操作系统原理课程设计

题目：主存空间的分配与回收

学 院： 计算机科学技术学院

班级学号： 软件19-3班26号

姓 名： 张思文

指导教师： 曹梅

同组成员： 李晏玮、王维卓

2021 年 12 月 9 日

**操作系统原理课程设计任务书**

**一、题目：**主存空间的分配与回收

**二、设计要求**

（1）同组分工：张思文负责设计与实现分页管理，使用位示图实现主存的分配与回收；李晏玮负责设计与实现可变式分区管理，使用空闲区链实现主存的分配与回收；王维卓负责设计与实现可变式分区管理，分别使用首次或最佳适应算法实现主存的分配与回收。

（2）查阅相关资料，自学具体课题中涉及到的新知识。

（3）采用结构化、模块化程序设计方法，功能要完善，具有一定的创新。

（4）所设计的程序应有输入、输出。

（5）按要求写出课程设计报告，于课程设计结束后2天内提交。其主要内容包括：封皮、课程设计任务书，指导教师评语与成绩、目录、概述、软件总体设计、详细设计、软件的调试、总结、致谢、附录(带中文注释的程序清单)、参考文献。**总体设计**应配合软件总体模块结构图来说明软件应具有的功能；**详细设计**应用传统或N-S流程图和屏幕抓图说明；**调试**的叙述应配合出错场景的抓图来说明出现了哪些错误，如何解决的。

**三、课程设计工作量**

一般每人的程序量在200行有效程序行左右，不得抄袭。

**四、课程设计工作计划**

2021年12月6日 指导教师讲解布置题目，学生根据题目准备资料；

2021年12月7日 进行总体方案设计；

2021年12月8日～12月12日 完成程序模块并通过独立编译；

2021年12月13日～12月15日 将各模块集成为完整系统，调试运行

2021年12月16日～12月19日 验收、撰写提交课程设计报告。

指导教师签章：

操作系统原理课程设计指导教师评语与成绩

|  |
| --- |
| 指导教师评语：  课程设计表现成绩：  课程设计验收成绩：  课程设计报告成绩：  课程设计 总成绩：    指导教师签章  2021年12月20 日 |

**目 录**

[一 概述 1](#_Toc90054844)

[二 需求分析 2](#_Toc90054845)

[三 概要设计 3](#_Toc90054846)

[四 详细设计 5](#_Toc90054847)

[五 调试分析、测试结果 10](#_Toc90054848)

[六 总结 12](#_Toc90054849)

[七 参考文献 13](#_Toc90054850)

[八 附录 14](#_Toc90054851)

一 概述

一个好的计算机系统不仅要有一个足够容量的、存取速度高的、稳定可靠的主存储器，而且要能合理地分配和使用这些存储空间。当用户提出申请存储器空间时，存储管理必须根据申请者的要求，按一定的策略分析主存空间的使用情况，找出足够的空闲区域分配给申请者。当作业撤离或主动归还主存资源时，则存储管理要收回作业占用的主存空间或归还部分主存空间。主存的分配和回收的实现虽与主存储器的管理方式有关的。

在这里我们使用这三种方法实现主存空间的分配与回收；

1.采用可变式分区管理，分别使用首次或最佳适应算法实现主存的分配与回收；

2.采用可变式分区管理，使用空闲区链实现主存的分配与回收；

3.采用分页管理，使用位示图实现主存的分配与回收。

二 需求分析

本次设计的目的主要是熟悉和掌握主存的分配与回收。同时在不同的存储管理方式下，如何实现主存空间的分配与回收，以及动态分区分配方式中的数据结构和分配算法及动态分区存储管理方式及其实现过程。

