Perf 在 Linux 程序性能评估中的应用

承刚
核心系统部 内核组

讲师介绍页面

- 核心系统部 - 内核组 - 承刚
- 淘宝新同学，在内核组从事 Perf 与调度器相关的工作。
- 如果同学们在 Perf 使用中有任何疑问，欢迎与我沟通。如果发现了 Bug ，更加欢迎与我沟通。

系列课程介绍页面

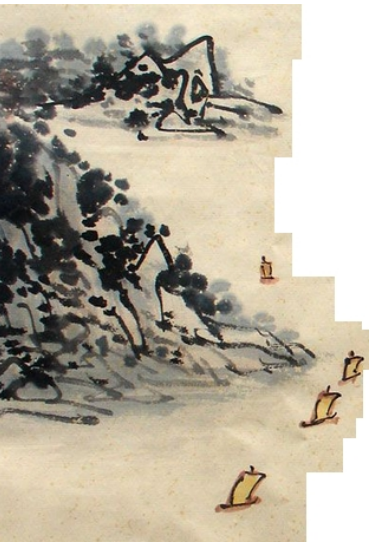
- 关于 Perf 的系列课程
  - 《 Perf 在 Linux 程序性能评估中的应用》
  - 《 Linux Perf 工具深入剖析》

课程目标与目标学员页

- 面向学员：对 Linux 程序的性能评估感兴趣的同学
- 课程目标：
  - 使同学们了解 Perf 的基本原理
  - 通过实例演示，使同学们了解 Perf 的功能和基本使用方法

课程大纲页

1. Perf 简介
2. Perf 原理
3. Perf 的使用方法
    - perf list
    - perf stat
    - perf top
    - perf record
    - perf report
    - Tracepoint
    - perf sched
    - perf timechart
    - perf script



Perf 简介

程序的系统级性能调优

- 算法优化： 空间复杂度 时间复杂度
- 代码优化： 提高执行速度 减少内存占用

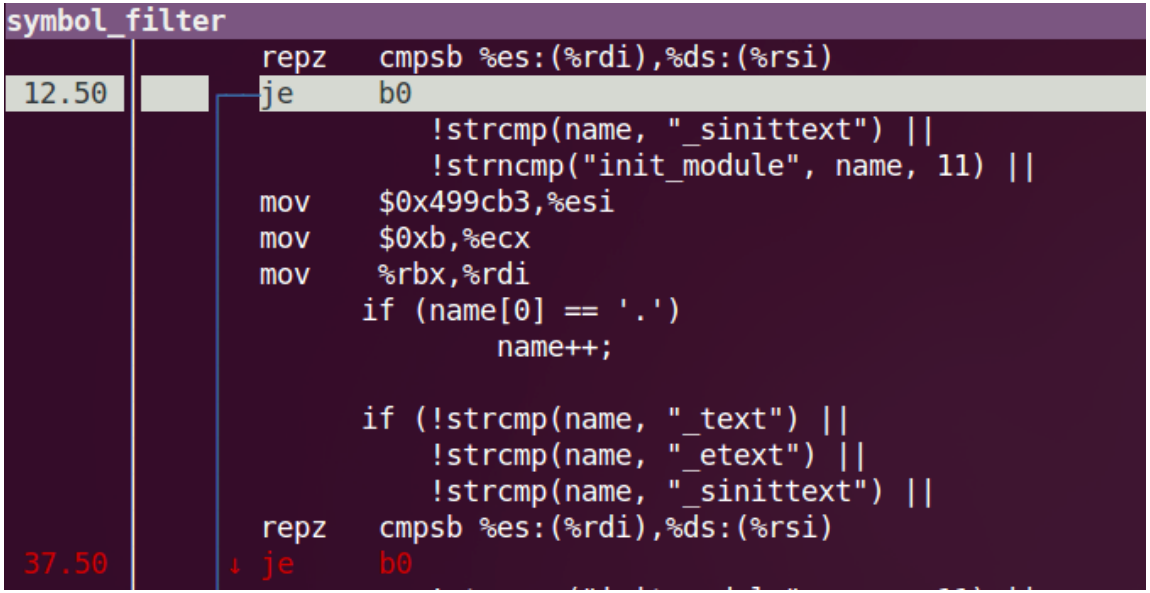
之后我们还能做些什么？ 系统级优化

- Cache 丢失率评估 --> 减少内存访问次数
- IPC 评估 --> 提高 CPU 利用率
- 上下文切换次数评估 --> 降低 OS 开销
- Page Fault 次数评估 --> 减少页面交换

