

Unisoc Confidential For hiar

稳定性调试指导手册

WWW.UNISOC.COM

紫光展锐科技



修改历史

版本号	日期	注释
V1.0	2020/10/10	初稿
V1.1	2021/02/26	<ul style="list-style-type: none">• 增加稳定性问题类型及分析• 新增Crash工具命令简介• 新增典型案例及分析方法

关键字

关键字：稳定性、Native Crash、Java Crash、ANR、重启、定屏

Unisoc Confidential For Internal Use Only

Unisoc Confidential For hiar

目录



01 概述

02 稳定性问题分类及分析

03 Crash工具命令简介

04 典型案例及分析方法

Unisoc Confidential For hiar

01

概述



● 稳定性问题简介

稳定性问题通常在CO描述中会出现“定屏”、“冻屏”、“黑屏”、“重启”、“sysdump”、“Framework Crash”、“Kernel Crash”、“应用退出”、“无法开机”等字样。

● 稳定性从系统架构上又可分为

- Android FW稳定性
- 内核稳定性

Unisoc Confidential For hiar

02

稳定性问题 分类及分析



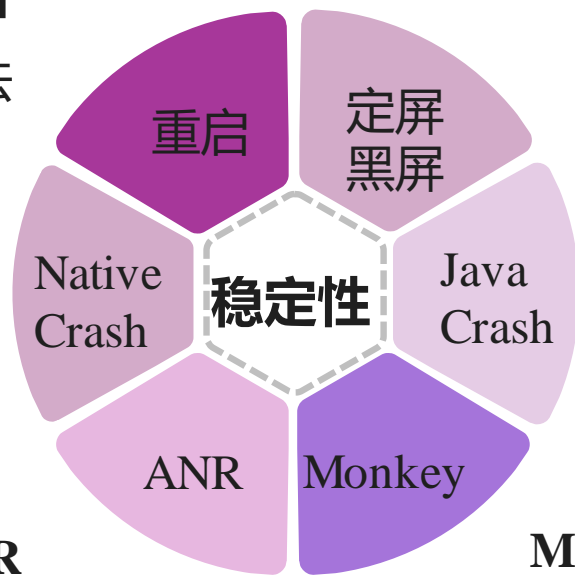
Unisoc Confidential For hiar

重启

如何判定重启，基本分析方法

定屏、黑屏、异常关机

定屏后的关键操作，基本分析方法



Native Crash

如何判定Native Crash，基本分析方法

Java Crash

如何判定Java Crash，基本分析方法

ANR

如何判定ANR，基本分析方法

Monkey

Monkey类问题包含以上5类问题

Java Crash 一般发生在应用Java层面。出现问题后，打印调用栈信息，通过分析调用栈并结合源代码找到解决问题的途径（其中相对复杂的一类为OOM问题）。

➤ hprof 文件

生成路径：/data/misc/hprofs/

➤ 获取方式

- AndroidStudio复现问题，点击工具栏上dump HPROF file按钮抓取hprof文件进行分析。
- adb shell am dumpheap -n <PID_TO_DUMP> /data/local/tmp/heap.hprof

➤ 分析hprof文件

通过Android_sdk/platform-tools/ hprof-conv工具将hprof文件进行转换，转换后通过mat工具进行解析分析。

转换命令如下：

```
$hprof-conv process_pid.hprof new_process_pid.hprof
```

Java Crash案例分析

问题原因：mCreateConnectionProcessor不为null导致重启。

分析方法：dropbox文件system_server_crash@14428907065758.txt&system log中查看调用栈信息并查找对应的代码行分析。

解决方案：向 checkState 传递参数前进行判断。

```
504 504 E AndroidRuntime: *** FATAL EXCEPTION IN SYSTEM PROCESS: main
504 504 E AndroidRuntime: java.lang.RuntimeException: Error receiving broadcast Intent
    { act=android.intent.action.NEW_OUTGOING_CALL flg=0x10000010
    (has extras) } in com.android.server.telecom.NewOutgoingCallIntentBroadcaster$
    NewOutgoingCallBroadcastIntentReceiver@1be0d54b
504 504 E AndroidRuntime:     at android.app.LoadedApk$ReceiverDispatcher$Args.run (LoadedApk.java:897)
504 504 E AndroidRuntime:     at android.os.Handler.handleCallback (Handler.java:739)
504 504 E AndroidRuntime:     at android.os.Handler.dispatchMessage (Handler.java:95)
504 504 E AndroidRuntime:     at android.os.Looper.loop (Looper.java:135)
504 504 E AndroidRuntime:     at com.android.server.SystemServer.run (SystemServer.java:290)
504 504 E AndroidRuntime: Caused by: java.lang.IllegalStateException
504 504 E AndroidRuntime:     at com.android.internal.util.Preconditions.checkState (Preconditions.java:68)
504 504 E AndroidRuntime:     at com.android.server.telecom.Call.startCreateConnection (Call.java:771)
504 504 E AndroidRuntime:     at com.android.server.telecom.CallsManager.placeOutgoingCall (CallsManager.java:727)
504 504 E AndroidRuntime:     at com.android.server.telecom.NewOutgoingCallIntentBroadcaster$
    NewOutgoingCallBroadcastIntentReceiver.onReceive
    (NewOutgoingCallIntentBroadcaster.java:141)
504 504 E AndroidRuntime:     at android.app.LoadedApk$ReceiverDispatcher$Args.run (LoadedApk.java:875)
504 504 E AndroidRuntime:     ... 9 more
```

Native Crash (1/6)

