diagnose-tools 2.0 功能使用说明

内容简介

本文档介绍diagnose-tools-2.0版本的使用方法。

# 支持的版本

## 支持的版本

目前的版本支持Centos 7.5 / 7.6

经过验证，工具也支持如下版本，相应的代码会陆续合入：

Centos 5.x / 6.x

Ubuntu

Linux 4.19

# 编译

推荐在centos 7.6中编译

make devel //安装编译环境

make deps //编译三方开源包

make module //编译内核模块

make tools //编译用户态工具

make rpm //制作rpm包

# 安装和卸载KO

在使用模块功能之前，需要用如下命令安装KO模块：

﻿diagnose-tools install

安装成功后，控制台有如下提示：

﻿installed successfully

在使用完模块功能后，需要用如下命令卸载KO模块：

﻿diagnose-tools uninstall

卸载成功后，控制台有如下提示：

*uninstalled successfully*

# 2.0正式版本的功能

目前，diagnose-tools-2.0正式发布的版本有如下几个功能：

**实用小工具pupil：**按照tid查询特定线程在主机上的PID/进程名称/进程链/堆栈等等。

**sys-delay：**监控syscall长时间运行引起调度不及时。间接引起系统Load高、业务RT高。

**sys-cost：**统计系统调用的次数及时间。

**sched-delay** : 监控系统调度延迟。找到引起调度延迟的进程。

**irq-delay：**监控中断被延迟的时间。

**irq-stats：**统计中断/软中断执行次数及时间。

**irq-trace：**跟踪系统中IRQ/定时器的执行。

**load-monitor：**监控系统Load值。每10ms查看一下系统当前Load，超过特定值时，输出任务堆栈。这个功能多次在线上抓到重大BUG。

可以分别监控Load/Load.R/Load.D/Task.D等指标。

**run-trace**：监控进程在某一段时间段内，在用户态/内核态运行情况。

**perf**: 对线程/进程进行性能采样，抓取用户态/内核态调用链。

**kprobe：**在内核任意函数中，利用kprobe监控其执行，并输出火焰图。

**uprobe：**在用户态应用程序中使用探针，在应用中挂接钩子。

**utilization**：监控系统资源利用率，找到CPU被哪些野进程干扰，以及进程对内存的使用情况。

**exit-monitor：**监控任务退出。在退出时，打印任务的用户态堆栈信息。

**mutex-monitor：**监控长时间持有mutex的流程。

**exec-monitor**: 监控进程调用exec系统调用创建新进程。

**alloc-top：**统计内存分配数量，按序输出内存分配多的进程

**high-order：**监控分配大内存的调用链

**drop-packet**：监控内核TCP/IP各个流程中的丢包。

**tcp-retrans**：监控TCP/IP套接字上的重传。

**ping-delay：**监控ping报文在内核中的路径，确认影响报文延迟的原因。

**rw-top：**监控写文件。找到突发引起文件读写的进程/调用链。

**fs-shm：**本功能监控当前打开的SHM文件。

**fs-orphan**：导出文件系统孤儿节点。

**fs-cache：监控文件系统缓存占用情况，统计每个文件占用的缓存数量**。

**reboot**：监控系统重启信息，打印出调用sys\_reboot系统调用的进程名称以及进程链。

## pupil小工具

### 查看diagnose-tools版本号

可以执行如下命令来查询版本号：

﻿diagnose-tools -v

﻿diagnose-tools -V

﻿diagnose-tools --version

结果如下：

﻿diagnose-tools tools version 2.0-rc4

### 查看线程信息

在容器或者宿主机上面，根据线程PID，输出其线程信息：

* 线程所在的CGROUP名称
* PID
* 进程名称
* 进程链信息
* 内核态堆栈

在控制台中运行如下命令查看线程信息：

diagnose-tools task-info --pid=$PID

其中，$PID是要查看的进程ID。

也可以用如下命令查看进程中所有线程的信息：

diagnose-tools task-info --tgid=$PID

最后，用如下命令获得结果：

diagnose-tools task-info --report

下图是运行结果示例：

线程详细信息： 4959

时间：[1584776688:293341].

进程信息： [/ / JS Helper]， PID： 4935 / 4959

##CGROUP:[/] 4959 [013] 采样命中

内核态堆栈：

#@ 0xffffffff8111ac34 futex\_wait\_queue\_me ([kernel.kallsyms])

#@ 0xffffffff8111b8a6 futex\_wait ([kernel.kallsyms])

#@ 0xffffffff8111dcb5 do\_futex ([kernel.kallsyms])

#@ 0xffffffff8111e055 SyS\_futex ([kernel.kallsyms])

#@ 0xffffffff81003c04 do\_syscall\_64 ([kernel.kallsyms])

#@ 0xffffffff8174bfce entry\_SYSCALL\_64\_after\_swapgs ([kernel.kallsyms])

用户态堆栈：

#~ 0x7f28339f9965 \_\_pthread\_cond\_wait ([symbol])

#~ 0x7f282bc82f25 \_ZN2JS15PerfMeasurement19canMeasureSomethingEv ([symbol])

#~ 0x7f282c080b9e \_ZN2JS19PeakSizeOfTemporaryEPK9JSContext ([symbol])

#~ 0x7f282c09fc02 \_ZN2JS14AddServoSizeOfEP9JSContextPFmPKvEPNS\_20ObjectPrivateVisitorEPNS\_10ServoSizesE ([symbol])

#~ 0x7f28339f5dd5 start\_thread ([symbol])

#\* 0xffffffffffffff JS Helper (UNKNOWN)

进程链信息：

#^ 0xffffffffffffff /usr/bin/gnome-shell (UNKNOWN)

#^ 0xffffffffffffff /usr/libexec/gnome-session-binary --session gnome-classic (UNKNOWN)

#^ 0xffffffffffffff gdm-session-worker [pam/gdm-password] (UNKNOWN)

#^ 0xffffffffffffff /usr/sbin/gdm (UNKNOWN)

#^ 0xffffffffffffff /usr/lib/systemd/systemd --switched-root --system --deserialize 22 (UNKNOWN)

##

线程详细信息： 4960

时间：[1584776688:293352].

进程信息： [/ / llvmpipe-0]， PID： 4935 / 4960

##CGROUP:[/] 4960 [014] 采样命中

内核态堆栈：

注意：启动进程的父进程可能已经退出，这样有可能找不到直接启动进程的父进程。

同样的，可以从上面的输出结果中提取出火焰图。

如：

diagnose-tools task-info --tgid=$PID --report > task.log

diagnose-tools flame --input=task.log --output=task.svg

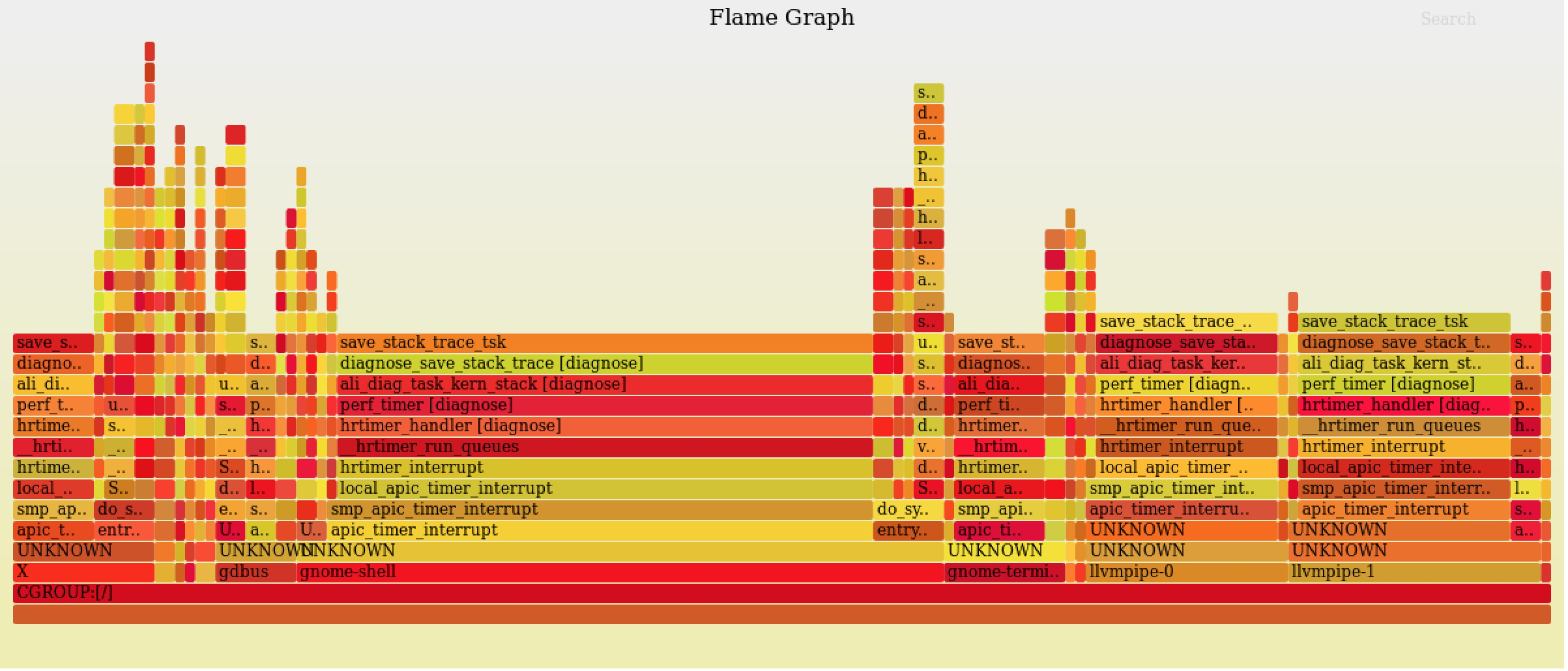
## 临时文件转火焰图

sys-delay / irq-delay / load-monitor / perf等功能都能够输出进程堆栈信息，可以将这些信息保存在临时文件中，例如tmp.txt中。

使用如下命令，可以将临时文件中的数据生成火焰图：

diagnose-tools flame --input=tmp.txt --output=perf.svg

该命令指定了数据来源文件为tmp.txt，并指定火焰图文件为perf.svg。成功后，可以使用浏览器直接打开perf.svg。如下所示：



你可以在浏览器中与火焰图互动：将鼠标移到不同层级的块中，看其详细信息，也可以点击块。

关于火焰图的说明，请参见：

<http://www.ruanyifeng.com/blog/2017/09/flame-graph.html>

## sys-delay

sys-delay功能抓取在syscall中长时间执行，导致调度被延迟的情况。

本工具的原理，是在定时器中监控当前任务的cond\_resched和schedule调用情况。如果在一段时间范围内都没有进行调度，说明在内核中有长时间执行的流程。这样的流程对业务RT和系统load都是有影响的。应当及时优化掉。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

﻿diagnose-tools sys-delay --help

结果如下：

sys-delay usage:

--help sys-delay help info

--activate

verbose VERBOSE

threshold THRESHOLD(MS)

style dump style: 0 - common, 1 - process chains

--deactivate

--settings print settings.

--report dump log with text.

--test loop in sys for 100ms, so triger this monitor.

