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

```
-----virtual-memory---- async
majpf minpf alloc  free| #aio
    0   240   147   279 |    0
    0    30    25    24 |    0
    0     0     1     1 |    0
    0     0     0     0 |    0
    0     0     1     1 |    0
```

dstat --vm --aio

```
--udp-- ----tcp-sockets---- ------sockets------
lis act|lis act syn tim clo|tot tcp udp raw frg
 25   0| 20   4   0   8   0|204  14  16   0   0
 25   0| 20   4   0   8   0|204  14  16   0   0
 25   0| 20   4   0   8   0|204  14  16   0   0
 25   0| 20   4   0   8   0|204  14  16   0   0
```

dstat --udp --tcp --socket

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

top

```
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!

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

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

	CPU0	CPU1	CPU2	CPU3	CPU4
0:	60	0	0	0	
3:	584	0	0	0	
4:	12	0	0	0	
8:	1	0	0	0	
9:	2	0	0	0	
10:	248	0	0	0	
22:	129	0	0	0	
23:	289	0	0	0	
104:	0	0	0	0	
105:	0	0	0	0	
106:	0	0	0	0	
107:	0	0	0	0	
108:	0	0	0	0	
109:	0	0	0	0	
110:	0	0	0	0	
111:	0	0	0	0	
112:	0	0	0	0	
113:	12497	0	301	0	1871
114:	2	0	0	0	
115:	2	0	0	0	
116:	2	0	0	0	

/proc/interrupts

/sys/block/[dev]/stat

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

- **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 dependecies
  - 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)

```
uncore_imc_1/cas_count_read/                       [Kernel PMU event]
uncore_imc_1/cas_count_write/                      [Kernel PMU event]
uncore_imc_1/clockticks/                           [Kernel PMU event]
uncore_imc_2/cas_count_read/                       [Kernel PMU event]
uncore_imc_2/cas_count_write/                      [Kernel PMU event]
uncore_imc_2/clockticks/                           [Kernel PMU event]
uncore_imc_3/cas_count_read/                       [Kernel PMU event]
uncore_imc_3/cas_count_write/                      [Kernel PMU event]
uncore_imc_3/clockticks/                           [Kernel PMU event]
uncore_qpi_0/clockticks/                           [Kernel PMU event]
uncore_qpi_0/drs_data/                             [Kernel PMU event]
uncore_qpi_0/ncb_data/                             [Kernel PMU event]
uncore_qpi_0/txl_flits_active/                     [Kernel PMU event]
uncore_qpi_1/clockticks/                           [Kernel PMU event]
uncore_qpi_1/drs_data/                             [Kernel PMU event]
uncore_qpi_1/ncb_data/                             [Kernel PMU event]
uncore_qpi_1/txl_flits_active/                     [Kernel PMU event]

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 Counters Listing in SDM

PERFORMANCE-MONITORING EVENTS

UMask = 0FH
Event Select = 27H

Table 19-5. Non-Architectural Performance Event... 3rd Generation Intel® Core™ i7, i5, i3 Pro...

Event Num.	Umask Value	Event Mask Mnemonic	Description	Comment
27H	08H	L2_STORE_LOCK_RQSTS.HIT_M	RFOs that hit cache lines in M state	
27H	0FH	L2_STORE_LOCK_RQSTS.ALL	RFOs that access cache lines in any state	
28H	01H	L2_L1D_WB_RQSTS.MISS	Not rejected writebacks that missed LLC.	
28H	04H	L2_L1D_WB_RQSTS.HIT_E	Not rejected writebacks from L1D to L2 cache lines in E state.	
28H	08H	L2_L1D_WB_RQSTS.HIT_M	Not rejected writebacks from L1D to L2 cache lines in M state.	
28H	0FH	L2_L1D_WB_RQSTS.ALL	Not rejected writebacks from L1D to L2 cache lines in any state.	
2EH	4FH	LONGEST_LAT_CACHE.REFERENCE	This event counts requests originating from the core that reference a cache line in the last level cache.	see Table 19-1
2EH	41H	LONGEST_LAT_CACHE.MISS	This event counts each cache miss condition for references to the last level cache.	see Table 19-1
3CH	00H	CPU_CLK_UNHALTED.THREAD_P	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.	see Table 19-1
3CH	01H	CPU_CLK_THREAD_UNHALTED.REF_XCLK	Increments at the frequency of XCLK (100 MHz) when not halted.	see Table 19-1

