**武汉轻工大学**

**操作系统课程设计报告**

**设计题目： 仿真模拟实现伙伴系统**

**小组成员：常凤迪 修梓凯 黄学凯 程代杨 徐沛柱**

**学 号：2105110007 2105110281 2105110305**

**2105110004 2105110170**

**院 (系)： 数学与计算机学院**

**班 级： 计科2103**

**指导教师： 余琛**

2023年 6月 20 日

目录

[一、课程设计目的 2](#_Toc138164979)

[二、课程设计要求和分析 2](#_Toc138164980)

[（一）实验要求 2](#_Toc138164981)

[（二）伙伴系统原理分析 2](#_Toc138164982)

[三、设计步骤及实现 3](#_Toc138164983)

[（一）算法实现分析 3](#_Toc138164984)

[（二）实验环境 3](#_Toc138164985)

[（三）算法实现 3](#_Toc138164986)

[（四）整体流程图 9](#_Toc138164987)

[四、结论 12](#_Toc138164988)

[（一）对伙伴系统的认识 12](#_Toc138164989)

[（二）伙伴系统的优缺点 12](#_Toc138164990)

[（三）总结 13](#_Toc138164991)

[小组分工 13](#_Toc138164992)

[参考文献 13](#_Toc138164993)

## 

## 一、课程设计目的

操作系统是计算机系统配置的基本软件之一。它在整个计算机系统软件中占有中心地位。其作用是对计算机系统进行统一的调度和管理，提供各种强有力的系统服务，为用户创造既灵活又方便的使用环境。本课程是计算机及应用专业的一门专业主干课和必修课。

通过课程设计，使学生掌握操作系统的基本概念、设计原理及实施技术,具有分析操作系统和设计、实现、开发实际操作系统的能力。

本实验的目的是通过模拟伙伴系统的原理和实现，了解连续内存分配的方法和优缺点，提高操作系统课程设计的能力和水平。

## 二、课程设计要求和分析

### （一）实验要求

通过所学知识模拟仿真伙伴系统过程，不限制编程语言

### （二）伙伴系统原理分析

#### 1.大致思路

伙伴系统是一种连续内存分配的方法，它可以有效地减少内存碎片和合并空闲块的开销。它的基本思想是将可用的内存空间划分为大小为2的幂次方的块，每次分配或释放内存时，都尝试将相邻且大小相同的块合并或拆分，以满足内存请求或回收内存空间。伙伴系统的优点是可以保证大对象的连续内存分配，可以快速地找到合适的空闲块，并且合并和拆分的操作也很简单。伙伴系统的缺点是会产生内部碎片，即分配出去的内存空间不能被完全利用，对合并的条件太过严格，只能是满足伙伴关系的块才能合并，一个很小的块往往会阻碍一个大块的合并，会增加内核维护空闲块和已分配块数组的开销。

#### 2.详细分析

假设系统的可利用空间容量为个字，则系统开始运行时，整个内存区是一个大小为的空闲分区。在系统运行过程中，由于不断的划分，可能会形成若干个不连续的空闲分区，将这些空闲分区根据分区的大小进行分类，对于每一类具有相同大小的所有空闲分区，单独设立一个空闲分区链表。这样，不同大小的空闲分区形成了k ( Okm）个空闲分区链表。

当需要为进程分配一个长度为n的存储空间时，首先计算一个i值，使-1<n<，然后在空闲分区大小为的空闲分区链表中查找。若找到，即把该空闲分区分配给进程。否则，表明长度为的空闲分区已经耗尽，则在分区大小为中的空闲分区链表中寻找。若存在中的一个空闲分区，则把该空闲分区分为相等的连个分区，这两个分区称为一对伙伴，其中的一个分区用于分配，而把另一个加入分区大小为的空闲分区链表中。若大小为的空闲分区不存在，则需要查找大小为的空闲分区，若找到则对其进行两次分割:第一次，将其分割为大小为的两个分区，一个用于分配，一个加入到大小为空闲分区链表中;第二次，将第一次用于分配的空闲分区分割为的两个分区，一个用于分配，一个加入到大小为空闲分区链表中。若仍然找不到，则继续查找大小为的空闲分区，以此类推。由此可见，在最坏的情况下，可能需要对的空闲分区进行k次分割才能得到所需分区。

与一次分配可能要进行多次分割一样，一次回收也可能要进行多次合并，如回收大小为的空闲分区时，若事先已存在的空闲分区时，则应将其与伙伴分区合并为大小为的空闲分区，若事先已存在的空闲分区时，又应继续与其伙伴分区合并为大小为的空闲分区，依此类推。

## 三、设计步骤及实现

### （一）算法实现分析

在模拟一系列进程对内存的请求与释放时，我们需要考虑以下几个方面：  
1、内存块的大小：伙伴系统将内存块分成大小相等的块，因此我们需要确定每个块的大小。可以根据实际情况选择合适的块大小。  
2、进程的内存请求：我们需要模拟一系列进程对内存的请求。每个进程请求的内存大小可能不同，因此我们需要在树中寻找一个大小合适的块，并将其分配给进程。  
3、进程的内存释放：当进程释放内存时，我们需要将该块与其伙伴块合并，并将其插入到树中适当的位置。  
4、内存的分配与释放顺序：我们需要考虑进程请求内存和释放内存的顺序，以及内存块的分配和释放顺序。这些顺序可能会影响内存的利用率和碎片情况。

