Linaro connect
Vancouver 2018

Writing Better Function Tests with GCOV

Masami Hiramatsu

Who Am I

Masami Hiramatsu (Linaro)

- Tech Lead in Socionext Landing Team
- Maintainer for:
    - kprobes and dynamic tracing in Linux kernel
    - ftracetest (a part of kselftest, function tests for ftrace)

Agenda

- Function Tests
  - Usual Issues
- GCOV and LCOV
  - How to use in userspace
- GCOV Kernel
  - Subsystem profiling
- Writing Function Tests with GCOV
- Ftracetest
  - Improving ftracetest with GCOV
  - Typical Untested Patterns
  - Pitfalls

Function(al) Tests

Tests each “function(feature)” of software

- Not function-level unit test :)
- Not a stress test
- It is a kind of regression test

Goal of function test

- For ensuring the “function” works as we expected
- Make sure no regressions while upgrading

Function Tests in Linux Kernel

There are several function tests

- Boot-time self tests
- Test (sample) modules
- Runtime tests
- Test collection: kselftests

Usual Issues on Writing Tests

A bug was found!

-> Why was not that tested?

Want to write a test!

-> What functions are not tested?

We need a measurement / visualizing tool for writing tests

GCOV and LCOV

GCOV: Coverage measurement tool for GCC

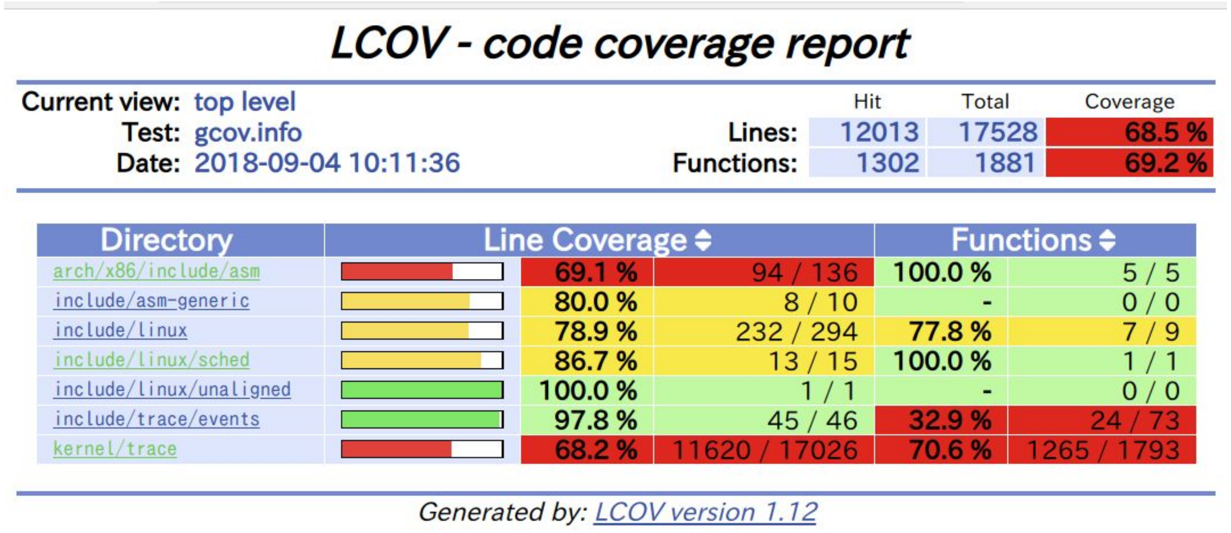
- Shows which “Line of code” is executed
- Calculate the coverage rate per line for each file

LCOV: Gcov visualizing wrapper tool

- Analyze multiple files at once
- Visualize the report in HTML
    - Show per-line and per-function coverage rate
    - Source-code based coverage report

LCOV Examples

Overview



LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level				
Test:	gcov.info	Lines:	12013	17528	68.5 %
Date:	2018-09-04 10:11:36	Functions:	1302	1881	69.2 %