Uncore Events

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

Event Num.[1]	Umask Value	Event Mask Mnemonic	Description	Comment
22H	01H	UNC_CBO_XSNP_RESPONSE.MISS	A snoop misses in some processor core.	Must combine with one of the umask values of 20H, 40H, 80H
22H	02H	UNC_CBO_XSNP_RESPONSE.INVAL	A snoop invalidates a non-modified line in some processor core.	
22H	04H	UNC_CBO_XSNP_RESPONSE.HIT	A snoop hits a non-modified line in some processor core.	
22H	08H	UNC_CBO_XSNP_RESPONSE.HITM	A snoop hits a modified line in some processor core.	
22H	10H	UNC_CBO_XSNP_RESPONSE.INVAL_M	A snoop invalidates a modified line in some processor core.	
22H	20H	UNC_CBO_XSNP_RESPONSE.EXTERNAL_FILTER	Filter on cross-core snoops initiated by this Cbox due to external snoop request.	Must combine with at least one of 01H, 02H, 04H, 08H, 10H
22H	40H	UNC_CBO_XSNP_RESPONSE.XCORE_FILTER	Filter on cross-core snoops initiated by this Cbox due to processor core memory request.	
22H	80H	UNC_CBO_XSNP_RESPONSE.EVICTION_FILTER	Filter on cross-core snoops initiated by this Cbox due to L3 eviction.	
34H	01H	UNC_CBO_CACHE_LOOKUP.M	L3 lookup request that access cache and found line in M-state.	Must combine with one of the umask values of 10H, 20H, 40H, 80H
34H	06H	UNC_CBO_CACHE_LOOKUP.ES	L3 lookup request that access cache and found line in E or S state.	
34H	08H	UNC_CBO_CACHE_LOOKUP.I	L3 lookup request that access cache and found line in I-state.	

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

```
#-----------------------------
IDX      : 37748738
PMU name : ix86arch (Intel X86 architectural PMU)
Name     : UNHALTED_REFERENCE_CYCLES
Equiv    : None
Flags    : None
Desc     : count reference clock cycles while the clock signal on the specific core is running. The reference clock operates a
t a fixed frequency, irrespective of core frequency changes due to performance state transitions
Code     : 0x13c
Modif-00 : 0x00 : PMU : [k] : monitor at priv level 0 (boolean)
Modif-01 : 0x01 : PMU : [u] : monitor at priv level 1, 2, 3 (boolean)
Modif-02 : 0x02 : PMU : [e] : edge level (may require counter-mask >= 1) (boolean)
Modif-03 : 0x03 : PMU : [i] : invert (boolean)
Modif-04 : 0x04 : PMU : [c] : counter-mask in range [0-255] (integer)
Modif-05 : 0x05 : PMU : [t] : measure any thread (boolean)
#-----------------------------
```

UMask = 01H
Event Select = 3CH

- sudo perf stat -e r13c -a sleep 1

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

       1,848,495      r13c

     1.000977098 seconds time elapsed
```

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
```

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

       1,113,116      cache-misses              #   27.387 % of all cache refs      [80.14%]
       4,064,355      cache-references                                            [80.15%]
     327,853,786      cpu-cycles                [80.16%]
      13,941,380      dTLB-loads                                                  [80.15%]
          22,766      iTLB-loads                                                  [79.73%]

