石家庄铁道大学

操作系统实验

2017 年 春 季学期

学 院： 信息科学与技术学院

专 业： 软件工程

实 验 名 称： 存储管理动态分区分配及回收算法

学 生 姓 名： 祖浩然

学 号： 20142903

指 导 教 师： 陈娜

完 成 日 期： 2017年 5月11日

**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 信息科学与技术 学院 信1405-2班

实验者姓名 祖浩然 实验日期 2017年 5月11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。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. **实验要求**

1、编写：First Fit Algorithm

2、编写：Best Fit Algorithm

3、编写：空闲区回收算法

**三、实验过程**

**1．准备**

A．查阅相关资料；

B．初步编写程序；

C．准备测试数据；

**2．上机调试**

**3．主要流程和源代码**

开始

Frist，best算法

将空闲队列按照地址由小到大排列

将空闲队列按照空间由小到大排列

判断是否有足够的空间

便利队列

获取合适的空间的位置

区域大于申请区域

分区

将找到的区域指针放入申请区

结束

开始

判断是否有对应的申请区

获将对应位置的申请去放入释放区

获将释放区插入空闲队列

是否存在前分区空闲

与释放区合并。前分区删除

是否存在后分区空闲

与释放区合并。后分区删除

结束

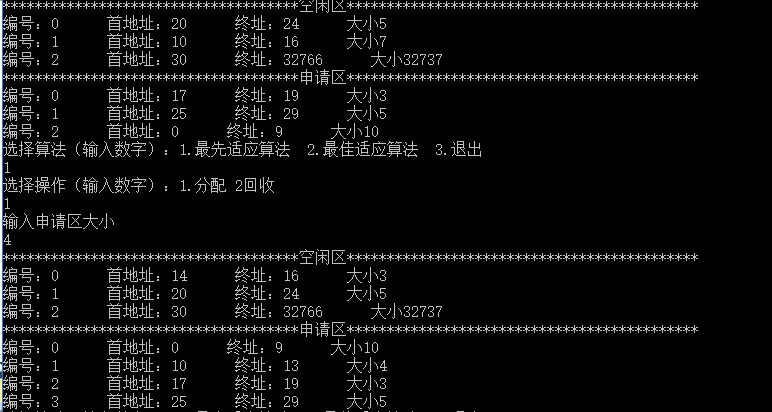
|  |
| --- |
| #include<iostream>  #include<string>  using namespace std;  typedef struct node  {  int adr=0;//分区首地址  int size=0;//分区大小  node \*next=NULL;//指向下一个分区的指针  }node;  void initlist(node\* &L)//第一个节点为空  {  L = new node;  L->next = NULL;  }  void listadd(node\* &head, node\* Point)//申请的空闲区添加到申请区  {  Point->next = head->next;  head->next = Point;  }  void freedel(node\* &headl, int wz)  {  node\* s;//遍历空闲区队列的指针  s = headl;  for (int i = 0; i < wz; i++)//找到申请位置的前一个空闲区  {  s = s->next;  }  s->next = s->next->next;//删除s的下一个节点  }  void outlist(node\* &head)//输出链表  {  node\* s;  s = head->next;  int i=0;  while (s!=NULL)  {  cout << "编号：" << i << " 首地址：" << s->adr << " 终址：" << s->adr + s->size-1 << " 大小" << s->size<<endl;  s = s->next;  i++;  }  }  void fenqv(node\* &s, node nod)//分区  {  node\* ns = new node;//新建分区  ns->adr = s->adr + nod.size;//新分区的地址赋值  ns->size = s->size - nod.size;//新分区的大小赋值  ns->next = s->next;//新分区的next赋值  s->size = nod.size;  s->next = ns;//s分区next为新分区  }  bool findfree(node\* &headl, node\* &assign,node nod) //backl找到合适的空闲区，并调去申请区  {  int length=0,wz=0;//记录链表长度，合适空闲区的位置  bool b=false;  node\* s;//遍历空闲区队列的指针  node\* assignPoint;//申请的空闲区    //初始化  s = headl->next;  assignPoint =NULL;  //判断是否存在合适的空闲区  while (s!=NULL)  {    if (s->size>=nod.size)  {  wz = length;  assignPoint = s;//获取申请空闲区  b = true;  break;  }  s = s->next;  length++;//headl的下一个为0；  }  if (b==false)  {  return b;  }  //判断空闲区是否有富余  if (assignPoint->size > nod.size)  {  //进行分区  fenqv(assignPoint, nod);  }  //将分好的空闲区前部分（或没有分区的选好的空闲区）指向申请区 并从空闲队列删除    freedel(headl, wz);  listadd(assign, assignPoint);  return b;  }  void adrmin(node\* &head)//寻找head节点 之后 的地址最小值，放在head的next(不包括head节点)  {  node\* min;//最小值的指针  node\* s;//遍历指针  min = head->next;  s = head->next;  int wz = 0,length=0;//位置，长度  //  while (s!=NULL)  {  if (s->adr < min->adr)//如果遍历到的地址小于min的地址，min赋值  {  min = s;  wz = length;//head->next的位置为0；  }  length++;  s = s->next;  }  s = head;  //从队列中删除最小值,找到最小值的前一个  for (int i = 0; i < wz; i++)  {  s = s->next;  }  s->next = s->next->next;  //将最小值插入对头  min->next = head->next;  head->next = min;  }  void sizemin(node\* &head)//寻找head节点 之后 的地址最小值，放在head的next(不包括head节点)  {  node\* min;//最小值的指针  node\* s;//遍历指针  min = head->next;  s = head->next;  int wz = 0, length = 0;//位置，长度  //  while (s != NULL)  {  if (s->size < min->size)//如果遍历到的空间小于min的空间，min赋值  {  min = s;  wz = length;//head->next的位置为0；  }  length++;  s = s->next;  }  s = head;  //从队列中删除最小值,找到最小值的前一个  for (int i = 0; i < wz; i++)  {  s = s->next;  }  s->next = s->next->next;  //将最小值插入队头  min->next = head->next;  head->next = min;  }  void adrPaiXu(node\* &headl)//将链表安地址排序  {  node\* s;//遍历空闲区队列的指针,用于排列  s = headl;  while (s->next)  {  adrmin(s);  s = s->next;//s移动到找到的节点上  }  }  void sizePaiXu(node\* &headl)//将链表安大小排序  {  node\* s;//遍历空闲区队列的指针,用于排列  s = headl;  while (s->next)  {  sizemin(s);  s = s->next;//s移动到找到的节点上  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //释放  //找到s的前一个节点，赋值给s  int findbef(node\* &head, node\* &point)  {  int length = 0, wz = 0;//记录链表长度，合适空闲区的位置  node\* s=head->next;//遍历空闲区队列的指针  while (s != NULL)  {  if (s->adr == point->adr)  {  wz = length;  }  s = s->next;  length++;//headl的下一个为0；  }  s = head;  for (int i = 0; i < wz; i++)//找到申请位置的前一个空闲区  {  s = s->next;  }  point = s;  return wz;  }  bool check(node\* &assign, node nod)//检查指定释放块的合法性  {  node \*s = assign->next;//s初始化为申请区的对头后一个  while (s != NULL)  {  if (s->adr == nod.adr)  {  if (s->size >= nod.size)  {  return true;  }  }  s = s->next;  }  return false;  }  //判断前一个区是否为空  bool befnull(node\* &headl, node\* &backl)  {  node\* s = headl->next;  bool panbef = true;  while (s != NULL)  {  if (s->adr + s->size == backl->adr)  {  panbef = false;//前一个为空闲状态  break;  }  s = s->next;  }  if (panbef == false)//前一个空闲，进行合并  {  backl->adr = s->adr;  backl->size += s->size;  freedel(headl, findbef(headl, s));  }  return panbef;  }  //判断后一个分区是否为空  bool aftnull(node\* &headl, node\* &backl)  {  node\* s = headl->next;  bool panbef = true;  while (s != NULL)  {  if (s->adr == backl->adr+backl->size)  {  panbef = false;//前一个为空闲状态  break;  }  s = s->next;  }  if (panbef == false)//前一个空闲，进行合并  {  backl->size += s->size;  freedel(headl, findbef(headl, s));  }  return panbef;  }  //在申请区找到对应的区域，并处理  bool findback(node\* &backl, node\* &assign, node nod)  {  if (check(assign, nod) == false)  {  cout << "输入释放区地址或大小错误"<<endl;  return false;  }  node\* s = assign->next;//s初始化为申请区的对头后一个  int length = 0,wz=0;//从对头后一个为0  //找到释放的位置  while (s != NULL)  {  if (s->adr == nod.adr)  {  break;  }  s = s->next;  wz++;  }    //如果回收空间小于使用空间则，进行分区，  //将前半部释放，从申请区删除（判断前分区是否空闲，空闲合并）  if (s->size > nod.size)  {  fenqv(s, nod);  backl = s;//赋值给释放指针  freedel(assign, wz);  }  else if (s->size = nod.size)  {  backl = s;//赋值给释放指针  freedel(assign, wz);  }  else  {  return false;  }  return true;  }  void assignment1(node\* &headl, node\* &assign, node nod)//实现First Fit Algorithm  {  adrPaiXu(headl);  if (findfree(headl, assign, nod) == false)  {  cout << "没有足够的空间"<<endl;  }  else  {  adrPaiXu(headl);  cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*空闲区\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;  outlist(headl);  adrPaiXu(assign);  cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*申请区\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;  outlist(assign);  }    }  void assignment2(node\* &headl, node\* &assign, node