Linux平台性能图形化分析快速入门

文件标识：RK-SM-YF-488

发布版本：V1.0.1

日期：2023-03-14

文件密级：□绝密 □秘密 □内部资料 ■公开

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

产品版本

芯片名称	内核版本
全系列	通用

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	陈谋春	2023-03-14	初始版本
V1.0.1	闫孝军	2024-09-11	修正错别字

目录

1. 启用Kernel Tracer 和 Debug FS

在分析开始前，我们需要确保内核已经启用了 Tracer 和 Debug FS，可以通过如下命令查看：

```
rk3399_Android11:/data/local/tmp # mount | grep debugfs
/sys/kernel/debug on /sys/kernel/debug type debugfs
(rw,seclabel,relatime,mode=755)
1|rk3399_Android11:/data/local/tmp # mount | grep trace
tracefs on /sys/kernel/tracing type tracefs (rw,seclabel,relatime,mode=755)
tracefs on /sys/kernel/debug/tracing type tracefs (rw,seclabel,relatime,mode=755)
```

如果没有看到以上节点，则需要 `make menuconfig` 启用如下选项，并烧写替换内核：

```
CONFIG_FTRACE
CONFIG_ENABLE_DEFAULT_TRACERS
CONFIG_DEBUG_FS
```

2. 下载 Catapult

为了能实现图形化分析，需要下载 Chrome 的性能分析工具 Catapult，具体如下：

```
git clone https://github.com/catapult-project/catapult.git
```

3. 代码插桩

Tracer 的原理在于代码插桩，内核的关键节点基本都有现成的桩，这些桩在内核里被叫做 event，可以通过如下命令看到：

```
rk3399_Android11:/data/local/tmp # ls /sys/kernel/debug/tracing/events/
alarmtimer  cgroup               drm         fib6          header_page  jbd2
module          pagemap  raw_syscalls  rtc     sock        thermal_ipa_power
   vmscan
android_fs  clk                  dwc3        filelock      i2c          kmem
namei           percpu   rcu           sched   spi
thermal_power_allocator  workqueue
asoc        cma                  emulation   filemap       initcall     libata
napi            power    regmap        scmi    swiotlb     thermal_virtual
   writeback
binder      compaction           enable      ftrace        iommu        mac80211
net             printk   regulator     scsi    sync_trace  timer
   xdp
block       cpufreq_interactive  ext4        gadget        ion          mdio
nvme            qdisc    rogue         signal  task        udp
   xhci-hcd
bridge      cpuhp                f2fs        gpio          ipi          migrate
oom             random   rpm           skb     tcp         v4l2
cfg80211    dma_fence            fib         header_event  irq          mmc
page_isolation  ras      rseq          smbus   thermal     vb2
```

你需要根据你的实际需要去打开这些 event，例如，如果你要分析存储性能瓶颈，通常你需要打开：文件系统，block 层和设备层，方法如下：

```
echo 1 > /sys/kernel/debug/tracing/events/ext4/enable       # 打开文件系统开关，假设你的文件系统是ext4
echo 1 > /sys/kernel/debug/tracing/events/block/enable      # 打开通用block层event开关
echo 1 > /sys/kernel/debug/tracing/events/mmc/enable        # 打开设备层event开关，假设你的存储介质是EMMC
```

3.1 内核态插桩

如果现有的内核 event 无法满足你的需求，则需要定义自己的 event，例如，我现在需要调试 dmc 变频，需要增加两个 dmc 变频的 event：ddr_frequency 和 ddr_load，可以通过如下步骤实现：

1. 声明新的 event，所有的内核 event 都需要在`include/trace/events/`目录下去声明，这个头文件目录下的每个头文件都对应于 tracefs 的一个 event 目录，例如：

   ```
   ls include/trace/events/power.h -l
   -rw-rw-r-- 1 cmc cmc 12129 2月  23  2022 include/trace/events/power.h      # 这是源码目录看到的头文件
   ls /sys/kernel/debug/tracing/events/ -l | grep " power"
   drwxr-xr-x  25 root root 0 1970-01-01 00:00 power                          # 这是设备的tracefs下看到event目录
   ```

