**实验二 存储管理动态分区粉配及回收算法**

课程名称 操作系统 信息科学与技术 学院 软件工程 学院 信1405-1班 学号 20143048

实验者姓名 张作秀 实验日期 2017 年 5 月 8 日

评分 教师签名

### 实验一 进程调度模拟算法

1. 目的和要求

分区管理是应用较广泛的一种存储管理技术。本实验要求用一种结构化高级语言构造分区描述器，编制动态分区分配算法和回收算法模拟程序，并讨论不同分配算法的特点。

二、实验内容  
1、编写： First Fit Algorithm  
2、编写： Best Fit Algorithm  
3、编写：空闲区回收算法

三、提示和说明  
（一）主程序  
1、定义分区描述器 node，包括 3 个元素：  
（1） adr——分区首地址  
（2） size——分区大小  
（3） next——指向下一个分区的指针  
2、定义 3 个指向 node 结构的指针变量：  
（1） head1——空闲区队列首指针  
（2） back1——指向释放区 node 结构的指针  
（3） assign——指向申请的内存分区 node 结构的指针  
3、定义 1 个整形变量：  
free——用户申请存储区的大小（由用户键入）  
（二）过程  
1、定义 check 过程，用于检查指定的释放块（由用户键入）的合法性  
2、定义 assignment1 过程，实现 First Fit Algorithm  
3、定义 assignment2 过程，实现 Best Fit Algorithm  
4、定义 acceptment1 过程，实现 First Fit Algorithm 的回收算法  
5、定义 acceptment2 过程，实现 Best Fit Algorithm 的回收算法  
6、定义 print 过程，打印空闲区队列  
（三）执行  
程序首先申请一整块空闲区，其首址为 0，大小为 32767；然后，提示用户使用哪种分  
配算法，再提示是分配还是回收；分配时要求输入申请区的大小，回收时要求输入释放区的  
首址和大小。  
（四）输出  
要求每执行一次，输出一次空闲区队列情况，内容包括：编号 首址 终址 大小

（五）源代码：

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MAX\_SIZE 32767

typedef struct node //定义分区描述器的结构

{

int id; //编号

int adr; //分区首地址

int size; //分区大小

struct node \*next;//指向下一个分区的指针

}Node;

Node \*head1, \*head2, \*back1, \*back2, \*assign;/\*head--空闲区队列首指针

back--指向释放区Node结构的指针

assign-指向申请的内存分区Node结构的指针\*/

int request; //用户申请存储区的大小（由用户输入）

int check(int add, int siz, char c)//用于检查指定的释放块（由用户键入）的合法性

{

Node \*p, \*head;

int check = 1;

if (add<0 || siz<0)

check = 0;/\*地址和大小不能为负\*/

if (c == 'f' || c == 'F')

head = head1;

else

head = head2;

p = head->next;

while ((p != NULL) && check)/\*地址不能和空闲区表中结点出现重叠\*/

if (((add<p->adr) && (add + siz>p->adr)) || ((add >= p->adr) && (add<p->adr + p->size)))

check = 0;

else

p = p->next;

if (check == 0)

printf("\t输入释放区地址或大小有错误！！！\n");

return check; /\*返回检查结果\*/

}

void init()//初始化，生成两个带头节点的空闲队列指针，

{ //head1指向first-fit的空闲队列头，head2指向best-fit的空闲队列头

Node \*p, \*q;

head1 = (Node\*)malloc(sizeof(Node));

head2 = (Node\*)malloc(sizeof(Node));

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

q = (Node\*)malloc(sizeof(Node));

head1->next = p;

head2->next = q;

p->size = q->size = MAX\_SIZE;

p->adr = q->adr = 0;

p->next = q->next = NULL;

p->id = q->id = 0;

}

Node\* assignment1(int num, int req)//实现首次适应算法的分配

{

Node \*before, \*after, \*ass;

ass = (Node\*)malloc(sizeof(Node));

before = head1;

after = head1->next;

ass->id = num;

ass->size = req;

while (after->size<req)

{

before = before->next;

after = after->next;

}

if (after == NULL)

{

ass->adr = -1;//分配失败

}

else

{

if (after->size == req)

{//空闲分区恰好等于所申请的内存大小

before->next = after->next;

ass->adr = after->adr;

}

else

{//空闲分区大于所申请的内存大小

after->size -= req;

ass->adr = after->adr;

after->adr += req;

}

}

return ass;

}

void acceptment1(int address, int siz, int rd)//实现首次适应算法的回收

{

Node \*before, \*after;

int insert = 0;

back1 = (Node\*)malloc(sizeof(Node));

before = head1;

after = head1->next;

back1->adr = address;

back1->size = siz;

back1->id = rd;

back1->next = NULL;

while (!insert&&after)

