Dynamic Tracing Tools on ARM/AArch64 platform
Updates and Challenges

Hiroyuki ISHII
Panasonic Corporation
Hiroyuki ISHII

• Panasonic Corporation, Automotive section, Japan
• Linux engineer
• Recent work:
  ◦ Linux debugging tools
• Previous work:
  ◦ Linux kernel engineer
  ◦ Bootloader: U-Boot, arm-trusted-firmware
This Talk

• Aim: to promote the **Dynamic Tracing Tools in Embedded**

• Introduction of the Dynamic Tracing Capability & Tools
  ◦ Including recent updates

• A Porting Example of the Dynamic Tracing Tool to AArch64
Panasonic IVI Products

- Panasonic is a Tier-1 IVI* supplier for various OEMs
  - Largest IVI supplier in the world
- Using Linux on ARM CPU in several IVI products

[*] In-Vehicle-Infotainment
Current IVI System Development

- IVI system is getting much larger & much more complex
  - Navigation, Voice recognition, Multimedia streaming & ripping
  - Database, Networking, Connectivity
  - Security, Fastboot, Update management ...

- Trying to use the latest hardware & various open source
  - ARMv8 64bit (aka AArch64) & Linux 4.x (will be in product in near future)
  - Hundreds of open source components

- "Tens of million lines" of unfamiliar source code
  - Consisting of open source and In-house code
  - Terribly complexed debugging – Tons of unknown issues
Q. How can we debug "Tens of million lines" of unfamiliar source code?
Q. How can we debug "Tens of million lines" of unfamiliar source code?

A. Dynamic Tracing
What's Dynamic Tracing?

A way of watching every function (w/o few exceptions), at anytime

- "Dynamic"
  - Attach to a live system
  - No precaution (pre-editing code)
  - Flexible & ad-hoc

- "Tracing"
  - Observe system/program behavior
  - Monitor a particular function
  - Profiling
  - Function call history

http://www.brendangregg.com/perf.html
Dynamic Tracing in Embedded

• Why now?
  ◦ Linux 4.x: powerful tracing capabilities
    ▪ kprobes for AArch64, BPF – mainlined
    ▪ uprobes for AArch64 – almost ready to merge
  ◦ Dynamic Tracing Tools
    ▪ Improve in past 2–3 years, by supporting BPF etc.

• What are the Advantages?
  ◦ Advanced Observability
    ▪ Break down the issues quickly & clearly
  ◦ Less Installation Cost
    ▪ No need to change the product code
Dynamic Tracing Capability
Linux Observability Tools – Traditional and New

Linux Performance Observability Tools

- strace
- lsof
- pstat
- pidstat
- ftrace
- stap
- lttng
- bcc (BPF)
- perf
- iostat
- iotop
- blkrade

- Operating System
- Applications
- System Libraries
- VFS
- File Systems
- Schedule
- System Call Interface
- Scheduler
- Kernel

- VFS
- File Systems
- Volume Manager
- Block Device Interface
- Device Drivers

- I/O Controller
- Disk
- Disk
- Swap

- Network Controller
- Port
- Port

- Interface Transports
- Ethtool
- Snmpget
- Lldptool

- I/O Bus
- Expander Interconnect
- I/O Bridge

- CPU
- Memory
- CPU Interconnect

- Various:
  - sar /proc dmesg
dstat
- turbostat
dmesg
- perf

- Hardware
- perf
- mpstat
- vmstat
- slabbtop
- free
- top ps
- pidstat
- numastat
- nicstat
- netstat
- ip
Linux Observability Tools – Traditional and New

Dynamic Tracing Tools

Linux Performance Observability Tools

- strace
- lsof
- pcstat
- ss
- ltrace
- perf
- ftrace
- stap
- lttng
- bcc (BPF)
- netstat
- sysdig
- hardware
- perf
- mpstat
- CPU Interconnect
- top ps
- pidstat
- vmstat
- slabtop
- free
- iperf
- tcpdump
- I/O Bus
- I/O Bridge
- ldfptool
- nicstat
- netstat
- ip
- swap
- swapon
- ethtool
- snmpget
- Port
- Port
- Network Controller
- Interface Transports
Linux Dynamic Tracing Frameworks

- Dynamic tracing tools works thanks to the various tracing frameworks
  - tracepoints - Statically defined tracing
  - kprobes - Kernel dynamic probing
  - uprobes - User space dynamic probing
  - ftrace - Kernel function tracing
  - perf_events - Hardware event tracing & sampling
  - BPF - In-kernel Virtual Machine, Trace data processing
    - Summarize
    - Efficient data transfer
    - Makes many things practical
Type of Dynamic Tracing Tools

1. Programable multi-tool - SystemTap, bcc, ply
   - Provides a programable framework for tracing
   - Bundled scripts as individual analyzing tools

2. NOT programable multi-tool - perf
   - All-in-one tracing tool
   - Use with command line arguments

3. Single-purpose tools - perf-tools
   - A group of individual analyzing tools
Dynamic Tracing Tools (example: perf-tools)
Dynamic Tracing Tools (example: bcc)
Dynamic Tracing Usage
Trace open() syscalls – prints all the file-open events

