SYSTEM WIDE TRACING AND PROFILING IN LINUX

Nadav Amit
Agenda

- System counters inspection
- Profiling with Linux perf tool
- Tracing using ftrace
Disclaimer

• Introductory level presentation

• We are not going to cover many tools

• We are not going to get deep into the implementation of the tools

• I am not an expert on many of the issues
Collect Statistics

- First step in analyzing the system behavior

- Option 1: Resource statistics tools
  - iostat, vmstat, netstat, ifstat
  - dstat

Examples:
- dstat
- dstat --udp --tcp --socket
- dstat --vm --aio
Watch system behavior online

• Option 2: Sample the counter
  • top
    • Use –H switch for thread specific
    • Use ‘f’ to choose additional fields: page faults, last used processor
    • Use ‘1’ to turn off cumulative mode

• iotop
  • Remember to run as sudoer
Fields Management for window 1:Def, whose current sort field is `%CPU`

Navigate with Up/Dn, Right selects for move then <Enter> or Left commits, 'd' or <Space> toggles display, 's' sets sort. Use 'q' or <Esc> to end!

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>Process Id</td>
</tr>
<tr>
<td>USER</td>
<td>Effective User Name</td>
</tr>
<tr>
<td>PR</td>
<td>Priority</td>
</tr>
<tr>
<td>NI</td>
<td>Nice Value</td>
</tr>
<tr>
<td>VIRT</td>
<td>Virtual Image (KiB)</td>
</tr>
<tr>
<td>RES</td>
<td>Resident Size (KiB)</td>
</tr>
<tr>
<td>SHR</td>
<td>Shared Memory (KiB)</td>
</tr>
<tr>
<td>S</td>
<td>Process Status</td>
</tr>
<tr>
<td>%CPU</td>
<td>CPU Usage</td>
</tr>
<tr>
<td>%MEM</td>
<td>Memory Usage (RES)</td>
</tr>
<tr>
<td>TIME+</td>
<td>CPU Time, hundredths</td>
</tr>
<tr>
<td>COMMAND</td>
<td>Command Name/Line</td>
</tr>
<tr>
<td>PPID</td>
<td>Parent Process pid</td>
</tr>
<tr>
<td>UID</td>
<td>Effective User Id</td>
</tr>
<tr>
<td>RUID</td>
<td>Real User Id</td>
</tr>
<tr>
<td>RUSER</td>
<td>Real User Name</td>
</tr>
<tr>
<td>SUID</td>
<td>Saved User Id</td>
</tr>
<tr>
<td>SUSER</td>
<td>Saved User Name</td>
</tr>
<tr>
<td>GID</td>
<td>Group Id</td>
</tr>
<tr>
<td>GROUP</td>
<td>Group Name</td>
</tr>
<tr>
<td>PGRP</td>
<td>Process Group Id</td>
</tr>
<tr>
<td>TTY</td>
<td>Controlling Tty</td>
</tr>
<tr>
<td>TPGID</td>
<td>Tty Process Grp Id</td>
</tr>
<tr>
<td>SID</td>
<td>Session Id</td>
</tr>
<tr>
<td>nTH</td>
<td>Number of Threads</td>
</tr>
<tr>
<td>P</td>
<td>Last Used Cpu (SMP)</td>
</tr>
<tr>
<td>TIME</td>
<td>CPU Time</td>
</tr>
<tr>
<td>SWAP</td>
<td>Swapped Size (KiB)</td>
</tr>
<tr>
<td>CODE</td>
<td>Code Size (KiB)</td>
</tr>
<tr>
<td>DATA</td>
<td>Data+Stack (KiB)</td>
</tr>
<tr>
<td>nMaj</td>
<td>Major Page Faults</td>
</tr>
<tr>
<td>nMin</td>
<td>Minor Page Faults</td>
</tr>
<tr>
<td>nDRT</td>
<td>Dirty Pages Count</td>
</tr>
<tr>
<td>WCHAN</td>
<td>Sleeping in Function</td>
</tr>
<tr>
<td>Flags</td>
<td>Task Flags &lt;sched.h&gt;</td>
</tr>
<tr>
<td>CGROUPS</td>
<td>Control Groups</td>
</tr>
<tr>
<td>SUPGIDS</td>
<td>Supp Groups IDs</td>
</tr>
<tr>
<td>SUPGRPS</td>
<td>Supp Groups Names</td>
</tr>
<tr>
<td>ENVIRON</td>
<td>Environment vars</td>
</tr>
<tr>
<td>TGID</td>
<td>Thread Group Id</td>
</tr>
<tr>
<td>vMj</td>
<td>Major Faults delta</td>
</tr>
<tr>
<td>vMn</td>
<td>Minor Faults delta</td>
</tr>
<tr>
<td>USED</td>
<td>Res+Swap Size (KiB)</td>
</tr>
<tr>
<td>nsIPC</td>
<td>IPC namespace Inode</td>
</tr>
<tr>
<td>nsMNT</td>
<td>MNT namespace Inode</td>
</tr>
<tr>
<td>nsNET</td>
<td>NET namespace Inode</td>
</tr>
<tr>
<td>nsPID</td>
<td>PID namespace Inode</td>
</tr>
<tr>
<td>nsUSER</td>
<td>USER namespace Inode</td>
</tr>
<tr>
<td>nsUTS</td>
<td>UTS namespace Inode</td>
</tr>
</tbody>
</table>
Inspect Raw Counters