   为了简化这个例子，我们直接把新的 event 加到`power.h`里，具体如下：

   ```
   diff --git a/include/trace/events/power.h b/include/trace/events/power.h
   index f7aece721aed..29c962a862ce 100644
   --- a/include/trace/events/power.h
   +++ b/include/trace/events/power.h
   ```

```
@@ -529,6 +529,37 @@ DEFINE_EVENT(dev_pm_qos_request,
dev_pm_qos_remove_request,

    TP_ARGS(name, type, new_value)
 );
+
+DECLARE_EVENT_CLASS(dram,
+
+   TP_PROTO(unsigned int state),
+
+   TP_ARGS(state),
+
+   TP_STRUCT__entry(
+       __field(    u32,        state       )
+   ),
+
+   TP_fast_assign(
+       __entry->state = state;
+   ),
+
+   TP_printk("state=%lu", (unsigned long)__entry->state)
+);
+
+DEFINE_EVENT(dram, ddr_frequency,                  /* 这里定义了新的event:
ddr_frequency，第二个参数就是event名，后面会用到 */
+
+   TP_PROTO(unsigned int state),
+
+   TP_ARGS(state)
+);
+
+DEFINE_EVENT(dram, ddr_load,                       /* 这里定义了新的event:
ddr_load */
+
+   TP_PROTO(unsigned int state),
+
+   TP_ARGS(state)
+);
 #endif /* _TRACE_POWER_H */

 /* This part must be outside protection */
```

2. 在合适的位置去引用这个 event，因为我们是要监控 dmc 的变频，所以我们需要在变频函数里去引用，具体如下：

```
diff --git a/drivers/devfreq/rockchip_dmc.c b/drivers/devfreq/rockchip_dmc.c
index d75c21e51854..7c3971ccf397 100644
--- a/drivers/devfreq/rockchip_dmc.c
+++ b/drivers/devfreq/rockchip_dmc.c
@@ -52,6 +52,7 @@
 #include <soc/rockchip/rockchip_opp_select.h>
 #include <soc/rockchip/scpi.h>
 #include <uapi/drm/drm_mode.h>
+#include <trace/events/power.h>

 #include "governor.h"
 #include "rockchip_dmc_timing.h"
```

```
@@ -2636,7 +2637,7 @@ static int devfreq_dmc_ondemand_func(struct devfreq
*df,
    /* Assume MAX if it is going to be divided by zero */
    if (stat->total_time == 0) {
        *freq = DEVFREQ_MAX_FREQ;
-       return 0;
+       goto set_new_freq;
    }

    /* Prevent overflow */
@@ -2649,20 +2650,20 @@ static int devfreq_dmc_ondemand_func(struct devfreq
*df,
    if (stat->busy_time * 100 >
        stat->total_time * upthreshold) {
        *freq = DEVFREQ_MAX_FREQ;
-       return 0;
+       goto set_new_freq;
    }

    /* Set MAX if we do not know the initial frequency */
    if (stat->current_frequency == 0) {
        *freq = DEVFREQ_MAX_FREQ;
-       return 0;
+       goto set_new_freq;
    }

    /* Keep the current frequency */
    if (stat->busy_time * 100 >
        stat->total_time * (upthreshold - downdifferential)) {
        *freq = max(target_freq, stat->current_frequency);
-       return 0;
+       goto set_new_freq;
    }

    /* Set the desired frequency based on the load */
@@ -2673,6 +2674,10 @@ static int devfreq_dmc_ondemand_func(struct devfreq
*df,
    b = div_u64(b, (upthreshold - downdifferential / 2));
    *freq = max_t(unsigned long, target_freq, b);

+set_new_freq:
+   trace_ddr_frequency(*freq);                /* 引用我们加的两个新event，格式：
trace_xxx，其中xxx就是前面定义的event名 */
+   trace_ddr_load(div_u64(stat->busy_time, stat->total_time) * 100);
+
    return 0;

 reset_last_status:
```

3. 打开新 event 的开关，方法和原生的 event 一样，具体如下：

```
echo 1 > /sys/kernel/debug/tracing/events/power/ddr_frequency/enable      #
打开ddr_frequency开关
echo 1 > /sys/kernel/debug/tracing/events/power/ddr_load/enable     # 打开
ddr_load开关
```