<table>
<thead>
<tr>
<th>PID</th>
<th>COMM</th>
<th>FD</th>
<th>ERR</th>
<th>PATH</th>
</tr>
</thead>
<tbody>
<tr>
<td>8550</td>
<td>gcc</td>
<td>3</td>
<td>0</td>
<td>/lib/x86_64-linux-gnu/libz.so.1</td>
</tr>
<tr>
<td>8550</td>
<td>gcc</td>
<td>-1</td>
<td>2</td>
<td>/home/foobar/local/lib/libc.so.6</td>
</tr>
<tr>
<td>8550</td>
<td>gcc</td>
<td>3</td>
<td>0</td>
<td>/lib/x86_64-linux-gnu/libc.so.6</td>
</tr>
<tr>
<td>8550</td>
<td>gcc</td>
<td>-1</td>
<td>2</td>
<td>/etc/ccache.conf</td>
</tr>
<tr>
<td>8550</td>
<td>gcc</td>
<td>3</td>
<td>0</td>
<td>/home/foobar/.ccache/ccache.conf</td>
</tr>
<tr>
<td>8550</td>
<td>gcc</td>
<td>4</td>
<td>0</td>
<td>/home/foobar/.ccache/f/stats</td>
</tr>
<tr>
<td>8550</td>
<td>gcc</td>
<td>4</td>
<td>0</td>
<td>/home/foobar/.ccache/f/stats.tmp.foobar.8550.t8HMvc</td>
</tr>
<tr>
<td>8550</td>
<td>gcc</td>
<td>-1</td>
<td>2</td>
<td>/home/foobar/local/lib/tls/x86_64/libc.so.6</td>
</tr>
<tr>
<td>8550</td>
<td>gcc</td>
<td>-1</td>
<td>2</td>
<td>/home/foobar/local/lib/tls/libc.so.6</td>
</tr>
<tr>
<td>8550</td>
<td>gcc</td>
<td>-1</td>
<td>2</td>
<td>/home/foobar/local/lib/x86_64/libc.so.6</td>
</tr>
<tr>
<td>8550</td>
<td>gcc</td>
<td>-1</td>
<td>2</td>
<td>/home/foobar/local/lib/libc.so.6</td>
</tr>
<tr>
<td>8550</td>
<td>gcc</td>
<td>4</td>
<td>0</td>
<td>/etc/ld.so.cache</td>
</tr>
<tr>
<td>8550</td>
<td>gcc</td>
<td>4</td>
<td>0</td>
<td>/lib/x86_64-linux-gnu/libc.so.6</td>
</tr>
<tr>
<td>8550</td>
<td>gcc</td>
<td>4</td>
<td>0</td>
<td>/usr/lib/locale/locale-archive</td>
</tr>
<tr>
<td>8535</td>
<td>configure</td>
<td>3</td>
<td>0</td>
<td>configure.log</td>
</tr>
</tbody>
</table>
execsnoop

Trace new processes via exec() syscalls

<table>
<thead>
<tr>
<th>PCOMM</th>
<th>PID</th>
<th>PPID</th>
<th>RET</th>
<th>ARGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>8550</td>
<td>8535</td>
<td>0</td>
<td>/home/foobar/.ccache/bin/gcc -v</td>
</tr>
<tr>
<td>gcc</td>
<td>8550</td>
<td>8535</td>
<td>0</td>
<td>/usr/bin/gcc -v</td>
</tr>
<tr>
<td>cat</td>
<td>8551</td>
<td>8535</td>
<td>0</td>
<td>/bin/cat ztest8535.c</td>
</tr>
<tr>
<td>gcc</td>
<td>8552</td>
<td>8535</td>
<td>0</td>
<td>/home/foobar/.ccache/bin/gcc -c ztest8535.c</td>
</tr>
<tr>
<td>gcc</td>
<td>8553</td>
<td>8552</td>
<td>0</td>
<td>/usr/bin/gcc -E ztest8535.c</td>
</tr>
<tr>
<td>cc1</td>
<td>8554</td>
<td>8553</td>
<td>0</td>
<td>/usr/lib/gcc/x86_64-linux-gnu/5/cc1 -E -quiet -imultiarch x86_64-linux-gnu ztest8535.c -mtune=generic -march=x86-64 -fstack-protector-strong -Wformat -Wformat-security</td>
</tr>
<tr>
<td>gcc</td>
<td>8555</td>
<td>8552</td>
<td>0</td>
<td>/usr/bin/gcc -c -o ztest8535.o /home/ishii/.ccache/tmp/ztest8535.stdout.foobar.8552.KMn6Zr.i</td>
</tr>
<tr>
<td>cc1</td>
<td>8556</td>
<td>8555</td>
<td>0</td>
<td>/usr/lib/gcc/x86_64-linux-gnu/5/cc1 -fpreprocessed /home/foobar/.ccache/tmp/ztest8535.stdout.foobar.8552.KMn6Zr.i -quiet -dumpbase ztest8535.stdout.foobar.8552.KMn6Zr.i -mtune=generic -march=x86-64 -auxbase-strip ztest8535.o -fstack-protector-strong -Wformat -Wformat-security -o /tmp/cc8HaudQ.s</td>
</tr>
<tr>
<td>gcc</td>
<td>8557</td>
<td>8555</td>
<td>-2</td>
<td>/home/foobar/.ccache/bin/as --64 -o ztest8535.o /tmp/cc8HaudQ.s</td>
</tr>
<tr>
<td>as</td>
<td>8557</td>
<td>8555</td>
<td>0</td>
<td>/home/foobar/local/bin/as --64 -o ztest8535.o /tmp/cc8HaudQ.s</td>
</tr>
</tbody>
</table>
Trace block device I/O with PID and latency

