USB Debugging and Profiling Techniques

Kishon Vijay Abraham I and Basak Partha
Agenda

• Introduction
• USB Generic Linux System Architecture
• USB Mass Storage Architecture
• Challenges in debugging
• USB Debugging Techniques (sysfs, usbmon, dynamic debug interface, tracepoint, protocol analyzer)
• Gadget Zero
• Other profiling tools
Introduction

- Widespread use of USB in embedded space

HOST MODE:
- To connect Ethernet/Hub
- To connect Modem
- To connect mass storage devices

DEVICE MODE
- Acts as mass storage device
- USB Speakers
- USB serial device
- USB webcam
Introduction

- Ease of use
- Hot pluggable
- Speed
- Reliability
- Power devices from the bus
- Bus expansion using hub
- USB On-The-Go
USB Generic Linux System Architecture

- Input, Sound, FS.. Subsystem
- Gadget Driver
- UDC
- USB Device Stack
- Firmware Driver
- Device Controller
- PHY

- Application
- libusb
- VFS
- Class Driver
- HCD/usbcore
- Firmware Driver
- Host Controller
- PHY

Device

HOST

HW comp

SW comp
Challenges in debugging

- Data on the bus is encoded
- Timing issues
- Out-of spec signaling errors
- Protocol errors
- Multiple layers makes it difficult to identify where exactly the problem originates
- Too many formatted prints lead to skewed results
USB debugging

• Using linux kernel facilities
  – sysfs/debugfs
  – usbmon
  – Dynamic debug interface
  – Tracepoints

• Using debug tools and Analyzers
  – Elisys/Lecroy/total Phase analyzer tools
  – ETM
Sysfs Entry in host

ls /sys/bus/usb/devices/

1-0:1.0  1-1  1-1.1  1-1.1:1.0  1-1:1.0  usb1

• The names that begin with "usb" refer to USB controllers
• The devices are named by a scheme
  bus-port.port.port
• The interfaces are indicated by suffixes having this form
  :config.interface
• All information about the device will be under this entry

The Documentation information for this entry can be obtained from http://www.linux-usb.org/FAQ.html#i6
Debugfs Entry in host

cat /sys/kernel/debug/usb/devices

T: Bus=01 Lev=00 Prnt=00 Port=00 Cnt=00 Dev#= 1 Spd=480 MxCh= 3
B: Alloc= 0/800 us ( 0%), #Int= 1, #Iso= 0
D: Ver= 2.00 Cls=09(hub ) Sub=00 Prot=00 MxPS=64 #Cfgs= 1
P: Vendor=1d6b ProdID=0002 Rev= 3.06
S: Manufacturer=Linux 3.7.0-rc2-next-20121026-00003-g72580a5 ehci_hcd
S: Product=OMAP-EHCI Host Controller
S: SerialNumber=ehci-omap.0
C:* #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr= 0mA
I:* If#= 0 Alt= 0 #EPs= 1 Cls=09(hub ) Sub=00 Prot=00 Driver=hub
E: Ad=81(I) Atr=03(Int.) MxPS= 4 Ivl=256ms

T: Bus=01 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#= 2 Spd=480 MxCh= 5
D: Ver= 2.00 Cls=09(hub ) Sub=00 Prot=02 MxPS=64 #Cfgs= 1
P: Vendor=0424 ProdID=9514 Rev= 2.00
S: SerialNumber=ehci-omap.0
C:* #Ifs= 1 Cfg#= 1 Atr=e0 MxPwr= 2mA
I: If#= 0 Alt= 0 #EPs= 1 Cls=09(hub ) Sub=00 Prot=01 Driver=hub
E: Ad=81(I) Atr=03(Int.) MxPS= 1 Ivl=256ms
I:* If#= 0 Alt= 1 #EPs= 1 Cls=09(hub ) Sub=00 Prot=02 Driver=hub
E: Ad=81(I) Atr=03(Int.) MxPS= 1 Ivl=256ms

The Documentation information for this entry can be obtained from
Documentation/usb/proc_usb_info.txt
Debugfs Entry in host contd..

