

Perf 使用说明

文件标识: RK-SM-YF-149

发布版本: V1.1.0

日期: 2023-03-08

文件密级: 绝密 秘密 内部资料 公开

免责声明

本文档按“现状”提供，瑞芯微电子股份有限公司（“本公司”，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

版权所有 © 2023 瑞芯微电子股份有限公司

超越合理使用范畴，非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

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	2017-12-25	初始版本
V1.0.1	黄莹	2021-03-02	修改格式
V1.1.0	陈谋春	2023-03-08	增加EVENT格式说明

目录

Perf 使用说明

1. 介绍
2. 功能
3. 在 Android 平台使用
 - 3.1 准备工作
 - 3.2 获取当前平台支持的事件
 - 3.3 获取系统热点进程
 - 3.4 获取进程的统计信息
 - 3.5 收集进程的 profile 数据
 - 3.6 分析 profile 数据
 - 3.7 FlameGraph
4. 在 Linux 平台使用
5. Simpleperf 使用
6. perf event格式说明

1. 介绍

Perf 是从 Linux 2.6 开始引入的一个 profiling 工具，通过访问包括 pmu 在内的软硬件性能计数器来分析性能，支持多架构，是目前 Kernel 的主要性能检测手段，和 Kernel 代码一起发布，所以兼容性良好。

2. 功能

性能瓶颈如果要分类的话，大致可以分为几个大类：cpu / gpu / mem / storage，其中 gpu 用 Perf 没法探测（这个目前比较好用的工具就只有 DS5），storage 只能用 tracepoint 来统计。总的说来，Perf 还是侧重于分析 cpu 的性能，其他功能都不是很好用。

```
$ perf

usage: perf [--version] [--help] COMMAND [ARGS]

The most commonly used perf commands are:
annotate           Read perf.data (created by perf record) and display annotated
code
archive            Create archive with object files with build-ids found in
perf.data file
bench               General framework for benchmark suites
buildid-cache      Manage <tt>build-id</tt> cache.
buildid-list       List the buildids in a perf.data file
diff              Read two perf.data files and display the differential profile
inject             Filter to augment the events stream with additional
information
kmem               Tool to trace/measure kernel memory(slab) properties
kvm                Tool to trace/measure kvm guest os
list               List all symbolic event types
lock               Analyze lock events
probe              Define new dynamic tracepoints
record             Run a command and record its profile into perf.data
report             Read perf.data (created by perf record) and display the
profile
sched              Tool to trace/measure scheduler properties (latencies)
script             Read perf.data (created by perf record) and display trace
output
stat               Run a command and gather performance counter statistics
test               Runs sanity tests.
timechart          Tool to visualize total system behavior during a workload
top               System profiling tool.

See 'perf help COMMAND' for more information on a specific command.
```

其中比较常用的功能有几个：

- **record**: 收集 profile 数据
- **report**: 根据 profile 数据生成统计报告

- stat: 打印性能计数统计值
- top: cpu 占有率实时统计

3. 在 Android 平台使用

3.1 准备工作

1. 首先按 Google 或芯片厂商的指导，构建一个完整的 Android 和 Kernel 的编译环境（如果不关心 Kernel 可以忽略），这样分析的时候符号表才能匹配上。
2. 编译 Perf

```
~$ . build/envsetup.sh
~$ lunch
~$ mmm external/linux-tools-perf
~$ adb root
~$ adb remount
~$ adb push perf /system/bin/
~$ adb shell sync
```

3. 准备符号文件

符号文件可以简单分为三类：

- a. 平台 native 代码，这部分代码在编译的过程中会自动生成符号表，不需要我们干预
- b. 平台 java 代码，对于 art 虚拟机来说（老版本的 dalvik 就不说了）最终的编译结果是 oat 文件，这也是正规的 elf 文件，但是默认是不带 debug 信息。而新版本的 Android 也提供了自动生成 java 符号表的工具：

```
bash art/tools/symbolize.sh
```