{//将要被回收的分区插入空闲区（按首址大小从小到大插入）

if ((after == NULL) || ((back1->adr <= after->adr) && (back1->adr >= before->adr)))

{

before->next = back1;

back1->next = after;

insert = 1;

}

else

{

before = before->next;

after = after->next;

}

}

if (insert)

{

if (back1->adr == before->adr + before->size)

{//和前边分区合并

before->size += back1->size;

before->next = back1->next;

free(back1);

}

else if (after&&back1->adr + back1->size == after->adr)

{//和后边分区合并

back1->size += after->size;

back1->next = after->next;

back1->id = after->id;

free(after);

after = back1;

}

printf("\t首先分配算法回收内存成功！！！\n");

}

else

printf("\t首先分配算法回收内存失败！！！\n");

}

Node\* assignment2(int num, int req)//实现最佳适应算法的分配

{

Node \*before, \*after, \*ass, \*q;

ass = (Node\*)malloc(sizeof(Node));

q = (Node\*)malloc(sizeof(Node));

before = head2;

after = head2->next;

ass->id = num;

ass->size = req;

while (after->size<req)

{

before = before->next;

after = after->next;

//printf("\nzphzph1\n");

}

if (after == NULL)

{

ass->adr = -1;//分配失败

//printf("\nzphzph2\n");

}

else

{

if (after->size == req)

{//空闲分区恰好等于所申请的内存大小

before->next = after->next;

ass->adr = after->adr;

//printf("\nzphzph3\n");

}

else

{//空闲分区大于所申请的内存大小

q = after;

before->next = after->next;

ass->adr = q->adr;

q->size -= req;

q->adr += req;

//printf("\nzphzph4\n");

before = head2;

after = head2->next;

//printf("\nzphzph4\n");

if (after == NULL)

{

//printf("\nzphzph5\n");

before->next = q;

q->next = NULL;

after = q;

}

else

{//printf("\nzphzph6\n");

while (after && ((after->size)<(q->size)))

{

printf("\nzphzph7\n");

before = before->next;

after = after->next;

}

//printf("\nzphzph8\n");

before->next = q;

q->next = after;

after = q;

}

}

}

return (ass);

}

void acceptment2(int address, int siz, int rd)//实现最佳适应算法的回收

{

Node \*before, \*after;

int insert = 0;//是否被回收的标志

back2 = (Node\*)malloc(sizeof(Node));

before = head2;

after = head2->next;

back2->adr = address;

back2->size = siz;

back2->id = rd;

back2->next = NULL;

if (head2->next == NULL)

{//空闲队列为空

head2->next = back2;

//head2->size=back2->size;

}

else

{//空闲队列不为空

while (after)

{

if (back2->adr == after->adr + after->size)

{//和前边空闲分区合并

before->next = after->next;

after->size += back2->size;

back2 = after;

after = before->next;

}

else

{

before = before->next;

after = after->next;

}

}

before = head2;

after = head2->next;

while (after)

{

if (after->adr == back2->adr + back2->size)

{//和后边空闲区合并

before->next = after->next;

back2->size += after->size;

after = before->next;

}

else

{

before = before->next;

after = after->next;

}

}

before = head2;

after = head2->next;

while (!insert)

{//将被回收的块插入到恰当的位置（按分区大小从小到大）

if (after == NULL || ((after->size>back2->size) && (before->size<back2->size)))

{

before->next = back2;

back2->next = after;

insert = 1;

break;

}

else

{

before = before->next;

after = after->next;

}

}

}

if (insert)

printf("\t最佳适应算法回收内存成功！！！\n");

else

printf("\t最佳适应算法回收内存失败！！！\n");

}

void print(char choice)//输出空闲区队列信息

{

Node \*p;

if (choice == 'f' || choice == 'F')

p = head1->next;

else

p = head2->next;

if (p)

{

printf("\n空闲区队列的情况为：\n");

printf("\t编号\t首址\t终址\t大小\n");

while (p)

{

printf("\t%d\t%d\t%d\t%d\n", p->id, p->adr, p->adr + p->size - 1, p->size);

p = p->next;

}

}

}

void menu()//菜单及主要过程

{

char chose;

int ch, num, r, add, rd;

while (1)

{

system("cls");

printf("\t\t存储管理动态分区分配及回收算法模拟\n\n");

printf("\tF.最先适应算法(First-Fit)\n\n");

printf("\tB.最佳适应算法(Best-Fit)\n\n");

printf("\tE.退出程序\n\n");

printf("\t你选择（f/b）:");

scanf("%c", &chose);

if (chose == 'e' || chose == 'E')

exit(0);

else

{

system("cls");

while (1)

