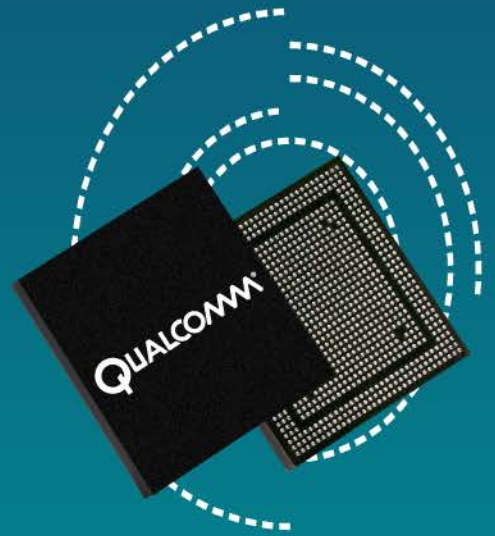
Qualcomm
TECHNOLOGIES, INC



Android Stability Issue Analysis Guide

80-NJ221-2 A

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision History

Revision	Date	Description
A	Aug 2013	Initial release

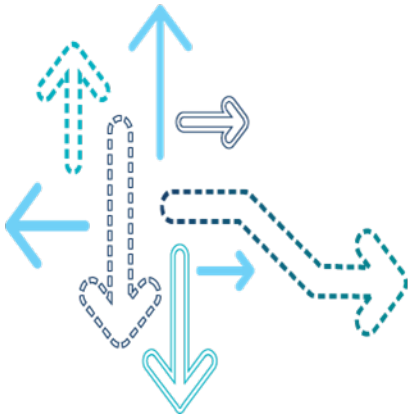
Note: There is no Rev. I, O, Q, S, X, or Z per Mil. standards.

Contents

Objectives

- At the end of this presentation, you will understand:
  - What the common Android stability issues are.
  - How to debug Android ANR issue.
  - How to debug Android Watchdog timeout issue.
  - How to debug Android system_server crash issue.
  - How to use JTAG to debug on live target.

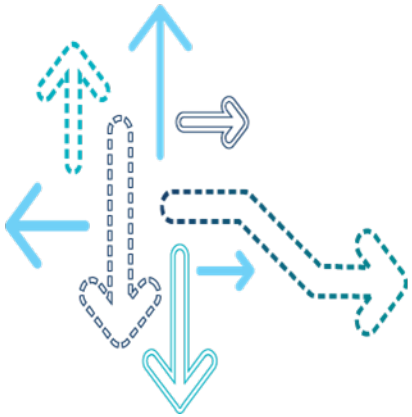
Overview



Overview

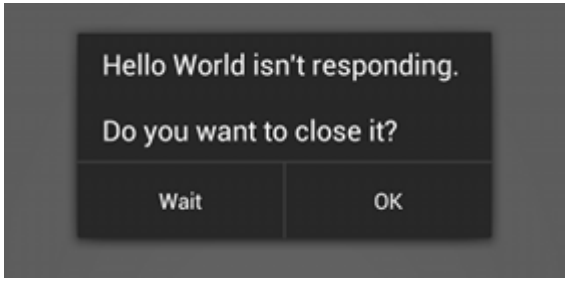
- Some of the Android stability issues are different from Kernel stability issues, so we need to use the correct ways for different issues.
- Here are the common Android Stability issues:
  - ANR(Application Not Responding)
  - Android Watchdog timeout
  - Android system_server crash

ANR



ANR

- It's possible to write code that wins every performance test, but it maybe still feels sluggish, hang or freeze for significant periods, or take too long to process input. The worst thing that can happen to your app's responsiveness is an "Application Not Responding" (ANR) dialog.
- In Android, the system guards against applications that are insufficiently responsive for a time period by displaying a dialog that says your app has stopped responding. At this point, your app has been unresponsive for a considerable time period so the system offers the user an option to quit the app.
- Note that system can't show this dialog sometimes due to internal problems, e.g. if ANR occurrs in the Activity or Window Manager.



ANR

- Generally, the system displays an ANR if an application cannot respond to user input. For example, if an application blocks on some I/O operation (frequently a network access) on the UI thread so the system can't process incoming user input events.
- In any situation, your app performs a potentially lengthy operation, you should not perform the work on the UI thread, but create a worker thread and do most of the work there instead . This keeps the UI thread (which drives the user interface event loop) running and prevents the system from concluding that your code has frozen.
- In Android, application responsiveness is monitored by the Activity Manager and Window Manager system services. Android will display the ANR dialog for a particular application when it detects one of the following conditions:
  - No response to an input event (such as key press or screen touch events) within 5 seconds.
  - A BroadcastReceiver hasn't finished executing within 10 seconds.

ANR

- Android applications normally run entirely on a single thread by default (the "UI thread" or "main thread"). It means anything your application is doing in the UI thread that takes a long time to complete. And it can trigger the ANR dialog because your application doesn't give itself a chance to handle the input event or intent broadcasts.
- Therefore, any method that runs in the UI thread should do as little work as possible on that thread.
- Potentially long running operations such as network or database operations (disk I/O operations), or computationally expensive calculations such as resizing bitmaps should be done in a worker thread.
- The specific constraint on BroadcastReceiver execution time emphasizes what broadcast receivers are meant to do: small, discrete amounts of work in the background such as saving a setting or registering a Notification. So as with other methods called in the UI thread, applications should avoid potentially long-running operations or calculations in a broadcast receiver. But instead of doing intensive tasks via worker threads, your application should start an IntentService if a potentially long running action needs to be taken in response to an intent broadcast.

ANR Issue Analysis

- For ANR issue, we should search the key word “ANR in” from the logcat log. From this log, we can get the information about which application got the ANR and the ANR reason (key dispatching time out or broadcast time out).
- The DVM ANR logging system writes to the /data/anr/traces.txt.bugreport log, which is considered as “Just Now” log, and is rewritten every time the ‘dumpstate’ command is called or the system decides to update this log. If ANR conditions occurred and the system considered the thread as “not responding,” it writes the ANR report to /data/anr/traces.txt, which is called Last ANR log.
- In ANR log, check the 'main' thread of the process since this is a thread where UI works.
- In ANR log “/data/anr/traces.txt”, check the state of your main thread. If it is in MONITOR state, it can be in 'dead lock' state. TIMED_WAIT state can also point to the problems with locking of your thread by 'sleep' or 'wait' functions.

ANR Issue Analysis

- Thread statuses:
  - ZOMBIE - terminated thread
  - RUNNABLE - runnable or running now
  - TIMED_WAIT - timed waiting in Object.wait()
  - MONITOR - blocked on a monitor
  - WAIT - waiting in Object.wait()
  - INITIALIZING - allocated, not yet running
  - STARTING - started, not yet on thread list
  - NATIVE - off in a JNI native method
  - VMWAIT - waiting on a VM resource
  - SUSPENDED - suspended, usually by GC or debugger
  - UNKNOWN - thread is in the undefined state

ANR Issue Analysis

- If the system itself (Activity or Window Managers) is in ANR condition and can't raise ANR, so you can't differentiate which thread in which process is responsible for the ANR, analyze the “Just Now” log.
- If necessary, we need to check the tombstones log for more information. We can get the tombstones log from /data/tombstones.
- At the same time, analyze nearby messages in the logcat log. Sometimes the system output information is why it decides that the thread is in the ANR state.

ANR Log Format

- ANR output displays threads information in the following order:
  - 1.DVM mutexes (only for ‘main’ thread)
  - 2.Thread’s information
  - 3.Thread stack
  - 4.DVM mutexes format

    "(mutexes: tll=%x tsl=%x tscl=%x ghl=%x hwl=%x hwll=%x)", where

    tll – thread list lock,

    tsl – thread suspend lock,

    tscl – thread suspend count lock,

    dhl – GC heap lock,

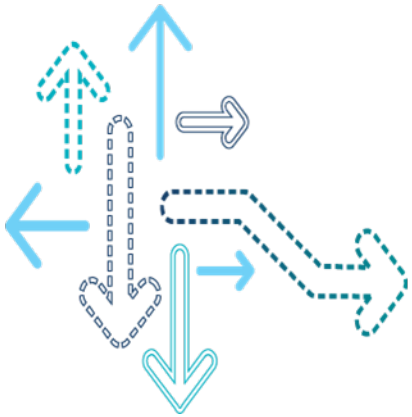
    hwl – heap worker lock,

    hwll – heap worker list lock
- Thread’s information format – First line
  - “%name %priority %tid %status”, where
    - name – Thread name
    - priority – Thread priority
    - tid – Thread ID
    - status – Thread status

ANR Log Format

- Thread's information format – Second line
  - "group=%s sCount=%d dsCount=%d obj=%p self=%p", where
    - group – Group name,
    - sCount – Suspend count,
    - dsCount – Debug suspend count,
    - obj – Linux thread that we are associated with
    - self – Self-reference
- Thread's information format – Third line
  - "sysTid=%d nice=%d sched=%d/%d cgrp=%s handle=%d", where sysTid – Linux thread ID
    - nice – Linux "nice" priority (lower numbers indicate higher priority)
    - sched – Scheduling priority
    - cgrp – Scheduling group buffer
    - handle – Thread handle

Case Study



Case Study 1:

- This is a UI main thread stuck issue, the main thread is stuck while do the IPC calling.
- From the logcat system log:

```
06-13 15:45:38.923 E/ActivityManager(  654): ANR in com.android.settings
(android/com.android.internal.app.RingtonePickerActivity)
06-13 15:45:38.923 E/ActivityManager(  654): Reason: keyDispatchingTimedOut
06-13 15:45:38.923 E/ActivityManager(  654): Load: 2.78 / 3.87 / 5.0
06-13 15:45:38.923 E/ActivityManager(  654): CPU usage from 16121ms to 3130ms ago:
06-13 15:45:38.923 E/ActivityManager(  654):   17% 654/system_server: 12% user + 4.3% kernel / faults: 2788 minor
06-13 15:45:38.923 E/ActivityManager(  654):   4.5% 177/surfaceflinger: 1% user + 3.4% kernel / faults: 9075 minor
06-13 15:45:38.923 E/ActivityManager(  654):   0% 218/adbd: 0% user + 0% kernel / faults: 22 minor
06-13 15:45:38.923 E/ActivityManager(  654):   3.6% 13086/kworker/0:1: 0% user + 3.6% kernel
```

- It’s key dispatching ANR which causes the setting application can’t response to the key pressing in time.
- The ANR will cause the ‘ActivityManager’ kills the application, so you should find ‘SIGSEGV’ or ‘SIGABRT’ logs from logcat main log.
- There is also CPU load information.