→ 更加有效地利用硬件与 OS 资源

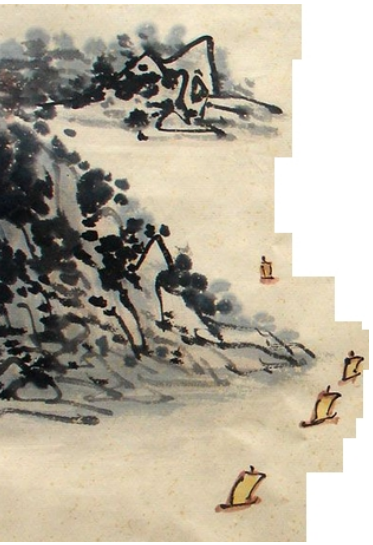
Perf 的功能

- 系统级性能分析的利器
- upstream 内核的一部分： from 2.6.31
  - kernel_path/tools/perf/
- 几乎能够处理所有与性能相关的事件
- 函数级与指令级的热点查找



Perf 的功能

- 评估程序对硬件资源的使用情况：
  各级 Cache 访问次数、各级 Cache 丢失次数、流水线停顿周期、前端总线访问次数 … …
- 评估程序对操作系统资源的使用情况：
  系统调用次数、 Page Fault 次数、上下文切换次数、任务迁移次数 … …
- 评估内核性能
  Benchmarks 、调度器性能分析、系统行为记录与重演
  动态添加探测点 … …



Perf 原理

Perf 原理

- 1. 性能事件
  - 1.1 硬件性能事件（PMU)
  - 1.2 软件性能事件
  - 1.3 Tracepoint

2. 基于时间的性能分析

3. 基于事件的性能分析

性能事件

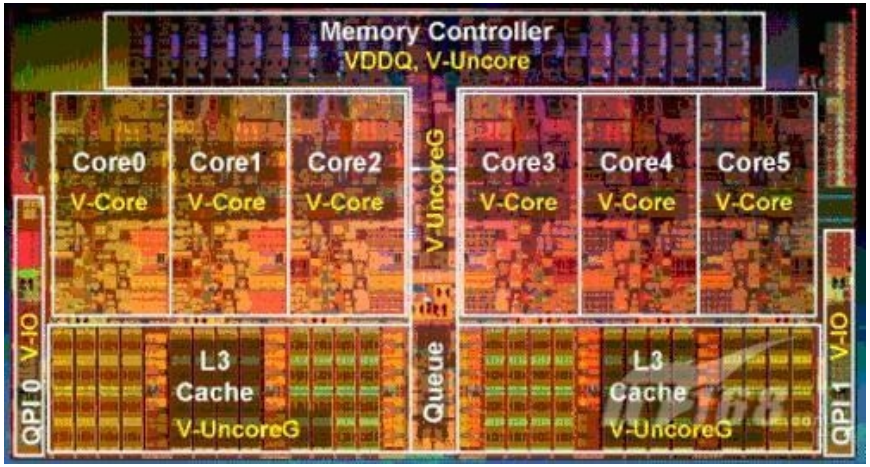
- 硬件相关
  - CPU Cycles、 Instructions、 Cache-References、 Cache-Misses、 Bus-Cycles、 Branch-Misses ... ...
- OS 相关
  - Page-Faults、 Context-Switches、 CPU-Migrations、 Emulation-Faults、 Task-Clock ...
- Tracepoint
  - 内核中所有的 Tracepoint，都可以作为 perf 的性能事件

PMU：性能监测单元

- PMU (Performance Monitoring Unit)
  - CPU 部件：监测处理器性能
  - Core PMU & Uncore PMU
  - 性能事件探测器：
    在特定条件下探测性能事件是否发生
  - 性能事件计数器：
    记录性能事件发生的次数

PMU：性能监测单元

- PMU 能够侦测的事件
  - Program Characterization
  - Memory Accesses
  - Pipeline stalls
  - Branch Prediction
  - Resource Utilization



软件性能事件

- 软件性能事件
  - 内置于 kernel，分布在各个
  - 功能模块中
  - 统计与操作系统相关的性能事件
  - 如：
    任务执行时间 -- HRTimer
    上下文切换次数
    任务迁移次数　等