Java应用、Native应用只要在Native层发生异常，就称为Native Crash。

产生原因：

- SIGSEGV

段错误，非法方式访问地址区域（越界访问，写只读的内存块等）。

- SIGABRT

主动终止程序运行，检查某种条件失败后主动退出执行的程序。

- SIGBUS

内存错误，非法方式访问了不可访问的区域（地址未对齐）。

- SIGILL

机器指令错误（未定义指令，PC跑飞等）。

- SIGFPE

算数错误（除0操作）。

ASan

AddressSanitizer (ASan)是一种基于编译器的快速检测工具，用于检测代码中的内存错误。

➤ ASan可以检测以下问题

- 堆栈和堆缓冲区上溢/下溢
- 释放之后的堆使用情况
- 超出范围的堆栈使用情况
- 重复释放/错误释放

➤ ASan的限制

- ASan不能检测未初始化的读取和内存泄漏。
- ASan无法检查Java代码，但可以检测JNI库中的错误。

malloc debug

malloc debug可以用来调试native memory问题, 它可以用来检测如下类型问题:

- memory corruption (内存覆盖)
- memory leaks (内存泄漏)
- use after free (释放后重用)

打开malloc debug需要在对应项目的init.common.rc中添加malloc debug属性。

➤ malloc.options: 设置需要检测内存错误类型, 如越界、覆盖等。

setprop libc.debug.malloc.options "backtrace guard fill"

➤ malloc.program: 设置对应发生问题进程的可执行文件名字, 如system_server进程对应的可执行文件名称为app_process。

setprop libc.debug.malloc.program app_process

Native Crash分析

● 从tombstone中分析信息

借助工具addr2line/objdump/c++filter对tombstone内容进行分析。工具路径：

- 64位：sprdroidr_trunk/bsp/toolchain/prebuilts/gcc/linux-x86/aarch64/aarch64-linux-android-4.9/bin/
- 32位：sprdroidr_trunk/bsp/toolchain/prebuilts/gcc/linux-x86/arm/arm-linux-androideabi-4.9/bin/

➤ 通过addr2line工具解析调用栈信息 **addr2line -f -e ./libc.so 0000582c**

- -f：显示文件名、行号输出信息的同时显示函数名信息。
- -e：指定需要转换地址的可执行文件名。
- libc.so：要解析的库。
- 0000582c：对应的PC相对地址。

➤ 通过objdump工具解析调用栈信息 **objdump -D -d -s -S ./libc.so >./libc_objdump.txt**

- -d：反汇编指令。
- -D：类似-d，反汇编所有section。
- -s：显示指定section的完整内容。默认所有非空section都会被显示，例如.dynsym、.dynstr等。
- -S：反汇编对应源码。
- libc.so：反汇编的库，尖括号后面的内容代表将反汇编信息重定向到文件libc_objdump.txt中。

● 从corefile中分析信息

➤ 准备工作

- 需要准备对应问题版本的symbols文件 (out/target/product/sc***/symbols/) 。
- corefile文件默认生成到/data/corefile 目录下面, corefile 文件的命名方式: core-问题线程的名字-pid。
- 需要格外注意log的导出, 建议使用logs4android2pc工具导出log。

➤ 分析, 进入gdb模式, 在gdb模式下执行如下命令:

- **file ./symbols/system/bin/app_process64**

file命令的作用是关联对应进程的可执行文件。如果是zygote启动的进程, 那么对应的可执行文件就是app_process(区分32位和64位)。如果是native进程, 例如surfaceflinger/mediaserver等进程, 对应的可执行文件是symbols/system/bin/下面相应的文件。

- **set solib-absolute-prefix ./symbols/ (set solib-search-path ./symbols/system/lib/)**

set solib 系列的命令主要作用是设置搜索共享库的路径, 上面两个命令执行其中一条即可。

- **.core ./core-.vorbis.decoder-208**

core命令: 开始解析corefile文件。

Native Crash案例分析—写越界导致的覆盖

问题原因：模块数组写越界导致覆盖。

分析方法：借助tombstone和corefile文件进行分析。

01-01 08:46:37.350 209 209 I DEBUG : pid: 614, tid: 1021, name: CTRL_PLANE_TASK >>> system_server <<<

01-01 08:46:37.350 209 209 I DEBUG : signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 00000000

01-01 08:46:37.530 209 209 I DEBUG : r0 00000000 r1 517a6a9a r2 517a6a9a r3 517a6a9a

01-01 08:46:37.530 209 209 I DEBUG : r4 00000000 r5 00000000 r6 00000000 r7 00000000

01-01 08:46:37.530 209 209 I DEBUG : r8 00000000 r9 565b05a7 sl 565e95b2 fp 565b065f

01-01 08:46:37.530 209 209 I DEBUG : ip 00000003 sp 5790d9c8 lr 55ff9ee7 pc 00000000 cpsr 60000010

01-01 08:46:37.530 209 209 I DEBUG : backtrace:

01-01 08:46:37.530 209 209 I DEBUG : #00 pc 00000000 <unknown>

01-01 08:46:37.530 209 209 I DEBUG : #01 pc 00018ee3 /system/lib/hw/gps.default.so (CG_cpgetPosition+158)

ANR (Application is Not Responding, 应用程序无响应)。在Android中, 当应用程序在规定时间内没有处理完毕相应的事件, 系统就会报出ANR。

产生原因

- InputDispatchingTimedOut: 应用程序主线程在5秒内没有完成用户的input事件。
- Service Timeout: 应用程序没有执行完成service的bind/create/start/destroy/unbind操作, 前台服务20秒超时, 后台服务200秒超时。
- Broadcast Timeout: 应用程序在规定时间内没有执行完成onReceive操作, 前台广播10秒超时, 后台广播60秒超时。
- Content Provider Timeout: 应用程序在20秒内没有执行完成ContentProvider相关操作。

