CS213,2001年秋
季
编写一个分配的动态存储分配器。2、到期时间：11月11日星期二。20,11：59PM

科里·威廉姆斯(cgw@andrew.cmu.edu)是这项任务的负责人。

1 介绍

在这个实验室中，您将为C程序编写一个动态存储分配器，即您自己的版本的malloc，免费和现实的例程。我们鼓励您创造性地探索设计空间，并实现一个正确、高效和快速的分配器。

2 物流

你最多可以和两个人一起工作。对作业的任何澄清和修改都将张贴在课程网页上。

3 分发说明

特定于网站：在这里插入一段，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执行工作的受保护的目录中。然后发出命令：tarxvfmalloclab-handout.tar。这将导致许多文件被解压缩到目录中。您将修改和提交的唯一文件是mm.c。mdriver.c程序是一个驱动程序，它允许您评估解决方案的性能。使用命令make生成驱动程序代码并使用该命令运行它。/mdriver-V.（-V标志显示有用的摘要信息。）

查看文件mm.c，你会注意到一个C结构团队，您应该在其中插入所请求的关于包含编程团队的一个或两个人的标识信息。**马上这么做，这样你就不会忘记了。**

当您完成实验室后，您将只提交一个文件(mm.c)，其中包含您的解决方案。

4 如何在实验室里工作

您的动态存储分配器将包括以下四个函数，它们以mm.h为单位声明
并用mm.c定义。

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

我们给您的mm.c文件实现了我们可以想到的最简单但在功能上仍然正确的malloc包。使用它作为起始位置，修改这些函数（并可能定义其他私有静态函数），以使它们服从以下语义：

- mminit：在调用mm或mm之前，应用程序（即您将用于评估实现的跟踪驱动驱动程序）调用mminit来执行任何必要的初始化，例如分配初始堆区域。如果在执行初始化时出现问题，返回值应该为-1，否则为0。

- mmmalloc：mmmalloc例程返回一个指向至少大小字节的分配块有效负载的指针。整个已分配的块应该位于堆区域内，并且不应该与任何其他已分配的块重叠。

  我们将将您的实现与标准C库(libc)中提供的malloc版本进行比较。由于libcmalloc总是返回与8字节对齐的有效负载指针，malloc实现应该这样做，并且总是返回8字节对齐的指针。

- mm自由：mm自由程序释放ptr指向的块。它什么也不回来。只有当传递的指针(ptr)被mmmalloc或mmrealloc返回且尚未释放时，此例程才能保证工作。

- mmrealloc例程返回一个指向至少大小的分配区域的指针
  具有以下约束条件的字节。

  - 如果ptr为NULL，则调用等于mmmalloc（大小）；
  - 如果大小等于零，则调用等于无mm的(ptr)；
  - 如果ptr不是空的，则必须通过对mmmalloc或mm实际值的早期调用返回。调用mmreaelloc会将ptr（旧块）指向的内存块的大小更改为字节大小，并返回新块的地址。注意，新块的地址可能与旧块相同，也可能不同，这取决于实现、旧块中内部碎片的数量和relloc请求的大小。

    新块的内容与旧ptr块的内容相同，最高可达到新旧大小的最小值。其他所有内容都未初始化。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

旧块的字节和最后4个字节都未初始化。类似地，如果旧块是8个字节，而新块是4个字节，则新块的内容与旧块的前4个字节相同。

这些语义与相应的libc、realloc和自由例程的语义相匹配。以获得完整的文档。

5 高一致性检查器

动态内存分配器是出了名的正确和高效编程的棘手问题。它们很难正确地编程，因为它们涉及到许多未键入类型的指针操作。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。

堆检查器可能要检查的一些示例是：

- 免费列表中的每个方块都被标记为免费的吗？
- 有没有任何连续的自由块以某种方式逃脱了合并？
- 每个免费块都在免费列表中吗？
- 免费列表中的指针是否指向有效的免费块？
- 是否有任何已分配的块存在重叠？
- 堆块中的指针是否指向有效的堆地址？

您的堆检查器将由mmmm检查（空）组成。它将检查您认为谨慎的任何不变量或一致性条件。当且仅当堆一致时，它才会返回一个非零值。您不限于所列出的建议，也不需要检查所有的建议。当mm检查失败时，系统鼓励您打印出错误消息。

此一致性检查器可用于您在开发期间进行自己的调试。当您提交mm.c时，确保删除任何调用mm检查，因为它们将减缓您的吞吐量。样式点将给出为您的毫米检查功能。确保发表评论并记录你正在检查的内容。

6 支持例程

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

- 无效*memsbrk(intincr)：按incr字节扩展堆，其中incr是一个正的非零整数，并返回一个指向新分配的堆区域的第一个字节的通用指针。这些语义与Unixsbrk函数相同，除了memsbrk只接受一个正的非零整数参数。

- 无效*Mem堆lo(void)：返回指向堆中第一个字节的通用指针。
- 无效*mem堆hi(void)：返回指向堆中最后一个字节的通用指针。
- 大小tmem堆大小（空）：以字节为单位返回堆的当前大小。
- 大小tmem页面大小（空）：以字节返回系统为单位的页面大小（Linux系统上的4K）。