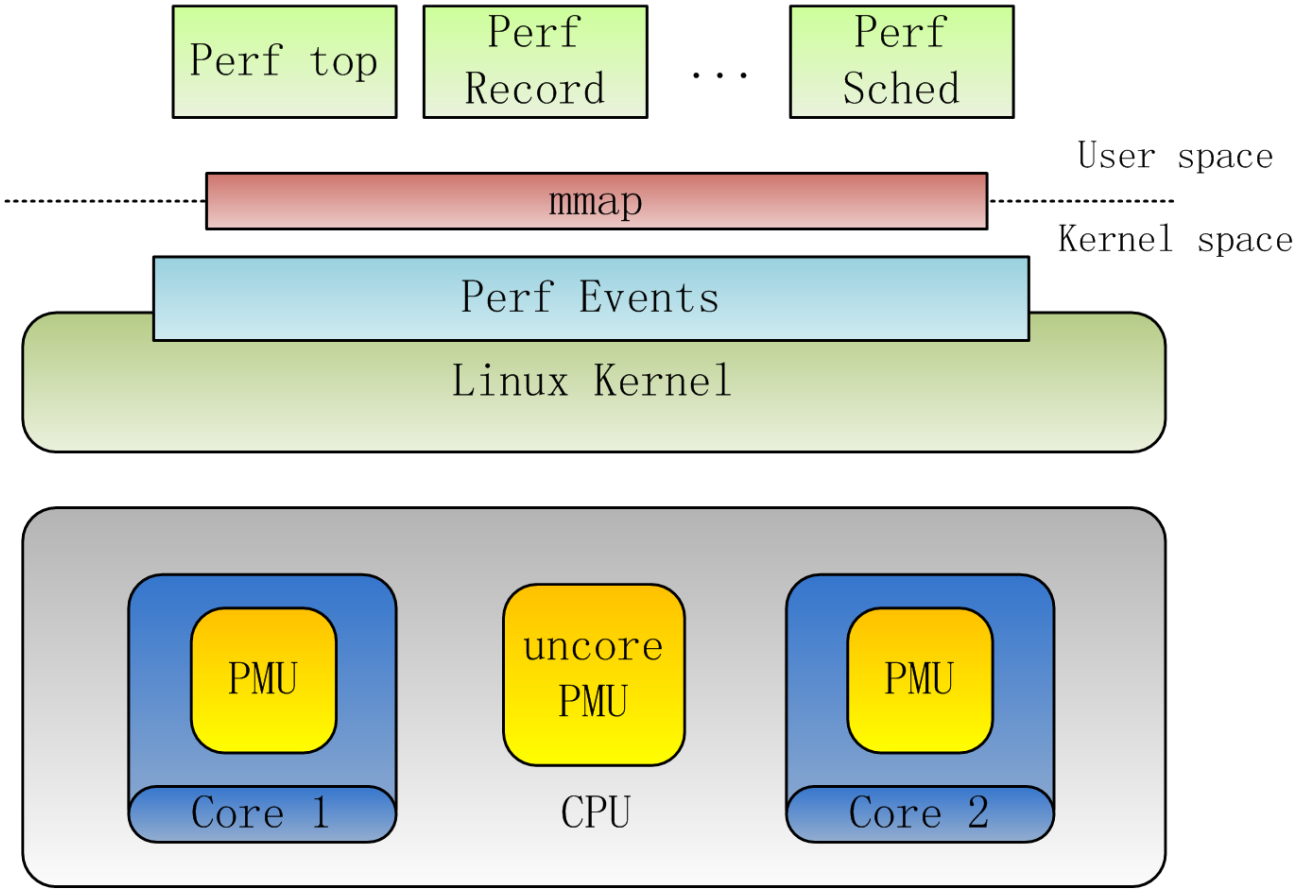
基于时间的性能分析

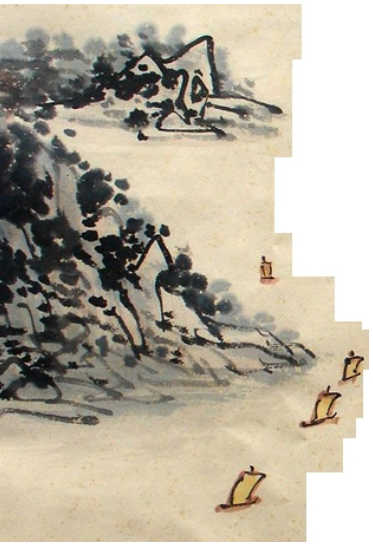
- 程序热点的查找
  - 如何找到程序中执行最频繁的热点代码？
- 周期性中断应用程序（周期性采样）
  - 记录当前指令地址（保存 Instruction Pointer [PC]）
  - IP -> function name
- 假定当前采样周期内一直在执行此函数
- 函数的处理器利用率：
  - 函数执行时间 / 程序总执行时间
- 采样频率 → 分析精度

基于事件的性能分析

- 针对时间无关的性能指标
  - 哪个函数 / 指令触发了最多的 Cache missing ？
  - 程序执行期间发生了多少次任务切换 ？
  - 哪个程序使用的系统调用数最多？
- 性能计数器
  - 性能计数器累积到一定数值（采样周期）时触发中断
  - ISR 记录当前进程的采样信息： PC， PID， TID 等
- 热点进程 / 热点函数 / 热点指令