Directory	Line Coverage ⇕			Functions ⇕	
arch/x86/include/asm		69.1 %	94 / 136	100.0 %	5 / 5
include/asm-generic		80.0 %	8 / 10	-	0 / 0
include/linux		78.9 %	232 / 294	77.8 %	7 / 9
include/linux/sched		86.7 %	13 / 15	100.0 %	1 / 1
include/linux/unaligned		100.0 %	1 / 1	-	0 / 0
include/trace/events		97.8 %	45 / 46	32.9 %	24 / 73
kernel/trace		68.2 %	11620 / 17026	70.6 %	1265 / 1793

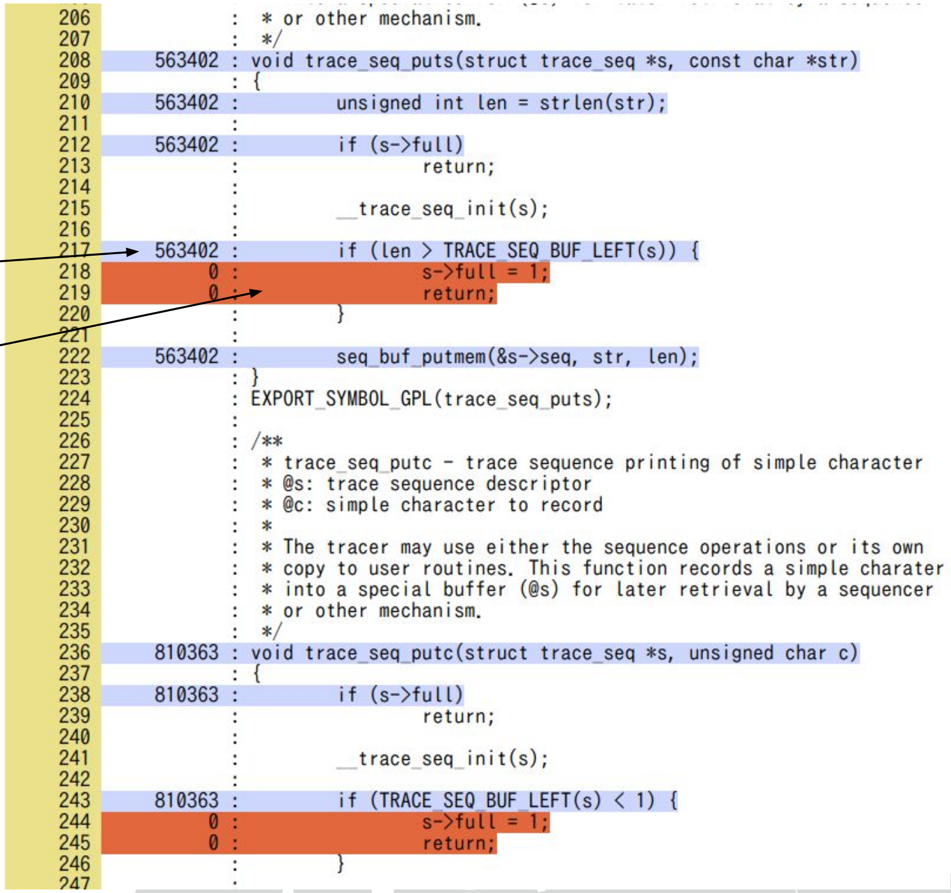
Generated by: LCOV version 1.12

LCOV Examples

Source view

# of Executed

Uncovered Lines



```
                 :  * or other mechanism.
                 :  */
          563402 : void trace_seq_puts(struct trace_seq *s, const char *str)
                 : {
          563402 :         unsigned int len = strlen(str);
                 :
          563402 :         if (s->full)
                 :                 return;
                 :
                 :         __trace_seq_init(s);
                 :
          563402 :         if (len > TRACE_SEQ_BUF_LEFT(s)) {
               0 :                 s->full = 1;
               0 :                 return;
                 :         }
                 :
          563402 :         seq_buf_putmem(&s->seq, str, len);
                 : }
                 : EXPORT_SYMBOL_GPL(trace_seq_puts);
                 :
                 : /**
                 :  * trace_seq_putc - trace sequence printing of simple character
                 :  * @s: trace sequence descriptor
                 :  * @c: simple character to record
                 :  *
                 :  * The tracer may use either the sequence operations or its own
                 :  * copy to user routines. This function records a simple charater
                 :  * into a special buffer (@s) for later retrieval by a sequencer
                 :  * or other mechanism.
                 :  */
          810363 : void trace_seq_putc(struct trace_seq *s, unsigned char c)
                 : {
          810363 :         if (s->full)
                 :                 return;
                 :
                 :         __trace_seq_init(s);
                 :
          810363 :         if (TRACE_SEQ_BUF_LEFT(s) < 1) {
               0 :                 s->full = 1;
               0 :                 return;
                 :         }
                 :
```

