CS 213, Fall 2001

Malloc Lab: Writing a Dynamic Storage Allocator

Assigned: Friday Nov. 2, Due: Tuesday Nov. 20, 11:59PM

Cory Williams (cgw@andrew.cmu.edu) is the lead person for this assignment.

**1 Introduction**

In this lab you will be writing a dynamic storage allocator for C programs, i.e., your own version of the malloc, free and realloc routines. You are encouraged to explore the design space creatively and implement an allocator that is correct, efficient and fast.

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。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 Logistics**

You may work in a group of up to two people. Any clarifications a nd revisions to the assignment will be posted on the course Web page.

您最多可以两个人组成一组。对作业的任何澄清和修订都将发布在课程网页上。

**3 Hand Out Instructions**

**SITE-SPECIFIC: Insert a paragraph here that explains how students should download the** malloclab-handout.tar **file.**

Start by copying malloclab-handout.tar to a protected directory in which you plan to do your work. Then give the command: tar xvf malloclab-handout.tar. This will cause a number of files to be unpacked into the directory. The only file you will b e modifying and handing in is mm.c. The mdriver.c program is a driver program that allows you to evaluate the performance of your solution. Use the command make to generate the driver code and run it with the command ./mdriver -V. (The -V flag displays helpful summary information.)

Looking at the file mm.c you'll notice a C structure team into which you should insert the requested identifying information about the one or two individuals comprising your programming team. **Do this right** **away so you don't forget.**

When you have completed the lab, you will hand in only one file ( mm.c), which contains your solution.

现场说明：在此处插入一段，说明学生应如何下载malloclab-handout.tar文件。

首先将malloclab-handout.tar复制到计划在其中进行工作的受保护目录。然后输入命令：tar xvf malloclab-handout.tar。这将导致许多文件解压缩到目录中。您将要修改和上交的唯一文件是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请求的有关组成您的编程团队的一个或两个人的标识信息。立即执行此操作，这样您就不会忘记。

完成实验后，您将只提交一个文件（mm.c），其中包含您的解决方案。

1

**4 How to Work on the Lab**

Your dynamic storage allocator will consist of the following four functions, which are declared in mm.h and defined in mm.c.

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

int mm\_init(void);

void \*mm\_malloc(size\_t size);

void mm\_free(void \*ptr);

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

The mm.c file we have given you implements the simplest but still funct ionally correct malloc package that we could think of. Using this as a starting place, modify these functions (and possibly define other private static functions), so that they obey the following semantics:

我们为您提供的mm.c文件实现了我们可以想到的最简单但仍在功能上正确的malloc程序包。以此为起点，修改这些函数（并可能定义其他私有静态函数），以使它们遵循以下语义：

* mm init: Before calling mm malloc mm realloc or mm free, the application program (i.e., the trace-driven driver program that you will use to evaluate your implementation) calls mm init to perform any necessary initializations, such as allocating the initial heap area. The return value should be -1 if there was a problem in performing the initialization, 0 otherwise.
* mm malloc: The mm malloc routine returns a pointer to an allocated block payload of at least size bytes. The entire allocated block should lie within the heap region and should not overlap with any other allocated chunk.

We will comparing your implementation to the version of malloc supplied in the standard C library (libc). Since the libc malloc always returns payload pointers that are aligned to 8 bytes, your malloc implementation should do likewise and always return 8-byte aligned pointers.

* mm free: The mm free routine frees the block pointed to by ptr. It returns nothing. This rou-tine is only guaranteed to work when the passed pointer (ptr) was returned by an earlier call to mm malloc or mm realloc and has not yet been freed.
* mm realloc: The mm realloc routine returns a pointer to an allocated region of at least size bytes with the following constraints.

