Malloc Lab

目标

本次实验的任务是完成一个 C 版本的动态内存分配管理程序，要求支持 *malloc, free* 和 *realloc* 操作且正确、尽可能的快速、高利用率。

要求

- 不要抄袭，可以与同学讨论，但不要直接抄袭同学的代码和实验报告。
- 请认真完成实验报告，重点讲解内存管理结构及算法、遇到的问题及解决方案、采用的优化手段及其他你认为重要的内容。

  > 算法和优化方法必须包括但不局限于最后提交的版本所采用的，其他效果较差的方法和已经写好但截止日期之前没有调试好的方法也可以写入实验报告。

- 请在截止时间（2020年 4月 5日 23:55）前将实验报告提交至 obe，不需要提交代码。

  > 为了方便助教批改，请以 pdf 形式提交实验报告，不要提交压缩包或 word。

指导

1. 基本步骤

- 登录服务器
  - 服务器地址 192.144.229.231
  - 连接服务器常用方式见《课前准备指南》。
- 实验代码已经分发至学生用户的主目录，使用 `tar -xvf malloclab-handout.tar` 命令解压实验材料。
- 仔细阅读 README 文件。malloclab-handout 目录下主要包括以下主要文件：
  - `mm.c` 需要完成的代码，其中已经完成了最初始版本的内存分配算法。正常情况下，只需要修改这个文件。
  - `mdriver.c` 测试驱动程序的源码，使用 make 命令编译后生成测试驱动程序。
  - `short{1,2}-bal.rep` 两个较小的测试样例，方便实验过程中测试正确性。
  - `Makefile` make 命令对应的脚本文件。
  - `config.h` 本实验的配置文件，由于评测时使用的是 `config.h` 的初始版本，为了自己的分数，**请不要对这个文件进行任何修改！**
  - `others` 其他测试框架中的源码文件。
- 填写 mm.c 中的信息。**本次实验不支持组队完成，队伍信息只需要填写前三项。前三项未填写完整会导致测试不通过。**
- 完成并测试动态内存分配管理程序。
- 提交代码
  - 按照要求修改并填写 `Makefile` 文件中的 *STUID*。
  - 确认无误后使用 `make handin` 命令提交代码。提交程序只会提交 `mm.c` 文件，因此为了保证程序在统一测试时程序的表现与实验时一致，***请不要修改除 `mm.c` 之外的任何源代码文***

件!

- 如果需要再次提交，需要将 `Makefile` 中的 *VERSION* 增加 1。统一测试时只以最大版本号为准。**每人最多可提交两次，提交次数大于两次后每多一次提交代码分数都会在基础得分上扣除百分之五，即提交两次之后代码得分会乘以系数** $(100 - 5 \times (Version - 2))\%$

