如何用 Perf 分析代码

下载 perf 源码

perf 的代码是 linux kernel 代码的一部分， 在./tools/perf/目录下面。linux kernel 代码可以从官网下载 https://www.kernel.org/

安装 perf

进入 linux 内核源码：linux/tools/perf，并执行

make

make install

提示：

1）可能在编译的时候，有报错大概是由于平台问题，数据类型不匹配，导致所有的 warning 都被当作 error 对待：出现这问题的原因是-Werror 这个 gcc 编译选项。只要在 makefile 中找到包含这个-Werror 选项的句子，将-Werror 删除，或是注释掉就行了

2）安装完毕，perf 可执行程序往往位于当前目录，可能不在系统的 PATH 路径中，此时需要改变环境变量 PATH。

Perf 是内置于 Linux 内核源码树中的性能剖析（profiling）工具。基于事件采样原理，以性能事件为基础，支持针对处理器相关性能指标与操作系统相关性能指标的性能剖析。可用于性能瓶颈的查找与热点代码的定位。

▲Perf 主要的用途在以下 2 个方面：

基于时间的性能分析

包括了可以再程序热点查找（最频繁执行的热点代码）；在周期性中断应用采样；函数的处理器利用率。

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 哪个程序使用的系统调用数最多等）；性能计数器（ISR 记录当前进程的采样信息：PC，PID，TID 性能计数器累积到一定数值（采样周期）时触发中断）；热点进程热点函数热点指令。

```
usage: perf [--version] [--help] [OPTIONS] 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 build-id cache.
   buildid-list    List the buildids in a perf.data file
   data            Data file related processing
   diff            Read perf.data files and display the differential profile
   evlist          List the event names in a perf.data file
   inject          Filter to augment the events stream with additional information
   kmem            Tool to trace/measure kernel memory properties
   kvm             Tool to trace/measure kvm guest os
   list            List all symbolic event types
   lock            Analyze lock events
   mem             Profile memory accesses
   **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.
   trace           strace inspired tool
   probe           Define new dynamic tracepoints

```

除此之外，Perf 还可以跟踪的性能事件非常全面，归结后可分为以下三种：

- Hardware Event 是由 PMU 硬件产生的事件，比如 cache 命中，当您需要了解程序对硬件特性的使用情况时，便需要对这些事件进行采样；
- Software Event 是内核软件产生的事件，比如进程切换，tick 数等 ；
- Tracepoint event 是内核中的静态 tracepoint 所触发的事件，这些 tracepoint 用来判断程序运行期间内核的行为细节，比如 slab 分配器的分配次数等。

我们只要了解下面这些常用就基本能判断问题了：

硬件（CPU 中的 Performance Monitor Unit）指标	解释
branch-instructions	# 执行的跳转指令数
branch-misses	# 分支预测失败数
cache-misses	# 缓存未命中
cache-references	# 缓存访问数
cpu-cycles or cycles	# CPU 时钟周期数
instructions	# 执行的指令数
stalled-cycles-frontend	# CPU前端等待（前端是指取指/译码、分支预测、访存等等东西，后端是指执行指令的过程）

软件指标	解释
alignment-faults	# 对齐错误，x86/x64 会自动处理访存不对齐，所以不用关注
bpf-output	# 跟 eBPF 有关
context-switches or cs	# 上下文切换数（程序被调度走了）
cpu-clock	#CPU 时钟（这里是软件统计的，不如上面的准)
cpu-migrations	# CPU 迁移（程序从 CPU1 执行，但是被调度到 CPU2 上去了）
major-faults	# 主要缺页错误（需要的页不在内存中，要去磁盘上取出再填进去）
minor-faults	# 次要缺页错误（需要的页在内存中，只是缺页表项，填上页表项就好了）
page-faults OR faults	# == major-faults + minor-faults

