**操作系统课程设计**

**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。

**二、课程设计内容和要求**

用高级语言编写程序，模拟实现一个简单功能的操作系统。

1. 实现作业调度（先来先服务）、进程调度功能（时间片轮转）
2. 实现内存管理功能（连续分配）
3. 实现文件系统功能（选做内容）
4. 这些功能要有机地连接起来

**三、软、硬件环境**

运行系统：window10

编译工具：Dev C++

**四、设计步骤**

1. **算法简介：**

先来先服务算法：

如果早就绪的进程排在就绪队列的前面，迟就绪的进程排在就绪队列的后面，那么先来先服务(FCFS: first come first service)总是把当前处于就绪队列之首的那个进程调度到运行状态。也就说，它只考虑进程进入就绪队列的先后，而不考虑它的下一个CPU周期的长短及其他因素。FCFS算法简单易行，但性能却不大好。

时间片轮转算法：

时间片轮转调度是一种最古老，最简单，最公平且使用最广的算法。每个进程被分配一个时间段，称作它的时间片，即该进程允许运行的时间。如果在时间片结束时进程还在运行，则CPU将被剥夺并分配给另一个进程。如果进程在时间片结束前阻塞或结束，则CPU当即进行切换。调度程序所要做的就是维护一张就绪进程列表，当进程用完它的时间片后，它被移到队列的末尾。

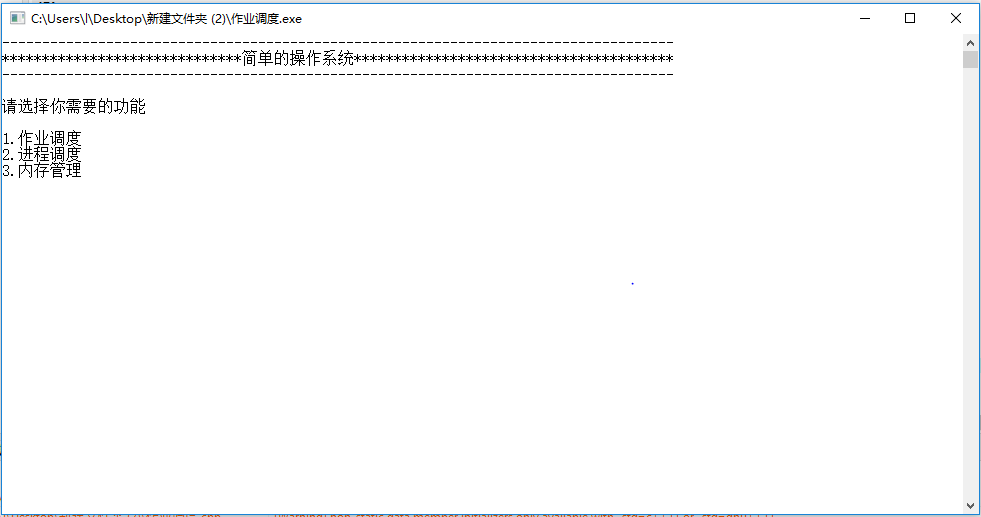
动态分配算法：

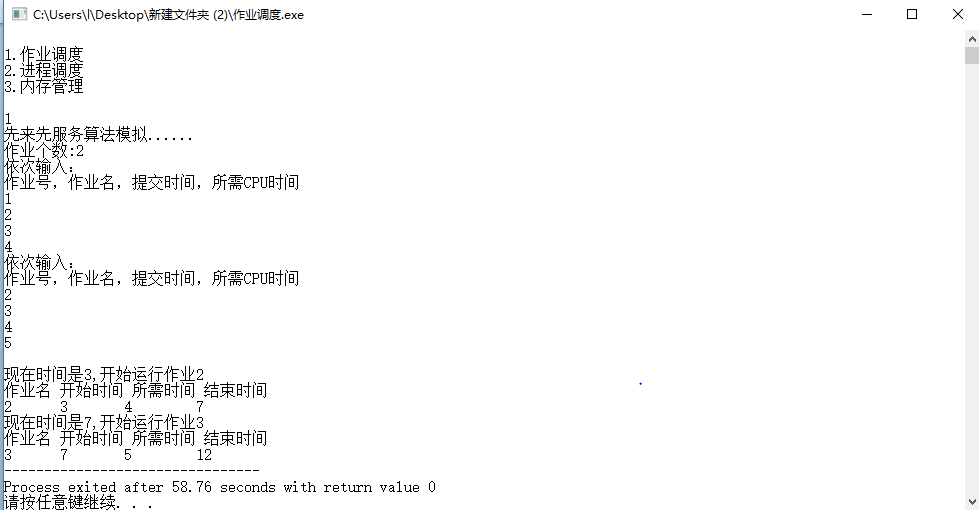
主要算法有首次适应算法、循环首次适应算法、最坏适应算法、最佳是适应算法等，是一种具有较高性能的内存分配算法。

