
Subject: [PATCH] namespaces: introduce sys_hijack (v7)

Posted by [serue](#) on Wed, 24 Oct 2007 22:21:25 GMT

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

This is just a first stab at doing hijack by cgroup files. I force using the 'tasks' file just so that I can (a) predict and check the name of the file, (b) make sure it's a cgroup file, and then (c) trust that taking `__d_cont(dentry parent)` gives me a legitimate container.

Seems to work at least.

Paul, does this look reasonable? `task_from_cgroup_fd()` in particular.

thanks,
-serge

>From 8ba6a4ae33da18b95d967a799c5edce4266d1f1f Mon Sep 17 00:00:00 2001
From: sergeh@us.ibm.com <sergeh@us.ibm.com>
Date: Tue, 16 Oct 2007 09:36:49 -0700
Subject: [PATCH] namespaces: introduce sys_hijack (v7)

Move most of `do_fork()` into a new `do_fork_task()` which acts on a new argument, `task`, rather than on `current`. `do_fork()` becomes a call to `do_fork_task(current, ...)`.

Introduce `sys_hijack` (for x86 only so far). It is like `clone`, but in place of a stack pointer (which is assumed null) it accepts a pid. The process identified by that pid is the one which is actually cloned. Some state - include the file table, the signals and `sighand` (and hence `tty`), and the `->parent` are taken from the calling process.

A process to be hijacked may be identified by process id. Alternatively, an open fd for a cgroup 'tasks' file may be specified. The first available task in that cgroup will then be hijacked.

In order to hijack a process, the calling process must be allowed to `ptrace` the target.

The effect is a sort of namespace enter. The following program uses `sys_hijack` to 'enter' all namespaces of the specified task. For instance in one terminal, do

```
mount -t cgroup -ons /cgroup
hostname
qemu
```

```
ns_exec -u /bin/sh
hostname serge
echo $$
1073
cat /proc/$$/cgroup
ns:/node_1073
```

In another terminal then do

```
hostname
qemu
cat /proc/$$/cgroup
ns:/
hijack pid 1073
hostname
serge
cat /proc/$$/cgroup
ns:/node_1073
hijack cgroup /cgroup/node_1073/tasks
```

Changelog:

Aug 23: send a stop signal to the hijacked process (like ptrace does).
Oct 09: Update for 2.6.23-rc8-mm2 (mainly pidns)
Don't take task_lock under rcu_read_lock
Send hijacked process to cgroup_fork() as the first argument.
Removed some unneeded task_locks.
Oct 16: Fix bug introduced into alloc_pid.
Oct 16: Add 'int which' argument to sys_hijack to allow later expansion to use cgroup in place of pid to specify what to hijack.
Oct 24: Implement hijack by open cgroup file.

```
=====
hijack.c
=====
#define _BSD_SOURCE
#include <unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sched.h>

void usage(char *me)
{
```

```

printf("Usage: %s pid <pid> | %s cgroup <cgroup_tasks_file>\n", me, me);
}

int exec_shell(void)
{
    execl("/bin/sh", "/bin/sh", NULL);
}

int main(int argc, char *argv[])
{
    int id;
    int ret;
    int status;
    int use_pid = 0;

    if (argc < 3 || !strcmp(argv[1], "-h")) {
        usage(argv[0]);
        return 1;
    }
    if (strcmp(argv[1], "pid") == 0)
        use_pid = 1;

    if (use_pid)
        id = atoi(argv[2]);
    else {
        id = open(argv[2], O_RDONLY);
        if (id == -1) {
            perror("cgroup open");
            return 1;
        }
    }

    ret = syscall(327, SIGCHLD, use_pid ? 1 : 2, (unsigned long)id);

    if (use_pid)
        close(id);
    if (ret == 0) {
        return exec_shell();
    } else if (ret < 0) {
        perror("sys_hijack");
    } else {
        printf("waiting on cloned process %d\n", ret);
        // ret = waitpid(ret, &status, __WALL);
        while(waitpid(-1, &status, __WALL) != -1)
            ;
        printf("cloned process exited with %d (waitpid ret %d)\n",
            status, ret);
    }
}

