DEPT Dependency Tracker

What Lockdep does and doesn’t

2022.9.13

Byungchul Park | LG Electronics

max.byungchul.park@gmail.com
<table>
<thead>
<tr>
<th>Date</th>
<th>Commit message</th>
<th>Author</th>
<th>Files</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020-01-24</td>
<td>rcu: Add basic support for kfree_skb batching</td>
<td>Byungchul Park</td>
<td>4</td>
<td>-6/+206</td>
</tr>
<tr>
<td>2019-08-01</td>
<td>rcu: Change return type of rcu_spawn_one_boost_kthread()</td>
<td>Byungchul Park</td>
<td>1</td>
<td>-9/+11</td>
</tr>
<tr>
<td>2018-08-30</td>
<td>rcu: Refactor rnu_irq(_enter/exit)</td>
<td>Byungchul Park</td>
<td>2</td>
<td>-22/+44</td>
</tr>
<tr>
<td>2018-07-12</td>
<td>rcu: Combine rnu_irq() and rnu_irq_cpu() into one function</td>
<td>Byungchul Park</td>
<td>3</td>
<td>-7/+5</td>
</tr>
<tr>
<td>2018-07-12</td>
<td>src: Move locking_for_file_stat to rnu</td>
<td>Byungchul Park</td>
<td>2</td>
<td>-12/+5</td>
</tr>
<tr>
<td>2018-05-15</td>
<td>rcu: Call wake_noob_leader_defer() with 'FORCE' when noob_a_count is high</td>
<td>Byungchul Park</td>
<td>1</td>
<td>-1/+1</td>
</tr>
<tr>
<td>2018-05-15</td>
<td>rcu: Inline rcu_preempt_do_callback() into its sole caller</td>
<td>Byungchul Park</td>
<td>1</td>
<td>-1/+1</td>
</tr>
<tr>
<td>2018-05-20</td>
<td>src: Remove dead code from rnu_get_tcb()</td>
<td>Byungchul Park</td>
<td>1</td>
<td>-1/+1</td>
</tr>
<tr>
<td>2017-11-14</td>
<td>iq: Add support for initializing complements with lockdep.map</td>
<td>Byungchul Park</td>
<td>4</td>
<td>-11/+23</td>
</tr>
<tr>
<td>2017-10-26</td>
<td>block: Add support for initializing complements with lockdep_map</td>
<td>Byungchul Park</td>
<td>3</td>
<td>-11/+23</td>
</tr>
<tr>
<td>2017-10-25</td>
<td>sched/completions: Add support for initializing completions with lockdep_map</td>
<td>Byungchul Park</td>
<td>1</td>
<td>-1/+1</td>
</tr>
<tr>
<td>2017-10-23</td>
<td>locking/lockdep: Introduce CONFIG_BOOTPARAM_LOCKDEP_CROSSRELEASE_FULLSTACK=y</td>
<td>Byungchul Park</td>
<td>2</td>
<td>0/+19</td>
</tr>
<tr>
<td>2017-10-23</td>
<td>locking/lockdep: Remove the BROKEN flag from CONFIG_LOCKDEP_CROSSRELEASE and ...</td>
<td>Byungchul Park</td>
<td>2</td>
<td>-2/+2</td>
</tr>
<tr>
<td>2017-10-23</td>
<td>locking/lockdep: Add a boot parameter allowing unwind in cross-release and di...</td>
<td>Byungchul Park</td>
<td>2</td>
<td>-2/+2</td>
</tr>
<tr>
<td>2017-10-23</td>
<td>locking/lockdep: sched/completions: Change the prefix of lock name for comple...</td>
<td>Byungchul Park</td>
<td>2</td>
<td>-2/+2</td>
</tr>
<tr>
<td>2017-10-23</td>
<td>locking/lockdep: Provide empty lockdep_map structure for CONFIG_LOCKDEP</td>
<td>Byungchul Park</td>
<td>2</td>
<td>-2/+2</td>
</tr>
<tr>
<td>2017-10-09</td>
<td>mm/malloc.c: Don't reinit the wheel but use existing liist API</td>
<td>Byungchul Park</td>
<td>1</td>
<td>-0/+14</td>
</tr>
<tr>
<td>2017-10-09</td>
<td>bracha: Don't reinit the wheel but use existing liist API</td>
<td>Byungchul Park</td>
<td>1</td>
<td>-0/+14</td>
</tr>
<tr>
<td>2017-10-08</td>
<td>fcout: Don't reinit the wheel but use existing liist API</td>
<td>Byungchul Park</td>
<td>1</td>
<td>-0/+14</td>
</tr>
<tr>
<td>2017-10-08</td>
<td>namespace: Don't reinit the wheel but use existing liist API</td>
<td>Byungchul Park</td>
<td>1</td>
<td>-0/+14</td>
</tr>
<tr>
<td>2017-09-17</td>
<td>locking/lockdep: Rename CONFIG_LOCKDEP_COMPLETE to CONFIG_LOCKDEP_COMPLETIONS</td>
<td>Byungchul Park</td>
<td>2</td>
<td>-6/+6</td>
</tr>
<tr>
<td>2017-09-17</td>
<td>locking/lockdep: Reword title of LOCKDEP_CROSSRELEASE config</td>
<td>Byungchul Park</td>
<td>1</td>
<td>-1/+1</td>
</tr>
<tr>
<td>2017-09-17</td>
<td>locking/lockdep: Make CONFIG_LOCKDEP_CROSSRELEASE part of CONFIG_PROVELOCKING</td>
<td>Byungchul Park</td>
<td>1</td>
<td>-1/+1</td>
</tr>
<tr>
<td>2017-09-14</td>
<td>locking/lockdep: Fix the rollback and overwrite detection logic in crossrelease</td>
<td>Byungchul Park</td>
<td>1</td>
<td>-1/+1</td>
</tr>
<tr>
<td>2017-09-14</td>
<td>locking/lockdep: Add a comment about crossrelease_hist_end() in lockdep_sys_e...</td>
<td>Byungchul Park</td>
<td>1</td>
<td>-1/+1</td>
</tr>
</tbody>
</table>

