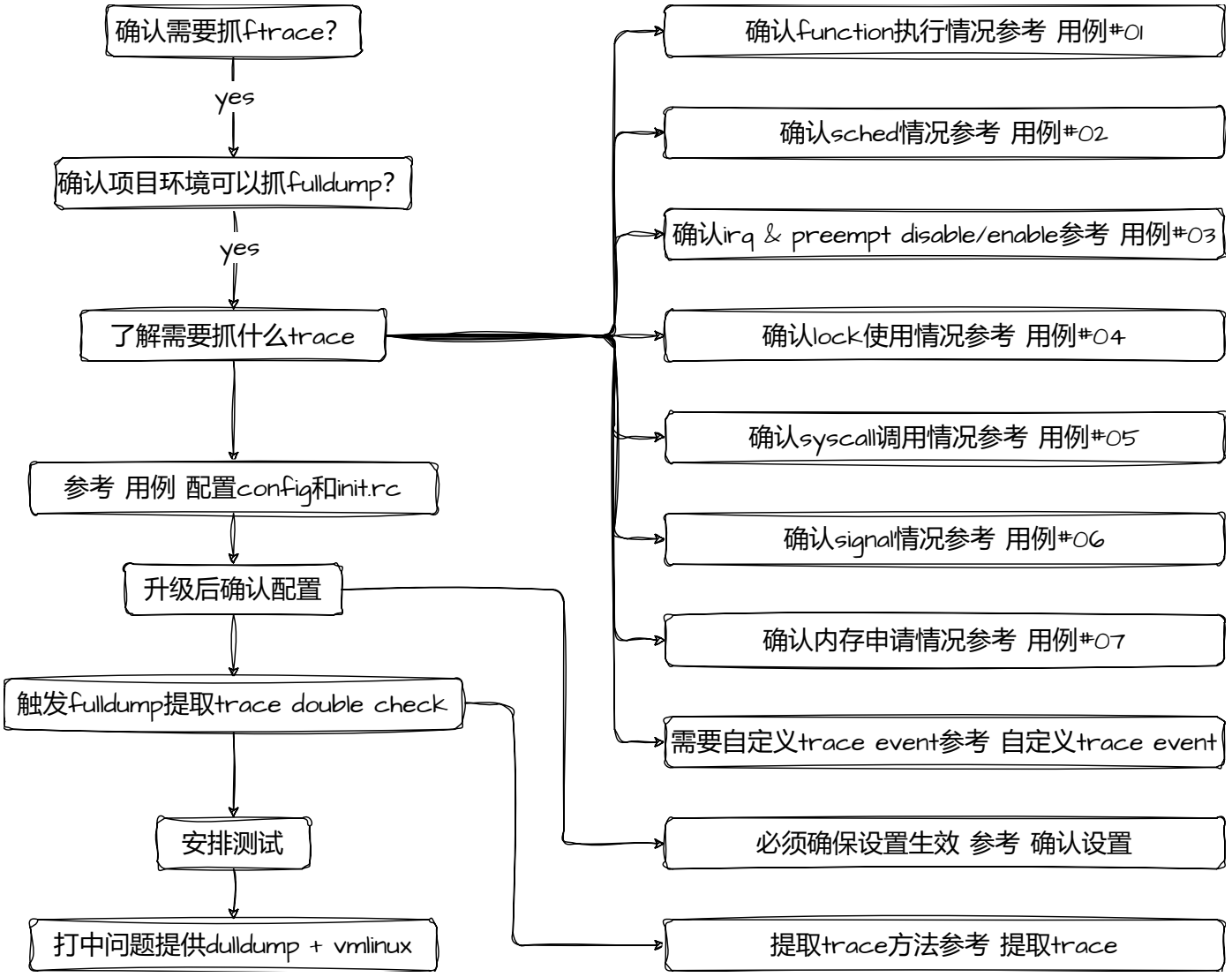
01.quick start

quick start

快速使用



Edit

介绍

本篇将介绍ftrace在android system stability问题分析中的基本使用、操作方法和配置方法。稳定性问题通常导致panic，panic后会进行内存转储，从内存转储中可以提取trace buffer中的记录，因此很多system stability问题都可以利用kernel原生的tracer和trace event进行分析，甚至可以自定义trace event来满足分析需求。本篇意在用于android system stabilty debug中ftrace使用的快速参考。