Case Study 1:

- From the log '/data/anr/traces_com.android.settings.txt'

```
"main" prio=5 tid=1 NATIVE
  | group="main" sCount=1 dsCount=0 obj=0x412876a0 self=0x41277150
  | sysTid=19557 nice=0 sched=0/0 cgrp=apps handle=1074794288
  | schedstat=( 0 0 0 ) utm=50 stm=34 core=0
  #00  pc 0000cb90  /system/lib/libc.so (__ioctl+8)
  #01  pc 00027ffd  /system/lib/libc.so (ioctl+16)
  #02  pc 00016b6d  /system/lib/libbinder.so (android::IPCThreadState::talkWithDriver(bool)+124)
  #03  pc 00017059  /system/lib/libbinder.so (android::IPCThreadState::waitForResponse(android::Parcel*, int*)+44)
  #04  pc 00017273  /system/lib/libbinder.so (android::IPCThreadState::transact(int, unsigned int, android::Parcel const&, android::Parcel*, unsigned int)+114)
  #05  pc 00014a6b  /system/lib/libbinder.so (android::BpBinder::transact(unsigned int, android::Parcel const&, android::Parcel*, unsigned int)+34)
  #06  pc 0003f2c5  /system/lib/libmedia.so
  #07  pc 0004400f  /system/lib/libmedia.so (android::AudioSystem::get_audio_flinger()+422)
  #08  pc 000440dd  /system/lib/libmedia.so (android::AudioSystem::newAudioSessionId()+2)
  #09  pc 00045999  /system/lib/libmedia.so (android::MediaPlayer::MediaPlayer()+204)
  #10  pc 000119db  /system/lib/libmedia_jni.so
  #11  pc 0001f330  /system/lib/libdvm.so (dvmPlatformInvoke+112)
  #12  pc 0004e079  /system/lib/libdvm.so (dvmCallJNIMethod(unsigned int const*, JValue*, Method const*, Thread*)+360)
  #13  pc 000287e0  /system/lib/libdvm.so
  #14  pc 0002cfa8  /system/lib/libdvm.so (dvmInterpret(Thread*, Method const*, JValue*)+180)
  #15  pc 0005f93f  /system/lib/libdvm.so (dvmInvokeMethod(Object*, Method const*, ArrayObject*, ArrayObject*, ClassObject*, bool)+374)
  #16  pc 000668e5  /system/lib/libdvm.so
  #17  pc 000287e0  /system/lib/libdvm.so
  #18  pc 0002cfa8  /system/lib/libdvm.so (dvmInterpret(Thread*, Method const*, JValue*)+180)
  #19  pc 0005f695  /system/lib/libdvm.so (dvmCallMethodV(Thread*, Method const*, Object*, bool, JValue*, std::__va_list)+272)
  #20  pc 0004a6a7  /system/lib/libdvm.so
  #21  pc 00049c6d  /system/lib/libandroid_runtime.so
  #22  pc 0004a711  /system/lib/libandroid_runtime.so (android::AndroidRuntime::start(char const*, char const*)+368)
  #23  pc 00000dcf  /system/bin/app_process
  #24  pc 00017193  /system/lib/libc.so (__libc_init+38)
  #25  pc 00000b34  /system/bin/app_process
  at android.media.MediaPlayer.native_setup(Native Method)
  at android.media.MediaPlayer.<init>(MediaPlayer.java:635)
  at com.android.internal.app.RingtonePickerActivity.onItemClick(RingtonePickerActivity.java:623)
  at android.widget.AdapterView.performItemClick(AdapterView.java:298)
```

Case Study 1:

- From the tombstond log, we can find similar stack.
- We can see the application's main thread is stuck while the function "native_setup(new WeakReference<MediaPlayer>(this));" is called in the file 'MediaPlayer.java'.
- Check the function "get_audio_flinger()", it will call 'binder = sm->getService(String16("media.audio_flinger"));' to make IPC calling to get the 'audio_flinger' service.
- The most possible reason is that there is a deadlock in the related service or the service cost too much time before it returns.
- For this issue, there is a deadlock in the AudioFlinger, it is resolved by making changes in the file "AudioFlinger.cpp".

Case Study 2:

- Another ANR issue which is caused because ‘cat proc/loadavg’ is called in main thread, but this calling is stuck because of a deadlock during fork a new process, the status out pipe will keep open but empty, read on this pipe will be blocked.

04-20 15:36:39.029: E/ActivityManager(332): ANR in com.into.stability (com.into.stability/.TestGPU)
04-20 15:36:39.029: E/ActivityManager(332): Reason: keyDispatchingTimedOut
04-20 15:36:39.029: E/ActivityManager(332): Load: 13.89 / 7.72 / 3.22
04-20 15:36:39.029: E/ActivityManager(332): CPU usage from 32569ms to 127ms ago:
04-20 15:36:39.029: E/ActivityManager(332): 162% 1988/com.into.stability: 152% user + 9.9% kernel / faults: 3517 minor
04-20 15:36:39.029: E/ActivityManager(332): 15% 109/surfaceflinger: 6.6% user + 8.6% kernel
04-20 15:36:39.029: E/ActivityManager(332): 1.9% 332/system_server: 0.9% user + 0.9% kernel / faults: 49 minor
.....
04-20 15:36:39.029: E/ActivityManager(332): 97% TOTAL: 82% user + 12% kernel + 3.2% iowait
04-20 15:36:39.029: E/ActivityManager(332): CPU usage from 896ms to 1428ms later with 99% awake:
04-20 15:36:39.029: E/ActivityManager(332): 184% 1988/com.into.stability: 184% user + 0% kernel / faults: 133 minor
04-20 15:36:39.029: E/ActivityManager(332): 90% 2093/Thread-119: 90% user + 0% kernel
04-20 15:36:39.029: E/ActivityManager(332): 84% 2209/Thread-120: 84% user + 0% kernel
04-20 15:36:39.029: E/ActivityManager(332): 5.7% 1996/FinalizerDaemon: 5.7% user + 0% kernel
.....
04-20 15:36:39.029: E/ActivityManager(332): 95% TOTAL: 95% user + 0% kernel
04-20 15:36:39.039: A/libc(1988): Fatal signal 6 (SIGABRT) at 0x0000014c (code=0)
04-20 15:36:40.139: I/DEBUG(107): *** *** *** *** *** *** *** *** *** *** *** *** *** *** *** ***
04-20 15:36:40.139: I/DEBUG(107): Build fingerprint:
'qcom/msm7627a/msm7627a:4.0.4/IMM76I/eng.junj.20121122.151519:eng/test-keys'
04-20 15:36:40.139: I/DEBUG(107): pid: 1988, tid: 1988 >>> com.into.stability <<<

Case Study 2:

- We can see from the logcat log, the application “com.into.stability” costs so much CPU.
- Check the trace log form the path '/data/anr/‘, We can see the fork is stuck there.
- It’s a know issue, you can get more information from this link: https://code.google.com/p/android/issues/detail?id=19916

```
DALVIK THREADS:
(mutexes: tll=0 tsl=0 tscl=0 ghl=0)
"main" prio=5 tid=1 NATIVE
  | group="main" sCount=1 dsCount=0 obj=0x40aeb5f8 self=0xf62e28
  | sysTid=1523 nice=0 sched=0/0 cgrp=ux handle=1075107176
  #00  pc 0000c674  /system/lib/libc.so (read+8)
  #01  pc 0000d15d  /system/lib/libnativehelper.so
  #02  pc 00020230  /system/lib/libdvm.so (dvmPlatformInvoke+112)
  | schedstat=( 0 0 0 ) utm=239 stm=240 core=1
  at java.lang.ProcessManager.exec(Native Method)
  at java.lang.ProcessManager.exec(ProcessManager.java:209)
  #00  pc 0000d7e8  /system/lib/libc.so (__futex_syscall3+8)
  #01  pc 00011f90  /system/lib/libc.so
  #02  pc 00012480  /system/lib/libc.so (pthread_mutex_unlock+140)
  #03  pc 0002746d  /system/lib/libc.so (fork+40)    __bionic_atfork_run_child
  #04  pc 0000d077  /system/lib/libnativehelper.so
  #05  pc 00020230  /system/lib/libdvm.so (dvmPlatformInvoke+112)
```

Case Study 3:

- A broadcase ANR sample

  For this kind of issue, we can check the related code which is handling the broadcase event in the receiver or check the ANR trace file. The event for this case is ‘ACTION_SCREEN_OFF’.

