[Protocol Buffers && Linux下的内存性能评测工具Profiler heap-check TCMalloc](http://blog.chinaunix.net/uid-17102734-id-3517896.html) 2013-03-12 21:56:44

分类： C/C++

[Protocol Buffers - Google's data interchange format  
(1)速度快，节省空间  
(2)跨平台，屏蔽了语言和平台之间的差异  
(3)支持协议的动态升级(by assigning and encoding the variable tag id)  
http://code.google.com/p/protobuf/](http://code.google.com/p/protobuf/)<https://developers.google.com/protocol-buffers/docs/overview>gperftools - [Fast, multi-threaded malloc() and nifty performance analysis tools  
  
These tools are for use by developers so that they can create more robust applications. Especially of use to those developing multi-threaded applications in C++ with templates. Includes **TCMalloc, heap-checker, heap-profiler and cpu-profiler**.](http://code.google.com/p/gperftools/)  
  
<http://code.google.com/p/gperftools/>  
<http://google-perftools.googlecode.com/svn/trunk/doc/>下面总结一下安装gperftools和使用方面的经验:  
  
(1)安装gperftools之前需要安装libunwind 

**安装gpreftools的时候需要按照install文件中的要求首先安装制定版本的libunwind.**

**安装libunwind时候出现的错误**

**点击(此处)折叠或打开**

1. gcc -DHAVE\_CONFIG\_H -I. -I../include -I../include -I../include/tdep-x86\_64 -I. -D\_GNU\_SOURCE -DNDEBUG -g -O2 -fexceptions -Wall -Wsign-compare -MT setjmp/longjmp.lo -MD -MP -MF setjmp/.deps/longjmp.Tpo -c setjmp/longjmp.c -fPIC -DPIC -o setjmp/.libs/longjmp.o
2. /usr/include/x86\_64-linux-gnu/bits/setjmp2.h:26:13: error: 'longjmp' aliased to undefined symbol '\_longjmp

解决方法: 在下载的代码包中 src中的MakeFile文件的CPPFLAGS增加编译选项 U\_FORTIFY\_SOURCE

**点击(此处)折叠或打开**

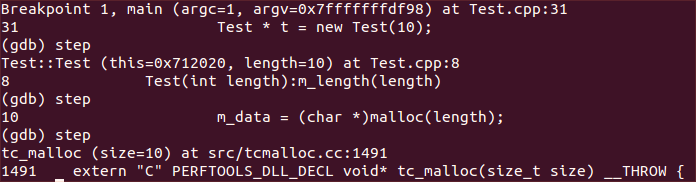
1. 之前的是
2. CPPFLAGS = -D\_GNU\_SOURCE -DNDEBUG
3. 之后的是:
4. CPPFLAGS = -D\_GNU\_SOURCE -DNDEBUG -U\_FORTIFY\_SOURCE

(2) TCMalloc使用  
  
**-->在对应的MakeFile中增加-lprofiler**

**点击(此处)折叠或打开**

1. **CC=g++**
2. **APP\_NAME = GPrefTest**
3. **OBJS = Test.o**
4. **INC = -I.**
5. **LIBS = -lrt -lpthread -lc -lstdc++ -ltcmalloc -lprofiler**
6. **FLAGS = -g -W -fpic**
7. **test:$(OBJS)**
8. **$(CC) $(FLAGS) -o $(APP\_NAME) $^ $(INC) $(LIBS)**
9. **$(OBJS): %.o: %.cpp**
10. **$(CC) $(FLAGS) $(INC) -c $<**
11. **clean:**
12. **rm -rf \*.o \*.lo \*.slo \*.la .libs**

**神奇的是代码不需要变化，就能实现之TCMalloc的功能:GDB来看-->使用的是钩子。**

  
  
(3) heap-check的使用  
   遇到问题:

<http://hi.baidu.com/xiaoyur347/item/c5e11594fc520bcab7253130>

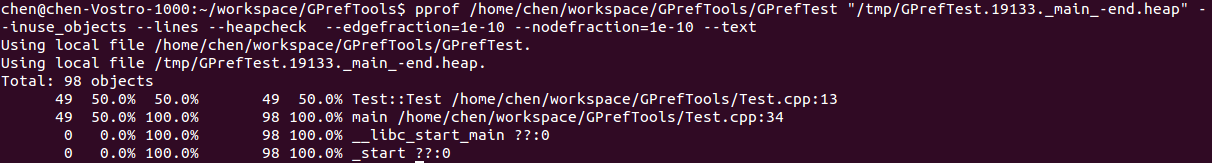
**点击(此处)折叠或打开**

1. Check failed: !do\_main\_heap\_check: should have done it

修改heap\_checker.cc中HeapLeakChecker\_AfterDestructors（在main退出后执行）从高亮代码do\_main\_heap\_check的位置移动到FLAGS\_heap\_check\_after\_destructors条件中。以解决heap-checker\_debug\_unittest.sh和heap-checker\_unittest.sh失败的问题。