$ ./biosnoop

<table>
<thead>
<tr>
<th>TIME(s)</th>
<th>COMM</th>
<th>PID</th>
<th>DISK</th>
<th>T</th>
<th>SECTOR</th>
<th>BYTES</th>
<th>LAT(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000000000</td>
<td>jbd2/sda1-8</td>
<td>253</td>
<td>sda</td>
<td>W</td>
<td>969250944</td>
<td>86016</td>
<td>0.33</td>
</tr>
<tr>
<td>0.017384000</td>
<td>jbd2/sda1-8</td>
<td>253</td>
<td>sda</td>
<td>W</td>
<td>969251112</td>
<td>4096</td>
<td>0.12</td>
</tr>
<tr>
<td>2.755776000</td>
<td>jbd2/sda3-8</td>
<td>564</td>
<td>sda</td>
<td>W</td>
<td>2911149624</td>
<td>8192</td>
<td>0.13</td>
</tr>
<tr>
<td>2.766396000</td>
<td>jbd2/sda3-8</td>
<td>564</td>
<td>sda</td>
<td>W</td>
<td>2911149640</td>
<td>4096</td>
<td>0.11</td>
</tr>
<tr>
<td>3.779816000</td>
<td>kworker/u16:3</td>
<td>1879</td>
<td>sda</td>
<td>W</td>
<td>1941794816</td>
<td>4096</td>
<td>0.14</td>
</tr>
<tr>
<td>3.780044000</td>
<td>kworker/u16:3</td>
<td>1879</td>
<td>sda</td>
<td>W</td>
<td>1941794856</td>
<td>4096</td>
<td>0.36</td>
</tr>
<tr>
<td>3.780058000</td>
<td>kworker/u16:3</td>
<td>1879</td>
<td>sda</td>
<td>W</td>
<td>2080206896</td>
<td>4096</td>
<td>0.37</td>
</tr>
<tr>
<td>3.780065000</td>
<td>kworker/u16:3</td>
<td>1879</td>
<td>sda</td>
<td>W</td>
<td>2080206976</td>
<td>4096</td>
<td>0.38</td>
</tr>
<tr>
<td>3.780073000</td>
<td>kworker/u16:3</td>
<td>1879</td>
<td>sda</td>
<td>W</td>
<td>2080207104</td>
<td>4096</td>
<td>0.38</td>
</tr>
<tr>
<td>3.780079000</td>
<td>kworker/u16:3</td>
<td>1879</td>
<td>sda</td>
<td>W</td>
<td>2080207248</td>
<td>4096</td>
<td>0.39</td>
</tr>
<tr>
<td>3.780170000</td>
<td>kworker/u16:3</td>
<td>1879</td>
<td>sda</td>
<td>W</td>
<td>2080207376</td>
<td>4096</td>
<td>0.48</td>
</tr>
<tr>
<td>3.780204000</td>
<td>kworker/u16:3</td>
<td>1879</td>
<td>sda</td>
<td>W</td>
<td>2080207768</td>
<td>4096</td>
<td>0.51</td>
</tr>
<tr>
<td>3.780296000</td>
<td>kworker/u16:3</td>
<td>1879</td>
<td>sda</td>
<td>W</td>
<td>2080208784</td>
<td>4096</td>
<td>0.60</td>
</tr>
<tr>
<td>3.780335000</td>
<td>kworker/u16:3</td>
<td>1879</td>
<td>sda</td>
<td>W</td>
<td>2080212720</td>
<td>4096</td>
<td>0.64</td>
</tr>
<tr>
<td>3.780389000</td>
<td>kworker/u16:3</td>
<td>1879</td>
<td>sda</td>
<td>W</td>
<td>2080276848</td>
<td>4096</td>
<td>0.69</td>
</tr>
<tr>
<td>3.780497000</td>
<td>kworker/u16:3</td>
<td>1879</td>
<td>sda</td>
<td>W</td>
<td>2138928736</td>
<td>4096</td>
<td>0.80</td>
</tr>
<tr>
<td>4.011546000</td>
<td>bash</td>
<td>26308</td>
<td>sda</td>
<td>W</td>
<td>382200192</td>
<td>4096</td>
<td>0.11</td>
</tr>
</tbody>
</table>
Kernel function call graph & durations

```bash
$ ./funcgraph -a -m 3 do_sys_open
Tracing "do_sys_open"... Ctrl-C to end.
# tracer: function_graph

#
#      TIME         CPU   TASK/PID          DURATION      FUNCTION CALLS
#       |           |      |     |            |    |                      |    |    |    |
28554.134852 | 6)   cat-26389 | | | | | | | | | | | do_sys_open() {  
28554.134854 | 6)   cat-26389 | | | | | | | | | | |     | getname() {  
28554.134854 | 6)   cat-26389 | | | 0.342 us | | | | | | | | | | |     | getname_flags();  
28554.134855 | 6)   cat-26389 | | | 1.062 us | | | | | | | | | | |     | }  
28554.134855 | 6)   cat-26389 | | | | | | | | | | | | get_unused_fd_flags() {  
28554.134856 | 6)   cat-26389 | | | 0.580 us | | | | | | | | | | | |     | __alloc_fd();  
28554.134857 | 6)   cat-26389 | | | 1.068 us | | | | | | | | | | | | }  
28554.134857 | 6)   cat-26389 | | | | | | | | | | | | do_filp_open() {  
28554.134857 | 6)   cat-26389 | | | 5.956 us | | | | | | | | | | | |     | path_openat();  
28554.134864 | 6)   cat-26389 | | | 0.098 us | | | | | | | | | | | |     | restore_nameidata();  
28554.134864 | 6)   cat-26389 | | | 7.117 us | | | | | | | | | | | | }  
28554.134865 | 6)   cat-26389 | | | | | | | | | | | | put_unused_fd() {  
28554.134865 | 6)   cat-26389 | | | 0.054 us | | | | | | | | | | | |     | _raw_spin_lock();
```
funccount