深层原因

- 系统原因: 由于kernel/Framework/Driver等存在问题, 导致系统不稳定最终表现出 ANR。
- 应用原因: 主线程死锁、阻塞或者性能低下, 需避免将耗时操作放在主线程。

- **ANR – Trace**

trace文件用于记录ANR时进程的堆栈信息。

- **ANR – Unisoc Cpu**

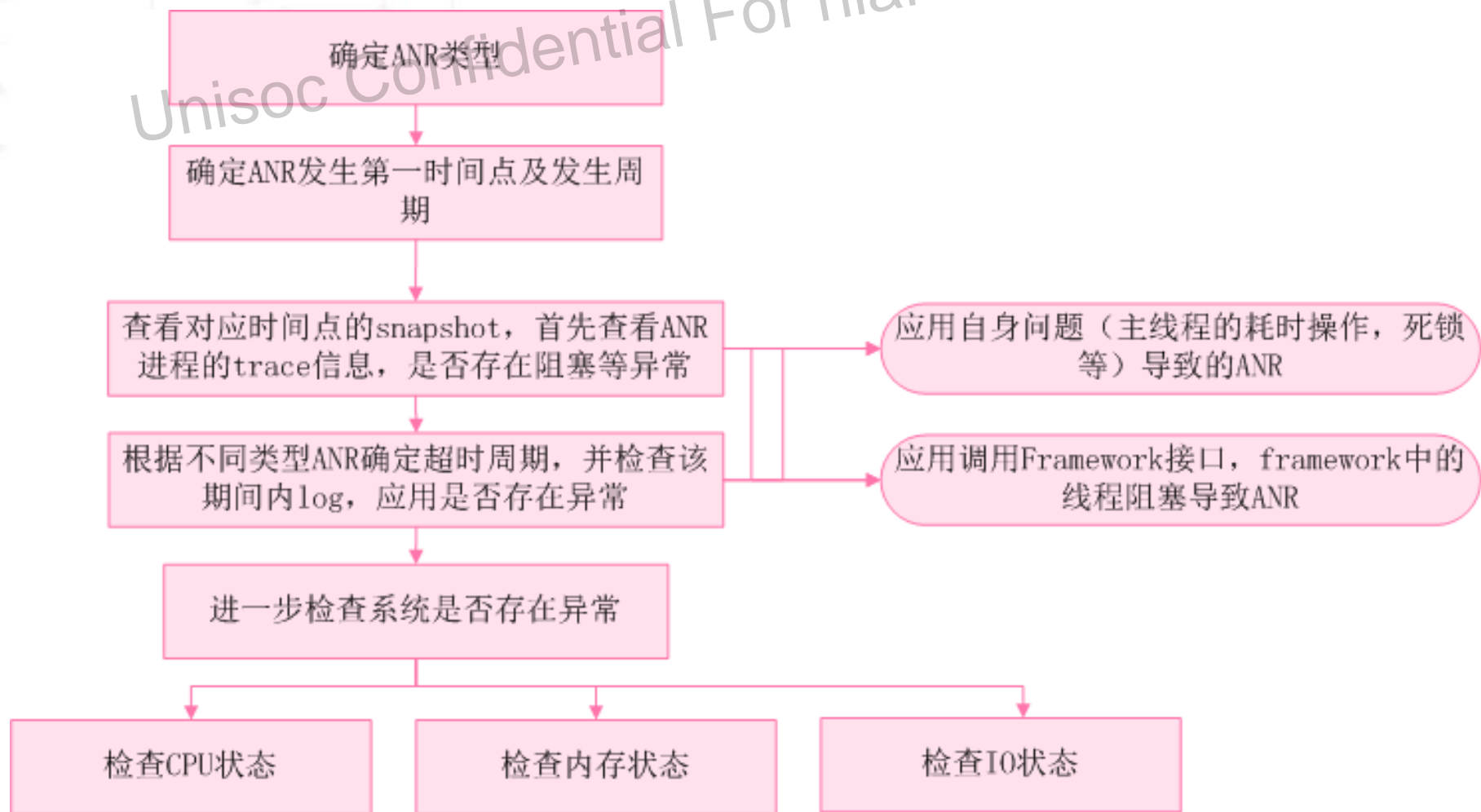
- unisoc_cpu_usage日志是展现平台针对多核实现的统计CPU占有率的日志信息，用于分析系统运行状态。它包含了一个采样周期内每个独立CPU和整个CPU的占有率、每个线程的CPU占有率、以及该采样周期内系统每个独立CPU和整个CPU的上下文切换数/pagefault数/pagemajfault数信息。

- 目前平台自动抓取unisoc_cpu_usage日志的场景主要有：ANR和watchdog。

- **ANR – memory**

sysinfo.log中cat /proc/meminfo命令之后的信息，用于记录系统内存相关的信息。

ANR 问题分析流程



ANR案例分析—应用多实例导致的ANR

问题原因：VolumeDialogImpl有多个实例。

分析方法：dropbox文件system_server_crash@14428907065758.txt&system log中查看调用栈信息并查找对应的代码行进行分析。

典型log：从Dump Win No.的信息可以看到VolumeDialogImpl有多个实例。

S0BD5F4 08-06 15:56:13.283 897 1152 I WindowManager: **Dump Win No.38** win= Window{6c4b60a u0 VolumeDialogImpl}, flags=0x18c0028, canReceive=false, View Visibility=0, isVisibleOrAdding=true