3.2 用户态插桩

用户态插桩需要用到一个 atrace 的库（从 Android 上移植来的），原理其实也是走的 trace_marker 接口，这里不细述，直接给出使用方法，目前 atrace 支持两套接口：c++ 和 c，分别对应两个不同的头文件：`atrace.h 和 trace.h`，同时还有一个库的源码文件`trace.c`，具体如下（==这些文件可以在本文末尾的附录中拷贝粘贴==）：

```
tar xvf atrace.tar.bz2
ls atrace
-rwx------ 1 cmc cmc  971 5月  14  2021 atrace.h              # 这是c++需要引用的头文件
-rwx------ 1 cmc cmc 2990 5月  14  2021 trace.c              # 这是库的源码，需要加到你的工程里编译
-rwx------ 1 cmc cmc  540 5月  14  2021 trace.h              # 这是c需要引用的头文件
```

c++ 的例子会简单一些，具体如下：

```
#include "atrace.h"

void ShaderCache::initShaderDiskCache() {
    ATRACE_NAME("iShaderDC");          // 最后在图形化分析的时候会看到iShaderDC的名字，就是对应这个函数的耗时
    std::lock_guard<std::mutex> lock(mMutex);

    // Emulators can switch between different renders either as part of config
    // or snapshot migration. Also, program binaries may not work well on some
    // desktop / laptop GPUs. Thus, disable the shader disk cache for emulator builds.
    if (!Properties::runningInEmulator && mFilename.length() > 0) {
        mBlobCache.reset(new FileBlobCache(maxKeySize, maxValueSize, maxTotalSize, mFilename));
        mInitialized = true;
    }
}

camera_status_t ACameraDevice_close(ACameraDevice* device) {
    ATRACE_CALL();                  // 自动以当前函数名命名，所以最后在图形化分析里会看到这个函数名
    if (device == nullptr) {
        ALOGE("%s: invalid argument! device is null", __FUNCTION__);
        return ACAMERA_ERROR_INVALID_PARAMETER;
    }
    delete device;
    return ACAMERA_OK;
}
```

c 函数调用则需要手动确定起始和结束，具体如下：

```
#include "trace.h"

void fun2() {
  /* do something */
}

void fun1() {
  atrace_begin_body("fun1_call_fun2");
  /* call fun2 */
  fun2();
  atrace_end_body();    /* 这样写，统计到的是fun1调用fun2的总时间 */
}
```

显然，如果 fun2 调用的地方非常多，想统计 fun2 的开销，放 fun2 内部是最合理的，此时必须要==注意确保在 fun2 的所有返回点都调用 `atrace_end_body`==，例如：

```
#include "trace.h"

int fun2() {
  atrace_begin_body("fun2");

  if (cond1) {
    atrace_end_body();
    return err;
  }

  /* do something */
  atrace_end_body();
  return ok;
}
```

==Note: trace.c 要记得加到编译脚本里==

4. 抓取原始 Trace

抓取原始 Trace 的方法其实和 Linux 传统的方法一样，不过为了方便，我们通常会写一个脚本，具体如下：

```
echo 4096 > /sys/kernel/debug/tracing/buffer_size_kb            # 设置buffer大小，太小的话，会导致event丢失，根据需要去调整
echo global > /sys/kernel/debug/tracing/trace_clock             # 启用全局时钟，确保每个核的时钟同步
echo 0 > /sys/kernel/debug/tracing/options/overwrite            # 不允许trace覆盖
echo 1 > /sys/kernel/debug/tracing/options/print-tgid           # 必须要打印线程id，这样图形化才不会混乱
echo 1 > /sys/kernel/debug/tracing/events/sched/sched_switch/enable # 下面是常用event的开关，通常都要开
echo 1 > /sys/kernel/debug/tracing/events/sched/sched_wakeup/enable
echo 1 > /sys/kernel/debug/tracing/events/sched/sched_waking/enable
echo 1 > /sys/kernel/debug/tracing/events/sched/sched_blocked_reason/enable
echo 1 > /sys/kernel/debug/tracing/events/irq/enable
```

```
echo 1 > /sys/kernel/debug/tracing/events/power/ddr_frequency/enable        # 根据自己的需要，去开启其他event
echo 1 > /sys/kernel/debug/tracing/events/power/ddr_load/enable
echo 1 > /sys/kernel/debug/tracing/tracing_on                               # 开始抓取
sleep $2                                                                    # 抓取的时间长度，通过参数传入
echo 0 > /sys/kernel/debug/tracing/tracing_on                               # 停止抓取
cat /sys/kernel/debug/tracing/trace > $1                                    # 保存trace到指定文件
```

