



# 系统 IO 快速排查指南

版本号: V0.3  
发布日期: 2023.03.08

## 版本历史

版本号	日期	制/修订人	内容描述
V0.1	2019.11.23	AWA1087	初始版本
V0.2	2020.01.13	AWA1087	新增默认值说明和参考文档说明等细节
V0.3	2023.03.08	AWA1579	新增新内核配置说明和 dram 调频等排除项



# 目 录

<b>1 前言</b>	<b>1</b>
1.1 概述	1
1.2 读者对象	1
<b>2 文件系统异常</b>	<b>2</b>
2.1 文件系统 crash	2
2.1.1 随机性 crash	2
2.1.1.1 排查 ddr	3
2.1.1.2 排查 cpu	3
2.1.1.3 排查供电正常	5
2.1.1.4 软件问题	5
2.1.2 固定位置 crash	5
2.1.2.1 排除内核原生代码	5
2.1.2.2 排查驱动代码	6
2.1.3 排查设备驱动	6
2.1.4 排查 flash 中数据是否异常	6
2.1.5 排查文件系统	8
<b>3 文件系统性能</b>	<b>9</b>
3.1 读写速度慢	9
3.1.1 排查设备速度	9
3.1.2 排查应用写行为	9
3.1.3 排查 mount 参数	9
3.1.4 排查 linux IO 参数	10
3.1.5 排查文件碎片	10
3.2 系统卡顿 ANR	11
3.2.1 排查内存页迁移	11
3.2.2 排查内存使用情况	11
3.2.3 排查 bypass 模式	11
3.2.4 排查 cpu 使用率	12
3.2.5 排查 cpu 中断	12
3.2.6 排查死锁	13
<b>4 总结</b>	<b>15</b>

## 插 图

图 2-1	TigerDebug 工具	7
图 2-2	H2testw 工具	7
图 3-1	windows-PC-SmartDefrag 工具	10
图 3-2	kswap 进程	11
图 3-3	bypass_depth 配置	12
图 3-4	cpu 中断	13



# 1 前言

## 1.1 概述

- 本文档用于快速排查系统 io 问题，包括系统性能问题，文件系统问题，IO 引起的系统卡顿问题等；
- 本文档提供的快速排查方案模型如下：

常规问题场景 + 基础软件配置排查 + 软件快速调试排查

- 本文档仅用于帮助开发人员快速排查和解决初级问题问题，如果按照文档中的指导仍无法解决问题，请联系全志原厂 FAE 进一步协助解决，并提供说明是否按照文档中的排查方案进行过必要的排查，提供排查过程的必要记录和数据

## 1.2 读者对象

本文档主要适用于以下工程师：

- io 技术基础的 RD 研发人员
- io 技术基础的 FAE 工程人员

## 2 文件系统异常

### 2.1 文件系统 crash

#### 问题现象

内核打印报错 oops/panic, 包含如下关键字:

```
Internal error: Oops: 817 [#1] PREEMPT SMP ARM  
end Kernel panic - not syncing: Fatal exception
```

#### 问题分析

当 linux kernel 遇到无法处理的异常的时候, 就会报错 oops、panic。

内核 oops、panic 可能是下面几种原因:

- 硬件问题

- ddr异常
- cpu异常
- 供电异常

- 软件问题

- 内核代码逻辑bug
- 内存异常篡改

系统 crash 的问题, 我们分为随机性 crash 和固定位置 crash 这两类, 下面讲这两类 crash 的排查思路。

#### 2.1.1 随机性 crash

如果是量产方案, 出现系统随机性的 crash, 这种情况更多的会是硬件问题, 可以从 ddr、cpu、供电三个方向排查。

### 2.1.1.1 排查 ddr

#### 步骤 1

memtester 测试实验。

可以通过 memtester 测试来验证是否 ddr 存在异常。如果测试存在报错，则需找 dram 专业人士调整 dram 参数或者排查 dram 硬件问题。

memtester测试大小随方案dram大小而定，如1G方案可以使用3个128M的测试  
memtester 128M &

#### 步骤 2

排除 dram 调频策略实验

由于在 androidS, androidT 及以上平台支持了 dram 调频策略；会对 dram 进行调频，可能会导致会运行在 dram 不稳定的频率等。

可以移除 dram 调频策略排除 dram 调频导致的不稳定可能。

获取persist.vendor.power.devfs.enabled是否为0；为0则是没有使能dram调频，可以直接排除dram调频的影响。  
失能dram调频的方法：  
setprop persist.vendor.power.devfs.enabled 0  
重启

