Maintaining a Real Time Stable Kernel
What’s different than a vanilla stable kernel?

Steven Rostedt
3/13/2018
Upstream Stable Releases

• After all mainline releases (Linus’s release)
  - A stable release is maintained while the next is developed
  - for example:
    • 4.13 - released 2017-09-03
    • 4.13.1 - released 2017-09-10
      - Follows once a week or when necessary
    • 4.13.12 - released 2017-11-08
    • 4.14 - released 2017-11-12
    • Four more 4.13 stable releases (4.13.13 - 4.13.16)
      - last one released 2017-11-24
    • 4.15-rc1 released 2017-11-26
    • No more 4.13 stable kernels are released
Upstream Stable Releases

• Long Term Kernel Releases
  - 4.14 - Released 2017-11-12, PEOL - Jan 2020
  - 4.9 - Released 2016-12-11, PEOL - Jan 2019
  - 4.4 - Released 2016-01-10, PEOL - Feb 2022
  - 4.1 - Released 2015-06-21, PEOL - May 2018
  - 3.16 - Released 2014-08-03, PEOL - Apr 2020
  - 3.2 - Released 2012-01-04, PEOL - May 2018
Real Time Stable Releases

• Development Branch
  - Follows ever other Mainline release (when possible)
  - Some times it will take three releases or one release
    • 4.1, 4.4, 4.6, 4.8, 4.9, 4.11, 4.13, 4.14
  - When 4.16 comes out, that will become the new Development Branch
  - Sebastian will take over 4.16
  - Steven will convert 4.14 into the next stable RT tree.

• This talk is only about the Stable RT Process
  - RT Development may not abide by this
Real Time Stable Releases

• Development Branch
  - 4.14 - Latest development, maintained by Sebastian Siewior

• Stable releases
  - 4.9 - Maintained by Julia Cartwright
  - 4.4 - Maintained by Daniel Wagner
  - 4.1 - Maintained by Julia Carwright
  - 3.18 (*) - Maintained by Tom Zanussi
  - 3.2 - Maintained by Steven Rostedt
    • No backports, just keeping with stable releases
Real Time Stable Process

• Keeping up with mainline stable
  – Each RT stable release maps to a long term stable release
  – Should sync up at least twice a month
  – A mainline stable merge into RT stable is done without other additions
    • No RT backports
    • Increments the appended “-rt” number
      – 4.9.43-rt33 - was last development release
      – 4.9.44-rt34 - was first “stable” RT release
      – 4.9.45-rt35 - Simple merge of 4.9.45

• Backports are done separately
  • You will see two -rt releases for the same release
    – 4.9.61-rt51 - Just merged 4.9.61
    – 4.9.61-rt52 - Backported patches from latest development RT release
Increment -rt with stable

- Originally, all new stable releases received a -rt tag
- `git merge v4.9.51`
- `vim localversion-rt`
  - change -rt40 to -rt41
- `git commit -a`
- `git tag -s v4.9.51-rt41`
- `git merge v4.9.52`
Increment -rt with stable

```
$ git tag |grep v4.9.5.
v4.9.50
v4.9.50-rt40
v4.9.51
v4.9.51-rt41
v4.9.52
v4.9.52-rt42
v4.9.53
v4.9.53-rt43
v4.9.54
v4.9.54-rt44
v4.9.55
v4.9.55-rt45
v4.9.56
v4.9.56-rt46
v4.9.57
v4.9.57-rt47
```
Increment -rt with stable (but not always?)

$ git tag |grep -e v4.9.6[89] -e v4.9.7.
  v4.9.68
  v4.9.68-rt60
  v4.9.68-rt60-rebase
  v4.9.69
  v4.9.70
  v4.9.71
  v4.9.72
  v4.9.73
  v4.9.74
  v4.9.75
  v4.9.76
  v4.9.76-rt61
  v4.9.76-rt61-rebase
  v4.9.77
  v4.9.78
  v4.9.79
Every stable or just when you update?

What happened?

- I was told to do it for every stable
- Julia did not see the rational for doing that
  - She only incremented the -rt for the latest stable she merged
  - If many stables were introduced, only tag the latest one you merged
- Why should we tag every stable?
  - Perhaps we don’t need to
  - But perhaps we should tag more than just the last one!
  - Compromise?
Increment -rt with stable (but not always?)

