课程名称 操作系统

实验者姓名 孟祥通 实验日期 2017 年 5 月 5 日

1. **实验目的**

通过使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。

1. **实验要求**
2. 定义分区描述器node。
3. 定义 3个指向node 结构的指针变量。
4. 定义 1个整形变量：用户申请存储区的大小（由用户键入）。
5. **实验过程**
6. 程序执行：

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。

1. 遇到的问题

在进行内存回收时，需要分四种情况，一种是与左边分区合并，一种是与右边分区合并，一种是即与左边分区合并又与右边分区合并，最后一种是不需合并。在左右分区都需合并这种情况下，空闲区指针和待释放区指针的指向问题一直困扰了我好长时间。

1. 程序源码：

#include<iostream>

#include<string>

using namespace std;

#define MAX\_SIZE 32767

typedef struct node{

int adr; //分区首地址

int size; //分区大小

struct node \*next; //指向下一个分区地址

}Node;

Node \*head1, \*back1, \*assign;

void init();

int check(int,int);

Node\* assignment1(int);

Node\* assignment2(int);

void acceptment1(int,int);

void acceptment2(int,int);

void print(string);

void main()

{

init();

string ch;

int chose;

int free;

int address;

while (1)

{

cout << "请选择地址分配算法（最先适应算法请输入F，最佳适应算法请输入B），退出程序请输入E" << endl;

cin >> ch;

if (ch == "E" || ch == "e")

{

exit(0);

}

else if (ch == "F" || ch == "f")

{

cout << "最先适应算法模拟：" << endl;

}

else if (ch == "B" || ch == "b")

{

cout << "最佳适应算法模拟：" << endl;

}

else

{

cout << "输入信息有误，请重新输入" << endl;

continue;

}

cout << "1.分配内存 2.回收内存 " << endl;

cin >> chose;

switch (chose)

{

case 1:

cout << "请输入申请的分区大小：(最大不能超过32767)" << endl;

cin >> free;

if (ch == "F" || ch == "f")

assign = assignment1(free);

else

assign = assignment2(free);

if (assign->adr == -1)

cout << "分配内存失败" << endl;

else

cout << "分配内存成功"<< endl;

print(ch);

break;

case 2:

cout << "请输入释放区的首地址和大小：" << endl;

cin >> address >> free;

int i = check(address,free);

if (i == 1)

{

if (ch == "F" || ch == "f")

acceptment1(address, free);

else

acceptment1(address, free);

}

else

{

cout << "释放区输入信息有误，无法找到所输释放区！" << endl;

}

print(ch);

break;

}

}

}

void init()

{

head1 = new Node;

back1 = new Node;

assign = new Node;

Node \*p = new Node;

head1->next = p;

p->adr = 0;

p->size = MAX\_SIZE;

p->next = NULL;

}

Node\* assignment1(int free)

{

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

ass = new Node;

before = head1;

after = head1->next;

ass->size = free;

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

{

before = before->next;

after = after->next;

}

if (after == NULL)

{

ass->adr = -1;

}

else

{

if (after->size == free)

{

before->next = after->next;

ass->adr = after->adr;

}

else

{

after->size -= free;

ass->adr = after->adr;

after->adr += free;

}

}

return ass;

}

Node\* assignment2(int free)

{

Node \*before, \*after,\*after1,\*ass;

int index=32767;

ass = new Node;

before = head1;

after = head1->next;

after1 = NULL;

ass->size = free;

while (after != NULL)

{

if (after->size == free)

{

before->next = after->next;

ass->adr = after->adr;

return ass;

break;

}

if ((after->size > free) && (after->size - free < index))

{

index = after->size - free;

after1 = after;

}

before = before->next;

after = after->next;

}

if (after1 == NULL)

{

ass->adr = -1;

return ass;

}

else

{

after1->size -= free;

ass->adr = after1->adr;

after1->adr += free;

}

return ass;

}

void acceptment1(int address,int size)

{

Node \*before, \*after;

int insert = 0;

back1 = new Node();

before = head1;

after = head1->next;

back1->adr = address;

back1->size = size;

back1->next = NULL;

while (!insert&&after)

{

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)

{

cout << "back1->adr:" << back1->adr << "back1->size:" << back1->size << endl;

cout << "after->adr:" << after->adr << "after->size:" << after->size << endl;

cout << "before->adr:" << before->next->adr << "before->size:" << before->next->size << endl;

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

{

before->size = before->size+back1->size+after->size;

back1->next = after->next;

after = after->next;

before->next = back1->next;