Interest: Task Scheduler, RCU, Locking

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/log/?qt=author&q=byungchul
Lock usage correctness
  Lock / unlock balance
  Relation between lock types
  Nest lock usage

RCU usage correctness
  Usage from illegal states
  Proper use of the APIs
  Delimitation for read section

Lock dependency correctness
  Lock acquisition order
  Lock recursive acquisitions
  Irq safe usage
**Lock usage correctness**

- Lock / unlock balance
- Relation between lock types
- Nest lock usage

**RCU usage correctness**

- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

**Lock dependency correctness**

- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage
Lock usage correctness
- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness
- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness
- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

// Holds A.

spin_lock A

// Terminates the task without releasing A.
exit

Lock and unlock must be used in pairs.

// Terminates the task without releasing A.
exit
// Holds A.
spin_lock A

// Terminates the task without releasing A.
exit
// Holds A.
spin_lock A

// Terminates the task without releasing A.
exit
Holds A.

spin_lock A

// Returns to user without releasing A.

return to user
// Holds A.
spin_lock A

// Returns to user without releasing A.
return to user
Lock usage correctness
- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness
- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness
- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

Relation between lock types
- mutex_lock shouldn’t be acquired in a spin_lock critical section.

// Holds spin lock A.
spin_lock A

// Holds mutex lock B.
mutex_lock B
mutex_unlock B
spin_unlock A
// Holds spin lock A.
spin_lock A

// Holds mutex lock B.
mutex_lock B

mutex_unlock B
spin_unlock A
// Holds spin lock A.
spin_lock A
...

// Holds mutex lock B.
mutex_lock B
...

mutex_unlock B
spin_unlock A
// Holds raw spin lock A.
raw_spin_lock A

// Holds spin lock B.
spin_lock B
spin_unlock B
raw_spin_unlock A

Relation between lock types
spin_lock shouldn’t be acquired in a raw_spin_lock critical section.
Lock usage correctness
- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness
- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness
- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

// Holds raw spin lock A.
raw_spin_lock A

... 