Count kernel & user function calls

```
$ ./funccount '/bin/bash:set*'
Traceing 49 functions for "/bin/bash:set*"... Hit Ctrl-C to end.
^C

<table>
<thead>
<tr>
<th>FUNC</th>
<th>COUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_working_directory</td>
<td>1</td>
</tr>
<tr>
<td>set_or_show_attributes</td>
<td>9</td>
</tr>
<tr>
<td>set_var_attribute</td>
<td>9</td>
</tr>
<tr>
<td>set_signal</td>
<td>31</td>
</tr>
<tr>
<td>set_debug_trap</td>
<td>31</td>
</tr>
<tr>
<td>set_dollar-vars_unchanged</td>
<td>73</td>
</tr>
<tr>
<td>set_shellopts</td>
<td>91</td>
</tr>
<tr>
<td>set_sigint_handler</td>
<td>104</td>
</tr>
<tr>
<td>set_dollar-vars_changed</td>
<td>146</td>
</tr>
<tr>
<td>set_sigchld_handler</td>
<td>195</td>
</tr>
<tr>
<td>setifs</td>
<td>223</td>
</tr>
<tr>
<td>set_current_prompt_level</td>
<td>271</td>
</tr>
<tr>
<td>set_pipestatus_from_exit</td>
<td>633</td>
</tr>
<tr>
<td>set_pipestatus_array</td>
<td>708</td>
</tr>
<tr>
<td>set_signal_handler</td>
<td>1789</td>
</tr>
</tbody>
</table>
```
Trace arbitrary functions with filters

```
$ ./trace 'sys_read (arg3 > 20000) "read %d bytes", arg3'

<table>
<thead>
<tr>
<th align="right">PID</th>
<th align="right">TID</th>
<th align="right">COMM</th>
<th>FUNC</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td align="right">4136</td>
<td align="right">4136</td>
<td align="right">bash</td>
<td>sys_read</td>
<td>read 306218 bytes</td>
</tr>
<tr>
<td align="right">32359</td>
<td align="right">32359</td>
<td align="right">cat</td>
<td>sys_read</td>
<td>read 131072 bytes</td>
</tr>
<tr>
<td align="right">32361</td>
<td align="right">32361</td>
<td align="right">grep</td>
<td>sys_read</td>
<td>read 32768 bytes</td>
</tr>
</tbody>
</table>
```

```
$ ./trace 'r:bash:readline "%s", retval'

<table>
<thead>
<tr>
<th align="right">PID</th>
<th align="right">TID</th>
<th align="right">COMM</th>
<th>FUNC</th>
</tr>
</thead>
<tbody>
<tr>
<td align="right">26308</td>
<td align="right">26308</td>
<td align="right">bash</td>
<td>readline</td>
</tr>
<tr>
<td align="right">26308</td>
<td align="right">26308</td>
<td align="right">bash</td>
<td>readline</td>
</tr>
<tr>
<td align="right">26308</td>
<td align="right">26308</td>
<td align="right">bash</td>
<td>readline</td>
</tr>
<tr>
<td align="right">26308</td>
<td align="right">26308</td>
<td align="right">bash</td>
<td>echo $PATH</td>
</tr>
</tbody>
</table>
```
Many Other Useful Tools...

- capable: Trace security capability checks
- biolatency: Summarize block device I/O latency
- biotop: Top for disks
- filetop: File reads and writes by filename and process
- ext4slower: Trace slow ext4 operations
- cachestat: Trace page cache hit/miss ratio
- argdist: Display function parameter values as a histogram

see: https://github.com/iovisor/bcc
Dynamic Tracing Tools

Summary and Recent Updates
perf-tools

- [https://github.com/brendangregg/perf-tools](https://github.com/brendangregg/perf-tools)
- Single-Purpose Observability Tools (Closed?)

**Pros**
- Few dependencies

**Cons**
- Direct dependency on intel architecture
- Less customizable
echo nop > current_tracer

offset=$($awk 'BEGIN { o = 0; } $1 == "#" & & $2 ~ /TASK/ & & NF == 6 { o = 1; } $2 ~ /TASK/ { print o; exit }' trace)
warn "echo > trace"
(if (( opt_duration )); then
  # wait then dump buffer
  sleep $duration
  cat trace
) | $awk -v o=$offset -v opt_name=$opt_name -v name=$name \
    -v opt_duration=$opt_duration -v opt_time=$opt_time \
    -v opt_fail=$opt_fail \
    -v opt_file=$opt_file -v file=$file'

# common fields

$1 ! = "#" {
  # task name can contain dashes and space
  split($0, line, ":")
  sub(/^[\t\r\n]+/", "", line[1])
  comm = line[1]
  if (opt_name & & match(comm, name) == 0)
    next
  sub(/^[.]*$/", "", line[2])
  pid = line[2]
}

(...)

opensnoop 55/255 lines: Written by Brendan Gregg, GPLv2
SystemTap

- [https://sourceware.org/systemtap/](https://sourceware.org/systemtap/)
- Powerful Multi-Tool with Nice Language

**Pros**
- High-level, Less-arch-dependent, easy to understand language
- Ready for AArch64, basically

**Cons**
- Takes long time before running a script
- Relatively unsafe
  - Both cons might be fixed with BPF support?
SystemTap – Script

