**实验二 存储管理动态分区分配及回收算法实验报告**

课程名称 操作系统 信息技术与科学 分院信1605-3 班 组 桌号

实验者姓名 杨蕾 实验日期 2019 年 4 月 29 日

评分 教师签名

一、实验目的

分区管理是应用较广泛的一种存储管理技术，了解存储管理动态分区分配及回收算法。

二、实验要求

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。

三、实验过程

1．准备

A． 查阅相关资料；

通过看书对最早适应算法及最先适应算法有更深的理解。

B． 初步编写程序；

精读提示与说明，创建node结构体，并定义 3个指向 node 结构的指针变量，用于对进程中的内容进行

C． 准备测试数据；

分别对最早适应算法及最佳适应算法准备测试数据。

如分配内存(1,100)（2,400），（3,1000），回收内存为首址为100;对不同算法分别进行调错检验。分配100000地址。

D． 运行。

2． 上机调试。

对最早适应算法及最佳适应算法分别进行调试。

3． 主要流程和源代码。

**主要流程：**

建立node结构及其他有关变量

定义3个指向node结构的指针变量，空闲区队列首址针，释放区node结构指针及申请的内存分区指针，及申请存储区大小。

定义不同的函数，check用于检查指定的释放块（由用户键入）的合法性，assignment1 过程，实现最早适应分配，acceptment1实现最早适应回收，assignment2实现最优适应分配，acceptment2实现最优适应回收，print实现打印空闲队列。

使用main函数将算法连接起来。

**源代码：**

/\*空闲区回收算法\*/

#include<iostream>

#define MAX\_SIZE 32767

using namespace std;

struct Node

{

int pid;

int adr;//分区首地址

int size;//分区大小

Node \*next;//指向下一个分区的指针

};

void print(Node \*head, Node \*pro)//输出空闲区队列信息

{

Node \*p, \*pl;

p = head;

if (p->next)

{

cout << endl << "空闲队列：" << endl << "编号\t首址\t终址\t大小" << endl;

while (p->next)

{

p = p->next;

if (p->pid < 0)

{

p->pid = 0;

}

cout << endl <<p->pid<<"\t"<< p->adr << "\t" << p->adr + p->size - 1 << "\t" << p->size << endl;

}

}

pl = pro;

if (pl->next)

{

cout << endl << "进程队列：" << endl << "编号\t首址\t终址\t大小" << endl;

while (pl->next)

{

pl = pl->next;

cout << endl << pl->pid << "\t" << pl->adr << "\t" << pl->adr + pl->size - 1 << "\t" << pl->size << endl;

}

}

}

//检测释放内存的合法性 查看其是否在进程队列

bool check(int pid, Node \*pro)

{

Node \*pl;

bool check = false;

pl = pro;

while (pl->next)

{

pl = pl->next;

if (pl->adr == pid)

{

check = true;

break;

}

}

if (!check)

{

cout << endl << "没有此进程" << endl;

}

return check;

}

void assignment1(int pid, int free, Node \*head, Node \*pro)

{

Node \*head1, \*after, \*assign, \*pl;

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

head1 = head;

after = head->next;

pl = pro;

while (pl->next)

{

pl = pl->next;

}

while (after != NULL&&after->size < free)

{

head1 = head1->next;

after = after->next;

}

if (after == NULL)

{

cout << endl << "分配失败" << endl;

}

else

{

if (after->size == free)

{

assign->pid = pid;

assign->adr = after->adr;

assign->size = free;

assign->next = NULL;

head1->next = after->next;

pl->next = assign;

pl = pl->next;

}

else

{

assign->pid = pid;

assign->adr = after->adr;

assign->size = free;

assign->next = NULL;

after->adr += free;

after->size -= free;

pl->next = assign;

pl = pl->next;

}

cout << endl << "分配成功" << endl;

}

}

void acceptment1(int pid, Node \*head, Node \*pro)

{

Node \*head1, \*after, \*back, \*pl;

head1 = head;

after = head->next;

pl = pro;

while (pl->next)

