**实验三 存储管理实验**

**一.  目的要求：**

1、通过编写和调试存储管理的模拟程序以加深对存储管理方案的理解。熟悉虚存管理的各种页面淘汰算法。

2、通过编写和调试地址转换过程的模拟程序以加强对地址转换过程的了解。

**二．实验内容：**

1、设计一个固定式分区分配的存储管理方案，并模拟实现分区的分配和回收过程。

可以假定每个作业都是批处理作业，并且不允许动态申请内存。为实现分区的分配和回收，可以设定一个分区说明表，按照表中的有关信息进行分配，并根据分区的分配和回收情况修改该表。

**算法描述：**

本算法将内存的用户区分成大小相等的四个的分区，设一张分区说明表用来记录分区，其中分区的表项有分区的大小、起始地址和分区的状态，当系统为某个作业分配主存空间时，根据所需要的内存容量，在分区表中找到一个足够大的空闲分区分配给它，然后将此作业装入内存。如果找不到足够大的空闲分区，则这个作业暂时无法分配内存空间，系统将调度另一个作业。当一个作业运行结束时，系统将回收改作业所占据的分区并将该分区改为空闲。

**算法原程序**

#include "stdio.h"

#include "windows.h"

#include <stdlib.h>

#include <conio.h>

#define PCB\_NUM 5 //模拟进程数量

#define INT 800//内存分区数量

struct MemInf

**{**

int addr**;** //分区起始地址

int size**;** //分区大小

int state**;** //0表示空闲，>0时表示已分配，存储的是进程ID

**};**

struct PCB

**{**

int pcbID**;** //进程ID

int size**;** //进程大小

int RunState**;** //运行状态,0表示就绪,1表示已分配内存正运行,2表示运行结束且退出内存

int TolTime**;** //总需要运行时间

int RunTime**;** //已运行时间

**}** pcbList**[**PCB\_NUM**];**

void menu**()** //菜单

**{**

int m**;**

system**(**"cls"**);**

printf**(**"\n\n\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\t\t\n"**);**

printf**(**"\t\t\t\t固定分区存储程序演示\n"**);**

printf**(**"\t\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\t\t\n"**);**

printf**(**"\n\t\t\t1.执行程序."**);**

printf**(**"\n\t\t\t0.退出程序."**);**

scanf**(**"%d"**,&**m**);**

**switch(**m**)**

**{**

**case** 1**:**

**break;**

**case** 0**:**

system**(**"cls"**);**

menu**();**

**break;**

**default:**

system**(**"cls"**);**

**break;**

**}**

**}**

void paixu**(**struct MemInf**\*** ComMem**,**int n**)**

**{**

int i**,**j**,**t**;**

**for(**j**=**0**;** j**<**n**-**1**;** j**++)**

**for(**i**=**0**;** i**<**n**-**j**-**1**;** i**++)**

**if(**ComMem**[**i**].**size**>**ComMem**[**i**+**1**].**size**)**

**{**

t**=**ComMem**[**i**].**size**;**

ComMem**[**i**].**size**=**ComMem**[**i**+**1**].**size**;**

ComMem**[**i**+**1**].**size**=**t**;**

**}**

**}**

void paixu2**()**

**{**

int i**,**j**,**t**;**

**for(**j**=**0**;** j**<**4**;** j**++)**

**for(**i**=**0**;** i**<**4**-**j**;** i**++)**

**if(**pcbList**[**i**].**size**>**pcbList**[**i**+**1**].**size**)**

**{**

t**=**pcbList**[**i**].**size**;**

pcbList**[**i**].**size**=**pcbList**[**i**+**1**].**size**;**

pcbList**[**i**+**1**].**size**=**t**;**

**}**

**}**

void main**()**

**{**

DD**:**

menu**();**

char ch**;**

int i**,**j**,**n**,**a**=**0**;**

struct MemInf**\*** ComMem**;**

system**(**"cls"**);**

printf**(**"你要分多少个分区呢，请输入数值吧:"**);**

scanf**(**"%d"**,&**n**);**

ComMem**=(**struct MemInf**\*)**malloc**(**n**\*sizeof(**struct MemInf**));**

printf**(**"请划分内存固定大小分区:\n"**);**//划分内存固定大小分区

**for(**i**=**0**;** i**<**n**;** i**++){**

printf**(**"输入固定分区%d分区的长度:"**,**i**);**//输入固定分区每个的长度

scanf**(**"%d"**,&**ComMem**[**i**].**size**);**

**if(**i**==**0**)** ComMem**[**i**].**addr**=**40**;**//定义第一个分区的起始地址为40

**else** ComMem**[**i**].**addr**=**ComMem**[**i**-**1**].**addr**+**ComMem**[**i**-**1**].**size**;**//表示下一个起始地址的起始状态表示的是分区长度加上起始地址长度