版本

本篇编写基于`linux-6.1`。

帮助文档

在首次使用ftrace时建议阅读内核文档ftrace和event了解ftrace说明，以上文档是对ftrace常见用法介绍最为完整的参考资料。

使用场景

使用ftrace主要用到ftracer和trace event。tracer常用的是function_graph tracer；trace event是内核中定义的静态插桩点，内核开发人员确保trace event接口的稳定，不同版本内核往往不会发生变化，具有跨内

核版本稳定特点。以下是常用的场景举例：

- #01 抓函数的调用流程使用function_graph tracer
- #02 抓调度情况使用sched event
- #03 抓中断和抢占情况使用preemptirq & irq event
- #04 抓锁使用情况使用lock event
- #05 抓系统调用情况使用syscall event
- #06 抓signal情况使用signal event
- #07 抓内存申请情况使用kmem event
- #08 抓block情况使用block event
- #09 抓task创建、退出等情况使用task event
- #10 抓f2fs情况使用f2fs event
- #11 抓kworker情况使用workqueue event
- #12 抓cpufreq变化使用cpufreq event
- #13 分析代码流程借助trace evne埋点理解关键code位置
- #14 自定义trace event用于log记录

以上列举一些可能用到的场景，实际使用中可以用到的场景更多，以上列表中常用top 3为#02、#03、#04。

instance

因为后续介绍的方法都是在对应的instance中进行设置的，在介绍其他内容之前先介instance，意为实例，参考Documentation ftrace了解更多信息。一个instance有一个trace buffer，用于保存这个instance的trace记录。在`/sys/kernel/tracing/trace`读到的trace记录是默认trace buffer中的内容，ftrace允许创建更多trace buffer，就是通过创建instance实现的，不同的trace buffer可以保存不同的trace记录。
在tracing目录下存在一个instances目录，即`/sys/kernel/tracing/instances`，要创建一个instance只需要在其下创建一个目录即可，目录的名称就是instance的name，例如，创建name为sched的instance用于保存sched trace则可以：

```
# mkdir /sys/kernel/tracing/instances/sched
```

查看一下新创建的instance sched下的内容：

```
/ # ls /sys/kernel/tracing/instances/sched/
available_tracers       set_event               trace_clock
buffer_percent          set_event_notrace_pid   trace_marker
buffer_size_kb          set_event_pid           trace_marker_raw
buffer_total_size_kb    set_ftrace_filter       trace_options
current_tracer          set_ftrace_notrace      trace_pipe
error_log               set_ftrace_notrace_pid  tracing_cpumask
events                  set_ftrace_pid          tracing_max_latency
```

```
free_buffer             snapshot                tracing_on
options                 timestamp_mode
per_cpu                 trace
```

可以发现sched下自动创建了很多文件，几乎与tracing下内容完全相同。

提取trace

先说明trace记录如何提取。

常用设置

提醒: 因为有些event依赖内核配置，如果找不到对应event，建议先检查内核配置。
提醒：以下为了方便介绍使用`echo`命令进行设置，在android系统测试中，需要在init.rc中通过write来设置，为了避免kernel panic后设置消失。

设置tracer

只介绍最常用的function_graph tracer。

开启tracer

设置要使用的tracer为functon_graph及确认：

```
/ # echo function_graph > /sys/kernel/tracing/current_tracer
/ # cat /sys/kernel/tracing/current_tracer
function_graph
```

查看系统支持哪些function的跟踪、设置要跟踪的function及确认，例如查看`vfs_open`：

```
/ # cat /sys/kernel/tracing/available_filter_functions | grep vfs_open
vfs_open
/ # echo vfs_open > /sys/kernel/tracing/set_graph_function
/ # cat /sys/kernel/tracing/set_graph_function
vfs_open
```

