls.S
+++ 2.6.21-mm2/arch/s390/kernel/syscalls.S
@@ -322,3 +322,4 @@ NI_SYSCALL       /* 310 sys_move_pages *
 SYSCALL(sys_getcpu,sys_getcpu,sys_getcpu_wrapper)
 SYSCALL(sys_epoll_pwait,sys_epoll_pwait,compat_sys_epoll_pwait_wrapper)
 SYSCALL(sys_utimes,sys_utimes,compat_sys_utimes_wrapper)
+SYSCALL(sys_bind_ns,sys_bind_ns,sys_bind_ns_wrapper)
Index: 2.6.21-mm2/arch/x86_64/ia32/ia32entry.S
===================================================================
--- 2.6.21-mm2.orig/arch/x86_64/ia32/ia32entry.S
+++ 2.6.21-mm2/arch/x86_64/ia32/ia32entry.S
@@ -721,4 +721,5 @@ ia32_sys_call_table:
  .quad sys_eventfd
  .quad sys_revokeat
  .quad sys_frevoke  /* 325 */
+ .quad sys_bind_ns
 ia32_syscall_end:
Index: 2.6.21-mm2/include/asm-ia64/unistd.h
===================================================================
--- 2.6.21-mm2.orig/include/asm-ia64/unistd.h
+++ 2.6.21-mm2/include/asm-ia64/unistd.h
@@ -294,11 +294,12 @@
 #define __NR_vmsplice   1302
 /* 1303 reserved for move_pages */
 #define __NR_getcpu   1304
+#define __NR_bind_ns   1305

 #ifdef __KERNEL__


-#define NR_syscalls   281 /* length of syscall table */
+#define NR_syscalls   282 /* length of syscall table */

 #define __ARCH_WANT_SYS_RT_SIGACTION
 #define __ARCH_WANT_SYS_RT_SIGSUSPEND
Index: 2.6.21-mm2/include/asm-powerpc/systbl.h
===================================================================
--- 2.6.21-mm2.orig/include/asm-powerpc/systbl.h
+++ 2.6.21-mm2/include/asm-powerpc/systbl.h
@@ -307,3 +307,4 @@ COMPAT_SYS_SPU(set_robust_list)
 COMPAT_SYS_SPU(move_pages)
 SYSCALL_SPU(getcpu)
 COMPAT_SYS(epoll_pwait)
+SYSCALL_SPU(bind_ns)
Index: 2.6.21-mm2/include/asm-powerpc/unistd.h
===================================================================
```

```
--- 2.6.21-mm2.orig/include/asm-powerpc/unistd.h
+++ 2.6.21-mm2/include/asm-powerpc/unistd.h
@@ -326,10 +326,11 @@
 #define __NR_move_pages  301
 #define __NR_getcpu  302
 #define __NR_epoll_pwait 303
+#define __NR_bind_ns  304

 #ifdef __KERNEL__

-#define __NR_syscalls  304
+#define __NR_syscalls  305

 #define __NR__exit __NR_exit
 #define NR_syscalls __NR_syscalls
Index: 2.6.21-mm2/include/asm-s390/unistd.h
===================================================================
--- 2.6.21-mm2.orig/include/asm-s390/unistd.h
+++ 2.6.21-mm2/include/asm-s390/unistd.h
@@ -251,8 +251,9 @@
 #define __NR_getcpu  311
 #define __NR_epoll_pwait 312
 #define __NR_utimes  313
+#define __NR_bind_ns  314

-#define NR_syscalls 314
+#define NR_syscalls 315

 /*
  * There are some system calls that are not present on 64 bit, some
Index: 2.6.21-mm2/include/asm-x86_64/unistd.h
===================================================================
--- 2.6.21-mm2.orig/include/asm-x86_64/unistd.h
+++ 2.6.21-mm2/include/asm-x86_64/unistd.h
@@ -627,6 +627,8 @@ __SYSCALL(__NR_signalfd, sys_signalfd)
 __SYSCALL(__NR_timerfd, sys_timerfd)
 #define __NR_eventfd  283
 __SYSCALL(__NR_eventfd, sys_eventfd)
+#define __NR_bind_ns  284
+__SYSCALL(__NR_bind_ns, sys_bind_ns)

 #ifndef __NO_STUBS
 #define __ARCH_WANT_OLD_READDIR
Index: 2.6.21-mm2/include/linux/sched.h
===================================================================
--- 2.6.21-mm2.orig/include/linux/sched.h
+++ 2.6.21-mm2/include/linux/sched.h
@@ -29,6 +29,8 @@
```

```
 #define CLONE_NEWPID  0x10000000      /* New pid namespace */
 #define CLONE_NEWUSER  0x20000000      /* New user namespace */

