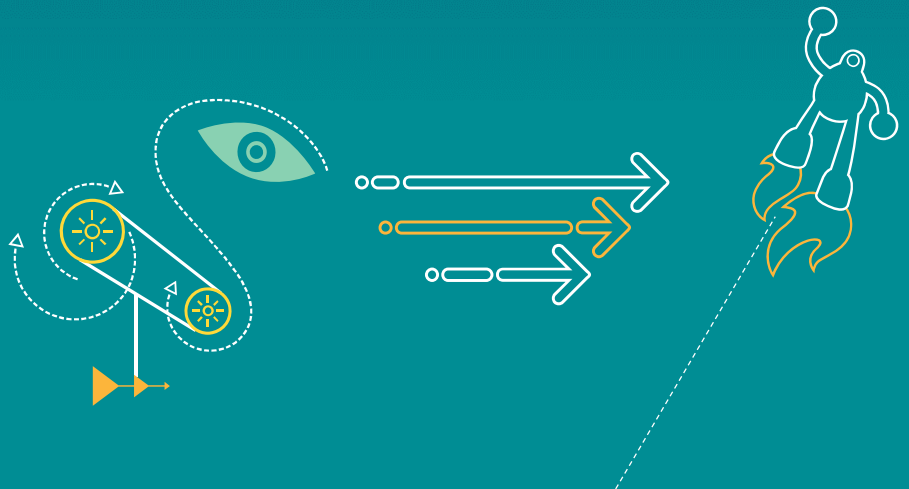
Introduction to Linux Kernel Debug Feature

Qualcomm Technologies, Inc.



80-P7139-4 A

Confidential and Proprietary – Qualcomm Technologies, Inc.

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

Revision History

Revision	Date	Description
A	June 2016	Initial release

Contents

Introduction

- This document is an overview of the Linux kernel debug feature for stability issues, and is intended for engineers that work in kernel driver development/integration or system crash/hang debug issues.

Kernel Debug Feature Overview

- Kconfig
  - arch/lib/Kconfig.debug
  - arch/lib/Kconfig.kgdb
  - arch/arm64/Kconfig.debug

- Menuconfig
  - make -C kernel O=../out/target/product/msm8996/obj/KERNEL_OBJ ARCH=arm64 CROSS_COMPILE=aarch64-linux-android- KCFLAGS=-mno-android menuconfig
  - Kernel hacking

- CONFIG_DEBUG_FS
- CONFIG_FRAME_POINTER

Kernel Debug Info

- CONFIG_DEBUG_INFO
  - gcc -g
- CONFIG_DEBUG_INFO_REDUCED
  - No structure debug info, but have line number
- CONFIG_DEBUG_INFO_SPLIT
- CONFIG_FRME_WARN
  - Warn if stack frame larger during build time; default 2048
- CONFIG_READABLE_ASM
  - Build kernel assembler code is more readable by disabling some optimizations

CONFIG_MAGIC_SYSRQ

/proc/sysrq-trigger

- c - Performs a system crash by a NULL pointer de-reference.
  A crashdump will be taken if configured.
- w - Dumps tasks that are in uninterruptable (blocked) state.
  6 normal (mpm, mdss, 2 audio, msm-core, mmc)
- l - Shows a stack backtrace for all active CPUs.
- m - Dumps current memory info to your console.
- q - Dumps per CPU lists of all armed hrtimers (but NOT regular timer_list timers) and detailed information about all clockevent devices.
- t - Dumps a list of current tasks and their information to your console.

Printk

- CONFIG_LOG_BUF_MAGIC
  - Add magic code to log buf record, to get from incomplete/corrupted dump
- CONFIG_DYNAMIC_DEBUG
  - Control print on/off dynamically

    pr_debug() dev_dbg()

  cat /sys/kernel/debug/dynamic_debug/control

  arch/arm64/kernel/traps.c:140 [traps]dump_backtrace =_ "%s(regs = %p tsk = %p)\0

  12"

  echo –n ‘func svc_process +p’ >dynamic_debug/control

  echo –n ‘file svcsock.c +p’

  echo -n 'module nfsd +p‘
- Log buffer
  - Kernel command line log_buf_len=1M 128K
  - memblock_virt_alloc_nopanic
- CONFIG_EARLY_PRINTK
  - early_printk

    CONFIG_EARLY_PRINTK_DIRECT=y

    CONFIG_DEBUG_LL=y
- Other API

  printk_ratelimited

  pr_devel   depend DEBUG

  pr_debug_once

  Pr_devel_once

Pstore

- Use to record kernel message of last crash
- CONFIG_PSTORE
  - CONFIG_PSTORE_CONSOLE all kernel message
  - PSTORE_PMSG usespace message /dev/pmsg0 /sys/fs/pstore/pmsg-ramoops-<ID>
  - CONFIG_PSTORE_RAM panic/oops message
  - CONFIG_PSTORE_FTRACE ftrace message

  - CONFIG_STRICT_MEMORY_RWX

- Need to reserver buffer

```
+       pstore_reserve_mem: pstore_reserve_mem_region@0 {
+               linux,reserve-contiguous-region;
+               linux,reserve-region;
+               linux,remove-completely;
+               reg = <0x0 0x9ff00000 0x0 0x00100000>;
+               label = "pstore_reserve_mem";
+       };
```

  - Solution 00028866 -- How to enable last kernel message - pstore on kernel 3.10

CONFIG_HAVE_HW_BREAKPOINT

- Use ARM hardware breakpoint register; can set in read-only segment
- A sample module to demo how to set
  Kernel/samples/hw_breakpoint/
  obj/KERNEL_OBJ/samples/hw_breakpoint/data_breakpoint.ko

- Demo code

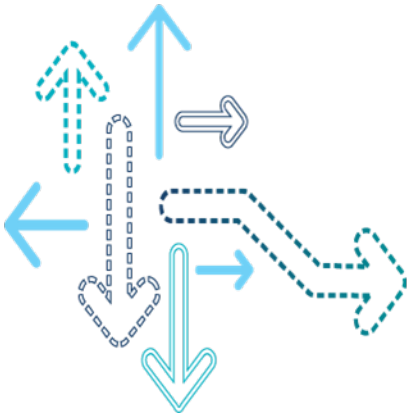
```
attr.bp_addr = kallsyms_lookup_name(ksym_name);
attr.bp_len = HW_BREAKPOINT_LEN_4;
attr.bp_type = HW_BREAKPOINT_W | HW_BREAKPOINT_R;
sample_hbp = register_wide_hw_breakpoint(&attr, sample_hbp_handler, NULL);
```

- Handler

```
static void sample_hbp_handler(struct perf_event *bp,
                               struct perf_sample_data *data,
                               struct pt_regs *regs)
{
    printk(KERN_INFO "%s value is changed\n", ksym_name);
    dump_stack();
    printk(KERN_INFO "Dump stack from sample_hbp_handler\n");
}
```

  - You can modify handler to dump_stack or panic or print other useful information
  - Solution 31443 -- How to enable Linux kernel hardware breakpoint on ARM/ARM64

Kernel Memory Debug



CONFIG_DEBUG_PAGEALLOC

This config can debug the software memory corruption

- When this debug config is set, kernel will generate an exception when read/write an un-allocated page, to catch code which corrupts memory
- The principle is that to poison/unpoison a page (set poison magic number and mark read-only) when allocate/free page; it is always used together with CONFIG_PAGE_POISONING and CONFIG_FREE_PAGES_RDONLY
- Parameters: “debug_guardpage_minorder=” : Kernel is trying to maximize memory usage, so there are usually not many free pages in the system and buggy code can corrupt some crucial data. This parameter allows control order of pages that will be intentionally kept free (therefore protected) by buddy allocator. Bigger value increases the probability of catching random memory corruption, but reduces amount of memory for normal system use. Maximum possible value is MAX_ORDER/2. Setting this parameter to 1 or 2 should be enough to identify most random memory corruption problems caused by bugs in kernel/drivers code when CPU writes to (or reads from) random memory location.

CONFIG_DEBUG_PAGEALLOC (cont.)

- Unmap pages from the kernel linear mapping after free_pages().
  - This results in a large slowdown, but helps to find certain types of memory corruption.
- CONFIG_PAGE_GUARD
- For architectures which don't enable ARCH_SUPPORTS_DEBUG_PAGEALLOC, fill the pages with poison patterns after free_pages() and verify the patterns before alloc_pages().
- CONFIG_PAGE_POISONING
- CONFIG_PAGE_GUARD
- Limitations:
  - Cannot catch the corruption caused by hardware or firmware bugs or by drivers badly programming DMA (when memory is written at bus level and CPU MMU is bypassed)
  - Downgrades performance

Map Read Only

- CONFIG_FREE_PAGES_RDOLY
  - Pages are always mapped in the kernel, i.e., anyone can write to the page if they have the address. Enable this option to mark pages as read-only to trigger a fault if any code attempts to write to a page on the buddy list. This may have a performance impact.