**–** ifptris NULL, the call is equivalent tomm malloc(size);

**–** ifsizeis equal to zero, the call is equivalent tomm free(ptr);

**–** ifptris not NULL, it must have been returned by an earlier call tomm mallocormm realloc.The call to mm realloc changes the size of the memory block pointed to by ptr (the *old* *block*) tosizebytes and returns the address of the new block. Notice that the address of thenew block might be the same as the old block, or it might be different, depending on your imple-mentation, the amount of internal fragmentation in the old block, and the size of the realloc request.

The contents of the new block are the same as those of the old ptr block, up to the minimum of the old and new sizes. Everything else is uninitialized. For example, if the old block is 8 bytes and the new block is 12 bytes, then the first 8 bytes of the new bl ock are identical to the first 8

2

bytes of the old block and the last 4 bytes are uninitialized. Similarly, if the old block is 8 bytes and the new block is 4 bytes, then the contents of the new block are identical to the first 4 bytes of the old block.

These semantics match the the semantics of the corresponding libc malloc, realloc, and free rou-tines. Type man malloc to the shell for complete documentation.

•mm init：在调用mmmalloc，mm realloc或mmfree之前，应用程序（即，将用于评估实现的跟踪驱动程序）将调用mm init以执行任何必要的初始化，例如分配初始堆区。如果执行初始化时有问题，则返回值应为-1，否则为0。

•mm malloc：mm malloc例程返回一个指针，该指针指向至少为大小字节的已分配块有效负载。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。

我们将把您的实现与标准C库（libc）中提供的malloc版本进行比较。由于libc malloc总是返回对齐8个字节的有效负载指针，因此您的malloc实现也应该这样做，并且始终返回8个字节的对齐指针。

•mm释放：mm释放例程释放ptr指向的程序段。它什么也不返回。只有在对mm malloc或mm realloc的较早调用返回了传递的指针（ptr）并且尚未释放该例程时，才能保证该例程正常工作。

•mm realloc：mm realloc例程返回一个指针，该指针指向具有以下约束的至少大小字节的已分配区域。

–如果ptr为NULL，则调用等效于mm malloc（size）；

–如果大小等于零，则调用等效于mm free（ptr）；

–如果ptr不为NULL，则必须由先前对mm malloc或mm realloc的调用返回了它。对mm realloc的调用将ptr（旧块）指向的存储块的大小更改为size字节，并返回新块的地址。请注意，新块的地址可能与旧块的地址相同，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重新分配请求的大小。

新块的内容与旧ptr块的内容相同，最大为新旧大小的最小值。其他所有内容均未初始化。例如，如果旧块为8个字节，新块为12个字节，则新块的前8个字节与前8个字节相同

2

未初始化旧块的字节和最后4个字节。同样，如果旧块为8字节，新块为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和自由例程的语义相匹配。在外壳上输入man malloc以获得完整的文档。

**5 Heap Consistency Checker**

Dynamic memory allocators are notoriously tricky beasts to program correctly and efficiently. They are difficult to program correctly because they involve a lot of u ntyped pointer manipulation. You will find it very helpful to write a heap checker that scans the heap and checks it for consistency.

Some examples of what a heap checker might check are:

* Is every block in the free list marked as free?
* Are there any contiguous free blocks that somehow escaped coalescing?
* Is every free block actually in the free list?
* Do the pointers in the free list point to valid free blocks?
* Do any allocated blocks overlap?
* Do the pointers in a heap block point to valid heap addresses?

Your heap checker will consist of the function int mm check(void) in mm.c. It will check any invari-ants or consistency conditions you consider prudent. It returns a nonzero value if and only if your heap is consistent. You are not limited to the listed suggestions nor are you required to check all of them. You are encouraged to print out error messages when mm check fails.

This consistency checker is for your own debugging during development. When you submit mm.c, make sure to remove any calls to mm check as they will slow down your throughput. Style points will be given for your mm check function. Make sure to put in comments and document what you are checking.

