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 计算机科学与技术

学　　号 1170300520

班　　级 1703005

学 生 姓 名 郭子阳

指 导 教 师 吴 锐

实 验 地 点 G712

实 验 日 期 2018.12.16

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

X86-64 CPU；2GHz；2G RAM；256GHD Disk

1.2.2 软件环境

Windows10 64 位；Vmware 15；Ubuntu 18.04 LTS 64 位

1.2.3 开发工具

gdb ； CodeBlocks

1.3 实验预习

1、上实验课前，必须认真预习实验指导书（PPT 或 PDF）

2、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。

3、熟知 C 语言指针的概念、原理和使用方法

4、了解虚拟存储的基本原理

5、熟知动态内存申请、释放的方法和相关函数

6、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）

动态存储分配器维护着一个进程的虚拟存储器区域，称为堆。堆是从低位地址向高位向上增长的，对于每个进程，内核维护着一个 brk，它指向堆的顶部。

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存储器片，要么是已分配的，要么是空闲的。已分配的显示地保留为供应应用程序使用。空闲块可用来分配。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示执行的，要么是存储分配器隐式执行的，它们的都是显式的来分配存储块的，不同之处在于由哪个实体来负责释放已分配的块。

a) 显式分配器，要求显式的释放已分配的块。如 C 标准库中的 malloc 和 free，C++中的 new 和 delete 操作符。

b) 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使用，那么就释放这个块。隐式分配器也叫做垃圾收集器(Grabage collector)。如 java 语言就依赖于类似分配器。