```

```
return ret;
}
```

=====
Signed-off-by: Serge Hallyn <serue@us.ibm.com>

```
arch/i386/kernel/process.c | 87 ++++++-----
arch/i386/kernel/syscall_table.S | 1 +
arch/s390/kernel/process.c | 12 +++++-
include/asm-i386/unistd.h | 3 +-
include/linux/cgroup.h | 10 +++++-
include/linux/ptrace.h | 1 +
include/linux/sched.h | 8 +++++
include/linux/syscalls.h | 1 +
kernel/cgroup.c | 69 ++++++-----
kernel/fork.c | 67 ++++++-----
kernel/ptrace.c | 7 +++
11 files changed, 237 insertions(+), 29 deletions(-)
```

```
diff --git a/arch/i386/kernel/process.c b/arch/i386/kernel/process.c
```

```
index eab6c62..4b631b9 100644
```

```
--- a/arch/i386/kernel/process.c
```

```
+++ b/arch/i386/kernel/process.c
```

```
@@ -39,6 +39,7 @@
```

```
#include <linux/personality.h>
```

```
#include <linux/tick.h>
```

```
#include <linux/percpu.h>
```

```
+#include <linux/cgroup.h>
```

```
#include <asm/uaccess.h>
```

```
#include <asm/pgtable.h>
```

```
@@ -455,8 +456,15 @@ int copy_thread(int nr, unsigned long clone_flags, unsigned long esp,
```

```
unsigned long unused,
struct task_struct * p, struct pt_regs * regs)
```

```
{
```

```
+ return copy_a_thread(current, nr, clone_flags, esp, unused,
+ p, regs);
```

```
+}
```

```
+
```

```
+int copy_a_thread(struct task_struct *tsk, int nr, unsigned long clone_flags,
```

```
+ unsigned long esp, unsigned long unused,
```

```
+ struct task_struct * p, struct pt_regs * regs)
```

```
+{
```

```
struct pt_regs * childregs;
```

```
- struct task_struct *tsk;
```

```
int err;
```

```

childregs = task_pt_regs(p);
@@ -471,7 +479,6 @@ int copy_thread(int nr, unsigned long clone_flags, unsigned long esp,

savesegment(gs,p->thread.gs);

- tsk = current;
if (unlikely(test_tsk_thread_flag(tsk, TIF_IO_BITMAP))) {
    p->thread.io_bitmap_ptr = kmemdup(tsk->thread.io_bitmap_ptr,
        IO_BITMAP_BYTES, GFP_KERNEL);
@@ -784,6 +791,82 @@ asmlinkage int sys_clone(struct pt_regs regs)
}

/*
+ * Called with task count bumped, drops task count before returning
+ */
+static int hijack_task(struct task_struct *task, struct pt_regs regs)
+{
+ unsigned long clone_flags = regs.ebx;
+ int ret = -EINVAL;
+
+ task_lock(task);
+ put_task_struct(task);
+ if (!ptrace_may_attach_locked(task)) {
+ ret = -EPERM;
+ goto out_put_task;
+ }
+ if (task->ptrace) {
+ ret = -EBUSY;
+ goto out_put_task;
+ }
+ force_sig_specific(SIGSTOP, task);
+
+ task_unlock(task);
+ ret = do_fork_task(task, clone_flags, regs.esp, &regs, 0,
+ NULL, NULL);
+ wake_up_process(task);
+ return ret;
+
+out_put_task:
+ task_unlock(task);
+ return ret;
+}
+
+static int hijack_pid(struct pt_regs regs)
+{
+ pid_t pid = regs.edx;
+ struct task_struct *task;
+

```

```

+ rcu_read_lock();
+ task = find_task_by_vpid(pid);
+ if (task)
+ get_task_struct(task);
+ rcu_read_unlock();
+
+ if (!task)
+ return -EINVAL;
+
+ return hijack_task(task, regs);
+}
+
+static int hijack_cgroup(struct pt_regs regs)
+{
+ unsigned int fd;
+ struct task_struct *task;
+
+ fd = (unsigned int) regs.edx;
+ task = task_from_cgroup_fd(fd);
+ if (IS_ERR(task))
+ return PTR_ERR(task);
+
+ return hijack_task(task, regs);
+}
+
+asmlinkage int sys_hijack(struct pt_regs regs)
+{
+ int which = regs.ecx;
+
+ switch (which) {
+ case HIJACK_PID:
+ return hijack_pid(regs);
+ case HIJACK_CGROUP:
+ return hijack_cgroup(regs);
+ default:
+ return -EINVAL;
+ }
+
+}
+
+/*
+ * This is trivial, and on the face of it looks like it
+ * could equally well be done in user mode.
+ */
diff --git a/arch/i386/kernel/syscall_table.S b/arch/i386/kernel/syscall_table.S
index df6e41e..495930c 100644
--- a/arch/i386/kernel/syscall_table.S
+++ b/arch/i386/kernel/syscall_table.S

