SONY

On ARM

Measuring Function Duration with Ftrace

By Tim Bird
Sony Corporation of America
<tim.bird (at) am.sony.com>

Outline

- Introduction to Ftrace
- Adding function graph tracing to ARM
- Duration Filtering
  - Trace coverage duration analysis
- Measuring kernel boot
- Post-trace analysis tools
- Performance impact
- Resources

Introduction to Ftrace

- What is Ftrace?
- Overview of operation
  - Instrumentation
  - Runtime operation
  - Data capture
  - Trace log output
- Function duration tracing

What is Ftrace?

- Ftrace is the first generic tracing system to get mainlined (Hurray!!)
  - Mainlined in 2.6.27
  - Derived from RT-preempt latency tracer
- Provides a generic framework for tracing
  - Infrastructure for defining tracepoints
  - Ability to register different kinds of tracers
  - Specialized data structure (ring buffer) for trace data storage

Overview of FTrace Operation

- Instrumentation
  - Explicit
    - Tracepoints defined by declaration
    - Calls to trace handler written in source code
  - Implicit
    - Automatically inserted by compiler
      - Uses gcc '-pg' option
    - Inserts call to 'mcount' in each function prologue
    - Easy to maintain – no source code modifications
    - Only practical way to maintain 20,000+ tracepoints

mcount Routine

- 'mcount' is called by every kernel function
  - Except inlines and a few special functions
- Must be a low-overhead routine
- Incompatible with some compiler optimizations
  - E.g. cannot omit frame-pointers on ARM
  - Compiler disables some optimizations automatically
  - Works with ARM EABI
  - Analysis of assembly indicates that mcount callers have well-defined frames
- Misc note:
  - New mcount routine (_gnu_mcount) is coming

Code to Call mcount

```
00000570 <sys_sync>:
570: e1a0c00d mov ip, sp
574: e92dd800 stmdb sp!, {fp, ip, lr, pc}
578: e24cb004 sub fp, ip, #4 ; 0x4



57c: e3a00001 mov r0, #1 ; 0x1
580: ebffffa0 bl 408 <do_sync>
584: e3a00000 mov r0, #0 ; 0x0
588: e89da800 ldmia sp, {fp, sp, pc}
```

```
00000570 <sys_sync>:
570: e1a0c00d mov ip, sp
574: e92dd800 stmdb sp!, {fp, ip, lr, pc}
578: e24cb004 sub fp, ip, #4 ; 0x4
57c: e1a0c00e mov ip, lr
580: ebfffffe bl 0 <mcount>
584: 00000028 andeq r0, r0, r8, lsr #32
588: e3a00001 mov r0, #1 ; 0x1
58c: ebffff9d bl 408 <do_sync>
590: e3a00000 mov r0, #0 ; 0x0
594: e89da800 ldmia sp, {fp, sp, pc}
```

Trace setup at run-time

- Pseudo-files in debugfs
  - e.g. mount debugfs –t debugfs /debug
- Select a tracer
  - e.g. echo function_duration >current_tracer
- Set tracing parameters
  - e.g. echo 100 >tracing_threshhold
  - echo duration-proc >trace_options

Trace Data Capture

- Ring Buffer
  - Specialized structure for collecting trace data
    - Manages buffer as list of pages
  - Latest version is lockless for writing
    - Ability to atomically reserve space for an event
  - Automatic timestamp management
  - Per-cpu buffers
    - Avoids requiring cross-CPU synchronization
    - Also avoids cache collisions
      - Very important for performance

Trace Output

- Output is human readable text
  - No special tools required to collect trace data
- Examples:
  - cat trace
    - Returns EOF at end of trace data
  - cat trace_pipe | grep foo >log.txt
    - Blocks at end of trace data
- Quick enable/disable
  - echo 0 >tracing_enabled

Ring Buffer Operations

- ring_buffer_lock_reserve
  - Atomically reserve space in buffer
- ring_buffer_event_data
  - Get pointer to place to fill with data
- ring_buffer_unlock_commit
  - Commit event data
- ring_buffer_discard_commit
  - Discard reserved data space

