int alloc(int tag)

{

int ID,size1;

cout<<"请输入作业号：";

cin>>ID;

cout<<"请输入所需内存大小：";

cin>>size1;

if (ID<=0 || size1<=0)

{

cout<<"输入错误！请输入正确的ID和请求大小："<<endl;

return 0;

}

if (tag==1)//采用首次适应算法

{

if(first\_fit(ID,size1))

{

cout<<"分配成功！"<<endl;

}

else cout<<"分配失败！"<<endl;

return 1;

}

else

{

if (best\_fit(ID,size1))

{

cout<<"分配成功！"<<endl;

}

else cout<<"分配失败！"<<endl;

return 1;

}

}

int first\_fit(int ID,int size)//首次适应算法

{

FNodeList temp=(FNodeList)malloc(sizeof(Free\_Node));

Free\_Node \*p=block\_first->next;

temp->date.ID=ID;

temp->date.size=size;

temp->date.flag=BUSY;

while(p)

{

if (p->date.flag==FREE && p->date.size==size)//请求大小刚好满足

{

p->date.flag=BUSY;

p->date.ID=ID;

return 1;

break;

}

if (p->date.flag==FREE && p->date.size>size)//说明还有其他的空闲区间

{

temp->next=p;

temp->front=p->front;

temp->date.address=p->date.address;

p->front->next=temp;

p->front=temp;

p->date.address=temp->date.address+temp->date.size;

p->date.size-=size;

return 1;

break;

}

p=p->next;

}

return 0;

}

int best\_fit(int ID,int size)//最佳适应算法

{

int surplus;//记录可用内存与需求内存的差值

FNodeList temp=(FNodeList)malloc(sizeof(Free\_Node));

Free\_Node \*p=block\_first->next;

temp->date.ID=ID;

temp->date.size=size;

temp->date.flag=BUSY;

Free\_Node \*q=NULL;//记录最佳位置

while(p)//遍历链表，找到第一个可用的空闲区间将他给q

{

if (p->date.flag==FREE&&p->date.size>=size)

{

q=p;

surplus=p->date.size-size;

break;

}

p=p->next;

}

while(p)//继续遍历，找到更加合适的位置

{

if (p->date.flag==FREE&&p->date.size==size)

{

p->date.flag=BUSY;

p->date.ID=ID;

return 1;

break;

}

if (p->date.flag==FREE&&p->date.size>size)

{

if (surplus>p->date.size-size)

{

surplus=p->date.size-size;

q=p;

}

}

p=p->next;

}

if (q==NULL)//如果没有找到位置

{

return 0;

}

else//找到了最佳位置

{

temp->next=q;

temp->front=q->front;

temp->date.address=q->date.address;

q->front->next=temp;

q->front=temp;

q->date.size=surplus;

q->date.address+=size;

return 1;

}

}

int free(int ID)//主存回收

{

Free\_Node \*p=block\_first->next;

while(p)

{

if (p->date.ID==ID)//找到要回收的ID区域

{

p->date.flag=FREE;

p->date.ID=FREE;

//判断P与前后区域关系

if (p->front->date.flag==FREE&&p->next->date.flag!=FREE)

{

p->front->date.size+=p->date.size;

p->front->next=p->next;

p->next->front=p->front;

}

if (p->front->date.flag!=FREE&&p->next->date.flag==FREE)

{

p->date.size+=p->next->date.size;

if(p->next->next)

{

p->next->next->front=p;

p->next = p->next->next;

}

else p->next=p->next->next;

}

if(p->front->date.flag==FREE&&p->next->date.flag==FREE)

{

p->front->date.size+=p->date.size+p->next->date.size;

if(p->next->next)

{

p->next->next->front=p->front;

p->front->next=p->next->next;

}

else p->front->next=p->next->next;

}

if(p->front->date.flag!=FREE&&p->next->date.flag!=FREE)

{//

}

break;

}

p=p->next;

}

cout<<"回收成功！"<<endl;

return 1;

}

void show()

{

cout<<"\*\*\*\*\*\*\*内存分配情况\*\*\*\*\*\*\*"<<endl;

Free\_Node \*p=block\_first->next;

while(p)