```

```

@@ -326,3 +326,4 @@ ENTRY(sys_call_table)
    .long sys_fallocate
    .long sys_revokeat /* 325 */
    .long sys_frevoke
+ .long sys_hijack
diff --git a/arch/s390/kernel/process.c b/arch/s390/kernel/process.c
index 70c5737..f256e7a 100644
--- a/arch/s390/kernel/process.c
+++ b/arch/s390/kernel/process.c
@@ -223,6 +223,14 @@ int copy_thread(int nr, unsigned long clone_flags, unsigned long
new_stackp,
    unsigned long unused,
    struct task_struct * p, struct pt_regs * regs)
{
+ return copy_a_thread(current, nr, clone_flags, new_stackp, unused,
+   p, regs);
+}
+
+int copy_a_thread(struct task_struct *task, int nr, unsigned long clone_flags,
+ unsigned long new_stackp, unsigned long unused,
+   struct task_struct * p, struct pt_regs * regs)
+{
    struct fake_frame
    {
        struct stack_frame sf;
@@ -251,8 +251,8 @@ int copy_thread(int nr, unsigned long clone_flags, unsigned long
new_stackp,
    * save fprs to current->thread.fp_regs to merge them with
    * the emulated registers and then copy the result to the child.
    */
- save_fp_regs(&current->thread.fp_regs);
- memcpy(&p->thread.fp_regs, &current->thread.fp_regs,
+ save_fp_regs(&task->thread.fp_regs);
+ memcpy(&p->thread.fp_regs, &task->thread.fp_regs,
    sizeof(s390_fp_regs));
    p->thread.user_seg = __pa((unsigned long) p->mm->pgd) | _SEGMENT_TABLE;
    /* Set a new TLS ? */
diff --git a/include/asm-i386/unistd.h b/include/asm-i386/unistd.h
index 006c1b3..fe6eeb4 100644
--- a/include/asm-i386/unistd.h
+++ b/include/asm-i386/unistd.h
@@ -332,10 +332,11 @@
#define __NR_fallocate 324
#define __NR_revokeat 325
#define __NR_frevoke 326
+#define __NR_hijack 327

#ifdef __KERNEL__

```

```

-#define NR_syscalls 327
+#define NR_syscalls 328

#define __ARCH_WANT_IPC_PARSE_VERSION
#define __ARCH_WANT_OLD_READDIR
diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
index 8747932..3edb820 100644
--- a/include/linux/cgroup.h
+++ b/include/linux/cgroup.h
@@ -26,7 +26,7 @@ extern int cgroup_init(void);
extern void cgroup_init_smp(void);
extern void cgroup_lock(void);
extern void cgroup_unlock(void);
-extern void cgroup_fork(struct task_struct *p);
+extern void cgroup_fork(struct task_struct *parent, struct task_struct *p);
extern void cgroup_fork_callbacks(struct task_struct *p);
extern void cgroup_post_fork(struct task_struct *p);
extern void cgroup_exit(struct task_struct *p, int run_callbacks);
@@ -304,12 +304,14 @@ struct task_struct *cgroup_iter_next(struct cgroup *cont,
    struct cgroup_iter *it);
void cgroup_iter_end(struct cgroup *cont, struct cgroup_iter *it);

+struct task_struct *task_from_cgroup_fd(unsigned int fd);
#else /* !CONFIG_CGROUPS */

static inline int cgroup_init_early(void) { return 0; }
static inline int cgroup_init(void) { return 0; }
static inline void cgroup_init_smp(void) {}
-static inline void cgroup_fork(struct task_struct *p) {}
+static inline void cgroup_fork(struct task_struct *parent,
+    struct task_struct *p) {}
static inline void cgroup_fork_callbacks(struct task_struct *p) {}
static inline void cgroup_post_fork(struct task_struct *p) {}
static inline void cgroup_exit(struct task_struct *p, int callbacks) {}
@@ -322,6 +324,10 @@ static inline int cgroupstats_build(struct cgroupstats *stats,
    return -EINVAL;
}

+static inline struct task_struct *task_from_cgroup_fd(unsigned int fd)
+{
+ return ERR_PTR(-EINVAL);
+}
#endif /* !CONFIG_CGROUPS */

#endif /* _LINUX_CGROUP_H */
diff --git a/include/linux/ptrace.h b/include/linux/ptrace.h
index ae8146a..727a4a9 100644