// Holds spin lock B.
spin_lock B

... 

spin_unlock B
raw_spin_unlock A
// Holds raw spin lock A.
raw_spin_lock A

…

// Holds spin lock B
spin_lock B

…

spin_unlock B
raw_spin_unlock A

---

**Lock usage correctness**
- Lock / unlock balance
- Relation between lock types
- Nest lock usage

**RCU usage correctness**
- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

**Lock dependency correctness**
- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage
Lock usage correctness
- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness
- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness
- Lock acquisition order
- Lock recursive acquisitions
- Irc safe usage

Nest lock usage
- Nest lock API should be called with the nest lock held.
- spin_lock A
- spin_lock_nest B, n=A
- spin_unlock B
- spin_unlock A
// Holds A.
spin_lock A

... // Holds B with nest lock A.
spin_lock_nest B, n=A

... spin_unlock B
spin_unlock A
// Holds B with nest lock A.

spin_lock_nest B, n=A

...
// Holds B with nest lock A.
spin_lock_nest B, n=A

spin_unlock B
Lock usage correctness
- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness
- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness
- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage
RCU read forbidden context

// Specifies a RCU read section.
rcu_read_lock
rcu_read_unlock

RCU usage correctness
- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness
- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

Usage from illegal states
RCU read forbidden context e.g. idle shouldn’t have RCU read sections.
RCU read forbidden context

// Specifies a RCU read section.
rcu_read_lock

rcu_read_unlock
RCU read forbidden context

// Specifies a RCU read section.
rcu_read_lock

...
// Specifies a RCU read section.

Proper use of the APIs

rcu_dereference_protected is supposed to be used with updater lock held.

rcu_dereference is supposed to be used in RCU read section.

rcu_read_unlock
// Specifies a RCU read section.
rcu_read_lock

// Calls without necessary lock held.
rcu_dereference_protected A

rcu_read_unlock
// Specifies a RCU read section.
rcu_read_lock

// Calls without necessary lock held.
rcu_dereference_protected

rcu_read_unlock
// Specifies a RCU read section.
rcu_read_lock

Delimitation for read section
rcu_read_unlock

RCU read APIs should be used in a section delimited by rcu_read_lock / rcu_read_unlock.

list_for_each_entry_rcu
// Specifies a RCU read section.
rcu_read_lock
...
rcu_read_unlock
...

// Traverses RCU list out of read section.
list_for_each_entry_rcu
...

Lock usage correctness
  Lock / unlock balance
  Relation between lock types
  Nest lock usage

RCU usage correctness
  Usage from illegal states
  Proper use of the APIs
  Delimitation for read section

Lock dependency correctness
  Lock acquisition order
  Lock recursive acquisitions
  Irq safe usage
// Specifies a RCU read section.
rcu_read_lock
...
rcu_read_unlock
...

// Traverses RCU list out of read section.
list_for_each_entry_rcu
...

Lock usage correctness
  Lock / unlock balance
  Relation between lock types
  Nest lock usage

RCU usage correctness
  Usage from illegal states
  Proper use of the APIs
  Delimitation for read section

Lock dependency correctness
  Lock acquisition order
  Lock recursive acquisitions
  Irq safe usage
Lock usage correctness
  Lock / unlock balance
  Relation between lock types
  Nest lock usage

RCU usage correctness
  Usage from illegal states
  Proper use of the APIs
  Delimitation for read section

Lock dependency correctness
  Lock acquisition order
  Lock recursive acquisitions
  Irq safe usage
// Holds A.
spin_lock A
...
// Holds B while holding A.
spin_lock B
...
// Holds B.
spin_lock B
...
// Holds A while holding B.
spin_lock A
...
spin_unlock A
spin_unlock B
spin_unlock A
spin_unlock B

Lock acquisition order

All the lock acquisition order should be kept across the kernel code.

Lock usage correctness
- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness
- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness
- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage
// Holds A.
spin_lock A
...
// Holds B while holding A.
spin_lock B
...
// Holds B.
spin_lock B
...
// Holds A while holding B.
spin_lock A
...
spin_unlock B
spin_unlock A
spin_unlock B
spin_unlock A
Lock usage correctness
  Lock / unlock balance
  Relation between lock types
  Nest lock usage

RCU usage correctness
  Usage from illegal states
  Proper use of the APIs
  Delimitation for read section

Lock dependency correctness
  Lock acquisition order
  Lock recursive acquisitions
  Irq safe usage

// Holds A.
spin_lock A
...
// Holds B while holding A.
spin_lock B
...
// Holds A while holding B.
spin_lock A
...
spin_unlock B
spin_unlock A
spin_unlock B
// Holds A.
spin_lock A

... 