GCOV in Userspace

To apply gcov in userspace

1. Pass “-fprofile-arcs -ftest-coverage” options to gcc when compiling a program
2. Run the program
3. You’ll see `<SOURCE>.gcda` and `<SOURCE>.gcno`
4. In the same directory, run “`gcov <SOURCE>.c`” command
5. Check generated `<SOURCE>.c.gcov`
   - This shows per-line execution count with source code.

LCOV in Userspace

To use lcov in userspace

1. Pass “-fprofile-arcs -ftest-coverage” options to gcc when compiling program
2. Run the program
3. You’ll see `<SOURCE>.gcda` and `<SOURCE>.gcno`
4. In the same directory, run “`lcov -c -d ./ -o lcov.info`”
5. Run “`genhtml -o html lcov.info` ”
6. Open `html/index.html`

GCOV in Kernel

Linux kernel can export GCOV logfile via debugfs

- Pseudo GCDA files and GCNO symlinks are exported under `/sys/kernel/debug/gcov/<build-path>`
- Test -> Copy the pseudo logfiles (make a snapshot) -> analyze it

Enablement

- **CONFIG_GCOV_KERNEL=y** compiles the framework
- CONFIG_GCOV_PROFILE_ALL=y profiles the whole kernel (**not recommended**)

Subsystem Profiling by GCOV

We can enable GCOV profiling on specific subsystem or file (**recommended**)

Add below lines in Makefile of the subsystem

- For profiling a file (e.g. sample.c)

  ```
  GCOV_PROFILE_sample.o := y
  ```

- For profiling all files under the directory

  ```
  GCOV_PROFILE := y
  ```

Writing Function Tests with GCOV

Instructions

1. Enable GCOV_PROFILE in target subsystem and build the kernel
2. Write a simple function test
3. Run the test
4. Check GCOV result by LCOV
5. Find what is **not** covered
6. Add a new test or improve existing one
7. Goto 3 until all functions(features) are covered

Goal of Function Tests

Don’t aim to 100% coverage of lines

- Test “functions(features)” not “implementation”
- The Linux implementation is always evolving
- Do not cover critical cases (Panic, BUG, etc)

Focus on

- What functions (of code) are not executed
- Are there any possible use-case?
- Is that a “feature” ?

Ftracetest Improvement

Ftrace - a collection of Linux kernel tracers

- 10 tracers + more than 1700 events + 42 options and more...
  - See /sys/kernel/debug/tracing/*
- All operations can be done via the tracefs interface (like debugfs)

Ftracetest - a collection of test cases for ftrace

- Shell-script based test framework and test cases under kselftests
  - See linux/tools/testing/selftests/ftrace/*
- Includes more than 50 test cases
- Show precise logs and summary