trace示例

针对`vfs_open`抓到的trace例子，如下：

```
/ # cat /sys/kernel/tracing/trace
# tracer: function_graph
#
# CPU  DURATION                  FUNCTION CALLS
# |     |    |                     |   |   |   |
...
 0)               |  vfs_open() {
```

```
0)               |    do_dentry_open() {
0)               |      path_get() {
0)   1.407 us    |        mntget();
0)   5.426 us    |      }
0)   1.074 us    |      try_module_get();
0)   1.555 us    |      security_file_open();
0)               |      nfs_file_open() {
0)               |        nfs_open() {
0)               |          alloc_nfs_open_context() {
0)               |            kmalloc_trace() {
0)               |              __kmem_cache_alloc_node() {
0)   1.037 us    |                should_failslab();
0)   4.574 us    |              }
0)   7.833 us    |            }
0)   1.333 us    |            nfs_sb_active();
0) + 12.963 us   |          }
0)               |          nfs_file_set_open_context() {
0)   1.167 us    |            get_nfs_open_context();
0)               |            nfs_inode_attach_open_context() {
0)               |              _raw_spin_lock() {
0)   1.111 us    |                preempt_count_add();
0)   3.260 us    |              }
0)               |              _raw_spin_unlock() {
0)   1.019 us    |                preempt_count_sub();
0)   3.000 us    |              }
0)   9.630 us    |            }
0) + 14.778 us   |          }
0)   1.185 us    |          __put_nfs_open_context();
0) + 33.389 us   |        }
0) + 35.815 us   |      }
0)               |      file_ra_state_init() {
0)   1.186 us    |        inode_to_bdi();
0)   3.556 us    |      }
0) + 55.685 us   |    }
0) + 66.333 us   |  }
...
```

设置event

启用event

在event所属目录下对event设置，例如，启用mmap_lock event下的所有event：

```
# echo 1 > /sys/kernel/tracing/instances/mmap_lock/events/mmap_lock/enable
```

如果不需要设置mmap_lock下的所有event，只需要设置mmap_lock_acquire_returned，则可以：

```
# echo 1 >
/sys/kernel/tracing/instances/mmap_lock/events/mmap_lock/mmap_lock_acquire_r
eturned/enable
```

设置filter

trace event可能频繁触发，为了在一定的buffer size下得到更长时间的trace记录，有时需要对trace event做filter设置，即过滤trace event，在抓lock event、kmem event时经常使用，详细用法参考event *5.2 Setting filters*

bufffer size

例如sched event、irq & preempt event、kmem event等触发非常频繁，默认buffer szie抓到的时间段比较短，经常需要增大buffer size，方法参考：

```
# echo 65536 > /sys/kernel/tracing/instances/mmap_lock/buffer_size_kb
```

提醒：`buffer_size_kb`表示percpu buffer szie，如果有8个cpu，则total buffer szie应为`buffer_size_kb * 8`。

options

options下可以对trace的行为或内容做细微设置，例如，有些情况想要抓出trace event的调用栈，则可以通过stacktrace选项设置，例如：

```
# echo 1 > /sys/kernel/tracing/instances/mmap_lock/options/stacktrace
# echo 1 > /sys/kernel/tracing/instances/mmap_lock/options/sym-offset
```

确认设置

在实际测试中需要在init.rc中设置，**必须** 要 **double check** 是否生效。 **init.rc中做了哪些设置就需要确认哪些设置，只要依次将设置项读出确认即可。**

自定义event

sample code

遇到printk不适合场景，定义trace event来分析kernel问题是一个不错的想法。定义trace event只需要参考kernel sample code就足够了，code注释几乎涵盖了需要注意的方方面面。