### （二）实验环境

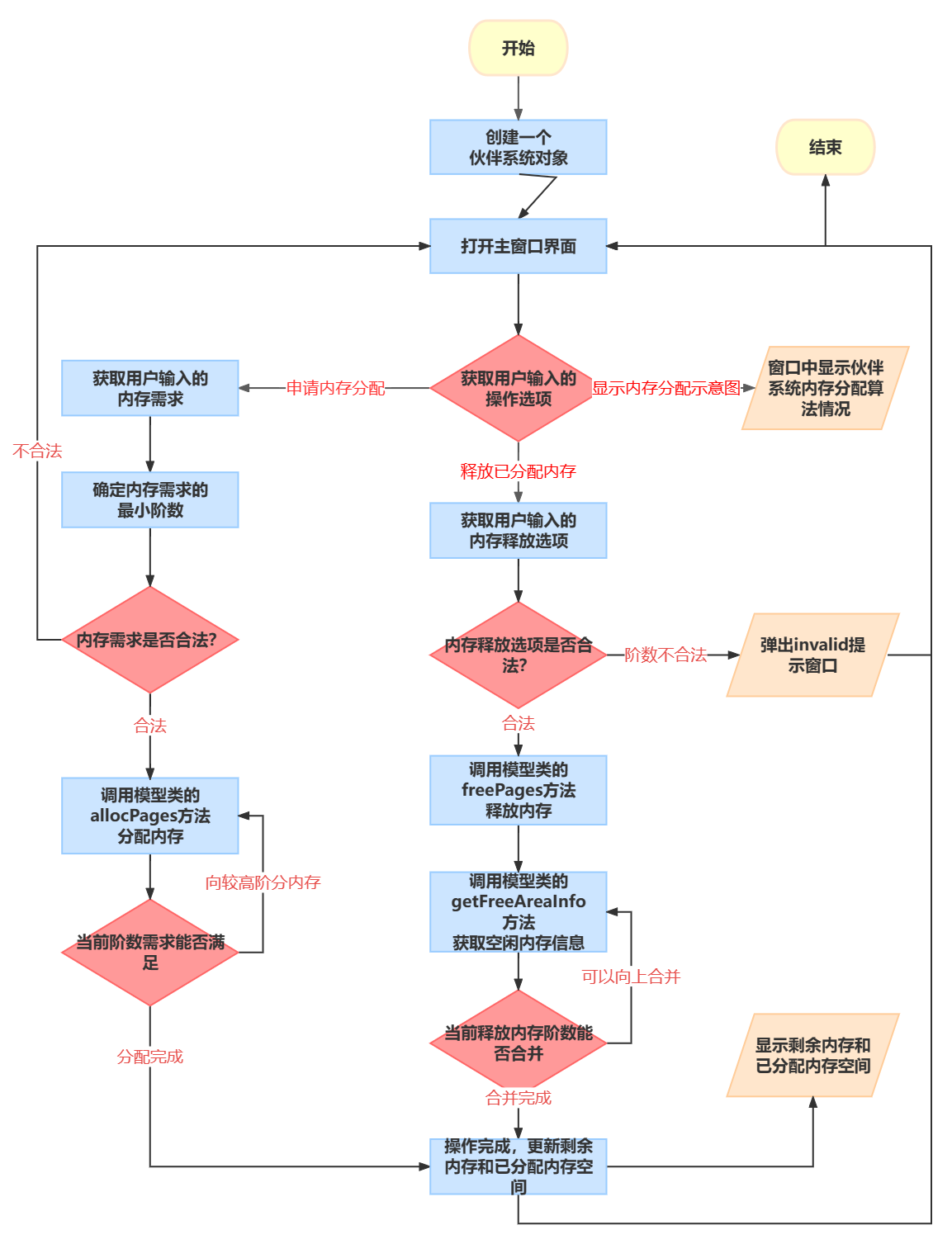
本实验使用Java编写程序，运行在Windows 11操作系统上，使用IDEA 专业版作为开发工具。

### （三）算法实现

#### 1.算法思路

定义一个Page类表示内存块，包含大小、起始地址、状态（空闲或已分配）、链表指针等字段。定义一个LinkList链表表示内存池，包含地址起点、内存块阶数、空闲块数组、已分配块位图等字段。  
初始化内存池：根据给定的总大小，初始化一个内存池，将其划分为一个最大的空闲块，并将其插入到空闲块数组中对应的链表中，将已分配块位图清零。  
分配内存：根据给定的请求大小，从空闲块数组中找到最小的满足条件的链表，如果链表为空，则向上查找更大的链表，如果找到了，则从中摘下一个空闲块，并对其进行二等分，将一半放回原来的链表，另一半继续比较和拆分，直到得到合适的可用空闲块。如果没有找到，则返回NULL表示分配失败。如果找到了，则将其从链表中摘下，并将其状态改为已分配，并将其在已分配块位图中对应的位设为1，并返回其起始地址表示分配成功。  
释放内存：根据给定的起始地址和大小，找到对应的内存块，并将其状态改为空闲，并将其在已分配块位图中对应的位设为0，并检查其是否有空闲的伙伴块，如果有，则从空闲块数组中摘下伙伴块，并将两个伙伴块合并成一个更大的空闲块，并将其插入到更高一级的链表中。然后继续检查合并后的空闲块是否有伙伴可以合并，直到不能合并或者达到最大的空闲块为止。

