
Subject: namespace and nsproxy syscalls

Posted by [Cedric Le Goater](#) on Tue, 26 Sep 2006 09:42:10 GMT

[View Forum Message](#) <> [Reply to Message](#)

Hello all,

A while ago, we expressed the need to have a new syscall specific to namespaces. the clone and unshare are good candidates but we are reaching the limit of the clone flags and clone has been hijacked enough.

So, I came up with unshare_ns. the patch for the core feature follows the email. Not much difference with unshare() for the moment but it gives us the freedom to diverge when new namespaces come in. I have faith also ! If you feel it's useful, i'll send the full patchset for review on the list.

I'd like to discuss of another syscall which would allow a process to bind to a set of namespaces (== nsproxy == container) :

```
bind_ns(ns_id_t id, int flags)
```

bind_ns binds the current nsproxy to an id. You can only bind once and you can use this id to bind another process to the same nsproxy.

a few comments :

- * ns_id_t could be an int, a const char*, a struct ns_addr*. this is to be defined.
- * bind_ns applies to nsproxy and not to namespaces. we could bind a specific namespace to an id using flags but i don't see the need. (but why not with a flags 0 defining ALL namespaces)
- * semantic is close to shmat but i don't think we need a shmdt because nsproxies are not resilient objects.

Thanks,

C.

From: Cedric Le Goater <clg@fr.ibm.com>

Subject: add unshare_ns syscall core routine

This patch adds the unshare_ns syscall core routine.

This syscall is an unshare dedicated to namespaces.

sample user program :

```
/*
```

```

* unshare_ns.c
*
* author: Cedric Le Goater <clg@fr.ibm.com>
*
* (C) Copyright IBM Corp. 2005, 2006
*
* This program is free software; you can redistribute it and/or
* modify it under the terms of the GNU General Public License
* as published by the Free Software Foundation; either version 2
* of the License, or (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
* 02110-1301, USA.
*/

```

```

#include <stdio.h>
#include <stdlib.h>
#include <sched.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>

```

```

#include <linux/unistd.h>

```

```

#ifndef __NR_unshare
#ifdef __i386__
# define __NR_unshare_ns 319
#elif __x86_64__
# define __NR_unshare_ns 280
#elif __ia64__
# define __NR_unshare_ns 1303
#elif __s390x__
# define __NR_unshare_ns 310
#elif __powerpc__
# define __NR_unshare_ns 301
#else
# error "Architecture not supported"
#endif
#endif

```

```

static inline _syscall1(int, unshare_ns, int, flags)

```

```

#define UNSHARE_NS_MNT 0x00000001
#define UNSHARE_NS_UTS 0x00000002
#define UNSHARE_NS_IPC 0x00000004
#define UNSHARE_NS_USER 0x00000008
#define UNSHARE_NS_NET 0x00000010
#define UNSHARE_NS_PID 0x00000020

static void usage(const char *name)
{
    printf("usage: %s [-Hiunmpeh]\n", name);
    printf("\t-H : unshare utsname namespace.\n");
    printf("\t-i : unshare ipc namespace.\n");
    printf("\t-u : unshare user namespace.\n");
    printf("\t-n : unshare net namespace.\n");
    printf("\t-m : unshare mount namespace.\n");
    printf("\t-p : unshare pid namespace.\n");
    printf("\t-e : exec command.\n");
    printf("\n");
    printf("(C) Copyright IBM Corp. 2005, 2006\n");
    printf("\n");
    exit(1);
}

int main(int argc, char* argv[])
{
    int c;
    unsigned long flag = 0;
    int exec = 0;

    while ((c = getopt(argc, argv, "+Hiunmpeh")) != EOF) {
        switch (c) {
            case 'i': flag |= UNSHARE_NS_IPC; break;
            case 'u': flag |= UNSHARE_NS_USER; break;
            case 'H': flag |= UNSHARE_NS_UTS; break;
            case 'n': flag |= UNSHARE_NS_NET; break;
            case 'm': flag |= UNSHARE_NS_MNT; break;
            case 'p': flag |= UNSHARE_NS_PID; break;
            case 'e': exec = 1; break;
            case 'h':
            default:
                usage(argv[0]);
        }
    };

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