{

pl = pl->next;

if (pl->adr == pid)

{

break;

}

}

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

back->pid = pl->pid;

back->adr = pl->adr;

back->size = pl->size;

back->next = NULL;

bool insert = false;

//空闲队列为空

if (after == NULL)

{

head1->next = back;

cout << endl << "回收成功" << endl;

}

//空闲队列不为空

else

{

while (!insert && after)

{

//将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 || back->adr >= head1->adr && back->adr <= after->adr)

{

head1->next = back;

back->next = after;

insert = true;

}

else

{

head1 = head1->next;

after = after->next;

}

}

if (insert)

{

if (back->adr == head1->adr + head1->size && after && back->adr + back->size == after->adr)

{

//和前后分区同时合并

head1->size += back->size;

head1->size += after->size;

head1->next = after->next;

back = NULL;

after = head1->next;

}

else if (back->adr == head1->adr + head1->size)

{

//和前分区合并

head1->size += back->size;

head1->next = back->next;

back = NULL;

}

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

{

//和后分区合并

back->size += after->size;

back->next = after->next;

after = back;

}

cout << endl << "回收成功" << endl;

}

else

{

cout << endl << "回收失败" << endl;

}

}

Node \*p;

p = pro;

while (p->next)

{

if (p->next->pid == pl->pid)

{

p->next = pl->next;

break;

}

p = p->next;

}

}

void assignment2(int pid, int free, Node \*head, Node \*pro)

{

Node \*head1, \*after, \*assign, \*p, \*pl;

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

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

head1 = head;

after = head->next;

pl = pro;

while (pl->next)

{

pl = pl->next;

}

while (after != NULL&&after->size < free)

{

head1 = head1->next;

after = after->next;

}

if (after == NULL)

{

cout << endl << "分配失败" << endl;

}

else

{

if (after->size == free)

{

assign->pid = pid;

assign->adr = after->adr;

assign->size = free;

assign->next = NULL;

head1->next = after->next;

pl->next = assign;

pl = pl->next;

}

else

{

assign->pid = pid;

assign->adr = after->adr;

assign->size = free;

assign->next = NULL;

head1->next = after->next;

pl->next = assign;

pl = pl->next;

//对分区重新排序

p = after;

p->adr += free;

p->size -= free;

head1 = head;

after = head->next;

if (after == NULL)

{

head1->next = p;

p->next = NULL;

}

else

{

while (after != NULL && after->size < p->size)

{

head1 = head1->next;

after = after->next;

}

head1->next = p;

p->next = after;

}

}

cout << endl << "分配成功" << endl;

}

}

void acceptment2(int pid, Node \*head, Node \*pro)

{

Node \*head1, \*after, \*back, \*pl;

head1 = head;

after = head->next;

pl = pro;

while (pl->next)

{

pl = pl->next;

if (pl->adr == pid)

{

break;

}

}

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

back->pid = pl->pid;

back->adr = pl->adr;

back->size = pl->size;

back->next = NULL;

bool insert = false;

if (after == NULL)

{

//空闲队列为空

head1->next = back;

cout << endl << "回收成功" << endl;

}

else

{

//空闲队列不为空

//和前分区合并

head1 = head;

after = head->next;

while (after)

{

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

{

head1->next = after->next;

after->size += back->size;

back = after;

back->next = NULL;

after = after->next;

break;

}

else

{

head1 = head1->next;

after = after->next;

}

}

//和后分区合并

head1 = head;

after = head->next;

while (after)

{

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

{

head1->next = after->next;

back->size += after->size;

back->next = NULL;

after = after->next;

break;

}

else

{

head1 = head1->next;

after = after->next;

}

}

head1 = head;

after = head->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 || back->size >= head1->size && back->size <= after->size)

{

head1->next = back;

back->next = after;

insert = true;

break;

}

else

{

head1 = head1->next;

after = after->next;

}

}

if (insert)

{

cout << endl << "回收成功" << endl;

}

else

{

cout << endl << "回收失败" << endl;