#### 2.算法流程图



#### 3.算法代码

public class BuddySystem {

public static final int MAX\_ORDER = 11; // 最大阶数

public static final int PAGE\_SIZE = 1 << 9; // 页面大小

public static final int MEM\_SIZE = 1 << 20; // 模拟内存大小

// 页面类

public static LinkedList<Page>[] freeArea; // 空闲页链表数组

public static Page[] memPool; // 模拟内存池

// 构造方法，初始化内存池和空闲页链表

public BuddySystem() {

int n = MEM\_SIZE / PAGE\_SIZE; // 总页数

memPool = new Page[n]; // 分配模拟内存池

for (int i = 0; i < n; i++) { // 初始化每个页面

memPool[i] = new Page(i);

}

freeArea = new LinkedList[MAX\_ORDER]; // 分配空闲页链表数组

for (int i = 0; i < MAX\_ORDER; i++) { // 初始化每个链表

freeArea[i] = new LinkedList<>();

}

freeArea[MAX\_ORDER - 1].add(memPool[0]); // 将整个内存池作为一个最大阶的空闲块

}

// 判断两个页面是否是伙伴

public boolean isBuddy(Page p1, Page p2, int order) {

if (p1 == null || p2 == null) return false; // 空指针不是伙伴

if (p1.allocated || p2.allocated) return false; // 已分配不是伙伴

int indexDiff = p1.index - p2.index; // 索引差值

if (indexDiff == (1 << order) || indexDiff == -(1 << order)) return true; // 索引差值等于阶数对应的页数则是伙伴

return false;

}

// 在空闲页链表中插入一个页面

public void insertPage(Page p, int order) {

p.order = order; // 设置阶数

p.allocated = false; // 设置未分配

freeArea[order].addFirst(p); // 插入到链表头部

}

// 在空闲页链表中删除一个页面

public void deletePage(Page p, int order) {

freeArea[order].remove(p); // 删除页面

}

// 分配一个指定阶数的内存块，返回页面对象，失败返回null

public Page allocPages(int order) {

if (order < 0 || order >= MAX\_ORDER) return null; // 阶数越界

if (!freeArea[order].isEmpty()) { // 如果有空闲块

Page p = freeArea[order].removeFirst(); // 取出链表头部的页面

p.allocated = true; // 设置已分配

p.order = order;

return p; // 返回页面对象

} else { // 如果没有空闲块

Page p = allocPages(order + 1); // 尝试分配更高阶的内存块

if (p == null) return null; // 分配失败返回null

Page buddy = memPool[p.index + (1 << order)]; // 找到伙伴页面

insertPage(buddy, order); // 将伙伴页面插入到空闲链表中

p.allocated = true; // 设置已分配

p.order = order;

return p; // 返回页面对象

}

}

// 释放一个指定阶数的内存块，传入页面对象

public void freePages(Page p, int order) {

// MainWindow.bs.printFreeArea();

if (p == null || order < 0 || order >= MAX\_ORDER) return; // 参数无效

Page buddy = null;

p.allocated = false;

for (Page curr : freeArea[order]) {

if (isBuddy(p, curr, order)) { // 如果找到伙伴页面

buddy = curr;

break;

}

}

if (buddy == null) { // 如果没有找到伙伴页面，直接插入到空闲链表中

insertPage(p, order);

} else { // 如果找到了伙伴页面，合并成更大的块，并递归释放

deletePage(buddy, order); // 删除伙伴页面

Page merged = p.index < buddy.index ? p : buddy; // 合并后的页面为索引较小者

// insertPage(merged, order+1);

freePages(merged, order + 1); // 递归释放更高阶的块

}

// MainWindow.bs.printFreeArea();

}

// 打印空闲页链表信息，用于调试

public void printFreeArea() {

for (int i = 0; i < MAX\_ORDER; i++) {

System.out.print("order " + i + ": ");

for (Page p : freeArea[i]) {

System.out.print(p.index + " ");

}

System.out.println();

}

}

public String getFreeAreaText() {

String res = new String(" free block address: <br>");

for (int i = 0; i < MAX\_ORDER; i++) {

res += "order " + Integer.toString(i) + ": ";

for (Page p : freeArea[i]) {

res += p.index + " ";

}

// System.out.println();

res += "<br>";

}

return res + "</html>";

}

public LinkedList<Page>[] getFreeArea() {

return freeArea;

}

public ArrayList<Page> turn() {

ArrayList<Page> res = new ArrayList<>();

for (int i = 0; i < MAX\_ORDER; i++) {

for (Page p : freeArea[i]) {

res.add(p);