01-06 08:27:24.009: WARN/ActivityManager(205): Timeout of broadcast BroadcastRecord{40a07b90 android.intent.action.SCREEN_OFF} - receiver=android.os.BinderProxy@409cc020, started 10000ms ago
01-06 08:27:24.009: WARN/ActivityManager(205): Receiver during timeout: BroadcastFilter{409cc340 ReceiverList{409cc2a0 329 com.android.phone/1001 remote:409cc020}}
01-06 08:27:24.939: ERROR/ActivityManager(205): ANR in com.android.phone
01-06 08:27:24.939: ERROR/ActivityManager(205): Reason: Broadcast of Intent { act=**android.intent.action.SCREEN_OFF** flg=0x50000000 }
01-06 08:27:24.939: ERROR/ActivityManager(205): Load: 9.25 / 2.81 / 0.98
01-06 08:27:24.939: ERROR/ActivityManager(205): CPU usage from 14100ms to -1ms ago:
01-06 08:27:24.939: ERROR/ActivityManager(205): 38% 19/kworker/0:2: 0% user + 38% kernel
……

01-06 08:27:24.939: ERROR/ActivityManager(205): 25% TOTAL: 13% user + 5.3% kernel + 6.5% iowait
01-06 08:27:24.939: ERROR/ActivityManager(205): CPU usage from 368ms to 881ms later:
01-06 08:27:24.939: ERROR/ActivityManager(205): 33% 19/kworker/0:2: 0% user + 33% kernel
01-06 08:27:24.939: ERROR/ActivityManager(205): 3.8% 205/system_server: 0% user + 3.8% kernel / faults: 5 minor
01-06 08:27:24.939: ERROR/ActivityManager(205): 3.8% 238/ActivityManager: 0% user + 3.8% kernel
01-06 08:27:24.939: ERROR/ActivityManager(205): 1.9% 211/Binder Thread #: 1.9% user + 0% kernel
01-06 08:27:24.939: ERROR/ActivityManager(205): 1.3% 74/yaffs-bg-1: 1.3% user + 0% kernel
01-06 08:27:24.939: ERROR/ActivityManager(205): 5.7% TOTAL: 1.9% user + 3.8% kernel

Case Study 3:

- From the ANR trace file ‘traces.txt’, we can find a deadlock which is blocked in the function ‘nativeSetScreenState’:

```
"PowerManagerService" prio=5 tid=21 MONITOR
  | group="main" sCount=1 dsCount=0 obj=0x416ff768 self=0x526a06a0
  | sysTid=556 nice=0 sched=0/0 cgrp=apps handle=1380613712
  | schedstat=( 0 0 0 ) utm=31 stm=39 core=0
  at com.android.server.PowerManagerService$6.onReceive(PowerManagerService.java:~1699)
  - waiting to lock <0x41497f38> (a com.android.server.PowerManagerService$LockList) held by tid=28 (InputReader)
  at android.app.LoadedApk$ReceiverDispatcher$Args.run(LoadedApk.java:755)
  at android.os.Handler.handleCallback(Handler.java:615)
  at android.os.Handler.dispatchMessage(Handler.java:92)
  at android.os.Looper.loop(Looper.java:137)
  at android.os.HandlerThread.run(HandlerThread.java:60)

"InputReader" prio=10 tid=28 MONITOR
  | group="main" sCount=1 dsCount=0 obj=0x4171fdb8 self=0x5136a458
  | sysTid=564 nice=-8 sched=0/0 cgrp=apps handle=1391804880
  | schedstat=( 0 0 0 ) utm=29 stm=27 core=0
  at com.android.server.PowerManagerService$ScreenBrightnessAnimator.isAnimating(PowerManagerService.java:~2516)
  - waiting to lock <0x416ff358> (a com.android.server.PowerManagerService$ScreenBrightnessAnimator) held by tid=20 (mScreenBrightnessUpdaterThread)
  at com.android.server.PowerManagerService.isScreenTurningOffLocked(PowerManagerService.java:2597)
  at com.android.server.PowerManagerService.userActivity(PowerManagerService.java:2685)
  at com.android.server.PowerManagerService.userActivity(PowerManagerService.java:2651)
  at dalvik.system.NativeStart.run(Native Method)

"mScreenBrightnessUpdaterThread" prio=5 tid=20 NATIVE
  | group="main" sCount=1 dsCount=0 obj=0x416ff358 self=0x52fd8810
  | sysTid=555 nice=-4 sched=0/0 cgrp=apps handle=1371346000
  | schedstat=( 0 0 0 ) utm=1 stm=16 core=0
  at com.android.server.PowerManagerService.nativeSetScreenState(Native Method)
  at com.android.server.PowerManagerService.setScreenStateLocked(PowerManagerService.java:1892)
  at com.android.server.PowerManagerService.screenOffFinishedAnimatingLocked(PowerManagerService.java:2103)
  at com.android.server.PowerManagerService.access$6100(PowerManagerService.java:83)
  at com.android.server.PowerManagerService$ScreenBrightnessAnimator.animateInternal(PowerManagerService.java:2433)
  at com.android.server.PowerManagerService$ScreenBrightnessAnimator.access$5500(PowerManagerService.java:2311)
  at com.android.server.PowerManagerService$ScreenBrightnessAnimator$1.handleMessage(PowerManagerService.java:2363)
  at android.os.Handler.dispatchMessage(Handler.java:99)
  at android.os.Looper.loop(Looper.java:137)
  at android.os.HandlerThread.run(HandlerThread.java:60)
```

Case Study 3:

- Check the code ‘PowerManagerService.java:~1699’, it is stuck by the lock mLocks.

```
private BroadcastReceiver mScreenOffBroadcastDone = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        synchronized (mLocks) {
            EventLog.writeEvent(EventLogTags.POWER_SCREEN_BROADCAST_DONE, 0,
                    SystemClock.uptimeMillis() - mScreenOffStart, mBroadcastWakeLock.mCount);
            mBroadcastWakeLock.release();
        }
    }
};
```

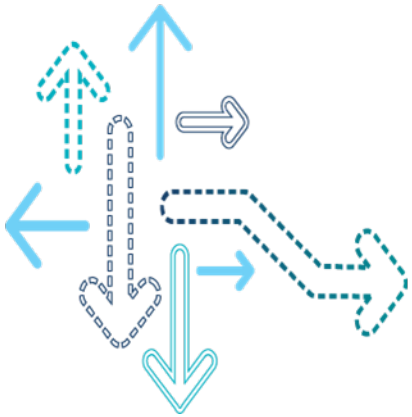
- mScreenOffBroadcastDone is set as below:

```
mContext.sendOrderedBroadcast(mScreenOffIntent, null,
                    mScreenOffBroadcastDone, mHandler, 0, null, null);
```

- mScreenOffIntent is defined as below:

```
mScreenOffIntent = new Intent(Intent.ACTION_SCREEN_OFF);
```

Android Watchdog



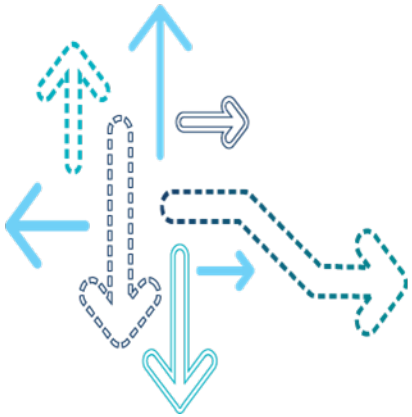
Android Watchdog

- Android has the mechanism for important android services (ActivityManagerService, WindowManagerService, PowerMangerService) stuck(deadlock) situation, it's android watchdog. If one of these important processes is stuck, while the watchdog is time out, the watchdog will kill this process. Sometimes the deadlock is caused because the status of hardware, kernel, driver or modem are not correct, or there is a deadlock in the kernel/driver, Android Framework or native code.
- This will cause "system_server" process be killed since the ActivityManagerService, WindowManagerService, PowerMangerService are running in the "system_server" process.
- Dalvik(Android process virtual machine) will kill itself if it finds "system_server" is killed because "system_server" is the most important service for the Android world.
- If Dalvik is killed, it will be restarted by 'init' process.
- So the watchdog will cause the whole Android world restart.

Android Watchdog

- If you want to disable the framework watchdog for debugging sometimes, you can use adb command below:
  - adb shell setprop **sys**.watchdog.disabled 1
- If you want to debug the watchdog issue on the live target, you need to disable the framework reboot using this command.

Case Study



Case Study 1:

- Some time the UI is frozen, not any response while touch the screen.
- From the logcat log we can find below log:

  06-28 04:12:00.391 506 715 W Watchdog: *** WATCHDOG KILLING SYSTEM PROCESS: com.android.server.am.ActivityManagerService
- So the system process ‘com.android.server.am.ActivityManagerService’ has been killed because of watchdog time out.
- Sometimes we can find the log like below while the watchdog issue happens:

  06-28 04:12:00.391 I/DEBUG (1775): pid: 214, tid: 314 >>> system_server <<<
  06-28 04:12:00.391 I/DEBUG (1775): signal 6 (SIGABRT), code 0 (?), fault addr --------
  06-28 04:12:00.391 I/DEBUG (1775): r0 00d3d228 r1 00000080 r2 00000002 r3 00000000
- The ActivityManagerService is killed while the watchdog issue happens which causes the system server aborted. The execution point of system_server is not important for analysing the issue in most of the situation, while the thread killed by watchdog is the most important thing we should check.

