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 1190301610

班　　级 1903603

学 生 姓 名 王家琪

指 导 教 师

实 验 地 点

实 验 日 期

计算机科学与技术学院

目 录

第 1 章 实验基本信息 - 4 -

1.1 实验目的 - 4 -
1.2 实验环境与工具 - 4 -
1.2.1 硬件环境 - 4 -
1.2.2 软件环境 - 4 -
1.2.3 开发工具 - 4 -
1.3 实验预习 - 4 -

第 2 章 实验预习 - 5 -

2.1 进程的概念、创建和回收方法（5 分） - 5 -
2.2 信号的机制、种类（5 分） - 7 -
2.3 信号的发送方法、阻塞方法、处理程序的设置方法（5 分） - 5 -
2.4 什么是 SHELL，功能和处理流程（5 分） - 6 -

第 3 章 TINYSHELL 测试 - 9 -

3.1 TINYSHELL 设计 - 9 -

第 4 章 总结 - 15 -

4.1 请总结本次实验的收获 - 15 -
4.2 请给出对本次实验内容的建议 - 15 -

参考文献 - 17 -

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

1.2.3 开发工具

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

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。

分配器分为两种：显式分配器、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，例如 malloc，free。

隐式分配器（垃圾收集器）：要求分配器检测一个已分配块何时不再使用，那么 就释放这个块，自动释放未使用的已经分配的块的过程叫做垃圾收集。

2.2 带边界标签的隐式空闲链表分配器原理（5 分）

隐式空闲链表结构如图所示：主要包含头部标志位，尾部标志位，有效载荷，填充位。其中标志位表示分配的块的大小，尾部的四 bit 是分配（0001）/空闲（0000）。

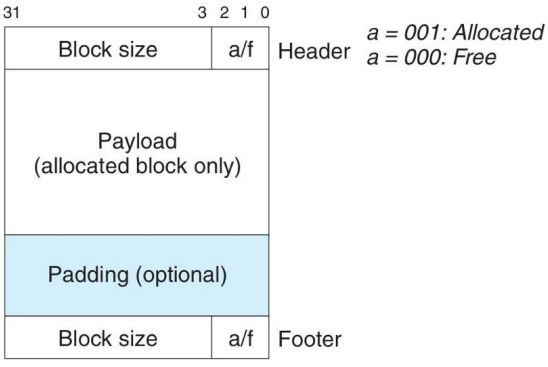

图 2-1 隐式空闲链表块结构

分配器要完成一系列操作，比如：分配（malloc），释放（free）。分配操作涉及到查找合适空闲块，有首次分配、再次分配、最佳分配等多种算法策略，然后对空闲块进行分割，放置分配的块。释放的操作比较简单，但是要考虑空闲块