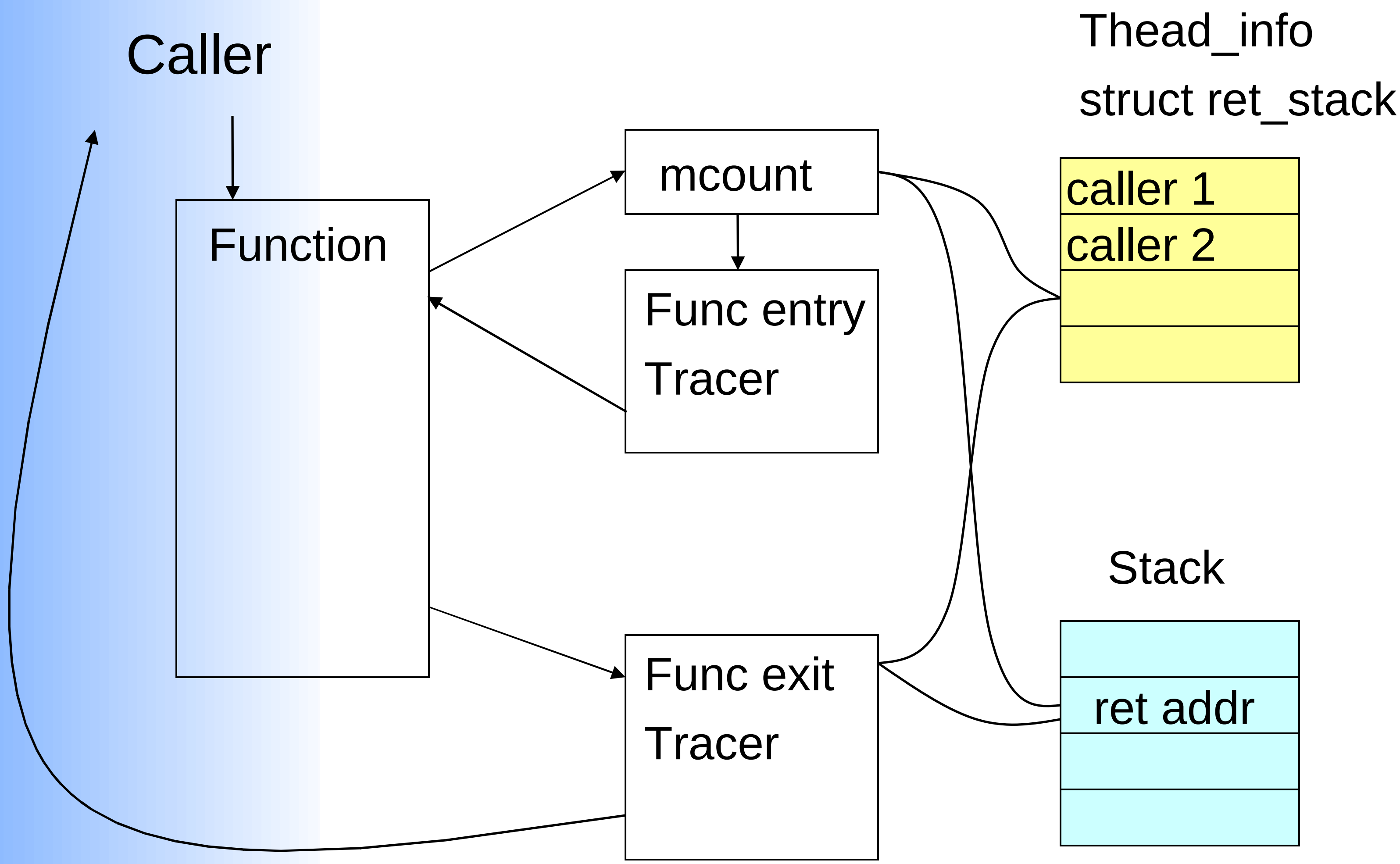
Function duration tracing

- Traces function entry and exit
- What is it good for?
  - See relationship between functions
    - Is a GREAT way to learn about kernel
    - Find unexpected/abnormal code paths
  - Measure function duration
    - Find long latencies and performance problems
- But, the -pg option only instruments function entry

Hooking function exit

- Normal ‘function’ tracer just traces function entry capture
- To capture function exit, a trampoline is used
  - mcount:
    - Saves real return address
    - Replaces return address with address of trampoline
  - In exit tracer, return to the real return address

Diagram of Trampoline



Why Filter by Duration?