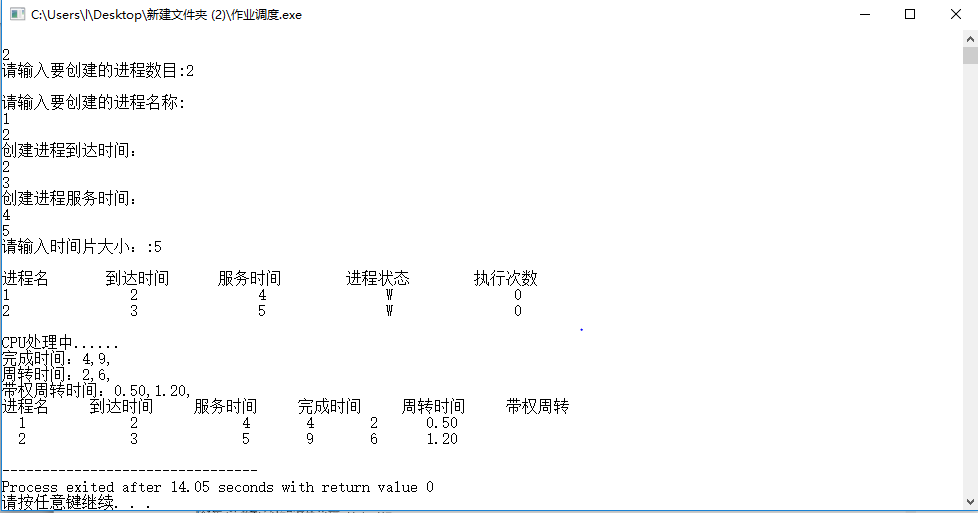
1. **主要方法介绍：**

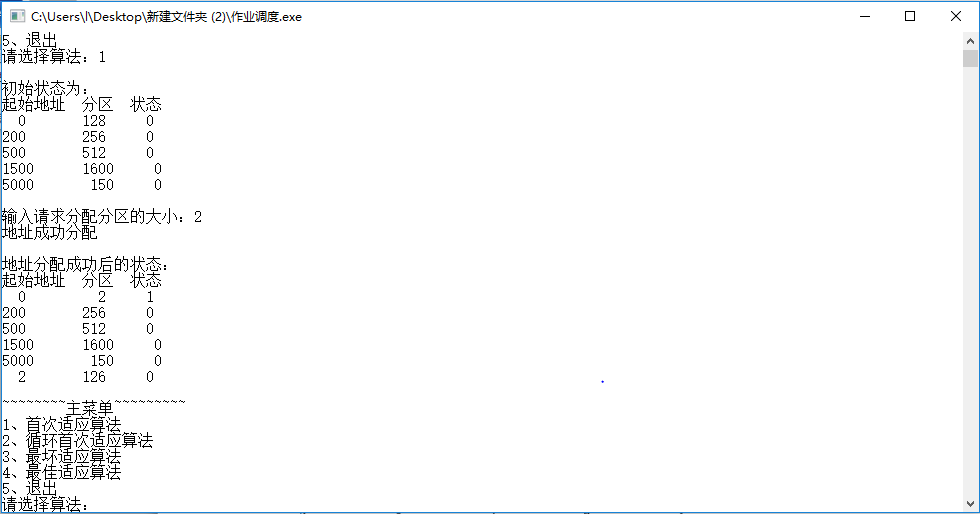
|  |  |
| --- | --- |
| 方法 | 功能 |
| OS() | 操作系统主界面 |
| jobS() | 作业调度算法（先来先服务） |
| courseS() | 进程调度算法（时间片轮转） |
| memory() | 内存管理算法 |

