Embedded Linux Conference Europe 2013

Linux Kernel Debugging And Profiling Tools
Agenda

- Introduction
- Kernel Features
- Tracers
- Debuggers
- Profilers
- Miscellaneous
- Questions
Introduction

Motivations:
- BSP adaptations to various hardwares
- Drivers development
- Optimizations

Debugging and profiling:
What are the available features in the kernel?
What tools exit in the community?
Definitions

Tracing
Specialized use of logging to collect state or behavior information about a program’s execution. Done by addition of trace points in the program's source code.

Interactive Debugging
Process of monitoring the execution of a program by interactively inspecting and/or modifying its internal state (memory, stacks, registers, etc). No source code modifications but requires specialized software.

Profiling
Process of sampling the execution of a program to analyze its performance.
Agenda

- Introduction
- Kernel Features
- Tracers
- Debuggers
- Profilers
- Miscellaneous
- Questions
Kernel Features

printk

linux/printk.h:
 asmlinkage __printf(1, 2) __cold
 int printk(const char *fmt, ...);

- Famous kernel's printf like function with log levels support
- Circular log buffer, can be called from any context
- CONFIG_PRINTK, /proc/sys/kernel/printk, dmesg tool
- Documentation/printk-formats.txt
- Probably the most commonly used routine for kernel debugging
- Unconditional trace point, overheads

    printk(KERN_DEBUG "Got there: %s:%i\n", __FILE__, __LINE__);
    printk(KERN_CRIT "I give up, bye\n");
Kernel Features

DebugFS

- Special RAM-based file system dedicated to kernel debugging
- No special rules, developers are free to put any info they want there
- `CONFIG_DEBUG_FS`
- `Documentation/filesystems/debugfs.txt`
- Typically mounted in `/sys/kernel/debug`
  ```
  # mount -t debugfs none /sys/kernel/debug
  ```

```c
linux/debugfs.h:
  debugfs_create_file
  debugfs_create_dir
  debugfs_create_blob
  debugfs_remove
  ...
```
Kernel Features
Dynamic Debug

- Dynamic activation/deactivation of kernel information code
- Currently, dynamic activation of `pr_debug/dev_dbg` per-callsite
- `CONFIG_DYNAMIC_DEBUG`
- `Documentation/dynamic-debug-howto.txt`
- Debugfs control file
  `<debugfs>/dynamic_debug/control`

```c
linux/printk.h:
#define pr_debug(fmt, ...) dynamic_pr_debug(fmt, ##__VA_ARGS__)

linux/device.h
#define dev_dbg(dev, format, ...) \
do { dynamic_dev_dbg(dev, format, ##__VA_ARGS__); } while (0)
```
Kernel Features

Probes

- **Static probes: tracing oriented**
  - Present in the source code
  - Optional code path: unlikely branch
  - Small overhead even when off
- **Dynamic probes: debug oriented**
  - Optional code path : breakpoint
  - No overhead when off, High overhead when on
  - Debug symbols and deep knowledge of the kernel mapping required
Kernel Features
Static probes

- Kernel Markers
  - First implementation
  - Mainline from 2.6.24 to 2.6.32

- Tracepoints
  - Dissociates instrumentation code from kernel code
  - Mainline since 2.6.28

- TRACE_EVENT macro
  - Tracepoints enhancement with automatic callback
  - Mainline since 2.6.32
Kernel Features
Static probes

User space
Kernel space

ftrace, lttng, perf

Tracepoint

ID
Binary format

Ring buffer

String format

ftrace, lttng, perf
Kernel Features
Static probes

Before 2.6.37

- Trace on?
  - Yes: Do trace
  - No: Continue

After 2.6.37

- NOP
  - Do trace

Trace on ? (Before 2.6.37)

Do trace

 NOP

Do trace (After 2.6.37)
Kernel Features
Dynamic probes

- **Kprobes**
  - Software breakpoint
  - Mainline since 2.6.9 (ARM: 2.6.25)
Kernel Features

Perf events

- API for performance measurements
- Hardware events
  - Special registers on most modern CPUs
  - Limited number, round robin when needed
  - Cycles, cache misses, branch hits ...
- Software events
  - Context switches, page faults ...
- Mainline since 2.6.31 (ARM: 2.6.34)
- On ARM, access to the PMU through the CP15
Kernel Features

Slabinfo

- Functionality
  - Memory usage on the slabs
  - Access through /proc/slabinfo
- Mainline since 2.6.24