{

cout<<"分区号：";

if (p->date.ID==FREE)

cout<<"FREE"<<endl;

else cout<<p->date.ID<<endl;

cout<<"起始地址："<<p->date.address<<endl;

cout<<"内存大小："<<p->date.size<<endl;

cout<<"分区状态：";

if (p->date.flag==FREE)

cout<<"空闲"<<endl;

else

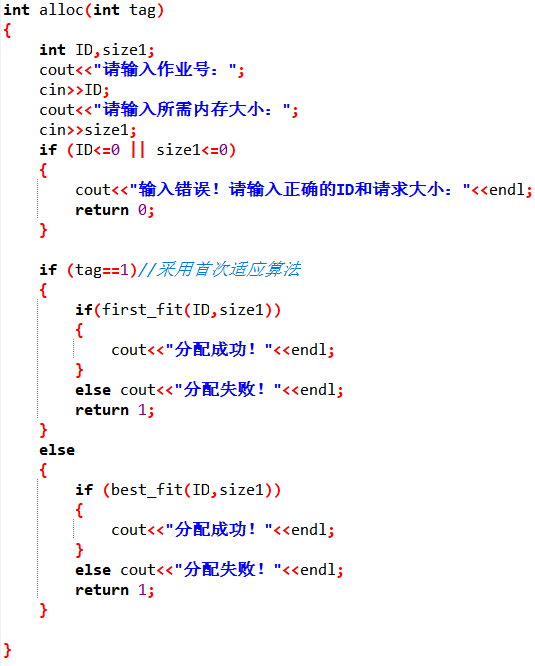
cout<<"已分配"<<endl;

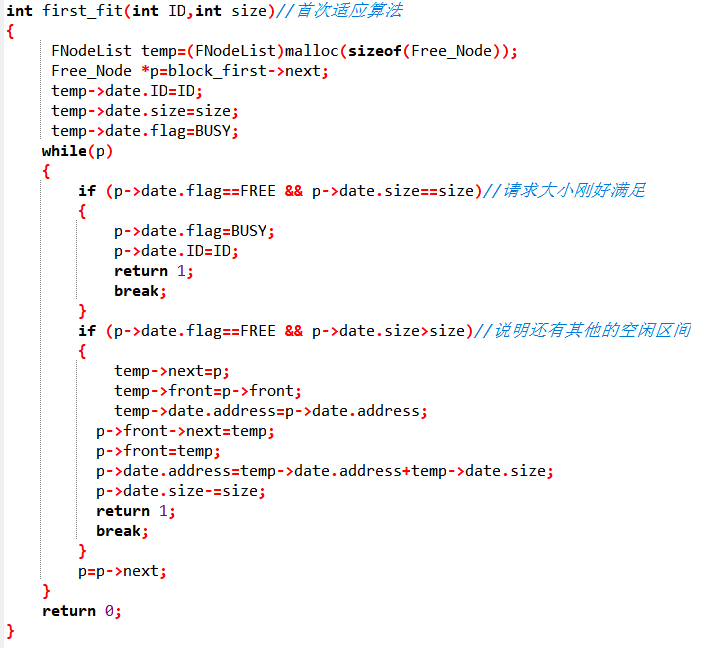
cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

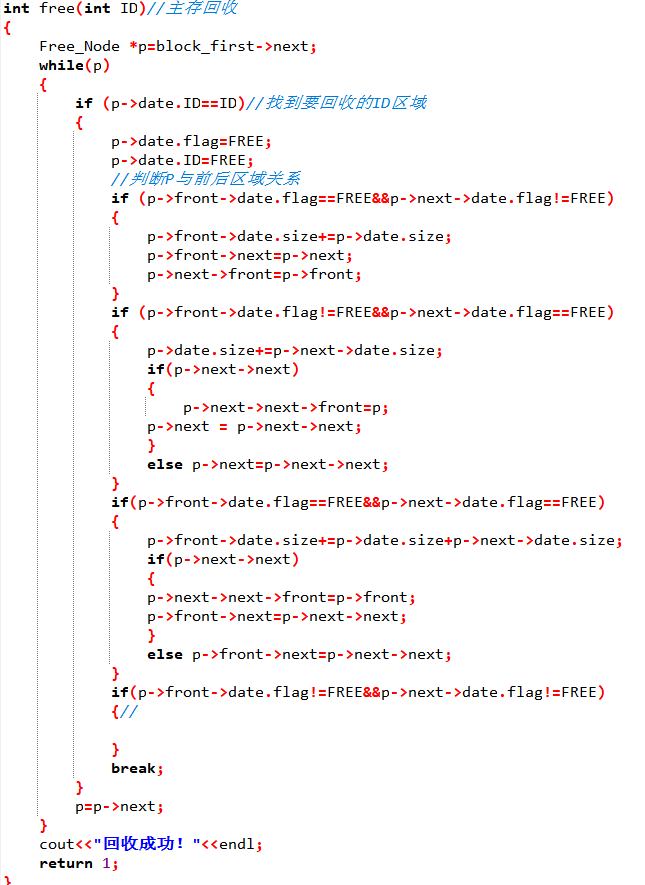
p=p->next;