ComMem**[**i**].**state**=**0**;**//表示状态为未分配

a**=**ComMem**[**i**].**size**+**a**;**

**if(**a**>=**INT**)**

**{**

printf**(**"超出规定内存范围"**);**

ch**=**getchar**();**

ch**=**getchar**();**

**goto** DD**;**

**}**

**}**

paixu**(**ComMem**,**n**);**

//初始化就绪进程队列

pcbList**[**0**].**pcbID **=**1**;**

pcbList**[**0**].**RunState **=**0**;** //运行状态,0表示就绪,1表示已分配内存正运行,2表示运行结束且退出内存

pcbList**[**0**].**size**=**30**;**

pcbList**[**0**].**RunTime **=**0**;**

pcbList**[**0**].**TolTime **=**5**;**

pcbList**[**1**].**pcbID **=**2**;**

pcbList**[**1**].**RunState **=**0**;**

pcbList**[**1**].**size**=**15**;**

pcbList**[**1**].**RunTime **=**0**;**

pcbList**[**1**].**TolTime **=**6**;**

pcbList**[**2**].**pcbID **=**3**;**

pcbList**[**2**].**RunState **=**0**;**

pcbList**[**2**].**size**=**50**;**

pcbList**[**2**].**RunTime **=**0**;**

pcbList**[**2**].**TolTime **=**3**;**

pcbList**[**3**].**pcbID **=**4**;**

pcbList**[**3**].**RunState **=**0**;**

pcbList**[**3**].**size**=**120**;**

pcbList**[**3**].**RunTime **=**0**;**

pcbList**[**3**].**TolTime **=**4**;**

pcbList**[**4**].**pcbID **=**5**;**

pcbList**[**4**].**RunState **=**0**;**

pcbList**[**4**].**size**=**125**;**

pcbList**[**4**].**RunTime **=**0**;**

pcbList**[**4**].**TolTime **=**9**;**

ch**=**getchar**();**

ch**=**getchar**();**

**while(**pcbList**[**PCB\_NUM**-**1**].**RunTime **<** pcbList**[**PCB\_NUM**-**1**].**TolTime**)**

**{**

**{**

**for(**j**=**0**;** j**<**PCB\_NUM**;** j**++)**

**{**//内存分配

**for(**i**=**0**;** i**<**n**;** i**++)**

**{**

**if(**ComMem**[**i**].**state **==**0 **&&** pcbList**[**j**].**RunState**==**0 **)** //内存分区为0空闲,且进程状态为就绪，即可以考虑分配该内存分区

**if(**ComMem**[**i**].**size **>=** pcbList**[**j**].**size**)** //如果该内存分区空间大于或等于进程空间，即可以把该空闲内存分区分配给该进程

**{**

ComMem**[**i**].**state **=**pcbList**[**j**].**pcbID **;**

pcbList**[**j**].**RunState**=**1**;**

**}**

**}**//内存回收

**if(**pcbList**[**j**].**RunTime **>=**pcbList**[**j**].**TolTime**)** //如果该进程运行时间大于或等于总需时间，即可回收该进程占用内存

**for(**i**=**0**;** i**<**n**;** i**++)**

**if(**ComMem**[**i**].**state **==** pcbList**[**j**].**pcbID**)**

**{**

ComMem**[**i**].**state **=** 0**;** //内存状态变为 "未分配"

pcbList**[**j**].**RunState**=**2**;** //进程状态变为 "运行完毕"

**}**

//运行时间加1

**for(**i**=**0**;** i**<**PCB\_NUM**;** i**++)**

**if** **(**pcbList**[**i**].**RunState**==**1 **&&** pcbList**[**i**].**RunTime **<** pcbList**[**i**].**TolTime**)** //处于运行状态且运行时间小于总需时间的进程，运行时间加1

pcbList**[**i**].**RunTime**++;**

//显示模块

printf**(**"进程ID\t 进程大小\t 状态\t 需要时间\t 运行时间\n"**);**

**for(**i**=**0**;** i**<**PCB\_NUM**;** i**++)**

printf**(**"%d\t %d\t\t %d\t %d\t\t %d\n"**,**pcbList**[**i**].**pcbID**,**pcbList**[**i**].**size**,** pcbList**[**i**].**RunState**,** pcbList**[**i**].**TolTime **,**pcbList**[**i**].**RunTime**);**

printf**(**"分区ID\t 分区大小\t 状态\n"**);**

**for(**i**=**0**;** i**<**n**;** i**++)**

printf**(**"%d\t %d\t\t %d\n"**,**i**,**ComMem**[**i**].**size **,**ComMem**[**i**].**state **);**