<table>
<thead>
<tr>
<th># name</th>
<th>&lt;active_objs&gt;</th>
<th>&lt;num_objs&gt;</th>
<th>&lt;object_size&gt;</th>
<th>&lt;objperslab&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext4_groupinfo_4k</td>
<td>24</td>
<td>24</td>
<td>168</td>
<td>24</td>
</tr>
<tr>
<td>SCTP</td>
<td>12</td>
<td>12</td>
<td>1280</td>
<td>12</td>
</tr>
<tr>
<td>rxrpc_call_jar</td>
<td>0</td>
<td>0</td>
<td>832</td>
<td>19</td>
</tr>
<tr>
<td>bsg_cmd</td>
<td>0</td>
<td>0</td>
<td>288</td>
<td>14</td>
</tr>
</tbody>
</table>

[...]
Agenda

- Introduction
- Kernel Features
- Tracers
- Debuggers
- Profilers
- Miscellaneous
- Questions
Tracers
Ftrace (Function Tracer)

- Functionality
  - Various tracers (function, graph function, irqs-off, wakeup, branch)
  - Various time sources (local and global clocks, atomic counter)
  - Function and event filters
  - Early boot tracing
  - Dump on Oops
  - No user tool required, debugfs access
- Mainline since 2.6.27 (ARM: 2.6.34)
- Based on mcount and debugfs, optionally on tracepoints and kprobes
Function tracer:

<table>
<thead>
<tr>
<th>TASK-PID</th>
<th>CPU#</th>
<th>TIMESTAMP</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;idle&gt;-0</td>
<td>[001]</td>
<td>6075.461563: mwait_idle &lt;-&gt;cpu_idle</td>
<td></td>
</tr>
<tr>
<td>bash-16939</td>
<td>[000]</td>
<td>6075.461563: __fsnotify_parent &lt;-&gt;vfs_write</td>
<td></td>
</tr>
</tbody>
</table>

Branch tracer:

<table>
<thead>
<tr>
<th>TASK-PID</th>
<th>CPU#</th>
<th>TIMESTAMP</th>
<th>CORRECT</th>
<th>FUNC:FILE:LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kworker/0:1-12</td>
<td>[000]</td>
<td>7.440216: [ ok ] console_unlock:printk.c:2108</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;idle&gt;-0</td>
<td>[000]</td>
<td>7.440383: [ MISS ] note_interrupt:spurious.c:284</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Stack tracer:

<table>
<thead>
<tr>
<th>Depth</th>
<th>Size</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>0)</td>
<td>2952</td>
<td>mempool_alloc_slab+0x15/0x17</td>
</tr>
<tr>
<td>1)</td>
<td>2936</td>
<td>mempool_alloc+0x52/0x104</td>
</tr>
</tbody>
</table>
Tracers
Ftrace

Irqs-off:
# _------=> CPU#
# / _------=> irqs-off
# | / _------=> need-resched
# || / _------=> hardirq/softirq
# ||| / _------=> preempt-depth
# |||| / 
# cmd pid time | caller
# \

<idle>-0 0d . . . 0us +: __dev_pm_qos_read_value <rpm_check_suspend_allowed
<idle>-0 0d . . . 31us : __rpm_callback <rpm_suspend

[...]

=normal context  s= soft irq  h=hard irq  H=hard irq occurred in a soft irq
<1µs  =<10µs  != >100µs
Tracers
Trace-cmd & Kernelshark
Tracers
Lttng

• Functionality:
  – Tracepoint tracer
  – Kprobes support
  – Mcount support
• Not in mainline (modules)
• Available since 2.6.9 (ARM: 2.6.20)
• Based on tracepoints, optionally on kprobes and mcount
Agenda

- Introduction
- Kernel Features
- Tracers
- Debuggers
- Profilers
- Miscellaneous
- Questions
Debuggers

KGDB

- Functionality
  - GDB server
  - Serial transport / kdb front-end
  - Breakpoints, watchpoints and step-by-step debugging
  - Early boot debug
- Mainline since 2.6.26 (KDB merge: 2.6.35)
Debuggers

KGDB
Debuggers
KGTP

- Functionality:
  - GDB server / kprobe
  - tracepoint support
  - Serial and Ethernet transports
- Not in mainline (module)
- Available since 2.6.18
- Based on kprobes, optionally on tracepoints
Agenda

- Introduction
- Kernel Features
- Tracers
- Debuggers
- Profilers
- Miscellaneous
- Questions
Profilers
Perf

• An API but also a set of tools
• Functionality:
  – Various profilers
  – Event based sampling
• Mainline since 2.6.31 (ARM: 2.6.34)
• Based on perf_events, optionally on tracepoints
• On ARM, Oprofile uses perf as backend
Profilers
Perf

Perf Top:

PerfTop:  279 irqs/sec  kernel:96.1%  exact: 0.0% [4000Hz cycles], (all, 1 CPU)

-----------------------------------------------------------------------------------