nod)//实现Best Fit Algorithm  {  sizePaiXu(headl);  if (findfree(headl, assign, nod) == false)  {  cout << "没有足够的空间";  }  else  {  sizePaiXu(headl);  cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*空闲区\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;  outlist(headl);  sizePaiXu(assign);  cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*申请区\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;  outlist(assign);  }    }  void acceptment1(node\* &headl,node\* &backl, node\* &assign,node nod)//实现First Fit Algorithm 的回收算法  {  adrPaiXu(headl);  //在assign找到对应位置，  //如果大于输出错误  //否则将释放区从申请区队列删除  findback(backl, assign, nod);  //将释放区插入空闲队列  listadd(headl, backl);    //判断后分区是否空闲，(将后分区的空间加到释放区上，并把后分区删除)  //空闲合并  aftnull(headl, backl);    //判断前分区是否空闲，（）  //空闲合并，  befnull(headl, backl);    //进行排序  adrPaiXu(headl);  cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*空闲区\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;  outlist(headl);    adrPaiXu(assign);  cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*申请区\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;  outlist(assign);  }  void acceptment2(node\* &headl, node\* &backl, node\* &assign, node nod)//实现Best Fit Algorithm 的回收算法  {  sizePaiXu(headl);  //在assign找到对应位置，  //如果大于输出错误  //否则将释放区从申请区队列删除  findback(backl, assign, nod);  //将释放区插入空闲队列  listadd(headl, backl);  //判断后分区是否空闲，(将后分区的空间加到释放区上，并把后分区删除)  //空闲合并  aftnull(headl, backl);  //判断前分区是否空闲，（）  //空闲合并，  befnull(headl, backl);  //进行排序  sizePaiXu(headl);  cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*空闲区\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;  outlist(headl);  sizePaiXu(assign);  cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*申请区\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;  outlist(assign);  }  void main()  {  node\* headl;//空闲区队列首指针  node\* backl; //指向释放区node结构的指针  node\* assign;//指向申请的内存分区node结构的指针  int free=0;//申请区的大小  int acsize=0;//释放区的大小  int acadr=0;//释放区的首地址  //指针初始化  initlist(headl);  initlist(backl);  initlist(assign);    //空闲队列赋初值  node\* n = new node;  n->adr = 0;  n->size = 32767;  n->next = NULL;  headl->next = n;  node nod;  //循环运行  int xz,xz2;  while (true)  {  //输入数值  cout << "选择算法（输入数字）：1.最先适应算法 2.最佳适应算法 3.退出"<<endl;  cin >> xz;  if (xz == 3)  {  break;  }  if (xz != 1 && xz != 2 && xz != 3)  {  cout << "输入错误" << endl;  continue;  }  cout << "选择操作（输入数字）：1.分配 2回收"<<endl;  cin >> xz2;    if (xz2 != 1 && xz2 != 2 )  {  cout << "输入错误" << endl;  continue;  }  if (xz2 == 1)  {  cout << "输入申请区大小"<<endl;  cin >> free;  nod.adr = 0;  nod.size = free;  }  else if (xz2 == 2)  {  cout << "输入释放区首地址和大小"<<endl;  cin >> acadr >> acsize;  nod.adr = acadr;  nod.size = acsize;  }  else  {  cout << "输入错误" << endl;  continue;  }    //进行运算  if (xz == 1)  {  if (xz2 == 1)  {  assignment1(headl,assign,nod);  }  else  {  acceptment1(headl,backl,assign,nod);  }  }  else if (xz==2)  {  if (xz2 == 1)  {  assignment2(headl, assign, nod);  }  else  {  acceptment2(headl, backl, assign, nod);  }  }  else if (xz==3)  {  break;  }  else  {  cout << "输入错误" << endl;  continue;  }  }  } |

