浅谈C++ IO优化——读优输优方法集锦

posted on 2018-11-17 22:55:24 | under 未分类 (.#type=未分类) | 2

零、写在前面

IO（读入/输出）优化是很实用&简单的常数优化~~（卡常技巧~~。C++为了兼容性导致 `cin` 、 `cout` 慢过天际，对于大量数据的读入和输出往往不堪重负。这个时候使用读入优化、输出优化可以节省数倍的时间。

很多人说Pascal读入快。其实Pascal的读入只比普通 `cin` 快（~~这点确实是碾压~~），在很多时候并不如 `scanf` 和关闭流同步的 `cin` 。

本文旨在介绍与操作系统无关的~~我会的~~C++的IO优化，因此 `Linux` 平台下的 `mmap` 优化不作讨论。

本文中所有读入优化、输出优化只适用于整数。由于实数读入优化/输出优化适用范围不广、效率不高，本文不作讨论。但如果你读完并理解本文，是可以写出好的实数读入优化/输出优化来的。只需要在原来的基础上处理小数点即可，留给读者自行思考。

本文中所有的代码均在luoguOJ、Windows上通过编译和测试。如果你的程序无法通过编译，可能是你把本文各部分拼凑起来导致重名了。

本文中所有的 **读优** 、 **输优** 均指 **读入优化** 、 **输出优化** 。~~就是不想叫快读~~

由于一些东西是挺久之前写的，所以可能码风什么的不一样，不要见怪。

文章目录：

一、基本读入优化

其实应该是谁都会的。防止萌新不知道~~凑字数~~还是写一下，大佬可以跳过该部分。

~~大佬为什么要看这种东西~~

先不解释，上代码。 给出最简单的读入优化：

```
inline int redn() {
    int ret=0,f=1;char ch=getchar();
    while (ch<'0'||ch>'9') {if (ch=='-') f=-f;ch=getchar();}
    while (ch>='0'&&ch<='9') ret=ret*10+ch-'0',ch=getchar();
    return ret*f;
}
```

也可以这样写方便重载:

```
inline int redn(int& ret) {
    ret=0,f=1;char ch=getchar();
    while (ch<'0'||ch>'9') {if (ch=='-') f=-f;ch=getchar();}
    while (ch>='0'&&ch<='9') ret=ret*10+ch-'0',ch=getchar();
    ret*=f;
}
```

当然也可以传指针进去……反正差不多，再写就没意思了。

优化核心在于 `getchar` 的速度很快。（头文件 `<cstdio>` ）

原理我想大家应该都懂吧。先一直读入字符直到出现数字，再一直读入数字同时累乘直到读入的不是数字。可以用 `isdigit` 写。（头文件 `<cctype>`）

有人说可以用位运算累乘。但是你如果懂一点汇编就知道这是没有区别的。

二、对于不定参数的读优重载

也可以叫可变参数、可变长参数。

需要C++ 11

怎么看是否支持C++ 11？包括luogu、codeforce等各大OJ都是支持的。本机支持不支持编译一下能通过就是支持。说到codeforce，据说在它上面 `cin` 比 `scanf` 快？

能不能提速这个不好说啊……反正用起来更方便是真的。

先上代码：

```
#include <iostream>
#include <cstdio>
#include <cctype>
#define ri register int
using namespace std;
template <typename T>inline void redi(T& t)
{
    ri c=getchar();t=0;
    while(!isdigit(c))c=getchar();
    while(isdigit(c))t=t*10+c-48,c=getchar();
}
template <typename T,typename... Args> inline void redi(T& t, Args&... args)
{
    redi(t);redi(args...);
}
int main (int argc,char const * argv[]) {
    int a,b,c,d;
    redi(a,b,c,d);
    cout<<a+b+c+d<<endl;
    return 0;
}
```

用来读无符号整数，要读负数的话稍微改一下第一个函数就好（见【基本读入优化】）。

首先你要知道什么是 `template` 。

代码中 template <typename T> 前缀表示一个类型。换句话说，你传入的t是 int 类型变量，T就表示 int ；如果传入的t是 long long 类型的，T就是 long long 。