S0BD5F6 08-06 15:56:13.283 897 1152 I WindowManager: **Dump Win No.39** win= Window{1b630bc u0 VolumeDialogImpl}, flags=0x18c0028, canReceive=false, View Visibility=0, isVisibleOrAdding=true

类别	描述	常见重启分类	具体重启类型划分
整机重启	手机发生整机重启	kernel panic	kernel fatal crash
		watchdog timeout	cm4 watchdog
		Alarm/charger	关机闹钟/充电开机
		Recovery	persist进程反复crash
		bootloader	init fatal crash
		调用reboot方法	应用行为
Android重启	kernel正常运行, framework服务发生了异常, 导致 framework reset	zygote进程died	zygote进程本身异常, 其它核心server进程died, 需要重启zygote进程
		system_server进程died	system_server java crash
			system_server native crash
			watchdog
			oom kill system_server

重启案例分析—刷机第一次开机手机重启

1. 在system.log中搜索 “WATCHDOG KILLING” 关键字，可以看到android.fg和main线程被阻塞
07-21 11:10:26.460 1000 744 769 W Watchdog: *** WATCHDOG KILLING SYSTEM PROCESS:
Blocked in handler on foreground thread (android.fg), Blocked in handler on main thread (main)
2. 在对应时间点trace文件里搜索 “system_server” , 查看android.fg和main线程的栈信息，从栈信息看main和android.fg都在等待获取0x0f4ab070对象锁，而该锁被线程128持有着。

```
"main" prio=5 tid=1 Blocked
| group="main" sCount=1 dsCount=0 flags=1 obj=0x729ad868 self=0xaed3de00
| sysTid=744 nice=-2 cgrp=default sched=0/0 handle=0xaf333dc0
| state=S schedstat=( 7274501267 2141946418 9432 ) utm=567 stm=159 core=3 HZ=100
| stack=0xbe0e8000-0xbe0ea000 stackSize=8192KB
| held mutexes=
at com.android.server.pm.PackageManagerService.getInstantAppPackageName(PackageManagerService.java:7314)
- waiting to lock <0x0f4ab070> (a android.util.ArrayMap) held by thread 128
at com.android.server.pm.PackageManagerService.getInstalledPackages(PackageManagerService.java:8680)
at android.app.ApplicationPackageManager.getInstalledPackagesAsUser(ApplicationPackageManager.java:884)
```

3. 查看线程128 当前的栈信息，确认该线程持有0x0f4ab070对象锁，但是该线程在等待获取另外一个对象锁0x0d8d15c8，而该对象锁被线程43持有着。

```
"Binder:744_7" prio=5 tid=128 Blocked
  | group="main" sCount=1 dsCount=0 flags=1 obj=0x14bc33b8 self=0x83592600
  | sysTid=1219 nice=0 cgrp=default sched=1073741824/0 handle=0x7cc7a230
  | state=S schedstat=( 495906402 720188368 979 ) utm=43 stm=6 core=1 HZ=100
  | stack=0x7cb7f000-0x7cb81000 stackSize=1008KB
  | held mutexes=
  at com.android.server.appop.AppOpsService.setUidMode(AppOpsService.java:1273)
  - waiting to lock <0x0d8d15c8> (a com.android.server.appop.AppOpsService) held by thread 43
  at android.app.AppOpsManager.setUidMode(AppOpsManager.java:4574)
  at com.android.server.pm.permission.DefaultPermissionGrantPolicy.grantRuntimePermissions(DefaultPermissionGrantPolicy.java:1251)
  at com.android.server.pm.permission.DefaultPermissionGrantPolicy.grantPermissionsToPackage(DefaultPermissionGrantPolicy.java:390)
  at com.android.server.pm.permission.DefaultPermissionGrantPolicy.grantPermissionsToSystemPackage(DefaultPermissionGrantPolicy.java:367)
  at com.android.server.pm.permission.DefaultPermissionGrantPolicy.grantPermissionsToSystemPackage(DefaultPermissionGrantPolicy.java:357)
  at com.android.server.pm.permission.DefaultPermissionGrantPolicy.grantDefaultPermissionsToEnabledImServices(DefaultPermissionGrantPolicy.java:833)
  at com.android.server.pm.PackageManagerService.grantDefaultPermissionsToEnabledImServices(PackageManagerService.java:25457)
  - locked <0x0f4ab070> (a android.util.ArrayMap)
```


4. 查看线程 43 的栈信息，其在等待获取 0x0f4ab070 对象锁，而该锁被线程 128 持有。到此可以看到线程 128 和线程 43 在相互等对方的锁，于是就发生了互锁触发看门狗重启。该问题需要梳理相关模块的代码逻辑。

```
"Binder:744_3" prio=5 tid=43 Blocked
| group="main" sCount=1 dsCount=0 flags=1 obj=0x148014f0 self=0x83494000
| sysTid=868 nice=-2 cgrp=default sched=1073741824/0 handle=0x8322c230
| state=S schedstat=( 320360402 405695273 764 ) utm=23 stm=9 core=3 HZ=100
| stack=0x83131000-0x83133000 stackSize=1008KB
| held mutexes=
at com.android.server.pm.PackageManagerService.getApplicationInfoInternal(PackageManagerService.java:4803)
- waiting to lock <0x0f4ab070> (a android.util.HashMap) held by thread 128
at com.android.server.pm.PackageManagerService.getApplicationInfo(PackageManagerService.java:4782)
at com.android.server.appop.AppOpsService.getOpsRawLocked(AppOpsService.java:2588)
at com.android.server.appop.AppOpsService.checkPackage(AppOpsService.java:1974)
- locked <0x0d8d15c8> (a com.android.server.appop.AppOpsService)
at com.android.server.appop.AppOpsService.checkOperationUnchecked(AppOpsService.java:1856)
- locked <0x0d8d15c8> (a com.android.server.appop.AppOpsService)
at com.android.server.appop.AppOpsService.checkOperationUnchecked(AppOpsService.java:1835)
```