把上面的脚本保存为 `atrace.sh`，即可通过如下命令来抓取原始 Trace：

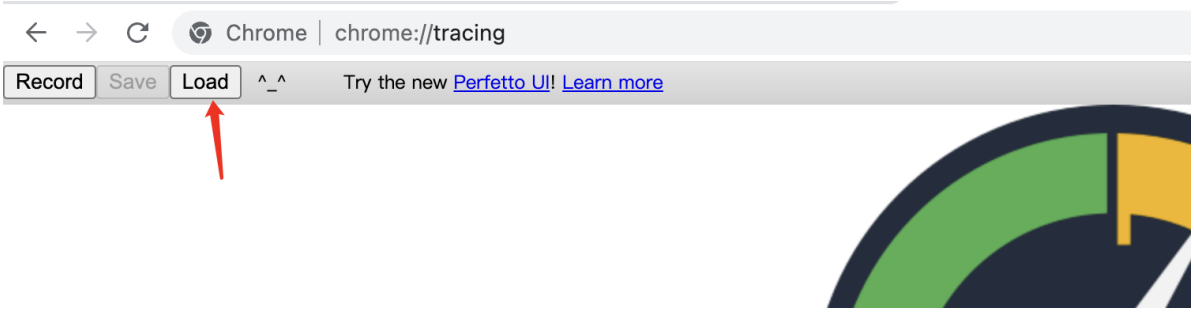
```
atrace.sh /path/to/my_trace 2               # 即抓取2秒的trace，并保存到/path/to/my_trace
```

5. 生成 HTML 和图形化分析

抓取到原始 trace 以后，就可以通过 Catapult 的 `trace2html` 命令来转成 HTML，具体如下：

```
cd catapult
./tracing/bin/trace2html --config chrome --output /path/to/my_trace.html /path/to/my_trace
ls -l /path/to/my_trace.html
-rw-r--r-- 1 cmc cmc   4712388 1月   7  2022 /path/to/my_trace.html
```

打开 Chrome 浏览器，同时在地址栏输入：`chrome://tracing/`，此时应该能看到如下界面：



点击 `load` 按钮，选择你之前转成功的 HTML，即可看到标准的 Android Systrace 界面，界面如下：

至于 Android Systrace 的 UI 使用方法，网上有很多介绍的，这里就不细述了。

6. 附录

6.1 atrace.h

```
#ifndef __ATRACE_H__
#define __ATRACE_H__

#define ATRACE_TAG   1

#include "trace.h"
#include <stdint.h>

// See <cutils/trace.h> for more ATRACE_* macros.

// ATRACE_NAME traces from its location until the end of its enclosing scope.
#define _PASTE(x, y) x ## y
#define PASTE(x, y) _PASTE(x,y)
#define ATRACE_NAME(name) ScopedTrace PASTE(___tracer, __LINE__)(ATRACE_TAG,
name)

// ATRACE_CALL is an ATRACE_NAME that uses the current function name.
#define ATRACE_CALL() ATRACE_NAME(__FUNCTION__)

static inline void atrace_begin(__attribute__((unused)) uint64_t tag, const char*
name)
{
    atrace_begin_body(name);
}

static inline void atrace_end(__attribute__((unused)) uint64_t tag)
{
    atrace_end_body();
}

class ScopedTrace {
public:
    inline ScopedTrace(uint64_t tag, const char* name) : mTag(tag) {
        atrace_begin(mTag, name);
    }

    inline ~ScopedTrace() {
        atrace_end(mTag);
    }

private:
    uint64_t mTag;
};

#endif // __ATRACE_H__
```