Ftracetest Example

Run by root user, and reported the result summary

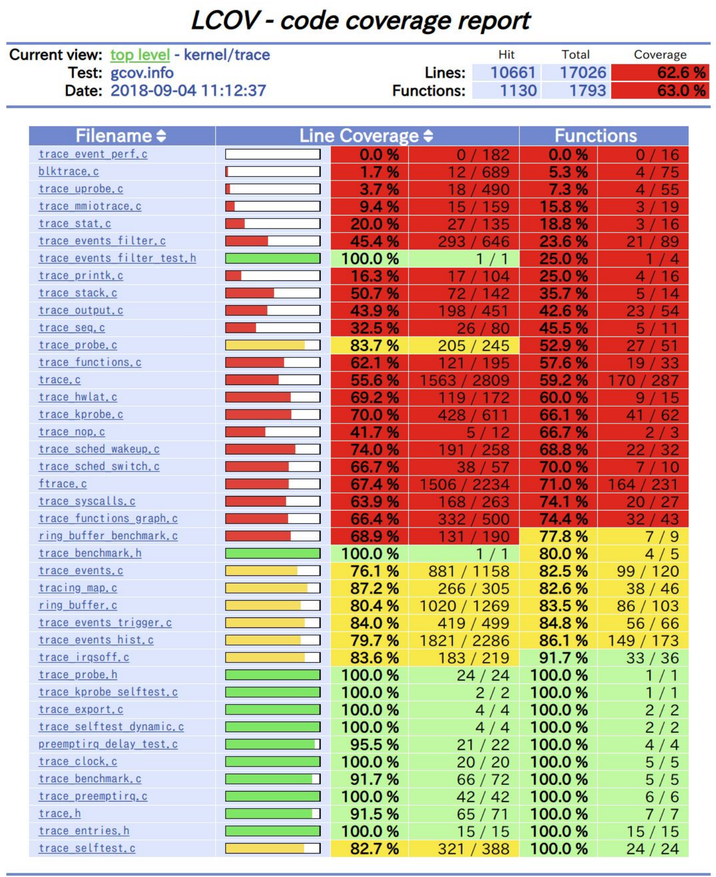
```
ftrace # ./ftracetest
=== Ftrace unit tests ===
[1] Basic trace file check   [PASS]
[2] Basic test for tracers   [PASS]
...
[68] (instance)  trace_marker trigger - test snapshot trigger    [PASS]

# of passed:  66
# of failed:  0
# of unresolved:  1
# of untested:  0
# of unsupported:  1
# of xfailed:  0
# of undefined(test bug):  0
```

Profiling Ftrace by GCOV

1. Add GCOV_PROFILE := y in kernel/trace/Makefile
   (This patch has been upstreamed, see 6b7dca401cb1)
2. Run ftracetest
```
$ cd tools/testing/selftests/ftrace
$ ./ftracetest
```
3. Copy GCOV data and analyze it
```
$ cp -r /sys/kernel/debug/gcov/<source-dir>/linux/kernel /opt/gcov-before
$ cd /opt/gcov-before
$ lcov -c -d ./trace -o lcov.info && genhtml -o html lcov.info
$ google-chrome html/kernel/trace/index.html
```
Target source directory

Let’s Check Code Coverage



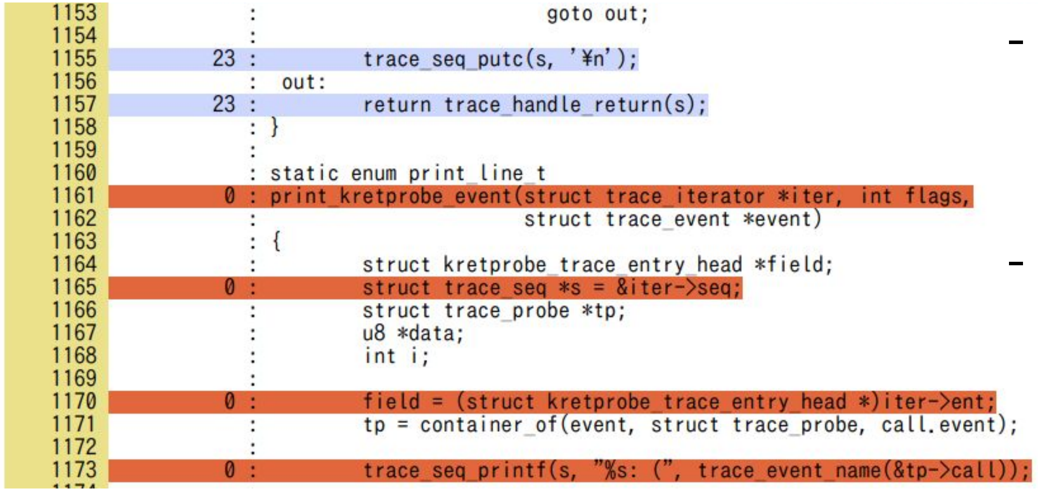
LCOV - code coverage report

				Hit	Total	Coverage
Current view:	top level - kernel/trace					
Test:	gcov.info		Lines:	10661	17026	62.6 %
Date:	2018-09-04 11:12:37		Functions:	1130	1793	63.0 %