+#define NS_ALL (CLONE_NEWNS|CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWUSER)
+
 /*
  * Scheduling policies
  */
Index: 2.6.21-mm2/arch/i386/kernel/syscall_table.S
===================================================================
--- 2.6.21-mm2.orig/arch/i386/kernel/syscall_table.S
+++ 2.6.21-mm2/arch/i386/kernel/syscall_table.S
@@ -325,3 +325,4 @@ ENTRY(sys_call_table)
  .long sys_eventfd
  .long sys_revokeat
  .long sys_frevoke  /* 325 */
+ .long sys_bind_ns
Index: 2.6.21-mm2/include/asm-i386/unistd.h
===================================================================
--- 2.6.21-mm2.orig/include/asm-i386/unistd.h
+++ 2.6.21-mm2/include/asm-i386/unistd.h
@@ -331,10 +331,11 @@
 #define __NR_eventfd  323
 #define __NR_revokeat  324
 #define __NR_frevoke  325
+#define __NR_bind_ns  326

 #ifdef __KERNEL__

-#define NR_syscalls 326
+#define NR_syscalls 327

 #define __ARCH_WANT_IPC_PARSE_VERSION
 #define __ARCH_WANT_OLD_READDIR
Index: 2.6.21-mm2/include/linux/nsproxy.h
===================================================================
--- 2.6.21-mm2.orig/include/linux/nsproxy.h
+++ 2.6.21-mm2/include/linux/nsproxy.h
@@ -36,16 +36,10 @@ extern struct nsproxy init_nsproxy;

 int copy_namespaces(int flags, struct task_struct *tsk);
 void get_task_nsproxy(struct task_struct *tsk);
-void free_nsproxy(struct nsproxy *ns);
 int unshare_nsproxy_namespaces(unsigned long, struct nsproxy **,
  struct fs_struct *);
-
-static inline void put_nsproxy(struct nsproxy *ns)
-{
```

```
- if (atomic_dec_and_test(&ns->count)) {
-  free_nsproxy(ns);
- }
-}
+void put_nsproxy(struct nsproxy *ns);
+struct nsproxy *find_nsproxy_by_id(int id);

 static inline void put_task_nsproxy(struct task_struct *p)
 {
```

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Cedric Le Goater on Mon, 18 Jun 2007 17:00:04 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Mon, Jun 18, 2007 at 02:02:19PM +0200, Cedric Le Goater wrote:
>>> on Linux-VServer,we have accounting for those
>>> proxies (and several other namespace related stuff)
>>> because we already suspected leakage and reference
>>> bugs in this area some time ago ... btw, I also
>>> suggested to put a similar functionality in mainline
>>> for the time being, but it was ignored, as usual ...
>> something like a kmem_cache ?
>
> maybe, but even simplest accounting of the
> different 'objects' would be more then enough
> for the test phase (or maybe as statistics :)

Sure but Linux-VServer would certainly beneficiate from a nsproxy
cache. Some scenarii do a lot of unshare(), nop ?

Anyway, i wouldn't know how to measure the difference :) but
it certainly helped me to identify the leak real fast !

>> and we are not ignoring vserver ! :)
>
> good for them, our project is called Linux-VServer :)

arg :)

Cheers,

C.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: - merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch removed from -mm tree
Posted by Badari Pulavarty on Mon, 18 Jun 2007 17:35:14 GMT
View Forum Message <> Reply to Message

On Mon, 2007-06-18 at 14:37 +0200, Cedric Le Goater wrote:
> Herbert Poetzl wrote:
> > On Sun, Jun 17, 2007 at 06:38:30PM +0400, Oleg Nesterov wrote:
> >> On 06/16, Herbert Poetzl wrote:
> >>> On Tue, May 08, 2007 at 07:45:35PM -0700, akpm@linux-foundation.org wrote:
> >>>> The patch titled
> >>>>   Merge sys_clone()/sys_unshare() nsproxy and namespace handling
> >>>> has been removed from the -mm tree. Its filename was
> >>>>   merge-sys_clone-sys_unshare-nsproxy-and-namespace.patch
> >>>>
> >>>> This patch was dropped because it was merged into mainline or a subsystem tree
> >>>>
> >>> .. [zapped] ...
> >>>
> >>> + * Called from unshare. Unshare all the namespaces part of nsproxy.
> >>> + * On sucess, returns the new nsproxy and a reference to old nsproxy
> >>> + * to make sure it stays around.
> >>> + */
> >>> +int unshare_nsproxy_namespaces(unsigned long unshare_flags,
> >>> + struct nsproxy **new_nsp, struct fs_struct *new_fs)
> >>> +{
> >>> this makes sys_unshare leak and nsproxy (reference)
> >>>
> >>> can be tested with the following command sequence:
> >>>   vcmd -nu ^17 -- vcmd -nu ^17 -- sleep 10
> >> I know almost nothing about this stuff, could you please explain in
> >> brief what this command does ...
> >
> > yeah, sure, it basically calls sys_unshare() with
> > bit 17 (CLONE_NEWNS) set then invokes the chained
> > command, so we get a sleep which is in a separate
> > namespace, unshared from a namespace != the main
> > one ...
> >
> >> ... and how do you detect a leak?
> >

> >>> (and some nsproxy accounting/debugging as used in
> >>> Linux-VServer)
> >
> > on Linux-VServer,we have accounting for those
> > proxies (and several other namespace related stuff)
> > because we already suspected leakage and reference
> > bugs in this area some time ago ... btw, I also
> > suggested to put a similar functionality in mainline
> > for the time being, but it was ignored, as usual ...
> >
> >>> we probably want to drop the reference to the old
> >>> nsproxy in sys_unshare() but I do not see a good reason
> >>> to take the reference in the first place (at least not
> >>> with the code in mainline 2.6.22-rc4)
> >> At first glance, sys_unshare() drops the reference to
> >> the old nsproxy,
> >
> > okay, the 'current' task has an nsproxy, and keeps
> > a reference to that (let's assume it is the only
> > task using this nsproxy, then the count will be 1)
> >
> > unshare_nsproxy_namespaces() now does get_nsproxy()
> > which makes the count=2, then it creates a new
> > nsproxy (which will get count=1), and returns ...
> >
> >>   old_nsproxy = current->nsproxy;
> >>   current->nsproxy = new_nsproxy;
> >>   new_nsproxy = old_nsproxy;
> >
> > sys_unshare, now replaces the current->nsproxy with
> > the new one, which will have the correct count=1,
> > and puts the old nsproxy (which has count=2), and
> > thus the nsproxy will not get released, although
> > it isn't referenced/used anymore ...
>
>
> Herbert,
>
> Could you give a try to the patch i've sent previously and this one
> which removes an extra get_nsproxy() ? It fixes the leak for me. I've
> run the ltp tests we have on namespace unsharing and i could see the
> no leaks in /proc/slabinfo.
>
> Badari,
>
> That extra get_nsproxy() seemed a superfluous remain from the 2.6.20.
> Do you see any issues with it ?
>

> If we're all happy with these fixes, i'll send them on lkml@ for review.
> They might deserve to be in 2.6.22.
>
> Thanks,
>
> C.

Cedric, Oleg and Herbert,

Thanks for working this out. Looks good.

Thanks,
Badari

>
> Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>

Acked.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH] create_new_namespaces: fix improper return of NULL
Posted by Oleg Nesterov on Tue, 19 Jun 2007 13:51:29 GMT
View Forum Message <> Reply to Message

Untested.

dup_mnt_ns() and clone_uts_ns() return NULL on failure. This is wrong,
create_new_namespaces() uses ERR_PTR() to catch an error. This means
that the subsequent create_new_namespaces() will hit BUG_ON() in
copy_mnt_ns() or copy_utsname().

Signed-off-by: Oleg Nesterov <oleg@tv-sign.ru>

