Subject: [PATCH v8 1/3] cgroups: read-write lock CLONE_THREAD forking per threadgroup
Posted by Ben Blum on Tue, 08 Feb 2011 01:37:41 GMT
View Forum Message <> Reply to Message

Adds functionality to read/write lock CLONE_THREAD fork()ing per-threadgroup

From: Ben Blum <bblum@andrew.cmu.edu>

This patch adds an rwsem that lives in a threadgroup's signal_struct that's
taken for reading in the fork path, under CONFIG_CGROUPS. If another part of
the kernel later wants to use such a locking mechanism, the CONFIG_CGROUPS
ifdefs should be changed to a higher-up flag that CGROUPS and the other system
would both depend on.

This is a pre-patch for cgroup-procs-write.patch.

Signed-off-by: Ben Blum <bblum@andrew.cmu.edu>
---
 include/linux/init_task.h | 9 +++++++++
 include/linux/sched.h | 37 ++
 kernel/fork.c | 10 ++++++++++
 3 files changed, 56 insertions(+), 0 deletions(-)

diff --git a/include/linux/init_task.h b/include/linux/init_task.h
index 6b281fa..b560381 100644
--- a/include/linux/init_task.h
+++ b/include/linux/init_task.h
@@ -15,6 +15,14 @@
 extern struct files_struct init_files;
 extern struct fs_struct init_fs;

+#ifdef CONFIG_CGROUPS
+#define INIT_THREADGROUP_FORK_LOCK(sig)   \
+ .threadgroup_fork_lock =   \
+ __RWSEM_INITIALIZER(sig.threadgroup_fork_lock),
+#else
+#define INIT_THREADGROUP_FORK_LOCK(sig)
+#endif
+
 #define INIT_SIGNALS(sig) {   \
  .nr_threads = 1,   \
  .wait_chldexit = __WAIT_QUEUE_HEAD_INITIALIZER(sig.wait_chldexit),\
@@ -31,6 +39,7 @@ extern struct fs_struct init_fs;
  },   \
  .cred_guard_mutex =   \
   __MUTEX_INITIALIZER(sig.cred_guard_mutex), \
+ INIT_THREADGROUP_FORK_LOCK(sig)   \

```
 }

 extern struct nsproxy init_nsproxy;
diff --git a/include/linux/sched.h b/include/linux/sched.h
index 8580dc6..2fdbeb1 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -509,6 +509,8 @@ struct thread_group_cputimer {
  spinlock_t lock;
 };

+#include <linux/rwsem.h>
+
 /*
  * NOTE! "signal_struct" does not have it's own
  * locking, because a shared signal_struct always
@@ -623,6 +625,16 @@ struct signal_struct {
  unsigned audit_tty;
  struct tty_audit_buf *tty_audit_buf;
 #endif
+#ifdef CONFIG_CGROUPS
+ /*
+  * The threadgroup_fork_lock prevents threads from forking with
+  * CLONE_THREAD while held for writing. Use this for fork-sensitive
+  * threadgroup-wide operations. It's taken for reading in fork.c in
+  * copy_process().
+  * Currently only needed write-side by cgroups.
+  */
+ struct rw_semaphore threadgroup_fork_lock;
+#endif

  int oom_adj;  /* OOM kill score adjustment (bit shift) */
  int oom_score_adj; /* OOM kill score adjustment */
@@ -2270,6 +2282,31 @@ static inline void unlock_task_sighand(struct task_struct *tsk,
  spin_unlock_irqrestore(&tsk->sighand->siglock, *flags);
 }

+/* See the declaration of threadgroup_fork_lock in signal_struct. */
+#ifdef CONFIG_CGROUPS
+static inline void threadgroup_fork_read_lock(struct task_struct *tsk)
+{
+ down_read(&tsk->signal->threadgroup_fork_lock);
+}
+static inline void threadgroup_fork_read_unlock(struct task_struct *tsk)
+{
+ up_read(&tsk->signal->threadgroup_fork_lock);
+}
+static inline void threadgroup_fork_write_lock(struct task_struct *tsk)
```

```
+{
+ down_write(&tsk->signal->threadgroup_fork_lock);
+}
+static inline void threadgroup_fork_write_unlock(struct task_struct *tsk)
+{
+ up_write(&tsk->signal->threadgroup_fork_lock);
+}
+#else
+static inline void threadgroup_fork_read_lock(struct task_struct *tsk) {}
+static inline void threadgroup_fork_read_unlock(struct task_struct *tsk) {}
+static inline void threadgroup_fork_write_lock(struct task_struct *tsk) {}
+static inline void threadgroup_fork_write_unlock(struct task_struct *tsk) {}
+#endif
+
 #ifndef __HAVE_THREAD_FUNCTIONS

 #define task_thread_info(task) ((struct thread_info *)(task)->stack)
diff --git a/kernel/fork.c b/kernel/fork.c
index 0979527..aefe61f 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -905,6 +905,10 @@ static int copy_signal(unsigned long clone_flags, struct task_struct *tsk)

  tty_audit_fork(sig);

+#ifdef CONFIG_CGROUPS
+ init_rwsem(&sig->threadgroup_fork_lock);
+#endif
+
  sig->oom_adj = current->signal->oom_adj;
  sig->oom_score_adj = current->signal->oom_score_adj;
  sig->oom_score_adj_min = current->signal->oom_score_adj_min;
@@ -1087,6 +1091,8 @@ static struct task_struct *copy_process(unsigned long clone_flags,
  monotonic_to_bootbased(&p->real_start_time);
  p->io_context = NULL;
  p->audit_context = NULL;
+ if (clone_flags & CLONE_THREAD)
+  threadgroup_fork_read_lock(current);
  cgroup_fork(p);
 #ifdef CONFIG_NUMA
  p->mempolicy = mpol_dup(p->mempolicy);
@@ -1294,6 +1300,8 @@ static struct task_struct *copy_process(unsigned long clone_flags,
  write_unlock_irq(&tasklist_lock);
  proc_fork_connector(p);
  cgroup_post_fork(p);
+ if (clone_flags & CLONE_THREAD)
+  threadgroup_fork_read_unlock(current);
  perf_event_fork(p);
```

```
  return p;

@@ -1332,6 +1340,8 @@ bad_fork_cleanup_policy:
  mpol_put(p->mempolicy);
 bad_fork_cleanup_cgroup:
 #endif
+ if (clone_flags & CLONE_THREAD)
+  threadgroup_fork_read_unlock(current);
  cgroup_exit(p, cgroup_callbacks_done);
  delayacct_tsk_free(p);
  module_put(task_thread_info(p)->exec_domain->module);
```

_____

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