printf**(**"按回车键继续...\n"**);**

getchar**();** //按任意键继续(分步执行，以便观察内存分配回收

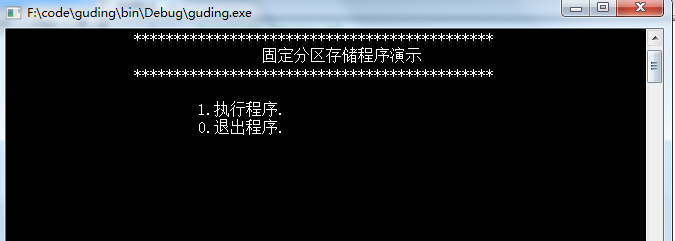
**}**

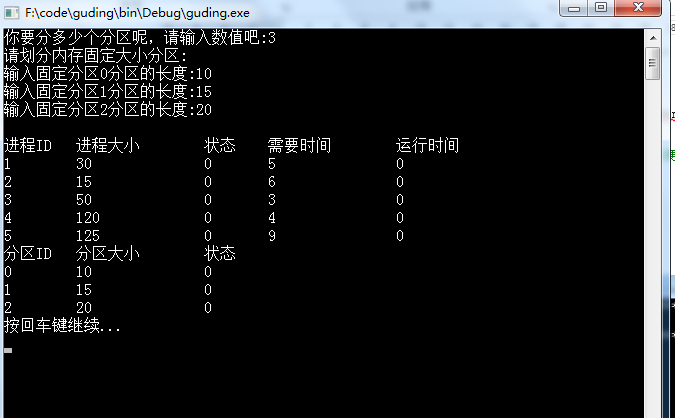
**}**

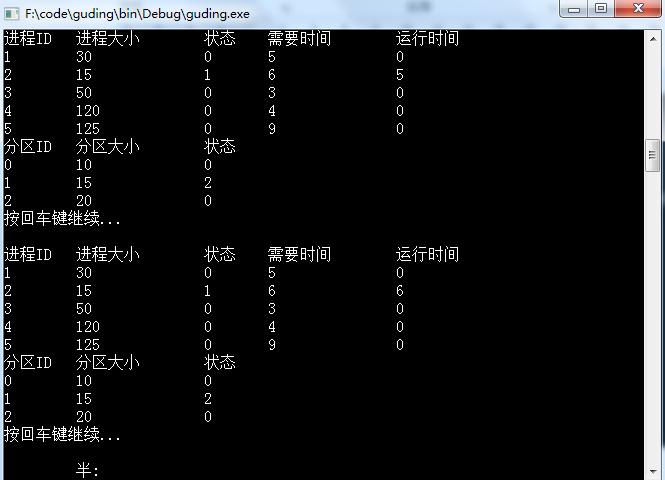
**}**

**}**

**运行结果截图：**







2、设计一个可变式分区分配的存储管理方案。并模拟实现分区的分配和回收过程。

对分区的管理法可以是下面三种算法之一：

首次适应算法

循环首次适应算法

最佳适应算法

* **代码实现部分：**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define max 100

**typedef** struct node

**{**

int start**;**

int length**;**

char tag**[**20**];**

**}**job**;**

job frees**[**max**];**

job occupys**[**max**];**

int free\_quantity**,**occupy\_quantity**;**

int SIZE**;**

void initial**()** //初始化函数

**{**

int i**;**

**for(**i**=**0**;**i**<**max**;**i**++)**

**{**

frees**[**i**].**start**=-**1**;**

frees**[**i**].**length**=**0**;**

strcpy**(**frees**[**i**].**tag**,**"free"**);**

occupys**[**i**].**start**=-**1**;**

occupys**[**i**].**length**=**0**;**

strcpy**(**occupys**[**i**].**tag**,**""**);**

**}**

free\_quantity**=**0**;**

occupy\_quantity**=**0**;**

**}**

void writedata**()** //把分区函数写入磁盘文件

**{**

FILE **\***fp**;**

char fname**[**20**];**

int i**,**j**;**

printf**(**"请输入初始空闲表文件名：\n"**);**

scanf**(**"%s"**,&**fname**);**

printf**(**"现在进行初始化空闲分区!\n"**);**

printf**(**"请输入您要建立的空闲分区数：\n"**);**

scanf**(**"%d"**,&**SIZE**);**

**for(**i**=**0**;**i**<**SIZE**;**i**++)**

**{**

j**=**i**+**1**;**

printf**(**"输入第%d个分区的起始地址：\n"**,**j**);**

scanf**(**"%d"**,&**frees**[**i**].**start**);**

printf**(**"输入第%d个分区的长度：\n"**,**j**);**

scanf**(**"%d"**,&**frees**[**i**].**length**);**

**}**

**if((**fp**=**fopen**(**fname**,**"wb"**))==NULL)**

printf**(**"错误,文件打不开,请检查文件名\n"**);**

**for(**i**=**0**;**i**<**SIZE**;**i**++)**

**if(**fwrite**(&**frees**[**i**],sizeof(**struct node**),**1**,**fp**)!=**1**)**

printf**(**"文件写入错误!\n"**);**

fclose**(**fp**);**

**}**

void readdata**()** //从文件读入分区表函数

**{**

FILE **\***fp**;**

char fname**[**20**];**

printf**(**"请输入读入空闲表文件名:"**);** //输入空闲表文件的文件名

scanf**(**"%s"**,&**fname**);**

**if((**fp**=**fopen**(**fname**,**"rb"**))==NULL)**

**{**

printf**(**"错误,文件打不开,请检查文件名\n"**);**

exit**(**0**);**

**}**

**else**

**{** int i**;**

**for(**i**=**0**;**i**<**SIZE**;**i**++)**

**{**

fread**(&**frees**[**i**],sizeof(**struct node**),**1**,**fp**);**

free\_quantity**++;**

**}**

**}**

fclose**(**fp**);**

**}**

void sort**()** //排序空闲表

**{** int i**;**

**for(**i**=**0**;**i**<**free\_quantity**-**1**;**i**++)**

**{**

int t**=**i**;**

int j**;**

**for(**j**=**i**+**1**;**j**<**free\_quantity**;**j**++)**

**{**

**if(**frees**[**j**].**start**<=**frees**[**t**].**start**)**

t**=**j**;**

**}**

frees**[**free\_quantity**].**start**=**frees**[**i**].**start**;**

frees**[**free\_quantity**].**length**=**frees**[**i**].**length**;**

frees**[**i**].**start**=**frees**[**t**].**start**;**

frees**[**i**].**length**=**frees**[**t**].**length**;**

frees**[**t**].**start**=**frees**[**free\_quantity**].**start**;**

frees**[**t**].**length**=**frees**[**free\_quantity**].**length**;**

**}**

**}**

void view**()** //显示分区信息

**{**int i**,**j**;**

printf**(**"空闲分区表显示如下：\n"**);**

printf**(**"起始地址\t长度\t状态标志\n"**);**

**for(**i**=**0**;**i**<**free\_quantity**;**i**++)**

printf**(**"%6dk\t%10dk\t%s\t\n"**,**frees**[**i**].**start**,**frees**[**i**].**length**,**frees**[**i**].**tag**);**