- c. 第三方 apk，如果是来自开源社区，则可以通过修改 makefile 和套用 Android 提供的 java 符号表工具来生成符号表文件，然后拷贝到 Android 的符号表目录，==注意路径必须要和设备上的完全一致==，可以通过 showmap 来获取设备上的路径。

```
~$ adb shell showmap apk_pid
38540      36296      36296      0      0      36216      80      0      3
/data/app/com.android.webview-2/lib/arm/libwebviewchromium.so
~$ cp libwebviewchromium.so
$ANDROID_PRODUCT_OUT/symbols/data/app/com.android.webview-
2/lib/arm/libwebviewchromium.so
```

如果是商业的 apk，基本上已经做过混淆和 strip，除非开发商能配合，不然就没招。

4. 稍微新一点的 Android 都开启了 Kernel 的指针保护，这也将影响 Perf 的 record，所以需要临时关闭保护：

```
~$ adb shell echo 0 > /proc/sys/kernel/kptr_restrict
```

5. 为了方便分析，一般会把 record 的数据 pull 到 host 端，在 host 端做分析，所以需要在设备端也安装一下 Perf 工具，ubuntu 下安装命令如下：

```
~$ sudo apt-get install linux-tools-common
```

6. 目前大部分的 Android 平台默认 Perf 功能都是打开的，所以一般不需要重新配置 Kernel，如果碰到 Perf 被关闭的情况，可以打开下面几个配置

```
CONFIG_PERF_EVENTS=y  
CONFIG_HW_PERF_EVENTS=y
```

3.2 获取当前平台支持的事件

```
rk3399:/data/local # ./perf list  
List of pre-defined events (to be used in -e):  
cpu-cycles OR cycles [Hardware event]  
instructions [Hardware event]  
cache-references [Hardware event]  
cache-misses [Hardware event]  
branch-instructions OR branches [Hardware event]  
branch-misses [Hardware event]  
bus-cycles [Hardware event]  
  
cpu-clock [Software event]  
task-clock [Software event]  
page-faults OR faults [Software event]  
context-switches OR cs [Software event]  
cpu-migrations OR migrations [Software event]  
minor-faults [Software event]  
major-faults [Software event]  
alignment-faults [Software event]  
emulation-faults [Software event]  
dummy [Software event]  
  
L1-dcache-loads [Hardware cache event]  
L1-dcache-load-misses [Hardware cache event]  
L1-dcache-stores [Hardware cache event]  
L1-dcache-store-misses [Hardware cache event]  
L1-dcache-prefetch-misses [Hardware cache event]  
L1-icache-loads [Hardware cache event]  
L1-icache-load-misses [Hardware cache event]  
dTLB-load-misses [Hardware cache event]  
dTLB-store-misses [Hardware cache event]  
iTLB-load-misses [Hardware cache event]  
branch-loads [Hardware cache event]  
branch-load-misses [Hardware cache event]
```

实际上 Android 移植的 Perf 还不完整，tracepoint 的事件还不支持，例如：block 事件，所以如果想要抓去一些内核子系统的性能信息就无法满足。Android 7.0 开始已经去掉了 Perf 工具，替代它的是 Simpleperf¹ 工具，对 tracepoint 的支持比原来的好很多。

3.3 获取系统热点进程

Perf 中的 top 工具可以列出当前 cpu 的热点，还可以附加 Kernel 的符号表让信息可方便分析。命令如下：

```
$ adb shell mkdir -p /data/local/symbols  
$ adb push vmlinux /data/local/symbols/vmlinux  
$ adb shell  
# perf top --vmlinux=/path/to/vmlinux -d 2
```

