Ftrace Kernel Hooks: More than just tracing

Presenter:
Steven Rostedt
rostedt@goodmis.org
Red Hat
Ftrace Function Hooks

- Function Tracer
- Function Graph Tracer
- Function Profiler
- Stack Tracer
- Kprobes
- Uprobes
- Perf
- Pstore
- SystemTap
```bash
# cd /sys/kernel/debug/tracing
# echo function > current_tracer
# cat trace
# tracer: function
# entries-in-buffer/entries-written: 205022/119956607   #P:4
#
#      _-----=> irqs-off
#    / _----=> need-resched
#   / _---=> hardirq/softirq
#  / _--=> preempt-depth
# || /     delay
# TASK-PID CPU# | | | | | | | | TIME | FUNCTION
# | | | | | | | | TIMESTAMP | |
# <idle>-0     [002] dN.1  1781.978299: rcu_eqs_exit <-rcu_idle_exit
# <idle>-0     [002] dN.1  1781.978300: rcu_eqs_exit_common <-rcu_eqs_exit
# <idle>-0     [002] .N.1  1781.978301: arch_cpu_idle_exit <-cpu_startup_entry
# <idle>-0     [002] .N.1  1781.978301: tick_nohz_idle_exit <-cpu_startup_entry
# <idle>-0     [002] dN.1  1781.978301: ktime_get <-tick_nohz_idle_exit
# <idle>-0     [002] dN.1  1781.978302: update_ts_time_stats <-tick_nohz_idle_exit
# <idle>-0     [002] dN.1  1781.978302: nr_iowait_cpu <-update_ts_time_stats
# <idle>-0     [002] dN.1  1781.978303: tick_do_update_jiffies64 <-tick_nohz_idle_exit
# <idle>-0     [002] dN.1  1781.978303: update_cpu_load_nohz <-tick_nohz_idle_exit
# <idle>-0     [002] dN.1  1781.978303: calc_load_exit_idle <-tick_nohz_idle_exit
```
```bash
# echo function_graph > current_tracer
# cat trace
# tracer: function_graph

# CPU   DURATION   FUNCTION CALLS
#      |   |                     |   |   |   |
# 2) 7.879 us   } /* context_tracking_user_exit */
# 2) 0.070 us   __do_page_fault() {
# 2) 0.057 us   down_read_trylock();
# 2) 0.096 us   __might_sleep();
# 2) 0.070 us   find_vma();
# 2) 0.074 us   handle_mm_fault() {
# 2) 0.057 us   __do_fault() {
# 2) 0.061 us   filemap_fault() {
# 2) 0.069 us   find_get_page() {
# 2) 0.063 us   __rcu_read_lock();
# 2) 0.061 us   __rcu_read_unlock();
# 2) 1.241 us   }
# 2) 0.070 us   __might_sleep();
# 2) 2.201 us   }
# 2) 0.070 us   _raw_spin_lock() {
# 2) 0.070 us   preempt_count_add();
# 2) 0.528 us   }
# 2) 0.070 us   add_mm_counter_fast();
# 2) 0.070 us   page_add_file_rmap();
# 2) 0.070 us   _raw_spin_unlock() {
# 2) 0.070 us   preempt_count_sub();
```
Dynamic Function Tracing

```bash
# echo '*sched*' > set_ftrace_filter
# echo function > current_tracer
# cat trace
# tracer: function
# entries-in-buffer/entries-written: 193727/240417   #P:4
`
How it works?

- gcc’s profiler option: -pg
- Adds special mcount function call
  - all functions call mcount
  - mcount is a trampoline
A function call

```c
asmlinkage __visible void __sched schedule(void) {
    struct task_struct *tsk = current;
    sched_submit_work(tsk);
    __schedule();
}
```
A function call

- Disassembled

```
<schedule>:
  55                  push %rbp
  48 8b 04 25 80 c0 0e mov 0xffffffff810ec080,%rax
  81
  48 89 e5           mov %rsp,%rbp
  48 8b 00           mov (%rax),%rax
  5d
  e9 db fa ff ff      jmpq fffffffff810bb100 <__schedule>
  66 66 2e 0f 1f 84 00 data16 nopw %cs:0x0(%rax,%rax,1)
  00 00 00 00
```
A function call With -pg option

- Disassembled

<schedule>:
55                      push   %rbp
48 89 e5                mov    %rsp,%rbp
e8 37 2e 00 00          callq  ffffffff810f7430 <mcount>
5d                      pop    %rbp
48 8b 04 25 80 d0 15    mov    0xffffffff8115d080,%rax
81                      mov     (%rax),%rax
48 8b 00                mov     (%rax),%rax
e9 96 fa ff ff          jmpq   ffffffff810f40a0 <__schedule>
66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
The kernel at boot up

<schedule>:
push   %rbp
mov    %rsp,%rbp
callq  <mcount>
pop    %rbp

<mcount>:
retq
Where’s the mcounts?

- Can’t just call the mcounts
  - too much overhead
  - retq only is 13% added overhead!
- Need to convert them to nops at boot up
- Need to know where they are
- Best to find them at compile time
recordmcount

- scripts/recordmcount.c (and a perl version)
- reads the object files one at a time
- reads the relocation tables
  - finds all the calls to mcount
  - creates a table
  - links the table back into the object file
  - New section called __mcount_loc
recordmcount

kernel/sched/core.o:

```assembly
<schedule>:
push %rbp
mov %rsp,%rbp
callq <mcount>
pop %rbp

[...] <preempt_schedule_irq>:
push %rbp
mov %rsp,%rbp
push %rbx
callq <mcount>
pop %rbp