Perf 的结构


Perf top
Perf Record
...
Perf Sched
User space
mmap
Kernel space
Perf Events
Linux Kernel
PMU
Core 1
uncore PMU
CPU
PMU
Core 2



Perf 的使用方法

Perf 使用方法

- perf list
- perf stat
- perf top
- perf record
- perf report
- Tracepoint
- perf sched
- perf timechart
- perf script

perf list

- 功能：
  - 查看当前软硬件环境支持的性能事件
  - 性能事件与 CPU 及内核版本相关
- 使用方法
  - # perf list

```
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
root@Latitude-E6400:perf# ./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]
  stalled-cycles-frontend OR idle-cycles-frontend    [Hardware event]
  stalled-cycles-backend OR idle-cycles-backend      [Hardware event]
  ref-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]

  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-prefetches                               [Hardware cache event]
  L1-dcache-prefetch-misses                          [Hardware cache event]
  L1-icache-loads                                    [Hardware cache event]
  L1-icache-load-misses                              [Hardware cache event]
  L1-icache-prefetches                               [Hardware cache event]
  L1-icache-prefetch-misses                          [Hardware cache event]
  LLC-loads                                          [Hardware cache event]
  LLC-load-misses                                    [Hardware cache event]
  LLC-stores                                         [Hardware cache event]
  LLC-store-misses                                   [Hardware cache event]
  LLC-prefetches                                     [Hardware cache event]
  LLC-prefetch-misses                                [Hardware cache event]
  dTLB-loads                                         [Hardware cache event]
  dTLB-load-misses                                   [Hardware cache event]
  dTLB-stores                                        [Hardware cache event]
  dTLB-store-misses                                  [Hardware cache event]
:
```

perf stat

- 功能
  - 分析程序的整体性能
- 使用方法
  - # perf stat ./your_prog
- 常用参数
  - ‘-e’: 指定性能事件
  - ‘-p’: 指定待分析进程的 PID
  - ‘-t’: 指定待分析线程的 TID

```
root@Latitude-E6400:perf# ./perf stat ./thread
old thread[4060]: 0 tid: 4061
old thread[4060]: 14 tid: 4075
old thread[4060]: 13 tid: 4074
old thread[4060]: 12 tid: 4073
old thread[4060]: 1 tid: 4062
old thread[4060]: 11 tid: 4072
old thread[4060]: 9 tid: 4070
old thread[4060]: 7 tid: 4068
old thread[4060]: 10 tid: 4071
old thread[4060]: 8 tid: 4069
old thread[4060]: 2 tid: 4063
old thread[4060]: 6 tid: 4067
old thread[4060]: 3 tid: 4064
old thread[4060]: 4 tid: 4065
old thread[4060]: 5 tid: 4066
^C./thread: 中断

 Performance counter stats for './thread':

      4263.359139 task-clock:HG              #    1.959 CPUs utilized
              501 context-switches:HG        #    0.118 K/sec
                8 CPU-migrations:HG          #    0.002 K/sec
              198 page-faults:HG             #    0.046 K/sec
    6,804,829,582 cycles:HG                  #    1.596 GHz
   <not supported> stalled-cycles-frontend:HG
   <not supported> stalled-cycles-backend:HG
    2,596,452,981 instructions:HG            #    0.38  insns per cycle
      124,329,311 branches:HG                #   29.162 M/sec
           39,409 branch-misses:HG           #    0.03% of all branches

      2.175925354 seconds time elapsed
```

perf stat

- 常用参数（ cont. ）
  - ‘-r N’：连续分析 N 次
  - ‘-d’：全面性能分析，采用更多的性能事件

CPU 密集型程序

- 输出信息

```
Performance counter stats for './thread':

      3347.518623 task-clock:HG              #    1.962 CPUs utilized
              372 context-switches:HG        #    0.111 K/sec
               13 CPU-migrations:HG          #    0.004 K/sec
              198 page-faults:HG             #    0.059 K/sec
    5,248,783,578 cycles:HG                  #    1.568 GHz                      [36.25%]
  <not supported> stalled-cycles-frontend:HG
  <not supported> stalled-cycles-backend:HG
    2,024,115,201 instructions:HG            #    0.39  insns per cycle          [49.49%]
       97,531,907 branches:HG                #   29.136 M/sec                    [50.29%]
           45,724 branch-misses:HG           #    0.05% of all branches          [53.11%]
    1,262,119,905 L1-dcache-loads:HG         #  377.031 M/sec                    [26.40%]
           63,767 L1-dcache-load-misses:HG   #    0.01% of all L1-dcache hits    [26.48%]
           46,879 LLC-loads:HG               #    0.014 M/sec                    [24.79%]
            2,703 LLC-load-misses:HG         #    5.77% of all LL-cache hits     [23.16%]

      1.706008343 seconds time elapsed
```