$ git tag |grep v4.9
[...]
v4.9.76-rt61
v4.9.76-rt61-rebase
v4.9.77
v4.9.78
v4.9.79
v4.9.80
v4.9.81
v4.9.82
v4.9.83
v4.9.84
v4.9.84-rt62
v4.9.84-rt62-rebase
Increment -rt with stable (but not always?)

$ git checkout v4.9-rt # currently at v4.9.76-rt61
$ git merge v4.9.84
CONFLICT (content): Merge conflict in kernel/workqueue.c
CONFLICT (content): Merge conflict in kernel/time/posix-timers.c
CONFLICT (content): Merge conflict in arch/x86/include/asm/thread_info.h

# Where did the conflict occur? What if we resolve it wrong?
Increment -rt with stable (but not always?)

```shell
$ git tag |grep v4.9
[...]
v4.9.76-rt61
v4.9.76-rt61-rebase
v4.9.77
v4.9.78  <- CONFLICT
v4.9.79
v4.9.80
v4.9.81  <- CONFLICT
v4.9.82  <- CONFLICT
v4.9.83
v4.9.84
v4.9.84-rt62
v4.9.84-rt62-rebase
```
Increment -rt with stable (What we agreed on)

$ git tag |grep v4.9
[...]  
v4.9.76-rt61  
v4.9.76-rt61-rebase  
v4.9.77  
v4.9.78-rt62 # resolve conflicts  
v4.9.79  
v4.9.80  
v4.9.81-rt63 # resolve conflicts  
v4.9.82-rt64 # resolve conflicts  
v4.9.83  
v4.9.84  
v4.9.84-rt65  
v4.9.84-rt65-rebase  

# Of course we wont change the current tags (but this is for the future)
Increment -rt with stable (Should we?)

$ git tag |grep v4.9
[...]
v4.9.76-rt61
v4.9.76-rt61-rebase
v4.9.77-rt62
v4.9.78-rt63 # resolve conflicts
v4.9.79
v4.9.80-rt64
v4.9.81-rt65 # resolve conflicts
v4.9.82-rt66 # resolve conflicts
v4.9.83
v4.9.84
v4.9.84-rt67
v4.9.84-rt67-rebase
Merging Mainline Stable into RT Stable

• If it has been a while, catch up first
  - Each stable release with conflicts gets its own -rt tag (or will in the future)
  - git merge v4.9.78
  - Solve conflicts
  - git commit
  - Increment “localversion-rt” file (-rt62)
  - git commit
  - git merge v4.9.80
  - etc

• When you catch up. Run the tests
  - This release will have a “rebase” (explained next)
The Rebase Branch

- After catching up to stable
- Run the tests, if it passes, then release
- To make a patch queue and tarball
  - Merging branches changes the original commits (because of conflicts)
  - Can't just cherry pick from the stable tree
  - Must keep the added patches at the end of the queue for the release
- v4.9-rt1 is equal to v4.9-rt1-rebase
- But let's look at v4.9.11-rt9
  - `git cherry v4.9.11 v4.9.11-rt9 > /tmp/list`
  - Convert the /tmp/list (of sha1s) into a quilt queue
  - Try to apply it!
From Sebastian Andrzej Siewior <bigeasy@linutronix.de>
Date: Wed, 14 Dec 2016 14:44:18 +0100
Subject: [PATCH] btrfs: drop trace_btrfs_all_work_done() from normal_work_helper()

For btrfs_scrubparity_helper() the ->func() is set to
scrub_parity_bio_endio_worker(). This function invokes invokes
scrub_free_parity() which kfree() the worked object. All is good as
long as trace events are not enabled because we boom with a backtrace
like this:

Workqueue: btrfs-endio btrfs_endio_helper
RIP: 0010:[<ffffffff812f81ae>] [<ffffffff812f81ae>] trace_event_raw_event_btrfs__work__done+0x4e/0xa0
Call Trace:
[<ffffffff8136497d>] btrfs_scrubparity_helper+0x59d/0x780
[<ffffffff81364c49>] btrfs_endio_helper+0x9/0x10
[<ffffffff810af8e>] process_one_work+0x26e/0x7b0
[<ffffffff810b516>] worker_thread+0x46/0x560
[<ffffffff81091c4>] kthread+0xee/0x110
[<ffffffff810e166>] ret_from_fork+0x2a/0x40

So in order to avoid this, I remove the trace point.