#!/usr/bin/stap

probe begin
{
    printf("%6s %6s %16s %4s %s\n", "UID", "PID", "COMM", "FD", "PATH");
}

probe nd_syscall.open.return
{
    /* trace on return to ensure the pathname has faulted in */
    printf("%6d %6d %16s %4d %s\n", uid(), pid(), execname(),
           retval(), user_string(entry(pointer_arg(1)), "-"));
}

opensnoop-nd.stp: Written by Brendan Gregg, GPLv2
bcc

- [https://github.com/iovisor/bcc](https://github.com/iovisor/bcc)
- Promising BPF Front-end Tool, under active development

**Pros**
- Low overhead & Extra capabilities, thanks to BPF
- Python / Lua / Go / C++ front-end support
- Many useful scripts

**Cons**
- Lower level language
- Direct dependency on intel architecture
from __future__ import print_function
from bcc import BPF
from time import sleep

b = BPF(text='''#include "uapi/linux/ptrace.h"

struct key_t {
  u64 ip;
};

BPF_TABLE("hash", struct key_t, u64, counts, 256);

int do_count(struct pt_regs *ctx) {
  struct key_t key = {};
  u64 zero = 0, *val;
  key.ip = PT_REGS_IP(ctx);
  val = counts.lookup_or_init(&key, &zero);
  (*val)++;
  return 0;
}
'''
)

b.attach_kprobe(event="vfs_*", fn_name="do_count")

# header
print("Tracing... Ctrl-C to end.")

# output
try:
  sleep(99999999)
except KeyboardInterrupt:
  pass

print("%-16s %-26s %8s" % ("ADDR", "FUNC", "COUNT"))
counts = b.get_table("counts")
for k, v in sorted(counts.items(), key=lambda counts: counts[1].value):
  print("%-16x %-26s %8d" % (k.ip, b.ksym(k.ip), v.value))

vfscount: Written by Brendan Gregg, Apache License v2.0
ply

- [https://github.com/wkz/ply](https://github.com/wkz/ply)
- Upcomming BPF Front-end tool
- Pros
  - Simple, High-level language
  - BPF support
- Cons
  - Unstable
  - Development seems relatively slow
  - AArch64 is not supported yet
#!/usr/bin/env ply

kprobe:SyS_*
{
    $syscalls[func].count()
}

$ ply syscall-count.ply
340 probes active
^Cde-activating probes

$syscalls:
sys_nanosleep            1
sys_fdatsasync           1
sys_timerfd_settime      1
sys_rt_sigaction         3
sys_alarm                5
sys_mmap                 6
[...]
sys_select               309
sys_setitimer            314
sys_close                350
sys_writev               719
sys_write                814
sys_read                 1288
sys_ioctl                2577
sys_poll                 3977
sys_recmmsg              5540
sys_futex                50567
perf

- [https://github.com/torvalds/linux/tree/master/tools/perf](https://github.com/torvalds/linux/tree/master/tools/perf)
- Performance Analyzing Tool, In Linux Source Tree
- Pros
  - Reliable & Ready to use, on many architecture
  - Advanced capabilities – CPU statics by PMU (Performance Monitoring Unit)
  - BPF support
- Cons
  - Not much programable
  - Much key strokes
# Summary of Tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Programable I/F</th>
<th>Capability</th>
<th>Overhead</th>
<th>Safety/ Stability</th>
<th>Code Change</th>
<th>BPF</th>
<th>AArch64 Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcc</td>
<td>Good</td>
<td>Great</td>
<td>Low</td>
<td>Safe</td>
<td>No</td>
<td>Yes</td>
<td>Partially Yes</td>
</tr>
<tr>
<td>SystemTap</td>
<td>Great</td>
<td>Good</td>
<td>High</td>
<td>Unsafe</td>
<td>No</td>
<td>wip</td>
<td>Partially Yes</td>
</tr>
<tr>
<td>perf</td>
<td>Not Good</td>
<td>Good</td>
<td>Low-Mid</td>
<td>-</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>perf-tools</td>
<td>Not Good</td>
<td>Good</td>
<td>Low-Mid</td>
<td>-</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ply</td>
<td>Great</td>
<td>Good</td>
<td>Low</td>
<td>Unstable</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

**NOTE:** My personal opinion, means no special recommendation
Port Dynamic Tracing Tools to AArch64
Port Dynamic Tracing Tools

- AArch64-Porting Example of bcc
- I'm also trying to port several other tools
  - Currently available: perf-tools
- All patches are put on my github
  - https://github.com/hiroishii/bcc
  - https://github.com/hiroishii/perf-tools
Environments

- Reference Board
  - Renesas R-Car Gen3 Salvator-X (AArch64)

- Linux 4.9
  - git://git.kernel.org/pub/scm/linux/kernel/git/horms/renesas-bsp.git v4.9/rcar-3.5.1

- Extra kernel patches
  - uprobes for AArch64 – Developed by Pratyush Anand at RedHat
    - https://github.com/pratyushanand/linux uprobe/upstream_arm64_devel_v3

```
arm64: kprobe: protect/ rename few definitions to be reused by uprobe
arm64: kgdb_step_brk_fn: ignore other's exception
arm64: Handle TRAP_TRACE for user mode as well
arm64: Handle TRAP_BRKPT for user mode as well
arm64: introduce mm context flag to keep 32 bit task information
arm64: Add uprobe support
```
## Kernel Configs