• Option 3: Go to the raw counters
  • General
    • /proc/stat
    • /proc/meminfo
    • /proc/interrupts
  • Process specific
    • /proc/[pid]/statm – process memory
    • /proc/[pid]/stat – process execution times
    • /proc/[pid]/status – human readable
  • Device specific
    • /sys/block/[dev]/stat
    • /proc/dev/net
  • Hardware
    • smartctl
    • /proc/interrupts
### /sys/block/[dev]/stat

<table>
<thead>
<tr>
<th>Name</th>
<th>units</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read I/Os</td>
<td>requests</td>
<td>number of read I/Os processed</td>
</tr>
<tr>
<td>read merges</td>
<td>requests</td>
<td>number of read I/Os merged with in-queue I/O</td>
</tr>
<tr>
<td>read sectors</td>
<td>sectors</td>
<td>number of sectors read</td>
</tr>
<tr>
<td>read ticks</td>
<td>milliseconds</td>
<td>total wait time for read requests</td>
</tr>
<tr>
<td>write I/Os</td>
<td>requests</td>
<td>number of write I/Os processed</td>
</tr>
<tr>
<td>write merges</td>
<td>requests</td>
<td>number of write I/Os merged with in-queue I/O</td>
</tr>
<tr>
<td>write sectors</td>
<td>sectors</td>
<td>number of sectors written</td>
</tr>
<tr>
<td>write ticks</td>
<td>milliseconds</td>
<td>total wait time for write requests</td>
</tr>
<tr>
<td>in_flight</td>
<td>requests</td>
<td>number of I/Os currently in flight</td>
</tr>
<tr>
<td>io_ticks</td>
<td>milliseconds</td>
<td>total time this block device has been active</td>
</tr>
<tr>
<td>time_in_queue</td>
<td>milliseconds</td>
<td>total wait time for all requests</td>
</tr>
</tbody>
</table>

- Sometimes this description are insufficient and you should look at the code.
x86 Hardware Debugging/Profiling

- Debug registers (breakpoints)

- Performance Counters
  - Cores (some support anythread)
  - Uncore (shared subsystems, e.g. L3, QPI)
  - Offcore (e.g., snoop information, sw prefetching)

- Precise Event Based Sampling (PEBS)

- More
  - Last Branch Store
  - Last Branch Records
  - Last Exception Records
  - Non-precise Event Based Sampling

- Using this facilities directly is difficult (and usually privileged)
Linux Perf Tool

- Can instrument CPU performance counters, tracepoints, kprobes, and uprobes (dynamic tracing)
- Capable of lightweight profiling
- Included in the Linux kernel, under tools/perf
- Frequently updated and enhanced

- But it can be more friendly

- Alternatives
  - oprofile – similar to perf, reportedly less stable
  - gprof – rebuilds your code, changes behavior
Installing Perf Tool

- Install package linux-tools-generic
- If you use custom kernel, make tools/perf
  - There are many dependencies that add functionality
  - Some distributions do not build the package with all dependencies
  - Install libunwind for call-graph tracing before building

- Some counters are only accessible to privileged user
  - You can tweak /proc/sys/kernel/perf_event_paranoid:
    - -1 - Not paranoid at all
    - 0 - Disallow raw tracepoint access for unpriv
    - 1 - Disallow cpu events for unpriv
    - 2 - Disallow kernel profiling for unpriv
perf stat