[...] <_cond_resched>:
push %rbp
mov %rsp,%rbp
push %rbx
callq <mcount>
pop %rbp

[...] <yield>:
push %rbp
mov %rsp,%rbp
push %rbx
callq <mcount>
pop %rbp

[...]<__mcount_loc>:
&schedule + 0x4
&preempt_schedule_irq + 0x4
&_cond_resched + 0x4
&yield + 0x4
```
<schedule>:
push %rbp
mov %rsp,%rbp
callq <mcount>
pop %rbp
[…]
<preempt_schedule_irq>:
push %rbp
mov %rsp,%rbp
push %rbx
callq <mcount>
pop %rbp
[…]
<_cond_resched>:
push %rbp
mov %rsp,%rbp
push %rbx
callq <mcount>
pop %rbp
[…]
<yield>:
push %rbp
mov %rsp,%rbp
push %rbx
callq <mcount>
pop %rbp
[…]
<__mcount_loc>:
&schedule + 0x4
&preempt_schedule_irq + 0x4
&_cond_resched + 0x4
&yield + 0x4
Linker Magic

- vmlinux.lds
  - include/linux/vmlinux.lds.h
  - arch/x86/kernel/vmlinux.lds.S
- Magic variables
  - __start_mcount_loc
  - __stop_mcount_loc

```c
#ifdef CONFIG_FTRACE_MCOUNT_RECORD
#define MCOUNT_REC()   . = ALIGN(8);                           
                      VMLINUX_SYMBOL(__start_mcount_loc) = .; \ 
                      *(__mcount_loc)                         \ 
                      VMLINUX_SYMBOL(__stop_mcount_loc) = .;
#else
#define MCOUNT_REC()
#endif
```
Linker Magic

vmlinux:

kernel/sched/core.o:

<__mcount_loc>:
&schedule + 0x4
&preempt_schedule_irq + 0x4
&cond_resched + 0x4
&yield + 0x4

mm/swap.o:

<__mcount_loc>:
&put_page + 0x4
&_get_page_tail + 0x4
&put_pages_list + 0x4
&get_kernel_pages + 0x4

fs/read_write.o:

<__mcount_loc>:
&new_sync_read + 0x4
&vfs_setpos + 0x4
&fixed_size_llseek + 0x4
&default_llseek + 0x4

...
vmlinux:

```
<__start_mcount_loc>:
&schedule + 0x4
&preempt_schedule_irq + 0x4
&cond_resched + 0x4
&yield + 0x4
&put_page + 0x4
&__get_page_tail + 0x4
&put_pages_list + 0x4
&get_kernel_pages + 0x4
&new_sync_read + 0x4
&vfs_setpos + 0x4
&fixed_size_llseek + 0x4
&default_llseek + 0x4
[...]
<__end_mcount_loc>:
```

kernel/sched/core.o:

```
<__mcount_loc>:
&schedule + 0x4
&preempt_schedule_irq + 0x4
&cond_resched + 0x4
&yield + 0x4
```

mm/swap.o:

```
<__mcount_loc>:
&put_page + 0x4
&__get_page_tail + 0x4
&put_pages_list + 0x4
&get_kernel_pages + 0x4
```

fs/read_write.o:

```
<__mcount_loc>:
&new_sync_read + 0x4
&vfs_setpos + 0x4
&fixed_size_llseek + 0x4
&default_llseek + 0x4
```

...
Linker Magic

vmlinux:

<__start_mcount_loc>:
0xfffffffff810f45f4
0xfffffffff810f4635
0xfffffffff810f4684
0xfffffffff810f4734
0xfffffffff81087ad4
0xfffffffff81087b14
0xfffffffff81087bd5
0xfffffffff81087c41
0xfffffffff810a7aa0
0xfffffffff810a7bd4
0xfffffffff810a7d34
0xfffffffff810a7d7d
[...]
<__end_mcount_loc>:

kernel/sched/core.o:

<__mcount_loc>:
&schedule + 0x4
&preempt_schedule_irq + 0x4
&_cond_resched + 0x4
&yield + 0x4

mm/swap.o:

<__mcount_loc>:
&put_page + 0x4
&_get_page_tail + 0x4
&put_pages_list + 0x4
&get_kernel_pages + 0x4

fs/read_write.o:

<__mcount_loc>:
&new_sync_read + 0x4
&vfs_setpos + 0x4
&fixed_size_llseek + 0x4
&default_llseek + 0x4

[...]

...
Finding Mcount

vmlinux:

<schedule>:
push %rbp
mov %rsp,%rbp
callq <mcount>
pop %rbp
[...]
<preempt_schedule_irq>:
push %rbp
mov %rsp,%rbp
push %rbx
callq <mcount>
pop %rbp
[...]
<_cond_resched>:
push %rbp
mov %rsp,%rbp
push %rbx
callq <mcount>
pop %rbp
[...]
<yield>:
push %rbp
mov %rsp,%rbp
push %rbx
callq <mcount>
pop %rbp
[...]

__/start_mcount_loc>:
[...]
<___end_mcount_loc>:
Finding Mcount

vmlinux:

```c
<schedule>:
    push   %rbp
    mov    %rsp,%rbp
    callq  <mcount>
    pop    %rbp
    [...]
</preempt_schedule_irq>:
    push   %rbp
    mov    %rsp,%rbp
    push   %rbx
    callq  <mcount>
    pop    %rbp
    [...]
<_cond_resched>:
    push   %rbp
    mov    %rsp,%rbp
    push   %rbx
    callq  <mcount>
    pop    %rbp
    [...]
<yield>:
    push   %rbp
    mov    %rsp,%rbp
    push   %rbx
    callq  <mcount>
    pop    %rbp
    [...]
<__start_mcount_loc>:
    [...]
<___end_mcount_loc>:
```
Finding Mcount

vmlinux:

```c
<schedule>:
push   %rbp
mov    %rsp,%rbp
nop    %rbp
pop    %rbp
[…]
<preempt_schedule_irq>:
push   %rbp
mov    %rsp,%rbp
push   %rbx
nop    %rbp
pop    %rbp
[…]
<_cond_resched>:
push   %rbp
mov    %rsp,%rbp
push   %rbx
nop    %rbp
pop    %rbp
[…]
<yield>:
push   %rbp
mov    %rsp,%rbp
push   %rbx
nop    %rbp
pop    %rbp
[…]
<__start_mcount_loc>:
[...]
<___end_mcount_loc>:
```
Finding Mcount

vmlinux:

<schedule>:
push   %rbp
mov    %rsp,%rbp
nop
pop    %rbp
[…]
<preempt_schedule_irq>:
push   %rbp
mov    %rsp,%rbp
push   %rbx
nop
pop    %rbp
[…]
<_cond_resched>:
push   %rbp
mov    %rsp,%rbp
push   %rbx
nop
pop    %rbp
[…]
<yield>:
push   %rbp
mov    %rsp,%rbp
push   %rbx
nop
pop    %rbp
[…]

__start_mcount_loc>:
[...]
__end_mcount_loc>: 

[...]

[...]

[...]

[...]

[...]

[...]

[...]

[...]
Finding Mcount

vmlinux:

```
<schedule>:
push %rbp
mov %rsp,%rbp
nop
pop %rbp
[…]
<preempt_schedule_irq>:
push %rbp
mov %rsp,%rbp
push %rbx
nop
pop %rbp
[…]
<_cond_resched>:
push %rbp
mov %rsp,%rbp
push %rbx
nop
pop %rbp
[…]
<yield>:
push %rbp
mov %rsp,%rbp
push %rbx
nop
pop %rbp
[…]
```
What about tracing?

- Need a way to enable tracing
- We threw away the mcount section
- The mcount section wasn’t enough for us
- Tracing also requires saving state
struct dyn_ftrace

struct dyn_ftrace {
   unsigned long ip;  /* address of mcount call-site */
   unsigned long flags;
   struct dyn_arch_ftrace arch;
};
**struct dyn_ftrace**

```c
struct dyn_ftrace {
    unsigned long ip; /* address of mcount call-site */
    unsigned long flags;
    struct dyn_arch_ftrace arch;
};
```

**arch/x86/include/asm/ftrace.h:**

```c
struct dyn_arch_ftrace {
    /* No extra data needed for x86 */
};
```
struct dyn_ftrace
{
    unsigned long ip; /* address of mcount call-site */
    unsigned long flags;
    struct dyn_arch_ftrace arch;
};

arch/powerpc/include/asm/ftrace.h:
struct dyn_arch_ftrace {
    struct module *mod;
};
Tracing data

• Copy from mcount_loc before deleting
• Sorted for quick lookup
• Allocated in groups of pages
  • details out of scope for this talk
• Data reported at boot up
  
  # dmesg |grep ftrace
  [ 0.139656] ftrace: allocating 24683 entries in 97 pages

  • Allocated 24,683 dyn_ftrace structures
  • Used up 97 (4K) pages to do so
  • Total of 397,312 bytes
Tracing data

vmlinux:

```
<schedule>:
push   %rbp
mov    %rsp,%rbp
nop
pop    %rbp
[

<preempt_schedule_irq>:
push   %rbp
mov    %rsp,%rbp
push   %rbx
nop
pop    %rbp
[

<_cond_resched>:
push   %rbp
mov    %rsp,%rbp
push   %rbx
nop
pop    %rbp
[

<yield>:
push   %rbp
mov    %rsp,%rbp
push   %rbx
nop
pop    %rbp
[

<__start_mcount_loc>:
[

<___end_mcount_loc>:
```

```
<__start_mcount_loc>:
[

<___end_mcount_loc>:
```
Tracing data

```plaintext
<trace_pages>

# cat available_filter_functions
put_page
__get_page_tail
put_pages_list
get_kernel_pages
new_sync_read
vfs_setpos
fixed_size_llseek
default_llseek
schedule
preempt_schedule_irq
_cond_resched
yield
```

```plaintext
ip = 0xffffffff81087ad4
flags = 0
ip = 0xffffffff81087b14
flags = 0
ip = 0xffffffff81087bd5
flags = 0
ip = 0xffffffff81087c41
flags = 0
ip = 0xffffffff810a7aa0
flags = 0
ip = 0xffffffff810a7bd4
flags = 0
ip = 0xffffffff810a7d34
flags = 0
ip = 0xffffffff810a7d7d
flags = 0
ip = 0xffffffff810f45f4
flags = 0
ip = 0xffffffff810f4635
flags = 0
ip = 0xffffffff810f4684
flags = 0
ip = 0xffffffff810f4734
flags = 0
[...]
```
Tracing data

```
<frtrace_pages>

| ip          | 0xfffffffff81087ad4 |
| flags       | 0 |
| ip          | 0xfffffffff81087b14 |
| flags       | 0 |
| ip          | 0xfffffffff81087bd5 |
| flags       | 0 |
| ip          | 0xfffffffff81087c41 |
| flags       | 0 |
| ip          | 0xfffffffff810a7aa0 |
| flags       | 0 |
| ip          | 0xfffffffff810a7bd4 |
| flags       | 0 |
| ip          | 0xfffffffff810a7d34 |
| flags       | 0 |
| ip          | 0xfffffffff810a7d7d |
| flags       | 0 |
| ip          | 0xfffffffff810f45f4 |
| flags       | 0 |
| ip          | 0xfffffffff810f4635 |
| flags       | 0 |
| ip          | 0xfffffffff810f4684 |
| flags       | 0 |
| ip          | 0xfffffffff810f4734 |
| flags       | 0 |

