
Subject: [PATCH -utrace] Move utrace into task_struct
Posted by [Alexey Dobriyan](#) on Wed, 02 May 2007 11:24:42 GMT
[View Forum Message](#) <> [Reply to Message](#)

This patch moves "struct utrace" into "struct task_struct" directly instead of being referenced by a pointer from task_struct. The main reason is utrace code leaving stale ->utrace pointer and freeing "struct utrace" itself. This manifests as crashes in __rcu_process_callbacks() and other nasties.

As side effect this makes attaching and detaching logic much simpler and streamlined. As you see ~200 lines are gone.

I can't oops 2-way Opteron box with this patch applied.

NOTE: patch is against (utrace patch against 2.6.21)

NOTE: "utrace_engine_cache" still leaks

NOTE: "XXX ptrace_report_death" leaks still happen

NOTE: most certainly some locking is still missed and I need to test on a couple more boxes.

Signed-off-by: Alexey Dobriyan <adobriyan@sw.ru>

```
include/linux/sched.h    | 7
include/linux/tracehook.h | 15 -
include/linux/utrace.h    | 39 +-
kernel/fork.c             | 9
kernel/utrace.c           | 457 ++++++++-----
5 files changed, 157 insertions(+), 370 deletions(-)
```

```
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -83,6 +83,7 @@ @@ #include <linux/resource.h>
#include <linux/timer.h>
#include <linux/hrtimer.h>
#include <linux/task_io_accounting.h>
+#include <linux/utrace.h>
```

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

```
@@ -940,11 +941,7 @@ @@ #endif
void *security;
struct audit_context *audit_context;
seccomp_t seccomp;
-
-#ifdef CONFIG_UTRACE
- struct utrace *utrace;
- unsigned long utrace_flags;
```

```

-#endif
+ struct utrace utrace;

/* Thread group tracking */
    u32 parent_exec_id;
--- a/include/linux/tracehook.h
+++ b/include/linux/tracehook.h
@@ -310,16 +310,6 @@ utrace_regset_copyin_ignore(unsigned int
    ***
    ***/

-
-/*
- * Called in copy_process when setting up the copied task_struct,
- * with tasklist_lock held for writing.
- */
-static inline void tracehook_init_task(struct task_struct *child)
-{
- utrace_init_task(child);
-}
-
/*
 * Called from release_task, no locks held.
 * After this, there should be no tracing entanglements.
@@ -327,8 +317,7 @@ static inline void tracehook_init_task(s
static inline void tracehook_release_task(struct task_struct *p)
{
    smp_mb();
- if (tsk_utrace_struct(p) != NULL)
- utrace_release_task(p);
+ utrace_release_task(p);
}

/*
@@ -339,7 +328,7 @@ static inline void tracehook_release_tas
*/
static inline int tracehook_check_released(struct task_struct *p)
{
- return unlikely(tsk_utrace_struct(p) != NULL);
+ return 0;
}

/*
--- a/include/linux/utrace.h
+++ b/include/linux/utrace.h
@@ -50,11 +50,30 @@ #include <linux/sched.h>

struct linux_binprm;

```

```

struct pt_regs;
-struct utrace;
+struct task_struct;
struct utrace_signal;
struct utrace_regset;
struct utrace_regset_view;

+#ifdef CONFIG_UTRACE
+struct utrace {
+ unsigned long flags;
+ union {
+  struct {
+   struct task_struct *cloning;
+   struct utrace_signal *signal;
+  } live;
+  struct {
+   unsigned long flags;
+  } exit;
+ } u;
+ struct list_head engines;
+ spinlock_t lock;
+};
+#else
+struct utrace {
+};
+#endif

/*
 * Flags in task_struct.utrace_flags and utrace_attached_engine.flags.
@@ -493,19 +512,8 @@ void utrace_signal_handler_singlestep(st
/*
 * <linux/tracehook.h> uses these accessors to avoid #ifdef CONFIG_UTRACE.
 */
-static inline unsigned long tsk_utrace_flags(struct task_struct *tsk)
-{
- return tsk->utrace_flags;
-}
-static inline struct utrace *tsk_utrace_struct(struct task_struct *tsk)
-{
- return tsk->utrace;
-}
-static inline void utrace_init_task(struct task_struct *child)
-{
- child->utrace_flags = 0;
- child->utrace = NULL;
-}
+#define tsk_utrace_flags(tsk) ((tsk)->utrace.flags)
+#define tsk_utrace_struct(tsk) (&(tsk)->utrace)