合并的操作，当找不到足够大的空闲块可以 malloc 的时候，就要考虑合并空闲块了。空闲块合并一共有四种情况，针对每种情况，改变头部和尾部的标志位：

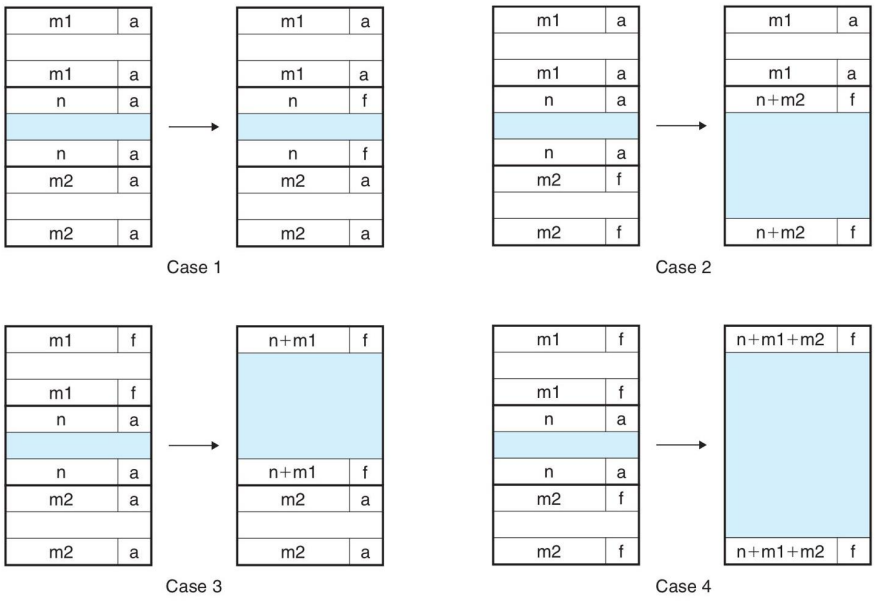

图 2-2 合并空闲块的四种情况

当合并空闲块也无法满足分配的要求的时候，这个时候就需要获取额外堆内存。分配器会调用 sbrk 函数，扩张指向堆顶的 brk 指针，向内核请求额外的内存。

2.3 显示空闲链表的基本原理（5 分）

显示空闲链表与隐式空闲链表不同，它的空闲块之间是用双指针进行链接的，每一个空闲块包含前驱和后驱的指针，形成了空闲块的双链表结构。

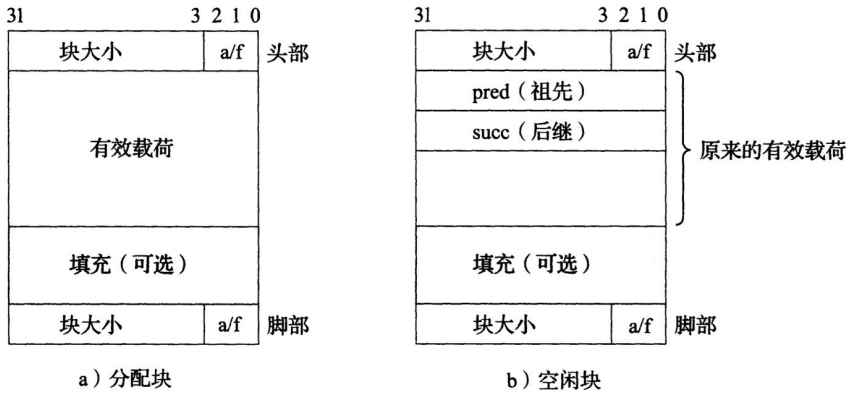

图 2-3 显示空闲链表的块结构

使用双向链表而不是隐式空闲链表，使得 malloc 分配的时候，首次分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。

当 free 块的时候，需要在链表中加入空闲块，维护链表的方式有两种：一种方法使用后进先出的顺序维护链表，将新释放的块在链表的开始处。使用 LIFO 的

顺序和首次适配的放置策略，分配器会最先检查最近使用过的块，在这种情况下，释放一个块可以在线性的时间内完成，如果使用了边界标记，那么合并也可以在常数时间内完成。另一种是按照地址顺序来维护链表，其中链表中的每个块的地址都小于它的后继的地址，在这种情况下，释放一个块需要线性时间的搜索来定位合适的前驱。平衡点在于，按照地址排序首次适配比 LIFO 排序的首次适配有着更高的内存利用率，接近最佳适配的利用率。

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，但是加入了染色等操作。从而保证了红黑树的查找、插入、删除的时间复杂度最坏为 O(log n)。

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。

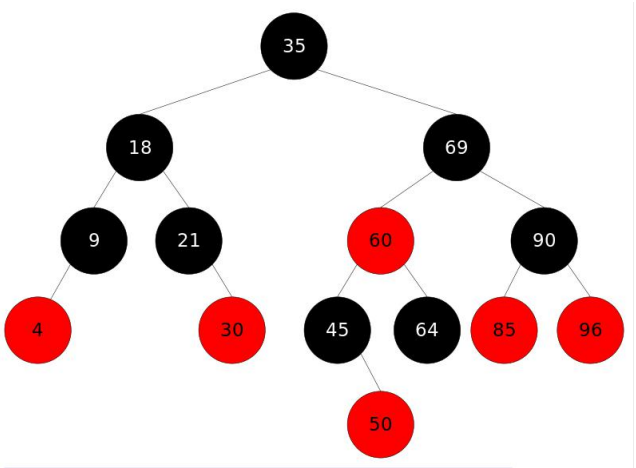

图 2-4 红黑树示例

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。调整可以分为两类：一类是颜色调整，即改变某个节点的颜色；另一类是结构调整，改变检索树的结构关系。结构调整过程包含两个基本操作：左旋（Rotate Left），右旋（RotateRight）。我从网上找到了两个左旋和右旋的示意图，网址如下图所示，示意图借鉴过来了。https://www.cnblogs.com/CarpenterLee/p/5503882.html