- CONFIG_KERNEL_TEXT_RDONLY
  - Make kernel code area read only

- CONFIG_DEBUG_RODATA
  - Kernel code and rodata will be read only

CONFIG_PAGE_OWNER

- This config can debug the page allocation stack trace
- Keeps track of what call chain is the owner of a page; may help to find bare alloc_page(s) leaks; uses a fair amount of memory
- See Documentation/page_owner.c for user-space helper
- When this debug config is set, the struct page will add several data structures:

```
#ifdef CONFIG_PAGE_OWNER
    int order;
    gfp_t gfp_mask;
    struct stack_trace trace;
    unsigned long trace_entries[8];
#endif
```

- This structure will be updated once allocation pages.
- It also can be read from debugfs page_owner entry.

Page_alloc.c

__alloc_pages_nodemask()->

set_page_owner():

```
#ifdef CONFIG_PAGE_OWNER
    struct stack_trace *trace = &page->trace;
    trace->nr_entries = 0;
    trace->max_entries = ARRAY_SIZE(page->trace_entries);
    trace->entries = &page->trace_entries[0];
    trace->skip = 3;
    save_stack_trace(&page->trace);

    page->order = (int) order;
    page->gfp_mask = gfp_mask;
#endif /* CONFIG_PAGE_OWNER */
```

CONFIG_PAGE_OWNER (cont.)

- Example
  - Parse dump by ‘Linux Ramdump Parser’

    Page structure addr: 0xcf290ee0L,

    page buff: phy:0x8207d000L, vir: 0xc207d000L

    flags 0x0, count 0x0

```
order = 0x0,
gfp_mask = 0x00211220,
trace = (
  nr_entries = 0x8,
  max_entries = 0x8,
  entries = 0xCF290F20,
  skip = 0x3),
trace_entries = (
  0xC00DEFC0,
  0xC00DA9CC,
  0xC0617DA8,
  0xC039FD98,
  0xC03A0584,
  0xC03BA75C,
  0xC004C810,
  0xC004C9F4))
```

```
('handle_irq_event', 60L)->
('handle_irq_event_percpu', 164L)->
('msm_udc_irq', 2032L)->
('rx_complete', 344L)->
('rx_submit', 240L)->
('__alloc_skb', 64L)->
('kmem_cache_alloc', 344L)->
('create_object', 40L)
```

CONFIG_SLUB_DEBUG

- This config can debug slub allocation issues
- When this debug config is set, every slub object will add some extra information for debugging

```
##ifdef CONFIG_SLUB_DEBUG
    atomic_long_t nr_slabs;
    atomic_long_t total_objects;
    struct list_head full;
#endif
```

- It can be checked from sysfs or dump

- CONFIG_SLUB_DEBUG_ON is another debug config which by default enabled the slub debug; it is equivalent to pass “slub_debug=”in commandline.

CONFIG_DEBUG_KMEMLEAK

/sys/kernel/debug/kmemleak

- CONFIG_DEBUG_KMEMLEAK_EARLY_LOG_SIZE
- CONFIG_DEBUG_KMEMLEAK_TEST
- CONFIG_DEBUG_KMEMLEAK_DEFAULT_OFF
- DEBUG_TASK_STACK_SCAN_OFF

Enable and Use Kmemleak

- Kernel configure
  - **CONFIG_DEBUG_KMEMLEAK**
  - Scans the memory manually or uses kthread to do it automatically; prints the number of new unreferenced objects found
  - Command line:

    kmemleak=off or kmemleak=on
- Debugfs
  - To trigger an intermediate memory scan

    echo scan > /sys/kernel/debug/kmemleak
  - To display the details of all possible memory leaks

    Cat /sys/kernel/debug/kmemleak
  - To clear the list of all current possible memory leaks

    echo clear > /sys/kernel/debug/kmemleak

CONFIG_DEBUG_KMEMLEAK

- Parameters written to /d/kmemleak

```
off                 - disable kmemleak (irreversible)
stack=on            - enable the task stacks scanning (default)
stack=off           - disable the tasks stacks scanning
scan=on             - start the automatic memory scanning thread (default)
scan=off            - stop the automatic memory scanning thread
scan=<secs>         - set the automatic memory scanning period in seconds
                      (default 600, 0 to stop the automatic scanning)
scan                - trigger a memory scan
clear               - clear list of current memory leak suspects, done by
                      marking all current reported unreferenced objects grey,
                      or free all kmemleak objects if kmemleak has been disabled.
dump=<addr>         - dump information about the object found at <addr>
```

Kmemleak Algorithm

- Can be tracked
  - Kmalloc, vmalloc, kmem_cache_alloc, …
- Can not be tracked
  - Ioremap, page allocations
  - How it is tracked
    - Pointers, size, stack trace, … saved in a rbtree
    - Freeing function calls are monitored ,so that to remove them from rbtree
  - How memory block is identified as orphan
    - Set all objects as white guys
    - Memory scan (data section and stack), to find any pointers to the start, or any location inside the memory block; if Yes, change the objects to be grey
    - The white objects left are considered as orphans

Kmemleak False Positive

- Wrongly reported as a leak
- How it happens
  - The allocated block itself cannot to be freed, and pointers are calculated by something like container_of
  - Temporarily stored in CPU registers or stacks

    Use MSECS_MIN_AGE to defer reporting

- APIs to avoid
  - kmemleak_not_leak(), kmemleak_ignore()
  - Use MSECS_MIN_AGE to defer reporting

Kmemleak False Negative

- Real leak, but not found
  - Stale information wrongly considered as the object pointer
  - Non-pointer variable, considered as pointer
    - Kernel would avoid this
- How to avoid
  - Task stack not scanned by default
  - APIs like kmemleak_scan_area /kmemleak_erase() to avoid
    - Add scan area inside a block
    - Erase an old value in a pointer variable
  - False negative eventually becomes positive, after stale information being replaced/erased if system keep running

Kmemleak – How to Analyze Logs

unreferenced object 0xffffffc05b48d130 (size 72):
  comm "init", pid 1, jiffies 4297168232 (age 80.080s)
- hex dump (first 32 bytes):
    25 00 00 00 25 00 00 00 06 00 00 00 57 02 04 00 %...%.......W...
    00 00 00 00 ff ff ff ff 01 00 00 00 00 00 00 00 ....
  backtrace:
    [<ffffffc00019d500>] create_object+0x140/0x274
    [<ffffffc000cf70a8>] kmemleak_alloc+0x7c/0xb4
    [<ffffffc000198a6c>] kmem_cache_alloc+0x170/0x220
    [<ffffffc0002dd3cc>] avc_alloc_node+0x2c/0x1f4
    [<ffffffc0002dd794>] avc_compute_av+0xb8/0x234
    [<ffffffc0002de274>] avc_has_perm_noaudit+0x64/0xd4
    [<ffffffc0002e198c>] selinux_inode_permission+0xc0/0x150
    [<ffffffc0002dba70>] security_inode_permission+0x20/0x34
    [<ffffffc0001ac5d8>] __inode_permission+0x84/0x98
    [<ffffffc0001ac62c>] inode_permission+0x40/0x4c
    [<ffffffc0001acde4>] may_open+0x94/0xec
    [<ffffffc0001af7e8>] do_last.isra.39+0x78c/0x9e4
    [<ffffffc0001afc44>] path_openat+0x204/0x590
    [<ffffffc0001b0cc0>] do_filp_open+0x2c/0x80
    [<ffffffc0001a26ac>] do_sys_open+0x160/0x1fc
    [<ffffffc0001a277c>] SyS_openat+0xc/0x18

Collect Kmemleak Log

- Use the following scripts to collect kmemleak log every 10 seconds:

```
#!/bin/sh
cat /proc/kmsg >/data/kmsg.txt &
num=1
while [ "$num" -ge 0 ]; do
echo ------------------------------
echo Cycle $num:
echo ------------------------------
echo clear > /sys/kernel/debug/kmemleak
cat /proc/meminfo
cat /proc/pagetypeinfo
cat /proc/slabinfo
top -n 1
free -m
cat /proc/zoneinfo
cat /proc/vmallocinfo
cat vmstat
sleep 10
echo scan > /sys/kernel/debug/kmemleak
cat /sys/kernel/debug/kmemleak > /data/$num.txt
let num=num+1
done
```

CONFIG_DEBUG_PER_CPU_MAPS

- This config is used to verify that the per_cpu map being accessed has been set up
  - Relate code:

```
static inline unsigned int cpumask_check(unsigned int cpu)
{
#ifdef CONFIG_DEBUG_PER_CPU_MAPS
        WARN_ON_ONCE(cpu >= nr_cpumask_bits);
#endif /* CONFIG_DEBUG_PER_CPU_MAPS */
        return cpu;
}
```