perf stat

- 例子：
  - 分支预测失效：

```
void func(void)
{
        int NUM = 20;
        int i, res;

        for (i=0; i<NUM; i++) {
                res += 1;
        }
}
```

```
int main(void)
{
        int i;
        for (i=0; i<100000000; i++)
                func();

        return 0;
}
```

perf stat

- 例子：

```
root@chenggang-Latitude:perf.ori# ./perf stat -d ./example/branch

 Performance counter stats for './example/branch':

       9207.651809 task-clock                #    0.999 CPUs utilized
                51 context-switches          #    0.006 K/sec
                60 CPU-migrations            #    0.007 K/sec
               112 page-faults               #    0.012 K/sec
    14,660,807,226 cycles                    #    1.592 GHz                      [24.94%]
   <not supported> stalled-cycles-frontend
   <not supported> stalled-cycles-backend
    11,375,098,307 instructions              #    0.78  insns per cycle          [37.53%]
     2,496,775,851 branches                  #  271.163 M/sec                    [37.54%]
           104,822 branch-misses             #    0.00% of all branches          [37.60%]
    10,736,990,047 L1-dcache-loads           # 1166.094 M/sec                    [25.07%]
           185,528 L1-dcache-load-misses     #    0.00% of all L1-dcache hits    [25.02%]
           114,685 LLC-loads                 #    0.012 M/sec                    [25.00%]
            11,364 LLC-load-misses           #    9.91% of all LL-cache hits     [24.96%]

      9.219138173 seconds time elapsed
```

perf stat

- 例子：
  - 修改后的代码：

```
void func(void)
{
        int NUM = 10;
        int i, res;

        for (i=0; i<NUM; i++) {
                res += 2;
        }
}
```

```
int main(void)
{
        int i;
        for (i=0; i<100000000; i++)
                func();

        return 0;
}
```

perf stat

- 例子：

```
root@chenggang-Latitude:perf.ori# ./perf stat -d ./example/branch2

 Performance counter stats for './example/branch2':

      4638.496184 task-clock                #    0.999 CPUs utilized
              25 context-switches          #    0.005 K/sec
              18 CPU-migrations            #    0.004 K/sec
             112 page-faults               #    0.024 K/sec
    7,387,750,439 cycles                   #    1.593 GHz                      [25.01%]
  <not supported> stalled-cycles-frontend
  <not supported> stalled-cycles-backend
    6,405,023,339 instructions             #    0.87  insns per cycle          [37.51%]
    1,501,057,047 branches                 #  323.609 M/sec                    [37.50%]
           35,446 branch-misses            #    0.00% of all branches          [37.51%]
    5,582,438,241 L1-dcache-loads          # 1203.502 M/sec                    [25.09%]
           65,248 L1-dcache-load-misses    #    0.00% of all L1-dcache hits    [25.06%]
           39,053 LLC-loads                #    0.008 M/sec                    [25.04%]
            2,379 LLC-load-misses          #    6.09% of all LL-cache hits     [25.01%]

      4.641335460 seconds time elapsed
```

perf top

- 功能
  - 实时显示系统 / 进程的性能统计信息
- 使用方法
  - # perf top
- 常用参数
  - ‘-e’: 指定性能事件 (默认事件： cycles)
  - ‘-p’: 指定待分析进程的 PID
  - ‘-t’: 指定待分析线程的 TID

perf top

- 常用参数（ cont. ）
  - ‘-a’：分析整个系统的性能（ Default ）
  - ‘-c’：事件的采样周期
  - ‘-d’：界面的刷新周期（ Default： 2s ）
  - ‘-E’：显示的函数条数
  - ‘-r <priority>’：将 perf top 作为实时任务，优先级为 <priority>
  - '-K'：不显示内核符号
  - '-U'：不显示用户符号