在分页式管理方式下采用位示图来表示主存分配情况，实现主存空间的分配和回收

(1) 假定系统的主存被分成大小相等的64个块，用0/1对应空闲/占用。

(2) 当要装入一个作业时，根据作业对主存的需求量，先查空闲块数是否能满足作业要求，若能满足，则查位示图，修改位示图和空闲块数。位置与块号的对应关系为：

块号 = j \* 8 + i，其中 i 表示位，j 表示字节。

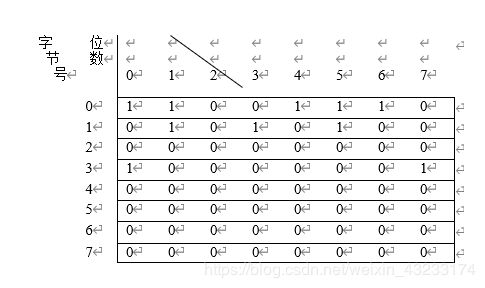
根据分配的块号建立页表。页表包括两项：页号和块号。

(3) 回收时，修改位示图和空闲块数。

要求能接受来自键盘的空间申请及释放请求，能显示位示图和空闲块数的变化，能显示进程的页表。

三 概要设计

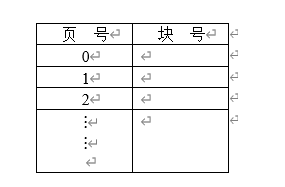
1. 分页式存储器把主存分成大小相等的若干块，作业的信息也按块的大小分页，作业装入主存时可把作业的信息按页分散存放在主存的空闲块中，为了说明主存中哪些块已经被占用，哪些块是尚未分配的空闲块，可用一张位示图来指出。位示图可由若干存储单元来构成，其中每一位与一个物理块对应，用0/1表示对应块为空闲/已占用。
2. 假设某系统的主存被分成大小相等的64块，则位示图可用8个字节来构成，另用一单元记录当前空闲块数。如果已有第0，1，4，5，6，9，11，13，24，31，共10个主存块被占用了，那么位示图情况如下：



1. 当要装入一个作业时，根据作业对主存的需要量，先查当前空闲块数是否能满足作业要求，若不能满足则输出分配不成功。若能满足，则查位示图，找出为“0”的一些位，置上占用标志“1”，从“当前空闲块数”中减去本次占用块数。  
   按找到的计算出对应的块号，其计算公式为：

块号= j\*8+i

其中，j表示找到的是第n个字节，I表示对应的是第n位。  
根据分配给作业的块号，为作业建立一张页表，页表格式：



1. 当一个作业执行结束，归还主存时，根据该作业的页表可以知道应归还的块号，由块号可计算出在位示图中的对应位置，把对应位的占用标志清成“0”，表示对应的块已成为空闲块。归还的块数加入到当前空闲块数中。由块号计算在位示图中的位置的公式如下：

字节号 j=块号/8  
位数 i=块号%8

1. 设计实现主存分配和回收的程序，要求能显示和打印分配或回收前后的位示图和当前空闲块数，对完成一次分配后还要显示或打印为作业建立的页表。

四 详细设计

1. 设计思路  
   （1）假定系统的主存被分成大小相等的64个块，用0/1对应空闲/占用。  
   （2）当要装入一个作业时，根据作业对主存的需求量，先查空闲块数是否能满足作业要求，若能满足，则查位示图，修改位示图和空闲块数。位置与块号的对应关系为：  
   块号=j\*8+i，其中i表示位，j表示字节。  
   根据分配的块号建立页表。页表包括两项：页号和块号。  
   （3）回收时，修改位示图和空闲块数。
2. 主要数据结构

truct BitMap //用于存储位示图

{

int map[8][8]; //8\*8=64，位示图

int free; //剩余的空闲块数

}bitmap;

typedef struct process //用于存储作业

{

int num; //作业序号

int size; //作业大小

int \*pagetable; //页表

struct process \*next; //下一作业

struct process \*pre; //上一作业

}process;

1. 算法设计

创建作业储存函数与作业回收函数以及展示位示图和页表函数

void SetProcess(process \*head) //创立作业，生成页表

{

int setnum;

int i=0,j=0,k=0,finish=0;

process \*temp = new process();

cout<<endl<<"请输入作业序号(数字且不为0)：";

cin>>setnum;

if(setnum == 0)

{

cout<<endl<<"错误！不能创建0号作业，返回。";

delete temp;

return;

}

temp->num = setnum;

cout<<endl<<"请输入作业大小(<=64)：";

cin>>temp->size;

if(temp->size > bitmap.free) //判断现有空间是否足够分配

{

cout<<endl<<"当前内存没有足够的空间分配，返回。";

delete temp;

return;

}

temp->next = head->next; //头插法插入新作业

if(head->next != NULL)

head->next->pre = temp;

temp->pre = head;

head->next = temp;

temp->pagetable = new int [temp->size]; //创立页表数组

for(i=0; i<8 && finish == 0; i++)

{

for(j=0; j<8 && finish == 0; j++)

{

if(bitmap.map[i][j] == 0) //修改页表数据

{

bitmap.map[i][j] = 1;

bitmap.free --;

temp->pagetable[k] = 8\*i + j;

k++;

}

if(k == temp->size) //用于结束循环

finish = 1;

}

}

cout<<endl<<"已装入作业，页表如下："<<endl<<"页号\t"; //输出页表

k = temp->size;

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

{

cout<<i+1<<'\t';

}

cout<<endl<<"块号\t";

for(j=0; j<k; j++)

{

cout<<temp->pagetable[j]<<'\t';