Impact:

- The following two CONFIGS will be force set

  CONFIG_CPUMASK_OFFSTACK cpumask_var_t

  CONFIG_DISABLE_OBSOLETE_CPUMASK_FUNCTIONS
- Adds a fair amount of code to kernel memory and decreases performance

CONFIG_DEBUG_STACK_USAGE

- This config is enabled to show the minimum free stack that each task ever had in the sysrq-T and sysrq-P debug output
- Usage: echo t > proc/sysrq-trigger

  Get the message in kernel log; the field in red is the stack information

```
<6>[ 5446.999722]  task                PC                stack   pid    father
<6>[ 5446.999735] init            S    ffffffc00008671c  9432    1      0 0x00000000
SNIP
<6>[ 5447.010547] rcu_preempt S    ffffffc00008671c  10776   7      2 0x00000000
SNIP
<6>[ 5447.034436] watchdog/2  S    ffffffc00008671c  13032   17     2 0x00000000
```

- Impact:
  - This option will slow down process creation somewhat

CC_STACKPROTECTOR

- Dependency
  - CONFIG_HAVE_CC_STACKPROTECTOR
  - CONFIG_CC_STACKPROTECTOR
  - ARM arch support this
- How it works
  - This option turns on the "stack-protector" GCC feature.
  - At the beginning of functions, a canary value put on the stack just before the return address validates the value just before actually returning.
  - Stack based buffer overflows (that need to overwrite this return address) now also overwrite the canary, trigger panic in the validation
- Choice
  - Menuconfig General Setup
  - CC_STACKPROTECTOR_NONE
    - Not enable gcc -fstack-protector feature
  - CC_STACKPROTECTOR_REGULAR
    - Functions will have the stack-protector canary logic added if they have an 8-byte or larger character array on the stack.
  - CC_STACKPROTECTOR_STRONG
    - Functions will have the stack-protector canary logic added in any of the following conditions:
      - local variable address used as part of the right hand side of an assignment or function argument
      - local variable is an array (or union containing an array), regardless of array type or length
      - uses register local variables

CC_STACKPROTECTOR Example

- Dump of assembler code for function workshop_stub2:

```
0xffffffc00043aa90 <+0>:    stp    x29, x30, [sp,#-128]!
0xffffffc00043aa94 <+4>:    mov    x1, x0
0xffffffc00043aa98 <+8>:    mov    x29, sp
0xffffffc00043aa9c <+12>:   str    x19, [sp,#16]
0xffffffc00043aaa0 <+16>:   adrp   x19, 0xffffffc0016ed000 <reset_devices>
0xffffffc00043aaa4 <+20>:   ldr    x0, [x19,#208]
0xffffffc00043aaa8 <+24>:   str    x0, [x29,#120]
0xffffffc00043aaac <+28>:   add    x0, x29, #0x28
0xffffffc00043aab0 <+32>:   bl     0xffffffc000318e7c <strcpy>
0xffffffc00043aab4 <+36>:   adrp   x0, 0xffffffc0011a6000
0xffffffc00043aab8 <+40>:   add    x1, x29, #0x28
0xffffffc00043aabc <+44>:   add    x0, x0, #0xfb5
0xffffffc00043aac0 <+48>:   bl     0xffffffc000cc3fbc <printk>
0xffffffc00043aac4 <+52>:   ldr    x1, [x29,#120]
0xffffffc00043aac8 <+56>:   ldr    x0, [x19,#208]
0xffffffc00043aacc <+60>:   cmp    x1, x0
0xffffffc00043aad0 <+64>:   b.eq   0xffffffc00043aad8 <workshop_stub2+72>
0xffffffc00043aad4 <+68>:   bl     0xffffffc0000a308c <__stack_chk_fail>
0xffffffc00043aad8 <+72>:   ldr    x19, [sp,#16]
0xffffffc00043aadc <+76>:   ldp    x29, x30, [sp],#128
0xffffffc00043aae0 <+80>:   ret
```

CONFIG_PANIC_ON_DATA_CORRUPTION

- This config is adebug feature which can detect the data corruption in time
- Usage: When this debug configure is set, kernel will trigger panic when detect data corruption.
- Principal: Detect the data corruption in earlier in order to closer the corruption scenarios for debug.
- Scenarios
  - The double linked list destroy
  - Workqueue leaked lock or atomic when scheduled
  - Spinlock wrong data
- How to detect
  - List add:

    prev->next != next || next->prev != prev || new == prev || new == next
    list del: entry->next==LIST_POISON1; entry->next== LIST_POISON2; prev->next != entry; next->prev != entry
    preempt_count() & ~PREEMPT_ACTIVE) != 0 || (tsk)->lockdep_depth > 0
    (spinlock->magic != SPINLOCK_MAGIC
- Reasons
  - bitflip issue
  - overwrite [much data corruption]
- Example: SR#02446787
  - (struct list_head) [D:0xC1D36E74] event_entry = (
  - (struct list_head *) [D:0xC1D36E74] next = 0xC5856400,
  - (struct list_head *) [D:0xC1D36E78] prev = 0xC5857C00 -> (
  - (struct list_head *) [D:0xC5857C00] next = 0xC5857C00,
  - (struct list_head *) [D:0xC5857C04] prev = 0xC5857C00)),

CONFIG_PANIC_ON_DATA_CORRUPTION (cont.)

- This config is a debug feature which can detect data corruption
- How to detect:

  (preempt_count() & ~PREEMPT_ACTIVE) != 0 || (tsk)->lockdep_depth > 0

  spinlock->magic != SPINLOCK_MAGIC
- The reasons:
  - bitflip issue
  - overwrite [a lot of data corruption]
- Example: SR#02446787

  (struct list_head) [D:0xC1D36E74] event_entry = (

  (struct list_head *) [D:0xC1D36E74] next = 0xC5856400,

  (struct list_head *) [D:0xC1D36E78] prev = 0xC5857C00 -> (

  (struct list_head *) [D:0xC5857C00] next = 0xC5857C00,

  (struct list_head *) [D:0xC5857C04] prev = 0xC5857C00)),

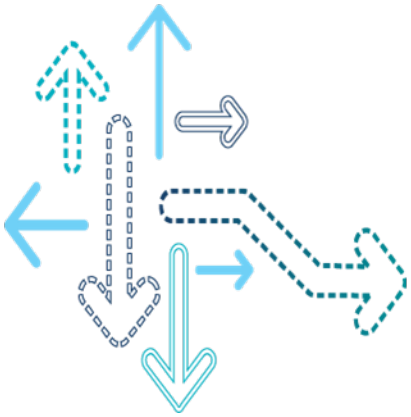
CONFIG_ARM64_PTDUMP

- It creates kernel_page_tables under /sys/kernel/debug. CONFIG_DEBUG_FS must be enabled

- Reference
  - http://www.spinics.net/lists/arm-kernel/msg499465.html

- For debugging purposes, it would be nice if we could export page tables other than the swapper_pg_dir to userspace. To enable this, this patch refactors the arm64 page table dumping code such that multiple tables may be registered with the framework, and exported under debugfs.

Lock Debug Feature



CONFIG_LOCKUP_DETECTOR

1. Detect hard and soft lockups, it will create thread watchdog/%u for each core
2. Echo a zero to /proc/sys/kernel/watchdog to disable the watchdog timer.
3. Echo a large number of /proc/sys/kernel/watchdog_thresh in order to reduce the frequency of OS jitter due to the watchdog timer down to a level that is acceptable for your workload.

- The priority of watchdog/%u is (MAX_RT_PRIO - 1);
- The watchdog task is a high priority kernel thread that updates a timestamp every time it is scheduled. If that timestamp is not updated for 2*watchdog_thresh seconds (the softlockup threshold) the 'softlockup detector' (coded inside the hrtimer callback function) will dump useful debug information to the system log, after which it will call panic if it was instructed to do so or resume execution of other kernel code.

CONFIG_BOOTPARAM_SOFTLOCKUP_PANIC_VALUE=1

CONFIG_LOCKUP_DETECTOR (cont.)

- HARDLOCKUP and SOFTLOCKUP

A 'softlockup' is defined as a bug that causes the kernel to loop in kernel mode for more than x defined seconds without giving other tasks a chance to run.

A 'hardlockup' is defined as a bug that causes the CPU to loop in kernel mode for more than defined seconds, without letting other interrupts have a chance to run. It uses a periodic NMI in order to detect if the system has become unresponsive. It is used to detect any kind of fault that can causes interrupt handling to fail. Examples include badly matched disables, spurious interrupts, and live locks inside critical sections. (Qualcomm does not use HARDLOCK NMI)

The soft and hard lockup detectors are built on top of the hrtimer and perf subsystems, respectively. A periodic hrtimer runs to generate interrupts and kick the watchdog task.

