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 计算机类

学　　号 1190501614

班　　级 1903006

学 生 姓 名 cgh

指 导 教 师 史先俊

实 验 地 点 G709

实 验 日 期 2021.6.15

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。

显式分配器：要求应用显式地释放任何已分配的块。C 语言中通过调用 free 函数来释放一个块。

隐式分配器：也叫做垃圾收集器，系统自动回收内存垃圾。

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和尾部（标记）、有效载荷、以及可能的填充组成的。头部编码了这个块的大小（包括头部和所有的填充），以及这个块是否已分配的标志位。强加一个双字的对齐约束条件后，块大小就总是 8 的倍数。我们用 29 个高位存储块大小信息，低 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）来指明这个块是否被分配

头部后面就是调用 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。

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 显式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，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。其中的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，通过一些着色和特殊约束，能够在对数复杂度时间内查找、删除、插入。红黑树是满足如下条件：

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。
5. 每个叶结点（包 NILL）是黑色的

- 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类似，通过比较当前结点的值与目标值决定查找左子树还是右子树。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。

- 红黑树的插入和更新：

每次插入一个结点，都将其着为红色，但可能违反红黑性质，通过旋转操作来保持红黑树的性质，插入时可分为三种情况进行调整：

1. 当前节点的父节点是红色，且当前节点的祖父节点的另一个子节点（叔节点）也是红色。

   操作：将父节点设为黑色, 将叔节点设为黑色, 将祖父节点设为红色, 将祖父节点设为当前节点(红色节点), 之后继续对当前节点进行操作。
2. 当前节点的父节点是红色，叔叔节点是黑色，且当前节点是其父节点的右

孩子

操作：将父节点作为新的当前节点。以新的当前节点为支点进行左旋。

3. 当前节点的父节点是红色，叔叔节点是黑色，且当前节点是其父节点的左孩子

操作：将父节点设为黑色；将祖父节点设为红色；以祖父节点为支点进行右旋。

- 红黑树结点删除

1. 将红黑树当作一颗二叉查找树，将节点删除。

1. 被删除节点没有儿子，即为叶节点。那么，直接将该节点删除就 OK 了。

2. 被删除节点只有一个儿子。那么，直接删除该节点，并用该节点的唯一子节点顶替它的位置。

3. 被删除节点有两个儿子。那么，先找出它的后继节点；然后把“它的后继节点的内容”复制给“该节点的内容”；之后，删除“它的后继节点”。

2. 通过"旋转和重新着色"等一系列来修正该树，使之重新成为一棵红黑树。

第 3 章 分配器的设计与实现

总分 50 分

3.1 总体设计（10 分）

- **堆**

动态内存分配器维护着一个进程的虚拟内存区域，称为堆。简单来说，动态分配器就是我们平时在 C 语言上用的 malloc 和 free,realloc，通过分配堆上的内存给程序，我们通过向堆申请一块连续的内存，然后将堆中连续的内存按 malloc 所需要的块来分配，不够了，就继续向堆申请新的内存，也就是扩展堆，这里设定，堆顶指针想上伸展（堆的大小变大）。

- **堆中内存块的组织结构**

用隐式空闲链表来组织堆，具体组织的算法在 mm_init 函数中。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。我们强加一个双字的对齐约束条件后，块大小就总是 8 的倍数，且块大小的最低 3 位总是 0。因此，我们只需要用 29 个高位存块大小，用剩余的 3 位来编码其他信息。我们用最低位（已分配位）来指明这个块是否已分配。

- **对于空闲块和分配块链表**

采用分离的空闲链表。全局变量 void *Lists[MAX_LEN]， 因为一个使用单向空闲块链表的分配器需要与空闲块数量呈线性关系的时间来分配块，而此堆的设计采用分离存储的来减少分配时间，就是维护多个空闲链表，每个链表中的块有大致相等的大小。将所有可能的块大小根据 2 的幂划分。

- **算法**

辅助函数

static void AddNode(void *bp, size_t size)

向空闲块链表中添加块 bp，大小为 size。通过块的大小找到对应的空闲链表，然后开始遍历直到找到合适的位置（维护链表的有序性），然后分情况加入块。

static void DeleteNode(void *bp)

