



***Liaoning Normal University***

采用最先适应算法的可变式分区的分配和回收

**课程名称：操作系统实验报告**

**专 业：计算机科学与技术（师范）**

**班 级：2016级（1）班**

**学 号：201621012299**

**姓 名：周惠馨**

**指导教师：张凤梅**

**2018年11月6日**

1. **实验目的**

通过本试验掌握可变式分区的分配和回收的原理。

1. **实验环境**

Linux下的C，Turbo C或Visual C

1. **实验内容**

**1．数据结构**

**空闲表：**

0

5K

10K

14k

26K

32K

128K

|  |
| --- |
| **OS** |
| **作业1** |
| **作业3** |
| **空闲区** |
| **作业2** |
| **空闲区** |

|  |  |  |
| --- | --- | --- |
| **起址** | **长度** | **状态** |
| **14K** | **12K** | **0** |
| **32K** | **96K** | **0** |
|  |  | **-1** |
|  |  | **-1** |

**1．分配流程：可变分区的分配算法**

**Loc＝F的起始地址**

**开始**

**输入申请一个Xk大小分区**

**置空闲区号F=0**

**F已超出最大项号，且F状态空表目**

**输出无法分配**

**Y**

**N**

**F的大小≥Xk**

**大于**

**F的大小－Xk＝新空闲块大小，loc＋Xk＝新起始地址**

**小于**

**等**

**于**

**F后非空表目上移，**

**删除空表目**

**结束**

**F=F+1**

**2．回收流程**

**开始**

**输入回收区R的起址loc和大小Size**

**i=0**

**table[i].add<loc**

**Y**

**N**

**F1即为table[i]**

**F2即为table[i-1]**

**分区R与F2上相邻，即**

**table[i-1].add+table[i-1].size==loc**

**块B**

**i=i+1**

**Y**

**块A**

**N**

**块A**

**分区R与F1下相邻，即**

**table[i].add==loc+size**

**table[i].size+=size**

**table[i].add=loc**

**结束**

**空闲表F1及其后空闲项下移。**

**table[i].add=loc**

**table[i].size=size**

**N**

**Y**

**块B**

**分区R与F1下相邻，即**

**table[i].add=loc+size**

**Y**

**size=size+ table[i].size**

**table[i-1].size=size**

**结束**

**空闲表删除F1**

**table[i-1].size=size**

**N**

**size=size+table[i-1]. size**

1. **实验代码**

#include "stdio.h"

#define N 5

typedef struct table

{

int address;

int length;

int state;

}table;

table tables[N]={{14,12,0},{32,96,0}};

//分配流程

int fenpei()

{

int i,flag=0;

int length;

printf("\n请输入长度：");

scanf("%d",&length);

for(i=0;i<N;i++)

{

if(tables[i].state==0 && tables[i].length>length)//空闲区长度大于作业长度

{

tables[i].address=tables[i].address+length;

tables[i].length=tables[i].length-length;

flag=1;

return tables[i].address-length;

}

if(tables[i].state==0 && tables[i].length==length)//空闲区长度等于作业长度

{

tables[i].state=-1;

flag=1;

return tables[i].address;

}

}

if(flag==0)

return -1;

}

//回收流程

void huishou()

{

int address,length;

int i,j,count=0;

int mark = 1;

printf("\n请输入地址：");

scanf("%d",&address);

printf("\n请输入长度：");

scanf("%d",&length);

for(i=0; i<N; i++)

{

if(tables[i].address>=address)//找到一个空闲区地址大于等于回收地址

{

mark = 0;

if(tables[i-1].address+tables[i-1].length>=address)//上一个空闲区与回收地址重叠

{

if(tables[i].address<=address+length)//空闲区与回收作业长度重叠

{

if(tables[i].address<address+length)//空闲区地址与回收作业长度重叠

{

printf("\n已分配！\n");

return ;

}

else{

tables[i].length=tables[i].address+tables[i].length-ables[i-1].address;

tables[i].address = tables[i-1].address;

tables[i-1].address=0;

tables[i-1].length=0;

tables[i-1].state=0;

}

}

else if(tables[i-1].address+tables[i-1].length<=address)//上相邻

{

tables[i-1].length+=length;

tables[i-1].state=0;

}else if((tables[i-1].address<=address)&&(tables[i-1].address+tables[i-1].length>=address + length))

{

printf("\n已分配！\n");

}

else{

tables[i-1].length = address + length - tables[i-1].address;

tables[i-1].state=0;

}

}

else if((tables[i].address>address)&&(tables[i].address>address+length))//不相邻

{

for(j=N-1; j>=i; j--)

{

tables[j+1] = tables[j];