```

```

if (unshare_ns(flag) == -1) {
    perror("unshare_ns");
    return 1;
}

if (exec) {
    execve(argv[0], argv, __environ);
    fprintf(stderr, "execve(%s) : %s\n", argv[0], strerror(errno));
    return 1;
}

return 0;
}

```

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

```

---
include/linux/sched.h | 10 ++++
include/linux/syscalls.h | 1
kernel/fork.c | 106 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
3 files changed, 117 insertions(+)

```

Index: 2.6.18-mm1/include/linux/sched.h

=====

```

--- 2.6.18-mm1.orig/include/linux/sched.h
+++ 2.6.18-mm1/include/linux/sched.h
@@ -28,6 +28,16 @@
#define CLONE_NEWIPC 0x08000000 /* New ipc */

/*
+ * unshare_ns flags:
+ */
+#define UNSHARE_NS_MNT 0x00000001
+#define UNSHARE_NS_UTS 0x00000002
+#define UNSHARE_NS_IPC 0x00000004
+#define UNSHARE_NS_USER 0x00000008
+#define UNSHARE_NS_NET 0x00000010
+#define UNSHARE_NS_PID 0x00000020
+
+/*
+ * Scheduling policies
+ */
#define SCHED_NORMAL 0

```

Index: 2.6.18-mm1/include/linux/syscalls.h

=====

```

--- 2.6.18-mm1.orig/include/linux/syscalls.h
+++ 2.6.18-mm1/include/linux/syscalls.h

```

```

@@ -580,6 +580,7 @@ asmlinkage long compat_sys_newfstatat(un
asmlinkage long compat_sys_openat(unsigned int dfd, const char __user *filename,
    int flags, int mode);
asmlinkage long sys_unshare(unsigned long unshare_flags);
+asmlinkage long sys_unshare_ns(unsigned long unshare_flags);

asmlinkage long sys_splice(int fd_in, loff_t __user *off_in,
    int fd_out, loff_t __user *off_out,
Index: 2.6.18-mm1/kernel/fork.c
=====
--- 2.6.18-mm1.orig/kernel/fork.c
+++ 2.6.18-mm1/kernel/fork.c
@@ -1761,3 +1761,109 @@ bad_unshare_cleanup_thread:
bad_unshare_out:
    return err;
}
+
+/*
+ * unshare_ns allows a process to 'unshare' one or more of its
+ * namespaces which were originally shared using clone.
+ */
+asmlinkage long sys_unshare_ns(unsigned long unshare_ns_flags)
+{
+ int err = 0;
+ struct nsproxy *new_nsproxy = NULL, *old_nsproxy = NULL;
+ struct fs_struct *fs, *new_fs = NULL;
+ struct mnt_namespace *mnt, *new_mnt = NULL;
+ struct uts_namespace *uts, *new_uts = NULL;
+ struct ipc_namespace *ipc, *new_ipc = NULL;
+ unsigned long unshare_flags = 0;
+
+ /* Return -EINVAL for all unsupported flags */
+ err = -EINVAL;
+ if (unshare_ns_flags & ~(UNSHARE_NS_MNT|UNSHARE_NS_UTS|UNSHARE_NS_IPC|
+    UNSHARE_NS_USER|UNSHARE_NS_NET|
+    UNSHARE_NS_PID))
+    goto bad_unshare_ns_out;
+
+ /* convert unshare_ns flags to clone flags */
+ if (unshare_ns_flags & UNSHARE_NS_MNT)
+    unshare_flags |= CLONE_NEWNS|CLONE_FS;
+ if (unshare_ns_flags & UNSHARE_NS_UTS)
+    unshare_flags |= CLONE_NEWUTS;
+ if (unshare_ns_flags & UNSHARE_NS_IPC)
+    unshare_flags |= CLONE_NEWIPC;
+
+ if ((err = unshare_fs(unshare_flags, &new_fs)))
+    goto bad_unshare_ns_out;