--log format:"sls=1.log[ syslog=1]" to store in file or syslog.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools sys-delay --activate

在激活本功能时，可以指定如下参数：

﻿ threshold 设置监控阀值，单位是ms。默认值是50。

style如果为1，输出进程链。其他值不输出。需在—activate激活前设置style为1，才能输出进程链；

verbose 设置输出级别，目前未用。

例如，使用如下命令，可以：

1. 监控阀值修改为60ms。当在syscall中执行时间超过60ms，就会在系统中记录下异常信息
2. 将输出级别修改为1。目前该参数无实际意义

diagnose-tools sys-delay --activate="threshold=60 verbose=1"

如果激活功能成功，将打印如下信息：

功能设置成功，返回值：0

阀值(ms)： 60

输出级别： 1

STYLE： 0

如果不能成功激活功能，将打印如下信息：

功能设置失败，返回值：-16

阀值(ms)： 60

输出级别： 1

STYLE： 0

### 查看设置参数

使用如下命令查看本功能的设置参数：

﻿diagnose-tools sys-delay --settings

结果如下：

功能设置：

是否激活： √

阀值(ms)： 60

输出级别： 1

STYLE： 0

### 测试用例

执行如下命令触发测试用例：

sh /usr/diagnose-tools/test.sh sys-delay

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools sys-delay --report

输出结果示例如下：

*抢占关闭, 时长： 55(ms).*

*时间：[1584003506:277464].*

*进程信息： [/ / diagnose-tools]， PID： 79757 / 79757*

*##CGROUP:[/] 79757 [001] 采样命中*

*内核态堆栈：*

*#@ 0xffffffff81025022 save\_stack\_trace\_tsk ([kernel.kallsyms])*

*#@ 0xffffffffa16483e9 diagnose\_save\_stack\_trace [diagnose] ([kernel.kallsyms])*

*#@ 0xffffffffa1648d5e ali\_diag\_task\_kern\_stack [diagnose] ([kernel.kallsyms])*

*#@ 0xffffffffa16510ed syscall\_timer [diagnose] ([kernel.kallsyms])*

*#@ 0xffffffffa164db09 hrtimer\_handler [diagnose] ([kernel.kallsyms])*

*#@ 0xffffffff810aa0d2 \_\_hrtimer\_run\_queues ([kernel.kallsyms])*

*#@ 0xffffffff810aa670 hrtimer\_interrupt ([kernel.kallsyms])*

*#@ 0xffffffff8104cbf7 local\_apic\_timer\_interrupt ([kernel.kallsyms])*

*#@ 0xffffffff81664cdf smp\_apic\_timer\_interrupt ([kernel.kallsyms])*

*#@ 0xffffffff81660432 apic\_timer\_interrupt ([kernel.kallsyms])*

*#@ 0xffffffff81317dc8 \_\_const\_udelay ([kernel.kallsyms])*

*#@ 0xffffffffa16512f6 sys\_delay\_syscall [diagnose] ([kernel.kallsyms])*

*#@ 0xffffffffa1649bdb trace\_sys\_enter\_hit [diagnose] ([kernel.kallsyms])*

*#@ 0xffffffff81023ec6 syscall\_trace\_enter ([kernel.kallsyms])*

*#@ 0xffffffff8165f331 tracesys ([kernel.kallsyms])*

*用户态堆栈：*

*#~ 0x50f199 syscall ([symbol])*

*#~ 0x4ace24 generic\_start\_main ([symbol])*

*#\* 0xffffffffffffff diagnose-tools (UNKNOWN)*

*进程链信息：*

*#^ 0xffffffffffffff diagnose-tools sys-delay --test (UNKNOWN)*

*#^ 0xffffffffffffff -bash (UNKNOWN)*

*#^ 0xffffffffffffff /usr/sbin/sshd -D -R (UNKNOWN)*

*#^ 0xffffffffffffff /usr/sbin/sshd -D (UNKNOWN)*

*#^ 0xffffffffffffff /usr/lib/systemd/systemd --switched-root --system --deserialize 21 (UNKNOWN)*

*##*

这是一个引起内核长时间运行的调用链。通过这个调用链，内核开发同学可以方便的找到问题原因。

### 生成火焰图

可以用如下命令获取结果并生成火焰图：

diagnose-tools sys-delay --report > sys-delay.log

diagnose-tools flame --input=sys-delay.log --output=sys-delay.svg

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools sys-delay --deactivate*

关闭成功后，将会有如下打印输出信息：

sys-delay is not activated

如果关闭失败，将会有如下打印：

deactivate sys-delay fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

## sys-cost

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools sys-cost --help

结果如下：

sys-cost usage:

--help sys-cost help info

--activate

verbose VERBOSE

tgid process group that monitored

pid thread id that monitored

comm comm that monitored

--deactivate

--settings dump settings

--report dump log with text.

--log

sls=/tmp/1.log store in file

syslog=1 store in syslog

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools sys-cost --activate

在激活本功能时，可用参数为：

verbose：该参数控制输出的详细程度，可以是任意整数。当前未用。

tgid：要监控的进程ID。

pid：要监控的线程ID。

comm：要监控的进程名称。

例如，如下命令设置要监控的进程ID为1234：

diagnose-tools sys-cost --activate='tgid=1234'

如果成功，将在控制台输出如下：

功能设置成功，返回值：0

进程ID：1234

线程ID：0

进程名称：

输出级别：0

如果失败，将在控制台输出如下：

功能设置失败，返回值：-16

进程ID：0

线程ID：1234

进程名称：

输出级别：0

### 测试用例

执行如下命令触发测试用例：

sh /usr/diagnose-tools/test.sh sys-cost

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools sys-cost --settings

结果如下：

功能设置：

是否激活：√

进程ID：1234

线程ID：0

进程名称：

输出级别：0

### 查看结果

系统会记录一段时间内系统调用的次数和执行时间。执行如下命令查看本功能的输出结果：

diagnose-tools sys-cost --report

输出结果示例如下：

*CPU：0，时间：[1587955169:532299]*

*SYSCALL：0, COUNT：3003, COST：96881178*

*SYSCALL：1, COUNT：2022, COST：1234512*

*SYSCALL：2, COUNT：1246, COST：2164344*

*SYSCALL：3, COUNT：2477, COST：342223*

*SYSCALL：4, COUNT：46, COST：76456*

*SYSCALL：5, COUNT：53, COST：30890*

*SYSCALL：6, COUNT：0, COST：0*

*SYSCALL：7, COUNT：3400, COST：7641641*

*SYSCALL：8, COUNT：3, COST：1783*

*SYSCALL：9, COUNT：70, COST：237337*

*SYSCALL：10, COUNT：32, COST：140419*

*SYSCALL：11, COUNT：23, COST：118006*

每次输出结果后，历史数据将被清空。

### 生成火焰图

可以用如下命令获取结果并生成火焰图：

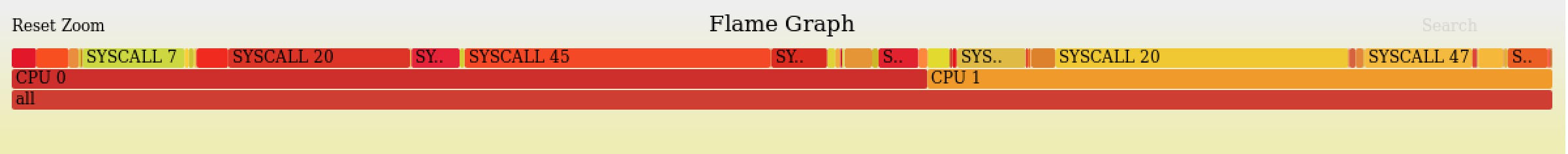
提取结果中以"\*\*"开头的行，然后用如下命令可以按调用次数输出火焰图：

diagnose-tools sys-cost --report | awk '{if (substr($1,1,2) == "\*\*") {print substr($0, 3)}}' | /usr/diagnose-tools/flame-graph/flamegraph.pl > sys-cost.count.svg

提取结果中以"\*#"开头的行，然后用如下命令可以按执行时间输出火焰图：

diagnose-tools sys-cost --report | awk '{if (substr($1,1,2) == "\*#") {print substr($0, 3)}}' | /usr/diagnose-tools/flame-graph/flamegraph.pl > sys-cost.cost.svg

火焰图如下所示：



### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools sys-cost --deactivate*

如果成功，控制台打印如下：

sys-cost is not activated

如果失败，控制台打印如下：

deactivate sys-cost fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

## sched-delay

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools sched-delay --help

结果如下：

sched-delay usage:

--help sched-delay help info

--activate

verbose VERBOSE

threshold THRESHOLD(MS)

tgid process group monitored

pid thread id that monitored

comm comm that monitored

--deactivate

--report dump log with text.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools sched-delay --activate

在激活本功能时，可用参数为：

verbose：该参数控制输出的详细程度，可以是任意整数。此参数目前未使用。

threshold：配置监控的阀值，当调度延迟超过该阀值将引起警告信息输出。时间单位是ms。

tgid 设置要监控的进程PID

pid 设置要监控的线程TID

comm 设置要监控的进程名称

例如，如下命令会将检测阀值设置为80ms。一旦系统有超过80ms的调度延迟，将输出其调用链：

diagnose-tools sched-delay --activate=”threshold=80”

如果成功，该命令在控制台上的输出如下：

功能设置成功，返回值：0

进程ID： 0

线程ID： 0

进程名称：

监控阈值(ms)： 80

输出级别： 0

如果失败，该命令在控制台上的输出如下：

功能设置失败，返回值：-38

进程ID： 0

线程ID： 0

进程名称：

监控阈值(ms)： 80

输出级别： 0

### 测试用例

执行如下命令触发测试用例：

sh /usr/diagnose-tools/test.sh sched-delay

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools sched-delay --settings

结果如下：

功能设置：

是否激活： √

进程ID： 0

线程ID： 0

进程名称：

监控阈值(ms)： 80

输出级别： 0

### 查看结果

系统会记录一段时间内调度延迟的调用链。执行如下命令查看本功能的输出结果：

diagnose-tools sched-delay --report

输出结果示例如下：

警告：调度被延迟 14 ms，NOW: 2065771, QUEUED: 2065757, 当前时间：[1584599791:768101]

##CGROUP:[/] 3868 [001] 采样命中

内核态堆栈：

#@ 0xffffffff8129b58b ep\_poll ([kernel.kallsyms])

#@ 0xffffffff8129c53e SyS\_epoll\_wait ([kernel.kallsyms])

#@ 0xffffffff81003c04 do\_syscall\_64 ([kernel.kallsyms])

#@ 0xffffffff81741c8e entry\_SYSCALL\_64\_after\_swapgs ([kernel.kallsyms])

用户态堆栈：

no address in memory maps

find vma failed

#~ 0x7f4a0573538d UNKNOWN ([symbol])

#\* 0xffffffffffffff X (UNKNOWN)

进程链信息：

#^ 0xffffffffffffff (UNKNOWN)

#^ 0xffffffffffffff (UNKNOWN)

#^ 0xffffffffffffff (UNKNOWN)

##

CPU 0，nr\_running:16

CPU 1，nr\_running:14

输出结果中包含引起调度延迟的调用链，以及每个CPU调度队列上的线程数量最大值。

每次输出结果后，历史数据将被清空。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools sched-delay --deactivate*

如果成功，将输出：

sched-delay is not activated

关闭功能后，本功能将不会对系统带来性能影响。

## irq-delay

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools irq-delay --help

结果如下：

irq-delay usage:

--help irq-delay help info

--activate

verbose VERBOSE

threshold threshold(ms)

--deactivate

--settings dump settings with text.

--report dump log with text.