7 跟踪驱动的驱动程序

malloclab-handout.tar分发中的驱动程序m驱动程序。c测试mm.c包的正确性、空间利用率和吞吐量。驱动程序程序由malloclab-handout.tar发行版中包含的一组跟踪文件控制。每个跟踪文件包含一个分配、重新分配和自由方向的序列，这些方向指示驱动程序按某些顺序调用mmmalloc、mmrelloc和mm自由例程。驱动程序和跟踪文件是我们将使用的相同的手动移动文件。

驱动程序m驱动程序。c接受以下命令行参数：

- -t<跟踪器>：在目录中查找默认的跟踪文件，而不是在config.h中定义的默认目录。
- -f<跟踪文件>：使用一个特定的跟踪文件进行测试，而不是默认的跟踪文件集。
- -h：打印命令行参数的摘要。
- -l：除了学生的课程包外，还要运行和测量课程包。
- -v：Verbose输出。在一个紧凑的表格中打印每个跟踪文件的性能明细。
- -V：更多的涡量输出。在处理每个跟踪文件时，将打印其他诊断信息。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

- 你不应该以mm.c的形式改变任何接口。
- 您不应该调用任何与内存管理相关的库调用或系统调用。这不包括使用malloc、calloc、免费、relalloc、sbrk、brk或代码中这些调用的任何变体。
- 不允许您在mm.c程序中定义任何全局或静态的复合数据结构，如数组、结构、树或列表。但是，您可以以mm.c格式声明全局标量变量，如整数、浮点数和指针。

- 为了与libcMalloc包保持一致，该包返回与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。司机将为您强制执行这一要求。

9 评价

如果你违反了任何规则，或者你的代码坏了驱动程序，你将得到零点。否则，您的成绩将计算如下：

- *正确性（20分）*。如果您的解决方案通过了驱动程序程序执行的正确性测试，您将获得全部积分。你将获得部分信用，每一个正确的跟踪。
- *性能（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：驱动程序使用的聚合内存量(即通过mmmalloc或mmrealloc分配，但尚未通过mm自由释放)与分配器使用的堆的大小之间的峰值比率。最优的比率等于1。您应该找到好的策略来最小化碎片化，以便使这个比率尽可能接近最佳值。
  - 吞吐量：每秒完成的平均操作次数。

  驱动程序通过计算性能索引来总结分配器的性能，
  P，这是空间利用率和吞吐量的加权之和

  $$P=wU+(1-w)，第1分，\quad \cdot \quad \frac{T^{\Sigma}}{T_{Iibc}}$$

  其中你是你的空间利用率，T是你的吞吐量，而T_{Iibc}是在默认跟踪上的libcmalloc在系统上的估计吞吐量。[1]性能指数有利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%。由于每个度量最多将分别对性能指数贡献w和1-w，因此不应该极端地优化内存利用率或吞吐量。要获得良好的分数，您必须在利用率和吞吐量之间取得平衡。
- 样式（10分）。
  - 您的代码应该被分解为函数，并使用尽可能少的全局变量。
  - 代码应该从一个标题注释开始，其中描述了免费块和分配块的结构、免费列表的组织结构，以及分配器如何操作免费列表。每个函数之前都应该有一个标题注释来描述该函数的操作。

[1]T的值$_{Iibc}$是您的教师在配置程序时建立的驱动程序中的一个常数（600Kops/s）。

– 每个子例程都应该有一个头注释，描述它是做什么以及如何做的。

– 您的堆一致性检查器mm检查应该是彻底的和充分的记录。

良好的堆一致性检查器将获得5分，良好的程序结构和评论将获得5分。

10 处理程序说明

特定地点：在这里插入一段，解释学生应该如何提交他们的解决方案mm.c文件。

11 铰链

- 使用mdriver-f选项。在初始开发过程中，使用小型跟踪文件将简化调试和测试。我们包含了两个可以用于初始调试的跟踪文件(short1,2-bal.rep)。
- 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，这将帮助您隔离错误。
- *用gcc-g编译并使用调试器。*调试器将帮助您隔离和识别边界外的内存引用。
- *了解教科书中马尔洛克实现的每一行。*该教科书上有一个基于隐式自由列表的简单分配器的详细示例。使用这是一个出发点。在了解关于简单隐式列表分配器的一切之前，不要开始处理分配器。
- *在C预处理器宏中封装指针算术。*内存管理器中的指针运算是混淆和容易错误的，因为所有的强制转换都是必要的。您可以通过为指针操作编写宏来显著降低复杂性。有关示例，请参见文本。
- *分阶段进行实现。*前9个跟踪包含对malloc和免费的请求。最后2条跟踪包含对真实、故障和免费的请求。我们建议您首先让您的错误和免费例程正确和有效地在前9条跟踪上工作。.只有这样，您才应该将注意力转向实际实现。首先，在现有的malloc和免费实现之上构建ralloc。但要获得真正好的性能，你需要建立一个独立的现实。
- *使用分析程序。*您可能会发现gprof工具有助于优化性能。
- *提前开始！*可以用几页的代码编写一个有效的malloc包。然而，我们可以保证这将是您在职业生涯中编写的一些最困难和最复杂的代码。所以早点开始，祝你好运！