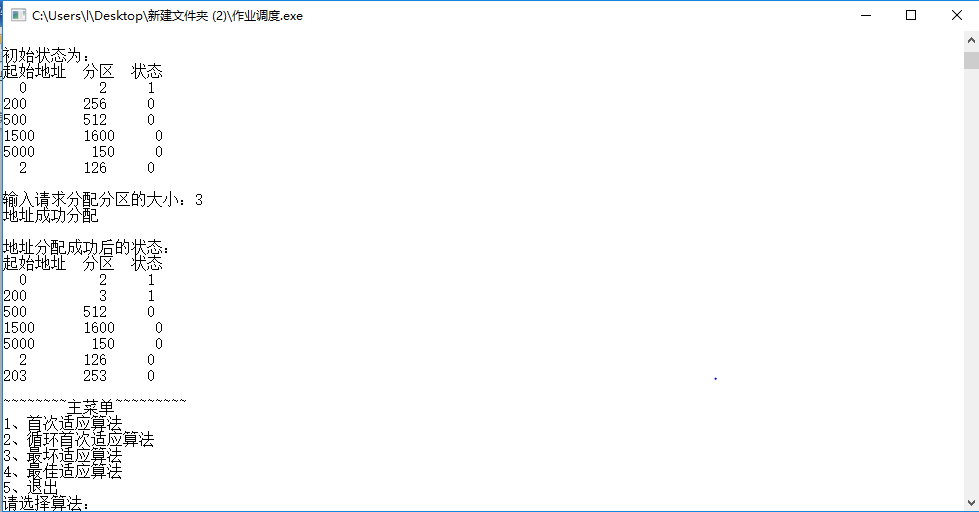
1. **实验结果截图：**

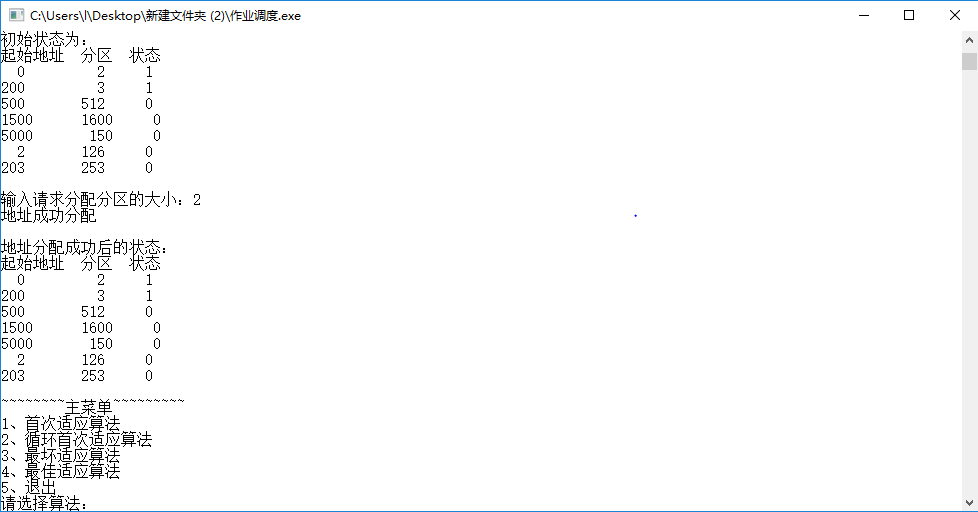


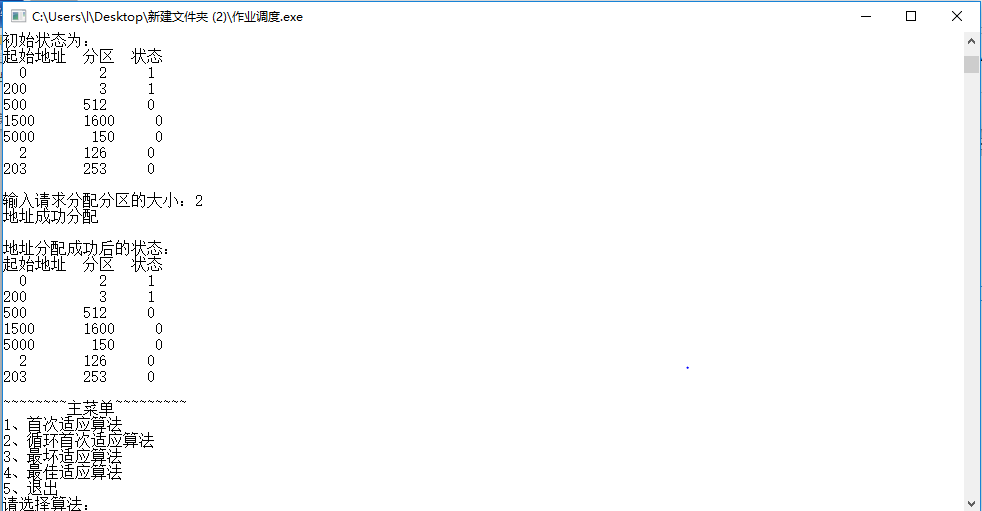












1. **源代码：**

#include <stdio.h>

#include <stdlib.h>

#include <tchar.h>

#include <stdio.h>

#include <string.h>

#include <windows.h>

#define OK 0

#define OVERFLOW 1

#define L 10

/\*

作业调度

\*/

typedef struct JCB { //定义进程控制块

char num[2]; //作业号

char name[10]; //作业名

char state; //运行状态

int tijiaotime; //提交作业时间

int starttime; //作业开始时间

int finishtime; //结束时间

int needtime; //运行需要时间

struct JCB \*next; //指向下个作业

}jcb;

int time=10000,n; //计时器

jcb \*head=NULL,\*p,\*q;

void run\_fcfo(jcb \*p1) {

time = p1->tijiaotime > time? p1->tijiaotime:time;

p1->starttime=time;

printf("\n现在时间是%d,开始运行作业%s\n",time,p1->name);

time+=p1->needtime;

p1->state='F';

p1->finishtime=time;

printf("作业名 开始时间 所需时间 结束时间\n");

printf("%s,%d,%d,%d",p1->name,p1->starttime,p1->needtime,p1->finishtime);

}

void fcfo() {

int i,j,t;

for(j=0;j<n;j++) {

p=head;

t=10000;

for(i=0;i<n;i++) { //找到当前未完成的作业

if(p->tijiaotime<t && p->state=='W') {

t=p->tijiaotime;

q=p; //标记当前未完成的作业

}

p=p->next;

}

run\_fcfo(q);

}

}

void getInfo() { //创建进程

int num;

printf("\n作业个数:");

scanf("%d",&n);

for(num=0;num<n;num++) {

p=(jcb \*)malloc(sizeof(jcb));

if(head==NULL) {head=p;q=p;}

printf("依次输入：\n作业号，作业名，提交时间，所需CPU时间\n");

scanf("%s\t%s\t%d\t%d",&p->num,&p->name,&p->tijiaotime,&p->needtime);

if(p->tijiaotime < time) time=p->tijiaotime;

q->next=p;

p->starttime=0;