2.2 带边界标签的隐式空闲链表分配器原理（5 分）

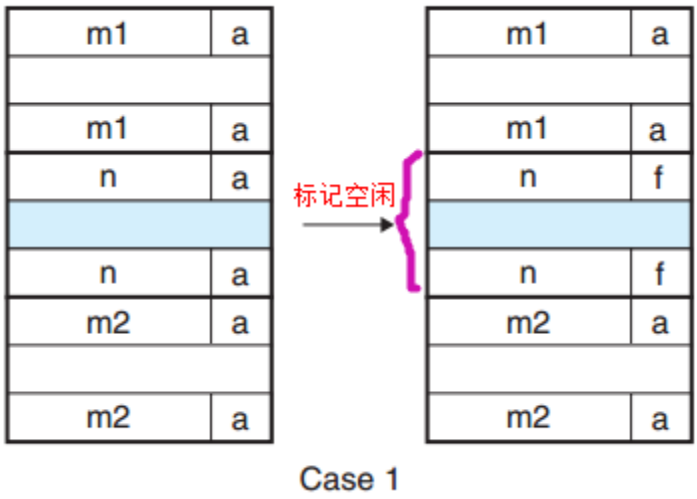
带边界标记主要是为了合并较为方便。

我们需要从新审视一下我们的隐式链表数据结构，加入新的边界标记形成如下结构：

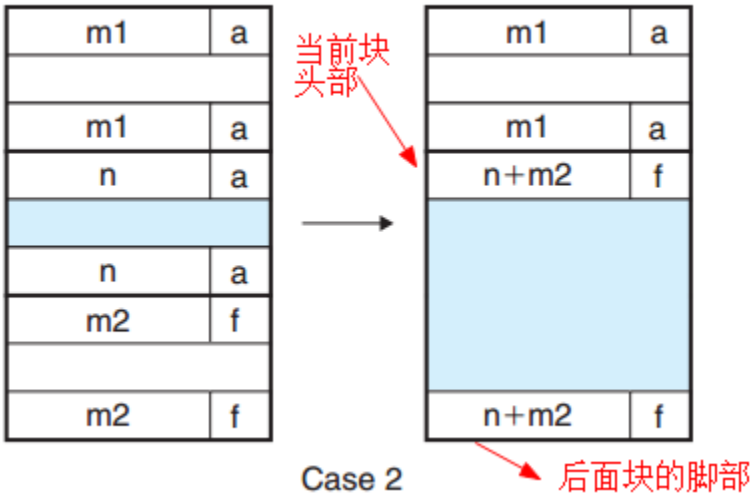
在链表的底部加入头部同样的格式，用 a 表示已分配、f 表示空闲

我们列举一下可能的所有情况：

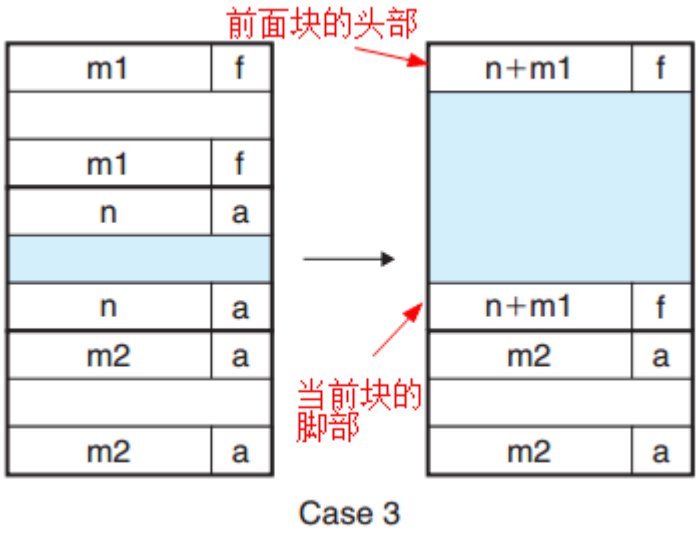
说明：



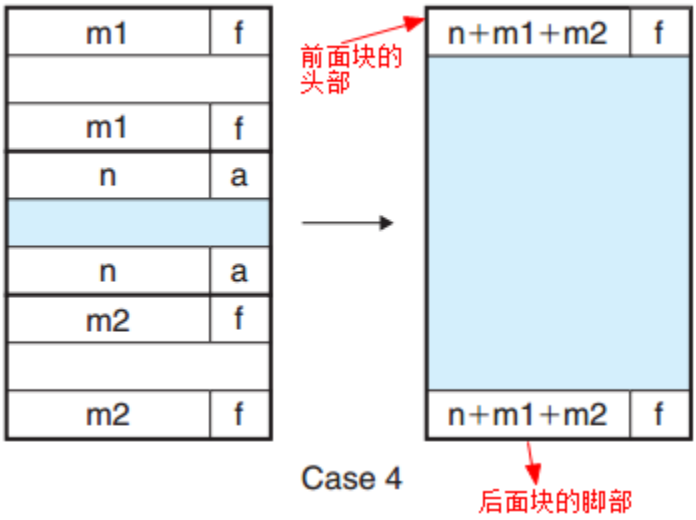
（a）：在合并的时候，由于前后都是已分配不执行合并，只是把当前块标记位空闲：



（b）：后面的块是空闲的，当前块与后面的块合并，用新的块的大小（当前块大小+后面块大小），更新当前块的头部和后面块的脚部



（c）：前面块是空闲，前面块与当前块合并，用新的块的大小（当前块的大小+前面块的大小），更新前面块的头部和当前块的脚部



（d）：三个块都是空闲，3 个块的大小来更新前面块的头部和后面块的脚部

注意：当（c）和（d）两种情况，前面的块是空闲的，才需要用到当前块的脚部。（a）不需要更新，（b）更新的是后面块的脚部+块大小。如果我们把前面的块的位存放在当前块头部未使用多出来的低位中，那么已分配的块就不需要脚部了。（当然空闲块仍然需要脚部）