第二个函数是第一个函数的重载。主要是 typename... Args 的问题吧。这个表示不定参。就是不确定有多少个参数。

可能讲的不是很清楚，但你自己用GDB跑一遍肯定就完全理解了。

或者去微软的MSDN上看看：https://msdn.microsoft.com/zh-cn/library/dn439779.aspx (https://msdn.microsoft.com/zh-cn/library/dn439779.aspx)

还有一个用 var_list 的版本，但没这个简单，有兴趣的可以自己了解一下（比如可以去看看 printf 函数的定义）。这里就不作讨论了。

三、基于fread的读入优化

考虑到 getchar 的速度还是不够快，我们可以使用 fread （头文件 <cstdio> ）来优化 getchar ，代码如下：

```
#define getchar()(p1==p2&&(p2=(p1=buf)+fread(buf,1,1<<21,stdin),p1==p2)?EOF:*p1++)
char buf[1<<21],*p1=buf,*p2=buf;
```

然后可以直接使用前文所述的读优。

打成三目运算符可能不太好理解哈。

关于逗号“，”，不理解的话看这个例子： return 1,2,3,4 最终返回的是4，也就是最后一个。还需要注意， && 是“短路运算符”，一旦当前值为假就不会继续往后执行，直接返回。

所以这句话的意思是：设置头指针p1和尾指针p2，fread一次读入1<<21个字符存在buf里，然后用p1来调用；当p1撞到p2时再次用fread读入1<<21个字符……

因此优化原理就是让 fread 一次性读入大量数据，再让 getchar 直接调用内存，如果刚开始读的不够多那就再调用 fread 读入数据，直到文件的末尾。要注意 fread 是很快的。

后文中的输优也差不多是这样的。

fread 函数的用法很简单，这里不再赘述了。百度搜索关键词：fread、fread返回值、fread的用法。

当然过多的宏定义很危险，所以也可以直接这样写成函数：

```
char buf[1<<21],*p1=buf,*p2=buf;
inline int getc(){
    return p1==p2&&(p2=(p1=buf)+fread(buf,1,1<<21,stdin),p1==p2)?EOF:*p1++;
}
inline int redi() {
    int ret = 0,f = 0;char ch = getc();
    while (!isdigit (ch)) {
        if (ch == '-') f = 1;
        ch = getc();
    }
    while (isdigit (ch)) {
        ret = ret * 10 + ch - 48;
        ch = getc();
    }
    return f?-ret:ret;
}
```

~~其实差不多吧~~

buf数组开大一点吧，太小的话读入不够快。

注意 `isdigit` 函数需要头文件 `cctype` 。

不定参数重载版本前面介绍过方法，就不写了。

注意

本机调试前请先检查你的电脑上的 `fread` 是否能读入数据！

检查代码：

```
#include <bits/stdc++.h>
using namespace std;
char buf[10];
int main() {
    fread(buf,1,sizeof(buf),stdin);
    cout << ferror(stdin) << endl;
    return 0;
}
```

随便输入一堆字，以回车+ `Ctrl+Z` +回车结尾。如： `123456` +回车+ `Ctrl+Z` +回车。

如果输出不是零，那么你的 `fread` 出锅了。

解决方法？我也不大清楚。这个问题是我在用云端电脑进行调试的时候遇到的。应该不常见的。

~~换一台电脑就好了~~

当然也可以用gdb调。

四、输出优化

说了这么多读入优化，那么输出优化呢？

按照刚才的思路，可以用 `putchar` （头文件 `<cstdio>` ）来优化。然而单纯使用 `putchar` 来优化**还没有 `printf` 快**。

不过还是给出输出优化的代码：

```
void wrtn(int x){
    if(x<0) {putchar('-');x=-x;}
    int y=10,len=1;
    while(y<=x) {y*=10;len++;}
    while(len--){y/=10;putchar(x/y+48);x%=y;}
}
```

