Linux低延迟图形化分析工具使用手册

1. 整体框架 2

2. 源码目录结构 2

3. 使用方法 3

3.1 编译Target端可执行程序 3

3.2 Target端部署 3

3.3 搜集数据 4

3.3.1 PC端驱动引擎所支持的功能 4

3.3.2 测试案例 5

3.3.3 数据捕获与生成图形化结果 6

4. 数据分析 8

4.1 内核态跟踪信息 9

4.2 用户态的跟踪信息 9

5. 用户态插桩方法 10

5.1 C++函数跟踪：ATRACE\_CALL 10

5.2 代码scenario的跟踪：atrace\_begin && atrace\_end 11

5.2.1 atrace\_begin函数原型 11

5.2.2 atrace\_end函数原型 12

5.2.3 trace tag头文件awtracetag.h 12

5.3 代码中调用初始化函数aw\_trace\_init 12

6. 内核函数跟踪 12

7. 如何只跟踪用户态event 13

8. 版本发布 13

# 1. 整体框架

Target端

HOST端

PC驱动引擎(python脚本)

通讯链路：ADB

TRACE-Collect

FTRACE

Host端：一套python写的引擎；用于驱动target端启动trace-collect后台守护进程去启动/搜集ftrace信息；通过adb将原始信息回传；借助模块生成图形化的html结果文件。

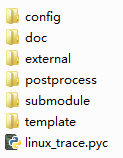
通讯链路：Host 与Target之间的通讯通过ADB完成

TRACE-Collect：运行在target端，即开发板上的用于搜集ftrace信息的后台守护进程。

FTRACE：linux内核ftrace机制

# 2. 源码目录结构

见如下截图：



config: 配置文件

doc : 文档目录

external : target端源码，包括TRACE-collect源码；用来对调试trace-event点进行插桩的libawtrace.so源码；以及一些测试案例源码。

postprocess：后处理脚本

submodule : PC端引擎用到的各种子模块

template : PC端用于生成图形化结果的模板

linux\_trace.pyc : PC端驱动引擎，由python编写而成

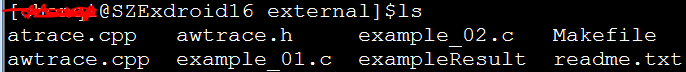
running\_log：运行日志文件

result : 用于保存结果的目录，运行后才会产生

# 3. 使用方法

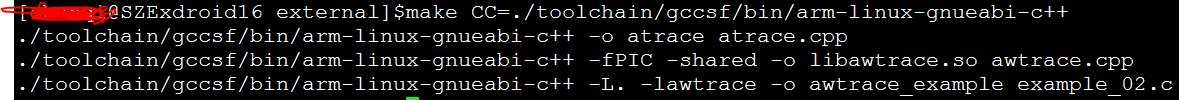
## 3.1 编译Target端可执行程序

进入第二章介绍的external目录：



截图3.1-01

执行make CC=./toolchain/gcchf/bin/arm-linux-gnueabihf-c++ （CC=/your/c++/compiler/path），必须确保编译以下程序所有的编译器和将要进行数据搜集的target目标板上所用的编译器是一致的：



截图3.1-02

即可得到：

atrace： target端TRACE-Collect后台进程

libawtrace.so：对C && C++源码进行trace点插桩的动态库

awtrace\_example : 一个使用libawtrace.so接口进行插桩的实例，演示如何使用ATRACE\_CALL()，atrace\_begin/atrce\_end对函数或者程序片段手动进行插桩跟踪。具体的使用请直接参考本目录下源码：example\_02.cpp。

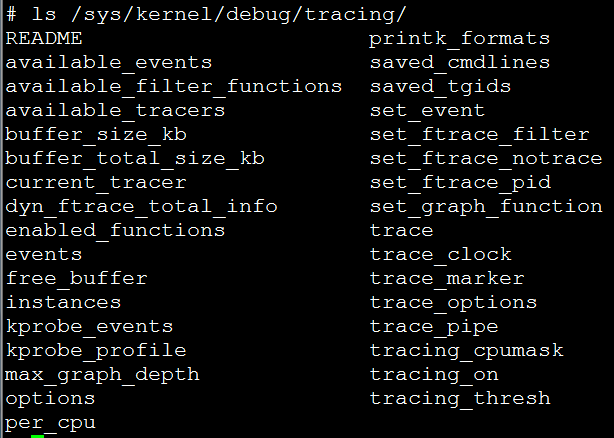
## 3.2 Target端部署

1. . 将atrace通过adb push到target的/bin目录下，并chmod 777 /bin/atrace
2. . 将libawtrace.so通过adb push到/usr/lib目录下
3. . 建议将awtrace\_example通过adb push到target任何目录下，建议放在/tmp目录，chmod 777
4. . 确保target端的debugfs是正常挂载的，并且是挂载在/sys/kernel/debug目录下