- Lists the supported events

```
List of pre-defined events (to be used in -e):
  cpu-cycles OR cycles  [Hardware event]
  instructions         [Hardware event]
  cache-references     [Hardware event]
  cache-misses         [Hardware event]
  branch-instructions OR branches [Hardware event]
  branch-misses        [Hardware event]
  bus-cycles           [Hardware event]
  stalled-cycles-frontend OR idle-cycles-frontend [Hardware event]
  stalled-cycles-backend OR idle-cycles-backend   [Hardware event]
  ref-cycles           [Hardware event]
  cpu-clock            [Software event]
  task-clock           [Software event]
  page-faults OR faults [Software event]
  context-switches OR cs [Software event]
  cpu-migrations OR migrations [Software event]
  minor-faults         [Software event]
  major-faults         [Software event]
  alignment-faults     [Software event]
  emulation-faults     [Software event]
  dummy                [Software event]
  L1-dcache-loads      [Hardware cache event]
  L1-dcache-load-misses [Hardware cache event]
  L1-dcache-stores     [Hardware cache event]
  L1-dcache-store-misses [Hardware cache event]
```

perf stat (2)

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uncore_imc_1/cas_count_read/</td>
<td>[Kernel PMU event]</td>
</tr>
<tr>
<td>uncore_imc_1/cas_count_write/</td>
<td>[Kernel PMU event]</td>
</tr>
<tr>
<td>uncore_imc_1/clockticks/</td>
<td>[Kernel PMU event]</td>
</tr>
<tr>
<td>uncore_imc_2/cas_count_read/</td>
<td>[Kernel PMU event]</td>
</tr>
<tr>
<td>uncore_imc_2/cas_count_write/</td>
<td>[Kernel PMU event]</td>
</tr>
<tr>
<td>uncore_imc_2/clockticks/</td>
<td>[Kernel PMU event]</td>
</tr>
<tr>
<td>uncore_imc_3/cas_count_read/</td>
<td>[Kernel PMU event]</td>
</tr>
<tr>
<td>uncore_imc_3/cas_count_write/</td>
<td>[Kernel PMU event]</td>
</tr>
<tr>
<td>uncore_imc_3/clockticks/</td>
<td>[Kernel PMU event]</td>
</tr>
<tr>
<td>uncore_qpi_0/clockticks/</td>
<td>[Kernel PMU event]</td>
</tr>
<tr>
<td>uncore_qpi_0/drs_data/</td>
<td>[Kernel PMU event]</td>
</tr>
<tr>
<td>uncore_qpi_0/ncb_data/</td>
<td>[Kernel PMU event]</td>
</tr>
<tr>
<td>uncore_qpi_0/txi_flits_active/</td>
<td>[Kernel PMU event]</td>
</tr>
<tr>
<td>uncore_qpi_1/clockticks/</td>
<td>[Kernel PMU event]</td>
</tr>
<tr>
<td>uncore_qpi_1/drs_data/</td>
<td>[Kernel PMU event]</td>
</tr>
<tr>
<td>uncore_qpi_1/ncb_data/</td>
<td>[Kernel PMU event]</td>
</tr>
<tr>
<td>uncore_qpi_1/txi_flits_active/</td>
<td>[Kernel PMU event]</td>
</tr>
</tbody>
</table>

rNNN [Raw hardware event descriptor]
cpu/t1=v1[,t2=v2,t3 ...]/modifier [Raw hardware event descriptor]
(see 'man perf-list' on how to encode it)

mem:<addr>[:access] [Hardware breakpoint]

[ Tracepoints not available: Permission denied ]

- To get tracepoints and global counters use privileged user (e.g., sudo ./perf ...)
Monitoring Hardware Events using Perf

- There are common “hardware events”
  - Those are aliases to performance counters

- When in doubt (or need something else) sample the raw counters

- Note that their accuracy is questionable

- Choosing a counter
  - Intel Software Development Manual
  - libpfm4
### PERFORMANCE-MONITORING EVENTS