```

```

+ if ((err = unshare_mnt_namespace(unshare_flags, &new_mnt, new_fs)))
+ goto bad_unshare_ns_cleanup_fs;
+ if ((err = unshare_utsname(unshare_flags, &new_uts)))
+ goto bad_unshare_ns_cleanup_mnt;
+ if ((err = unshare_ipcs(unshare_flags, &new_ipc)))
+ goto bad_unshare_ns_cleanup_uts;
+
+ if (new_mnt || new_uts || new_ipc) {
+ old_nsproxy = current->nsproxy;
+ new_nsproxy = dup_namespaces(old_nsproxy);
+ if (!new_nsproxy) {
+ err = -ENOMEM;
+ goto bad_unshare_ns_cleanup_ipc;
+ }
+ }
+
+ if (new_fs || new_mnt || new_uts || new_ipc) {
+
+ task_lock(current);
+
+ if (new_nsproxy) {
+ current->nsproxy = new_nsproxy;
+ new_nsproxy = old_nsproxy;
+ }
+
+ if (new_fs) {
+ fs = current->fs;
+ current->fs = new_fs;
+ new_fs = fs;
+ }
+
+ if (new_mnt) {
+ mnt = current->nsproxy->mnt_ns;
+ current->nsproxy->mnt_ns = new_mnt;
+ new_mnt = mnt;
+ }
+
+ 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_ns_cleanup_ipc:
+ if (new_ipc)
+ put_ipc_ns(new_ipc);
+
+bad_unshare_ns_cleanup_uts:
+ if (new_uts)
+ put_uts_ns(new_uts);
+
+bad_unshare_ns_cleanup_mnt:
+ if (new_mnt)
+ put_mnt_ns(new_mnt);
+
+bad_unshare_ns_cleanup_fs:
+ if (new_fs)
+ put_fs_struct(new_fs);
+
+bad_unshare_ns_out:
+ return err;
+}

```

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

Subject: Re: namespace and nsproxy syscalls
Posted by [Herbert Poetzl](#) on Tue, 26 Sep 2006 15:52:53 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Tue, Sep 26, 2006 at 11:42:10AM +0200, Cedric Le Goater wrote:

```

> Hello all,
>
> A while ago, we expressed the need to have a new syscall specific to
> namespaces. the clone and unshare are good candidates but we are reaching
> the limit of the clone flags and clone has been hijacked enough.
>
> So, I came up with unshare_ns. the patch for the core feature follows
> the email. Not much difference with unshare() for the moment but it gives
> us the freedom to diverge when new namespaces come in. I have faith also !
> If you feel it's useful, i'll send the full patchset for review on the list.
>

```

> I'd like to discuss of another syscall which would allow a process to
 > bind to a set of namespaces (== nsproxy == container) :
 >
 > bind_ns(ns_id_t id, int flags)
 >
 > bind_ns binds the current nsproxy to an id. You can only bind once and
 > you can use this id to bind another process to the same nsproxy.
 >
 > a few comments :
 >
 > * ns_id_t could be an int, a const char*, a struct ns_addr*. this is
 > to be defined.
 > * bind_ns applies to nsproxy and not to namespaces. we could bind a
 > specific namespace to an id using flags but i don't see the need.

what if you have a setup like this:

- guest using a full set of namespaces
- host admin wants to 'adjust' a mount inside

typically the host admin would enter the filesystem namespace (at least it is done so in Linux-VServer) but would avoid entering the other namespaces, like the pid-space or the container itself, just to do the mount without 'appearing' inside the guest and more important, with more priviledges than the guest processes would have ...

IMHO that would require to setup two nsproxies which _share_ a certain namespace, but how would I do that? the only way I see here is to create some kind of nsproxy hierarchy when the guest is started like this:

- create filesystem namespace
- do a bind_ns()
- create network namespace
- do another bind_ns()
- create uts/ipc/pid namespace
- do another bind_ns()
- ...

what if I only want to enter the network namespace at some point?