```

```

#else /* !CONFIG_UTRACE */

@@ -517,9 +525,6 @@ static struct utrace *tsk_utrace_struct(
{
    return NULL;
}
-static inline void utrace_init_task(struct task_struct *child)
-{
-}

/*
 * The calls to these should all be in if (0) and optimized out entirely.
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -1241,8 +1241,13 @@ #endif

    if (likely(p->pid)) {
        add_parent(p);
-    tracehook_init_task(p);
-
+
+#ifdef CONFIG_UTRACE
+    p->utrace.flags = 0;
+    p->utrace.u.live.cloning = NULL;
+    p->utrace.u.live.signal = NULL;
+    INIT_LIST_HEAD(&p->utrace.engines);
+    spin_lock_init(&p->utrace.lock);
+#endif
        if (thread_group_leader(p)) {
            p->signal->tty = current->signal->tty;
            p->signal->pgrp = process_group(current);
--- a/kernel/utrace.c
+++ b/kernel/utrace.c
@@ -10,13 +10,13 @@
 * Red Hat Author: Roland McGrath.
 */

-#include <linux/utrace.h>
#include <linux/tracehook.h>
#include <linux/err.h>
#include <linux/sched.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/slab.h>
+#include <linux/utrace.h>
#include <asm/tracehook.h>

```

```

@@ -29,48 +29,11 @@ #define CHECK_INIT(p) do { } while (0)
#define CHECK_DEAD(p) do { } while (0)
#endif

-/*
- * Per-thread structure task_struct.utrace points to.
- *
- * The task itself never has to worry about this going away after
- * some event is found set in task_struct.utrace_flags.
- * Once created, this pointer is changed only when the task is quiescent
- * (TASK_TRACED or TASK_STOPPED with the siglock held, or dead).
- *
- * For other parties, the pointer to this is protected by RCU and
- * task_lock. Since call_rcu is never used while the thread is alive and
- * using this struct utrace, we can overlay the RCU data structure used
- * only for a dead struct with some local state used only for a live utrace
- * on an active thread.
- */
-struct utrace
-{
- union {
- struct rcu_head dead;
- struct {
- struct task_struct *cloning;
- struct utrace_signal *signal;
- } live;
- struct {
- unsigned long flags;
- } exit;
- } u;
-
- struct list_head engines;
- spinlock_t lock;
- atomic_t check_dead;
-};
-
-static struct kmem_cache *utrace_cachep;
static struct kmem_cache *utrace_engine_cachep;

static int __init
utrace_init(void)
{
- utrace_cachep =
- kmem_cache_create("utrace_cache",
- sizeof(struct utrace), 0,
- SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
utrace_engine_cachep =
kmem_cache_create("utrace_engine_cache",

```

```

    sizeof(struct utrace_attached_engine), 0,
@@ -79,106 +42,6 @@ utrace_init(void)
}
subsys_initcall(utrace_init);

-
-/*
- * Make sure target->utrace is allocated, and return with it locked on
- * success. This function mediates startup races. The creating parent
- * task has priority, and other callers will delay here to let its call
- * succeed and take the new utrace lock first.
- */
-static struct utrace *
-utrace_first_engine(struct task_struct *target,
-    struct utrace_attached_engine *engine)
-__acquires(utrace->lock)
-{
-    struct utrace *utrace, *ret;
-
-    /*
-     * If this is a newborn thread and we are not the creator,
-     * we have to wait for it. The creator gets the first chance
-     * to attach. The PF_STARTING flag is cleared after its
-     * report_clone hook has had a chance to run.
-     */
-    if ((target->flags & PF_STARTING)
-        && (current->utrace == NULL
-            || current->utrace->u.live.cloning != target)) {
-        yield();
-        return (signal_pending(current)
-            ? ERR_PTR(-ERESTARTNOINTR) : NULL);
-    }
-
-    utrace = kmem_cache_alloc(utrace_cachep, GFP_KERNEL);
-    if (unlikely(utrace == NULL))
-        return ERR_PTR(-ENOMEM);
-
-    utrace->u.live.cloning = NULL;
-    utrace->u.live.signal = NULL;
-    INIT_LIST_HEAD(&utrace->engines);
-    list_add(&engine->entry, &utrace->engines);
-    spin_lock_init(&utrace->lock);
-    CHECK_INIT(utrace);
-
-    ret = utrace;
-    spin_lock(&utrace->lock);
-    task_lock(target);
-    if (likely(target->utrace == NULL)) {