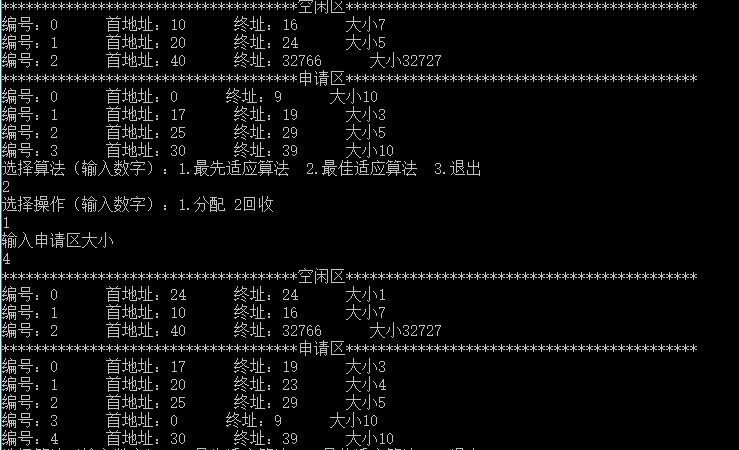
**4．遇到的主要问题和解决方法**

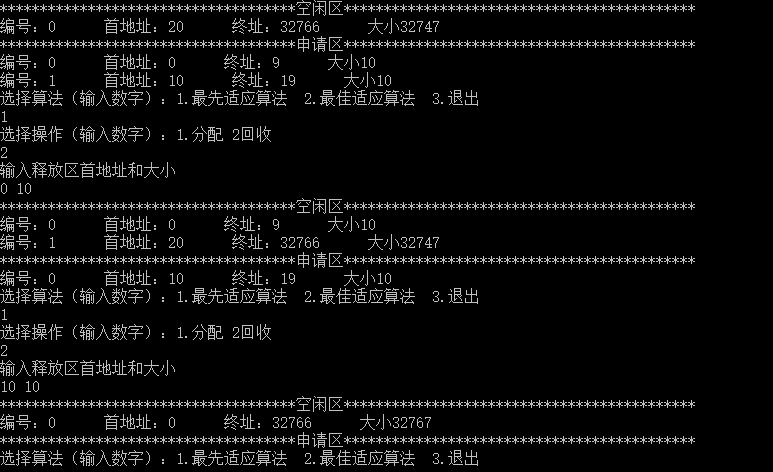
A．frist fit 和 best fit不能交替使用；将数据进行查找前进行排序，而不是在输出前进行排序。

B．回收时分区的判断。先将释放区插入空闲队列，然后便利查找是否有空闲的前分区，和后分区，如果有则和释放区合并。

1. **实验结果**







**五、实验总结**

通过本次实验，我学到了存储管理的最先适应算法和最佳适应算法，了解了最先适应算法和最佳适应算法的运行过程，掌握了算法的使用方法，提高了对算法的掌握程度。