perf top

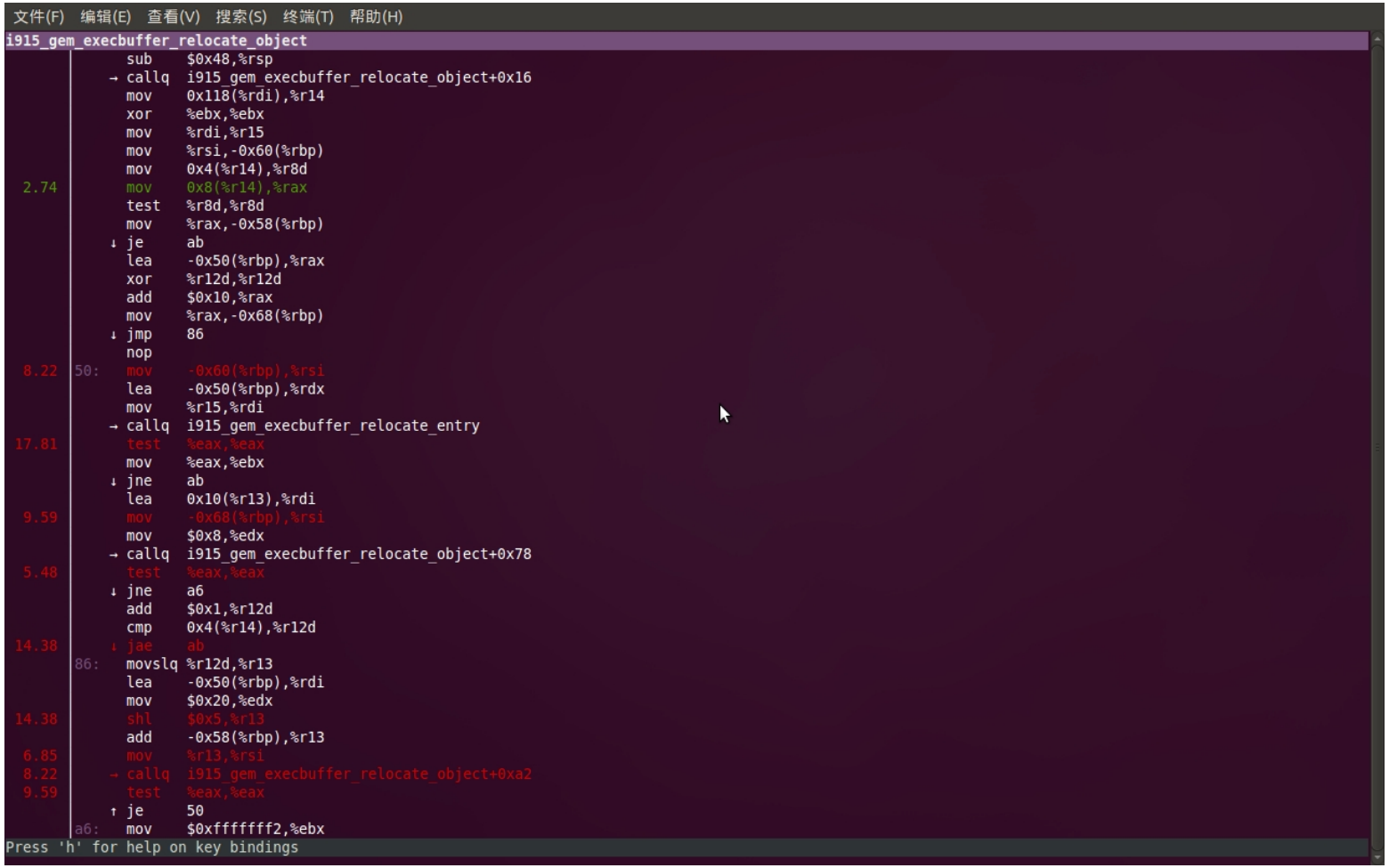
- 输出信息

文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)

```
Samples: 24K of event 'cycles', Event count (approx.): 2143399325
  5.71%  [kernel]                          [k] read_hpet
  3.28%  libcairo.so.2.11000.2             [.] 0x0000000000052603
  3.17%  [drm]                             [k] drm_clflush_pages
  2.82%  [unknown]                         [.] 0x00007f6ea69c53ee
  2.13%  [kernel]                          [k] copy_user_generic_string
  1.90%  [kernel]                          [k] __ticket_spin_lock
  1.82%  libdrm_intel.so.1.0.0             [.] 0x00000000000049ff
  1.53%  libglib-2.0.so.0.2800.6           [.] g_hash_table_lookup
  1.47%  Xorg                              [.] 0x000000000000c20a2
  1.45%  libpthread-2.13.so                [.] pthread_mutex_lock
  1.09%  libgtk-x11-2.0.so.0.2400.4        [.] 0x0000000000147a78
  1.07%  libc-2.13.so                      [.] _int_malloc
  1.06%  intel_drv.so                      [.] 0x00000000000223d7
  1.03%  libpthread-2.13.so                [.] pthread_mutex_unlock
  0.99%  [nls_iso8859_1]                   [.] 0x000000005f14217b
  0.98%  libgobject-2.0.so.0.2800.6        [.] 0x000000000001fdc2
  0.98%  libc-2.13.so                      [.] __GI___strcmp_ssse3
  0.92%  [kernel]                          [k] unix_poll
  0.91%  libc-2.13.so                      [.] memcpy
  0.87%  libpng12.so.0.44.0                [.] 0x000000000000bf00
  0.81%  libc-2.13.so                      [.] malloc
  0.80%  [kernel]                          [k] clear_page_c
  0.77%  libgdk-x11-2.0.so.0.2400.4        [.] 0x0000000000058140
```

