**石 家 庄 铁 道 大 学**

**实 验 报 告**

|  |  |  |
| --- | --- | --- |
| **课程名称**： 操作系统A | **任课教师：** 陈娜 | **实验日期**： 2021.4.26 |
| **班 级**： 信1805-2 | **姓 名**： 宋泊然 | **学 号**： 20183553 |

1. 实验目的

通过对存储管理动态分区分配及回收算法的实验，了解到了首次适应算法与最佳适应算法，体会到了两种调度算法区别，掌握了两种算法的分配与回收编码的实现，提高了对它们的理解。

1. 实验要求

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。

1. 实验过程
2. 准备
3. 查阅相关资料

c++list、制表符、首次适应算法、最佳适应算法

1. 初步编写程序

准备两个list，与两种list排序方式...

1. 准备测试数据

申请内存：

内存大小：200

内存大小：2445

回收内存：

首地址和大小：0 200

1. 上机调试
2. 主要流程和源代码
3. 主要流程：

首先程序自动申请32767大小的空间，然后提示选择算法

选择首次适应算法

提示用户选择申请还是释放或者退出

选择申请

提示用户输入申请区大小

从空闲区地址从小到大查找该大小的空间

若无，申请失败

若有申请成功，提示用户申请的首地址，返回提示

选择释放

输入释放区首地址和大小，检查输入是否合法

不合法提示用户重新输入

合法则回收

回收后按首地址从小到大排序，并合并相邻区间

选择最佳适应算法

提示用户选择申请还是释放或者退出

选择申请

提示用户输入申请区大小

从空闲区按大小从小到大查找该大小的空间

若无，申请失败

若有申请成功，提示用户申请的首地址，返回提示

选择释放

输入释放区首地址和大小，检查输入是否合法

不合法提示用户重新输入

合法则回收

回收后按区大小从小到大排序，并合并相邻区间

1. 源代码

#include <bits/stdc++.h>

using namespace std;

const int N = 32767;

int number[N];

struct node

{

int num;//编号

int adr,size,next;

};

list <node> l;

list <node> l2;

int n,m;

class my\_greater1

{

public:

bool operator () (const node s1, const node s2)

{

if (s1.size == s2.size)

return s1.adr < s2.adr;

else

return s1.size < s2.size;

}

};

class my\_greater2

{

public:

bool operator () (const node s1, const node s2)

{

return s1.adr < s2.adr;

}

};

void start\_0()

{

node p;

p.adr = 0;

p.size = N;

p.num = 0;

number[0] = 1;

p.next = p.adr + p.size;

l.push\_back(p);

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*程序已申请一整块空闲区，其首址为0，大小为32767\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*请选择算法\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*First Fit Algorithm请输入1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Best Fit Algorithm请输入2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

}

void start\_1()

{

cout<<"请选择:1.分配内存，2.回收内存，3.退出"<<endl;

}

void start\_2()

{

cout<<"输入申请的分区大小"<<endl;

}

void print\_free()

{

cout<<endl<<"空闲区队列情况为："<<endl;

list<node>::iterator it;

for (it = l.begin();it != l.end(); ++it)

{

node p = \*it;

cout<<"编号"<<"\t"<<"首地址"<<"\t\t"<<"终止地址"<<"\t"<<"大小"<<"\t"<<endl;

cout<<p.num<<"\t"<<p.adr<<"\t\t"<<p.next<<"\t\t"<<p.size<<"\t"<<endl;

}

}

void print\_use()

{

cout<<endl<<"使用区队列情况为："<<endl;

list<node>::iterator it;

for (it = l2.begin();it != l2.end(); ++it)

{

node p = \*it;

cout<<"编号"<<"\t"<<"首地址"<<"\t\t"<<"终止地址"<<"\t"<<"大小"<<"\t"<<endl;

cout<<p.num<<"\t"<<p.adr<<"\t\t"<<p.next<<"\t\t"<<p.size<<"\t"<<endl;

}

}

int main(int argc, char const \*argv[])

{

start\_0();

cin>>m;

while(1)

{

start\_1();

cin>>n;

if (n == 1)

{

bool flag = false;

int start\_sq;

cout<<"请输入申请区的大小："<<endl;

node pp;

cin>>pp.size;

list<node>::iterator it;

for (it = l.begin();it != l.end(); ++it)

{

node p = \*it;

if (p.size >= pp.size)

{

start\_sq = p.adr;

p.adr += pp.size;

p.size = p.next - p.adr;

pp.adr = start\_sq;

pp.next = pp.adr + pp.size;

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

{

//cout<<"asd"<<endl;

if (number[i] == 0)

{

number[i] = 1;

pp.num = i;

break;

}

}

l2.push\_back(pp);

l.emplace(it,p);

l.erase(it);

flag = true;

break;

}

}

if (flag) cout<<"申请成功！申请区起始地址为："<<start\_sq<<endl;

else cout<<"申请失败！"<<endl;

}

else if (n == 2)

{

if (l2.empty())

{

cout<<"暂无申请区可以释放！请先分配内存！"<<endl;

continue;