- To extend the capture duration time
  - By reducing, at runtime, the amount of trace data
  - Without a duration filter, you can only capture about 0.4 seconds worth of data
- To see only long-duration functions
  - When looking for long-lasting functions, you don’t need to see the short ones (in most cases)

Filtering by Duration - first try

- Added duration filter to 'function_graph' tracer
- Method:
  - Compare duration to threshhold
  - Discard function entry and exit events
- Its easy to discard exit event
  - Just don’t commit data
- Trickier to discard entry event
  - ring_buffer_event_discard() converts event to padding if subsequent events have been committed to buffer
    - Wastes a lot of space
    - Severely constrains the time coverage for a trace

Filtering by Duration - second try

- Created new 'function_duration' tracer
- Method:
  - Don't save function *entries* to trace log at all
    - Only save call time on function return stack
  - At function exit, compare duration to threshhold
  - Omit exit entry events for short duration functions
- Results in simpler, *and faster* code
- Only issue is that log is displayed in order of function exit (not function *entry*)
  - Can be solved with a simple sort on trace output

Trace time coverage: graph vs duration tracer

Tracer	Duration Filter Value	Total Function Count	Time Covered by Trace	Trace Event Count	Projected Trace Time Coverage
Graph	0	3.295M	0.39 s	27316	0.39 s
Graph	1000	3.310M	1.29 s	26630	1.39 s
Graph	100000	3.309M	1.34 s	26438	1.34 s
Duration	0	2.906M	0.38 s	27597	0.38 s
Duration	1000	2.788M	21.70 s †	3943	154.00 s
Duration	100000	2.795M	21.31 s †	208	2868.00 s

† The test finished without filling the buffer.

= Estimate

Example of Use

```
$ mount debugfs -t debugfs /debug
$ cd /debug/tracing
$ cat available_tracers
function_graph function_duration function sched_switch nop
$ echo 0 >tracing_enabled
$ echo 100 >tracing_thresh
$ echo function_duration >current_tracer
$ echo 1 >tracing_enabled ; do \
   ls /bin | sed s/a/z/g ; done ; echo 0 >tracing_enabled
$ echo duration-proc >trace_options
$ cat trace >/tmp/trace.txt
$ cat /tmp/trace.txt | sort –k3 > /tmp/trace.txt.sorted
```

Function Duration Results (sorted)

```
# tracer: function_duration
#
# CPU  TASK/PID        CALLTIME      DURATION                  FUNCTION CALLS
# |     |     |           |              |   |                     |   |   |   |
 0)    sed-562     |   502.854252393 |  ! 436.833 us |       bprm_mm_init
 0)    sed-562     |   502.854254893 |  ! 321.500 us |         mm_alloc
 0)    sed-562     |   502.854270893 |  ! 296.500 us |           mm_init
 0)    sed-562     |   502.854279393 |  ! 266.166 us |             get_pgd_slow
 0)    sed-562     |   502.854744059 |  ! 229.500 us |       prepare_binprm
 0)    sed-562     |   502.854765393 |  ! 198.666 us |         kernel_read
 0)    sed-562     |   502.854769226 |  ! 183.333 us |           vfs_read
 0)    sed-562     |   502.854780393 |  ! 142.000 us |             do_sync_read
 0)    sed-562     |   502.854785559 |  ! 120.667 us |               nfs_file_read
 0)    sed-562     |   502.854982393 |  ! 538.000 us |       copy_strings_kernel
 0)    sed-562     |   502.854985726 |  ! 521.667 us |         copy_strings
 0)    sed-562     |   502.854993893 |  ! 470.000 us |           get_arg_page
 0)    sed-562     |   502.854997226 |  ! 455.500 us |             get_user_pages
 0)    sed-562     |   502.855000059 |  ! 421.667 us |               __get_user_pages
 0)    sed-562     |   502.855031393 |  ! 285.666 us |                 handle_mm_fault
 0)    sed-562     |   502.855037726 |  ! 101.833 us |                   __pte_alloc
```

Measuring kernel boot

- Can start tracer early in boot sequence
- Use “ftrace=function_duration” on kernel command line
  - Can specify “tracing_thresh=<value>”
- Tracer is initialized after kernel core (timers, memory, interrupts), but before all initcalls
  - On my hardware, tracer starts about 50 milliseconds after start_kernel()
- Had to restore instrumentation to functions in _init segment
- Need to stop trace after point of interest