**Table 19-5. Non-Architectural Performance Events for 3rd Generation Intel® Core™ i7, i5, i3 Processors**

<table>
<thead>
<tr>
<th>Event Num.</th>
<th>Umask Value</th>
<th>Event Mask Mnemonic</th>
<th>Description</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>27H</td>
<td>08H</td>
<td>L2_STORE_LOCK_RQSTS.HIT_M</td>
<td>RFOs that hit cache lines in M state</td>
<td></td>
</tr>
<tr>
<td>27H</td>
<td>0FH</td>
<td>L2_STORE_LOCK_RQSTS.ALL</td>
<td>RFOs that access cache lines in any state</td>
<td></td>
</tr>
<tr>
<td>28H</td>
<td>01H</td>
<td>L2_L1D_WB_RQSTS.MISS</td>
<td>Not rejected writebacks that missed LLC.</td>
<td></td>
</tr>
<tr>
<td>28H</td>
<td>04H</td>
<td>L2_L1D_WB_RQSTS.HIT_M</td>
<td>Not rejected writebacks from L1D to L2 cache lines in M state.</td>
<td></td>
</tr>
<tr>
<td>28H</td>
<td>08H</td>
<td>L2_L1D_WB_RQSTS.HIT_E</td>
<td>Not rejected writebacks from L1D to L2 cache lines in E state.</td>
<td></td>
</tr>
<tr>
<td>28H</td>
<td>0FH</td>
<td>L2_L1D_WB_RQSTS.ALL</td>
<td>Not rejected writebacks from L1D to L2 cache lines in any state.</td>
<td></td>
</tr>
<tr>
<td>2EH</td>
<td>4FH</td>
<td>LONGEST_LAT_CACHE,REFERENCE</td>
<td>This event counts requests originating from the core that reference a cache line in the last level cache.</td>
<td>see Table 19-1</td>
</tr>
<tr>
<td>2EH</td>
<td>41H</td>
<td>LONGEST_LAT_CACHE,MISS</td>
<td>This event counts each cache miss condition for references to the last level cache.</td>
<td>see Table 19-1</td>
</tr>
<tr>
<td>3CH</td>
<td>00H</td>
<td>CPU_CLK_UNHALTED.THREAD_P</td>
<td>Counts the number of thread cycles while the thread is not in a halt state. The thread enters the halt state when it is running the HLT instruction. The core frequency may change from time to time due to power or thermal throttling.</td>
<td>see Table 19-1</td>
</tr>
<tr>
<td>3CH</td>
<td>01H</td>
<td>CPU_CLK_THREAD_UNHALTED.REF_XCLK</td>
<td>Increments at the frequency of XCLK (100 MHz) when not halted.</td>
<td>see Table 19-1</td>
</tr>
</tbody>
</table>
# Uncore Events

## Table 19-4. Non-Architectural Uncore Performance Events In the 4th Generation Intel® Core™ Processors

<table>
<thead>
<tr>
<th>Event Num.</th>
<th>Umask Value</th>
<th>Event Mask Mnemonic</th>
<th>Description</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>22H</td>
<td>01H</td>
<td>UNC_CBO_XSNP_RESPONSE_MISS</td>
<td>A snoop misses in some processor core.</td>
<td>Must combine with one of the umask values of 20H, 40H, 80H</td>
</tr>
<tr>
<td>22H</td>
<td>02H</td>
<td>UNC_CBO_XSNP_RESPONSE_INVAL</td>
<td>A snoop invalidates a non-modified line in some processor core.</td>
<td></td>
</tr>
<tr>
<td>22H</td>
<td>04H</td>
<td>UNC_CBO_XSNP_RESPONSE_HIT</td>
<td>A snoop hits a non-modified line in some processor core.</td>
<td></td>
</tr>
<tr>
<td>22H</td>
<td>08H</td>
<td>UNC_CBO_XSNP_RESPONSE_HITM</td>
<td>A snoop hits a modified line in some processor core.</td>
<td></td>
</tr>
<tr>
<td>22H</td>
<td>10H</td>
<td>UNC_CBO_XSNP_RESPONSE_INVAL_M</td>
<td>A snoop invalidates a modified line in some processor core.</td>
<td></td>
</tr>
<tr>
<td>22H</td>
<td>20H</td>
<td>UNC_CBO_XSNP_RESPONSE_EXTERNAL_FILTER</td>
<td>Filter on cross-core snoops initiated by this Cbox due to external snoop request.</td>
<td>Must combine with at least one of 01H, 02H, 04H, 08H, 10H</td>
</tr>
<tr>
<td>22H</td>
<td>40H</td>
<td>UNC_CBO_XSNP_RESPONSE_CORE_FILTER</td>
<td>Filter on cross-core snoops initiated by this Cbox due to processor core memory request.</td>
<td></td>
</tr>
<tr>
<td>22H</td>
<td>80H</td>
<td>UNC_CBO_XSNP_RESPONSE_VICTION_FILTER</td>
<td>Filter on cross-core snoops initiated by this Cbox due to L3 eviction.</td>
<td></td>
</tr>
<tr>
<td>34H</td>
<td>01H</td>
<td>UNC_CBO_CACHE_LOOKUP_M</td>
<td>L3 lookup request that access cache and found line in M-state.</td>
<td>Must combine with one of the umask values of 10H, 20H, 40H, 80H</td>
</tr>
<tr>
<td>34H</td>
<td>06H</td>
<td>UNC_CBO_CACHE_LOOKUP_ES</td>
<td>L3 lookup request that access cache and found line in E or S state.</td>
<td></td>
</tr>
<tr>
<td>34H</td>
<td>08H</td>
<td>UNC_CBO_CACHE_LOOKUP_I</td>
<td>L3 lookup request that access cache and found line in I-state.</td>
<td></td>
</tr>
</tbody>
</table>
Performance Counters Listing using libpfm

