Wait/Wakeup and waketorture

Boqun Feng (Intel)

A bit about me

- Working for Intel OTC VMM Enabling Team
  - UMIP/SGX on xen
- Dedicated Reviewer for ATOMIC INFRASTRUCTURE
- Co-maintainer of restartable sequence(not merged)
  - git://git.kernel.org/pub/scm/linux/kernel/git/rseq/linux-rseq.git
- Co-maintainer of Linux Kernel Memory Model(WIP)

How many of you...

- Know about multithreading?
- Have used or learned primitives in <linux/wait.h>?
- Have read Documentation/memory-barriers.txt?

Warm-up: Is this safe?

```
struct wait_queue_head q; // a queue for block tasks

<TASK A>
1  DEFINE_WAIT(wait); // define a wait structure
2  add_wait_queue(&q, &wait); // add this task to queue
3  while (!condition) { // check condition
4    prepare_to_wait(&q, &wait, TASK_UNINTERRUTBILE);
5    schedule(); // ask scheduler to schedule this out
6  }

  // do something after condition is satisfied

<TASK B>
1  condition = true; // the condition is satisfied.
2  wake_up(q); // wake up!
```

Warm-up: What if this happens?

```
<TASK A>
  DEFINE_WAIT(wait);
  add_wait_queue(&q, &wait);
  while (!condition) {

<SWITCH TO TASK B>
  condition = true;
  wake_up(q);

<Back to TASK A>
    prepare_to_wait(&q, &wait, TASK_UNINTERRUTBILE);
    schedule();
  }
```

Warm-up: Try to fix. Work?

```
<TASK A>
  DEFINE_WAIT(wait);
  add_wait_queue(&q, &wait);
  while (!condition) {

<Switch TO TASK B>
  condition = true;
  wake_up(q);

<Back to TASK A>

+   if (condition)
+     break;

    prepare_to_wait(&q, &wait, TASK_UNINTERRUTBILE);
    schedule();
  }
```

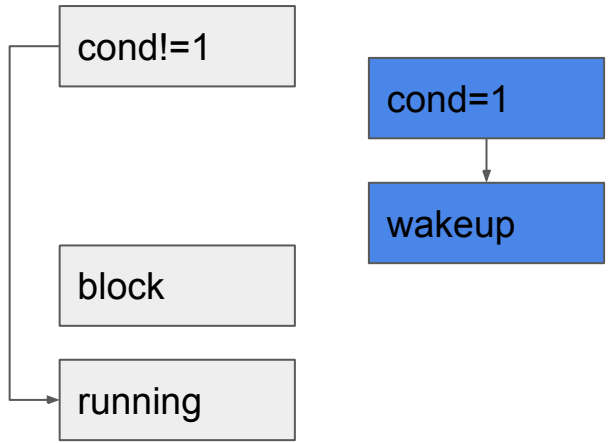
Warm-up: Try to fix. Work?(cont.)

```
<TASK A>
  DEFINE_WAIT(wait);
  add_wait_queue(&q, &wait);
  while (!condition) {

+    if (condition)
+      break;

<Switch TO TASK B>
  condition = true;
  wake_up(q);

<Back to TASK A>
    prepare_to_wait(&q, &wait, TASK_UNINTERRUTBILE);
    schedule();
  }
```

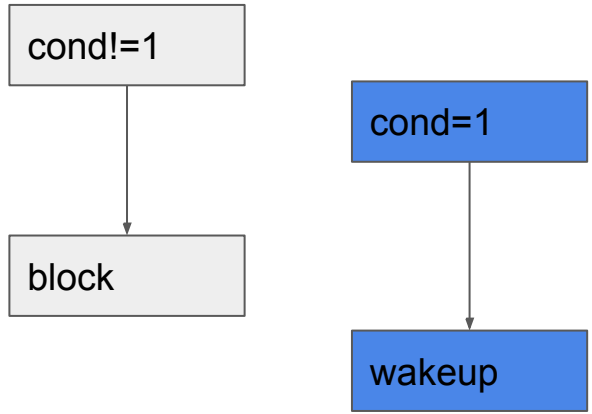
Warm-up: Try to fix

- If the waker sets the @condition to **true** before the wakee tries to block:
  - Either the wakee would **observe** the @condition and **not** block(Trivial)
  - Otherwise the waker must prevent the wakee from blocking forever.

