百度智能云

读写信号量bug分析与修复

Xie Yongji (Baidu AI Cloud)

Zhang Yu (Baidu AI Cloud)

背景

百度智能云

- 线上持续出现多台物理机load负载异常，执行ps等命令会hang住
- 线下未能成功复现

```
[                                    ~]# uptime
 18:44:37 up 24 days,  2:39,  1 user,  load average: 15394.11, 15394.07, 15394.06
```

追查-负载异常

百度智能云

- 大量进程D住，导致loadavg上升

```
crash> ps -m | grep UN | tail
[12 23:08:14.476] [UN]  PID: 29444  TASK: ffff881cc0a61f40  CPU: 29  COMMAND: "pgrep"
[12 23:08:20.666] [UN]  PID: 29101  TASK: ffff881e113d5dc0  CPU: 31  COMMAND: "pgrep"
[12 23:08:23.836] [UN]  PID: 28822  TASK: ffff881cc0a65dc0  CPU: 26  COMMAND: "ps"
[12 23:08:24.007] [UN]  PID: 7854   TASK: ffff881ff2f53e80  CPU: 15  COMMAND: "server_inspecto"
[12 23:08:24.542] [UN]  PID: 27729  TASK: ffff881c4561be80  CPU: 20  COMMAND: "pgrep"
[12 23:08:24.553] [UN]  PID: 27705  TASK: ffff881fc20f8000  CPU: 21  COMMAND: "pgrep"
[12 23:08:24.709] [UN]  PID: 23778  TASK: ffff881cdaf1be80  CPU: 16  COMMAND: "instance_inspec"
[12 23:08:26.224] [UN]  PID: 26531  TASK: ffff881c63799f40  CPU: 18  COMMAND: "pgrep"
[12 23:08:27.229] [UN]  PID: 26451  TASK: ffff881d5ee1be80  CPU: 1   COMMAND: "sudo"
[12 23:08:27.232] [UN]  PID: 26447  TASK: ffff881f8fb81f40  CPU: 1   COMMAND: "sudo"
```

追查-D进程

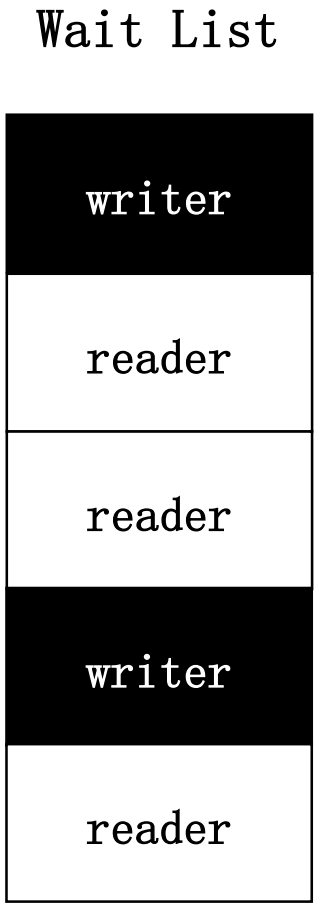
百度智能云

```
PID: 21553  TASK: ffff8813dde41f40  CPU: 0   COMMAND: "sudo"
 #0 [ffff8816cf633d70] __schedule at ffffffff817114f2
 #1 [ffff8816cf633dc8] schedule at ffffffff81711bd9
 #2 [ffff8816cf633dd8] rwsem_down_read_failed at ffffffff81713115
 #3 [ffff8816cf633e60] call_rwsem_down_read_failed at ffffffff813d1728
 #4 [ffff8816cf633eb0] down_read at ffffffff81710027
 #5 [ffff8816cf633ec0] __do_page_fault at ffffffff81718af6
 #6 [ffff8816cf633f20] do_page_fault at ffffffff81718be5
 #7 [ffff8816cf633f50] page_fault at ffffffff81715048
    RIP: 00007f68c8a6e625  RSP: 00007ffe536f1c48  RFLAGS: 00010206
    RAX: ffffffffffffffea  RBX: 00007f68b7fff700  RCX: 00007f68b7fff700
    RDX: 00000000003d0f00  RSI: 00007f68b7ffef50  RDI: 00007f68c6c86780
    RBP: 0000000000000000   R8: 00007f68b7fff9d0   R9: 00007f68b7fff700
    R10: 00007f68b7ffef60  R11: 0000000000000246  R12: 00007f68c6c90200
    R13: 00007f68b7fff9c0  R14: 0000000000000000  R15: 0000000000000003
    ORIG_RAX: ffffffffffffffff  CS: 0033  SS: 002b
```

追查-分析读写锁

百度智能云

- 处于读者持锁状态



追查-找持锁进程

百度智能云

- 遍历所有运行进程的调用栈，未找到有持锁记录的进程
- 遍历所有等待队列的进程调用栈，未找到有死锁的场景
- 未开启Debug选项，锁的元数据内，未记录持锁进程

追查-持锁进程

百度智能云

- 疑点：通过遍历该读写信号量wait_list里面的所有进程，发现系统中有一个D状态的进程（尝试拿锁失败后睡眠）并没有在该锁的等待队列里面

- 推测：该进程就是“持锁”进程，但该进程却没有被唤醒，导致该锁没有被释放，后续进程全部都hang住

定位-持锁进程未被成功唤醒

百度智能云

定位-持锁进程未被成功唤醒

百度智能云

- 新版本(v4.7)引入lockless wakeup机制

定位-持锁进程未被成功唤醒

百度智能云

- 新版本(v4.7)引入lockless wakeup机制(Before)

```
up_write(写者解锁):

if atomic_sub_fetch(rwstate, RWSEM_ACTIVE_WRITE_BIAS) >= 0 {
  return;
}
spin_lock();
if first waiter is writer {
  wake up the waiter;
  rwstate = -1;
} else {
  wake up the waiters in wait list until we meet the writer
  rwstate = woken_readers;
}
if wait_list is empty
  rwstate -= RWSEM_WAITING_BIAS
spin_unlock();
```

定位-持锁进程未被成功唤醒

百度智能云

- 新版本(v4.7)引入lockless wakeup机制(Before)

```
up_write(写者解锁):

if atomic_sub_fetch(rwstate, RWSEM_ACTIVE_WRITE_BIAS) >= 0 {
  return;
}
spin_lock();
if first waiter is writer {
  add the waiter to local wake_queue;
  rwstate = -1;
} else {
  add the waiters to local wake_queue until we meet writer;
  rwstate = woken_readers;
}
if wait_list is empty
  rwstate -= RWSEM_WAITING_BIAS
spin_unlock();
wake up the local wake_queue
```

定位-持锁进程未被成功唤醒

百度智能云

```
/* 遍历读写信号量的等待队列 */
while (!slist_empty(&sem->wait_list)) {
  struct task_struct *tsk;

  /* 找到等待队列第一个entry */
  waiter = list_entry(sem->wait_list.next, typeof(*waiter), list)

  if (waiter->type = RWSEM_WAITING_FOR_WRITE)
    break;

  woken++;
  tsk = waiter->task;

  wake_q_add(wake_q, tsk);  // 将waiter加入本地唤醒队列
  slist_del(&waiter->list, &sem->wait_list);  // 将waiter移除出等待队列
  smp_store_release(&waiter->task, NULL);  // 清空waiter的睡眠条件
}
spin_unlock();
wake_up_q(&wake_q);  // 执行唤醒操作
```

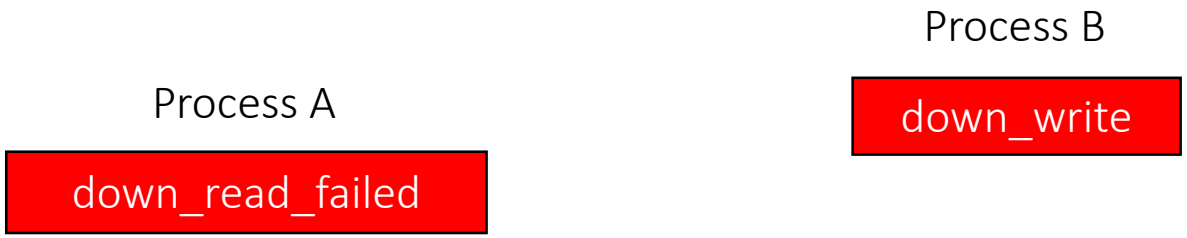
定位-持锁进程未被成功唤醒

百度智能云

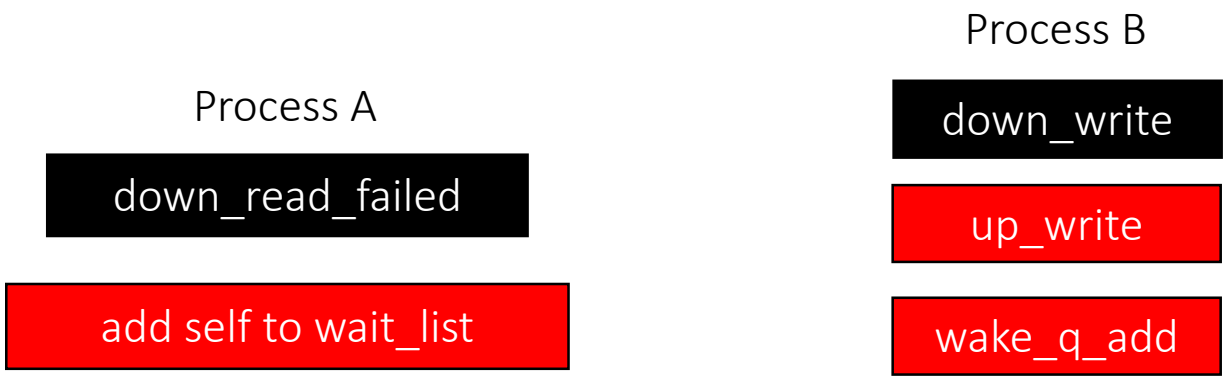
```
/* 遍历读写信号量的等待队列 */
while (!slist_empty(&sem->wait_list)) {
  struct task_struct *tsk;

  /* 找到等待队列第一个entry */
  waiter = list_entry(sem->wait_list.next, typeof(*waiter), list)

  if (waiter->type = RWSEM_WAITING_FOR_WRITE)
    break;

  woken++;
  tsk = waiter->task;

  wake_q_add(wake_q, tsk);  // 将waiter加入本地唤醒队列
  slist_del(&waiter->list, &sem->wait_list);  // 将waiter移除出等待队列
  smp_store_release(&waiter->task, NULL);  // 清空waiter的睡眠条件
}
spin_unlock();
wake_up_q(&wake_q);  // 执行唤醒操作
```

```
void wake_q_add(struct wake_q_head *head,
struct task_struct *task)
{
struct wake_q_node *node = &task->wake_q;

/*
  * if ->wake_q is !nil already it means its
  * already queued (either by us or someone
  * else) and will get the wakeup later
  */
if (cmpxchg(&node->next, NULL, WAKE_Q_TAIL))
  return;

get_task_struct(task);

*head->lastp = node;
head->lastp = &node->next;
}
```

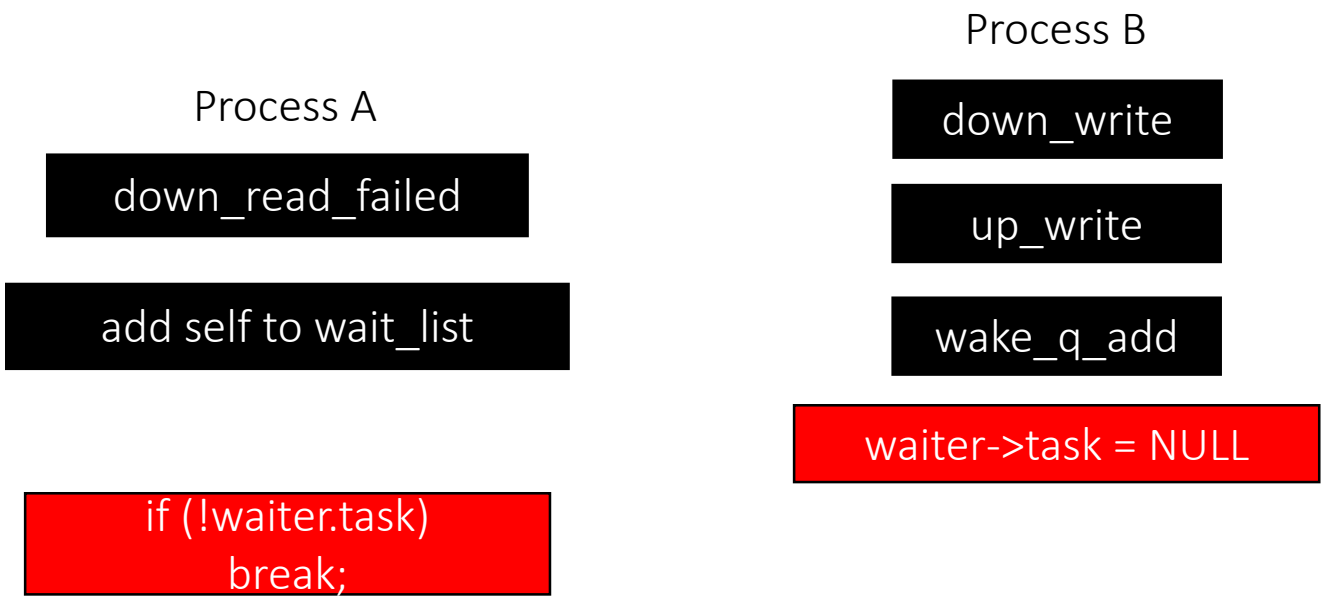
定位-Example



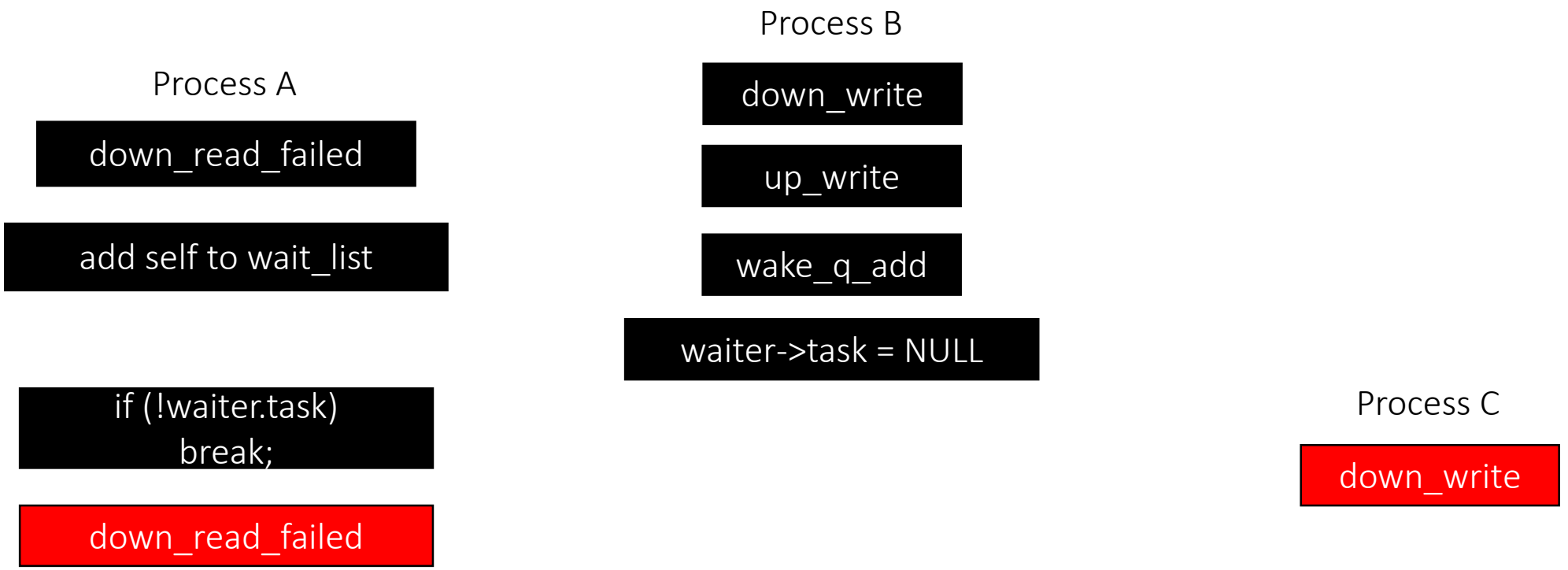
定位-Example


Process A
down_read_failed
add self to wait_list
Process B
down_write
up_write
wake_q_add