2. 需要完成的函数

动态内存分配管理程序应该包含以下四个函数，它们在 `mm.h` 文件中被声明：

```
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` 文件中已经实现了最原始但是功能正确的版本。可以以此为基础修改函数，也可以从头开始并且自定义其他需要的函数。这四个函数的具体意义如下：

- `int mm_init(void)`

  这个函数会在其他三个函数被执行前调用，需要在其中实现必要的初始化。如果在初始化过程中出现问题，应返回 1，否则应该返回 0。

- `void *mm_malloc(size_t size)`

  这个函数返回一个指向分配好的至少 *size* 字节大小的可用内存块，整个块必须在堆区域内，且不能与其他已分配块重叠，如果不能分配则返回 *NULL*。

- `void mm_free(void *ptr)`

  释放 *ptr* 指针指向的内存块，无返回值。只需要保证传入的参数 *ptr* 是之前调用的 `mm_malloc` 或 `mm_realloc` 函数的返回值且未被释放过时，该函数可以正常工作即可。（不需要检查错误）

- `void *mm_realloc(void *ptr, size_t size)`

  这个函数返回一个指向重新分配的至少 *size* 字节大小的可用内存块。

  - 如果 *ptr* 参数为 *NULL*，函数等价于 `mm_malloc(size)`；
  - 如果 *size* 参数为 0，函数等价于 `mm_free(ptr)`；
  - 如果 *ptr* 参数不为空，则它必须是之前调用的 `mm_malloc` 或 `mm_realloc` 函数的返回值，且不曾被释放过。函数需要更改分配过的内存块的大小，可以原地修改内存块（*ptr* 不变），或者重新寻找可用的地址（*ptr* 改变），这取决于同学们自己的实现。但注意，新内存块的内容应保持与原内存块相同。 如旧内存块 8 字节，新内存块 12 字节，则新内存块的前 8 个字节应与旧内存块完全一致；如旧内存块 8 字节，新内存块 4 字节，则新内存块应与旧内存块的前 4 字节完全一致。

这些函数的规则与 C 标准库的对应函数规则完全相同，如果还存在疑问，可通过 `man malloc` 命令查阅官方文档。不了解 *man page* 的同学建议阅读 OSTEP 原书 44 页 **ASIDE: *RFTM — READ THE MAN PAGE*** 部分（会心一笑😁）。

3. 关于 Debug

- 助教不会回答任何关于具体程序实现的问题，**包括 Debug!**
- 当使用较复杂的算法时，动态内存分配管理程序很容易出错，又由于**助教不帮助 Debug**，我们提供以下几个建议：
  - 在自己认为可能出错的地方打印必要的信息，包括但不限于：内存内容、指针地址。

- 使用 gdb 在自己程序对应位置设置断点进行调试，注意程序的入口是 `mdriver` 而不是 `mm`。
- 自己实现一个 Heap Checker 来帮助 Debug。Checker 可检查以下内容：
  - 每个 free list 中的内存块是否真正被释放了?
  - 相邻的内存块是否有重叠?
  - 每个被释放的内存块是否当前可用?
  - 其他可能存在问题的部分。

  可以把自己的 Heap Checker 定义在 `mm.c` 文件中的 `int mm_check(void)` 函数中，自定义返回值和输出，在希望检查的时候调用，并根据错误信息修改程序。

  此外，Checker 应只存在于开发过程中，测试和提交程序之前**请注释或删除检查部分，防止程序性能变差。**

以上调试方法仅为助教建议，可以选择任意自己喜欢的方式 debug（除了把错误程序交给助教），**如果足够强，你甚至不需要 debug。**

4. 一些需要用到的函数

动态内存分配函数管理的是一段连续的内存区间，我们称之为堆，它最大是 20MB（定义在`config.h` 中），应用程序向内存分配程序申请内存，程序根据需求对堆进行操作，为了简化学生实验过程，方便助教评测，我们提供了一些关于堆的操作函数（定义在 `memlib.c` 中）。

- `void *mem_sbrk(int incr)`

  将堆扩展 *incr* 字节，其中 *incr* 是一个正整数。正常情况下返回值是一个指向扩展部分第一个字节的指针，如果增加后将超出堆的最大值，则不进行扩展，并返回 -1。 注意，在程序开始的时候，管理的内存大小是 0 字节，需要先向堆申请内存，然后才能进行管理并分配给应用程序。*incr* 必然是正整数，意味着所使用的内存大小不能变小，而最后测评程序将根据程序所申请的堆的大小来计算程序的空间利用率。
- `void *mem_heap_lo(void)`

  返回申请的堆中的第一个字节的指针。
- `void*mem_heap_lo(void)`
  返回申请的堆中的最后一个字节的指针。
- `size_t mem_heapsize(void)`
  返回当前申请的堆的大小。
- `size_t mem_pagesize(void)`
  返回系统内存页的大小（Linux 系统是 4K）。

5. 测试程序