perf top

- Annotate: 将性能分析细化到指令



perf record

- 功能
  - 记录一段时间内系统 / 进程的性能事件
- 使用方法
  - # perf record [options] [<command>]
  - # perf record [options] -- <command> [options]
- 常用参数
  - ‘-e’: 指定性能事件 (默认事件： cycles)
  - ‘-p’: 指定待分析进程的 PID
  - ‘-t’: 指定待分析线程的 TID

perf record

- 常用参数（cont.）
  - ‘-a’：分析整个系统的性能（ Default ）
  - ‘-c’：事件的采样周期
  - ‘-o’：指定输出文件（ Default: perf.data ）
  - ‘-g’: 记录函数间的调用关系
  - ‘-r <priority>’：将 perf top 作为实时任务，优先级为 <priority>
  - ‘-u <uid>’：只分析用户 <uid> 创建的进程

perf record

- 输出信息
  - 默认在当前目录下生成数据文件：
    perf.data

perf report

- 功能
  - 读取 perf record 生成的 perf.data 文件，并显示分析数据
- 使用方法
  - # perf report [-i <file> | --input=file]
- 常用参数
  - ‘-i’：输入文件名
  - ‘-v’: 显示每个符号的地址
  - ‘-d <dso>’：只显示指定 dso 的符号

perf report

- 常用参数 **(cont.)**
  - ‘-n’：显示每个符号对应的事件数
  - ‘-v’: 显示每个符号的地址
  - ‘--comms=<comm>’只显示指定 comm 的信息
  - ‘-S <symbol name>’只考虑指定符号
  - ‘-U’只显示已解析的符号
  - ‘-g [type,min]’按照 [type,min] 指定的方式显示函数调用图

perf report

- 常用参数 (cont.)
  - ‘-g [type,min]’ 按照 [type,min] 指定的方式显示函数调用图

    type： flat - 线性展开所有调用链

    graph – 显示调用树，并显示每个调用树对应的绝对开销率

    fractal – 显示调用树，并显示每个调用树对应的相对开销率

    min：只显示开销率大于 min 的符号

perf report

- flat :

```
Samples: 32K of event 'cycles:HG', Event count (approx.): 12745485611
-  99.82%  thread  thread              [.] do_pi
     do_pi
     start_thread
+   0.04%  thread  [kernel.kallsyms]   [k] read_hpet
-   0.01%  thread  [kernel.kallsyms]   [k] schedule
     retint_careful
     do_pi
     do_pi
+   0.01%  thread  [kernel.kallsyms]   [k] hrtimer_interrupt
```

perf report

- graph :

```
Samples: 32K of event 'cycles:HG', Event count (approx.): 12745485611
-  99.82%  thread  thread                [.] do_pi
     do_pi
     start_thread
+   0.04%  thread  [kernel.kallsyms]     [k] read_hpet
-   0.01%  thread  [kernel.kallsyms]     [k] schedule
   - schedule
      - 0.01% retint_careful
           do_pi
        0.00% do_pi
+   0.01%  thread  [kernel.kallsyms]     [k] hrtimer_interrupt
-   0.01%  thread  [kernel.kallsyms]     [k] update_curr
   - update_curr
      + 0.00% reweight_entity
      + 0.00% put_prev_entity
+   0.01%  thread  [kernel.kallsyms]     [k] __math_state_restore
```

perf report

- fractal :

```
Samples: 32K of event 'cycles:HG', Event count (approx.): 12745485611
-  99.82%  thread  thread                [.] do_pi
     do_pi
     start_thread
+   0.04%  thread  [kernel.kallsyms]    [k] read_hpet
-   0.01%  thread  [kernel.kallsyms]    [k] schedule
   - schedule
      - 66.67% retint_careful
           do_pi
        33.33% do_pi
+   0.01%  thread  [kernel.kallsyms]    [k] hrtimer_interrupt
-   0.01%  thread  [kernel.kallsyms]    [k] update_curr
   - update_curr
      + 50.00% reweight_entity
      + 50.00% put_prev_entity
+   0.01%  thread  [kernel.kallsyms]    [k]   math_state_restore
```