Signed-off-by: Sebastian Andrzej Siewior <bigeasy@linutronix.de>
---
fs/btrfs/async-thread.c | 2 --
1 file changed, 2 deletions(-)
diff --git a/fs/btrfs/async-thread.c b/fs/btrfs/async-thread.c
index e0f071f6b5a7..d0dfc3d2e199 100644
--- a/fs/btrfs/async-thread.c
+++ b/fs/btrfs/async-thread.c
@@ -318,8 +318,6 @@ static void normal_work_helper(struct btrfs_work *work)
     run_ordered_work(wq);
 }
-
-if (!need_order)
-    trace_btrfs_all_work_done(work);
}

void btrfs_init_work(struct btrfs_work *work, btrfs_work_func_t uniq_func,
diff --git a/fs/btrfs/async-thread.c b/fs/btrfs/async-thread.c
index e0f071f6b5a7..d0dfc3d2e199 100644
--- a/fs/btrfs/async-thread.c
+++ b/fs/btrfs/async-thread.c
@@ -318,8 +318,6 @@ static void normal_work_helper(struct btrfs_work *work)
    set_bit(WORK_DONE_BIT, &work->flags);
    run_ordered_work(wq);
 }
-
 if (!need_order)
-    trace_btrfs_all_work_done(work);
-
 void btrfs_init_work(struct btrfs_work *work, btrfs_work_func_t uniq_func, --
static void normal_work_helper(struct btrfs_work *work)
    run_ordered_work(wq);
}
if (!need_order)
-     trace_btrfs_all_work_done(work);
+     trace_btrfs_all_work_done(wq->fs_info, wtag);
}

void btrfs_init_work(struct btrfs_work *work, btrfs_work_func_t uniq_func,
From the Mainline Stable

@@ -333,7 +340,7 @@ static void normal_work_helper(struct btrfs_work *work)
       run_ordered_work(wq);
   }
   if (!need_order)
-      trace_btrfs_all_work_done(work);
+      trace_btrfs_all_work_done(wq->fs_info, wtag);
   }

void btrfs_init_work(struct btrfs_work *work, btrfs_work_func_t uniq_func,

@@ -318,8 +318,6 @@ static void normal_work_helper(struct btrfs_work *work)
       run_ordered_work(wq);
   }
   if (!need_order)
-      trace_btrfs_all_work_done(work);
-  }

void btrfs_init_work(struct btrfs_work *work, btrfs_work_func_t uniq_func,
$ quilt push -a
[...]
Applying patch patches/0001-locking-percpu-rwsem-use-swait-for-the-wating-writer.patch
patching file include/linux/percpu-rwsem.h
patching file kernel/locking/percpu-rwsem.c

Applying patch patches/0001-btrfs-drop-trace_btrfs_all_work_done-from-normal_wor.patch
patching file fs/btrfs/async-thread.c
Hunk #1 FAILED at 318.
1 out of 1 hunk FAILED -- rejects in file fs/btrfs/async-thread.c
Patch patches/0001-btrfs-drop-trace_btrfs_all_work_done-from-normal_wor.patch can be reverse-applied
The Rebase Branch

- `git checkout v4.9-rt`
- `git rebase -i v4.9.11`
  - Make sure you are on the right branch!
- Fix all conflicts
- Tag with "-rebase" appended
- Make sure v4.9.11-rt9 is the same as v4.9.11-rt9-rebase
  - `git diff v4.9.11-rt9 v4.9.11-rt9-rebase`
- Create the patch tarball from the rebase branch
RT overview
What makes Linux into an RTOS?

• Spin locks into mutex
  - Allow to preempt (sleep) in “spin lock” critical sections
  - raw_spin_locks are still spin locks

• Threaded interrupts
  - Interrupt handlers contain “spin locks”

• Priority Inheritance
  - All sleeping locks are given Priority Inheritance
### Priority Inversion

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>blocked</td>
</tr>
<tr>
<td>B</td>
<td>preempted</td>
<td>preempted</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Priority Inheritance

<table>
<thead>
<tr>
<th></th>
<th>blocked</th>
<th>sleeps</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>wakes up</td>
</tr>
<tr>
<td>C</td>
<td>preempted</td>
<td>releases lock</td>
</tr>
</tbody>
</table>
Priority Inheritance is Hard

- Can get really complex
- Can not handle Multiple owners (think reader writer locks)
- Do one of three (or two)
  - Make readers serialize (turn multi owner into single owner lock)
  - Do not convert rwlocks to use priority inheritance (buyer beware)
  - Implement Multiple owner Priority Inheritance algorithm
    - Becomes exponentially more complex
    - Lots of race conditions
    - But hey! It *can* be done
Disabling Preemption