     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`

system-wide

Do not aggregate across CPUs

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

CPU0                    0      minor-faults
CPU1                    0      minor-faults
CPU2                    0      minor-faults
CPU3                    0      minor-faults
CPU4                    0      minor-faults
CPU5                    0      minor-faults
CPU6                    0      minor-faults
CPU7                    0      minor-faults
CPU8                    0      minor-faults
CPU9                    0      minor-faults
CPU10                   0      minor-faults
CPU11                   0      minor-faults
CPU12                   0      minor-faults
CPU13                   0      minor-faults
CPU14                 256      minor-faults
CPU15                  26      minor-faults
CPU16                   0      minor-faults
CPU17                   7      minor-faults
CPU18                   0      minor-faults
CPU19                   0      minor-faults
CPU20                   0      minor-faults
CPU21                   0      minor-faults
CPU22                   0      minor-faults
CPU23                   0      minor-faults

       0.001905321 seconds time elapsed
```

- 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

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 -g -- ls`
- `perf report`

call-graph recording

```
#
# Samples: 8  of event 'minor-faults'
# Event count (approx.): 587
#
# Overhead  Command      Shared Object                      Symbol
# ........  .......  .................  ..........................
#
    65.08%       ls  libc-2.19.so       [.] _nl_intern_locale_data
                 |
                 --- _nl_intern_locale_data

    26.41%       ls  ld-2.19.so         [.] _dl_load_cache_lookup
                 |
                 --- _dl_load_cache_lookup
                     0x5f6c636100312e6f

     5.45%       ls  ld-2.19.so         [.] dl_main
                 |
                 --- dl_main
                     _dl_sysdep_start

     2.04%       ls  ld-2.19.so         [.] _dl_start
                 |
                 --- _dl_start
                     0x7f6f5f5e02d8

                 ls  ld-2.19.so         [.] 0x00000000000012d0
                 |
                 --- 0x7f6f5f5e02d0

     0.34%       ls  [kernel.kallsyms]  [k] __clear_user
                 |
                 --- __clear_user
                     clear_user
                     padzero
                     load_elf_binary
                     search_binary_handler
                     do_execve_common.isra.27
                     sys_execve
                     stub_execve
                     0x7f745a5b1177

     0.17%       ls  [kernel.kallsyms]  [k] copy_user_generic_string
                 |
                 --- copy_user_generic_string
                     search_binary_handler
```

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]

```
             Disassembly of section .text:

             00000000004004ed <main>:
             int main()
             {
               push   %rbp
               mov    %rsp,%rbp
                     volatile int k;
                     for (int i = 0; i < 1000000; i++) {
               movl   $0x0,-0x4(%rbp)
             ↓ jmp    18
                             k = 100;
 10.00    d:┌─→movl   $0x64,-0x8(%rbp)
            │int main()
            │{
            │        volatile int k;
            │        for (int i = 0; i < 1000000; i++) {
            │  addl   $0x1,-0x4(%rbp)
         18:│  cmpl   $0xf423f,-0x4(%rbp)
 90.00      └──jle    d
                             k = 100;
                     }
                     return 0;
               mov    $0x0,%eax
             }
               pop    %rbp
             ← retq
```

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

```
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: {
```

Usage

Memory accesses sampling

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

