河南工業大學

课　程　设　计

课程设计名称：操作系统原理

专 业 班 级 ：软件 1601

学 生 姓 名 ：高天

学　　号 ：201616030213

指 导 教 师 ：刘於勋

课程设计时间：2018/6/25 至 2018/6/29

目录

1 需求分析

用C语言实现采用循环首次适应算法的动态分区分配过程alloc()和回收过程 free()。其中，空闲分区通过空闲分区链表来管理，在进行内存分配时，系统优先使用空闲区低端的空间。采用循环首次适应算法进行内存块的分配和回收，同时显示内存块分配和回收后空闲内存分区链的情况。

假设初始状态如下，可用的内存空间为640KB，并按照下列的请求序列进行内存的分配与回收：

作业1申请130KB；作业2申请60KB；作业3申请100KB；作业2释放60KB；作业4申请200 KB；作业3释放100 KB；作业1释放130 KB；作业5申请140 KB；作业6申请60 KB；作业7申请50KB；作业6释放60 KB

基本功能：设计与实现动态分区分配的数据结构与算法。根据作业大小，对空闲分区按照循环首次适应算法进行分配，回收分区时，按照回收算法进行合并回收。分配、回收后显示空闲分区状态。

扩展功能：同时实现首次适应算法、最佳适应算法、最坏适应算法。通过绘制分区状态图更直观地显示分配和回收过程，对比各种算法的差异和优劣。

2 概要设计

总体结构：

在C语言面向过程编程方法中，常用分层方法进行编程，降低各模块之间的耦合度。在操作系统中，也用到了分层思想进行设计，每一步设计都建立在可靠的基础上，这样易于扩充和维护系统。

在此次的课程设计中，同样也使用分层的方法，可以将各种算法很简单方便的一起实现。最底层是基本的数据结构，用来存储分区的信息。上一层是分配分区的函数alloc()和回收分区的函数free()。通过调用alloc()和free()函数可以实现四种算法。通过图形展示算法可以直观地显示分配和回收过程。最上一层是程序的界面，允许用户可以自由的对内存进行分配和回收，程序展示动态结果。

总体结构如图1。

图 1 程序总体结构

数据结构：

在实际的操作系统中，分配出去的内存记录在进程控制块中，为了加快内存的分配过程，动态分区最常采用的数据结构是空闲表或空闲链表，数据结构只记录未分配的内存分区。为了降低模拟程序的复杂度，本模拟将分配出去的和未分配出去的内存分区记录在同一个数据结构中，用一个标志表示是否已被分配。各分区按地址递增次序排列。内存的分配和回收只要是对数据结构做插入和删除操作，采用双向链表结构。

为使程序具有扩展性灵活性，分区大小采用宏定义方式（#define MEMORY_SIZE 640），可修改分区的大小，展示在不同分区大小下的分配情况。

```
typedef struct SubAreaNode *SubAreaList;
struct SubAreaNode
{
    int addr;     //分区起始地址
    int size;     //分区大小
    int stat;     //分区状态  宏定义  ALLOCED 1    FREE 0
    int pid;      //已分配分区的进程 id

    SubAreaList pre;
    SubAreaList next;
};
SubAreaList head;     //全局变量  分区链表首指针
SubAreaList nowList;   //当前位置分区指针  用于循环首次适应算法
```

分区分配算法：

接口：int alloc(int pid, int psize, SubAreaList p);

在链表位置 p 处为编号为 pid，大小为 psize 的作业分配空间，链表 p 通过各种查找空闲分区算法（首次适应、循环首次适应、最佳适应、最坏适应）提供。

返回本次分配成功与否。

算法流程图见图 2。

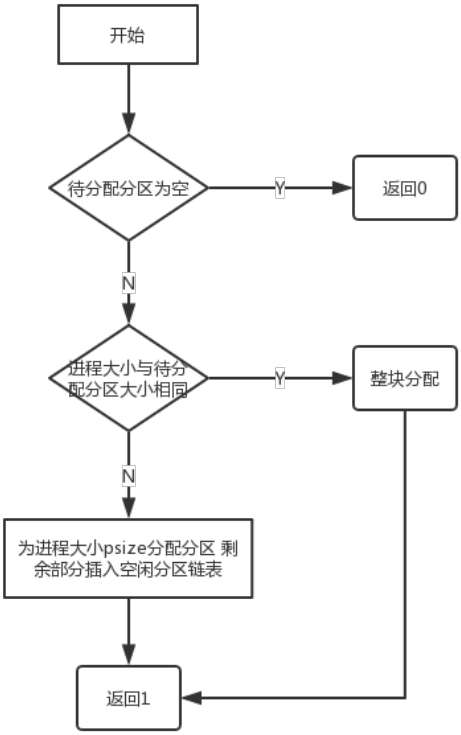

图 2 分区分配算法流程图

回收分区算法：

接口：int freeAlloc(int pid);