[...]  
```
dyn_ftrace.flags

- First 29 bits are for counter
  - Every registered callback increments +1
- bit 29 (starts from zero) – ENABLED
- bit 30 – REGS
- bit 31 – REGS_EN
Enabling tracing

vmlinux:

```
<schedule>:
push   %rbp
mov    %rsp,%rbp
nop
pop    %rbp
[...]
<preempt_schedule_irq>:
push   %rbp
mov    %rsp,%rbp
push   %rbx
nop
pop    %rbp
[...]
<_cond_resched>:
push   %rbp
mov    %rsp,%rbp
push   %rbx
nop
pop    %rbp
[...]
<yield>:
push   %rbp
mov    %rsp,%rbp
push   %rbx
nop
pop    %rbp
[...]
```

```
<trace_pages>

ip    = 0xffffffff81087ad4
flags = 0
ip    = 0xffffffff81087b14
flags = 0
ip    = 0xffffffff81087bd5
flags = 0
ip    = 0xffffffff81087c41
flags = 0
ip    = 0xffffffff810a7aa0
flags = 0
ip    = 0xffffffff810a7bd4
flags = 0
ip    = 0xffffffff810a7d34
flags = 0
ip    = 0xffffffff810a7d7d
flags = 0
ip    = 0xffffffff810f45f4
flags = 0x20000001
ip    = 0xffffffff810f4635
flags = 0
ip    = 0xffffffff810f4684
flags = 0
ip    = 0xffffffff810f4734
flags = 0xa0000001
[...]```
Enabling tracing

vmlinux:

<schedule>:
push   %rbp
mov    %rsp,%rbp
call ftrace_caller
pop    %rbp
[…]
<preempt_schedule_irq>:
push   %rbp
mov    %rsp,%rbp
push   %rbx
nop
pop    %rbp
[…]
<__cond_resched>:
push   %rbp
mov    %rsp,%rbp
push   %rbx
nop
pop    %rbp
[…]
<yield>:
push   %rbp
mov    %rsp,%rbp
push   %rbx
call ftrace_regs_caller
pop    %rbp
[…]

<frtrace_pages>

ip    = 0xffffffff81087ad4
flags = 0
ip    = 0xffffffff81087b14
flags = 0
ip    = 0xffffffff81087bd5
flags = 0
ip    = 0xffffffff81087c41
flags = 0
ip    = 0xffffffff810a7aa0
flags = 0
ip    = 0xffffffff810a7bd4
flags = 0
ip    = 0xffffffff810a7d34
flags = 0
ip    = 0xffffffff810a7d7d
flags = 0
ip    = 0xffffffff810f45f4
flags = 0x20000001
ip    = 0xffffffff810f4635
flags = 0
ip    = 0xffffffff810f4684
flags = 0
ip    = 0xffffffff810f4734
flags = 0xe0000001
[…]

Modifying code at runtime

- Not the same as at boot up
- SMP boxes must be careful
- Other CPUs may be executing that code
- x86 has non uniform machine instructions
- Instructions may cross cache boundaries
Modifying code at runtime

<schedule>:

55                      push   %rbp
48 89 e5                mov    %rsp,%rbp
0f 1f 44 00 00          nop
5d                      pop    %rbp
48 8b 04 25 80 d0 15    mov    0xffffffff8115d080,%rax
81                      mov     0xffffffff810f40a0 <__schedule>
48 8b 00                mov     (%rax),%rax
e9 96 fa ff ff         jmpq   fffffff810f40a0 <__schedule>
66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
Modifying code at runtime

```assembly
<schedule>:
  55                      push   %rbp
  48 89 e5                mov    %rsp,%rbp
  e8 37 2e 00 00          callq  ffffffff810f7430 <ftrace_caller>
  5d                      pop    %rbp
  48 8b 04 25 80 d0 15    mov    0xffffffff8115d080,%rax
  81
  48 8b 00                mov    (%rax),%rax
  e9 96 fa ff ff          jmpq   ffffffff810f40a0 __schedule>
  66 0f 1f 44 00 00        nopw   0x0(%rax,%rax,1)
```
Modifying code at runtime

CPU 0

<schedule>:
55
48 89 e5
0f 1f 44 00 00
5d
48 8b 04 25 80 d0 15
81
48 8b 00
e9 96 fa ff ff
66 0f 1f 44 00 00

CPU 1

<schedule>:
55
48 89 e5
0f 1f 44 00 00
5d
48 8b 04 25 80 d0 15
81
48 8b 00
e9 96 fa ff ff
66 0f 1f 44 00 00
Modifying code at runtime

CPU 0

<schedule>:
  55
  48 89 e5
  e8 37 2e 00 00
  5d
  48 8b 04 25 80 d0 15
  81
  48 8b 00
  e9 96 fa ff ff
  66 0f 1f 44 00 00

CPU 1

<schedule>:
  55
  48 89 e5
  0f 1f 44 00 00
  5d
  48 8b 04 25 80 d0 15
  81
  48 8b 00
  e9 96 fa ff ff
  66 0f 1f 44 00 00
Modifying code at runtime

CPU 0

<schedule>:
  55
  48 89 e5
  e8 37 2e 00 00
  5d
  48 8b 04 25 80 d0 15
  81
  48 8b 00
  e9 96 fa ff ff
  66 0f 1f 44 00 00

CPU 1

<schedule>:
  55
  48 89 e5
  0f 1f 2e 00 00
  5d
  48 8b 04 25 80 d0 15
  81
  48 8b 00
  e9 96 fa ff ff
  66 0f 1f 44 00 00
0f 1f 2e 00???
0f 1f 2e 00???

