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 1170301027

班　　级 1703010

学　　生 冯帅

指 导 教 师 史先俊

实 验 地 点 **G712**

实 验 日 期 20181216

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

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）

分配器将堆视为一组不同大小的块 (blocks)的集合来维护，每个块要么是已分配的，要么是空闲的。

动态内存分配器的类型有两种：

1. 显式分配器 ： 要求应用显式地释放任何已分配的块
2. 隐式分配器 ： 应用检测到已分配块不再被程序所使用，就释放 这个块

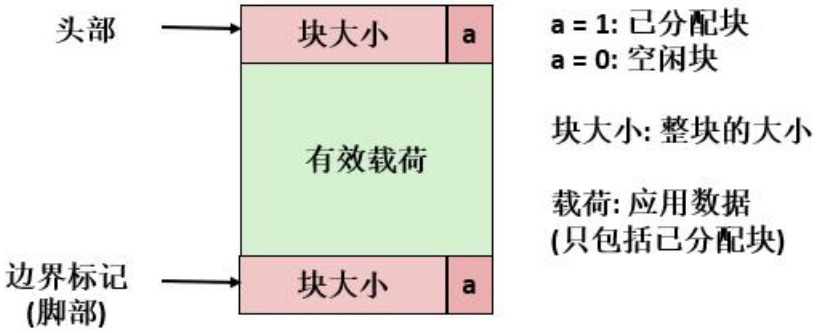
基本的实现原理就是维护一个内存空闲链表，当申请内存空间时，搜索内存空闲链表，找到适配的空闲内存空间，然后将空间分割成两个内存块，一个变成分配块，一个变成新的空闲块。如果没有搜索到，那么就会用 sbrk()才推进 brk 指针来申请内存空间。

搜索空闲块最常见的算法有：首次适配，下一次适配，最佳适配（详见 2.2 节）。

2.2 带边界标签的隐式空闲链表分配器原理（5 分）

隐式空闲链表（边界标记）：通过头部中的大小字段隐含的连接空闲块

1.堆及堆中内存块的组织结构：



隐式空闲链表中堆中块组织结构

2.适配方法：

首次适配 (First fit):

从头开始搜索空闲链表，选择第一个 合适的空闲块:

可以取总块数 (包括已分配和空闲块) 的线性时间

在靠近链表起始处留下小空闲块的 “碎片”；

下一次适配 (Next fit):

和首次适配相似，只是从链表中上一次查询结束的地方开始

比首次适应更快: 避免重复扫描那些无用块

一些研究表明，下一次适配的内存利用率要比首次适配低得多；

最佳适配 (Best fit):

查询链表，选择一个 最好的 空闲块： 适配，剩余最少空闲空间

保证碎片最小——提高内存利用率

通常运行速度会慢于首次适配；

3.分割空闲块：

在分配块小于空闲块的时候我们可以把空闲块分割成两部分；

4. 释放已分配块：

在程序中不没有用的块或者已经用完了的块需要释放回收；

5. 合并相邻的空闲块：

立即合并 (Immediate coalescing): 每次释放都合并

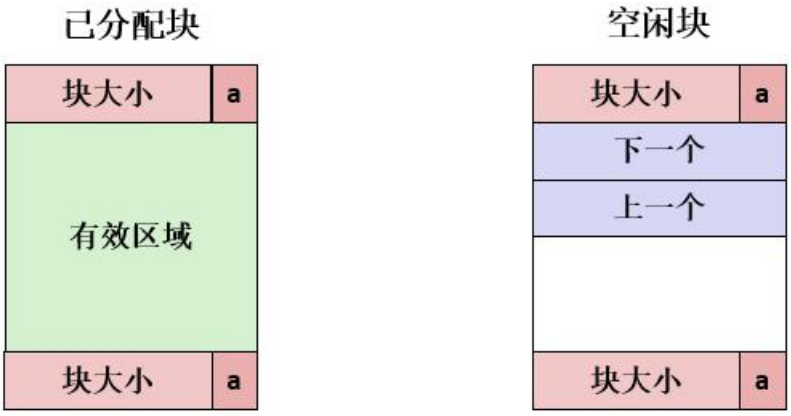
延迟合并 (Deferred coalescing): 尝试通过延迟合并，即直到需要才合并来提高释放的性能.例如:为 malloc 扫描空闲链表时可以合并；外部碎片达到阈值时可以合并