截图3.2-01

1. . 确保target端的ftrace功能正常，查看/sys/kernel/debug/tracing目录是否存在



截图3.2-02

1. . 确保PC与target之间的adb通讯正常

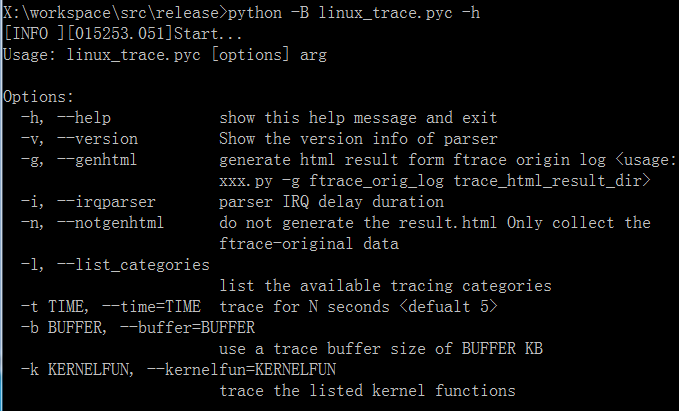
## 3.3 搜集数据

本章结合测试程序awtrace\_example，详细阐述如何捕获target端的ftrace数据。

### 3.3.1 PC端驱动引擎所支持的功能

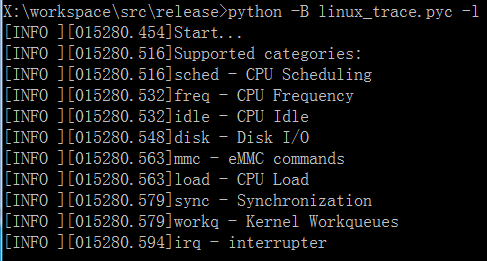
进入第2章介绍的源码主目录，就能看到python文件linux\_trace.pyc，这个就是工具PC端的驱动总引擎，工具所有命令均由它进行驱动。

通过help命令（python -B linux\_trace.pyc -h）可知道工具当前所支持的功能：



截图3.3.1-01

当前版本支持的内核跟踪策略（请参考命令“python -B linux\_trace.pyc -l”的运行结果）：

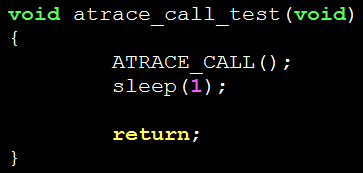


截图3.3.1-02

### 3.3.2 测试案例

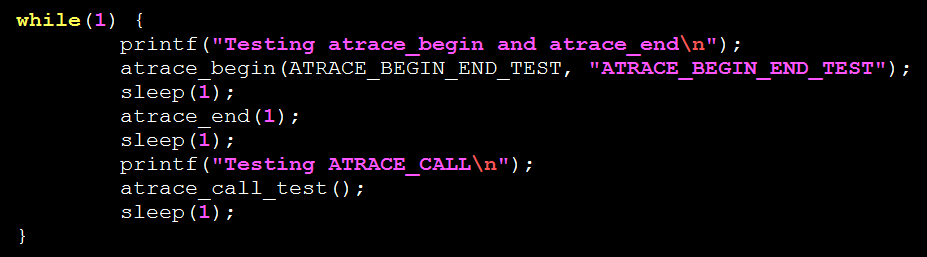
本小节阐述下文用到的测试程序awtrace\_example，源码文件为：external/example\_02.cpp。