众所周知，动态内存分配器是棘手的野兽，无法正确有效地进行编程。它们很难正确编程，因为它们涉及很多非类型化的指针操作。您会发现编写一个用于检查堆并检查其一致性的堆检查器非常有帮助。

堆检查程序可能检查的一些示例是：

•空闲列表中的每个块是否都标记为空闲？

•是否存在以某种方式逃脱的连续自由块？

•每个空闲块实际上是否在空闲列表中？

•空闲列表中的指针是否指向有效的空闲块？

•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int mm check（void）in mm.c函数组成。它将检查您认为谨慎的所有不变量或一致性条件。当且仅当堆是一致的时，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。鼓励您在mm检查失败时打印出错误消息。

此一致性检查器用于在开发过程中进行自己的调试。提交mm.c时，请确保删除所有对mm check的调用，因为它们会减慢您的吞吐量。将为您的mm检查功能提供样式点。确保添加评论并记录您要检查的内容。

**6 Support Routines**

The memlib.c package simulates the memory system for your dynamic memory allocator. You can invoke the following functions in memlib.c:

* void \*mem sbrk(int incr): Expands the heap by incr bytes, where incr is a positive non-zero integer and returns a generic pointer to the first by te of the newly allocated heap area. The semantics are identical to the Unix sbrk function, except that mem sbrk accepts only a positive non-zero integer argument.

3

* void \*mem heap lo(void): Returns a generic pointer to the first byte in the heap.
* void \*mem heap hi(void): Returns a generic pointer to the last byte in the heap.
* size t mem heapsize(void): Returns the current size of the heap in bytes.
* size t mem pagesize(void): Returns the system's page size in bytes (4K on Linux systems).

memlib.c软件包为您的动态内存分配器模拟了内存系统。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是非零的正整数，并返回指向新分配堆区域的第一个字节的通用指针。语义与Unix sbrk函数相同，除了mem sbrk仅接受一个正的非零整数参数。

3

•void \* mem堆lo（void）：返回指向堆中第一个字节的通用指针。

•void \* mem堆hi（void）：返回指向堆中最后一个字节的通用指针。

•size t mem heapsize（void）：以字节为单位返回堆的当前大小。

•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 The Trace-driven Driver Program**

The driver program mdriver.c in the malloclab-handout.tar distribution tests your mm.c pack-

age for correctness, space utilization, and throughput. The driver program is controlled by a set of *trace files*

that are included in the malloclab-handout.tar distribution. Each trace file contains a sequence of

allocate, reallocate, and free directions that instruct the driver to call your mm malloc, mm realloc, and

1. free routines in some sequence. The driver and the trace files are t he same ones we will use when we grade your handin mm.c file.

The driver mdriver.c accepts the following command line arguments:

* -t <tracedir>: Look for the default trace files in directory tracedir instead of the default directory defined in config.h.
* -f <tracefile>: Use one particular tracefile for testing instead of the default set of trace-files.
* -h: Print a summary of the command line arguments.
* -l: Run and measure libc malloc in addition to the student's malloc package.
* -v: Verbose output. Print a performance breakdown for each tracefile in a compact table.
* -V: More verbose output. Prints additional diagnostic information as each trace file is processed. Useful during debugging for determining which trace file is c ausing your malloc package to fail.

malloclab-handout.tar发行版中的驱动程序mdriver.c测试了您的mm.c pack-

正确性，空间利用率和吞吐量的年龄。驱动程序由一组跟踪文件控制

包含在malloclab-handout.tar发行版中。每个跟踪文件包含一个序列

指示驱动程序调用mm malloc，mm realloc和

amfree例程按一定顺序进行。驱动程序和跟踪文件与我们为您的handin mm.c文件评分时将使用的驱动程序和跟踪文件相同。

驱动程序mdriver.c接受以下命令行参数：

•-t 注意：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 ：使用一个特定的跟踪文件进行测试，而不使用默认的跟踪文件集。

•-h：打印命令行参数摘要。

•-l：除了学生的malloc软件包之外，还运行和测量libc malloc。

•-v：详细输出。在紧凑表中为每个跟踪文件打印性能明细。

•-V：更详细的输出。在处理每个跟踪文件时打印其他诊断信息。在调试过程中很有用，可用于确定哪个跟踪文件导致您的malloc包失败。

8编程规则

1. **Programming Rules**
   * You should not change any of the interfaces in mm.c.
   * You should not invoke any memory-management related library calls or system calls. This excludes

the use of malloc, calloc, free, realloc, sbrk, brk or any variants of these calls in your code.