--test testcase for irq-delay.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools irq-delay --activate

在激活本功能时，可用参数为：

verbose：该参数控制输出的详细程度，可以是任意整数。当前未用。

threshold：配置长时间关中断的阀值，超过该阀值将引起警告信息输出。时间单位是ms。

例如，如下命令会将检测阀值设置为80ms。一旦系统有超过80ms的关中断代码，将输出其调用链：

diagnose-tools irq-delay --activate="threshold=80"

如果成功，将在控制台输出如下：

功能设置成功，返回值：0

阀值(ms)： 80

输出级别： 0

如果失败，将在控制台输出如下：

功能设置失败，返回值：-16

阀值(ms)： 80

输出级别： 0

### 测试用例

执行如下命令触发测试用例：

sh /usr/diagnose-tools/test.sh irq-delay

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools irq-delay --settings

结果如下：

功能设置：

是否激活：√

阀值(ms)：80

输出级别：0

### 查看结果

系统会记录一段时间内中断被延迟的调用链。执行如下命令查看本功能的输出结果：

diagnose-tools irq-delay --report

输出结果示例如下：

*中断延迟，PID： 164390[diagnose-tools]， CPU：39, 96 ms, 时间：[1583993047:186455]*

*时间：[1583993047:186455].*

*进程信息： [/ / diagnose-tools]， PID： 164390 / 164390*

*##CGROUP:[/] 164390 [001] 采样命中*

*内核态堆栈：*

*#@ 0xffffffff81025022 save\_stack\_trace\_tsk ([kernel.kallsyms])*

每次输出结果后，历史数据将被清空。

### 生成火焰图

可以用如下命令获取结果并生成火焰图：

diagnose-tools irq-delay --report > irq-delay.log

diagnose-tools flame --input=irq-delay.log --output=irq-delay.svg

该命令将生成的火焰图保存到irq-delay.svg中。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools irq-delay --deactivate*

如果成功，控制台打印如下：

irq-delay is not activated

如果失败，控制台打印如下：

deactivate irq-delay fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

## irq-stats

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools irq-stats --help

结果如下：

irq-stats usage:

--help irq-stats help info

--activate

verbose VERBOSE

--deactivate

--report dump log with text.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools irq-stats --activate

在激活本功能时，有如下可供设置的参数：

verbose，该参数控制输出的详细程度，可以是任意整数。当值大于等于0时，会输出每个中断在每个核上面执行的次数/时间。

通过如下命令设置verbose参数为1,以打印详细的信息：

diagnose-tools irq-stats --activate="verbose=1"

如果激活成功，控制台将会输出如下：

功能设置成功，返回值：0

输出级别：1

如果失败，控制台将会输出如下：

功能设置失败，返回值：-16

输出级别：1

### 测试用例

执行如下命令运行本功能的测试用例：

sh /usr/diagnose-tools/test.sh irq-stats

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools irq-stats --settings

结果如下：

功能设置：

是否激活：×

输出级别：1

### 查看结果

系统会记录一段时间内中断/软中断执行次数/执行时间。执行如下命令查看本功能的输出结果：

diagnose-tools irq-stats --report

输出结果示例如下：

中断统计：[1580784677:664146]

core0 2558 125155482 21 971594

core1 106 8753619 21 1425487

IRQ: core0 irq: 1, handler: 0xffffffff815809b0, runtime(ns): 395 / 7867442

IRQ: core0 irq: 15, handler: 0xffffffffa006a750, runtime(ns): 72 / 1217346

IRQ: core0 irq: 19, handler: 0xffffffffa00a6480, runtime(ns): 92 / 1600734

IRQ: core0 irq: 21, handler: 0xffffffffa00cbef0, runtime(ns): 9 / 1192626

IRQ: core0 irq: 20, handler: 0xffffffffa03277f0, runtime(ns): 1990 / 113277334

IRQ: core1 irq: 21, handler: 0xffffffffa00cbef0, runtime(ns): 106 / 8753619

SOFT-IRQ: core0 soft-irq: 0, count: 0 / 0, runtime(ns): 0 / 0

SOFT-IRQ: core0 soft-irq: 1, count: 33396 / 22587824, runtime(ns): 39 / 70468

SOFT-IRQ: core0 soft-irq: 2, count: 0 / 0, runtime(ns): 0 / 0

SOFT-IRQ: core0 soft-irq: 3, count: 92 / 6564714, runtime(ns): 6 / 45302

每次输出结果后，历史数据将被清空。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools irq-stats --deactivate*

如果执行成功，控制台将打印如下：

irq-stats is not activated

如果执行失败，控制台将打印如下：

deactivate irq-stats fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

## irq-trace

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools irq-trace --help

结果如下：

irq-trace usage:

--help irq-trace help info

--activate

verbose VERBOSE

irq set irq threshold(ms)

sirq set soft-irq threshold(ms)

timer set timer threshold(ms)

--deactivate

--report dump log with text.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools irq-trace --activate

激活本功能时，可用参数为：

verbose：该参数控制输出的详细程度，可以是任意整数。当值大于等于0时，会在日志文件中输出每次中断/软中断/定时器的执行时刻、类型、函数名称。

irq：设置中断监控阈值(ms)

sirq：设置软中断监控阈值(ms)

timer：设置定时器监控阈值(ms)

如下命令监控超过1ms的IRQ，超过5ms的软中断／定时器：

diagnose-tools irq-trace --activate='irq=1 sirq=5 timer=5'

如果成功，将输出如下信息：

功能设置成功，返回值：0

输出级别：0

IRQ：1(ms)

SIRQ：5(ms)

TIMER：5(ms)

如果失败，将输出如下信息：

功能设置失败，返回值：-16

输出级别：0

IRQ：1(ms)

SIRQ：5(ms)

TIMER：5(ms)

### 测试用例

运行如下命令运行测试用例，以查看本功能是否正常：

sh /usr/diagnose-tools/test.sh irq-trace

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools irq-trace --settings

结果如下：

功能设置：

是否激活：×

输出级别：0

IRQ：1(ms)

SIRQ：5(ms)

TIMER：5(ms)

### 查看结果

执行如下命令查看本功能的输出结果：

﻿﻿diagnose-tools irq-trace --report

如果系统中有长时间执行的中断/软中断/定时器，工具将输出相应函数的名称，以及执行时长，异常时间点。

每次输出结果后，历史数据将被清空。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools irq-trace --deactivate*

如果成功，将输出如下：

irq-trace is not activated

如果失败，将输出如下：

deactivate irq-trace fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

## load-monitor

load-monitor功能实时抓取系统Load值，一旦Load值超过设置的阀值，就输出系统中所有处于Running/Uninterruptale状态的线程调用链。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools load-monitor --help

结果如下：

load-monitor usage:

--help load-monitor help info

--activate

verbose VERBOSE

style dump style: 0 - common, 1 - process chains

load threshold for load(ms)

load.r threshold for load.r(ms)

load.d threshold for load.d(ms)

task.d threshold for task.d(ms)

--settings print settings.

--deactivate

--report dump log with text.

--sls save detail into sls files.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools load-monitor --activate

激活本功能时，可用的参数为：

verbose 设置输出级别，目前未用。

style如果为1，输出进程链。其他值不输出。--load设置监控阀值，一旦Load值超过此值，就触发Load报警输出。默认值是0，表示不监控此值。

load.r 设置监控阀值，一旦Load.R值超过此值，就触发Load.R报警输出。默认值是0，表示不监控此值。

load.d 设置监控阀值，一旦Load.D值超过此值，就触发Load.D报警输出。默认值是0，表示不监控此值。

task.d 设置监控阀值，一旦Task.D值超过此值，就触发Task.D报警输出。默认值是0，表示不监控此值。

Load/Load.R/Load.D/Task.d分别代表几个被监控的负载值。

* Load: 系统load值
* Load.R: 正在运行的任务引起的Load
* Load.D: D状态任务引起的Load
* Task.D: 当前处于D状态的任务数量

一般情况下，仅仅需要监控Load指标即可。如：

diagnose-tools load-monitor --activate="load=50"

该命令会将Load监控值设置为50,一旦系统Load超过50就输出调用链。

如果成功，将会在控制台输出：

功能设置成功，返回值：0

Load： 50

Load.R： 0

Load.D： 0

Task.D： 0

输出级别： 0

STYLE： 0

如果失败，将会在控制台输出：

功能设置失败，返回值：-16

Load： 50

Load.R： 0

Load.D： 0

Task.D： 0

输出级别： 0

STYLE： 0

### 测试用例

运行如下命令运行测试用例，以查看本功能是否正常：

sh /usr/diagnose-tools/test.sh load-monitor

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools load-monitor --settings

结果如下：

功能设置：

是否激活： √

Load： 50

Load.R： 0

Load.D： 0

Task.D： 0

输出级别： 0

STYLE： 0

### 查看结果

激活本功能后，一旦系统负载超过阀值，就会记录下日志。执行如下命令查看本功能的输出结果：

diagnose-tools load-monitor --report

输出结果示例如下：

*Load飙高：[1583992747:970548]*

*Load: 6.10, 6.14, 6.14*

*Load.R: 6.08, 6.12, 6.09*

*Load.D: 0.01, 0.02, 0.04*

*##CGROUP:[/] 156654 [022] 采样命中[R]*

*内核态堆栈：*

*#@ 0xffffffff81025022 save\_stack\_trace\_tsk ([kernel.kallsyms])*

*#@ 0xffffffffa0fc0419 diagnose\_save\_stack\_trace [diagnose] ([kernel.kallsyms])*

*#@ 0xffffffffa0fc0d8e ali\_diag\_task\_kern\_stack [diagnose] ([kernel.kallsyms])*

*#@ 0xffffffffa0fc70a0 ali\_diagnose\_load\_timer [diagnose] ([kernel.kallsyms])*

*#@ 0xffffffffa0fc5b41 hrtimer\_handler [diagnose] ([kernel.kallsyms])*

*#@ 0xffffffff810aa0d2 \_\_hrtimer\_run\_queues ([kernel.kallsyms])*

*#@ 0xffffffff810aa670 hrtimer\_interrupt ([kernel.kallsyms])*

*#@ 0xffffffff8104cbf7 local\_apic\_timer\_interrupt ([kernel.kallsyms])*

*#@ 0xffffffff81664cdf smp\_apic\_timer\_interrupt ([kernel.kallsyms])*

*#@ 0xffffffff81660432 apic\_timer\_interrupt ([kernel.kallsyms])*

*#\* 0xffffffffffffff yes (UNKNOWN)*

*进程链信息：*

*#^ 0xffffffffffffff yes (UNKNOWN)*

*#^ 0xffffffffffffff -bash (UNKNOWN)*

*#^ 0xffffffffffffff sshd: root@pts/6 (UNKNOWN)*

*#^ 0xffffffffffffff /usr/sbin/sshd -D (UNKNOWN)*

*#^ 0xffffffffffffff /usr/lib/systemd/systemd --switched-root --system --deserialize 21 (UNKNOWN)*

*##*

内核开发同学根据这些调用链，就能知道引起系统Load高的原因。

### 生成火焰图

可以用如下命令获取结果并生成火焰图：

diagnose-tools load-monitor --report > load-monitor.log

diagnose-tools flame --input=load-monitor.log --output=load-monitor.svg

该命令将生成的火焰图保存到load-monitor.svg中。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools load-monitor --deactivate*

如果执行成功，控制台将输出如下内容：

load-monitor is not activated

如果执行失败，控制台将输出如下内容：

deactivate load-monitor fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。不过本功能对系统性能的影响很小。

## run-trace

本功能监控统计多个进程或者线程的运行状况，以及用户态/内核态热点调用链。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools run-trace --help

结果如下：

run-trace usage:

--help run-trace help info

--activate

verbose VERBOSE

threshold default THRESHOLD(MS), you may set special value in code

threshold-us default THRESHOLD(US)

buf-size-k set buf size(k) for per-thread

timer-us perf timer(us)

--deactivate

--settings print settings.

--report dump log with text.

--test testcase for run-trace.

