哈爾濱工業大學

实验报告

实 验（八）

题　　目 Dynamic Storage Allocator

动态内存分配器

专　　业 计算机

学　　号 _____

班　　级 _____

学 生 姓 名 _____

指 导 教 师 _____

实 验 地 点 _____

实 验 日 期 2021.6.10

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

Vscode

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区域，称为堆。分配器将堆视为一组不同大小的块的集合，来维护，每个块就是一个连续的虚拟内存片，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。

分配器有两种基本风格：显式分配器和隐式分配器。两种风格都要求应用显式地分配块。它们的不同之处在于由哪个实体来负责释放已分配的块。

显式分配器：要求应用显式地释放任何已分配的块。例如 C 程序通过调用 malloc 函数来分配一个块，通过调用 free 函数来释放一个块。C++中的 new 和 delete 操作符与 c 中的 malloc 和 free 相当。

隐式分配器：要求分配器检测一个已分配的块何时不再被程序所使用，那么久释放这个块。也叫做垃圾收集器，而自动释放未使用的块的过程叫做垃圾收集。例如，诸如 Lisp、ML、以及 Java 之类的高级语言就依赖垃圾收集来释放已分配的块。

2.2 带边界标签的隐式空闲链表分配器原理（5 分）

对于带边界标签的隐式空闲链表分配器，一个块是由一个字的头部、有效载荷、可能的一些额外的填充，以及在块的结尾处的一个字的脚部组成的。头部编码了这个块的大小（包括头部和所有的填充），以及这个块是已分配的还是空闲的。如果我们强加一个双字的对齐约束条件，那么块大小就总是 8 的倍数，且块大小的最低 3 位总是 0。因此，我们只需要内存大小的 29 个高位，释放剩余的 3 位来编码其他信息。在这种情况中，我们用其中的最低位（已分配位）来指明这个块是已分配的还是空闲的。

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，其大小可以是任意的。需要填充有很多原因。比如，填充可能是分

配器策略的一部分，用来对付外部碎片。或者也需要用它来满足对齐要求。

我们将对组织为一个连续的已分配块和空闲块的序列，这种结构称为隐式空闲链表，是因为空闲块是通过头部中的大小字段隐含地连接着的。分配器可以通过遍历堆中所有的块，从而间接地遍历整个空闲块的集合。注意：此时我们需要某种特殊标记的结束块，可以是一个设置了已分配位而大小为零的终止头部。

Knuth 提出了一种边界标记技术，允许在常数时间内进行对前面快的合并。这种思想是在每个块的结尾处添加一个脚部，其中脚部就是头部的一个副本。如果每个块包括这样一个脚部，那么分配器就可以通过检查它的脚部，判断前面一个块的起始位置和状态，这个脚部总是在距当前块开始位置一个字的距离。

2.3 显示空间链表的基本原理（5 分）

显式空闲链表结构将堆组织成一个双向空闲链表，在每个空闲块的主体中，都包含一个 pred（前驱）和 succ（后继）指针。

使用双向链表而不是隐式空闲链表，使首次适配的分配时间从块总数的线性时间减少到了空闲块数量的线性时间。不过，释放一个块的时间可以是线性的，也可能是个常数，这取决于空闲链表中块的排序策略。

一种方法是用后进先出（LIFO）的顺序维护链表，将新释放的块放置在链表的开始处。另一种方法是按照地址顺序来维护链表，其中链表中每个块的地址都小于它后继的地址。

2.4 红黑树的结构、查找、更新算法（5 分）

红黑树的结构：

红黑树是一种近似平衡的二叉查找树，它能够确保任何一个节点的左右子树的高度差不会超过二者中较低那个的一陪。具体来说，红黑树是满足如下条件的二叉查找树（binary search tree）：

1. 每个节点要么是红色，要么是黑色。
2. 根节点必须是黑色
3. 红色节点不能连续（也即是，红色节点的孩子和父亲都不能是红色）。
4. 对于每个节点，从该点至 null（树尾端）的任何路径，都含有相同个数的黑色节点。

在树的结构发生改变时（插入或者删除操作），往往会破坏上述条件 3 或条件 4，需要通过调整使得查找树重新满足红黑树的条件。

红黑树的查找：

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。红黑树的主要是想对 2-3 查找树进行编码，尤其是对 2-3 查找树中的 3-nodes 节点添加额外的信息。红黑树中将节点之间的链接分为两种不同类型，红色链接，他用来链接两个 2-nodes 节点来表示一个 3-nodes 节点。黑色链接用来链接普通的 2-3 节点。特别的，使用红色链接的两个 2-nodes 来表示一个 3-nodes 节点，并且向左倾斜，即一个 2-node 是另一个 2-node 的左子节点。这种做法的好处是查找的时候不用做任何修改，和普通的二叉查找树相同。