结果输出如下：

perf top 还可以只抓取指定进程的 pid，这一般是用在要优化某个程序是非常有用，命令如下：

```
perf top --vmlinux=/path/to/vmlinux -d 2 -p pid_of_prog
```

perf top 还和系统的 top 一样可以指定刷新间隔²，以上命令中的-d 选项就是这个功能，单位是秒。

3.4 获取进程的统计信息

perf stat 用于获取进程某个时间段内的 pmu 统计信息，命令如下：

```
# ./perf stat -p 1415
```

ctrl+c 退出，或发信号让 Perf 进程退出都可以看到统计结果，例如：

一些明显的异常值会被标注为红色，例如上图是浏览器跑 fishtank 时候抓的统计信息，可以看到分支预测的失败率非常高，结合 Perf 的热点分析工具可以进一步缩小范围找到分支预测失败的原因。

3.5 收集进程的 profile 数据

perf record 用于记录详细的 profile 数据，可以指定记录某个进程，还可以记录调用栈，命令如下：

```
# perf record -g -p pid -o /data/local/perf.data
```

也可以指定只抓取某个事件，事件列表可以通过上面的 perf list 得到，例如：

```
# ./perf record -e cache-misses -p 1415
```

3.6 分析 profile 数据

perf report 用户分析抓到的 profile 数据，一般会先把数据发到 pc 上再分析，命令如下：

```
adb pull /data/local/perf.data  
perf report --objdump=aarch64-linux-android-objdump --vmlinux=/path/to/vmlinux -  
-symfs ANDROID_PRODUCT_OUT/symbols -i perf.data
```

结果如图：

上图有‘+’的地方可以用‘enter’键来遍历其调用关系。

3.7 FlameGraph

还可以通过一些脚本来方便分析调用关系，Flame Graph 就是一个比较好用的可视化分析工具。

下载：

```
git clone https://github.com/brendangregg/FlameGraph.git
```

生成图形：

```
perf script --vmlinux=<kernel_folder>/vmlinux --symfs  
$ANDROID_PRODUCT_OUT/symbols -i perf.data | FlameGraph/stackcollapse-perf.pl |  
FlameGraph/flamegraph.pl > flamegraph.html
```

4. 在 Linux 平台使用

arm 版本的 linux 发行版很多都没有提供 Perf 的包，所以需要自己手动编译一个 Perf，由于 Perf 依赖的 elfutils/binutils/zlib，所以实际上需要交叉编译四个东西。

首先编译 zlib，[源码地址](#)

```
CC=aarch64-linux-gnu-gcc ./configure --  
prefix=/home/cmc/workspace/linaro/toolchain/armlinux/aarch64/gcc-linaro-6.3.1-  
2017.02-x86_64_aarch64-linux-gnu/aarch64-linux-gnu/libc/usr  
make && make install
```

==Note: prefix 要指向你的交叉编译工具的库目录==

编译 elfutils，我直接用的最新的版本的：

```
git clone git://sourceware.org/git/elfutils.git
```

配置：

```
cd /path/to/elfutils  
mkdir build  
.configure --enable-maintainer-mode --host=aarch64-linux-gnu --  
prefix=/home/cmc/workspace/linaro/elfutils/build
```

修改 Makefile：删除 elfutils 根目录下 Makefile 里面的 libcpu

修改 backends/Makefile：删除 backends/Makefile 中的 libebbl_i386 和 libebbl_x86_64 有关的所有东西

编译：

```
make && make install
```

编译 binutils，这个要考虑和 gcc 版本的兼容，我用的 2.28.1 的版本，[源代码地址](#)

```
cd /path/to/binutils
mkdir build
../configure --target=aarch64-linux-gnu --host=aarch64-linux-gnu --
prefix=/home/cmc/workspace/linaro/binutils-2.28.1/build
make && make install
```

编译 Perf，Perf 是 Kernel 一起发布的，所以直接下载一个 Kernel 就有了，但是交叉编译的话，需要改一些东西：

修改 Makefile.perf，在前面加入：

```
EXTRA_CFLAGS=-I/path/to/elfutils/build/include -L/path/to/elfutils/build/lib -
I/path/to/binutils/build/include -L/path/to/binutils/build/lib
WERROR=0
NO_LIBPERL=1
NO_LIBPYTHON=1
```