--set-syscall PID SYSCALL THRESHOLD monitor special syscall

--clear-syscall PID do not monitor syscall

--uprobe set uprobe to start/stop trace.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools run-trace --activate

在激活本功能时，可用参数为：

verbose VERBOSE 设置输出信息的级别，目前未用。

threshold 该参数设置监控阀值，单位是ms。默认值是500。当用户态应用在一段代码中运行超过阀值，就会详细的输出这段代码超时的详细信息。应用程序可以传递参数指定自己想要监控的阀值，将忽略此设置。

threshold-us 该参数设置监控阀值，单位是us。此参数优先级高于threshold参数。

timer-us 指定采样周期，单位是us。如果指定此参数，将定期采集当前线程的行为。

buf-size-k 设置每个线程的监控缓冲区大小，默认为200K，最大可以上调到10M。单位为K。

例如，如下命令将设置采样周期为10us，以及输出级别为1：

diagnose-tools run-trace --activate='timer-us=10 verbose=1'

如果成功，将输出：

功能设置成功，返回值：0

阀值(us)：500000

输出级别：1

TIMER\_US：10

BUF-SIZE-K：0

如果失败，将输出：

功能设置失败，返回值：-16

阀值(us)：500000

输出级别：1

TIMER\_US：10

BUF-SIZE-K：0

### 设置参数

--set-syscall 设置将对哪个进程监控哪个系统调用，以及其监控阀值。在不同的环境中，同一个系统调用的编号并不相同。可以使用/usr/diagnose-tools/get\_sys\_call.sh脚本来获得某个系统调用号。例如下面的命令将获得open系统调用的编号：

sh /usr/diagnose-tools/get\_sys\_call.sh open

--clear-syscall 清除要监控的进程，不再对其系统调用进行监控。

--uprobe 设置用户态探针位置。例如：

--uprobe="tgid=`pgrep run-trace.out | head -1` start-file=/usr/diagnose-tools/bin/run-trace.out start-offset=1875 stop-file=/usr/diagnose-tools/bin/run-trace.out stop-offset=1885"

该命令会监控run-trace.out文件，并在其偏移1875地方设置探针，开始对RT进行计时，并在第1885的地方再次设置探针，结束对RT进行计时。

这样，就不需要修改应用程序就可以监控其run-trace结果了。

例如，如下命令将监控阀值编号为1234的进程的第35号系统调用，其监控阀值为900ms：

diagnose-tools run-trace --set-syscall="1234 35 900"

如果成功，将输出：

set-syscall for run-trace: pid 1234, syscall 35, threshold 900ms, ret is 0

如果失败，将输出：

set-syscall for run-trace: pid 1234, syscall 35, threshold 900ms, ret is -1

如下命令将清除编号为1234的进程监控，不再对其所有系统调用进行监控：

diagnose-tools run-trace --clear-syscall="1234"

如果成功，将输出：

clear-syscall for run-trace: pid 1234, ret is 0

如果失败，将输出：

clear-syscall for run-trace: pid 1234, ret is -1

### 测试用例

运行如下命令运行测试用例，以查看本功能是否正常：

sh /usr/diagnose-tools/test.sh run-trace

### 应用改造

一般情况下，需要修改应用程序，也不需要重启应用程序。

有3种方式对应用程序的RT进行监控。

其中一种方式是监控应用的某个系统调用RT高。这种情况不需要对应用进程改造。

另一种方式是监控应用程序某一段代码的RT超时，需要应用程序在计算RT开始和结束的地方，按照diagnose-tools工具的要求来修改应用程序。

C代码示例如下：

﻿ for (i = 0; i < count; i++) {

syscall(ALI\_DIAG\_RUN\_TRACE\_START, 100);

sleep(1);

sleep(1);

syscall(ALI\_DIAG\_RUN\_TRACE\_STOP);

}

其中syscall(ALI\_DIAG\_RUN\_TRACE\_START, 100)启动run-trace监控功能，这样run-trace就会开始对当前程序进行监控，如果在syscall(ALI\_DIAG\_RUN\_TRACE\_STOP)之前，程序运行时间超过100ms，将记录下系统日志。

syscall(ALI\_DIAG\_RUN\_TRACE\_STOP)告诉run-trace结束监控。如果在启动/结束之间的运行时间超过100ms，就会输出警告信息。

你也可以写一段java代码来告诉run-trace启动/结束监控：

﻿static class ali\_diagnose\_settings

{

static void start\_run\_trace()

{

FileOutputStream out = null;

try {

File file = new File("/proc/ali-linux/diagnose/kern/run-trace-settings");

if (file.exists()) {

out = new FileOutputStream("/proc/ali-linux/diagnose/kern/run-trace-settings");

out.write("start 100\n\0".getBytes());

out.write(0);

out.write(System.getProperty("line.separator").getBytes());

}

} catch (IOException e) {

e.printStackTrace();

} finally {

if (out != null) {

try {

out.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

static void stop\_run\_trace()

{

FileOutputStream out = null;

try {

File file = new File("/proc/ali-linux/diagnose/kern/run-trace-settings");

if (file.exists()) {

out = new FileOutputStream("/proc/ali-linux/diagnose/kern/run-trace-settings");

out.write("stop\n".getBytes());

out.write(0);

out.write(System.getProperty("line.separator").getBytes());

}

} catch (IOException e) {

e.printStackTrace();

} finally {

if (out != null) {

try {

out.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

}

当然了，在Java中，也可以通过JNI的方式来调用syscall，来告诉run-trace启动/结束监控。

第三种方法是利用btrace或者uprobe功能，直接在应用程序中挂接钩子，这样就不需要对应用程序进行修改了。

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools run-trace --settings

结果如下：

功能设置：

是否激活：×

阀值(ms)：500

输出级别：0

线程监控项：0

系统调用监控项：0

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools run-trace --report

该命令会以文件的方式输出监控结果。一般情况下，业务同学不应当使用此命令。应当使用网页来查看监控结果。

每次输出结果后，历史数据将被清空。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools* run-trace --*deactivate*

如果成功，将输出：

run-trace is not activated

如果失败，将输出：

deactivate run-trace fail, ret is -1

## perf

本功能每10ms对系统内的进程进行采样。采集进程名称/所在容器信息/内核态堆栈/用户态堆栈。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools perf --help

结果如下：

perf usage:

--help perf help info

--activate

style dump style: 0 - common, 1 - process chains

verbose VERBOSE

tgid process group that monitored

pid thread id that monitored

comm comm that monitored

cpu cpu-list that monitored

idle set 1 if want monitor idle

bvt set 1 if want monitor idle

sys set 1 if want monitor syscall only

--deactivate

--report dump log with text.

--test testcase for perf.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools perf --activate

在激活功能时，可用参数为：

verbose 该参数设置输出级别，暂时未用。

style如果为1，输出进程链。其他值不输出。

﻿ tgid 要采样的进程PID

pid 要采样的线程TID

comm 要采样的进程名称

cpu 要采样的CPU列表，如0-16,23这样的格式

bvt 是否采集离线任务，默认不采集。

idle 是否采集IDLE任务，默认不采集。

sys 如果为1,表示只采集SYS，忽略用户态。用于专查SYS高。

例如，如下命令开始对gnome-shell进程进行采样：

diagnose-tools perf --activate="tgid=`pidof gnome-shell`"

如果成功，将输出：

功能设置成功，返回值：0

STYLE： 0

输出级别： 0

进程ID： 11570

线程ID： 0

进程名称：

CPUS：

IDLE： 0

BVT： 0

SYS： 0

如果失败，将输出：

功能设置失败，返回值：-16

STYLE： 0

输出级别： 0

进程ID： 11570

线程ID： 0

进程名称：

CPUS：

IDLE： 0

BVT： 0

SYS： 0

### 测试用例

运行如下命令运行测试用例，以查看本功能是否正常：

sh /usr/diagnose-tools/test.sh perf

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools perf --settings

结果如下：

功能设置：

是否激活： √

进程ID： 0

线程ID： 0

进程名称：

CPUS： 0-1

IDLE： 1

BVT： 1

SYS： 0

STYLE： 0

输出级别： 0

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools perf --report

在输出时，可以指定container参数，这样就可以在容器中使用perf工具了。如：

diagnose-tools perf --report="container=1"

在输出时，还可以指定reverse参数，一旦此参数为1，就会将输出结果翻转。也就是说，内核态调用链在火焰图的最底端 。用于配合sys参数查系统SYS高。如：

diagnose-tools perf --report="reverse=1"

输出结果中，包含perf命中的线程PID/名称，线程所在CGROUP组，内核态堆栈，用户态堆栈，进程链等信息。可以使用这些结果生成火焰图。

每次输出结果后，历史数据将被清空。

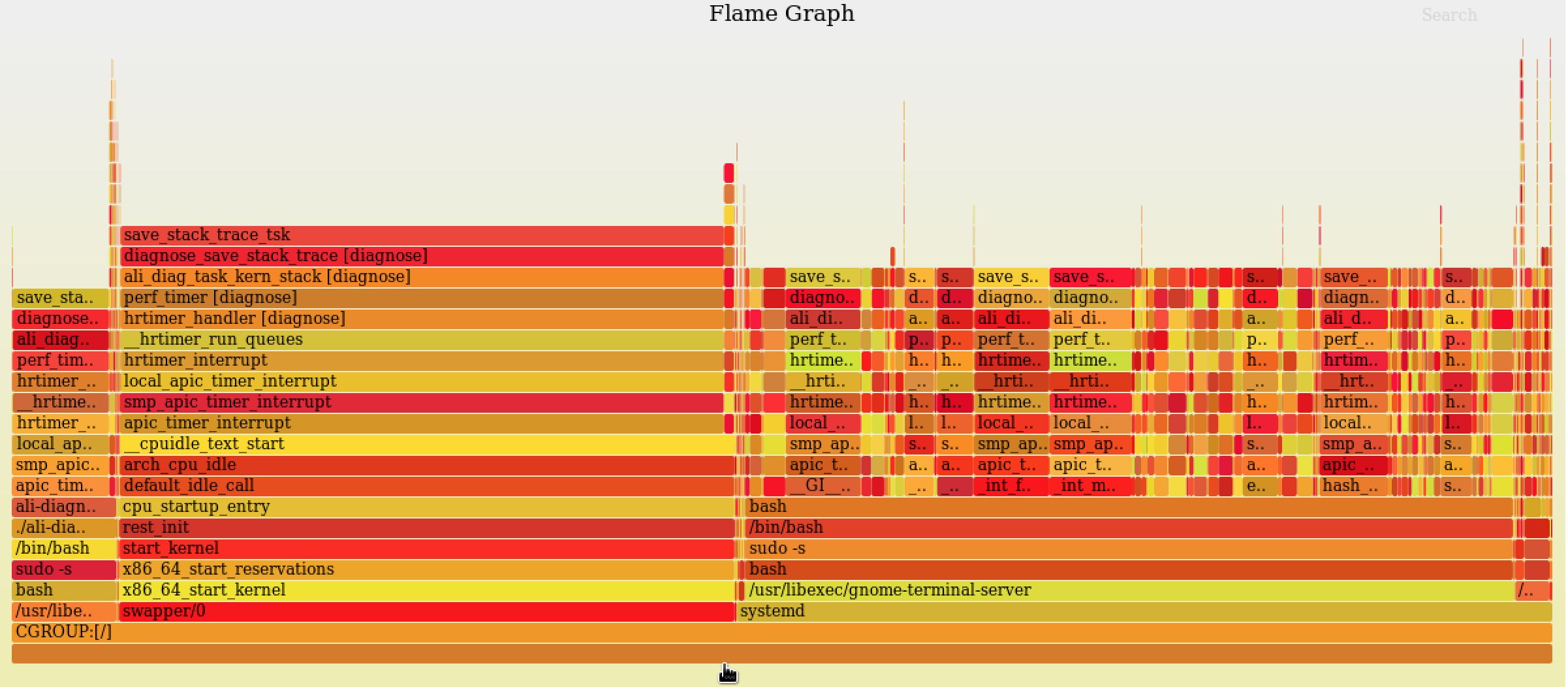
### 输出火焰图

diagnose-tools perf --report > perf.log

diagnose-tools flame --input=perf.log --output=perf.svg

该命令将生成的火焰图保存到perf.svg中。

使用浏览器打开perf.svg，如下所示：



## kprobe

本功能监控任意内核函数的执行情况，并生成火焰图。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools kprobe --help

结果如下：

kprobe usage:

--help kprobe help info

--activate

verbose VERBOSE

tgid process group that monitored

pid thread id that monitored

comm comm that monitored

cpu cpu-list that monitored

probe function that monitored

dump-style dump style for kprobe. dump to dmesg if it is 1.

