**Problem:**

<6>[125.150865699 C1 814 ] spa\_batt\_work : charger type = 5, charging status = 1, 0

<4>[125.421304238 C0 4 ] ANDROID BOOT HAS NOT COMPLETED IN 125s -> PANIC!

<4>[125.427307161 C0 4 ] ------------[ cut here ]------------

<2>[125.432617622 C0 4 ] kernel BUG at /projects/mobcom\_andrwks\_ext18\_scratch/users/nirm/sdb-android-4.4-eos/android/kernel/arch/arm/mach-shmobile/ramdump.c:50!

<0>[125.446326622 C0 4 ] Internal error: Oops - BUG: 0 [#1] PREEMPT SMP ARM

<4>[125.452670392 C0 4 ] Modules linked in:

<4>[125.456249776 C0 4 ] CPU: 0 PID: 4 Comm: kworker/0:0 Not tainted 3.10.17 #1

<4>[125.462945853 C0 4 ] Workqueue: events r8a66597\_vbus\_work

<4>[125.468081545 C0 4 ] task: ef05ccc0 ti: ef074000 task.ti: ef074000

<4>[125.473996007 C0 4 ] PC is at android\_booted\_callback+0x30/0x3c

<4>[125.479645238 C0 4 ] LR is at vprintk\_emit+0x208/0x564

<4>[125.484514622 C0 4 ] pc : [<c002fc30>] lr : [<c0035c28>] psr: 600f0113

<4>[125.484514622 C0 4 ]sp : ef075ce0 ip : ef075c40 fp : ef075cec

On android boot complete, in init.vendor.rc

# boot complete

on property:dev.bootcomplete=1

write /dev/console [AUTOTEST]BOOT\_COMPLETE

write /sys/kernel/PowerManagement/boot\_stop 1

write /sys/module/ramdump/parameters/android\_booted 1

So if the write to android\_booted does not happen even after 125 second, a timer will be fired and crashes the kernel. This is to catch any kind of hang-ups during booting.

**Debugging:**

Most of the debugging is done using crash utility. Arguments passed to crash\_utility are –m phys\_base=0x49000000 –cpus=2

Backtrace of init shows,

crash> bt 1

PID: 1 TASK: ef05c000 CPU: 1 COMMAND: "init"

#0 [<c0680eb8>] (\_\_schedule) from [<c068135c>]

#1 [<c068135c>] (schedule) from [<c0038b90>]

#2 [<c0038b90>] (do\_wait) from [<c003993c>]

#3 [<c003993c>] (sys\_wait4) from [<c0010240>]

pc : [<000197dc>] lr : [<00008717>] psr: 600f0010

sp : bee56de0 ip : 00000000 fp : 0003481c

r10: 00000000 r9 : 00000000 r8 : 00000000

r7 : 00000072 r6 : bee56e18 r5 : 00032ff4 r4 : 00058a30

r3 : 00000000 r2 : 00000000 r1 : bee56de4 r0 : 0000050b

Flags: nZCv IRQs on FIQs on Mode USER\_32 ISA ARM

crash>

Which is basically waiting for a pid 1291( 0x50b).

1291 is again init(which is forked by init for fstab mount). And the main tab waits for the second init(pid 1291) to finish by calling a wait system call.

crash> bt 1291

PID: 1291 TASK: ef218cc0 CPU: 1 COMMAND: "init"

#0 [<c0680eb8>] (\_\_schedule) from [<c068135c>]

#1 [<c068135c>] (schedule) from [<c067f424>]

#2 [<c067f424>] (schedule\_timeout) from [<c06809ec>]

#3 [<c06809ec>] (wait\_for\_common) from [<c0680aa0>]

#4 [<c0680aa0>] (wait\_for\_completion) from [<c004ef18>]

#5 [<c004ef18>] (flush\_work) from [<c0051220>]

#6 [<c0051220>] (schedule\_on\_each\_cpu) from [<c00eea1c>]

#7 [<c00eea1c>] (lru\_add\_drain\_all) from [<c0157ce8>]

#8 [<c0157ce8>] (invalidate\_bdev) from [<c01c53c4>]

#9 [<c01c53c4>] (ext4\_put\_super) from [<c0123f9c>]

#10 [<c0123f9c>] (generic\_shutdown\_super) from [<c012402c>]

#11 [<c012402c>] (kill\_block\_super) from [<c01242e8>]