printf**(**"\n\n已分配分区表显示如下：\n"**);**

printf**(**"起始地址\t长度\t占用作业名\n"**);**

**for(**j**=**0**;**j**<**occupy\_quantity**;**j**++)**

printf**(**"%6dk\t%10dk\t%s\t\n"**,**occupys**[**j**].**start**,**occupys**[**j**].**length**,**occupys**[**j**].**tag**);**

getchar**();**

getchar**();**

**}**

void earliest**()** //首次适应算法

**{**

char jobname**[**20**];**

int joblength**,**f**=**0**;**

int i**,**j**;**

printf**(**"请输入作业名：\n"**);**

scanf**(**"%s"**,&**jobname**);**

printf**(**"输入作业的长度：\n"**);**

scanf**(**"%d"**,&**joblength**);**

**for(**i**=**0**;**i**<**free\_quantity**;**i**++)**

**{**

**if(**frees**[**i**].**length**>=**joblength**)**

f**=**1**;**

**}**

**if(**f**==**0**)**

**{**

printf**(**"\n当前没有能满足你申请长度的空闲内存,请稍候再试\n"**);**

getchar**();**

**}**

**else**

**{** //找到了满足的空间

int t**=**0**;**

j**=**0**;**

**while(**t**==**0**)**

**{**

**if(**frees**[**j**].**length**>=**joblength**)**

**{**

t**=**1**;**

**}**

j**++;**

**}**

j**--;**

occupys**[**occupy\_quantity**].**start**=**frees**[**j**].**start**;** //分配满足条件的空间