p->finishtime=0;

p->next=NULL;

p->state='W';

q=p;

}

}

void jobS() {

printf("先来先服务算法模拟......");

getInfo();

fcfo();

}

/\*

进程调度

\*/

char pro[20] ; //进程

int processNum; //进程数

int timeSlice = 0; //时间片

typedef char QlemTypeChar;

typedef int QlemTypeInt;

typedef int Status;

typedef struct QNode {

QlemTypeChar data;

QlemTypeInt timeArrive = 0;

QlemTypeInt timeService = 0;

QlemTypeInt timeCount = 0;

QlemTypeInt runCount = 0;

QlemTypeInt timeFinal = 0; //完成时间

QlemTypeInt timeRound = 0; //周转时间

float timeRightRound = 0; //带权周转时间

QlemTypeChar proState = 'W'; //进程的状态，W--就绪态，R--执行态，F--完成态

struct QNode \*next; //链表指针

}QNode, \*QueuePtr;

typedef struct {

QueuePtr front; //队头指针

QueuePtr rear; //队尾指针

}LinkQueue;

Status InitQueue(LinkQueue &Q) {

Q.front = Q.rear = (QueuePtr)malloc(sizeof(QNode));

if(!Q.front) exit(OVERFLOW);

Q.front->next = NULL;

return OK;

}

Status EnQueue(LinkQueue &Q, QlemTypeChar e) {

QueuePtr p;

p = (QueuePtr)malloc(sizeof(QNode));

if (!p) exit(OVERFLOW);

p->data = e;

p->next = NULL;

Q.rear->next = p;

Q.rear = p;

return OK;

}

Status DeQueue(LinkQueue &Q, QlemTypeChar &e) {

QueuePtr p;

if (Q.front == Q.rear) return ERROR;

p = Q.front->next;

e = p->data;

Q.front->next = p->next;

if (Q.rear == p) Q.rear = Q.front;

free(p);

return OK;

}

LinkQueue QPro;

QNode qq[10];

void ProGetFirst() { //取出就绪队列队首进程

InitQueue(QPro);

printf("请输入要创建的进程名称:\n");

for (int i = 0; i < processNum-1; i++) {

fflush(stdin);

scanf("%c", &pro[i]);

