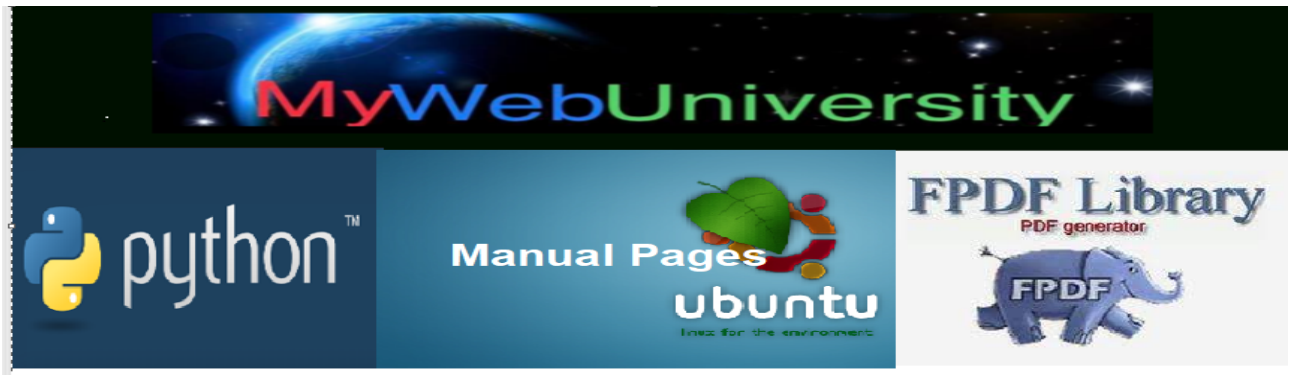

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'trace-cmd-sqlhist.1'

$ man trace-cmd-sqlhist.1

TRACE-CMD-SQLHIST(1) libtracefs Manual TRACE-CMD-SQLHIST(1)

NAME

trace-cmd-sqlhist - Use SQL language to create / show creation of tracefs histograms and synthetic events

SYNOPSIS

trace-cmd sqlhist [OPTIONS] [SQL-select-command]

DESCRIPTION

The trace-cmd sqlhist(1) will take an SQL like statement to create tracefs histograms and synthetic events that can perform various actions for various handling of the data.

The tracefs file system interfaces with the Linux tracing infrastructure that has various dynamic and static events through out the kernel. Each of these events can have a "histogram" attached to it, where the fields of the event will define the buckets of the histogram.

A synthetic event is a way to attach two separate events and use the fields and time stamps of those events to create a new dynamic event. This new dynamic event is call a synthetic event. The fields of each event can have simple calculations done on them where, for example, the delta between a field of one event to a field of the other event can be taken. This also works for the time stamps of the events where the time delta between the two events can also be extracted and placed into the synthetic event.

Other actions can be done from the fields of the events. A snapshot can be taken of the kernel ring buffer a variable used in the synthetic event creating hits a max, or simply changes.

The commands to create histograms and synthetic events are complex and not easy to remember.

trace-cmd sqlhist is used to convert SQL syntax into the commands needed to create the histogram or synthetic event.

The SQL-select-command is a SQL string defined by tracefs_sqlhist(3).

Note, this must be run as root (or sudo) as interacting with the tracefs directory requires root privilege, unless the -t option is given with a copy of the tracefs directory and its events.

OPTIONS

-n name

The name of the synthetic event to create. This event can then be used like any other event, and enabled via trace-cmd record(1).

-t tracefs-dir

In order to test this out as non root user, a copy of the tracefs directory can be used, and passing that directory with this option will allow the program to work. Obviously, -e will not work as non-root because it will not be able to execute.

```
# mkdir /tmp/tracing
# cp -r /sys/kernel/tracing/events /tmp/tracing
# exit
$ trace-cmd sqlhist -t /tmp/tracing ...
```

-e

Not only display the commands to create the histogram, but also execute them. This requires root privilege.

-f file

Instead of reading the SQL commands from the command line, read them from file. If file is - then read from standard input.

-m var

Do the given action when the variable var hits a new maximum. This can not be used with -c. The var must be defined in the SQL-select-command.

