CS 213，2001年秋季

Malloc实验室：编写动态存储分配器

分配时间：11月2日星期五，截止日期：11月20日星期二晚上11:59

科里·威廉姆斯（cgw@andrew.cmu.edu）是这项任务的负责人。

**1导言**

在本实验中，您将为C程序编写一个动态存储分配器，即您自己版本的malloc、free和realloc例程。我们鼓励您创造性地探索设计空间，并实现正确、高效和快速的分配器。

**2物流**

你可以在最多两个人的小组中工作。对作业的任何澄清和修改都将发布在课程网页上。

**3分发说明**

特定站点：在这里插入一段话，解释学生应该如何下载malloclab-handout.tar文件。

首先将malloclab-handout.tar复制到您计划在其中完成工作的受保护目录中。然后发出命令：tar xvf malloclab-handout.tar。这将导致许多文件被解压缩到目录中。您将修改和上交的唯一文件是mm.c。mdriver.c程序是一个驱动程序，允许您评估解决方案的性能。使用make命令生成驱动程序代码，并使用./mdriver-v命令运行它。（-V标志显示有用的摘要信息。）

查看文件mm.c，您会注意到一个C结构团队，您应该在其中插入所请求的关于组成您的编程团队的一两个人的识别信息。马上做，这样你就不会忘记了。

完成实验后，您将只上交一个文件(mm.c)，其中包含您的解决方案。

1

**4如何在实验室工作**

您的动态存储分配器将由以下四个函数组成，它们在mm.h中声明，在mm.c中定义。

intmm\_init(void)；

void\*mm\_malloc(size\_t size)；

void mm\_free(void\*ptr)；

void\*mm\_realloc(void\*ptr，size\_t size)；

我们给你的mm.c文件实现了我们能想到的最简单但功能仍然正确的malloc包。以此为起点，修改这些函数（并可能定义其他私有静态函数），使它们服从以下语义：

* mm init：在调用mm malloc mm realloc或mm free之前，应用程序（即，您将用于评估您的实现的跟踪驱动程序程序）调用mm init来执行任何必要的初始化，例如分配初始堆区域。如果执行初始化时出现问题，返回值应该是-1，否则返回值应该是0。
* mm malloc：mm malloc例程返回一个指向至少大小为字节的已分配块有效负载的指针。整个分配的块应该位于堆区域内，并且不应该与任何其他分配的块重叠。

我们将把您的实现与标准C库(libc)中提供的malloc版本进行比较。因为libc malloc总是返回对齐到8字节的有效负载指针，所以您的malloc实现也应该这样做，并且总是返回8字节对齐的指针。

* mm free：mm free例程释放ptr指向的块。它不返回任何内容­。只有当传递的指针(ptr)由先前对mm malloc或mm realloc的调用返回并且尚未被释放时，此例程才能保证工作。
* mm realloc：mm realloc例程返回一个指向至少大小为字节的已分配区域的指针，并具有以下约束。

-如果ptr为空，则调用等效于mm malloc(大小)；

-如果size等于零，则呼叫相当于mm free(ptr)；

-如果ptr不为空，则它必须是由先前对mm malloc或mm realloc的调用返回的。对mm realloc的调用将ptr（旧块）指向的内存块的大小更改为大小字节，并返回新块的地址。请注意，新块的地址可能与旧块相同­，也可能不同，这取决于您的实现、旧块中的内部碎片数量以及realloc请求的大小。

新块的内容与旧ptr块的内容相同，直到新旧大小的最小值。其他一切都是未初始化的。例如，如果旧块是8字节，而新块是12字节，则新块的前8字节与前8字节相同

2

旧块的字节和最后4个字节是未初始化的。类似地，如果旧块是8字节，而新块是4字节，那么新块的内容与旧块的前4字节相同。

这些语义与相应的libc malloc、realloc和free例程的语义相­匹配。在shell中键入man malloc以获取完整的文档。

**5堆一致性检查器**

众所周知，动态内存分配器很难正确有效地编程。它们很难正确编程，因为它们涉及大量非类型化指针操作。您会发现编写一个扫描堆并检查其一致性的堆检查器非常有帮助。

堆检查器可能检查的一些示例包括：

* 空闲列表中的每个块都标记为空闲吗？
* 是否有任何连续的自由块以某种方式逃脱了合并？
* 每个空闲块实际上都在空闲列表中吗？
* 空闲列表中的指针是否指向有效的空闲块？
* 是否有分配的块重叠？
* 堆块中的指针指向有效的堆地址吗？

您的堆检查器将由mm.c中的函数int mm check(v­oid)组成。它将检查您认为谨慎的任何不变量或一致性条件。当且仅当您的堆是一致的，它才返回一个非零值。您不限于列出的建议，也不需要检查所有建议。当mm检查失败时，我们鼓励您打印出错误消息。

这个一致性检查器是供您在开发过程中自己调试的。当您提交mm.c时，请确保删除对mm check的任何调用，因为它们会降低您的吞吐量。将为您的mm检查功能提供风格点。确保加入评论并记录您正在检查的内容。

**6支持例程**

memlib.c包模拟动态内存分配器的内存系统。您可以在memlib.c中调用以下函数：

* void\*mem sbrk(int incr)：以incr字节扩展堆，其中incr是非零正整数，返回指向新分配的堆区域第一个字节的泛型指针。语义与Unix sbrk函数相同，只是mem sbrk只接受一个正的非零整数参数。

3

* 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）：返回系统的页面大小，单位为字节（Linux系统上为4K）。