```
# To display the perf.data header info, please use --header/--header-only options.
#
# Samples: 16  of event 'cpu/mem-loads/pp'
# Total weight : 149
# Sort order   : local_weight,mem,sym,dso,symbol_daddr,dso_daddr,snoop,tlb,locked
#
# Overhead       Samples  Local Weight             Memory access                   Symbol      Shared Object                        Data Symbol
# ........  ............  ............  ........................  ......................  .................  ...............................
#
    14.77%             1  22            L1 hit                    [k] handle_mm_fault     [kernel.kallsyms]  [k] 0xffff88041947b0b0
     9.40%             1  14            L1 hit                    [k] acpi_map_lookup     [kernel.kallsyms]  [k] acpi_ioremaps+0x0
     9.40%             1  14            L2 hit                    [k] perf_event_aux      [kernel.kallsyms]  [k] 0xffff88081934be28
     8.72%             1  13            L1 hit                    [k] prepare_creds       [kernel.kallsyms]  [k] 0xffff880419ba96f8
     8.05%             1  12            L1 hit                    [.] get_next_seq        libc-2.19.so       [.] 0x00007fff956615a0
     5.37%             1  8             L1 hit                    [k] perf_event_aux_ctx  [kernel.kallsyms]  [k] 0xffff8800c8c31dc0
     4.70%             1  7             L1 hit                    [.] get_next_seq        libc-2.19.so       [.] 0x00007fff956615e8
     4.70%             1  7             L1 hit                    [.] __strcoll_l         libc-2.19.so       [.] 0x00007fff956616e8
     4.70%             1  7             L1 hit                    [k] perf_event_aux_ctx  [kernel.kallsyms]  [k] 0xffff8800b1ec3878
     4.70%             1  7             L1 hit                    [k] vunmap_page_range   [kernel.kallsyms]  [k] 0xffff88041fce6d78
     4.70%             1  7             L1 hit                    [k] memcpy              [kernel.kallsyms]  [k] 0xffff8800c8c31bb8
     4.70%             1  7             L1 hit                    [k] acpi_map_lookup     [kernel.kallsyms]  [k] 0xffff88081923c680
     4.03%             1  6             L1 hit                    [.] _dl_catch_error     ld-2.19.so         [.] 0x00007fff95661838
     4.03%             1  6             L1 hit                    [.] _IO_getdelim        libc-2.19.so       [.] 0x00007fff95661cac
     4.03%             1  6             L1 hit                    [k] syscall_trace_leave [kernel.kallsyms]  [k] 0xffff8800c8c31f38
     4.03%             1  6             L1 hit                    [k] put_prev_task_fair  [kernel.kallsyms]  [k] 0xffff88041688db0c
```

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)

```
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

```
# tracer: nop
#
# entries-in-buffer/entries-written: 15318/15318   #P:24
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
            bash-24796 [018] d.h. 36936.265283: softirq_raise: vec=1 [action=TIMER]
          <idle>-0     [000] d.h. 36936.265284: softirq_raise: vec=1 [action=TIMER]
          <idle>-0     [000] d.h. 36936.265285: softirq_raise: vec=9 [action=RCU]
            bash-24796 [018] d.h. 36936.265286: softirq_raise: vec=9 [action=RCU]
          <idle>-0     [000] d.h. 36936.265286: softirq_raise: vec=7 [action=SCHED]
          <idle>-0     [000] ..s. 36936.265288: softirq_entry: vec=1 [action=TIMER]
            bash-24796 [018] d.h. 36936.265289: softirq_raise: vec=7 [action=SCHED]
            bash-24796 [018] ..s. 36936.265291: softirq_entry: vec=1 [action=TIMER]
          <idle>-0     [000] .Ns. 36936.265291: softirq_exit: vec=1 [action=TIMER]
          <idle>-0     [000] .Ns. 36936.265291: softirq_entry: vec=7 [action=SCHED]
          <idle>-0     [000] .Ns. 36936.265292: softirq_exit: vec=7 [action=SCHED]
          <idle>-0     [000] .Ns. 36936.265292: softirq_entry: vec=9 [action=RCU]
          <idle>-0     [000] .Ns. 36936.265293: softirq_exit: vec=9 [action=RCU]
            bash-24796 [018] .Ns. 36936.265303: softirq_exit: vec=1 [action=TIMER]
            bash-24796 [018] .Ns. 36936.265303: softirq_entry: vec=7 [action=SCHED]
            bash-24796 [018] .Ns. 36936.265307: softirq_exit: vec=7 [action=SCHED]
            bash-24796 [018] .Ns. 36936.265307: softirq_entry: vec=9 [action=RCU]
            bash-24796 [018] .Ns. 36936.265308: softirq_exit: vec=9 [action=RCU]
          <idle>-0     [002] d.h. 36936.265466: irq_handler_entry: irq=130 name=eth0-tx-0
          <idle>-0     [002] d.h. 36936.265468: softirq_raise: vec=3 [action=NET_RX]
          <idle>-0     [002] d.h. 36936.265469: irq_handler_exit: irq=130 ret=handled
```