Introducing a stop trigger

- Use “trace_stop_fn=<func_name>” on kernel command line
- Trace stops on ENTRY to named function
- To use, figure out a fairly unique function, which runs immediately after the area of interest
- An initcall works very well
  - Initcall functions have unique names in kernel

Example of early boot trace

- To trace most of kernel boot:
  - Add this to the kernel command line:
    - “ftrace=function_duration tracing_thresh=200 trace_stop_fn=run_init_process”
  - If the trace doesn't cover the whole boot, increase tracing_thresh and try again
- To trace an individual initcall:
  - Find initcall following the one you are interested in
    - Can use initcall_debug on kernel command line
    - ex: pty_init follows tty_init
  - Kernel command line:
    - “ftrace=function_duration trace_stop_fn=pty_init”

Post-trace analysis

- fdd tool is provided to analyze data
- What fdd shows:
  - function counts, total time, average duration
  - sub-routine with the longest duration, how many times it was called
  - Local time = total time minus sub-routine total time
    - Is approximately the cost of the local execution of a function
- Notes:
  - Total time may be wrong if process is scheduled out or if a filter was active
    - May need an option to subtract time that function was scheduled out
  - You can filter, sort, select output columns,etc.

fdd Output

```
$ fdd /tmp/trace.txt –n 15
Function                            Count Time       Average  Local
----------------------------------- ----- ---------- -------- ----------
schedule                               59 1497735270 25385343 1476642939
sys_write                              56 1373722663 24530761    2892665
vfs_write                              56 1367969833 24428032    3473173
tty_write                              54 1342476332 24860672 1212301170
do_path_lookup                         95 1076524931 11331841   34682198
__link_path_walk                       99 1051351737 10619714    6702507
rpc_call_sync                          87 1033211085 11875989    1700178
path_walk                              94 1019263902 10843233    3425163
rpc_run_task                           87  960080412 11035407    2292360
rpc_execute                            87  936049887 10759194    2316635
__rpc_execute                          87  932779083 10721598   11383353
do_lookup                             191  875826405  4585478    9510659
call_transmit                         100  785408085  7854080    5871339
__nfs_revalidate_inode                 38  696216223 18321479    1652173
nfs_proc_getattr                       38  690552053 18172422    1234634
```

Performance issues

- Overhead of tracing can be big
  - Average function duration = 3.22 μs
  - Overhead = 11.4 microseconds per function
- Use a CPU-bound test to measure overhead
  - “find /sys >/dev/null”
  - With an I/O-bound test (or a real-workload), the ratio of overhead to average function duration should be lower
- With ftrace compiled into kernel, but the 'NOP' tracer selected, the overhead in my test was about 12%

Overhead Measurements

Tracer status	Elapsed time	Function count	Time per function	Overhead per function
TRACE=n	8.85 s	2.751M *	3.22 us	-
Tracer=nop	9.94 s	2.757M *	3.61 us	0.39 us
Tracer=duration, enabled=0	21.57 s	2.816M	7.66 us	4.44 us
Tracer=duration, thresh=0	42.55 s	2.911M	14.62 us	11.40 us
thresh=1	42.80 s	2.923M	14.64 us	11.42 us
thresh=10	30.87 s	2.850M	10.83 us	7.61 us
thresh-=100	24.58 s	2.824M	8.70 us	5.48 us
thresh=1000	21.40 s	2.802M	7.64 us	4.42 us
thresh=1000000	21.43 s	2.803M	7.64 us	4.42 us

* = estimated

Roadmap and future work

- Mainline try 2
  - Patches:
    - ARM function graph assembly support
    - function_duration tracer
    - changes to ftrace for use at boot time
- Need to use functionality to improve bootup time
  - Have already identified a few problems
    - call_usermode_helper *(may already be done)*
    - ip_auto_config

References

- Ftrace tutorial at OLS 2008
  - http://people.redhat.com/srostedt/ftrace-tutorial.odp
  - Video: http://free-electrons.com/pub/video/2008/ols/
    ols2008-steven-rostedt-ftrace.ogg
- “The world of Ftrace” at Spring 2009 LF Collaboration Summit
  - http://people.redhat.com/srostedt/ftrace-world.odp
- Patches and tools for this talk
  - http://elinux.org/Ftrace_Function_Graph_ARM

Questions & Answers