IMHO some kind of unique (e.g. the virtual address of the kernel structure) identifier for every namespace would be a very good idea to have, in

combination with a method to 'build' a proxy on the fly (e.g. by simply 'joining' the namespaces you want to use) makes the above trivial and simple

an alternative would be to have some 'copy partial' method to 'pick' certain namespaces from an existing nsproxy, but personally I would prefer to keep the nsproxy invisible and to work on namespaces only (IIRC, that was what we planned at the beginning of the nsproxy discussion)

best,
Herbert

```
> (but why not with a flags 0 defining ALL namespaces)
> * semantic is close to shmat but i don't think we need a shmdt because
> nsproxies are not resilient objects.
>
> Thanks,
>
> C.
>
> From: Cedric Le Goater <clg@fr.ibm.com>
> Subject: add unshare_ns syscall core routine
>
> This patch adds the unshare_ns syscall core routine.
>
> This syscall is an unshare dedicated to namespaces.
>
> sample user program :
>
> /*
>  * unshare_ns.c
>  *
>  * author: Cedric Le Goater <clg@fr.ibm.com>
>  *
>  * (C) Copyright IBM Corp. 2005, 2006
>  *
>  * This program is free software; you can redistribute it and/or
>  * modify it under the terms of the GNU General Public License
>  * as published by the Free Software Foundation; either version 2
>  * of the License, or (at your option) any later version.
>  *
>  * This program is distributed in the hope that it will be useful,
>  * but WITHOUT ANY WARRANTY; without even the implied warranty of
>  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
>  * GNU General Public License for more details.
```

```

> *
> * You should have received a copy of the GNU General Public License
> * along with this program; if not, write to the Free Software
> * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
> * 02110-1301, USA.
> */
>
> #include <stdio.h>
> #include <stdlib.h>
> #include <sched.h>
> #include <unistd.h>
> #include <string.h>
> #include <errno.h>
>
> #include <linux/unistd.h>
>
> #ifndef __NR_unshare
> #if __i386__
> #   define __NR_unshare_ns 319
> #elif __x86_64__
> #   define __NR_unshare_ns 280
> #elif __ia64__
> #   define __NR_unshare_ns 1303
> #elif __s390x__
> #   define __NR_unshare_ns 310
> #elif __powerpc__
> #   define __NR_unshare_ns 301
> #else
> #   error "Architecture not supported"
> #endif
> #endif
>
> static inline _syscall1(int, unshare_ns, int, flags)
>
> #define UNSHARE_NS_MNT 0x00000001
> #define UNSHARE_NS_UTS 0x00000002
> #define UNSHARE_NS_IPC 0x00000004
> #define UNSHARE_NS_USER 0x00000008
> #define UNSHARE_NS_NET 0x00000010
> #define UNSHARE_NS_PID 0x00000020
>
> static void usage(const char *name)
> {
>     printf("usage: %s [-Hiunmpeh]\n", name);
>     printf("\t-H : unshare utsname namespace.\n");
>     printf("\t-i : unshare ipc namespace.\n");
>     printf("\t-u : unshare user namespace.\n");
>     printf("\t-n : unshare net namespace.\n");

```

```

> printf("\t-m : unshare mount namespace.\n");
> printf("\t-p : unshare pid namespace.\n");
> printf("\t-e : exec command.\n");
> printf("\n");
> printf("(C) Copyright IBM Corp. 2005, 2006\n");
> printf("\n");
> exit(1);
> }
>
> int main(int argc, char* argv[])
> {
>     int c;
>     unsigned long flag = 0;
>     int exec = 0;
>
>     while ((c = getopt(argc, argv, "+Hiunmpeh")) != EOF) {
>         switch (c) {
>             case 'i': flag |= UNSHARE_NS_IPC; break;
>             case 'u': flag |= UNSHARE_NS_USER; break;
>             case 'H': flag |= UNSHARE_NS_UTS; break;
>             case 'n': flag |= UNSHARE_NS_NET; break;
>             case 'm': flag |= UNSHARE_NS_MNT; break;
>             case 'p': flag |= UNSHARE_NS_PID; break;
>             case 'e': exec = 1; break;
>             case 'h':
>             default:
>                 usage(argv[0]);
>             }
>         };
>
>         argv = &argv[optind];
>         argc = argc - optind;
>
>         if (unshare_ns(flag) == -1) {
>             perror("unshare_ns");
>             return 1;
>         }
>
>         if (exec) {
>             execve(argv[0], argv, __environ);
>             fprintf(stderr, "execve(%s) : %s\n", argv[0], strerror(errno));
>             return 1;
>         }
>
>         return 0;
>     }
>
>
>