- BOOM!
- CRASH!
- GENERAL PROTECTION FAULT!
- REBOOT!
How to go from this

<schedule>:

```
55                      push %rbp
48 89 e5                mov %rsp,%rbp
0f 1f 44 00 00          nop
5d                      pop %rbp
48 8b 04 25 80 d0 15    mov 0xffffffff8115d080,%rax
81                      mov 0xffffffff8115d080,%rax
48 8b 00                mov (%rax),%rax
e9 96 fa ff ff          jmpq fffffffff810f40a0 <__schedule>
66 0f 1f 44 00 00        nopw 0x0(%rax,%rax,1)
```
<schedule>:

55                      push %rbp
48 89 e5                mov %rsp,%rbp
e8 37 2e 00 00          callq ffffffff810f7430 <ftrace_caller>
5d                      pop %rbp
48 8b 04 25 80 d0 15    mov 0xffffffff8115d080,%rax
81                      jmpq ffffffff810f40a0 <__schedule>
48 8b 00                mov %rax
48 8b 00                (%rax),%rax
66 0f 1f 44 00 00       jmpq ffffffff810f40a0 <__schedule>
66 0f 1f 44 00 00       nopw 0x0(%rax,%rax,1)
Breakpoints
**Breakpoints**

<table>
<thead>
<tr>
<th>&lt;schedule&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>push %rbp</td>
</tr>
<tr>
<td>mov %rsp,%rbp</td>
</tr>
<tr>
<td>nop</td>
</tr>
<tr>
<td>pop %rbp</td>
</tr>
<tr>
<td>mov 0xffffffff8115d080,%rax</td>
</tr>
<tr>
<td>mov 0xffffffff8115d080,%rax</td>
</tr>
<tr>
<td>jmpq fffffff810f40a0 &lt;__schedule&gt;</td>
</tr>
<tr>
<td>nopw 0x0(%rax,%rax,1)</td>
</tr>
</tbody>
</table>
Breakpoints

<table>
<thead>
<tr>
<th>Address</th>
<th>Machine Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>5d</td>
<td>pop %rbp</td>
</tr>
<tr>
<td>48 89 e5</td>
<td>mov %rsp,%rbp</td>
<td></td>
</tr>
<tr>
<td>cc 1f 44 00 00</td>
<td>&lt;bp&gt;nop</td>
<td></td>
</tr>
<tr>
<td>5d</td>
<td>pop %rbp</td>
<td></td>
</tr>
<tr>
<td>48 8b 04 25 80 d0 15</td>
<td>mov 0xffffffff8115d080,%rax</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>48 8b 00</td>
<td>mov (%rax),%rax</td>
<td></td>
</tr>
<tr>
<td>e9 96 fa ff ff</td>
<td>jmpq fffffffff810f40a0</td>
<td>&lt;__schedule&gt;</td>
</tr>
<tr>
<td>66 0f 1f 44 00 00</td>
<td>nopw 0x0(%rax,%rax,1)</td>
<td></td>
</tr>
</tbody>
</table>
Breakpoints

\texttt{<schedule>:
  55                  \texttt{push} \texttt{%rbp}
  48 89 e5           \texttt{mov} \texttt{%rsp},\texttt{%rbp}
  \texttt{cc} 37 2e 00 00  \texttt{<bp>callq} \texttt{fffffffff810f7430 <ftrace_caller>}
  5d                  \texttt{pop} \texttt{%rbp}
  48 8b 04 25 80 d0 15 \texttt{mov} \texttt{0xffffffff8115d080},\texttt{%rax}
  81
  48 8b 00
  e9 96 fa ff ff     \texttt{mov} \texttt{(%rax),%rax}
  66 0f 1f 44 00 00   \texttt{jmpq} \texttt{fffffffff810f40a0 <__schedule>}
  nopw 0x0(%rax,%rax,1)\texttt{<bp>}}</texttt>
Breakpoints

```
<schedule>:
  55                      push   %rbp
  48 89 e5                mov    %rsp,%rbp
  e8 37 2e 00 00          callq  fffffffff810f7430 <ftrace_caller>
  5d                      pop    %rbp
  48 8b 04 25 80 d0 15    mov    0xffffffff8115d080,%rax
  81                      mov    0xffffffff8115d080,%rax
  48 8b 00                mov    (%rax),%rax
  e9 96 fa ff ff          jmpq   fffffffff810f40a0 <__schedule>
  66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
```
Registering with Ftrace

- Call to register_ftrace_function()
- Requires an ftrace_ops descriptor
- Static ftrace_ops
  - function and function_graph
  - function probes (schedule:traceoff)
  - stack tracer
  - latency tracers
- Dynamic ftrace_ops
  - perf
  - kprobes
```c
struct ftrace_ops {
    ftrace_func_t func;
    struct ftrace_ops *next;
    unsigned long flags;
    int __percpu *disabled;
    void *private;

    ifdef CONFIG_DYNAMIC_FTRACE
    struct ftrace_hash *notrace_hash;
    struct ftrace_hash *filter_hash;
    struct mutex regex_lock;
    endif
};
```
ftrace_ops.flags

- ENABLED
  - set by ftrace, when ops is recording
- DYNAMIC
  - set by ftrace when ops is dynamically allocated
- CONTROL
  - set by perf
ftrace_ops.flags

- SAVE_REGS
  - set by caller, to record regs
  - fails if saving regs is not supported
- SAVE_REGS_IF_SUPPORTED
  - set by caller, save regs if supported
  - doesn’t fail register if not supported
- RECURSION_SAFE
  - If ftrace_ops.func handles recursion
  - Otherwise, ftrace will handle it
ftrace_ops.flags

- **STUB**
  - used by ftrace for stub functions

- **INITIALIZED**
  - used by ftrace when ftrace_ops is first used

- **DELETED**
  - ftrace_ops has been deleted
  - used by ftrace buffer instances
ftrace_ops hashes