-c var

Do the given action when the variable var changes its value. This can not be used with -m. The var must be defined in the SQL-select-command.

-s

Perform a snapshot instead of calling the synthetic event.

-T

Perform both a snapshot and trace the synthetic event.

-S fields[,fields]

Save the given fields. The fields must be fields of the "end" event given in the

SQL-select-command

-B instance

For simple statements that only produce a histogram, the instance given here will be

where the histogram will be created. This is ignored for full synthetic event creation,

as sythetic events have a global affect on all tracing instances, where as, histograms

only affect a single instance.

EXAMPLES

As described above, for testing purposes, make a copy of the event directory:

$ mkdir /tmp/tracing

$ sudo cp -r /sys/kernel/tracing/events /tmp/tracing/

$ sudo chmod -R 0644 /tmp/tracing/

For an example of simple histogram output using the copy of the tracefs directory.

$ trace-cmd sqlhist -t /tmp/tracing/ 'SELECT CAST(call_site as SYM-OFFSET), bytes_req, CAST(bytes_alloc AS _COUNTER_) FROM kmalloc'

Produces the output:

echo 'hist:keys=call_site.sym-offset,bytes_req:vals=bytes_alloc' > /sys/kernel/tracing/events/kmem/kmalloc/trigger

Which could be used by root:

echo 'hist:keys=call_site.sym-offset,bytes_req:vals=bytes_alloc' > /sys/kernel/tracing/events/kmem/kmalloc/trigger

cat /sys/kernel/tracing/events/kmem/kmalloc/hist

event histogram

#

trigger info: hist:keys=call_site.sym-offset,bytes_req:vals=hitcount,bytes_alloc:sort=hitcount:size=2048 [active]

#

{ call_site: [ffffffff813f8d8a] load_elf_phdrs+0x4a/0xb0 , bytes_req: 728 } hitcount: 1 bytes_alloc: 1024

{ call_site: [ffffffffc0c69e74] nf_ct_ext_add+0xd4/0x1d0 [nf_conntrack] , bytes_req: 128 } hitcount: 1 bytes_alloc: 128

{ call_site: [ffffffff818355e6] dma_resv_get_fences+0xf6/0x440 , bytes_req: 8 } hitcount: 1 bytes_alloc: 8

{ call_site: [ffffffffc06dc73f] intel_gt_get_buffer_pool+0x15f/0x290 [i915] , bytes_req: 424 } hitcount: 1 bytes_alloc: 512

{ call_site: [ffffffff813f8d8a] load_elf_phdrs+0x4a/0xb0 , bytes_req: 616 } hitcount: 1 bytes_alloc: 1024

{ call_site: [ffffffff8161a44c] __sg_alloc_table+0x11c/0x180 , bytes_req: 32 } hitcount: 1 bytes_alloc: 32

{ call_site: [ffffffffc070749d] shmem_get_pages+0xad/0x5d0 [i915] , bytes_req: 16 } hitcount: 1 bytes_alloc: 16

{ call_site: [ffffffffc07507f5] intel_framebuffer_create+0x25/0x60 [i915] , bytes_req: 408 } hitcount: 1 bytes_alloc: 512

{ call_site: [ffffffffc06fc20f] eb_parse+0x34f/0x910 [i915] , bytes_req: 408 } hitcount: 1 bytes_alloc: 512

{ call_site: [ffffffffc0700ebd] i915_gem_object_get_pages_internal+0x5d/0x270 [i915] , bytes_req: 16 } hitcount: 1 bytes_alloc: 16

{ call_site: [ffffffffc0771188] intel_frontbuffer_get+0x38/0x220 [i915] , bytes_req: 400 } hitcount: 1 bytes_alloc: 512

{ call_site: [ffffffff8161a44c] __sg_alloc_table+0x11c/0x180 , bytes_req: 128 } hitcount: 1 bytes_alloc: 128

{ call_site: [ffffffff813f8f45] load_elf_binary+0x155/0x1680 , bytes_req: 28 } hitcount: 1 bytes_alloc: 32