- Preemption is disabled by spin locks
  - PREEMPT_RT lets spin locks sleep, and keeps preemption enabled
- Per CPU critical sections
  - Preemption is disabled for modifying local data to the CPU
  - RT adds “local_locks”
    - Keeps preemption enabled
    - Adds a mutex to the critical section
    - Disables migration (task stays on CPU)
- Watch out for “local_locks” in modified stable code
  - New ones may be needed
Disabling Interrupts

- If per CPU data shared with interrupts is performed
- Interrupts may be disabled
- PREEMPT_RT runs interrupt handlers as threads
- `local_lock` is good enough
- Either have “`local_lock_irqsave()`” or `local_irq_disable_nort()`
  - `local_irq_disable_nort()` only disables interrupts when PREEMPT_RT is not configured
Dealing with Conflicts (during stable merges)

- Merge of mainline stable release into RT stable
  - git merge v4.9.35
- Conflict occurs (This is good, we know a problem exists)
  - No conflicts does not mean a good merge
  - Remember, preempt_disable and local_irq_save may have been added
- Here’s what to do
Dealing with Conflicts (Easy)

$ git merge v4.9.35
Auto-merging kernel/time/timekeeping.c
Auto-merging kernel/signal.c
CONFLICT (content): Merge conflict in kernel/signal.c
Auto-merging fs/exec.c
Auto-merging drivers/usb/gadget/function/f_fs.c
Automatic merge failed; fix conflicts and then commit the result.
$ git dif

diff --cc kernel/signal.c
index c88464795f7,deb04d5983ed..000000000000
--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -583,7 -526,13 +584,17 @@ static void collect_signal(int sig, str
still_pending:
  list_del_init(&first->list);
  copy_siginfo(info, &first->info);
+<<<<< HEAD
+sigqueue_free_current(first);
++++++
++
+  *resched_timer =
++  (first->flags & SIGQUEUE_PREALLOC) &&
++  (info->si_code == SI_TIMER) &&
++  (info->si_sys_private);
++
++  __sigqueue_free(first);
++++>>>> v4.9.35
)+ else { /*
+  * Ok, it wasn't in the queue. This must be
+  * v4.9.10 -561,10 -565,9 +627,11 @@ static int __dequeue_signal(struct sig
+  *int dequeue_signal(struct task_struct *tsk, sigset_t *mask, siginfo_t *info)
+  { +     bool resched_timer = false;
+     int signs;
+     WARN_ON_ONCE(tsk != current);
+     /* We only dequeue private signals from ourselves, we don't let
+     * signalfd steal them
+     */
+     
+     int mask  = 0;
+     siginfo_t info;
Dealing with Conflicts (Easy)

--- a/kernel/signal.c
+++ b/kernel/signal.c
@@ -583,7 -526,13 +584,17 @@@
 static void collect_signal(int sig, struct task_struct *tsk, struct siginfo *info, str
 still_pending:
    list_del_init(&first->list);
    copy_siginfo(info, &first->info);
+              sigqueue_free_current(first);
+              *resched_timer =
+               (first->flags & SIGQUEUE_PREALLOC) &&
+               (info->si_code == SI_TIMER) &&
+               (info->si_sys_private);
+              __sigqueue_free(first);
+>>>>>>> v4.9.35
 else {
    /* Ok, it wasn't in the queue. This must be
@@ -616,10 -565,9 +627,11 @@@
 static int __dequeue_signal(struct task_struct *tsk, sigset_t *mask, siginfo_t *info)
 {+
    bool resched_timer = false;
    int signr;
+              WARN_ON_ONCE(tsk != current);
+              */
    /* We only dequeue private signals from ourselves, we don't let
+       signalfd steal them
+*/
Dealing with Conflicts (Easy)

```
$ git diff v4.9.34 v4.9.34-rt24 kernel/signal.c
@@ -525,7 +583,7 @@ static void collect_signal(int sig, struct sigpending *list, siginfo_t *info)
     still_pending:
         list_del_init(&first->list);
         copy_siginfo(info, &first->info);
-        __sigqueue_free(first);
+        sigqueue_free_current(first);
     } else {
         /*
           * Ok, it wasn't in the queue. This must be
@@ -560,6 +618,8 @@ int dequeue_signal(struct task_struct *tsk, sigset_t *mask, siginfo_t *info)
     } int signr;
+        WARN_ON_ONCE(tsk != current);
+        /* We only dequeue private signals from ourselves, we don't let
           * signalfd steal them
           */
```
Dealing with Conflicts (Easy)

```
$ git diff v4.9.34 v4.9.35 kernel/signal.c
-static void collect_signal(int sig, struct sigpending *list, siginfo_t *info)
+static void collect_signal(int sig, struct sigpending *list, siginfo_t *info,
 +    bool *resched_timer)
 {
 struct sigqueue *q, *first = NULL;

@@ -525,6 +526,12 @@ static void collect_signal(int sig, struct sigpending *list, siginfo_t *info)
      still_pending:
       list_del_init(&first->list);
       copy_siginfo(info, &first->info);
-      
+      *resched_timer =
+      (first->flags & SIQUEUE_PREALLOC) &&
+      (info->si_code == SI_TIMER) &&
+      (info->si_sys_private);
+      __sigqueue_free(first);
 } else {
  /*
 @@ -558,15 +565,16 @@ static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
      */
 int dequeue_signal(struct task_struct *tsk, sigset_t *mask, siginfo_t *info)
 {  
+    bool resched_timer = false;
     int signr;
     /* We only dequeue private signals from ourselves, we don't let
      * signalfd steal them
      */
-    signr = __dequeue_signal(&tsk->pending, mask, info);
+    signr = __dequeue_signal(&tsk->pending, mask, info, &resched_timer);
     if (!signr) {
         signr = __dequeue_signal(&tsk->signal->shared_pending,
```
Dealing with Conflicts (Easy)