1.



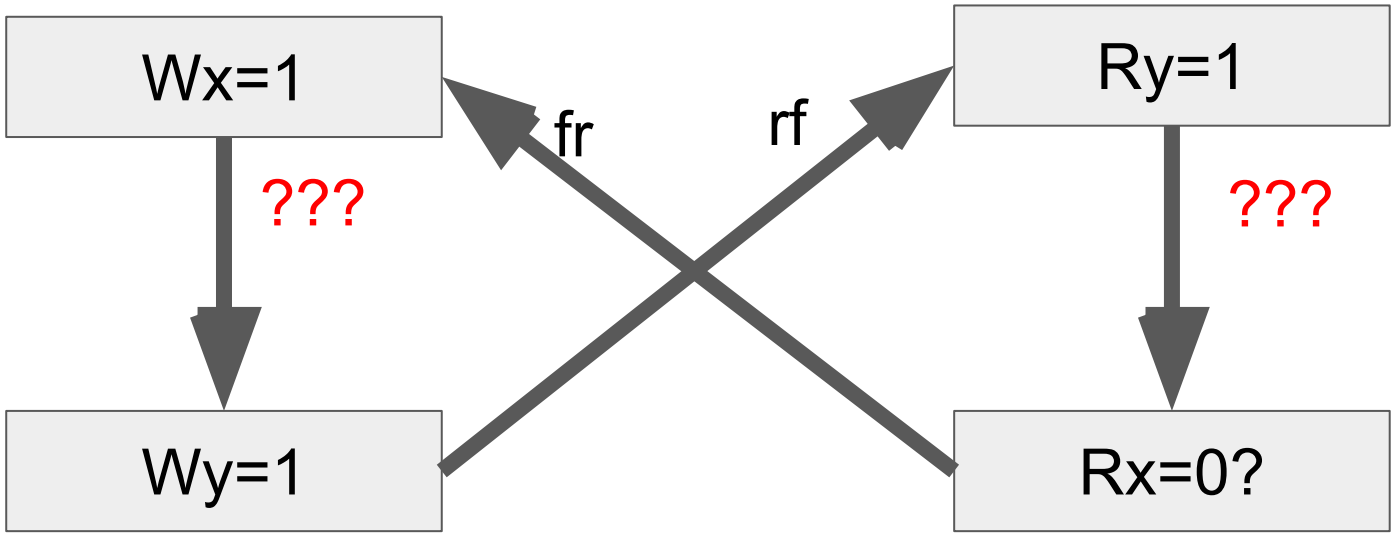
2.



Warm-up: How?

```
struct wait_queue_head q;

<TASK A>
  DEFINE_WAIT(wait)
  add_wait_queue(q, &wait);
  while (!condition) {
    prepare_to_wait(&q, &wait, TASK_UNINTERRUTBILE);
+   if (condition)
+     break;
    schedule();
  }

<TASK B>
  condition = true;
  wake_up(q);
```

Memory Model 101: Message Passing



prepare_to_wait() magic #1

```
current->on_rq = 1;
<???>
prepare_to_wait(...):
  current->state = !TASK_RUNNING;
```

```
wake_up(...):
  try_to_wake_up():
    if (->state)
      goto out; // give up waking
    smp_rmb();

    if (->on_rq);
      ->state = TASK_RUNNING;
```

```
schedule():
  if (->state)
    deactivate_task(); // block
```

Bug: the smp_rmb() was missing

- Spotted at 2016
- Fixed by commit 135e8c9250dd ("sched/core: Fix a race between try_to_wake_up() and a woken up task")