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
```

Collect all raw counters

- 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
```

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

```
           sshd-24773 [019] d... 39314.558014: load_balance <-idle_balance
           sshd-24773 [019] d... 39314.558016: load_balance <-idle_balance
          <idle>-0    [019] d... 39314.558021: nohz_balance_enter_idle <-tick_nohz_stop_sched_tick
           bash-24796 [015] d... 39314.558169: idle_balance <-__schedule
           bash-24796 [015] d... 39314.558171: load_balance <-idle_balance
           bash-24796 [015] d... 39314.558173: load_balance <-idle_balance
          <idle>-0    [015] d... 39314.558178: nohz_balance_enter_idle <-tick_nohz_stop_sched_tick
       rcuos/14-23    [016] d... 39314.558239: idle_balance <-__schedule
       rcuos/14-23    [016] d... 39314.558240: load_balance <-idle_balance
       rcuos/14-23    [016] d... 39314.558243: load_balance <-idle_balance
          <idle>-0    [016] d... 39314.558247: nohz_balance_enter_idle <-tick_nohz_stop_sched_tick
      rcu_sched-8     [018] d... 39314.558271: idle_balance <-__schedule
      rcu_sched-8     [018] d... 39314.558273: load_balance <-idle_balance
      rcu_sched-8     [018] d... 39314.558275: load_balance <-idle_balance
 migration/14-143     [014] d... 39314.558293: idle_balance <-__schedule
 migration/14-143     [014] d... 39314.558296: load_balance <-idle_balance
 migration/14-143     [014] d... 39314.558298: load_balance <-idle_balance
          <idle>-0    [014] d... 39314.558302: nohz_balance_enter_idle <-tick_nohz_stop_sched_tick
       rcuos/15-24    [017] d... 39314.558574: idle_balance <-__schedule
       rcuos/15-24    [017] d... 39314.558575: load_balance <-idle_balance
       rcuos/15-24    [017] d... 39314.558578: load_balance <-idle_balance
```

Tracing Specific Module

- echo :mod:nfs > set_ftrace_filter
- cat trace

```
ls-25378 [014] .... 39690.410695: nfs_readdir <-iterate_dir
ls-25378 [014] .... 39690.410695: nfs_block_sillyrename <-nfs_readdir
ls-25378 [014] .... 39690.410695: nfs_attribute_cache_expired <-nfs_readdir
ls-25378 [014] .... 39690.410696: nfs_readdir_get_array <-nfs_readdir
ls-25378 [014] .... 39690.410696: cache_page_release.isra.22 <-nfs_readdir
ls-25378 [014] .... 39690.410697: nfs_readdir_get_array <-nfs_readdir
ls-25378 [014] .... 39690.410697: cache_page_release.isra.22 <-nfs_readdir
ls-25378 [014] .... 39690.410697: nfs_readdir_get_array <-nfs_readdir
ls-25378 [014] .... 39690.410698: cache_page_release.isra.22 <-nfs_readdir
ls-25378 [014] .... 39690.410698: nfs_readdir_get_array <-nfs_readdir
ls-25378 [014] .... 39690.410698: cache_page_release.isra.22 <-nfs_readdir
ls-25378 [014] .... 39690.410698: nfs_readdir_get_array <-nfs_readdir
ls-25378 [014] .... 39690.410699: cache_page_release.isra.22 <-nfs_readdir
ls-25378 [014] .... 39690.410699: nfs_readdir_get_array <-nfs_readdir
ls-25378 [014] .... 39690.410699: cache_page_release.isra.22 <-nfs_readdir
ls-25378 [014] .... 39690.410699: nfs_readdir_get_array <-nfs_readdir
ls-25378 [014] .... 39690.410700: cache_page_release.isra.22 <-nfs_readdir
ls-25378 [014] .... 39690.410700: nfs_readdir_get_array <-nfs_readdir
ls-25378 [014] .... 39690.410700: cache_page_release.isra.22 <-nfs_readdir
ls-25378 [014] .... 39690.410701: nfs_readdir_get_array <-nfs_readdir
ls-25378 [014] .... 39690.410701: cache_page_release.isra.22 <-nfs_readdir
```