* + You are not allowed to define any global or static compound data structures such as arrays, structs, trees, or lists in your mm.c program. However, you *are* allowed to declare global scalar variables such as integers, floats, and pointers in mm.c.

4

* For consistency with the libc malloc package, which returns blocks aligned on 8-byte boundaries, your allocator must always return pointers that are aligned to 8-byte boundaries. The driver will enforce this requirement for you.

•您不应在mm.c中更改任何接口。

•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。

4

•为了与libc malloc软件包保持一致，后者返回以8字节为边界对齐的块，分配器必须始终返回以8字节为边界对齐的指针。驾驶员将为您强制执行此要求。

9评价

**9 Evaluation**

You will receive **zero points** if you break any of the rules or your code is buggy and crashes the driver.

Otherwise, your grade will be calculated as follows:

* *Correctness (20 points).* You will receive full points if your solution passes the correctness testsperformed by the driver program. You will receive partial credit for each correct trace.
* *Performance (35 points).* Two performance metrics will be used to evaluate your solution:

**–** *Space utilization*: The peak ratio between the aggregate amount of memory used by the driver(i.e., allocated via mm malloc or mm realloc but not yet freed via mm free) and the size of the heap used by your allocator. The optimal ratio equals to 1. You should find good policies to minimize fragmentation in order to make this ratio as close as possible to the optimal.

**–** *Throughput*: The average number of operations completed per second.

The driver program summarizes the performance of your allocator by computing a *performance index*,

* , which is a weighted sum of the space utilization and throughput

|  |  |  |  |
| --- | --- | --- | --- |
| P = WU + (1 − W) MIN 1, | T |  |  |
|  |  |
| TLIBC |  |

where U is your space utilization, T is your throughput, and TLIBC is the estimated throughput of libc malloc on your system on the default traces.1 The performance index favors space utilization over throughput, with a default of W = 0.6.

Observing that both memory and CPU cycles are expensive system resources, we adopt this formula to encourage balanced optimization of both memory utilization and throughput. Ideally, the performance index will reach P = W + (1 − W) = 1 or 100%. Since each metric will contribute at most W and 1 − W to the performance index, respectively, you should not go to extremes to optimize either the memory utilization or the throughput only. To receive a good score, you must achieve a balance between utilization and throughput.

* Style (10 points).

**–** Your code should be decomposed into functions and use as few global variables as possible.

**–** Your code should begin with a header comment that describes the structure of your free andallocated blocks, the organization of the free list, and how your allocator manipulates the free list. each function should be preceeded by a header comment that describes what the function does.

1 The value for TLIBC is a constant in the driver (600 Kops/s) that your instructor established when they configured the program.

5

**–** Each subroutine should have a header comment that describes what it does and how it does it.

**–** Your heap consistency checkermm checkshould be thorough and well-documented.

You will be awarded 5 points for a good heap consistency checker and 5 points for good program structure and comments.

如果您违反任何规则或代码有错误并导致驱动程序崩溃，您将得到零分。

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。您将为每条正确的踪迹获得部分信用。

•性能（35分）。两个性能指标将用于评估您的解决方案：

–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释放的内存）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大程度地减少碎片，以使该比率尽可能接近最佳比率。

–吞吐量：每秒完成的平均操作数。

驱动程序通过计算性能指标来总结分配器的性能，

P，是空间利用率和吞吐量的加权和

P = WU +（1- W）最小值1，T

TLIBC

其中U是您的空间利用率，T是您的吞吐量，而TLIBC是系统在默认跟踪上的libc malloc的估计吞吐量。1性能指数优先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，我们采用此公式来鼓励均衡地优化内存利用率和吞吐量。理想情况下，性能指标应达到P = W +（1- W）= 1或100％。由于每个指标最多分别对性能指标有W和1-W的贡献，因此您不应极端地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。