}

}

void RecProcess(process \*head) //回收作业，退还内存

{

int recnum;

int i=0,j=0,k=0,l=0,found=0;

process \*temp = new process();

cout<<endl<<"请输入回收作业序号(数字且不为0)：";

cin>>recnum;

if(recnum == 0)

{

cout<<endl<<"错误！不能删除0号作业，返回。";

return;

}

for(temp=head; temp != NULL; temp = temp->next) //找到进程

{

if(temp->num == recnum)

{

found = 1;

k = temp->size;

for(l=0; l<k; l++) //删除数据

{

i = temp->pagetable[l] / 8;

j = temp->pagetable[l] % 8;

bitmap.map[i][j] = 0;

bitmap.free ++;

}

if(temp->pre != NULL) //不允许删除头结点

{

if(temp->next != NULL) //不是尾结点

{

temp->pre->next = temp->next;

temp->next->pre = temp->pre;

delete temp;

}

else //是尾结点

{

temp->pre->next = NULL;

delete temp;

}

}

cout<<endl<<"已找到并回收序号为"<<recnum<<"的作业。";

cout<<endl<<"该作业大小为"<<k<<"。"<<endl;

continue;

}

}

if(found == 0) //未找到进程

{

cout<<endl<<"未找到该作业，返回。";

return;

}

cout<<endl<<"完成回收，返回。";

return;

}

void ShowBitMap() //展示位示图

{

cout<<endl<<"当前位示图状态如下："<<endl;

cout<<'\t'<<"0\t"<<"1\t"<<"2\t"<<"3\t"<<"4\t"<<"5\t"<<"6\t"<<"7\t";

cout<<endl<<"\*\*\*\*";

int i=0,j=0;

for(i=0; i<8; i++)

{

cout<<endl<<i<<'\t';

for(j=0; j<8; j++)

{

cout<<bitmap.map[i][j]<<'\t';

}

}

cout<<endl<<endl<<"完成位示图展示，返回。";

}

void ShowPageTable(process \*head) //展示页表

{

int shownum;

int i=0,j=0,k=0,l=0,found=0;

process \*temp = new process();

cout<<endl<<"请输入查看作业序号(数字且不为0)：";

cin>>shownum;

if(shownum == 0)

{

cout<<endl<<"错误！不能查看0号作业，返回。";

return;

}

for(temp=head; temp != NULL; temp = temp->next) //找到进程

{

if(temp->num == shownum)

{

found = 1;

cout<<endl<<"序号为"<<shownum<<"的作业页表如下："<<endl<<"页号\t"; //输出页表

k = temp->size;

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

{

cout<<i+1<<'\t';

}

cout<<endl<<"块号\t";

for(j=0; j<k; j++)

{

cout<<temp->pagetable[j]<<'\t';

}

cout<<endl<<"已找到并展示序号为"<<shownum<<"的作业页表。";

cout<<endl<<"该作业大小为"<<k<<"。"<<endl;

continue;

}

}

if(found == 0) //未找到进程

{

cout<<endl<<"未找到该作业，返回。";

return;

}

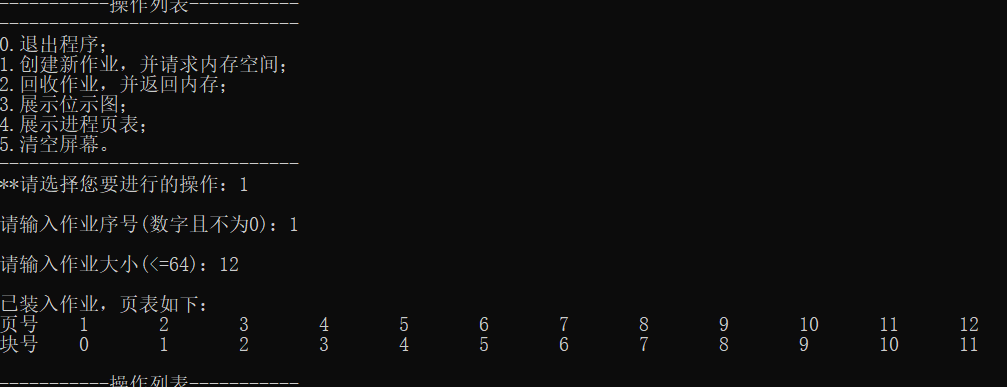
cout<<endl<<"完成作业页表展示，返回。";