```



```

--- a/include/linux/ptrace.h
+++ b/include/linux/ptrace.h
@@ -97,6 +97,7 @@ extern void __ptrace_link(struct task_struct *child,
extern void __ptrace_unlink(struct task_struct *child);
extern void ptrace_untrace(struct task_struct *child);
extern int ptrace_may_attach(struct task_struct *task);
+extern int ptrace_may_attach_locked(struct task_struct *task);

static inline void ptrace_link(struct task_struct *child,
                               struct task_struct *new_parent)
diff --git a/include/linux/sched.h b/include/linux/sched.h
index 3436b98..6478cbb 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -29,6 +29,12 @@
#define CLONE_NEWNET 0x40000000 /* New network namespace */

/*
+ * Hijack flags
+ */
+#define HIJACK_PID 1 /* 'id' is a pid */
+#define HIJACK_CGROUP 2 /* 'id' is an open fd for a cgroup dir */
+
+/*
 * Scheduling policies
 */
#define SCHED_NORMAL 0
@@ -1643,6 +1649,7 @@ extern struct mm_struct *get_task_mm(struct task_struct *task);
extern void mm_release(struct task_struct *, struct mm_struct *);

extern int copy_thread(int, unsigned long, unsigned long, unsigned long, struct task_struct *,
struct pt_regs *);
+extern int copy_a_thread(struct task_struct *, int, unsigned long, unsigned long, unsigned long,
struct task_struct *, struct pt_regs *);
extern void flush_thread(void);
extern void exit_thread(void);

@@ -1658,6 +1665,7 @@ extern int allow_signal(int);
extern int disallow_signal(int);

extern int do_execve(char *, char __user * __user *, char __user * __user *, struct pt_regs *);
+extern long do_fork_task(struct task_struct *task, unsigned long, unsigned long, struct pt_regs *,
unsigned long, int __user *, int __user *);
extern long do_fork(unsigned long, unsigned long, struct pt_regs *, unsigned long, int __user *,
int __user *);
struct task_struct *fork_idle(int);

diff --git a/include/linux/syscalls.h b/include/linux/syscalls.h

```

```

index f696874..31f0c79 100644
--- a/include/linux/syscalls.h
+++ b/include/linux/syscalls.h
@@ -616,5 +616,6 @@ int kernel_execve(const char *filename, char *const argv[], char *const
envp[]);

asmlinkage long sys_revokeat(int dfd, const char __user *filename);
asmlinkage long sys_frevoked(unsigned int fd);
+asmlinkage long sys_hijack(unsigned long flags, int which, unsigned long id);

#endif
diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index 8720881..2d1d67e 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -45,6 +45,7 @@
#include <linux/sort.h>
#include <linux/delayacct.h>
#include <linux/cgroupstats.h>
+#include <linux/file.h>

#include <asm/atomic.h>

@@ -2460,12 +2461,12 @@ static struct file_operations proc_cgroupstats_operations = {
 * At the point that cgroup_fork() is called, 'current' is the parent
 * task, and the passed argument 'child' points to the child task.
 */
-void cgroup_fork(struct task_struct *child)
+void cgroup_fork(struct task_struct *parent, struct task_struct *child)
{
- task_lock(current);
- child->cgroups = current->cgroups;
+ task_lock(parent);
+ child->cgroups = parent->cgroups;
  get_css_set(child->cgroups);
- task_unlock(current);
+ task_unlock(parent);
  INIT_LIST_HEAD(&child->cg_list);
}

@@ -2819,3 +2820,63 @@ static void cgroup_release_agent(struct work_struct *work)
  spin_unlock(&release_list_lock);
  mutex_unlock(&cgroup_mutex);
}
+
+static inline int task_available(struct task_struct *task)
+{
+ switch(task->state) {