If any CPU in the system does not receive any hrtimer interrupt during that time, the 'hardlockup detector will generate a kernel warning or call panic, depending on the configuration.

CONFIG_LOCKUP_DETECTOR (cont.)

- How SOFTLOCKUP works

Process Watchdog/%u will touch watchdog_touch_ts to get_timestamp by __touch_watchdog, and hrtimer will check if it has softlock when now is after touch_ts + get_softlockup_thresh()

```
static void watchdog(unsigned int cpu)
{
    __this_cpu_write(soft_lockup_hrtimer_cnt,
                     __this_cpu_read(hrtimer_interrupts));
    __touch_watchdog();
}
```

```
/* Commands for resetting the watchdog */
static void __touch_watchdog(void)
{
    __this_cpu_write(watchdog_touch_ts, get_timestamp());
}
```

judge if is softlock by checking watchdog_touch_ts

```
static int is_softlockup(unsigned long touch_ts)
{
    unsigned long now = get_timestamp();

    /* Warn about unreasonable delays: */
    if (time_after(now, touch_ts + get_softlockup_thresh()))
        return now - touch_ts;

    return 0;
}
```

```
static int get_softlockup_thresh(void)
{
    return watchdog_thresh * 2;
}
```

CONFIG_LOCKUP_DETECTOR (cont.)

How HARDLOCKUP_DETECTOR_OTHER_CPU works

```
static void watchdog_enable(unsigned int cpu)
{
    struct hrtimer *hrtimer = &__raw_get_cpu_var(watchdog_hrtimer);

    /* kick off the timer for the hardlockup detector */
    hrtimer_init(hrtimer, CLOCK_MONOTONIC, HRTIMER_MODE_REL);
    hrtimer->function = watchdog_timer_fn;
```

```
static enum hrtimer_restart watchdog_timer_fn(struct hrtimer *hrtimer)
{
    unsigned long touch_ts = __this_cpu_read(watchdog_touch_ts);
    struct pt_regs *regs = get_irq_regs();
    int duration;

    /* kick the hardlockup detector */
    watchdog_interrupt_count();

    /* test for hardlockups on the next cpu */
    watchdog_check_hardlockup_other_cpu();

    /* kick the softlockup detector */
    wake_up_process(__this_cpu_read(softlockup_watchdog));
```

```
static void watchdog_check_hardlockup_other_cpu(void)
{
    unsigned int next_cpu;

    /*
     * Test for hardlockups every 3 samples.  The sample period is
     *  watchdog_thresh * 2 / 5, so 3 samples gets us back to slightly over
     *  watchdog_thresh (over by 20%).
     */
    if (__this_cpu_read(hrtimer_interrupts) % 3 != 0)
        return;

    /* check for a hardlockup on the next cpu */
    next_cpu = watchdog_next_cpu(smp_processor_id());
    if (next_cpu >= nr_cpu_ids)
        return;

    smp_rmb();

    if (per_cpu(watchdog_nmi_touch, next_cpu) == true) {
        per_cpu(watchdog_nmi_touch, next_cpu) = false;
        return;
    }

    if (is_hardlockup_other_cpu(next_cpu)) {
```

A 'hardlockup' is defined as a bug that causes the CPU to loop in kernel mode for more than 10 seconds, default value is 12 seconds ((watchdog_thresh * 2/5) *3).

Hrtimer will call watchdog_timer_fn every watchdog_thresh * 2/5 (default 4) seconds to increase hrtimer_interrupts

```
static void watchdog_interrupt_count(void)
{
    __this_cpu_inc(hrtimer_interrupts);
}
```

In watchdog_check_hardlockup_other_cpu, only when __this_cpu_read(hrtimer_interrupts) % 3 == 0, it will check if it has hardlock

```
static int is_hardlockup_other_cpu(unsigned int cpu)
{
    unsigned long hrint = per_cpu(hrtimer_interrupts, cpu);

    if (per_cpu(hrtimer_interrupts_saved, cpu) == hrint)
        return 1;

    per_cpu(hrtimer_interrupts_saved, cpu) = hrint;
    return 0;
}
```

CONFIG_DETECT_HUNG_TASK

- Kernel configure
  - CONFIG_DETECT_HUNG_TASK
  - CONFIG_BOOTPARAM_HUNG_TASK_PANIC_VALUE
  - CONFIG_DEFAULT_HUNG_TASK_TIMEOUT
  - When a task keeps in D state for longer than certain threshold, one kernel daemon will detect that, and print the current stack trace
  - It is with negligible overhead, and configurable whether to trigger panic when finding hung task
- procfs
  - echo 0 > /proc/sys/kernel/hung_task_timeout_secs
  - By-default setting is 120 seconds.
  - There is a daemon named khungtaskd, which wakes up periodically and checks every thread's context switch count, which is recorded in the task structure

Detect Hung Tasks – Log Example

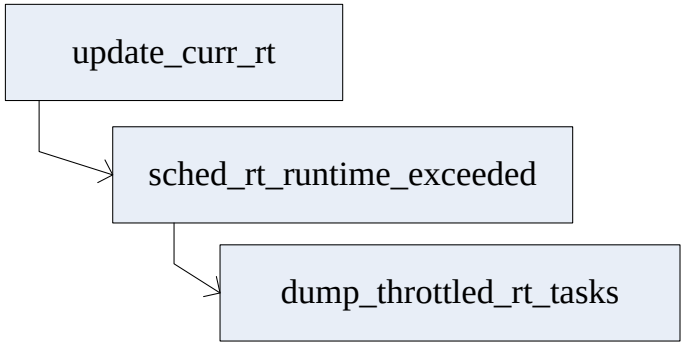
- task PonMgr:510 blocked for more than 120 seconds.

```
PonMgr          D c03e08cc     0   510    509 0x00000000
Backtrace:
[<c03e0514>] (__schedule+0x0/0x494) from [<c03e0ad0>] (schedule+0x84/0x8c)
[<c03e0a4c>] (schedule+0x0/0x8c) from [<c03dee44>] (schedule_timeout+0x20/0x1e4)
[<c03dee24>] (schedule_timeout+0x0/0x1e4) from [<c03dfe98>] (__down+0x80/0xb4)
[<c03dfe18>] (__down+0x0/0xb4) from [<c012e190>] (down+0x34/0x48)
[<c012e15c>] (down+0x0/0x48) from [<bf0229a8>] (gpon_evt_read+0x1c/0xc8 [gpon_drv])
[<bf02298c>] (gpon_evt_read+0x0/0xc8 [gpon_drv]) from [<c0195e80>] (vfs_read+0xb8/0x134)
[<c0195dc8>] (vfs_read+0x0/0x134) from [<c0195f40>] (sys_read+0x44/0x70)
[<c0195efc>] (sys_read+0x0/0x70) from [<c000d7a0>] (ret_fast_syscall+0x0/0x30)
```

http://m.blog.chinaunix.net/uid-25564582-id-5204177.html
http://www.cnblogs.com/openix/p/4136274.html
http://blog.csdn.net/wh_19910525/article/details/50503269

CONFIG_PANIC_ON_RT_THROTTLING

- RT Throttling
  - If an RT task runs into unexpected bugs, without RT throttling other non-RT tasks will not be able to run, and the system will hang; for example, for 1 second, the RT is granted to hog CPU for 0.95 seconds
- Kernel configure
  - CONFIG_PANIC_ON_RT_THROTTLING
    - When detect RT task run out of its time slices, trigger panic
- procfs
  - /proc/sys/kernel/sched_rt_period_us
    /proc/sys/kernel/sched_rt_runtime_us
  - http://book.2cto.com/201302/16291.html
  - https://lwn.net/Articles/296419/
  - http://www.oenhan.com/task-group-sched



