**河北科技大学**

课程设计报告

学生姓名：­­­ 韩浩然 学 号： 180706207

专业班级： 网络工程182

课程名称： 数据结构

学年学期： 2 0 2 0 —2 0 2 1 学年第 一 学期

指导教师： 王建亮

2 0 2 0年1 2月

**课程设计成绩评定表**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 学生姓名 | 韩浩然 | 学 号 | 180706207 | | 成绩 |  |
| 专业班级 | 网络182 | 起止时间 | 2020.12.21 – 2020.12.27 | | | |
| 设计题目 | 模拟内存管理 | | | | | |
| 工作态度  （10%） | ○工作积极主动，遇到问题可相互讨论 10  ○工作态度一般，缺乏积极性和主动性 5  ○工作态度不端正，出勤率低1  ○无出勤记录0 | | | | | |
| 设计方案  （10%） | ○设计方案合理、完善，数据结构选择恰当10  ○设计方案较合理、完整，数据选择恰当 7  ○设计方案考虑不周到，数据结构选择不太合理4 | | | | | |
| 完成情况  （10%） | ○完成了要求的全部功能 10  ○完成了要求的主要功能 8  ○完成了基本功能 5  ○完成的功能太少 （0 - 3） | | | | | |
| 代码质量  （5%） | □标识符命名规范，可读性好3 □有适当注释1  □函数规模适中1 | | | | | |
| 操作演示  （10%） | ○能够熟练演示所有功能，知道程序存在的具体缺陷10  ○可以演示主要功能，知道程序存在的主要缺陷7  ○可以演示部分非主要功能，不清楚程序存在的缺陷3  ○不能够进行任何演示操作0 | | | | | |
| 答辩表现  （20%） | ○能正确回答和分析所有问题20 ○能正确回答主要问题15  ○可以回答一些难度较低的问题10 ○可以回答简单问题5  ○不能解答问题0 | | | | | |
| 功能变更  （10%） | ○能够在规定时间内实现变更10  ○能够在规定时间内完成部分变更8  ○不能在规定时间内完成变更，但能给出思路、方法5  ○不能完成变更，也不能给出解决方法0 | | | | | |
| 报告质量  （10%） | ○条理清楚，叙述流畅10 ○条理比较清楚，文字通顺7  ○组织较混乱，叙述不太清楚5 ○组织混乱，表达不清2 | | | | | |
| 书写质量  （10%） | ○工整10 ○较整齐8 ○一般5 ○较差2 | | | | | |
| 总结  （5%） | ○深刻5 ○真实、可信4 ○一般3 ○肤浅2 ○没有0 | | | | | |
| 总评 | ○优秀 ○良好 ○中等  ○及格 ○不及格 | | | 指导教师：  年 月 日 | | |

**目录 （手写，页码待定）**

**1　人员组成及分工…………………………………………………………….1**

**2　分析和设计…………………………………………………………………**

**3　实现…………………………………………………………………………**

**4　关键代码……………………………………………………………………**

**5　测试…………………………………………………………………………**

**6　总结…………………………………………………………………………**

1. **人员组成及分工**

2人组（韩浩然、杨攀）

分工：

韩浩然：实现内存分配的三种算法（首次适应算法，最佳适应算法，最坏适应算法）

杨攀：实现内存初始化、内存回收和显示内存使用情况

1. **分析和设计**

内存是计算机系统的重要资源，内存管理是操作系统的主要功能之一，目的是对有限的内存进行有效的管理，以满足大多数程序的需求。

模拟实现内存管理机制。设计和实现关于内存管理的内存布局初始化以及按不同策略进行内存分配、内存回收、查看内存使用情况等基本功能操作

分配内存策略使用首次适应算法，最佳适应算法，最坏适应算法三种算法

使用结构体数组来初始化整块内存，每个节点包含起始地址，当前内存大小，使用状态，回收内存时考虑右侧或左侧是否也为空闲，并择机进行合并操作

通过遍历整个结构体数组，找到已用的内存，进行加和，并记录当前内存使用状态，打印到控制台来显示内存使用情况

1. **实现**

**1.按策略进行内存分配（此部分由韩浩然完成）：**

1.首次适应算法

将空闲的内存区按其在储存空间中的起始地址递增的顺序排列，分配储存空间时，从空闲区链的始端开始查找，选择第一个满足要求的空闲区，而不管它究竟有多大

优点:

①在释放内存分区的时候，如果有相邻的空白区就进行合并，使其成为一个较大的空白区

②此算法的实质是尽可能的利用储存器的低地址部分，在高地址部分则保留多的或较大的空白区，以后如果需要较大的空白区，就容易满足

缺点：

在低地址部分很快集中了许多非常小的空白区，因而在空白区分配时，搜索次数增加，影响工作效率。

2.最佳适应算法

从全部空闲区中找出满足作业要求的，且大小最小的空闲分区的一种计算方法，这种方法能使得碎片尽量小，为适应此算法，空闲分区表中的空闲分区要按从小到大进行排序，自表头开始查找第一个满足要求的自由分区分配