```

```

- rcu_assign_pointer(target->utrace, utrace);
- /*
-  * The task_lock protects us against another thread doing
-  * the same thing. We might still be racing against
-  * tracehook_release_task. It's called with ->exit_state
-  * set to EXIT_DEAD and then checks ->utrace with an
-  * smp_mb() in between. If EXIT_DEAD is set, then
-  * release_task might have checked ->utrace already and saw
-  * it NULL; we can't attach. If we see EXIT_DEAD not yet
-  * set after our barrier, then we know release_task will
-  * see our target->utrace pointer.
-  */
- smp_mb();
- if (target->exit_state == EXIT_DEAD) {
- /*
-  * The target has already been through release_task.
-  */
- target->utrace = NULL;
- goto cannot_attach;
- }
- task_unlock(target);
- }
- else {
- /*
-  * Another engine attached first, so there is a struct already.
-  * A null return says to restart looking for the existing one.
-  */
- cannot_attach:
- ret = NULL;
- task_unlock(target);
- spin_unlock(&utrace->lock);
- kmem_cache_free(utrace_cachep, utrace);
- }
-
- return ret;
-}
-
-static void
-utrace_free(struct rcu_head *rhead)
-{
- struct utrace *utrace = container_of(rhead, struct utrace, u.dead);
- kmem_cache_free(utrace_cachep, utrace);
-}
-
-/*
- * Called with utrace locked. Clean it up and free it via RCU.
- */
-static void

```

```

-rcu_utrace_free(struct utrace *utrace)
- __releases(utrace->lock)
-{
- CHECK_DEAD(utrace);
- spin_unlock(&utrace->lock);
- INIT_RCU_HEAD(&utrace->u.dead);
- call_rcu(&utrace->u.dead, utrace_free);
-}
-
static void
utrace_engine_free(struct rcu_head *rhead)
{
@@ -200,16 +63,10 @@ rcu_engine_free(struct utrace_attached_e
* forced signal (or it's quiescent in utrace_get_signal).
*/
static inline void
-utrace_clear_tsk(struct task_struct *tsk, struct utrace *utrace)
+utrace_clear_tsk(struct utrace *utrace)
{
- if (utrace->u.live.signal == NULL) {
- task_lock(tsk);
- if (likely(tsk->utrace != NULL)) {
- rcu_assign_pointer(tsk->utrace, NULL);
- tsk->utrace_flags &= UTRACE_ACTION_NOEAP;
- }
- task_unlock(tsk);
- }
+ if (utrace->u.live.signal == NULL)
+ utrace->flags &= UTRACE_ACTION_NOEAP;
}

/*
@@ -218,12 +75,11 @@ utrace_clear_tsk(struct task_struct *tsk
* pending, utrace is left locked and not freed, but is removed from the task.
*/
static void
-remove_engine(struct utrace_attached_engine *engine,
- struct task_struct *tsk, struct utrace *utrace)
+remove_engine(struct utrace_attached_engine *engine, struct utrace *utrace)
{
list_del_rcu(&engine->entry);
if (list_empty(&utrace->engines))
- utrace_clear_tsk(tsk, utrace);
+ utrace_clear_tsk(utrace);
rcu_engine_free(engine);
}

@@ -254,7 +110,7 @@ check_dead_utrace(struct task_struct *ts

```

```

    * If tracing was preventing a SIGCHLD or self-reaping
    * and is no longer, we'll do that report or reaping now.
    */
- if (((tsk->utrace_flags &~ flags) & UTRACE_ACTION_NOREAP)
+ if (((tsk->utrace.flags &~ flags) & UTRACE_ACTION_NOREAP)
    && tsk->exit_state) {
    /*
    * While holding the utrace lock, mark that it's been done.
@@ -304,11 +160,8 @@ check_dead_utrace(struct task_struct *ts

}

- tsk->utrace_flags = flags;
- if (flags)
- spin_unlock(&utrace->lock);
- else
- rcu_utrace_free(utrace);
+ tsk->utrace.flags = flags;
+ spin_unlock(&utrace->lock);

/*
* Now we're finished updating the utrace state.
@@ -348,7 +201,7 @@ quiesce(struct task_struct *target, int
{
    int quiescent;

- target->utrace_flags |= UTRACE_ACTION QUIESCE;
+ target->utrace.flags |= UTRACE_ACTION QUIESCE;
    read_barrier_depends();

    quiescent = (target->exit_state
@@ -399,93 +252,63 @@ struct utrace_attached_engine *
    utrace_attach(struct task_struct *target, int flags,
        const struct utrace_engine_ops *ops, void *data)
    {
- struct utrace *utrace;
+ struct utrace *utrace = &target->utrace;
    struct utrace_attached_engine *engine;
+ int wait_for_creator = 0;

- restart:
- rcu_read_lock();
- utrace = rcu_dereference(target->utrace);
- smp_rmb();
- if (unlikely(target->exit_state == EXIT_DEAD)) {
+ if (unlikely(target->exit_state == EXIT_DEAD))
+ if (unlikely(target->exit_state == EXIT_DEAD))
    /*
    * The target has already been reaped.