Case Study 1:

- From the ANR log ‘/data/anr/traces.txt’, search ‘com.android.server.am.ActivityManagerService’, we can find many lines like below:

```
"android.server.ServerThread" prio=5 tid=11 MONITOR
  | group="main" sCount=1 dsCount=0 obj=0x42472d10 self=0x577d0948
  | sysTid=521 nice=-2 sched=0/0 cgrp=apps handle=1077763776
  | schedstat=( 0 0 0 ) utm=244 stm=64 core=0
  at com.android.server.am.ActivityManagerService.monitor(ActivityManagerService.java:~15275)
  - waiting to lock <0x4247cec0> (a com.android.server.am.ActivityManagerService) held by tid=64
(Binder_5)
  at com.android.server.Watchdog$HeartbeatHandler.handleMessage(Watchdog.java:134)
  at android.os.Handler.dispatchMessage(Handler.java:99)
  at android.os.Looper.loop(Looper.java:137)
  at com.android.server.ServerThread.run(SystemServer.java:900)
```

Case Study 1:

- From the ANR log '/data/anr/traces.txt', search 'tid=64', we can find call stack as below:

"Binder_5" prio=5 tid=64 **NATIVE**
| group="main" sCount=1 dsCount=0 obj=0x426ba6b0 self=0x5cddf218
| sysTid=788 nice=0 sched=0/0 cgrp=apps handle=1558054920
| schedstat=(0 0 0) utm=49 stm=9 core=0
#00 pc 0000cb90 /system/lib/libc.so (__ioctl+8)
#01 pc 00027fed /system/lib/libc.so (ioctl+16)
#02 pc 00016cc9 /system/lib/libbinder.so (android::IPCThreadState::talkWithDriver(bool)+124)
#03 pc 000171b5 /system/lib/libbinder.so (android::IPCThreadState::waitForResponse(android::Parcel*, int*)+44)
#04 pc 000173cf /system/lib/libbinder.so (android::IPCThreadState::transact(int, unsigned int, android::Parcel const&, android::Parcel*, unsigned int)+114)
#05 pc 00014b83 /system/lib/libbinder.so (android::BpBinder::transact(unsigned int, android::Parcel const&, android::Parcel*, unsigned int)+34)
#06 pc 000228bf /system/lib/libgui.so
#07 pc 000258d3 /system/lib/libgui.so (android::ScreenshotClient::update(unsigned int, unsigned int, unsigned int, unsigned int)+64)
#08 pc 00057dad /system/lib/libandroid_runtime.so
……

Case Study 1:

#25 pc 000125c8 /system/lib/libc.so (pthread_create+172)

at android.view.Surface.screenshot(Native Method)

at com.android.server.wm.WindowManagerService.screenshotApplications(WindowManagerService.java:5785)

at com.android.server.am.ActivityStack.screenshotActivities(ActivityStack.java:937)

at com.android.server.am.ActivityStack.startPausingLocked(ActivityStack.java:962)

at com.android.server.am.ActivityStack.resumeTopActivityLocked(ActivityStack.java:1507)

at com.android.server.am.ActivityStack.resumeTopActivityLocked(ActivityStack.java:1397)

at com.android.server.am.ActivityStack.finishTaskMoveLocked(ActivityStack.java:4289)

at com.android.server.am.ActivityStack.moveTaskToFrontLocked(ActivityStack.java:4284)

at com.android.server.am.ActivityStack.startActivityUncheckedLocked(ActivityStack.java:2785)

at com.android.server.am.ActivityStack.startActivityLocked(ActivityStack.java:2647)

at com.android.server.am.ActivityStack.startActivityMayWait(ActivityStack.java:3195)

at com.android.server.am.ActivityManagerService.startActivity(ActivityManagerService.java:2433)

at android.app.ActivityManagerNative.onTransact(ActivityManagerNative.java:130)

at com.android.server.am.ActivityManagerService.onTransact(ActivityManagerService.java:1676)

at android.os.Binder.transact(Binder.java:326)

at com.lbe.security.service.core.internal.d.onTransact((null):-1)

at android.os.Binder.execTransact(Binder.java:367)

at dalvik.system.NativeStart.run(Native Method)

Case Study 1:

- So the ‘com.android.server.am.ActivityManagerService’ process has been stuck by the native method ‘**screenshot**’.
- Now we should check this native method for more information and find the root cause.
- For this situation, we can also try to check the surfaceflinger's tombstone log as the last call in View.Surface.
- From the tombstone log, if we find it has relationship with other processes (e.g. mediaserver, surfaceflinger), we need to get the correct tombstone log with the processes.
- About how to get the tombstone of the processes, please refer to the Section ‘**Create tombstones**’.

Case Study 2:

- Another deadlock issue caused watchdog timeout

  From the '/data/anr/traces_SystemServer_WDT.txt', search 'waiting to lock', many logs like below:

  - waiting to lock <0x42addea0> (a com.android.server.am.ActivityManagerService) held by tid=76 (Binder_C)

  - waiting to lock <0x42b54af0> (a java.util.HashMap) held by tid=22 (WindowManager)

  - waiting to lock <0x42c04ad0> (a java.util.HashMap) held by tid=79 (GpsLocationProvider)

Case Study 2:

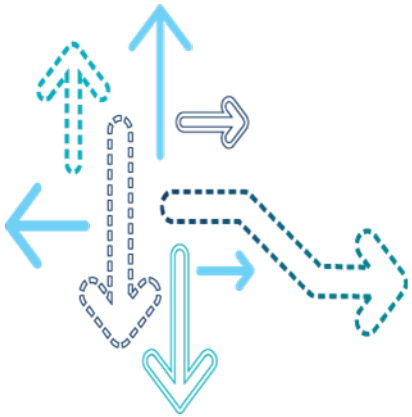
- Check these threads stack:

```
"Binder_C" prio=5 tid=76 MONITOR
  | group="main" sCount=1 dsCount=0 obj=0x43157f48 self=0x71f92530
 | sysTid=1074 nice=0 sched=0/0 cgrp=apps handle=1096888200
 | schedstat=( 0 0 0 ) utm=1690 stm=492 core=1
  at com.android.server.wm.WindowManagerService.resumeKeyDispatching(WindowManagerService.java:~6848)
 - waiting to lock <0x42b54af0> (a java.util.HashMap) held by tid=22 (WindowManager)
  at com.android.server.am.ActivityRecord.resumeKeyDispatchingLocked(ActivityRecord.java:663)
  at com.android.server.am.ActivityStack.completePauseLocked(ActivityStack.java:1156)
  at com.android.server.am.ActivityStack.activityPaused(ActivityStack.java:1043)
  at com.android.server.am.ActivityManagerService.activityPaused(ActivityManagerService.java:4329)
  at android.app.ActivityManagerNative.onTransact(ActivityManagerNative.java:381)
 at com.android.server.am.ActivityManagerService.onTransact(ActivityManagerService.java:1617)
  at android.os.Binder.execTransact(Binder.java:367)
 at dalvik.system.NativeStart.run(Native Method)

"WindowManager" prio=5 tid=22 MONITOR
 | group="main" sCount=1 dsCount=0 obj=0x42b55f38 self=0x723e3d28
  | sysTid=800 nice=-4 sched=0/0 cgrp=apps handle=1927383888
 | schedstat=( 0 0 0 ) utm=334 stm=144 core=0
  at android.app.ActivityThread.releaseProvider(ActivityThread.java:~4380)
  - waiting to lock <0x42c04ad0> (a java.util.HashMap) held by tid=79 (GpsLocationProvider)
  at android.app.ContextImpl$ApplicationContentResolver.releaseProvider(ContextImpl.java:1770)
  at android.content.ContentResolver.insert(ContentResolver.java:873)
  at android.provider.Settings$NameValueTable.putString(Settings.java:681)
  at android.provider.Settings$System.putString(Settings.java:884)
  at android.provider.Settings$System.putInt(Settings.java:967)

"GpsLocationProvider" prio=5 tid=79 MONITOR
  | group="main" sCount=1 dsCount=0 obj=0x4320f000 self=0x4032b698
  | sysTid=1126 nice=10 sched=0/0 cgrp=apps/bg_non_interactive handle=1077356040
 | schedstat=( 0 0 0 ) utm=8 stm=2 core=2
  at com.android.server.am.ActivityManagerService.refContentProvider(ActivityManagerService.java:~6646)
  - waiting to lock <0x42addea0> (a com.android.server.am.ActivityManagerService) held by tid=76 (Binder_C)
  at android.app.ActivityThread.incProviderRefLocked(ActivityThread.java:4314)
 at android.app.ActivityThread.acquireExistingProvider(ActivityThread.java:4368)
 at android.app.ActivityThread.acquireProvider(ActivityThread.java:4258)
 at android.app.ContextImpl$ApplicationContentResolver.acquireProvider(ContextImpl.java:1760)
```

Case Study 2:

- For this issue, there is a deadlock between thread 22, 76 and 79.
- It should be Google original issue. We can find some similar issues from Google web site:
- Issue 52890:
  Watchdog ANR due to deadlock between LocationManagerService and ActivityManagerService - Android 4.2.2 Build JDQ39
  https://code.google.com/p/android/issues/detail?id=52890
- Issue 41465:
  Watchdog by GpsLocationProvider and ActivityThreads
  https://code.google.com/p/android/issues/detail?id=41465
- Issue 46806:
  Watchdog timer reboot between Binder and GpsLocationProvider on ActivityManagerService
  https://code.google.com/p/android/issues/detail?id=46806