CONFIG_PANIC_ON_RT_THROTTLING (cont.)

```
[ 59.437232] sched: RT throttling activated for rt_rq
ffffffc0d822e000 (cpu 4)
[ 59.437232] potential CPU hogs:
[ 59.437232] ts-kthread (33)
[ 59.450756] ------------[ cut here ]------------
[ 59.455353] Kernel BUG at ffffffc000b03610
[verbose debug info unavailable]
[ 59.462297] Internal error: Oops - BUG: 0 [#1]
PREEMPT SMP
[ 59.467764] Modules linked in: core_ctl(PO)
wlan(O)
[ 59.472632] CPU: 4 PID: 33 Comm: ts-kthread
Tainted: P O 3.10.49-perf-ga375302-dirty #2
[ 59.481918] task: ffffffc0e1bb1f80 ti:
ffffffc0e1bc8000 task.ti: ffffffc0e1bc8000
[ 59.489387] PC is at
dump_throttled_rt_tasks+0x64/0x134
[ 59.494588] LR is at
dump_throttled_rt_tasks+0x64/0x134
.....
[ 59.945111] []
dump_throttled_rt_tasks+0x64/0x134
[ 59.951366] [] update_curr_rt+0x184/0x1f0
[ 59.956918] [] task_tick_rt+0x10/0xec
[ 59.962127] [] scheduler_tick+0x1c8/0x24c
[ 59.967682] [] update_process_times+0x50/0x6c
[ 59.973587] [] tick_sched_handle.isra.13+0x40/0x54
[ 59.979921] [] tick_sched_timer+0x7c/0x18c
[ 59.985562] [] __run_hrtimer+0x148/0x224
[ 59.991030] [] hrtimer_interrupt+0xd8/0x1fc
[ 59.996764] [] arch_timer_handler_virt+0x28/0x38
[ 60.002925] [] handle_percpu_devid_irq+0xd8/0x16c
[ 60.009175] [] generic_handle_irq+0x28/0x3c
[ 60.014905] [] handle_IRQ+0x7c/0xa0
[ 60.019936] [] gic_handle_irq+0x58/0xa4
.....
[ 60.105183] [] el1_irq+0x60/0xd0
[ 60.109956] [] printk+0x6c/0x78
[ 60.114645] [] ts_scan_switch+0x184/0x1a0
[ 60.120199] [] synaptics_kthread_dclick_proximity_switch+0xa4/0xb4
[ 60.127927] [] touchscreen_thread+0x198/0x214
[ 60.133826] [] kthread+0xac/0xb8
```

CONFIG_DEBUG_MUTEXES

- Usage for this config
  - Debug mutex bad magic issue
  - Debug mutex recursion issue
  - Mutex dead lock
  - Mutex waiterlist corruption
  - initialization

    lock->magic = lock;
- Mutex lock
  1. Set waiter as MUTEX_DEBUG_INIT; also initialization waiter‘s magic as waiter itself
  2. Check if waiter list is NULL or not
  3. Check if current task is the task mutex wake up
  4. If no waiter is there, set waiter as MUTEX_DEBUG_FREE
  5. Set mutex owner as current task

CONFIG_DEBUG_MUTEXES (cont.)

- Mutex unlock

  debug_mutex_unlock will be used to unlock

- Check lock->magic equal to lock or not, not equal print debug message
  1. Check owner if task current or not
  2. Check waiter list if corruption or not
  3. Clear owner

CONFIG_DEBUG_SPINLOCK

Usage for this config

- Debug spinlock bad magic issue
- Debug spinlock recursion issue
- Spinlock dead lock

How it works

- Initialization

```
# define SPIN_DEBUG_INIT(lockname)                \
    .magic = SPINLOCK_MAGIC,                      \
    .owner_cpu = -1,                              \
    .owner = SPINLOCK_OWNER_INIT,
```

- Before trying to get this lock

  debug_spin_lock_before -- Try to check magic num and owner_cpu to see if it is bad magic or recursion issue; if so, all spin_dump

CONFIG_DEBUG_SPINLOCK (cont.)

- Lock operation
  - spin_lock_debug will call arch_spin_trylock loops_per_jiffy * HZ times, if still not get the spinlock, then it will call spin_dump to print all spinlock information (owner, magic_num), and call stacks on all cpu cores
- CONFIG_DEBUG_SPINLOCK_BITE_ON_BUG – trigger bite
- CONFIG_DEBUG_SPINLOCK_PANIC_ON_BUG – trigger panic
- Unlock operation
  - Check the following situations

```
SPIN_BUG_ON(lock->magic != SPINLOCK_MAGIC, lock, "bad magic");
SPIN_BUG_ON(!raw_spin_is_locked(lock), lock, "already unlocked");
SPIN_BUG_ON(lock->owner != current, lock, "wrong owner");
SPIN_BUG_ON(lock->owner_cpu != raw_smp_processor_id(),
                                        lock, "wrong CPU")
```

- If above issues, will call spin_dump to dump the spinlock and call stack information

CONFIG_DEBUG_LOCK_ALLOC

- This config checks the following:
  - Whether any held lock (spinlock, rwlock, mutex or rwsem) is incorrectly freed by the kernel, in memory-freeing routines (kfree(), kmem_cache_free(), free_pages(), vfree(), etc.)
  - Whether a live lock is incorrectly reinitialized via spin_lock_init()/mutex_init()
  - Whether there is any lock held during task exit
    - Example:

```
void __raw_spin_lock_init(raw_spinlock_t *lock, const char *name,
                          struct lock_class_key *key)
{
#ifdef CONFIG_DEBUG_LOCK_ALLOC
    /*
     * Make sure we are not reinitializing a held lock:
     */
    debug_check_no_locks_freed((void *)lock, sizeof(*lock));
    lockdep_init_map(&lock->dep_map, name, key, 0);
#endif
    lock->raw_lock = (arch_spinlock_t)__ARCH_SPIN_LOCK_UNLOCKED;
lock->magic = SPINLOCK_MAGIC;
    lock->owner = SPINLOCK_OWNER_INIT;
    lock->owner_cpu = -1;
}
```

CONFIG_DEBUG_ATOMIC_SLEEP

- This affects the definition of might_sleep(); when it's not defined, might_sleep, which does real work, will not be called
- A function that has sleep possibility and is NOT expected be called in atomic context, probably calls might_sleep to warn customer by log, or bug when it's called from atomic context

kernel/include/linux/kernel.h

```
#ifdef CONFIG_DEBUG_ATOMIC_SLEEP
    void __might_sleep(const char *file, int line, int preempt_offset);
   /**
    * might_sleep - annotation for functions that can sleep
    *
    * this macro will          print a stack trace if it is executed in an atomic
    * context (spinlock, irq-handler,  ...).
    *
    * This is a useful debugging help to be able to catch problems early and not
    * be bitten later when the calling function happens to sleep when it is not
    * supposed to. */
     # define might_sleep() \
              do { __might_sleep(__FILE__, __LINE__, 0); might_resched(); } while (0)
#else
     static inline void __might_sleep(const char *file, int line, int preempt_offset) { }
     # define might_sleep() do { might_resched(); } while (0)
#endif
```

- There is also might_sleep_if, which does the same thing with a condition

```
#define might_sleep_if(cond) do { if (cond) might_sleep(); } while (0)
```

CONFIG_DEBUG_ATOMIC_SLEEP (cont.)

The implementation of __might_sleep is located in kernel/kernel/sched/core.c and kernel/kernel/sched/qhmp_core.c

```
void __might_sleep(const char *file, int line, int preempt_offset)
{
    static unsigned long prev_jiffy;    /* ratelimiting */

    rcu_sleep_check(); /* WARN_ON_ONCE() by default, no rate limit reqd. */
    if ((preempt_count_equals(preempt_offset) && !irqs_disabled() &&
         !is_idle_task(current)) || oops_in_progress)
        return;
    if (system_state != SYSTEM_RUNNING &&
        (!__might_sleep_init_called || system_state != SYSTEM_BOOTING))
        return;
    if (time_before(jiffies, prev_jiffy + HZ) && prev_jiffy)
        return;
    prev_jiffy = jiffies;

    printk(KERN_ERR
        "BUG: sleeping function called from invalid context at %s:%d\n",
            file, line);
    printk(KERN_ERR
        "in_atomic(): %d, irqs_disabled(): %d, pid: %d, name: %s\n",
            in_atomic(), irqs_disabled(),
            current->pid, current->comm);

    debug_show_held_locks(current);
    if (irqs_disabled())
        print_irqtrace_events(current);
#ifdef CONFIG_DEBUG_PREEMPT
    if (!preempt_count_equals(preempt_offset)) {
        pr_err("Preemption disabled at:");
        print_ip_sym(current->preempt_disable_ip);
        pr_cont("\n");
    }
#endif
#ifdef CONFIG_PANIC_ON_SCHED_BUG
    BUG();
#endif
    dump_stack();
} ? end __might_sleep ?
```

Check if kernel's in atomic context, i.e., spinlock, irq context…

Those logs will be printed out if __might_sleep is called in atomic context

Kernel may crash, depending on this switch

CONFIG_DEBUG_ATOMIC_SLEEP (cont.)