```

```

    * Check this first; a race with reaping may lead to restart.
    */
- rcu_read_unlock();
  return ERR_PTR(-ESRCH);
- }
- if (utrace == NULL) {
- rcu_read_unlock();
-
- if (!(flags & UTRACE_ATTACH_CREATE))
- return ERR_PTR(-ENOENT);
+ /*
+ * If this is a newborn thread and we are not the creator,
+ * we have to wait for it. The creator gets the first chance
+ * to attach. The PF_STARTING flag is cleared after its
+ * report_clone hook has had a chance to run.
+ */
+ if (target->flags & PF_STARTING) {
+ struct utrace *u = &current->utrace;

- engine = kmem_cache_alloc(utrace_engine_cache, GFP_KERNEL);
- if (unlikely(engine == NULL))
- return ERR_PTR(-ENOMEM);
- engine->flags = 0;
- CHECK_INIT(engine);
+ spin_lock(&u->lock);
+ wait_for_creator = (u->u.live.cloning != target);
+ spin_unlock(&u->lock);
+ }
+ if (wait_for_creator) {
+ yield();
+ return (signal_pending(current)
+ ? ERR_PTR(-ERESTARTNOINTR) : NULL);
+ }

- first:
- utrace = utrace_first_engine(target, engine);
- if (IS_ERR(utrace) || unlikely(utrace == NULL)) {
- kmem_cache_free(utrace_engine_cache, engine);
- if (unlikely(utrace == NULL)) /* Race condition. */
- goto restart;
- return ERR_PTR(PTR_ERR(utrace));
- }
+ spin_lock(&utrace->lock);
+ if (!(flags & UTRACE_ATTACH_CREATE)) {
+ engine = matching_engine(utrace, flags, ops, data);
+ spin_unlock(&utrace->lock);
+ return engine;
+ }

```

```

- else {
- if (!(flags & UTRACE_ATTACH_CREATE)) {
- engine = matching_engine(utrace, flags, ops, data);
- rcu_read_unlock();
- return engine;
- }
- rcu_read_unlock();
-
- engine = kmem_cache_alloc(utrace_engine_cachep, GFP_KERNEL);
- if (unlikely(engine == NULL))
- return ERR_PTR(-ENOMEM);
- engine->flags = 0;
- CHECK_INIT(engine);
-
- rcu_read_lock();
- utrace = rcu_dereference(target->utrace);
- if (unlikely(utrace == NULL)) { /* Race with detach. */
- rcu_read_unlock();
- goto first;
- }
- spin_lock(&utrace->lock);
+ spin_unlock(&utrace->lock);

- if (flags & UTRACE_ATTACH_EXCLUSIVE) {
- struct utrace_attached_engine *old;
- old = matching_engine(utrace, flags, ops, data);
- if (!IS_ERR(old)) {
- spin_unlock(&utrace->lock);
- rcu_read_unlock();
- kmem_cache_free(utrace_engine_cachep, engine);
- return ERR_PTR(-EEXIST);
- }
- }
+ engine = kmem_cache_alloc(utrace_engine_cachep, GFP_KERNEL);
+ if (unlikely(engine == NULL))
+ return ERR_PTR(-ENOMEM);
+ engine->flags = 0;
+ engine->ops = ops;
+ engine->data = data;
+ CHECK_INIT(engine);

- if (unlikely(rcu_dereference(target->utrace) != utrace)) {
- /*
-  * We lost a race with other CPUs doing a sequence
-  * of detach and attach before we got in.
-  */
+ spin_lock(&utrace->lock);
+ if (flags & UTRACE_ATTACH_EXCLUSIVE) {