- 保留现场确认adb是否可连

- adb可连: bugreport等命令。
- adb不可连: trace32或sysdump等。

- nHMonitor (Native Hang Monitor) 机制介绍

在手机实际使用中, 可能会出现定屏之类的异常, 此时希望手机不要卡在这个定屏界面, 而是保存当时的现场信息, 然后重启。

该功能涉及Kernel和Android两个层次。

- 当Android自己能监测到异常时, 自己主动保存Log并重启Android。
- 当Android自己无法监测到定屏异常时, 则由Kernel负责, Kernel监测会保存dump, 并重启系统。

- **nHMonitor适用场景**

- **SystemServer.Watchdog Blocked**

这类问题更多的发生在monkey测试中，看门狗在回调controller.systemNotResponding时阻塞导致定屏。而非monkey测试中，一般是看门狗重启。

- **SystemServer启动时阻塞，且还没有执行到watchdog.start**

这种定屏发生在开机或看门狗重启过程中，一般是sf/vold之类的服务启动中被阻塞了。由于这个时候看门狗还没有启动，所以system_server.watchdog层暂时无法监测到这个问题，需要在nhMonitorService中增加这类问题的监听。但这类问题一般需要重启才有可能恢复，简单的Android重启是无效的，需要依赖Kernel nhMonitor的超时重启。

➤ 系统反复重启，界面黑屏

这类问题一般是zygote或者system_server在反复crash(Java or Native)。这类问题也无法通过system_server.watchdog监听，但可以通过nhMonitorService监控，同样一般需要重启。

➤ Init没有起来

这类问题一般是Kernel出现问题，或者需要通过sysdump抓取init corefile。Android层无法监控，完全需要依赖Kernel nhMonitor。

➤ System_server进程suspend失败

System_server已经停止工作，检测工作停止，只能依靠Kernel nhMonitor，但只能通过minidump保存数据。在这个机制下，没有合适机会去保存system_server的debuggerd信息，出现这类问题的概率极低。

定屏/黑屏类问题 (4/11)

现象上

- 触发sysdump
- 定屏
- 异常重启
- 开机卡开机logo
- 开机卡开机动画

```
Sysdumping now, keep power on.
-----
Reset mode: kernel_crash
exception_file_info:
not-bugon
exception_panic_reason:
Crash Key
exception_stack_info:
[<25b3ebfe>] get_exception_stack_info+0x15c/0x264
[<2b259495>] prepare_exception_info+0x158/0x174
[<0bd3fea4>] sysdump_panic_event+0x588/0x740
[<4c3e5675>] notifier_call_chain+0x94/0xc8
[<55bcec1b>] atomic_notifier_call_chain+0x40/0x54
[<7d5c8f43>] panic+0x154/0x2b8
[<b9d43351>] crash_note_save_cpu+0x0/0x1c4
[<9e56c49b>] 0xffffffff
-----
init_mmc_fat failed, Please check SD Card!!!.
sysdump to sdcard fail , we try to do dump to pc now ??
Dump to PC start..
check usb cable's status or check key volumn up pressed to abort
```


● sysdump简介

sysdump 是对 Android 系统进行调试的工具之一。当系统发生故障或者主动触发故障时就会触发 sysdump 的机制，对当前现场进行保护，进而将信息保存到T卡中。

● 触发原因

1. 用户主动触发

- magic key: 同时按住音量上下键之后双击power键。
- echo 'c' > /proc/sysrq-trigger
- 7s reset: 可在uboot中配成单键（power键）或双键（power键+音量上键）。
- 前两种是软件行为，第三种7s reset是硬件行为。
- 7s reset和正常panic()流程相比，软件flush cache等动作，导致save的RAM内容不完整，最近的关键信息看不到，用7s reset的sysdump成功分析到问题点的案例很少，不推荐使用。

2. 程序运行异常触发sysdump

- Kernel crash。
- 异常信息需要使用crash工具解析sysdump之后从Log中查看。
- 发生Kernel crash之后Kernel会运行到panic()函数，执行panic()会重启手机，在uboot中进行sysdump。

Kernel crash分类

- DDR和CACHE数据不一致
- 未定义指令
- 外部异常
- watchdog
- lockup
- CP crash
- 踩内存

硬件watchdog

● 配置差异

- User版本：sysdump默认关闭，watchdog默认开启，adb需要在手机上授权才能连接。
- Userdebug版本：sysdump默认开启，watchdog默认关闭，adb无需授权可以直接连接。

● 发生apwatchdog表象差别

- User版本
 - 打开sysdump：生成sysdump文件。
 - 关闭sysdump（默认）：重启。
- Userdebug版本
 - 定屏。

Lockup

- **Soft lockup**

每个core上有一个[watchdog/x]进程，此进程长时间得不到调度，就会触发soft lockup。

- **Hard lockup**

hrtimer_interrupts计数器表征系统是否能响应中断，如果长时间计数器值没变化，就会触发hard lockup。

发生lockup后的表现：

- 没打开lockup detector功能，表现为系统卡顿、定屏。
- 打开lockup detector功能，表现为触发Kernel crash。

定屏

定屏问题需要确认是Kernel卡死还是上层卡死。

- **判断问题点**

adb是否可连、或在设备管理器中查看是否有新端口识别。

- **上层卡死**

bugreport, 并连接电源保留现场, 等待上层同事分析。

- **Kernel卡死**

- 查看magic能否触发sysdump, 提供sysdump。
- magic不能触发sysdump, 打开lockup机制。
- 以上手段均失效, 长按 “power键+音量上键” 触发sysdump, 初步分析Log。
- 长按 “power键+音量上键” 触发sysdump无法定位, 则需要连接trace32查看现场。