}

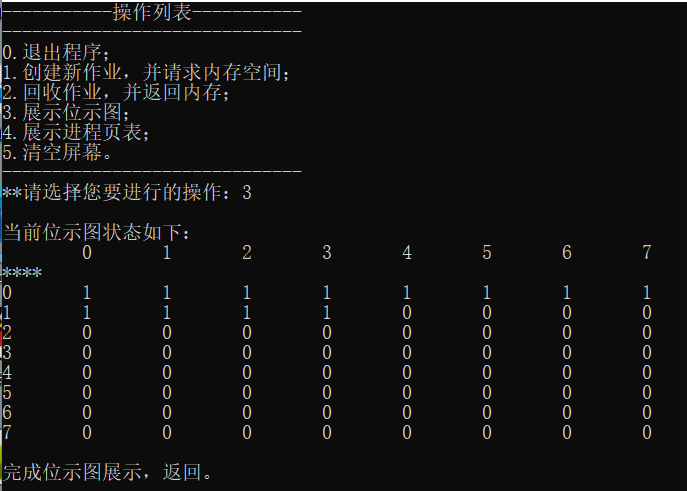
五 调试分析、测试结果

程序可以实现用户自定义输入操作序号，并根据操作序号，进行相关操作

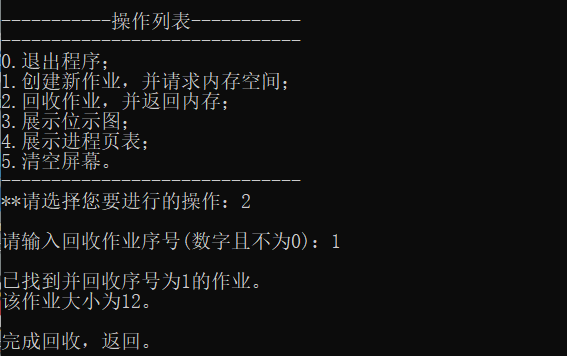
1.创建新作业



2.展示位示图



3回收作业



六 总结

1.我使用了一个二维数组作为位示图的数据结构，这样简单明了。而进程页表使用一个一维数组，用数组下标即可表示页号，节省了存储空间。  
2.在程序中充分考虑到出错处理。比如在功能选择菜单中，若用户输入非法字符则提示“输入错误！请重新输入”；在用户申请内存时，若要求的空间大小大于空闲空间大小，则提示“空闲空间不足”；在回收内存空间和显示进程页表时，若用户输入的进程名不存在，则提示“没有该进程”。这样提高了程序的健壮性。

3.通过本实验，我进一步加深了对使用位示图方式表示和实现主存分配回收的理解，同时也得到了将系统理论实践的机会，熟练了编程能力。

七 参考文献

[1]孟静.《操作系统原理教程》. 北京: 清华大学出版社,2000

[2]周苏、金海溶.《操作系统原理实验》. 北京: 科学出版社,2000

[3]谭浩强. C程序设计（第四版）. 北京：清华大学出版社，2010.6

[4]张尧学. 计算机操作系统教程习题解答与实验指导.北京:清华大学出版社，2000

八 附录

#include<stdio.h>

#include<string.h>

#include<malloc.h>

#include<conio.h>

#include<stdlib.h>

#include <iostream>

using namespace std;

#define NULL 0

#define LEN sizeof(struct PCB)

struct PCB{

int InitAdd;

int EndAdd;

int Length;

int Release;

char name[5];

char state[5];

struct PCB \*next;

};

struct PCB a,b,c,d,e,f,\*Head;

//初始化

PCB\* init(PCB \*Head){

struct PCB \*p;

a.InitAdd=0;a.Length=10;a.EndAdd=10;strcpy(a.name,"系统");strcpy(a.state,"空表目");

b.InitAdd=10;b.Length=10;b.EndAdd=20;strcpy(b.name,"作业");strcpy(b.state,"已分配");

c.InitAdd=20;c.Length=25;c.EndAdd=45;strcpy(c.name,"作业");strcpy(c.state,"已分配");

d.InitAdd=45;d.Length=20;d.EndAdd=65;strcpy(d.name,"空闲");strcpy(d.state,"未分配");

e.InitAdd=65;e.Length=45;e.EndAdd=110;strcpy(e.name,"作业");strcpy(e.state,"已分配");

f.InitAdd=110;f.Length=146;f.EndAdd=256;strcpy(f.name,"空闲");strcpy(f.state,"未分配");

Head=&a;

a.next=&b;

b.next=&c;

c.next=&d;

d.next=&e;

e.next=&f;

f.next=NULL;

printf("起始地址 长度 终止地址 name 状态 \n");

p=Head;

while(p!=NULL){

printf("%4d%10d%8d%12s%11s\n",p->InitAdd,p->Length,p->EndAdd,p->name,p->state);

p=p->next;

}

return Head;

}

//空闲区分配

void compare(struct PCB \*Head,int l){

struct PCB \*p,\*s;

int k=0;

p=Head;

while(p!=NULL){

if(p->Length>=l&&strcmp(p->name,"空闲")==0){

k=1;

printf("作业分配成功！\n");

if(p->Length==l){

strcpy(p->name,"作业");

strcpy(p->state,"已分配");

break;

