Recursive read deadlocks and Where to find them

冯博群 Boqun Feng

HUAWEI

Agenda

- Deadlock cases
- Lockdep
- Flavors of read/write locks
- More deadlock cases
- (Recursive) read deadlock detection

Deadlock cases

- self deadlock

```
P0

spin_lock(&A);
...
spin_lock(&A);
```

- ABBA deadlock

```
P0                      P1

spin_lock(&A);          spin_lock(&B);
...                     ...
spin_lock(&B);          spin_lock(&A);
```

Deadlock cases (cont.)

- IRQ safe->unsafe deadlocks
  - IRQs bring more "code combinations"

```
P0

<irq enabled>
spin_lock(&A);
...
<in irq handler>
 spin_lock(&A);
```

```
P0                        P1

<irq enabled>             <irq disabled>
spin_lock(&A);            spin_lock(&B);
...                       ...
<in irq handler>
 spin_lock(&B);           spin_lock(&A);
```

Deadlock cases (cont.)

- ABBCCA deadlocks
  - or more

```
P0                      P1                      P2

spin_lock(&A);          spin_lock(&B);          spin_lock(&C);
...                     ...                     ...
spin_lock(&B);          spin_lock(&C);          spin_lock(&A);
```

Lockdep

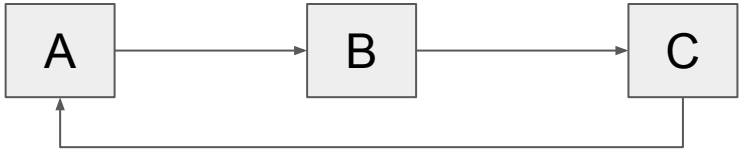
- Detect deadlock possibility
  - Assume the worst case: all the combinations of code sequences can happen
- Lock dependency
  - A -> B
  - Assume we can catch all dependencies

```
P0

spin_lock(&A);
...
spin_lock(&B);
```

- Dependency graph

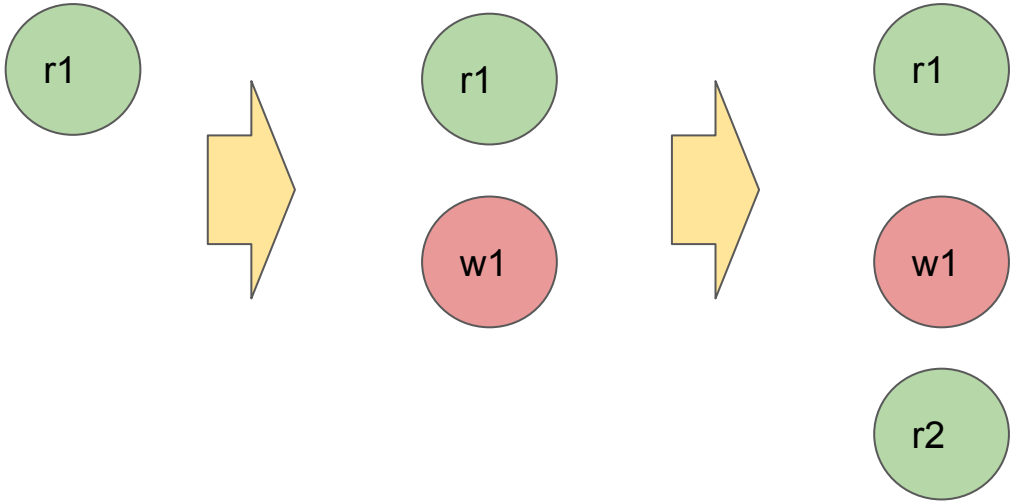
Lockdep (cont.)