```
--- ns/fs/namespace.c~1_NS_NULL 2007-05-21 13:57:56.000000000 +0400
+++ ns/fs/namespace.c 2007-06-19 17:26:35.000000000 +0400
@@ -1457,7 +1457,7 @@ static struct mnt_namespace *dup_mnt_ns(

  new_ns = kmalloc(sizeof(struct mnt_namespace), GFP_KERNEL);
  if (!new_ns)
-  return NULL;
+  return ERR_PTR(-ENOMEM);
```

```
  atomic_set(&new_ns->count, 1);
  INIT_LIST_HEAD(&new_ns->list);
@@ -1471,7 +1471,7 @@ static struct mnt_namespace *dup_mnt_ns(
  if (!new_ns->root) {
   up_write(&namespace_sem);
   kfree(new_ns);
-  return NULL;
+  return ERR_PTR(-ENOMEM);
  }
  spin_lock(&vfsmount_lock);
  list_add_tail(&new_ns->list, &new_ns->root->mnt_list);
--- ns/kernel/utsname.c~1_NS_NULL 2007-05-21 13:57:59.000000000 +0400
+++ ns/kernel/utsname.c 2007-06-19 17:35:22.000000000 +0400
@@ -13,6 +13,7 @@
 #include <linux/uts.h>
 #include <linux/utsname.h>
 #include <linux/version.h>
+#include <linux/err.h>

 /*
  * Clone a new ns copying an original utsname, setting refcount to 1
@@ -24,10 +25,11 @@ static struct uts_namespace *clone_uts_n
  struct uts_namespace *ns;

  ns = kmalloc(sizeof(struct uts_namespace), GFP_KERNEL);
- if (ns) {
-  memcpy(&ns->name, &old_ns->name, sizeof(ns->name));
-  kref_init(&ns->kref);
- }
+ if (!ns)
+  return ERR_PTR(-ENOMEM);
+
+ memcpy(&ns->name, &old_ns->name, sizeof(ns->name));
+ kref_init(&ns->kref);
  return ns;
 }
```

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers