# QEMU CORE分析之死锁

[mnstory.net](http://mnstory.net)

## main线程8003

QEMU出CORE，死在pthread\_mutex\_lock里：

Core was generated by `/usr/bin/kvm -id 3114792937754 -chardev socket,id=qmp,path=/var/run/qemu-server'.

Program terminated with signal SIGSEGV, Segmentation fault.

#0 0x00007fe1ed7d8cec in \_\_lll\_lock\_wait () from /lib/x86\_64-linux-gnu/libpthread.so.0

(gdb) bt

#0 0x00007fe1ed7d8cec in \_\_lll\_lock\_wait () from /lib/x86\_64-linux-gnu/libpthread.so.0

#1 0x00007fe1ed7d4339 in \_L\_lock\_926 () from /lib/x86\_64-linux-gnu/libpthread.so.0

#2 0x00007fe1ed7d415b in pthread\_mutex\_lock () from /lib/x86\_64-linux-gnu/libpthread.so.0

#3 0x00007fe1f4b84739 in qemu\_mutex\_lock (mutex=mutex@entry=0x7fe1f5223940 <qemu\_global\_mutex>) at util/qemu-thread-posix.c:73

#4 0x00007fe1f485d096 in qemu\_mutex\_lock\_iothread () at /home/jenkins/workspace/Compile/HCI5.2\_Compile/src/app/vtp-qemu-kvm/qemu-2.5.1/cpus.c:1235

#5 0x00007fe1f4afb264 in os\_host\_main\_loop\_wait (timeout=15740693) at main-loop.c:279

#6 main\_loop\_wait (nonblocking=<optimized out>) at main-loop.c:530

#7 0x00007fe1f48290b0 in main\_loop () at vl.c:2240

#8 main (argc=<optimized out>, argv=<optimized out>, envp=<optimized out>) at vl.c:5167

查看一下，是在等待qemu\_global\_mutex锁：

(gdb) f 3

#3 0x00007fe1f4b84739 in qemu\_mutex\_lock (mutex=mutex@entry=0x7fe1f5223940 <qemu\_global\_mutex>) at util/qemu-thread-posix.c:73

73 util/qemu-thread-posix.c: No such file or directory.

(gdb) p mutex

$1 = (QemuMutex \*) 0x7fe1f5223940 <qemu\_global\_mutex>

(gdb) p \*mutex

$2 = {lock = {\_\_data = {\_\_lock = 2, \_\_count = 0, \_\_owner = 8417, \_\_nusers = 9, \_\_kind = 0, \_\_spins = 0, \_\_list = {\_\_prev = 0x0, \_\_next = 0x0}},

\_\_size = "\002\000\000\000\000\000\000\000\341 \000\000\t", '\000' <repeats 26 times>, \_\_align = 2}}

通过\_\_owner看出，qemu\_global\_mutex 已经被人加锁了，线程ID为8417。

## migrate线程8417

切换到线程8417：

(gdb) t 22

[Switching to thread 22 (Thread 0x7fddd17fa700 (LWP 8417))]

#0 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

(gdb) bt

#0 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

#1 0x00007fe1f4b84959 in qemu\_cond\_wait (cond=cond@entry=0x7fe1f6e0ce80, mutex=mutex@entry=0x7fe1f6e0ce50) at util/qemu-thread-posix.c:132

#2 0x00007fe1f4b9642a in rfifolock\_lock (r=r@entry=0x7fe1f6e0ce50) at util/rfifolock.c:59

#3 0x00007fe1f4ae8f91 in aio\_context\_acquire (ctx=ctx@entry=0x7fe1f6e0cdf0) at async.c:371

#4 0x00007fe1f4b42fdb in bdrv\_drain\_all () at block/io.c:299

#5 0x00007fe1f485dd15 in do\_vm\_stop (state=RUN\_STATE\_FINISH\_MIGRATE) at /home/jenkins/workspace/Compile/HCI5.2\_Compile/src/app/vtp-qemu-kvm/qemu-2.5.1/cpus.c:737

#6 vm\_stop (state=state@entry=RUN\_STATE\_FINISH\_MIGRATE) at /home/jenkins/workspace/Compile/HCI5.2\_Compile/src/app/vtp-qemu-kvm/qemu-2.5.1/cpus.c:1416

#7 0x00007fe1f485ddfc in vm\_stop\_force\_state (state=state@entry=RUN\_STATE\_FINISH\_MIGRATE) at /home/jenkins/workspace/Compile/HCI5.2\_Compile/src/app/vtp-qemu-kvm/qemu-2.5.1/cpus.c:1424

#8 0x00007fe1f4a94d43 in migration\_completion (start\_time=<synthetic pointer>, old\_vm\_running=<synthetic pointer>, current\_active\_state=4, s=0x7fe1f51a6200 <current\_migration>)

at migration/migration.c:1614

#9 migration\_thread (opaque=0x7fe1f51a6200 <current\_migration>) at migration/migration.c:1754

#10 0x00007fe1ed7d1b50 in start\_thread () from /lib/x86\_64-linux-gnu/libpthread.so.0

#11 0x00007fe1ed51ba7d in clone () from /lib/x86\_64-linux-gnu/libc.so.6

#12 0x0000000000000000 in ?? ()

可以看到，migration线程是在迁移完成vm\_stop时，等待rfifolock\_lock：

(gdb) f 2

(gdb) p r

$13 = (RFifoLock \*) 0x7fe1f6e0ce50

(gdb) p \*r

$14 = {lock = {lock = {\_\_data = {\_\_lock = 0, \_\_count = 0, \_\_owner = 0, \_\_nusers = 1, \_\_kind = 0, \_\_spins = 0, \_\_list = {\_\_prev = 0x0, \_\_next = 0x0}},

\_\_size = '\000' <repeats 12 times>, "\001", '\000' <repeats 26 times>, \_\_align = 0}}, head = 45679, tail = 45681, cond = {cond = {\_\_data = {\_\_lock = 0, \_\_futex = 85,

\_\_total\_seq = 43, \_\_wakeup\_seq = 42, \_\_woken\_seq = 42, \_\_mutex = 0x7fe1f6e0ce50, \_\_nwaiters = 2, \_\_broadcast\_seq = 42},

\_\_size = "\000\000\000\000U\000\000\000+\000\000\000\000\000\000\000\*\000\000\000\000\000\000\000\*\000\000\000\000\000\000\000P\316\340\366\341\177\000\000\002\000\000\000\*\000\000",

\_\_align = 365072220160}}, owner\_thread = {thread = 140608218035968 = 0x7FE1E6E37700 = thread 2}, nesting = 2, cb = 0x7fe1f4ae8c40 <aio\_rfifolock\_cb>, cb\_opaque = 0x7fe1f6e0cdf0}

通过owner\_thread可以看出，rfifolock\_lock目前被线程 Thread 2 (Thread 0x7fe1e6e37700 (LWP 8240)) 持有。

## io线程8240

切换到线程8240：

Thread 2 (Thread 0x7fe1e6e37700 (LWP 8240)):

#0 0x00007fe1ed7d8cec in \_\_lll\_lock\_wait () from /lib/x86\_64-linux-gnu/libpthread.so.0

#1 0x00007fe1ed7d4339 in \_L\_lock\_926 () from /lib/x86\_64-linux-gnu/libpthread.so.0

#2 0x00007fe1ed7d415b in pthread\_mutex\_lock () from /lib/x86\_64-linux-gnu/libpthread.so.0

#3 0x00007fe1f4b84739 in qemu\_mutex\_lock (mutex=mutex@entry=0x7fe1f5223940 <qemu\_global\_mutex>) at util/qemu-thread-posix.c:73

#4 0x00007fe1f485d096 in qemu\_mutex\_lock\_iothread () at /home/jenkins/workspace/Compile/HCI5.2\_Compile/src/app/vtp-qemu-kvm/qemu-2.5.1/cpus.c:1235

#5 0x00007fe1f4830335 in prepare\_mmio\_access (mr=0x7fe1f7743800, mr=0x7fe1f7743800) at /home/jenkins/workspace/Compile/HCI5.2\_Compile/src/app/vtp-qemu-kvm/qemu-2.5.1/exec.c:2500

#6 0x00007fe1f4835407 in address\_space\_stl\_internal (endian=DEVICE\_LITTLE\_ENDIAN, result=0x0, attrs=..., val=16481, addr=140608486299080, as=0x90)

at /home/jenkins/workspace/Compile/HCI5.2\_Compile/src/app/vtp-qemu-kvm/qemu-2.5.1/exec.c:3300

#7 address\_space\_stl\_le (as=as@entry=0x7fe1f7e04510, addr=addr@entry=4276092928, val=val@entry=16481, attrs=..., result=result@entry=0x0)

at /home/jenkins/workspace/Compile/HCI5.2\_Compile/src/app/vtp-qemu-kvm/qemu-2.5.1/exec.c:3349

#8 0x00007fe1f4a35b13 in msi\_send\_message (dev=0x7fe1f7e04310, msg=...) at hw/pci/msi.c:298

#9 0x00007fe1f4a3462c in msix\_notify (dev=<optimized out>, vector=<optimized out>) at hw/pci/msix.c:450

#10 0x00007fe1f48a94ae in virtio\_scsi\_complete\_req (req=0x7fddc40008b0) at /home/jenkins/workspace/Compile/HCI5.2\_Compile/src/app/vtp-qemu-kvm/qemu-2.5.1/hw/scsi/virtio-scsi.c:78

#11 0x00007fe1f48a9643 in virtio\_scsi\_complete\_cmd\_req (req=0x7fddc40008b0) at /home/jenkins/workspace/Compile/HCI5.2\_Compile/src/app/vtp-qemu-kvm/qemu-2.5.1/hw/scsi/virtio-scsi.c:438

#12 virtio\_scsi\_command\_complete (r=<optimized out>, status=0, resid=0) at /home/jenkins/workspace/Compile/HCI5.2\_Compile/src/app/vtp-qemu-kvm/qemu-2.5.1/hw/scsi/virtio-scsi.c:465

#13 0x00007fe1f4a4270c in scsi\_req\_complete (req=0x7fddc4031340, status=<optimized out>) at hw/scsi/scsi-bus.c:1734

#14 0x00007fe1f4a3c663 in scsi\_write\_do\_fua (r=0x7fddc4031340) at hw/scsi/scsi-disk.c:235

#15 0x00007fe1f4957ef4 in dma\_complete (ret=<optimized out>, dbs=0x7fddc4030990) at dma-helpers.c:113

#16 dma\_blk\_cb (opaque=0x7fddc4030990, ret=<optimized out>) at dma-helpers.c:135

#17 0x00007fe1f4b401fb in bdrv\_co\_complete (acb=0x7fddc4031d30) at block/io.c:2114

#18 bdrv\_co\_complete (acb=0x7fddc4031d30) at block/io.c:2110

#19 0x00007fe1f4b97b0a in coroutine\_trampoline (i0=<optimized out>, i1=<optimized out>) at util/coroutine-ucontext.c:80

#20 0x00007fe1ed484020 in ?? () from /lib/x86\_64-linux-gnu/libc.so.6

#21 0x00007fdddd7f7f80 in ?? ()

#22 0x0000000000000000 in ?? ()

这是一协程上下文，同样是在等待qemu\_mutex\_lock (mutex=mutex@entry=0x7fe1f5223940 <qemu\_global\_mutex>)，上面已经分析了，qemu\_global\_mutex 是被线程migrate线程8417持有的，而migrate线程8417要等待的rfifolock\_lock被线程8240持有，相当于，8417和8240相互持有锁而相互等待对方的锁解锁，但是8240是协程上下文，其协程栈内并没有地方持有rfifolock\_lock，那么加锁rfifolock\_lock的代码在哪儿呢？

我们在切换入协程的时候，保存了原有栈的指针到caller\_sp：

(gdb) ptype current

type = struct Coroutine {

CoroutineEntry \*entry;

void \*entry\_arg;

Coroutine \*caller;

void \*caller\_sp;

struct {

struct Coroutine \*sle\_next;

} pool\_next;

struct {

struct Coroutine \*tqh\_first;

struct Coroutine \*\*tqh\_last;

} co\_queue\_wakeup;

struct {

struct Coroutine \*tqe\_next;

struct Coroutine \*\*tqe\_prev;

} co\_queue\_next;

} \*

这个caller\_sp是在进入协程前赋值的原有栈的栈变量：

void qemu\_coroutine\_enter(Coroutine \*co, void \*opaque)

{

Coroutine \*self = qemu\_coroutine\_self();

CoroutineAction ret;

trace\_qemu\_coroutine\_enter(self, co, opaque);

if (co->caller) {

fprintf(stderr, "Co-routine re-entered recursively\n");

abort();

}

co->caller = self;

co->entry\_arg = opaque;

co->caller\_sp = &self;

ret = qemu\_coroutine\_switch(self, co, COROUTINE\_ENTER);

...

}

(gdb) p \*current

$13 = {entry = 0x7fe1f4b42400 <bdrv\_co\_do\_rw>, entry\_arg = 0x0, caller = 0x7fe1e6e375e0, caller\_sp = 0x7fe1e6e36970, pool\_next = {sle\_next = 0x7fe1f8e79930}, co\_queue\_wakeup = {

tqh\_first = 0x0, tqh\_last = 0x7fe1f89fdd48}, co\_queue\_next = {tqe\_next = 0x0, tqe\_prev = 0x0}}

直接dump切换前的栈内容：

(gdb) x /100a current->caller\_sp

0x7fe1e6e36970: 0x7fe1e6e375e0 0x47f6cf4982fdcb00

0x7fe1e6e36980: 0x3 0x7fddc4031de0

0x7fe1e6e36990: 0x7fddc4031db0 0x7fe1f4b3afe9 <qemu\_laio\_completion\_bh+201>

//static void qemu\_laio\_completion\_bh(void \*opaque) -> static void qemu\_laio\_process\_completion(struct qemu\_laio\_state \*s, struct qemu\_laiocb \*laiocb)

0x7fe1e6e369a0: 0x7fe100000000 0x47f6cf4982fdcb00

0x7fe1e6e369b0: 0x7fe1e308cfe0 0x47f6cf4982fdcb00

0x7fe1e6e369c0: 0x0 0x0

0x7fe1e6e369d0: 0x3 0x47f6cf4982fdcb00

0x7fe1e6e369e0: 0x7fe1e308cfe0 0x7fe1f6e0c710

0x7fe1e6e369f0: 0x7fe1f6e0cdf0 0x1

0x7fe1e6e36a00: 0x0 0x7fe1f6e0cf18

0x7fe1e6e36a10: 0x3 0x7fe1f4ae881d <aio\_bh\_poll+125>

//int aio\_bh\_poll(AioContext \*ctx)

0x7fe1e6e36a20: 0x3 0x47f6cf4982fdcb00

0x7fe1e6e36a30: 0x0 0x0

0x7fe1e6e36a40: 0x7fe1f6e0cdf0 0x0

0x7fe1e6e36a50: 0x0 0x7fe1f4afd45b <aio\_dispatch+43>

//bool aio\_dispatch(AioContext \*ctx)

0x7fe1e6e36a60: 0x0 0x47f6cf4982fdcb00

0x7fe1e6e36a70: 0x0 0x7fe1f6e0cdf0

0x7fe1e6e36a80: 0x0 0x0

0x7fe1e6e36a90: 0x0 0x7fe1f4afd6e3 <aio\_poll+371>

//bool aio\_poll(AioContext \*ctx, bool blocking)

0x7fe1e6e36aa0: **0x7fe1f6e0ce50** 0x29

0x7fe1e6e36ab0: 0x0 0x7fe1f4b8501d <qemu\_thread\_get\_self+29>

0x7fe1e6e36ac0: 0x0 0x47f6cf4982fdcb00

0x7fe1e6e36ad0: **0x7fe1f6e0ce50** 0x7fe1f4b96437 <rfifolock\_lock+103>

0x7fe1e6e36ae0: 0x0 0x47f6cf4982fdcb00

0x7fe1e6e36af0: 0x7fe1f6e0c920 0x7fe1f6e0c958

0x7fe1e6e36b00: 0x7ffdd3b789f0 0x7fe1e6e379c0

0x7fe1e6e36b10: 0x7fe1f467d040 <\_rtld\_global> 0x7fe1f494b402 <iothread\_run+114>

// static void \*iothread\_run(void \*opaque)

0x7fe1e6e36b20: 0x0 0x47f6cf4982fdcb00

0x7fe1e6e36b30: 0x0 0x0

0x7fe1e6e36b40: 0x0 0x7fe1ed7d1b50 <start\_thread+208>

如果细心一些，可以看出**0x7fe1f6e0ce50**地址就是上面线程migrate线程8417等待的RFifoLock \*地址，记不得这个地址也没关系，对照源码分析下栈:

static void \*iothread\_run(void \*opaque)

{

IOThread \*iothread = opaque;

...

while (!iothread->stopping) {

**aio\_context\_acquire(iothread->ctx);**

blocking = true;

while (!iothread->stopping && **aio\_poll(iothread->ctx, blocking)**) {

blocking = false;

}

**aio\_context\_release(iothread->ctx);**

}

...

}

在调用aio\_poll之前，有aio\_context\_acquire(iothread->ctx)的调用，这个调用展开就是：

void aio\_context\_acquire(AioContext \*ctx)

{

rfifolock\_lock(&ctx->lock);

}

void rfifolock\_lock(RFifoLock \*r)

{

qemu\_mutex\_lock(&r->lock);

/\* Take a ticket \*/

unsigned int ticket = r->tail++;

if (r->nesting > 0 && qemu\_thread\_is\_self(&r->owner\_thread)) {

r->tail--; /\* put ticket back, we're nesting \*/

} else {

while (ticket != r->head) {

/\* Invoke optional contention callback \*/

if (r->cb) {

r->cb(r->cb\_opaque);

}

qemu\_cond\_wait(&r->cond, &r->lock);

}

}

qemu\_thread\_get\_self(&r->owner\_thread);

r->nesting++;

qemu\_mutex\_unlock(&r->lock);

}

细看iothread\_run与aio\_poll之间的栈变量残留：：

0x7fe1e6e36aa0: 0x7fe1f6e0ce50 0x29

0x7fe1e6e36ab0: 0x0 0x7fe1f4b8501d <qemu\_thread\_get\_self+29>

0x7fe1e6e36ac0: 0x0 0x47f6cf4982fdcb00

0x7fe1e6e36ad0: 0x7fe1f6e0ce50 0x7fe1f4b96437 <rfifolock\_lock+103>

0x7fe1e6e36ae0: 0x0 0x47f6cf4982fdcb00

0x7fe1e6e36af0: 0x7fe1f6e0c920 0x7fe1f6e0c958

0x7fe1e6e36b00: 0x7ffdd3b789f0 0x7fe1e6e379c0

其中蓝色标注部分，都可以和源码重叠，表示，栈上曾经跑过rfifolock\_lock操作。

现在的问题就是，这里lock的参数，是否就是线程migrate线程8417等待的RFifoLock？

如果你不记得地址0x7fe1f6e0ce50，也可以一个一个地址dump出来看，dump每个指针指向的结构，通过地址查找蛛丝马迹，例如0x7fe1f6e0ce50：

(gdb) x /16g 0x7fe1f6e0ce50

0x7fe1f6e0ce50: 0x0 0x100000000

0x7fe1f6e0ce60: 0x0 0x0

0x7fe1f6e0ce70: 0x0 0xb2710000b26f

0x7fe1f6e0ce80: 0x5500000000 0x2b

0x7fe1f6e0ce90: 0x2a 0x2a

0x7fe1f6e0cea0: 0x7fe1f6e0ce50 0x2a00000002

0x7fe1f6e0ceb0: 0x7fe1e6e37700(线程8240的地址) 0x2

0x7fe1f6e0cec0: **0x7fe1f4ae8c40 <aio\_rfifolock\_cb>** 0x7fe1f6e0cdf0

这里的突破点是aio\_rfifolock\_cb，全局搜索代码发现，只有一个地方会操作aio\_rfifolock\_cb地址：

rfifolock\_init(&ctx->lock, aio\_rfifolock\_cb, ctx);

void rfifolock\_init(RFifoLock \*r, void (\*cb)(void \*), void \*opaque)

{

qemu\_mutex\_init(&r->lock);

r->head = 0;

r->tail = 0;

qemu\_cond\_init(&r->cond);

r->nesting = 0;

r->cb = cb;

r->cb\_opaque = opaque;

}

很明显，aio\_rfifolock\_cb是当成cb指针赋值给了RFifoLock结构，

typedef struct {

QemuMutex lock; /\* protects all fields \*/

/\* FIFO order \*/

unsigned int head; /\* active ticket number \*/

unsigned int tail; /\* waiting ticket number \*/

QemuCond cond; /\* used to wait for our ticket number \*/

/\* Nesting \*/

QemuThread owner\_thread; /\* thread that currently has ownership \*/

unsigned int nesting; /\* amount of nesting levels \*/

/\* Contention callback \*/

**void (\*cb)(void \*);** /\* called when thread must wait, with ->lock

\* held so it may not recursively lock/unlock

\*/

void \*cb\_opaque;

} RFifoLock;

对照一下，上面一个值nesting=0x2，owner\_thread里首地址等于0x7fe1e6e37700正好是线程8240的地址，推测这个0x7fe1f6e0ce50很有可能是RFifoLock指针：

(gdb) p \*(RFifoLock\*)0x7fe1f6e0ce50

$23 = {lock = {lock = {\_\_data = {\_\_lock = 0, \_\_count = 0, \_\_owner = 0, \_\_nusers = 1, \_\_kind = 0, \_\_spins = 0, \_\_list = {\_\_prev = 0x0, \_\_next = 0x0}},

\_\_size = '\000' <repeats 12 times>, "\001", '\000' <repeats 26 times>, \_\_align = 0}}, head = 45679, tail = 45681, cond = {cond = {\_\_data = {\_\_lock = 0, \_\_futex = 85,

\_\_total\_seq = 43, \_\_wakeup\_seq = 42, \_\_woken\_seq = 42, \_\_mutex = 0x7fe1f6e0ce50, \_\_nwaiters = 2, \_\_broadcast\_seq = 42},

\_\_size = "\000\000\000\000U\000\000\000+\000\000\000\000\000\000\000\*\000\000\000\000\000\000\000\*\000\000\000\000\000\000\000P\316\340\366\341\177\000\000\002\000\000\000\*\000\000",

\_\_align = 365072220160}}, owner\_thread = {thread = 140608218035968}, nesting = 2, cb = 0x7fe1f4ae8c40 <aio\_rfifolock\_cb>, cb\_opaque = 0x7fe1f6e0cdf0}

这下总该眼熟了吧，我们上面分析过一次，通过owner\_thread分析谁持有锁，两个内容一模一样，再看本来就是migrate线程8417等待的RFifoLock地址。

以上，证明 RFifoLock 被io线程8240持有，然后8240调度进入协程，而协程里需要等待被migrate线程8417持有的qemu\_global\_mutex锁。

## 分析migrate线程源码

再回过头，我们结合源码分析一下migrate线程8417是如何持有qemu\_global\_mutex锁的：

(gdb) bt

#0 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

#1 0x00007fe1f4b84959 in qemu\_cond\_wait (cond=cond@entry=0x7fe1f6e0ce80, mutex=mutex@entry=0x7fe1f6e0ce50) at util/qemu-thread-posix.c:132

#2 0x00007fe1f4b9642a in rfifolock\_lock (r=r@entry=0x7fe1f6e0ce50) at util/rfifolock.c:59

#3 0x00007fe1f4ae8f91 in aio\_context\_acquire (ctx=ctx@entry=0x7fe1f6e0cdf0) at async.c:371

#4 0x00007fe1f4b42fdb in bdrv\_drain\_all () at block/io.c:299

#5 0x00007fe1f485dd15 in do\_vm\_stop (state=RUN\_STATE\_FINISH\_MIGRATE) at /home/jenkins/workspace/Compile/HCI5.2\_Compile/src/app/vtp-qemu-kvm/qemu-2.5.1/cpus.c:737

#6 vm\_stop (state=state@entry=RUN\_STATE\_FINISH\_MIGRATE) at /home/jenkins/workspace/Compile/HCI5.2\_Compile/src/app/vtp-qemu-kvm/qemu-2.5.1/cpus.c:1416

#7 0x00007fe1f485ddfc in **vm\_stop\_force\_state** (state=state@entry=RUN\_STATE\_FINISH\_MIGRATE) at /home/jenkins/workspace/Compile/HCI5.2\_Compile/src/app/vtp-qemu-kvm/qemu-2.5.1/cpus.c:1424

#8 0x00007fe1f4a94d43 in **migration\_completion** (start\_time=<synthetic pointer>, old\_vm\_running=<synthetic pointer>, current\_active\_state=4, s=0x7fe1f51a6200 <current\_migration>)

at migration/migration.c:1614

#9 migration\_thread (opaque=0x7fe1f51a6200 <current\_migration>) at migration/migration.c:1754

#10 0x00007fe1ed7d1b50 in start\_thread () from /lib/x86\_64-linux-gnu/libpthread.so.0

#11 0x00007fe1ed51ba7d in clone () from /lib/x86\_64-linux-gnu/libc.so.6

#12 0x0000000000000000 in ?? ()

(gdb) f 8

#8 0x00007fe1f4a94d43 in migration\_completion (start\_time=<synthetic pointer>, old\_vm\_running=<synthetic pointer>, current\_active\_state=4, s=0x7fe1f51a6200 <current\_migration>)

at migration/migration.c:1614

1614 migration/migration.c: No such file or directory.

(gdb) p s

$1 = (MigrationState \*) 0x7fe1f51a6200 <current\_migration>

(gdb) p \*s

$2 = {bandwidth\_limit = 8589934592, bytes\_xfer = 0, xfer\_limit = 0, thread = {thread = 140590679303936}, cleanup\_bh = 0x7fe1f866a750, file = 0x7fe1fb5c2030, parameters = {1, 8, 2, 20, 10},

**state = 4**, params = {blk = false, shared = false}, rp\_state = {from\_dst\_file = 0x0, rp\_thread = {thread = 0}, error = false}, mbps = 943.80247272727274, total\_time = 351512593,

downtime = 0, expected\_downtime = 59, dirty\_pages\_rate = 1713, dirty\_bytes\_rate = 7016448, enabled\_capabilities = {true, false, true, true, true, false, false},

xbzrle\_cache\_size = 1073741824, setup\_time = 89, dirty\_sync\_count = 4, start\_postcopy = false, migration\_thread\_running = true, src\_page\_req\_mutex = {lock = {\_\_data = {\_\_lock = 0,

\_\_count = 0, \_\_owner = 0, \_\_nusers = 0, \_\_kind = 0, \_\_spins = 0, \_\_list = {\_\_prev = 0x0, \_\_next = 0x0}}, \_\_size = '\000' <repeats 39 times>, \_\_align = 0}}, src\_page\_requests = {

sqh\_first = 0x0, sqh\_last = 0x7fe1f51a62e8 <current\_migration+232>}, last\_req\_rb = 0x0}

./qapi-types.h:816: MIGRATION\_STATUS\_ACTIVE = 4,

static void migration\_completion(MigrationState \*s, int current\_active\_state,

bool \*old\_vm\_running,

int64\_t \*start\_time)

{

int ret;

if (s->state == MIGRATION\_STATUS\_ACTIVE) {

**qemu\_mutex\_lock\_iothread();** //这里持有锁

\*start\_time = qemu\_clock\_get\_ms(QEMU\_CLOCK\_REALTIME);

qemu\_system\_wakeup\_request(QEMU\_WAKEUP\_REASON\_OTHER);

\*old\_vm\_running = runstate\_is\_running();

ret = global\_state\_store();

if (!ret) {

**ret = vm\_stop\_force\_state(RUN\_STATE\_FINISH\_MIGRATE);** //代码在这里vm\_stop\_force\_state

if (ret >= 0) {

qemu\_file\_set\_rate\_limit(s->file, INT64\_MAX);

qemu\_savevm\_state\_complete\_precopy(s->file, false);

}

}

qemu\_mutex\_unlock\_iothread();

if (ret < 0) {

goto fail;

}

}

...

}

s->state = 4，结合代码，可以看到，在进入vm\_stop\_force\_state前，有qemu\_mutex\_lock\_iothread调用，此函数用于加锁了qemu\_global\_mutex。

## 结论

从堆栈上都找到证据，从代码上找到理论支撑，那这个死锁就可以明确下来。

(gdb) info threads

Id Target Id Frame

22 Thread 0x7fddd17fa700 (LWP 8417) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

21 Thread 0x7fddd1ffb700 (LWP 8409) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

20 Thread 0x7fddd2ffd700 (LWP 8397) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

19 Thread 0x7fddd27fc700 (LWP 8401) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

18 Thread 0x7fddd37fe700 (LWP 8385) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

17 Thread 0x7fddd8ffa700 (LWP 8378) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

16 Thread 0x7fddd3fff700 (LWP 8383) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

15 Thread 0x7fddda7fd700 (LWP 8360) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

14 Thread 0x7fddd97fb700 (LWP 8373) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

13 Thread 0x7fdddc3ff700 (LWP 8394) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

12 Thread 0x7fdddbbfe700 (LWP 8395) 0x00007fe1ed510e33 in poll () from /lib/x86\_64-linux-gnu/libc.so.6

11 Thread 0x7fddddffa700 (LWP 8324) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

10 Thread 0x7fdddd7f9700 (LWP 8325) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

9 Thread 0x7fddde7fb700 (LWP 8323) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

8 Thread 0x7fdddeffc700 (LWP 8322) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

7 Thread 0x7fdddf7fd700 (LWP 8321) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

6 Thread 0x7fdddfffe700 (LWP 8320) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

5 Thread 0x7fe1e5433700 (LWP 8318) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

4 Thread 0x7fe1e4c32700 (LWP 8319) 0x00007fe1ed7d62d4 in pthread\_cond\_wait@@GLIBC\_2.3.2 () from /lib/x86\_64-linux-gnu/libpthread.so.0

3 Thread 0x7fe1e7638700 (LWP 8005) 0x00007fe1ed5182f9 in syscall () from /lib/x86\_64-linux-gnu/libc.so.6

2 Thread 0x7fe1e6e37700 (LWP 8240) 0x00007fe1ed7d8cec in \_\_lll\_lock\_wait () from /lib/x86\_64-linux-gnu/libpthread.so.0

\* 1 Thread 0x7fe1f457caa0 (LWP 8003) 0x00007fe1ed7d8cec in \_\_lll\_lock\_wait () from /lib/x86\_64-linux-gnu/libpthread.so.0

**线程22为migrate线程，持有qemu\_global\_mutex但是需要获取RFifoLock被线程2持有；**

**线程2为io线程切换后的协程栈，原io线程持有RFifoLock，但协程需要等待被线程22持有的qemu\_global\_mutex。**

2016/7/14