}

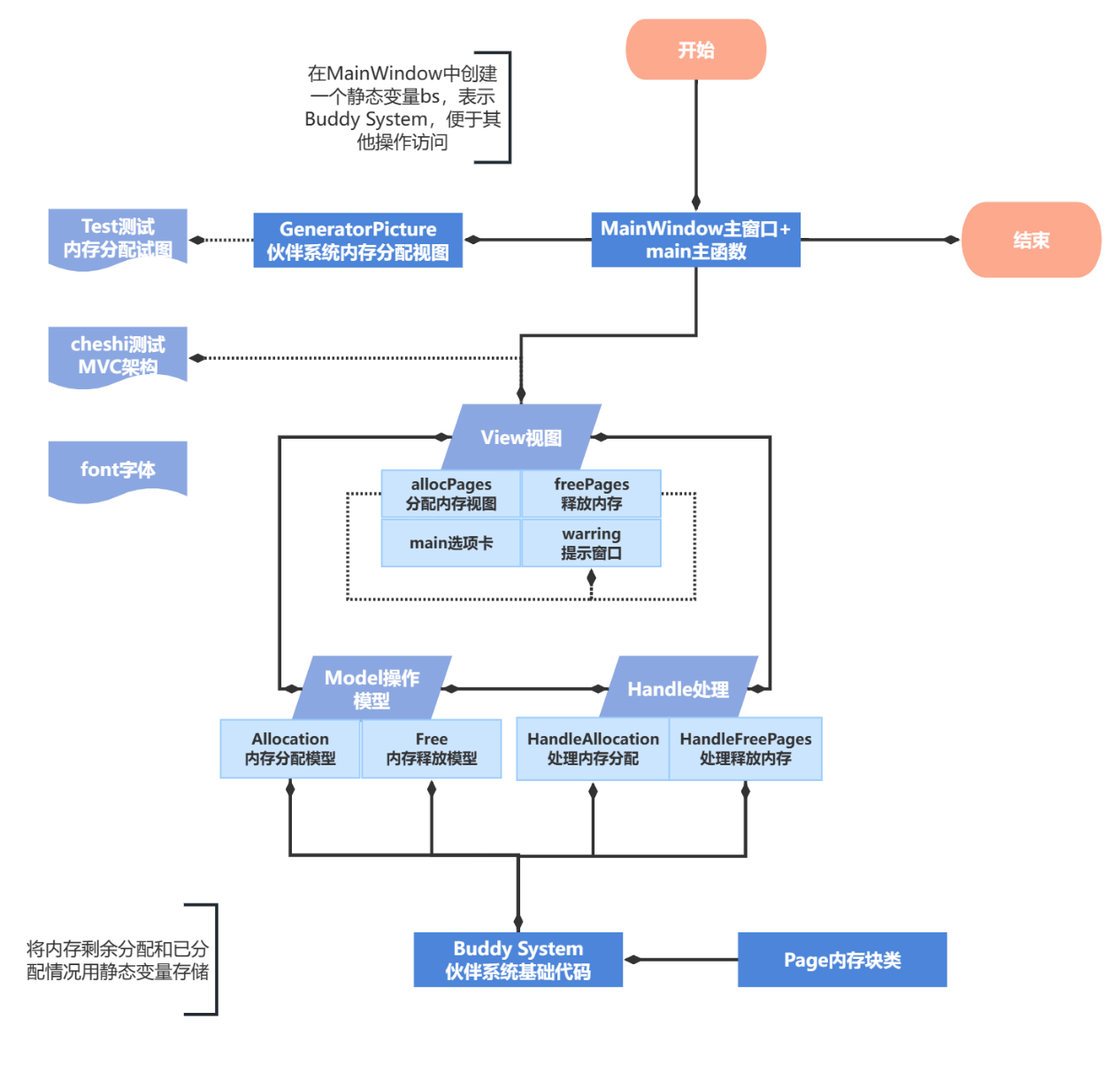
}

return res;

