Linux性能优化

代码写完，功能实现了，性能怎么样?

优化：

1. 算法优化，选择优秀的数据结构和算法。
2. 代码优化，提高执行速度，减少内存占用。

我们还能做些什么?

系统级优化

更加有效地利用硬件和操作系统资源

- 上下文切换次数评估 ---> 降低OS开销
- 缺页异常次数评估 ---> 减少页面交换
- IPC 评估 ---> 提高CPU利用率

如何做优化?

对症下药：系统性能瓶颈在哪里?

1、任务调度带来任务切换的开销

避免不必要的上下文切换

引起上下文切换的原因：

- 时间片用完，CPU正常调度下一个任务
- 被其他优先级更高的任务抢占
- 执行任务碰到IO阻塞，调度器挂起当前任务，切换执行下一个任务
- 用户代码主动挂起当前任务让出CPU时间 pthread_cond_wait
- 多任务抢占资源，由于没有抢到被挂起 mutex
- 硬件中断/软中断

perf性能检测工具的安装

```
sudo apt install linux-tools-common
sudo apt install linuex-tools-4.15***-generic
sudo apt install linux-tools-generic
```

```
ken@ubuntu:~$ sudo perf stat ls
Desktop    Downloads         Linux  Pictures  target.cap  test.c
Documents  examples.desktop  Music  Public    Templates   Videos

 Performance counter stats for 'ls':

// 任务真正占用的处理器时间，单位为ms。
          0.724643      task-clock (msec)         #    0.586 CPUs utilized //CPU
的占用率。
// 程序在运行过程中上下文的切换次数
                 6      context-switches          #    0.008 M/sec
// 程序在运行过程中发生的处理器迁移次数。
```

```
                 0      cpu-migrations            #    0.000 K/sec
// 缺页异常的次数。
                92      page-faults               #    0.127 M/sec

   <not supported>      cycles

   <not supported>      instructions

   <not supported>      branches

   <not supported>      branch-misses


       0.001237099 seconds time elapsed
```

查看使用系统调用次数

```
sudo perf stat -e raw_syscalls:sys_enter ls
```

CPU-migrations：程序在运行过程中发生的处理器迁移次数。Linux为了维持多个处理器的负载均衡，在特定条件下会将某个任务从一个CPU迁移到另一个CPU。
CPU迁移和上下文切换：发生上下文切换不一定会发生CPU迁移，而发生CPU迁移时肯定会发生上下文切换。发生上下文切换有可能只是把上下文从当前CPU中换出，下一次调度器还是将进程安排在这个CPU上执行。
page-faults：缺页异常的次数。当应用程序请求的页面尚未建立、请求的页面不在内存中，或者请求的页面虽然在内存中，但物理地址和虚拟地址的映射关系尚未建立时，都会触发一次缺页异常。另外TLB不命中，页面访问权限不匹配等情况也会触发缺页异常。
cycles：消耗的处理器周期数。如果把被ls使用的cpu cycles看成是一个处理器的，那么它的主频为2.486GHz。可以用cycles / task-clock算出。
stalled-cycles-frontend：指令读取或解码的质量步骤，未能按理想状态发挥并行左右，发生停滞的时钟周期。
`stalled-cycles-backend:` 指令执行步骤，发生停滞的时钟周期。
`instructions:` 执行了多少条指令。IPC为平均每个cpu cycle执行了多少条指令。
`branches:` 遇到的分支指令数。branch-misses是预测错误的分支指令数。

避免锁引起的调度，实现无锁设计(http://wetest.qq.com/lab/view/283.html)

2、系统调用带来用户态和内核切换的开销

系统调用造成用户态上下文栈压栈弹栈，有损耗。降低系统调用使用次数，例如
反复的read，write，就改为mmap，消息队列没有共享内存实现的消息队列快
(http://wetest.qq.com/lab/view/283.html)
多次的malloc可以升级为一次malloc
持久使用一块内存可以用大页 4K 2M 1G hugepages 避免发生tlb miss
通过使用sendfile及splice降低系统调用带来的开销 零COPY

http://blog.csdn.net/jasonliuvip/article/details/22600569

3、代码级别优化

热点函数，文件收集

```
sudo perf record -g ./test_branch
sudo perf report -i perf.data
```

高频的文件打开，需要持久打开，或者常驻内存
打开以后，队列里面去
Mysql数据库 访问过多，拒绝访问，怎么办 100个连接，队列。
建立连接池 队列
使用Redis
lsof统计系统打开的所有文件

函数热点分支收集

性能路径频繁调用的小函数内联化
修改为static inline实现函数内联

cache tlb 及分支预取优化
http://www.cnblogs.com/lirunzhou/p/5883698.html
http://blog.csdn.net/adam040606/article/details/49563453

性能优化工具列表
perf，oprofile，gprof，systemtap
linux有许多非常好用的性能分析工具，这里挑选几款最常用的简单介绍下：

1. perf应该是最全面最方便的一个性能检测工具。由linux内核携带并且同步更新，基本能满足日常使用。推荐大家使用。
2. oprofile，我觉得是一个较过时的性能检测工具了，基本被perf取代，命令使用起来也不太方便。比如opcontrol –no-vmlinux , opcontrol –init等命令启动，然后是opcontrol –start， opcontrol –dump， opcontrol -h停止，opreport查看结果等，一大串命令和参数。有时候使用还容易忘记初始化，数据就是空的。
3. gprof主要是针对应用层程序的性能分析工具，缺点是需要重新编译程序，而且对程序性能有一些影响。不支持内核层面的一些统计，优点就是应用层的函数性能统计比较精细，接近我们对日常性能的理解，比如各个函数时间的运行时间，，函数的调用次数等，很人性易读。
4. systemtap 其实是一个运行时程序或者系统信息采集框架，主要用于动态追踪，当然也能用做性能分析，功能最强大，同时使用也相对复杂。不是一个简单的工具，可以说是一门动态追踪语言。如果程序出现非常麻烦的性能问题时，推荐使用 systemtap。

Gcov,lcov 统计代码覆盖率
Benchmark

kememleak memcheck

valgrind 检测内存泄漏

网络是瓶颈，下面是socket参数调优
http://www.cnblogs.com/zengkefu/p/5749009.html
查看系统性能常用命令
http://www.infoq.com/cn/news/2015/12/linux-performance
https://github.com/chyyuu/os_course_exercises/blob/master/all/2-intr.md
操作系统一些题目学习
http://wetest.qq.com/lab/view/?id=103
概率性浮现问题定位案例分析
针对http://wetest.qq.com/lab/view/283.html这篇文章中出现的eventfd及singlefd的使用讲解
http://blog.csdn.net/freeelinux/article/details/53511331
http://blog.csdn.net/yusiguyuan/article/details/22934743