- There are many examples available in CASE, for example*:* **02428387**
  - The following log will be printed out when the situation is captured by might_sleep

[1448.551880] BUG: sleeping function called from invalid context at ../../../../../../kernel/kernel/locking/mutex.c:97
[1448.551898] in_atomic(): 0, irqs_disabled(): 0, pid: 20242, name: essaging.vzmsgs
[1448.551905] Preemption disabled at:[<ffffffc000d9ae64>] printk+0x6c/0x78
[1448.551925]
[1448.551937] ------------[cut here]------------
[1448.551944] kernel BUG at ../../../../../../kernel/kernel/sched/core.c:10169!
[1448.551951] Internal error: Oops - BUG: 0 [#1] PREEMPT SMP
[1448.551957] Modules linked in: wlan(O) v4l2_hal(O) gb_vibrator(O) gb_vendor_moto(O) gb_usb_ext(O) gb_raw(O) gb_ptp(O) gb_phy(O) gb_mods(O) gb_loopback(O) gb_light(O) gb_hid(O) gb_display(O) gb_db3(O) gb_camera_ext(O) gb_camera(O) gb_battery(O) greybus(O) moto_crypto(O)
[1448.552041] CPU: 0 PID: 20242 Comm: essaging.vzmsgs Tainted: G W O 3.18.24-g82285f4-00012-g271f546 #1
[1448.552047] Hardware name: Sheridan (DT)
[1448.552054] task: ffffffc06c648c80 ti: ffffffc047350000 task.ti: ffffffc047350000
[1448.552066] PC is at __might_sleep+0x15c/0x16c
[1448.552073] LR is at __might_sleep+0x15c/0x16c
[1448.552080] pc : [<ffffffc0000c4fb4>] lr : [<ffffffc0000c4fb4>] pstate: 80000145
[1448.552085] sp : ffffffc047353d70
[1448.552090] x29: ffffffc047353d70 x28: ffffffc047350000
[1448.552102] x27: ffffffc09c09c780 x26: 0000000000000036
[1448.552114] x25: 00000000ffccf310 x24: ffffffc074b7ca00
[1448.552124] x23: ffffffc002173000 x22: 00000000c0306201
[1448.552135] x21: ffffffc001e5c000 x20: ffffffc001f2d000
[1448.552146] x19: 0000000000000000 x18: 0000000000000000
[1448.552156] x17: 0000000000000000 x16: ffffffc0001e9f94
[1448.552167] x15: 0000000000000000 x14: 0fffffffffffffe
[1448.552178] x13: 0000000000000000 x12: 0101010101010101
[1448.552188] x11: ffffffff7f7f7f7f x10: fefefebf463439ff
[1448.552199] x9 : 7f7f7f7f7f7f7f7f x8 : 5d302c3532393135
[1448.552209] x7 : 0000000000000000 x6 : ffffff8001c2cc53
[1448.552220] x5 : ffffff8001c00000 x4 : 0000000000000007
[1448.552230] x3 : 0000000000000007 x2 : 0000000000000000
[1448.552241] x1 : 0000000000400000 x0 : 0000000000000000

CONFIG_SCHED_STACK_END_CHECK

- When this debug config is set, the event of a stack overrun will be checked
- The principle is as follows:
  - STACK_END_MAGIC(0x57AC6E9D) will be set at the end of stack when create a task, :

```
void set_task_stack_end_magic(struct task_struct *tsk)
{
    unsigned long *stackend;

    stackend = end_of_stack(tsk);
    *stackend = STACK_END_MAGIC;     /* for overflow detection */
}
```

  - Then it checks when task is scheduling, the change of magic number representing stack overrun, and a kernel exception will be triggered

```
static inline void schedule_debug(struct task_struct *prev)
{
#ifdef CONFIG_SCHED_STACK_END_CHECK
    BUG_ON(unlikely(task_stack_end_corrupted(prev)));
#endif
```

```
#define task_stack_end_corrupted(task) \
        (*(end_of_stack(task)) != STACK_END_MAGIC)
```

CONFIG_DEBUG_LIST

- When this CONFIG_DEBUG_LIST is set, it turns on extended checks in the linked-list walking routines

1. Add manipulation checks list next, to see if prev is corrupted; also checks if new node is double add or not
2. Delete manipulation checks if list is corrupted; also checks if del mode is double delete by LIST_POISON1/LIST_POISON2
3. Open CONFIG_PANIC_ON_DATA_CORRUPTION, which gets ram dump; without this, only a warning message is printed out in kernel log

- When this CONFIG_DEBUG_PI_LIST is set, it turns on extended checks in the priority-ordered linked-list (plist) walking routines; this adds plist_check_head to check the entire list multiple times during each manipulation; a warning message will be printed out in kernel log if corruption is detected

CONFIG_TIMER_STATS

- This config collects information about the timer events which are fired in a Linux system over a sample period
  - Name of the process which initialized the timer
  - Function where the timer was initialized
  - PID of the task (process) which initialized the timer
  - Callback function which is associated to the timer
  - Number of events (callbacks)

  - To activate a sample period issue:

    # echo 1 >/proc/timer_stats
  - To stop a sample period issue:

    # echo 0 >/proc/timer_stats
  - The statistics can be retrieved by:

    # cat /proc/timer_stats

CONFIG_TIMER_STATS (cont.)

Timer Stats Version: v0.3 （Test Result on MTP8996）
Sample period: 8.703 s
Collection: inactive

```
   47,     0 swapper/4         hrtimer_start_range_ns (tick_sched_timer)
   34,     7 rcu_preempt       rcu_gp_kthread (process_timeout)
    9,  9693 appsearch_threa   hrtimer_start_range_ns (hrtimer_wakeup)
  14D,    29 ksoftirqd/4       add_timer (cpufreq_interactive_timer)
   33,     0 swapper/5         hrtimer_start_range_ns (tick_sched_timer)
....
    6,    11 watchdog/0        start_bandwidth_timer (sched_rt_period_timer)
....
    1,  3458 logd.reader.per   add_timer (cpufreq_interactive_nop_timer)
    1,  5836 Google Conversi   hrtimer_start_range_ns (hrtimer_wakeup)
   1D,  1473 Binder_2          add_timer (cpufreq_interactive_timer)
    1,   384 mmc-cmdqd/0       blk_add_timer (blk_rq_timed_out_timer)
    1,  3457 logcat            add_timer (cpufreq_interactive_nop_timer)
   1D,  2409 Binder_10         add_timer (cpufreq_interactive_timer)
597 total events, 68.597 events/sec
```

The first column is the number of events, the second column the pid, the third column is the name of the process. The forth column shows the function which initialized the timer and in parenthesis the callback function which was executed on expiry.

Added flag to indicate 'deferrable timer' in /proc/timer_stats. A deferrable timer will appear as follows:

```
   1D,  2409 Binder_10         add_timer (cpufreq_interactive_timer)
```

CONFIG_KGDB

1. Turn on KGDB
   - CONFIG_HAVE_ARCH_KGDB=y
   - CONFIG_KGDB=y
   - CONFIG_KGDB_SERIAL_CONSOLE=y
   - CONFIG_KGDB_KDB=y

2. Turn off CONFIG_MSM_WATCHDOG_V2
   - Otherwise, when kgdb breaks, dog will trigger reset

3. Add uart support for KGDB; there are 2 ways to do this:
   - Configure kgdboc at boot using kernel parameters:

     kgdboc=ttyS0,115200

   - Configure kgdboc after the kernel has booted:

     echo ttyS0 > /sys/module/kgdboc/parameters/kgdboc

     Stop kernel execution (break into the debugger)

4. Use gdb on a host PC, connect with uart

CONFIG_KGDB GDB

1. Connect with GDB

   To connect to gdb via kgdboc, the kernel must first be stopped. There are several ways to stop the kernel which include using kgdbwait as a boot argument, via a sysrq-g, or running the kernel until it takes an exception where it waits for the debugger to attach.

2. When logged in as root or with a super user session you can run:

   echo g > /proc/sysrq-trigger

   Example using minicom 2.2

   1. Press Ctrl+ A
   2. Press F
   3. Press G

3. When you have telneted to a terminal server that supports sending a remote break

   1. Press Ctrl+]
   2. Type “send break”
   3. Press Enter
   4. Press G

4. Connect from gdb

   Example (using a directly connected port):