#12 [<c01242e8>] (deactivate\_locked\_super) from [<c0124f44>]

#13 [<c0124f44>] (deactivate\_super) from [<c013f09c>]

#14 [<c013f09c>] (mntput\_no\_expire) from [<c014027c>]

#15 [<c014027c>] (sys\_umount) from [<c0010240>]

pc : [<00025edc>] lr : [<0000e0a1>] psr: 40000010

sp : bee56d78 ip : 00000000 fp : 0002b032

r10: 00064900 r9 : 00000003 r8 : 000645e8

r7 : 00000034 r6 : 000648e8 r5 : 000648b0 r4 : 00064900

r3 : 0000040a r2 : 00064900 r1 : 00000000 r0 : 000648e8

Flags: nZcv IRQs on FIQs on Mode USER\_32 ISA ARM

**Backtrace of user space is bleow,**

**To take user space back trace, have a look** [**here.**](http://www.hackers4hackers.blogspot.in/2014/03/crash-utility-how-to-view-user-space.html)

**BT for pid 1**

(gdb) bt

#0 wait4 () at bionic/libc/arch-arm/syscalls/wait4.S:10

#1 0x00008716 in do\_mount\_all (nargs=Unhandled dwarf expression opcode 0xf3

) at bionic/libc/include/string.h:217

#2 0x0000992a in execute\_one\_command () at system/core/init/init.c:939

#3 0x00009e56 in main (argc=Unhandled dwarf expression opcode 0xf3

) at system/core/init/init.c:939

(gdb)

**BT for pid 1291**

(gdb) bt

#0 umount2 () at bionic/libc/arch-arm/syscalls/umount2.S:10

#1 0x0000e0a0 in check\_fs (blk\_device=0x648b0 "/dev/block/platform/e6bd0000.mmcif/by-name/MODEMLOG",

fs\_type=0x64900 "ext4", target=0x648e8 "/mnt/modemlog") at system/core/fs\_mgr/fs\_mgr.c:512

#2 0x0000e7ea in fs\_mgr\_mount\_all (fstab=0x645c0) at system/core/fs\_mgr/fs\_mgr.c:512

#3 0x0000873a in do\_mount\_all (nargs=Unhandled dwarf expression opcode 0xf3

) at bionic/libc/include/string.h:217

#4 0x0000992a in execute\_one\_command () at system/core/init/init.c:939

#5 0x00009e56 in main (argc=Unhandled dwarf expression opcode 0xf3

) at system/core/init/init.c:939

Second init process is waiting at an unmount system call which never returned and system crashes.

From the kernel stack trace of process 1291, during an umount system call, kernel calls invalidate\_bdev. Which schedules a work on each cpu.

Name of the work function: lru\_add\_drain\_per\_cpu

schedule\_on\_each\_cpu queues work on system\_wq

**Step 1) Get the address of the system\_wq**

crash> system\_wq

system\_wq = $5 = (struct workqueue\_struct \*) 0xef03d000

**Step2) view the wq struct**

crash> struct workqueue\_struct 0xef03d000

struct workqueue\_struct {

pwqs = {

next = 0xc18bed60,

prev = 0xc18b5d60

},

list = {

next = 0xc09ec0fc,

prev = 0xef03d0c8

},

mutex = {

count = {

counter = 1

},

wait\_lock = {

{

rlock = {

raw\_lock = {

{

slock = 0,

tickets = {

owner = 0,

next = 0

}

}

**Step 2) List all pool work queues in this wq**

crash> list 0xc18bed60

c18bed60

c18b5d60

ef03d000

**Step4) Find the offset of linked list in pwq**

crash> pool\_workqueue.pwqs\_node

struct pool\_workqueue {

[0x60] struct list\_head pwqs\_node;