--deactivate

--report dump log with text.

--settings dump settings.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools kprobe --activate

在激活本功能时，可用参数为：

verbose VERBOSE该参数设置输出级别，暂时未用

tgid 要采样的进程PID

pid 要采样的线程TID

comm 要采样的进程名称

cpu cpus 要采样的CPU列表，如0-16,23这样的格式d

probe 要监控的函数名称

dump-style 输出格式，如果为1,表示输出到dmesg中

例如，如下命令表示监控hrtimer\_interrupt函数：

diagnose-tools kprobe --activate='probe=hrtimer\_interrupt'

如果成功，将输出：

功能设置成功，返回值：0

进程ID：0

线程ID：0

进程名称：

函数名称：hrtimer\_interrupt

CPUS：

输出级别：0

如果失败，将输出：

功能设置失败，返回值：-16

进程ID：0

线程ID：0

进程名称：

函数名称：hrtimer\_interrupt

CPUS：

输出级别：0

### 测试用例

运行如下命令，将启动本功能的测试用例：

sh /usr/diagnose-tools/test.sh kprobe

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools kprobe --settings

结果如下：

功能设置：

是否激活：×

进程ID：0

线程ID：0

进程名称：

函数名称：hrtimer\_interrupt

CPUS：0-1

输出级别：0

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools kprobe --report

输出结果示例如下：

*KPROBE命中：PID： 106526[h2o]，时间：[1584004225:651374]*

*##CGROUP:[h2o] 106526 [4564] KPROBE命中，时间：[1584004225:651374]*

*内核态堆栈：*

*#@ 0xffffffff81025022 save\_stack\_trace\_tsk ([kernel.kallsyms])*

*#@ 0xffffffffa141f419 diagnose\_save\_stack\_trace [diagnose] ([kernel.kallsyms])*

*#@ 0xffffffffa141fd8e ali\_diag\_task\_kern\_stack [diagnose] ([kernel.kallsyms])*

*#@ 0xffffffffa142eaab kprobe\_pre [diagnose] ([kernel.kallsyms])*

*#@ 0xffffffff81659e1c kprobe\_ftrace\_handler ([kernel.kallsyms])*

*#@ 0xffffffff8113559e ftrace\_ops\_list\_func ([kernel.kallsyms])*

*#@ 0xffffffff81663d44 ftrace\_regs\_call ([kernel.kallsyms])*

*#@ 0xffffffff81664cdf smp\_apic\_timer\_interrupt ([kernel.kallsyms])*

*#@ 0xffffffff81660432 apic\_timer\_interrupt ([kernel.kallsyms])*

*用户态堆栈：*

*#~ 0xcbf32d \_ZNKSt7\_\_cxx1112basic\_stringIcSt11char\_traitsIcESaIcEE4findEPKcmm ([symbol])*

*#\* 0xffffffffffffff h2o (UNKNOWN)*

*进程链信息：*

*#^ 0xffffffffffffff h2o (UNKNOWN)*

*#^ 0xffffffffffffff auditd (UNKNOWN)*

*#^ 0xffffffffffffff systemd (UNKNOWN)*

*##*

输出结果中，包含kprobe命中的线程PID/名称，线程所在CGROUP组，内核态堆栈，用户态堆栈，进程链等信息。

每次输出结果后，历史数据将被清空。

### 输出火焰图

执行如下命令生成火焰图：

diagnose-tools kprobe --report > kprobe.log

diagnose-tools flame --input=kprobe.log --output=kprobe.svg

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools kprobe --deactivate*

如果成功，将输出：

kprobe is not activated

如果失败，将输出：

deactivate kprobe fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

## uprobe

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools uprobe --help

结果如下：

uprobe usage:

--help uprobe help info

--activate launch file and offset

verbose VERBOSE

tgid process group that monitored

pid thread id that monitored

comm comm that monitored

cpu cpu-list that monitored

--deactivate

--settings dump settings

--report dump log with text.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools uprobe --activate

在激活本功能时，需要指定激活参数：

* file参数指定在哪个文件中设置探针
* offset参数指定在文件中什么位置设置探针

命令示例：

diagnose-tools uprobe --activate='verbose=1 file=/usr/diagnose-tools/bin/uprobe.out offset=1875'

同时，可以指定其他一些参数：

本命令可用参数为：

tgid 要探测的进程ID

pid 要探测的线程ID

comm 要探测的进程名称

cpu cpus 要探测的CPU列表，如0-16,23这样的格式

例如，如下命令限制仅对进程1234进行探测：

diagnose-tools uprobe --activate='verbose=1 file=/usr/diagnose-tools/bin/uprobe.out offset=1875,tgid=1234'

如果成功，将输出：

功能设置成功，返回值：0

进程ID：1234

线程ID：0

进程名称：

CPUS：

输出级别：1

文件名：

偏移：1875

如果失败，将输出：

功能设置失败，返回值：-16

进程ID：1234

线程ID：0

进程名称：

CPUS：

输出级别：1

文件名：

偏移：1875

### 测试用例

运行如下命令测试本功能：

sh /usr/diagnose-tools/test.sh uprobe

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools uprobe --settings

结果如下：

功能设置：

是否激活：√

进程ID：1234

线程ID：0

进程名称：

CPUS：0-1

输出级别：1

文件名：/usr/diagnose-tools/bin/uprobe.out

偏移：1875

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools uprobe --report

输出结果示例如下：

*UPROBE命中：PID： 9215[run-trace.out]，时间：[1587959184:307057]*

*##CGROUP:[/] 9215 [001] UPROBE命中，时间：[1587959184:307057]*

*用户态堆栈：*

*#~ 0x400773 mytest ([symbol])*

*#~ 0x7fdc46516495 \_\_libc\_start\_main ([symbol])*

*#\* 0xffffffffffffff run-trace.out (UNKNOWN)*

*进程链信息：*

*#^ 0xffffffffffffff /usr/diagnose-tools/bin/run-trace.out (UNKNOWN)*

*#^ 0xffffffffffffff sh test.sh uprobe (UNKNOWN)*

*#^ 0xffffffffffffff /bin/bash (UNKNOWN)*

*#^ 0xffffffffffffff sudo -s (UNKNOWN)*

*#^ 0xffffffffffffff bash (UNKNOWN)*

*#^ 0xffffffffffffff /usr/libexec/gnome-terminal-server (UNKNOWN)*

*#^ 0xffffffffffffff /usr/lib/systemd/systemd --switched-root --system --deserialize 22 (UNKNOWN)*

*##*

输出结果中，包含uprobe命中的线程PID/名称，线程所在CGROUP组，内核态堆栈，用户态堆栈，进程链等信息。

每次输出结果后，历史数据将被清空。

### 输出火焰图

可以将输出结果转储到文件中，然后使用diagnose-tools的flame命令生成火焰图。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools uprobe --deactivate*

如果成功，将输出：

uprobe is not activated

如果失败，将输出：

deactivate uprobe fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

## utilization

本功能监控系统资源利用率，找到CPU被哪些野进程干扰，以及进程对内存的使用情况。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools utilization --help

结果如下：

utilization usage:

--activate

verbose VERBOSE

style dump style: 0 - common, 1 - process chains

cpu cpu-list that monitored

--deactivate

--settings print settings.

--report dump log with text.

--isolate CPU CGROUP set isolated cgroup name for cpu.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools utilization --activate

在激活本功能时，可用的参数有：

cpu 被监控的cpu列表；

verbose 该参数设置输出级别，暂时未用。

style如果为1，输出进程链。其他值不输出。

例如，如下命令将设置输出类型为1,即输出进程链：

diagnose-tools utilization --activate='style=1'

如果成功，将输出：

功能设置成功，返回值：0

STYLE： 1

输出级别： 0

CPUS：

如果失败，将输出：

功能设置失败，返回值：-16

STYLE： 1

输出级别： 0

CPUS：

### 设置参数

本功能可用参数为：

--isolate CPU CGROUP 设置某个CPU上独享CPU的CGROUP组，这些CGROUP组的进程不作为野进程对待。

如下命令设置CPU 1的独占CGROUP名称为tdc：

diagnose-tools utilization --isolate="1 tdc"

如果成功，将输出：

set isolate for utilization: 1, tdc, ret is 0

如果失败，将输出：

set isolate for utilization: 1, tdc, ret is -1

### 测试用例

运行如下命令启动本功能的测试用例：

sh /usr/diagnose-tools/test.sh utilization

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools utilization --settings

结果如下：

功能设置：

是否激活： ×

输出级别： 0

STYLE： 0

CPUS： 0-1

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools utilization --report

输出结果中包含野进程在CPU上运行的时间，分配的内存数量。

每次输出结果后，历史数据将被清空。

### 输出火焰图

可以生成三种火焰图：

1. CPU执行时间火焰图
2. 内存分配火焰图
3. 野进程干扰火焰图

以下三个命令分别用于生成这三个图：

1、CPU执行时间火焰图

diagnose-tools utilization --report | awk '{if (substr($1,1,2) == "\*\*") {print substr($0, 3)}}' | /usr/diagnose-tools/flame-graph/flamegraph.pl > utilization.svg

2、内存分配火焰图

diagnose-tools utilization --report | awk '{if (substr($1,1,2) == "\*#") {print substr($0, 3)}}' | /usr/diagnose-tools/flame-graph/flamegraph.pl > utilization.svg

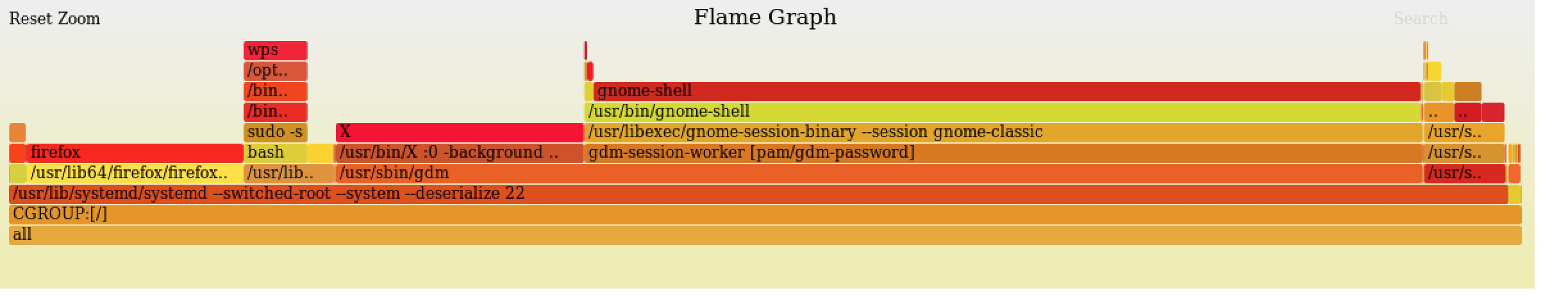
3、野进程干扰火焰图

diagnose-tools utilization --report | awk '{if (substr($1,1,2) == "\*^") {print substr($0, 3)}}' | /usr/diagnose-tools/flame-graph/flamegraph.pl > utilization.svg

该命令将生成的火焰图保存到utilization.svg中。该命令的含义是：

首先调用report命令输出内核中保存的结果，然后使用awk命令将”\*”开头的行提取出来，最后的一个pl脚本将堆栈符号生成火焰图。

使用浏览器打开utilization.svg，如下所示：



### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools* utilization *--deactivate*

如果成功，将输出：

utilization is not activated

如果失败，将输出：

deactivate utilization fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

## exit-monitor

有时候，进程莫名其妙的退出了。很难找到是被kill还是程序自身异常引起。特别是某些三方库会调用abort/exit直接退出系统。

exit-monitor可以监控特定进程退出时的调用链。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools exit-monitor --help

结果如下：

exit-monitor usage:

--help exit-monitor help info

--activate

verbose VERBOSE

tgid process group that monitored

comm comm that monitored

--deactivate

--report dump log with text.