$ git diff v4.9.34 v4.9.35-rt25 kernel/signal.c
- static void collect_signal(int sig, struct sigpending *list, siginfo_t *info)
+ static void collect_signal(int sig, struct sigpending *list, siginfo_t *info,
  + bool *resched_timer)
{
  struct sigqueue *q, *first = NULL;
@@ -525,7 +584,13 @@ static void collect_signal(int sig, struct sigpending *list, siginfo_t *info)
 still_pending:
                     list_del_init(&first->list);
                     copy_siginfo(info, &first->info);
-                   __sigqueue_free(first);
+                   ...
+                   *resched_timer =
+                     (first->flags & SIGQUEUE_PREALLOC) &&
+                     (info->si_code == SI_TIMER) &&
+                     (info->si_sys_private);
+                   sigqueue_free_current(first);
  } else {
    /*
      Ok, it wasn't in the queue. This must be
@@ -558,15 +623,18 @@ static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
 int dequeue_signal(struct task_struct *tsk, sigset_t *mask, siginfo_t *info)
{
  bool resched_timer = false;
  int signr;
+  WARN_ON_ONCE(tsk != current);
+  /* We only dequeue private signals from ourselves, we don't let
+   * signalfd steal them
+  */
-  signr = __dequeue_signal(&tsk->pending, mask, info);
+  signr = __dequeue_signal(&tsk->pending, mask, info, &resched_timer);


Dealing with Conflicts (non-trivial)

```bash
$ git merge v4.9.61
Auto-merging fs/namei.c
Auto-merging drivers/pci/access.c
CONFLICT (content): Merge conflict in drivers/pci/access.c
Automatic merge failed; fix conflicts and then commit the result.
$ git diff
diff --cc drivers/pci/access.c
index 223bbb9acb03,7b5cf6d1181a..000000000000
--- a/drivers/pci/access.c
+++ b/drivers/pci/access.c
@@ -672,8 -672,9 +672,13 @@@
   void pci_cfg_access_unlock(struct pci_d
   WARN_ON(!dev->block_cfg_access);
         dev->block_cfg_access = 0;
     +     + void pci_cfg_access_unlock(struct pci_d
     +     +     + WARN_ON(!dev->block_cfg_access);
     +     +       + warn_all_locked(&pci_cfg_wait);
     +     +       + ++++++++ v4.9.61
     +     +       + raw_spin_unlock_irqrestore(&pci_lock, flags);
     +     +       + +)
     +     } +
   +     +     + wake_up_all(&pci_cfg_wait);
     +     + EXPORT_SYMBOL_GPL(pci_cfg_access_unlock);
```
Dealing with Conflicts (non-trivial)