}

}

### （四）整体流程图

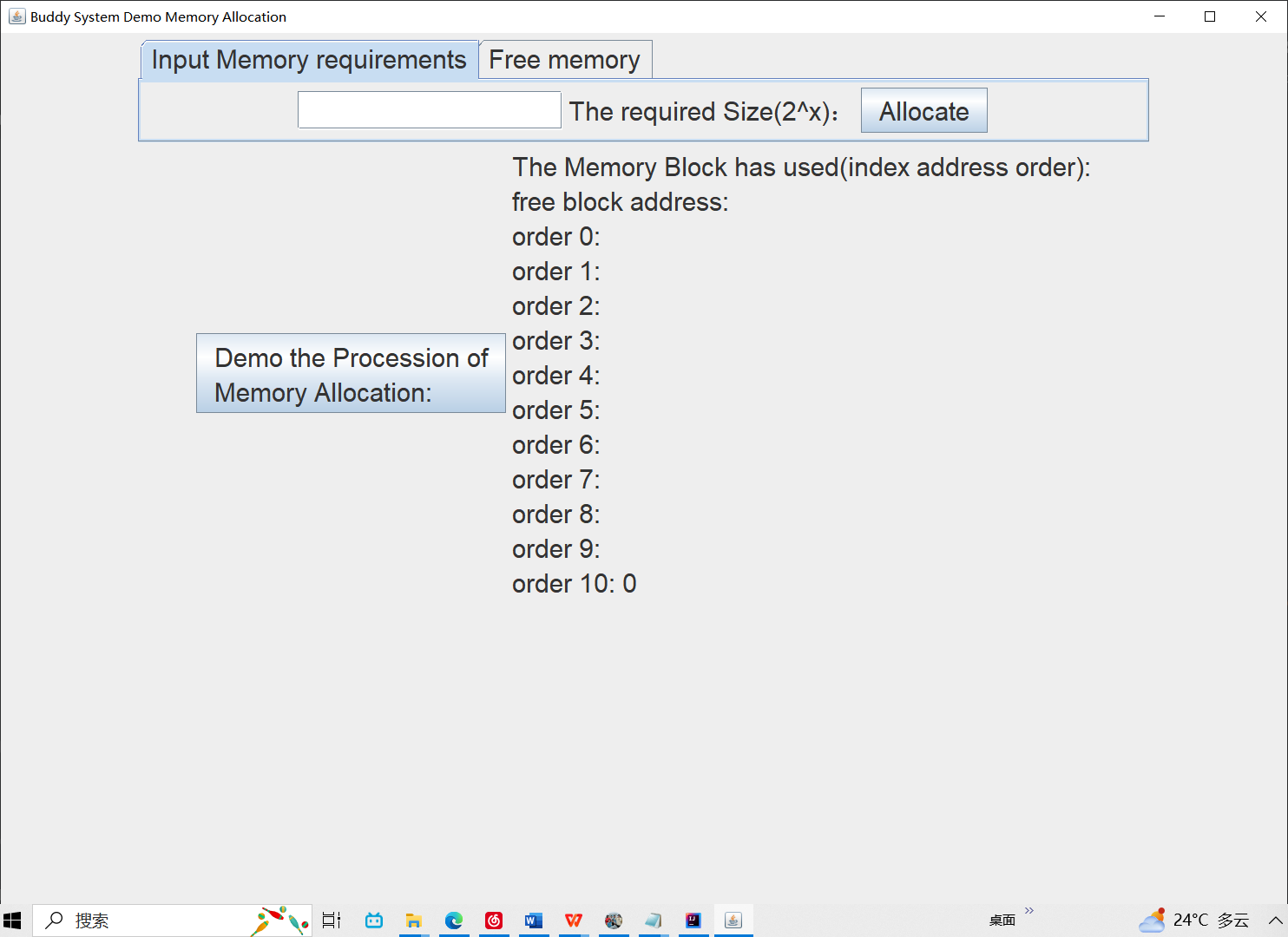


如上图

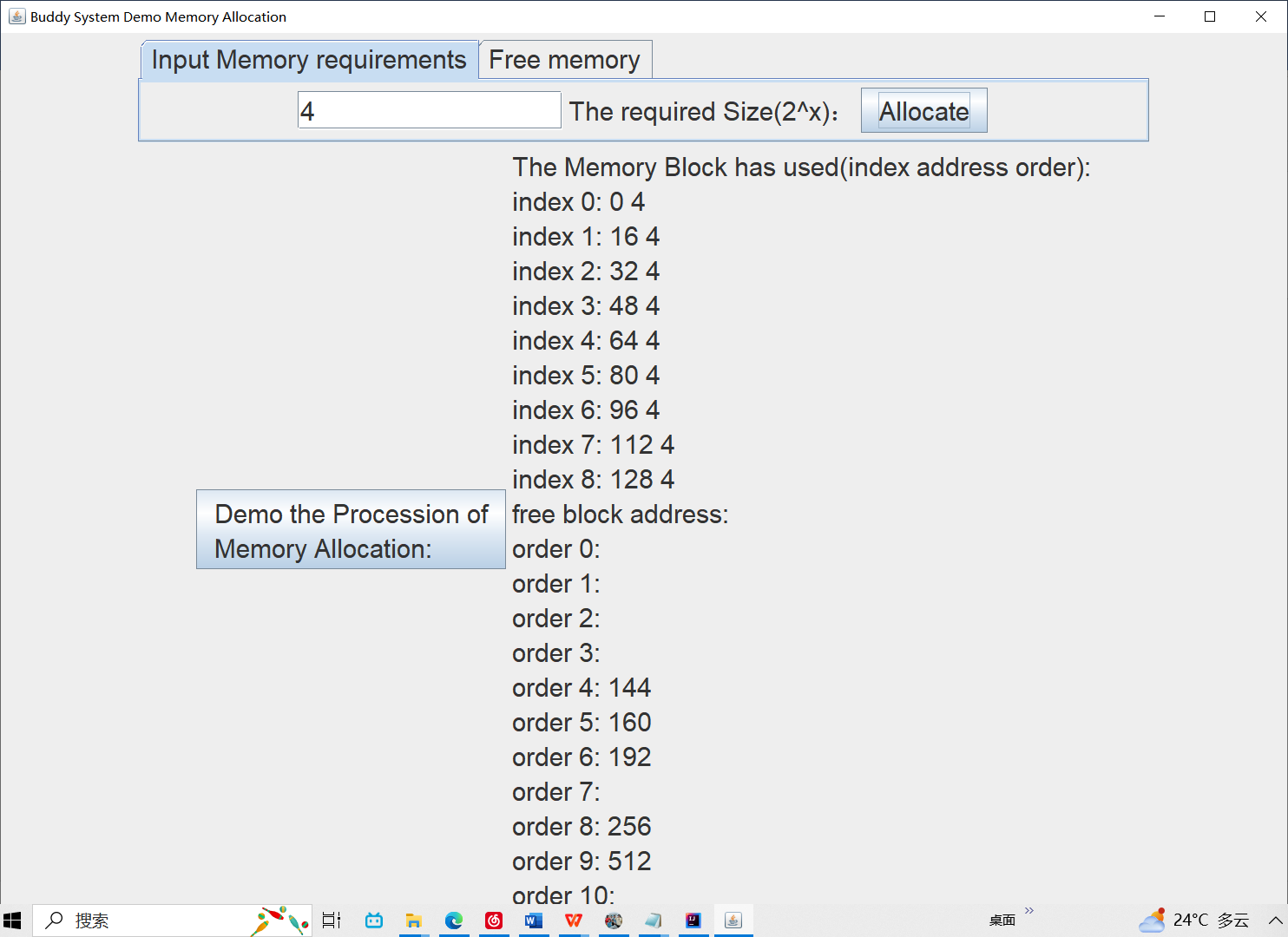