- Deadlock detection
  - A closed path (circle) in the dependency graph



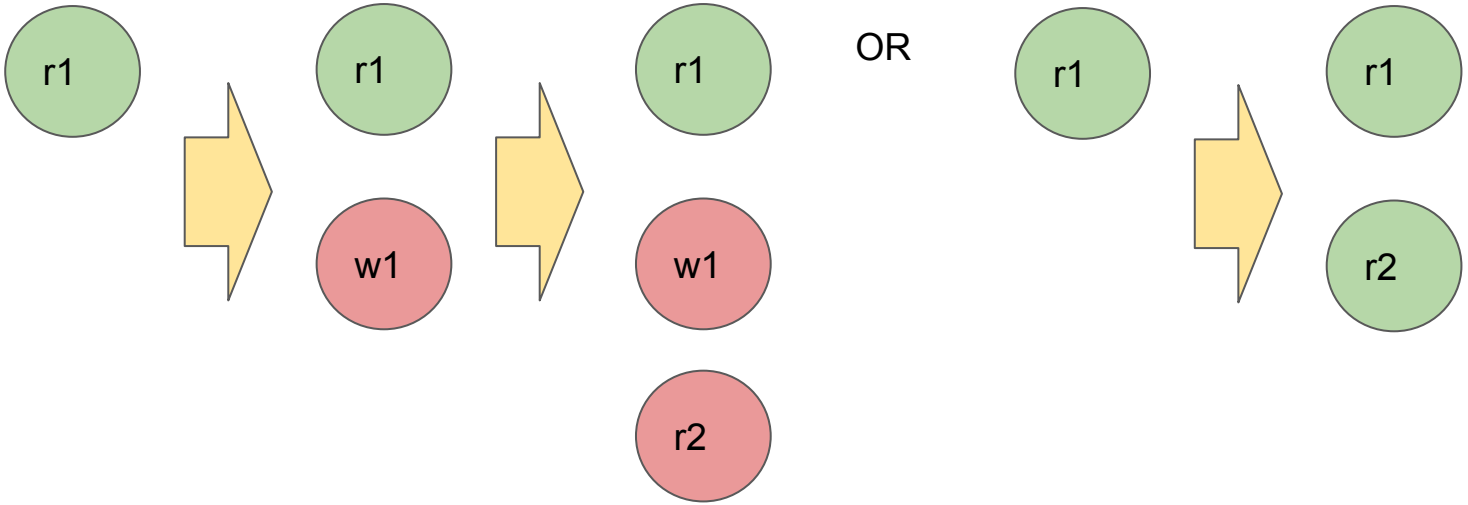
Flavors of read/write locks

- Recursive/unfair rwlocks
  - readers are preferable



Flavors of read/write locks (cont.)

- Non-recursive/fair rwlocks



Flavors of read/write locks (cont.)

flavors	multiple readers	recursive c.s	a reader blocks another reader
recursive	Y	Y	N
non-recursive	Y	N	Y* (via a waiting writer)

Flavors of read/write locks (cont.)

- Block condition
  - Recursive readers can get blocked by writers
  - Non-recursive readers can get blocked by non-recursive readers (via a waiting writer) or writers

	reader(recursive or not)	writer
recursive reader	No	Yes
non-recursive(r & w)	Yes	Yes

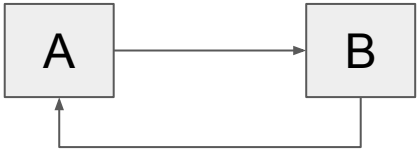
More deadlock cases

- For non-recursive read/write locks
  - Same as spinlocks, since readers can block each other via a waiting writer

```
P0                      P1                      P2

read_lock(&A);          spin_lock(&B);
...                     ...                     write_lock(&A);
spin_lock(&B);          read_lock(&A);
```

More deadlock cases

- For recursive locks, things get interesting:
  - This is not a deadlock

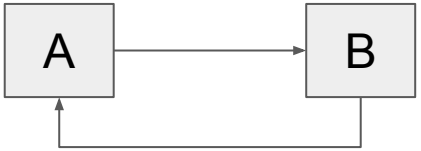



```
P0                          P1

read_lock(&A);              spin_lock(&B);
...                         ...
spin_lock(&B);              read_lock(&A);
```

More deadlock cases

- But this is a deadlock

```
P0                          P1

read_lock(&A);              spin_lock(&B);
...                         ...
spin_lock(&B);              write_lock(&A);
```

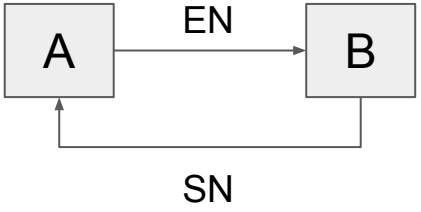


More deadlock cases

- Things get complicated when we mixed recursive and non-recursive read locks
- queued rwlock
  - non-recursive read lock in process context
  - recursive read lock in irq context

More deadlock cases

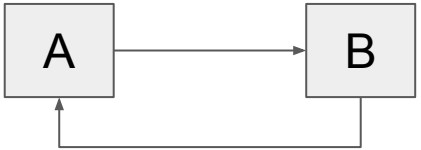
- Recursive deadlock case




```
P0                          P1                          P2

<in irq handler>
 read_lock(&B);             spin_lock_irq(&A);
                                                        write_lock_irq(&B);
 spin_lock(&A);             read_lock(&B);
```

More deadlock cases

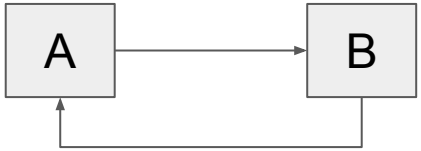
- Recursive *not* deadlock case




```
P0                          P1                          P2

<in irq handler>
 spin_lock(&A);             read_lock(&B);
                                                        write_lock_irq(&B);
 read_lock(&B);             spin_lock_irq(&A);
```

Recursive read deadlock detection

- Limitation of current lockdep
  - circles mean deadlocks
  - while not all the circles mean deadlocks if we consider recursive readers.