#### 步骤 3

减低 dram 频率实验。

减低 dram 频率到 360M，可以保证 dram 的稳定性，可能 dram 在板子上运行不稳定或者参数不适配等。如果此实验正常下工作正常，则需找 dram 专业人士调整 dram 参数或者排查 dram 硬件问题。

dram频率修改方法  
sys\_config中的dram\_para参数  
[dram\_para]  
dram\_clk=360

### 2.1.1.2 排查 cpu

cpu 异常可能是问题 ic 是不良品、或者是性能不足、或者是焊接不良等。

#### 步骤 1

排查是否为 ic 性能不足。可以做定频定核提压实验，如果在此实验下工作正常，再排查供电是否正常，如果供电正常，则可定论为 ic 性能不足，需反馈问题给硬件同事针对性能不足 ic 做出筛选。

## 步骤 1.1

定频提压：配置 sys\_config 中的**所有** dvfs(改方法在 linux5.15 以上的内核已经失效，使用 dts 的修改) table 中**所有**频点为 1G 或者修改 dts 中的 opp 表所有频点为 1G，并使用最高频点电压。

```
L_LV_count = 8
-L_LV1_freq = 1104000000
+L_LV1_freq = 1008000000
L_LV1_volt = 1300
L_LV2_freq = 1008000000
-L_LV2_volt = 1200
+L_LV2_volt = 1300
-L_LV3_freq = 912000000
-L_LV3_volt = 1120
+L_LV3_freq = 1008000000
+L_LV3_volt = 1300
...
```

如果 dvfs 配置在 dts 中的 opp\_table，那么同理的，按上面的方法修改 dts 中的 opp 表即可。（具体可参考《Linux\_CPUFREQ\_开发指南》）

提高 40mv sys 电压。查看硬件原理图 sys 是用的哪路电，再修改 sys\_config 中的供电配置，如 sys 是使用 dc3 并默认值为 900mv，可做如下修改：

```
[power_sply]
dc3_vol=100940
```

## 步骤 1.2

关闭温控系统。

```
make ARCH=arm menuconfig或者make ARCH=arm64 menuconfig或者./build.sh menuconfig
设置THERMAL=n.具体可参考《Linux_Thermal_开发指南》
```

## 步骤 1.3

打开所有核，并锁定。

```
如果ic为四核cpu，则锁定四核：
echo 4 > /sys/kernel/autohotplug/lock
注：具体使用方法可参考《Linux_CPUHOTPLUG_开发指南》文档
```

## 步骤 2

排查是否为焊接不良。可重新补焊验证。

## 步骤 3

验证 ic 是否为不良品。如果 crash 问题为单颗 ic 问题，并且步骤 1、2 均不能查出问题，可通过替换 ic 的操作验证是否为 ic 不良。如果是不良 ic，交由硬件同事确认，并需增加筛选条件将此类不良品筛选。

### 2.1.1.3 排查供电正常

#### 步骤 1

测量 cpu 供电电压纹波看是否符合规格。

#### 步骤 2

用示波器抓取 cpu 电压，看 cpu 供电电压是否存在抖动。

### 2.1.1.4 软件问题

随机性 crash 问题一般都是与硬件相关。如果上述排查实验过后依然找不出问题，可能是软件问题了。软件方向可以从代码逻辑 bug 和内存篡改方向排查。

## 2.1.2 固定位置 crash

固定位置 crash 的判断标准：如果 3 次以上 crash 情况的都是在同一个位置。此类固定位置 crash 的问题一般为软件问题，可采用 addr2line、objdump 等工具进行正面分析，查明原因。