```

```

+ struct utrace_attached_engine *old;
+
+ old = matching_engine(utrace, flags, ops, data);
+ if (!IS_ERR(old)) {
+     spin_unlock(&utrace->lock);
- rcu_read_unlock();
+ kmem_cache_free(utrace_engine_cachep, engine);
- goto restart;
+ return ERR_PTR(-EEXIST);
+ }
- rcu_read_unlock();
-
- list_add_tail_rcu(&engine->entry, &utrace->engines);
+ }
-
- engine->ops = ops;
- engine->data = data;
-
+ list_add_tail_rcu(&engine->entry, &utrace->engines);
+ spin_unlock(&utrace->lock);

return engine;
@@ -627,13 +450,10 @@ get_utrace_lock_attached(struct task_str
    struct utrace_attached_engine *engine)
    __acquires(utrace->lock)
{
- struct utrace *utrace;
+ struct utrace *utrace = &target->utrace;
+ int dead_engine;

- rcu_read_lock();
- utrace = rcu_dereference(target->utrace);
- smp_rmb();
- if (unlikely(utrace == NULL)
-     || unlikely(target->exit_state == EXIT_DEAD))
+ if (unlikely(target->exit_state == EXIT_DEAD))
/*
 * If all engines detached already, utrace is clear.
 * Otherwise, we're called after utrace_release_task might
@@ -641,22 +461,18 @@ get_utrace_lock_attached(struct task_str
 * callback might already be in progress or engine might
 * even have been freed already.
 */
- utrace = ERR_PTR(-ESRCH);
- else {
-     spin_lock(&utrace->lock);
-     if (unlikely(rcu_dereference(target->utrace) != utrace)
-         || unlikely(rcu_dereference(engine->ops)

```

```

- == &dead_engine_ops)) {
- /*
-  * By the time we got the utrace lock,
-  * it had been reaped or detached already.
-  */
- spin_unlock(&utrace->lock);
- utrace = ERR_PTR(-ESRCH);
- }
- }
+ return ERR_PTR(-ESRCH);
+ rcu_read_lock();
+ dead_engine = rcu_dereference(engine->ops) == &dead_engine_ops;
+ rcu_read_unlock();
+ if (unlikely(dead_engine))
+ /*
+  * By the time we got the utrace lock,
+  * it had been reaped or detached already.
+  */
+ return ERR_PTR(-ESRCH);

+ spin_lock(&utrace->lock);
+ return utrace;
+ }

@@ -691,7 +507,7 @@ utrace_detach(struct task_struct *target
+ rcu_assign_pointer(engine->ops, &dead_engine_ops);

+ if (quiesce(target, 1)) {
- remove_engine(engine, target, utrace);
+ remove_engine(engine, utrace);
+ wake_quiescent(flags, utrace, target);
+ }
+ else
@@ -733,8 +549,7 @@ restart:
+ }
+ rcu_engine_free(engine);
+ }
-
- rcu_utrace_free(utrace);
+ spin_unlock(&utrace->lock);
+ }

/*
@@ -743,20 +558,12 @@ restart:
void
utrace_release_task(struct task_struct *target)
{
- struct utrace *utrace;

```

```

-
- task_lock(target);
- utrace = target->utrace;
- rcu_assign_pointer(target->utrace, NULL);
- task_unlock(target);
-
- if (unlikely(utrace == NULL))
- return;
+ struct utrace *utrace = &target->utrace;

spin_lock(&utrace->lock);
utrace->u.exit.flags |= EXIT_FLAG_REAP;

- if (target->utrace_flags & (UTRACE_EVENT(DEATH)
+ if (target->utrace.flags & (UTRACE_EVENT(DEATH)
    | UTRACE_EVENT(QUIESCE)))
/*
 * The target will do some final callbacks but hasn't
@@ -797,7 +604,7 @@ #endif

restart: /* See below. */

- old_utrace_flags = target->utrace_flags;
+ old_utrace_flags = target->utrace.flags;
old_flags = engine->flags;

if (target->exit_state
@@ -833,12 +640,12 @@ restart: /* See below. */
    spin_unlock(&utrace->lock);
    return ret;
}
- target->utrace_flags |= flags;
+ target->utrace.flags |= flags;
read_unlock(&tasklist_lock);
}

engine->flags = flags;
- target->utrace_flags |= flags;
+ target->utrace.flags |= flags;
ret = 0;

report = 0;
@@ -861,7 +668,7 @@ restart: /* See below. */
 * in user mode to get those effects, even if the target is
 * not going to be quiescent right now.
 */
- if (!(target->utrace_flags & UTRACE_ACTION QUIESCE)
+ if (!(target->utrace.flags & UTRACE_ACTION QUIESCE)