```

```

> Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
>
> ---
> include/linux/sched.h | 10 ++++
> include/linux/syscalls.h | 1
> kernel/fork.c | 106 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
> 3 files changed, 117 insertions(+)
>
> Index: 2.6.18-mm1/include/linux/sched.h
> =====
> --- 2.6.18-mm1.orig/include/linux/sched.h
> +++ 2.6.18-mm1/include/linux/sched.h
> @@ -28,6 +28,16 @@
> #define CLONE_NEWIPC 0x08000000 /* New ipc */
>
> /*
> + * unshare_ns flags:
> + */
> +#define UNSHARE_NS_MNT 0x00000001
> +#define UNSHARE_NS_UTS 0x00000002
> +#define UNSHARE_NS_IPC 0x00000004
> +#define UNSHARE_NS_USER 0x00000008
> +#define UNSHARE_NS_NET 0x00000010
> +#define UNSHARE_NS_PID 0x00000020
> +
> +/*
> * Scheduling policies
> */
> #define SCHED_NORMAL 0
> Index: 2.6.18-mm1/include/linux/syscalls.h
> =====
> --- 2.6.18-mm1.orig/include/linux/syscalls.h
> +++ 2.6.18-mm1/include/linux/syscalls.h
> @@ -580,6 +580,7 @@ asmlinkage long compat_sys_newfstatat(un
> asmlinkage long compat_sys_openat(unsigned int dfd, const char __user *filename,
> int flags, int mode);
> asmlinkage long sys_unshare(unsigned long unshare_flags);
> +asmlinkage long sys_unshare_ns(unsigned long unshare_flags);
>
> asmlinkage long sys_splice(int fd_in, loff_t __user *off_in,
> int fd_out, loff_t __user *off_out,
> Index: 2.6.18-mm1/kernel/fork.c
> =====
> --- 2.6.18-mm1.orig/kernel/fork.c
> +++ 2.6.18-mm1/kernel/fork.c
> @@ -1761,3 +1761,109 @@ bad_unshare_cleanup_thread:
> bad_unshare_out:
> return err;

```

```

> }
> +
> +/*
> + * unshare_ns allows a process to 'unshare' one or more of its
> + * namespaces which were originally shared using clone.
> + */
> +asmlinkage long sys_unshare_ns(unsigned long unshare_ns_flags)
> +{
> + int err = 0;
> + struct nsproxy *new_nsproxy = NULL, *old_nsproxy = NULL;
> + struct fs_struct *fs, *new_fs = NULL;
> + struct mnt_namespace *mnt, *new_mnt = NULL;
> + struct uts_namespace *uts, *new_uts = NULL;
> + struct ipc_namespace *ipc, *new_ipc = NULL;
> + unsigned long unshare_flags = 0;
> +
> + /* Return -EINVAL for all unsupported flags */
> + err = -EINVAL;
> + if (unshare_ns_flags & ~(UNSHARE_NS_MNT|UNSHARE_NS_UTS|UNSHARE_NS_IPC|
> +   UNSHARE_NS_USER|UNSHARE_NS_NET|
> +   UNSHARE_NS_PID))
> + goto bad_unshare_ns_out;
> +
> + /* convert unshare_ns flags to clone flags */
> + if (unshare_ns_flags & UNSHARE_NS_MNT)
> +   unshare_flags |= CLONE_NEWNS|CLONE_FS;
> + if (unshare_ns_flags & UNSHARE_NS_UTS)
> +   unshare_flags |= CLONE_NEWUTS;
> + if (unshare_ns_flags & UNSHARE_NS_IPC)
> +   unshare_flags |= CLONE_NEWIPC;
> +
> + if ((err = unshare_fs(unshare_flags, &new_fs)))
> + goto bad_unshare_ns_out;
> + if ((err = unshare_mnt_namespace(unshare_flags, &new_mnt, new_fs)))
> + goto bad_unshare_ns_cleanup_fs;
> + if ((err = unshare_utsname(unshare_flags, &new_uts)))
> + goto bad_unshare_ns_cleanup_mnt;
> + if ((err = unshare_ipcs(unshare_flags, &new_ipc)))
> + goto bad_unshare_ns_cleanup_uts;
> +
> + if (new_mnt || new_uts || new_ipc) {
> +   old_nsproxy = current->nsproxy;
> +   new_nsproxy = dup_namespaces(old_nsproxy);
> +   if (!new_nsproxy) {
> +     err = -ENOMEM;
> +     goto bad_unshare_ns_cleanup_ipc;
> +   }
> + }