使用MVC架构实现输入，输出的抽象可视化

三、结论

（五）实验结果及分析



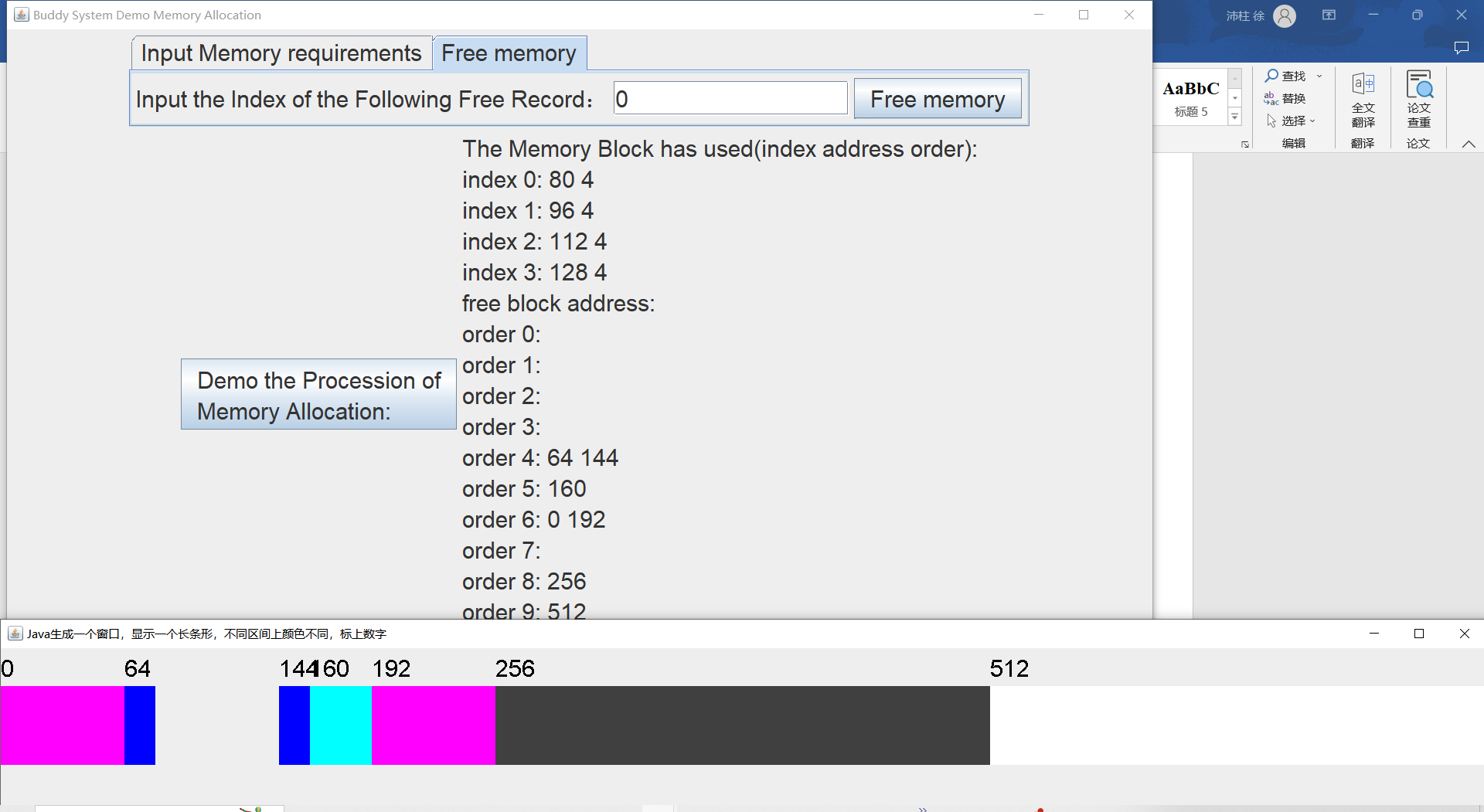
主界面如上图布局。