}

fflush(stdin);

for (int i = 0; i<processNum-1; i++) {

qq[i].data = pro[i];

EnQueue(QPro, qq[i].data);

}

}

void scanfData() {

printf("请输入要创建的进程数目:");

scanf("%d", &processNum);

processNum++;

fflush(stdin);

printf("\n");

ProGetFirst();

printf("创建进程到达时间：\n");

int time\_Arr[10];

for (int i = 0; i < processNum-1; i++) {

scanf("%d", &time\_Arr[i]);

}

for (int i =0; i<processNum-1; i++) {

qq[i].timeArrive = time\_Arr[i];

EnQueue(QPro, qq[i].timeArrive);

}

printf("创建进程服务时间：\n");

int time\_Ser[10];

for (int i = 0; i < processNum-1; i++) {

scanf("%d", &time\_Ser[i]);

}

for (int i = 0; i<processNum-1; i++) {

qq[i].timeService = time\_Ser[i];

EnQueue(QPro, qq[i].timeService);

}

printf("请输入时间片大小：:");

scanf("%d", &timeSlice);

printf("\n");

}

void ProOutPut1(){ //获取进程信息

printf("进程名\t 到达时间\t 服务时间\t 进程状态\t 执行次数\n");

for (int i = 0; i < processNum - 1; i++){

printf("%c\t\t%d\t\t%d\t\t%c\t\t%d\n", qq[i].data, qq[i].timeArrive, qq[i].timeService, qq[i].proState, qq[i].runCount);

}

}

void CalculatetimeFinal() { //计算完成时间

int timecou=0;

int countTemp = 0;

QlemTypeChar ee;

for (int i = 0; i < processNum - 1; i++) {

countTemp += qq[i].timeService;

}

while (timecou < countTemp) {

for (int i = 0; i < processNum - 1; i++) {

if (qq[i].timeFinal == 0) {

if (qq[i].timeService - qq[i].timeCount >= timeSlice) {

timecou += timeSlice;

}

else {

timecou += (qq[i].timeService - qq[i].timeCount);

//DeQueue(QPro, ee);

}

if (timeSlice < qq[i].timeService) { //时间片大小< 服务时间

int timetemp = timeSlice > qq[i].timeService ? qq[i].timeService : timeSlice;

if ((qq[i].timeCount + timetemp) < qq[i].timeService) {

if (qq[i].timeService - qq[i].timeCount >= timeSlice) {

qq[i].timeCount += timeSlice;

}

else {

qq[i].timeCount += (qq[i].timeService - qq[i].timeCount);

}

}

else {

if (qq[i].timeFinal == 0) {

qq[i].timeFinal = timecou;

}

}

}

else { //时间片大小>= 服务时间

qq[i].timeFinal = timecou; //该进程的完成时间=count

}

}

}

}

for (int i = 0; i < processNum - 1; ++i) {

qq[i].timeRound = qq[i].timeFinal - qq[i].timeArrive;

qq[i].timeRightRound = (float)qq[i].timeRound / qq[i].timeService;

}

}

void ProOutPut2() { //获取进程处理后的信息

printf("进程名\t 到达时间 服务时间 完成时间 周转时间 带权周转\n");

for (int i = 0; i < processNum - 1; i++) {

printf("%c\t\t%d\t%d\t%d\t%d\t%.2f\n",

qq[i].data, qq[i].timeArrive, qq[i].timeService, qq[i].timeFinal, qq[i].timeRound, qq[i].timeRightRound);

}

}

int courseS() {

scanfData();

ProOutPut1();

CalculatetimeFinal();

printf("\n");

printf("CPU处理中......\n");

printf("完成时间：");

for (int i = 0; i < processNum - 1; i++) {

printf("%d,", qq[i].timeFinal);

}

printf("\n");

printf("周转时间：");

for (int i = 0; i < processNum - 1; i++) {

printf("%d,",qq[i].timeRound);

}

printf("\n");

printf("带权周转时间：");

for (int i = 0; i < processNum - 1; i++) {

printf("%.2f,", qq[i].timeRightRound);

}

printf("\n");

ProOutPut2();

return 0;

}

/\*

内存管理

\*/

typedef struct LNode {

int startaddress;

int size;

int state;

}LNode;

LNode

P[L]={{0,128,0},{200,256,0},{500,512,0},{1500,1600,0},{5000,150,0}};

int N=5; int f=0;

void print() {

int i;

printf("起始地址 分区 状态\n");