$ git diff v4.9.61 v4.9.61-rt50 drivers/pci/access.c
@@ -672,7 +672,7 @@ void pci_cfg_access_unlock(struct pci_dev *dev)
        WARN_ON(!dev->block_cfg_access);

        dev->block_cfg_access = 0;
-       wake_up_all(&pci_cfg_wait);
+       wake_up_all_locked(&pci_cfg_wait);
        raw_spin_unlock_irqrestore(&pci_lock, flags);
    }
EXPORT_SYMBOL_GPL(pci_cfg_access_unlock);
Dealing with Conflicts (non-trivial)

```
$ git diff v4.9.61 v4.9.62 drivers/pci/access.c
@@ -672,8 +672,9 @@ void pci_cfg_access_unlock(struct pci_dev *dev)
               WARN_ON(!dev->block_cfg_access);

               dev->block_cfg_access = 0;
-              wake_up_all(&pci_cfg_wait);
              raw_spin_unlock_irqrestore(&pci_lock, flags);
+              wake_up_all(&pci_cfg_wait);
+              wake_up_all(&pci_cfg_wait);
 }
 EXPORT_SYMBOL_GPL(pci_cfg_access_unlock);
```
Dealing with Conflicts (non-trivial)

Obvious answer:

```bash
$ git diff v4.9.61 v4.9.62 drivers/pci/access.c
@@ -672,8 +672,9 @@ void pci_cfg_access_unlock(struct pci_dev *dev)
    WARN_ON(!dev->block_cfg_access);

    dev->block_cfg_access = 0;
-   wake_up_all(&pci_cfg_wait);
+    raw_spin_unlock_irqrestore(&pci_lock, flags);
+   wake_up_all_locked(&pci_cfg_wait);
}
EXPORT_SYMBOL_GPL(pci_cfg_access_unlock);
```
Dealing with Conflicts (non-trivial)

Obvious answer:

WRONG!

```bash
$ git diff v4.9.61 v4.9.62 drivers/pci/access.c
@@ -672,8 +672,9 @@ void pci_cfg_access_unlock(struct pci_dev *dev)
         WARN_ON(!dev->block_cfg_access);

         dev->block_cfg_access = 0;
-        wake_up_all(&pci_cfg_wait);
+        raw_spin_unlock_irqrestore(&pci_lock, flags);
+        wake_up_all_locked(&pci_cfg_wait);
 }
 EXPORT_SYMBOL_GPL(pci_cfg_access_unlock);
```
wake_up_all_locked?

• In mainline, but nice for RT
• If the wait queue is protected by a lock
  – No need to grab the wait queue lock
• Mainline uses this for optimization
  – Why grab another lock if one is already taken?
• RT uses this for wait queues in raw_spin_locks
wake_up_all_locked usage

```c
void pci_cfg_access_unlock(struct pci_dev *dev)
{
    unsigned long flags;

    raw_spin_lock_irqsave(&pci_lock, flags);

    /* This indicates a problem in the caller, but we don't need
     * to kill them, unlike a double-block above. */
    WARN_ON(!dev->block_cfg_access);

    dev->block_cfg_access = 0;
    wake_up_all_locked(&pci_cfg_wait);
    raw_spin_unlock_irqrestore(&pci_lock, flags);
}
```
Dealing with Conflicts (non-trivial)

Correct answer:

```bash
$ git diff v4.9.61 v4.9.62 drivers/pci/access.c
@@ -672,8 +672,9 @@ void pci_cfg_access_unlock(struct pci_dev *dev)
    WARN_ON(!dev->block_cfg_access);

    dev->block_cfg_access = 0;
-   wake_up_all(&pci_cfg_wait);
+   raw_spin_unlock_irqrestore(&pci_lock, flags);
+   wake_up_all(&pci_cfg_wait);
}
EXPORT_SYMBOL_GPL(pci_cfg_access_unlock);
```
Dealing with No Conflicts (Hard)

$ git merge v4.9.66
Auto-merging kernel/sched/sched.h
Auto-merging kernel/sched/rt.c
CONFLICT (content): Merge conflict in kernel/sched/rt.c
[...
$ git diff
diff --cc kernel/sched/rt.c
index b0691f4e7d49,9c131168d933..000000000000
--- a/kernel/sched/rt.c
+++ b/kernel/sched/rt.c
@@@ -96,14 -91,6 +92,17 @@ void init_rt_rq(struct rt_rq *rt_rq
   rt_rq->rt_nr_migratory = 0;
   rt_rq->overloaded = 0;
   plist_head_init(&rt_rq->pushable_tasks);
+<<<<<<<<< HEAD
+ +#ifdef HAVE_RT_PUSH_IPI
+    rt_rq->push_flags = 0;
+    rt_rq->push_cpu = nr_cpu_ids;
+    raw_spin_lock_init(&rt_rq->push_lock);
+    init_irq_work(&rt_rq->push_work, push_irq_work_func);
+    rt_rq->push_work.flags |= IRQ_WORK_HARD_IRQ;
+  +#endif
+>>>>>> v4.9.66
  #endif /* CONFIG_SMP */
Dealing with No Conflicts (hard)