异常重启

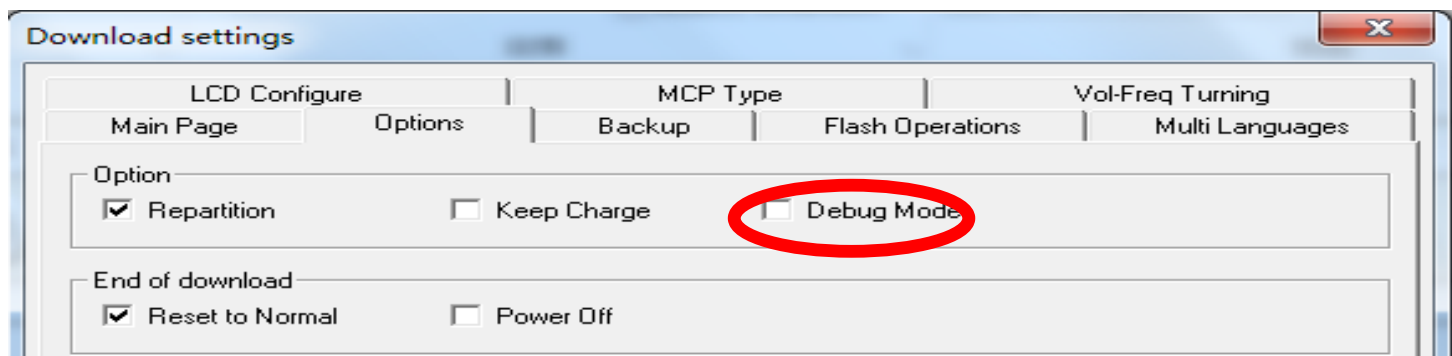
- 上层system_server、zygote、surfaceflinger等关键进程发生crash造成上层重启
需要提供YLog查看重启原因，但若是Kernel异常只能看到重启类型，无法查看到具体的问题点。
- 在没有打开sysdump enable开关时，Kernel卡死触发看门狗重启或Kernel crash之后直接进入了主系统。

打开sysdump或者用debug版本测试，抓取sysdump分析。

开机定屏

● 开机卡开机logo

- 需要抓取开机串口Log确认卡死的地方。
- Researchdownload->settings->options->勾选debug mode, 只更新FDL1和FDL2即可。(此操作等同于设置loglevel=7)。



● 开机卡开机动画

- 确认adb是否可连。
- 开机动画能够播放说明bootanim进程已经启动, 但zygote没能正常启动。
- 抓取串口Log, 另外需要将logcat重定向至串口Log, 查看上层是否有异常。

Unisoc Confidential For hiar

03

Crash工具命令简介



● 解析sysdump方法

➤ 需要与下载的pac包编译, 同时生成vmlinux文件

- `cat sysdump.core.* > vmcore`

➤ 需确定vmlinux与生成sysdump的软件版本是否一致

- `strings vmlinux | grep "gcc version"`
- `strings vmcore | grep "gcc version"`

```
Linux version 4.4.83 (android@c0496) (gcc version 4.8 (GCC) ) #1 SMP PREEMPT Thu May 9  
21:51:17 CST 2019
```

上述两条命令显示的编译时间、主机、用户应当完全一致

- 64位: `crash_arm64 -m phys_offset=0x80000000 vmlinux vmcore`
- 32位: `crash_arm -m phys_base=0x80000000 vmlinux vmcore`

Crash工具命令简介 (2/7)

- **help**

列出当前crash工具支持的所有命令, 以及关于某条命令的详细描述。

```
crash_arm64> help
```

*	extend	mach	runq	tree
alias	files	mod	search	union
ascii	foreach	mount	set	vm
bpf	fuser	net	sha1	vtop
bt	gdb	p	sig	waitq
bttop	help	ps	struct	whatis
compare	ipcs	pte	swap	wr
dev	irq	ptob	sym	q
dis	kmem	ptov	sys	
eval	list	rd	task	
exit	log	repeat	timer	

For help on any command above, enter "help <command>".

Crash工具命令简介 (3/7)

- log

输出Kernel log buffer里的内容。

```
crash_arm64> log | grep -A 10 "Kernel panic"
[ 42.695186] c0 Kernel panic - not syncing: test gnss
[ 42.700061] c0 CPU: 0 PID: 385 Comm: kworker/0:2 Tainted: G S      O  4.14.78+ #72
[ 42.708016] c0 Hardware name: Unisoc UMS510 1H10 board (DT)
[ 42.713570] c0 Workqueue: events pre_gnss_download_firmware
[ 42.719093] c0 Call trace:
[ 42.721783] c0 [<ffffff800808a744>] dump_backtrace+0x0/0x248
[ 42.727400] c0 [<ffffff800808a9ac>] show_stack+0x20/0x28
[ 42.732684] c0 [<ffffff80089b8908>] dump_stack+0xa4/0xdc
[ 42.737964] c0 [<ffffff80080aabdb>] panic+0x134/0x294
[ 42.742980] c0 [<ffffff80086cd870>] sprd_send_cmd+0x514/0x730
[ 42.748688] c0 [<ffffff80086cdb54>] sprd_sdhc_request+0xc8/0x13c
```