硬件指标（Hardware cache event）	解释
L1-dcache-load-misses	# L1 数据缓存未命中
L1-dcache-loads	# L1 数据缓存访问数
L1-dcache-prefetch-misses	# L1 数据缓存预取未命中
L1-dcache-store-misses	# L1 数据缓存RFO未命中（RFO: read-for-ownership，读取并且失效这个缓存）
L1-dcache-stores	# L1 数据缓存RFO
L1-icache-load-misses	# L1 指令缓存未命中
LLC-load-misses	# LLC: Last Level Cache，即 L2/L3 缓存。
branch-load-misses	# BPU 缓存未命中（BPU：Branch Processing Unit)
branch-loads	# 与分支预测有关的缓存
dTLB-load-misses	# 数据 TLB 缓存未命中（TLB: Translation Lookaside Buffer）
dTLB-loads	# TLB 是页表的缓存，用来加速虚拟地址->物理地址(VA->PA)转换的
iTLB-load-misses	# 指令 TLB 缓存未命中
node-load-misses	# 本地访存未命中，需要跨 NUMA node（其实就是 CPU）访存
node-loads	# 访存数

了解 perf 的功能和独特优势之后，接下来使用 Perf。

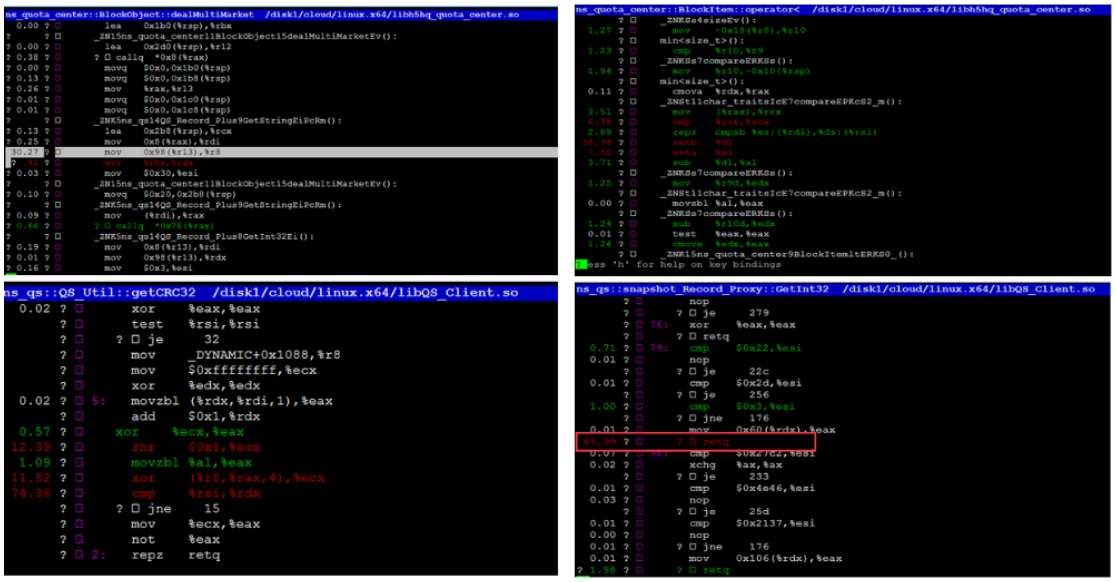
（1） perf top – 实时显示系统/进程的性能统计信息

```
perf  top  命令说明
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
 '-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）
```

先用 perf top 看看 CPU 实时热点情况，对资源的整体消耗有一个基本了解，并对照源码做一个基本定位。

```
Samples: 30K of event 'cpu-clock', Event count (approx.): 7090550756
Overhead  Shared Object                 Symbol
  16.12%  libc-2.12.so                  [.] memcpy
  14.63%  libh5hq_quota_center.so       [.] ns_quota_center::BlockItem::operator<
  12.74%  libh5hq_quota_center.so       [.] ns_quota_center::BlockObject::dealMultiMarket
   7.82%  libQS_Client.so               [.] ns_qs::snapshot_Record_Proxy::GetInt32
   7.13%  libc-2.12.so                  [.] _int_free
   5.75%  libc-2.12.so                  [.] malloc
   4.89%  libstdc++.so.6.0.13           [.] __gnu_cxx::__exchange_and_add
   3.75%  libc-2.12.so                  [.] _int_malloc
   2.92%  libstdc++.so.6.0.13           [.] std::basic_string<char, std::char_traits<char>, std::allocator<char> >::basic_string
   2.55%  libh5hq_quota_center.so       [.] std::_Rb_tree<ns_quota_center::BlockItem, ns_quota_center::BlockItem, std::_Identity<ns_quota
   1.82%  libh5hq_quota_center.so       [.] ns_quota_center::SRSIterator::get
   1.21%  libQS_Client.so               [.] ns_qs::snapshot_Record_Proxy::GetInt64
   1.04%  libQS_Client.so               [.] ns_qs::snapshot_Record_Proxy::GetString
   0.97%  libh5hq_quota_center.so       [.] ns_quota_center::FieldBigger::operator()
   0.94%  libc-2.12.so                  [.] __strlen_sse42
   0.82%  libh5hq_quota_center.so       [.] ns_quota_center::AllSnapshot::buildSortField
   0.70%  libstdc++.so.6.0.13           [.] std::basic_string<char, std::char_traits<char>, std::allocator<char> >::basic_string
   0.64%  libc-2.12.so                  [.] free
   0.60%  libstdc++.so.6.0.13           [.] operator new
   0.56%  libh5hq_quota_center.so       [.] ns_quota_center::BlockItem::BlockItem
   0.51%  [kernel]                      [k] _spin_unlock_irqrestore
   0.50%  libstdc++.so.6.0.13           [.] std::string::_Rep::_S_create
   0.45%  libh5hq_quota_center.so       [.] ns_quota_center::SRSIterator::next
```