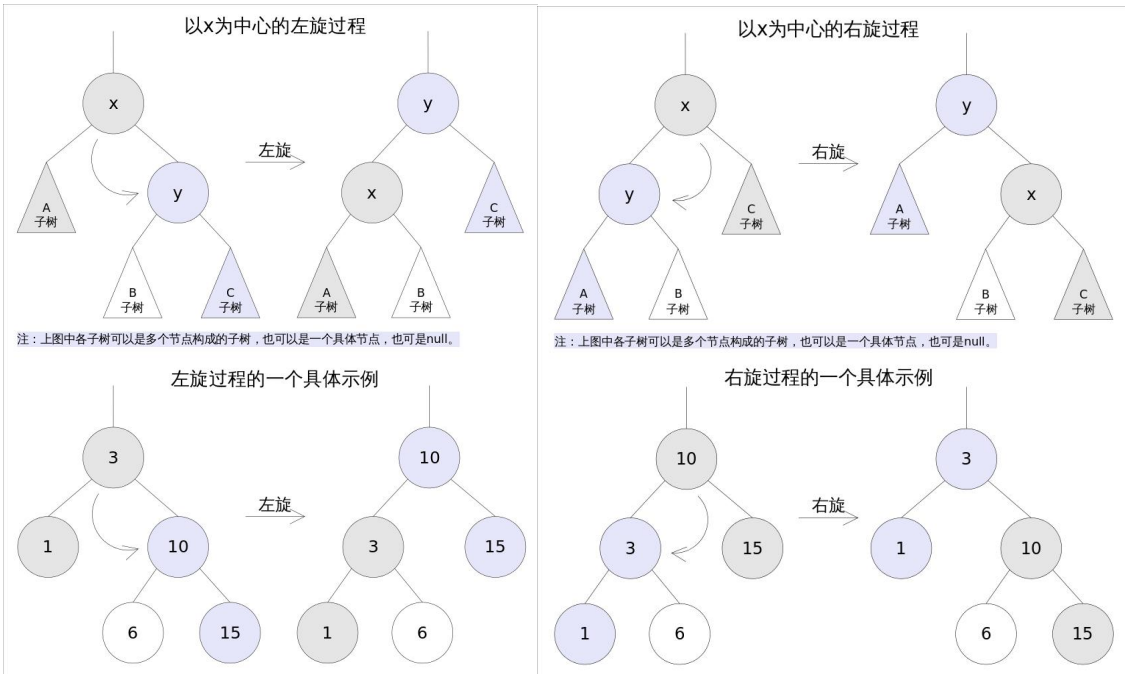

图 2-4 左旋 右旋示意图

红黑树的查找算法比较简单，直接借鉴二叉搜索树的查找算法即可，从根节点开始找，如果大于左，小于右，直至找到或者到 null 为止。插入和删除算法可以借鉴二叉搜索树的算法，但是要注意重新染色，和通过左旋右旋保持红黑树的性质不变。

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。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。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。

堆中内存块的组织结构: 用隐式空闲链表来组织堆，初始化隐式空闲链表的算法在 mm_init 函数中，标记了一个堆的序言块和尾块，拓展堆大小。

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（byte）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（bit），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（a/f）。

采用的空闲块、分配块链表/树结构：为了优化堆的性能，我们采用一个链表 Lists 来维护空闲块，将空闲块按照从小到大的顺序存储到 Lists 里面，每个 Lists[i] 里面存储的是块大小介于 2^(i-1)到 2^i 之间的，便于 malloc 的时候找到大小最合适的块，即使采用首次适配也可以达到最佳分配的效果。为了维护 Lists，定义 InsertNode 和 DeleteNode 两个函数，在每次 free 和 malloc 的时候更新 Lists。