释放 pid 进程的分区空间，重新插入到空闲分区链表。返回值为本次回收是否成功。

回收时先遍历链表，找到进程 pid 所在的位置 p，如果存在，可能出现下面四种情况之一：

1） 回收区与前一空闲分区相邻，此时应将回收区与前一分区合并。

2） 回收区与后一空闲分区相邻，此时应将回收区与后一分区合并。

3） 回收区同时与前后两个空闲分区相邻，此时将三个分区合并。

4） 回收区不相邻前后空闲分区，直接改为空闲分区。

算法流程图见图 3。

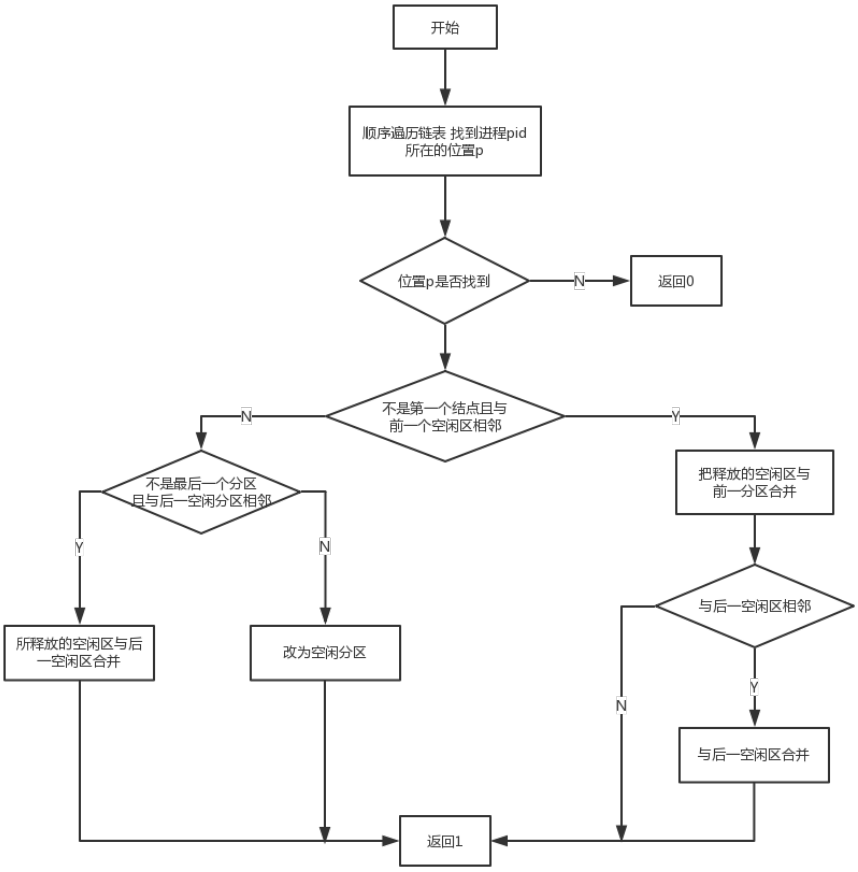

图 3 回收分区算法流程图

首次适应算法寻找分区位置：

接口：SubAreaList ffFindFreeSubArea(int psize);

按照首次适应算法从分区链表中找到第一个不小于 psize 的空闲分区，返回此分区地址。如果找不到可用的空闲分区返回 NULL。

算法流程图见图 4。

循环首次适应算法寻找分区位置：

接口：SubAreaList nfFindFreeSubArea(int psize);

按照循环首次适应算法从分区链表的当前位置开始寻找第一个不小于psize的空闲分区，返回此分区地址。寻找结束后更新当前位置。当链表循环一圈后仍没有找到合适的空闲分区则返回NULL。

