

## 实验 一 题目 进程调度 第      周星期

### 一、实验目的

用高级语言编写和调试一个进程调度程序，以加深对进程的概念及进程调度算法的理解。

### 二、实验内容和要求

编写并调试一个模拟的进程调度程序，采用“短进程优先”调度算法对五个进程进行调度。以加深对进程的概念及进程调度算法的理解。

每个进程有一个进程控制块（PCB）表示。进程控制块可以包含如下信息：进程名、优先数、到达时间、需要运行时间、已用 CPU 时间、进程状态等等。

进程的优先数及需要的运行时间可以事先人为地指定（也可以由随机数产生）。进程的到达时间为进程输入的时间。

进程的运行时间以时间片为单位进行计算。

每个进程的状态可以是就绪 W（Wait）、运行 R（Run）、或完成 F（Finish）三种状态之一。

就绪进程获得 CPU 后都只能运行一个时间片。用已占用 CPU 时间加 1 来表示。

如果运行一个时间片后，进程的已占用 CPU 时间已达到所需要的运行时间，则撤消该进程，如果运行一个时间片后进程的已占用 CPU 时间还未达所需要的运行时间，也就是进程还需要继续运行，此时应将进程的优先数减 1（即降低一级），然后把它插入就绪队列等待 CPU。

每进行一次调度程序都打印一次运行进程、就绪队列、以及各个进程的 PCB，以便进行检查。重复以上过程，直到所要进程都完成为止。

### 三、实验主要仪器设备和材料

#### 实验环境

硬件环境：个人台式机 Microsoft Windows7

软件环境：C 语言编程环境，VS 2010

### 四、实验原理及设计方案

#### 1、实验原理

进程作为基本数据处理单元，需要对进程的基本信息进行相关的描述。进程的基本

信息包括进程进程名、到达的时间、预计的进程运行时间、进程开始运行时间、进程仍需运行的时间、进程完成的时间、进程运行的次数等。在此，可以定义一个结构体链表来储存进程信息。并在此基础上进行其他操作

短进程优先调度算法 对强占式短进程优先调度算法而言，其本质特征便是按进程的预计运行时间长短进行排序，先执行短进程。若内存中运行的进程优先级比就绪队列中的某进程优先级低（即运行的进程预计运行时间比就绪队列中的某进程长），此运行的进程让出内存并进入就绪队列，优先级更高的短进程强占内存资源并运行直到结束或者遇到优先级更高的进程强占为止。

## 2、设计方案

```
struct pcb {  
    char name[10];    //进程名  
    char state;        //进程状态 w: 等待 r: 运行  
    int queue;        //进程队列 1: 2: 3  
    int priority;    // 数字越小优先级越高  
    int needtime;    //需运行时间  
    int runtime;    //已经运行时间  
    pcb *link;        //进程队列下一指针  
}*ready=NULL,*run;  
  
typedef struct pcb PCB;  
  
//分别设置三条就绪队列头指针和尾指针  
PCB *First=NULL,*Second=NULL,*Third=NULL,*end1=NULL,*end2=NULL,*end3=NULL;  
  
//为三条就绪队列设置三个时间片: 1: 2: 3  
int time[3];
```

## 3、程序流程图

## 五、实验结果及分析

短进程优先调度算法.exe,出现进程初始化界面,输入初始进程数目,接着必须按提示输入所有初始化进程,如下图所示:



```
-----进程调度-----
-----张大川3110005966-----
输入进程个数:
2
输入进程名:      aaa
输入进程 0 的到达时间和服务时间:
0
4
输入进程名:      2
输入进程 1 的到达时间和服务时间:
1
2
第 1 个调度进程:      进程名: aaa      到达时间: 0      服务时间: 4
完成时间<下一个进程开始时间>: 4 周转时间: 4      带权周转时间: 1
第 2 个调度进程:      进程名: 2      到达时间: 1      服务时间: 2
完成时间<下一个进程开始时间>: 6 周转时间: 5      带权周转时间: 3
////////////////////////////////平均周转时间: 6.5
////////////////////////////////平均带权周转时间: 1.3
请按任意键继续. . .
```

## 六、调试总结及心得体会

第一次实验相对比较简单,依照教材的例子,照设计思想,不断调试不断完善,总算完成了,加深了进程调度的理解,搞清楚了各个调度的优缺点,收获不少。

## 七、源程序代码:

```
#include<iostream>
#include<string>
#include<stdio.h>
using namespace std;
//进程结构
class JCB
{
public:
    string name;
    int cometime;
    int sertime;
    bool run;
    JCB * next;
public:
    JCB(string tn="", int tc=-1, int ser=0, bool tr=false, JCB *nex=NULL)
        :name(tn), cometime(tc), sertime(ser), run(tr), next(nex)
    {}
    void show()
```

```

    {
        cout<<"\t 进 程 名 : " <<name<<"\t 到 达 时 间 : " <<cometime<<"\t 服 务 时 间 : " <<sertime<<endl;
    }
};