红黑树的插入：

情况 1：红黑树为空树

这种情况直接插入到根节点就行，然后通过变色将插入节点变为黑色。因为红黑树根节点必须是黑色。

把插入结点作为根结点，并把结点设置为黑色。

情况 2：插入节点的 key 存在

这种情况不改变当前节点颜色，直接更新对应节点的 value 值即可。

把 I 设为当前结点的颜色

更新当前结点的值为插入结点的值

情况 3：插入节点的父节点是黑色节点。

因为红黑树的平衡是黑色平衡，所以这种在黑色节点下面插入红节点不会违背红黑树的性质。

直接插入

情况 4：插入节点父节点是红色节点

这种情况就是红黑树的主要要讨论的情况了。因为根节点是黑色，所以父节点是红色时，插入节点必定有祖父节点。所以必定可以进行左旋 or 右旋。

红黑树的更新：

情况 1：替换节点是红色节点

这种情况最简单，直接删除掉替换节点即可，不会影响红黑树平衡。

情况 2：替换节点是黑色节点

如果替换节点时黑色节点，那么删除替换节点必然会影响子树的平衡，因此必须做自平衡处理。

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。

1.堆：堆区是一个动态的存储区域，使用库函数 malloc()和 free()，和操作符 new 和 delete 以及一些相关变量来进行分配和回收，在堆区中，对象的生命周期可以比它存在内存中的生命周期短，换句话说：程序可以获得一片内存区域而不用马上对它进行初始化，同时，在对象被销毁后，也不用马上收回它所占用的内存区，在这段时间内，用户可以还可以用 void*型的指针访问这片区域，但是原始对象的非静态区以及成员函数都不能被访问或者操纵，因为我们知道实际上对象已经不存在了。

2.堆中内存块的组织结构：用隐式空闲链表来组织堆，对于带边界标签的隐式空闲链表分配器，一个块是由一个字的头部、有效载荷、可能的一些额外的填充，以及在块的结尾处的一个字的脚部组成的。头部编码了这个块的大小（包括头部和所有的填充），以及这个块是已分配的还是空闲的。如果我们强加一个双字的对齐约束条件，那么块大小就总是 8 的倍数，且块大小的最低 3 位总是 0。因此我们可以使用最低三位来记录其他我们需要的信息。

3.空闲块与分配块链表：采用的是分离的空闲链表，使用 LIST 数组来储存我们分配或未分配的块，当需要的时候找到合适的组之后找到合适的块进行分配。

放置策略：首次适配。经过测试首次适配的效果已经相当不错，因此不再使用其他的放置方法进行优化。

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：初始化内存分配系统。

处理流程：

第一步：初始化分离空闲链表，把申请的空间的内容全部填为 NULL

第二步：初始化堆，并创建一个空的空闲链表，创建空的空闲链表可以分为如下几个步骤：

1. 第一个字用于对齐
2. 第二部分是用于指向分配的空间的大小的序言块，也就是头部和脚部用于存储空间大小的块
3. 第三部分是用于标记结尾的块,初始化时表示这是一个大小为 0 的已分配空间。

要点分析：

1. 在空闲链表创建之后需要使用 extend 函数来扩展堆的大小。

2. 初始化空的空闲链表的时候是需要遵守对齐的，因此第一个字是用于对齐的字。

3.2.2 void mm_free(void *ptr)函数（5 分）

函数功能：释放一个块
参　　数：指向需要释放的块的首地址的指针。
处理流程：首先使用函数获得已分配块的大小，之后将头部与脚部记录的字段设置为大小为 0，标志这个块已经被释放了。在释放成功之后将空闲块插入到空闲链表中，并使用 coalesce 进行有可能需要的将这个刚释放的空闲块与前后的空闲块的合并。
要点分析：在释放空闲块之后由于可能后续还需要进行分配这一块空间，因此需要将其插入空闲链表中，同时为了简化操作，采用的合并策略是立即合并，在每次释放空闲块之后都需要判断是否需要进行合并，如果需要的话将其余前面或者后面的空闲块进行合并。

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

函数功能：向 ptr 指向的地址重新分配一个 size 大小的载荷。
参　　数：ptr 是需要分配的载荷的首地址的指针，size 表示需要分配的载荷的大小。
处理流程：
第一步：与 malloc 的基本步骤一致，首先判断是否需要重新分配，需要注意的是分配强制 8 字节对齐，最小的大小是 16 字节，其中 8 字节作为分配的载荷使用，其余 8 字节用于记录头部与脚部信息，因此当申请的空间不是 8 的整数倍的时候向上舍入到最近的 8 的整数倍。
第二步：在分配载荷的时候可以分为如下几种情况：
1. 如果需要分配的空间比原来的块小，那么返回原来的块
2. 如果比原来的块大，那么判断地址的连续的下一个空间是不是未分配的块，尽量使用临近的块，减小碎片的产生；如果加上下一个空间满足要求那么就这样操作，如果还是不满足则需要扩展堆。
3. 如果已经不存在连续地址上的可用空间，就只能申请使用非连续地址上的空间。