算法流程图见图5。

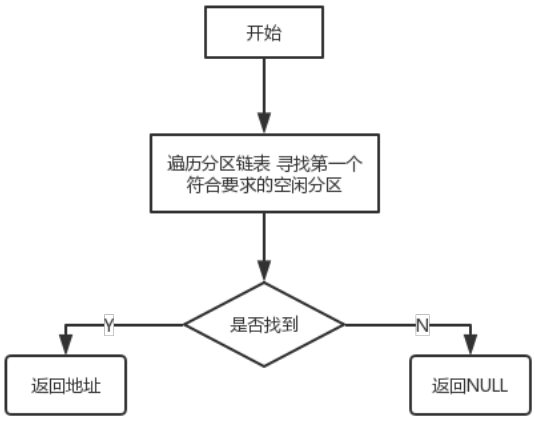

图4 首次适应算法寻找分区位置

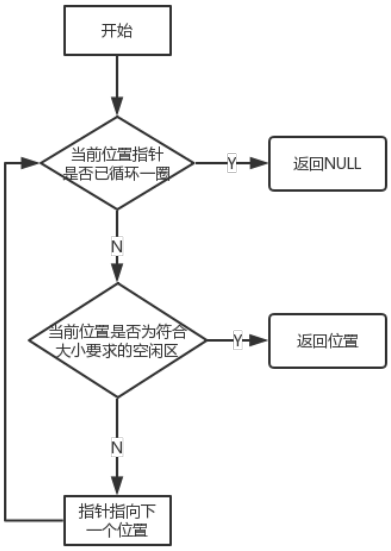

图5 循环首次适应算法寻找分区位置

最佳适应算法寻找分区位置：

接口：SubAreaList bfFindFreeSubArea(int psize);

可以将所有空闲分区按照大小升序排列，从头寻找第一个符合要求的空间分区进行分配。在这里为简单起见，对链表线性搜索一轮，找到符合空间大小要求的最小的空闲分区。

算法流程图如图6。

最坏适应算法寻找分区位置：

接口：SubAreaList wfFindFreeSubArea(int psize);

可以将所有空闲分区按照大小降序排列，从头寻找第一个符合要求的空间分区进行分配。在这里为简单起见，对链表线性搜索一轮，找到符合空间大小要求的最大的空闲分区。

算法流程图如图 7。

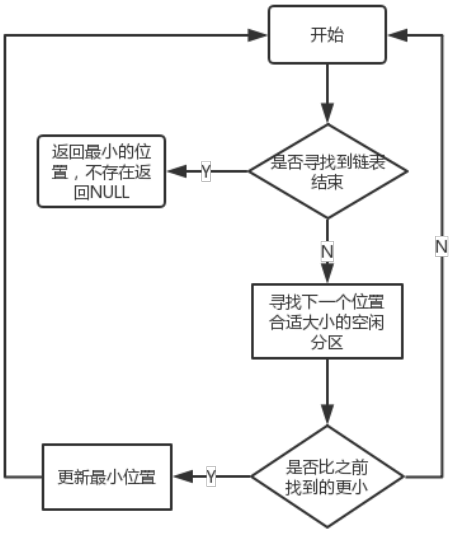

图 6 最佳适应算法寻找分区位置

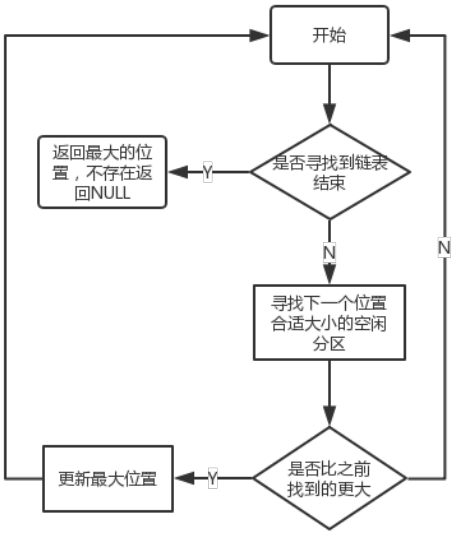

图 7 最坏适应算法寻找分区位置

绘制状态图算法：

接口：void disAllocGraph(int prec);

用于绘制精度为 prec 的内存状态图。也即每 MEMORY_SIZE/prec 绘制一格。

程序界面：

接口：void selectAlogrithm();

可选择进入不同算法进行内存分配和回收的演示，每个算法既支持手动输入分配回收序列，也支持从文件中输入分配回收序列。每进行一步即展示当前这一步的分配结果。展示时内存分配表和分配图并排展示，可以更直观清晰的看到内存分配回收情况，加深对算法的理解程度。

3 运行环境

硬件环境：

cpu： Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz × 4

内存：8G

软件环境：

OS：Deepin Linux 15.6 64 位

Linux 内核：Linux 4.15.0-21deepin-generic

4 开发工具和编程语言

编辑软件：

Sublime Text 3

编译软件：

gcc version 7.2.0

编程语言：

C 语言

5 详细设计

链表初始化：

```
SubAreaList getASubArea() {
    return (SubAreaList)malloc(sizeof(struct SubAreaNode));
}

//对空闲分区链表进行初始化，一整块空间为空
SubAreaList initSubArea() {
    SubAreaList p = getASubArea();
    p->addr = 0;
    p->size = MEMORY_SIZE;
    p->stat = FREE;
    p->pid = -1;
    p->pre = NULL;
    p->next = NULL;

    return p;
}
```