--test testcase for exit-monitor.

--log

sls=/tmp/1.log store in file

syslog=1 store in syslog

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools exit-monitor --activate

在激活本功能时，可用参数有：

本功能可用参数为：

tgid 设置要监控的进程pid

comm 设置要监控的进程名称

verbose 设置输出级别，目前未用。

例如：

diagnose-tools exit-monitor --activate="comm=sleep"

如果设置成功，该命令在控制台中会有如下输出：

功能设置成功，返回值：0

进程ID：0

进程名称：sleep

输出级别：0

如果设置失败，该命令在控制台中会有如下输出：

功能设置失败，返回值：-16

进程ID：0

进程名称：sleep

输出级别：0

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools exit-monitor --settings

结果如下：

功能设置：

是否激活：√

线程ID：0

进程名称：sleep

输出级别：0

### 测试用例

执行如下命令触发测试用例：

﻿sh /usr/diagnose-tools/test.sh exit-monitor

### 查看结果

激活本功能后，一旦被监控的进程退出，就会记录下日志。执行如下命令查看本功能的输出结果：

diagnose-tools exit-monitor --report

输出结果示例如下：

线程退出，PID： 13796[sleep]，退出时间：[1580733810:211866]

0xffffffff8103ddff, save\_stack\_trace\_tsk

0xffffffffa0656409, diagnose\_save\_stack\_trace [diagnose]

0xffffffffa0658631, ali\_diag\_task\_kern\_stack [diagnose]

0xffffffffa065d444, kprobe\_do\_exit\_pre [diagnose]

0xffffffff810657b3, kprobe\_ftrace\_handler

0xffffffff8115ecb1, ftrace\_ops\_assist\_func

0xffffffffa06470d5, cleanup\_module [isofs]

0xffffffff8108f9a5, do\_exit

0xffffffff81090583, do\_group\_exit

0xffffffff81090604, SyS\_exit\_group

0xffffffff81003c04, do\_syscall\_64

0xffffffff81741c8e, entry\_SYSCALL\_64\_after\_swapgs

用户态堆栈：

0x00007fcc5f832359,no address in memory maps

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools exit-monitor --deactivate*

如果执行成功，控制台将会有如下输出：

exit-monitor is not activated

如果执行失败，控制台将会有如下输出：

deactivate exit-monitor fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。不过本功能对系统性能的影响很小。

## mutex-monitor

本功能监控内核中长时间持有mutex的情况。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools mutex-monitor --help

结果如下：

mutex-monitor usage:

--help mutex-monitor help info

--activate

verbose VERBOSE

threshold threshold(ms)

style dump style: 0 - common, 1 - process chains

--deactivate

--settings dump settings with text.

--report dump log with text.

--test testcase for mutex-monitor.

--log

sls=/tmp/1.log store in file

syslog=1 store in syslog.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools mutex-monitor --activate

激活本功能时，可用参数为：

verbose 本参数目前未用。

style如果为1，输出进程链。

threshold 该参数设置监控阀值，单位是ms。默认值是1000。当某个函数持有mutex超过1000 ms时，就会打印这个函数的调用链。

例如，如下命令将监控阀值设置为900ms：

diagnose-tools mutex-monitor --activate="threshold=900"

如果成功，将输出如下：

功能设置成功，返回值：0

阀值(ms)： 900

输出级别： 0

STYLE： 0

如果失败，将输出如下：

功能设置失败，返回值：-16

阀值(ms)： 900

输出级别： 0

STYLE： 0

### 测试用例

运行如下命令运行测试用例，以查看本功能是否正常：

sh /usr/diagnose-tools/test.sh mutex-monitor

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools mutex-monitor --settings

结果如下：

功能设置：

是否激活： ×

阀值(ms)： 0

输出级别： 0

STYLE： 1

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools mutex-monitor --report

预期结果如下：

*MUTEX延迟： 0xffffffffa0fd80c0，PID： 183817[diagnose-tools]， 1503 ms, 时间：[1583993176:208214]*

*时间：[1583993176:208214].*

*进程信息： [/ / diagnose-tools]， PID： 183817 / 183817*

*内核态堆栈：*

*#@ 0xffffffff81025022 save\_stack\_trace\_tsk ([kernel.kallsyms])*

......

结果中包含延迟时间/造成延迟的锁名称/造成延迟的调用链。

每次输出结果后，历史数据将被清空。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools mutex-monitor --deactivate*

如果成功，将输出：

mutex-monitor is not activated

如果失败，将输出：

deactivate mutex-monitor fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

## exec-monitor

本功能监控进程创建过程。对于那些引起系统抖动的小脚本，例如ps -eL命令，能抓到调用这些命令的进程组。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools exec-monitor --help

结果如下：

exec-monitor usage:

--help exec-monitor help info

--activate

verbose VERBOSE

--deactivate

--report dump log with text.

--log

sls=/tmp/1.log store in file

syslog=1 store in syslog.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools exec-monitor --activate

激活本功能时，本命令可用参数为：

verbose 该参数设置输出级别，暂时未用。

如下命令，将设置verbose参数为1：

diagnose-tools exec-monitor --activate="verbose=1"

如果成功，将输出：

功能设置成功，返回值：0

输出级别：1

如果失败，将输出：

功能设置失败，返回值：-16

输出级别：1

### 测试用例

运行如下命令启动测试用例：

sh /usr/diagnose-tools/test.sh exec-monitor

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools exec-monitor --settings

结果如下：

功能设置：

是否激活：×

输出级别：0

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools exec-monitor --report

输出结果示例如下：

创建进程： [./diagnose-tools]，CGROUP：[/], 当前进程：10493[diagnose-tools], tgid： 10493，当前时间：[1580804064:266908]

进程链信息：

./diagnose-tools exec-monitor --report

/bin/bash

sudo -s

bash

/usr/libexec/gnome-terminal-server

输出结果中，包含了被创建进程的名称/启动参数，所在CGROUP组，父进程/祖父进程的名称。

每次输出结果后，历史数据将被清空。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools exec-monitor --deactivate*

如果成功，将输出：

exec-monitor is not activated

如果失败，将输出：

deactivate exec-monitor fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

## alloc-top

本功能统计一段时间内，进程分配的内存数量（不统计释放数量），并按照分配数量按序输出。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools alloc-top --help

结果如下：

alloc-top usage:

--help alloc-top help info

--activate

verbose VERBOSE

top max count in top list

--deactivate

--report dump log with text.

--settins dump settings with text.

--test testcase for alloc-top.

--log

sls=/tmp/1.log store in file

syslog=1 store in syslog.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools alloc-top --activate

在激活本功能时，可用参数为：

verbose 输出级别，目前未用。

top 设置输出结果的行数。

例如，下面的命令将输出行数限制为20行：

diagnose-tools alloc-top --activate='top=20'

如果成功，将输出：

功能设置成功，返回值：0

TOP-N：20

输出级别：0

如果失败，将输出：

功能设置失败，返回值：-16

TOP-N：20

输出级别：0

### 测试用例

使用如下命令将启动本功能的测试用例：

sh /usr/diagnose-tools/test.sh alloc-top

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools alloc-top --settings

结果如下：

功能设置：

是否激活：×

TOP-N：20

输出级别：0

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools alloc-top --report

结果示例如下：

序号 TGID COMM PG-COUNT CGROUP

1 3883 X 2443251 /

序号 TGID COMM PG-COUNT CGROUP

1 4959 gnome-shell 201975 /

这几列数据分别代表：序号/进程号/进程名称/分配页面数量/进程所在CGROUP名称。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools alloc-top --deactivate*

如果成功，将输出：

alloc-top is not activated

如果失败，将输出：

deactivate alloc-top fail, ret is -1

关闭功能后，本功能将不会对系统带来任何影响。

## high-order

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools high-order --help

结果如下：

high-order usage:

--help high-order help info

--activate

verbose VERBOSE

order threshold value

--deactivate

--settins dump settings with text.

--report dump log with text.

--test testcase for high-order.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools alloc-top --activate

在激活本功能时，可用参数为：

verbose 输出级别，目前未用。

order只有当分配的内存阶数高于此值才输出。

例如，下面的命令将分配阶数设置为2：

diagnose-tools high-order --activate='order=2'

如果成功，将输出：

功能设置成功，返回值：0

ORDER：2

输出级别：0

如果失败，将输出：

功能设置失败，返回值：-16

ORDER：2

输出级别：0

### 测试用例

使用如下命令将启动本功能的测试用例：

sh /usr/diagnose-tools/test.sh high-order

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools high-order --settings

结果如下：

功能设置：

是否激活：×

ORDER：2

输出级别：0

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools high-order --report

结果示例如下：

##CGROUP:[/] 5180 [48191] 采样命中[3]

内核态堆栈：

#@ 0xffffffff8103ddef save\_stack\_trace\_tsk ([kernel.kallsyms])

#@ 0xffffffffa1b49289 diagnose\_save\_stack\_trace [diagnose] ([kernel.kallsyms])

#@ 0xffffffffa1b49bae ali\_diag\_task\_kern\_stack [diagnose] ([kernel.kallsyms])

#@ 0xffffffffa1b5d232 trace\_mm\_page\_alloc\_hit [diagnose] ([kernel.kallsyms])

#@ 0xffffffff811d20ac \_\_alloc\_pages\_nodemask ([kernel.kallsyms])

#@ 0xffffffff81229ba5 alloc\_pages\_current ([kernel.kallsyms])

#@ 0xffffffff81629889 alloc\_skb\_with\_frags ([kernel.kallsyms])

#@ 0xffffffff816249d0 sock\_alloc\_send\_pskb ([kernel.kallsyms])

#@ 0xffffffff816fdf17 unix\_stream\_sendmsg ([kernel.kallsyms])

#@ 0xffffffff8161ea48 sock\_sendmsg ([kernel.kallsyms])

#@ 0xffffffff8161eae5 sock\_write\_iter ([kernel.kallsyms])

#@ 0xffffffff81262f89 do\_iter\_readv\_writev ([kernel.kallsyms])

#@ 0xffffffff8126499e do\_readv\_writev ([kernel.kallsyms])

#@ 0xffffffff81264c8c vfs\_writev ([kernel.kallsyms])

#@ 0xffffffff81264d01 do\_writev ([kernel.kallsyms])

#@ 0xffffffff81265ee0 SyS\_writev ([kernel.kallsyms])

#@ 0xffffffff81003c04 do\_syscall\_64 ([kernel.kallsyms])

#@ 0xffffffff817691ce entry\_SYSCALL\_64\_after\_swapgs ([kernel.kallsyms])

用户态堆栈：

#~ 0x7efef7266b80 \_\_writev ([symbol])

#\* 0xffffffffffffff gnome-shell (UNKNOWN)

进程链信息：

#^ 0xffffffffffffff /usr/bin/gnome-shell (UNKNOWN)

#^ 0xffffffffffffff /usr/libexec/gnome-session-binary --session gnome-classic (UNKNOWN)

#^ 0xffffffffffffff gdm-session-worker [pam/gdm-password] (UNKNOWN)

#^ 0xffffffffffffff /usr/sbin/gdm (UNKNOWN)

#^ 0xffffffffffffff /usr/lib/systemd/systemd --switched-root --system --deserialize 22 (UNKNOWN)

##

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools* high-order *--deactivate*

如果成功，将输出：

high-order is not activated

如果失败，将输出：

deactivate high-order fail, ret is -1

关闭功能后，本功能将不会对系统带来任何影响。

## drop-packet

本功能统计内核态一段时间内，在各个TCP/UDP连接上，各个环节的报文数量。统计丢包发生的位置。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools drop-packet --help

结果如下：

drop-packet usage:

--help drop-packet help info

--activate

verbose VERBOSE

source-addr source addr you want monitor

source-port source port you want monitor

dest-addr dest addr you want monitor

dest-port dest port you want monitor

--deactivate

--report dump log with text.