当输入 `make` 指令时，代码将被整合进测试驱动程序，驱动程序将根据测试数据测试程序。下面介绍 `mdrvier` 的参数和使用。

使用：在 `make` 成功之后直接使用 `./mdriver` 命令进行测试。如果有其他需求请参见下面的参数部分。

参数：

- `-h` 打印帮助信息。
- `-t <tracedir>` 在指定目录而不是 config.h 定义的目录中寻找测试数据。
- `-f <tracefile>` 使用特定的一个测试数据测试程序，默认是将测试数据集全部测试。

- `-l` 除了测试代码外额外调用 C 标准库接口并测试。
- `-v` 详细的输出模式，输出程序对于每一个测试文件的表现。
- `-V` 更详细的输出模式，对于每个测试文件输出额外的诊断信息，用于判断哪个文件使程序出错。

测试使用的 trace 文件统一放在 `/home/handin-malloc/traces` 中，由于实验难度较大，允许同学们把 trace 复制到**本地**针对数据做一些优化。**由于 trace 文件较大，请同学们不要在自己的目录下保存全部 trace 文件。**

6. 规则

- 不应修改 `mm.c` 中函数调用接口的参数形式。
- 不应在代码中以任何方式使用内存分配管理的库函数，如 `malloc`, `calloc`, `free`, `realloc`, `sbrk`, `brk` 等。
- 不可以在 `mm.c` 中定义任何全局和静态变量。（课本上代码里使用的全局指针由 `memlib.c` 中的 `mem_heap_lo`函数获得）。
- 为了与网上可以查到的博客作一些小的区分，我们额外要求程序不可以使用结构体定义。（以上两条要求会给程序带来一些实现细节的变化，参考网上的不规范代码时请务必修改）。
- 必须保证内存块是 8 字节对齐的，测试驱动会判断程序是否符合要求。

7. 测试文件

一个测试文件分为两部分，头部和数据部分。

- 头部：
  - 第一行：建议的申请的堆的大小
  - 第二行：数据中涉及的内存对象的数量
  - 第三行：操作的数量
  - 第四行：测试文件的权重（本实验中无意义）
- 数据部分：
  - `a <id> <bytes>` 为指定的内存对象申请 *bytes* 大小的空间（malloc）。
  - `r <id> <bytes>` 为指定的内存对象重新分配 *bytes* 大小的空间（realloc）。
  - `f <id>` 释放指定的内存对象（free）。

可以在 `/home/handin-malloc/traces` 中获取 trace 文件的详细信息。

8. 提示

- `./mdriver -f` 可用于前期使用 handout 中的小的数据文件调试程序正确性。
- `./mdriver -v or -V` 可以看到性能表现和错误。
- 使用 `gcc -g` 编译程序以使用 gdb（助教已经完成了这一步，直接 make 就可以使用 gdb。）
- 把 csapp 上的 malloc 代码都弄懂。（请确定自己是真的懂了）
- 使用 C 的宏定义实现指针转换算法和弥补不能使用结构体带来的不便。
- 分阶段实现程序。前 9 个测试文件只包含 malloc 和 free 操作。可以通过先结合使用 malloc和 free 来实现 realloc，但为了获得更好的性能，需要实现一个独立的 realloc 算法。
- 本实验对指针的使用要求较高，请确保自己**非常熟悉指针的使用和转换**以避免出现 bug。对指针概念和操作不够了解的同学可以先阅读文档包中提供的 *K_R Pointers*。
- 早点开始！这可能是**大部分**同学学习计算机以来将要完成的最底层最复杂的代码，如果想要获得令人满意的分数，一定要尽早开始。**对于大多数人来说这不是一个可以一天肝完的实验。**

评分标准

- 报告（50%）：
  - 完成的部分
  - 遇到的问题及解决方案
  - 实现的内存分配管理算法
  - 反思总结
- 代码（50%）：
  - 正确性（30%）：

    评测时会使用 13 个测试文件（在 `config.h` 中定义），包含下发的两个小的数据文件。测评程序根据正确通过的百分比评定正确性分数。
  - 性能分（70%）：

    如果程序在 13 个测试文件中**全部通过正确性测试**，评测程序将根据性能表现评定性能分，程序的性能分取决于性能参数 Performance Index，最高为 100 点。
    - 吞吐量（40%）：

      测试程序会计算程序的平均 Kops，记为 x，程序将获得 $100 \times 40\% \times \frac{x}{15000}$ 的性能参数。当 $x > 15000$ 时程序将获得这部分全部的 40 点性能参数。
    - 空间利用率（60%）：

      测试程序会计程序的平均空间利用率，记为 y，程序将获得 $100 \times 40\% \times y$ 的性能参数。

    如果程序获得 95 点及以上的性能参数，即可获得全部的性能分，否则只可以得到 $Performance\ Index\%$ 的性能分。
- 按时按要求提交：
  - 晚交酌情扣分
  - obe 上只提交 pdf 格式的实验报告，提交其他格式如 word（排版可能会乱）、txt（不会有吧）、jpg（。。。）等、代码或压缩包的减分。

祝大家实验愉快~~☺