Lock recursive acquisition
spin_lock A

The same lock shouldn’t be acquired twice in a single context.
spin_unlock A
// Holds A.

```c
spin_lock A
```

...  

// Holds A while holding A.

```c
spin_lock A
```

...  

```c
spin_unlock A
```

```c
spin_unlock A
```
// Holds A.
spin_lock A
...
// Holds A while holding A.
spin_lock A
...
spin_unlock A
spin_unlock A
Lock usage correctness
  Lock / unlock balance
  Relation between lock types
  Nest lock usage

RCU usage correctness
  Usage from illegal states
  Proper use of the APIs
  Delimitation for read section

Lock dependency correctness
  Lock acquisition order
  Lock recursive acquisitions
  Irq safe usage

interrupt context

spin_lock_irq A
...

Irq safe usage

Irq safe APIs should be used if the lock is used in either process or interrupt context.

// Holds A with irq disabled.
spin_lock_irq A
...

spin_unlock_irq A
Lock usage correctness
  - Lock / unlock balance
  - Relation between lock types
  - Nest lock usage

RCU usage correctness
  - Usage from illegal states
  - Proper use of the APIs
  - Delimitation for read section

Lock dependency correctness
  - Lock acquisition order
  - Lock recursive acquisitions
  - Iro safe usage

interrupt context

spin_lock_irq A
...
spin_unlock_irq A

process context

// Holds A with irq disabled.
spin_lock_irq A
...
spin_unlock_irq A
interrupt context

spin_lock A

... 

spin_unlock A

process context

// Holds A with irq enabled.

spin_lock A

... 

spin_unlock A
process context

// Holds A with irq enabled.
spin_lock A

...  
interrupt context

spin_lock A

...  
spin_unlock A

spin_unlock A
process context

// Holds A with irq enabled.
spin_lock A
...

interrupt context

spin_lock A
...

spin_unlock A
spin_unlock A

Lock usage correctness
  Lock / unlock balance
  Relation between lock types
  Nest lock usage

RCU usage correctness
  Usage from illegal states
  Proper use of the APIs
  Delimitation for read section

Lock dependency correctness
  Lock acquisition order
  Lock recursive acquisitions
  Irq safe usage
Lock usage correctness
- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness
- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness
- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

Lockdep is a quite good tool for checking these.
mutex_lock A
...
// Waits for event B.
wait_for_event B
...
// Releases A.
mutex_unlock A

// Waits for A to be released.
mutex_lock A
...
mutex_lock A
...
mutex_unlock A
...
// Wakes up wait_for_event B.
event B
mutex_lock A
...
// Waits for event B.
wait_for_event B
...
// Releases A.
mutex_unlock A
...
// Waits for A to be released.
mutex_lock A
...
mutex_unlock A
...
// Wakes up wait_for_event B.
event B
Lock usage correctness
- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness
- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness
- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

```c
mutex_lock A
...
// Waits for event B.
wait_for_event B
...
// Releases A.
mutex_unlock A
```

```c
// Waits for A to be released.
mutex_lock A
...
mutex_unlock A
...
// Wakes up wait_for_event B.
event B
```
Lock usage correctness
  Lock / unlock balance
  Relation between lock types
  Nest lock usage
RCU usage correctness
  Usage from illegal states
  Proper use of the APIs
  Delimitation for read section
Lock dependency correctness
  Lock acquisition order
  Lock recursive acquisitions
  Irq safe usage

mutex_lock A
...
// Releases A.
mutex_unlock A

// Waits for A to be released.
mutex_lock A
...
mutex_unlock A
...
Lock usage correctness
  Lock / unlock balance
  Relation between lock types
  Nest lock usage

RCU usage correctness
  Usage from illegal states
  Proper use of the APIs
  Delimitation for read section

Lock dependency correctness
  Lock acquisition order
  Lock recursive acquisitions
  Irq safe usage

mutex_lock A // Waits for A to be released.
mutex_lock A
...
mutex_lock A
mutex_unlock A
...
mutex_unlock A
...
mutex_unlock A
...
mutex_unlock A

// Releases A.
Lock usage correctness
- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness
- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness
- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

What makes the case missed?
Lock usage correctness
- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness
- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

**Lock dependency correctness**
- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

How does it track dependency?
Define Dependency

Lock A
Lock B (while holding A.)