--log

sls=/tmp/1.log store in file

syslog=1 store in syslog.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools drop-packet --activate

在激活本功能时，可用参数为：

verbose 该参数设置输出级别，暂时未用。

source-addr 要监控的源地址，可以不设置。

source-port 要监控的源端口，可以不设置。

dest-addr 要监控的目的地址，可以不设置。

dest-port 要监控的目的端口，可以不设置。

例如，如下命令设置输出级别为1：

diagnose-tools drop-packet --activate='verbose=1'

如果成功，将输出：

功能设置成功，返回值：0

输出级别：1

如果失败，将输出：

功能设置失败，返回值：-16

输出级别：1

### 测试用例

运行如下命令启动本功能的测试用例：

sh /usr/diagnose-tools/test.sh drop-packet

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools drop-packet --settings

结果如下：

功能设置：

是否激活：√

输出级别：0

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools drop-packet --report

输出结果示例如下：

协议类型：UDP, 源IP：10.0.2.15, 源端口：54578, 目的IP：30.14.128.1, 目的端口：53

ETH\_RECV: pkg-count: 1, true-size: 896, len: 194, datalen: 0

GRO\_RECV: pkg-count: 1, true-size: 896, len: 180, datalen: 0

GRO\_RECV\_ERR: pkg-count: 0, true-size: 0, len: 0, datalen: 0

RECV\_SKB: pkg-count: 1, true-size: 896, len: 180, datalen: 0

RECV\_SKB\_DROP: pkg-count: 0, true-size: 0, len: 0, datalen: 0

IP\_RCV: pkg-count: 0, true-size: 0, len: 0, datalen: 0

IP\_RCV\_FINISH: pkg-count: 0, true-size: 0, len: 0, datalen: 0

DST\_INPUT: pkg-count: 1, true-size: 896, len: 180, datalen: 0

LOCAL\_DELIVER: pkg-count: 1, true-size: 896, len: 180, datalen: 0

LOCAL\_DELIVER\_FINISH: pkg-count: 1, true-size: 896, len: 180, datalen: 0

UDP\_RCV: pkg-count: 1, true-size: 896, len: 160, datalen: 0

TCP\_V4\_RCV: pkg-count: 0, true-size: 0, len: 0, datalen: 0

SEND\_SKB: pkg-count: 1, true-size: 768, len: 85, datalen: 0

输出结果中，包含了报文在各个阶段被接收/发送的次数。

每次输出结果后，历史数据将被清空。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools* drop-packet --*deactivate*

如果成功，将输出：

drop-packet is not activated

如果失败，将输出：

deactivate drop-packet fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

## tcp-retrans

本功能统计内核态一段时间内，各个TCP连接上面的重传计数。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools tcp-retrans --help

结果如下：

tcp-retrans usage:

--help tcp-retrans help info

--activate

verbose VERBOSE

source-addr source addr you want monitor

source-port source port you want monitor

dest-addr dest addr you want monitor

dest-port dest port you want monitor

--deactivate

--report dump log with text.

--log

sls=/tmp/1.log store in file

syslog=1 store in syslog.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools tcp-retrans --activate

在激活本功能时，可用的参数有：

verbose 该参数设置输出级别，暂时未用。

例如，如下命令设置输出级别为1：

diagnose-tools tcp-retrans --activate='verbose=1'

如果成功，将输出:

功能设置成功，返回值：0

输出级别：1

如果失败，将输出：

功能设置失败，返回值：-16

输出级别：1

### 测试用例

运行如下命令启动本功能的测试用例：

sh /usr/diagnose-tools/test.sh tcp-retrans

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools tcp-retrans --settings

结果如下：

功能设置：

是否激活：√

输出级别：1

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools tcp-retrans --report

可以在report的时候，指定ignore参数，这样就不会打印重传数量较少的五元组，如：

diagnose-tools tcp-retrans --report="ignore=5"

这样，重传次数在5次以下的将被忽略。

输出结果示例如下：

TCP重传调试统计：：

分配次数：0

tcp\_retransmit\_skb调用次数：10

tcp\_rtx\_synack调用次数：0

tcp\_dupack调用次数：15

tcp\_send\_dupack调用次数：0

源地址： 10.0.2.15[703]， 目的地址： 180.101.49.12[9999]， SYNC重传次数: 0, 报文重传次数： 4

源地址： 10.0.2.15[14498]， 目的地址： 180.101.49.11[9999]， SYNC重传次数: 0, 报文重传次数： 6

输出结果中包含一些调试统计值，以及每个连接上的重传统计。包含sync重传和报文重传统计。

每次输出结果后，历史数据将被清空。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools tcp-retrans --deactivate*

如果成功，将输出：

tcp-retrans is not activated

如果失败，将输出：

deactivate tcp-retrans fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

## ping-delay

本功能追踪ping包的时间延迟。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools ping-delay --help

结果如下：

ping-delay usage:

--help ping\_delay help info

--activate

verbose VERBOSE

addr filtered ip address.

--deactivate

--settings dump settings

--report dump log with text.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

﻿ ﻿ ﻿diagnose-tools ping-delay --activate

在激活本功能时，可用参数为：

verbose 该参数设置输出级别，当该值为1时，输出详细的报文信息。

addr 设置要过滤的IP地址。

例如，如下命令设置输出级别为1：

diagnose-tools ping-delay --activate='verbose=1'

如果成功，将输出：

*功能设置成功，返回值：0*

*输出级别：1*

*过滤地址：0.0.0.0*

如果失败，将输出：

*功能设置失败，返回值：-16*

*输出级别：1*

*过滤地址：0.0.0.0*

### 测试用例

运行如下命令运行测试用例，以查看本功能是否正常：

sh /usr/diagnose-tools/test.sh ping-delay

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools ping-delay --settings

结果如下：

功能设置：

是否激活：×

输出级别：0

过滤地址：0.0.0.0

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools ping-delay --report

输出结果示例如下：

PING延时信息, 源IP：[10.0.2.15], 目的IP：[180.101.49.12], ID：6074, SEQ: 2, 时间：[1589802125:725029]

PD\_ETH\_RECV: 14277527179528

PD\_GRO\_RECV: 14277527180325

PD\_GRO\_RECV\_ERR: 0

PD\_RECV\_SKB: 14277527181863

PD\_RECV\_SKB\_DROP: 0

PD\_IP\_RCV: 0

PD\_IP\_RCV\_FINISH: 0

PD\_DST\_INPUT: 14277527220383

PD\_LOCAL\_DELIVER: 14277527220563

PD\_LOCAL\_DELIVER\_FINISH: 14277527220652

PD\_ICMP\_RCV: 14277527234109

PD\_SEND\_SKB: 14277491867008

输出结果中包含ping报文在各个阶段的时间，以ns为单位。

每次输出结果后，历史数据将被清空。

### 关闭功能

通过如下命令关闭本功能：

diagnose-tools ping-delay --*deactivate*

关闭功能后，本功能将不会对系统带来性能影响。

## rw-top

本功能监控一段时间内执行文件写的进程和文件。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools rw-top --help

结果如下：

rw-top usage:

--help rw-top help info

--activate

verbose VERBOSE

top how many items to dump

shm set 1 if want dump shm

perf set 1 if want perf detail

--deactivate

--report dump log with text.

--log

sls=/tmp/1.log store in file

syslog=1 store in syslog.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools rw-top --activate

在激活本功能时，可用参数为：

verbose 设置输出信息的详细程度

top 设置输出列表的长度，默认值是20。

shm 如果设置为1,将只监控对共享内存文件的读写。

例如，如下命令设置输出列表长度为100：

diagnose-tools rw-top --activate='top=100'

如果成功，将输出：

功能设置成功，返回值：0

TOP：100

SHM：0

PERF: 0

输出级别：0

如果失败，将输出：

功能设置失败，返回值：-16

TOP：100

SHM：0

PERF: 0

输出级别：0

### 测试用例

运行如下命令启动本功能的测试用例：

sh /usr/diagnose-tools/test.sh rw-top

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools rw-top --settings

结果如下：

功能设置：

是否激活：×

TOP：0

SHM：0

PERF1

输出级别：1

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools rw-top --report

输出结果示例如下：

序号 R-SIZE W-SIZE MAP-SIZE RW-SIZE 文件名

1 0 66375 0 66375 /apsarapangu/tmp.txt

输出结果中，包含了写数量排名前100名的文件名/读写长度。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools* rw-top --*deactivate*

如果成功，将输出：

rw-top is not activated

如果失败，将输出：

deactivate rw-top fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

## fs-shm

本功能监控当前打开的SHM文件。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools fs-shm --help

结果如下：

fs-shm usage:

--help fs-shm help info

--activate

verbose VERBOSE

top how many items to dump

--deactivate

--report dump log with text.

--log

sls=/tmp/1.log store in file

syslog=1 store in syslog.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools fs-shm --activate

在激活本功能时，可用的参数有：

verbose 设置输出信息的详细程度

top 设置输出列表的长度，默认值是20。

例如，如下命令设置输出列表长度为100：

diagnose-tools fs-shm --activate='top=100'

如果成功，将输出：

功能设置成功，返回值：0

TOP：100

输出级别：0

如果失败，将输出：

功能设置失败，返回值：-16

TOP：100

输出级别：0

### 测试用例

运行如下命令将启动本功能的测试用例：

sh /usr/diagnose-tools/test.sh fs-shm

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools fs-shm --settings

结果如下：

功能设置：

是否激活：×

TOP：100

输出级别：0

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools fs-shm --report

输出结果示例如下：

序号 FILE-SIZE 容器 PID 进程名 文件名

0 8388608 / 1584 systemd-journal /run/log/journal/aebdf2677ae545de8ce26bb89f163484/system.journal

1 8388608 / 3458 rsyslogd /run/log/journal/aebdf2677ae545de8ce26bb89f163484/system.journal

2 132 / 4794 gdm /run/gdm/auth-for-baoyou.xie-oaCIhg/database

3 132 / 4794 gdm /run/gdm/auth-for-gdm-RhBRA1/database

4 5 / 4831 VBoxService /run/vboxadd-service.sh

5 5 / 3480 atd /run/atd.pid

6 5 / 2969 abrtd /run/abrt/abrtd.pid

7 5 / 3484 crond /run/crond.pid

8 5 / 1608 lvmetad /run/lvmetad.pid

9 4 / 3471 libvirtd /run/libvirtd.pid

10 0 / 3471 libvirtd /run/libvirt/network/nwfilter.leases

11 0 / 2962 rpcbind /run/rpcbind.lock

输出结果中，包含了写数量排名前50名的SHM文件。

每次输出结果后，历史数据将被清空。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools fs-shm --deactivate*

如果成功，将输出：

fs-shm is not activated

如果失败，将输出：

deactivate fs-shm fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

## fs-orphan

输出ext4文件系统下的孤儿节点信息，只适配了v3.10与v4.9版本的内核。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools fs-orphan--help

结果如下：

fs-orphan usage:

--help fs-orphan help info

--activate

--deactivate

--settings print settings.