}

else

{

cout<<"请输入释放区的首地址："<<endl;

node pp;

cin>>pp.adr;

cout<<"请输入释放区的大小："<<endl;

cin>>pp.size;

pp.next = pp.adr + pp.size;

list<node>::iterator it;

bool flag = true;

for (it = l2.begin();it != l2.end(); ++it)

{

node p = \*it;

if (p.size == pp.size && p.adr == pp.adr && p.next == pp.next)

{

flag = false;

cout<<"释放成功！"<<endl;

l.push\_back(p);

l2.erase(it);

break;

}

}

if (flag) cout<<"释放失败，请检查输入的首地址与大小！"<<endl;

}

}

else break;

list <node> lt;

stack<node> s;

l.sort(my\_greater2());

list<node>::iterator it;

it = l.begin();

node p = \*it;

s.push(p);

it++;

for (;it != l.end(); ++it)

{

node pp = \*it;

p = s.top();

s.pop();

if (p.next == pp.adr)

{

p.next = pp.next;

p.size = p.next - p.adr;

p.num = min(p.num,pp.num);

number[max(p.num,pp.num)] = 0;

s.push(p);

}

else

{

s.push(p);

s.push(pp);

}

}

while(!s.empty())

{

node pt = s.top();

s.pop();

lt.push\_back(pt);

}

swap(l,lt);

if (m == 1) l.sort(my\_greater2());

else l.sort(my\_greater1());

print\_free();

//print\_use();

}

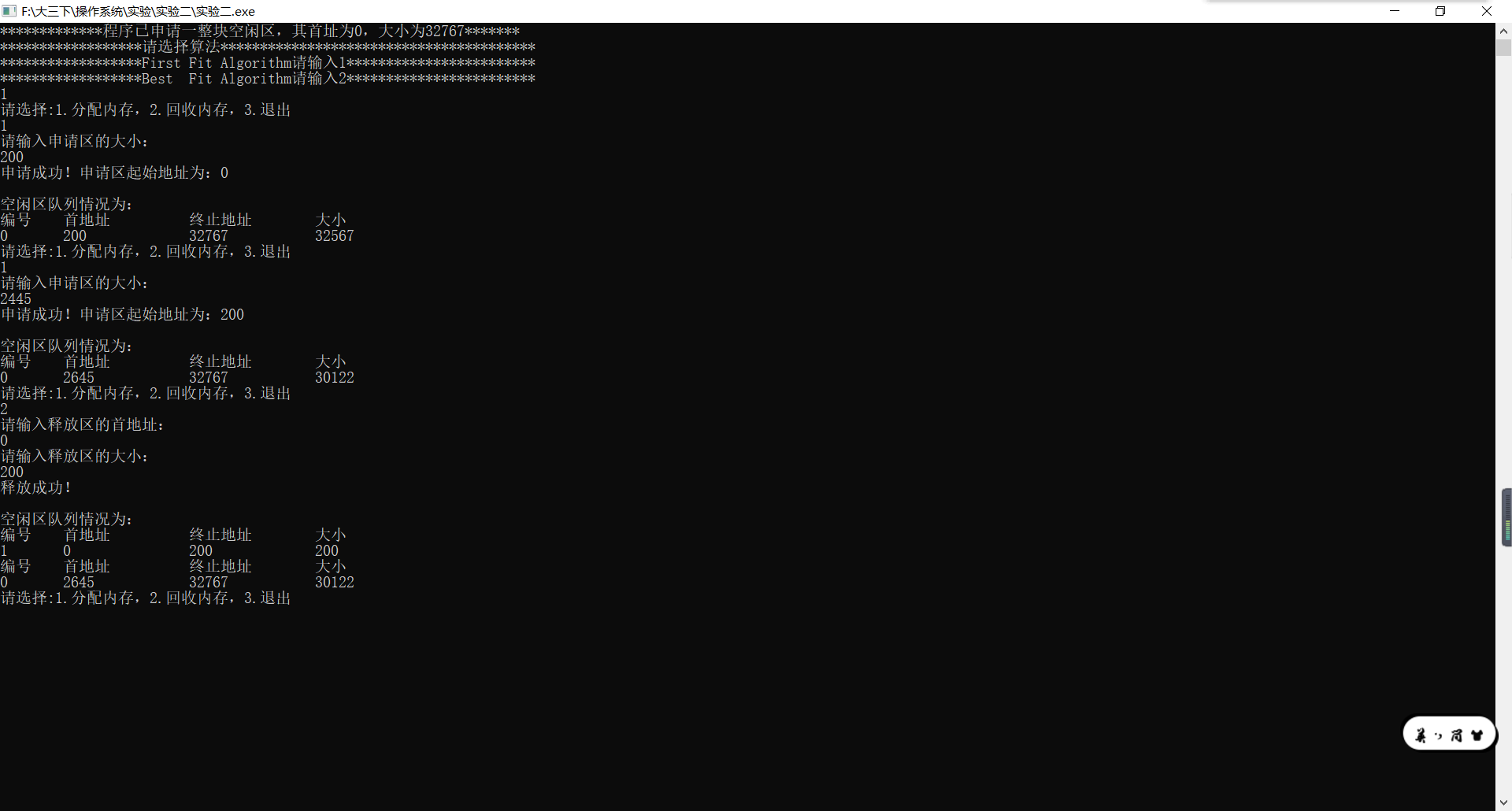
return 0;