T:   Bus=01 Lev=02 Prnt=02 Port=00 Cnt=01 Dev#=  3 Spd=480  MxCh=  0
D:   Ver= 2.00 Cls=ff(vend.) Sub=00 Prot=01 MxPS=64  #Cfgs=  1
P:   Vendor=0424 ProdID=ec00 Rev= 2.00
C:*  #Ifs=  1 Cfg#=  1 Atr=e0 MxPwr=  2mA
I:*  If#=  0 Alt=  0 #EPs=  3 Cls=ff(vend.) Sub=00 Prot=ff Driver=smsc95xx
E:   Ad=81(I) Atr=02(Bulk) MxPS= 512 Ivl=0ms
E:   Ad=02(O) Atr=02(Bulk) MxPS= 512 Ivl=0ms
E:   Ad=83(I) Atr=03(Int.) MxPS=  16 Ivl=1ms

The Documentation information for this entry can be obtained from Documentation/usb/proc_usb_info.txt
Debugfs (in device controller)

- drivers/usb/dwc3/debugfs.c
- Show the entire register dump
- Change the port mode (device, host, OTG)
- Force the controller to work at a particular speed (super, high..)
Debug example using debugfs

• PROBLEM STATEMENT

Added PM support for dwc3 and realized once the system goes to off mode and comes back the device is not enumerating. Adding some debug prints showed the we are not getting any interrupts in dwc3.

Configuration Diagram
Debug example using debugfs

• Took the register dump before the system goes to suspend and after resuming using “cat /debug/dwc3.0/regdump”

• After going to OFF mode and coming back, DEVTEN register lost the contents.

• DEVTEN controls the generation of device specific events.

• The fix is to restore the contents of DEVTEN after coming back from OFF mode.
usbmon

- Facility in kernel to be used to collect URB traces
- USB monitoring facilities for Linux consists of a kernel part and user part
- Reports requests made by peripheral-specific drivers to Host Controller Drivers (HCD)
- Requires a reliable working HCD
- Has a "text" and "binary" API
**usbmon Architecture**

- **Binary**
  - Character device (for each host controller)
  - `/dev/usbmonX`
  - tcpdump
  - wireshark

- **Text**
  - Debugfs entry (for each host controller)
  - `/sys/kernel/debug/usb/usbmon/Xu`
  - vusb-analyzer

- **usbmon main**
- **HCD**
- `urb_submit`
- `urb_complete`
- `add/remove bus`

- **User space tools**
  - Linux kernel comp
  - User space tools
usbmon ASCII capture

• mount -t debugfs none_debugs /sys/kernel/debug
• modprobe usbmon
• ls /sys/kernel/debug/usb/usbmon/
  0s 0u 1s 1t 1u
• cat /sys/kernel/debug/usbmon/1u > usbmon.mon
• ./vusb-analyzer usbmon.mon
usbmon binary capture

• modprobe usbmon

• ls /dev/usbmon

  usbmon0 usbmon1

• sudo tcpdump -i usbmon1 -w usbmon.pcap &

• ./wireshark usbmon.pcap
Decoding usbmon captures

- Usbmon trace

```
d2263780 469874560 S Ci:3:003:0 s 80 06 0100
0000 0008 8 <

d2263780 469875939 C Ci:3:003:0 0 8 = 12010002 ef020140
```

<table>
<thead>
<tr>
<th>URB Address</th>
<th>Time stamp</th>
<th>Urb Event</th>
<th>Transfer &amp; Direction</th>
<th>Bus Number</th>
<th>Device Number</th>
<th>Endpoint Number</th>
<th>URB Status</th>
<th>bmRequest type</th>
<th>bRequest</th>
</tr>
</thead>
<tbody>
<tr>
<td>d2263780</td>
<td>469874560</td>
<td>S</td>
<td>Ci</td>
<td>3</td>
<td>003</td>
<td>0</td>
<td>s</td>
<td>80</td>
<td>06</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>wValue</th>
<th>wIndex</th>
<th>wLength</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URB Address</th>
<th>Time stamp</th>
<th>Urb Event</th>
<th>Transfer &amp; Direction</th>
<th>Bus Number</th>
<th>Device Number</th>
<th>Endpoint Number</th>
<th>URB Status</th>
<th>Length</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>d2263780</td>
<td>469875939</td>
<td>C</td>
<td>Ci</td>
<td>3</td>
<td>003</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>= 12010002 ef020140</td>
</tr>
</tbody>
</table>
Debug examples using usbmon

• PROBLEM STATEMENT

  usb stick works fine under older kernels (tested on 2.6.34 and 2.6.37) but stopped working for 3.0 and 3.1

• Obtain usbmon traces for working and non working setup

• Filter the usbmon traces for only the required devices

  cat usb_nw.mon | grep 2:011 > usb_nw_dev.mon
  cat usb_w.mon | grep 1:019 > usb_w_dev.mon

• ./vusb-analyzer usb_nw_dev.mon usb_w_dev.mon

• The non working log has some additional commands compared to working log
Debug examples using usbmon contd.