strcpy**(**occupys**[**occupy\_quantity**].**tag**,**jobname**);**

occupys**[**occupy\_quantity**].**length**=**joblength**;**

occupy\_quantity**++;**

**if(**frees**[**j**].**length**>**joblength**)**

**{**

frees**[**j**].**start**+=**joblength**;**

frees**[**j**].**length**-=**joblength**;**

**}**

**else**

**{**

**for(**i**=**j**;**i**<**free\_quantity**-**1**;**i**++)**

**{**

frees**[**i**].**start**=**frees**[**i**+**1**].**start**;**

frees**[**i**].**length**=**frees**[**i**+**1**].**length**;**

**}**

free\_quantity**--;**

**}**

printf**(**"作业申请内存空间成功!\n"**);**

getchar**();**

getchar**();**

**}**

**}**

void excellent**()** //最佳适应法

**{**

char jobname**[**20**];**

int joblength**,**f**=**0**;**

int i**,**j**;**printf**(**"请输入作业名：\n"**);**

scanf**(**"%s"**,&**jobname**);**

printf**(**"输入作业的长度：\n"**);**

scanf**(**"%d"**,&**joblength**);**

**for(**i**=**0**;**i**<**free\_quantity**;**i**++)**

**{**

**if(**frees**[**i**].**length**>=**joblength**)**

f**=**1**;**

**}**

**if(**f**==**0**)**

**{**

printf**(**"\n当前没有能满足你申请长度的空闲内存,请稍候再试\n"**);**

getchar**();**

**}**

**else** //找到了满足的空间

**{**

int t**=**0**;**

j**=**0**;**

**while(**t**==**0**)**

**{**

**if(**frees**[**j**].**length**>=**joblength**)**

**{**

t**=**1**;**

**}**

j**++;**

**}**

j**--;**

**for(**i**=**0**;**i**<**free\_quantity**;**i**++)**

**{**

**if(**frees**[**i**].**length**>=**joblength**&&**frees**[**i**].**length**<**frees**[**j**].**length**)**

j**=**i**;**

**}**

occupys**[**occupy\_quantity**].**start**=**frees**[**j**].**start**;** //分配空闲空间

strcpy**(**occupys**[**occupy\_quantity**].**tag**,**jobname**);**

occupys**[**occupy\_quantity**].**length**=**joblength**;**

occupy\_quantity**++;**

**if(**frees**[**j**].**length**>**joblength**)**

**{**

frees**[**j**].**start**+=**joblength**;**

frees**[**j**].**length**-=**joblength**;**

**}**

**else**

**{**

**for(**i**=**j**;**i**<**free\_quantity**-**1**;**i**++)**

**{**

frees**[**i**].**start**=**frees**[**i**+**1**].**start**;**

frees**[**i**].**length**=**frees**[**i**+**1**].**length**;**

**}**

free\_quantity**--;**

**}**

printf**(**"作业申请内存空间成功!\n"**);**

getchar**();**

getchar**();**

**}**

**}**

void worst**()**

**{**

char jobname**[**20**];**

int joblength**,**f**=**0**;**int i**,**j**;**

printf**(**"请输入作业名：\n"**);**

scanf**(**"%s"**,&**jobname**);**

printf**(**"输入作业的长度：\n"**);**

scanf**(**"%d"**,&**joblength**);**

**for(**i**=**0**;**i**<**free\_quantity**;**i**++)**

**{**

**if(**frees**[**i**].**length**>=**joblength**)**

f**=**1**;**

**}**

**if(**f**==**0**)**

**{**

printf**(**"\n当前没有能满足你申请长度的空闲内存,请稍候再试\n"**);**

getchar**();**

getchar**();**

**}**

**else** //找到了满足的空间

**{**

int t**=**0**;**

j**=**0**;**

**while(**t**==**0**)**

**{**

**if(**frees**[**j**].**length**>=**joblength**)**

**{**

t**=**1**;**

**}**

j**++;**

**}**

j**--;**

**for(**i**=**0**;**i**<**free\_quantity**;**i**++)**

**{**

**if(**frees**[**i**].**length**>=**joblength**&&**frees**[**i**].**length**>**frees**[**j**].**length**)**

j**=**i**;**

**}**

occupys**[**occupy\_quantity**].**start**=**frees**[**j**].**start**;** //分配空闲空间