向空闲块链表中删除块 bp。通过找到对应的空闲链表，然后开始扫描链表找到合适位置，分情况删除相应的块。

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：初始化空闲链表和堆
处理流程：

1. 初始化空闲链表
2. 初始化堆，用八个字节初始化头部和脚部，用 4 个字节初始化堆的结尾标志
3. 然后调用 extend_heap 函数扩展堆的大小

要点分析：

1. 初始化空闲链表时，全指向 NULL
2. 初始化对注意判断内存分配是否成功
3. 最后要记得扩展堆的大小

3.2.2 void mm_free(void *ptr)函数（5 分）

函数功能：释放内存块
参　　数：void* ptr
处理流程：1　首先获得内存块的大小
2　将块的头部和脚部置 0
3　将块加入到空闲链表
4　合并空闲块

要点分析：1　注意释放之后要加入到空闲链表中
2　注意最后需要合并空闲块

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

函数功能：向 ptr 指向的块重新分配大小至少为 size 的块
参　　数：void *ptr，size_t　size
处理流程：1. 判断 size 的大小，然后根据 8 对齐原则重置 size 的大小
2. 如果块的大小大于等于 size，则直接返回 ptr
3. 如果相邻的下一个块是未分配，则加上下一个块，如果依旧不够，则需要重新申请新的空闲块，并将内容复制到该块，释放之前的块
4. 如果内存不够，则需要扩展堆的大小

要点分析：1. 注意需要内存对齐，调整 size 的大小
2. 先查找相邻的内存块，减少外部内存碎片
3. 然后再判断是否需要新的内存块

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：1. 指向序言块的全局变量，检查序言块是否 8 字节已分配
2 判断块的头部和脚部，检查双字对齐
3 如果 verbose 不为 0，打印块信息
4 最后检查结尾块是否大小为 0 的已分配块

要点分析：主要检查序言块，结尾块，双字对齐，头部和脚部是否匹配

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

函数功能：分配大小至少为 size 的内存块
参　　数：size_t　size
处理流程：1 调整 size 的大小，强制 8 字节对齐
2 在空闲链表中找到合适的空闲块，然后放置
3 如果找不到，则扩展堆的大小后再分配

要点分析：注意调整 size 为 8 字节对齐，最后要判断是否需要扩张堆的大小

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

函数功能：合并空闲块
处理流程：1. 主要分四种情况合并
2 前后都是已分配，无需合并
3 前面已分配，后面未分配，和后面的块合并
4 前面未分配，后面已分配，和前面的块合并
5 前后均未分配，全部合并为一块

要点分析：注意最后要将这个块加入到空闲链表中，
每次需要合并时，需要先删除已加入的对应块

第 4 章测试

总分 10 分

4.1 测试方法

$:make
$:./mdriver -av -t traces/

4.2 测试结果评价

测试结果 95 分，除了采取 ppt 上方案 1 的优化以外，还采取了一些细节优化，减少不必要的循环，调整每次堆扩展的大小，对一些特殊的测试文件 7 和 8 有很大的改善。

4.3 自测试结果

```
cgh1190501614@ubuntu:~/Desktop/hitics/HITICS-Lab8 malloc/malloclab-handout/malloclab-handout$ ./mdriver -av -t traces/
Using default tracefiles in traces/
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   97%    5694  0.000277 20571
 1       yes   98%    5848  0.000220 26618
 2       yes   99%    6648  0.000208 31931
 3       yes   99%    5380  0.000174 30849
 4       yes   66%   14400  0.000208 69331
 5       yes   94%    4800  0.000263 18286
 6       yes   91%    4800  0.000246 19552
 7       yes   95%   12000  0.000290 41422
 8       yes   88%   24000  0.002614  9180
 9       yes   99%   14401  0.000157 91551
10       yes   86%   14401  0.000149 96457
Total          92%  112372  0.004805 23385

Perf index = 55 (util) + 40 (thru) = 95/100
cgh1190501614@ubuntu:~/Desktop/hitics/HITICS-Lab8 malloc/malloclab-handout/malloclab-handout$
```

第 5 章 总结

5.1 请总结本次实验的收获

了解分配器的基本原理

5.2 请给出对本次实验内容的建议

参考文献