Android system_server Crash



Android system_server Crash

- Sometimes we can find the SEGFAULT (signal 11) or SIGBUS (signal 7) from the logcat logs, usually it's because the application attempts to access a memory location that it is not allowed to access, or in a way that is not allowed.
- Every time a native crash happens, it generates a file called tombstone, which is written for the process. A tombstone is a file containing important information about the process when it crashes.
- We can check the tombstone file from "/data/tombstones/" which corresponds to the crash.
- We should restore the call stack from tomsbaones log as below to get the related code files and lines:
  - /development/scripts/stack --symbols-dir=out/target/product/<board>/symbols <tombstone_file>

Crash During Garbage Collection

```
01-05 05:38:52.612 I/DEBUG   (  224):        #00  pc 0004420c  /system/lib/libdvm.so
01-05 05:38:52.612 I/DEBUG   (  224):        #01  pc 00044404  /system/lib/libdvm.so
01-05 05:38:52.612 I/DEBUG   (  224):        #02  pc 00034900  /system/lib/libdvm.so
(_Z25dvmCollectGarbageInternalPK6GcSpec)
01-05 05:38:52.612 I/DEBUG   (  224):        #03  pc 0007aa7a  /system/lib/libdvm.so
01-05 05:38:52.622 I/DEBUG   (  224):        #04  pc 0005fc96  /system/lib/libdvm.so
01-05 05:38:52.622 I/DEBUG   (  224):        #05  pc 00012dac  /system/lib/libc.so (__thread_entry)
01-05 05:38:52.622 I/DEBUG   (  224):        #06  pc 00012900  /system/lib/libc.so (pthread_create)
```

- Debug tips for such issues:
  - 1) Disable JIT In build.prop: dalvik.vm.execution-mode = int: fast
    - NOTE: build.prop is auto-generated from buildinfo.sh. So, to make this change, easiest thing will be to pull build.prop from device, change it and push the changed file back
  - 2) Enable extra dalvik GC logs: In build.prop: dalvik.vm.extra-opts "-Xgc:preverify -Xgc:postverify"
  - 3) Collect dalvik heap dumps at the time of crash Change code in the android tree to collect heap dumps.There are 2 ways of doing it:
    - To do it in JAVA: android.os.Debug.dumpHprofData(String filename) API can be used
    - To do it in CPP: dalvik.vm.hprof.Hprof.hprofDumpHeap(String filename,-1,false) API can be used
  - 4) Enable malloc debug logs

    adb shell
    #setprop dalvik.vm.checkjni true
    #setprop libc.debug.malloc 10
    #setprop dalvik.vm.jniopts forcecopy
    #stop
    #start

Unhandled Exception in system_server

- The uncaught exception in the following services will cause the framework reboot
- ActivityManagerService
- PackageManagerService
- ContentService
- WindowManagerService
- NetworkTimeUpdateService
- and some more...

Unhandled Exception in system_server

02-08 06:04:21.549 207 2057 D QMI_FW : QCCI: QMI_CCI_TX: cntl_flag - 00, txn_id - 0003, msg_id - 0020, msg_len - 0012
02-08 06:04:21.549 207 2057 D QMI_FW : QCCI: Sent[18]: 25 bytes to port 1792
02-08 06:04:21.549 473 766 E AndroidRuntime: * FATAL EXCEPTION IN SYSTEM PROCESS: NetworkTimeUpdateService**
02-08 06:04:21.549 473 766 E AndroidRuntime: java.lang.NullPointerException
02-08 06:04:21.549 473 766 E AndroidRuntime: at com.android.server.NetworkTimeUpdateService.onPollNetworkTime(NetworkTimeUpdateService.java:177)
02-08 06:04:21.549 473 766 E AndroidRuntime: at com.android.server.NetworkTimeUpdateService.access$300(NetworkTimeUpdateService.java:51)
02-08 06:04:21.549 473 766 E AndroidRuntime: at com.android.server.NetworkTimeUpdateService$MyHandler.handleMessage(NetworkTimeUpdateService.java:265)
02-08 06:04:21.549 473 766 E AndroidRuntime: at android.os.Handler.dispatchMessage(Handler.java:99)
02-08 06:04:21.549 473 766 E AndroidRuntime: at android.os.Looper.loop(Looper.java:137)
02-08 06:04:21.549 473 766 E AndroidRuntime: at android.os.HandlerThread.run(HandlerThread.java:60)
……
02-08 06:04:21.649 183 183 W AudioFlinger: power manager service died !!!
02-08 06:04:21.649 1869 1880 W Sensors : sensorservice died [0x126bf08]
02-08 06:04:21.649 777 1518 W Sensors : sensorservice died [0x11756a0]
02-08 06:04:21.649 777 1518 I ActivityThread: Removing dead content provider: settings
……
02-08 06:04:21.649 750 765 I ActivityThread: Removing dead content provider: settings
02-08 06:04:21.649 692 705 I ActivityThread: Removing dead content provider: settings
02-08 06:04:21.659 883 883 E StrictMode: RemoteException trying to handle StrictMode violation
02-08 06:04:21.659 883 883 E StrictMode: android.os.DeadObjectException
02-08 06:04:21.659 883 883 E StrictMode: at android.os.BinderProxy.transact(Native Method)
02-08 06:04:21.659 883 883 E StrictMode: at android.app.ActivityManagerProxy.handleApplicationStrictModeViolation(ActivityManagerNative.java:2900)

Unhandled Exception in system_server

```
02-08 06:04:21.659   883   883 E StrictMode:          at
android.os.StrictMode$AndroidBlockGuardPolicy.handleViolation(StrictMode.java:1306)
02-08 06:04:21.659   883   883 E StrictMode:          at
android.os.StrictMode$AndroidBlockGuardPolicy$1.run(StrictMode.java:1206)
02-08 06:04:21.659   883   883 E StrictMode:          at android.os.Handler.handleCallback(Handler.java:605)
02-08 06:04:21.659   883   883 E StrictMode:          at android.os.Handler.dispatchMessage(Handler.java:92)
02-08 06:04:21.659   883   883 E StrictMode:          at android.os.Looper.loop(Looper.java:137)
02-08 06:04:21.659   883   883 E StrictMode:          at android.app.ActivityThread.main(ActivityThread.java:4424)
02-08 06:04:21.659   883   883 E StrictMode:          at java.lang.reflect.Method.invokeNative(Native Method)
02-08 06:04:21.659   883   883 E StrictMode:          at java.lang.reflect.Method.invoke(Method.java:511)
02-08 06:04:21.659   883   883 E StrictMode:          at
com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:787)
02-08 06:04:21.659   883   883 E StrictMode:          at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:554)
02-08 06:04:21.659   883   883 E StrictMode:          at dalvik.system.NativeStart.main(Native Method)
02-08 06:04:21.659   180   321 D memalloc: ion: Freeing buffer base:0x40038000 size:4096 fd:38
02-08 06:04:21.659   180   321 D memalloc: ion: Unmapping buffer  base:0x40038000 size:4096
```

- The block which is highlighted in **RED** is the reason for the framework reboot and the blocks which are highlighted in **BLACK** are the framework reboot messages.
- You can find the reason for the framework reboot within one or two seconds (timestamp in the log) just before the ***eof*** message in the UI log.

Debugging Segmentation Fault

- Sometimes we see crashes in the kernel log with the thread id and the register values, but we do not see the tombstones or the backtrace in the UI log.

```
<7>[29216.377747] PowerManagerSer: unhandled page fault (11) at 0x00001eef, code 0x017
<1>[29216.377747] pgd = d7748000
<1>[29216.377777] [00001eef] *pgd=9ea1e831, *pte=00000000, *ppte=00000000
<4>[29216.377777]
<4>[29216.377808] Pid: 591, comm: PowerManagerSer
<4>[29216.377808] CPU: 0 Tainted: G W (3.0.8-00016-g9392d0c #2)
<4>[29216.377838] PC is at 0x406698c0
<4>[29216.377838] LR is at 0x406cf724
<4>[29216.377838] pc : [<406698c0>] lr : [<406cf724>] psr: 20000010
```

- We need the *pc* and *pid.* From the *pid* we can find out the process which crashed (You may get this info from the UI log or you can always run the *top* command just before you start the test).
- Check whether the new instance of the same process is running (If not, start it).
- Get the current pid for that process.
- Get the maps info for that process from the device:
  - adb shell cat /proc/<pid>/maps

Debugging Segmentation Fault

- Find the address range for the *pc* value

4063a000-40762000 r-xp 00000000 b3:0c 1002 /system/lib/libskia.so

- Get the objdump:

arm-linux-androideabi-objdump -S out/target/product/msm8960/symbols/system/lib/libskia.so | tee objdumpSkia.txt

- Search for the offset value which is from the *pc* value in the objdump file (base: 0x406XXXXX Offset: 698c0)

```
static void src_over_4444(SkPMColor16 dst[], SkPMColor16 color,
                          SkPMColor16 other, unsigned invScale, int count) {
  int twice = count >> 1;
  while (--twice >= 0) {
      *dst = color + SkAlphaMulQ4(*dst, invScale);
698a8: e1d3c0b0  ldrh  ip, [r3]
698ac: e30f70f0  movw  r7, #61680  ; 0xf0f0
      dst++;
      *dst = other + SkAlphaMulQ4(*dst, invScale);
698b0: e1d300b2  ldrh  r0, [r3, #2]
698b4: e3407000  movt  r7, #0
698b8: e3001f0f  movw  r1, #3855 ; 0xf0f
698bc: e1a04007  mov r4, r7
698c0: e3401000  movt  r1, #0
698c4: e00c7007  and r7, ip, r7
698c8: e1a08001  mov r8, r1
698cc: e0004004  and r4, r0, r4
698d0: e00c1001  and r1, ip, r1
698d4: e0002008  and r2, r0, r8
698d8: e1818607  orr r8, r1, r7, lsl #12
698dc: e1824604  orr r4, r2, r4, lsl #12
```

