
Subject: [PATCH 0/8] user namespace: Introduction
Posted by [serue](#) on Tue, 19 Dec 2006 22:59:02 GMT
[View Forum Message](#) <> [Reply to Message](#)

This patchset adds a user namespace, which allows a process to unshare it's user_struct table, allowing for separate accounting per user namespace. It appends a user namespace to vfsmounts and fown_structs, so that uid1==uid2 checks can be extended to be false if uid1 and uid2 are in different namespaces.

A vfsmount generally cannot be accessed by another user namespace than that in which it was mounted. A vfsmount can be mounted "shared-ns", in which case it can be accessed by any user namespace. This is needed at least to bootstrap a container so it can get far enough to create it's own private file system tree, and can be used in conjunction with read-only bind mounts to provide shared /usr trees, for instance. However, for more useful, more fine-grained sharing accross user namespaces, it has been suggested that a new filesystem specifying global userid's be used.

Patches are as follows:

1. make exit_task_namespaces extern to prevent future compile failure.
2. add userns framework.
3. add userns pointer to vfsmount
4. hook permission to check current against vfsmount userns
5. prepare for copy_mnt and copy_tree to return -EPERM
6. implement shared mounts which can cross user namespaces
7. implement user ns checks for file sigio
8. implement user namespace unshare

```
#include <unistd.h>
#include <linux/unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <sys/mount.h>
#include <sys/syscall.h>
#include <errno.h>
#include <signal.h>
```

```
#ifndef MNT_DETACH
#define MNT_DETACH 2
#endif
```

```
#ifndef MS_SHARE_NS
```

```

#define MS_SHARE_NS 1<<22
#endif

#ifndef CLONE_NEWUSER
#define CLONE_NEWUSER 0x10000000
#endif

#define PIPEREAD 0
#define PIPEWRITE 1

static inline __syscall2(int, clone, int, flags, int, foo)
int to1[2], from1[1], to2[2], from2[2];

void do_test1(void)
{
    int ret;
    int fd;
    char buf[100];

    rmdir("/mnt1");
    ret = mkdir("/mnt1", 0666);
    if (ret == -1) {
        perror("mkdir mnt1");
        _exit(1);
    }
    ret = mount("/", "/mnt1", "none", MS_BIND, "");
    if (ret == -1) {
        perror("mount mnt1");
        _exit(1);
    }
    fd = open("/mnt1/testme", O_RDWR|O_CREAT);
    if (fd == -1) {
        printf("thread 1: unable to write /testme: ERROR\n");
    } else {
        write(fd, "a", 1);
        printf("thread 1: able to write /testme: GOOD\n");
        close(fd);
    }
    write(from1[PIPEWRITE], "mount", 6);

    read(to1[PIPEREAD], buf, 2);
    umount("/mnt1");
    printf("thread 1: exiting.\n");
    write(from1[PIPEWRITE], "done", 5);
    rmdir("/mnt1");
}

void do_test2(void)

```

```

{
int ret;
char buf[100];
int fd;

rmdir("/mnt2");
ret = mkdir("/mnt2", 0666);
if (ret == -1) {
    perror("mkdir mnt2");
    _exit(1);
}
ret = read(to2[PIPEREAD], buf, 10);
fd = open("/testme", O_RDWR|O_CREAT);
if (fd == -1) {
    printf("thread 2: unable to write /testme: ERROR\n");
} else {
    write(fd, "b", 1);
    printf("thread 2: able to write /testme: GOOD\n");
    close(fd);
}

fd = open("/mnt1/testme", O_RDWR|O_CREAT);
if (fd == -1) {
    printf("thread 2: unable to open /mnt1/testme: GOOD\n");
} else {
    write(fd, "c", 1);
    printf("thread 2: able to open /mnt1/testme: ERROR\n");
    close(fd);
}

/* now try remounting /mnt1 to /mnt2 */
ret = mount("/mnt1", "/mnt2", "none", MS_BIND, "");
if (ret == -1) {
    printf("thread2: unable to remount /mnt1 to /mnt2: GOOD\n");
} else {
    perror("thread2: able to remount /mnt1 to /mnt2: BAD\n");
    fd = open("/mnt2/testme", O_RDWR|O_CREAT);
    if (fd == -1) {
        printf("thread 2: unable to open /mnt2/testme: GOOD\n");
    } else {
        write(fd, "d", 1);
        printf("thread 2: able to open /mnt2/testme: ERROR\n");
        close(fd);
    }
    umount ("/mnt2");
}

write(from2[PIPEWRITE], "done", 5);

```

```

printf("thread 2: exiting\n");
rmdir("/mnt2");
}

int main(int argc, char *argv[])
{
    int childpid1, childpid2;
    int ret;
    char buf[100];

    ret = mount("/", "/mnt", "none", MS_SHARE_NS | MS_BIND, "");
    if (ret == -1) {
        perror("failed to mount /mnt shared_ns.\n");
        _exit(1);
    }

    chdir("/mnt");
    chroot("/mnt");

    ret = pipe(to1);
    if (ret == -1) {
        perror("failed to create child pipe 1\n");
        _exit(1);
    }
    ret = pipe(to2);
    if (ret == -1) {
        perror("failed to create child pipe 2\n");
        _exit(1);
    }
    ret = pipe(from1);
    if (ret == -1) {
        perror("failed to create child pipe 1\n");
        _exit(1);
    }
    ret = pipe(from2);
    if (ret == -1) {
        perror("failed to create child pipe 2\n");
        _exit(1);
    }

    childpid1 = clone(CLONE_NEWUSER | SIGCHLD, 0);
    if (childpid1 != 0) {
        do_test1();
        _exit(0);
    }
    childpid2 = clone(CLONE_NEWUSER | SIGCHLD, 0);
    if (childpid2 != 0) {
        do_test2();
    }
}

```

```
_exit(0);  
}  
  
read(from1[PIPEREAD], buf, 5);  
write(to2[PIPEWRITE], "go", 3);  
read(from2[PIPEREAD], buf, 4);  
write(to1[PIPEWRITE], "go", 3);  
read(from1[PIPEREAD], buf, 4);  
return 0;  
}
```

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