From the diff, there are additional commands in the non-working case like the UDISK shown in the command.

Then the URB return status shows broken PIPE error.

The host sends CLEAR ENDPOINT HALT and this happens again and again.

So it's concluded some user space program send bogus commands causing the issue.
Debug examples using usbmon

• PROBLEM STATEMENT

  Webcam does not work when connected to USB 2 port but works on USB 3

• Obtain usbmon traces for working and non working setup

• Filter the usbmon traces for only the required devices

  cat usb2.mon | grep 1:006 > usb2_nw.mon
  cat usb3.mon | grep 3:003 > usb3_w.mon

• ./vusb-analyzer usb2_nw.mon usb3_w.mon
• Both traces show that the webcam stopped being used for a short time and was suspended

• when it was resumed again, it worked okay in USB3 but did not work in USB-2

• The workaround was to disable auto-suspend (and debug the device)
Dynamic Debug Interface

• Extended version of printk

• Use dev_dbg and dev_vdbg to control debug messages

• Enabled using DDEBUG and DVERBOSE_DEBUG compiler options

• If CONFIG_DYNAMIC_DEBUG is NOT set, everything under dev_dbg turns to normal printk

• If CONFIG_DYNAMIC_DEBUG is set
  − a new debugfs entry /sys/kernel/debug/dynamic_debug/control gets created
  − Writing to this file will enable or disable specific debugging functions
  − e.g., echo file gadget.c line 269 +p > .../dynamic_debug/control
GOOD: out transfer on dwc3 device mode:
[ 274.694458] =>>>queing request ed7cdea0 to ep2out-bulk length 512
[ 274.694458] dwc3 dwc3.0: ep2out-bulk: req ed7cdea0 dma ac9f0000 length 512 last
[ 274.713378] dwc3 dwc3.0: ep2out-bulk: cmd 'Start Transfer' params 00000000 ad5ab110 00000000
[ 274.713378] dwc3 dwc3.0: Command Complete -- > 0
[ 274.726989] dwc3 dwc3.0: ep2out-bulk: Transfer Complete
[ 274.732482] =>>request ed7cdea0 from ep2out-bulk completed 16/512
===> 0

BAD: out transfer on dwc3 device mode:
[ 217.927062] =>>>queing request ecb1ecc0 to ep1out-bulk length 24
[ 219.333374] dwc3 dwc3.0: ep1out-bulk: req ecb1ecc0 dma aca1b000 length 24 last
[ 219.333374] dwc3 dwc3.0: ep1out-bulk: cmd 'Start Transfer' params 00000000 ad55e000 00000000
[ 219.349822] dwc3 dwc3.0: Command Complete -- > 0
[ 219.360168] dwc3 dwc3.0: ep1out-bulk: endpoint busy
The log shows OUT transfers of 512 bytes is getting succeeded but OUT transfer of size 24 is failing

Confirms maxpacket aligned transfers for out direction are succeeded and there is a problem with short packet transfer

The problem is that the controller requires OUT transfers to be aligned on wMaxPacketSize, even if we *know* the correct size

So the fix is done in the gadget driver to give maxpacket aligned buffers to the controller.
Protocol Analyzer

- A protocol analyzer decodes, filters, and displays USB data
- Some analyzers also have an exerciser along with it that can generate data on the bus
Debug Example using Protocol Analyzer

- USB tethering is not working on omap5
- Captured CATC trace in omap4(working) and in omap5(non-working)
Debug Example using Protocol Analyzer

• The host sends a QUERY message for OID_802_3_PERMANENT_ADDRESS

• In the working case the device responds with a QUERY_COMPLT message with some DATA in it.

• In the non working case the device responds with a QUERY_COMPLT without any DATA in it.