Filename	Line Coverage			Functions	
trace_event_perf.c		0.0 %	0 / 182	0.0 %	0 / 16
blktrace.c		1.7 %	12 / 689	5.3 %	4 / 75
trace_uprobe.c		3.7 %	18 / 490	7.3 %	4 / 55
trace_mmiotrace.c		9.4 %	15 / 159	15.8 %	3 / 19
trace_stat.c		20.0 %	27 / 135	18.8 %	3 / 16
trace_events_filter.c		45.4 %	293 / 646	23.6 %	21 / 89
trace_events_filter_test.h		100.0 %	1 / 1	25.0 %	1 / 4
trace_printk.c		16.3 %	17 / 104	25.0 %	4 / 16
trace_stack.c		50.7 %	72 / 142	35.7 %	5 / 14
trace_output.c		43.9 %	198 / 451	42.6 %	23 / 54
trace_seq.c		32.5 %	26 / 80	45.5 %	5 / 11
trace_probe.c		83.7 %	205 / 245	52.9 %	27 / 51
trace_functions.c		62.1 %	121 / 195	57.6 %	19 / 33
trace.c		55.6 %	1563 / 2809	59.2 %	170 / 287
trace_hwlat.c		69.2 %	119 / 172	60.0 %	9 / 15
trace_kprobe.c		70.0 %	428 / 611	66.1 %	41 / 62
trace_nop.c		41.7 %	5 / 12	66.7 %	2 / 3
trace_sched_wakeup.c		74.0 %	191 / 258	68.8 %	22 / 32
trace_sched_switch.c		66.7 %	38 / 57	70.0 %	7 / 10
ftrace.c		67.4 %	1506 / 2234	71.0 %	164 / 231
trace_syscalls.c		63.9 %	168 / 263	74.1 %	20 / 27
trace_functions_graph.c		66.4 %	332 / 500	74.4 %	32 / 43
ring_buffer_benchmark.c		68.9 %	131 / 190	77.8 %	7 / 9
trace_benchmark.h		100.0 %	1 / 1	80.0 %	4 / 5
trace_events.c		76.1 %	881 / 1158	82.5 %	99 / 120
tracing_map.c		87.2 %	266 / 305	82.6 %	38 / 46
ring_buffer.c		80.4 %	1020 / 1269	83.5 %	86 / 103
trace_events_trigger.c		84.0 %	419 / 499	84.8 %	56 / 66
trace_events_hist.c		79.7 %	1821 / 2286	86.1 %	149 / 173
trace_irqsoff.c		83.6 %	183 / 219	91.7 %	33 / 36
trace_probe.h		100.0 %	24 / 24	100.0 %	1 / 1
trace_kprobe_selftest.c		100.0 %	2 / 2	100.0 %	1 / 1
trace_export.c		100.0 %	4 / 4	100.0 %	2 / 2
trace_selftest_dynamic.c		100.0 %	4 / 4	100.0 %	2 / 2
preemptirq_delay_test.c		95.5 %	21 / 22	100.0 %	4 / 4
trace_clock.c		100.0 %	20 / 20	100.0 %	5 / 5
trace_benchmark.c		91.7 %	66 / 72	100.0 %	5 / 5
trace_preemptirq.c		100.0 %	42 / 42	100.0 %	6 / 6
trace.h		91.5 %	65 / 71	100.0 %	7 / 7
trace_entries.h		100.0 %	15 / 15	100.0 %	15 / 15
trace_selftest.c		82.7 %	321 / 388	100.0 %	24 / 24

- 63.0% functions are covered
- 22 / 41 files are under 75% coverage of functions.
- There is room for improvement in ftracetest