### 2.1.2.1 排除内核原生代码

如果是内核原生代码，基于某种测试下出现，请在开源社区查看 Bug 记录。

eg:

```
[32m[ 378.048331] [33mWARNING[0m: CPU: 1 PID: 7329 at lib/percpu-refcount.c:322 percpu_ref_kill_and_confirm+0x64/0x13c
[32m[ 378.048342] [0mpercpu_ref_kill_and_confirm called more than once on blk_queue_usage_counter_release!
[32m[ 378.048371] [33mModules linked in[0m: xr829 gslX680new xradio_btfdi xradio_btldm vin_v4l2 gc0310_mipi gc2355_mipi gc030a_mipi gc2385_mipi vin_io videobuf2_v4l2 videobuf2_dma_contig videobuf2_memops videobuf2_core mali(O)
[32m[ 378.048379] [33mCPU[0m: 1 PID: 7329 Comm: Binder:1779_6 Tainted: G O 4.9.170 #33
[32m[ 378.048381] [33mHardware name[0m: sun8iw15
[32m[ 378.048403] [0m[<c0110a68>] (unwind_backtrace) from [<c010c5ac>] (show_stack+0x20/0x24)
[32m[ 378.048415] [0m[<c010c5ac>] (show_stack) from [<c04985d0>] (dump_stack+0x78/0x94)
[32m[ 378.048426] [0m[<c04985d0>] (dump_stack) from [<c0129b98>] (__warn+0xdc/0x110)
[32m[ 378.048434] [0m[<c0129b98>] (__warn) from [<c0129c20>] (warn_slowpath_fmt+0x54/0x74)
[32m[ 378.048442] [0m[<c0129c20>] (warn_slowpath_fmt) from [<c04b2374>] (percpu_ref_kill_and_confirm+0x64/0x13c)
[32m[ 378.048452] [0m[<c04b2374>] (percpu_ref_kill_and_confirm) from [<c0480e68>] (blk_mq_freeze_queue_start+0x64/0x7c)
[32m[ 378.048461] [0m[<c0480e68>] (blk_mq_freeze_queue_start) from [<c0480e9c>] (blk_freeze_queue+0x1c/0x28)
[32m[ 378.048468] [0m[<c0480e9c>] (blk_freeze_queue) from [<c0480ec0>] (blk_mq_freeze_queue+0x18/0x1c)
[32m[ 378.048479] [0m[<c0480ec0>] (blk_mq_freeze_queue) from [<c058f6d4>] (lo_release+0xa4/0xf0)
[32m[ 378.048491] [0m[<c058f6d4>] (lo_release) from [<c02d1968>] (__blkdev_put+0x140/0x1f8)
[32m[ 378.048500] [0m[<c02d1968>] (__blkdev_put) from [<c02d228c>] (blkdev_put+0x114/0x124)
```

```
[32m[ 378.048508] [0m[<c02d228c>] (blkdev_put) from [<c02d233c>] (blkdev_close+0x28/0x30)
[32m[ 378.048518] [0m[<c02d233c>] (blkdev_close) from [<c0296334>] (__fput+0xf4/0x1d0)
[32m[ 378.048527] [0m[<c0296334>] (__fput) from [<c0296480>] (____fput+0x18/0x1c)
[32m[ 378.048537] [0m[<c0296480>] (____fput) from [<c0148fc4>] (task_work_run+0xc8/0xd8)
[32m[ 378.048546] [0m[<c0148fc4>] (task_work_run) from [<c010bc70>] (do_work_pending+0xac/0xcc)
[32m[ 378.048556] [0m[<c010bc70>] (do_work_pending) from [<c0107dac>] (slow_work_pending+0xc/0x20)
[32m[ 378.048567] [31mblk_mq_freeze_queue_wait 76 wait zero @ 1 Binder:1779_6
```

### 2.1.2.2 排查驱动代码

如果是内核 crash，log 里有关于具体某驱动打印，请 addr2line、objdump 等工具进行正面分析，或提交 readmine 给具体驱动负责人。文件系统只读文件系统挂载时，有一个 errors 参数，一般选择 ro(只读)，当文件系统报错时，会被重新挂载为只读或者 panic，当然目前平台 error 参数基本都是只读。

### 2.1.3 排查设备驱动

查看在文件系统被挂载为只读之前，对应的设备驱动是否有异常 log，启动常见的设备包括 sd，emmc，nand，nor 等。如发现 emmc，nand 或者 nor 驱动异常 log，可先参考对应的排查指南进行快速定位，包括《NAND 快速排查指南.doc》《MMC 量产问题快速排查指南.doc》，如无进展可联系驱动负责人处理。