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

printf("%3d %8d %4d\n",P[i].startaddress,P[i].size,P[i].state);

}

void First() {

int i,l=0,m;

printf("\n输入请求分配分区的大小：");

scanf("%d",&m);

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

if(P[i].size<m)

continue;

else if(P[i].size==m) {

P[i].state=1;

l=1;

break;

}

else {

P[N].startaddress=P[i].startaddress+m;

P[N].size=P[i].size-m;

P[i].size=m;P[i].state=1;

l=1;N++;

break;

}

}

if(l==1||i<N) {

printf("地址成功分配\n\n");

printf("地址分配成功后的状态：\n");

print();

}

else

printf("没有可以分配的地址空间\n");

}

void CirFirst() {

int l=0,m,t=0;

printf("\n输入请求分配分区的大小：");

scanf("%d",&m);

while(f<N) {

if(P[f].size<m) {

f=f+1;

if(f%N==0) {

f=0;t=1;

}

continue;

}

if(P[f].size==m && P[f].state!=1) {

P[f].state=1;

l=1;f++;

break;

}

if(P[f].size>m && P[f].state!=1) {

P[N].startaddress=P[f].startaddress+m;

P[N].size=P[f].size-m;

P[f].size=m;

P[f].state=1;

l=1;N++;f++;

break;

}

}

if(l==1) {

printf("地址成功分配\n\n");

printf("地址分配成功后的状态：\n");print();

}

else printf("没有可以分配的地址空间\n");

}

void Worst() {

int i,t=0,l=0,m;

int a[L];

printf("\n输入请求分配分区的大小：");

scanf("%d",&m);

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

a[i]=0;

if(P[i].size<m)

continue;

else if(P[i].size==m) {

P[i].state=1;

l=1;break;

}

else a[i]=P[i].size-m;

}

if(l==0) {

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

if(a[i]!=0)

t=i;

}

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

if(a[i]!=0 && a[i]>a[t])

t=i;

}

P[N].startaddress=P[t].startaddress+m;

P[N].size=P[t].size-m;

P[t].size=m;P[t].state=1;

l=1;N++;

}

if(l==1||i<N) {

printf("地址成功分配\n\n");

printf("地址分配成功后的状态：\n");

print();

}

else printf("没有可以分配的地址空间\n");

}

void Best() {

int i,t=0,l=0,m;

int a[L];

printf("\n输入请求分配分区的大小：");

scanf("%d",&m);

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

a[i]=0;

if(P[i].size<m)

continue;

else if(P[i].size==m) {

P[i].state=1;

l=1;

break;

}

else a[i]=P[i].size-m;

}

if(l==0) {

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

if(a[i]!=0)

t=i;

}

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

if(a[i]!=0 && a[i]<a[t])

t=i;

}

P[N].startaddress=P[t].startaddress+m;

P[N].size=P[t].size-m;

P[t].size=m;P[t].state=1;

l=1;N++;

}

if(l==1||i<N) {

printf("地址成功分配\n\n");

printf("地址分配成功后的状态：\n");

print();

}

else printf("没有可以分配的地址空间\n");

}

void pr() {

int k=0;

printf("动态分区分配算法：");

while(k!=5) {

printf("\n~~~~~~~~主菜单~~~~~~~~~");

printf("\n1、首次适应算法\n2、循环首次适应算法");

printf("\n3、最坏适应算法\n4、最佳适应算法");

printf("\n5、退出\n");

printf("请选择算法：");

scanf("%d",&k);

switch(k) {

case 1:

printf("\n初始状态为：\n");

print();

First();

continue;

case 2:

printf("\n初始状态为：\n");

print();

CirFirst();

continue;

case 3:

printf("\n初始状态为：\n");

print();

Worst();

continue;

case 4:

printf("\n初始状态为：\n");

print();

Best();

continue;

case 5:

break;

default:printf("选择错误，请重新选择。\n");

}

}

}

int memoryM() {

pr();

return 0;

}

void OS() {

printf("------------------------------------------------------------------------------------\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*简单的操作系统\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("------------------------------------------------------------------------------------\n");

printf("\n");

printf("请选择你需要的功能\n");

printf("\n");

printf("1.作业调度\n");

printf("2.进程调度\n");

printf("3.内存管理\n");

printf("\n");

scanf("%d",&n);

switch(n) {

case 1:

jobS();

break;

case 2:

courseS();

break;

case 3:

memoryM();

break;

default:

printf("请输入有效数字\n");

}

}

int main() {

OS();

}