既然不够快，给出有什么用？因为我们可以用 `fwrite` （头文件 `<cstdio>` ）来优化 `putchar` ，这样就能比 `printf` 快了 ~~滑稽~~

代码如下：

```
char buf[1<<21],a[20];int p,p2=-1;
inline void flush() {
    fwrite (buf,1,p2+1,stdout),p2=-1;
}
inline void print(int x) {
    if (p2>1<<20) flush();
    if (x<0) buf[++p2]=45,x=-x;
    do {
        a[++p]=x%10+48;
    }while(x/=10);
    do {
        buf[++p2]=a[p];
    }while(--p);
    buf[++p2]='\n';//按需要改，不要照抄
}
```

递归实现有点慢，所以改了下。

好像while语句也可以优化一下，~~懒得写子~~ 留给读者思考。

~~然后就好像看不出是优化 `putchar` 子~~

就是把要输出的东西全部放起来，到一定量时再输出。

在上文的例子中是以 `\n` 分隔输出的。使用时要按需要改。

在程序结束之前一定要再调用一次 `flush` 函数

为什么？其实看懂这个函数的人应该都知道， `print` 函数中只有第一行（调用 `flush` 的那行）才有可能输出。

如果题目强制在线，在每个回答结束后都要调用 `flush` 。

平常的时候buf数组也是老规矩，稍微大点吧。

在使用以上代码进行调试或运行时，如果 `stdin` 没有重定向至文件，也就是在控制台进行输入时，需要在输出末尾加上回车+ `Ctrl+Z` +回车结束输入（Windows操作系统下）。

五、基于streambuf的IO优化

如果你的电脑用不了 `fread` 的话这个好像也是不行的。。。

输入输出似乎已经不能再优化了。但是笔者在查询资料时发现了一种新的优化方法：使用 `iostream` 底层的 `streambuf` 进行优化。

~~（头文件当然是 `<iostream>` ）~~

原理还是用各种函数代替 `getchar` 、 `putchar` ，看完你就会发现和 `fread` 的版本没有本质上的区别，只不过这里是用streambuf类实现的。。。

streambuf的介绍篇幅过长，这里简单讲一下。streambuf可以看作一块缓冲区，用来存储数据。想要深入了解可以参考 这位大佬的博客 (https://blog.csdn.net/man_sion/article/details/78110842) 。

为了方便区分还是把两个streambuf分别取名为inbuf和outbuf吧。

所以下文中的 `static std::streambuf *inbuf = cin.rdbuf();` 相当于获取 `cin` 的缓存地址。注意 `static` 表示这是静态变量。

