

Malloc Lab 实验讲解

张学峰

2023 年 12 月 6 日

实验要求

- 编写一个动态内存分配的 C 语言程序
- 实现动态内存分配、释放、重新分配等功能
- 单人独立完成作业

实验方法

- 从网络学堂获取 `malloclab-handout.tar`
- 解压缩: `unzip malloclab-handout.zip`
- 仅需要且仅允许修改 `mm.c` 文件
- 通过 `mdriver` (文件为 `mdriver.c`) 进行评测
- 运行方法:
 - ▶ `make`
 - ▶ `./mdriver -V`

提交要求

- 在 mm.c 中的 Team 结构体中填充自己的姓名和学号
- 在实验截止日期前将 mm.c 放置到服务器上 ~/mm.c 路径
- 将实验报告提交至网络学堂，命名为学号.pdf（如 2021000000.pdf）

动态存储分配器的实现

- 需要完成以下 4 个函数（声明位于 mm.h，定义位于 mm.c）
 - ▶ `int mm_init(void);`
 - ▶ `void *mm_malloc(size_t size);`
 - ▶ `void mm_free(void *ptr);`
 - ▶ `void *mm_realloc(void *ptr, size_t size);`
- mm.c 中包含动态存储分配器的最简单的实现，可以跑过一部分测试，大家需要修改相关函数的实现。

mm_init

- 接口: `int mm_init(void);`
- 如果执行初始化时出现问题返回 -1, 如果正常结束返回 0。
- mdriver 在每次测试新 trace 前调用该函数进行初始化。
- 不要在此函数中调用 memlib 中的 `mem_init`。

mm_malloc

- 接口: `void *mm_malloc(size_t size);`
- 返回一个指向至少为 `size` 字节的连续内存块的指针。
- 分配的内存块应完全位于堆内, 且不与任何其他已分配的块重叠。
- 若无法分配, 需调用 `void *mem_sbrk(int incr)` 申请一块新的堆区域。
- 在 64 位的环境下进行实验, 故根据 C 语言标准, 应对齐到 **16 字节**。

mm_free

- 接口: `void mm_free(void *ptr);`
- 释放 `ptr` 指向的内存块, 无返回值。若 `ptr` 为 `NULL`, 该调用不进行任何操作。
- 保证 `ptr` 为之前 `malloc` 或 `realloc` 返回的指针, 且未被释放。

mm_realloc

- 接口: `void *mm_realloc(void *ptr, size_t size);`
- 返回一个指向至少为 `size` 字节的连续内存块的指针。有如下要求:
 - ▶ 如果 `ptr` 为 `NULL`, 相当于调用 `mm_malloc(size)`
 - ▶ 如果 `size` 为 0, 相当于调用 `mm_free(ptr)`
 - ▶ 如果 `ptr` 不为 `NULL`, 一定为之前 `malloc` 或 `realloc` 返回的指针, 且未被释放。该调用将 `ptr` 指向的内存块 (旧块) 的大小更改为 `size` 字节并返回新块的地址。新块的地址可以与旧块相同, 也可以不同。
 - ▶ 新块需复制旧块的内容, 大小由新旧两块的较小者决定, 例如:
 - ★ 旧块 8 字节, 新块 12 字节, 则复制前 8 字节
 - ★ 旧块 8 字节, 新块 4 字节, 则复制前 4 字节

支撑函数

- memlib.c 包为动态内存分配器模拟内存系统。你可以调用其中的以下函数：
 - ▶ `void *mem_sbrk(int incr)`: 扩展 `incr` 字节的堆，其中 `incr` 是一个正整数。该函数返回一个通用指针，指向新分配的堆的第一个字节。语义与 Unix 的 `sbrk` 相同，只是 `mem_sbrk` 只接受正的参数。
 - ▶ `void *mem_heap_lo(void)`: 返回指向堆中第一个字节的通用指针
 - ▶ `void *mem_heap_hi(void)`: 返回指向堆中最后一个字节的通用指针
 - ▶ `size_t mem_heapsize(void)`: 返回堆的当前大小，单位为字节
 - ▶ `size_t mem_pagesize(void)`: 返回系统的页大小，单位为字节。Linux 为 4K。

- 用于测试 mm.c 中各函数的正确性、空间利用率和吞吐量
- 测试的 trace 位于 traces 文件夹中
- 使用方式可通过 ./mdriver -h 查看。其中 -V 可用于定位报错出现的文件，-f 可用于指定 trace 进行测试。

编程规则

- 不允许改变 mm.c 的接口函数
- 不允许调用系统的库函数
- 不允许在 mm.c 程序中定义全局或静态的复合数据结构，如数组、结构、树或列表。但是可以在 mm.c 中声明全局标量变量，如整数、浮点数和指针。
- 返回的内存块应 16 字节对齐

评分标准

- 正确性：10 分
- 性能：50 分
 - ▶ 空间利用率：程序使用的最大内存量与分配程序使用的堆大小之间的比率，最佳比率为 1。
 - ▶ 吞吐量：Kops (kilo operations per second)
 - ▶ 评分公式：

$$P = wU + (1 - w) \min(1, \frac{T}{T_{libc}})$$

U 为空间利用率，T 为吞吐量， T_{libc} 是助教在课程集群上测试的 libc malloc 的吞吐量，具体值以 config.h 的 AVG_LIBC_THRUPUT 为准。
 $w = 0.6$ ，需要均衡地考虑内存利用率和吞吐量的优化。

- ▶ 代码风格：10 分
- ▶ 实验报告：30 分

一些建议

- 利用 `mdriver -f` 使用小文件简化调试，如 `traces/short{1,2}-bal.rep`
- 利用 `-v` 和 `-V` 查看详细输出
- 详细了解书中 `malloc` 实现的每一行代码。该示例实现了一个基于隐式自由列表的简单分配器。
- 将指针算术封装在 C 的预处理器宏 (`#define`) 中，可以显著降低复杂性。
- 前 9 条 trace 仅包括 `malloc` 和 `free`，后两条包含 `malloc`, `free` 和 `realloc`。建议在 `malloc` 和 `free` 能够在前 9 条 trace 上正常工作后再调试 `realloc`。
- `realloc` 可以构建在 `malloc` 和 `free` 之上，但要获得非常好的性能，需要单独进行设计。

Q&A