```

//进程的排序，按照进程的提交时间先后排序

```

void sort(JCB *head)
{
    JCB *q,*s,*p=head->next;
    head->next=NULL;
    while(p!=NULL)
    {
        q=p->next;
        if(head->next==NULL)
        {
            p->next=head->next;
            head->next=p;
        }
        else
        {
            s=head;
            while(p->cometime>s->next->cometime)
            {
                s=s->next;
            }
            p->next=s->next;
            s->next=p;
        }
        p=q;
    }
}

```

//辅助函数，查找JCB

```

JCB * getJCB(JCB *q, JCB *p)
{
    JCB *find=q;
    int minsertime=q->sertime;
    while(q!=p)
    {
        if(q->sertime<minsertime&&q->run==false)
        {
            minsertime=q->sertime;
            find=q;
        }
    }
}

```

```

        q=q->next;
    }
    return find;
}
//shadule调度函数
//
void shadule(JCB *head, int n)
{
    JCB *s,*q,*p=head->next;
    JCB *sp;
    int sum_rolltime=p->sertime-p->cometime;
    int waittime=p->cometime;
    int sum_rollsertime=p->sertime;
    int i=1;
    sp=p;
    for(int num=0;num<n;num++)
    {
        while(p->run==true)
            p=p->next;
        q=p;
        while(p!=NULL&& p->cometime<=waittime)
            p=p->next;
        s=getJCB(q, p);
        cout<<"第 "<<i++<<" 个调度进程: ";
        s->show();
        s->run=true;
        waittime+=s->sertime;
        cout<<" 完成时间 (下一个进程开始时间): "<<waittime<<"\t 周转时间: "
        <<waittime-(s->cometime)
        <<"\t带权周转时间: "<<waittime/float(s->sertime)<<endl;
        sum_rolltime+=(waittime-(s->cometime));
        sum_rollsertime+=s->sertime;
        p=sp;
    }
    cout<<"////////////////////平均周转时间: "<<sum_rolltime/float(n)<<endl;
    cout<<"//////////////////// 平 均 带 权 周 转 时 间 : "
    <<sum_rolltime/float(sum_rollsertime)<<endl;
    system("pause");
}

void main()
{

```

```

int num;
string name;
int cometime,sertime;
cout<<" -----进程调度-----"<<endl;
cout<<" -----张大川|3110005966-----"<<endl;

cout<<"      输入进程个数: "<<endl;
cin>>num;
JCB *head=new JCB();

for(int i=0;i<num;++i){
    cout<<"输入进程名:\t";
    cin>>name;
    cout<<"输入进程 "<<i<<" 的到达时间和服务时间:"<<endl;
    cin>>cometime>>sertime;
    JCB *p=new JCB(name,cometime,sertime,false,head->next);
    head->next=p;
}
sort(head);
shadule(head,num);
}

```

实验 二 题目 作业调度 第      周星期     

### 一、实验目的

用高级语言编写和调试一个或多个作业调度的模拟程序，以加深对作业调度算法的理解。

### 二、实验内容和要求

1. 写并调试一个单道处理系统的作业等待模拟程序。
2. 作业等待算法：分别采用先来先服务（FCFS）、响应比高者优先（HRN）的调度算法。
3. 由于在单道批处理系统中，作业一投入运行，它就占有计算机的一切资源直到作业完成为止，因此调度作业时不必考虑它所需要的资源是否得到满足，它所占用的 CPU 时限等因素。
4. 每个作业由一个作业控制块 JCB 表示，JCB 可以包含如下信息：作业名、提交时间、所需的运行时间、所需的资源、作业状态、链指针等等。作业的状态可以是等待 W(Wait)、运行 R(Run) 和完成 F(Finish) 三种状态之一。每个作业的最初状态总是等待 W。
5. 对每种调度算法都要求打印每个作业开始运行时刻、完成时刻、周转时间、带权周转时间，以及这组作业的平均周转时间及带权平均周转时间。

### 三、实验主要仪器设备和材料

#### 实验环境

硬件环境：个人台式机 Microsoft Windows 7

软件环境：C 语言编程环境，vs 2010

## 四、实验原理及设计方案

### 1、实验原理

先来先服务算法：是按照作业进入输入井的先后次序来挑选作业，先进入输入井的作业优先被挑选，当系统中现有的尚未分配的资源不能满足先进入输入井的作业，那么顺序挑选后面的作业。FCFS 算法简单易行，但性能却不大好。

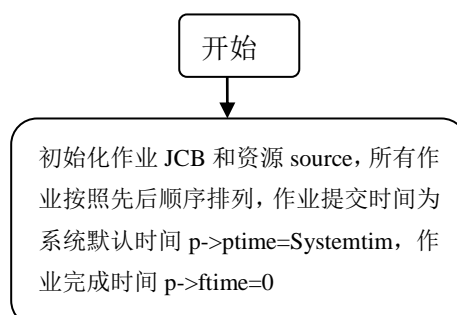
短作业优先算法：总是按照作业要求运行时间来选择作业，每次挑选要求作业执行时间短且资源要求能满足的作业优先分派处理机，通常后来的短作业不抢先正在执行的作业。SJF 改善平均周转时间和平均带权周转时间，缩短作业的等待时间，提高系统的吞吐量，但对长作业非常不利。