几个函数稍微提一下：

- 函数 `sputc` 用法类似 `putchar` ：

```
outbuf -> sputc(const char ch);
```

对，有 sgetc 函数的。

- 函数 sgetn

类似 fread：

```
inbuf -> sgetn(char* buf, MAX_INPUT);
```

本来要用类似 fwrite 的 sputn 的，但好像没有 sputc 快？

所以给出 sputn 格式，有兴趣的可以试一下： `outbuf -> sputn(char* buf,max_output)`

经过笔者的整理和封装后测试程序代码如下：

```
#include<iostream>
#include<cctype>
using namespace std;
using std::cin;
using std::cout;
using std::endl;
namespace IN {
    const int MAX_INPUT = 1000000;
    #define getc() (p1 == p2 && (p2 = (p1 = buf) + inbuf -> sgetn(buf, MAX_INPUT), p1 == p
2) ? EOF : *p1++)
    char buf[MAX_INPUT], *p1, *p2;
    template <typename T> inline bool redi(T &x) {
        static std::streambuf *inbuf = cin.rdbuf();
        x = 0;
        register int f = 0, flag = false;
        register char ch = getc();
        while (!std::isdigit(ch)) {
            if (ch == '-') f = 1;
        ch = getc();
        }
        if (std::isdigit(ch)) x = x * 10 + ch - '0', ch = getc(),flag = true;
        while (std::isdigit(ch)) {
            x = x * 10 + ch - 48;
            ch = getc();
        }
        x = f ? -x : x ;
        return flag;
    }
    template <typename T,typename ...Args> inline bool redi(T& a,Args& ...args) {
       return redi(a) && redi(args...);
    }
    #undef getc
}

namespace OUT {
    template <typename T> inline void put(T x) {
        static std::streambuf *outbuf = cout.rdbuf();
        static char stack[21];
        static int top = 0;
        if (x < 0) {
            outbuf -> sputc('-');
            x=-x;
        }
        if (!x) {
            outbuf -> sputc('0');
            outbuf -> sputc('\n');
            return;
        }
        while (x) {
```

```
            stack[++top] = x % 10 + '0';
            x /= 10;
        }
        while (top) {
            outbuf -> sputc(stack[top]);
            -- top;
        }
        outbuf -> sputc('\n');
    }
    inline void putc (const char ch) {
        static std::streambuf *outbuf = cout.rdbuf();
        outbuf -> sputc(ch);
    }
    template <typename T> inline void put(const char ch,T x)
    {
        static std::streambuf *outbuf = cout.rdbuf();
        static char stack[21];
        static int top = 0;
        if (x < 0) {
            outbuf -> sputc('-');
            x=-x;
        }
        if (!x) {
            outbuf -> sputc('0');
            outbuf -> sputc(ch);
            return;
        }
        while (x) {
            stack[++top] = x % 10 + '0';
            x /= 10;
        }
        while (top) {
            outbuf -> sputc(stack[top]);
            --top;
        }
        outbuf -> sputc(ch);
    }
    template<typename T,typename ...Args> inline void put(T a,Args ...args) {
        put(a);put(args...);
    }
    template<typename T,typename ...Args> inline void put(const char ch,T a,Args ...args) {
        put(ch,a);put(ch,args...);
    }
}
using IN::redi;
using OUT::put;
using OUT::putc;
int main(int argc, char const *argv[])
{
    freopen("testdata.in","r",stdin);
```

```
    freopen("testdata.out","w",stdout);
    std::ios::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    int a,b;
    redi(a,b);
    put(' ',a,b);
    putc('\n');
    put('\n',a,b,a+b,a*b);
    fclose(stdin);fclose(stdout);
    return 0;
}
```

输入文件：

```
2 3
```

输出文件：

```
2 3
2
3
5
6
```

同理，MAX_INPUT稍微大一点，空间不会爆的。

在控制台进行调试时不要忘了回车+ Ctrl+Z +回车。

读优这种东西不懂的话GDB跑一遍嘛。。。

应该是比 fread 什么要快的。可能快的不明显，有些地方还可以再优化一下。

说明：

```
std::ios::sync_with_stdio(false);
cin.tie(NULL);
cout.tie(NULL);
```

这个都懂的吧，取消同步，取消绑定。用了之后快过 scanf 。但是不能再用 cstdio 了。

- `redi` 函数：

类似于 `scanf` ，只是不用加格式字符串。返回一个 `bool` 值，当且仅当所有数据读入成功时返回 `true` ，可以配合 `while` 使用。

- `putc` 函数：

用法同 `putchar` 。

- `put` 函数：

格式一：类似于 `printf` ，传入任意个变量，以 `\n` 分隔输出；

格式二：在格式一的基础上，传入一个常量字符，并以之为分隔输出。

- getc：

看到定义和上文的 `fread` 优化的应该都懂了吧。优化过的 `getchar` 。可以写成函数。

六、注意事项

关于streambuf优化：

- 类型

以上函数仅支持 `int` 、 `long long` 、 `usigned long long` 等**整形变量** 。除 `redi` 函数外其余函数均无返回值。

- 速度

如果追求速度，请去掉 `redi` 的返回值，并删掉重载过的函数。也就是只用第一个 `redi` 和第一个 `put` 。而且如果读入数据没有负数的话可以稍作修改。