eg:

```
[ 18.570632] sunxi-mmc 1c0f000.sdmmc: smc 1 p0 err, cmd 24, WR EBE !!
[ 18.577837] sunxi-mmc 1c0f000.sdmmc: *****retry:start*****
[ 18.585349] sunxi-mmc 1c0f000.sdmmc: REG_DRV_DL: 0x00030000
[ 18.585388] sunxi-mmc 1c0f000.sdmmc: REG_SD_NTSR: 0x81710110
[ 18.585448] sunxi-mmc 1c0f000.sdmmc: *****retry:re-send cmd*****
[ 42.163782] vidioc_g_fmt_vid_cap: signal is not locked.
[ 42.195251] vidioc_s_fmt_vid_cap:dev_sel=0
[ 42.500101] hpd :tvd_num[] = 4,g_status[] =400,tvd_onsignal=0
[ 50.525527] sunxi-mmc 1c0f000.sdmmc: smc 1 p0 err, cmd 25, WR EBE !!
[ 50.532749] sunxi-mmc 1c0f000.sdmmc: *****retry:start*****
[ 50.532852] sunxi-mmc 1c0f000.sdmmc: REG_DRV_DL: 0x00030000
[ 50.532879] sunxi-mmc 1c0f000.sdmmc: REG_SD_NTSR: 0x81710110
[ 50.532969] sunxi-mmc 1c0f000.sdmmc: *****retry:re-send cmd*****
[ 50.533050] sunxi-mmc 1c0f000.sdmmc: smc 1 p0 err, cmd 25, WR RTO !!
```

### 2.1.4 排查 flash 中数据是否异常

如果报错的文件系统本身就是只读文件系统。

eg:

SQUASHFS error: Unable to read fragment cache entry \[aeabc\  
 SQUASHFS error: Unable to read page, block aeabc, size 13188。

使用 TigerDebug 工具（参考 TigerDebug 使用手册.pdf 文档）将出问题机器的所有逻辑区镜像 dump 出来，并连同 dump 时串口日志文件、压测日志文件、原始镜像文件、sys\_partition.fex 文件发给全志工程师分析。



图 2-1: TigerDebug 工具

如果出错的是非只读文件，USB 设备/sd 卡设备文件，重点查看对应设备驱动是否有异常错误，并在 PC 使用 H2testw 测试设备读写是否可正常，如该工具报错，说明设备存在问题，不建议使用。



图 2-2: H2testw 工具

## 2.1.5 排查文件系统

如果非只读文件系统有异常 log。

eg:

```
+-----+
| fat_free导致内核挂掉问题:                |
|                                           |
| fat(Thread-63:363)-&fat_free(299): skip = 0, inode-&i_nlink = |
| 0                                         |
| .....                                    |
| [12:30:37][ 928.675501] \[&c01da3f8&]      |
| (fat_truncate_blocks+0xac0x358) from \[&c01db508&] |
| (fat_evict_inode+0x48/0x6c)                |
| [12:30:37][ 928.675501] \[&c01db508&]      |
| (fat_evict_inode+0x48/0x6c) from \[&c0128df4&] |
| (evict+0xac/0x170)                         |
| [12:30:37][ 928.675501] \[&c0128df4&] (evict+0xac/0x170) |
| from \[&c0129848&] (iput+0x144/0x14c)      |
| [12:30:37][ 928.675501] \[&c0129848&] (iput+0x144/0x14c) |
| from \[&c011ff50&] (do_unlinkat+0x1f8/0x274) |
| [12:30:37][ 928.675501] \[&c011ff50&]      |
| (do_unlinkat+0x1f80x274) from \[&c0120030&] |
| (SyS_unlink+0x24/0x28)                    |
| [12:30:37][ 928.675501] \[&c0120030&]      |
| (SyS_unlink+0x24/0x28) from \[&c000f9e0&] |
| (ret_fast_syscall+0x0/0x30)               |
+-----+
```

查考章节 1 排查。

## 3 文件系统性能

### 3.1 读写速度慢

遇到系统读写性能问题，比如：写卡丢帧，MTP copy 速度不达标等问题

#### 3.1.1 排查设备速度

使用 iosimu, dd 等测试工具测试设备性能是否符合预期，一般情况下 dd 测试连续读写性能，iosimu 可支持 flash 连续读写，随机读写性能测试工具，测试中注意，测试数据一般要大于内存大小。

```
iosimu测试，可以执行
/data/test/Run_Performance_test.sh XXX指定目录----性能测试
/data/test/Run_Stress_test.sh XXX----读写老化测试（可跑72小时）
/data/test/run_fulldisk_write.sh XXX----全盘读写测试---（一般在测试性能之前运行）
```

