[gperftools使用详解](http://qiaoyong.lofter.com/post/2b36f_1140c82)

官方文档地址：

https://code.google.com/p/gperftools/wiki/GooglePerformanceTools

先说一下gperftools的优点吧

* 大厂出品， 质量放心
* valgrind这种单线程分析的工具无力分析多线程工具
* valgrind内存需要好几倍。对于大内存程序是无力分析的
* 用起来方便， 无需修改程序。比较适合敏捷

安装环境

下载地址如下：https://code.google.com/p/gperftools/downloads/list

安装这里需要注意编译选项。 一般需要libunwind库支持。 但x86\_64有内建的库。 所以需要configure的时候， 这样去做即可。打开frame-poionters的开关即可

./configure  --prefix=/home/qiaoyong/qiaoyong/memleak/gperftools --enable-frame-pointers

make

make install

安装的结果如下：

gperftools

├── bin

├── include

├── lib

└── share

我们只用到了lib下的东西。几个.so.

注意：我尝试了安装libunwind的库， 都未遂。 水深， 慎入。 非x86\_64的环境，就自求多福吧。

2， 一些通用的对程序的要求

* 程序需要自己能安全退出。 被信号杀掉无法生成分析文件。当然你可以信号处理函数里面自己做， 但比较麻烦。细节自己去查。
* 一般而言程序自己要做到内存释放干净。 否则测试的时候不容易看出来泄露。
* 还需要装个库：sudo apt-get install graphviz -y

3,  用tcmalloc测试内存泄露

试用tcmalloc非常简单， 编译的时候改一下配置就好

g++ -Wall  -g  -fno-omit-frame-pointer  py.cpp pool.cpp contents.cpp sug.cpp md5\_64bit.cpp  -I ./  -I ./sparsehash/include/ -I ./sparsehash/include/google -I ./sparsehash/include/sparsehash -lpthread -lcrypto -L/home/qiaoyong/qiaoyong/memleak/gperftools/lib  -ltcmalloc\_and\_profiler -o sug

有4个地方需要注意：

* 需要加一个 -fno-omit-frame-pointer﻿﻿选项。-g当然也是默认要有的。
* 添加动态库。 注意有单独的tcmalloc库和profiler的库。一般而言tcmalloc检查内存泄露；profiler检查cpu性能情况。
* 当然目前更多的常见用法， 是用tcmalloc作为内存分配器。这对于无内存泄露的线上程序是无问题的。**但常规的测试用tcmalloc，会隐藏很多内存上的问题。所以常规的测试应该去掉tcmalloc测试。**
* profiler会影响性能。切忌随便带上。

测试程序如下方式执行

HEAPCHECK=normal  LD\_LIBRARY\_PATH=/home/qiaoyong/qiaoyong/memleak/gperftools/lib ./sug

具体的参数可以看文档。 然后开始跑压力。

这里的参数有， 一般strict就够了

1. minimal
2. normal
3. strict
4. draconian

程序运行结束， 会生成一个分析文件结果。

利用pprof工具， 可以看结果。

./pprof ./sug "/tmp/sug.21854.\_main\_-end.heap" --inuse\_objects --lines --heapcheck  --edgefraction=1e-10 --text

这里可以有多样的参数， --pdf， --dot等， 内存泄露一般text就够了。

结果显示如下

qiaoyong@qiaoyong-VirtualBox:~/qiaoyong/sug$ ./pprof ./sug "/tmp/sug.21854.\_main\_-end.heap" --inuse\_objects --lines --heapcheck  --edgefraction=1e-10 --text

Using local file ./sug.

Using local file /tmp/sug.21854.\_main\_-end.heap.