主要源码：



截图3.3.2-01

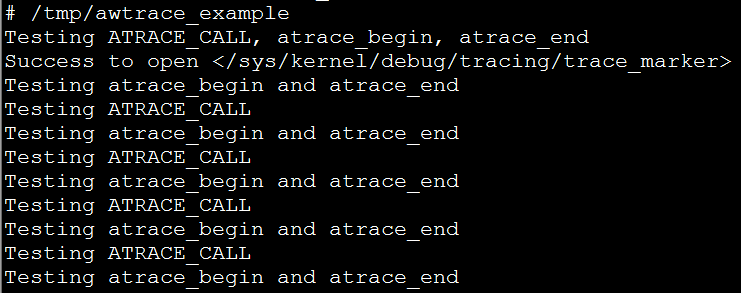
“截图3.3.2-01”中ATRACE\_CALL()：可用于用户态的函数跟踪，此处跟踪函数atrace\_call\_test()



截图3.3.2-02

“截图3.3.2-01”中atrace\_begin()与atrace\_end()：用于跟踪用户态的指定代码段，此处仅跟踪sleep(1)这一行。

按照上面“3.1”小节的步骤编译并adb push awtrace\_example到target板子端后，并在板子上运行awtrace\_example，将会反复输出下面语句（对应源码中的while语句）：



截图3.3.2-03

### 3.3.3 数据捕获与生成图形化结果

依据“3.3.2”中方法在target端运行测试程序awtrace\_example，即可在PC端启动数据捕获总引擎。

参考命令：python -B linux\_trace.pyc -t 6 -b 4096 sched irq freq

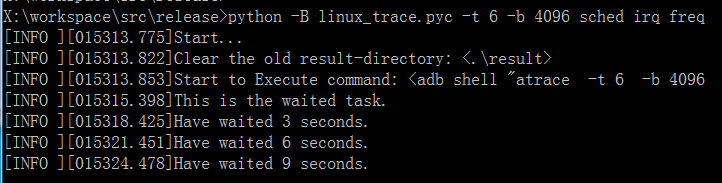
此命令含义：

-t 6 : 捕获6秒钟

-b 4096 ： buffer设置为4096KB

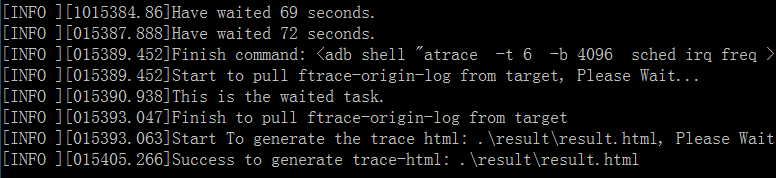
sched irq freq：所跟踪的eveve。sched：调度，irq：中断，freq：频率

命令(python -B linux\_trace.pyc -t 6 -b 4096 sched irq freq)执行截图如下：



......

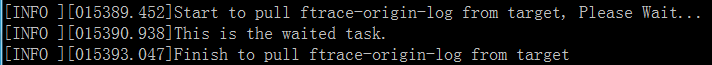
......



截图3.3.3-01

注意按照上面截图的提示，命令完全执行完毕需要超过72秒（**后续考虑优化，特别是加入irq跟踪后时间有点长**），但命令抓取的仅仅是命令启动后6秒钟的信息，即终端打印“Have waited 6 seconds.”之后的信息系统是不捕获的。

从“Have waited 6 seconds.”开始，一直到“Finish command:”这段时间，是在等待target板子端复制ftrace原始信息到文件中。



截图3.3.3-02

上图是PC引擎将ftrace原始信息通过adb pull到PC端。



截图3.3.3-03

上图是将原始的ftrace信息转换为HTML，得到图像化分析结果。

如以上命令执行成功，在源码主目录下将生成一个result子目录，子目录里面有两个文件：

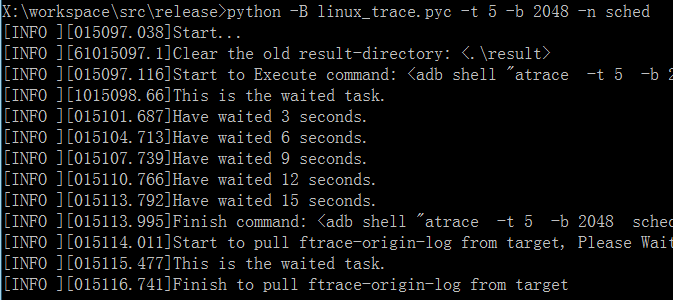


ftrace\_orig.log：target端捕获的ftrace原始跟踪数据

result.html：图形化分析结果，可使用google浏览器chrome打开。

为了方便拓展，增加了一个选项“-n”，如将命令由“python -B linux\_trace.pyc -t 6 -b 4096 sched”变为“python -B linux\_trace.pyc -t 6 -b 4096 -n sched”。则此命令只采集ftrace的原始log，不会自动生成图形化分析结果（**即只生成ftrace\_orig.log，不会自动生成result.html**）。

