**链接tcmalloc与不链接tcmalloc内存分配性能对比测试**

目录

[一 测试环境](#_Toc27599_WPSOffice_Level1) [2](#_Toc27599_WPSOffice_Level1)

[二 测试方法](#_Toc16355_WPSOffice_Level1) [2](#_Toc16355_WPSOffice_Level1)

[三 测试代码编写及分析](#_Toc9246_WPSOffice_Level1) [2](#_Toc9246_WPSOffice_Level1)

[四 测试数据汇总](#_Toc27352_WPSOffice_Level1) [5](#_Toc27352_WPSOffice_Level1)

[五 测试结论](#_Toc6301_WPSOffice_Level1) [7](#_Toc6301_WPSOffice_Level1)

**一 测试环境**

IP：192.168.97.30

OS：18.04.1-Ubuntu SMP Fri Mar 15 15:27:12 UTC 2019

CPU：6Core，Intel(R) Core(TM) i5-9400F CPU @ 2.90GHz

内存：16G

g++版本：g++ (Ubuntu 8.2.0-1ubuntu2~18.04) 8.2.0

机器类别：实体机

**二 测试方法**

1. 测试程序是单进程，单线程；
2. 测试程序定义带模板参数类，可生成不同字节的对象，从64字节到2M；
3. 测试程序循环测试100w次new与delete操作；
4. 在链接tcmalloc与不链接tcmalloc两种情况下，统计这100w次new与delete时间延迟；
5. 测试程序多次测试，采用求均值的方法得出测试数据。

**三 测试代码编写及分析**

#include <sys/time.h>

#include <sys/time.h>

#include <string.h>

#include <iostream>

#include <string>

using namespace std;

static const int LOOP = 1000000;

static const int TRYCNT = 5;

static const size\_t SIZE64 = 64;

static const size\_t SIZE128 = 128;

static const size\_t SIZE512 = 512;

static const size\_t SIZE1024 = 1024;

static const size\_t SIZE2048= 1024 \* 2;

static const size\_t SIZE4196 = 1024 \* 4;

static const size\_t SIZE8192 = 1024 \* 8;

static const size\_t SIZE16K = SIZE8192 \* 2;

static const size\_t SIZE64K = SIZE16K \* 4;

static const size\_t SIZE128K = SIZE64K \* 2;

static const size\_t SIZE256K = SIZE128K \* 2;

static const size\_t SIZE512K = SIZE256K \* 2;

static const size\_t SIZE1M = SIZE1024 \* SIZE1024;

static const size\_t SIZE2M = 2 \* SIZE1024 \* SIZE1024;

static const size\_t SIZE4M = 4 \* SIZE1024 \* SIZE1024;

template<auto SIZE>

class Test

{

public:

Test()

{

memset(buf, 'A', SIZE);

}

private:

char buf[SIZE];

};

int main()

{

struct timeval start\_time = { 0 };

struct timeval end\_time = { 0 };

long total\_use\_useconds = 0;

long use\_useconds = 0;

for (int cnt = 0; cnt < TRYCNT; cnt++)

{

gettimeofday(&start\_time, NULL);

for (int i = 0; i < LOOP; i++)

{

//Test<SIZE64>\*p = new Test<SIZE64>; // tcmalloc -- 0.009886s no tcmalloc -- 0.012395s

//Test<SIZE128>\* p = new Test<SIZE128>; // tcmalloc -- 0.010792s no tcmalloc -- 0.013048s

//Test<SIZE512>\* p = new Test<SIZE512>; // tcmalloc -- 0.014859s no tcmalloc -- 0.021042s

//Test<SIZE1024>\* p = new Test<SIZE1024>; // tcmalloc -- 0.017297s no tcmalloc -- 0.026832s

//Test<SIZE2048>\* p = new Test<SIZE2048>; // tcmalloc -- 0.025106s no tcmalloc -- 0.037326s