分配分区函数：

```
//在链表位置 p 处为编号为 pid，大小为 psize 的作业分配空间
int alloc(int pid, int psize, SubAreaList p) {
    if (p == NULL)     //无合适空间可分配
        return 0;
```

```
        if (p->size == psize) {     //分区整块分配
            p->stat = ALLOCED;
            p->pid = pid;
        } else {                            //分割分区
            SubAreaList newSubArea = getASubArea();
            newSubArea->addr = p->addr + psize;
            newSubArea->size = p->size - psize;
            newSubArea->stat = FREE;
            newSubArea->pid = -1;

            p->size = psize;
            p->stat = ALLOCED;
            p->pid = pid;

            newSubArea->next = p->next;
            p->next = newSubArea;
            newSubArea->pre = p;

        }

        return 1;
    }
```

回收分区函数：

```
    int freeAlloc(int pid) {
        //寻找作业所在分区
        SubAreaList p = head;

        while (p) {
            if (p->pid == pid) {
                break;
            }
            p = p->next;
        }

        if (p == NULL)
            return 0;

        //不是首块分区且与前一空闲分区相连
        if (p != head && p->pre->stat == FREE &&
                p->pre->addr + p->pre->size == p->addr) {
            SubAreaList preNode = p->pre;
            SubAreaList nextNode = p->next;
```

```
            preNode->size += p->size;
            preNode->next = p->next;
            nextNode->pre = preNode;

            //与后一空闲分区相连
            if (p->next->stat == FREE &&
                        p->addr + p->size == p->next->addr) {
                preNode->size += nextNode->size;
                preNode->next = nextNode->next;
                nextNode->next->pre = preNode;

                free(nextNode);
            }

            free(p);

        } else {
            //不是最后一个分区且与后一空闲分区相连
            if (p->next != NULL && p->next->stat == FREE &&
                        p->addr + p->size == p->next->addr) {
                SubAreaList nextNode = p->next;
                p->size += nextNode->size;
                p->next = nextNode->next;
                nextNode->next->pre = p;
                p->stat = FREE;
                p->pid = -1;

                free(nextNode);
            } else {
                p->stat = FREE;
                p->pid = -1;
            }
        }

        return 1;
    }
```

首次适应算法：

```
    SubAreaList ffFindFreeSubArea(int psize) {
        SubAreaList p = head;
        while (p) {
            if (p->stat == FREE && p->size >= psize)
                return p;
```

```
            p = p->next;
        }

        return NULL;
    }


    int ffAlloc(int pid, int psize) {
        return alloc(pid, psize, ffFindFreeSubArea(psize));
    }
```

循环首次适应算法：

```
    SubAreaList nfFindFreeSubArea(int psize) {
        SubAreaList tmp = nowList;

        while (nowList) {
            if (nowList->stat == FREE && nowList->size >= psize) {
                return nowList;
            }
            nowList = nowList->next == NULL ? head : nowList->next;
            if (nowList == tmp) return NULL;
        }
    }

    int nfAlloc(int pid, int psize) {
        int ret = alloc(pid, psize, nfFindFreeSubArea(psize));
        nowList = nowList->next == NULL ? head : nowList->next;
        return ret;
    }
```

最佳适应算法：

```
    SubAreaList bfFindFreeSubArea(int psize) {
        SubAreaList p = head, minP = NULL;
        int minSize = MEMORY_SIZE + 1;

        while (p) {
            if (p->stat == FREE && p->size >= psize) {
                if (p->size < minSize) {
                    minSize = p->size;
                    minP = p;
                }
            }
            p = p->next;
```

```
        }

        return minP;
    }

    int bfAlloc(int pid, int psize) {
        return alloc(pid, psize, bfFindFreeSubArea(psize));
    }
```

最坏适应算法：

```
    SubAreaList wfFindFreeSubArea(int psize) {
        SubAreaList p = head, maxP = NULL;
        int maxSize = -1;

        while (p) {
            if (p->stat == FREE && p->size >= psize) {
                if (p->size > maxSize) {
                    maxSize = p->size;
                    maxP = p;
                }
            }
            p = p->next;
        }

        return maxP;
    }


    int wfAlloc(int pid, int psize) {
        return alloc(pid, psize, wfFindFreeSubArea(psize));
    }
```