}

else{

s=(struct PCB \*)malloc(LEN);

s->next=p->next;

p->next=s;

s->EndAdd=p->EndAdd;

strcpy(s->name,"空闲");

strcpy(s->state,"未分配");

s->InitAdd=p->InitAdd+l;

s->Length=p->Length-l;

p->Length=l;

p->EndAdd=p->InitAdd+l;

strcpy(p->name,"作业");

strcpy(p->state,"已分配");

break;

}

}

p=p->next;

}

if(k==0)

printf("作业等待！\n");

printf("起始地址 长度 终止地址 name 状态 \n");

p=Head;

while(p!=NULL){

printf("%4d%10d%8d%12s%11s\n",p->InitAdd,p->Length,p->EndAdd,p->name,p->state);

p=p->next;

}

}

//作业输入

void input(struct PCB \*Head){

int N,l;

printf("请输入申请作业个数：\n");

scanf("%d",&N);

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

printf("输入第%d个作业长度：\n",i+1);

scanf("%d",&l);

compare(Head,l);

}

}

//作业释放

void mix(PCB \*Head,int temp){

PCB \*p,\*q;

p=Head;

while(p->next!=NULL){

if(strcmp(p->name,"空闲")==0&&strcmp(p->next->name,"空闲")==0){

p->Length=p->Length+p->next->Length;

p->EndAdd=p->InitAdd+p->Length;

p->next=p->next->next;

continue;

}

p=p->next;

}

printf("起始地址 长度 终止地址 name 状态 \n");

p=Head;

while(p!=NULL){

printf("%4d%10d%8d%12s%11s\n",p->InitAdd,p->Length,p->EndAdd,p->name,p->state);

p=p->next;

}

}

//作业释放顺序

void release(PCB \*Head){

PCB \*p;

int sum=0,temp=1;

p=Head;

while(p!=NULL){

if(strcmp(p->name,"作业")==0)

sum++;

p=p->next;

}

printf("作业个数为：%d\n",sum);

printf("输入作业释放顺序：\n");

p=Head;

while(p!=NULL){

if(strcmp(p->name,"作业")==0)

scanf("%d",&p->Release);

p=p->next;

}

while(temp<=sum){

p=Head;

while(p!=NULL){

if(p->Release==temp){

strcpy(p->name,"空闲");

strcpy(p->state,"未分配");

mix(Head,temp);

break;

}

p=p->next;

}

temp++;

}

}

//主函数

int linkPCB(){

PCB \*p;

Head=init(Head);

input(Head);

release(Head);

}

//空闲区链实现主存的分配与回收

#define SIZE 1

#define ERROR 0 //出错

typedef int Status;

typedef struct free\_table//定义一个空闲区说明表结构

{

int num; //分区序号

long begin; //起始地址

long size; //分区大小

int status; //分区状态

}ElemType;

typedef struct Node// 线性表的双向链表存储结构

{

ElemType data;

struct Node \*prior; //前趋指针

struct Node \*next; //后继指针

}Node,\*LinkList;

LinkList first; //头结点

LinkList end; //尾结点

int flag;//记录要删除的分区序号

Status Initblock()//开创带头结点的内存空间链表

{

first=(LinkList)malloc(sizeof(Node));

end=(LinkList)malloc(sizeof(Node));

first->prior=NULL;

first->next=end;

end->prior=first;

end->next=NULL;

end->data.num=1;

end->data.begin=40;

end->data.size=600;

end->data.status=0;

return SIZE;

}

//菜单

void menu()

{

printf("\n |\*\*\*\*\*\*\*\*\*\*\*\*\*内存分配和回收\*\*\*\*\*\*\*\*\*\*\*|\n");

printf(" |======================================|\n");

printf(" | 0.退出 |\n");

printf(" | 1.首次适应算法 |\n");

printf(" | 2.最佳适应算法 |\n");

printf(" |======================================|\n");

}

void sort()//分区序号重新排序

{

Node \*p=first->next,\*q;

q=p->next;

for(;p!=NULL;p=p->next)

{

for(q=p->next;q;q=q->next)

{

if(p->data.num>=q->data.num)

{

q->data.num+=1;

}

}

}

}

//显示主存分配情况

void show()

{

int flag=0;//用来记录分区序号

Node \*p=first;

p->data.num=0;

p->data.begin=0;

p->data.size=40;

p->data.status=1;

sort();

printf("\n\t\t》主存空间分配情况《\n");

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

printf("分区序号\t起始地址\t分区大小\t分区状态\n\n");

while(p)