60.77% [kernel]  [k] cpuidle_wrap_enter
10.29% [kernel]  [k] finish_task_switch.constprop.59
8.99%  [kernel]  [k] tick_nohz_idle_enter
3.29%  [kernel]  [k] tick_nohz_idle_exit
2.33%  [kernel]  [k] __do_softirq
0.96%  [kernel]  [k] ftrace_likely_update
0.92%  [kernel]  [k] rcu_idle_enter

[...]
### Perf report:

- # Samples: 8K of event 'cycles'
- # Event count (approx.): 1745355260

<table>
<thead>
<tr>
<th>Overhead</th>
<th>Command</th>
<th>Shared Object</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.33%</td>
<td>hackbench</td>
<td>[kernel.kallsyms]</td>
<td>[k] ftrace_likely_update</td>
</tr>
<tr>
<td>9.21%</td>
<td>hackbench</td>
<td>[kernel.kallsyms]</td>
<td>[k] finish_task_switch.constprop.59</td>
</tr>
<tr>
<td>6.37%</td>
<td>hackbench</td>
<td>[kernel.kallsyms]</td>
<td>[k] memchr_inv</td>
</tr>
<tr>
<td>4.75%</td>
<td>hackbench</td>
<td>[kernel.kallsyms]</td>
<td>[k] unix_stream_recvmsg</td>
</tr>
<tr>
<td>2.78%</td>
<td>hackbench</td>
<td>[kernel.kallsyms]</td>
<td>[k] __wake_up_sync_key</td>
</tr>
</tbody>
</table>

[...]
Profilers

Perf

Perf annotate:

Percent | Source code & Disassembly of libpthread-2.15.so

-----------------------

: Disassembly of section .text:

: 0000f7a0 __read:

7.32 : f7a0: ldr ip, [pc, #-40] ; f780 __local_syscall_error+0x28>

3.25 : f7a4: ldr ip, [pc, ip]

6.50 : f7a8: teq ip, #0

0.00 : f7ac: push {r7} ; (str r7, [sp, #-4]!)

0.00 : f7b0: bne f7cc __read+0x2c>

[...]
Profilers
Kmemleak

• Functionality
  – Garbage collector based memory profiler
  – Kmalloc, vmalloc and friends
  – Log pointer, size and backtrace, unlog at free
  – Periodic scan (default: 10 min)

• Mainline since 2.6.31
# cat /sys/kernel/debug/kmemleak
unreferenced object 0xef42d000 (size 28):
comm "khubd", pid 189, jiffies 4294937550 (age 2543.370s)
hex dump (first 28 bytes):
00 01 10 00 00 02 20 00 08 d0 42 ef 08 d0 42 ef
00 00 00 00 00 00 00 ff ff ff ff
backtrace:
[<c0080fe1>] create_object+0xa1/0x1ac
[<c007eac5>] kmem_cache_alloc+0x8d/0xdc
[<c01a966d>] isp1760_urb_enqueue+0x2f9/0x358
[<c019bbbd>] usb_hcd_submit_urb+0x75/0x574
[<c019d8f1>] usb_start_wait_urb+0x29/0x80
[<c019daad>] usb_control_msg+0x89/0xac
Agenda

- Introduction
- Kernel Features
- Tracers
- Debuggers
- Profilers
- Miscellaneous
- Questions
Miscellaneous
SystemTap

• Functionality:
  – Lightweight debug
  – Kprobes front-end using a scripting language
  – Trace dump on Oops
  – Tracepoint support
  – Perf support

• Not in mainline (module)

• Available since 2.6.9

• Based on kprobe, optionally on tracepoint and perf_event
SystemTap

- **SystemTap script**
- **Tapset**
- **Debug-info**

**Parse (stap)**

**probe.c**

**Build**

**Host**

**Target**

- **Load probe.ko** (staprun)
- **SystemTap output** (stapio)
%
#include<linux/netdevice.h>
%

function change_mtu (dev:long, val:long) %{
    int ret;
    struct net_device *netdevice;
    netdevice = (struct net_device *)(long)THIS->dev;
    dev_set_mtu(netdevice, THIS->val);
%

probe module("tg3").function("tg3_get_stats") {
    change_mtu($dev, 2000)
    printf("Changing tg3 MTU\n")
}
Miscellaneous
Kexec & Crash

• Functionality
  – Post-mortem analysis
  – Boot on a new kernel instance on panic
  – Old kernel memory dump through /proc/vmcore

• Mainline since 2.6.13
Agenda

- Introduction
- Kernel Features
- Tracers
- Debuggers
- Profilers
- Miscellaneous
- Questions
Questions & Answers

Thank you for your attention

Any Question ?