分配状态显示图与表：

```
    //绘制内存分配图
    void disAllocGraph(int prec) {
        SubAreaList p = head;

        for (int i = 0; i < MEMORY_SIZE/prec; i++)
            printf("-");
        printf("\n");

        int addr[MAX_PRO];
        int pid[MAX_PRO];
```

```
int minSize = MEMORY_SIZE;
int n = 0;
while (p) {
	addr[n] = p->addr;
	pid[n++] = p->pid;

	minSize = p->size < minSize ? p->size : minSize;
	p = p->next;
}
addr[n] = MEMORY_SIZE;
pid[n++] = -1;
if (minSize < 10) {
	printf("内存间隔过小，不显示分配图\n");
	return;
}

for (int l = 0; l < 3; l++) {
	for (int i = 0, j = 0, k = 0; i < MEMORY_SIZE/prec; i++) {
		if (i == addr[j]/prec) {
			printf("|");
			j++;
		} else
			printf(" ");

		if (l == 1 && i == addr[k]/prec +
				(addr[k+1]/prec-addr[k]/prec)/2) {
			if (pid[k] != -1) {
				printf("%d", pid[k]);
				i = pid[k] < 10 ? i+1 : i+2;
			}
			k++;
		}

		if (i == MEMORY_SIZE/prec - 2) printf("|");
	}
	printf("\n");
}


for (int i = 0; i < MEMORY_SIZE/prec; i++)
		printf("-");
printf("\n");
int cnt = 0;
for (int i = 0, k = 0; i < MEMORY_SIZE/prec; i++) {
```

```
            int digit = 0;
            int tmp = addr[k];
            while (tmp) { digit++; tmp /= 10; }
            if (i == addr[k]/prec - digit/2) {
                printf("%d", addr[k]);
                k++;
                cnt = digit;
            } else {
                if (cnt > 1) { cnt--; continue; }
                printf(" ");
            }
        }

        printf("\n");
    }


    void printSepLine() {
        printf("\n**************************************************
                                            ************************\n\n");
    }

    void displayAlloc() {
        SubAreaList p = head;

        printf("\n%3s %3s %3s %3s %3s\n", "起始", "终止", "长度", "状态
                                                                ", "ID");

        while (p) {
            printf("%3d    %3d    %3d    %3d    %3d\n", p->addr,
                p->addr + p->size-1, p->size, p->stat, p->pid);
            p = p->next;
        }

        printf("\n");
        disAllocGraph(10);

        printSepLine();
    }
```

输入分配回收序列控制函数：

```
    void inputCtrl(int (*allocAlogrithm)(int,int)) {
        system("clear");
```

```
    printf("分配输入：  A 作业号 大小\n");
    printf("回收输入：  F 作业号\n");
    printf("退出输入：  Q\n\n");

    char T[5];
    scanf("%s", T);
    while (T[0] != 'Q') {
        if (T[0] == 'A') {
            int pid, size;
            scanf("%d%d", &pid, &size);
            int ret = allocAlogrithm(pid, size);
            if (ret) {
                printf("作业号 %d   分配   %d KB\n", pid, size);
                displayAlloc();
            }
            else {
                printf("\n 内存不足 分配失败\n\n");
                printSepLine();
            }
        } else if (T[0] == 'F') {
            int pid;
            scanf("%d", &pid);
            int ret = freeAlloc(pid);
            if (ret) {
                printf("作业号 %d   已回收\n", pid);
                displayAlloc();
            } else {
                printf("未找到相关作业，回收失败\n\n");
                printSepLine();
            }
        } else
            exit(0);
        scanf("%s", T);
    }
}

void fileInputCtrl(int (*allocAlogrithm)(int,int)) {
    freopen(FILE_NAME, "r", stdin);
    inputCtrl(allocAlogrithm);
    freopen("/dev/tty", "r", stdin);
}
```

6 调试分析

分区分配与回收测试输入数据：

A 1 130

A 2 60

A 3 100

F 2

A 4 200

F 3

F 1

A 5 140

A 6 60

A 7 50

F 6

Q

四种算法通过以上测试数据可以很正确的得出结果，分配状态图也可以清晰直观的显示分区状态。

不足的是，如果请求分配的进程大小过于小或进程过多，会导致分区、碎片都比较小，这时状态图显示算法会错误显示。这里的解决方案是，如果分区间隔小于 10 就不做显示，只显示分区表。