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIG_BPF=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_BPF_JIT=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_BPF_SYSCALL=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_BPF_EVENTS=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_KPROBES=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_KPROBE_EVENT=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_UPROBES=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_UPROBE_EVENT=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_BRANCH_PROFILE_NONE=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_CONTEXT_SWITCH_TRACER=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_DYNAMIC_FTRACE=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_EVENT_TRACING=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_FTRACE=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_FTRACE_MCOUNT_RECORD=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_FTRACE_SYSCALLS=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_FUNCTION_GRAPH_TRACER=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_FUNCTION_PROFILER=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_FUNCTION_TRACER=y</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIG_GENERIC_TRACER=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_NOP_TRACER=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_PREEMPT_TRACER=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_RELAY=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_RING_BUFFER=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_RING_BUFFER_ALLOW_SWAP=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_SCHED_TRACER=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_STACKTRACE=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_STACK_TRACER=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_TRACEPOINTS=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_TRACER_MAX_TRACE=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_TRACER_SNAPSHOT=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_TRACER_SNAPSHOT_PER_CPU_SWAP=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_TRACE_CLOCK=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_TRACE_ENUM_MAP_FILE=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_TRACING=y</td>
<td></td>
</tr>
<tr>
<td>CONFIG_FRAME_POINTER=y</td>
<td></td>
</tr>
</tbody>
</table>
Other Environments

• Ubuntu 16.04 host
  ○ gcc 5.4.0, make 4.1
  ○ cmake 3.7

• Cross compiler
  ○ gcc v5.2.1 – Linaro Toolchain (aarch64-poky-linux-gcc)

• Target userland (AArch64)
  ○ Yocto 2.x based
  ○ Multilib environment – "lib64" directory for 64bit libraries
  ○ Login via serial console, bash 4.3
Port bcc

- **Version**
  - master on github (5 Feb 2017, c510eff)

- **Dependencies (noteworthy)**
  - llvm/clang
  - elfutils, flex
  - python (optional)
Port bcc cont.

Porting Steps

- Cross-compile llvm/clang
- Cross-compile bcc
- Fix 1. Multilib support
- Fix 2. *(Workaround)* Module linkage problem
- Fix 3. *(Workaround)* asm macro
- Fix 4. Port each script
Cross-compile llvm/clang

• Version
  ○ llvm, clang, compiler-rt version 3.9.1 (branch: release_39)
    https://github.com/llvm-mirror/llvm.git
    https://github.com/llvm-mirror/clang.git
    https://github.com/llvm-mirror/compiler-rt.git
Cross-compile llvm/clang cont.

Creating directory structure for build

$ git clone https://github.com/llvm-mirror/llvm.git --branch release_39
$ cd ${llvm_root_dir}/tools
$ git clone https://github.com/llvm-mirror/clang.git --branch release_39
$ cd ${llvm_root_dir}/projects
$ git clone https://github.com/llvm-mirror/compiler-rt.git --branch release_39

- $ ...: command on the host PC
- # ...: command on the target board
- Abstracted paths such as ${llvm_root_dir}, ${sysroot}, etc
$ cd ${llvm_root_dir}
$ mkdir build && cd build
$ cmake .. -DCMAKE_CROSSCOMPIRING=1 -DCMAKE_SYSROOT=${sysroot} \
   -DCMAKE_FIND_ROOT_PATH=${sysroot} -DCMAKE_FIND_ROOT_PATH_MODE_PROGRAM=NEVER \
   -DCMAKE_FIND_ROOT_PATH_MODE_LIBRARY=ONLY -DCMAKE_FIND_ROOT_PATH_MODE_INCLUDE=ONLY \
   -DCMAKE_INSTALL_PREFIX=${sysroot}/usr -DCMAKE_INSTALL_LIBDIR=lib64 \
   -DCMAKE_C_COMPILER=aarch64-poky-linux-gcc -DCMAKE_CXX_COMPILER=aarch64-poky-linux-g++ \
   -DCMAKE_C_FLAGS="-mabi=lp64 -march=armv8-a -mtune=cortex-a57.cortex-a53 -Wall --sysroot=${sysroot}" \
   -DCMAKE_CXX_FLAGS="-mabi=lp64 -march=armv8-a -mtune=cortex-a57.cortex-a53 -Wall --sysroot=${sysroot} -std=c++11" \
   -DCMAKE_EXE_LINKER_FLAGS="--sysroot=${sysroot}" -DCMAKE_SHARED_LINKER_FLAGS="--sysroot=${sysroot}" \
   -DCMAKE_BUILD_TYPE=Release -DLLVM_TABLEGEN=/home/foobar/local/bin/llvm-tblgen \
   -DCLANG_TABLEGEN=/home/foobar/local/bin/clang-tblgen -DLLVM_LIBDIR_SUFFIX=64 \
   -DLLVM_DEFAULT_TARGET_TRIPLE=aarch64-unknown-linux-gnu
$ make && make install

TIPS: Recommend not to use the build parallelization (might cause an oom error)
Cross-compile bcc

Build instructions - Almost same as llvm/clang

```
$ cd ${bcc_root_dir}
$ mkdir build && cd build
$ cmake .. -DCMAKE_CROSSCOMPILING=1 -DCMAKE_SYSROOT=${sysroot} \ 
  -DCMAKE_FIND_ROOT_PATH=${sysroot} -DCMAKE_FIND_ROOT_PATH_MODE_PROGRAM=NEVER \ 
  -DCMAKE_FIND_ROOT_PATH_MODE_LIBRARY=ONLY -DCMAKE_FIND_ROOT_PATH_MODE_INCLUDE=ONLY \ 
  -DCMAKE_INSTALL_PREFIX=${sysroot}/usr -DCMAKE_INSTALL_LIBDIR=lib64 \ 
  -DCMAKE_C_COMPILER=aarch64-poky-linux-gcc -DCMAKE_CXX_COMPILER=aarch64-poky-linux-g++ \ 
  -DCMAKE_C_FLAGS="-mabi=lp64 -march=armv8-a -mtune=cortex-a57.cortex-a53 -Wall --sysroot=${sysroot} \ 
  -DCMAKE_CXX_FLAGS="-mabi=lp64 -march=armv8-a -mtune=cortex-a57.cortex-a53 -Wall --sysroot=${sysroot} -std=c++11" \ 
  -DCMAKE_EXE_LINKER_FLAGS="--sysroot=${sysroot}" -DCMAKE_SHARED_LINKER_FLAGS="--sysroot=${sysroot}" \ 
  -DMULTILIB_LIB64=ON -DCMAKE_BUILD_TYPE=Release
$ make && make install
```
Port bcc – Fix 1. Multilib Support