- Install the package libpfm4 sources
  - apt-get source libpfm4
  - make
  - cd examples
  - make
  - ./showevtinfo
# Monitoring Hardware Counters

- **libpfm** – running examples/showevtinfo

```bash
# Example code snippet

sudo perf stat -e r13c -a sleep 1
```

### Performance counter stats for 'system wide':

<table>
<thead>
<tr>
<th>Counter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>r13c</td>
<td>1,848,495</td>
</tr>
</tbody>
</table>

1.000977098 seconds time elapsed

**UMask = 01H**

**Event Select = 3CH**
Hardware Counters Limitations

- The system has limited number of hardware performance counters.
- If you exceed them, `perf` would arbitrate

```
./perf stat -e cache-misses -e cache-references -e cpu-cycles -e dTLB-loads -e iTLB-loads -a -- sleep 1
```

<table>
<thead>
<tr>
<th>Counter</th>
<th>Value</th>
<th>Percentage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache-misses</td>
<td>1,113,116</td>
<td>27.387%</td>
<td># of cache misses</td>
</tr>
<tr>
<td>cache-references</td>
<td>4,064,355</td>
<td>80.14%</td>
<td># of cache references</td>
</tr>
<tr>
<td>cpu-cycles</td>
<td>327,853,786</td>
<td>80.16%</td>
<td>Total cpu cycles</td>
</tr>
<tr>
<td>dTLB-loads</td>
<td>13,941,380</td>
<td>80.15%</td>
<td>Total dTLB loads</td>
</tr>
<tr>
<td>iTLB-loads</td>
<td>22,766</td>
<td>79.73%</td>
<td>Total iTLB loads</td>
</tr>
</tbody>
</table>

1.000972757 seconds time elapsed
Software Events

• Perdefined software events can be monitored

• `perf stat -e minor-faults -- ls`

```
Performance counter stats for 'ls':
  254  minor-faults
  0.001568812 seconds time elapsed
```
• `perf stat -e minor-faults -a -A -- ls`

- In many cases you would just run `sleep [X]` as your process for the duration you want to sample
- Use `-x`, for comma delimited file

Do not aggregate across CPUs
Event Modifiers

- You can tell when the event counter should take place

```
u - user-space counting
k - kernel counting
h - hypervisor counting
G - guest counting (in KVM guests)
H - host counting (not in KVM guests)
p - precise level
S - read sample value (PERF_SAMPLE_READ)
D - pin the event to the PMU
```

- `perf stat -e minor-faults:u -e minor-faults:k -- ls`

```
Performance counter stats for 'ls':

247 minor-faults:u
8 minor-faults:k
0.001965714 seconds time elapsed
```
Recording

• Recording and reporting is possible

• `perf record -e minor-faults -- ls`

• `perf report`
Profiling your Application

• For analysis which program/function should be optimized:
  • `perf report --sort comm,dso,symbol`

• Build your program with `-ggdb` flag to get debug information and being able to annotate it

• Don’t build with `-fomit-frame-pointer` (i.e., disable most optimizations)
Annotating the Source

- You can use `perf annotate [func]` or `perf report` to use annotation facilities.
- You can extract `vmlinux` and use `-k [vmlinux]`.
Annotating the Source (2)

• You can use extract-vmlinux script to extract vmlinux
  • Personally – It didn’t work for me

• If you want debugging of glibc
  • Install the debug package
  • Install the dev sources
Creating Trace Points

- You can create your own trace-points (but not likely get them upstream)
- See and include linux/tracepoint.h

```c
TRACE_EVENT(kvm_userspace_exit,
    TP_PROTO(__u32 reason, int errno),
    TP_ARGS(reason, errno),

    TP_STRUCT__entry(
        __field(    __u32,           reason           ),
        __field(    int,             errno           )
    ),

    TP_fast_assign(
        __entry->reason   = reason;
        __entry->errno    = errno;
    ),

    TP_printk("reason %s (%d)",
        __entry->errno < 0 ?
            (__entry->errno == -EINTR ? "restart" : "error") !=
            __print_symbolic(__entry->reason, kvm_trace_exit_reason),
        __entry->errno < 0 ? -__entry->errno : __entry->reason)
);

goto out;
r = kvm_arch_vcpu_ioctl_run(vcpu, vcpu->run);
trace_kvm_userspace_exit(vcpu->run->exit_reason, r);
break;
se KVM GET_REGS: {
```
Memory accesses sampling