$ git diff v4.9.65 v4.9.66 kernel/sched/
--- a/kernel/sched/core.c
+++ b/kernel/sched/core.c
@@ -5878,6 +5877,12 @@ static int init_rootdomain(struct root_domain *rd)
     if (!zalloc_cpumask_var(&rd->rto_mask, GFP_KERNEL))
         goto free_dlo_mask;
+
+#ifdef HAVE_RT_PUSH_IPI
+    rd->rto_cpu = -1;
+    raw_spin_lock_init(&rd->rto_lock);
+    init_irq_work(&rd->rto_push_work, rto_push_irq_work_func);
+#endif
+
 init_dl_bw(&rd->dl_bw);
     if (cpudl_init(&rd->cpudl) != 0)
         goto free_dlo_mask;
--- a/kernel/sched/rt.c
+++ b/kernel/sched/rt.c
@@ -95,13 +91,6 @@ void init_rt_rq(struct rt_rq *rt_rq)
     rt_rq->rt_nr_migratory = 0;
     rt_rq->overloaded = 0;
     plist_head_init(&rt_rq->pushable_tasks);
-
-#ifdef HAVE_RT_PUSH_IPI
-    rt_rq->push_flags = 0;
-    rt_rq->push_cpu = nr_cpu_ids;
-    raw_spin_lock_init(&rt_rq->push_lock);
-    init_irq_work(&rt_rq->push_work, push_irq_work_func);
-#endif

 #endif /* CONFIG_SMP */
/* We start is dequeued state, because no RT tasks are queued */
rt_rq->rt_queued = 0;
Dealing with No Conflicts (hard)

```bash
$ git diff v4.9.65 v4.9.65-rt57 kernel/sched/
--- a/kernel/sched/rt.c
+++ b/kernel/sched/rt.c
@@ -101,6 +102,7 @@ void init_rt_rq(struct rt_rq *rt_rq)
     rt_rq->push_cpu = nr_cpu_ids;
     raw_spin_lock_init(&rt_rq->push_lock);
     init_irq_work(&rt_rq->push_work, push_irq_work_func);
+   rt_rq->push_work.flags |= IRQ_WORK_HARD_IRQ;
@endif
@endif /* CONFIG_SMP */
/* We start is dequeued state, because no RT tasks are queued */
```
Why was that Hard?

- It wasn’t, because there was a conflict
- This patch was backported to 3.18-rt (for RHEL)
- It introduced the irq_work
- That irq_work required adding the IRQ_WORK_HARD_IRQ flag
- If not, the system would not behave properly (but would still boot!)
- How do you catch these?
  - You need to know how PREEMPT_RT works!
Backporting RT patches

- Stable is the easy part
- When a new development RT comes out
  - Need to look at what was added
  - Must haves will be tagged: Cc: stable-rt@vger.kernel.org
  - But there may be some that are required
  - If not the next stable, even “stable-rt” commits may not be applicable
## Backporting RT patches