Find Untested Code



```
           :                         goto out;
           :
        23 :                 trace_seq_putc(s, '¥n');
           :  out:
        23 :                 return trace_handle_return(s);
           : }
           :
           : static enum print_line_t
         0 : print_kretprobe_event(struct trace_iterator *iter, int flags,
           :                       struct trace_event *event)
           : {
           :                 struct kretprobe_trace_entry_head *field;
         0 :                 struct trace_seq *s = &iter->seq;
           :                 struct trace_probe *tp;
           :                 u8 *data;
           :                 int i;
           :
         0 :                 field = (struct kretprobe_trace_entry_head *)iter->ent;
           :                 tp = container_of(event, struct trace_probe, call.event);
           :
         0 :                 trace_seq_printf(s, "%s: (", trace_event_name(&tp->call));
```

- “`print_kretprobe_event()`” is not tested
  - This function is for printing out the “function-return” kretprobe event
- ftracetest has a kretprobe event testcase. But it does NOT test kretprobe event **“output”**

Improve Test Case

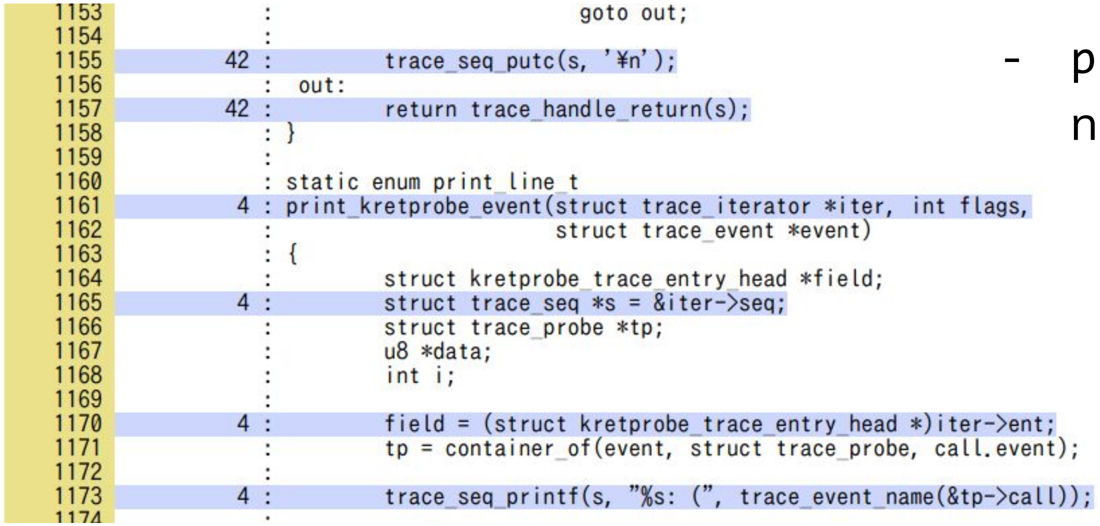
Not only setting the event, but also **ensure the trace output**

```
 echo 'r:testprobe2 _do_fork $retval' > kprobe_events
-grep testprobe2 kprobe_events
+grep testprobe2 kprobe_events | grep -q 'arg1=\$retval'
 test -d events/kprobes/testprobe2
+
 echo 1 > events/kprobes/testprobe2/enable
 ( echo "forked")
+
+cat trace | grep testprobe2 | grep -q '<- _do_fork'
+
 echo 0 > events/kprobes/testprobe2/enable
 echo '-:testprobe2' >> kprobe_events
 clear_trace
```

(Ensure the setting is correctly done)

(Ensure the trace output)

Improvement Result



```
         :                          goto out;
         :
      42 :              trace_seq_putc(s, '¥n');
         :   out:
      42 :              return trace_handle_return(s);
         : }
         :
         : static enum print_line_t
       4 : print_kretprobe_event(struct trace_iterator *iter, int flags,
         :                        struct trace_event *event)
         : {
         :              struct kretprobe_trace_entry_head *field;
       4 :              struct trace_seq *s = &iter->seq;
         :              struct trace_probe *tp;
         :              u8 *data;
         :              int i;
         :
       4 :              field = (struct kretprobe_trace_entry_head *)iter->ent;
         :              tp = container_of(event, struct trace_probe, call.event);
         :
       4 :              trace_seq_printf(s, "%s: (", trace_event_name(&tp->call));
```