= A depends on B. (A → B)
Example

// Holds A.
spin_lock A
...

// Holds B while holding A.
spin_lock B
...

spin_unlock B
spin_unlock A
Build Dependency Graph
// Holds B.
spin_lock B

...
Build Dependency Graph
Circular Dependency
REPORT!

Circular Dependency
REPORT!
Is a deadlock caused by incorrect lock acquisition order?
Is a deadlock caused by incorrect lock acquisition order?

Partially yes.
A deadlock is caused by **Waiters** that cannot be woken up.
A deadlock is caused by **Events** that are not reachable.
Deadlock detection tool should focus on waits and events themselves.
Define Dependency

Lock A
Lock B (while holding A.)

= A depends on B. (A → B)
Redefine Dependency

Event A occurrence depends on Event B occurrence.

= A depends on B. (A \rightarrow B)
Event A occurrence depends on Event B occurrence.
Example

mutex_lock A

... 

// Waits for event B.

wait_for_event B

... 

// Depends on event B.

mutex_unlock A
Build Dependency Graph
Example

// Waits for A to be released.
mutex_lock A
...
mutex_unlock A
...
// Depends on releasing A.
event B
Build Dependency Graph
Circular Dependency
CIRCULAR DEPENDENCY

REPORT!

Circular Dependency
REWIND ...
// Holds A.
spin_lock A
...

// Holds B while holding A.
spin_lock B
...

// Holds B.
spin_lock B
...

// Holds A while holding B.
spin_lock A
...

spin_unlock B
spin_unlock A
spin_unlock A
spin_unlock B
Lock usage correctness
- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness
- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness
- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

// Holds A.
spin_lock A
...
// Holds B while holding A.
waiter B
waiter A
...
// Holds B.
spin_lock B
...
// Holds A while holding B.
spin_lock A
spin_unlock B
spin_unlock A
spin_unlock B
// Holds A.

spin_lock A

... 

// Holds A while holding A.

spin_lock A

...

spin_unlock A

... 

spin_unlock A
// Holds A.

spin_lock A

...  

// Holds A while holding A.

spin_lock A

waiter A

spin_unlock A

...

spin_unlock A

event A
process context

// Holds A with irq enabled.

spin_lock A

...  

interrupt context

spin_lock A

...

spin_unlock A  

spin_unlock A
process context

// Holds A with irq enabled.
spin_lock A

interrupt context

spin_unlock A

waiter A

event A

spin_unlock A
mutex_lock A
...
// Waits for event B.
wait_for_event B
...
// Releases A.
mutex_unlock A

// Waits for A to be released.
mutex_lock A
...
mutex_lock A
...
// Wakes up wait_for_event B.
event B
Lock usage correctness
- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness
- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness
- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage

mutex_lock A
... // Waits for event B.
wait_for_event B
... // Releases A.
mutex_unlock A
... // Waits for A to be released.
mutex_lock A
... // Waits for event B.
wait_for_event B
... // Wakes up wait_for_event B.
mutex_unlock A
... // Wakes up event B.
Lockdep is a great tool for checking these two.

Lock usage correctness
- Lock / unlock balance
- Relation between lock types
- Nest lock usage

RCU usage correctness
- Usage from illegal states
- Proper use of the APIs
- Delimitation for read section

Lock dependency correctness
- Lock acquisition order
- Lock recursive acquisitions
- Irq safe usage
Lock dep is not the best tool for this purpose.
Lockdep is not the best tool for this purpose.

So I developed a new tool named **DEPT** to replace the 3\textsuperscript{rd} part to deal with dependency in the best way.
So I developed a new tool named DEPT to replace the 3\textsuperscript{rd} part to deal with dependency in the best way.

DEPT is the best tool for this purpose.
DEPT detects situations where waiters cannot be woken up.
DEPT detects situations where events are not reachable.
Pros

Works with all types of wait.
Supports multiple reports efficiently.
Provides easy APIs to annotate wait / event.

Cons

Just started so still has false positives.
LKML Discussion
https://lkml.org/lkml/2022/5/4/218

Github Repository
https://github.com/lgebyungchulpark/linux-dept
WANTED