Malloclab实验指导

ICS2-24Spring

实验目标

模仿实现一个动态内存分配器（即自己实现库函数**malloc、free和realloc**)。
你需要修改下发文件中的**mm.c**的相关代码，最终完成一个正确且高效的分配器。

实验步骤

1. 登录课程平台**obe.ruc.edu.cn**
2. 下载本次实验的相关材料，将**malloclab-handout**文件夹上传至服务器或虚拟机
3. 按要求完成作业

下面是具体的说明

你的分配器

你的分配器将由以下四个函数组成（mm.*中声明并定义）：

```
int mm_init(void);
void * mm_malloc(size_t size);
void mm_free(void * ptr);
void * mm_realloc(void * ptr, size_t size);
```

我们将给你提供一个简单实现的**mm.c**，你可以以此为基础修改。每一个函数应当完成的工作如下：

- **mm_init**：用来评估你的分配器的进程预先进行初始化，例如分配初始堆空间。如果初始化过程出现问题，返回值应当设为-1，否则设置为0
- **mm_malloc**：类似*libc*中的*malloc*，返回一个指向大小至少为**size**的已分配块的指针，且该指针应当是8字节对齐的。也就是说*ret & 0x7 = 0*
- **mm_free**：释放由*mm_malloc*或者*mm_realloc*分配的空间
- **mm_realloc**
  - 如果**ptr==NULL**，等价于**mm_malloc(size)**
  - 如果**size==0**，等价于**mm_free(ptr)**
  - 否则，返回指向新块的指针，新块的前min($size_{new}, size_{old}$)字节应当与旧的块保持一致，剩下的内容保持未初始化状态。注意，新块和旧块的地址可以一样，这取决于你的实现。

你可以在shell中输入下面的命令获得完整的libc相关内容文档。（用处不大）

```
man malloc
```

模拟内存系统

memlib.c中定义了一个模拟内存系统，你可以调用其中的函数。

```
void * mem_sbrk(int incr); // 使堆增加incr字节，返回新分配堆区域的第一个字节地址
void * mem_heap_lo(void); // 返回堆的第一个字节地址
void * mem_heap_hi(void); // 返回堆的最后字节地址
size_t mem_heapsize(void); // 返回当前堆的大小
size_t mem_pagesize(void); // 返回系统的Page大小， Linux上为4096
```

自动化测试系统

mdriver程序测试你的分配器的相关指标。我们使用同样的**trace**和**mdriver**对你的分配器进行评测。在**make**后，可以使用**./mdriver -h**查看测试系统使用方式。

实验要求

代码规范

- 你不能更改任何已定义的函数原型。
- 你不能进行任何内存管理相关的库调用或系统调用。
- 在**mm.c**中，不允许定义任何全局或静态的复合数据结构（如数组、结构体、树、链表）。但是你可以定义全局的scalar变量，包括整数、浮点数和指针
- 注意对齐问题。

提交

你需要在规定的时间前提交实验报告和代码，按照下面的格式命名：

```
学号-malloc
|-- report.pdf
|-- mm.c
```

然后将这个文件夹打包成**学号-malloc.zip**提交至OBE平台。我们会使用全自动的批改工具测试正确性和性能分，不按照这个格式提交的作业将无法被正常识别。
在截止日期后k天提交的实验，其成绩为：

$$\text{Grade} = \text{Grade}_{\text{original}} \cdot \max\left(0.6, 0.6 + 0.4 \cdot \frac{14-k}{14}\right)$$

评分标准

- 代码（70%）
  - 正确性
  - 性能
  - 代码风格
- 报告（30%）

性能评分根据下面的指标：

- 空间利用率$\mathcal{U}$：使用的内存总量（包括分配但未释放的内存）与使用的堆大小的比值的最大值，需要使其尽可能接近1。

- 吞吐量$\mathcal{T}$：平均每秒钟完成的操作数。
  使用下面的公式计算性能指标 $\mathcal{E}$：

$$\mathcal{E} = w\mathcal{U} + (1-w)\min\left(1, \frac{\mathcal{T}}{\mathcal{T}_{\text{libc}}}\right)$$

其中, w 是一个可能会根据班级情况适当调整的权重, 默认值为0.6。

一堆提示们

堆一致性检查器

你可以编写一个堆一致性检查器，帮助你实现这个lab，其可以对以下方面进行检查：

- 空闲链表中的块都标记为空闲了吗?
- 是否有连续的空闲块未被合并?
- 每一个空闲块都在空闲链表中吗?
- 空闲链表的指针指向的都是有效的空闲块吗?
- 有相交的或者大小不对的分配块吗?
- 返回的指针是有效的吗?
- ……

你的堆检查器会由 mm.c 中的**int mm_check(void)** 组成，当且仅当你的堆满足一致性时返回非0值。
你不需要局限于上面的建议，也不需要完全检查这些建议。鼓励在**mm_check**失败时打印错误信息。
当你提交 **mm.c** 时，请移除所有 **mm_check** 的调用，因为这会影响到你的吞吐量。

其他温馨的提示

- 可以使用调试器GDB
- 理解课本上的malloc实现的每一行
- 可以使用宏定义处理你的指针算术操作
- **realloc**可以调用已有代码，但是为了得到良好的性能，建议重新实现
- 可以参考英文指导**malloclab-instructions.pdf**
- 不要在精神状态不好的时候Debug，多走出去看看春天 😀

特别重要的提示

- **严禁抄袭，网上有很多malloclab的实现代码，你可以找到各种师兄师姐，甚至可能我们几个助教的代码。但是本次实验会采用非常严格的查重方式，除非你觉得你的检索能力比我强，否则还是别抄网上的代码**
- **查重不仅会和网上代码比较，也会在你们之间比较，如有抄袭者，本次实验0分，其他所有实验按不及格处理**
- **如有明显参考，请在实验报告里指出，不会被当做抄袭**
- **助教很仁慈，不会刻意为难你们，分都给的高高的**
- **尽早开始！**