Before Fix:

```bash
$ cmake .. -DCMAKE_CROSSCOMPILING=1 -DCMAKE_SYSROOT=${sysroot} \ 
    -DCMAKE_FIND_ROOT_PATH=${sysroot} -DCMAKE_FIND_ROOT_PATH_MODE_PROGRAM=NEVER \ 
 [...] 
-- Found BISON: /usr/bin/bison (found version "3.0.4")
-- Found FLEX: /usr/bin/flex (found version "2.6.0")
CMake Error at ${sysroot}/usr/lib/cmake/llvm/LLVMExports.cmake:1028 (message):
    The imported target "LLVMSupport" references the file
    "${sysroot}/usr/lib/libLLVMSupport.a"
[...]
```

Cause:

- Lack of multilib support
Fix 1: bcc/CMakeLists.txt

--- a/CMakeLists.txt
+++ b/CMakeLists.txt
@@ -11,6 +11,13 @@
  include(cmake/GetGitRevisionDescription.cmake)
  include(cmake/version.cmake)
  include(GNUInstallDirs)

+option(MULTILIB_LIB64 "use lib64 instead of lib to find library" OFF)
+if(MULTILIB_LIB64)
+  set_property(GLOBAL PROPERTY FIND_LIBRARY_USE_LIB64_PATHS TRUE)
+else()
+  set_property(GLOBAL PROPERTY FIND_LIBRARY_USE_LIB64_PATHS FALSE)
+endif()
+
+set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} ${CMAKE_CURRENT_SOURCE_DIR}/cmake)

if(NOT PYTHON_ONLY)

- Also need to add `-DMULTILIB_LIB64=ON` to cmake
Port bcc – Fix 2. Lack of Module

Before Fix:

```
# /usr/share/bcc/examples/cpp/HelloWorld
Could not create ExecutionEngine: Interpreter has not been linked in.
bpf: Invalid argument
Failed to load on_sys_clone: -1
```

Cause:

- Lack of the BPF module of clang
  - It had not built on the llvm/clang's compilation
  - Also bcc doesn't try to link it (even if it were exist)
Fix 2-1: llvm/clang compilation

- Remove `LLVM_TARGET_ARCH`, `LLVM_TARGETS_TO_BUILD` option
  - These options make the BPF module not to be built

- Cross compilation still works well because of `-DLLVM_DEFAULT_TARGET_TRIPLE=aarch64-unknown-linux-gnu` option
Port bcc – Fix 2. Lack of Module cont.

Fix 2–2 (workaround): bcc/src/cc/CMakeLists.txt

```
--- a/src/cc/CMakeLists.txt
+++ b/src/cc/CMakeLists.txt
@@ -56,6 +56,8 @@ endif()
   llvm_map_components_to_libnames(llvm_libs ${llvm_raw_libs})
   llvm_expand_dependencies(expanded_libs ${llvm_libs})

+set(llvm_raw_libs all)
+
+  # order is important
+  set(clang_libs ${libclangFrontend} ${libclangSerialization} ${libclangDriver} ${libclangParse}
+      ${libclangSema} ${libclangCodeGen} ${libclangAnalysis} ${libclangRewrite} ${libclangEdit}
```
Port bcc - Fix 3. asm macro

Before Fix:

```
# /usr/share/bcc/tools/execsnoop
LLVM ERROR: Inline asm not supported by this streamer because we don't have an asm parser for this target
```

Cause:

- Inline asm code exists in `linux/arch/arm64/include/asm/sysreg.h`

Ref: [https://www.mail-archive.com/iovisor-dev@lists.iovisor.org/msg00337.html](https://www.mail-archive.com/iovisor-dev@lists.iovisor.org/msg00337.html)
Port bcc – Fix 3. asm macro cont.

Fix 3 (workaround):  `linux/arch/arm64/include/asm/sysreg.h`

```c
--- a/arch/arm64/include/asm/sysreg.h
+++ b/arch/arm64/include/asm/sysreg.h
@@ -239,6 +239,7 @@

#include <linux/types.h>

+#ifndef __clang__
    asm(
        "       .irp num,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30\n        "        .equ .L__reg_num__x\num, \num\n    @ @ -253,6 +254,7 @@ asm(
        "       .inst 0xd5000000|(|\sreg)|(.L__reg_num__\rt)\n        "        .endm\n    );
+endif

/*
 * Unlike read_cpuid, calls to read_sysreg are never expected to be
```
Port bcc – Run

Now bcc works

```
# /usr/share/bcc/examples/hello_world.py
telnetd-1479 [001] d..1 90624.696430: : Hello, World!
```

But some scripts require extra fix

```
# /usr/share/bcc/tools/opensnoop
PID  COMM          FD  ERR  PATH
(no outputs)
```
Port bcc – Fix 4. opensnoop

Before Fix:

<table>
<thead>
<tr>
<th>PID</th>
<th>COMM</th>
<th>FD</th>
<th>ERR</th>
<th>PATH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(no outputs)

Cause:

- Probe point mismatch
  - AArch64 kernel uses `sys_openat()` instead of `sys_open()`
Port bcc - Fix 4. opensnoop cont.