{

printf("%d\t\t%d\t\t%d",p->data.num,p->data.begin,p->data.size);

if(p->data.status==0) printf("\t\t空闲\n\n");

else printf("\t\t已分配\n\n");

p=p->next;

}

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

}

//首次适应算法

Status First\_fit(int request)

{

//为申请作业开辟新空间且初始化

Node \*p=first->next;

LinkList temp=(LinkList)malloc(sizeof(Node));

temp->data.size=request;

temp->data.status=1;

p->data.num=1;

while(p)

{

if((p->data.status==0)&&(p->data.size==request))

{

//有大小恰好合适的空闲块

p->data.status=1;

return SIZE;

break;

}

else if((p->data.status==0) && (p->data.size>request))

{

//有空闲块能满足需求且有剩余

temp->prior=p->prior;

temp->next=p;

temp->data.begin=p->data.begin;

temp->data.num=p->data.num;

p->prior->next=temp;

p->prior=temp;

p->data.begin=temp->data.begin+temp->data.size;

p->data.size-=request;

p->data.num+=1;

return SIZE;

break;

}

p=p->next;

}

return ERROR;

}

Status Best\_fit(int request)

{

int ch; //记录最小剩余空间

Node \*p=first;

Node \*q=NULL; //记录最佳插入位置

LinkList temp=(LinkList)malloc(sizeof(Node));

temp->data.size=request;

temp->data.status=1;

p->data.num=1;

while(p) //初始化最小空间和最佳位置

{

if((p->data.status==0) && (p->data.size>=request) )

{

if(q==NULL)

{

q=p;

ch=p->data.size-request;

}

else if(q->data.size > p->data.size)

{

q=p;

ch=p->data.size-request;

}

}

p=p->next;

}

if(q==NULL) return ERROR;//没有找到空闲块

else if(q->data.size==request)

{

q->data.status=1;

return SIZE;

}

else

{

temp->prior=q->prior;

temp->next=q;

temp->data.begin=q->data.begin;

temp->data.num=q->data.num;

q->prior->next=temp;

q->prior=temp;

q->data.begin+=request;

q->data.size=ch;

q->data.num+=1;

return SIZE;

}

return SIZE;

}

//分配主存

Status allocation(int a)

{

int request;//申请内存大小

printf("请输入申请分配的主存大小(单位:KB):");

scanf("%d",&request);

if(request<0 ||request==0)

{

printf("分配大小不合适，请重试！");

return ERROR;

}

switch(a)

{

case 1: //默认首次适应算法

if(First\_fit(request)==SIZE) printf("\t\*\*\*\*分配成功！\*\*\*\*");

else printf("\t\*\*\*\*内存不足，分配失败！\*\*\*\*");

return SIZE;

break;

case 2: //选择最佳适应算法

if(Best\_fit(request)==SIZE) printf("\t\*\*\*\*分配成功！\*\*\*\*");

else printf("\t\*\*\*\*内存不足，分配失败！\*\*\*\*");

return SIZE;

break;

}

}

Status deal1(Node \*p)//处理回收空间

{

Node \*q=first;

for(;q!=NULL;q=q->next)

{

if(q==p)

{

if(q->prior->data.status==0&&q->next->data.status!=0)

{

q->prior->data.size+=q->data.size;

q->prior->next=q->next;

q->next->prior=q->prior;

q=q->prior;

q->data.status=0;

q->data.num=flag-1;

}

if(q->prior->data.status!=0&&q->next->data.status==0)

{

q->data.size+=q->next->data.size;

q->next=q->next->next;

q->next->next->prior=q;

q->data.status=0;

q->data.num=flag;

}

if(q->prior->data.status==0&&q->next->data.status==0)

{

q->prior->data.size+=q->data.size;

q->prior->next=q->next;

q->next->prior=q->prior;

q=q->prior;

q->data.status=0;

q->data.num=flag-1;

}

if(q->prior->data.status!=0&&q->next->data.status!=0)

{

q->data.status=0;

}

}

}

return SIZE;

}

Status deal2(Node \*p)//处理回收空间

{

Node \*q=first;

for(;q!=NULL;q=q->next)

{

if(q==p)

{

if(q->prior->data.status==0&&q->next->data.status!=0)

{

q->prior->data.size+=q->data.size;

q->prior->next=q->next;

q->next->prior=q->prior;

q=p->prior;

q->data.status=0;

q->data.num=flag-1;

}

if(q->prior->data.status!=0&&q->next->data.status==0)

{

q->data.status=0;

}

if(q->prior->data.status==0&&q->next->data.status==0)

