CS 213,2001 年秋
季 Malloc 实验室:编写动态存储分配器 指定时间:11 月 2 日星期五,截止时间:11 月 20 日星期二晚上 11:59

Cory Williams (cgw@andrew.cmu.edu) 是这项任务的负责人。

1 简介

在本实验中,您将为 C 程序编写一个动态存储分配器,即您自己的 malloc、free 和 realloc 例程版本。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

您最多可以在两人的团队中工作。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

特定于站点:在此处插入一段内容,解释学生应如何下载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目录。然后给出命令:tar xvf malloclab-handout.tar。这将导致许多文件被解压到该目录中。您要修改和提交的唯一文件是 mm.c。 mdriver.c 程序是一个驱动程序,可让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命令 ./mdriver -V 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需的关于组成您的编程团队的一两个人的识别信息。立即执行此操作,以免忘记。

完成实验后,您只需提交一个文件 (mm.c),其中包含您的解决方案。

4 如何在实验室工作

您的动态存储分配器将包含以下四个函数,它们在 mm.h 中声明并在 mm.c 中定义。

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

我们为您提供的 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）,以便它们遵循以下语义：

mm init:在调用 mm malloc mm realloc 或 mm free 之前,应用程序（即,将用于评估实现的跟踪驱动的驱动程序）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,则返回值应为 -1,否则为 0。

- mm malloc:mm malloc 例程返回指向至少为 size 字节的已分配块有效负载的指针。整个分配的块应位于堆区域内,并且不应与任何其他分配的块重叠。

  我们会将您的实现与标准 C 库 (libc) 中提供的 malloc 版本进行比较。由于 libc malloc 始终返回与 8 字节对齐的有效负载指针,因此您的 malloc 实现也应该这样做,并且始终返回 8 字节对齐的指针。

- mm free:mm free 例程释放ptr 指向的块。它什么也不返回。仅当传递的指针 (ptr) 由先前对 mm malloc 或 mm realloc 的调用返回且尚未释放时,才能保证此例程正常工作。

- mm realloc:mm realloc 例程返回一个指向至少大小的已分配区域的指针
  具有以下限制的字节。

  –如果 ptr 为 NULL,则调用相当于 mm malloc(size)； –如果 size 等于 0,则
  调用相当于 mm free(ptr)； –如果 ptr 不为 NULL,则它必须是由之前对 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size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,具体取决于您的实现、旧块中的内部碎片量以及重新分配请求的大小。

  新块的内容与旧 ptr 块的内容相同,最多为新旧大小中的最小值。其他一切都未初始化。例如,如果旧块为 8 字节,新块为 12 字节,则新块的前 8 个字节与前 8 个字节相同

旧块的字节和最后 4 个字节未初始化。类似地,如果旧块是 8 字节,新块是 4 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 malloc、realloc 和 free 例程的语义相匹配。在 shell 中键入 man malloc 以获得完整的文档。

5 堆一致性检查器

动态内存分配器是出了名的难以正确有效地编程的野兽。它们很难正确编程,因为它们涉及大量无类型指针操作。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是否都标记为空闲?
- 是否有任何连续的空闲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?
- 任何分配的块是否重叠?
- 堆块中的指针是否指向有效的堆地址?

您的堆检查器将由 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。当且仅当您的堆一致时,它才返回非零值。您不仅限于列出的建议,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。

这个一致性检查器供您在开发过程中自己调试。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。将为您的毫米检查功能提供样式点。请务必添加评论并记录您正在检查的内容。

6 个支持例程

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):将堆扩展 incr 字节,其中 incr 是非零正整数,并返回指向新分配的堆区域的第一个字节的通用指针。语义与 Unix sbrk 函数相同,只是 mem sbrk 仅接受正非零整数参数。

- 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（以字节为单位）。
- size t mem pagesize(void):返回系统的页面大小（以字节为单位）（在 Linux 系统上为 4K）。

7 Trace驱动的驱动程序

malloclab-handout.tar 发行版中的驱动程序 mdriver.c 测试 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分配、重新分配和释放指令，指示驱动程序按某种顺序调用 mm malloc、mm realloc 和 mm free 例程。驱动程序和跟踪文件与我们对交接 mm.c 文件进行评分时将使用的文件相同。 - - -