```

```

    && !target->exit_state
    && ((flags &~ old_utrace_flags)
    & (UTRACE_ACTION_SINGLESTEP | UTRACE_ACTION_BLOCKSTEP
@@ -934,10 +741,10 @@ #endif
    */)
    if (((ret ^ engine->flags) & (UTRACE_ACTION_STATE_MASK
        & ~UTRACE_ACTION QUIESCE)))
-   tsk->utrace_flags |= UTRACE_ACTION QUIESCE;
+   tsk->utrace.flags |= UTRACE_ACTION QUIESCE;
    engine->flags &= ~UTRACE_ACTION_STATE_MASK;
    engine->flags |= ret & UTRACE_ACTION_STATE_MASK;
-   tsk->utrace_flags |= engine->flags;
+   tsk->utrace.flags |= engine->flags;
    spin_unlock(&utrace->lock);
    }
    else
@@ -965,7 +772,7 @@ remove_detached(struct task_struct *tsk,

    list_for_each_entry_safe(engine, next, &utrace->engines, entry) {
        if (engine->ops == &dead_engine_ops)
-       remove_engine(engine, tsk, utrace);
+       remove_engine(engine, utrace);
        else
            flags |= engine->flags | UTRACE_EVENT(REAP);
    }
@@ -987,8 +794,8 @@ check_detach(struct task_struct *tsk, u3
    * getting into utrace_report_death.
    */
    BUG_ON(tsk != current);
-   spin_lock(&tsk->utrace->lock);
-   action = remove_detached(tsk, tsk->utrace, action, ~0UL);
+   spin_lock(&tsk->utrace.lock);
+   action = remove_detached(tsk, &tsk->utrace, action, ~0UL);
    }
    return action;
    }
@@ -1011,7 +818,7 @@ void
utrace_report_clone(unsigned long clone_flags, struct task_struct *child)
{
    struct task_struct *tsk = current;
-   struct utrace *utrace = tsk->utrace;
+   struct utrace *utrace = &tsk->utrace;
    struct list_head *pos, *next;
    struct utrace_attached_engine *engine;
    unsigned long action;
@@ -1056,7 +863,7 @@ int
utrace_report_jctl(int what)
{

```

```

    struct task_struct *tsk = current;
- struct utrace *utrace = tsk->utrace;
+ struct utrace *utrace = &tsk->utrace;
    struct list_head *pos, *next;
    struct utrace_attached_engine *engine;
    unsigned long action;
@@ -1122,7 +929,7 @@ sigkill_pending(struct task_struct *tsk)
    int
    utrace_quiescent(struct task_struct *tsk, struct utrace_signal *signal)
    {
- struct utrace *utrace = tsk->utrace;
+ struct utrace *utrace = &tsk->utrace;
    unsigned long action;

restart:
@@ -1151,7 +958,7 @@ restart:
    * Never stop when there is a SIGKILL bringing us down.
    */
    killed = sigkill_pending(tsk);
- if (!killed && (tsk->utrace_flags & UTRACE_ACTION QUIESCE)) {
+ if (!killed && (tsk->utrace.flags & UTRACE_ACTION QUIESCE)) {
    set_current_state(TASK_TRACED);
    /*
    * If there is a group stop in progress,
@@ -1171,7 +978,7 @@ restart:
    * u.live.signal is set, see check_dead_utrace.
    * This makes it safe to clear its pointer here.
    */
- BUG_ON(tsk->utrace != utrace);
+ BUG_ON(&tsk->utrace != utrace);
    BUG_ON(utrace->u.live.signal != signal);
    utrace->u.live.signal = NULL;
    }
@@ -1186,12 +993,12 @@ restart:
    * longer quiescent, so don't need to do any RCU locking.
    * But we do need to check our utrace pointer anew.
    */
- utrace = tsk->utrace;
- if (tsk->utrace_flags
+ utrace = &tsk->utrace;
+ if (tsk->utrace.flags
    & (UTRACE_EVENT(QUIESCE) | UTRACE_ACTION_STATE_MASK))
    goto restart;
    }
- else if (tsk->utrace_flags & UTRACE_ACTION QUIESCE) {
+ else if (tsk->utrace.flags & UTRACE_ACTION QUIESCE) {
    /*
    * Our flags are out of date.