trace_print_helper

`trace_print_helper`是对一个只打印字符串的自定义的trace event的封装，当遇到打印量大的场景，printk会导致系统性能问题，且kernel log buffer size可能不够记录，这种情况可以考虑使用`trace_print_helper`，可以在任何上下文使用，使用方法与printk用法相同。这是基于linux-6.1实现`trace_print_helper`的diff，linux-5.10、linux-5.15也适用：

```
commit 0098626333457ca8c4df9fba90ce39871f95bc1f
Author: andrew <mrju.email@gmail.com>
Date:   Mon Nov 20 22:12:13 2023 +0800

    implement trace_print_helper

    Signed-off-by: andrew <mrju.email@gmail.com>

diff --git a/include/linux/printk.h b/include/linux/printk.h
index 8c81806c2e99..192588e089b0 100644
--- a/include/linux/printk.h
+++ b/include/linux/printk.h
@@ -142,6 +142,8 @@ void early_printk(const char *s, ...) { }

 struct dev_printk_info;

+void trace_print_helper(const char *fmt, ...);
+
 #ifdef CONFIG_PRINTK
 asmlinkage __printf(4, 0)
 int vprintk_emit(int facility, int level,
diff --git a/include/trace/events/print.h b/include/trace/events/print.h
new file mode 100644
index 000000000000..b24692f82aea
--- /dev/null
+++ b/include/trace/events/print.h
@@ -0,0 +1,30 @@
+/* SPDX-License-Identifier: GPL-2.0 */
+#undef TRACE_SYSTEM
+#define TRACE_SYSTEM print
+
+#if !defined(_TRACE_PRINT_H) || defined(TRACE_HEADER_MULTI_READ)
+#define _TRACE_PRINT_H
+#include <linux/tracepoint.h>
+
+#undef TP_printk
+#define TP_printk(fmt, args...) fmt, args
+
+TRACE_EVENT(print,
+    TP_PROTO(const char* str),
+    TP_ARGS(str),
+
+    TP_STRUCT__entry(
```

```
+        __string(str, str)
+    ),
+
+    TP_fast_assign(
+        __assign_str(str, str);
+    ),
+
+    TP_printk("%s", __get_str(str))
+);
+
+#endif
+
+/* This part must be outside protection */
+#include <trace/define_trace.h>
diff --git a/kernel/printk/Makefile b/kernel/printk/Makefile
index f5b388e810b9..51e8cac32c5e 100644
--- a/kernel/printk/Makefile
+++ b/kernel/printk/Makefile
@@ -1,5 +1,6 @@
 # SPDX-License-Identifier: GPL-2.0-only
 obj-y  = printk.o
+obj-y  += trace_print.o
 obj-$(CONFIG_PRINTK)   += printk_safe.o
 obj-$(CONFIG_A11Y_BRAILLE_CONSOLE)     += braille.o
 obj-$(CONFIG_PRINTK_INDEX)     += index.o
diff --git a/kernel/printk/trace_print.c b/kernel/printk/trace_print.c
new file mode 100644
index 000000000000..64f551e2169e
--- /dev/null
+++ b/kernel/printk/trace_print.c
@@ -0,0 +1,17 @@
+#define CREATE_TRACE_POINTS
+#include <trace/events/print.h>
+
+#define TP_BUF_SIZE (1024)
+
+void trace_print_helper(const char *fmt, ...)
+{
+    char buf[TP_BUF_SIZE];
+    va_list ap;
+
+    va_start(ap, fmt);
+    vsnprintf(buf, sizeof(buf), fmt, ap);
```

```
+    va_end(ap);
+
+    trace_print(buf);
+}
+EXPORT_SYMBOL_GPL(trace_print_helper);
```

提醒：`trace_print_helper`在stack中定义了1024 bytes的buffer，如果使用`trace_print_helper`导致的栈溢出，那么需要留意code本身是否存在栈溢出风险。

Tips

trace的含义

trace记录的含义需要通过查看trace event的定义和插桩点来了解。trace event必须在`.h`文件中定义，在`.c`文件中使用。根据event name在code总搜索找到插桩点，规则，event name为xxx，则搜索trace_xxx，例如，对于name为`sched_switch`的event，通过搜索`trace_sched_switch`，搜到在`kernel/sched/core.c`中被使用，即插桩点在这个文件中，则必定会包含sched_switch定义的`.h`文件，即`include/trace/events/sched.h`。

