



author Frederic Weisbecker <frederic@kernel.org> 2025-03-04 14:54:46 +0100
committer Greg Kroah-Hartman <gregkh@linuxfoundation.org> 2025-04-20 10:22:22 +0200
commit [1267bd38f161c1a27d9b722de017027167a225a0](#) (patch)
tree [4afa1a8197470119f7c938b66fd64ef67ec4ab3e](#)
parent [1209b0b29fd472e7dbd2b06544b019dd9f9b7e51](#) (diff)
download [linux-1267bd38f161c1a27d9b722de017027167a225a0.tar.gz](#)

diff options

context:
space:
mode:

perf: Fix hang while freeing sigtrap event

[Upstream commit 56799bc035658738f362acec3e7647bb84e68933]

Perf can hang while freeing a sigtrap event if a related deferred signal hadn't managed to be sent before the file got closed:

```
perf_event_overflow()
    task_work_add(perf_pending_task)

fput()
    task_work_add(____fput())

task_work_run()
    ____fput()
    perf_release()
    perf_event_release_kernel()
    _free_event()
    perf_pending_task_sync()
    task_work_cancel() -> FAILED
    rcuwait_wait_event()
```

Once task_work_run() is running, the list of pending callbacks is removed from the task_struct and from this point on task_work_cancel() can't remove any pending and not yet started work items, hence the task_work_cancel() failure and the hang on rcuwait_wait_event().

Task work could be changed to remove one work at a time, so a work running on the current task can always cancel a pending one, however the wait / wake design is still subject to inverted dependencies when remote targets are involved, as pictured by Oleg:

T1

```
fd = perf_event_open(pid => T2->pid);
close(fd)
<IRQ>
perf_event_overflow()
    task_work_add(perf_pending_task)
</IRQ>
fput()
    task_work_add(____fput())

task_work_run()
    ____fput()
    perf_release()
    perf_event_release_kernel()
```

T2

```
fd = perf_event_open(pid => T1->pid);
close(fd)
<IRQ>
perf_event_overflow()
    task_work_add(perf_pending_task)
</IRQ>
fput()
    task_work_add(____fput())

task_work_run()
    ____fput()
    perf_release()
    perf_event_release_kernel()
```

```

_free_event()
perf_pending_task_sync()
rcuwait_wait_event()
_free_event()
perf_pending_task_sync()
rcuwait_wait_event()

```

Therefore the only option left is to acquire the event reference count upon queueing the perf task work and release it from the task work, just like it was done before 3a5465418f5f ("perf: Fix event leak upon exec and file release") but without the leaks it fixed.

Some adjustments are necessary to make it work:

- * A child event might dereference its parent upon freeing. Care must be taken to release the parent last.
- * Some places assuming the event doesn't have any reference held and therefore can be freed right away must instead put the reference and let the reference counting to its job.

Reported-by: "Yi Lai" <yil.lai@linux.intel.com>
 Closes: <https://lore.kernel.org/all/Zx9Losv4YcJowaP%2F@ly-workstation/>
 Reported-by: syzbot+3c4321e10eea460eb606@syzkaller.appspotmail.com
 Closes: <https://lore.kernel.org/all/673adf75.050a0220.87769.0024.GAE@google.com/>
 Fixes: 3a5465418f5f ("perf: Fix event leak upon exec and file release")
 Signed-off-by: Frederic Weisbecker <frederic@kernel.org>
 Signed-off-by: Peter Zijlstra (Intel) <peterz@infradead.org>
 Link: <https://lkml.kernel.org/r/20250304135446.18905-1-frederic@kernel.org>
 Signed-off-by: Sasha Levin <sashal@kernel.org>

Diffstat

```

-rw-r--r-- include/linux/perf_event.h 1
-rw-r--r-- kernel/events/core.c      64

```

2 files changed, 18 insertions, 47 deletions

```

diff --git a/include/linux/perf_event.h b/include/linux/perf_event.h
index 677f80249458e4..93ea9c6672f0e1 100644

```

```

--- a/include/linux/perf_event.h
+++ b/include/linux/perf_event.h
@@ -833,7 +833,6 @@ struct perf_event {
     struct irq_work          pending_disable_irq;
     struct callback_head     pending_task;
     unsigned int             pending_work;
-    struct rcuwait           pending_work_wait;

     atomic_t                 event_limit;

```

```

diff --git a/kernel/events/core.c b/kernel/events/core.c
index 3a69e816d6f125..ee6b7281a19943 100644

```

```

--- a/kernel/events/core.c
+++ b/kernel/events/core.c
@@ -5303,30 +5303,6 @@ static bool exclusive_event_installable(struct perf_event *event,
     static void perf_addr_filters_splice(struct perf_event *event,
                                           struct list_head *head);

-static void perf_pending_task_sync(struct perf_event *event)
-{
-    struct callback_head *head = &event->pending_task;
-
-    if (!event->pending_work)
-        return;
-
-    /*
-     * If the task is queued to the current task's queue, we
-     * obviously can't wait for it to complete. Simply cancel it.
-     */
-    if (task_work_cancel(current, head)) {

