Memset 优化实验报告

2193211079 王嘉禾 计算机试验班 001

实验目标：

对 memset 函数进行优化，验证优化后的 memset 功能是否正确，对比优化前后 memset 函数的性能。

试验设定：

对 `char a[100001000]` 进行 memset 操作。

直接用朴素的遍历方式对 memset 函数的正确性进行验证，见如下的 `bool check(char c)` 函数，其功能为验证静态全局数组 a 中的元素是否全部设置为 c。

```
bool check(char c)
{
    bool flag=true;
    for(int i=0;i<sizeof(a);++i)
    {
        if(a[i]!=c)
        {
            flag=false;
            break;
         }
     }
     return flag;
}
```

实验步骤：

1. 编写普通的 memset_naive 函数和优化后的 memset_op1 和 memset_op2 函数，对于优化后的 memset 函数，只需在遍历数组并填充时将数组名视为不同类型的类型即可，例如在 4*1 优化中，首先将每个字节填充的 char 值拼接为 4 个字节的 int，遍历时将 a 视为整形指针，则每次填充会填入 4 个字节的值。对于 8*1 优化类似，每次填入 8 个字节的值。
2. 对于每个 memset 函数，调用两次，将 a 数组设为不同的值，第二次用于计时测试，第一次用于控制变量，防止类似 cold miss 的情况。
3. 调用 check 函数，检查 memset 函数功能的正确性。
4. 输出记录的 cpu 时间，比较几个 memset 函数的性能。