#### 3.1.2 排查应用写行为

如果写性能不达标，但设备速度测试符合预期，就需要排查应用写操作是否使能 DirectIO 或者频繁调用 sync，可检查应用 open 是否使能了 O\_DIRECT 或者 O\_SYNC 选项，写过程是否调用 sync/fsync/fdatasync 等操作。这些操作随可保证数据安全性，但是会对写操作性能有一定的影响。

#### 3.1.3 排查 mount 参数

检测 mount 参数，比如 ext4 文件系统 data 参数，该参数具体区分如下，会影响系统性能，一般使用 ordered 模式即可。

```
ext4有三种data模式：ordered,journal,writeback。文件在ext4中分两部分存储，一部分是文件的metadata，另一部分是data。metadata和data的操作日志journal也是分开管理的。这取决于mount ext4时的data参数
这三种mode的区别是：
1. data=journal
   在将data写入文件系统前，必须等待metadata和data的journal已经落盘了。性能最差，并且不支持文件操作的delalloc，O_DIRECT flag（参考 man open）。
2. data=ordered
   这个模式不记录data的journal，只记录metadata的journal日志，但是在写metadata的journal前，必须先确保data已经落盘。
```

3. data=writeback  
不记录data journal，仅记录metadata journal。并且不保证data比metadata先落盘。

### 3.1.4 排查 linux IO 参数

Linux 系统有一些 IO 参数调试接口，这些接口在不同场景，不同内存方案中做调整，已达到性能和流畅性平衡。

/proc/sys/vm/dirty\_writeback\_centisecs: 回刷线程唤醒间隔时间，单位ms，默认500ms  
/proc/sys/vm/dirty\_expire\_centisecs: 回刷线程判断，若文件上次回刷离现在的时间超过此值才会回刷，默认值200ms  
/proc/sys/vm/dirty\_background\_ratio: 当文件系统缓存脏页数量达到系统内存百分之多少时触发pdflush等后台回写进程运行，将一定缓存的脏页异步地刷入存储，默认值5  
/proc/sys/vm/dirty\_ratio: 当文件系统缓存脏页数量达到系统内存百分之多少时，系统阻塞应用操作，回刷脏数据，默认值20。  
/sys/block/xxx/queue/scheduler: 驱动选择的IO调度策略，cfq, noop, deadline, 这三个不同的策略，针对性不对，对于多进程交付频繁大的系统，如android可采用cfq调度，强调多个进程间之间的公平竞争；noop是最简单的调度，按照先来先处理的策略去处理请求，适合使用场景单一，如非触屏的行车记录仪，运动DV等产品；deadline是保证不会有请求被饿死，对读请求更友好的一种调度策略，会对请求进行合并和排序。（linux4.9及其以下内核适用，linux5.4开始使用的是mq-deadline, bfq, none,可以简单的认为cfq=bfq, noop=none, deadline=mq-deadline）  
/sys/block/xxx/queue/max\_sectors\_kb: 设备允许的最大请求大小，bio的最大限制,默认128  
/sys/block/xxx/queue/nr\_requests: 磁盘队列长度。默认只有 128 个队列,可以提高到 512 个.会更加占用内存,但能更加多的合并读写操作,提高flash带宽利用率  
/sys/block/sda/queue/read\_ahead\_kb: 一次提前读多少内容,无论实际需要多少.默认一次读 128kb 远小于要读的,设置大些对读大文件非常有用,可以有效的减少读 seek 的次数  
\*\*xxx 是指对应的设备名\*\*, mmc--mmcblk0, nand---nand0

### 3.1.5 排查文件碎片

其中 fat、ntfs 文件系统和 ext4 文件系统存在检测碎片工具和在线碎片整理工具。fat 和 ntfs 文件系统可以用 windows PC SmartDefrag 工具查看文件碎片。



图 3-1: windows-PC-SmartDefrag 工具

## 3.2 系统卡顿 ANR

在写测试的同时，操作 UI 界面，系统出现卡顿，甚至出现“假死”，点击运行 app 没反应，android logcat 会报 ANR，直到测试数据写入完成后，才恢复正常。

### 3.2.1 排查内存页迁移

文件系统数据写过程中，会把用户态数据 copy 到内核态，这一过程申请的内存都带有 \_\_GFP\_WRITE | \_\_GFP\_MOVABLE | \_\_GFP\_HIGHUSER 标记，而系统在使能 CMA 情况下，会对标记 \_\_GFP\_MOVABLE 的优先分配 CMA 内存，导致 Android 应用 ion\_alloc 申请缓慢（1~4s），系统 ANR。可使用 vmstat 或 cpu\_monitor 进行内存检测，查看 kswap 进程是否异常。如果是可打上链接补丁测试是否有改善。<http://gerrit.allwinnertech.com:8081/#/c/lichee/linux-4.9/+90095/>