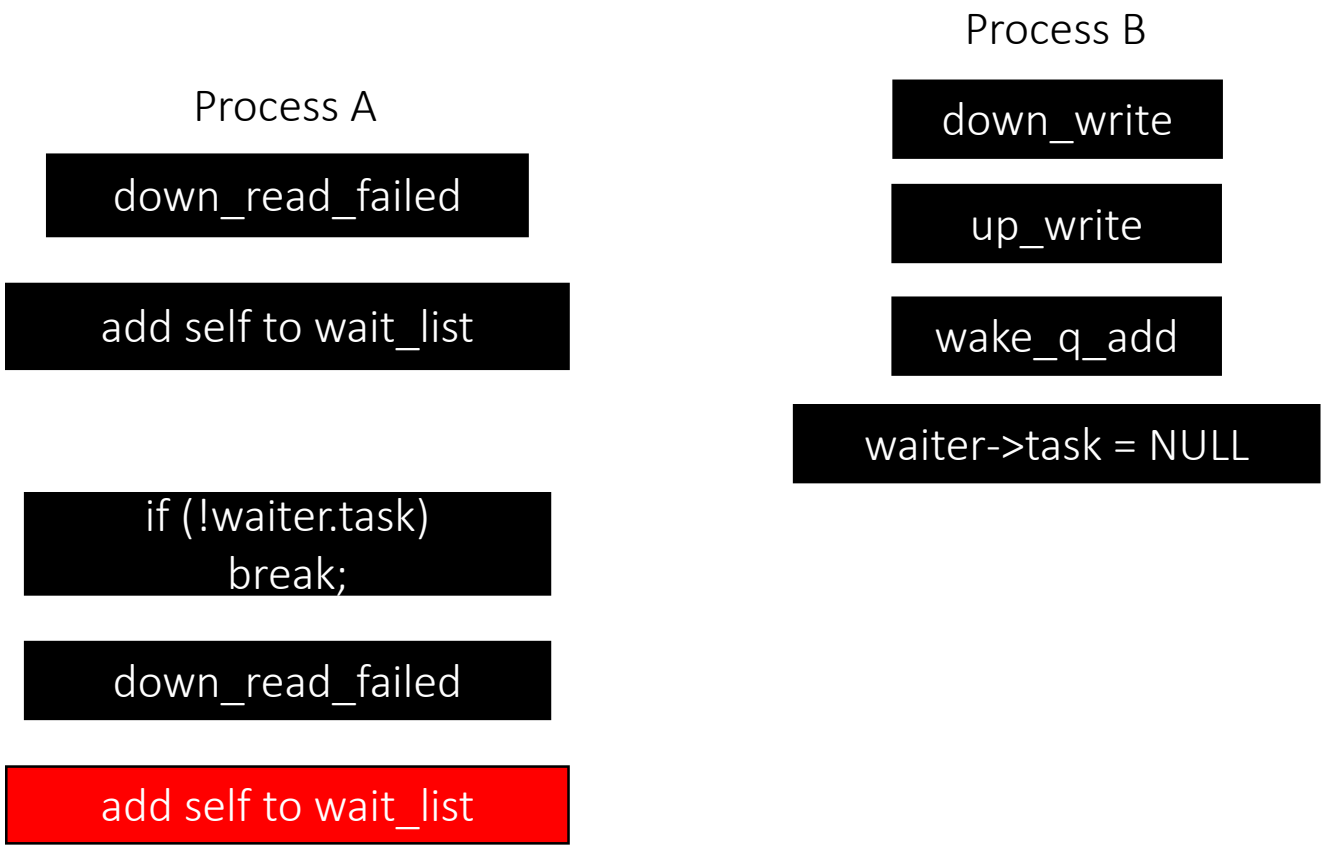
定位-Example


Process A
down_read_failed
add self to wait_list
if (!waiter.task)
break;
Process B
down_write
up_write
wake_q_add
waiter->task = NULL