**点击(此处)折叠或打开**

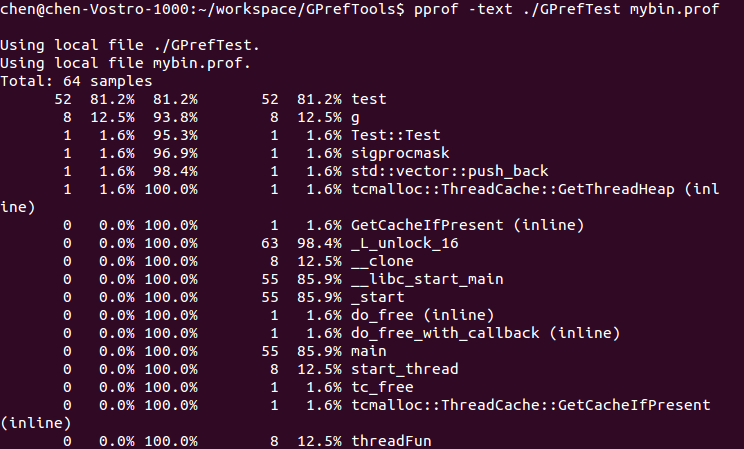
1. // This function is executed after all global object destructors run.
2. void HeapLeakChecker\_AfterDestructors() {
3. { SpinLockHolder l(&heap\_checker\_lock);
4. // can get here (via forks?) with other pids
5. if (heap\_checker\_pid != getpid()) return;
6. }
7. if (FLAGS\_heap\_check\_after\_destructors) {
8. if (HeapLeakChecker::DoMainHeapCheck()) {
9. const struct timespec sleep\_time = { 0, 500000000 };  // 500 ms
10. nanosleep(&sleep\_time, NULL);
11. // Need this hack to wait for other pthreads to exit.
12. // Otherwise tcmalloc find errors
13. // on a free() call from pthreads.
14. }
15. //added
16. SpinLockHolder l(&heap\_checker\_lock);
17. RAW\_CHECK(!do\_main\_heap\_check, "should have done it");
18. }
19. //deleted
20. //SpinLockHolder l(&heap\_checker\_lock);
21. //RAW\_CHECK(!do\_main\_heap\_check, "should have done it");
22. }

and rebuild ./configure && make && sudo make install    
  
测试程序  
代码:  
省略...  
命令:  
env HEAPCHECK=normal "待执行的程序"  
内存泄漏位置分析  


(4) CPU-Profiler 使用

    使用方法  
       (A)首先加入-lprofiler  
    (B)In your code, bracket the code you want profiled in calls to ProfilerStart() and ProfilerStop(). (These functions are declared in .) ProfilerStart() will take the profile-filename as an argument.  
测试代码  
点击(此处)折叠或打开

* 1. #include <stdio.h>
  2. #include <stdlib.h>
  3. #include <vector>
  4. #include <time.h>
  5. #include <iostream>
  6. #include <google/profiler.h>
  7. class Test
  8. {
  9. public:
  10. Test(int length):m\_length(length)
  11. {
  12. //m\_data = new char(10);
  13. m\_data = (char \*)malloc(10);
  14. }
  15. ~Test()
  16. {
  17. delete m\_data;
  18. m\_data = NULL;
  19. }
  20. private:
  21. int m\_length;
  22. char \* m\_data;
  23. /\* data \*/
  24. };
  25. void test()//time consuming
  26. {
  27. for (int i = 0; i < 100000000; ++i)
  28. {
  29. /\* code \*/
  30. }
  31. }
  32. void g()//time consuming
  33. {
  34. for (int i = 0; i < 100; ++i)
  35. {
  36. /\* code \*/
  37. usleep(5000);
  38. std::cout<<"thread is running"<<std::endl;
  39. for (int i = 0; i < 100000; ++i);
  40. }
  41. }
  42. void \* threadFun(void \*)
  43. {
  44. g();
  45. }
  46. int main(int argc, char const \*argv[])
  47. {
  48. /\* code \*/
  49. ProfilerStart("mybin.prof");
  50. std::vector<Test \* > testVec;
  51. time\_t startT = time( NULL );
  52. for (int i = 0; i < 50000; ++i)
  53. {
  54. /\* code \*/
  55. Test \* t = new Test(10);
  56. testVec.push\_back(t);
  57. }
  58. test();
  59. pthread\_t thread\_id;
  60. int error = pthread\_create(&thread\_id, NULL, threadFun, NULL);
  61. if (error != 0)
  62. printf("can't create threadn");
  63. for(std::vector<Test \* >::iterator vecIt = testVec.begin();vecIt != testVec.end();vecIt++)
  64. {
  65. if(\*vecIt != NULL)
  66. {
  67. delete \*vecIt;
  68. }
  69. }
  70. char \* p = new char(100);
  71. p = NULL;
  72. testVec.clear();
  73. time\_t endT = time(NULL);
  74. std::cout<<"time:"<<endT-startT<<std::endl;
  75. pthread\_join(thread\_id, NULL);
  76. ProfilerStop();
  77. getchar();
  78. return 0;
  79. }

(2) 测试结果  
  


Number of profiling samples in this function

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