编译

```
cd /path/to/kernel/tools/perf
make -f Makefile.perf perf ARCH=arm64
CROSS_COMPILE=/home/cmc/workspace/linaro/toolchain/armlinux/aarch64/gcc-linaro-
6.3.1-2017.02-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu- -j8
```

理论上在 arm 的 linux 发行版上直接编译 Perf 应该也是可以的，但是我没有试过。用法的话和 Android 是一样的，这里就不叙说了。

5. Simpleperf 使用

Android 7.0 开始提供了一个更完整的 Perf 版本 Simpleperf：

```
source build/envsetup.sh
lunch
mma system/extras/simpleperf
```

Simpleperf 相对之前 google 移植的 Perf 有以下改进

- 支持剖析 apk 中兼容的共享库，从 .gnu_debugdata 段读取符号表和调试信息
- 提供更方便分析的脚本
- 纯静态，所以和 Android 版本无关，只要指令集兼容都能跑

ndk r13 开始就提供了 Simpleperf 工具，所以也可以直接下载编译好的工具：

```
git clone https://aosp.tuna.tsinghua.edu.cn/platform/prebuilt/simpleperf
```

用法上和 Perf 是类似的，命令基本通用，可以直接参考上面 Perf 的命令。

Simpleperf 更多信息，特别是调试 java 程序的方法，请参考[官方手册](#)

6. perf event格式说明

目前perf的event支持如下几种格式:

- symbolic event: 这是最常见的用法, 即给出perf list命令看到的完整名字, 下面是一个例子:

```
rk3399_Android11:/data/local/tmp # ./perf_aarch64 list pmu
    CCI_500/cci_rq_stall_addr_hazard/                                [Kernel PMU event]
    CCI_500/cci_snoop_access_filter_bank_0_1/                          [Kernel PMU event]
    CCI_500/cci_snoop_access_filter_bank_2_3/                          [Kernel PMU event]
    CCI_500/cci_snoop_access_filter_bank_4_5/                          [Kernel PMU event]
    CCI_500/cci_snoop_access_filter_bank_6_7/                          [Kernel PMU event]
    CCI_500/cci_snoop_access_miss_filter_bank_0_1/                     [Kernel PMU event]
    CCI_500/cci_snoop_access_miss_filter_bank_2_3/                     [Kernel PMU event]
    CCI_500/cci_snoop_access_miss_filter_bank_4_5/                     [Kernel PMU event]
    CCI_500/cci_snoop_access_miss_filter_bank_6_7/                     [Kernel PMU event]
    CCI_500/cci_snoop_back_invalidation/                               [Kernel PMU event]
    CCI_500/cci_snoop_cd_hs/                                         [Kernel PMU event]
    CCI_500/cci_snoop_rq_stall_tt_full/                                [Kernel PMU event]
    CCI_500/cci_snoop_rq_tzmp1_prot/                                 [Kernel PMU event]
    CCI_500/cci_snoop_stall_alloc_busy/                               [Kernel PMU event]
    CCI_500/cci_snoop_stall_tt_full/ # 例
如我想抓这个event
    CCI_500/cci_wrq/                                              [Kernel PMU event]
    CCI_500/mi_r_data_beat_any,source=?/                            [Kernel PMU event]
    CCI_500/mi_r_data_stall,source=?/                             [Kernel PMU event]
    CCI_500/mi_rrq_stall,source=?/                               [Kernel PMU event]
    CCI_500/mi_w_data_beat_any,source=?/                           [Kernel PMU event]
    CCI_500/mi_w_data_stall,source=?/                            [Kernel PMU event]
    CCI_500/mi_w_resp_stall,source=?/                           [Kernel PMU event]
    CCI_500/mi_wrq_stall,source=?/                               [Kernel PMU event]
    CCI_500/si_r_data_stall,source=?/                            [Kernel PMU event]
    CCI_500/si_r_stall_arbit,source=?/                           [Kernel PMU event]
    CCI_500/si_rq_stall_ot_limit,source=?/                         [Kernel PMU event]
rk3399_Android11:/data/local/tmp # ./perf_aarch64 stat -e
cci_snoop_stall_tt_full # 直接给出event的完整名字即可
^C
Performance counter stats for 'system wide':
      29      cci_snoop_stall_tt_full
 4.512917370 seconds time elapsed
```

