《面向对象程序设计》实验报告

姓名：龙永奇
学院：计算机科学与技术学院
专业：计算机科学与技术
学号：3220105907
邮箱：3220105907@zju.edu.cn

报告日期：2024年6月25日

1 实验目的

本实验的目的是实现一个自定义的内存池模块用于 Allocator，减少内存分配和释放的开销，从而提高程序的性能，同时与 C++ 标准的 STL allocator 进行性能比较

1. **内存池实现**：实现简单的内存池类 MemoryPool，预先分配内存，并管理多个内存块
2. **Allocator 实现**：使用自定义内存池实现带有内存缓冲池的 Allocator
3. **性能测试**：比较自定义内存池和与 C++ 标准的 STL allocator 向 `vector` 中分配和释放内存时的性能差异

2 实验环境

- 系统：Windows 11 企业版
- 编译器：tdm64-gcc-10.3.0-2
- IDE：Visual Stdio Code

3 实验步骤

3.1 Memory Pool 实现

该类用于管理预先分配的一块连续内存，并将其作为内存池。

在构造函数中，通过 `operator new` 分配足够大小的内存块，并将每个块的地址存储在 `m_freeBlocks` 中。`allocate` 函数用于分配内存块，从 `m_freeBlocks` 中取出一个可用的块，并返回其地址。`deallocate` 函数将不用的内存块地址加回到 `m_freeBlocks` 中，用于下次分配使用。

运行过程：

创建 `MemoryPool` 对象时，指定每个内存块的大小和总数。内存池在构造函数中分配了一块大小为 `m_blockSize * m_blockCount` 的连续内存。每个内存块的地址存储在 `m_freeBlocks` 向量中，表示它们当前可用。然后通过 `allocate` 和 `deallocate` 函数进行动态管理

类的实现如下：

```
class MemoryPool {
public:
    MemoryPool(size_t blockSize, size_t blockCount) : m_blockSize(blockSize),
m_blockCount(blockCount) {
        m_pool = operator new(m_blockSize * m_blockCount);
        for (size_t i = 0; i < m_blockCount; ++i) {
            m_freeBlocks.push_back(static_cast<char*>(m_pool) + i * m_blockSize);
        }
    }

    ~MemoryPool() {
        operator delete(m_pool);
    }

    void* allocate() {
        if (m_freeBlocks.empty()) {
            throw bad_alloc();
        }
        void* block = m_freeBlocks.back();
```

```
        m_freeBlocks.pop_back();
        return block;
    }

    void deallocate(void* block) {
        m_freeBlocks.push_back(static_cast<char*>(block));
    }

private:
    size_t m_blockSize;
    size_t m_blockCount;
    void* m_pool;
    vector<void*> m_freeBlocks;
};
```

3.2 Allocator 实现

`Allocator` 用于分配和释放块

通过 `allocate` 函数用于分配 `n` 个对象的内存空间。`deallocate` 函数用于释放由 `allocate` 分配的内存空间。如果内存池存在且 `n` 为 1，则从内存池中分配；否则，调用全局的 `operator new`。

`construct` 和 `destroy` 函数用于在已分配的内存上构造和销毁对象。`rebind` 结构体定义了 Allocator 的重新绑定机制，用于支持不同类型的 Allocator。

`operator==` 和 `operator!=` 用于比较两个 Allocator 是否相等，主要比较其内部的内存池指针是否相同。

运行过程:

1. 创建 `MyAllocator` 对象时，可以选择提供一个 `MemoryPool` 对象，或者使用默认构造函数，此时内存池为空指针。
2. 调用 `allocate` 函数时，根据传入的数量 `n` 和内存池的存在性，决定是从内存池还是全局分配内存。调用 `deallocate` 函数时，根据先前分配的方式，将内存块归还给内存池或者使用全局的方式释放内存。