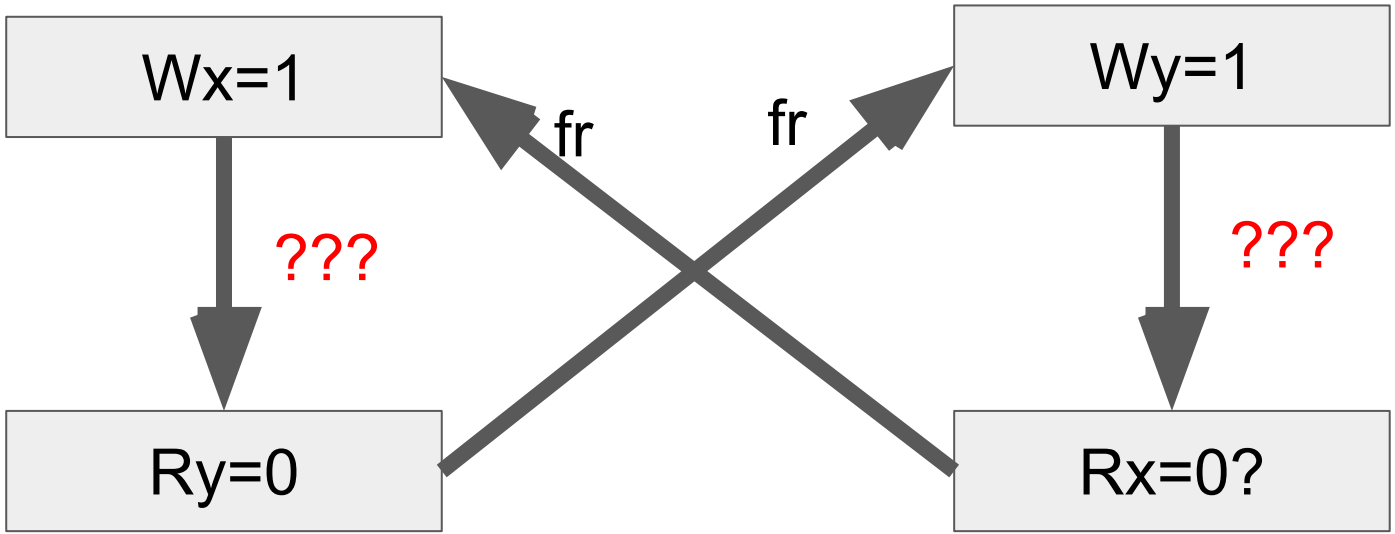
try_to_wake_up() magic #1

```
wake_up(...):
  try_to_wake_up():
```

```
schedule():
  rq_lock();
  if (->state)
    deactivate_task(); // block
    ->on_rq = 0;
  <...>
  smp_store_release(->on_cpu, 0);
  rq_unlock();
```

```
rq_lock()
if (->on_rq);
  ->state = TASK_RUNNING;
rq_unlock();
smp_load_cond_acquire(->on_cpu, 0)
```

Memory Model 101: Store Buffer



prepare_to_wait() magic #2

```
prepare_to_wait(...):
  smp_store_mb(->state, state):
    current->state = !TASK_RUNNING;
    smp_mb();

if (cond)
  break; // stop blocking
```

```
cond = 1;
wake_up(...):
  try_to_wake_up():
    <???>
    if (->state)
      goto out; // give up waking
    <continue to wake up>
```

Bug: Missing a smp_mb()

- Spotted at 2017
- Fixed by commit 35a2897c2a30 ("sched/wait: Remove the lockless swait_active() check in swake_up*()")

```
prepare_to_swait(...):
  raw_spin_lock_irqsave(...);
  list_add(&wait, &q);
  smp_store_mb(->state, state):
  raw_spin_unlock_irqsave(...);
if (cond)
  break; // stop blocking
```

```
cond = 1;
swake_up():
  if (swait_active()) // list_empty()
    raw_spin_lock_irqsave(...);
    swake_up_lock();
```

Kernel API for wait/wakeup

- wake_up*() and wait_event*()
- swake_up*() and swait_event*()
  - Bounded IRQ and lock hold time.
- swake_event_idle()
  - Do not contribute load to system
- complete() and wait_for_comletion()
  - Guarded by CROSS_RELEASE
- For more
  - "Much Ado About Blocking: Wait/Wake in the Linux Kernel" by Davidlohr Bueso.