   % aarch64-linux-android-gdb ./vmlinux

   (gdb) set remotebaud 115200

   (gdb) target remote /dev/ttyS0

5. Once connected, debug the kernel the way an application program is debugged

CONFIG_FTRACE

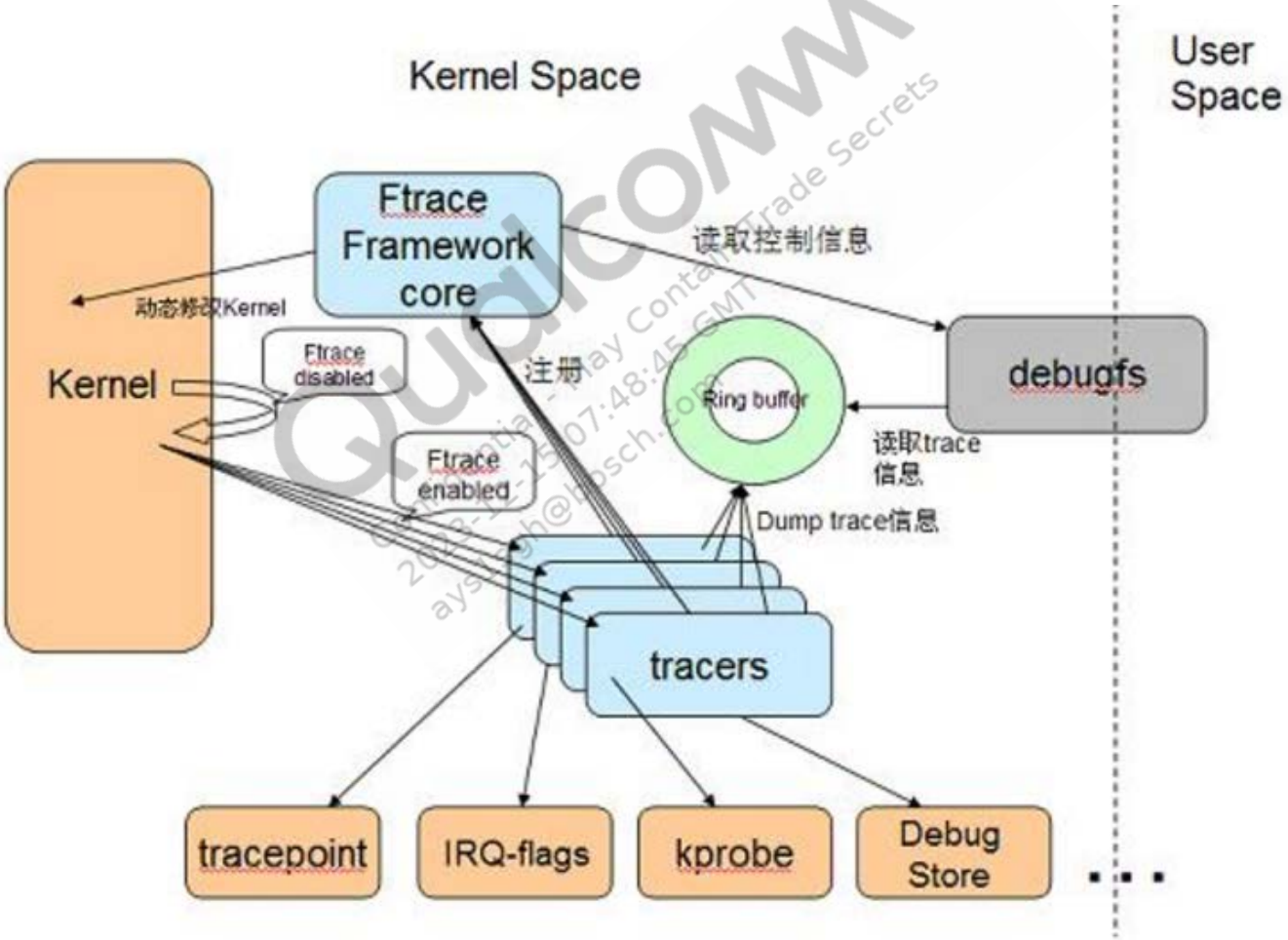
- Usage for ftrace
  - Stability issue debug
    - Events info gets more history info,like clock/regulator/cpufreq/HW controller(I2C/SPI/USB), more info than RTB; include the setting value
    - Enable related tracer for other stability issues such as ETM
  - Performance tuning
    - Scheduler tuning, latency, irq/wakeup
    - Android systrace
  - Power tuning
    - CPU usage /cluster LPM distribution/ bus frequency
    - DDR frequency/regulator

CONFIG_FTRACE (cont.)

- FTRACE configurations
  - CONFIG_TRACEPOINTS
  - CONFIG_FUNCTION_TRACER
  - CONFIG_CONTEXT_SWITCH_TRACER
  - CONFIG_IRQSOFF_TRACER
  - CONFIG_PREEMPT_TRACER
  - CONFIG_SCHED_TRACER
  - CONFIG_NOP_TRACER
  - CONFIG_STACK_TRACER
  - CONFIG_FUNCTION_GRAPH_TRACER

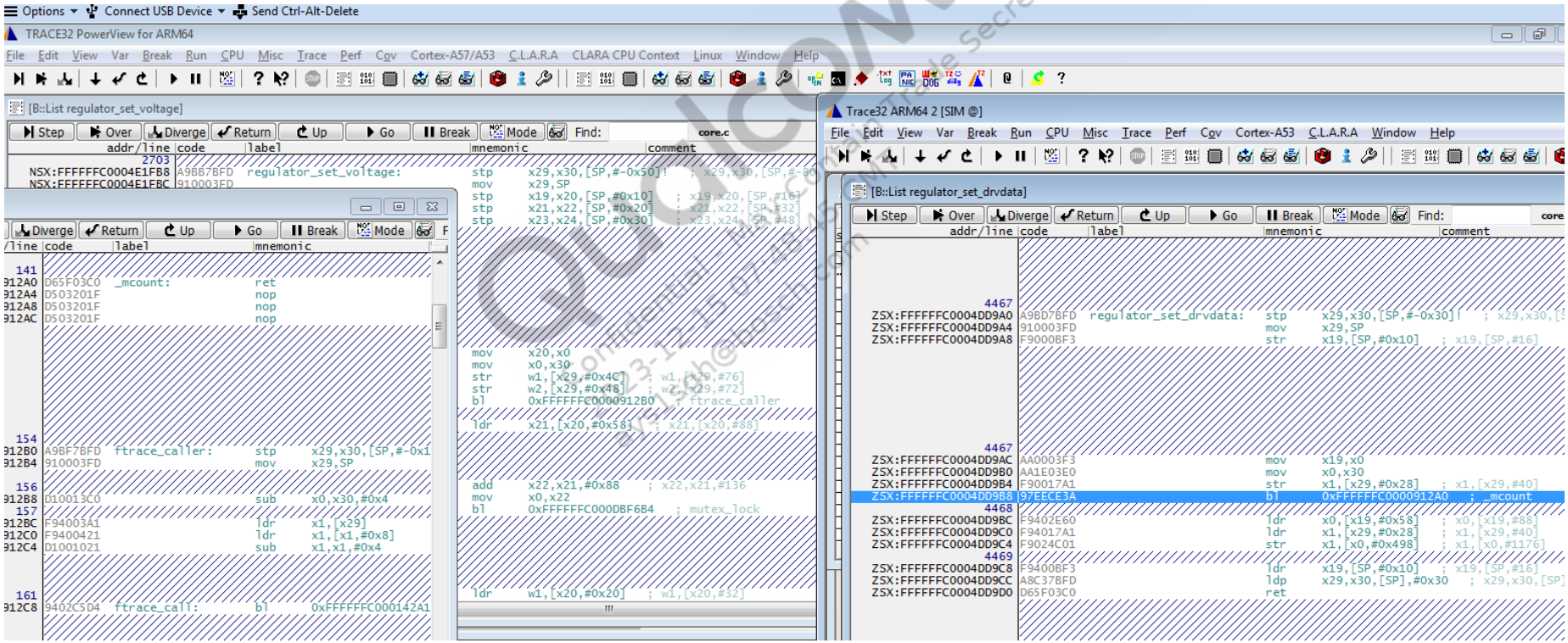
CONFIG_FTRACE (cont.)

- Block diagram



CONFIG_FTRACE (cont.)

- CONFIG_HAVE_DYNAMIC_FTRACE reduce overhead



Example1 for Stability

- events

  #Ftrace

  echo 1 > /sys/kernel/debug/tracing/tracing_on

  #Clock

  echo 1 > /sys/kernel/debug/tracing/events/power/clock_disable/enable

  echo 1 > /sys/kernel/debug/tracing/events/power/clock_enable/enable

  echo 1 > /sys/kernel/debug/tracing/events/power/clock_set_rate/enable

  echo 1 > /sys/kernel/debug/tracing/events/power/clock_state/enable

  echo 1 > /sys/kernel/debug/tracing/events/power/cpu_frequency_switch_end/enable

  echo 1 > /sys/kernel/debug/tracing/events/power/cpu_frequency_switch_start/enable

Example1 for Stability (cont.)

- **Retrieving ftrace** from dump

  crash> extend ../crash/crash7.1.0/extensions/trace.so

  ../crash/crash-7.1.0/extensions/trace.so: shared object loaded

  crash> trace dump -t rawtracedata

  adroidbug$ trace-cmd report -l rawtracedata

- trace-cmd:

  website: http://git.kernel.org/cgit/linux/kernel/git/rostedt/trace-cmd.git/

Example1 for Stability (cont.)