Set Tracing Trigger

- echo > trace
- echo 0 > tracing_on
- echo nf_nat_ipv4_in:traceon > set_ftrace_filter

```
# tracer: function
#
# entries-in-buffer/entries-written: 199754/246299   #P:24
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
          <idle>-0     [008] ..s. 40233.920949: nf_nat_ipv4_in <-nf_iterate
          <idle>-0     [008] ..s. 40233.920950: nf_nat_ipv4_fn <-nf_nat_ipv4_in
          <idle>-0     [008] ..s. 40233.920950: nf_nat_packet <-nf_nat_ipv4_fn
          <idle>-0     [008] ..s. 40233.920951: ip_rcv_finish <-ip_rcv
          <idle>-0     [008] ..s. 40233.920951: tcp_v4_early_demux <-ip_rcv_finish
          <idle>-0     [008] ..s. 40233.920951: __inet_lookup_established <-tcp_v4_early_demux
          <idle>-0     [008] ..s. 40233.920951: inet_ehashfn <-__inet_lookup_established
          <idle>-0     [008] ..s. 40233.920952: ipv4_dst_check <-tcp_v4_early_demux
          <idle>-0     [008] ..s. 40233.920953: __skb_dst_set_noref <-tcp_v4_early_demux
```

Function graph tracer

- echo ‘function_graph’ > current_tracer

```
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |   |                     |   |   |   |
 19)                  |  set_all_modules_text_ro() {
 22)                  |  __hrtimer_start_range_ns() {
 10)                  |  __hrtimer_start_range_ns() {
 12)                  |  cpuidle_enter_state() {
  4)   0.433 us       |  menu_reflect();
  6)   0.307 us       |  ns_to_timeval();
  2)                  |  cpuidle_enter_state() {
 19)                  |    mutex_lock() {
 19)   0.080 us       |      _cond_resched();
 19)   1.189 us       |    }
 19)                  |    set_memory_ro() {
 19)                  |      change_page_attr_set_clr() {
 12)   0.160 us       |    ktime_get();
  2)   0.274 us       |    ktime_get();
 10)                  |    lock_hrtimer_base.isra.19() {
 19)                  |        vm_unmap_aliases() {
 22)                  |    lock_hrtimer_base.isra.19() {
 10)   0.193 us       |      _raw_spin_lock_irqsave();
 12)                  |    intel_idle() {
 19)                  |          __purge_vmap_area_lazy() {
 22)   0.137 us       |      _raw_spin_lock_irqsave();
```

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

PEBS

Event Name
INSTR_RETIRED.ANY_P
X87_OPS_RETIRED.ANY
BR_INST_RETIRED.MISPRED
SIMD_INST_RETIRED.ANY
MEM_LOAD_RETIRED.L1D_MISS
MEM_LOAD_RETIRED.L1D_LINE_MISS
MEM_LOAD_RETIRED.L2_MISS
MEM_LOAD_RETIRED.L2_LINE_MISS
MEM_LOAD_RETIRED.DTLB_MISS

Libpfm

- sudo perf stat -e r13c -a sleep 1