命令“python -B linux\_trace.pyc -t 5 -b 2048 -n sched”的运行截图如下：



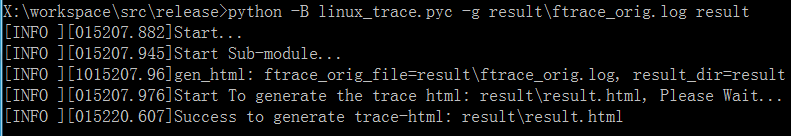
截图3.3.3-04

如想基于这个原始的ftrace log生成图形化分析结果，手动运行命令“python -B linux\_trace.pyc -g result\ftrace\_orig.log result”即可，结果文件result.html还是保存在同一个result目录下。

result/ftrace\_orig.log：ftrace原始log文件

result：图形化分析结果文件result.html的保存目录

命令“python -B linux\_trace.pyc -g result\ftrace\_orig.log result”运行截图如下：



截图3.3.3-05

# 4. 数据分析

已经“3.3.3”小节得到HTML图形化分析结果文件result.html，打开google浏览器chrome，地址栏输入“<chrome://tracing/>”，通过“Load”按钮打开result.html文件。即可像android systrace生成的html结果文件一样进行分析。

解析结果参考文件：

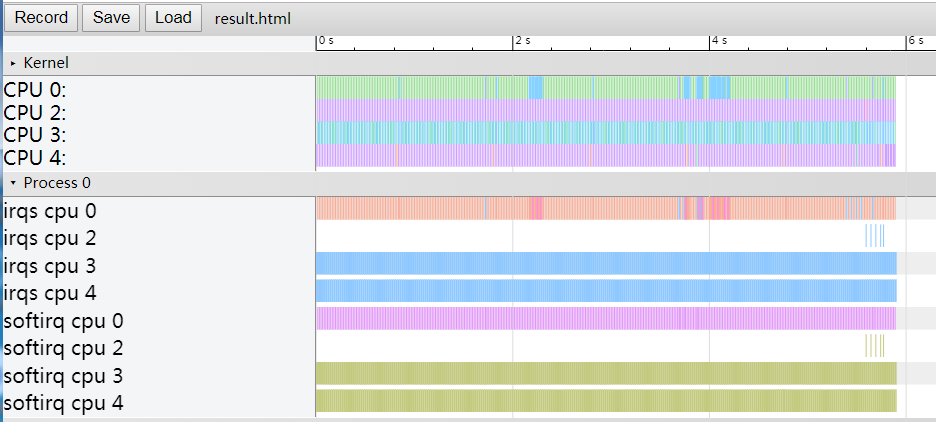
exampleResult/usertrace/ftrace\_orig.log：ftrace原始log文件

exampleResult/usertrace/result.html：图形化分析结果

## 4.1 内核态跟踪信息

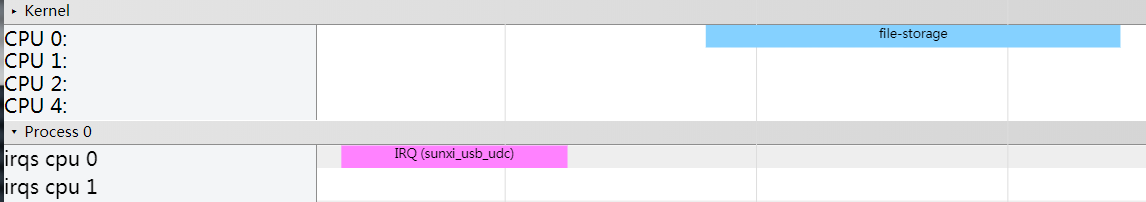
内核跟踪功能被划分为组租跟踪策略，当前所能支持的内核跟踪策略见上面“截图3.3.1-02”，也可以通命令“python -b linux\_trace.pyc -l”得到当前版本的详细信息。