}

**Step 5) View the struct pool\_workqueue by subtracting 0x60 from addr.**

crash> struct pool\_workqueue c18bed00

struct pool\_workqueue {

pool = 0xc18bb900,

wq = 0xef03d000,

work\_color = 0,

flush\_color = -1,

refcnt = 1,

nr\_in\_flight = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

nr\_active = 0,

max\_active = 256, //just for verification

delayed\_works = {

next = 0xc18bed58,

prev = 0xc18bed58

},

pwqs\_node = {

next = 0xc18b5d60,

prev = 0xef03d000

},

mayday\_node = {

next = 0xc18bed68,

prev = 0xc18bed68

},

unbound\_release\_work = {

data = {

counter = -32

},

entry = {

next = 0xc18bed74,

prev = 0xc18bed74

},

func = 0xc00513fc <pwq\_unbound\_release\_workfn>

},

rcu = {

next = 0x0,

func = 0

}

}

**Step6 ) view the associated worker pool**

struct worker\_pool {

cpu = 1,

node = 0,

id = 2,

flags = 0,

worklist = {

next = 0xc18bb914,

prev = 0xc18bb914

},

nr\_workers = 3,

nr\_idle = 3,

idle\_list = {

**step7 ) From step 2 pic the next address**

**View the struct pool\_workqueue by subtracting 0x60 from addr.**

crash> struct pool\_workqueue c18b5d00

struct pool\_workqueue {

pool = 0xc18b2900,

wq = 0xef03d000,

work\_color = 0,

flush\_color = -1,

refcnt = 9,

nr\_in\_flight = {7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

nr\_active = 7,

max\_active = 256,

delayed\_works = {

next = 0xc18b5d58,

prev = 0xc18b5d58

},

pwqs\_node = {

next = 0xef03d000,

prev = 0xc18bed60

},

mayday\_node = {

next = 0xc18b5d68,

prev = 0xc18b5d68

},

unbound\_release\_work = {

data = {

counter = -32

},

entry = {

next = 0xc18b5d74,

prev = 0xc18b5d74

},

func = 0xc00513fc <pwq\_unbound\_release\_workfn>

**step8) View associated worker\_pool**

struct worker\_pool {

cpu = 0,

node = 0,

id = 0,

flags = 0,

worklist = {

next = 0xc18b1f18,

prev = 0xef2cde94

},

nr\_workers = 6,

nr\_idle = 5,

idle\_list = {

next = 0xef139000,

prev = 0xef139600

},

idle\_timer = {

**Step10) list of pending works**

crash> list 0xc18b1f18

c18b1f18

ef01c368

c18b8d50

ee5f3dfc

f00d6004

ef01c438

ef2cdea4

ef2cde94

c18b2914

crash>

**Step11) Work struct.entry offset**

crash> work\_struct.entry

struct work\_struct {

[4] struct list\_head entry;

}

crash>

**Step12) see each work struct substracing 4**

**A)**

crash> struct work\_struct c18b1f14

struct work\_struct {

data = {

counter = 0xc18b5d05

},

entry = {

next = 0xef01c368,

prev = 0xc18b2914

},

func = 0xc00fbab0 <vmstat\_update>

}

**B)**

crash> struct work\_struct ef01c364

struct work\_struct {

data = {

counter = 0xc18b5d05

},

entry = {

next = 0xc18b8d50,

prev = 0xc18b1f18

},

func = 0xc03fd880 <d2153\_monitor\_temperature\_work>

}

**C)**

crash> struct work\_struct c18b8d4c

struct work\_struct {

data = {

counter = 0xc18b5d0d

},

entry = {

next = 0xee5f3dfc,

prev = 0xef01c368

},

func = 0xc00ee9e0 <l**ru\_add\_drain\_per\_cpu**>

}

crash>

**D)**

crash> struct work\_struct ee5f3df8

struct work\_struct {

data = {

counter = 0xc18b5df5

},

entry = {

next = 0xf00d6004,

prev = 0xc18b8d50

},

func = 0xc004dce0 <wq\_barrier\_func>

}

**E)**

crash> struct work\_struct f00d6000

struct work\_struct {

data = {

counter = 0xc18b7a05

},

entry = {

next = 0xef01c438,

prev = 0xee5f3dfc

},

func = 0xc03c7c90 <bcmtch\_deferred\_worker>

**F)**

crash> struct work\_struct ef01c434

struct work\_struct {

data = {

counter = 0xc18b5d05

},

entry = {

next = 0xef2cdea4,

prev = 0xf00d6004

},

func = 0xc037bc6c <init\_vdd\_fault\_work>

}

crash>

**G) ef2cdea4**

crash> struct work\_struct ef2cdea0

struct work\_struct {

data = {

counter = 0xc18b5d05

},

entry = {

next = 0xef2cde94,

prev = 0xef01c438

},

func = 0xc04001c4 <fan5405\_timer\_work\_func>

}

crash>

**H) ef2cde94**

crash> struct work\_struct ef2cde90

struct work\_struct {

data = {

counter = 0xc18b5d05

},

entry = {

next = 0xc18b2914,

prev = 0xef2cdea4

},

func = 0xc0400098 <fan5405\_intr\_work\_func>

}

**I) c18b2914**

crash> struct work\_struct c18b2910

struct work\_struct {

data = {

counter = 0

},

entry = {

next = 0xc18b1f18,

prev = 0xef2cde94

},

func = 0x6

}

crash>

During a filesystem umount requested by init, kernel schedule work on each cpu for a flush.