需要合并块的四种情况

2.3 显示空间链表的基本原理（5 分）

显示空闲链表：在空闲块中用指针连接空闲块

堆中块的结构：



显示空闲链表中堆中块的结构

将空闲块组织成链表形式的数据结构。堆可以组织成一个双向空闲链表，在每个空闲块中，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。

操作大致上与隐式空闲链表所要完成的操作相同，但要点是，我们只保留空闲块链表，而不是所有块；再者由于无法确定下一个块的位置以及大小（可以在任何地方），故而我们需要存储空闲块前后指针，而不仅仅是大小；同时需要合并边界标记；

插入原则：针对已释放的块，我们有

LIFO（last-in-first-out）policy,后进先出法：

将新释放的块放置在链表开始处；

地址顺序法：

按照地址顺序维护链表：

Addr（祖先）<Addr（当前回收快）<Addr（后继）

2.4 红黑树的结构、查找、更新算法（5 分）

红黑书树是一种自平衡的二叉查找树，红黑树和 AVL 树类似，都是在进行插入、删除操作时通过特定操作保持二叉查找树的平衡，从而获得更高的查找性能。

数据结构设计：用抽象的数据类型表示红黑树的结点，使用指针保存结点间的相互关系，基本属性有：结点的颜色、左子节点指针、右子节点指针、父节点指针、结点的值。一棵红黑树必须满足以下几个条件：根节点必须是黑色；任意从根到叶子结点的路径不包含连续的红色结点；任意从根到叶子的路径的黑色节点的数目相同。

```
public class RBTNODE{
    keytype key;//关键字
    colortype color;//颜色设置
    RBTNODE lchild,rchild,parent;//节点的左右儿子以及父节点
}
```

红黑树类型定义

查找算法：按照二分查找的方式递归寻找要查找的结点。

更新算法：按照递归搜索的方式寻找插入点。不过要考虑边界条件--当树为空时，我们直接用树根记录这个结点，并设置为黑色，否则做递归查找插入。默认插入的节点颜色都是红色，因为插入黑色节点会破坏根路径上的黑色节点总数，但即使如此，也会出现连续红色节点的情况。因此在一般的插入操作之后，出现红黑树约束条件不满足的情况（称为失去平衡）时，就必须要根据当前的红黑树的情况做相应的调整。和 AVL 树的平衡调整通过旋转操作的实现类似，红黑树的调整操作一般都是通过旋转结合节点的变色操作来完成的。

第 3 章 分配器的设计与实现

总分 50 分

3.1 总体设计（10 分）

此分配器采用课本 memlin.c 包所提供的一个内存系统模型。模型的目的在于允许我们在不干涉已存在的系统层 malloc 包的情况下，运行分配器。

Mem_init 函数将对于堆来说可用的虚拟内存模型化为一个大的、双字对齐的字节数组。在 mem_heap 和 mem_brk 之间的字节表示已分配的虚拟内存。Mem_init 函数初始化分配器，如果成功就返回 0，否则就返回-1.mem_malloc 和 mm_free 函数与它们对应的系统函数有相同的接口和语义。

分配器采用的**块**格式：最小空闲块的大小为 6 字。空闲链表组织成为一个显示空闲链表。对堆初始化时获取一个 6 字的块，其中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，这是一个 8 字节的已分配块，只由一个头和一个脚部组成。序言块是在初始化时创建的，并且永不释放。在空闲块中存在两个指针，分别作为前驱和后继指针，以此形成双向空闲链表。**堆**总是以一个特殊的结尾块来结束，这个块是一个大小为 0 的已分配块，只有一个头部组成。

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：

创建一个初始空堆

处理流程：

Mm_init 函数从内存系统得到 6 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。

要点分析：

mm_init 函数将链表头初始化都指向 NULL。

Mm_init 函数中会调用 extend_heap 函数，extend_heap 函数会在两种不同的环境中被调用：当堆被初始化和 mm_malloc 不能找到一个合适的匹配块时。为了保持对齐，extend_heap 将请求大小向上舍入为最接近的 2 字的倍数，然后向内存系统请求额外的堆空间

3.2.2 void mm_free(void *ptr)函数（5 分）

函数功能：

释放一个块，并使用边界标记合并将之与所有的邻接空闲块在常数时间内合并

参　　数：

一个名为 ptr 的指针

处理流程：

应用通过调用 mm_free 函数，来释放一个以前分配的块（意味着如果当前指针为未分配块指针则不作处理），这个函数释放所请求的块，然后使用边界标记合并技术将之与邻接的空闲块合并起来。