}

}

Node \*p;

p = pro;

while (p->next)

{

if (p->next->pid == pl->pid)

{

p->next = pl->next;

break;

}

p = p->next;

}

}

void start(Node \*head)

{

Node \*pro;

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

pro->next = NULL;

int option, choose, free, pid,adr;

print(head, pro);

cout << endl << "1.最先适应算法" << endl << endl << "2.最佳适应算法" << endl;

cout << endl << "请选择:";

cin >> option;

while (true)

{

switch (option)

{

case 1://最先适应算法

cout << endl << "1.分配内存" << endl << endl << "2.回收内存" << endl << endl << "3.返回" << endl;

cout << endl << "请选择:";

cin >> choose;

switch (choose)

{

case 1://分配内存

cout << endl << "请输入进程ID: ";

cin >> pid;

cout << endl << "请输入大小：";

cin >> free;

assignment1(pid, free, head, pro);

print(head, pro);

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

break;

case 2://回收内存

cout << endl << "请输入首址: ";

cin >> adr;

if (check(adr, pro))

{

acceptment1(adr, head, pro);

print(head, pro);

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

}

break;

case 3:

return;

print(head, pro);

cout << endl << "1.最先适应算法" << endl << endl << "2.最佳适应算法" << endl;

cout << endl << "请选择:";

cin >> option;

break;

default:

cout << endl << "输入有误，请重新输入" << endl;

break;

}

break;

case 2://最佳适应算法

cout << endl << "1.分配内存" << endl << endl << "2.回收内存" << endl << endl << "3.返回" << endl;

cout << endl << "请选择:";

cin >> choose;

switch (choose)

{

case 1://分配内存

cout << endl << "请输入进程ID: ";

cin >> pid;

cout << endl << "请输入大小：";

cin >> free;

assignment2(pid, free, head, pro);

print(head, pro);

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

break;

case 2://回收内存

cout << endl << "请输入首址: ";

cin >> adr;

if (check(adr, pro))

{

acceptment2(adr, head, pro);

print(head, pro);

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

}

break;

case 3:

return;

print(head, pro);

cout << endl << "1.最先适应算法" << endl << endl << "2.最佳适应算法" << endl;

cout << endl << "请选择:";

cin >> option;

break;

default:

cout << endl << "输入有误，请重新输入" << endl;

break;

}

break;

default:

cout << endl << "输入有误，请重新输入" << endl;

print(head, pro);

cout << endl << "1.最先适应算法" << endl << endl << "2.最佳适应算法" << endl;

cout << endl << "请选择:";

cin >> option;

break;

}

}

}

void main()

{

while (true)

{

//初始化

Node \*head, \*p;

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

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

p->adr = 0;

p->size = MAX\_SIZE;

p->next = NULL;

head->next = p;

//初始化完成

start(head);

free(head);

head = NULL;

free(p);

p = NULL;