```

```

> +
> + if (new_fs || new_mnt || new_uts || new_ipc) {
> +
> + task_lock(current);
> +
> + if (new_nsproxy) {
> +     current->nsproxy = new_nsproxy;
> +     new_nsproxy = old_nsproxy;
> + }
> +
> + if (new_fs) {
> +     fs = current->fs;
> +     current->fs = new_fs;
> +     new_fs = fs;
> + }
> +
> + if (new_mnt) {
> +     mnt = current->nsproxy->mnt_ns;
> +     current->nsproxy->mnt_ns = new_mnt;
> +     new_mnt = mnt;
> + }
> +
> + 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_ns_cleanup_ipc:
> + if (new_ipc)
> +     put_ipc_ns(new_ipc);
> +
> +bad_unshare_ns_cleanup_uts:
> + if (new_uts)
> +     put_uts_ns(new_uts);
> +

```

```
> +bad_unshare_ns_cleanup_mnt:
> + if (new_mnt)
> + put_mnt_ns(new_mnt);
> +
> +bad_unshare_ns_cleanup_fs:
> + if (new_fs)
> + put_fs_struct(new_fs);
> +
> +bad_unshare_ns_out:
> + return err;
> +}
>
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers
```

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

Subject: Re: namespace and nsproxy syscalls
Posted by [serue](#) on Tue, 26 Sep 2006 17:17:01 GMT
[View Forum Message](#) <> [Reply to Message](#)

Quoting Herbert Poetzl (herbert@13thfloor.at):
> On Tue, Sep 26, 2006 at 07:56:49AM -0500, Serge E. Hallyn wrote:
> > Quoting Cedric Le Goater (clg@fr.ibm.com):
> > > Hello all,
> > >
> > > A while ago, we expressed the need to have a new syscall specific to
> > > namespaces. the clone and unshare are good candidates but we are reaching
> > > the limit of the clone flags and clone has been hijacked enough.
> > >
> > > So, I came up with unshare_ns. the patch for the core feature follows
> > > the email. Not much difference with unshare() for the moment but it gives
> > > us the freedom to diverge when new namespaces come in. I have faith also !
> > > If you feel it's useful, i'll send the full patchset for review on the list.
> > >
> > > I'd like to discuss of another syscall which would allow a process to
> > > bind to a set of namespaces (== nsproxy == container) :
> > >
> > > bind_ns(ns_id_t id, int flags)
> > >
> > > What about just using a pid instead of introducing some ns_id_t? I'm
> > > guessing that any time you want to bind to some other nsproxy, it will
> > > be the nsproxy of a decendent nsproxy, so even if it is in a new
> > > pidspace, you will have a pid in your pidspace to reference it.

>
> what about lightweight containers where the process
> creating the namespace(s) goes away after starting
> a few scripts inside the guest?

So long as the scripts are running, those processes have a pid which could be used.

But I guess your concern is how the sysadmin can know which pids to use, since he might have only known the pid which started the container?

Dunno. Good question. Guess it might imply that either (a) we need namespace id's after all, or (b) we need to keep init processes around even for application containers.

> how to avoid having duplicate identifiers when there
> is a chance that the same pid will be used again
> to create a second namespace?