Fix 4: bcc/tools/opensnoop.py

--- a/tools/opensnoop.py
+++ b/tools/opensnoop.py
@@ -17,6 +17,7 @@
    from bcc import BPF
    import argparse
    import ctypes as ct
+import subprocess

    # arguments
    examples = "examples:
      -68,7 +69,7 @@
    BPF_HASH(infotmp, u64, struct val_t);
    BPF_PERF_OUTPUT(events);

    -int trace_entry(struct pt_regs *ctx, const char __user *filename)
+trace_entry = "int trace_entry(struct pt_regs *ctx, \
+  const char __user *filename)"
+    +
+    +arch = subprocess.Popen(["uname", ",-m"], stdout=subprocess.PIPE).stdout.read()
+    +if arch[7] == "aarch64":
+    +    sys_open = "sys_openat"
+    +    trace_entry = "int trace_entry(struct pt_regs *ctx, \
+      int __user dirfd, const char __user *filename)"
+    +
+    +bpf_text = bpf_text.replace('TRACE_ENTRY', 'sys_open')
+    +
+    +if args.tid: # TID trumps PID
+    +    bpf_text = bpf_text.replace('FILTER', \
+      'if (tid != %s) { return 0; }' % args.tid)
@@ -111,6 +138,8 @@
+
    # initialize BPF
    b = BPF(text=bpf_text)
+    b.attach_kprobe(event=sys_open, fn_name=trace_entry)
+    b.attach_kretprobe(event=sys_open, fn_name=trace_return)
Port bcc – Run Again

- Works fine!

<table>
<thead>
<tr>
<th>PID</th>
<th>COMM</th>
<th>FD</th>
<th>ERR</th>
<th>PATH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1483</td>
<td>telnetd</td>
<td>6</td>
<td>0</td>
<td>/dev/ptmx</td>
</tr>
<tr>
<td>1483</td>
<td>telnetd</td>
<td>7</td>
<td>0</td>
<td>/etc/group</td>
</tr>
<tr>
<td>1703</td>
<td>telnetd</td>
<td>-1</td>
<td>2</td>
<td>/var/run/utmp</td>
</tr>
<tr>
<td>1703</td>
<td>telnetd</td>
<td>-1</td>
<td>2</td>
<td>/var/run/utmp</td>
</tr>
<tr>
<td>1703</td>
<td>telnetd</td>
<td>-1</td>
<td>2</td>
<td>/var/run/utmp</td>
</tr>
<tr>
<td>1703</td>
<td>telnetd</td>
<td>-1</td>
<td>2</td>
<td>/var/log/wtmp</td>
</tr>
<tr>
<td>1703</td>
<td>telnetd</td>
<td>0</td>
<td>0</td>
<td>/dev/pts/0</td>
</tr>
<tr>
<td>1703</td>
<td>telnetd</td>
<td>-1</td>
<td>2</td>
<td>/etc/issue.net</td>
</tr>
<tr>
<td>1703</td>
<td>bash</td>
<td>-1</td>
<td>2</td>
<td>/lib64/tls/aarch64/libdl.so.2</td>
</tr>
<tr>
<td>1703</td>
<td>bash</td>
<td>-1</td>
<td>2</td>
<td>/lib64/tls/libdl.so.2</td>
</tr>
<tr>
<td>1703</td>
<td>bash</td>
<td>-1</td>
<td>2</td>
<td>/lib64/aarch64/libdl.so.2</td>
</tr>
<tr>
<td>1703</td>
<td>bash</td>
<td>4</td>
<td>0</td>
<td>/lib64/libdl.so.2</td>
</tr>
<tr>
<td>1703</td>
<td>bash</td>
<td>-1</td>
<td>2</td>
<td>/lib64/tls/libc.so.6</td>
</tr>
<tr>
<td>1703</td>
<td>bash</td>
<td>4</td>
<td>0</td>
<td>/lib64/libc.so.6</td>
</tr>
<tr>
<td>1703</td>
<td>bash</td>
<td>4</td>
<td>0</td>
<td>/dev/tty</td>
</tr>
<tr>
<td>1703</td>
<td>bash</td>
<td>4</td>
<td>0</td>
<td>/proc/meminfo</td>
</tr>
<tr>
<td>1703</td>
<td>bash</td>
<td>-1</td>
<td>2</td>
<td>/etc/nsswitch.conf</td>
</tr>
</tbody>
</table>
Summary
Summary

• Let's GET STARTED with Dymanic Tracing Tools on AArch64!
  ○ bcc – Extra capabilities and Low overhead
  ○ SystemTap – Powerful but slow & unsafe, expecting BPF support
  ○ perf – Easy to start on AArch64, not so programable
  ○ perf-tools – Few dependencies

• What comes NEXT
  ○ BPF front-end with High-level Language
    ▪ bcc / perf extention?
    ▪ SystemTap BPF support?
    ▪ ply?
Future Works

- Solve remaining issues
  - bcc – Correct workarounds, Port much more scripts, Test
- Contributions
- Port tools to Linux v3.x/ARMv7 (maybe)
References

- http://www.brendangregg.com/
- https://www.linaro.org/blog/kprobes-event-tracing-armv8/
- https://clang.llvm.org/get_started.html
- http://llvm.org/docs/CMake.html
- http://llvm.org/docs/HowToCrossCompileLLVM.html
- https://www.mail-archive.com/iovisor-dev@lists.iovisor.org/msg00337.html
Thank You!

Any Questions?

https://github.com/hiroishii