Crash工具命令简介 (4/7)

- ps

ps命令可以列出当前系统task信息。

```
crash_arm> ps
```

	PID	PPID	CPU	TASK	ST	%MEM	VSZ	RSS	COMM
>	0	0	0	c09f2c38	RU	0.0	0	0	[swapper/0]
>	0	0	1	df4cc600	RU	0.0	0	0	[swapper/1]
>	0	0	2	df4ccd00	RU	0.0	0	0	[swapper/2]
	0	0	3	df4cd400	RU	0.0	0	0	[swapper/3]
	1	0	2	df4c8000	IN	0.1	9624	2016	init
	2	0	2	df4c8700	IN	0.0	0	0	[kthreadd]
	3	2	0	df4c8e00	IN	0.0	0	0	[ksoftirqd/0]
	4	2	0	df4c9500	IN	0.0	0	0	[kworker/0:0]
	5	2	0	df4c9c00	IN	0.0	0	0	[kworker/0:0H]

- **bt**

查看CPU上当前线程堆栈。

```
crash_arm> bt | head
PID: 907  TASK: c40e8e00 CPU: 3  COMMAND: "Binder:893_1"
#0 [<c0341a0c>] (sysdump_enter) from [<c00f3ab8>]
#1 [<c00f3a50>] (panic) from [<c00135b8>]
#2 [<c0013328>] (die) from [<c001ed5c>]
#3 [<c001ed04>] (__do_kernel_fault) from [<c001ba24>]
#4 [<c001b9ac>] (do_bad_area) from [<c001baec>]
#5 [<c001ba4c>] (do_translation_fault) from [<c0009278>]
#6 [<c0009240>] (do_DataAbort) from [<c0013cd8>]
  pc : [<c011c454>]  lr : [<c011c47c>]  psr: a0010013
  sp : c2661d98 ip : c2661d98 fp : c2661dcc
```

- **runq**

查看CPU上运行队列中线程列表。

```
crash_arm> runq | head
```

```
CPU 0 RUNQUEUE: dff4a2c0
```

```
  CURRENT: PID: 0   TASK: c09f2c38  COMMAND: "swapper/0"
```

```
  RT PRIO_ARRAY: dff4a3c0
```

```
    [no tasks queued]
```

```
  CFS RB_ROOT: dff4a330
```

```
    [no tasks queued]
```

```
CPU 1 RUNQUEUE: dff572c0
```

```
  CURRENT: PID: 0   TASK: df4cc600  COMMAND: "swapper/1"
```

```
  RT PRIO_ARRAY: dff573c0
```

Crash工具命令简介 (7/7)

- rd

读取某地址开始的多个数据。

```
crash_arm> rd c42ab000 16
```

```
c42ab000: a6074000 eb010022 eb000041 c42a7f78  .@.."...A...x.*.
```

```
c42ab010: c42ab069 00000000 00000000 00000000  i.*.....
```

```
c42ab020: daef3500 00000e9f 00000070 00000000  .5.....p.....
```

```
c42ab030: 00000000 00000000 00000000 c42ab03c  .....<.*.
```

Unisoc Confidential For hiar

04

典型问题分析



- 先用log命令查看logbuffer中的kernellog, 找到panic的log信息

[1446.809997] c3 Unable to handle kernel NULL pointer dereference at virtual address 00000000[13.691026]

查看问题处的反汇编

```
crash_arm64> dis -xl fffffff8008264d44
```

```
/home/disk4/project/9190WO_T310/kernel4.14/fs/buffer.c: 3275
```

```
0xffffffff8008264d44 <drop_buffers+0x3c>: ldr    x2, [x0]
```

● 查看调用栈

```
crash_arm64> bt
PID: 37 TASK: fffffc0ba80c080 CPU: 3 COMMAND: "kcompactd0"
#6 [ffff800979b750] do_mem_abort at fffff8008080aa8
#7 [ffff800979b950] ell_ia at fffff8008082898
    PC: fffff8008265ff8 [drop_buffers+60]
    LR: fffff80082671b8 [try_to_free_buffers+160]
    SP: fffff800979ba60 PSTATE: 00c00045
    X29: fffff800979ba60 X28: fffffb00bf10c0 X27: fffff80081e57b0
    X26: 0000000000000000 X25: 0000000000000000 X24: 0000000000000001
    X23: fffff800979bad0 X22: fffffb013dc880 X21: fffffc0284779d0
    X20: 0000000000000001 X19: fffffb00bf10c0 X18: 0000000000000000
    X17: b3598c50622c000d X16: c691e4006907578a X15: 941f2498f95cdcc5
    X14: 34c7c801c0801848 X13: 0de823babf820150 X12: a30f50df00ffa84a
    X11: 11e0c17890cfde28 X10: 627e32101ad13e36 X9: d32478c6a0a0dea4
    X8: e9001479f9b02273 X7: 69923a8e09b1b302 X6: 00000040b702b000
    X5: 0000000000000031 X4: fffff80082671ac X3: fffff800926e000
    X2: 0000000000000000 X1: 0000000000000000 X0: 0000000000000000
#8 [ffff800979ba60] drop_buffers at fffff8008265ff4
#9 [ffff800979baa0] try_to_free_buffers at fffff80082671b4
#10 [ffff800979bae0] try_to_release_page at fffff80081b9b04
#11 [ffff800979bb10] move_to_new_page at fffff8008218994
#12 [ffff800979bb90] migrate_pages at fffff8008219170
#13 [ffff800979bc50] compact_zone at fffff80081e8c30
#14 [ffff800979bcf0] kcompactd_do_work at fffff80081e9344
#15 [ffff800979bdd0] kcompactd at fffff80081e95e8
#16 [ffff800979be60] kthread at fffff80080cc5dc]
```