Recursive read deadlock detection

- Goals
  - Compatible with original lockdep detection.
  - Handle qrwlock semantics.
  - No false positive.

Recursive read deadlock detection

- Overview
  - Classification for lock dependencies
  - Definition of "strong" dependencies
  - Deadlock Condition

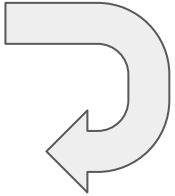
Classification of lock dependencies

- We used to treat all lock dependencies as the same
- but they are really not.
- {R reader, reader, writer} -> {R reader, reader, writer} : 9 combinations

Classification of locks

- (S)hared locks: reader (recursive or not)
- (E)clusive locks: writer (or plain spinlocks)
- (R)ecurisve readers
- (N)on-recursive (readers and writers)

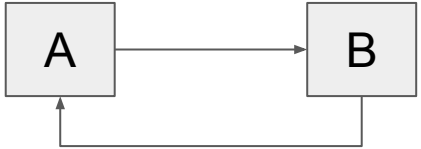
blocked by	reader(recursive or not)	writer
recursive reader	No	Yes
non-recursive(r & w)	Yes	Yes



blocked by	S	E
R	No	Yes
N	Yes	Yes

Classification of lock dependencies

- Groups things into 4
  - -(SN)-> : {R reader, reader} -> {reader, writer}
  - -(SR)-> : {R reader, reader} -> {R reader}
  - -(EN)-> : {writer} -> {reader, writer}
  - -(ER)-> : {writer} -> {R reader}
- Why? Because for a dependency A -> B, we cares:
  - Whether A can block anyone
  - Whether B can get blocked by anyone



blocked by	S	E
R	No	Yes
N	Yes	Yes

```
P0                          P1

read_lock(&A);              spin_lock(&B);
...                         ...
spin_lock(&B);              write_lock(&A);
```

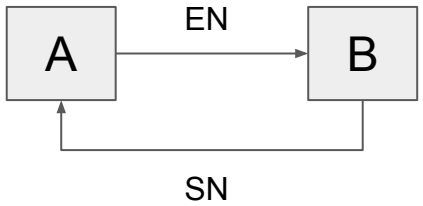
Definition of "strong" dependencies

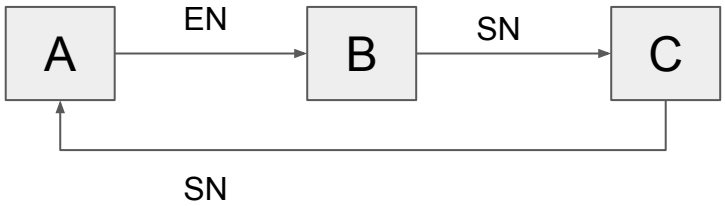
- Chaining lock dependencies via block conditions
- For dependencies A -> B and B -> C
  - A -> B -> C is a "strong" dependency path iff
    - A -> B : -(*R)-> and B -> C : -(E*)->, or
    - A -> B : -(*N)-> and B -> C : -(S*)->, or
    - A -> B : -(*N)-> and B -> C : -(E*)->
  - IOW, -(*R)-> -(S*)-> will break the dependency
- works for "A -> B, B -> C and C -> D" case, and so on

blocked by	S	E
R	No	Yes
N	Yes	Yes

Deadlock condition

- A strong dependency chain/path forms a circle






```
P0                      P1                      P2

spin_lock(&A);          read_lock(&B);          read_lock(&C);
...                     ...                     ...
write_lock(&B);         write_lock(&C);         spin_lock(&A);
```

Implementation

- Extend __bfs() to walk on strong dependency path
- Make LOCK*_STATE* part of the chainkeys
- Add test cases
  - also unleash irq_read_recursion2
- Enable this for srcu
- Code
  - git.kernel.org/pub/scm/linux/kernel/git/boqun/linux.git arr-rfc-wip

欢迎加入华为OS内核实验室

华为操作系统部:华为端、管、云核心的OS软件基础设施

OS Kernel Lab

- Linux内核(ARM/x86/异构等)的技术研发与创新
- 低时延、高安全、高可靠、高智能的下一代OS内核技术的研究和成果转化

招聘岗位

下一代操作系统研究员/高级工程师

形式化技术研究员/高级工程师

Linux内核架构师/高级工程师

Linux内核测试专家/高级工程师

工作地

杭州、北京、上海

简历投递

Tel:王先生/18658102676

Email:hr.kernel@huawei.com