7 测试结果

下面给出上述测试序列选用循环首次适应算法时的运行结果：

```
分配输入：  A 作业号 大小
回收输入：  F 作业号
退出输入：  Q

作业号 1  分配  130 KB

起始 终止 长度 状态  ID
  0  129  130    1    1
130  639  510    0   -1

-----------------------------------------------------------------
|            |                                                  |
|      1     |                                                  |
|            |                                                  |
-----------------------------------------------------------------
0           130                                                640

*********************************************************************************

作业号 2  分配  60 KB

起始 终止 长度 状态  ID
  0  129  130    1    1
130  189   60    1    2
190  639  450    0   -1

-----------------------------------------------------------------
|            |     |                                            |
|      1     |   2 |                                            |
|            |     |                                            |
-----------------------------------------------------------------
0           130   190                                          640

*********************************************************************************
```

图 8 作业 1 分配 130KB 作业 2 分配 60KB 时的分区状态

```
作业号 3  分配  100 KB

起始 终止 长度 状态  ID
  0  129  130    1    1
130  189   60    1    2
190  289  100    1    3
290  639  350    0   -1

-----------------------------------------------------------------
|            |     |       |                                    |
|      1     |   2 |   3   |                                    |
|            |     |       |                                    |
-----------------------------------------------------------------
0           130   190       290                                640

*********************************************************************************

作业号 2  已回收

起始 终止 长度 状态  ID
  0  129  130    1    1
130  189   60    0   -1
190  289  100    1    3
290  639  350    0   -1

-----------------------------------------------------------------
|            |     |       |                                    |
|      1     |     |   3   |                                    |
|            |     |       |                                    |
-----------------------------------------------------------------
0           130   190       290                                640

*********************************************************************************
```