•样式（10分）。

–您的代码应分解为函数，并使用尽可能少的全局变量。

–您的代码应以标头注释开头，该注释描述您的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纵空闲列表。每个函数之前都应有描述函数功能的标题注释。

1 TLIBC的值是教师在配置程序时建立的驱动程序中的常数（600 Kops / s）。

5

–每个子例程都应具有标题注释，以描述其功能以及执行方式。

–您的堆一致性检查器的mm检查应彻底且有据可查。

优秀的堆一致性检查器将为您授予5分，而良好的程序结构和注释将获得5分。

**10 Handin Instructions**

**SITE-SPECIFIC: Insert a paragraph here that explains how the students should hand in their solution** mm.c **files.**

现场说明：在此处插入一段，说明学生应如何提交其解决方案mm.c文件。

1. **Hints**
   * *Use the* mdriver -f *option.* During initial development, using tiny trace files will simp lify debug-ging and testing. We have included two such trace files ( short1,2-bal.rep) that you can use for initial debugging.
   * *Use the* mdriver -v *and* -V *options.* The-voption will give you a detailed summary for eachtrace file. The -V will also indicate when each trace file is read, which will hel p you isolate errors.
   * *Compile with* gcc -g *and use a debugger.* A debugger will help you isolate and identify out ofbounds memory references.
   * *Understand every line of the malloc implementation in the textbook.* The textbook has a detailedexample of a simple allocator based on an implicit free list. Use this is a point of departure. Don't start working on your allocator until you understand everything about the simple implicit list allocator.
   * *Encapsulate your pointer arithmetic in C preprocessor macros.* Pointer arithmetic in memory man-agers is confusing and error-prone because of all the casting that is necessary. You can reduce the complexity significantly by writing macros for your pointer operations. See the text for examples.
   * *Do your implementation in stages.* The first 9 traces contain requests tomallocandfree. Thelast 2 traces contain requests for realloc, malloc, and free. We recommend that you start by getting your malloc and free routines working correctly and efficiently on the first 9 trac es. Only then should you turn your attention to the realloc implementation. For starters, build realloc on top of your existing malloc and free implementations. But to get really good performance, you will need to build a stand-alone realloc.
   * *Use a profiler.* You may find thegproftool helpful for optimizing performance.
   * *Start early!* It is possible to write an efficient malloc package with a few p ages of code. However, wecan guarantee that it will be some of the most difficult and sop histicated code you have written so far in your career. So start early, and good luck!

•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。我们包含了两个这样的跟踪文件（short1,2-bal.rep），您可以将它们用于初始调试。

•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还指示何时读取每个跟踪文件，这将帮助您隔离错误。

•使用gcc -g编译并使用调试器。调试器将帮助您隔离和识别超出范围的内存引用。

•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。使用这是出发点。在您了解有关简单隐式列表分配器的所有知识之前，不要开始使用分配器。

•将指针算法封装在C预处理器宏中。由于必须进行所有强制转换，因此内存管理器中的指针算术令人困惑且容易出错。通过为指针操作编写宏，可以大大降低复杂性。有关示例，请参见文本。

•分阶段实施。前9个跟踪包含对malloc和free的请求。最后2条跟踪包含对realloc，malloc和free的请求。我们建议您首先使malloc和free例程在前9个跟踪中正确有效地工作。只有这样，您才应将注意力转移到realloc实现上。首先，请在您现有的malloc和免费实现之上构建realloc。但是要获得真正好的性能，您将需要构建独立的重新分配。

•使用探查器。您可能会发现gprof工具有助于优化性能。

•尽早开始！可以编写带有少量代码的高效malloc程序包。但是，我们可以保证这将是您迄今为止在您的职业生涯中编写的最困难，最糟糕的代码。因此，尽早开始，祝你好运！

6