选择 Annotate，找出热点（可以精确到 CPU 指令):



从跟踪情况可以看到 CPU 主要消耗在，MOV ，CMP，SETB，RETQ 等指令上面，而尤其 CMP 和 RETQ 指令在链表功能中使用时资源耗费最多，基于该处的怀疑，我们用 Perf stat 来对程序/线程进行分析。

（2） perf stat 来统计–用于分析指定程序/线程的性能概况

```
perf stat  命令说明
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
 ‘-r N’ : 连续分析N次
 ‘-d’ : 全面性能分析，采用更多的性能事件
```

**perf stat -e cache-references,cache-misses,task-clock , context-switches ,

cpu-migrations ,page-faults ,cycles ,L1-dcache-loads,L1-dcache-load-misses-t -a -d **

```
Performance counter stats for process id '12738':

     5583.553701      task-clock (msec)         #    1.100 CPUs utilized
         995,063      context-switches          #    0.009 M/sec
          12,038      cpu-migrations            #    0.162 K/sec
          13,992      page-faults               #    0.251 K/sec
      37,731,213      cycles                    #    2.904GHZ
         213,981      cache-misses              #    53.46% of all cache hits
       1,432,134      cache-references          #    87.871 M/sec
       1,134,981      L1-dcache-loads           #    66.674 M/sec
          91,213      L1-dcache-load-misses     #    22.88% of all L1-dcache hits

     50.740709471 seconds time elapsed
```

▲perf stat 输出解读如下

- task-clock

用于执行程序的 CPU 时间，单位是 ms(毫秒)。第二列中的 CPU utillized 则是指这个进程在运行 perf 的这段时间内的 CPU 利用率，该数值是由 task-clock 除以最后一行的 time elapsed 再除以 1000 得出的。

- context-switches

进程切换次数，记录了程序运行过程中发生了多少次进程切换，应该避免频繁的进程切换。

- cpu-migrations

程序在运行过程中发生的 CPU 迁移次数，即被调度器从一个 CPU 转移到另外一个 CPU 上运行。

- page-faults

缺页。指当内存访问时先根据进程虚拟地址空间中的虚拟地址通过 MMU 查找该内存页在物理内存的映射，没有找到该映射，则发生缺页，然后通过 CPU 中断调用处理函数，从物理内存中读取。

- Cycles

处理器时钟，一条机器指令可能需要多个 cycles。

- Cache-references

cache 命中的次数。

- Cache-misses

cache 失效的次数。

- L1-dcache-load-missed

一级数据缓存读取失败次数。

- L1-dcache-loads

一级数据缓存读取次数。

从 Perf stat 输出数据可以看到 CPU 的迁移次数较多， 程序上下文切换比较频繁，Cache 命中率不是很理想，内存缺页也比较多 ，对照源码定位到本次问题是因为为保证数据一致性在获取数据时增加了 CRC 码校验过程，只有当 CRC 码校验正确的情况下才进行后续工作，该处单线程处理，且有锁操作，导致程序频繁上下文切换、内存缺页严重。