--report dump log with text.

--verbose VERBOSE

--dev devname that monitored, for instance dba

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools fs-orphan --activate

激活本功能时，可用参数为：

verbose VERBOSE该参数设置输出级别，暂时未用

dev 要分析的设备，如sda

例如，如下命令设置要分析的磁盘设备名称为sda：

diagnose-tools fs-orphan --activate='dev=sda'

如果成功，将输出：

功能设置成功，返回值：0

输出级别：0

DEV：sda

如果失败，将输出：

功能设置失败，返回值：-16

输出级别：0

DEV：sda

### 测试用例

运行如下命令运行测试用例，以查看本功能是否正常：

sh /usr/diagnose-tools/test.sh fs-orphan

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools fs-orphan --settings

结果如下：

功能设置：

是否激活：√

输出级别：0

DEV：sda

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools fs-orphan --report

每次输出结果后，历史数据将被清空。

### 输出火焰图

理论上，可以输出孤儿节点相关的火焰图，但是目前还未实现此功能。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools* fs-orphan *--deactivate*

如果成功，将输出：

deactivate fs-orphan

如果失败，将输出：

deactivate fs-orphan fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

## fs-cache

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools fs-cache --help

结果如下：

fs-cache usage:

--help fs-cache help info

--activate

verbose VERBOSE

top how many items to dump

size filter size

--deactivate

--report dump log with text.

--drop invalid file cache

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools df-du --activate

在激活本功能时，可用参数为：

verbose VERBOSE该参数设置输出级别，暂时未用

top 指定输出数据的数量

size 当指定此参数时，只考虑那些缓存大小超过此值的文件。

例如：

diagnose-tools fs-cache -activate='top=100'

如果成功，将输出：

功能设置成功，返回值：0

TOP：100

输出级别：0

如果失败，将输出：

功能设置失败，返回值：-16

TOP：100

输出级别：0

### 测试用例

使用如下命令测试本功能：

sh /usr/diagnose-tools/test.sh fs-cache

### 删除文件缓存

使用如下命令删除特定文件的缓存

diagnose-tools fs-cache --drop='inode=$ADDR'

其中$ADDR是inode节点的内核地址，例如 18446612134122553728

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools fs-cache --settings

结果如下：

功能设置：

是否激活：√

TOP：100

输出级别：0

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools fs-cache --report

每次输出结果后，历史数据将被清空。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools fs-cache --deactivate*

如果成功，将输出：

fs-cache is not activated

如果失败，将输出：

deactivate fs-cache fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

## reboot

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools reboot --help

结果如下：

reboot usage:

--help reboot help info

--activate

--deactivate

--verbose VERBOSE

--settings dump settings

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools reboot --activate

如果成功，将输出:

reboot activated

如果失败，将输出：

reboot is not activated, ret 0

### 设置参数

本功能可用参数为：

-v, --verbose 该参数设置输出级别，暂时未用。

例如，如下命令设置输出级别为1：

diagnose-tools reboot --verbose=1

如果成功，将输出：

set verbose for reboot: 1, ret is 0

如果输出，将输出：

set verbose for reboot: 1, ret is -1

### 测试用例

无

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools reboot --settings

结果如下：

功能设置：

是否激活：√

输出级别：1

### 查看结果

在复位后，通过串口日志查看本命令的输出结果。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools* reboot *--deactivate*

如果成功，将输出：

reboot is not activated

如果失败，将输出：

deactivate reboot fail, ret is -1

关闭功能后，本功能将不会对系统带来性能影响。

# 测试命令

## test-md5

这是一个测试CPU速率的小工具。

使用方法：

diagnose-tools test-md5

该命令默认执行1000,0000次md5计算。

也可以使用-c参数指定计算次数，如：

﻿[root@localhost diagnoise-tool]# diagnose-tools test-md5 -c 5000000

加密前:admin

加密后:21232f297a57a5a743894a0e4a801fc3

real 0m3.600s

user 0m3.581s

sys 0m0.006s

## test-pi

这是另一个测试CPU速率的小工具。

使用方法：

diagnose-tools test-pi

可以附带两个参数，其中：

-c --cpu可以指定将测试用例绑定到哪一个CPU上运行。

-v --verbose可以打开详细信息

## test-memcpy

这是一个测试内存速率的小工具。

使用方法：

diagnose-tools test-memcpy

可以附带两个参数，其中：

-c --cpu可以指定将测试用例绑定到哪一个CPU上运行。

-v --verbose可以打开详细信息

## test-run-trace

## test-run-trace-java

## test-presure-java

# 实验版本的功能

目前，diagnose-tools-2.0实验版本有如下几个功能：

**kern-demo：**展示如何在diagnose-tools中添加一个功能，供开发同学使用。

sys-broken：监控系统调用被中断/软中断/定时器打断的时间。

**mm-leak：**统计内核态一段时间内，分配了但是没有释放的内存。并输出分配这些内存的调用链，以及泄漏次数。

## pupil小工具

无

## kern-demo

﻿略

## mm-leak

本功能统计内核态一段时间内，分配了但是没有释放的内存。并输出分配这些内存的调用链，以及泄漏次数。

### 查看帮助信息

通过如下命令查看本功能的帮助信息：

diagnose-tools mm-leak --help

结果如下：

mm-leak usage:

--help mm-leak help info

--activate

--deactivate

--verbose VERBOSE

--report dump log with text.

### 安装KO

参见《安装和卸载KO》一节

### 激活功能

激活本功能的命令是：

diagnose-tools mm-leak --activate

如果成功，将输出：

mm-leak activated

### 设置参数

本功能可用参数为：

-v, --verbose 该参数设置输出级别，暂时未用。

例如，如下命令设置输出级别为1：

diagnose-tools mm-leak --verbose=1

该命令在控制台会输出如下结果：

set verbose for mm-leak: 1, ret is 0

### 测试用例

无

### 查看设置参数

使用如下命令查看本功能的设置参数：

diagnose-tools mm-leak --settings

结果如下：

功能设置：

是否激活：√

输出级别：1

### 查看结果

执行如下命令查看本功能的输出结果：

diagnose-tools mm-leak --report

输出结果示例如下：

内存泄漏，次数：25337

内核态堆栈：

#@ 0xffffffff8103ddef save\_stack\_trace\_tsk ([kernel.kallsyms])

#@ 0xffffffffa1b09289 diagnose\_save\_stack\_trace [diagnose] ([kernel.kallsyms])

#@ 0xffffffffa1b0d7cf ali\_stack\_desc\_find\_alloc [diagnose] ([kernel.kallsyms])

#@ 0xffffffffa1b24461 trace\_kmem\_cache\_alloc\_hit [diagnose] ([kernel.kallsyms])

#@ 0xffffffff812264f7 kmem\_cache\_alloc ([kernel.kallsyms])

#@ 0xffffffff8128c141 alloc\_buffer\_head ([kernel.kallsyms])

#@ 0xffffffff8128c467 alloc\_page\_buffers ([kernel.kallsyms])

#@ 0xffffffff8128c4de create\_empty\_buffers ([kernel.kallsyms])

#@ 0xffffffff8128c621 create\_page\_buffers ([kernel.kallsyms])

#@ 0xffffffff8128e9aa \_\_block\_write\_begin\_int ([kernel.kallsyms])

#@ 0xffffffff812bb631 iomap\_write\_begin ([kernel.kallsyms])

#@ 0xffffffff812bb8c8 iomap\_write\_actor ([kernel.kallsyms])

#@ 0xffffffff812bbf31 iomap\_apply ([kernel.kallsyms])

#@ 0xffffffff812bc020 iomap\_file\_buffered\_write ([kernel.kallsyms])

#@ 0xffffffffa017144c xfs\_file\_buffered\_aio\_write [xfs] ([kernel.kallsyms])

#@ 0xffffffffa0171743 xfs\_file\_write\_iter [xfs] ([kernel.kallsyms])

#@ 0xffffffff8125251e new\_sync\_write ([kernel.kallsyms])

#@ 0xffffffff812526c6 \_\_vfs\_write ([kernel.kallsyms])

#@ 0xffffffff81252ce5 vfs\_write ([kernel.kallsyms])

#@ 0xffffffff812541a5 SyS\_write ([kernel.kallsyms])

#@ 0xffffffff81003c04 do\_syscall\_64 ([kernel.kallsyms])

#@ 0xffffffff8174bfce entry\_SYSCALL\_64\_after\_swapgs ([kernel.kallsyms])

内存泄漏，次数：13562

每次输出结果后，历史数据将被清空。

### 关闭功能

通过如下命令关闭本功能：

*diagnose-tools mm-leak --deactivate*

如果成功，将输出：

*mm-leak is not activated*

关闭功能后，本功能将不会对系统带来性能影响。

# btrace和uprobe

## btrace

btrace类似arthas，但是由于btrace可以定义脚本，所以在使用上相对arthas更加灵活，在某些arthas无法解决的场景，可以考虑使用btrace进行定位，其帮助文档位于:

https://github.com/btraceio/btrace/wiki/BTrace-Annotations?spm=ata.13261165.0.0.249b2086sNijQN

注意：本机需要指定JAVA\_HOME环境变量，供btrace使用，同时btrace内部提供了比较多的samples脚本:

export JAVA\_HOME=/opt/taobao/java/

## 示例：查看ThreadPoolExecutor初始化的堆栈

用法：

./bin/btrace 2184 ./samples/ThreadPoolExecutorInit.class > /tmp/init.log

代码：

$more ThreadPoolExecutorInit.java

package samples;

import com.sun.btrace.BTraceUtils;

import com.sun.btrace.annotations.BTrace;

import com.sun.btrace.annotations.OnMethod;

import com.sun.btrace.annotations.ProbeClassName;

import com.sun.btrace.annotations.ProbeMethodName;

import static com.sun.btrace.BTraceUtils.println;

@BTrace public class ThreadPoolExecutorInit {

@OnMethod(

clazz = "java.util.concurrent.ThreadPoolExecutor",

method = "<init>"

)

public static void logOnInit(@ProbeClassName String probeClass, @ProbeMethodName String probeMethod){

println("==== " + probeClass + " " + probeMethod);

BTraceUtils.Threads.jstack();

println("==== ================================");

}

}

## 示例：btrace使用diagnose-tools脚本

在btrace中使用diagnose-tools脚本进行排查定位，在进入com.taobao.tair.comm.TairClientFactory.createClient 的时候开始开启btrace，在该接口返回后退出btrace

注意使用unsafe=true 才可以在btrace脚本中调用外部代码，同时指定需要将btrace 增加启动参数： -Dcom.sun.btrace.unsafe=true ， 开启非安全模式后，方可执行非安全脚本

./bin/btrace 3496 ./samples/BtraceMain.java

代码：

package com.sun.btrace.samples;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import com.sun.btrace.BTraceUtils;

import com.sun.btrace.annotations.\*;

@BTrace(unsafe=true)

public class BtraceMain {

@OnMethod(clazz = "com.taobao.tair.comm.TairClientFactory", method = "createClient")

public static void start\_run\_trace() {

FileOutputStream out = null;

try {

File file = new File("/proc/ali-linux/diagnose/kern/run-trace-settings");

if (file.exists()) {

out = new FileOutputStream("/proc/ali-linux/diagnose/kern/run-trace-settings");

out.write("start\0".getBytes());

out.write(0);

out.write(System.getProperty("line.separator").getBytes());

}

} catch (IOException e) {

e.printStackTrace();

} finally {

if (out != null) {

try {

out.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

BTraceUtils.print("enter");

}

}