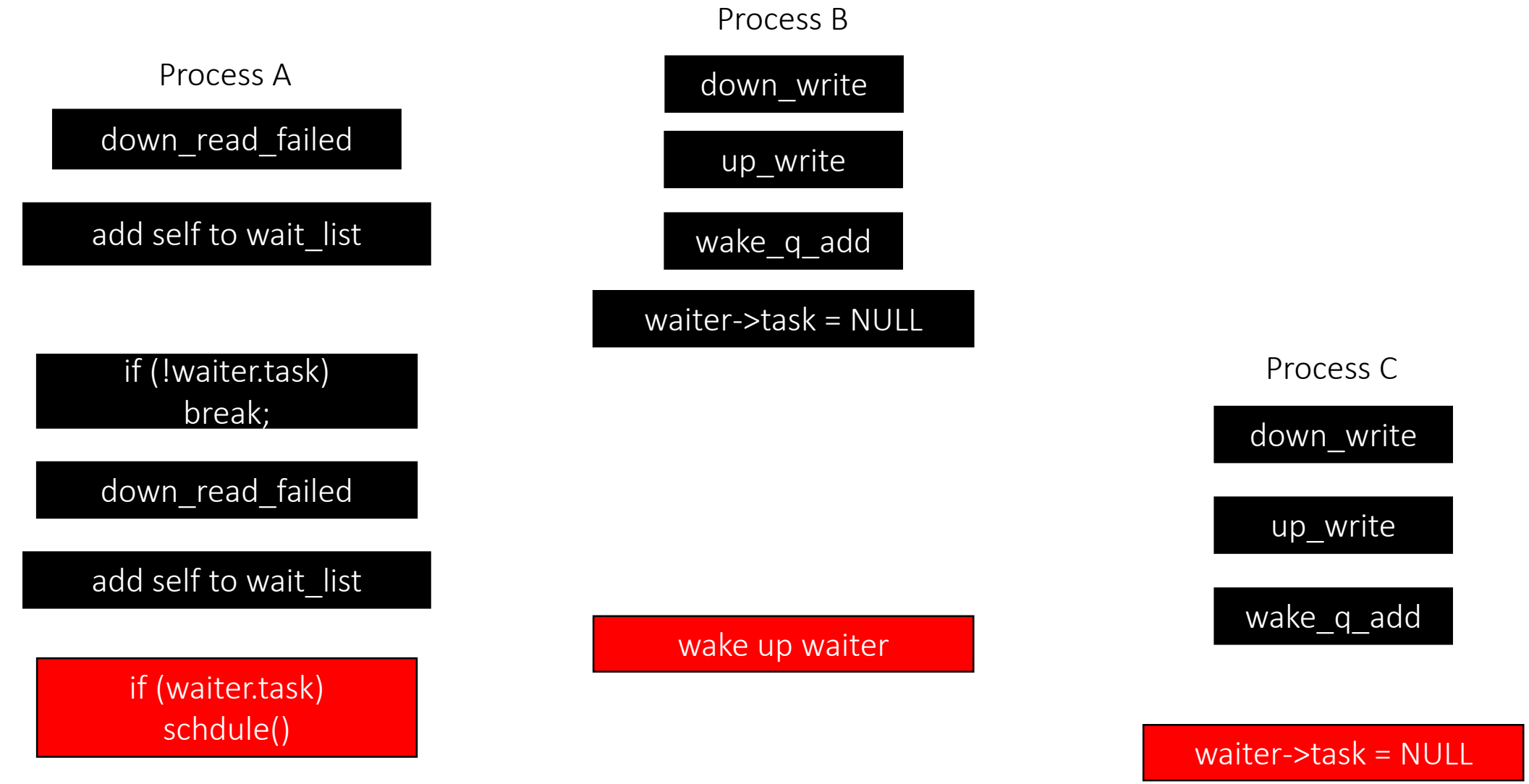
定位-Example


Process A
down_read_failed
add self to wait_list
if (!waiter.task)
break;
down_read_failed
Process B
down_write
up_write
wake_q_add
waiter->task = NULL
Process C
down_write