{

if (chose == 'f' || chose == 'F')

printf("\t\t最先适应算法(First-Fit)模拟\n\n");

if (chose == 'b' || chose == 'B')

printf("\t\t最佳适应算法(Best-Fit)模拟\n\n");

printf("\t1.分配内存\t\t2.回收内存\n\n");

printf("\t3.查看内存\t\t4.返回\n\n");

printf("\t你选择（1/2/3/4）:");

scanf("%d", &ch);

fflush(stdin);

switch (ch)

{

case 1:

printf("输入分区号："); scanf("%d", &num);

printf("输入申请的分区大小："); scanf("%d", &r);

if (chose == 'f' || chose == 'F')

assign = assignment1(num, r);

else

assign = assignment2(num, r);

if (assign->adr == -1)

{

printf("分配内存失败！\n");

}

else

printf("分配成功！分配的内存的首址为:%d\n", assign->adr);

break;

case 2:

printf("输入释放的内存的首址："); scanf("%d", &add);

printf("输入释放的内存的大小："); scanf("%d", &r);

printf("输入释放的内存的编号："); scanf("%d", &rd);

if (check(add, r, chose))

{

if (chose == 'f' || chose == 'F')

acceptment1(add, r, rd);

else

acceptment2(add, r, rd);

}

break;

case 3:

print(chose);

break;

case 4:

menu();

break;

}

}

}

}

}

void main()//主函数

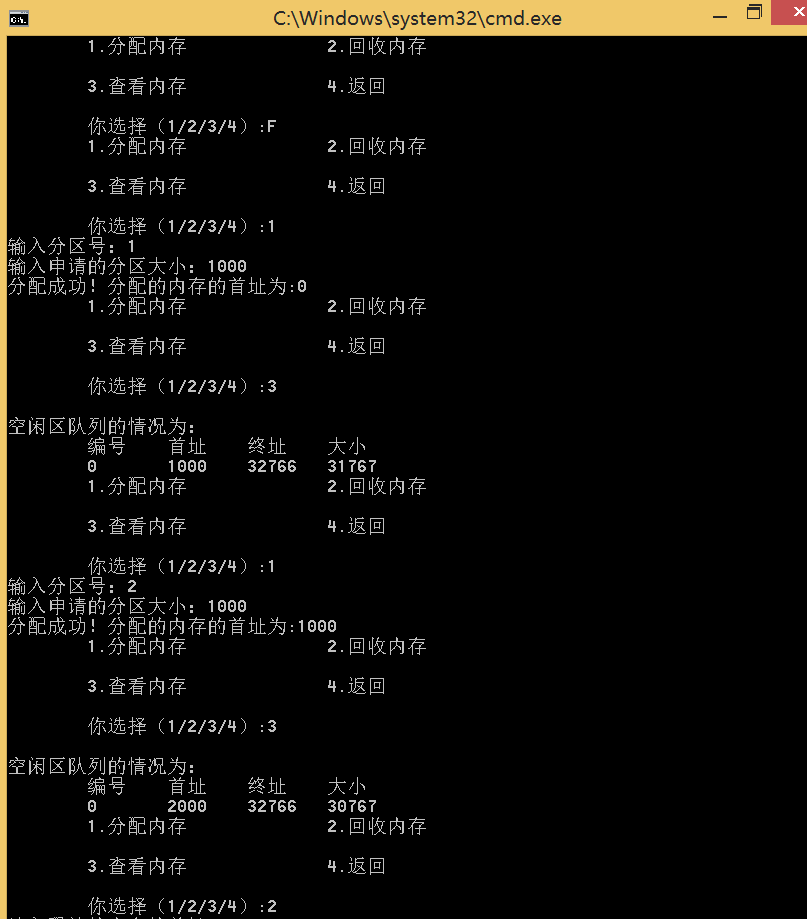
{

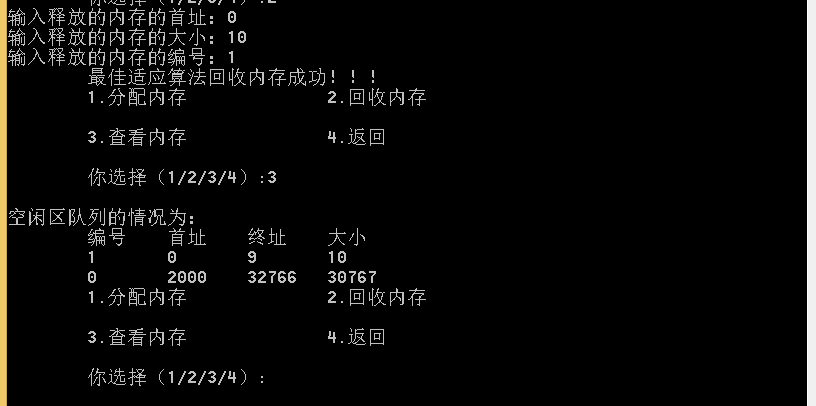
init();