```

```

+ case TASK_RUNNING:
+ case TASK_INTERRUPTIBLE:
+ return 1;
+ default:
+ return 0;
+ }
+}
+
+/*
+ * Takes an integer which is a open fd in current for a valid
+ * cgroupfs file. Returns a task in that cgroup, with its
+ * refcount bumped.
+ */
+struct task_struct *task_from_cgroup_fd(unsigned int fd)
+{
+ struct file *file;
+ struct cgroup *cgroup;
+ struct cgroup_iter it;
+ struct task_struct *task = NULL;
+
+ file = fget(fd);
+ if (!file)
+ return ERR_PTR(-EINVAL);
+
+ if (!file->f_dentry || !file->f_dentry->d_sb)
+ goto out_fput;
+ if (file->f_dentry->d_parent->d_sb->s_magic != CGROUP_SUPER_MAGIC)
+ goto out_fput;
+ if (strcmp(file->f_dentry->d_name.name, "tasks"))
+ goto out_fput;
+
+ rcu_read_lock();
+ cgroup = __d_cont(file->f_dentry->d_parent);
+ printk(KERN_NOTICE "cgroup is %lu\n", (unsigned long)cgroup);
+ if (!cgroup)
+ goto out_unlock;
+ cgroup_iter_start(cgroup, &it);
+ do {
+ task = cgroup_iter_next(cgroup, &it);
+ if (task)
+ printk(KERN_NOTICE "task %d state %d\n",
+ task->pid, task->state);
+ } while (task && !task_available(task));
+ cgroup_iter_end(cgroup, &it);
+ printk(KERN_NOTICE "task is %lu\n", (unsigned long)task);
+ if (task)
+ get_task_struct(task);
+
+

```

```

+out_unlock:
+ rcu_read_unlock();
+out_fput:
+ fput(file);
+ if (!task)
+ return ERR_PTR(-EINVAL);
+ return task;
+}
diff --git a/kernel/fork.c b/kernel/fork.c
index 64860ef..20b4902 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -621,13 +621,14 @@ struct fs_struct *copy_fs_struct(struct fs_struct *old)

```

```
EXPORT_SYMBOL_GPL(copy_fs_struct);
```

```
-static int copy_fs(unsigned long clone_flags, struct task_struct *tsk)
```

```
+static inline int copy_fs(unsigned long clone_flags,
```

```
+ struct task_struct * src, struct task_struct * tsk)
```

```
{
```

```
if (clone_flags & CLONE_FS) {
```

```
- atomic_inc(&current->fs->count);
```

```
+ atomic_inc(&src->fs->count);
```

```
return 0;
```

```
}
```

```
- tsk->fs = __copy_fs_struct(current->fs);
```

```
+ tsk->fs = __copy_fs_struct(src->fs);
```

```
if (!tsk->fs)
```

```
return -ENOMEM;
```

```
return 0;
```

```
@@ -973,7 +974,8 @@ static void rt_mutex_init_task(struct task_struct *p)
```

```
* parts of the process environment (as per the clone
```

```
* flags). The actual kick-off is left to the caller.
```

```
*/
```

```
-static struct task_struct *copy_process(unsigned long clone_flags,
```

```
+static struct task_struct *copy_process(struct task_struct *task,
```

```
+ unsigned long clone_flags,
```

```
unsigned long stack_start,
```

```
struct pt_regs *regs,
```

```
unsigned long stack_size,
```

```
@@ -1007,15 +1009,17 @@ static struct task_struct *copy_process(unsigned long clone_flags,
```

```
goto fork_out;
```

```
retval = -ENOMEM;
```

```
- p = dup_task_struct(current);
```

```
+ p = dup_task_struct(task);
```

```
if (!p)
```

```
goto fork_out;
```

```

rt_mutex_init_task(p);

#ifdef CONFIG_TRACE_IRQFLAGS
- DEBUG_LOCKS_WARN_ON(!p->hardirqs_enabled);
- DEBUG_LOCKS_WARN_ON(!p->softirqs_enabled);
+ if (task == current) {
+ DEBUG_LOCKS_WARN_ON(!p->hardirqs_enabled);
+ DEBUG_LOCKS_WARN_ON(!p->softirqs_enabled);
+ }
#endif
retval = -EAGAIN;
if (atomic_read(&p->user->processes) >=
@@ -1084,7 +1088,7 @@ static struct task_struct *copy_process(unsigned long clone_flags,
#endif
p->io_context = NULL;
p->audit_context = NULL;
- cgroup_fork(p);
+ cgroup_fork(task, p);
#ifdef CONFIG_NUMA
p->mempolicy = mpol_copy(p->mempolicy);
if (IS_ERR(p->mempolicy)) {
@@ -1132,7 +1136,7 @@ static struct task_struct *copy_process(unsigned long clone_flags,
goto bad_fork_cleanup_audit;
if ((retval = copy_files(clone_flags, p)))
goto bad_fork_cleanup_semundo;
- if ((retval = copy_fs(clone_flags, p)))
+ if ((retval = copy_fs(clone_flags, task, p)))
goto bad_fork_cleanup_files;
if ((retval = copy_sighand(clone_flags, p)))
goto bad_fork_cleanup_fs;
@@ -1144,7 +1148,7 @@ static struct task_struct *copy_process(unsigned long clone_flags,
goto bad_fork_cleanup_mm;
if ((retval = copy_namespaces(clone_flags, p)))
goto bad_fork_cleanup_keys;
- retval = copy_thread(0, clone_flags, stack_start, stack_size, p, regs);
+ retval = copy_a_thread(task, 0, clone_flags, stack_start, stack_size, p, regs);
if (retval)
goto bad_fork_cleanup_namespaces;

@@ -1164,7 +1168,7 @@ static struct task_struct *copy_process(unsigned long clone_flags,
p->pid = pid_nr(pid);
p->tgid = p->pid;
if (clone_flags & CLONE_THREAD)
- p->tgid = current->tgid;
+ p->tgid = task->tgid;

p->set_child_tid = (clone_flags & CLONE_CHILD_SETTID) ? child_tidptr : NULL;