一些分析结果的参考截图如下：



截图4.1-01

以上截图反映的是跟踪的这6秒时间段内，各个CPU的调度情况和中断产生情况；放大即可得到详细信息进程调度信息，举例如下：



截图4.1-02

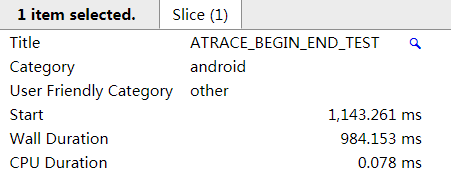
## 4.2 用户态的跟踪信息

这些信息是用户态人为手动添加的跟踪信息，与“3.3.2”小节里面分析的example\_02.cpp源码是一一对应的。

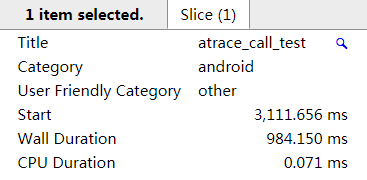
在结合“4.1”，即可将内核态与用户态的跟踪情况放在一起分析，依据时间戳分析其系统性能问题。



截图4.2-01



截图4.2-02



截图4.2-03

# 5. 用户态插桩方法

上面第4章ftrace跟踪信息包括两部分：用户态与内核态。

内核态：使用者不用考虑，所有功能由底层开发人员支持。

用户态：工具提供一种插桩方法与动态库。针对特殊的场景，模块开发人员需在其关键通路上插入跟踪点。当前支持C && C++的函数与代码scenario跟踪两种跟踪方式。

下文例子的参考代码见源码文件：

awsystrace/external/example\_02.cpp

## 5.1 C++函数跟踪：ATRACE\_CALL

用于跟踪C++的函数（可参考上面4.2小节），参考源码：external/example\_02.cpp。

**提醒：建议在C++源码中使用ATRACE\_CALL，C源码中使用下文介绍的atrace\_begin/atrace\_end。如您在C中使用ATRACE\_CALL，并使用C编译进行编译，编译会出错。**

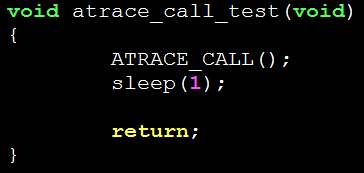
解析结果参考文件：

exampleResult/usertrace/ftrace\_orig.log：ftrace原始log文件

exampleResult/usertrace/result.html：图形化分析结果

使用方法：

1. . 导入头文件：#include "awcpptrace.h"
2. . 在需要跟踪的C++ 函数的入口，加上一行代码： ATRACE\_CALL()，例如下面的例子，跟踪函数atrace\_call\_test：



截图5.1-01

1. . 链接时候：-lawtrace (动态库libawtrace.so的编译方法将3.1小节)
2. . 跟踪结果参考上面4.2小节中的截图。

ATRACE\_CALL用于跟踪函数，此测试案例跟踪的是函数atrace\_call\_test，图形化分析显示出来的就是这个函数名，请参考“截图4.2-01”中图形化显示出来的字符串“atrace\_call\_test”。

## 5.2 代码scenario的跟踪：atrace\_begin && atrace\_end

这两个函数用于跟踪指定的C && C++代码片段；请可参考源码：external/example\_02.cpp

**提醒：C与C++中均可使用。如用这一对函数在C代码中进行函数跟踪，函数所有return地方均需要加上atrace\_end。**

解析结果参考文件：

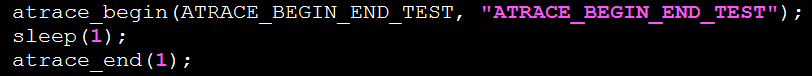
exampleResult/usertrace/ftrace\_orig.log：ftrace原始log文件

exampleResult/usertrace/result.html：图形化分析结果

使用方法：

(1). 导入头文件：#include "awctrace.h"