● 查看代码意义

try_to_release_page()->try_to_free_buffers()->drop_buffers()

释放与page相关的buffer_head。

```
crash_arm64> page -x ffffffffbf00bf10c0 | grep private
```

```
private = 0xffffffffc0284779d0,
```

```
crash_arm64> list -o 8 0xffffffffc0284779d0 -s buffer_head.b_state,b_page
```

```
fffffffc0284779d0
```

```
b_state = 33
```

```
b_page = 0xfffffffffbf00bf10c0
```

```
.....
```

```
fffffffc028477d40
```

```
b_state = 33
```

```
b_page = 0xfffffffffbf00bf10c0
```

```
fffffffc028476008
```

```
b_state = 0
```

```
b_page = 0x0
```

可能原因：地址跳变或踩内存

- 查看该地址附件数据

```
crash_arm64> rd fffffffc028476008 20
```

```
ffffffc028476008: 0000000000000000 0000000000000000 .....
```

```
ffffffc028476018: 0000000000000000 0000000000000000 .....
```

```
ffffffc028476028: 0000000000000000 0000000000000000 .....
```

```
ffffffc028476038: 0000000000000000 0000000000000000 .....
```

```
ffffffc028476048: 0000000000000000 fffffffc028476050 .....P`G(...
```

```
ffffffc028476058: fffffffc028476050 0000000000000000 P`G(.....
```

可能原因：地址跳变或踩内存

● 通过KASAN检测是否有越界问题

[3054.011339] c3 BUG: KASAN: slab-out-of-bounds in prepare_addba+0x4c/0x1c8 [sprdwl_ng]
[3054.011343] c2 WCN SLP_MGR: PUB_INT_STS0-0x4
[3054.011355] c3 Read of size 8 at addr fffffffc05e225e10 by task wifi@1.0-servic/344
[3054.011357] c3
[3054.011368] c3 CPU: 3 PID: 344 Comm: wifi@1.0-servic Tainted: G S O 4.14.98+ #5
[3054.011371] c3 Hardware name: Unisoc UMS312 2H10 board (DT)
[3054.011375] c3 Call trace:
[3054.011388] c3 [<ffffff900808d008>] dump_backtrace+0x0/0x28c
[3054.011401] c3 [<ffffff900808d2b4>] show_stack+0x20/0x28
[3054.011410] c3 [<ffffff9008dbef94>] dump_stack+0xac/0xe4
[3054.011418] c3 [<ffffff90082a9094>] print_address_description+0x78/0x2e0
[3054.011421] c2 WCN SLP_MGR: allow sleep
[3054.011432] c3 [<ffffff90082a9618>] kasan_report+0x268/0x2b8
[3054.011439] c3 [<ffffff90082a7ed8>] __asan_load8+0x90/0x98
[3054.011581] c3 [<ffffff9001252814>] prepare_addba+0x4c/0x1c8 [sprdwl_ng]
[3054.011727] c3 [<ffffff9001252e3c>] sprdwl_tx_msg_func+0x4ac/0x5b0 [sprdwl_ng]

- 内存覆盖问题分析方法

- slub

- KASAN

- MPU

slub

- 在内存分配/释放的接口(kmalloc()/kfree()/kmem_cache_alloc()/kmem_cache_free()等接口), slub会对debug数据做检测。
- 若没有调用到Kernel内存alloc/free接口, slub无法进行检测。

KASAN

- 运行时监控str/ldr指令访问内存的执行。
- 只能在arm64/x86平台上使用，arm平台并不支持KASAN。
- 需要额外消耗内存，对于小内存的设备，可能对性能和功能上造成影响。
- 监控str/ldr指令本身会影响性能。

MPU

- UNISOC自行研发设计的硬件模块。
- 监控可能被覆盖的内存区域，定位踩内存的子系统。
- 只能通过配置物理地址进行内存保护（对于AP侧软件行为的踩内存分析不是很有效）。

Unisoc Confidential For hiar

谢谢



本文件所含数据和信息都属于紫光展锐（上海）科技有限公司（以下简称紫光展锐）所有的机密信息，紫光展锐保留所有相关权利。本文件仅为信息参考之目的提供，不包含任何明示或默示的知识产权许可，也不表示有任何明示或默示的保证，包括但不限于满足任何特殊目的、不侵权或性能。当您接受这份文件时，即表示您同意本文件中内容和信息属于紫光展锐机密信息，且同意在未获得紫光展锐书面同意前，不使用或复制本文件的整体或部分，也不向任何其他方披露本文件内容。紫光展锐有权在未经事先通知的情况下，在任何时候对本文件做任何修改。紫光展锐对本文件所含数据和信息不做任何保证，在任何情况下，紫光展锐均不负责任与本文件相关的直接或间接的、任何伤害或损失。请参照交付物中说明文档对紫光展锐交付物进行使用，任何人对紫光展锐交付物的修改、定制化或违反说明文档的指引对紫光展锐交付物进行使用造成的任何损失由其自行承担。紫光展锐交付物中的性能指标、测试结果和参数等，均为在紫光展锐内部研发和测试系统中获得的，仅供参考，若任何人需要对交付物进行商用或量产，需要结合自身的软硬件测试环境进行全面的测试和调试。