图 9 作业 3 分配 100KB 作业 2 被回收时的分区状态

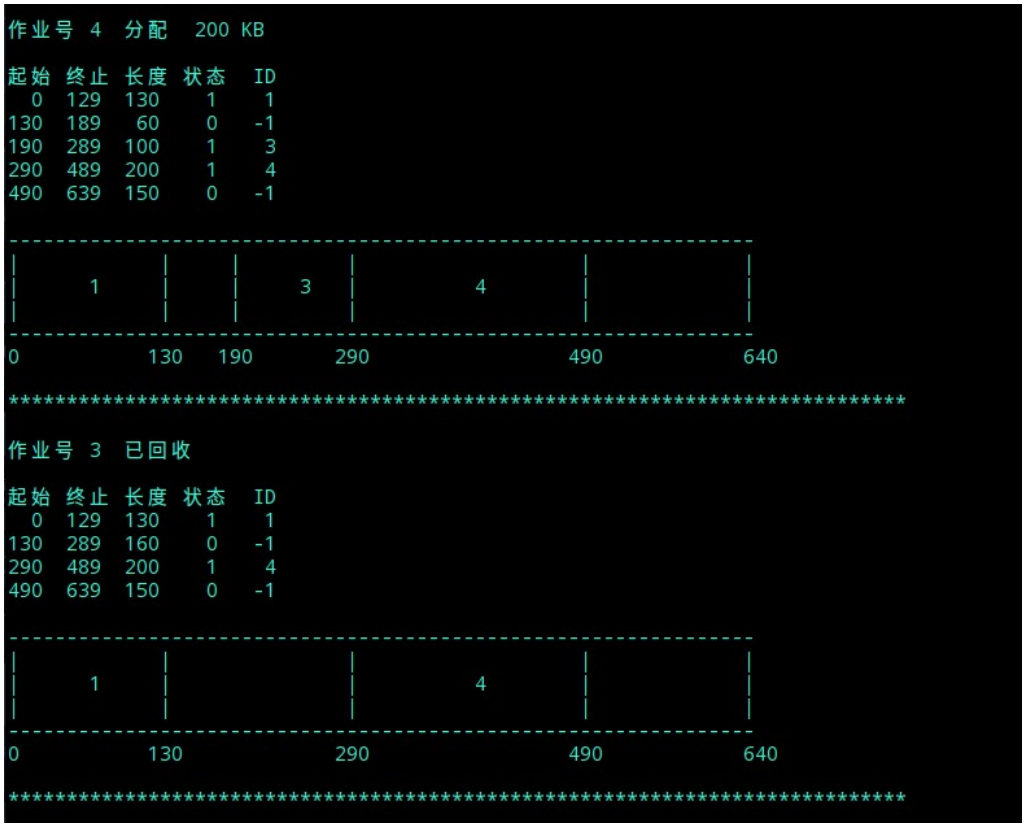

图 10 作业 4 分配 200KB 作业 3 被回收时的分区状态

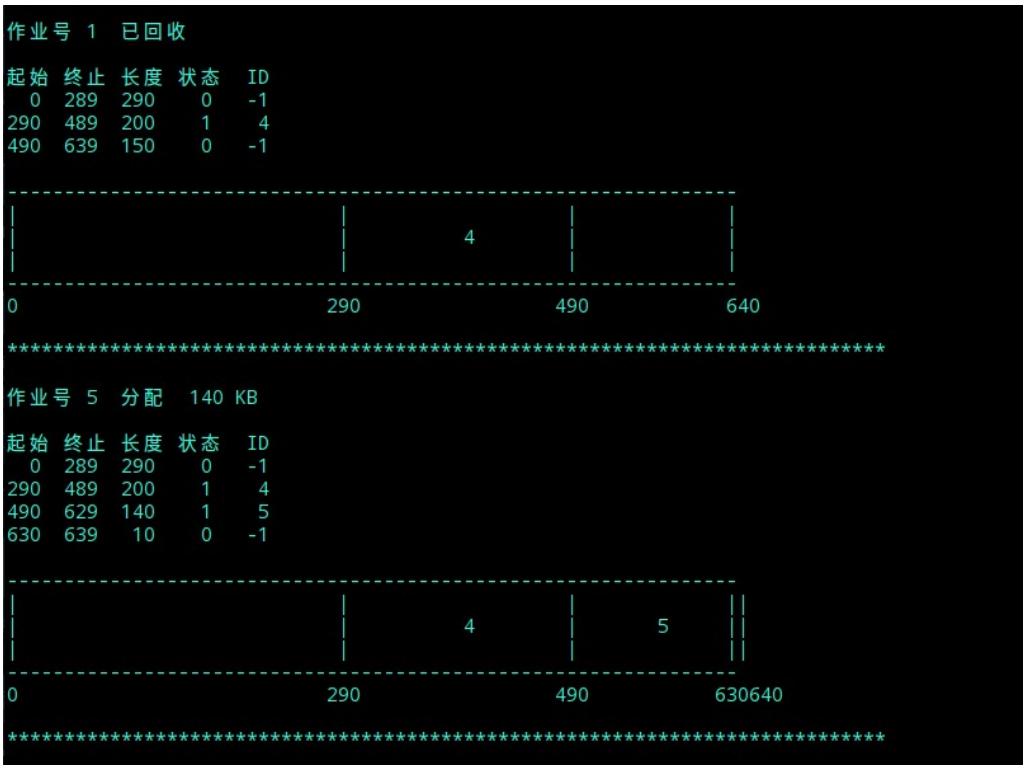

图 11 作业 1 被回收 作业 5 分配 140KB 时的分区状态

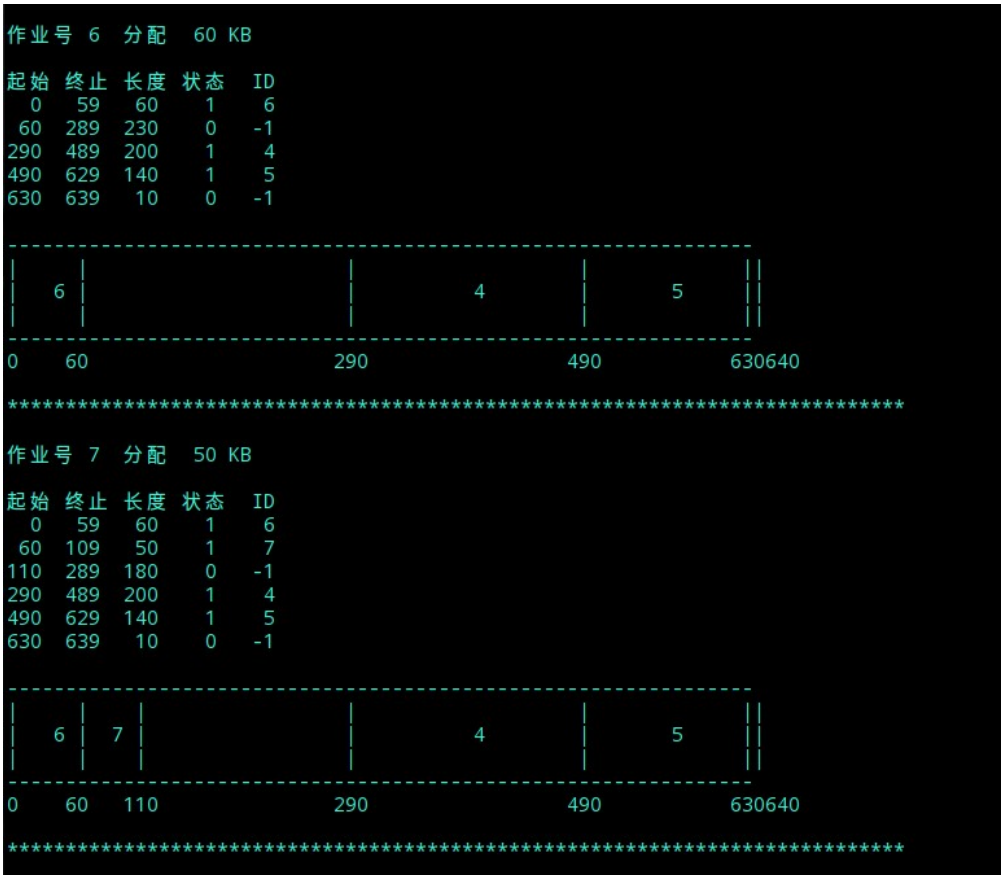

图 12 作业 6 分配 60KB 作业 7 分配 50KB 时的分区状态

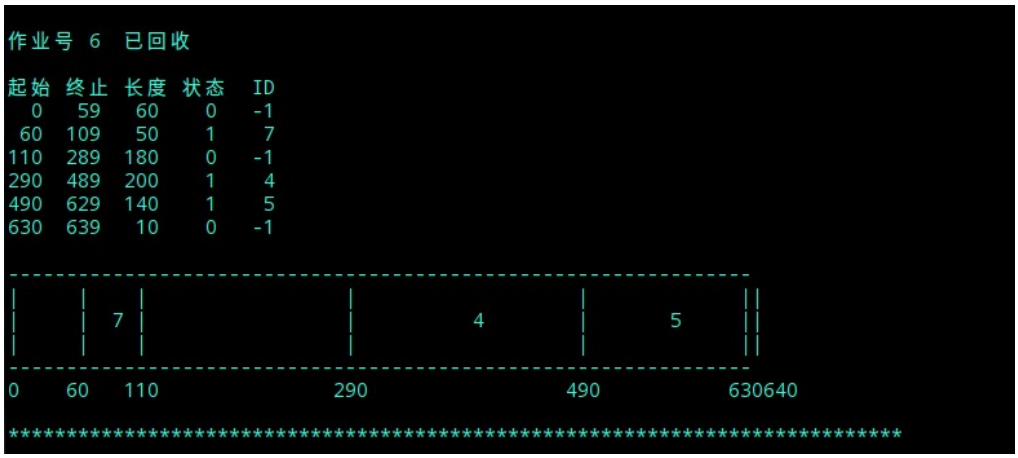

图 13 作业被回收时的分区状态

参考文献

[1] 汤子瀛，计算机操作系统（第四版），西安：西安电子科技大学出版社，2004.05，127-131

[2] 梁红兵 汤小丹，计算机操作系统（第四版）学习指导与题解，西安：西安电子科技大学出版社，2005.02，97-98

[3] Peter Baer Galvin、Greg Gagne 著、 郑扣根译，操作系统概念（第七版），北京：高等教育出版社，2010.01，243-246

[4] 严蔚敏 吴伟民，数据结构（C 语言版），北京：清华大学出版社，2016.12，18-43

[5] 邓曦辉，动态分区分配与回收算法的模拟[J]，电脑开发与应用，2013，26（4），61-63