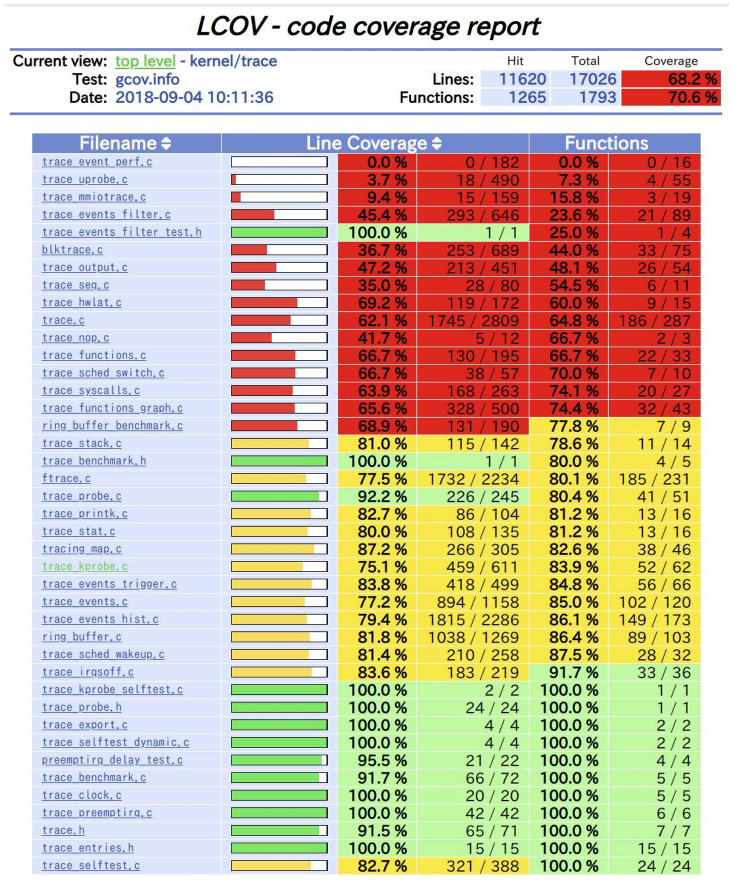
- `print_kretprobe_event()` is now tested :-)

Typical Untested Patterns

Typical patterns of uncovered function-tests

- Functions that are just not touched
    - Function is documented, but not tested
    - Main function is tested, but sub options are not
- Setting without verified
    - Setting the function but just set. Not verified.
    - Not only check the result, but also verify if possible
        - set_XXX -> get_XXX
        - write_XXX -> read_XXX
        - echo 1 > XXX -> cat XXX
- Undocumented features
    - New feature is not documented, no one knows.
    - Testing a feature which will be dropped in the future

Improvement Summary



LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level - kernel/trace	Lines:	11620	17026	68.2 %
Test:	gcov.info	Functions:	1265	1793	70.6 %
Date:	2018-09-04 10:11:36				