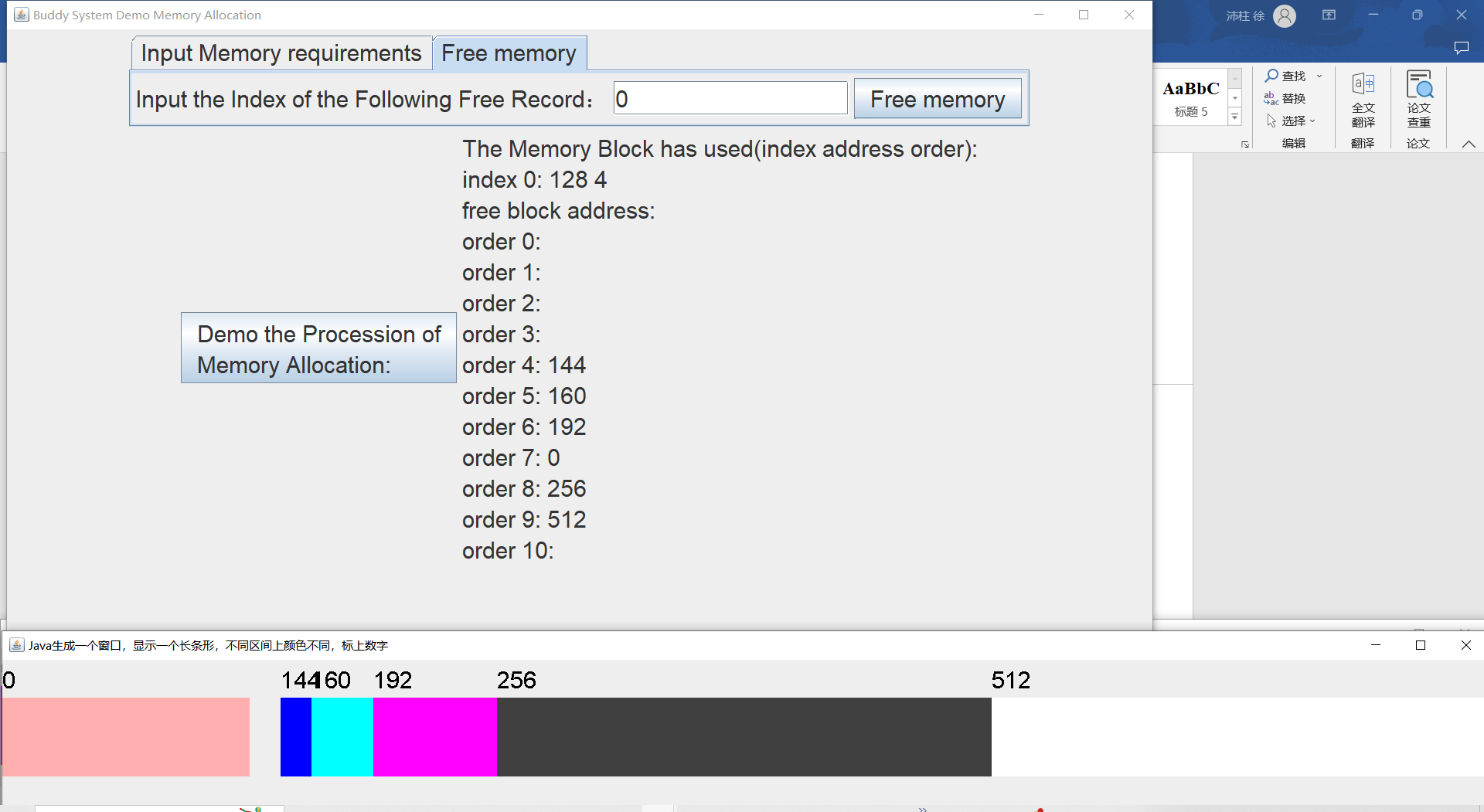
如上图，分配9个需求内存空间阶数大小为4的块，显示每个块的内存分配地址和大小（阶数），显示剩余未分配空间的情况。



