项目 #1：Malloc 库
第1部分

ECE 650 – 春季

2023年

截止日期为 1/26 晚上 11:59

一般说明

1. 你将单独完成这个项目。

2. 本作业的代码应在基于 UNIX 的环境中开发和测试，特别是 Duke Linux 环境，网址为登录.oit.duke.edu 或者 https://vcm.duke.edu/ .

3. 您必须仔细遵守此作业规范，并按要求上交所有内容（并按照描述的正确格式）。由于班级规模较大，因此需要这样做才能提高评分效率。

4. 你应该计划尽早开始这个项目，并随着时间的推移取得稳步进展。完成作业需要时间和仔细思考。

5. 如果您在截止日期前 2 天在 Gradescope 上提交作业，您就有机会预览我们的自动评分器测试的程序结果。然后，您可以在截止日期前改进您的课程并重新评分。

介绍

对于此作业，您将实现自己版本的 C 标准库中的多个内存分配函数（实际上，您将有机会实现和研究如下所述的多个不同版本）。您的实施将在 C 代码中完成。

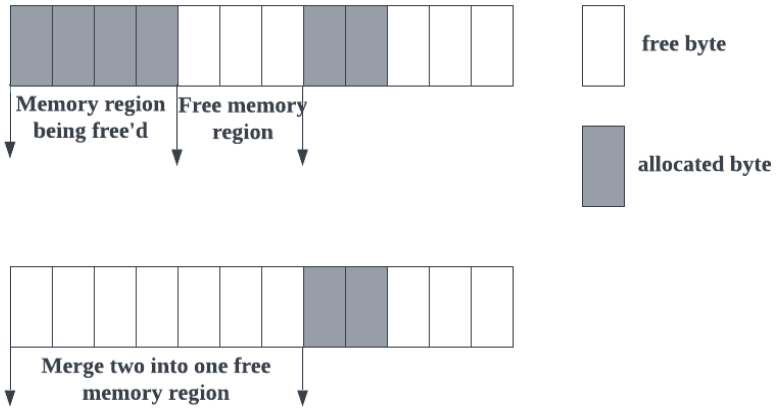
C标准库包括4个malloc相关的库函数：malloc()、free()、calloc()和realloc()。在此作业中，您只需要实现 malloc() 和 free() 的版本：

void *malloc(size_t size); 免费无效
（无效* ptr）；

有关这些功能的预期操作的完整说明，请参阅手册页。本质上，malloc() 接受内存分配的大小（字节数），在程序的数据区域中定位一个地址，那里有足够的空间来容纳指定的字节数，并返回该地址供调用程序使用. free() 函数获取一个地址（由先前的 malloc 操作返回）并将该数据区域标记为再次可用以供使用。

本作业描述末尾的提交说明提供了有关要创建哪些代码文件、如何命名新版本的 malloc 函数等的具体详细信息。

当您实现 malloc() 和 free() 时，您会发现随着内存分配和释放的发生，您有时会释放与其他空闲内存区域相邻的内存区域。**你的实施是*必需的*在这种情况下，通过将相邻的空闲区域合并为一个内存空闲区域来合并。同样，如果理想的空闲区域大于请求的大小，也需要拆分空闲区域。**下图说明了区域的合并。