**7跟踪驱动程序程序**

malloclab-handout.tar发行版中的驱动程序mdriver­.c测试您的mm.c包的正确性、空间利用率和吞吐量。驱动程序由malloclab-handout.tar发行版中包含的一组跟踪文件控制。每个跟踪文件都包含一系列allocate、reallocate和free指令，这些指令指示驱动程序以某种顺序调用mm malloc、mm realloc和mm free例程。驱动程序和跟踪文件与我们对您的handin mm.c文件进行评分时使用的文件相同。

驱动程序mdriver.c接受以下命令行参数：

* -t<tracedir>：在tracedir目录中查找默认跟踪文件，而不是在config.h中定义的默认目录中查找。
* -f<tracefile>：使用一个特定的tracefile进行测试，而不是默认的tracefile集。
* -h：打印命令行参数的摘要。
* -l：除了学生的malloc包之外，还运行和测量libc malloc。
* -v：详细输出。在紧凑的表格中打印每个跟踪文件的性能明细。
* -V：更详细的输出。在处理每个跟踪文件时打印附加诊断信息。在调试过程中用于确定哪个跟踪文件导致malloc包失败。

**8编程规则**

* 您不应该更改mm.c中的任何接口。
* 您不应该调用任何与内存管理相关的库调用或系统调用。这不包括在代码中使用malloc、calloc、free、realloc、sbrk、brk或这些调用的任何变体。
* 不允许在mm.c程序中定义任何全局或静态复合数据结构，如数组、结构、树或列表。但是，允许在mm.c中声明全局标量变量，如整数、浮点和指针。

4

* 为了与libc malloc包保持一致，libc malloc包返回与8字节边界对齐的块，分配器必须始终返回与8字节边界对齐的指针。司机会为你强制执行这一要求。

**9评价**

如果你违反了任何规则，或者你的代码有问题，导致驱动程序崩溃，你将得到零分。否则，您的成绩将按以下方式计算：

* 正确性（20分）。如果您的解决方案通过了驱动程序执行的正确性测试，您将获得满分。您将获得每条正确跟踪的部分积分。
* 表现（35分）。将使用两个性能指标来评估您的解决方案：

-空间利用率：驱动程序使用的内存总量（即通过mm malloc或mm realloc分配但尚未通过mm free释放）与分配器使用的堆大小之间的峰值比率。最佳比率等于1。您应该找到好的策略来最小化碎片化，以便使该比率尽可能接近最佳。

-吞吐量：平均每秒完成的操作数。

驱动程序通过计算性能指数P来总结分配器的性能，P是空间利用率和吞吐量的加权和

<UNK>

P=wU+(1-W)min 1,T

*Tlibc*

其中U是您的空间利用率，T是您的吞吐量，Tlibc是默认跟踪上系统上libc malloc的估计吞吐量。1性能指数更倾向于空间利用率而不是吞吐量，默认值为w=0.6。

观察到内存和CPU周期都是昂贵的系统资源，我们采用这个公式来鼓励内存利用率和吞吐量的平衡优化。理想情况下，性能指数将达到P=w+(1-W)=1或100%。由于每个指标对性能指数的贡献最多分别为w和1-W，因此您不应该走极端，只优化内存利用率或吞吐量。要获得好的分数，您必须在利用率和吞吐量之间取得平衡。

* 风格（10分）。

-你的代码应该分解成函数，尽可能少地使用全局变量。

-您的代码应该以一个头注释开始，描述您的空闲和已分配块的结构、空闲列表的组织以及您的分配器如何操作空闲列表。每个函数前面都应该有一个描述函数功能的头注释。

1 Tlibc的值是驱动程序中的一个常量（600 Kops/s），由讲师在配置程序时确定。

5

-每个子程序都应该有一个标题注释，描述它做什么以及如何做。-您的堆一致性检查器mm检查应该是彻底的并且有良好的文档记录。

一个好的堆一致性检查器会给你5分，一个好的程序结构和注释会给你5分。

**10手动说明**

特定站点：在此插入一段，解释学生应该如何提交他们的解决方案mm.c文件。

**11提示**

* 使用mdriver-f选项。在初始开发期间，使用微小的跟踪文件将简化调试和测­试。我们包含了两个这样的跟踪文件(short1,2-bal.rep),您可以使用它们进行初始调试。
* 使用mdriver-v和-v选项。-v选项将为您提供每个跟踪文件的详细摘要。-V还将指示何时读取每个跟踪文件，这将有助于您隔离错误。
* 用gcc-g编译并使用调试器。调试器将帮助您隔离和识别越界内存引用。
* 理解教科书中malloc实现的每一行。教科书中有一个基于隐式空闲列表的简单分配器的详细示例。使用这是一个出发点。在理解了简单隐式列表分配器的所有内容之前，不要开始处理分配器。
* 将指针算法封装在C预处理器宏中。内存管理器中的指­针算法令人困惑且容易出错，因为所有必要的强制转换。您可以通过为指针操作编写宏来显著降低复杂性。示例见正文。
* 分阶段实施。前9个跟踪包含对malloc和free的请求。最后两个跟踪包含对realloc、malloc和free的请求。我们建议您首先让您的malloc和free例程在前9个跟踪上正确有效地工作。只有这样，您才应该将注意力转向realloc实现。首先，在现有的malloc和free实现的基础上构建realloc。但是要获得真正好的性能，您需要构建一个独立的realloc。
* 使用分析器。您可能会发现gprof工具有助于优化性能。
* 早点开始！用几页代码编写一个高效的malloc包是可能的。然而，我们可以保证，这将是您职业生涯中迄今为止编写的最困难和最复杂的代码。所以早点开始，祝你好运！

6