- Memory access overhead
- `sudo ./perf mem record`
- `sudo ./perf mem report`
- Use `-g` to generate call-graph
Other perf features

The most commonly used perf commands are:

- **annotate**: Read perf.data (created by perf record) and display annotated code
- **archive**: Create archive with object files with build-ids found in perf.data file
- **bench**: General framework for benchmark suites
- **buildid-cache**: Manage build-id cache.
- **buildid-list**: List the buildids in a perf.data file
- **diff**: Read perf.data files and display the differential profile
- **evlist**: List the event names in a perf.data file
- **inject**: Filter to augment the events stream with additional information
- **kmem**: Tool to trace/measure kernel memory(slab) properties
- **kvm**: Tool to trace/measure kvm guest os
- **list**: List all symbolic event types
- **lock**: Analyze lock events
- **mem**: Profile memory accesses
- **record**: Run a command and record its profile into perf.data
- **report**: Read perf.data (created by perf record) and display the profile
- **sched**: Tool to trace/measure scheduler properties (latencies)
- **script**: Read perf.data (created by perf record) and display trace output
- **stat**: Run a command and gather performance counter statistics
- **test**: Runs sanity tests.
- **timechart**: Tool to visualize total system behavior during a workload
- **top**: System profiling tool.
- **trace**: strace inspired tool
- **probe**: Define new dynamic tracepoints
Guest events

- You can record guest events from the host
  - Only HW counters are supported

- First copy the guest symbols and modules to the host
  - `# ssh guest "cat /proc/kallsyms" > /tmp/guest.kallsyms`
  - `# ssh guest "cat /proc/modules" > /tmp/guest.modules`

- Then run:
  - `perf kvm --host --guest --guestkallsyms=/tmp/guest.kallsyms --guestmodules=/tmp/guest.modules record -a`
  - `perf kvm --guestkallsyms=/tmp/guest.kallsyms --guestmodules=/tmp/guest.modules --guest report`

```
Samples:  153  of event 'cycles', Event count (approx.):  49295839
 10.04%  : 2202 [guest.kernel.kallsyms] [g] _raw_spin_lock
  4.53%  : 2202 [guest.kernel.kallsyms] [g] native_write_msr_safe
  3.43%  : 2202 [guest.kernel.kallsyms] [g] _raw_spin_lock_irqsave
  3.37%  : 2202 [guest.kernel.kallsyms] [g] async_page_fault
  2.55%  : 2202 [guest.kernel.kallsyms] [g] reschedule_interrupt
  2.47%  : 2202 [guest.kernel.kallsyms] [g] _raw_spin_lock_irq
  2.45%  : 2202 [guest.kernel.kallsyms] [g] generic_exec_single
  2.38%  : 2202 [guest.kernel.kallsyms] [g] pvclock_clocksource_read
  2.10%  : 2202 [guest.kernel.kallsyms] [g] rcu_check_callbacks
```
Ftrace

- Tracing capability in the Linux kernel

- Enable by including in the config:
  - CONFIG_FUNCTION_TRACER=Y
  - CONFIG_FUNCTION_GRAPH_TRACER=Y
  - CONFIG_STACK_TRACE=Y
  - CONFIG_DYNAMIC_FTRACE=Y

- If you are lazy use `trace-cmd` wrapper application instead of everything shown in next slides

- You may need to mount the debugfs system
  - `mount -t debugfs nodev /sys/kernel/debug`
Tracers

- Go into tracing directory (/sys/kernel/debug/tracing)

```bash
cat available_tracers
blk mmiotrace function_graph wakeup_dl wakeup_rt wakeup function nop
```
nop tracer