//Test<SIZE4196>\* p = new Test<SIZE4196>; // tcmalloc -- 0.048244s no tcmalloc -- 0.081024s

//Test<SIZE8192>\* p = new Test<SIZE8192>; // tcmalloc -- 0.079766s no tcmalloc -- 0.13078s

//Test<SIZE1M>\* p = new Test<SIZE1M>; // tcmalloc -- 66.8704s no tcmalloc -- 20.6404s

//Test<SIZE2M>\* p = new Test<SIZE2M>; // tcmalloc -- 132.338s no tcmalloc -- 40.8435s

//Test<SIZE4M>\* p = new Test<SIZE4M>; // tcmalloc -- 132.338s no tcmalloc -- 40.8435s

//Test<SIZE16K>\* p = new Test<SIZE16K>; // tcmalloc -- 0.144789s no tcmalloc -- 0.230068s

//Test<SIZE64K>\* p = new Test<SIZE64K>; // tcmalloc -- 0.862808s no tcmalloc -- 1.1384s

//Test<SIZE128K>\* p = new Test<SIZE128K>; // tcmalloc -- 1.81849s no tcmalloc -- 2.2705s

//Test<SIZE256K>\* p = new Test<SIZE256K>; // tcmalloc -- 3.9121s no tcmalloc -- 4.65727s

Test<SIZE512K>\* p = new Test<SIZE512K>; // tcmalloc -- 9.60905s no tcmalloc -- 4.65727s

delete p;

}

gettimeofday(&end\_time, NULL);

use\_useconds = 1000000 \* (end\_time.tv\_sec - start\_time.tv\_sec) + (end\_time.tv\_usec - start\_time.tv\_usec);

total\_use\_useconds += use\_useconds;

}

cout << "average elapse time = " << total\_use\_useconds / TRYCNT / 1e6 << "s" << endl;

return 0;

}

代码里定义Test类，可实例化不同字节的对象，在main函数中循环测试了5次，最终以平均值作为测试数据，并输出。

编译脚本：

g++ -std=c++17 -g -o TcmallocTest main\_tcmalloc.cpp -ltcmalloc -pthread

g++ -std=c++17 -g -o NoTcmallocTest main\_tcmalloc.cpp -pthread

**四 测试数据汇总**

本次测试使用相同环境与方法对链接tcmalloc与不链接tcmalloc进行性能测试，最终测试数据取多次测试数据的平均值，测试数据如下：

|  |  |  |  |
| --- | --- | --- | --- |
| **消息字节** | **不链接tcmalloc耗时(S)** | **链接tcmalloc耗时(S)** | **性能提升** |
| 64 | 0.012395 | 0.009886 | 20.24% |
| 128 | 0.013048 | 0.010792 | 17.29% |
| 512 | 0.021042 | 0.014859 | 29.38% |
| 1K | 0.026832 | 0.017297 | 35.54% |
| 2K | 0.037326 | 0.025106 | 32.74% |
| 4K | 0.081024 | 0.048244 | 40.46% |
| 8K | 0.13078 | 0.079766 | 39.01% |
| 16K | 0.230068 | 0.144789 | 37.07% |
| 64K | 1.1384 | 0.862808 | 24.21% |
| 128K | 2.2705 | 1.81849 | 19.91% |
| 256K | 4.65727 | 3.9121 | 16.00% |
| 512K | 9.60905 | 32.9355 | -242,8% |
| 1M | 20.6404 | 66.8704 | -223,9% |
| 2M | 40.8435 | 132.338 | -224,0% |

**五 测试结论**

1. tcmalloc使用简单，只要在编译时链接tcmalloc库即可，不需要包含头文件；
2. 对于256k以下内存分配和释放，tcmalloc优势明显，，对于4K~8K内存分配，程序性能最高可以提升40%；
3. 对于256K以上的内存分配与释放，tcmalloc性能下降很严重，因此tcmalloc适合小内存分配（几百字节到256K）的应用场景。