{ call_site: [ffffffffc07038c8] __assign_mmap_offset+0x208/0x3d0 [i915] , bytes_req: 288 } hitcount: 1 bytes_alloc: 512

{ call_site: [ffffffff813737b2] alloc_bprm+0x32/0x2f0 , bytes_req: 416 } hitcount: 1 bytes_alloc: 512

{ call_site: [ffffffff813f9027] load_elf_binary+0x237/0x1680 , bytes_req: 64 } hitcount: 1 bytes_alloc: 64

{ call_site: [ffffffff8161a44c] __sg_alloc_table+0x11c/0x180 , bytes_req: 64 } hitcount: 1 bytes_alloc: 64

{ call_site: [ffffffffc040ffe7] drm_vma_node_allow+0x27/0xe0 [drm] , bytes_req: 40 } hitcount: 2 bytes_alloc: 128

{ call_site: [ffffffff813cda98] __do_sys_timerfd_create+0x58/0x1c0 , bytes_req: 336 } hitcount: 2 bytes_alloc: 1024

{ call_site: [ffffffff818355e6] dma_resv_get_fences+0xf6/0x440 , bytes_req: 40 } hitcount: 2

bytes_alloc: 128

{ call_site: [ffffffff8139b75a] single_open+0x2a/0xa0 , bytes_req: 32 } hitcount: 2

bytes_alloc: 64

{ call_site: [ffffffff815df715] bio_kmalloc+0x25/0x80 , bytes_req: 136 } hitcount: 2

bytes_alloc: 384

{ call_site: [ffffffffc071e5cd] i915_vma_work+0x1d/0x50 [i915] , bytes_req: 416 } hitcount: 3

bytes_alloc: 1536

{ call_site: [ffffffff81390d0d] alloc_fdtable+0x4d/0x100 , bytes_req: 56 } hitcount: 3

bytes_alloc: 192

{ call_site: [ffffffffc06ff65f] i915_gem_do_execbuffer+0x158f/0x2440 [i915] , bytes_req: 16 } hitcount:

4 bytes_alloc: 64

{ call_site: [ffffffff8137713c] alloc_pipe_info+0x5c/0x230 , bytes_req: 384 } hitcount: 5

bytes_alloc: 2560

{ call_site: [ffffffff813771b4] alloc_pipe_info+0xd4/0x230 , bytes_req: 640 } hitcount: 5

bytes_alloc: 5120

{ call_site: [ffffffff81834cdb] dma_resv_list_alloc+0x1b/0x40 , bytes_req: 40 } hitcount: 6

bytes_alloc: 384

{ call_site: [ffffffff81834cdb] dma_resv_list_alloc+0x1b/0x40 , bytes_req: 56 } hitcount: 9

bytes_alloc: 576

{ call_site: [ffffffff8120086e] tracing_map_sort_entries+0x9e/0x3e0 , bytes_req: 24 } hitcount:

60 bytes_alloc: 1920

Totals:

Hits: 122

Entries: 30

Dropped: 0

Note, although the examples use uppercase for the SQL keywords, they do not have to be.

SELECT could also be select or even sElEcT.

By using the full SQL language, synthetic events can be made and processed. For example,

using trace-cmd sqlhist along with trace-cmd record(1), wake up latency can be recorded by

creating a synthetic event by attaching the sched_waking and the sched_switch events.

trace-cmd sqlhist -n wakeup_lat -e -T -m lat 'SELECT end.next_comm AS comm, (end.TIMESTAMP_USECS -

start.TIMESTAMP_USECS) AS lat FROM ' \

'sched_waking AS start JOIN sched_switch AS end ON start.pid = end.next_pid WHERE end.next_prio == 100 &&