Well at least that's simple, the pid will no longer be a valid handle to the first namespace ever since that process died :)

-serge

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

Subject: Re: namespace and nsproxy syscalls
Posted by [Cedric Le Goater](#) on Tue, 26 Sep 2006 17:51:02 GMT
[View Forum Message](#) <> [Reply to Message](#)

Herbert Poetzl wrote:

> On Tue, Sep 26, 2006 at 11:42:10AM +0200, Cedric Le Goater wrote:
>> Hello all,
>>
>> A while ago, we expressed the need to have a new syscall specific to
>> namespaces. the clone and unshare are good candidates but we are reaching
>> the limit of the clone flags and clone has been hijacked enough.
>>
>> So, I came up with unshare_ns. the patch for the core feature follows
>> the email. Not much difference with unshare() for the moment but it gives
>> us the freedom to diverge when new namespaces come in. I have faith also !
>> If you feel it's useful, i'll send the full patchset for review on the list.
>>
>> I'd like to discuss of another syscall which would allow a process to
>> bind to a set of namespaces (== nsproxy == container) :

```

>>
>> bind_ns(ns_id_t id, int flags)
>>
>> bind_ns binds the current nsproxy to an id. You can only bind once and
>> you can use this id to bind another process to the same nsproxy.
>>
>> a few comments :
>>
>> * ns_id_t could be an int, a const char*, a struct ns_addr*. this is
>> to be defined.
>> * bind_ns applies to nsproxy and not to namespaces. we could bind a
>> specific namespace to an id using flags but i don't see the need.
>
> what if you have a setup like this:
>
> - guest using a full set of namespaces
> - host admin wants to 'adjust' a mount inside
>
> typically the host admin would enter the filesystem
> namespace (at least it is done so in Linux-VServer)
> but would avoid entering the other namespaces, like
> the pid-space or the container itself, just to do
> the mount without 'appearing' inside the guest and
> more important, with more priviledges than the guest
> processes would have ...

```

the flags would be useful for such scenarii.

a process unshares some namespaces, this operation creates a new nsproxy. the process binds (identifies) to someid with

```
bind_ns(someid, UNSHARE_NS_ALL)
```

The process does some other stuff, exec, fork, etc. and dies. but we keep the nsproxy bound to someid because some processes are still using it.

Now, i'm back in the 'host' and I want to fully bind to the identified nsproxy, I would use the same command

```
bind_ns(someid, UNSHARE_NS_ALL)
```

The above operation would simply attach the process to the nsproxy.

Now, if only want to bind to the filesystem namespace of this nsproxy, I would use :

```
bind_ns(someid, UNSHARE_NS_MNT)
```

The above operation creates a new nsproxy object and populates it with the identified nsproxy namespaces specified by flags (UNSHARE_NS_MNT) and keeps the current namespaces for the others. It's a reverse unshare but not exactly a clone().

It does some stuff again like mount a new fs and when the process dies the nsproxy gets recycled.

you use bind_ns() for 2 different purposes in 2 different contexts :

- * to identify a nsproxy and its set of namespaces in the context of the process (guest) which created the nsproxy.
- * to bind to some or all of a nsproxy namespaces in the context of another process (host)

[the UNSHARE_NS_ prefix is not well chosen. NS_ would be more appropriate i think]

- > IMHO that would require to setup two nsproxies
- > which _share_ a certain namespace, but how would
- > I do that? the only way I see here is to create some
- > kind of nsproxy hierarchy when the guest is started
- > like this:
- >
- > - create filesystem namespace
- > - do a bind_ns()
- > - create network namespace
- > - do another bind_ns()
- > - create uts/ipc/pid namespace
- > - do another bind_ns()
- > ...
- >
- > what if I only want to enter the network namespace
- > at some point?

I think the above sequence i've described covers your need, nop ?

- > IMHO some kind of unique (e.g. the virtual address
- > of the kernel structure) identifier for every
- > namespace would be a very good idea to have, in
- > combination with a method to 'build' a proxy on
- > the fly (e.g. by simply 'joining' the namespaces
- > you want to use) makes the above trivial and simple

what about :

unsigned long bind_ns(unsigned long id, int flags)

when identifying the nsproxy, use the virtual address of the kernel structure if id == 0 . returns the identifier.

> an alternative would be to have some 'copy partial'
> method to 'pick' certain namespaces from an existing
> nsproxy, but personally I would prefer to keep the
> nsproxy invisible and to work on namespaces only
> (IIRC, that was what we planned at the beginning of
> the nsproxy discussion)

Yes. I think we are reaching that now.

thanks,

C.