Total: 26278 objects

    9357  35.6%  35.6%    18671  71.1% tcpsendmsg /home/qiaoyong/qiaoyong/sug/sug.cpp:360

    9314  35.4%  71.1%     9314  35.4% PendingPool::check\_item /home/qiaoyong/qiaoyong/sug/pool.cpp:308

    6957  26.5%  97.5%     6957  26.5% cn2py\_segment::fullfil\_hash /home/qiaoyong/qiaoyong/sug/py.cpp:178

     650   2.5% 100.0%      650   2.5% std::\_\_move\_median\_first /usr/include/c++/4.6/bits/stl\_algo.h:119

       0   0.0% 100.0%     9314  35.4% PendingPool::close\_not\_busy\_socket /home/qiaoyong/qiaoyong/sug/pool.cpp:191

       0   0.0% 100.0%    26278 100.0% \_\_libc\_start\_main ??:0

       0   0.0% 100.0%     6957  26.5% alloc\_impl /home/qiaoyong/qiaoyong/sug/./sparsehash/include/sparsehash/internal/densehashtable.h:1193

       0   0.0% 100.0%     6957  26.5% cn2py\_segment::do\_cn2py /home/qiaoyong/qiaoyong/sug/py.cpp:53

       0   0.0% 100.0%    19321  73.5% cn2py\_segment::do\_cn2py /home/qiaoyong/qiaoyong/sug/py.cpp:54

       0   0.0% 100.0%    19321  73.5% insert\_one\_query /home/qiaoyong/qiaoyong/sug/sug.cpp:743 [@](http://qiaoyong.lofter.com/post/2b36f_1140c82)

看到头4行没有。 这里显示了内存泄露的地方。

4， 试用profiler测试cpu上的性能热点

编译方式一样。

执行这样做

CPUPROFILE=./sug.prof LD\_LIBRARY\_PATH=/home/qiaoyong/qiaoyong/memleak/gperftools/lib ./sug

程序退出的时候， 性能的初始分析文件就是 ./sug.prof

也是利用pprof工具看这个文件

./pprof --text ./sug ./sug.prof

这里的参数随意吧， --pdf，--dot都可以。

--text可以看到这样的结果

qiaoyong@qiaoyong-VirtualBox:~/qiaoyong/sug$ ./pprof --text ./sug ./sug.prof

Using local file ./sug.

Using local file ./sug.prof.

Removing \_L\_unlock\_16 from all stack traces.

Total: 243 samples

      87  35.8%  35.8%       87  35.8% \_\_write\_nocancel

      14   5.8%  41.6%       14   5.8% \_\_nss\_hosts\_lookup

       9   3.7%  45.3%      199  81.9% child\_main

       7   2.9%  48.1%        7   2.9% \_\_close\_nocancel

       6   2.5%  50.6%        7   2.9% std::\_Iter\_base::\_S\_base

       5   2.1%  52.7%        5   2.1% \_\_accept\_nocancel

       5   2.1%  54.7%        8   3.3% std::\_\_niter\_base

       5   2.1%  56.8%        9   3.7% std::\_\_unguarded\_linear\_insert

       5   2.1%  58.8%        5   2.1% strlen

 .......

什么意思呢

Analyzing Text Output

Text mode has lines of output that look like this:

       14   2.1%  17.2%       58   8.7% std::\_Rb\_tree::find

Here is how to interpret the columns:

1. Number of profiling samples in this function
2. **Percentage of profiling samples in this function**
3. Percentage of profiling samples in the functions printed so far
4. Number of profiling samples in this function and its callees
5. **Percentage of profiling samples in this function and its callees**
6. Function name

基本上看第二列， 就知道热点在哪里了

如何分析数据

1， 一般好的程序， 时间片分布非常均匀

2， 有性能瓶颈的程序， 可能某个函数占据了60%+的性能。那么这个就需要重点优化点。我曾经的一个例子

1116 //int fuzzy::generate\_fuzzy\_query(vector<string> &py\_segment\_list, set<string> & results, int ed\_len)

1117 int fuzzy::generate\_fuzzy\_query(fuzzy\_py\_results &py\_segment\_list, unsigned int list\_seg,  set<string> & results, int ed     \_len)

发现这个函数， 占据了90%的时间。等等例子吧