Mem State (unit:MB)															
Memory-Total:	1981	Swap-Total:	1486	Swap-Free:	1486	Uma-Total:	256895	Uma-Free:	256895						
Buffer:	18	KernStack:	48	Shmem:	0	Gpu:	0								
Ion cma:	0	caverout:	0	sytem:	0	contig:	0								
Total-used:	1861	Total-freed:	26	Total-lost:	93										
Anon	Slab	Cache	Sysfre	Cmafree	pgal	pgfree	pgfalt	fimap	kswap	dirty	whack	rbio	wbio		
252	113	1428	22	4!	27963	27702	383	0	31	12527	2873	49684	80182		
253	121	1417	22	4!	29247	29704	1876	0	26	12931	2832	51456	12489		
255	129	1404	23	7!	26262	28299	2032	0	25	11496	3294	45872	11448		
256	136	1334	22	70!	27615	43585	1109	0	9	12050	2844	47976	8420		
257	139	1363	22	38!	9857	1449	749	0	3	4070	2832	16228	13517		
258	142	1379	23	17!	6582	1767	763	0	2	2590	3256	10364	13496		
259	144	1384	24	8!	6784	4766	800	0	6	2642	2844	10752	12400		
260	146	1383	22	8!	7459	6842	669	0	10	3101	3068	12032	10430		
253	147	1380	30	12!	3577	6705	1647	0	6	1158	1571	4540	8432		
249	149	1392	22	8!	7504	4762	411	0	6	3154	3107	12368	14536		
250	151	1389	23	8!	7112	7155	776	0	10	2920	3059	11776	7375		
250	153	1388	23	8!	6938	6635	557	0	9	2931	3624	11520	12516		
251	155	1385	22	8!	7406	7367	1280	0	11	2939	3086	11776	13407		
251	157	1384	21	8!	6328	6197	235	0	10	2706	3090	10752	5305		
252	159	1382	24	7!	6314	7300	491	1	10	2566	2314	10364	11504		
252	161	1379	24	8!	5640	5767	200	0	11	2390	3356	9472	7353		

图 3-2: kswap 进程

### 3.2.2 排查内存使用情况

可使用 vmstat 或 cpu\_monitor 进行内存检测，查看是否是剩余内存不足，kswap 进程异常导致系统卡顿，具体可参考《量产系统问题性问题快速排查指南》

### 3.2.3 排查 bypass 模式

驱动如果使能 IO bypass 模式，会使读写操作不经过任何 IO 调度策略，该操作可能会带来 IO 读写性能的提高，但是如果系统不经过 IO 调度，会导致大量读写数据时，系统卡顿现象。可在环境中

nand 代码里搜索 bypass\_depth 变量，把对该变量操作的地方注释掉，测试是否有效，如有效请上该补丁（已在 A50/H3 Q 项目中测试过）。

```
gd->private_data = dev;
dev->disk = gd;
tr->rq->bypass_depth++;
gd->queue = tr->rq;

dev->disable_access = 0;
dev->readonly = 0;
dev->writeonly = 0;
mutex_init(&dev->lock);
device_add_disk(ndfc_dev->parent, gd);

return 0;
```