6.2 trace.h

```
#ifndef __TRACE_H__
#define __TRACE_H__

#include <stdint.h>

#if defined(__cplusplus)
#define __BEGIN_DECLS extern "C" {
#define __END_DECLS }
#else
#define __BEGIN_DECLS
#define __END_DECLS
#endif

__BEGIN_DECLS

void atrace_begin_body(const char* name);
void atrace_end_body();
void atrace_async_begin_body(const char* name, int32_t cookie);
void atrace_async_end_body(const char* name, int32_t cookie);
void atrace_int_body(const char* name, int32_t value);
void atrace_int64_body(const char* name, int64_t value);

__END_DECLS

#endif
```

6.3 trace.c

```
#include "trace.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <inttypes.h>
#include <string.h>

//#define TRACE_ON

#ifdef TRACE_ON

#ifdef __cplusplus
#define CC_LIKELY( exp )    (__builtin_expect( !!(exp), true ))
#define CC_UNLIKELY( exp )  (__builtin_expect( !!(exp), false ))
#else
#define CC_LIKELY( exp )    (__builtin_expect( !!(exp), 1 ))
#define CC_UNLIKELY( exp )  (__builtin_expect( !!(exp), 0 ))
#endif
```

```
#define ATRACE_MESSAGE_LENGTH 1024

static int atrace_marker_fd     = -1;
static int init_ok = 0;

static int trace_init_once(void)
{
    if (init_ok == 0)
    {
         atrace_marker_fd = open("/sys/kernel/debug/tracing/trace_marker",
O_WRONLY | O_CLOEXEC);
        if (atrace_marker_fd == -1) {
            printf("Error opening trace file: %s (%d)\n", strerror(errno),
errno);
            return -1;
        }
    }

    init_ok = 1;
    return 0;
}

#define WRITE_MSG(format_begin, format_end, name, value) { \
    char buf[ATRACE_MESSAGE_LENGTH]; \
    if (CC_UNLIKELY(!init_ok)) \
        trace_init_once(); \
    int pid = getpid(); \
    int len = snprintf(buf, sizeof(buf), format_begin "%s" format_end, pid, \
        name, value); \
    if (len >= (int) sizeof(buf)) { \
        /* Given the sizeof(buf), and all of the current format buffers, \
         * it is impossible for name_len to be < 0 if len >= sizeof(buf). */ \
        int name_len = strlen(name) - (len - sizeof(buf)) - 1; \
        /* Truncate the name to make the message fit. */ \
        printf("Truncated name in %s: %s\n", __FUNCTION__, name); \
        len = snprintf(buf, sizeof(buf), format_begin "%.*s" format_end, pid, \
            name_len, name, value); \
    } \
    write(atrace_marker_fd, buf, len); \
}

void atrace_begin_body(const char* name)
{
    WRITE_MSG("B|%d|", "%s", name, "");
}

void atrace_end_body()
{
    WRITE_MSG("E|%d", "%s", "", "");
}

void atrace_async_begin_body(const char* name, int32_t cookie)
{
    WRITE_MSG("S|%d|", "|%" PRId32, name, cookie);
}

void atrace_async_end_body(const char* name, int32_t cookie)
{
```

```
    WRITE_MSG("F|%d|", "|%" PRId32, name, cookie);
}

void atrace_int_body(const char* name, int32_t value)
{
    WRITE_MSG("C|%d|", "|%" PRId32, name, value);
}

void atrace_int64_body(const char* name, int64_t value)
{
    WRITE_MSG("C|%d|", "|%" PRId64, name, value);
}

#else

void atrace_begin_body(__attribute__((unused)) const char* name)
{
}

void atrace_end_body()
{
}

void atrace_async_begin_body(__attribute__((unused)) const char* name,
__attribute__((unused)) int32_t cookie)
{
}

void atrace_async_end_body(__attribute__((unused)) const char* name,
__attribute__((unused)) int32_t cookie)
{
}

void atrace_int_body(__attribute__((unused)) const char* name,
__attribute__((unused)) int32_t value)
{
}

void atrace_int64_body(__attribute__((unused)) const char* name,
__attribute__((unused)) int64_t value)
{
}

#endif
```