```

```

-         event->pending_work = 0;
-         local_dec(&event->ctx->nr_no_switch_fast);
-         return;
-     }
-
-     /*
-      * All accesses related to the event are within the same RCU section in
-      * perf_pending_task(). The RCU grace period before the event is freed
-      * will make sure all those accesses are complete by then.
-      */
-     rcuwait_wait_event(&event->pending_work_wait, !event->pending_work, TASK_UNINTERRUPTIBLE);
-}
-
/* vs perf_event_alloc() error */
static void __free_event(struct perf_event *event)
{
@@ -5379,7 +5355,6 @@ static void _free_event(struct perf_event *event)
{
    irq_work_sync(&event->pending_irq);
    irq_work_sync(&event->pending_disable_irq);
-    perf_pending_task_sync(event);

    unaccount_event(event);

@@ -5472,10 +5447,17 @@ static void perf_remove_from_owner(struct perf_event *event)

static void put_event(struct perf_event *event)
{
+    struct perf_event *parent;
+
    if (!atomic_long_dec_and_test(&event->refcount))
        return;

+    parent = event->parent;
+    _free_event(event);
+
+    /* Matches the refcount bump in inherit_event() */
+    if (parent)
+        put_event(parent);
}

/*
@@ -5559,11 +5541,6 @@ again:
    if (tmp == child) {
        perf_remove_from_context(child, DETACH_GROUP);
        list_move(&child->child_list, &free_list);
-
-        /*
-         * This matches the refcount bump in inherit_event();
-         * this can't be the last reference.
-         */
-        put_event(event);
    } else {
        var = &ctx->refcount;
    }
@@ -5589,7 +5566,8 @@ again:
    void *var = &child->ctx->refcount;

    list_del(&child->child_list);
-    free_event(child);
+    /* Last reference unless ->pending_task work is pending */
+    put_event(child);

    /*
     * Wake any perf_event_free_task() waiting for this event to be
@@ -5600,7 +5578,11 @@ again:

```

```

}

no_ctx:
-   put_event(event); /* Must be the 'last' reference */
+   /*
+    * Last reference unless ->pending_task work is pending on this event
+    * or any of its children.
+    */
+   put_event(event);
+   return 0;
}
EXPORT_SYMBOL_GPL(perf_event_release_kernel);
@@ -7015,12 +6997,6 @@ static void perf_pending_task(struct callback_head *head)
    int rctx;

    /*
-   * All accesses to the event must belong to the same implicit RCU read-side
-   * critical section as the ->pending_work reset. See comment in
-   * perf_pending_task_sync().
-   */
-   rcu_read_lock();
-   /*
    * If we 'fail' here, that's OK, it means recursion is already disabled
    * and we won't recurse 'further'.
    */
@@ -7030,9 +7006,8 @@ static void perf_pending_task(struct callback_head *head)
    event->pending_work = 0;
    perf_sigtrap(event);
    local_dec(&event->ctx->nr_no_switch_fast);
-   rcuwait_wake_up(&event->pending_work_wait);
-   }
-   rcu_read_unlock();
+   put_event(event);

    if (rctx >= 0)
        perf_swevent_put_recursion_context(rctx);
@@ -9978,6 +9953,7 @@ static int __perf_event_overflow(struct perf_event *event,
    !task_work_add(current, &event->pending_task, notify_mode)) {
        event->pending_work = pending_id;
        local_inc(&event->ctx->nr_no_switch_fast);
+       WARN_ON_ONCE(!atomic_long_inc_not_zero(&event->refcount));

        event->pending_addr = 0;
        if (valid_sample && (data->sample_flags & PERF_SAMPLE_ADDR))
@@ -12325,7 +12301,6 @@ perf_event_alloc(struct perf_event_attr *attr, int cpu,
    init_irq_work(&event->pending_irq, perf_pending_irq);
    event->pending_disable_irq = IRQ_WORK_INIT_HARD(perf_pending_disable);
    init_task_work(&event->pending_task, perf_pending_task);
-   rcuwait_init(&event->pending_work_wait);

    mutex_init(&event->mmap_mutex);
    raw_spin_lock_init(&event->addr_filters.lock);
@@ -13466,8 +13441,7 @@ perf_event_exit_event(struct perf_event *event, struct perf_event_context *ctx)
    * Kick perf_poll() for is_event_hup();
    */
    perf_event_wakeup(parent_event);
-   free_event(event);
-   put_event(parent_event);
+   put_event(event);
    return;
}

@@ -13585,13 +13559,11 @@ static void perf_free_event(struct perf_event *event,
    list_del_init(&event->child_list);
    mutex_unlock(&parent->child_mutex);

```

```
-      put_event(parent);  
-  
      raw_spin_lock_irq(&ctx->lock);  
      perf_group_detach(event);  
      list_del_event(event, ctx);  
      raw_spin_unlock_irq(&ctx->lock);  
-      free_event(event);  
+      put_event(event);  
    }  
  
    /*
```