在此作业中，您将开发一个 malloc 实现并研究不同的分配策略。在作业 2 中，您将使该实现成为线程安全的。

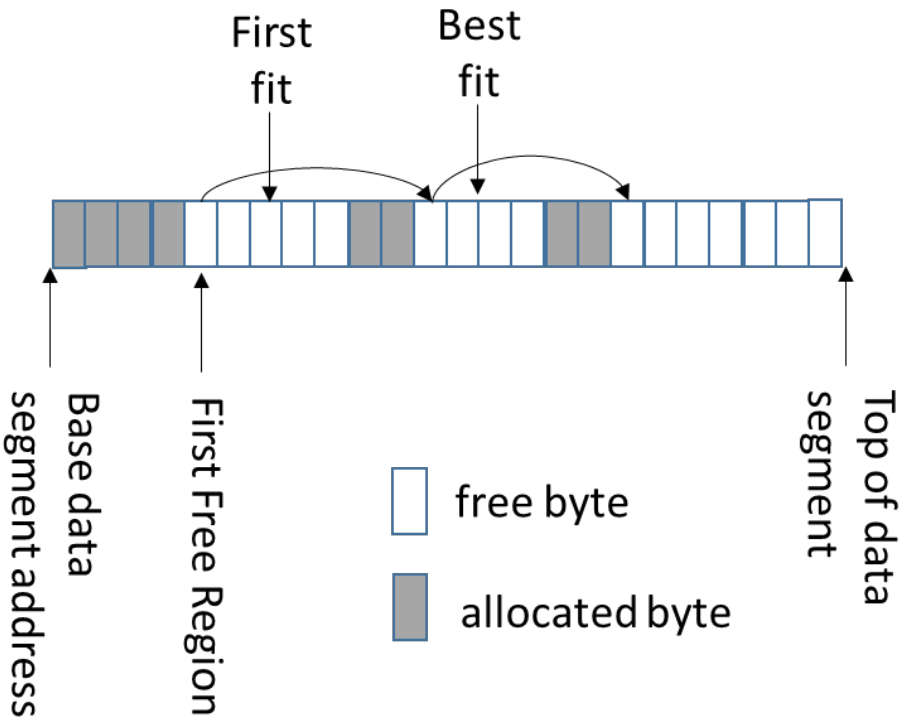
内存分配策略研究

您的任务是实现 malloc 和 free 的 2 个版本，每个版本基于确定要分配的内存区域的不同策略。这两个策略是：

1. **首次适配**：检查空闲空间跟踪器（例如空闲列表），并从具有足够空间的第一个空闲区域分配一个地址以适应请求的分配大小。

2. **最合适**：检查所有空闲空间信息，从空闲区域中分配一个地址，该地址具有最小的大于或等于请求的分配大小的字节数。

下图说明了对于 2 个字节的 malloc() 请求，每个策略将如何运行（假设空闲区域按从左到右的顺序遍历）：



要求

Malloc 实现

为了实施您的分配策略，您将创建 4 个函数：

```
//首先适配malloc/free void
*ff_malloc(size_t size); void ff_free(void
*ptr);

//最适合malloc/free
void *bf_malloc(size_t size); void
bf_free(void *ptr);
```

请注意，在所有情况下，都应使用最小化进程数据段大小的策略。换句话说，如果没有适合分配请求的可用空间，则应使用 sbrk() 来创建该空间。此外，当收到分配请求时，**只能分配请求大小的内存**(除了剩余的可用空间太小而无法跟踪）。但是，您不需要执行任何类型的垃圾收集（例如，减少进程数据段的大小，即使数据段顶部的分配已被释放）。

在 free() 上，您的实现需要将新释放的区域与任何当前空闲的相邻区域合并。换句话说，您的簿记数据结构不应包含多个相邻的空闲区域，因为这会导致 malloc 期间区域选择不当。由于我们是自己管理内存空间，所以不需要调用原来的free()函数。

性能研究报告

除了实现这些 malloc 函数之外，您的任务是对不同分配策略下的 malloc() 性能进行性能研究。将提供几个实验程序，以不同的模式（例如频率、大小）执行 malloc() 和 free() 请求。感兴趣的指标将是：1）程序的运行时间，因为不同分配策略的实施可能会导致不同数量的内存分配开销，以及 2）碎片（即未分配的数据段空间量除以总数据段空间）。为了捕获#2，您还应该实现两个额外的库函数：

```
unsigned long get_data_segment_size(); //以字节为单位
```

unsigned long get_data_segment_free_space_size(); //以字节为单位

这些功能还必须包括用于元数据的空间。
get_data_segment_size() = 整个堆内存（这包括用于保存元数据的内存）

get_data_segment_free_space_size() =“空闲列表”的大小=空闲列表中块的（实际可用空闲空间+元数据占用的空间）

然后，使用这些功能，您可以使用包含的测试程序以及您自己创建的测试程序来评估和比较算法。

提示

暗示：要实现 malloc()，您应该熟悉 sbrk() 系统调用。此系统调用可用于：1) 返回表示进程数据段当前结尾的地址（称为程序中断），以及 2) 按“increment”指定的量增加进程数据段的大小。

void *sbrk(intptr_t 增量);

暗示：实现 malloc() / free() 和管理内存空间的常用方法是**保留表示空闲内存区域列表的数据结构**（想想为什么不保留所有内存区域的列表和实现的时间复杂度）。当调用 malloc() 和 free() 以分配和释放进程数据段中的内存区域时，此空闲内存范围集合将发生变化。您可以使用您认为最合适的结构和状态跟踪来设计和实现您的 malloc 和 free。

入门套件

Saka 上的文件中包含一个入门工具包我在资源下的文件夹中:**homework1-kit.tgz**

此存档可以下载到本地计算机并使用以下命令传输到 VMscp为了
* nix/Mac 系统（最简单的方法）或使用命令行检索wget如果托管在网站上或通过使用 Windows 上的文件传输应用程序（Windows 版 Mobaxterm 具有内置的拖放文件传输）。然后可以使用“tar xvzf homework1-kit.tgz”。

该套件包括：

- **生成文件**: libmymalloc.so 的示例 Makefile。

- **general_tests/**: 一般正确性测试，详见README.txt。
- **alloc_policy_tests/**: 分配策略测试用例，详见README.txt。
- **alloc_policy_tests_osx/**: 同样适用于 MacOS。
  注意：Mac 操作系统是不是 评分平台；这些文件是为方便起见而提供的。此外，您可能需要将#include "time.h" 更改为 #include "sys/time.h"

测试你的系统

tl;dr：编写你自己的测试！ 至少一个基本的打印数据结构和最少的测试可以帮助您发现几个错误。

提供的代码用于最少的测试，但提供的材料不会详尽地评估您的实施的正确性。建议创建您自己的使用您的库的测试软件，以在开发过程中提供帮助并确保在各种情况下的正确性。如果您想了解有关开发测试用例的一般介绍，请参阅Sarah Heckman 的软件测试 .

