哈爾濱工業大學

实验报告

实 验（七）

题　　目 Dynamic Storage Allocator

动态内存分配器

专　　业 计算机类

学　　号 1161800218

班　　级 1636101

学 生 姓 名 陈翔

指 导 教 师 史先俊

实 验 地 点 G709

实 验 日 期 2017.12.29

计算机科学与技术学院

目 录

第 1 章 实验基本信息

1.1 实验目的

理解现代计算机系统虚拟存储的基本知识

掌握 C 语言指针相关的基本操作

深入理解动态存储申请、释放的基本原理和相关系统函数

用 C 语言实现动态存储分配器，并进行测试分析

培养 Linux 下的软件系统开发与测试能力

1.2 实验环境与工具

1.2.1 硬件环境

X64 CPU；2GHz；2G RAM；256GHD Disk 以上

1.2.2 软件环境

Windows7 64 位以上；VirtualBox/Vmware 11 以上；Ubuntu 16.04 LTS 64 位/优麒麟 64 位

1.2.3 开发工具

Ubuntu 16.04，CodeBlocks

1.3 实验预习

上实验课前，必须认真预习实验指导书（PPT 或 PDF）

了解实验的目的、实验环境与软硬件工具、实验操作步骤，复习与实验有关的理论知识。

熟知 C 语言指针的概念、原理和使用方法

了解虚拟存储的基本原理

熟知动态内存申请、释放的方法和相关函数

熟知动态内存申请的内部实现机制：分配算法、释放合并算法等

第 2 章 实验预习

总分 20 分

2.1 动态内存分配器的基本原理（5 分）

动态内存分配器维护着一个进程的虚拟内存空间，称为堆（heap）。系统之间细节不同，但是不失通用型。假设堆是一个请求二进制零的区域，它紧接着在未初始化的数据区域后开始，并向上生长（向更高的地址）。对于每个进程，内核始终维护着一个变量 brk，它指向堆的顶部。

分配器将堆视为一组不同大小的块（block）的集合来维护，每个块就是一个连续的虚拟内存片（chunk），要么是已分配的，要么是空闲的。已分配的块显式地保留为供应用程序使用，空闲块可用来分配。空闲块保持空闲，直到它显式地被应用所分配。一个已分配的块保持已分配的状态，直到它被释放，这种释放要么是应用程序显式执行的，要么是内存分配器自身隐式执行的。

2.2 带边界标签的隐式空闲链表分配器原理（5 分）

分配器需要一些数据结构来区别块边界，区别已分配的块和空闲块。大多数分配器将这些信息嵌入块本身。

一个块由一个字的头部，有效载荷以及可能的一些额外的填充组成的。头部包括块的大小，已分配还是空闲。头部后面就是应用调用 malloc 时请求的有效载荷。有效载荷后面是一片不使用的填充块，大小可以是任意的。

可以将堆组织成一个连续的已分配块和空闲块的序列。

这种结构为隐式空闲链表。空闲块是通过头部中的大小字段隐含地连接着的。

2.3 显示空间链表的基本原理（5 分）

将空闲块组织成某种形式的显式数据结构。实现这个数据结构的指针可以存放在这些空闲块的主体里面。例如：堆可以组织成一个双向空闲链表，在每个空闲块内，都包含一个前驱（pred）和一个后继（succ）指针。

2.4 红黑树的结构、查找、更新算法（5 分）

红黑树是一种具有红色和黑色链接的平衡查找树，同时满足： 红色节点向左倾斜 一个节点不可能有两个红色链接 整个书完全黑色平衡，即从根节点到所以叶子结点的路径上，黑色链接的个数都相同。

在二叉查找树的每一个节点上增加一个新的表示颜色的标记。该标记指示该节点指向其父节点的颜色。每个结点都只会有一条指向自己的链接（从它的父结点指向它），我们将链接的颜色保存在表示结点的 Node 数据类型的布尔变量 color 中（若指向它的链接是红色的，那么该变量为 true，黑色则为 false）。 当我们提到一个结点颜色时，我们指的是指向该结点的链接的颜色。

红黑树是一种特殊的二叉查找树，他的查找方法也和二叉查找树一样，不需要做太多更改。但是由于红黑树比一般的二叉查找树具有更好的平衡，所以查找起来更快。

旋转又分为左旋和右旋。通常左旋操作用于将一个向右倾斜的红色链接旋转为向左链接。对比操作前后，可以看出，该操作实际上是将红线链接的两个节点中的一个较大的节点移动到根节点上。

插入操作，一般地插入操作就是先执行标准的二叉查找树插入，然后再进行平衡化。

具体操作方式如下： 如果节点的右子节点为红色，且左子节点位黑色，则进行左旋操作 如果节点的左子节点为红色，并且左子节点的左子节点也为红色，则进行右旋操作 如果节点的左右子节点均为红色，则执行 FlipColor 操作，提升中间结点。