}

tables[i].address = address;

tables[i].length = length;

tables[i].state=0;

}

else if((tables[i].address>=address)&&(tables[i].address>address+length))

{

length = tables[i].address-address;

tables[i].length = length + tables[i].length;

tables[i].address = address;

tables[i].state=0;

}

else if(tables[i].address<=address+length)//下相邻

{

if(tables[i].address<address+length)

{

printf("已分配！\n");

return ;

}

else{

length = tables[i].address-address;

tables[i].length = length + tables[i].length;

tables[i].address=address;

tables[i].state=0;

}

}

else {

printf("\n已分配！\n");

return ;

}

printf("成功！\n");

break;

}

}

if(mark==1)

{

for(i=0; i<N; i++)

{

if((tables[i].address!=0)&&(tables[i].length!=0)&&(tables[i].state==0))

continue;

else{

tables[i].address = address;

tables[i].length = length;

tables[i].state = 0;

printf("\n成功！\n");

break;

}}}}

//输出

void print()

{

int i;

printf("\n 起址 \t 长度 \t 状态 \t\n");

for(i=0;i<N;i++)

{

if(tables[i].state!=0)

continue;

if(tables[i].state==0)

printf(" %4d \t %4d \t %4d \t\n",tables[i].address,tables[i].length,tables[i].state);

}

}

int main()

{

int m;

print();

while(1)

{

printf("\n1.分配\n");

printf("2.回收\n");

printf("3.退出\n");

printf("请输入你的选择：");

scanf("%d",&m);

switch(m)

{

case 1:fenpei();print();break;

case 2:huishou();print();break;

case 3:exit(0);

}

}

return 0;