```

```

    * Update the set of events of interest from the union
@@ -1202,11 +1009,12 @@ restart:
    * still needs to process a pending forced signal.
    */
    unsigned long flags;
-   utrace = rcu_dereference(tsk->utrace);
+
+   utrace = &tsk->utrace;
    spin_lock(&utrace->lock);
    flags = rescan_flags(utrace);
    if (flags == 0)
-   utrace_clear_tsk(tsk, utrace);
+   utrace_clear_tsk(utrace);
    check_dead_utrace(tsk, utrace, flags);
}

@@ -1226,7 +1034,7 @@ #ifdef ARCH_HAS_BLOCK_STEP
    else
        tracehook_disable_block_step(tsk);
#endif
-   if (tsk->utrace_flags & UTRACE_EVENT_SYSCALL)
+   if (tsk->utrace.flags & UTRACE_EVENT_SYSCALL)
        tracehook_enable_syscall_trace(tsk);
    else
        tracehook_disable_syscall_trace(tsk);
@@ -1242,7 +1050,7 @@ void
utrace_report_exit(long *exit_code)
{
    struct task_struct *tsk = current;
-   struct utrace *utrace = tsk->utrace;
+   struct utrace *utrace = &tsk->utrace;
    struct list_head *pos, *next;
    struct utrace_attached_engine *engine;
    unsigned long action;
@@ -1386,17 +1194,9 @@ utrace_report_death(struct task_struct *
void
utrace_report_delayed_group_leader(struct task_struct *tsk)
{
-   struct utrace *utrace;
+   struct utrace *utrace = &tsk->utrace;

-   rcu_read_lock();
-   utrace = rcu_dereference(tsk->utrace);
-   if (unlikely(utrace == NULL)) {
-       rcu_read_unlock();
-       return;
-   }
    spin_lock(&utrace->lock);

```

```

- rcu_read_unlock();
-
  utrace->u.exit.flags |= EXIT_FLAG_DELAYED_GROUP_LEADER;

/*
@@ -1404,7 +1204,7 @@ utrace_report_delayed_group_leader(struc
  * started already, there is nothing more to do now.
  */
  if ((utrace->u.exit.flags & (EXIT_FLAG_DEATH | EXIT_FLAG_REAP))
-   || !likely(tsk->utrace_flags & UTRACE_ACTION_NOEAP))
+   || !likely(tsk->utrace.flags & UTRACE_ACTION_NOEAP))
    spin_unlock(&utrace->lock);
  else
    report_delayed_group_leader(tsk, utrace);
@@ -1417,7 +1217,7 @@ void
utrace_report_vfork_done(pid_t child_pid)
{
  struct task_struct *tsk = current;
- struct utrace *utrace = tsk->utrace;
+ struct utrace *utrace = &tsk->utrace;
  struct list_head *pos, *next;
  struct utrace_attached_engine *engine;
  unsigned long action;
@@ -1442,7 +1242,7 @@ void
utrace_report_exec(struct linux_binprm *bprm, struct pt_regs *regs)
{
  struct task_struct *tsk = current;
- struct utrace *utrace = tsk->utrace;
+ struct utrace *utrace = &tsk->utrace;
  struct list_head *pos, *next;
  struct utrace_attached_engine *engine;
  unsigned long action;
@@ -1467,7 +1267,7 @@ void
utrace_report_syscall(struct pt_regs *regs, int is_exit)
{
  struct task_struct *tsk = current;
- struct utrace *utrace = tsk->utrace;
+ struct utrace *utrace = &tsk->utrace;
  struct list_head *pos, *next;
  struct utrace_attached_engine *engine;
  unsigned long action, ev;
@@ -1586,7 +1386,7 @@ void
utrace_signal_handler_singlestep(struct task_struct *tsk, struct pt_regs *regs)
{
  u32 action;
- action = report_signal(tsk, regs, tsk->utrace, UTRACE_SIGNAL_HANDLER,
+ action = report_signal(tsk, regs, &tsk->utrace, UTRACE_SIGNAL_HANDLER,
    UTRACE_EVENT_SIGNAL_ALL,