hash pointer

issue

内核安全需求要保护指针避免泄漏导致在trace记录中看到指针打印不全，下面是抓mmap_lock event时的例子，可以看到`mm=000000005a63e32e`应该时task->mm，记录的地址并不是内核地址空间，有些版本可能会看到`mm=(__ptrval__)`：

```
# tracer: nop
#
# entries-in-buffer/entries-written: 891/891   #P:4
#
#                                _-----=> irqs-off/BH-disabled
#                               / _----=> need-resched
#                              | / _---=> hardirq/softirq
#                              || / _--=> preempt-depth
#                              ||| / _-=> migrate-disable
#                              |||| /     delay
#           TASK-PID     CPU#  |||||  TIMESTAMP  FUNCTION
#              | |         |   |||||     |         |
              sh-119     [001] ...1.  1834.010166: mmap_lock_start_locking:
mm=000000005a63e32e memcg_path=/ write=true
              sh-119     [001] ...1.  1834.010175:
mmap_lock_acquire_returned: mm=000000005a63e32e memcg_path=/ write=true
success=true
              sh-119     [001] ...1.  1834.010177: mmap_lock_start_locking:
```

```
mm=00000000974f3337 memcg_path=/ write=true
              sh-119     [001] ...1.  1834.010180:
mmap_lock_acquire_returned: mm=00000000974f3337 memcg_path=/ write=true
success=true
```

sulution

linux-5.15和linux-6.1中可以设置hash-ptr关闭，举例，通过以下方法关闭hash-ptr：

```
# echo 0 > /sys/kernel/tracing/instances/mmap_lcok/options/hash-ptr
```

如果linux版本不支持hash-ptr设置，还可以找到trace event的定义，修改定义中的打印方式，例如，对于mmap_lock event，可以修改，即将指针打印的格式化`%p`改为`%px`即可：

```
diff --git a/include/trace/events/mmap_lock.h
b/include/trace/events/mmap_lock.h
index 14db8044c1ff..56c8aae0b905 100644
--- a/include/trace/events/mmap_lock.h
+++ b/include/trace/events/mmap_lock.h
@@ -32,7 +32,7 @@ DECLARE_EVENT_CLASS(mmap_lock,
        ),

        TP_printk(
-               "mm=%p memcg_path=%s write=%s",
+               "mm=%px memcg_path=%s write=%s",
                __entry->mm,
                __get_str(memcg_path),
                __entry->write ? "true" : "false"
@@ -71,7 +71,7 @@ TRACE_EVENT_FN(mmap_lock_acquire_returned,
        ),

        TP_printk(
-               "mm=%p memcg_path=%s write=%s success=%s",
+               "mm=%px memcg_path=%s write=%s success=%s",
                __entry->mm,
                __get_str(memcg_path),
                __entry->write ? "true" : "false",
```

注意：以上方法都不能带到release到市场版本。

用例

android system stability问题多会导致kernel panic，因此不能runtime时手动设置ftrace，否则kernel panic后的下次启动将不带ftrace设置，这就需要将设置ftrace放在android的init.rc中来做，以下用例将在`on init`时设置，已经过很多项目实作检验。

#01 function_graph tracer

应用场景

需要了解函数的执行情况可以抓function_graph trace，例如，当发现userland某一个系统调用执行失败，调查发现在内核中返回的error，则可以抓function_graph trace查看系统调用的执行流程。以下假设对`vfs_open`抓functon_graph trace。

开启方法

config：

```
CONFIG_FUNCTION_TRACER=y
CONFIG_FUNCTION_GRAPH_TRACER=y
CONFIG_DYNAMIC_FTRACE=y
```

init.rc：

```
on init
    write /sys/kernel/tracing/buffer_size_kb 65536
    write /sys/kernel/tracing/options/funcgraph-irqs 0
    write /sys/kernel/tracing/options/trace_printk 0
    write /sys/kernel/tracing/options/markers 0
    write /sys/kernel/tracing/options/funcgraph-proc 1
    write /sys/kernel/tracing/options/funcgraph-abstime 1
    write /sys/kernel/tracing/current_tracer function_graph
    write /sys/kernel/tracing/set_graph_function vfs_open
    write /sys/kernel/tracing/tracing_on 1
```

注意事项

- 自定义instance不支持设置function_graph tacer，因此在default buffer中抓function_graph trace，如果有其他程序同时使用default buffer则还会录到其他trace；
- 建议设置较大buffer size；
- options中关闭markers和trace_printk减少其他trace干扰；
- 通常关注irq的情况，因此关闭funcgraph-irqs；
- 通常设置funcgraph-abstime方便与kernel log时间对齐；
- 建议不要对过多funtion同时抓function_graph trace；

#02 sched event

应用场景

尤其在hang detect问题中常常遇到cpu loading过高或rt task不当使用cpu导致的逻辑死锁问题，抓sched trace可以直接证明问题出在哪些task上。