Random Unhandled Exception

- Sometimes we can find the SEGV_MAPERR or SEGV_ACCERR error randomly like below:

```
06-13 15:44:04.472 I/DEBUG   (  175): pid: 18463, tid: 18463, timestamp: 06-13 15:44:04.480 , name:
ndroid.settings  >>> com.android.settings <<<
06-13 15:44:04.472 I/DEBUG   (  175): signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr c018620d
06-13 15:44:04.652 I/DEBUG   (  175):     r0 00000000  r1 bea2e0e4  r2 00000000  r3 00000000
```

```
03-28 09:40:05.330 137 137 I DEBUG : pid: 499, tid: 501, name: GC >>> system_server <<<
03-28 09:40:05.330 137 137 I DEBUG : signal 11 (SIGSEGV), code 2 (SEGV_ACCERR), fault addr 4cbbb8a6
03-28 09:40:05.480   137   137 I DEBUG   :     r0 04000000  r1 41f2a828  r2 00000014  r3 40df9000
```

- The error can happen in any of the process, one of the possible reason is that the memory is not stable. Need check the DDR.
- If this issue happen every time with the same phenomenon(**Not randomly**), perhaps there is a memory usage issue at somewhere, refer to the chapter “Debugging segmentation fault” about how to debug this kind of issue.

Kernel OOM Killer Cause Framework Reboot

- Low memory killer is part of the kernel monitoring remaining available memory.
- If available memory goes under the determined threshold, it chooses victim process and kills it.
- It is calculating available memory by summing up *active* + *nonactive* pages. You can check that with the command: ‘cat /proc/meminfo’.
- When the kernel OOM killer kills one of the following process, the framework will reboot.
  - system_server
  - surfaceflinger
  - Servicemanager
- We can check the kernel log abut the OOM, if necessary, we need to debug the memory leak in kernel or userspace.

Kernel OOM Killer Cause Framework Reboot

- Here is a sample kernel log about OOM:

<4>[239.031619] [(2013-08-01 02:29:47.940946647 UTC)] [cpuid: 0] geomagneticd: page allocation failure: order:7, mode:0xc0d0
<4>[239.031701] [(2013-08-01 02:29:47.941026647 UTC)] [cpuid: 0] [<c0014888>] (unwind_backtrace+0x0/0x11c) from [<c0116d18>] (warn_alloc_failed+0xe8/0x110)
<4>[239.031741] [(2013-08-01 02:29:47.941066647 UTC)] [cpuid: 0] [<c0116d18>] (warn_alloc_failed+0xe8/0x110) from [<c01193d0>] (__alloc_pages_nodemask+0x654/0x758)
<4>[239.031776] [(2013-08-01 02:29:47.941101647 UTC)] [cpuid: 0] [<c01193d0>] (__alloc_pages_nodemask+0x654/0x758) from [<c01194e4>] (__get_free_pages+0x10/0x24)
<4>[239.031811] [(2013-08-01 02:29:47.941134981 UTC)] [cpuid: 0] [<c01194e4>] (__get_free_pages+0x10/0x24) from [<c0142dbc>] (kmalloc_order_trace+0x20/0xd4)
<4>[239.031848] [(2013-08-01 02:29:47.941173314 UTC)] [cpuid: 0] [<c0142dbc>] (kmalloc_order_trace+0x20/0xd4) from [<c03b8158>] (evdev_open+0x94/0x210)
<4>[239.031883] [(2013-08-01 02:29:47.941208314 UTC)] [cpuid: 0] [<c03b8158>] (evdev_open+0x94/0x210) from [<c03b6980>] (input_open_file+0x98/0x100)
<4>[239.031918] [(2013-08-01 02:29:47.941243314 UTC)] [cpuid: 0] [<c03b6980>] (input_open_file+0x98/0x100) from [<c014bb70>] (chrdev_open+0x11c/0x144)
……
(do_filp_open+0x2c/0x78)
<4>[239.032091] [(2013-08-01 02:29:47.941416647 UTC)] [cpuid: 0] [<c0154a6c>] (do_filp_open+0x2c/0x78) from [<c0147574>] (do_sys_open+0xd8/0x170)
<4>[239.032126] [(2013-08-01 02:29:47.941451647 UTC)] [cpuid: 0] [<c0147574>] (do_sys_open+0xd8/0x170) from [<c000ddc0>] (ret_fast_syscall+0x0/0x30)
<4>[239.032151] [(2013-08-01 02:29:47.941474981 UTC)] [cpuid: 0] Mem-info:
<4>[239.032169] [(2013-08-01 02:29:47.941494981 UTC)] [cpuid: 0] Normal per-cpu:
<4>[239.032189] [(2013-08-01 02:29:47.941514981 UTC)] [cpuid: 0] CPU 0: hi: 186, btch: 31 usd: 0
<4>[239.032211] [(2013-08-01 02:29:47.941536647 UTC)] [cpuid: 0] CPU 1: hi: 186, btch: 31 usd: 29
<4>[239.032243] [(2013-08-01 02:29:47.941566647 UTC)] [cpuid: 0] active_anon:57346 inactive_anon:75 isolated_anon:0
<4>[239.032258] [(2013-08-01 02:29:47.941583314 UTC)] [cpuid: 0] active_file:7476 inactive_file:7495 isolated_file:0
<4>[239.032274] [(2013-08-01 02:29:47.941598314 UTC)] [cpuid: 0] unevictable:639 dirty:0 writeback:49 unstable:0
<4>[239.032289] [(2013-08-01 02:29:47.941614981 UTC)] [cpuid: 0] free:1399 slab_reclaimable:2671 slab_unreclaimable:2794
<4>[239.032306] [(2013-08-01 02:29:47.941629981 UTC)] [cpuid: 0] mapped:13148 shmem:92 pagetables:2762 bounce:0
<4>[239.032384] [(2013-08-01 02:29:47.941709981 UTC)] [cpuid: 0] Normal free:5596kB min:5120kB low:6400kB high:7680kB active_anon:229384kB inactive_anon:300kB active_file:29904kB inactive_file:29980kB unevictable:2556kB isolated(anon):0kB isolated(file):0kB present:408988kB mlocked:0kB dirty:0kB writeback:196kB mapped:52592kB shmem:368kB slab_reclaimable:10684kB slab_unreclaimable:11176kB kernel_stack:6520kB pagetables:11048kB unstable:0kB bounce:0kB writeback_tmp:0kB pages_scanned:29 all_unreclaimable? no

Kernel OOM Killer Cause Framework Reboot

<4>[239.032479] [(2013-08-01 02:29:47.941804981 UTC)] [cpuid: 0] lowmem_reserve[]: 0 0 0

<4>[239.032578] [(2013-08-01 02:29:47.941903314 UTC)] [cpuid: 0] Normal: 111*4kB 58*8kB 33*16kB 8*32kB 5*64kB 0*128kB 0*256kB 1*512kB 1*1024kB 1*2048kB 0*4096kB = 5596kB

<4>[239.032639] [(2013-08-01 02:29:47.941963314 UTC)] [cpuid: 0] 15693 total pagecache pages

<4>[239.042563] [(2013-08-01 02:29:47.951889981 UTC)] [cpuid: 0] 117504 pages of RAM

<4>[239.042589] [(2013-08-01 02:29:47.951914981 UTC)] [cpuid: 0] 2555 free pages

<4>[239.042609] [(2013-08-01 02:29:47.951933314 UTC)] [cpuid: 0] 19677 reserved pages

<4>[239.042628] [(2013-08-01 02:29:47.951951647 UTC)] [cpuid: 0] 3905 slab pages

<4>[239.042644] [(2013-08-01 02:29:47.951969981 UTC)] [cpuid: 0] 230256 pages shared

<4>[239.042663] [(2013-08-01 02:29:47.951988314 UTC)] [cpuid: 0] 0 pages swap cached

<6>[239.174963] [(2013-08-01 02:29:48.084289981 UTC)] [cpuid: 1] init: waitpid returned pid 2309, status = 00000000

<5>[239.175006] [(2013-08-01 02:29:48.084331648 UTC)] [cpuid: 1] init: process 'geomagneticd', pid 2309 exited

<5>[239.175036] [(2013-08-01 02:29:48.084361648 UTC)] [cpuid: 1] init: process 'geomagneticd' killing any children in process group

<6>[239.294181] [(2013-08-01 02:29:48.203508315 UTC)] [cpuid: 1] init: waitpid returned pid 2308, status = 00000100

<5>[239.294223] [(2013-08-01 02:29:48.203548315 UTC)] [cpuid: 1] init: process 'orientationd', pid 2308 exited

<5>[239.294253] [(2013-08-01 02:29:48.203578315 UTC)] [cpuid: 1] init: process 'orientationd' killing any children in process group

<4>[239.964594] [(2013-08-01 02:29:48.873923315 UTC)] [cpuid: 1] select 1020 (ndroid.contacts), adj 470, size 6508, to kill