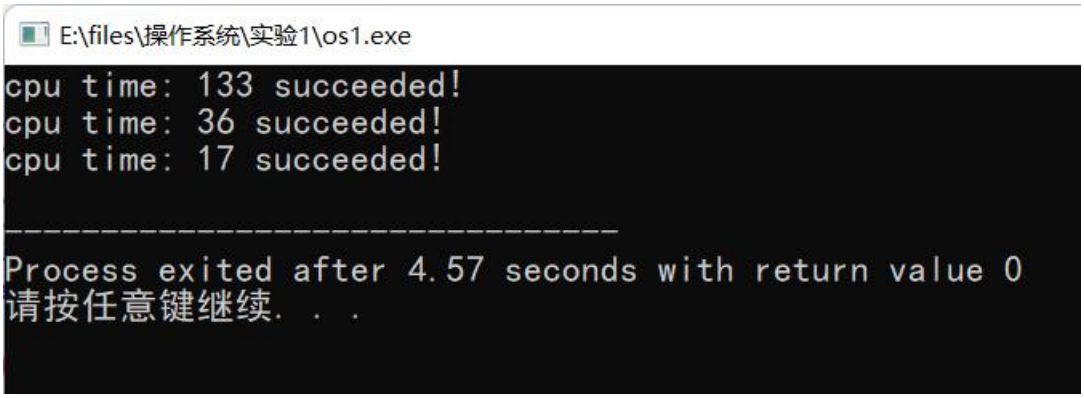
代码实现：

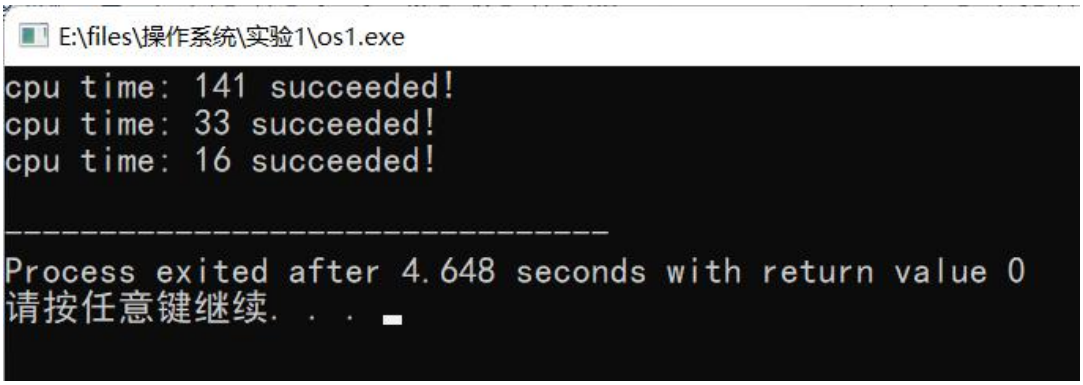
```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <ctime>
using namespace std;
static char a[100001000];
void *memset_naive(void* s,int c,size_t n)
{
    for(int i=0;i<n;++i) ((char*)s)[i]=(unsigned char)(c);
}
```

```
11. void *memset_op1(void* s,int c,size_t n)
12. {
13.     char c1=(unsigned char)(c);
14.     unsigned int num_to_fill=(c1<<24)+(c1<<16)+(c1<<8)+c1;
15.     long long i=0;
16.     for(;i<=n/4;i++) ((int*)s)[i]=num_to_fill;
17.     i*=4;
18.     for(;i<n;++i) ((char*)s)[i]=c1;
19. }
20. void *memset_op2(void* s,int c,size_t n)
21. {
22.     char c1=(unsigned char)(c);
23.     unsigned long long num_to_fill=((unsigned long long)c1<<56)+((unsigned
     long long)c1<<48)+((unsigned long long)c1<<40)+((unsigned long long)c1<<3
    2)+(c1<<24)+(c1<<16)+(c1<<8)+c1;
24.     long long i=0;
25.     for(;i<=n/8;i++) ((long long*)s)[i]=num_to_fill;
26.     i*=8;
27.     for(;i<n;++i) ((char*)s)[i]=c1;
28. }
29. bool check(char c)
30. {
31.     bool flag=true;
32.     for(int i=0;i<sizeof(a);++i)
33.     {
34.         if(a[i]!=c)
35.         {
36.             flag=false;
37.             break;
38.         }
39.     }
40.     return flag;
41. }
42. int main()
43. {
44.     clock_t time;
45.
46.     memset_naive(a,'1',sizeof(a));
47.     time=clock();
48.     memset_naive(a,'2',sizeof(a));
49.     cout << "cpu time: " << clock()-time << " ";
50.     cout << (check('2')?"succeeded!":"failed!") << endl;
51.
52.     memset_op1(a,'1',sizeof(a));
```

```
53.     time=clock();
54.     memset_op1(a,'3',sizeof(a));
55.     cout << "cpu time: " << clock()-time << " ";
56.     cout << (check('3')?"succeeded!":"failed!") << endl;
57.
58.     memset_op2(a,'1',sizeof(a));
59.     time=clock();
60.     memset_op2(a,'4',sizeof(a));
61.     cout << "cpu time: " << clock()-time << " ";
62.     cout << (check('4')?"succeeded!":"failed!") << endl;
63.     return 0;
64. }
```

测试结果：





多次测试后每次调用的 cpu 时间波动幅度不超过 10%左右。可见，采用 4*1 优化和 8*1 优化时，同样成功将 a 数组全部赋值时，cpu 时间缩短了 4 倍和 8 倍。这是因为在 memset_naive 函数中，每次只填充 1 个字节的长度，假设需要填充的空间为 n 个字节，则需要填充 n 次。利用循环展开的方式，每次填充 k 个字节，只需要填充⌊ xn/k⌋次，并处理最后多出的几个字节即可。这种方法可以减少循环次数，最终可以降低程序的 CPE，提高程序运行速度。

实验感想：

本次实验中总体没遇到什么困难，唯一花费时间 debug 的地方是指针的迭代出现错误（用(int*)指针遍历时每次只需+1 就可以读四个字节，第一次实现时错误写成了+4），但很快通过查看 memset 的结果修改成功。本次实验的收获主要是加深了对代码优化的理解和掌握，提升编程时考虑代码性能和注意优化的意识，使我获益匪浅。