开启方法

config：

```
CONFIG_SCHED_TRACER=y
```

init.rc：

```
on init
    mkdir /sys/kernel/tracing/instances/sched
    write /sys/kernel/tracing/instances/sched/buffer_size_kb 65536
    write
/sys/kernel/tracing/instances/sched/events/sched/sched_switch/enable 1
    write
/sys/kernel/tracing/instances/sched/events/sched/sched_wakeup/enable 1
    write
/sys/kernel/tracing/instances/sched/events/sched/sched_waking/enable 1
    write
/sys/kernel/tracing/instances/sched/events/sched/sched_wakeup_new/enable 1
    write /sys/kernel/tracing/instances/sched/tracing_on 1
```

注意事项

- 需要设置较大buffer szie。

#03 irq & preempt event

应用场景

发生如关中断或关抢占过久而导致的调度问题、关中断过久或中断频繁导致watchdog timeout问题、关抢占下发生调度等等问题，无法确定哪些中断或什么task导致时可以考虑抓irq & preempt event。在与sched event配合时常常用于分析hang detect问题。

开启方法

config：

```
CONFIG_IRQSOFF_TRACER=y
CONFIG_PREEMPT_TRACER=y
CONFIG_TRACE_IRQFLAGS=y
```

init.rc：

```
on init
    mkdir /sys/kernel/tracing/instances/preemptirq
    write /sys/kernel/tracing/instances/preemptirq/buffer_size_kb 65536
    write /sys/kernel/tracing/instances/preemptirq/events/irq/enable 1
    write /sys/kernel/tracing/instances/preemptirq/events/preemptirq/enable
```

```
1
    write /sys/kernel/tracing/instances/preemptirq/tracing_on 1
```

注意事项

- 需要设置较大buffer size；
- 有时需要开启stacktrace，当开启stacktrace后，提取trace需要花费更多时间。

#04 lock event

应用场景

遇到卡锁问题无法查出持锁者可以使用，所有锁类型，例如，spinlock、mutex、rwsem等等都支持。

开启方法

lock_name需要根据lock初始化接口的参数确定，例如：

- 若初始化语句为`init_rwsem(&rwsem);`，则lock_name就是`&rwsem`，filter条件就是`"name == '&rwsem'"`；
- 若初始化语句为`init_rwsem(&container->rwsem);`，则lock_name就是`&container->rwsem`，filter条件就是`"name == '&container->rwsem'"`；
- 若初始化语句为`init_rwsem(rwsem);`，则lock_name就是`rwsem`，filter条件就是`"name == 'rwsem'"`；

这里假设所名称为lock_name，实际问题中需要具体确定。
config：

```
CONFIG_LOCKDEP=y
CONFIG_LOCK_STAT=y
```

diff：

```
diff --git a/lib/debug_locks.c b/lib/debug_locks.c
index a75ee30b77cb..794bb0c1f538 100644
--- a/lib/debug_locks.c
+++ b/lib/debug_locks.c
@@ -22,7 +22,7 @@
  * that would just muddy the log. So we report the first one and
  * shut up after that.
  */
-int debug_locks __read_mostly = 1;
+int debug_locks __read_mostly = 0;
 EXPORT_SYMBOL_GPL(debug_locks);
```

```
 /*
```

init.rc：

```
on init
    mkdir /sys/kernel/tracing/instances/lock
    write /sys/kernel/tracing/instances/lock/buffer_size_kb 65536
    write /sys/kernel/tracing/instances/lock/options/stacktrace 1
    write /sys/kernel/tracing/instances/lock/options/sym-offset 1
    write /sys/kernel/tracing/instances/lock/options/hash-ptr 1
    write /sys/kernel/tracing/instances/lock/events/lock/lock_acquire/filter
"name == 'lock_name'"
    write
/sys/kernel/tracing/instances/lock/events/lock/lock_contended/filter "name
== 'lock_name'"
    write
/sys/kernel/tracing/instances/lock/events/lock/lock_acquired/filter "name ==
'lock_name'"
    write /sys/kernel/tracing/instances/lock/events/lock/lock_release/filter
"name == 'lock_name'"
    write /sys/kernel/tracing/instances/lock/events/lock/lock_acquire/enable
1
    write
/sys/kernel/tracing/instances/lock/events/lock/lock_contended/enable 1
    write
/sys/kernel/tracing/instances/lock/events/lock/lock_acquired/enable 1
    write /sys/kernel/tracing/instances/lock/events/lock/lock_release/enable
1
    write /sys/kernel/tracing/instances/lock/tracing_on 1
```