```

```

        UTRACE_ACTION_SINGLESTEP|UTRACE_ACTION_BLOCKSTEP,
        NULL, NULL, NULL);
@@ -1605,7 +1405,7 @@ utrace_get_signal(struct task_struct *ts
    __releases(tsk->sigand->siglock)
    __acquires(tsk->sigand->siglock)
    {
- struct utrace *utrace = tsk->utrace;
+ struct utrace *utrace = &tsk->utrace;
    struct utrace_signal signal = { info, return_ka, 0 };
    struct k_sigaction *ka;
    unsigned long action, event;
@@ -1634,7 +1434,7 @@ utrace_get_signal(struct task_struct *ts
    * First stash a pointer to the state on our stack,
    * so that utrace_inject_signal can tell us what to do.
    */
- if (tsk->utrace_flags & UTRACE_ACTION QUIESCE) {
+ if (tsk->utrace.flags & UTRACE_ACTION QUIESCE) {
    int killed = sigkill_pending(tsk);
    if (!killed) {
        spin_unlock_irq(&tsk->sigand->siglock);
@@ -1698,7 +1498,7 @@ utrace_get_signal(struct task_struct *ts
    * If noone is interested in intercepting signals, let the caller
    * just dequeue them normally.
    */
- if ((tsk->utrace_flags & UTRACE_EVENT_SIGNAL_ALL) == 0)
+ if ((tsk->utrace.flags & UTRACE_EVENT_SIGNAL_ALL) == 0)
    return 0;

    /*
@@ -1749,7 +1549,7 @@ utrace_get_signal(struct task_struct *ts
    action = UTRACE_SIGNAL_TERM;
    }

- if (tsk->utrace_flags & event) {
+ if (tsk->utrace.flags & event) {
    /*
    * We have some interested engines, so tell them about the
    * signal and let them change its disposition.
@@ -1964,23 +1764,19 @@ EXPORT_SYMBOL_GPL(utrace_regset);
    struct task_struct *
    utrace_tracer_task(struct task_struct *target)
    {
- struct utrace *utrace;
+ struct utrace *utrace = &target->utrace;
    struct task_struct *tracer = NULL;
    struct list_head *pos, *next;
    struct utrace_attached_engine *engine;
    const struct utrace_engine_ops *ops;

```

```

- utrace = rcu_dereference(target->utrace);
- if (utrace != NULL) {
-   struct list_head *pos, *next;
-   struct utrace_attached_engine *engine;
-   const struct utrace_engine_ops *ops;
-   list_for_each_safe_rcu(pos, next, &utrace->engines) {
-     engine = list_entry(pos, struct utrace_attached_engine,
-       entry);
-     ops = rcu_dereference(engine->ops);
-     if (ops->tracer_task) {
-       tracer = (*ops->tracer_task)(engine, target);
-       if (tracer != NULL)
-         break;
-     }
+ list_for_each_safe_rcu(pos, next, &utrace->engines) {
+   engine = list_entry(pos, struct utrace_attached_engine, entry);
+   ops = rcu_dereference(engine->ops);
+   if (ops->tracer_task) {
+     tracer = (*ops->tracer_task)(engine, target);
+     if (tracer != NULL)
+       break;
+   }
+ }

```

```

@@ -1990,26 +1786,21 @@ utrace_tracer_task(struct task_struct *t
int
utrace_allow_access_process_vm(struct task_struct *target)
{
- struct utrace *utrace;
+ struct utrace *utrace = &target->utrace;
+ struct list_head *pos, *next;
+ struct utrace_attached_engine *engine;
+ const struct utrace_engine_ops *ops;
  int ret = 0;

  rcu_read_lock();
- utrace = rcu_dereference(target->utrace);
- if (utrace != NULL) {
-   struct list_head *pos, *next;
-   struct utrace_attached_engine *engine;
-   const struct utrace_engine_ops *ops;
-   list_for_each_safe_rcu(pos, next, &utrace->engines) {
-     engine = list_entry(pos, struct utrace_attached_engine,
-       entry);
-     ops = rcu_dereference(engine->ops);
-     if (ops->allow_access_process_vm) {
-       ret = (*ops->allow_access_process_vm)(engine,

```

```

-         target,
-         current);
-     if (ret)
-         break;
- }
+ list_for_each_safe_rcu(pos, next, &utrace->engines) {
+     engine = list_entry(pos, struct utrace_attached_engine, entry);
+     ops = rcu_dereference(engine->ops);
+     if (ops->allow_access_process_vm) {
+         ret = (*ops->allow_access_process_vm)(engine, target,
+         current);
+         if (ret)
+             break;
+     }
+ }
    rcu_read_unlock();
@@ -2024,7 +1815,7 @@ utrace_allow_access_process_vm(struct ta
int
utrace_unsafe_exec(struct task_struct *tsk)
{
- struct utrace *utrace = tsk->utrace;
+ struct utrace *utrace = &tsk->utrace;
    struct list_head *pos, *next;
    struct utrace_attached_engine *engine;
    const struct utrace_engine_ops *ops;

```