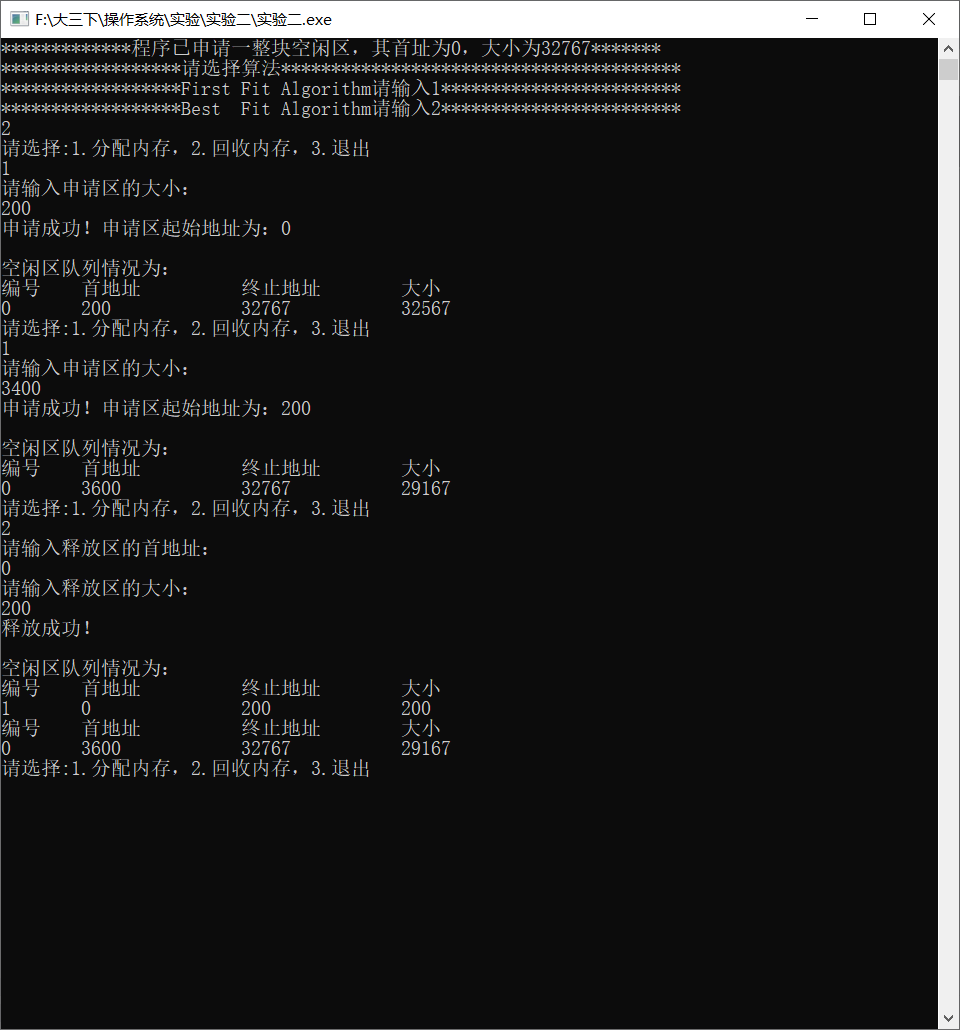
}

1. 遇到的主要问题和解决方法

1.要将list进行自定义排序，通过上网查解决

1. 实验结果





1. 实验总结

通过本次实验，我了解了首次适应，最佳适应的流程，优缺点。

一、首次适应算法（First Fit）：该算法从空闲分区链首开始查找，直至找到一个能满足其大小要求的空闲分区为止。然后再按照作业的大小，从该分区中划出一块内存分配给请求者，余下的空闲分区仍留在空闲分区链中。

     特点： 该算法倾向于使用内存中低地址部分的空闲区，在高地址部分的空闲区很少被利用，从而保留了高地址部分的大空闲区。显然为以后到达的大作业分配大的内存空间创造了条件。

     缺点：低地址部分不断被划分，留下许多难以利用、很小的空闲区，而每次查找又都从低地址部分开始，会增加查找的开销。

  二、最佳适应算法（Best Fit）：该算法总是把既能满足要求，又是最小的空闲分区分配给作业。为了加速查找，该算法要求将所有的空闲区按其大小排序后，以递增顺序形成一个空白链。这样每次找到的第一个满足要求的空闲区，必然是最优的。孤立地看，该算法似乎是最优的，但事实上并不一定。因为每次分配后剩余的空间一定是最小的，在存储器中将留下许多难以利用的小空闲区。同时每次分配后必须重新排序，这也带来了一定的开销。

     特点：每次分配给文件的都是最合适该文件大小的分区。

     缺点：内存中留下许多难以利用的小的空闲区。