strcpy**(**occupys**[**occupy\_quantity**].**tag**,**jobname**);**

occupys**[**occupy\_quantity**].**length**=**joblength**;**

occupy\_quantity**++;**

**if(**frees**[**j**].**length**>**joblength**)**

**{**

frees**[**j**].**start**+=**joblength**;**

frees**[**j**].**length**-=**joblength**;**

**}**

**else**

**{**

**for(**i**=**j**;**i**<**free\_quantity**-**1**;**i**++)**

**{**

frees**[**i**].**start**=**frees**[**i**+**1**].**start**;**

frees**[**i**].**length**=**frees**[**i**+**1**].**length**;**

**}**

free\_quantity**--;**

**}**

printf**(**"作业申请内存空间成功!\n"**);**

getchar**();**

getchar**();**

**}**

**}**

void main**()**

**{**

initial**();**

int n**;**

writedata**();**

system**(**"cls"**);**

readdata**();**

**for(;;)**

**{**

sort**();**

printf**(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"**);**

printf**(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"**);**

printf**(**"\*\* 欢迎使用可变分区存储管理系统 \*\*\n"**);**

printf**(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"**);**

printf**(**"\*\* 1.显示空闲表和分配表 \*\*\n"**);**

printf**(**"\*\* 2.首次适应算法 \*\*\n"**);**

printf**(**"\*\* 3.最佳适应算法 \*\*\n"**);**

printf**(**"\*\* 4.最坏适应算法 \*\*\n"**);**

printf**(**"\*\* 0.退出系统 \*\*\n"**);**

printf**(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"**);**

printf**(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"**);**

printf**(**"请输入您要选择的项目：\n"**);**

scanf**(**"%d"**,&**n**);**

**for(;;)**

**{**

**if(**n**<**0**||**n**>**4**)**

**{**

printf**(**"没有这个选项，请重新输入!"**);**

scanf**(**"%d"**,&**n**);**

**}**

**else**

**break;**

**}**

**switch(**n**)**

**{**

**case** 0**:**printf**(**"感谢您的使用!再见!\n"**);**exit**(**0**);**

**case** 1**:**view**();break;**

**case** 2**:**earliest**();break;**

**case** 3**:**excellent**();break;**

**case** 4**:**worst**();break;**

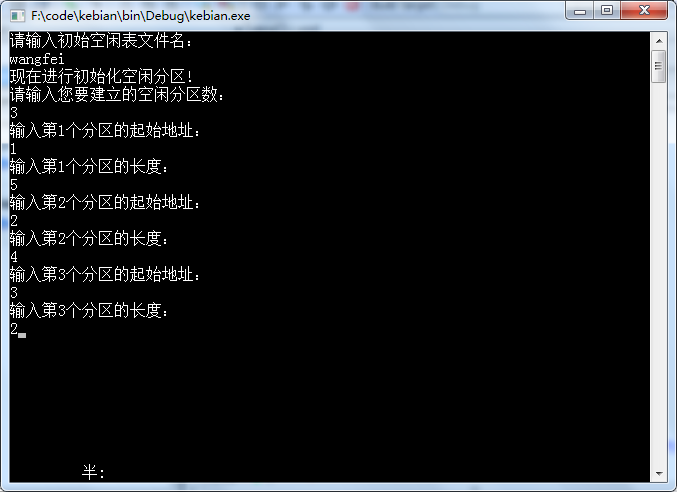
**}**

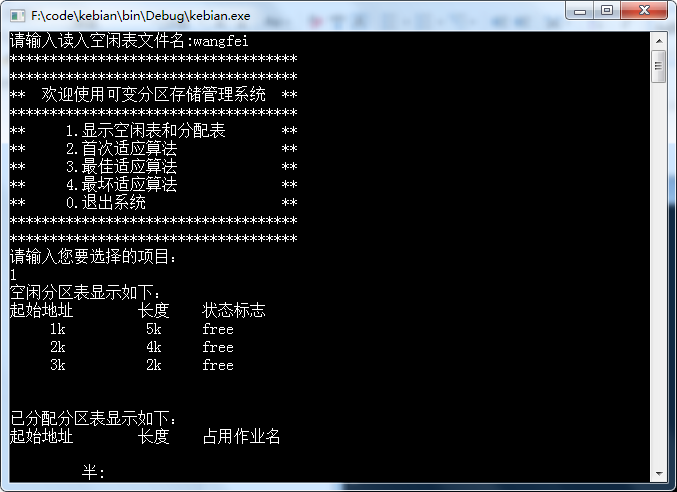
system**(**"cls"**);**

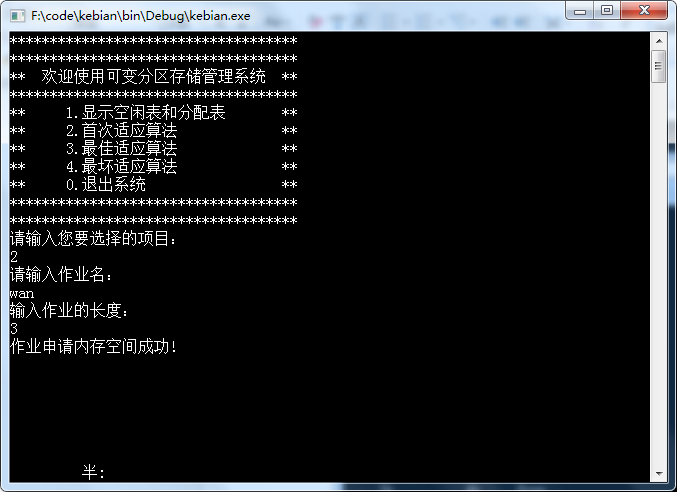
**}**

**}**

**测试结果：**

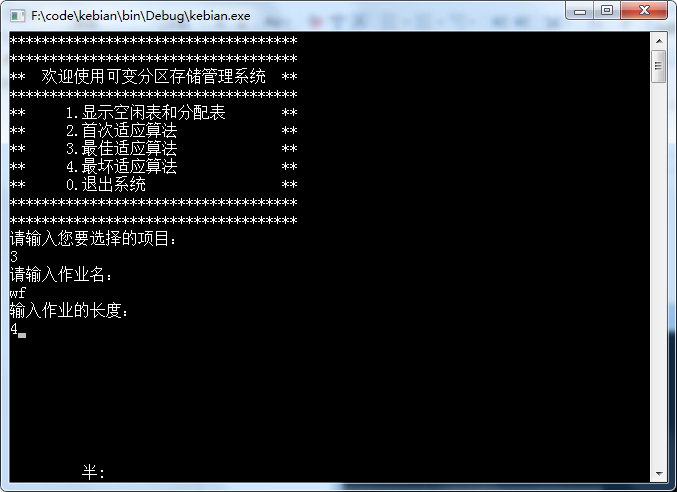


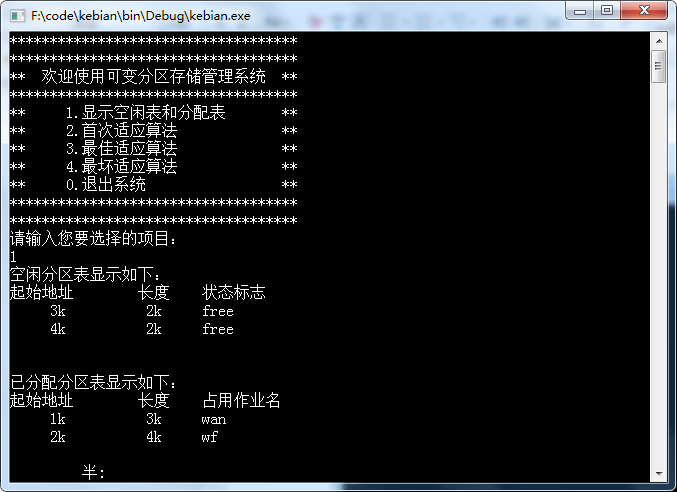


**使用首次适应算法的结果：**



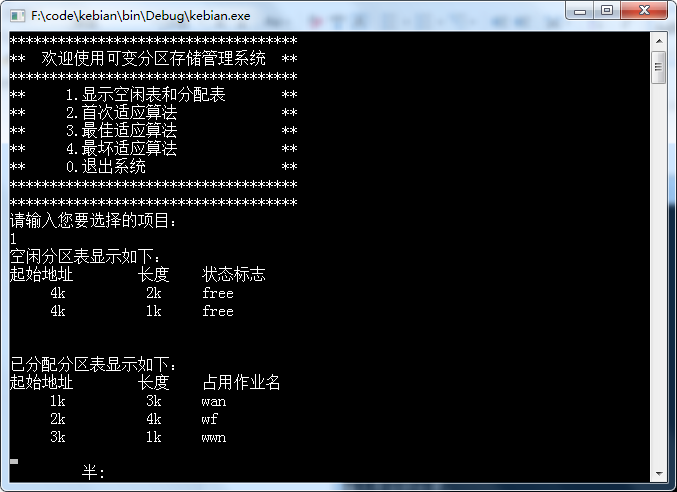
**使用最佳适应算法：**





**使用最坏适应算法：**





**内存过满：**



3、编写并调试一个段页式存储管理的地址转换的模拟程序。

　　首先设计好段表、页表，然后给出若干个有一定代表性的地址，通过查找段表页表后得到转换的地址。

要求打印转换前的地址，相应的段表，页表条款及转换后的地址，以便检查。

* **代码实现**

#include<iostream>

#include<string>

**using** **namespace** std**;**

**typedef** struct Quick

**{**

int qs**;**//快表段号

int qp**;**//快表页号

int qb**;**//快表段号

**}** Quick**;**

**typedef** struct Data

**{**

int num**;**//内存的块数

string str**;**//对应数据块的作业内容，简化起见说明内容为一串字符。

**}** Data**;**

//页表

**typedef** struct Page

**{**

int num**;**//页号

int flag**;**//页状态，即是否在内存。

int block**;**//该页对应的块号

**}** Page**;**

**typedef** struct Stack

**{**

int num**;**//段号

int flag**;**//段状态

int plen**;**//页表长度

int psta**;**//页表始址

**}** Stack**;**

//段表寄存器

**typedef** struct Stare

**{**

int ssta**;**//段表始址

int slen**;**//段表长度

**}** Stare**;**

Stack ss**[**10**];**////全局变量

Stare st**;**///////全局变量

Data work**[**20**];**//全局变量

Quick qu**;**//////全局变量

Page page**[**5**][**5**];**

bool menuflag**=**0**;**

int bbs**;**//内存块大小

int bs**;**//内存大小

void menu**();**

void start**();**