}

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

{

//和前边分区合并

before->size += back1->size;

before->next = back1->next;

}

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

{

//和后边分区合并

back1->size += after->size;

back1->next = after->next;

after = back1;

}

else

{

}

cout << "回收内存成功" << endl;

}

else

cout << "回收内存失败！" << endl;

}

void acceptment2(int address, int free)

{

Node \*after, \*ass;

ass = new Node;

after = head1->next;

ass->size = free;

}

int check(int address, int free)

{

int check = 1;

Node \*p,\*head;

if (address < 0 || free < 0)

check = 0;

head = head1;

p = head1->next;

while ((p != NULL) && check)

{

if (((address < p->adr)&&(address+free>p->adr))||((address>=p->adr)&&(address<p->adr+p->size)))

{

check = 0;

}

else

{

p = p->next;

}

}

return check;

}

void print(string choice)

//输出空闲区队列信息

{

Node \*p = new Node;

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

p = head1->next;

else

p = head1->next;

if (p)

{

cout << "空闲区队列的情况为：" << endl;

cout << "首址\t终址\t大小" << endl;

while (p)

{

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

p = p->next;

}

}

}#include<iostream>

#include<string>

using namespace std;

#define MAX\_SIZE 32767

typedef struct node{

int adr; //分区首地址

int size; //分区大小

struct node \*next; //指向下一个分区地址

}Node;

Node \*head1, \*back1, \*assign;

void init();

int check(int,int);

Node\* assignment1(int);

Node\* assignment2(int);

void acceptment1(int,int);

void acceptment2(int,int);

void print(string);

void main()

{

init();

string ch;

int chose;

int free;

int address;

while (1)

{

cout << "请选择地址分配算法（最先适应算法请输入F，最佳适应算法请输入B），退出程序请输入E" << endl;

cin >> ch;

if (ch == "E" || ch == "e")

{

exit(0);

}

else if (ch == "F" || ch == "f")

{

cout << "最先适应算法模拟：" << endl;

}

else if (ch == "B" || ch == "b")

{

cout << "最佳适应算法模拟：" << endl;

}

else

{

cout << "输入信息有误，请重新输入" << endl;

continue;

}

cout << "1.分配内存 2.回收内存 " << endl;

cin >> chose;

switch (chose)

{

case 1:

cout << "请输入申请的分区大小：(最大不能超过32767)" << endl;

cin >> free;

if (ch == "F" || ch == "f")

assign = assignment1(free);

else

assign = assignment2(free);

if (assign->adr == -1)

cout << "分配内存失败" << endl;

else

cout << "分配内存成功"<< endl;

print(ch);

break;

case 2:

cout << "请输入释放区的首地址和大小：" << endl;

cin >> address >> free;

int i = check(address,free);

if (i == 1)

{

if (ch == "F" || ch == "f")

acceptment1(address, free);

else

acceptment1(address, free);

}

else

{

cout << "释放区输入信息有误，无法找到所输释放区！" << endl;

}

print(ch);

break;

}

}

}

void init()

{

head1 = new Node;

back1 = new Node;

assign = new Node;

Node \*p = new Node;

head1->next = p;

p->adr = 0;

p->size = MAX\_SIZE;

p->next = NULL;

}

Node\* assignment1(int free)

{

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

ass = new Node;

before = head1;

after = head1->next;

ass->size = free;

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

{

before = before->next;

after = after->next;

}

if (after == NULL)

{

ass->adr = -1;

}

else

{

if (after->size == free)

{

before->next = after->next;

ass->adr = after->adr;

}

else

{

after->size -= free;

ass->adr = after->adr;

after->adr += free;

}

}

return ass;

}

Node\* assignment2(int free)

{

Node \*before, \*after,\*after1,\*ass;

int index=32767;

ass = new Node;

before = head1;

after = head1->next;

after1 = NULL;

ass->size = free;

while (after != NULL)

{

if (after->size == free)

{

before->next = after->next;

ass->adr = after->adr;

return ass;

break;

}

if ((after->size > free) && (after->size - free < index))

{

index = after->size - free;

after1 = after;

}

before = before->next;

after = after->next;

}

if (after1 == NULL)

{

ass->adr = -1;

return ass;

}

else

{

after1->size -= free;

ass->adr = after1->adr;

after1->adr += free;

}

return ass;

}

void acceptment1(int address,int size)

{

Node \*before, \*after;

int insert = 0;

back1 = new Node();

before = head1;

after = head1->next;

back1->adr = address;

back1->size = size;

back1->next = NULL;

while (!insert&&after)

{

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)