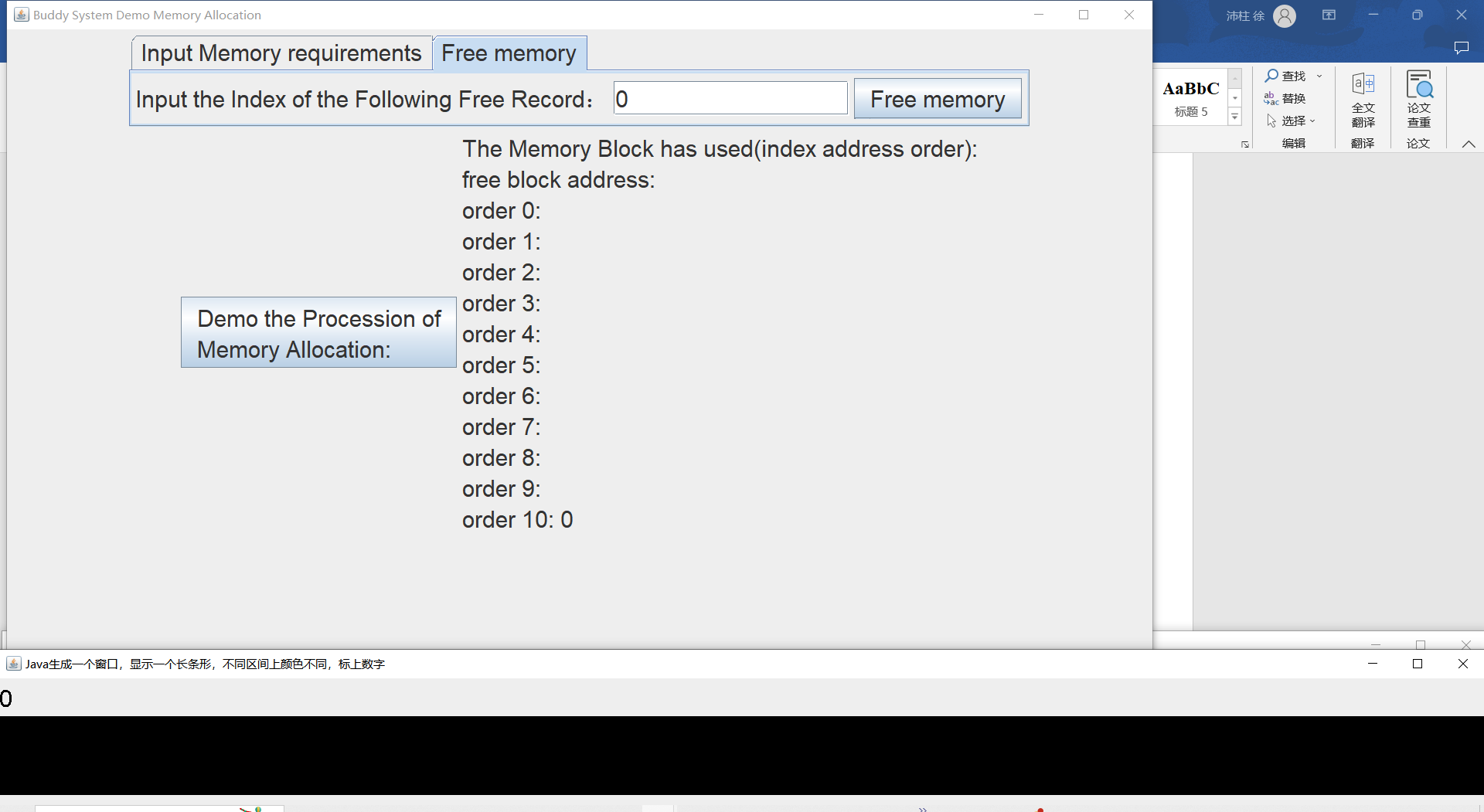
这是伙伴系统内存分配示意图



释放前五个内存需求后，剩余内存块需求和已用内存发生了变化，伙伴系统内存使用情况发生变化。



再释放三个内存需求后，剩余内存块需求和已用内存发生了变化，伙伴系统内存使用情况发生变化。



完全释放后，恢复成最开始的状态，内存块全被合并成一个最大的块。伙伴算法采用2的幂次方进行分配内存块，可以避免把大的内存块拆分的过小，更重要的是可以加快分配和释放速度，但如果所需要的空间不是2的整数次幂，则会产生许多内部碎片。分配和合并采用链表和位图操作，操作方便，但是开销比较大。一个很小的块往往会阻碍一个大块的合并，一个系统中，对内存块的分配，大小是随机的，一片内存中仅一个小的内存块没有释放，旁边两个大的内存块就不能合并。

## 四、结论

### （一）对伙伴系统的认识

伙伴系统是一种连续内存分配的方法，它可以有效地减少内存碎片和合并空闲块的开销。它的基本思想是将可用的内存空间划分为大小为2的幂次方的块，每次分配或释放内存时，都尝试将相邻且大小相同的块合并或拆分，以满足内存请求或回收内存空间。

### （二）伙伴系统的优缺点

伙伴系统的优点是：  
1.它可以保证大对象（不超过1024个页面）能够得到连续的内存分配，从而避免了外部碎片的产生 。  
2.它可以快速地找到合适的空闲块，并且合并和拆分的操作也很简单，只需要对位图进行异或操作即可 。  
3.它可以动态地调整内存分配的粒度，根据内存需求的变化而变化，从而提高了内存利用率。  
伙伴系统的缺点是：  
1.它会产生内部碎片，即分配出去的内存空间不能被完全利用，因为每次分配都要向上对齐到2的幂次方，可能会浪费一些空间 。  
2.它对合并的条件太过严格，只能是满足伙伴关系的块才能合并，一个很小的块往往会阻碍一个大块的合并，从而影响了合并效率 。  
3.它会增加内核维护空闲块和已分配块数组的开销，需要申请额外的内存空间来存储位图和链表指针。

（三）总结  
通过本实验，本小组深入了解了伙伴系统的原理和实现，以及它是如何解决连续内存分配的问题的。小组各个成员掌握了伙伴系统的数据结构和算法，以及它是如何维护空闲块和已分配块的数组的。组员们分析了伙伴系统的优缺点，以及它是如何影响内存碎片和分配效率的。我认为伙伴系统是一种简单而有效的连续内存分配方法，但也有一些局限性和改进空间。

## 小组分工

材料收集 常凤迪 修梓凯 徐沛柱 黄学凯 程代杨

代码算法实现 常凤迪

PPT制作 黄学凯 徐沛柱 程代杨 修梓凯

PPT展示 常凤迪

论文编写 修梓凯 徐沛柱

## 参考文献

1.Andrew S·Tanenbaum《现代操作系统》（第四版）. 机械工业出版社，2017年7月

2.西尔伯查茨 高尔文加根《操作系统概念》（第十版）. 高等教育出版社，2007年3月

3.陈莉君《深入理解 Linux 内核》（第三版）. 中国电力出版社，2007年10月