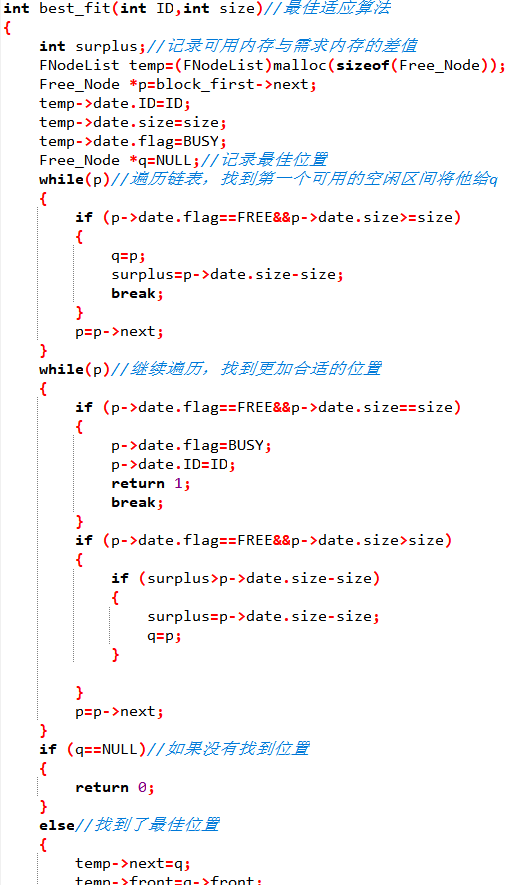
}

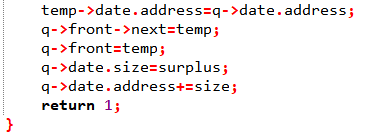
}

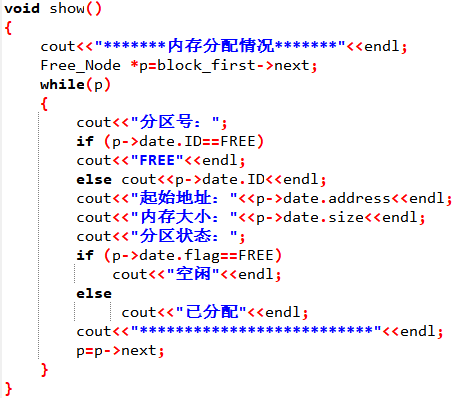












存储管理可以有效地对外部存储资源和内存进行管理，可以完成存储分配，存储共享，存储保护，存储扩充，地址映射等重要功能，对操作系统的性能有很重要的影响。首次适应算法和最佳适应算法是存储管理中两个十分重要的页面置换算法。

首次适应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。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，为以后到达的大作业分配大的内存空间创造了条件。但是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。

最佳适应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，该算法将所有的空闲区按大小排序后，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。这样，每次分配给文件的都是最合适该文件大小的分区，但是内存中留下许多难以利用的小的空闲区。

通过本次实验，上机代码模拟实现了三种动态分区分配算法，对操作系统内部的空闲分区分配方式有了更深刻的认识和感受。FF 首次适应算法在低址部分不断被划分，会留下很多难以利用的、很小的空闲碎片。BF 最佳适应算法似乎是最优的，但是可能会在存储器中留下许多难以利用的碎片。WF 最坏适应算法与最佳适应算法刚好相反，它会导致存储器中缺乏大的空闲分区，但是未必是最坏的，它产生碎片的可能性最小。