要点分析：

要设置 head 和 foot 之后再执行 coalesce 函数

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

函数功能：

改变块的内存大小

参　　数：

ptr 指针和块的大小 size

处理流程：

1.内存对齐，2.如果 size 小于原来块大小直接返回原来的块，3.否则先检查地址连续的下一个块是否为空闲的或者块是否是堆的结束块，3.1.如果即使加上后面连续的空闲块空间还不够则要扩展块，3.2.删除刚刚利用空闲块并设置新的头尾，4.没有可以利用的连续空闲块，而且 size 大于原来的块，这时只能申请新的不连续的空闲块，复制原块内容、释放原块。

要点分析：

其中 size 为 0，相当于 free，其中 ptr 为 NULL，相当于 malloc，memmove 实现旧数据移动，包括缩小空间和扩大空间两种情况。

3.2.4 int mm_check(void)函数（5 分）

函数功能：

检查重要的不变量和一致性条件

处理流程：

依此检查堆中各个块，包括空闲链表，分配的块，当且仅当堆是一致的，才能返回非 0 值

要点分析：

空闲链表中的块是否为空闲的，连续的空闲块是否合并了，空闲块是否在空闲链表中，指针指向的空闲块是否有效，分配的块是否重叠，堆中的指针指向是否有效

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

函数功能：

向内存请求大小为 size 字节的块

参　　数：

向内存申请块的大小 size

处理流程：

至少申请 2DSIZE 的堆空间然后调用 find_fit 和 place 函数或者先利用 extend_heap 函数扩展堆之后调用 place 函数。

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。最小块的大小是 16 字节：8 字节用来满足对齐要求，而另外 8 个用来放头部和脚部。对于超过 8 个字节的请求，一般的规则是加上开销字节，然后向上舍入到最接近的 8 的整数倍

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

函数功能：

合并空闲块（主要在于 free 掉的块与前后空闲块的合并）

处理流程：

分前后无空闲、后空闲、前空闲、前后均空闲四种情况讨论，决定合并方式和处理需要合并块

要点分析：

合并的情况要先删除空闲块再插入，要分前后无空闲、后空闲、前空闲、前后均空闲四种情况讨论。我们选择的空闲链表格式（它的序言块和结尾块总是标记为已分配）允许我们忽略潜在的麻烦边界情况，也就是，请求块 bp 在堆的起始处或者是在堆的结尾处。如果没有这些特殊块，代码将混乱很多，并且容易出错，

并且更慢，所以我们将不得不在每次释放请求时，都去要检查这些并不常见的边界情况。

第 4 章测试

总分 10 分

4.1 测试方法

依次测试 traces 文件夹里的文件，应用的指令为
./mdriver -v -f ./traces/file_name 单测
./mdriver -v -t traces/ 总测

4.2 测试结果评价

测试结果显示，对于 mm_malloc 和 mm_free 的测试方面无论是空间利用率还是吞吐率来说都比较理想，而在测试 mm_realloc 的过程中看到反馈回来的结果不算太理想，多次试验无果，没有追踪到症结所在，但好在综测还算理想。

```
fs1170301027@ubuntu:~/hitics/malloclab-handout$ ./mdriver -v -t traces/
Team Name:1170301027
Member 1 :Feng Shuai:1170301027@stu.hit.edu.cn
Using default tracefiles in traces/
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   89%    5694  0.000259 21951
 1       yes   92%    5848  0.000146 40137
 2       yes   94%    6648  0.000416 15988
 3       yes   96%    5380  0.000293 18337
 4       yes   99%   14400  0.000135106984
 5       yes   88%    4800  0.000508  9454
 6       yes   85%    4800  0.000517  9293
 7       yes   55%   12000  0.004693  2557
 8       yes   51%   24000  0.011615  2066
 9       yes   26%   14401  0.168714    85
10       yes   34%   14401  0.004757  3027
Total          74%  112372  0.192052   585

Perf index = 44 (util) + 39 (thru) = 83/100
```