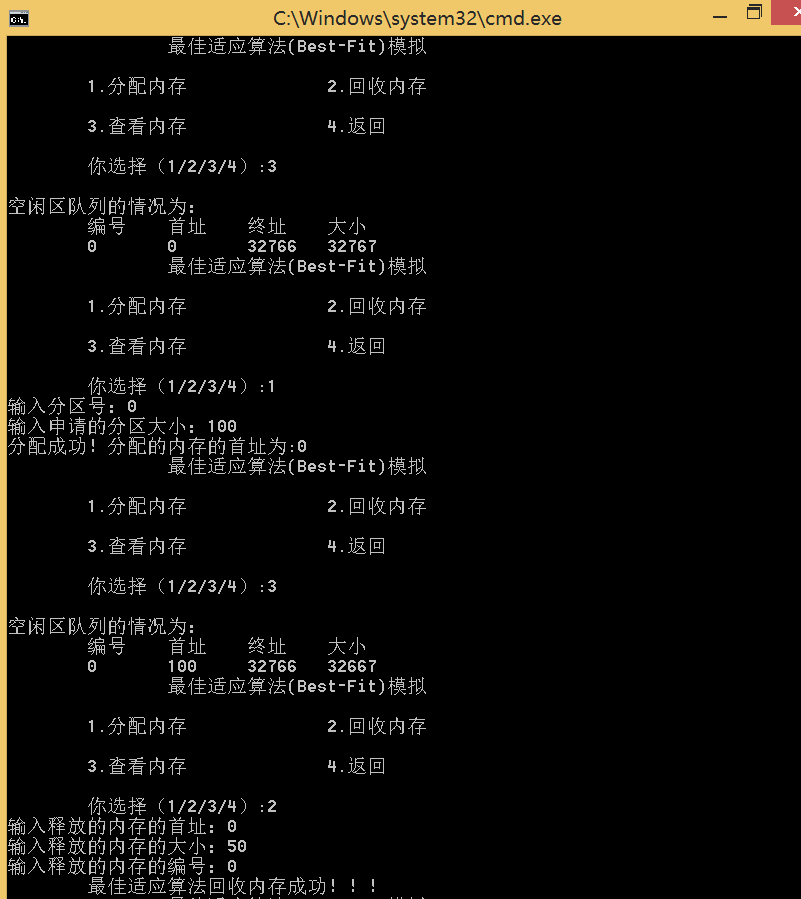
menu();

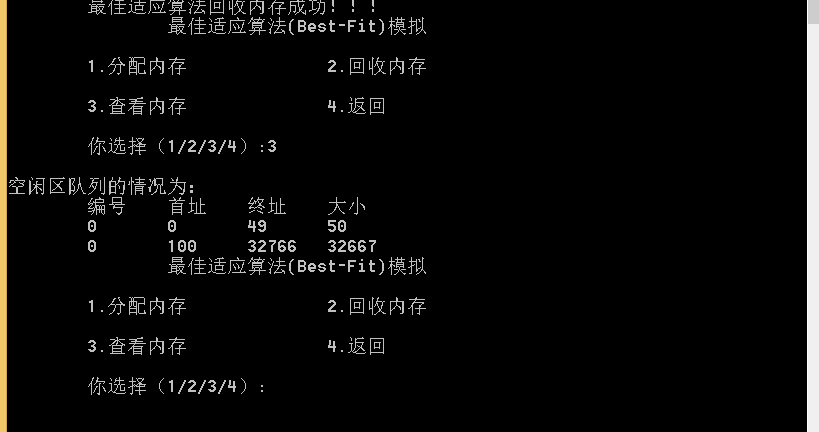
}

四、实验结果









五、实验总结

通过这次实验，我对于可变存储分区管理有了更深刻的认识能够更好地认识。刚开始的时候无从下手，后来从网上找了一下相关的代码看了看，根据课上讲的知用程序语言去理解代码，最终整合了出来。把课上的内容算法，课上讲的知识，融入到程序语言中，对于自己的编程能力也是一种提升。