- events

  cfinteractive-272 [004] 25370.268716: cpu_frequency_switch_start: start=691200 end=1017600 cpu_id=0

  cfinteractive-272 [004] 25370.268722: clock_set_rate: a53_clk state=1017600000 cpu_id=4

  cfinteractive-272 [004] 25370.268727: regulator_set_voltage: name=apc0_corner (4-9)

  cfinteractive-272 [004] 25370.268741: regulator_set_voltage: name=pm8950_s6_level_ao (384-448)

  cfinteractive-272 [004] 25370.268748: regulator_set_voltage_complete: name=pm8950_s6_level_ao, val=384

  cfinteractive-272 [004] 25370.268750: regulator_set_voltage: name=pm8950_s5 (975000-1165000)

  cfinteractive-272 [004] 25370.268862: regulator_set_voltage_complete: name=pm8950_s5, val=975000

  cfinteractive-272 [004] 25370.268877: regulator_set_voltage_complete: name=apc0_corner, val=4

Ftrace sysfs

- Configuration and get interface

  cat /sys/kernel/debug/tracing/trace_pipe

```
root@msm8996:/sys/kernel/debug/tracing # ls -l
-r--r--r-- root     root              0 1970-01-01 00:00 README
-r--r--r-- root     root              0 1970-01-01 00:00 available_events
-r--r--r-- root     root              0 1970-01-01 00:00 available_tracers
-rw-rw-r-- root     shell             0 1970-01-01 00:00 buffer_size_kb
-r--r--r-- root     root              0 1970-01-01 00:00 buffer_total_size_kb
-rw-r--r-- root     root              0 1970-01-01 00:00 cpu_freq_switch_profile_enabled
-rw-r--r-- root     root              0 1970-01-01 00:00 current_tracer
drwxr-xr-x root     root                1970-01-01 00:00 events
--w------- root     root              0 1970-01-01 00:00 free_buffer
drwxr-xr-x root     root                1970-01-01 00:00 instances
drwxr-xr-x root     root                1970-01-01 00:00 options
drwxr-xr-x root     root                1970-01-01 00:00 per_cpu
-r--r--r-- root     root              0 1970-01-01 00:00 printk_formats
-r--r--r-- root     root              0 1970-01-01 00:00 saved_cmdlines
-rw-r--r-- root     root              0 1970-01-01 00:00 saved_cmdlines_size
-r--r--r-- root     root              0 1970-01-01 00:00 saved_tgids
-rw-r--r-- root     root              0 1970-01-01 00:00 set_event
-rw-rw---- root     shell             0 1970-01-01 00:00 trace
-rw-rw-r-- root     shell             0 1970-01-01 00:00 trace_clock
--w--w--w- root     root              0 1970-01-01 00:00 trace_marker
-rw-r--r-- root     root              0 1970-01-01 00:00 trace_options
-r--r--r-- root     root              0 1970-01-01 00:00 trace_pipe
-rw-r--r-- root     root              0 1970-01-01 00:00 tracing_cpumask
-rw-rw-r-- root     shell             0 1970-01-01 00:00 tracing_on
-rw-r--r-- root     root              0 1970-01-01 00:00 tracing_thresh
```

Tracer Example

- Function

```
# tracer: function
#
function latency trace v1.1.5 on 2.6.29
--------------------------------------------------------------------
 latency: 0 us, #60136/129495, CPU#0 | (M:preempt VP:0, KP:0, SP:0 HP:0)
    -----------------
    | task: -0 (uid:0 nice:0 policy:0 rt_prio:0)
    -----------------

#                  _------=> CPU#
#                 / _-----=> irqs-off
#                | / _----=> need-resched
#                || / _---=> hardirq/softirq
#                ||| / _--=> preempt-depth
#                |||| /
#                |||||     delay
#  cmd     pid   ||||| time  |   caller
#     \   /      |||||    \  |   /
##### CPU 0 buffer started ####
      sh-1155    0...1 1357960us+: kmem_cache_free (__mmdrop)
      sh-1155    0.... 1357973us+: _read_lock (do_wait)
      sh-1155    0...1 1357986us+: wait_consider_task (do_wait)
      sh-1155    0...1 1358000us+: pid_vnr (wait_consider_task)
      sh-1155    0...1 1358006us+: pid_nr_ns (pid_vnr)
      sh-1155    0...1 1358020us+: thread_group_cputime (wait_consider_task)
      sh-1155    0...1 1358033us+: _spin_lock_irq (wait_consider_task)
      sh-1155    0d..2 1358046us+: _spin_unlock_irq (wait_consider_task)
```

Tracer Example (cont.)

- Sched_switch

```
# tracer: sched_switch
#
#           TASK-PID    CPU#    TIMESTAMP  FUNCTION
#              | |       |          |         |
          <idle>-0     [000]    65.260792:      0:140:R   + [000]  1148:120:D
          <idle>-0     [000]    65.260846:      0:140:R ==> [000]  1148:120:R
              sh-1148  [000]    65.261732:   1148:120:R   + [000]  1149:120:R
              sh-1148  [000]    65.261786:   1148:120:R ==> [000]  1149:120:R
           sleep-1149  [000]    65.262146:   1149:120:R   + [000]   533:115:S
           sleep-1149  [000]    65.262219:   1149:120:R ==> [000]   533:115:R
 USB mass_storag-533   [000]    65.262352:    533:115:S ==> [000]  1149:120:R
           sleep-1149  [000]    65.263039:   1149:120:R   + [000]   533:115:S
           sleep-1149  [000]    65.263126:   1149:120:R ==> [000]   533:115:R
```

Tracer Example (cont.)

- Irqsoff

```
# tracer: irqsoff
#
irqsoff latency trace v1.1.5 on 2.6.29
--------------------------------------------------------------------
 latency: 1453 us, #173/173, CPU#0 | (M:preempt VP:0, KP:0, SP:0 HP:0)
    -----------------
    | task: sh-1142 (uid:0 nice:0 policy:0 rt_prio:0)
    -----------------

#                  _------=> CPU#
#                 / _-----=> irqs-off
#                | / _----=> need-resched
#                || / _---=> hardirq/softirq
#                ||| / _--=> preempt-depth
#                |||| /
#                |||||     delay
#  cmd     pid   ||||| time  |   caller
#     \   /      |||||  \    |   /
      sh-1142    0d..2   13us+: trace_hardirqs_off (__irq_svc)
      sh-1142    0d..3   20us+: asm_do_IRQ (__irq_svc)
      sh-1142    0d..3   27us+: irq_enter (asm_do_IRQ)
      sh-1142    0d..3   33us+: idle_cpu (irq_enter)
      sh-1142    0d.h3   47us+: tick_check_idle (irq_enter)
      sh-1142    0d.h3   53us+: tick_nohz_stop_idle (tick_check_idle)
      sh-1142    0d.h3   60us+: ktime_get (tick_nohz_stop_idle)
      sh-1142    0d.h3   67us+: ktime_get_ts (ktime_get)
      sh-1142    0d.h3   73us+: getnstimeofday (ktime_get_ts)
      sh-1142    0d.h3   80us+: msm_dgt_read (getnstimeofday)
      sh-1142    0d.h3   93us+: msm_read_timer_count (msm_dgt_read)
      sh-1142    0d.h3  100us+: set_normalized_timespec (ktime_get_ts)
```

Tracer Example (cont.)

- preemptirqsoff

```
# tracer: preemptirqsoff
#
preemptirqsoff latency trace v1.1.5 on 2.6.29
--------------------------------------------------------------------
 latency: 1546 us, #181/181, CPU#0 | (M:preempt VP:0, KP:0, SP:0 HP:0)
    -----------------
    | task: adbd-933 (uid:0 nice:0 policy:0 rt_prio:0)
    -----------------

#                  _------=> CPU#
#                 / _-----=> irqs-off
#                | / _----=> need-resched
#                || / _---=> hardirq/softirq
#                ||| / _--=> preempt-depth
#                |||| /
#                |||||     delay
#  cmd     pid   ||||| time  |   caller
#     \   /      |||||  \    |   /
    adbd-933     0...1    7us+: _spin_lock (0)
    adbd-933     0d..2   20us+: _spin_unlock_irqrestore (skb_dequeue)
    adbd-933     0d..3   33us+: asm_do_IRQ (__irq_svc)
    adbd-933     0d..3   40us+: irq_enter (asm_do_IRQ)
    adbd-933     0d..3   47us+: idle_cpu (irq_enter)
    adbd-933     0d.h3   60us+: irq_to_desc (asm_do_IRQ)
    adbd-933     0d.h3   67us+: handle_edge_irq (asm_do_IRQ)
    adbd-933     0d.h3   73us+: _spin_lock (handle_edge_irq)
    adbd-933     0d.h4   80us+: msm_irq_ack (handle_edge_irq)
    adbd-933     0d.h4   87us+: _spin_unlock (handle_edge_irq)
    adbd-933     0d.h3  100us+: handle_IRQ_event (handle_edge_irq)
    adbd-933     0d.h3  107us+: msm_timer_interrupt (handle_IRQ_event)
    adbd-933     0d.h3  113us+: hrtimer_interrupt (msm_timer_interrupt)
    adbd-933     0d.h3  120us+: ktime_get (hrtimer_interrupt)
```

Questions?

https://createpoint.qti.qualcomm.com