{

q->prior->data.size+=q->data.size;

q->prior->next=q->next;

q->next->prior=q->prior;

q=q->prior;

q->data.status=0;

q->data.num=flag-1;

}

if(q->prior->data.status!=0&&q->next->data.status!=0)

{

q->data.status=0;

}

}

}

return SIZE;

}

//主存回收

Status recovery(int flag)

{

Node \*p=first;

for(;p!=NULL;p=p->next)

{

if(p->data.num==flag)

{

if(p->prior==first)

{

if(p->next!=end)//当前P指向的下一个不是最后一个时

{

if(p->next->data.status==0) //与后面的空闲块相连

{

p->data.size+=p->next->data.size;

p->next->next->prior=p;

p->next=p->next->next;

p->data.status=0;

p->data.num=flag;

}

else p->data.status=0;

}

if(p->next==end)//当前P指向的下一个是最后一个时

{

p->data.status=0;

}

}

//结束if(p->prior==block\_first)的情况

else if(p->prior!=first)

{

if(p->next!=end)

{

deal1(p);

}

else

{

deal2(p);

}

}//结束if(p->prior!=block\_first)的情况

}//结束if(p->data.num==flag)的情况

}

printf("\t\*\*\*\*回收成功\*\*\*\*");

return SIZE;

}

int best\_sf()

{

int i; //操作选择标记

int a;//算法选择标记

while(1)

{

menu();

printf("请选择输入所使用的内存分配算法 (0~2):");

scanf("%d",&a);

while(a<0||a>3)

{

printf("\n输入错误，请重新选择输入所使用的内存分配算法 (0~2):");

scanf("%d",&a);

}

switch(a)

{

case 1:printf("\n\t\*\*\*\*使用首次适应算法：\*\*\*\*\n");

break;

case 2:printf("\n\t\*\*\*\*使用最佳适应算法：\*\*\*\*\n");

break;

case 0:printf("\n\t\*\*\*\*退出内存分配与回收\*\*\*\*\n");

return 0;

}

Initblock(); //开创空间表

while(1)

{

show();

printf("\t1: 分配内存\t2: 回收内存\t0: 退出当前内存分配算法\n");

printf("请输入您的操作：");

scanf("%d",&i);

if(i==1)

allocation(a); // 分配内存

else if(i==2) // 内存回收

{

printf("请输入您要释放的分区号：");

scanf("%d",&flag);

recovery(flag);

}

else if(i==0)

{

printf("\n退出当前内存分配算法，返回主菜单\n");

break; //退出

}

else //输入操作有误

{

printf("输入有误，请重新输入！");

continue;

}

}

}

}

//适应算法 end

struct BitMap //用于存储位示图

{

int map[8][8]; //8\*8=64，位示图

int free; //剩余的空闲块数

}bitmap;

typedef struct process //用于存储作业

{

int num; //作业序号

int size; //作业大小

int \*pagetable; //页表

struct process \*next; //下一作业

struct process \*pre; //上一作业

}process;

void SetProcess(process \*head) //创立作业，生成页表

{

int setnum;

int i=0,j=0,k=0,finish=0;

process \*temp = new process();

cout<<endl<<"请输入作业序号(数字且不为0)：";

cin>>setnum;

if(setnum == 0)

{

cout<<endl<<"错误！不能创建0号作业，返回。";

delete temp;

return;

}

temp->num = setnum;

cout<<endl<<"请输入作业大小(<=64)：";

cin>>temp->size;

if(temp->size > bitmap.free) //判断现有空间是否足够分配

{

cout<<endl<<"当前内存没有足够的空间分配，返回。";

delete temp;

return;

}

temp->next = head->next; //头插法插入新作业

if(head->next != NULL)

head->next->pre = temp;

temp->pre = head;

head->next = temp;

temp->pagetable = new int [temp->size]; //创立页表数组

for(i=0; i<8 && finish == 0; i++)

{

for(j=0; j<8 && finish == 0; j++)

{

if(bitmap.map[i][j] == 0) //修改页表数据

{

bitmap.map[i][j] = 1;

bitmap.free --;

temp->pagetable[k] = 8\*i + j;

k++;

}

if(k == temp->size) //用于结束循环

finish = 1;

}

}

cout<<endl<<"已装入作业，页表如下："<<endl<<"页号\t"; //输出页表

k = temp->size;

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

{

cout<<i+1<<'\t';

}

cout<<endl<<"块号\t";

for(j=0; j<k; j++)

{

cout<<temp->pagetable[j]<<'\t';

}

}

void RecProcess(process \*head) //回收作业，退还内存

{

int recnum;

int i=0,j=0,k=0,l=0,found=0;

process \*temp = new process();

cout<<endl<<"请输入回收作业序号(数字且不为0)：";

cin>>recnum;

if(recnum == 0)