类的实现如下:

```
template <typename T>
class MyAllocator {
public:
    using value_type = T;

    MyAllocator() : m_pool(nullptr) {}

    MyAllocator(MemoryPool& pool) : m_pool(&pool) {}

    template <typename U>
    MyAllocator(const MyAllocator<U>& other) noexcept : m_pool(other.m_pool) {}

    T* allocate(size_t n) {
        if (n > numeric_limits<size_t>::max() / sizeof(T)) {
            throw bad_alloc();
        }
        if (m_pool && n == 1) {
            return static_cast<T*>(m_pool->allocate());
```

```
        } else {
            return static_cast<T*>(::operator new(n * sizeof(T)));
        }
    }

    void deallocate(T* p, size_t n) noexcept {
        if (m_pool && n == 1) {
            m_pool->deallocate(p);
        } else {
            operator delete(p);
        }
    }

    template <typename U, typename ...Args>
    void construct(U* p, Args&&... args) {
        new(p) U(forward<Args>(args)...);
    }

    template <typename U>
    void destroy(U* p) {
        p->~U();
    }

    template <typename U>
    struct rebind {
        using other = MyAllocator<U>;
    };

    MemoryPool* m_pool;
};

template <typename T, typename U>
bool operator==(const MyAllocator<T>& lhs, const MyAllocator<U>& rhs) noexcept {
    return lhs.m_pool == rhs.m_pool;
}

template <typename T, typename U>
bool operator!=(const MyAllocator<T>& lhs, const MyAllocator<U>& rhs) noexcept {
    return !(lhs == rhs);
}
```

3.3 性能测试

通过以下代码对 Memory Pool Allocator 向 `vector` 中分配和释放内存时的性能进行测试：

```
MemoryPool pool(sizeof(int), TestSize * 50);
random_device rd;
mt19937 gen(rd());
uniform_int_distribution<> dis(1, TestSize);

// Measure time for custom allocator
auto start_custom = high_resolution_clock::now();

// Vector creation
using IntVec = vector<int, MyAllocator<int>>;
```

```
vector<IntVec, MyAllocator<IntVec>> vecints(TestSize, MyAllocator<IntVec>(pool));
for (int i = 0; i < TestSize; i++)
    vecints[i].resize(dis(gen));

using PointVec = vector<Point2D, MyAllocator<Point2D>>;
vector<PointVec, MyAllocator<PointVec>> vecpts(TestSize, MyAllocator<PointVec>
(pool));
for (int i = 0; i < TestSize; i++)
    vecpts[i].resize(dis(gen));

// Vector resize
for (int i = 0; i < PickSize; i++) {
    int idx = dis(gen) - 1;
    int size = dis(gen);
    vecints[idx].resize(size);
    vecpts[idx].resize(size);
}

// Vector element assignment
{
int val = 10;
int idx1 = dis(gen) - 1;
int idx2 = vecints[idx1].size() / 2;
vecints[idx1][idx2] = val;
if (vecints[idx1][idx2] == val)
    cout << "correct assignment in vecints: " << idx1 << endl;
else
    cout << "incorrect assignment in vecints: " << idx1 << endl;
}
{
Point2D val(11, 15);
int idx1 = dis(gen) - 1;
int idx2 = vecpts[idx1].size() / 2;
vecpts[idx1][idx2] = val;
if (vecpts[idx1][idx2] == val)
    cout << "correct assignment in vecpts: " << idx1 << endl;
else
    cout << "incorrect assignment in vecpts: " << idx1 << endl;
}

auto end_custom = high_resolution_clock::now();
auto duration_custom = duration_cast<milliseconds>(end_custom -
start_custom).count();

cout << "Memory pool allocator time: " << duration_custom << " ms" << endl;
```

通过以下代码对 STL 中 allocator 向 `vector` 中分配和释放内存时的性能进行测试：

```
// Measure time for allocator
auto start = high_resolution_clock::now();

// Vector creation
using StdIntVec = vector<int>;
vector<StdIntVec> std_vecints(TestSize);
for (int i = 0; i < TestSize; i++)
```

```
    std_vecints[i].resize(dis(gen));

using StdPointVec = vector<Point2D>;
vector<StdPointVec> std_vecpts(TestSize);
for (int i = 0; i < TestSize; i++)
    std_vecpts[i].resize(dis(gen));

// Vector resize
for (int i = 0; i < PickSize; i++) {
    int idx = dis(gen) - 1;
    int size = dis(gen);
    std_vecints[idx].resize(size);
    std_vecpts[idx].resize(size);
}
// Vector element assignment
{
    int val = 10;
    int idx1 = dis(gen) - 1;
    int idx2 = std_vecints[idx1].size() / 2;
    std_vecints[idx1][idx2] = val;
    if (std_vecints[idx1][idx2] == val)
        cout << "correct assignment in std_vecints: " << idx1 << endl;
    else
    cout << "incorrect assignment in std_vecints: " << idx1 << endl;
}
{
    Point2D val(11, 15);
    int idx1 = dis(gen) - 1;
    int idx2 = std_vecpts[idx1].size() / 2;
    std_vecpts[idx1][idx2] = val;
    if (std_vecpts[idx1][idx2] == val)
        cout << "correct assignment in std_vecpts: " << idx1 << endl;
    else
        cout << "incorrect assignment in std_vecpts: " << idx1 << endl;
}

auto end = high_resolution_clock::now();
auto duration = duration_cast<milliseconds>(end - start).count();

cout << "allocator time: " << duration << " ms" << endl;
```