Ordering implied by wait/wakeup

- No ordering outside the wait/wakeup subsystem
  - The wait and wakeup may not happen
- If a task is actually woken by another, the wakee is guaranteed to observe all the states of the waker before the wakeup
  - Program Order Guarantee
  - so do not put smp_*mb() between "cond=1" and "wake_up()" simply for wakee to observe @cond.

waketorture

- Proposed by Paul Mckenney
- Basic idea:
  - multiple tasks wait for/wake up each other
  - doing CPU online/offline in the same time
  - introduce jitters at host

```
struct wake_torture_ops {
    signed long (*wait)(signed long timeout);
    const char *name;
}

static int wake_torture_wait(void *arg); // nr_cpus threads
static int wake_torture_checker(void *arg);
static int wake_torture_onoff(void *arg);
```

waketorture

- Improvement
  - make it work ;-)
  - Dynamic wakeup topology

static int cond[...]; // the cond a thread is waiting
static int to_wake[...]; // the cond a thread is to wake up

Example:
cond: [0, 1, 2, 3]
to_wake: [1, 2, 3, 0]

A circular wait/wake topology.

waketorture

- Still WIP -- To detect the bugs we mention before.
- Could detect timer related wait/wake bug:
  - https://marc.info/?l=linux-sparc&m=150323406031064&w=2
- Need more real world scenarios of wait/wake bugs

Summarize

- Understand synchronization primitives via memory model
- Try to fix the section for wait/wake in memory-barriers.txt
- Feedback to waketorture

Q & A

Thanks!

LOCKDEP_CROSSRELEASE

```
mutex_lock(L1);

wait_for_completion(C1);
```

```
mutex_lock(L1);
<...>
mutex_unlock(L1);
complete(C1);
```

Some restartable sequence

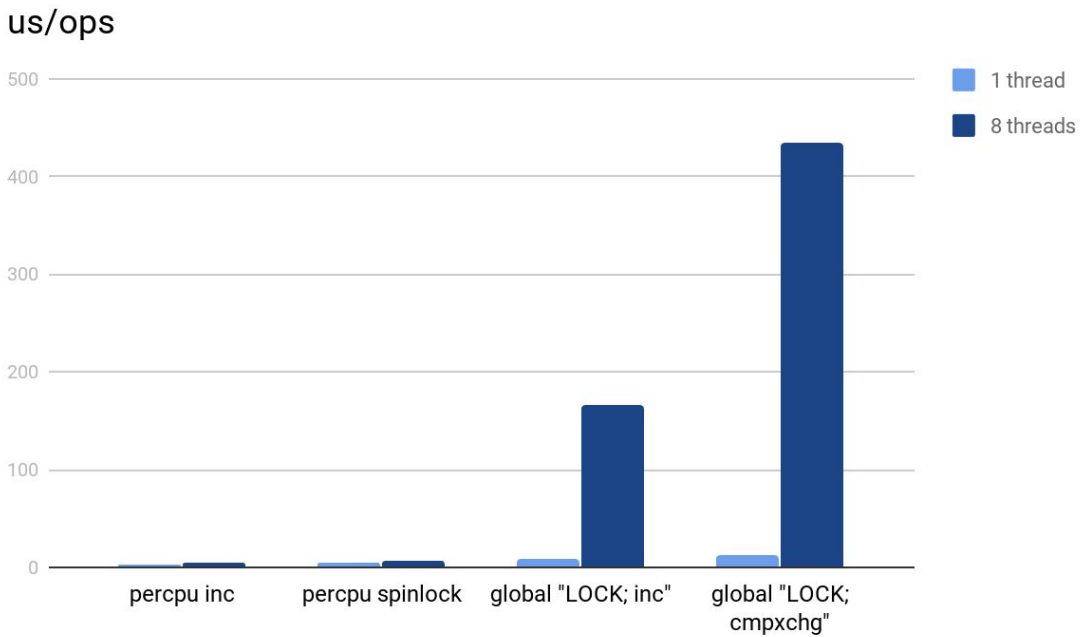
- Have you ever dreamed about userspace preemption disable?
- Restartable sequence(rseq)
  - per-cpu atomics
  - poor man's transactional memory
  - give a little bit power to userspace to run code without be worried with preemption.

Some restartable sequence

- userspace register an abi data structure via syscall
- set the (start_ip, post_commit_ip, abort_ip) to the data structure
- run some code at [start_ip, post_commit_ip)
- if a preemption happens in the middle, set the userspace ip to abort_ip
- the instruction before post_commit_ip indicating the finish of some critical section.

Some restartable sequence

Performance numbers(from Mathieu, on Xeon E5-2630)



Some restartable sequence

I do not hate this series, and I'd be happy to apply it, but I will repeat what I've asked for EVERY SINGLE TIME this series has come up:

I want to see real numbers from real issues.

-- Linus Torvalds

So help or trying out is welcome!