2.3 显式空闲链表的基本原理（5 分）

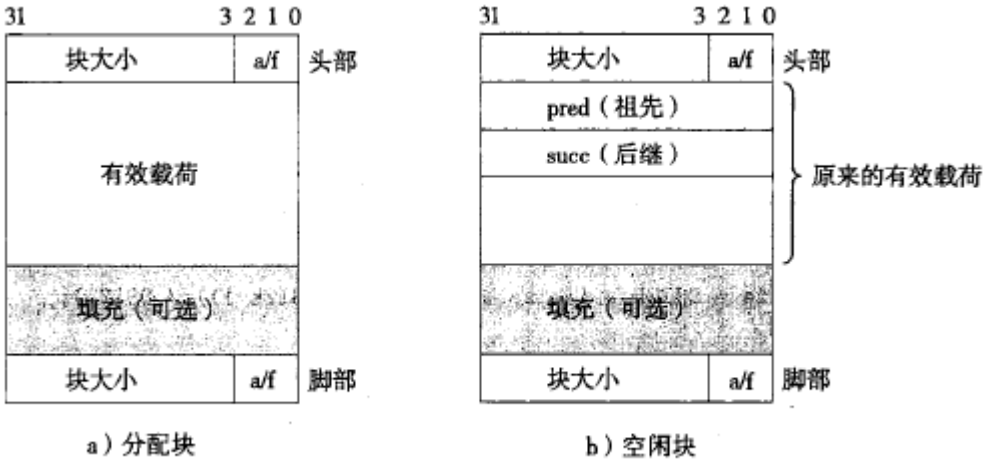


图 9-48　使用双向空闲链表的堆块的格式

上图中的显式空闲链表，把分配块和空闲块用不同的数据结构进行组织，每个空闲块中，包含一个前驱和后继指针，所有的空闲块形成了一个双向空闲链表，如果我们依然采取首次适配的方式，那么分配时间就可以从块总数的线性时间减少到了空闲块数量的线性时间。

2.4 红黑树的结构、查找、更新算法（5 分）

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。

查找：

假定红黑树的根节点指针为 root，给定的关键字值为 key，则查找算法可描述为：

1. 置初值：p = root ；
2. 如果 key = p -> data ，则查找成功，算法结束；
3. 否则，如果 key < p->data ，而且 p 的左子树非空，则将 p 的左子树根送 p ，转步骤 2 ；否则，查找失败，算法结束；

4. 否则，如果 key > p->data ，而且 p 的右子树非空，则将 p 的右子树根送 p ，转步骤 2 ；否则，查找失败，算法结束。

第 3 章 分配器的设计与实现

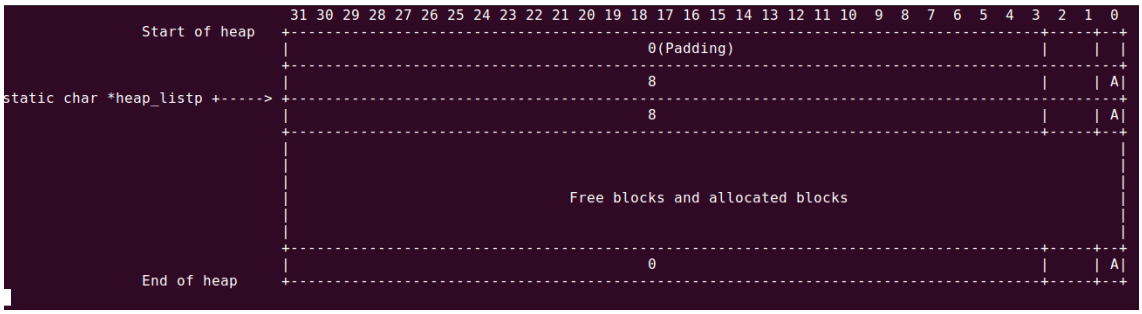
总分 50 分

3.1 总体设计（10 分）

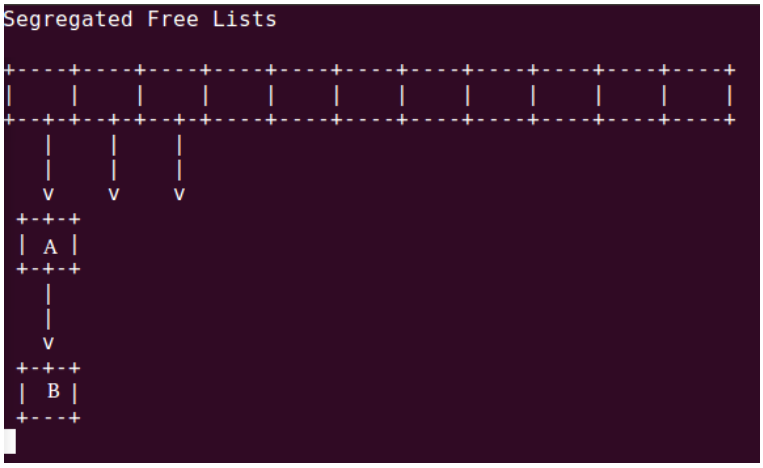
介绍堆、堆中内存块的组织结构，采用的空闲块、分配块链表/树结构和相应算法等内容。