<4>[239.964668] [(2013-08-01 02:29:48.873994981 UTC)] [cpuid: 1] select 1114 (hter.testcamera), adj 647, size 5904, to kill

<4>[239.964726] [(2013-08-01 02:29:48.874051648 UTC)] [cpuid: 1] select 1185 (droid.deskclock), adj 647, size 6340, to kill

<4>[239.964858] [(2013-08-01 02:29:48.874184981 UTC)] [cpuid: 1] select 1348 (com.android), adj 705, size 5903, to kill

<4>[239.964903] [(2013-08-01 02:29:48.874229981 UTC)] [cpuid: 1] select 1370 (ei.android.gpms), adj 705, size 5919, to kill

<4>[239.965133] [(2013-08-01 02:29:48.874459981 UTC)] [cpuid: 1] send sigkill to 1370 (ei.android.gpms), adj 705, size 5919

<4>[242.171961] [(2013-08-01 02:29:51.081288314 UTC)] [cpuid: 1] select 1020 (ndroid.contacts), adj 470, size 6508, to kill

<4>[242.171998] [(2013-08-01 02:29:51.081321647 UTC)] [cpuid: 1] select 1114 (hter.testcamera), adj 647, size 5904, to kill

<4>[242.172021] [(2013-08-01 02:29:51.081346647 UTC)] [cpuid: 1] select 1185 (droid.deskclock), adj 647, size 6340, to kill

<4>[242.172048] [(2013-08-01 02:29:51.081373314 UTC)] [cpuid: 1] select 1348 (com.android), adj 705, size 5903, to kill

<4>[242.172096] [(2013-08-01 02:29:51.081419981 UTC)] [cpuid: 1] send sigkill to 1348 (com.android), adj 705, size 5903

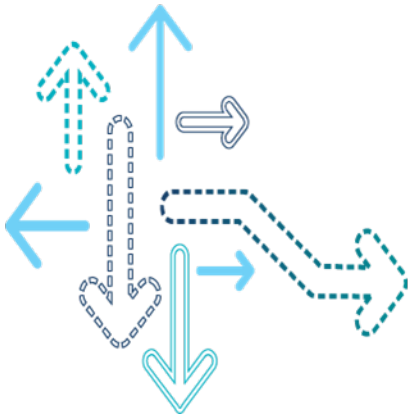
Excessive JNI References

- JNI is the interface used for Java code to communicate with native (C/C++) code. Google has set a limit of 2001 references per dalvik virtual machine, i.e. per dalvik process. If this limit is exceeded, the VM will be killed. This condition is called Excessive JNI references.
- JNI Global references are created to prevent garbage collection of Java objects.
- Check JNI enabled to prevent JNI problems in eng/non-user builds
- If you see the following messages in the UI log, it's because there are too many JNI references which haven't been released:

  01-19 16:03:22.953 405 7636 E dalvikvm: Excessive JNI global references (2001)
  01-19 16:03:22.953 405 7636 E dalvikvm: VM aborting
  01-19 16:03:23.163 405 7636 F libc : Fatal signal 11 (SIGSEGV) at 0xdeadd00d (code=1)

- We can collect the *.**hprof** from the folder /data and collect logcat logs to analyze.
- Firstly, check the JNI reference table in the logcat logs to see which object has too many references. Then use MAT (Memory Analyzer Tool) to analyze the *.hprof files.
- 'StrictMode' can also be used to find the root cause.
- Please refer to Q3 for how to use MAT and 'StrictMode'.

JTAG Debugging on Live Target



JTAG Debugging on Live Target

- When there is a target freeze, that is, the device does not show up on adb and QPST, you can get some information by debugging over JTAG:
  - Get dmesg logs over JTAG
  - Look at the states, stacktraces of tasks.

JTAG Debugging on Live Target

- Please follow these steps to debug over JTAG:
  - 1. Open T32 sessions for RPM and krait
  - 2. In RPM T32

    sys.m a
  - 3. In Krait T32

    sys.m a

    b

    data.load.elf <vmlinux> /nocode

    data.save.binary C:\Dropbox\dmesg.txt __log_buf++40000
  - 4. If you want the stacktrace of a CPU (i.e. what it is doing at a particular time), you can issue the follow command

    v.F
  - 5. To know where the program counter (PC) is, try

    snoop.pc on
  - 6. If PC is in 0xCyyyyyyy, the CPU is executing in kernel and not in userspace.
  - 7. If the issue is in android framework, you can load the linux aware module and look at individual tasks, their individual stacktraces. This is especially helpful when certain userspace tasks are in D state

    task.config C:\T32\demo\arm\kernel\linux\linux.t32

    menu.reprogram C:\T32\demo\arm\kernel\linux\linux.men

  Note that the above commands will create a new menu "Linux" where you can select the tasks.

JTAG Debugging on Live Target

- 8. Once you have the Linux menu along with other menu items at the top of your T32 session, you can click on "Linux" -> "show tasks" to see the kernel tasks:

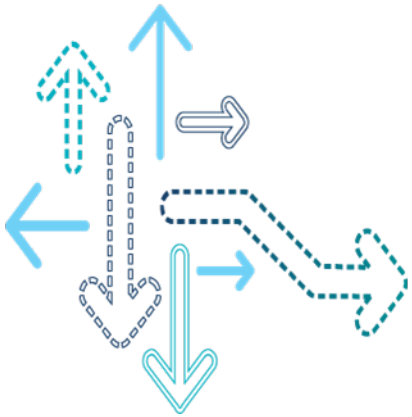
  The following steps (9-11) are based on an example set of logs of a black screen issue where ADB is up. One of system_server's threads is stuck on Surface.nativeScreenshot(). This is a binder call to surfaceflinger. Surfaceflinger in turn is in D state.
- 9. Since we see surfaceflinger in D state, we need to know what surfaceflinger is doing in the kernel.

  We can do this by selecting the process (surfaceflinger) -> right click -> Select StackFrame which will give us the stacktrace for the surfaceflinger task.
- 10. Surfaceflinger is waiting on a mutex held by some other task.

  Find the owner by looking at "comm" field of the owner.
- 11. Once you fine the owner in Step 10, you can find what the owner is doing by following the same method as Step 9:

  Right click on the owner -> Select stackframe

Appendix



Create Tombstones

- Below function ‘createTombstone()’ can be used to create the tombstones for some processes. For example, call ‘createTombstone(“mediaserver,surfaceflinger”)’ can get the tombstones of ‘mediaserver’ and ‘surfaceflinger’.

```
import java.util.StringTokenizer;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Iterator;
import java.util.List;

    private boolean isProcessAlive (String name)
    {
        String cmd;
        String line;

        StringBuffer sb = new StringBuffer();
        sb.append ("ps ");
        sb.append (name);
        cmd = sb.toString ();

        try {
            java.lang.Process p = Runtime.getRuntime().exec(cmd);
            BufferedReader input = new BufferedReader
                (new InputStreamReader(p.getInputStream()));
            while ((line = input.readLine()) != null) {
                String[] tokens = line.split ("\\s+");
                if (tokens[7].equals ("S")) {
                    if (tokens[8].indexOf (name) >= 0) {
                        Slog.w (TAG, line);
                        return true;
                    }
                }
            }
            input.close();
        }
```


Create Tombstones

```
    catch (Exception ex) {
        Slog.w(TAG, ex);
    }

    return false;
}

private void createTombstone(String procList)
{
    String cmd = "ps";
    List<String> procList4Tombstones = new ArrayList<String>();
    String line;

    StringBuffer sb = new StringBuffer();
    sb.append (",");
    sb.append (procList);
    sb.append (",");
    procList = sb.toString ();

    try {
        java.lang.Process p = Runtime.getRuntime().exec(cmd);
        BufferedReader input = new BufferedReader
            (new InputStreamReader(p.getInputStream()));
        while ((line = input.readLine()) != null) {
            String[] tokens = line.split ("\\s+");
            if (tokens.length == 9) {
                String delimiter1 = "/";
                String[] procname = tokens[8].split (delimiter1);
                StringBuffer sb1 = new StringBuffer();
                sb1.append (",");
                sb1.append (procname[procname.length-1]);
                sb1.append (",");
                String pn = sb1.toString ();
                if (procList.indexOf (pn) >= 0) {
                    procList4Tombstones.add (tokens[1]);
                    procList4Tombstones.add (procname[procname.length-1]);
                }
            }
        }
        input.close();
    }
    catch (Exception ex) {
        Slog.w(TAG, ex);
    }
```


Create Tombstones

```
    Iterator<String> it = procList4Tombstones.iterator();
    while ( it.hasNext()) {
        /*
         * Sometimes the debuggerd will exit after the tombstones creation which
         * will not allow the next tombstones generation.
         * Wait till the debuggerd is back.
         */
        Slog.w(TAG, "Waiting for the debuggerd process...");
        while (false == isProcessAlive ("debuggerd")) {
            SystemClock.sleep(500);
        }
        String pid = it.next();
        Slog.w(TAG, "Creating tombstone file for process[" + it.next() + "]");
        SystemClock.sleep(1000);
        Process.sendSignal(Integer.parseInt(pid), 6);
        SystemClock.sleep(1000);
        Process.sendSignal(Integer.parseInt(pid), 6);
        SystemClock.sleep(2000);
    }
}
```

- Please refer to the patch below for detailed information: https://git.quicinc.com/?p=platform%2Fframeworks%2Fbase.git;a=commit;h=bc66849d53e388c0791216493e189c9ca1df0362

Print Android Java Stack Trace

- Use the following code snippet to log a thread's call-stack in Android apps or framework code.

```
import android.util.Log;
...
StackTraceElement[] stackframes = Thread.currentThread().getStackTrace();
Log.e("Stacktrace *** gfx", "**********BEGIN**********");
for(int i=0; i < stackframes.length; i++) {
Log.e("Stacktrace *** gfx", stackframes[i].toString());
}
Log.e("Stacktrace *** gfx", "**********END**********");
```

Print C++ Stack Trace

- Use the following code snippet to log a thread's call-stack in C++ code.

```
#include <utils/CallStack.h>
...
CallStack stack;
stack.update();
stack.dump();
```


Enable Kernel stacktrace

- Kernel stacktraces are not enabled by default on multicore targets. We need it in cases where the java and native stacktrace of a thread are not enough to root cause an issue. In such cases, it helps to know what the userspace thread is doing in the kernel level. Once it is enabled, you can see what a userspace thread is doing by checking its corresponding kernel stacktrace either in /proc</<pid>/stack or in traces.txt
- A framework reboot will result in traces.txt getting collected which should have kernel stacktraces of the threads with it's enabled.
- If you need to see the kernel stacktrace of many different tasks, you can collect bugreport (adb bugreport > bugreport.txt) and then look at the stacktraces of all currently running userspace threads.