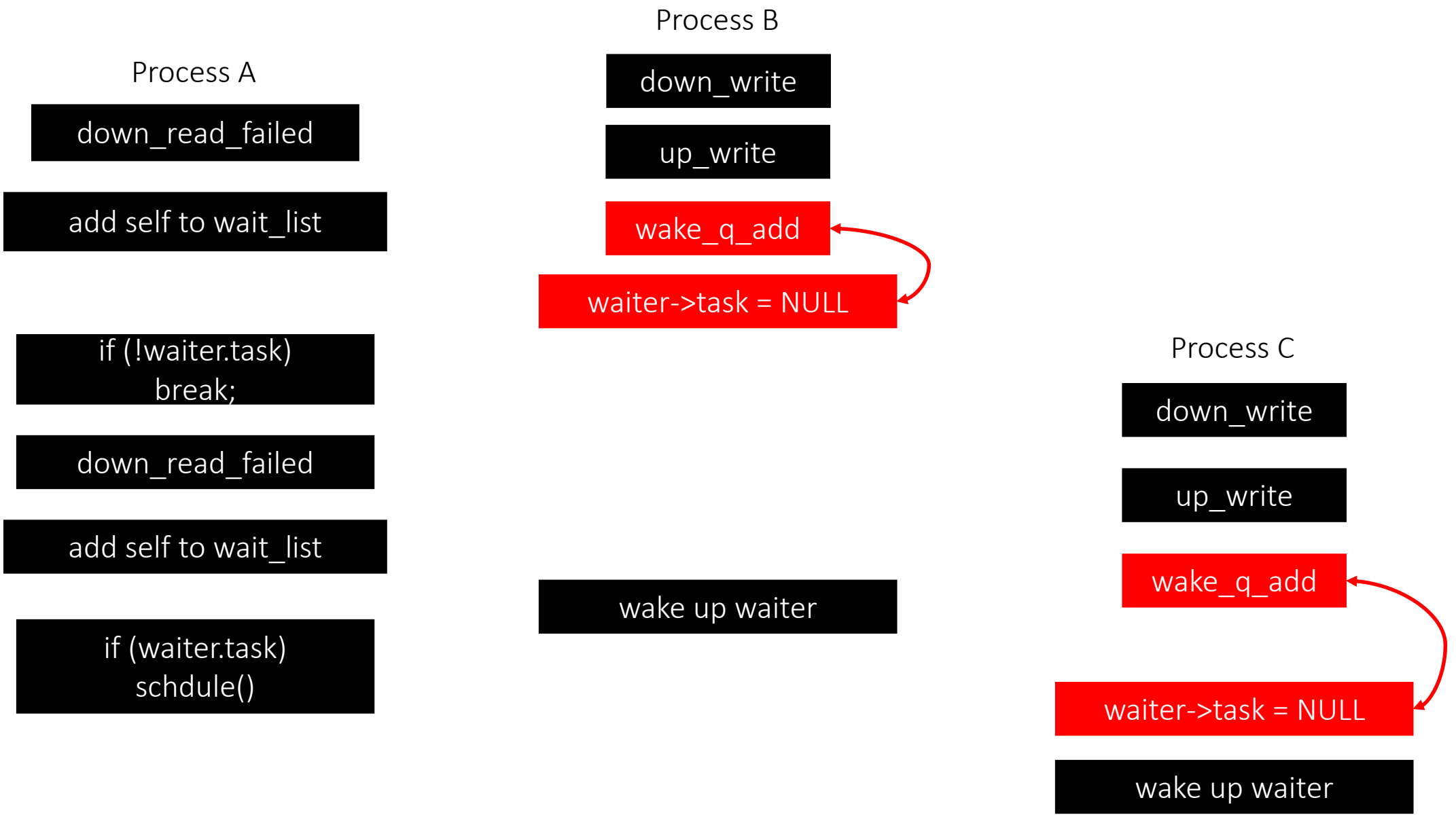
定位-Example


Process A
down_read_failed
add self to wait_list
if (!waiter.task)
break;
down_read_failed
add self to wait_list
Process B
down_write
up_write
wake_q_add
waiter->task = NULL
Process C
down_write
up_write
wake_q_add

定位-Example


Process A
down_read_failed
add self to wait_list
if (!waiter.task)
break;
down_read_failed
add self to wait_list
if (waiter.task)
schdule()
Process B
down_write
up_write
wake_q_add
waiter->task = NULL
wake up waiter
Process C
down_write
up_write
wake_q_add
waiter->task = NULL

修复方案


Process A
down_read_failed
add self to wait_list
if (!waiter.task)
break;
down_read_failed
add self to wait_list
if (waiter.task)
schdule()
Process B
down_write
up_write
wake_q_add
waiter->task = NULL
wake up waiter
Process C
down_write
up_write
wake_q_add
waiter->task = NULL
wake up waiter

修复方案

Process A

down_read_failed

add self to wait_list

if (!waiter.task)
break;

down_read_failed

add self to wait_list

if (waiter.task)
schdule()