- regex_lock
  - used to protect the hashes
- notrace_hash
  - what functions not to trace
  - empty means OK to trace all
- filter_hash
  - what functions to trace
  - empty means to trace all
- Functions in notrace_hash will not be traced even if they exist in filter_hash
ftrace_caller trampoline

```assembly
<schedule>:
push %rbp
mov %rsp,%rbp
call ftrace_caller
pop %rbp
[…]
<preempt_schedule_irq>:
push %rbp
mov %rsp,%rbp
push %rbx
nop
pop %rbp
[…]
<cond_resched>:
push %rbp
mov %rsp,%rbp
push %rbx
nop
pop %rbp
[…]
<yield>:
push %rbp
mov %rsp,%rbp
push %rbx
call ftrace_regs_caller
pop %rbp
[…]
```

```assembly
ftrace_caller:
save regs
load args
ftrace_call:
call ftrace_stub
restore regs
ftrace_stub:
retq
```
ftrace_caller trampoline

```asm
<schedule>:
push %rbp
mov %rsp,%rbp
```call ftrace_caller
```asm
pop %rbp
[
```...
<preempt_schedule_irq>:
push %rbp
mov %rsp,%rbp
push %rbx
nop
pop %rbp
[
```...
<cond_resched>:
push %rbp
mov %rsp,%rbp
push %rbx
nop
pop %rbp
[
```...
<yield>:
push %rbp
mov %rsp,%rbp
push %rbx
```call ftrace_regs_caller
```asm
pop %rbp
[
```...

ftrace_caller:
save regs
load args

ftrace_call:
call func_trace
restore regs

ftrace_stub:
retq

void func_trace()
{
    /* trace */
}
```
ftrace_caller trampoline

```
schedule:
    push   %rbp
    mov    %rsp,%rbp
    call ftrace_caller
    pop    %rbp
    [...]

preempt_schedule_irq:
    push   %rbp
    mov    %rsp,%rbp
    push   %rbx
    nop
    pop    %rbp
    [...]

_cond_resched:
    push   %rbp
    mov    %rsp,%rbp
    push   %rbx
    nop
    pop    %rbp
    [...]

yield:
    push   %rbp
    mov    %rsp,%rbp
    push   %rbx
    call ftrace_regs_caller
    pop    %rbp
    [...]
```

```
ftrace_caller:
    save regs
    load args
    ftrace_call:
    call func_trace
    restore regs
ftrace_stub:
    retq
```

```
void func_trace()
{
    /* trace */
}
```

```
ftrace_ops.func
```
Multiple callbacks?

- Direct call works fine
- Multiple calls requires a list operation
- All functions being traced will call the list function
Multiple callbacks?

```c
void func1_trace()
{
    /* trace */
}

void func2_trace()
{
    /* trace */
}

void list_func()
{
    /* iterate */
}
```

```c
void ftrace_stub()
{
    retq
}

ftrace_call:  
    load args
    call list_func
    restore regs

ftrace_caller:  
    save regs
    call ftrace_caller
    pop %rbp

[schedule]:
    push %rbp
    mov %rsp,%rbp
    call ftrace_caller
    pop %rbp

[preempt_schedule_irq]:
    push %rbp
    mov %rsp,%rbp
    push %rbx
    nop
    pop %rbp

[cond_resched]:
    push %rbp
    mov %rsp,%rbp
    push %rbx
    nop
    pop %rbp

[yield]:
    push %rbp
    mov %rsp,%rbp
    push %rbx
    call ftrace_regs_caller
    pop %rbp
```

```c
ftrace_regs_caller:
    push %rbx
    call ftrace_caller
    pop %rbp
```

```c
void func1_trace()
{
    /* trace */
}

void list_func()
{
    /* iterate */
}

void func2_trace()
{
    /* trace */
}
```
Example

- Run function tracer on all functions
- Run perf on just the scheduler
Example

Want to trace `do_IRQ()`?

`list_func()`

Nope

`perf`

Yep!

`function_tracer`
Example

Want to trace `raw_spin_lock()`?

list_func()

Nope

perf

Yep!

function_tracer
Example

Want to trace `rcu_read_lock()`?

list_func()

Nope

perf

Yep!

function_tracer
Example

Want to trace `rcu_read_unlock()`?

list_func()

Nope

per

Yep!

function_tracer
Example

Want to trace `schedule()`?

Yep! list_func()

Yep! perf

Yep! function_tracer
Multiple callbacks?

```c
void function_trace()
{
    /* function tracing */
}

void list_func()
{
    /* iterate and check hash of ops */
}

void perf_func()
{
    /* perf profiling */
}
```
Multiple callbacks?

```
void function_trace()
{
    /* function tracing */
}

void list_func()
{
    /* iterate and check hash of ops */
}

void perf_func()
{
    /* perf profiling */
}

dynamic_trampoline:
    save regs
    load args
    call function_trace
    restore regs
    retq

ftrace_caller:
    save regs
    load args
    ftrace_call
    restore regs
    ftrace_stub:
    retq

<schedule>:
    push %rbp
    mov %rsp,%rbp
    call ftrace_caller
    pop %rbp

<preempt_schedule_irq>:
    push %rbp
    mov %rsp,%rbp
    push %rbx
    call ftrace_caller
    pop %rbp

<_cond_resched>:
    push %rbp
    mov %rsp,%rbp
    push %rbx
    call ftrace_caller
    pop %rbp

<yield>:
    push %rbp
    mov %rsp,%rbp
    push %rbx
    call ftrace_caller
    pop %rbp
```
Problems with Dynamic Trampolines

- When can you free them?
- When are they not in use?
- Would RCU help?
Dynamic trampolines

```
dynamic_trampoline:
    save regs
    load args
    call function_trace
    restore regs
    retq
```
Dynamic trampolines