测试结果如下:

```
PS D:\Code\Objects\OOP\Project7\code> cd "d:\Code\Objects\OOP\Project7\code\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
correct assignment in vecints: 5769
correct assignment in vecpts: 715
Memory pool allocator time: 660 ms
correct assignment in std_vecints: 9341
correct assignment in std_vecpts: 7569
std::allocator time: 830 ms
PS D:\Code\Objects\OOP\Project7\code> cd "d:\Code\Objects\OOP\Project7\code\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
correct assignment in vecints: 8453
correct assignment in vecpts: 6146
Memory pool allocator time: 675 ms
correct assignment in std_vecints: 3152
correct assignment in std_vecpts: 5398
std::allocator time: 836 ms
PS D:\Code\Objects\OOP\Project7\code> cd "d:\Code\Objects\OOP\Project7\code\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
correct assignment in vecints: 1901
correct assignment in vecpts: 7164
Memory pool allocator time: 836 ms
correct assignment in std_vecints: 9897
correct assignment in std_vecpts: 2960
std::allocator time: 935 ms
PS D:\Code\Objects\OOP\Project7\code> cd "d:\Code\Objects\OOP\Project7\code\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
correct assignment in vecints: 8267
correct assignment in vecpts: 9159
Memory pool allocator time: 802 ms
correct assignment in std_vecints: 9430
correct assignment in std_vecpts: 6963
std::allocator time: 980 ms
```

```
PS D:\Code\Objects\OOP\Project7\code> cd "d:\Code\Objects\OOP\Project7\code\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
correct assignment in vecints: 7642
correct assignment in vecpts: 3366
Memory pool allocator time: 599 ms
correct assignment in std_vecints: 8569
correct assignment in std_vecpts: 8075
std::allocator time: 737 ms
PS D:\Code\Objects\OOP\Project7\code> cd "d:\Code\Objects\OOP\Project7\code\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
correct assignment in vecints: 6457
correct assignment in vecpts: 7
Memory pool allocator time: 850 ms
correct assignment in std_vecints: 9282
correct assignment in std_vecpts: 3693
std::allocator time: 829 ms
PS D:\Code\Objects\OOP\Project7\code> cd "d:\Code\Objects\OOP\Project7\code\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
correct assignment in vecints: 3234
correct assignment in vecpts: 3454
Memory pool allocator time: 812 ms
correct assignment in std_vecints: 1219
correct assignment in std_vecpts: 6009
std::allocator time: 777 ms
PS D:\Code\Objects\OOP\Project7\code> cd "d:\Code\Objects\OOP\Project7\code\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
correct assignment in vecints: 2818
correct assignment in vecpts: 6603
Memory pool allocator time: 598 ms
correct assignment in std_vecints: 8293
correct assignment in std_vecpts: 2852
std::allocator time: 820 ms
```

测试集大小：

- Test Size = 10000
- Pick Size = 1000

平均测试时间：

- MemoryPool Allocator：735.875 ms
- STL Allocator：843.125ms

实验总结

本实验实现了一个简单的内存池（MemoryPool）并用于 Memory Pool Allocator，同时也是本学期 OOP 的最后一次实验。在实现的过程中，学习了解了 Cpp 的内存分配机制。

测试结果表明，自定义内存池分配器在大量小对象分配的情况下比标准 C++ 分配器相对的性能提升。

优化方向

1. **多级内存池**：针对不同大小的内存块使用不同的内存池，减少内存浪费
2. **线程本地存储**：在多线程环境下，为每个线程分配独立的内存池，减少锁竞争
3. **内存对齐**：确保分配的内存块对齐到缓存行，提高缓存命中率