Filename	Line Coverage			Functions	
trace_event_perf.c		0.0 %	0 / 182	0.0 %	0 / 16
trace_uprobe.c		3.7 %	18 / 490	7.3 %	4 / 55
trace_mmiotrace.c		9.4 %	15 / 159	15.8 %	3 / 19
trace_events_filter.c		45.4 %	293 / 646	23.6 %	21 / 89
trace_events_filter_test.h		100.0 %	1 / 1	25.0 %	1 / 4
blktrace.c		36.7 %	253 / 689	44.0 %	33 / 75
trace_output.c		47.2 %	213 / 451	48.1 %	26 / 54
trace_seq.c		35.0 %	28 / 80	54.5 %	6 / 11
trace_hwlat.c		69.2 %	119 / 172	60.0 %	9 / 15
trace.c		62.1 %	1745 / 2809	64.8 %	186 / 287
trace_nop.c		41.7 %	5 / 12	66.7 %	2 / 3
trace_functions.c		66.7 %	130 / 195	66.7 %	22 / 33
trace_sched_switch.c		66.7 %	38 / 57	70.0 %	7 / 10
trace_syscalls.c		63.9 %	168 / 263	74.1 %	20 / 27
trace_functions_graph.c		65.6 %	328 / 500	74.4 %	32 / 43
ring_buffer_benchmark.c		68.9 %	131 / 190	77.8 %	7 / 9
trace_stack.c		81.0 %	115 / 142	78.6 %	11 / 14
trace_benchmark.h		100.0 %	1 / 1	80.0 %	4 / 5
ftrace.c		77.5 %	1732 / 2234	80.1 %	185 / 231
trace_probe.c		92.2 %	226 / 245	80.4 %	41 / 51
trace_printk.c		82.7 %	86 / 104	81.2 %	13 / 16
trace_stat.c		80.0 %	108 / 135	81.2 %	13 / 16
tracing_map.c		87.2 %	266 / 305	82.6 %	38 / 46
trace_kprobe.c		75.1 %	459 / 611	83.9 %	52 / 62
trace_events_trigger.c		83.8 %	418 / 499	84.8 %	56 / 66
trace_events.c		77.2 %	894 / 1158	85.0 %	102 / 120
trace_events_hist.c		79.4 %	1815 / 2286	86.1 %	149 / 173
ring_buffer.c		81.8 %	1038 / 1269	86.4 %	89 / 103
trace_sched_wakeup.c		81.4 %	210 / 258	87.5 %	28 / 32
trace_irqsoff.c		83.6 %	183 / 219	91.7 %	33 / 36
trace_kprobe_selftest.c		100.0 %	2 / 2	100.0 %	1 / 1
trace_probe.h		100.0 %	24 / 24	100.0 %	1 / 1
trace_export.c		100.0 %	4 / 4	100.0 %	2 / 2
trace_selftest_dynamic.c		100.0 %	4 / 4	100.0 %	2 / 2
preemptirq_delay_test.c		95.5 %	21 / 22	100.0 %	4 / 4
trace_benchmark.c		91.7 %	66 / 72	100.0 %	5 / 5
trace_clock.c		100.0 %	20 / 20	100.0 %	5 / 5
trace_preemptirq.c		100.0 %	42 / 42	100.0 %	6 / 6
trace.h		91.5 %	65 / 71	100.0 %	7 / 7
trace_entries.h		100.0 %	15 / 15	100.0 %	15 / 15
trace_selftest.c		82.7 %	321 / 388	100.0 %	24 / 24

You can find the series (v3) here (https://lkml.org/lkml/2018/8/30/497)

- Add 13 new test cases
- 70.6% functions are covered
- 15 / 41 files are under 75% coverage of functions.
- Still there is room for improvement

Pitfalls

Bad signals...

- Break something (e.g. testing critical error path / panic)
    - BTW, if you find it easily, it must be a **BUG** and must be fixed soon.
- Give a stress on the system (e.g. OOM)
    - That's a stress test.
- Start using error injection

No, you are stepping into the dark side...

Side Effects

- Improves documentation
  - docs: tracing: Add stacktrace filter command
- Orphaned functions found
  - 72809cbf ("tracing: Remove orphaned function using_ftrace_ops_list_func()")
  - 7b144b6c ("tracing: Remove orphaned function ftrace_nr_registered_ops()")
- Unused(obsoleted) features found
  - test_nop_accept/refuse are tentative function
  - hex/raw/bin output format will be replaced by trace_pipe_raw
- Real bugs :)
  - 757d9140 ("tracing/blktrace: Fix to allow setting same value")
  - Stack tracer filter doesn't work correctly
  - GCOV kernel was broken on some arch!

Conclusion

- Using GCOV is very easy
  - For Linux kernel, you just need CONFIG_GCOV_KERNEL=y and add GCOV_PROFILE:=y
  - Show how to use gcov and lcov commands

- Function tests can be improved by GCOV
  - Easy to find untested functions
  - Explained by ftracetest case

- Ftracetest was improved by GCOV
  - ~7% coverage improved with 14 new test cases
  - Found some real bugs etc.

Future Work

- Continue to improve ftracetest
  - Check what is not tested and add new tests
- Improve other selftests
  - We can also find untested functions for other tests
- FCOV: we can use ftrace instead of GCOV for profiling “function” coverage.
  - We can dynamically change the target subsystem
  - Inline functions can be covered by kprobe dynamic event

Questions?

Thank You!!

Masami Hiramatsu <mhiramat@kernel.org>
or <masami.hiramatsu@linaro.org>