{

cout<<endl<<"错误！不能删除0号作业，返回。";

return;

}

for(temp=head; temp != NULL; temp = temp->next) //找到进程

{

if(temp->num == recnum)

{

found = 1;

k = temp->size;

for(l=0; l<k; l++) //删除数据

{

i = temp->pagetable[l] / 8;

j = temp->pagetable[l] % 8;

bitmap.map[i][j] = 0;

bitmap.free ++;

}

if(temp->pre != NULL) //不允许删除头结点

{

if(temp->next != NULL) //不是尾结点

{

temp->pre->next = temp->next;

temp->next->pre = temp->pre;

delete temp;

}

else //是尾结点

{

temp->pre->next = NULL;

delete temp;

}

}

cout<<endl<<"已找到并回收序号为"<<recnum<<"的作业。";

cout<<endl<<"该作业大小为"<<k<<"。"<<endl;

continue;

}

}

if(found == 0) //未找到进程

{

cout<<endl<<"未找到该作业，返回。";

return;

}

cout<<endl<<"完成回收，返回。";

return;

}

void ShowBitMap() //展示位示图

{

cout<<endl<<"当前位示图状态如下："<<endl;

cout<<'\t'<<"0\t"<<"1\t"<<"2\t"<<"3\t"<<"4\t"<<"5\t"<<"6\t"<<"7\t";

cout<<endl<<"\*\*\*\*";

int i=0,j=0;

for(i=0; i<8; i++)

{

cout<<endl<<i<<'\t';

for(j=0; j<8; j++)

{

cout<<bitmap.map[i][j]<<'\t';

}

}

cout<<endl<<endl<<"完成位示图展示，返回。";

}

void ShowPageTable(process \*head) //展示页表

{

int shownum;

int i=0,j=0,k=0,l=0,found=0;

process \*temp = new process();

cout<<endl<<"请输入查看作业序号(数字且不为0)：";

cin>>shownum;

if(shownum == 0)

{

cout<<endl<<"错误！不能查看0号作业，返回。";

return;

}

for(temp=head; temp != NULL; temp = temp->next) //找到进程

{

if(temp->num == shownum)

{

found = 1;

cout<<endl<<"序号为"<<shownum<<"的作业页表如下："<<endl<<"页号\t"; //输出页表

k = temp->size;

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

{

cout<<i+1<<'\t';

}

cout<<endl<<"块号\t";

for(j=0; j<k; j++)

{

cout<<temp->pagetable[j]<<'\t';

}

cout<<endl<<"已找到并展示序号为"<<shownum<<"的作业页表。";

cout<<endl<<"该作业大小为"<<k<<"。"<<endl;

continue;

}

}

if(found == 0) //未找到进程

{

cout<<endl<<"未找到该作业，返回。";

return;

}

cout<<endl<<"完成作业页表展示，返回。";

}

int postMain()

{

char choice;

bitmap.map[8][8] = 0;

bitmap.free = 64;

process \*head = new process();

head->next = NULL;

head->pre = NULL;

head->num = 0;

do

{

cout<<endl;

cout<<endl<<"-----------操作列表-----------";

cout<<endl<<"------------------------------";

cout<<endl<<"0.退出程序；";

cout<<endl<<"1.创建新作业，并请求内存空间；";

cout<<endl<<"2.回收作业，并返回内存；";

cout<<endl<<"3.展示位示图；";

cout<<endl<<"4.展示进程页表；";

cout<<endl<<"5.清空屏幕。";

cout<<endl<<"------------------------------";

cout<<endl<<"\*\*请选择您要进行的操作：";

cin>>choice;

switch(choice)

{

case '1':

SetProcess(head);

break;

case '2':

RecProcess(head);

break;

case '3':

ShowBitMap();

break;

case '4':

ShowPageTable(head);

break;

case '5':

system("cls");

break;

default:

cout<<endl<<"输入错误！请重新输入！";

}

}while(choice != '0');

return 0;

}

//位示图 end

void w\_view(){

printf("\n\t\*\*\*\*1.使用适应算法：\*\*\*\*\n");

printf("\n\t\*\*\*\*2.空闲区链：\*\*\*\*\n");

printf("\n\t\*\*\*\*3.位示图：\*\*\*\*\n");

printf("\n\t\*\*\*\*4.退出\*\*\*\*\n");

printf("请输入您的操作:");

}

int main(){

int opera = 4;

bool flag = true;

while(flag){

w\_view();

scanf("%d",&opera);

switch(opera){

case 1:

best\_sf();

break;

case 2:

linkPCB();

break;

case 3:

postMain();

break;

default:

flag = false;

break;

}

}

}