{

cout << "back1->adr:" << back1->adr << "back1->size:" << back1->size << endl;

cout << "after->adr:" << after->adr << "after->size:" << after->size << endl;

cout << "before->adr:" << before->next->adr << "before->size:" << before->next->size << endl;

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

{

before->size = before->size+back1->size+after->size;

back1->next = after->next;

after = after->next;

before->next = back1->next;

}

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

{

//和前边分区合并

before->size += back1->size;

before->next = back1->next;

}

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

{

//和后边分区合并

back1->size += after->size;

back1->next = after->next;

after = back1;

}

else

{

}

cout << "回收内存成功" << endl;

}

else

cout << "回收内存失败！" << endl;

}

void acceptment2(int address, int free)

{

Node \*after, \*ass;

ass = new Node;

after = head1->next;

ass->size = free;

}

int check(int address, int free)

{

int check = 1;

Node \*p,\*head;

if (address < 0 || free < 0)

check = 0;

head = head1;

p = head1->next;

while ((p != NULL) && check)

{

if (((address < p->adr)&&(address+free>p->adr))||((address>=p->adr)&&(address<p->adr+p->size)))

{

check = 0;

}

else

{

p = p->next;

}

}

return check;

}

void print(string choice)

//输出空闲区队列信息

{

Node \*p = new Node;

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

p = head1->next;

else

p = head1->next;

if (p)

{

cout << "空闲区队列的情况为：" << endl;

cout << "首址\t终址\t大小" << endl;

while (p)

{

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

p = p->next;

}

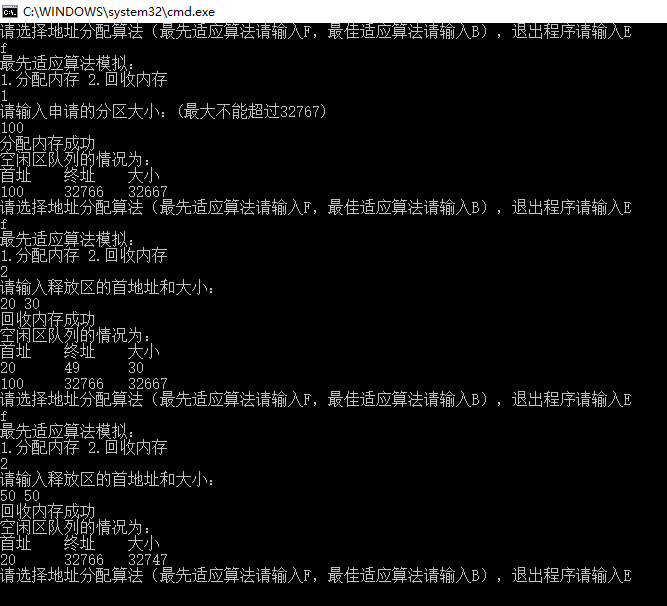
}

}

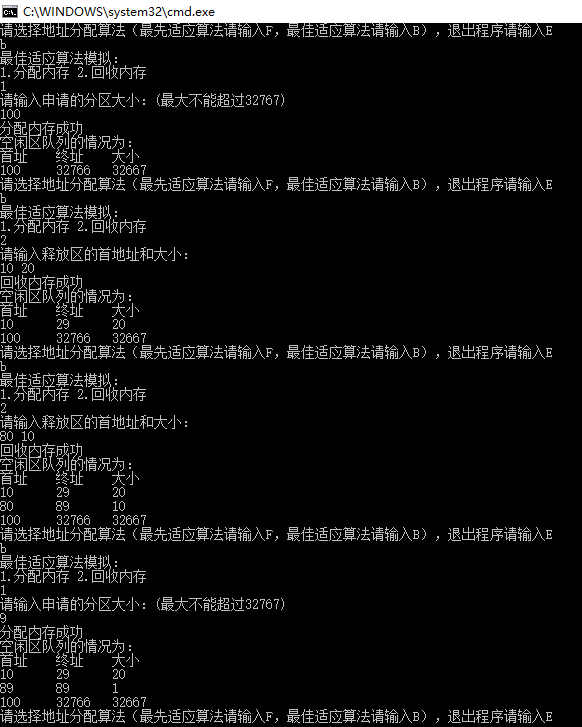
1. **实验结果**

实验结果截图：

最先适应算法结果截图：

****

最佳适应算法结果截图：



1. **实验总结**

通过此次实验我学会了最佳适应算法、最先适应算法和内存回收算法，了解了动态分区分配算法和回收算法模拟程序的基本原理，掌握了如何利用队列来实现动态分区分配算法和回收算法。