end.next_comm == "cyclictest"'

```
# trace-cmd start -e all -e wakeup_lat -R stacktrace
# cyclictest -l 1000 -p80 -i250  -a -t -q -m -d 0 -b 1000 --tracemark
# trace-cmd show -s | tail -30
     <idle>-0       [002] dNh4 23454.902246: sched_wakeup: comm=cyclictest pid=12272 prio=120 target_cpu=002
     <idle>-0       [005] ...1 23454.902246: cpu_idle: state=4294967295 cpu_id=5
     <idle>-0       [007] d..1 23454.902246: cpu_idle: state=0 cpu_id=7
     <idle>-0       [002] dNh1 23454.902247: hrtimer_expire_exit: hrtimer=0000000037956dc2
     <idle>-0       [005] d..1 23454.902248: cpu_idle: state=0 cpu_id=5
     <idle>-0       [002] dNh1 23454.902248: write_msr: 6e0, value 4866ce957272
     <idle>-0       [006] ...1 23454.902248: cpu_idle: state=4294967295 cpu_id=6
     <idle>-0       [002] dNh1 23454.902249: local_timer_exit: vector=236
     <idle>-0       [006] d..1 23454.902250: cpu_idle: state=0 cpu_id=6
     <idle>-0       [002] .N.1 23454.902250: cpu_idle: state=4294967295 cpu_id=2
     <idle>-0       [002] dN.1 23454.902251: rcu_utilization: Start context switch
     <idle>-0       [002] dN.1 23454.902252: rcu_utilization: End context switch
     <idle>-0       [001] ...1 23454.902252: cpu_idle: state=4294967295 cpu_id=1
     <idle>-0       [002] dN.3 23454.902253: prandom_u32: ret=3692516021
     <idle>-0       [001] d..1 23454.902254: cpu_idle: state=0 cpu_id=1
         <idle>-0       [002] d..2 23454.902254: sched_switch: prev_comm=swapper/2 prev_pid=0 prev_prio=120
prev_state=R ==> next_comm=cyclictest next_pid=12275 next_prio=19
     <idle>-0       [002] d..4 23454.902256: wakeup_lat: next_comm=cyclictest lat=17
     <idle>-0       [002] d..5 23454.902258: <stack trace>
 => trace_event_raw_event_synth
 => action_trace
 => event_hist_trigger
 => event_triggers_call
 => trace_event_buffer_commit
 => trace_event_raw_event_sched_switch
 => __traceiter_sched_switch
 => __schedule
 => schedule_idle
 => do_idle
```

=> cpu_startup_entry

=> secondary_startup_64_no_verify

Here?s the options for above example explained:

-n wakeup_lat

Name the synthetic event to use wakeup_lat.

-e

Execute the commands that are printed.

-T

Perform both a trace action and then a snapshot action (swap the buffer into the static snapshot buffer).

-m lat

Trigger the actions whenever lat hits a new maximum value.

Now a breakdown of the SQL statement:

'SELECT end.next_comm AS comm, (end.TIMESTAMP_USECS - start.TIMESTAMP_USECS) AS lat FROM ' \

'sched_waking AS start JOIN sched_switch AS end ON start.pid = end.next_pid WHERE end.next_prio < 100 && end.next_comm == "cyclictest"'

end.next_comm AS comm

Save the sched_switch field next_comm and place it into the comm field of the wakeup_lat synthetic event.

(end.TIMESTAMP_USECS - start.TIMESTAMP_USECS) AS lat

Take the delta of the time stamps from the sched_switch event and the sched_waking event. As time stamps are usually recorded in nanoseconds, TIMESTAMP would give the full nanosecond time stamp, but here, the TIMESTAMP_USECS will truncate it into microseconds. The value is saved in the variable lat, which will also be recorded in the synthetic event.

FROM sched_waking AS start JOIN sched_switch AS end ON start.pid = end.next_pid

Create the synthetic event by joining sched_waking to sched_switch, matching the sched_waking pid field with the sched_switch next_pid field. Also make start an alias for sched_waking and end an alias for sched_switch which then an use start and end as a subsitute for sched_waking and sched_switch respectively through out the rest of the SQL statement.

WHERE end.next_prio < 100 && end.next_comm == "cyclictest"

Filter the logic where it executes only if sched_waking next_prio field is less than

100. (Note, in the Kernel, priorities are inverse, and the real-time priorities are

represented from 0-100 where 0 is the highest priority). Also only trace when the

next_comm (the task scheduling in) of the sched_switch event has the name "cyclictest".