• So the host sends a HALT message

• The issue was around copying data from the bounce buffer to the gadget layer.
tracepoint

- Used to record data at a specific point in the kernel for later retrieval
- Light weight hooks added to the kernel
- Two types: static tracepoint and dynamic tracepoint
- can be used by a number of tools for kernel debugging and performance problem diagnosis
- They have zero overhead when disabled and minimal overhead when enabled
Defining Static tracepoint

• Macros existing to create a static tracepoint include
  – TRACE_EVENT()
  – TP_PROTO()
  – TP_ARGS()
  – TP_STRUCT__entry()
  – TP_fast_assign()
  – TP_printk()

• The TRACE_EVENT() is created by

  TRACE_EVENT(name, proto, args, struct, assign, print)
Defining Static tracepoint

- Macros existing to create a static tracepoint to do more than printing
  - DECLARE_TRACE()
  - DECLARE_TRACE_NOARGS()
  - DEFINE_TRACE()

- To hook into the tracepoints
  
  register_trace_[tracepoint_func]()

- To unregister the hook
  
  unregister_trace_[tracepoint_func]()
Tracepoint Sample

• Patch to log usb_request from gadget layer
  
  http://gitorious.org/linux-usb/linux-usb/commit/a7e7fb69808c

• Patch to log every read/write in dwc3
  
  http://comments.gmane.org/gmane.linux.usb.general/64821
  ([PATCH] usb: dwc3: add trace support)
Gadget Zero

- Obtain performance characteristics
- Two configuration device
  - sinks and sources bulk data
  - loops back a configurable number of transfers
- A kernel driver (host), drivers/usb/misc/usbtest.c, where the actual test cases live
- Userland software to call that driver, such as testusb.c and test.sh
- Can't be used for class or vendor-specific functionality
Sample test.sh output

** Control test cases:
  test 9: ch9 postconfig
  /dev/bus/usb/001/027 test 9, 64.183046 secs
  test 10: control queueing
  /dev/bus/usb/001/027 test 10, 11.738630 secs
  test 14: control writes
  /dev/bus/usb/001/027 test 14, 5.432296 secs

assuming sink-src configuration

** Host Write (OUT) test cases:
  test 1: 5000 transfers, same size
  /dev/bus/usb/001/027 test 1, 1.624243 secs
  test 3: 5000 transfers, variable/short size
  /dev/bus/usb/001/027 test 3, 1.090523 secs
  test 5: 2000 scatterlists, same size entries
  /dev/bus/usb/001/027 test 5, 13.404251 secs
  test 7a: 2000 scatterlists, variable size/short entries
  /dev/bus/usb/001/027 test 7, 8.839292 secs

Test Case No 1,3,5,7 are Write test cases
Test Case No 2,4,6,8 are read test cases
Sample test.sh output contd.

test 7b: 2000 scatterlists, variable size/bigger entries
/dev/bus/usb/001/027 test 7, 6.701336 secs

test 7c: 2000 scatterlists, variable size/micro entries
/dev/bus/usb/001/027 test 7, 4.173024 secs

** Host Read (IN) test cases:
test 2: 5000 transfers, same size
/dev/bus/usb/001/027 test 2, 0.803351 secs

test 4: 5000 transfers, variable size
/dev/bus/usb/001/027 test 4, 0.784580 secs

test 6: 2000 scatterlists, same size entries
/dev/bus/usb/001/027 test 6, 5.442718 secs

test 8: 2000 scatterlists, variable size entries
/dev/bus/usb/001/027 test 8, 3.621307 secs
Other Profiling Tools

• iozone
  – Can be used to profile mass-storage devices
  – iozone can perform 13 types of test: Read, write, re-read, re-write, read backwards, read strided, fread etc.,
  – Sudo iozone -Rab massstorage.xls -i0 -i1 -e -f /dev/sdb

• FFSB

• dd
  – Copy a file, converting and formatting according to the operands
  – dd if=/dev/zero of=/dev/sdb bs=128 count=1024
References

• Documentation/usb/*
• Documentation/trace/*
• Drivers/usb/*
• lwn.net
• Bootstrap Yourself with Linux-USB Stack
• linux-usb list
• http://vusb-analyzer.sourceforge.net/
THANK YOU

For Queries and Feedback

kishon@ti.com, kishonvijayabraham@gmail.com
partha_basak2000@yahoo.com