Task

Preempted!

dynamic_trampoline:
  save regs
  load args
  call function_trace
  restore regs
  retq
What about dynamic ftrace_ops

- Remember, there’s a difference between dynamic and static ftrace_ops
- Ftrace detects ftrace_ops that are dynamic
- Always uses the list function
  - it disables preemption
  - and is static
Dynamic ftrace_ops

```
<schedule>:
  push  %rbp
  mov   %rsp,%rbp
  call ftrace_caller
  pop   %rbp
  [...]
<preempt_schedule_irq>:
  push  %rbp
  mov   %rsp,%rbp
  push  %rbx
  nop
  pop   %rbp
  [...]
<_cond_resched>:
  push  %rbp
  mov   %rsp,%rbp
  push  %rbx
  nop
  pop   %rbp
  [...]
<yield>:
  push  %rbp
  mov   %rsp,%rbp
  push  %rbx
  call ftrace_regs_caller
  pop   %rbp
  [...]
```

```
ftrace_caller:
  save regs
  load args
  ftrace_call:
  call list_func
  restore regs
  ftrace_stub:
  retq
```

```
void list_func()
{
  preempt_disable_notrace();
  /* iterate */
  preempt_enable_notrace();
}
```

```
void dynamic_ops_func()
{
  /* trace */
}
```
Knowing when to free

- If there was a way to know no more tasks were on the trampoline or function
- There will be a way in coming 3.18
call_rcu_tasks()

• Call a function after all tasks
  • have voluntarily scheduled
  • in userspace
  • are idle
Dynamic trampolines

```assembly
<schedule>:
push   %rbp
mov    %rsp,%rbp
call dynamic_tramp
pop    %rbp
[...]

dynamic_trampoline:
save regs
load args
call function_trace
restore regs
retq
```
Dynamic trampolines

```
<schedule>:
push   %rbp
mov    %rsp,%rbp
call   dynamic_tramp
pop    %rbp
[...]
```

Preempted!

```
dynamic_trampoline:
save   regs
load   args
call   function_trace
restore_regs
retq
```
Dynamic trampolines

<schedule>:
push   %rbp
mov    %rsp,%rbp
**call ftrace_stub**
pop    %rbp
[…]

Preempted!

dynamic_trampoline:
save regs
load args
**call function_trace**
restore regs
retq
Dynamic trampolines

```
<schedule>:
push   %rbp
mov    %rsp,%rbp
call ftrace_stub
pop    %rbp
[...]
```

```
dynamic_trampoline:
save regs
load args
call function_trace
restore regs
retq
```

`call_rcu_task()`
Dynamic trampolines

\[
\text{<schedule>}: \\
push \ %rbp \\
mov \ %rsp,%rbp \\
call \texttt{ftrace_stub} \\
pop \ %rbp \\
[...]
\]

dynamic_trampoline: \\
save \texttt{regs} \\
load \texttt{args} \\
call \texttt{function_trace} \\
restore \texttt{regs} \\
retq

- Voluntary schedule
- In idle
- In userspace

call\_rcu\_task()
Dynamic trampolines

<schedule>:
push  %rbp
mov   %rsp,%rbp
**call ftrace_stub**
pop   %rbp
[…]

- Voluntary schedule
- In idle
- In userspace

```assembly
dynamic_trampoline:
  save regs
  load args
  call function_trace
  restore regs
  retq
```
fentry

- mcount can’t record parameters
- New feature of gcc
  - starting with gcc 4.6.0
    - Added by Andi Kleen
  - for x86_64 only (for now)
- gcc -pg -mfentry
fentry

<schedule>:

55                      push  %rbp
48 89 e5                mov    %rsp,%rbp
e8 37 2e 00 00          callq  ffffffff810f7430 <mcount>
5d                      pop    %rbp
48 8b 04 25 80 d0 15    mov    0xffffffff8115d080,%rax
81                      mov    (%rax),%rax
48 8b 00                jmpq   ffffffff810f40a0 <__schedule>
66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
<posix_cpu_timer_set>:

55                      push   %rbp
48 89 e5                mov    %rsp,%rbp
41 57                   push   %r15
41 56                   push   %r14
41 55                   push   %r13
41 54                   push   %r12
53                      push   %rbx
48 83 ec 30             sub    $0x30,%rsp
e8 1a 81 0b 00          callq  ffffffff810f7430 <mcount>
48 8b 47 70             mov    0x70(%rdi),%rax
49 89 ff                mov    %rdi,%r15
<posix_cpu_timer_set>:

e8 eb 3c 0b 00
41 57
41 56
49 89 ff
41 55
41 54
49 89 d5
55
53

callq fffffffff810f0af0 <__fentry__>
push %r15
push %r14
mov %rdi,%r15
push %r13
push %r12
mov %rdx,%r13
push %rbp
push %rbx
<posix_cpu_timer_set>:
call ftrace_caller
push %r15
push %r14
mov %rdi,%r15
push %r13
push %r12

ftrace_caller:
    save regs
    load args
ftrace_call:
    call func_trace
    restore regs
ftrace_stub:
    retq

void func_trace()
{
    /* trace */
}

Live Kernel Patching!

```c
void lkp()
{
    /* change
    return reg */
}
```

```assembly
<posix_cpu_timer_set>:
call ftrace_caller
push   %r15
push   %r14
mov    %rdi,%r15
push   %r13
push   %r12

ftrace_caller:
    save regs
    load args
ftrace_call:
    call func_trace
    restore regs
ftrace_stub:
    retq

<posix_cpu_timer_set>:
nop
push   %r15
push   %r14
mov    %rdi,%r15
push   %r13
push   %r12
```
Another Solution