内核 tracepoint 的使用

- Tracepoint
  - 在内核的 tracepoint 中可以插入 hook function，以追踪内核的执行流。
  - Taobao 2.6.32 内核中共有 801 个 tracepoint
  - Upstream 内核中共有 906 个 tracepoint
  - Perf 将 tracepoint 作为性能事件
    module:function
- 使用方法
  - 使用 perf list 查看当前系统支持的 tracepoint
  - perf top [record] –e module:function <other options>

内核 tracepoint 的使用

- Example:

  统计程序使用的系统调用数：

  perf stat -e raw_syscalls:sys_enter ls

Performance counter stats for 'ls‘:

128 raw_syscalls:sys_enter

0.002457299 seconds time elapsed

---

ls 在执行期间共调用了 128 次系统调用

perf timechart

- 功能

  将系统的运行状态以 SVG 图的形式输出。

  1. 各处理器状态（run, idle)
  2. 各进程的时间图谱（ run, sleep, blocked … ）

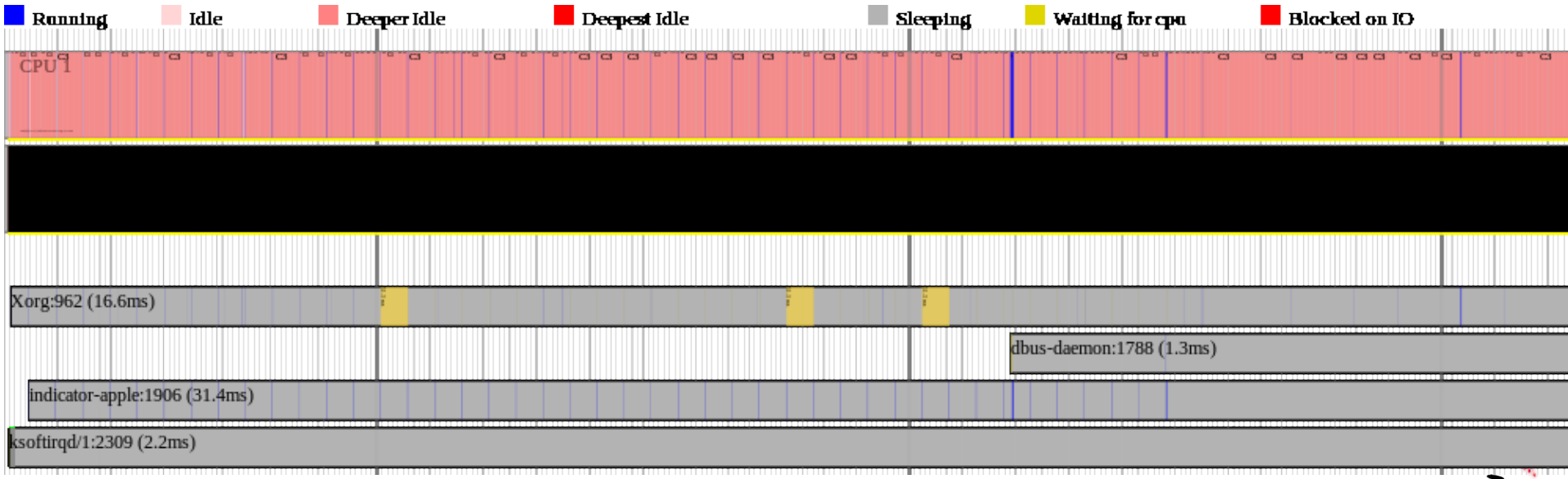
- 使用方法
  - 记录系统状态

    # perf timechart record
  - 绘制系统状态图

    # perf timechart

    输出： output.svg

perf timechart



perf script

- 功能
  - 查看 perf 的数据文件（ perf.data ）
  - 执行基于 python/perl 的扩展功能 (需要 python 环境)
- 使用方法

  1.查看 perf 的数据文件

  # perf script

  2.使用 perf 的扩展脚本

  3.2.1 查看系统中当前可用的扩展脚本

  # perf script -l

perf script

- 使用方法 (cont.)
- 2 使用 perf 的扩展脚本 (cont.)

1.2.2 扩展脚本

- - syscall-count
- 统计监测时段内，各个系统调用被调度的次数

2. # ./perf script syscall-count （演示）

3. - sctop

- 实时查看各个系统调用被调用的次数
- # ./perf script sctop （演示）

- - sched-migration
- 以图形显示监测时段内，各个任务在处理器间的
- 迁移情况。
- # ./perf script sched-migration （演示）

Q&A

谢 谢 ！