The work (lru\_add\_drain\_per\_cpu) was queued on system\_wq. Listing the number of work items pending on system\_wq, shows that lru\_add\_drain\_per\_cpu was still pending.

Below is the list of works pending on workerqueue 0(ie on core 0 normal work queue)

crash> list work\_struct.entry -s work\_struct.func -h 0xc18b1f14

c18b1f14

func = 0xc00fbab0 <vmstat\_update>

ef01c364

func = 0xc03fd880 <d2153\_monitor\_temperature\_work>

c18b8d4c

func = 0xc00ee9e0 **<lru\_add\_drain\_per\_cpu>**

ee5f3df8

func = 0xc004dce0 <wq\_barrier\_func>

f00d6000

func = 0xc03c7c90 <bcmtch\_deferred\_worker>

ef01c434

func = 0xc037bc6c <init\_vdd\_fault\_work>

ef2cdea0

func = 0xc04001c4 <fan5405\_timer\_work\_func>

ef2cde90

func = 0xc0400098 <fan5405\_intr\_work\_func>

c18b2910

func = 0x6

crash>

crash>

Worker pool shows that out of 6 workers only one is active for cpu 0 and that is [kworker/0:0] which is current process during crash. Below is the dump of worker pool.

struct worker\_pool {

cpu = 0,

node = 0,

id = 0,

flags = 0,

worklist = {

next = **0xc18b1f18**,

prev = 0xef2cde94

},

**nr\_workers = 6,**

**nr\_idle = 5,**

Now looking at the back trace of Kworker/0:0 , we can see that kworker is hanging in work function r8a66597\_vbus\_work.

000 |android\_booted\_callback(ignored = ?)

001 |call\_timer\_fn(timer = 0xC09EA1F4, fn = 0xC002FC00, data = 0)

002 |run\_timer\_softirq(?)

003 |\_\_do\_softirq()

004 |do\_softirq()

005 |irq\_exit()

006 |handle\_IRQ(irq = 126, ?)

007 |gic\_handle\_irq(regs = 0xEF075E30)

008 |\_\_irq\_svc(asm)

--> |exception

**009 |usb\_phy\_read(?)**

**010 |r8a66597\_vbus\_work**(work = 0xEF175E90)

011 |process\_one\_work(worker = 0xEF03A500, work = 0xEF175E90)

012 |process\_scheduled\_works(worker = 0xEF03A500)

013 |worker\_thread(\_\_worker = 0xEF03A500)

014 |kthread(\_create = 0xEF061E18)

015 |ret\_from\_fork(asm)

--> |exception

--- |end of frame

|

In source code, "drivers/usb/gadget/r8a66597-udc.c".

The code in red below looks buggy. There should have been a timeout for the do while loop.

/\*\*

\*\*\* usb\_phy\_read - Read data of TUSB1211 register via PHY

\*\*\* @return None

\*\*\*/

static u8 usb\_phy\_read(u64 reg)

{

u8 monreg;

\_\_raw\_writew((reg & PHY\_READ\_MASK), USB\_SPADDR);

\_\_raw\_writew(((reg >> 6) << 4), USB\_SPEXADDR);

/\* Issue read comand \*/

\_\_raw\_writew(\_\_raw\_readw(USB\_SPCTRL) | USB\_SPRD, USB\_SPCTRL);

udc\_log("%s: Waiting for read Permission\n", \_\_func\_\_);

**do {**

**cpu\_relax();**

**monreg = \_\_raw\_readw(USB\_SPCTRL);**

**} while (monreg & USB\_SPRD);**

udc\_log("%s: Read Permission granted\n", \_\_func\_\_);

monreg = \_\_raw\_readw(USB\_SPRDAT);

printk(KERN\_INFO "Value of PHY\_READ is %x \n", monreg);

return monreg;

}