第 3 章 分配器的设计与实现

总分 50 分

3.1 总体设计（10 分）

介绍堆、堆中内存块的组织结构，采用的空闲块、分配块链表/树结构和相应算法等内容。

基于隐式空闲链表，使用立即边界标记合并方式。

mm_init 函数初始化分配器，如果成功就返回 0，失败就返回-1.

mm_malloc 函数和 mm_free 函数与它们对应的系统函数有相同的借口和语义。

第一个字是一个双字边界对齐的不使用的填充字。填充后面紧跟着一个特殊的序言块(prologue block)，这是一个 8 字节的已分配块，只由一个头部和一个脚部组成。序言块是在初始化时创建的，并且永不释放。在序言块后紧跟的是零个或者多个由 malloc 或者 free 调用创建的普通块。堆总是以一个特殊的结尾块(epilogue block)来结束，这个块是一个大小为零的已分配块，只由一个头部组成。序言块和结尾块是一种消除合并时边界条件的技巧。分配器使用一个单独的私有(static)全局变量(heap_listp)，它总是指向序言块。(作为一个小优化，我们可以让它指向下一个块，而不是这个序言块。)

分配器使用的块格式，最小的 16 字节，空闲链表将组织成一个隐式空闲链表。

字的大小（WSIZE），双字的大小（DSIZE），初始空闲块的大小和扩展堆时的默认大小（CHUNKSIZE）

定义一小组宏来访问和遍历空闲链表。

PACK 将大小和已分配的位结合起来并返回一个值，可以把它存放在头部或者脚部中。GET 读取和返回参数 p 引用的字。PUT 将 val 存放在参数 p 指向的字中。

GET 宏(第 12 行)读取和返回参数 p 引用的字。这里强制类型转换是至关重要的。参数 p 典型地是一个(viod*)指针，不可以直接进行间接引用。类似地，PUT 宏(第 13 行)将 val 存放在参数 p 指向的字中。

GET_SIZE 和 GET_ALLOC 宏(第 16～17 行)从地址 p 处的头部或者脚部分别返回大小和已分配位。剩下的宏是对块指针(block pointer，用 bp 表示)的操作，块指针指向第一个有效载荷字节。给定一个块指针 bp，HDRP 和 FTRP 宏(第 20～21 行)分别返回指向这个块的头部和脚部的指针。NEXT_BLKP 和 PREV_BLKP 宏(第 24～25 行)分别返回指向后面的块和前面的块的块指针。

可以用多种方式来编辑宏，以操作空闲链表。比如，给定一个指向当前块的指针 bp，我们可以使用下面的代码行来确定内存中后面的块的大小：

```
size_t size = GET_SIZE(HDRP(NEXT_BLKP(bp)));
```

3.2 关键函数设计（40 分）

3.2.1 int mm_init(void)函数（5 分）

函数功能：将对于堆来说可用的虚拟内存模型化为一个大的双子对齐的字节数组
处理流程：
初始化堆
从内存系统得到 4 个字，并将它们初始化，创建一个空的空闲链表。然后它调用 extend_heap 函数，这个函数将堆扩展 CHUNKSIZE 字节，并且创建初始的空闲块。此刻，分配器已经初始化了，并且准备好接受来自应用的分配和释放请求。
要点分析：
extend_heap 函数会在两个不同的环境下被调用：当堆初始化时；当 mm_malloc 不能找到一个合适的分配块时。为了保持对齐，entend_heap 将请求大小向上舍入为最接近 2 字（8 字节）的倍数，然后向内存系统请求额外的对空间。

3.2.2 void mm_free(void *ptr)函数（5 分）

函数功能：一个应用通过调用 mm_free 函数，来释放一个以前分配的块
参　　数：所请求的块 ptr
处理流程：通过边界标记合并技术预制邻接的空闲块合并起来
要点分析：

3.2.3 void *mm_realloc(void *ptr, size_t size)函数（5 分）

函数功能：调用 mm_realloc 是为了将 ptr 所指向内存块（旧块)的大小变为 size，并返回新内存块的地址。
参　　数：块 ptr，大小 size
处理流程：
- 如 ptr 是空指针 NULL,等价于 mm_malloc(size)
- 如果参数 size 为 0，等价于 mm_free(ptr)
- 如 ptr 非空, 它应该是之前调用 mm_malloc 或 mm_realloc 返回的数值，指向一个已分配的内存块。

注意：返回的地址与原地址可能相同，也可能不同，这依赖于算法的实现、旧块内部碎片大小、参数 size 的数值。新内存块中，前 min(旧块 size，新块 size)个字节的内容与旧块相同，其他字节未做初始化。

3.2.4 int mm_check(void)函数（5 分）

函数功能：堆的一致性检查，检查重要的不变量和一致性条件。当且仅当堆是一致的，才能返回非 0 值。
处理流程：
- 空闲列表中的每个块是否都标识为 free（空闲）？
- 是否有连续的空闲块没有被合并？