注意事项

- 需要设置较大buffer size；
- 视情况而定是否开启stacktrace；
- 需要关闭hash pointer，参考 **hash pointer**。

#05 syscall event

应用场景

需要监控系统调用的情时使用。

开启方法

确定需要跟踪哪些系统调用，支持跟踪的系统调用可以查看`/sys/kernel/tracing/events/syscalls`下的内容，这里假设跟踪mmap系统调用的enter和exit。
config：

```
CONFIG_FTRACE_SYSCALLS=y
```

init.rc：

```
on init
    mkdir /sys/kernel/tracing/instances/mmap
    write /sys/kernel/tracing/instances/mmap/buffer_size_kb 10240
    write /sys/kernel/tracing/instances/mmap/options/hash-ptr 1
    write
/sys/kernel/tracing/instances/mmap/events/syscalls/sys_enter_mmap/enable 1
    write
/sys/kernel/tracing/instances/mmap/events/syscalls/sys_exit_mmap/enable 1
    write /sys/kernel/tracing/instances/mmap/tracing_on 1
```

注意事项

- 如果所跟踪的系统调用trace包含kernel addr，则需要关闭hash pointer，参考 **hash pointer。**

#06 signal event

应用场景

需要跟踪signal的发出者和接收者，则可以开启signal event。

开启方法

init.rc：

```
on init
    mkdir /sys/kernel/tracing/instances/signal
    write /sys/kernel/tracing/instances/signal/buffer_size_kb 8192
    write /sys/kernel/tracing/instances/signal/events/signal/enable
    write /sys/kernel/tracing/instances/signal/tracing_on 1
```

注意事项

- 通常不需要抓stacktrace，如果有必要也可以考虑。

#07 kmem event

应用场景

分析内存分配问题可以考虑开启kmem event，用来分析一段时间内存分配的速度，内存释放的速度，是否存在程序实例分配的内存过多，是否可能存在潜在内存泄漏等等，通常是这类问题的可选分析手段之一。

本方法存在一个缺点，就是trace量过大会导致一定buffer size可记录时间段偏短。因此使用本方法时常常需要提前做好初步分析，已经定位好order或size后进行filter抓取比较有效。
kmem下包含event有：

```
kfree
kmalloc
kmem_cache_alloc
kmem_cache_free
mm_page_alloc
mm_page_alloc_extfrag
mm_page_alloc_zone_locked
mm_page_free
mm_page_free_batched
mm_page_pcpu_drain
rss_stat
```

具体需要跟踪哪些event需要根据问题而定。

开启方法

假设跟踪mm_page_alloc和mm_page_free，并且order为2的情况。
init.rc：

```
on init
    mkdir /sys/kernel/tracing/instances/kmem
    write /sys/kernel/tracing/instances/kmem/buffer_size_kb 65536
    write /sys/kernel/tracing/instances/kmem/options/stacktrace 1
    write /sys/kernel/tracing/instances/kmem/options/sym-offset 1
    write /sys/kernel/tracing/instances/kmem/options/hash-ptr 1
    write
/sys/kernel/tracing/instances/kmem/events/kmem/mm_page_alloc/filter
"order==2"
    write /sys/kernel/tracing/instances/kmem/events/kmem/mm_page_free/filter
"order==2"
    write
/sys/kernel/tracing/instances/kmem/events/kmem/mm_page_alloc/enable 1
    write /sys/kernel/tracing/instances/kmem/events/kmem/mm_page_free/enable
1
    write /sys/kernel/tracing/instances/kmem/tracing_on 1
```

注意事项

- 需要关闭hash pointer，参考 **hash pointer**；
- 需要设置较大的buffer size；
- 以上设置会抓stacktrace，实际问题中要视情况而定，可以不抓stacktrace以便节约buffer空间。

结语

希望本篇文档可以为阅者在分析android system stabilty问题时有所帮助。有任何建议可以反馈给作者。