心得体会

通过这次实验，我更加了解了内存连续分配的方式，动态分区分配过程。通过亲自的编程实践，把课程上学的操作系统原理很好的联系了起来，更加深了对于理论的理解。同时，通过理论的学习，也让编程更加得心应手。

我所选的这一题仅要求实现循环首次适应分配算法，由于这几个算法分配分区和回收分区的算法都是相同的，仅仅是各自所寻找的合适的位置不同，所以可以很容易的一起实现，作为扩展功能。只要底层的 alloc()函数将各个算法返回的位置作为参数即可同时实现。这也体现了分层的思想。

对于分区管理的数据结构，最开始考虑有两种解决方案，一个是设置一个空闲分区链表和一个已分配分区链表，另一个是只设置一个分区链表，通过标志位来区别是否被分配或空闲。综合考虑各种因素后决定选择后者。

在已经实现基础功能的情况下，为了可以更直观清晰的显示分配和回收分区的过程，自己也设计了一个绘制内存状态图的算法。由于要输出的美观，可以按照内存分配的实际比例绘图，而且图里的数字尽量居中对齐，自己设计调试了好久才达到满意的程度。最终完成时也很有成就感。

最后感谢老师的指导，让我对程序有了更进一步的认识和改进的空间。操作系统的五次实验和这次的课程设计使自己的理论知识和编程能力都有了较大的提高，在今后的学习与编程中，要把理论和实践结合起来不断学习，这样自己的知识能力体系才能更加完备，为更深入的学习打下良好的基础。