- Hierarchy of events is based in `/sys/kernel/debug/tracing`
- You can enable a subset
  - For example `echo 1 > /sys/kernel/debug/tracing/events/irq`
- Then enable tracing
  - `echo 1 > /sys/kernel/debug/tracing/tracing_on`
- To clear the trace
  - `echo > /sys/kernel/debug/tracing/trace`
- To see the trace
  - `cat /sys/kernel/debug/tracing/trace`
  - Consuming read: `cat /sys/kernel/debug/tracing/trace_pipe`
echo 1 > /sys/kernel/debug/tracing/events/irq

```bash
# tracer: nop
#
# entries-in-buffer/entries-written: 15318/15318  #P:24

-----> irqs-off
  |-----> need-resched
  |      |-----> hardirq/softirq
  |      |      |-----> preempt-depth
  |      |      |      |-----> delay

| TASK-PID | CPU# | TIME | TIMESTAMP | FUNCTION
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bash-24796 [018] d.h. 36936.265283: softirq_raise: vec=1 [action=TIMER]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;idle&gt;-0 [000] d.h. 36936.265284: softirq_raise: vec=1 [action=TIMER]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;idle&gt;-0 [000] d.h. 36936.265285: softirq_raise: vec=9 [action=RCU]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bash-24796 [018] d.h. 36936.265286: softirq_raise: vec=9 [action=RCU]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;idle&gt;-0 [000] d.h. 36936.265286: softirq_raise: vec=7 [action=SCHED]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;idle&gt;-0 [000] s.s. 36936.265286: softirq_entry: vec=1 [action=TIMER]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bash-24796 [018] d.h. 36936.265289: softirq_raise: vec=7 [action=SCHED]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bash-24796 [018] s.s. 36936.265291: softirq_entry: vec=1 [action=TIMER]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;idle&gt;-0 [000] s.s. 36936.265291: softirq_exit: vec=1 [action=TIMER]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;idle&gt;-0 [000] s.s. 36936.265291: softirq_entry: vec=7 [action=SCHED]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;idle&gt;-0 [000] s.s. 36936.265292: softirq_exit: vec=7 [action=SCHED]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;idle&gt;-0 [000] s.s. 36936.265292: softirq_entry: vec=9 [action=RCU]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;idle&gt;-0 [000] s.s. 36936.265293: softirq_exit: vec=9 [action=RCU]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bash-24796 [018] s.s. 36936.265303: softirq_exit: vec=1 [action=TIMER]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bash-24796 [018] s.s. 36936.265303: softirq_entry: vec=7 [action=SCHED]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bash-24796 [018] s.s. 36936.265307: softirq_exit: vec=7 [action=SCHED]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bash-24796 [018] s.s. 36936.265307: softirq_entry: vec=9 [action=RCU]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bash-24796 [018] s.s. 36936.265308: softirq_exit: vec=9 [action=RCU]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;idle&gt;-0 [002] d.h. 36936.265466: irq_handler_entry: irq=130 name=eth0-tx-0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;idle&gt;-0 [002] d.h. 36936.265468: softirq_raise: vec=3 [action=NET_RX]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;idle&gt;-0 [002] d.h. 36936.265469: irq_handler_exit: irq=130 ret=handled</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Writing to the Trace from Kernel

- Use `trace_printk(…)` instead of `printk`

- Why not `printk`?
  - Changes scheduling
  - Slow
  - Harder to tell order with trace messages

- `trace_printk` will print the calling function on the stack
  - So it is inconsistent with the actual function if it is inlines
Snapshot; CPU Buffers

- Reading the buffer can cause events to be lost
- You can use snapshot instead:
  - `echo 1 > snapshot` (allocates spare buffer and clears it)
  - `cat snapshot`
  - If done – `echo 0 > snapshot` (free the buffer)

- Per CPU buffers exist in `per_cpu` directory
  - Note that their data is not interleaved in the global trace
uprobes

• perf probe -x /lib/x86_64-linux-gnu/libc.so.6 malloc

  Added new event:
  probe_libc:malloc (on 0x83590)

  You can now use it in all perf tools, such as:
  perf record -e probe_libc:malloc -aR sleep 1

• perf record -g -e probe_libc:malloc -aR sleep 10

• perf report

  Samples: 96K of event 'probe_libc:malloc', Event count (approx.): 96965
  +  97.28% command-not-fou  libc-2.19.so  [.] malloc
  +   2.63%      find  libc-2.19.so  [.] malloc
  +   0.04%  irqbalance  libc-2.19.so  [.] malloc
  +   0.03%       sleep  libc-2.19.so  [.] malloc
  +   0.01%  automount  libc-2.19.so  [.] malloc
  +   0.00%          cron  libc-2.19.so  [.] malloc

  Collect all raw counters
Function tracer

- `echo 'function > current_tracer'`