- 注意函数的命名

我记得C++是有一个函数叫 `read` 的。之所以没出问题可能是因为没有用到那个库、那个命名空间，或者因为参数数量不同直接被重载了。~~这就是为什么我的函数名…~~

- MAX_INPUT

MAX_INPUT尽量大一点。

其他：

- 再次提醒

使用 `std::ios::sync_with_stdio(false);` 之后不能用 `scanf` 等 `cstdio` 里的函数。

部分代码 **需要C++ 11** 。如果没有C++11，请把所有带 `args` 的函数删除。~~换句话说，不确定NOIP能不能用。~~

- 小数据

小数据（ 10^3 个字符以内）的读入用上面的优化都有可能变慢。这个时候还是用 `scanf` 吧。

- 兼容性

除了最简单的读优和输优以外，使用读优之后会导致 `scanf` 、 `printf` 、 `getchar` 等函数不能使用

- luogu

有大佬说洛谷上计算程序的运行时间是看命令执行次数的，经过我自己的检验好像确实如此（检验方法：循环用 `register` 和不用做对比，多次测试取平均值）。不排除别的oj这样做的可能。那么这时候用任何的读优都有可能起反作用。但用cena、ccr之类的测评软件线下测评时是没有问题的。

七、关于竞赛

- C++ 11

主要是NOIP。

先看CCF官网上的资料：http://www.noi.cn/newsview.html?id=559&hash=E4E249&type=11 (http://www.noi.cn/newsview.html?id=559&hash=E4E249&type=11)。

只有一点点。。。不过可以看到C++的编译器是 GNU G++ 4.8.4。这个版本是支持C++11的。在笔者的Windows系统下不定参数重载能够通过编译（VSC编译命令：`"-g"," ${file}","-o","$ {fileBasenameNoExtension}.exe"` ）。然而你们自己编译过就知道编译器会有Warning。提示信息是：`variadic templates only available with -std=c++11 or -std=gnu++11` 。什么意思呢？就是说只有在把标准设置成C++11或以上时才能通过编译。对于CCF的Linux电脑，我一无所知。所以**不保证NOIP可以使用。如果有人因此造成的损失本人概不承担。**

当然，NOI及以上的赛事、ACM等肯定是可以的。

~~其实不用根本没关系啊~~

- 其他

基本读优、 `fread` 读优，在考场上背都背得出来。相比之下streambuf读优看起来可能稍微繁琐一点。然而只是看起来而已。其实核心函数也就两个，按需求缩减一下不见得比fread版本长多少。

不过从另一个方面讲， `fread` 读优已经很优了，在考场上完全够用（~~卡 `fread` 的出题人可以去世了~~）。事实上，最基本的读优就足以应付NOIP等大部分竞赛（~~卡 `getchar` 的出题人在想什么~~）。但我们学习信息并不是完全为了竞赛啊，开阔一下视野总是好的。

八、IO优化的弊端

IO优化很实用，然而其并非没有缺点。

基于fread的读入优化、基于streambuf的读入优化都需要一个buf数组。这是比较占内存的（~~如果你只开到5什么的话当我没说~~。而且除了最基本的读优输优以外，本文介绍的所有函数都会导致不兼容其他IO函数（输优要好一点）。不光是 `cstdio` ， `cout` 等函数都不行。遇到读入数据有字符串的情况下，还需要自己再写一个函数。

当然最基本的读优完全没有问题，几乎能代替 `scanf` 进行整数读入。但在字符和数字混用时使用可能会出错，因为它会“吃掉”一个不是数字的字符。

不过毫无疑问，总的来说，IO优化的利是大于弊的。

THE END

完结撒花(￣▽￣)~*

关于C++的IO优化的介绍就此告一段落。如果有更好的优化，或者有什么不对的地方欢迎轰炸评论区~

~~赶紧水一发A+B Problem~~

在洛谷，

享受Coding的欢乐