vmlinux:

```c
<schedule>:
  nop
  push %rbp
  mov %rsp,%rbp
  pop %rbp
  [...]  
<preempt_schedule_irq>:
  nop
  push %rbp
  mov %rsp,%rbp
  push %rbx
  pop %rbp
  [...]  
<_cond_resched>:
  nop
  push %rbp
  mov %rsp,%rbp
  push %rbx
  pop %rbp
  [...]  
<yield>:
  nop
  push %rbp
  mov %rsp,%rbp
  push %rbx
  pop %rbp
  [...]  
```

```text
ip    = 0xffffffff81087ad4
flags = 0
ip    = 0xffffffff81087b14
flags = 0
ip    = 0xffffffff81087bd5
flags = 0
ip    = 0xffffffff81087c41
flags = 0
ip    = 0xffffffff81087a7aa0
flags = 0
ip    = 0xffffffff81087a7bd4
flags = 0
ip    = 0xffffffff81087a7d34
flags = 0
ip    = 0xffffffff81087a7d7d
flags = 0
ip    = 0xffffffff8108f45f4
flags = 0
ip    = 0xffffffff8108f4635
flags = 0
ip    = 0xffffffff8108f4684
flags = 0
ip    = 0xffffffff8108f4734
flags = 0
[...]```
Another Solution

vmlinux:

```assembly
<schedule>:
  nop
  push %rbp
  mov %rsp,%rbp
  pop %rbp
  [...]
</schedule>

<preempt_schedule_irq>:
  nop
  push %rbp
  mov %rsp,%rbp
  push %rbx
  pop %rbp
  [...]
</preempt_schedule_irq>

<_cond_resched>:
  nop
  push %rbp
  mov %rsp,%rbp
  push %rbx
  pop %rbp
  [...]
</_cond_resched>

<yield>:
  nop
  push %rbp
  mov %rsp,%rbp
  push %rbx
  pop %rbp
  [...]
</yield>

<ftrace_pages>

```
Another Solution

vmlinux:

```c
<schedule>:
  nop
  push  %rbp
  mov   %rsp,%rbp
  pop   %rbp
[...]
<preempt_schedule_irq>:
  nop
  push  %rbp
  mov   %rsp,%rbp
  push  %rbx
  pop   %rbp
[...]
<_cond_resched>:
  nop
  push  %rbp
  mov   %rsp,%rbp
  push  %rbx
  pop   %rbp
[...]
<yield>:
  nop
  push  %rbp
  mov   %rsp,%rbp
  push  %rbx
  pop   %rbp
[...]
</ftrace_pages>
```

```c
ip    = 0xffffffff81087ad4
flags = 0
ip    = 0xffffffff81087b14
flags = 0
ip    = 0xffffffff81087bd5
flags = 0
ip    = 0xffffffffa0014466
flags = 0
ip    = 0xffffffff810a7aa0
flags = 0
ip    = 0xffffffff810a7bd4
flags = 0
ip    = 0xffffffff810a7d34
flags = 0
ip    = 0xffffffff810a7d7d
flags = 0
ip    = 0xffffffff810f45f4
flags = 0
ip    = 0xffffffff810f4635
flags = 0
ip    = 0xffffffff810f4684
flags = 0
ip    = 0xffffffff810f4734
flags = 0
[...]
```
Another Solution

vmlinux:

```
<schedule>:
  nop
  push %rbp
  mov %rsp,%rbp
  pop %rbp

<preempt_schedule_irq>:
  jmp preempt_sched2
  push %rbp
  mov %rsp,%rbp
  push %rbx
  pop %rbp

<_cond_resched>:
  nop
  push %rbp
  mov %rsp,%rbp
  push %rbx
  pop %rbp

<yield>:
  nop
  push %rbp
  mov %rsp,%rbp
  push %rbx
  pop %rbp
```

```
<preempt_schedule_irq>:
  push %r15
  push %r14
  mov %rdi,%r15
  push %r13
  push %r12
```

```
<trace_pages>

ip    = 0xffffffff81087ad4
flags = 0
ip    = 0xffffffff81087b14
flags = 0
ip    = 0xffffffff81087bd5
flags = 0
ip    = 0xffffffffa0014466
flags = 0
ip    = 0xffffffff810a7aa0
flags = 0
ip    = 0xffffffff810a7bd4
flags = 0
ip    = 0xffffffff810a7d34
flags = 0
ip    = 0xffffffff810a7d7d
flags = 0
ip    = 0xffffffff810f45f4
flags = 0
ip    = 0xffffffff810f4635
flags = 0
ip    = 0xffffffff810f4684
flags = 0
ip    = 0xffffffff810f4734
flags = 0
```

Instead of this

```c
void lkp()
{
    /* change
     * return reg */
}
```

```c
<posix_cpu_timer_set>:
call ftrace_caller
push   %r15
push   %r14
mov    %rdi,%r15
push   %r13
push   %r12
```

```asm
ftrace_caller:
save regs
load args
ftrace_call:
call func_trace
restore regs
ftrace_stub:
retq
```

```asm
<posix_cpu_timer_set>:
nop
push   %r15
push   %r14
mov    %rdi,%r15
push   %r13
push   %r12
```
Have this!

```plaintext
<posix_cpu_timer_set>:
    jmp posix_cpu2
    push %r15
    push %r14
    mov %rdi,%r15
    push %r13
    push %r12

<posix_cpu_timer_set>:
    call ftrace_caller
    push %r15
    push %r14
    mov %rdi,%r15
    push %r13
    push %r12
```

```plaintext
ftrace_caller:
    save regs
    load args
ftrace_call:
    call func_trace
    restore regs
ftrace_stub:
    retq
```
Questions?
Questions?

Yeah right!
Like we have time