块的结构如下，其中低三位由于内存对齐的原因总会是 0，A 代表最低位为 1，即该块已经 allocated：

堆的起始和结束结构如下：



Free list 的结构如下，每条链上的块按大小由小到大排列，这样我们用“first hit”策略搜索链表的时候就能获得“best hit”的性能，例如第一条链，A 是 B 的 successor，B 是 A 的 predecessor，A 的大小小于等于 B；不同链以块大小区分，依次为{1}{2}{3~4}{5~8}...{1025~2048}... ：



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：初始化堆结构
处理流程：先初始化分离空闲链表，再初始化堆结构，最后扩展堆
要点分析：
在初始化堆结构的最后还需要扩展一下堆

3.2.2 void mm_free(void *ptr)函数（5 分）

函数功能：释放一个块
参　　数：块的头指针
处理流程：将块设为空闲的，并插入分离空闲链表
要点分析：
插入空闲链表后还需要和前后的空闲块合并

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

函数功能：调整块
参　　数：快的指针，块的大小
处理流程：首先检查一下内存对齐。如果 size 小于原来的块大小则直接返回原来的块，否则检查地址的下一块是否 free，如果即使加上后面连续地址上的 free 块空间也不够，则需要扩展块。如果没有可以利用的连续 free 块，而且 size 大于原来的块，这时只能申请新的不连续的 free 块、复制原块内容、释放原块。
要点分析：
要考虑到申请不连续的空闲块的情况

3.2.4 int mm_check(void)函数（5 分）

函数功能：扫描堆并检查其状态
处理流程：遍历堆中的块，检查是否对齐，每个块的头和脚的信息是否相同等
要点分析：
还需要保证没有两个连续的空闲块，如果发现应该及时合并

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

函数功能：申请一个块
参　　数：申请块的大小
处理流程：寻找大小对应的链，并在链上找对应大小的块
要点分析：
在找不到对应大小的块时，需要进行堆扩展

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

函数功能：合并相邻的空闲块
处理流程：分四种情况遍历合并空闲块
要点分析：
合并完成的空闲块要加入空闲链表

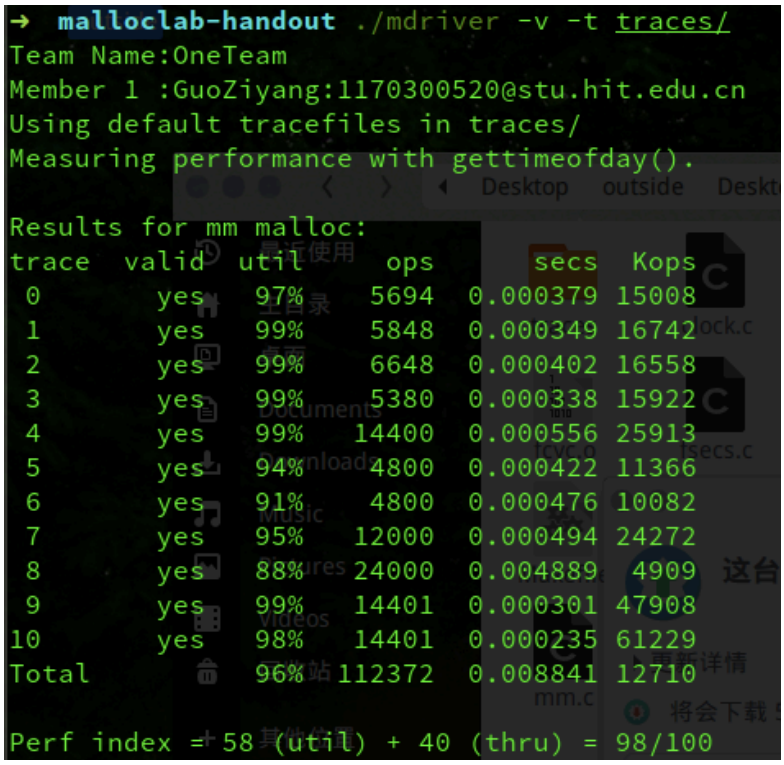
第 4 章测试

总分 10 分

4.1 测试方法

./mdriver -v -t traces/，测试 traces 下所有的轨迹文件

4.2 测试结果评价



4.3 自测试结果

测试结果较好，有几种情况无法处理碎片导致减分

第 5 章 总结

5.1 请总结本次实验的收获

深入理解了动态内存分配器的原理与实现方法

5.2 请给出对本次实验内容的建议