相应算法：算法主要涉及到 Lists 链表的链表的查找、插入和删除，还有动态内存分配等内容。动态内存分配比较有难度的算法是 coalesce，分四种情况分别操作即可（原理见第二章预习）。动态内存分配需要实现的功能其实可以概括为 malloc——分配内存，free——释放内存。为了优化性能，减少内部碎片和外部碎片，我们采用了 Lists 来根据大小管理空闲块，并加入 InsertNode 和 DeleteNode 两个函数来维护 Lists 的不变性，最后利用 checkheap，checkblock 来进行堆的一致性检查。

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：初始化堆区域。返回值为 0 表示正常，-1 表示有错误；
处理流程：

1. 初始化分离空闲链表，每个指针为 NULL
2. 设置堆的头部尾部，分别有填充字，序言块（一个头部和脚部），结尾块
3. 拓展堆大小为 INITIALSIZE（48B）
4. 堆的一致性检查

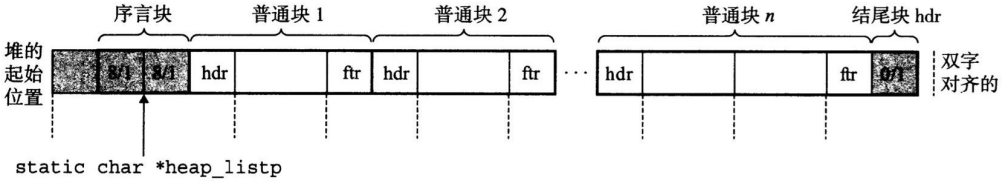

图 3-1 隐式空闲链表的恒定形式

要点分析：注意堆格式。

3.2.2 void mm_free(void *ptr)函数（5 分）

函数功能：释放参数“ptr”指向的已分配内存块，没有返回值。指针值 ptr 应该是之前调用 mm_malloc 或 mm_realloc 返回的值，并且没有释放过。
参　　数：ptr
处理流程：

1. 计算释放的块大小
2. 改写头部尾部的标记后三位为 0
3. 将释放的块插入到显式分离空闲链表
4. 立即进行空闲块合并
5. 堆的一致性检查

要点分析：在释放分配块之后则该 block 已经空闲下来了，需要将该 block 加入到显式分离空闲链表之中，添加之后因为位于堆之中，地址前后的块可能存在空闲块，所以我们需要调用 colesce 进行空闲块的合并，colesce 之中包括如果发生合 并产生的必要的链表操作逻辑。

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

函数功能：
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。
注意：
(1)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。
(2)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。
- 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。

参　　数：块指针 ptr，块大小 size

处理流程：

1. 对齐块。如果块大小小于两个字节，手动对齐，否则调用 ALIGN 函数
2. 判断要求的 size 是否小于原来块的大小，如果小于，直接返回原指针。
3. 如果要拓展块的大小，考虑以下几种情况：

   1）如果后面的块为结束块，按缺少的大小拓展堆，更新显式内存分配链表

   2）如果后一个为空闲块，考虑合并。如果合并后大小够 size，直接更新即可。

   3）其他情况，则 malloc 动态内存分配，将内容复制到新块里面，返回新指针，释放当前块。
4. 堆的一致性检查

要点分析：为了优化性能，最好不要直接 malloc 一个新的内存，考虑可不可以就地解决。当 size<块大小的时候，直接返回，如果块后面有空闲块或者直接就是堆结束块，也不用 malloc，直接在当前块的基础上拼接即可。其他情况可以直接 malloc 新内存。

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。当且仅当堆是一致的，才能返回非 0 值。

处理流程：利用 checkheap 和 checkblock 函数来检查一致性。

要点分析：为了简化代码，我们特意设置了 checkblock 函数，来检查每一块是否合法，主要检查块的大小是否是 8 的倍数，块的头部和脚部是否一直这两点。我们还设置了 checkheap 函数来检查堆的一致性，主要检查堆的序言块和结尾块是否合法，堆里面分配的每一块是否合法（checkblock）等等。

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

函数功能：申请有效载荷至少是参数“size”指定大小的内存块，返回该内存块地址首地址（可以使用的区域首地址）。申请的整个块应该在对齐的区间内，并且不能与其他已经分配的块重叠。返回的地址应该是 8 字节对齐的（地址%8==0）。