- raw event: 用上面介绍的符号名有个缺陷, 就是有一些非架构通用的event并不会出现在perf list里, 即PMU驱动并不会把所有可支持的event全部符号化。这时候就需要用到raw event, 其格式是rNNN, 其中NNN是event的16进制格式, event的16进制值可以在CPU手册的PMU章节找到, 下面是一个例子:

```
rk3399_Android11:/data/local/tmp # ./perf_aarch64 stat -e cpu_cycles ./mhz
1798 MHz, 0.5562 nanosec clock

Performance counter stats for './mhz':
      1688410875      cpu_cycles
      (50.00%)
  0.944476845 seconds time elapsed
```

```

0.941888000 seconds user
0.000000000 seconds sys

rk3399_Android11:/data/local/tmp # ./perf_aarch64 stat -e r11 ./mhz      #
cpu_cycles对应的16进制值是0x11
1798 MHz, 0.5562 nanosec clock

Performance counter stats for './mhz':

1686673192          r11

0.941898803 seconds time elapsed

0.939286000 seconds user
0.000000000 seconds sys

```

- 带修饰的raw event或symbolic event: 格式就是在上面两种event的后面加上 :x，其中X的意义如下：

```

u - user-space counting
k - kernel counting
h - hypervisor counting
I - non idle counting
G - guest counting (in KVM guests)
H - host counting (not in KVM guests)
p - precise level
P - use maximum detected precise level
S - read sample value (PERF_SAMPLE_READ)
D - pin the event to the PMU
W - group is weak and will fallback to non-group if not schedulable

rk3399_Android11:/data/local/tmp # perf record -a -e cpu-cycles:p ...      #
use ibs op counting cycles
rk3399_Android11:/data/local/tmp # perf record -a -e r11:p ...           #
same as -e cpu-cycles:p

```

- 带格式参数的event: 目前比较常见的PMU有三种：CPU、BUS、自定义，其中CPU就是对应我们常见的Cortex-A5x/Cortex-A7x这些，BUS则对应CCI和DSU(还有CMN等BUS，我司SOC都没用到过)，自定义则是vendor自己加的一些PMU event，每一个PMU设备都有自己的格式要求，这个可以通过如下命令看到：

```

rk3399_Android11:/data/local/tmp # ls
/sys/bus/event_source/devices/CCI_500/format/
event source
# 上面的event和source就是cci 500这个device的参数名了

```

找到格式以后，就可以通过如下命令来指定具体event：

```

./perf_aarch64 stat -e CCI_500/event=0xa,source=0xf/      #
cci_snoop_stall_tt_full
./perf_aarch64 stat -e cci_snoop_stall_tt_full           # 等同于上面命令

```

这种格式的好处是方便指定同一种event的不同实例，以CCI500为例，它有7个slave和6个master接口都有同一套event list，用这种方式就很容易指定不同的接口，下面是一个例子：

```
./perf_aarch64 stat -e CCI_500/event=0x1a,source=0x0/      # slave 0的  
si_w_data_stall  
./perf_aarch64 stat -e CCI_500/event=0x1a,source=0x1/      # slave 1的  
si_w_data_stall
```

不同格式参数的含义，需要结合不同的PMU驱动和手册去看，这里就不具体叙述了。

1. 后面也会简单介绍一些Simpleperf²
2. 这个是指top统计信息的刷新间隔而不是采样间隔²