For the trace-cmd(3) command:

trace-cmd start -e all -e wakeup_lat -R stacktrace

trace-cmd start

Enables tracing (does not record to a file).

-e all

Enable all events

-e wakeup_lat -R stacktrace

have the "wakeup_lat" event (our synthetic event) enable the stacktrace trigger, were

for every instance of the "wakeup_lat" event, a kernel stack trace will be recorded in

the ring buffer.

After calling cyclictest (a real-time tool to measure wakeup latency), read the snapshot

buffer.

trace-cmd show -s

trace-cmd show reads the kernel ring buffer, and the -s option will read the snapshot

buffer instead of the normal one.

<idle>-0 [002] d..4 23454.902256: wakeup_lat: next_comm=cyclictest lat=17

We see on the "wakeup_lat" event happened on CPU 2, with a wake up latency 17 microseconds.

This can be extracted into a trace.dat file that trace-cmd(3) can read and do further

analysis, as well as kernelshark.

trace-cmd extract -s

trace-cmd report --cpu 2 | tail -30

<idle>-0 [002] 23454.902238: prandom_u32: ret=1633425088

<idle>-0 [002] 23454.902239: sched_wakeup: cyclictest:12275 [19] CPU:002

<idle>-0 [002] 23454.902241: hrtimer_expire_exit: hrtimer=0xffffbbd68286fe60

<idle>-0 [002] 23454.902241: hrtimer_cancel: hrtimer=0xffffbbd6826efe70

<idle>-0 [002] 23454.902242: hrtimer_expire_entry: hrtimer=0xffffbbd6826efe70 now=23455294430750 function=hrtimer_wakeup/0x0

<idle>-0 [002] 23454.902243: sched_waking: comm=cyclictest pid=12272 prio=120 target_cpu=002

<idle>-0 [002] 23454.902244: prandom_u32: ret=1102749734

<idle>-0 [002] 23454.902246: sched_wakeup: cyclictest:12272 [120] CPU:002

```
            <idle>-0     [002] 23454.902247: hrtimer_expire_exit:  hrtimer=0xffffbbd6826efe70
            <idle>-0     [002] 23454.902248: write_msr:            6e0, value 4866ce957272
            <idle>-0     [002] 23454.902249: local_timer_exit:     vector=236
            <idle>-0     [002] 23454.902250: cpu_idle:             state=4294967295 cpu_id=2
            <idle>-0     [002] 23454.902251: rcu_utilization:      Start context switch
            <idle>-0     [002] 23454.902252: rcu_utilization:      End context switch
            <idle>-0     [002] 23454.902253: prandom_u32:          ret=3692516021
            <idle>-0     [002] 23454.902254: sched_switch:         swapper/2:0 [120] R ==> cyclictest:12275 [19]
            <idle>-0     [002] 23454.902256: wakeup_lat:           next_comm=cyclictest lat=17
            <idle>-0     [002] 23454.902258: kernel_stack:         <stack trace >
        => trace_event_raw_event_synth (ffffffff8121a0db)
        => action_trace (ffffffff8121e9fb)
        => event_hist_trigger (ffffffff8121ca8d)
        => event_triggers_call (ffffffff81216c72)
        => trace_event_buffer_commit (ffffffff811f7618)
        => trace_event_raw_event_sched_switch (ffffffff8110fda4)
        => __traceiter_sched_switch (ffffffff8110d449)
        => __schedule (ffffffff81c02002)
        => schedule_idle (ffffffff81c02c86)
        => do_idle (ffffffff8111e898)
        => cpu_startup_entry (ffffffff8111eba9)
        => secondary_startup_64_no_verify (ffffffff81000107)
```

SEE ALSO

trace-cmd(1), tracefs_sqlhist(3)

AUTHOR

Written by Steven Rostedt, <rostedt@goodmis.org[1]>

RESOURCES

https://git.kernel.org/pub/scm/utils/trace-cmd/trace-cmd.git/

COPYING

NOTES

1. rostedt@goodmis.org

mailto:rostedt@goodmis.org