```

```

/*
@@ -1379,7 +1383,7 @@ struct task_struct * __cpuinit fork_idle(int cpu)
    struct task_struct *task;
    struct pt_regs regs;

- task = copy_process(CLONE_VM, 0, idle_regs(&regs), 0, NULL,
+ task = copy_process(current, CLONE_VM, 0, idle_regs(&regs), 0, NULL,
    &init_struct_pid);
    if (!IS_ERR(task))
        init_idle(task, cpu);
@@ -1404,12 +1408,12 @@ static int fork_traceflag(unsigned clone_flags)
}

/*
- * Ok, this is the main fork-routine.
- *
- * It copies the process, and if successful kick-starts
- * it and waits for it to finish using the VM if required.
+ * if called with task!=current, then caller must ensure that
+ * 1. it has a reference to task
+ * 2. current must have ptrace permission to task
*/
-long do_fork(unsigned long clone_flags,
+long do_fork_task(struct task_struct *task,
+ unsigned long clone_flags,
    unsigned long stack_start,
    struct pt_regs *regs,
    unsigned long stack_size,
@@ -1420,13 +1424,23 @@ long do_fork(unsigned long clone_flags,
    int trace = 0;
    long nr;

+ if (task != current) {
+ /* sanity checks */
+ /* we only want to allow hijacking the simplest cases */
+ if (clone_flags & CLONE_SYSVSEM)
+ return -EINVAL;
+ if (current->ptrace)
+ return -EPERM;
+ if (task->ptrace)
+ return -EINVAL;
+ }
    if (unlikely(current->ptrace)) {
        trace = fork_traceflag (clone_flags);
        if (trace)
            clone_flags |= CLONE_PTRACE;
    }
}

```

```

- p = copy_process(clone_flags, stack_start, regs, stack_size,
+ p = copy_process(task, clone_flags, stack_start, regs, stack_size,
  child_tidptr, NULL);
/*
 * Do this prior waking up the new thread - the thread pointer
@@ -1481,6 +1495,23 @@ long do_fork(unsigned long clone_flags,
  return nr;
}

+/*
+ * Ok, this is the main fork-routine.
+ *
+ * It copies the process, and if successful kick-starts
+ * it and waits for it to finish using the VM if required.
+ */
+long do_fork(unsigned long clone_flags,
+  unsigned long stack_start,
+  struct pt_regs *regs,
+  unsigned long stack_size,
+  int __user *parent_tidptr,
+  int __user *child_tidptr)
+{
+ return do_fork_task(current, clone_flags, stack_start,
+ regs, stack_size, parent_tidptr, child_tidptr);
+}
+
+ #ifndef ARCH_MIN_MMSTRUCT_ALIGN
+ #define ARCH_MIN_MMSTRUCT_ALIGN 0
+ #endif
diff --git a/kernel/ptrace.c b/kernel/ptrace.c
index 7c76f2f..c65c9fe 100644
--- a/kernel/ptrace.c
+++ b/kernel/ptrace.c
@@ -159,6 +159,13 @@ int ptrace_may_attach(struct task_struct *task)
  return !err;
}

+int ptrace_may_attach_locked(struct task_struct *task)
+{
+ int err;
+ err = may_attach(task);
+ return !err;
+}
+
+ int ptrace_attach(struct task_struct *task)
+ {
+ int retval;
--

```

1.5.1

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>