对于调试，请确保使用 gdb 和 valgrind 等工具。如果您不习惯使用这些工具，请联系助教并获得帮助。另外，请使用可能对您有帮助的相关在线资源。确保检查 valgrind 中的任何错误，因为这些可能是您程序中未捕获的错误。

回顾一下 emacs 中的 gdb

- 在您的 Makefile 中，将 -O3 更改为 -ggdb3。-O3 是优化标志，-ggdb3 是调试标志。-ggdb3 需要让 gdb 将信息添加到已编译的二进制文件中，这将帮助它查明确切的行号、函数名称等。
- 以下是如何在 emacs 中使用 gdb 的链接：AoD_1：在 Emacs 中使用 GDB 进行调试 您可能有更好的参考视频，请随时与他人分享。

- 在emacs 中，打开您拥有需要运行的二进制文件的目录中的任何文件。
  例如，如果它是 general_tests 中的 mymalloc_test，则导航到该目录并打开一个文件。

- 然后运行Mx gdb
- 然后键入(r) 运行。
- 您的代码以段错误结束——如果您执行了上述所有步骤，emacs 中的 gdb 已经向您显示发生段错误的行。
- 然后您可以运行(bt) for backtrace，它会显示是哪一系列的函数调用将您引导到这里，例如导致此函数调用的主测试文件中的行号。
- 请查阅此 GDB 备忘单GDB 快速参考 GDB 版本 4 更多。

Valgrind

一些可能对 valgrind 有用的标志：

```
valgrind -v --leak-check=full --track-origins=yes ./program
```

获得帮助

请阅读 Eric Steven Raymond 的如何聪明地提问 . 如果您不知道如何执行文档中提到的任何事情，请确保先在线查找，然后在您的问题中寻求更多细节的进一步说明。助教总是很乐意提供帮助。但是，请记住，您需要帮助他们帮助您。

评分标准

代码正确性（54'），总共 12 个测试用例（您可以假设所有测试用例的输入始终有效）

代码检查（16'），需要拆分和合并块

报告(30')，包括实施描述、绩效结果展示和结果分析

除非表现太差，否则该项目不会根据表现评分。IE：
如果运行单个测试用例需要 1 或 2 分钟，则可能会扣分。

额外学分

额外的学分是值得的**10%**作业等级。

对于额外的学分，有一个单独的**项目 1 中点检查作业**在 Gradescope 上，将于**1/21 晚上 11:59**. 如果您希望获得此额外学分，请完成**malloc() 的 First Fit 实现（确保您的 free() 也有效）**，并在上述截止日期前提交您的源代码（加分部分不需要书面报告，请按照**详细提交说明**用于源代码提交）。您将需要通过所有自动评分器测试用例才能获得学分。

此项目 1 中点检查作业不会影响您在 1 月 26 日晚上 11:59 到期的项目 1 作业成绩。项目 1 中点检查作业的自动评分器只会测试您的**首次适配实施**而 Project 1 Assignment 将测试 First Fit 和 Best Fit。

详细的提交说明

请将以下资料提交至**等级范围**:

1. 书面报告称**报告.pdf**包括了：
   - 概述您如何实施分配策略，
   - 性能实验的结果，
   - 结果分析（例如，为什么您认为您观察到了针对具有不同 malloc/free 模式的不同策略所做的结果，您对哪种策略似乎最有效有建议等）。

2.所有源代码在一个名为“**my_malloc**”

- 应该有一个头文件名“**my_malloc.h**” 以及所有 *_malloc() 和 *_free() 函数的函数定义。
- 您可以在“**my_malloc.c**”。如果您想使用不同的 C 源文件，请在报告中说明这些文件的内容。
- 应该有一个“**生成文件**” 其中至少包含两个目标：

  1) “all”应该将你的代码构建到一个名为“libmymalloc.so”的共享库中

  2) “clean”应该删除除源代码文件之外的所有文件。提供的 Makefile 可以按原样使用、扩展或完全替换。如果你之前没有将代码编译成共享库，你应该可以在网上找到很多资料，或者找导师或TA指导一下！

  有了这个“Makefile”基础设施，测试程序将能够：1)
  # 包含“my_malloc.h”和 2) 链接到 libmymalloc.so (-lmymalloc)，然后可以访问新的 malloc 函数。就这样，您将在 C 标准库中创建自己版本的 malloc 例程！