}

}

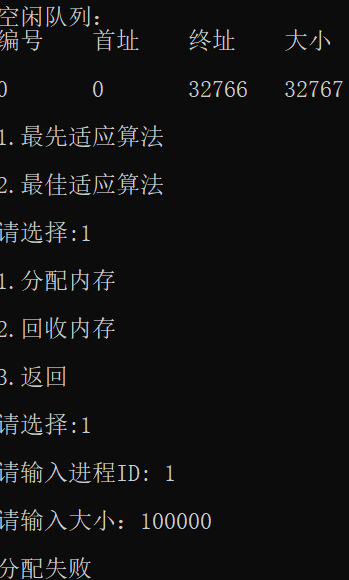
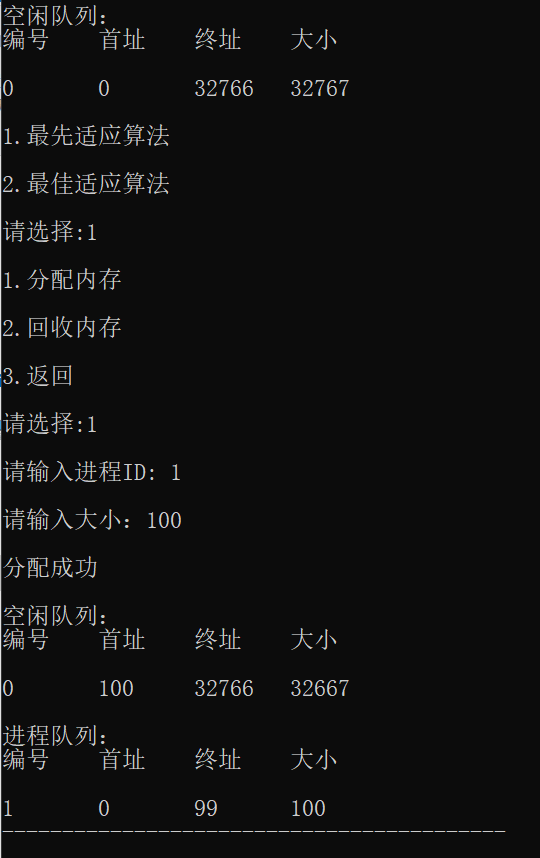
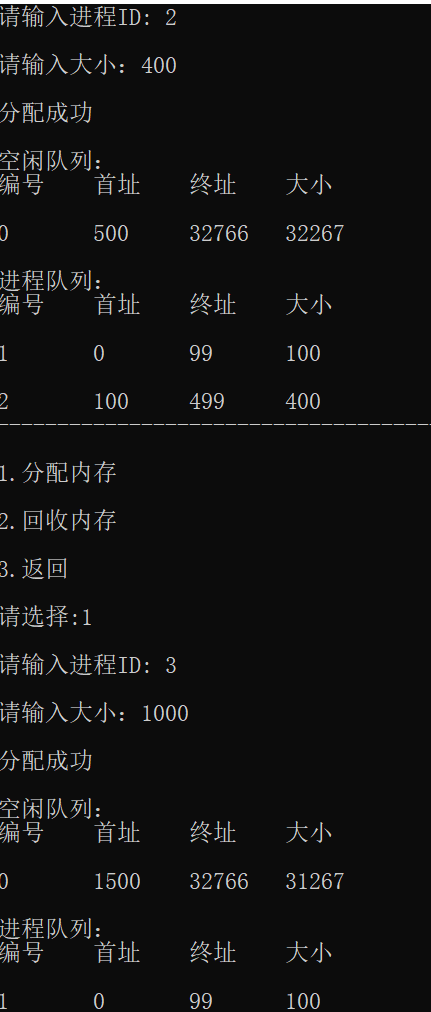
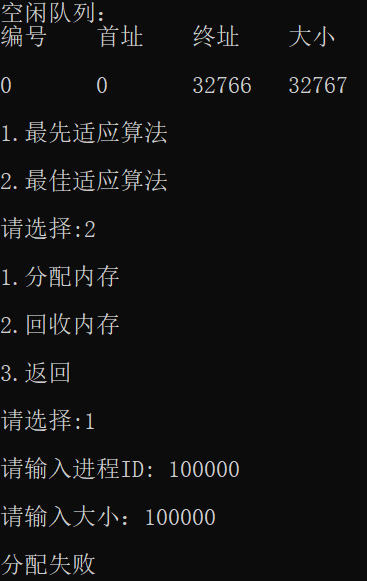
4． 遇到的主要问题和解决方法

A．对空链表重新生成一直使head为null而导致next读不出，导致出错，解决直接ruturn返回。

B．对于分配内存过大的内容,通过在判断after->size < p->size和大小同时，加入after!=null

C．本题回收为回收一块，对于直接输入回收块的首址即可，回收最近的回收块大小。

四、实验结果

五、实验总结

通过本次实验，我学到了最先适应算法及最优适应算法在代码中的实现方法，了解了两者在实现细节的不同，掌握了列表的回收与分配、释放，提高了编程能力与对指针的灵活应用，对return、if、while等语句的使用更加熟悉。