<table>
<thead>
<tr>
<th></th>
<th>Patches</th>
<th>Upstream commit</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>4.9</td>
<td>4.4</td>
<td>4.1</td>
<td>3.18</td>
</tr>
<tr>
<td>2</td>
<td>Revert &quot;ts. jbd: pull your plug when waiting for space&quot;</td>
<td>3b5d23e6687a</td>
<td>N/A</td>
<td>N/A</td>
<td>Review</td>
<td>Review</td>
</tr>
<tr>
<td>3</td>
<td>timer/hrtimer: check property for a running timer</td>
<td>f2249cc4b1f6</td>
<td>Original</td>
<td>Applied</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>rtmutex: Make lock_killable work</td>
<td>9edcb2cd71f</td>
<td>Original</td>
<td>Applied</td>
<td>Applied</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>random: avoid preempt_disable(jud section)</td>
<td>21ca6915e696</td>
<td>Original</td>
<td>Applied</td>
<td>Applied</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>sched: Prevent task state corruption by spurious lock wakeup</td>
<td>2f9624e150e8</td>
<td>Original</td>
<td>Applied</td>
<td>Applied</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>sched/migrate disable: handle updated task-mask mg-dis</td>
<td>1dcb9be37874</td>
<td>Original</td>
<td>Applied</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>8</td>
<td>kernel/locking: use an exclusive wait_q for sleepers</td>
<td>2d8a1c72a3780</td>
<td>Original</td>
<td>Applied</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>fs: convert two more BH_Update=Lock related bitspinlocks</td>
<td>a043e49ad549</td>
<td>Applied</td>
<td>Applied</td>
<td>Applied</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>md/raid: do not disable interrupts</td>
<td>4b8ad0200c0</td>
<td>Applied</td>
<td>Applied</td>
<td>Applied</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Revert &quot;fs. jbd2: pull your plug when waiting for space&quot;</td>
<td>95bc2c7b2c0</td>
<td>Applied</td>
<td>Applied</td>
<td>Applied</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>cpu_pm: replace raw_nodtler to atomic_nodtler</td>
<td>f0f38a50ac4c</td>
<td>Applied</td>
<td>Applied</td>
<td>Applied</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>kernel/hrtimer: migrate deferred timer on CPU down</td>
<td>b3c66bfddcdd</td>
<td>Applied</td>
<td>Applied</td>
<td>Applied</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>kernel/hrtimer: don't wakeup a process while holding the</td>
<td>ca577cb1bba1</td>
<td>Applied</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>kernel/hrtimer/hotplug: don't wake ktimersoftd while holding</td>
<td>f53c48477839</td>
<td>Applied</td>
<td>Applied</td>
<td>Applied</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Bluetooth: avoid recursive locking in hci_send_to_channel()</td>
<td>a785161d39be</td>
<td>Applied</td>
<td>N/A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>rt/locking: allow recursive local_trylock()</td>
<td>40df995cda2a</td>
<td>Applied</td>
<td>Applied</td>
<td>Applied</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>net: use trylock in lomp_sk</td>
<td>80f3b603c32c</td>
<td>Applied</td>
<td>Applied</td>
<td>Applied</td>
<td></td>
</tr>
</tbody>
</table>
Backporting RT patches

- Have a branch with last rt-devel pull (let’s say v4.14.15-rt13)
- `git checkout -b temp last_rt_devel (branch)`
- Want to update to latest rt-devel (let’s say v4.14.24-rt19)
- `git merge v4.14.24 (make it match the same mainline stable release)`
  - Force fixing of commits (really doesn’t matter here)
- `git cherry -v --abbrev=12 HEAD v4.14.24-rt19`
- Examine the commits that are new
$ git cherry -v --abbrev=12 HEAD v4.14.24-rt19
+ 56e250a796bc v4.14.18-rt14
+ 4eb01a8a3d90 brd: remove unused brd_mutex
+ 0ab208e3c1d2 tracing: Update the "tracing: Inter-event (e.g. latency) support" patch
+ 51331ec00036 v4.14.18-rt15
+ 8f7c88961b2e v4.14.20-rt16
+ 96867e66ef1d RCU: skip the "schedule() in RCU section" warning on UP, too
+ d3a66fffd1c4f net: use task_struct instead of CPU number as the queue owner on -RT
+ f5a5c5e7d006 v4.14.20-rt17
+ 79d03355044c v4.14.20-rt18
+ 932c5783d443 Revert "rt,ntp: Move call to schedule_delayed_work() to helper thread"
+ ee6c0574c45a v4.14.24-rt19
$ git cherry -v --abbrev=12 HEAD v4.14.24-rt19
+ 56e250a796bc v4.14.18-rt14
+ 4eb01a8a3d90 brd: remove unused brd_mutex
+ 0ab208e3c1d2 tracing: Update the "tracing: Inter-event (e.g. latency) support" patch
+ 51331ec00036 v4.14.18-rt15
+ 8f7c88961b2e v4.14.20-rt16
+ 96867e66ef1d RCU: skip the "schedule() in RCU section" warning on UP, too
+ d3a66ff6d1c4f net: use task_struct instead of CPU number as the queue owner on -RT
+ f5a5c5e7d006 v4.14.20-rt17
+ 79d03355044c v4.14.20-rt18
+ 932c5783d443 Revert "rt,ntp: Move call to schedule_delayed_work() to helper thread"
+ ee6c0574c45a v4.14.24-rt19
Thank You