void change**();**

int main**()**

**{**

menu**();**

**return** 0**;**

**}**

void menu**()**

**{**

cout**<<**"请选择:"**<<**endl**;**

cout**<<**endl**;**

cout**<<**" 1、初始化表 "**<<**endl**;**

cout**<<**" 2、物理地址转换 "**<<**endl**;**

cout**<<**" 3、退出 "**<<**endl**;**

int menu1**;**

cin**>>**menu1**;**

**if(**menu1**!=**1**&&**menu1**!=**2**&&**menu1**!=**3**)**

**{**

cout**<<**"请输入正确的选项"**<<**endl**;**

menu**();**

**}**

**switch(**menu1**)**

**{**

**case** 1**:**

**{**

menuflag**=**1**;**

start**();**

**break;**

**}**

**case** 2**:**

**{**

**if(**menuflag**==**0**)**

**{**

cout**<<**"请初始化表"**<<**endl**;**

menu**();**

**}**

change**();**

**break;**

**}**

**case** 3**:**

**return;**

**}**//switch

**}**

void start**()**

**{**

cout**<<**"请输入内存大小(K)"**<<**endl**;**

cin**>>**bs**;**

cout**<<**"请输入内存块的大小(k)"**<<**endl**;**

cin**>>**bbs**;**

int blocknum**;**

blocknum**=**bs**/**bbs**;**

cout**<<**"内存一共被分为"**<<**blocknum**<<**"块，每块"**<<**bbs**<<**"k"**<<**"一共"**<<**bs**<<**"k"**<<**endl**;**

cout**<<**"请输入进程个数"**<<**endl**;**

int pn**;**

cin**>>**pn**;**

//下面求所有进程的总段数和段表,并为每段创建页表

int sums**=**0**;**

**for** **(**int pn1**=**0**;** pn1**<**pn**;** pn1**++)**

**{**

cout**<<**"请输入第"**<<**pn1**<<**"个进程的段数"**<<**endl**;**

int ppn**;**

cin**>>**ppn**;**

sums**+=**ppn**;**

**}**

**for(**int ss1**=**0**;** ss1**<**sums**;** ss1**++)**

**{**

cout**<<**"请输入第"**<<**ss1**<<**"个段表数据：段号，状态，页表长度，页表始址"**<<**endl**;**

cin**>>**ss**[**ss1**].**num**>>**ss**[**ss1**].**flag**>>**ss**[**ss1**].**plen**>>**ss**[**ss1**].**psta**;**

cout**<<**"请初始化第"**<<**ss1**<<**"段的页表，输入两个数据页表状态和对应块号"**<<**endl**;**

**for(**int sss1**=**0**;** sss1**<**ss**[**ss1**].**plen**;** sss1**++)**

**{**

page**[**ss1**][**sss1**].**num**=**sss1**;**

cout**<<**"请输入该段第"**<<**sss1**<<**"个页表的页表状态和对应块号"**<<**endl**;**

cin**>>**page**[**ss1**][**sss1**].**flag**>>**page**[**ss1**][**sss1**].**block**;**

**}**

**}**

//初始化段表寄存器

cout**<<**"初始化段表寄存器的段表始址"**<<**endl**;**

cin**>>**st**.**ssta**;**

st**.**slen**=**sums**;**

//初始化内存中物理地址每一块的数据区

cout**<<**"简单起见，我们对物理地址的每一块用字符串进行简单的初始化，没有具体到每一物理地址"**<<**endl**;**

**for** **(**int bn**=**0**;** bn**<**blocknum**;** bn**++)**

**{**

work**[**bn**].**num**=**bn**;**

cout**<<**"请输入第"**<<**bn**<<**"个内存块里的作业内容"**<<**endl**;**

cin**>>**work**[**bn**].**str**;**

**}**

//初始化快表

cout**<<**"简单起见，我们初始化快表只有一个"**<<**endl**;**

cout**<<**"请输入要作为快表的段号和页号"**<<**endl**;**

cin**>>**qu**.**qb**>>**qu**.**qp**;**

**while(**ss**[**qu**.**qb**].**flag**!=**1**||**page**[**qu**.**qb**][**qu**.**qp**].**flag**!=**1**)**

**{**

cout**<<**"该页不在内存请输入一页在内存中的作为快表,请输入要作为快表的段号和页号"**<<**endl**;**

cin**>>**qu**.**qb**>>**qu**.**qp**;**

**}**

qu**.**qs**=**page**[**qu**.**qb**][**qu**.**qp**].**block**;**

menu**();**

**}**

void change**()**

**{**

cout**<<**"请输入要转化的逻辑地址，段号s，段内页号p，页内偏移地址d(B)"**<<**endl**;**

int snum**,**pnum**,**dnum**;**

cin**>>**snum**>>**pnum**>>**dnum**;**

//首先查快表

**if(**snum**==**qu**.**qb**&&**pnum**==**qu**.**qp**)**

**{**

cout**<<**"快表命中"**<<**"对应块号是"**<<**qu**.**qs**<<**endl**;**

cout**<<**"该块中作业数据是"**<<**work**[**page**[**qu**.**qb**][**qu**.**qp**].**block**].**str**<<**endl**;**