- 是否每个空闲块都在空闲链表中？
- 空闲链表中的指针是否均指向有效的空闲块？
- 分配的块是否有重叠？
- 堆块中的指针是否指向有效的堆地址？

提交代码文件 mm.c 的时候，将 mm_check 的所有调用注释，以免影响速度，降低吞吐率。

3.2.5 void *mm_malloc(size_t size)函数（10 分）

函数功能：应用通过调用 mm_malloc 函数来向内存请求大小为 size 字节的块
参　　数：块的大小 size
处理流程与要点分析：
在检查完请求的真假之后，分配器必须调整请求块的大小，从而为头部和脚部留有空间，并满足双字对齐的要求。if 条件强制了最小块大小是 16 字节；8 字节用来满足对齐要求，而另外 8 字节用来放头部和脚部。对于超过 8 字节的请求，一般的规则是加上开销字节，然后向上舍入到最接近 8 的整数倍。
一旦分配器调整了请求块的大小，它就会搜索空闲链表，寻找一个合适的空闲块。如果有合适的，那么分配器就会放置这个块，并可选地分割多余的部分，然后返回新分配块的地址。
如果分配器不能发现一个匹配的块，那么就用一个新的空闲块来扩展堆，把请求块放置在这个新的空闲块里，可选地分割这个块，然后返回一个指针，指向这个新分配的块。

3.2.6 static void *coalesce(void *bp)函数（10 分）

函数功能：将要回收的空闲块和临近的空闲块（如果有的话）合并成一个大的空闲块。
流程与要点：想要释放的块为当前快，那么合并内存中的下一个空闲块很简单并且高效。当前快的头部指向下一个块的头部，可以检查这个指针以判断下一个块是否是空闲的。如果是，那么就将它简单地指向下一个块的头部。

搜索整个链表，记住前面块的位置，直到我们到达当前块。

边界标记，在每个块的尾部添加一个脚部，脚部是头部的一个副本。分配器检查脚部，判断前一个块的起始位置和状态。

分析四种情况：

1.前面的块和后面的块都是已分配的

2.前面的块是已分配的，后面的块是空闲的

3.前面的块是空闲的，后面的是已分配的

4.前面的和后面的都是空闲的

1 中，不可能进行合并，只是简单地从已分配变成空闲；2 中当前块和后面的块合并，用当前块和后面块的大小来更新当前块的头部和后面块的脚部；3 中，前面的块和当前块合并，用两个块的大小的和来更新前面快的头部和当前块的脚部；4 中，合并所有的三个块称为一个单独的空闲块，用三个块的大小的和来更新前面块的头部和后面块的脚部。

第 4 章测试

总分 10 分

4.1 测试方法

make
./mdriver -v -t traces/

4.2 测试结果评价

中等水平

4.3 自测试结果

```
xiangxiang@xiangxiang-Lenovo-G50-75m:~/桌面/实验七/malloclab-handout/malloclab-h
andout$ make
gcc -Wall -O2 -m32   -c -o mm.o mm.c
gcc -Wall -O2 -m32 -o mdriver mdriver.o mm.o memlib.o fsecs.o fcyc.o clock.o ftimer.o
xiangxiang@xiangxiang-Lenovo-G50-75m:~/桌面/实验七/malloclab-handout/malloclab-h
andout$ ./mdriver -v -t traces
Team Name:chenxiang
Member 1 :chenxiang:1208141062@qq.com
Using default tracefiles in traces/
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.023560   242
 1       yes   99%    5848  0.021518   272
 2       yes   99%    6648  0.028138   236
 3       yes  100%    5380  0.021325   252
 4       yes   66%   14400  0.000272 52961
 5       yes   92%    4800  0.014396   333
 6       yes   92%    4800  0.013810   348
 7       yes   55%   12000  0.245985    49
 8       yes   51%   24000  0.580992    41
 9       yes   27%   14401  0.154835    93
10       yes   34%   14401  0.008627  1669
Total          74%  112372  1.113458   101

Perf index = 44 (util) + 7 (thru) = 51/100
```

第 5 章 总结

5.1 请总结本次实验的收获

知道了如何制造一个简单的分配器。

5.2 请给出对本次实验内容的建议

注：本章为酌情加分项。

参考文献

为完成本次实验你翻阅的书籍与网站等

[1] 林来兴. 空间控制技术[M]. 北京：中国宇航出版社，1992：25-42.

[2] 辛希孟. 信息技术与信息服务国际研讨会论文集：A 集[C]. 北京：中国科学出版社，1999.

[3] 赵耀东. 新时代的工业工程师[M/OL]. 台北：天下文化出版社，1998 [1998-09-26]. http://www.ie.nthu.edu.tw/info/ie.newie.htm（Big5）.

[4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨：哈尔滨工业大学，1992：8-13.

[5] KANAMORI H. Shaking Without Quaking[J]. Science，1998，279（5359）：2063-2064.

[6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science，1998，281：331-332[1998-09-23]. http://www.sciencemag.org/cgi/collection/anatmorp.