}

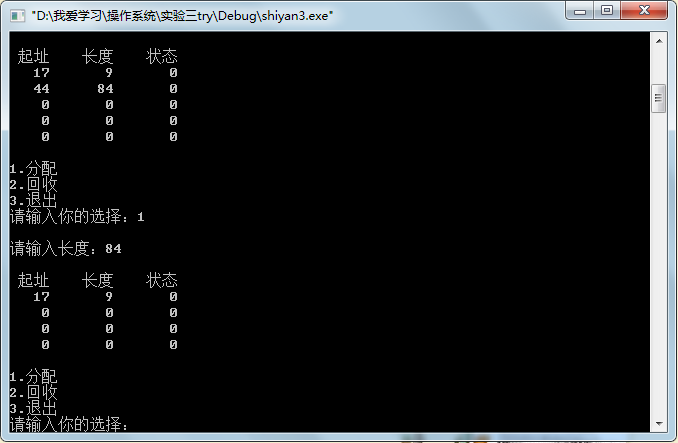
1. **实验结果**
2. 分配
3. 分配空间大小小于第一个空间长度



1. 分配空间大小大于第一个空闲区长度小于第二个空闲区长度



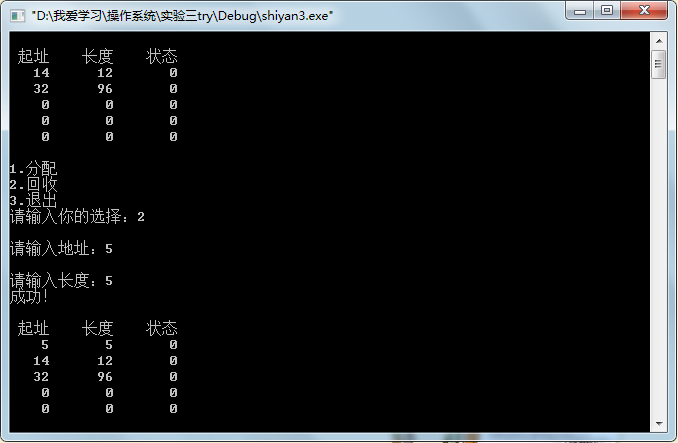
1. 分配空间大小等于某一个空闲区长度



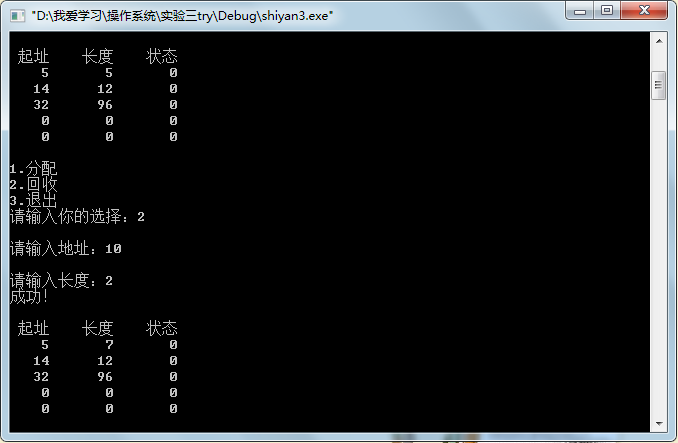
1. 分配空间大小大于所有空闲区长度



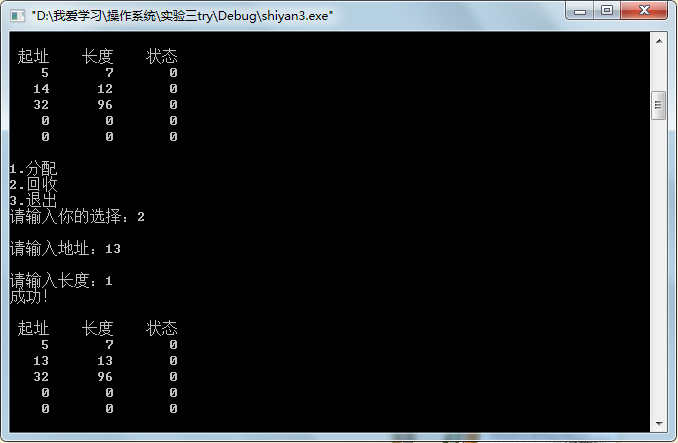
1. 回收
2. 回收空间地址与空闲区不相邻



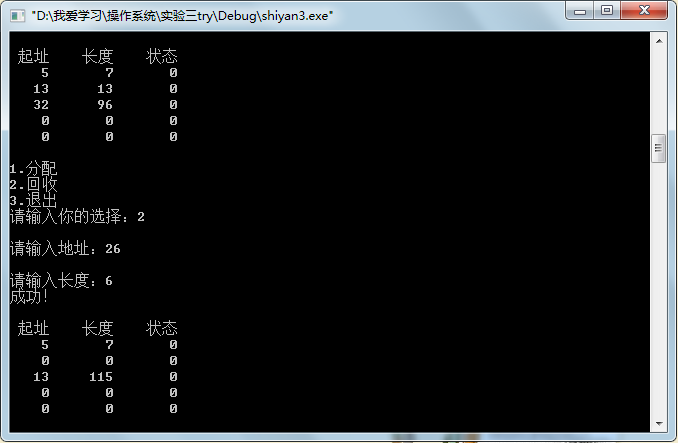
1. 回收空间地址与某个空闲区上相邻



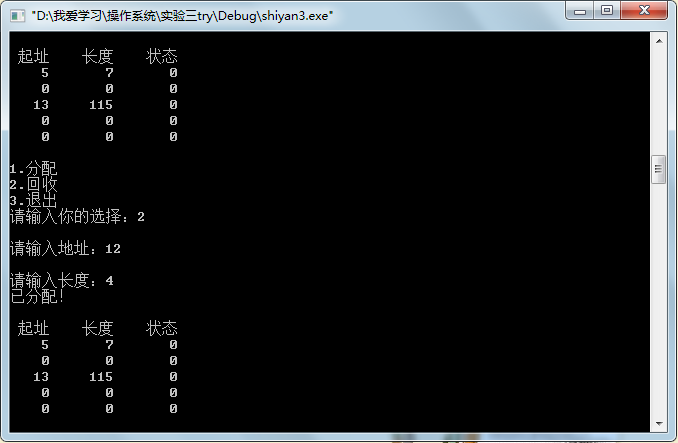
1. 回收空间地址与某个空闲区下相邻



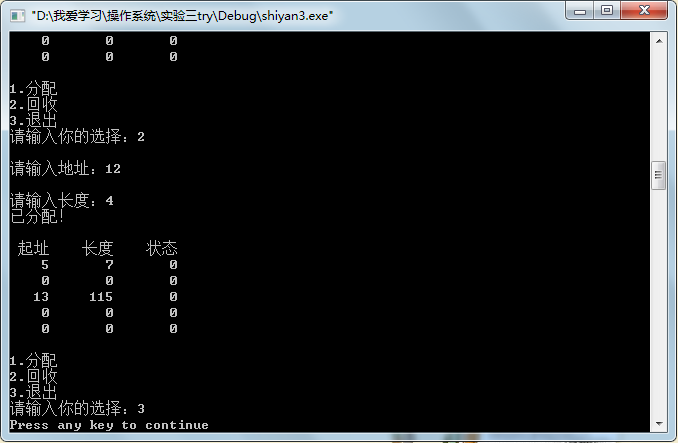
1. 回收空间地址与空闲区上下都相邻



1. 回收空间与空闲区有重叠



1. 退出



1. **思考分析**

在做回收的时候要将数据的地址考虑进去，不能只改变状态。