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

Subject: Re: namespace and nsproxy syscalls
Posted by [Herbert Poetzl](#) on Tue, 26 Sep 2006 22:09:48 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Tue, Sep 26, 2006 at 12:17:01PM -0500, Serge E. Hallyn wrote:
> Quoting Herbert Poetzl (herbert@13thfloor.at):
> > On Tue, Sep 26, 2006 at 07:56:49AM -0500, Serge E. Hallyn wrote:
> > > Quoting Cedric Le Goater (clg@fr.ibm.com):
> > > > Hello all,
> > > >
> > > > A while ago, we expressed the need to have a new syscall
> > > > specific to namespaces. the clone and unshare are good
> > > > candidates but we are reaching the limit of the clone flags and
> > > > clone has been hijacked enough.
> > > >
> > > > So, I came up with unshare_ns. the patch for the core feature
> > > > follows the email. Not much difference with unshare() for
> > > > the moment but it gives us the freedom to diverge when new
> > > > namespaces come in. I have faith also ! If you feel it's useful,
> > > > i'll send the full patchset for review on the list.
> > > >
> > > > I'd like to discuss of another syscall which would allow
> > > > a process to bind to a set of namespaces (== nsproxy ==
> > > > container) :
> > > >
> > > > bind_ns(ns_id_t id, int flags)
> > >

> > > What about just using a pid instead of introducing some ns_id_t?
> > > I'm guessing that any time you want to bind to some other nsproxy,
> > > it will be the nsproxy of a decendent nsproxy, so even if it is in
> > > a new pidspace, you will have a pid in your pidspace to reference
> > > it.
> >
> > what about lightweight containers where the process
> > creating the namespace(s) goes away after starting
> > a few scripts inside the guest?
>
> So long as the scripts are running, those processes have a pid which
> could be used.
>
> But I guess your concern is how the sysadmin can know which pids to use,
> since he might have only known the pid which started the container?

not only, just consider a lightweight guest which
does nothing more but 'running' /etc/rc to start
services. quite naturally this script is not running
very long (a few seconds usually) but you might want
to enter the guest namespace at a later time too :)

> Dunno. Good question. Guess it might imply that either (a) we need
> namespace id's after all, or (b) we need to keep init processes around
> even for application conatiners.

that's just a waste of resources ... IMHO it is
a little weird to actually consider having an init
process 'just' to have a reference for a bunch of
namespaces, given that you might want to access
them individually, am I missing something?

for me this suggestion sounds like making a dog
mandatory for each household, so that when you
want to get the younger son on the phone you
can refer to him as 'the younger son of the family
with the dog charly' :) ...

> > how to avoid having duplicate identifiers when there
> > is a chance that the same pid will be used again
> > to create a second namespace?
>

> Well at least that's simple, the pid will no longer be a valid handle to
> the first namespace ever since that process died :)

which simply makes it inaccessible which is not
what you actually want, sorry ...

best,
Herbert

> -serge

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

Subject: Re: namespace and nsproxy syscalls
Posted by [serue](#) on Tue, 03 Oct 2006 14:43:57 GMT
[View Forum Message](#) <> [Reply to Message](#)

Quoting Herbert Poetzl (herbert@13thfloor.at):

> > > how to avoid having duplicate identifiers when there
> > > is a chance that the same pid will be used again
> > > to create a second namespace?

> >

> > Well at least that's simple, the pid will no longer be a valid handle to
> > the first namespace ever since that process died :)

>

> which simply makes it inaccessible which is not
> what you actually want, sorry ...

Nonsense. It is still accessible via any other pids for processes in that namespace. (i.e. if you're in pidns 1, and (pidns 2, pid 1) has started (pidns 2, pid 2) and then exited, then (pidns 2, pid 2) will also be known by some (pidns 1, pid X), so you can access the namespace via pid X from your pidns 1 process.

How to actually find a pid that will last long enough for you to find it and then access it is an exercise left to the reader :)

In other words, I was saying that the duplicate identifiers is not a bug, but I thought I had left it clearly implied that the approach not practical, and we will need namespace ids.

-serge

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