```
<idle>-0 [000] ..s. 39251.463281: arch_scale_smt_power <-update_group_power
<idle>-0 [000] ..s. 39251.463281: arch_scale_freq_power <-update_group_power
<idle>-0 [000] ..s. 39251.463281: target_load <-find_busiest_group
<idle>-0 [000] ..s. 39251.463281: idle_cpu <-find_busiest_group
<idle>-0 [000] ..s. 39251.463282: source_load <-find_busiest_group
<idle>-0 [000] ..s. 39251.463282: idle_cpu <-find_busiest_group
<idle>-0 [000] ..s. 39251.463282: msecs_to_jiffies <-rebalance_domains
<idle>-0 [000] ..s. 39251.463282: load_balance <-rebalance_domains
<idle>-0 [000] ..s. 39251.463282: idle_cpu <-load_balance
<idle>-0 [000] ..s. 39251.463282: find_busiest_group <-load_balance
<idle>-0 [000] ..s. 39251.463283: update_group_power <-find_busiest_group
<idle>-0 [000] ..s. 39251.463283: msecs_to_jiffies <-update_group_power
<idle>-0 [000] ..s. 39251.463283: target_load <-find_busiest_group
<idle>-0 [000] ..s. 39251.463283: idle_cpu <-find_busiest_group
<idle>-0 [000] ..s. 39251.463283: target_load <-find_busiest_group
<idle>-0 [000] ..s. 39251.463283: idle_cpu <-find_busiest_group
<idle>-0 [000] ..s. 39251.463283: source_load <-find_busiest_group
<idle>-0 [000] ..s. 39251.463284: idle_cpu <-find_busiest_group
<idle>-0 [000] ..s. 39251.463284: source_load <-find_busiest_group
<idle>-0 [000] ..s. 39251.463284: idle_cpu <-find_busiest_group
<idle>-0 [000] ..s. 39251.463284: source_load <-find_busiest_group
<idle>-0 [000] ..s. 39251.463284: idle_cpu <-find_busiest_group
```
Setting ftrace filter

- echo ‘*balance*’ > set_ftrace_filter
- cat trace
Tracing Specific Module

- `echo :mod:nfs > set_ftrace_filter`
- `cat trace`
Set Tracing Trigger

- echo > trace
- echo 0 > tracing_on
- echo nf_nat_ipv4_in:traceon > set_ftrace_filter
Function graph tracer

- `echo 'function_graph' > current_tracer`
Ftrace in userspace

- You can enable trace from userspace in the critical section by writing to 1 to tracing_on file
  - Examples on LWN

- Record userspace events in the trace
  - echo hello world > trace_marker
Controlling ftrace from the kernel

- You can disable/enable tracing in the kernel
  - `tracing_on()` and `tracing_off()`

- Dumping ftrace to console
  - `echo 1 > /proc/sys/kernel/ftrace_dump_on_oops`
  - Can also be set as kernel parameter (`ftrace_dump_on_oops`)
  - You can initiate dump using `ftrace_dump()`
  - `[ instead of `dump_stack()` ]`
Other useful features

- CPU mask for tracing (tracing_cpumask)
- Change buffer sizes (buffer_size_kb and buffer_size_total_kb)
Ftrace clocks

- **trace_clock** - change the clock used to order events
  - local: Per cpu clock but may not be synced across CPUs
  - global: Synced across CPUs but slows tracing down.
  - counter: Not a clock, but just an increment
  - uptime: Jiffy counter from time of boot
  - perf: Same clock that perf events use
  - x86-tsc: TSC cycle counter
References

- https://perf.wiki.kernel.org/index.php/Main_Page
- http://www.linux-kvm.org/page/Perf_events
- http://lwn.net/Articles/365835/
- http://lwn.net/Articles/366796/
- Documentation/trace/ftrace.txt
- Documentation/trace/uprobetracer.txt
- Documentation/trace/tracepoints.txt
Backup
<table>
<thead>
<tr>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTR_RETIRED.ANY_P</td>
</tr>
<tr>
<td>X87_OPS_RETIRED.ANY</td>
</tr>
<tr>
<td>BR_INST_RETIRED.MISPRED</td>
</tr>
<tr>
<td>SIMD_INST_RETIRED.ANY</td>
</tr>
<tr>
<td>MEM_LOAD_RETIRED.L1D_MISS</td>
</tr>
<tr>
<td>MEM_LOAD_RETIRED.L1D_LINE_MISS</td>
</tr>
<tr>
<td>MEM_LOAD_RETIRED.L2_MISS</td>
</tr>
<tr>
<td>MEM_LOAD_RETIRED.L2_LINE_MISS</td>
</tr>
<tr>
<td>MEM_LOAD_RETIRED.DTLB_MISS</td>
</tr>
</tbody>
</table>
Libpfm

- `sudo perf stat -e r13c -a sleep 1`