Enable Kernel stacktrace

- How to enable it?
- In the file: kernel/arch/arm/kernel/stacktrace.c

```
-#ifdef CONFIG_SMP
-                       /*
-                        * What guarantees do we have here that 'tsk' is not
-                        * running on another CPU?  For now, ignore it as we
-                        * can't guarantee we won't explode.
-                        */
-                       if (trace->nr_entries < trace->max_entries)
-                               trace->entries[trace->nr_entries++] = ULONG_MAX;
-                       return;
-#else
                        data.no_sched_functions = 1;
                        frame.fp = thread_saved_fp(tsk);
                        frame.sp = thread_saved_sp(tsk);
                        frame.lr = 0;                   /* recovered from the stack */
                        frame.pc = thread_saved_pc(tsk);
-#endif
```

Monkey Test

- adb shell monkey -s 100 --throttle 500 -v-v-v --ignore-timeouts --ignore-crashes --ignore-security-exceptions 1000000
- Please refer to R2 for more details.

Collecting Logs Through adb

- Get logcat log, kernel log, bugreport, ANR log and tombstones:
  - adb shell logcat -v threadtime > main.txt
  - adb shell logcat -v threadtime -b events > Event.txt
  - adb shell logcat -v threadtime -b radio > Radio.txt
  - adb shell cat /proc/kmsg > dmesg.txt
  - adb bugreport > bugreport.txt
  - adb pull /data/anr .
  - adb pull /data/tombstones .
- Synchronize adb and kmsg log timestamps:
  - adb shell
  - logcat -v time -f /dev/kmsg | cat /proc/kmsg > /data/log.txt
  - exit
  - adb pull /data/log.txt .

Useful adb Command

- Get linux process list:
  - adb shell top -t -d 1 -n 5 > linux_top_{date-time}.txt
- Get userspace dump:
  - adb shell dumpsys > dumpsys.txt
  - adb shell dumpsys SurfaceFlinger > sf_dump.txt
- Get dump state:
  - adb shell dumpstate > dumpstate.txt
- Get the process stack:
  - debuggerd -b <process_pid>
- Get threads information:
  - adb shell ps -t > threads.txt
- Get memory information:
  - adb shell procrank > memory.txt

Useful adb Command

- Mount debugfs:
  - adb shell mount -t debugfs none /sys/kernel/debug
- Trigger kernel panic
  - echo c > /proc/sysrq-trigger
- Strace tool:
  - strace -p <process_pid>
- Make the system partition read/write:
  - adb shell mount -o rw,remount /dev/block/system /system
- Some other commands:
  - cat /proc/version
  - cat /proc/wakelocks
  - cat /proc/<process_pid>/maps

Useful adb Command

- Start and stop application:
  - adb shell am start -a android.intent.action.VIEW
  - adb shell am start -a android.intent.action.MAIN -c android.intent.category.HOME
  - adb shell am start -n com.android.gallery3d/com.android.camera.Camera
  - adb shell am start -d /data/video/Qtc88.mp4 -t video/mp4 -a android.intent.action.VIEW
  - adb shell am start -a android.intent.action.VIEW -d http://www.cnn.com
  - adb shell am force-stop app_name
  - Please refer to R3 for more information.

Android Prop

- property_set and property_get defined in the file '/system/core/include/cutils/properties.h' for 'C' code to use.
- SystemProperties.get and SystemProperties.set defined in the file '/frameworks/base/core/java/android/os/SystemProperties.java' for 'Java' code to use.
- Some ways to get and set prop from adb shell:
  - adb shell getprop > android_prop.txt
  - adb shell getprop <property>
  - adb shell setprop <property> <value>
  - adb pull /system/build.prop .

Android Prop

- To define/add prop:
  - Change the file ‘buildinfo.sh’:
    - /build/tools/buildinfo.sh
  - Use below from your Android.mk:
    - ADDITIONAL_BUILD_PROPERTIES = <PROP NAME>=<VALUE>

Debug Native Process Memory Allocations

- Turn on Android's Bionic memory debugger by doing the following in adb shell:
  - setprop dalvik.vm.checkjni true
  - setprop libc.debug.malloc 10
  - setprop dalvik.vm.jniopts forcecopy
  - stop
  - start
- Supported values for libc.debug.malloc (debug level values) are:
  - 1 - perform leak detection
  - 5 - fill allocated memory to detect overruns
  - 10 - fill memory and add sentinels to detect overruns
  - 20 - use special instrumented malloc/free routines for the emulator

Debugging UI Freeze

- When a UI freeze is encountered and USB / ADB are still responding, collect the following logs while the device is frozen:
  - adb shell top -t -n 1 > top.txt
  - adb shell bugreport > bugreport.txt
  - adb pull /data/tombstones .
  - adb pull /data/anr .
- If you are debugging a particular process, you can get stacktraces for it by sending signals:
  - Create traces
    - adb shell kill -3 <process_pid>
  - Create tombstones
    - adb shell kill -6 <process_pid>
    - debuggerd <process_pid>
  - Create ANR
    - adb shell kill -19 <process_pid>

Debugging UI Freeze

- Please take note of the followings on the actual device:
  - What screen is the UI stuck on (e.g., home screen, app screen, etc.)?
  - Please run the following experiment: First run the command "adb shell getevent"
    - Touch the touch screen and observe the output of "adb shell getevent“. Do you see touch events being printed on the screen?
    - Press the power and volume up / down buttons. Do you see button press events being printed on the screen? Moreover, the volume up / down buttons usually cause a sound effect when pressed. Do you hear this sound when the buttons are pressed?
  - Check if there is an actual UI freeze, or if the UI is simply not responding to touch input:a
    - Run the command "adb shell input keyevent 3". Refer KeyEvent.java. Replace the "3" with other numbers to see if the UI responds to inputs given through adb.
    - Try to start activites through adb, for example, using: "# am start com.android.settings" etc.
  - If possible, take note of the time shown on the device (in the top right corner, or in the middle for the lock screen). Is the time changing/ticking or is it frozen indefinitely? This will help correlate the freeze with a specific time in the logs.
  - In 'top' logs, see if any of the threads of system_server or surfaceflinger are in D state. If so, you will have to connect JTAG and see their respective kernel level tasks.

Kernel memleak Enable

- Enable leak detector tool in kernel config. CONFIG_DEBUG_KMEMLEAK.
- Mount debugfs:
  - adb shell mount -t debugfs none /sys/kernel/debug
- You can find the file in /sys/kernel/debug/kmemleak

Disable checkjni & JIT

- Disable checkjni:
  - Method 1
    - adb pull /system/build.prop
    - Edit this file and set ro.kernel.android.checkjni=0
    - adb push build.prop /system/build.prop
  - Method 2
    - Change main.mk in build/core: ro.kernel.android.checkjni=1 change to 0
    - Also touch build/core/main.mk and build/tools/buildinfo.sh
- Disable JIT:
  - Pull /system/build.prop
  - Add this line to build.prop
  - dalvik.vm.execution-mode=int:fast
  - Push back build.prop and restart the board.

References

Ref.	Document	
Qualcomm Technologies		
Q1	*Android User Space Debugging*	80-NC051-1
Q2	*QRD7x27A AND 8x25 STABILITY DEBUG GUIDE*	80-NE759-1
Q3	*Android Memory Leak Analysis Guide*	80-NJ221-1
Resources		
R1	http://developer.android.com/training/articles/perf-anr.html	Keeping Your App Responsive
R2	http://developer.android.com/tools/help/monkey.html	UI/Application Exerciser Monkey
R3	http://developer.android.com/tools/help/adb.html	Android Debug Bridge
R4	http://developer.android.com/tools/debugging/systrace.html	Analyzing Display and Performance with Systrace
R5	http://developer.android.com/tools/help/systrace.html	Systrace
R6	http://www.kandroid.org/online-pdk/guide/debugging_native.html	Debugging Native Code
R7	http://qwiki.qualcomm.com/quic/Android_Framework_Reboot_Debugging	Android Framework Reboot Debugging
R8	http://qwiki.qualcomm.com/quic/JTAG_Debugging_On_Live_Target	TAG Debugging On Live Target

Questions?

https://support.cdmatech.com