图 3-3: bypass\_depth 配置

### 3.2.4 排查 cpu 使用率

使用 top 命令，查看卡顿过程中，是否有线程异常占用 cpu。如存在，就联系对应模块负责人。

### 3.2.5 排查 cpu 中断

#### 问题现象

MTP 拷贝大于 4G 文件时卡顿，或其他使用场景中的卡顿

#### 步骤 1

cat /proc/interrupts 查看 cpu 中断是否正常

eg: 图 cpu0 出现大量的 sunxi\_usb\_udc 中断

	CPU0	CPU1	CPU2	CPU3	
2:	135577	0	0	0	- hnsdh_sdmmc
4:	420	0	0	0	- bluetooth hostwake
27:	124161304	126326630	285246791	283369879	GIC arch_timer
32:	1	0	0	0	GIC uart0
33:	1935	0	0	0	GIC uart1
36:	77470459	0	0	0	GIC twi0
45:	154701768	0	0	0	GIC sunxi-gpadc
52:	0	0	0	0	GIC audio_jack_irq
54:	7071	0	0	0	GIC sunxi_usb_udc
57:	0	0	0	0	GIC ehci_hcd:usb1
58:	0	0	0	0	GIC ohci_hcd:usb2
65:	736	0	0	0	GIC sunxi-mmc
66:	712029	0	0	0	GIC sunxi-mmc
67:	429375	0	0	0	GIC sunxi-mmc
70:	0	0	0	0	GIC 3002000.dma-controller
71:	74271510	0	0	0	GIC nanohub-irq1
75:	30329400	0	0	0	GIC sunxi_timer
78:	0	0	0	0	GIC PIN_GRP
79:	167	0	0	0	GIC PIN_GRP
80:	0	0	0	0	GIC PIN_GRP
81:	13	0	0	0	GIC PIN_GRP
86:	11968550	0	0	0	GIC dispaly
103:	24745094	0	0	0	GIC gpu
104:	23100035	0	0	0	GIC gpu
105:	0	0	0	0	GIC gpu
106:	0	0	0	0	GIC ss
108:	0	0	0	0	GIC cedar_dev
109:	0	0	0	0	GIC googleup9_dev
111:	135996	0	0	0	GIC PIN_GRP
112:	22	0	0	0	GIC axp803_irq_chip
117:	0	0	0	0	GIC rtc
247:	167	0	0	0	- sdc0 cd
307:	1	0	0	0	- key0_gpio
308:	11	0	0	0	- key1_gpio
309:	1	0	0	0	- key2_gpio
IP10:	280972223	345759693	292046350	238938221	Rescheduling interrupts
IP11:	57	57	87	77	Function call interrupts
IP12:	6	5	6	7	Single function call interrupts
IP13:	0	0	0	0	CPU stop interrupts
IP14:	0	922018	114182	184948	Timer broadcast interrupts
Err:	0				

图 3-4: cpu 中断

## 3.2.6 排查死锁

### 问题现象

- 使用串口，按键，屏幕等外设都没有反应
- 某应用进程阻塞

### 步骤 1

利用sysrq-trigger  
dump出内核所有线程的堆栈信息，帮助我们分析系统中的用锁情况。  
echo t > /proc/sysrq-trigger  
在串口卡死，无法输入时，可以使用快捷键（ctrl+break 再按t）。  
注：如果需要使用快捷键，在进行问题复现前，需确保/proc/sys/kernel/sysrq节点值设置为1；

### 步骤 2

打开锁的 debug 调试选项，然后复现问题。在不清楚是哪类死锁问题的时候可以把 **kernel hacking** 中所有的锁 debug 选项都打开，然后再复现问题。如果是死锁，内核会把死锁类型、死锁处的堆栈等信息 dump 出来，帮助开发人员快速定位问题。内核锁相关配置如下：

```
CONFIG_LOCKUP_DETECTOR=y      死锁检查器
CONFIG_DEBUG_LOCK_ALLOC=y
CONFIG_PROVE_LOCKING=y
CONFIG_LOCKDEP=y
CONFIG_DEBUG_LOCKDEP=y
CONFIG_DETECT_HUNG_TASK=y     hung task检查机制
CONFIG_DEFAULT_HUNG_TIMEOUT=120
CONFIG_DEBUG_MUTEXES=y        mutex锁调试
CONFIG_DEBUG_RT_MUTEXES=y
CONFIG_DBEUG_SPINLOCK=y       spin lock锁调试
CONFIG_LOCK_STAT              锁使用信息
CONFIG_DEBUG_ATOMIC_SLEEP      debug原子环境是否会睡眠
CONFIG_DBEUG_LOCKING_API_SELFTESTS  锁自检
或者在linux5.15以上版本内核打开AW_LOCK_DEBUG;该配置会打开以上所有的锁调试的相关配置。
```



## 4 总结

---

经过以上排查，问题依然得不到澄清，请联系全志对应模块负责工程师处理，并反馈排查实验和初步结论。






## 著作权声明

版权所有 ©2023 珠海全志科技股份有限公司。保留一切权利。

本档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本档内容的部分或全部，且不得以任何形式传播。

## 商标声明

、、**全志科技**、（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

## 免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本档作为使用指导仅供参考。由于产品版本升级或其他原因，本档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本档中提供准确的信息，但并不确保内容完全没有错误，因使用本档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。