优点：能使得碎片尽量的小,保留了最大空闲区

缺点：造成了许多小的空闲区

3.最坏适应算法**（新添加的部分）**

扫描整个空闲分区或者链表，总是挑选一个最大的空闲分区来使用

优点：可以使链表中的节点大小趋于均匀，产生碎片的几率最小，对中小作业有利，同时该算法的查找效率很高

缺点：会使得储存器中缺乏大的空闲分区

**2.内存初始化、内存回收、查看内存使用情况（此部分由杨攀完成）**

1.内存初始化：

将整块内存分成多份（比如100份），创建一个结构体数组，每一份作为一个节点，每个节点包含此片内存的大小、起始地址和使用状态，初始化每块大小为零，再将第一份设为最大，起始地址为零，内存使用状态默认为空闲

2.内存回收

从键盘输入得到要回收的某一块内存的序号，若状态为已用，则回收，并判断右侧是否为空闲，若是则与右侧合并，再将此块内存之后的内存序号往后移，再判断左侧是否为空闲，同理，若是则与左侧合并，同样当前内存之后的内存序号往后移，若否，则不作合并操作

3.查看内存使用情况

内置一个变量，遍历整个结构体数组，将状态为已用的内存大小加和，遍历时记录每块内存的状态，打印到控制台

1. **关键代码**
2. **首次适应算法**

遍历整个结构体数组，找到满足所需要的大小,且是空闲的内存，即

If(MemList[i].Size >= request && MemList[i].status == 'f')

还需要在判断一下此块内存剩余大小如果小于等于规定的最小差则将整个空间分配出去，将此快内存状态标记为已用，即MemList[i].status = 'u';

否则将此块分成两份，将当前元素后的元素后移，即

for (j = MEMSIZE - 2; j > i; j--)

{

MemList[j + 1] = MemList[j];

}

使用低地址部分，即重新计算这两份的大小和起始地址，前一份状态标记为已用

1. **最佳适应算法**

遍历找到符合大小要求的空间，保存此块空间的大小和位置，若下一个符合要求的大小小于刚保存的那一块的大小，则重新保存此块空间大小，并记录位置，即

if (MemList[i].Size < t)

{

t = MemList[i].Size;

j = i;

}

分配后的剩余大小操作部分同上

1. **最坏适应算法（新添加的部分）**

遍历找到符合大小要求的空间，保存此块空间的大小和位置，若下一个符合要求的大小大于刚保存的那一块的大小，则重新保存此块空间大小，并记录位置，即

if (MemList[i].Size > k)

{

k = MemList[i].Size;

j = i;

}

分配后的剩余大小操作部分同上

1. **内存初始化**

创建一个结构体数组，每一份作为一个节点，每个节点包含此片内存的大小、起始地址和使用状态，初始化大小为零，再将第一份设为最大，起始为零，内存状态默认为空闲

typedef struct memoryInfomation

{

int start;

int Size;

char status;

} MEMINFO;

MEMINFO MemList[MEMSIZE];

初始化

MEMINFO item = {0, 0, 'e'};

for (int i = 0; i < MEMSIZE; i++)

{

MemList[i] = item;

}

起始地址为0 ：MemList[0].start = 0;

空间初始为最大：MemList[0].Size = MEMSIZE;

状态为空闲：MemList[0].status = 'f';

1. **内存回收**

从输入得到要回收的某一块内存的序号：scanf("%d", &number);

若状态为已用，则回收，

if (MemList[number].status == 'u')

{

// 标记为空闲

MemList[number].status = 'f';

}

判断右侧是否为空闲，若是则与右侧合并，

if (MemList[number + 1].status == 'f')

{

// 大小合并

MemList[number].Size += MemList[number + 1].Size;

}

此块内存之后的内存序号往后移，

for (i = number + 1; i < MEMSIZE - 1 && MemList[i].status != 'e'; i++)

{

if (i > 0)

{

MemList[i] = MemList[i + 1];

}

}

同理，再判断左侧是否为空闲，若是则与左侧合并，之后的内存序号往后移，若否，不作合并操作

1. **显示内存使用情况**

内置一个变量 used ，遍历整个结构体数组，将状态为已用的内存大小加和，遍历时记录每块内存的状态，打印到控制台

if (MemList[i].status == 'u')

{

used += MemList[i].Size;

}

printf("| %4d%12d%14d%15s |\n", i + 1, MemList[i].start, MemList[i].Size, MemList[i].status == 'u' ? "已用" : "空闲");

1. **测试**

**见附录截图**

1. **总结**

内存是计算机系统的重要资源，内存管理是操作系统的主要功能之一，目的是对有限的内存进行有效的管理，以满足大多数程序的需求。

内存分配的三种算法：

首次适应算法：按内存地址大小从小到大开始适配

最佳实挺算法：按内存大小从小到大开始适配

最坏适应算法：扫描整块分区，总是选一块最大的来分配