cout**<<**"物理地址是"**<<**qu**.**qs**\***bbs**\***1024**+**dnum**<<**endl**;;**

menu**();**

**}**

//访问段表寄存器

**else**

**{**

cout**<<**"快表没有命中,访问段表寄存器，段号等于段表始址加上偏移地址"**<<**endl**;**

int ssnum**;**

ssnum**=**st**.**ssta**+**snum**;**

**if(**ssnum**>**st**.**slen**-**1**)**

**{**

cout**<<**"越界中断"**<<**endl**;**

menu**();**

**}**

//访问段表

**else**

**{**

**if(**ssnum**>=**0**&&**ssnum**<=**st**.**slen**-**1**)**

**{**

//是否缺段

cout**<<**"段表有效"**<<**endl**;**

**if(**ss**[**ssnum**].**flag**==**0**)**

**{**

cout**<<**"缺段中断"**<<**endl**;**

menu**();**

**}**

**else**

**{**

//查询页表，根据段号查出页表始址，加上段内偏移页数，即得到页表项。 //缺页中断测试

**if(**pnum**>**ss**[**ssnum**].**plen**-**1**)**

**{**

cout**<<**"缺页中断"**<<**endl**;**

menu**();**

**}**

**else**

**{**

**if(**pnum**>=**0**&&**pnum**<=**ss**[**ssnum**].**plen**-**1**)**

**{**

**if(**page**[**ssnum**][**pnum**].**flag**==**0**)**

**{**

cout**<<**"缺页中断"**<<**endl**;**

menu**();**

**}**

**else**

**{**

cout**<<**"找到该页"**<<**"查询页表后对应块号"**<<**page**[**ssnum**][**pnum**].**block**<<**endl**;**

cout**<<**"该块内存的数据是"**<<**work**[**page**[**ssnum**][**pnum**].**block**].**str**<<**endl**;**

cout**<<**"转化得到的物理地址是："**<<**page**[**ssnum**][**pnum**].**block**\***bbs**\***1024**+**dnum**<<**endl**;**

menu**();**

**}**

**}**

**}**

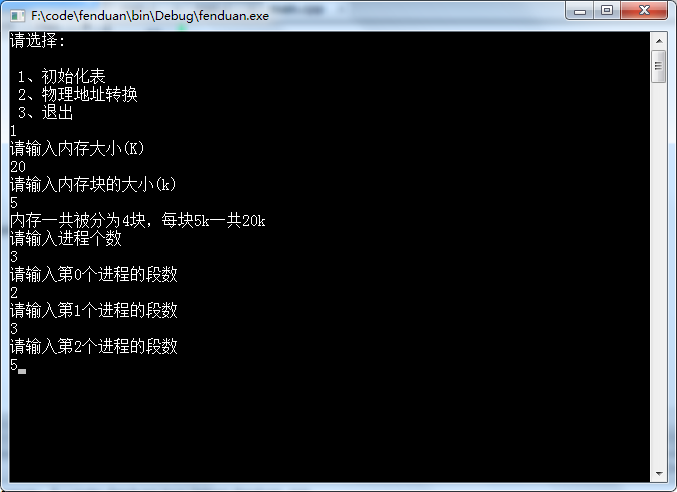
**}**

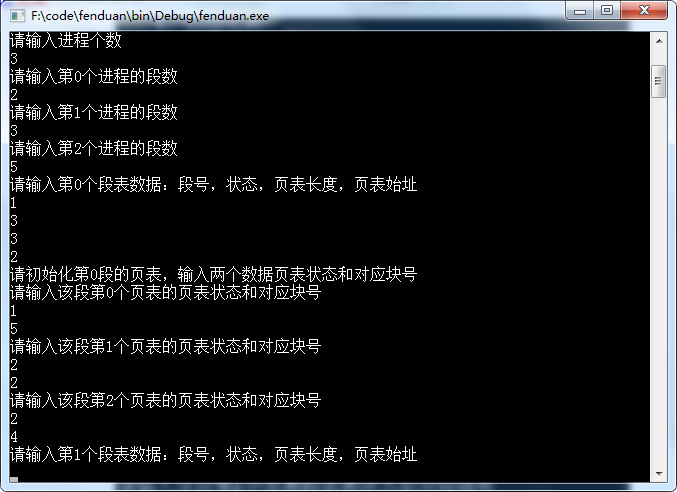
**}**

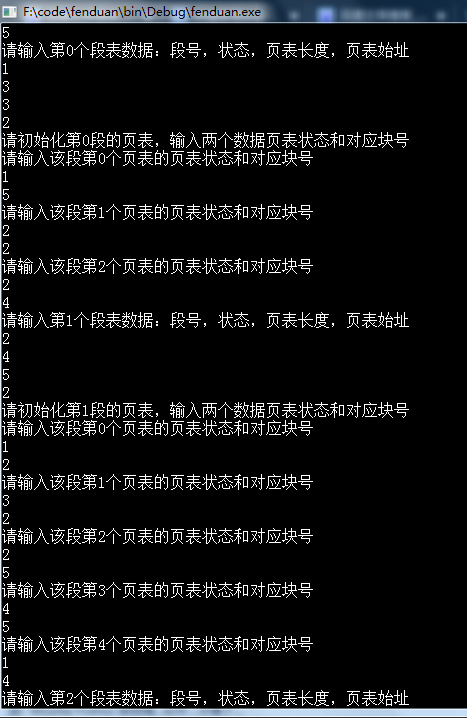
**}**

**}**

**}**

**实验结果：**





**四、实验总结**

本次实习内容存储管理，通过编写和调试存储管理的模拟程序加深了对存储管理方案的理解。熟悉虚存管理的各种页面淘汰算法。对算法有了更深刻的理解，就是动手能力太差。