(2). 在需要跟踪的代码开始处加上一行代码atrace\_begin()，结束跟踪的地方加上一行代码atrace\_end()，例子如下：



截图5.2-01

以上截图中的代码跟踪的是sleep(1)。

1. . 链接时候：-lawtrace (动态库libawtrace.so的编译方法将3.1小节)
2. . 跟踪结果参考上面4.2小节中的截图。

### 5.2.1 atrace\_begin函数原型

函数原型：static inline void atrace\_begin(unsigned long tag, const char\* name)

tag: 一个长整形数，跟踪类别，每个跟踪都应有自己唯一的tag。建议将所有定义好的跟踪tag（跟踪类别）以宏定义的方式定义在头文件awtracetag.h中。

name: 字符串，跟踪名字，图形化显示的时候将显示这个字符串。参考“截图4.2-01”中图形化显示出来的字符串“ATRACE\_BEGIN\_END\_TEST”。

### 5.2.2 atrace\_end函数原型

函数原型：static inline void atrace\_end(unsigned long tag)

tag: 与上“5.2.1”小节介绍的含义一致；atrace\_end使用的tag必须和与之对应atrace\_begin所用的tag一样。

### 5.2.3 trace tag头文件awtracetag.h

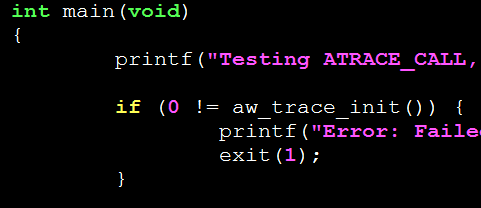
atrace\_begin与atrace\_end的参数均含有一个tag，此tag在后续版本中将可独立进行控制，动态enable/disable；一个模块定义一个tag，这样即可动态的打开关闭模块跟踪事件。

建议将统一平台下所有的tag均定义在头文件awtracetag.h中，统一引用。

## 5.3 代码中调用初始化函数aw\_trace\_init

aw\_trace\_init()函数负责在trace运行前进行必要的初始化。

因此，建议在您mai()主函数开始调用这个函数，如awsystrace/external/example\_02.cpp的如下源码：



# 6. 内核函数跟踪

通过命令的“-k”选项，可以跟踪内核export出来的任何函数（即内核符号表里面的任何内核函数）。“-k”之后是一个跟踪函数名的列表，函数名之间使用逗号隔开。

内核函数参考命令：python -B linux\_trace.pyc -b 4096 -k generic\_handle\_irq,gic\_handle\_irq

此命令跟踪两个内核函数：generic\_handle\_irq,gic\_handle\_irq

解析结果参考文件：

exampleResult/kfuntrace/ftrace\_orig.log：ftrace原始log文件

exampleResult/kfuntrace/result.html：图形化分析结果

图形化分析结果截图：



截图6-01

依据此方法使用者可以跟踪任何export到内核符号表的函数，并图形化显示出来。

# 7. 如何只跟踪用户态event

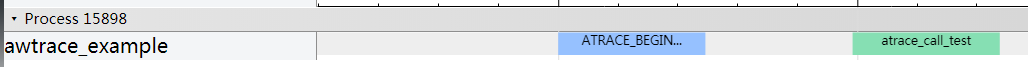
为方便分析，如不想看内核态event，只关注用户态事件，可依如下面步骤：

1. . 在Target板子上运行测试程序：awtrace\_example （3.3.2 测试案例）
2. . PC端运行命令：python -B linux\_trace.pyc -t 6 -b 4096 idle

idle这个内核category产生的内核event少，方便用于观察用户态事件。

以上步骤产生的结果文件见目录：exampleResult/onlyuser/。

Google浏览器Chrome打开exampleResult/onlyuser/result.html，结果截图如下：



截图7-01

从Chrome打开的图形结果看，内核event很少，可以方便分析用户态事件。

# 8. 版本发布

本工具套件基本是由python脚本组成，发布版本会将源码过滤，使用编译得到的.pyc。因为不同python版本，其“Image Number”是不一样的。当前发布版本运行环境为“Python 2.7.13”，如你本地python不是2.7版本，可能会遇到“Image Number error”类似的错误。此时请替换本地机器上的python版本，或者联系工具开发组生成新的发布版本。

工具套件发布版本的生成（需要工具套件源码）：

1. . 请使用者提供使用环境所用的python版本号
2. . cd tolls，运行脚本release.py，在目录下会生成目录”release”，拷贝此目录，名字替换即可得到本工具的发布版本。