要点分析：与 malloc 的要点类似，在重新分配载荷的时候需要考虑到对齐的问题，因此需要在适当的时候修改 size 的大小；同时需要注意的是为了尽量减小分配时产生的碎片问题，每次在重新分配空间的时候如果原有的空间不够应该优先考虑连续空间上的分配。

3.2.4 int mm_check(void)函数（5 分）

函数功能：检查堆的一致性
处理流程：主要的目的是为了查看在创建和扩展堆的时候是否按照我们预先定义好的方法进行操作。最初是检查序言块，如果序言块不是 8 字节的已分配块，则会打印 Bad prologue header。接着检查是否进行双字对齐，并通过获得每一个块的头部和脚部信息匹配是否一致，如果对齐的格式不对或是头部和脚部信息不一致的话输出报错信息；最后检查的是结尾快，如果结尾块不是一个标志已分配的块的输出报错信息。
要点分析：这个函数的主要功能是检测堆的一致性，主要检查的就是序言块，头部和脚部信息以及结尾块的信息是否满足要求。

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

函数功能：向内存请求分配一块至少 size 大小的载荷
参　　数：size 标志需要分配的载荷的最小大小。
处理流程：
1.和 remalloc 的第一步类似，在判断命令的真伪之后首先判断是否需要重新分配，需要注意的是分配强制 8 字节对齐，最小的大小是 16 字节，其中 8 字节作为分配的载荷使用，其余 8 字节用于记录头部与脚部信息，因此当申请的空间不是 8 的整数倍的时候向上舍入到最近的 8 的整数倍。
2.在空闲链表中搜索是否有满足需要的块
3.如果有满足要求的块则使用 place 函数来分配这一载荷，返回分配的内存空间的首地址；如果搜索之后发现没有满足条件的内存空间的话使用一个空的空闲链表申请新的空间，并将新申请的空间中分配合适的大小给需要的块之后返回分配的内存空间的首地址。
要点分析：注意每次都需要使用 place 函数来处理需要分配的空间，主要目的是从空闲的空间中分割出合适的大小给需要分配的块。

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

函数功能：边界标记合并
处理流程：这一部分主要实现的就是书上说的释放已分配块之后需要边界合并的四种情况，四种情况叙述如下；
1.前面的和后面的块都已分配。此时无法合并，直接返回当前块即可。
2.前面的块已分配，后面的块空闲。先把当前块和后面的块从分离空闲链表中删除。然后此时当前块和后面的块合并，用当前块和后面块的大小的和来更新当前块的头部和脚部。
3.前面块空闲，后面块已分配。这时和上面类似，先把当前块和前面块从分离链表中删除。然后此时将前面块和当前块合并，用两个块大小的和来更新前面块的头部和当前块的脚部。
4.前面块和后面块都空闲。先把前面块、当前块和后面块从分离链表中删除。然后

合并所有的三个块形成一个单独的空闲块，用三个块大小的和来更新前面块的头部和后面块的脚部。

要点分析：在合并的时候需要先将需要合并的块从空闲链表中删除之后合并完成之后再将合并之后的块插入空闲链表中。

第 4 章测试

总分 10 分

4.1 测试方法

生成可执行评测程序文件的方法：linux>make
评测方法:mdriver [-hvVa] [-f <file>]
选项：

-a	不检查分组信息
-f <file>	使用 <file>作为单个的测试轨迹文件
-h	显示帮助信息
-l	也运行 C 库的 malloc
-v	输出每个轨迹文件性能
-V	输出额外的调试信息

轨迹文件：指示测试驱动程序 mdriver 以一定顺序调用
性能分 pindex 是空间利用率和吞吐率的线性组合
获得测试总：linux>./mdriver -av -t traces/

4.2 自测试结果

最终测试的分数为 87 分，优化的还不是很到位，但是已经减少了很多碎片的出现与搜索合适空间所需要的时间，因此不再进行优化。

4.3 测试结果评价

```
zsh@zsh-virtual-machine:~/code/c/lab8/malloclab-handout-hit$ ./mdriver -t traces
/ -v
Team Name:1190300321
Member 1 :Zheng Shenghe:531905990@qq.com
Using default tracefiles in traces/
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   44%    5694  0.003393  1678
 1       yes   31%    5848  0.003205  1824
 2       yes   50%    6648  0.006053  1098
 3       yes   68%    5380  0.005537   972
 4       yes   99%   14400  0.000383 37559
 5       yes   94%    4800  0.000800  6004
 6       yes   93%    4800  0.000749  6413
 7       yes   95%   12000  0.007972  1505
 8       yes   88%   24000  0.007050  3404
 9       yes   99%   14401  0.000213 67547
10       yes   98%   14401  0.000168 85669
Total          78%  112372  0.035523  3163

Perf index = 47 (util) + 40 (thru) = 87/100
```

第 5 章 总结

5.1 请总结本次实验的收获

对于动态内存分配器有了基本了解。

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.