Process B

down_write

up_write

wake_q_add

waiter->task = NULL

wake up waiter

```
void wake_q_add()
{
……
get_task_struct();
……
}
```

Process C

down_write

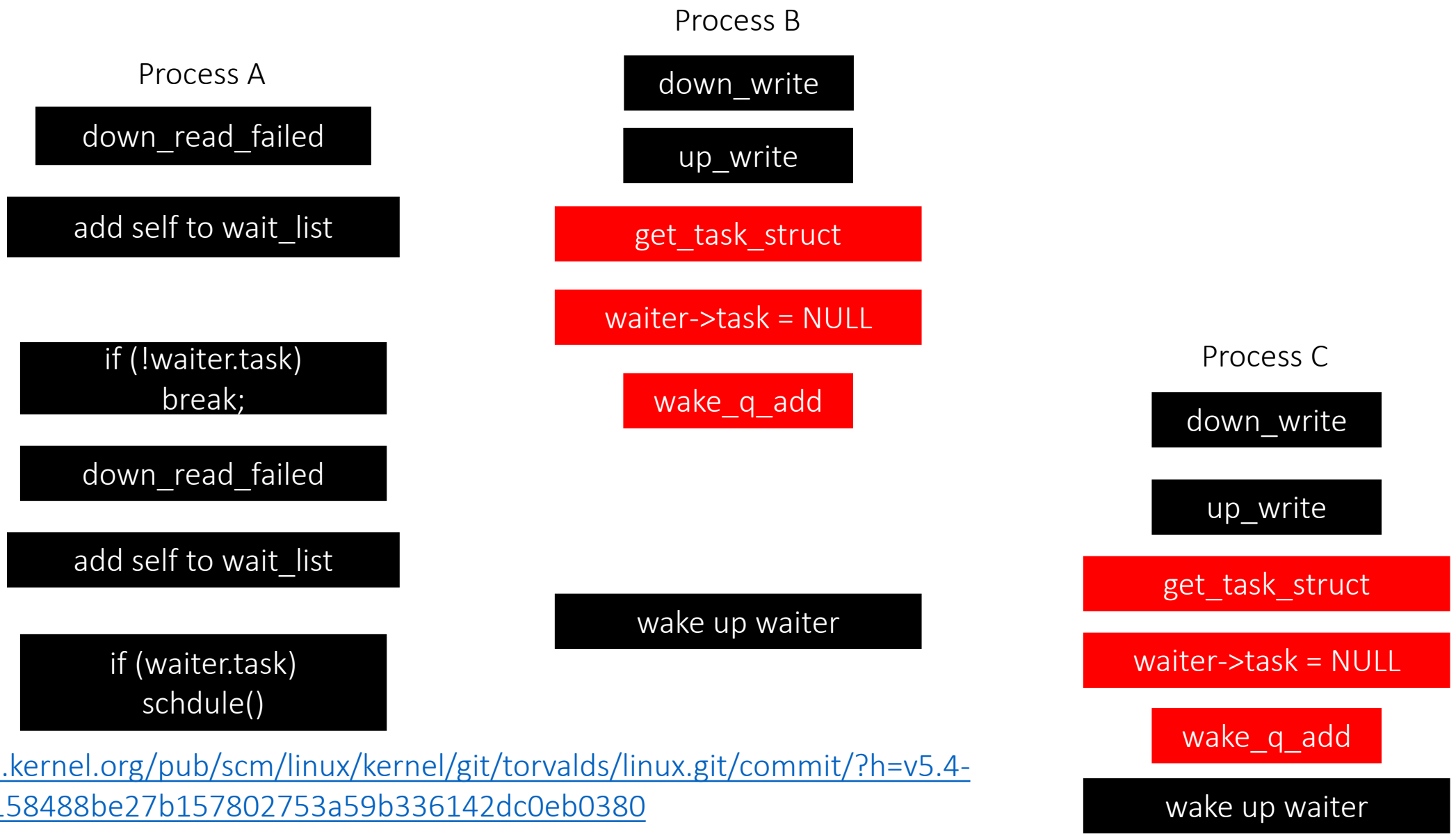
up_write

wake_q_add

waiter->task = NULL

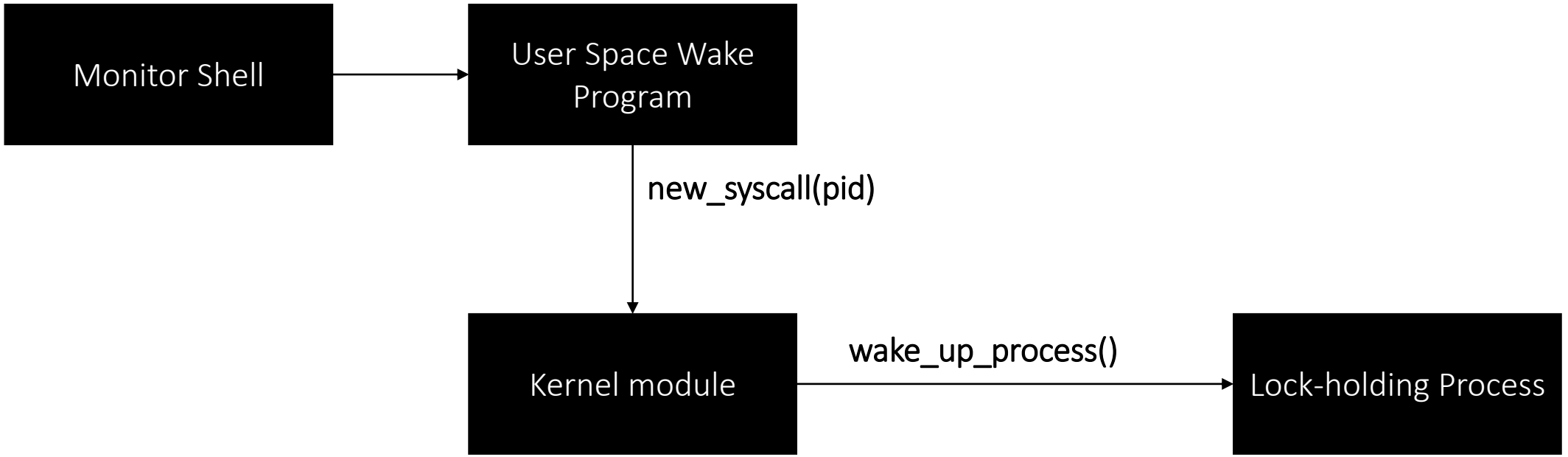
wake up waiter

修复方案



https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?h=v5.4-rc3&id=e158488be27b157802753a59b336142dc0eb0380

Workaround方案



百度智能云

THANK YOU

CLOUD.BAIDU.COM