驱动程序 mdriver.c 接受以下命令行参数：

- -t <tracedir>：在目录tracedir 中查找默认跟踪文件，而不是在config.h 中定义的默认目录。
- -f <tracefile>：使用一个特定的跟踪文件进行测试，而不是默认的一组跟踪文件。
  文件。
- -h：打印命令行参数的摘要。
- -l：除了学生的malloc 包之外，还运行并测量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细分。
- -V：更详细的输出。处理每个跟踪文件时打印附加诊断信息。
  在调试过程中非常有用，可用于确定哪个跟踪文件导致 malloc 包失败。

8 编程规则

- 您不应更改mm.c 中的任何接口。
- 您不应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。
- 不允许您在mm.c 程序中定义任何全局或静态复合数据结构，例如数组、结构、树或列表。但是，您可以在 mm.c 中声明全局标量变量，例如整数、浮点数和指针。

· 为了与libc malloc 包（返回按8 字节边界对齐的块)保持一致,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此要求。

9 评价

如果您违反任何规则或您的代码有错误并导致驱动程序崩溃,您将获得零分。
否则,您的成绩将按如下方式计算：

·正确性（20 分）。如果您的解决方案通过正确性测试,您将获得满分
由驱动程序执行。对于每条正确的轨迹,您都将获得部分积分。

·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。最优比率等于 1。您应该找到好的策略来最小化碎片,以使该比率尽可能接近最优。

–吞吐量:每秒完成的平均操作数。

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

$$P = wU + (1 - w)\min\left(1, \frac{\text{时间}}{\text{Tlibc}}\right)$$

其中 U 是您的空间利用率,T 是您的吞吐量， Tlibc是您的系统上默认跟踪上 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,性能指标将达到 P = w + (1 − w) = 1或100%。由于每个指标最多分别对性能指数贡献 w 和 1 − w,因此您不应走极端,仅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。

· 风格（10 分）。

–您的代码应该分解为函数并使用尽可能少的全局变量。

–您的代码应以标头注释开始,该注释描述空闲块和已分配块的结构、空闲列表的组织以及分配器如何操作空闲列表。每个函数前面都应该有一个标题注释,描述该函数的作用。

1Tlibc的值是驱动程序中的常量 (600 Kops/s),是您的讲师在配置程序时建立的。

–每个子例程应该有一个标题注释来描述它的作用以及它是如何做的。

–您的堆一致性检查器 mm 检查应该彻底且有据可查。

如果您有良好的堆一致性检查器,您将获得 5 分;如果您有良好的程序结构和评论,您将获得 5 分。

10 上交须知

特定于站点:在此处插入一段内容,解释学生应如何提交解决方案mm.c文件。

11 提示

·使用mdriver -f选项。在初始开发过程中,使用微小的跟踪文件将简化调试和测试。我们提供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 还将指示读取每个跟踪文件的时间,这将帮助您隔离错误。

·使用gcc -g进行编译并使用调试器。调试器将帮助您隔离和识别
限制内存引用。

·了解教科书中malloc 实现的每一行。教科书有一个基于隐式空闲列表的简单分配器的详细示例。使用这个是一个出发点。在了解有关简单隐式列表分配器的所有内容之前,不要开始使用分配器。

·将指针算术封装在C 预处理器宏中。由于所有必要的转换,内存管理器中的指针算术令人困惑且容易出错。您可以通过为指针操作编写宏来显着降低复杂性。请参阅文本中的示例。

·分阶段实施。前 9 个跟踪包含对 malloc 和 free 的请求。最后 2 个跟踪包含对 realloc、malloc 和 free 的请求。我们建议您首先让 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。对于初学者来说,在现有的 malloc 和 free 实现之上构建 realloc。但为了获得真正好的性能,您需要构建一个独立的重新分配。

·使用分析器。您可能会发现 gprof 工具有助于优化性能。

·尽早开始!用几页代码就可以编写一个高效的 malloc 包。然而,我们可以保证,这将是您职业生涯迄今为止编写的最困难、最复杂的代码之一。所以早点开始,祝你好运!