4.3 自测试结果

```
fs1170301027@ubuntu:~/hitics/malloclab-handout$ ./mdriver -v -f traces/amptjp-bal.rep
Team Name:1170301027
Member 1 :Feng Shuai:1170301027@stu.hit.edu.cn
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   89%    5694  0.001149  4957
Total          89%    5694  0.001149  4957

Perf index = 53 (util) + 40 (thru) = 93/100
```

```
fs1170301027@ubuntu:~/hitics/malloclab-handout$ ./mdriver -v -f traces/cccp-bal.rep
Team Name:1170301027
Member 1 :Feng Shuai:1170301027@stu.hit.edu.cn
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   92%    5848  0.000156 37415
Total          92%    5848  0.000156 37415

Perf index = 55 (util) + 40 (thru) = 95/100
```

```
fs1170301027@ubuntu:~/hitics/malloclab-handout$ ./mdriver -v -f traces/cp-decl-bal.rep
Team Name:1170301027
Member 1 :Feng Shuai:1170301027@stu.hit.edu.cn
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   94%    6648  0.000302 21999
Total          94%    6648  0.000302 21999

Perf index = 57 (util) + 40 (thru) = 97/100
```

```
fs1170301027@ubuntu:~/hitics/malloclab-handout$ ./mdriver -v -f traces/expr-bal.rep
Team Name:1170301027
Member 1 :Feng Shuai:1170301027@stu.hit.edu.cn
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   96%    5380  0.000341 15782
Total          96%    5380  0.000341 15782

Perf index = 58 (util) + 40 (thru) = 98/100
```

```
fs1170301027@ubuntu:~/hitics/malloclab-handout$ ./mdriver -v -f traces/coalescing-bal.rep
Team Name:1170301027
Member 1 :Feng Shuai:1170301027@stu.hit.edu.cn
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%   14400  0.000153 94241
Total          99%   14400  0.000153 94241

Perf index = 60 (util) + 40 (thru) = 100/100
```

```
fs1170301027@ubuntu:~/hitics/malloclab-handout$ ./mdriver -v -f traces/random-bal.rep
Team Name:1170301027
Member 1 :Feng Shuai:1170301027@stu.hit.edu.cn
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   88%    4800  0.000771  6230
Total          88%    4800  0.000771  6230

Perf index = 53 (util) + 40 (thru) = 93/100
```

```
fs1170301027@ubuntu:~/hitics/malloclab-handout$ ./mdriver -v -f traces/random2-bal.rep
Team Name:1170301027
Member 1 :Feng Shuai:1170301027@stu.hit.edu.cn
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   85%    4800  0.000783  6128
Total          85%    4800  0.000783  6128

Perf index = 51 (util) + 40 (thru) = 91/100
```

```
fs1170301027@ubuntu:~/hitics/malloclab-handout$ ./mdriver -v -f traces/binary-bal.rep
Team Name:1170301027
Member 1 :Feng Shuai:1170301027@stu.hit.edu.cn
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   55%   12000  0.004662  2574
Total          55%   12000  0.004662  2574

Perf index = 33 (util) + 40 (thru) = 73/100
```

```
fs1170301027@ubuntu:~/hitics/malloclab-handout$ ./mdriver -v -f traces/binary2-bal.rep
Team Name:1170301027
Member 1 :Feng Shuai:1170301027@stu.hit.edu.cn
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   51%   24000  0.010499  2286
Total          51%   24000  0.010499  2286

Perf index = 31 (util) + 40 (thru) = 71/100
```

```
fs1170301027@ubuntu:~/hitics/malloclab-handout$ ./mdriver -v -f traces/realloc-bal.rep
Team Name:1170301027
Member 1 :Feng Shuai:1170301027@stu.hit.edu.cn
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   26%   14401  0.131937   109
Total          26%   14401  0.131937   109

Perf index = 16 (util) + 7 (thru) = 23/100
```

```
fs1170301027@ubuntu:~/hitics/malloclab-handout$ ./mdriver -v -f traces/realloc2-bal.rep
Team Name:1170301027
Member 1 :Feng Shuai:1170301027@stu.hit.edu.cn
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   34%   14401  0.002879  5002
Total          34%   14401  0.002879  5002

Perf index = 20 (util) + 40 (thru) = 60/100
```

第 5 章 总结

5.1 请总结本次实验的收获

通过本次实验理解现代计算机系统虚拟存储的基本知识，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，设计过程中也意识到自己的诸多不足，以及与大佬之间的差距。

5.2 请给出对本次实验内容的建议

没什么更好的建议

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.