参　　数：块的大小 size

处理流程：

1. 非法判断，如果 size==0 返回空指针
2. 对齐，如果 size<8 个字节，手动对齐，否则调用 ALIGN 函数
3. 在显式分配空闲链表里面寻找大小合适的空闲块。
4. 如果显式分配空闲链表没有合适的空闲块，拓展堆，更新链表

要点分析：如何在显式分配空闲链表里面寻找大小合适的空闲块？在 Lists 里面，空闲块的大小从小到大排列，根据 2 的幂次分组，利用 size<<1 找到对应的 Lists[i], 在这个链表里面找比 size 大的的空闲块即可。

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。

参数：bp 是要回收的空闲块指针

处理流程：分成四种情况分别考虑（合并原理见第二章实验预习）

1. 存储 bp 前面和后面的块大小
2. 如果前后都是已分配，直接返回即可
3. 如果只有后面是空闲块，则合并，bp 指针不变，重写头部和脚部的标志，更新 Lists。
4. 如果只有前面是空闲块，则合并，bp 指针指向前面的块，重写头部和脚部的标志，更新 Lists。
5. 如果前面和后面的都是空闲块，则三块合并，bp 指针指向前面的块，重写头部和脚部的标志，更新 Lists。

要点分析：更新 List 主要涉及到两个操作，插入和删除。为了简化代码，我们特意写了两个函数 DeleteNode，InsertNode。这涉及到我们维护 Lists 的策略，除了在 Lists 里面，空闲块的大小从小到大排列，根据 2 的幂次分组，我们还希望每个 List[i] 里面的空闲块也是安装从小到大排列，这样分配的时候，可以减少内部碎片提高堆的性能。所以我们删除和插入的时候要选择合适的位置，而不是简单的头插法。

InsertNode：首先遍历 Lists，寻找到块所在的 Lists[i]，然后往里面插入新块。插入的时候要维护 Lists[i]里面的块大小从小到大排列，有四种情况：第一次往 Lists[i]里面插，中间插，头部插，尾部插，分别讨论即可。DeleteNode：DeleteNode 和 InsertNode 的思路差不多，删除的时候分三种情况即可，没有初始化的情况。

第 4 章测试

总分 10 分

4.1 测试方法

实验目标：能正确、高效、快速地运行
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
对于本测试，只要执行： ./mdriver -t traces/ -v 即可。

4.2 自测试结果

```
kolerk@ubuntu:~/Documents/malloclab-handout-hit$  ./mdriver -t traces/ -v
Team Name:ateam
Member 1 :Harry Bovik:bovik@cs.cmu.edu
Using default tracefiles in traces/
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   97%    5694  0.000441 12900
 1       yes   99%    5848  0.000455 12861
 2       yes   99%    6648  0.000399 16683
 3       yes   99%    5380  0.000299 17981
 4       yes   99%   14400  0.000757 19030
 5       yes   94%    4800  0.000517  9293
 6       yes   91%    4800  0.000512  9375
 7       yes   95%   12000  0.000634 18930
 8       yes   88%   24000  0.002448  9805
 9       yes   99%   14401  0.000285 50530
10       yes   98%   14401  0.000214 67200
Total          96%  112372  0.006960 16146

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

4.3 测试结果评价

可以看到大部分的 util 都在 90%以上，只有第 8 个测试还是结果不太好。总分为 98 分。

通过分数的分析可以看到我们的动态内存还有很多不太完美的地方，比如虽然用了一个链表 Lists 来根据大小管理空闲块，首次适配达到最佳适配的目的，但是查找还是耗费了很多时间，另外对空闲块的合并是立即合并的机制，也造成了时间上的浪费，现实中常常是先放着不管，直到找不到足够大小的块来 malloc 的时候，才进行合并操作。

改进的措施还有利用显式空闲链表来分配，彻底改变堆的块内部结构和空闲块的管理模式等等。

第 5 章 总结

5.1 请总结本次实验的收获

深入了解了动态内存分配的机制，实践了隐式空闲链表的堆管理，并利用了一个额外的链表来管理空闲块，达到了优化堆性能的目的。

5.2 请给出对本次实验内容的建议

希望老师可以讲的详细一下。

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.