### 设计方案

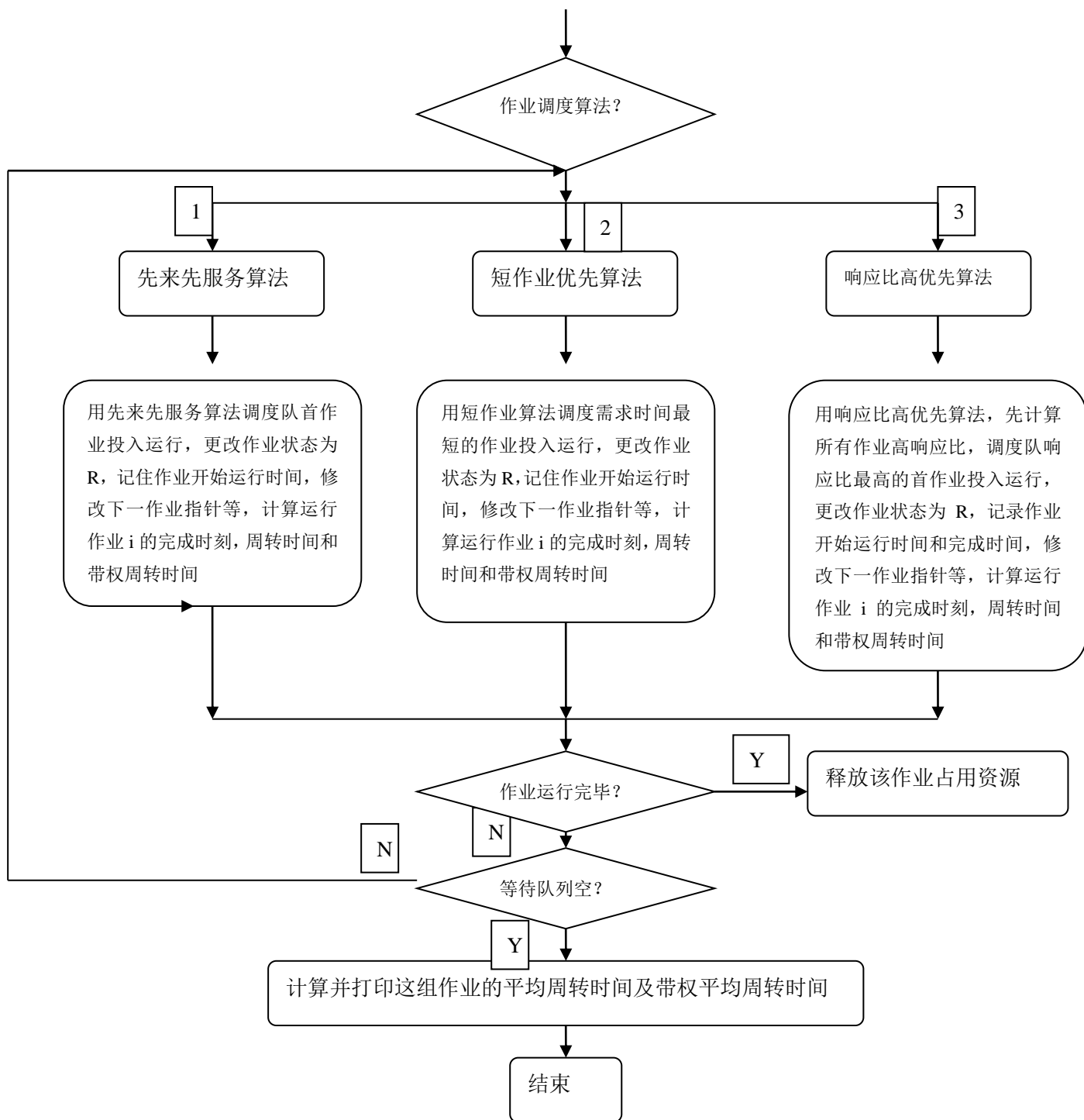
```
struct source{                                /*定义资源需求结构*/
    char    memery[5];                        /*主存需求*/
    int     machine;                          /*磁带机数量*/
};

struct jcb {                                  /* 定义作业控制块 PCB */
    char    name[10];
    char    state;                            /* 状态 */
    double  super;                            /* 响应比优先权 */
    int     ntime;                            /* 需要运行时间 */
    int     rtime;                            /* 开始运行时间 */
    int     ptime;                            /*提交时间*/
    int     ftime;                            /*完成时间*/
    source *needsources;                      /*资源需求链*/
    struct jcb* link;                         /* 下一个作业控制块的地址 */
}*ready=NULL,*run,*p;
```

### 2、程序流程图

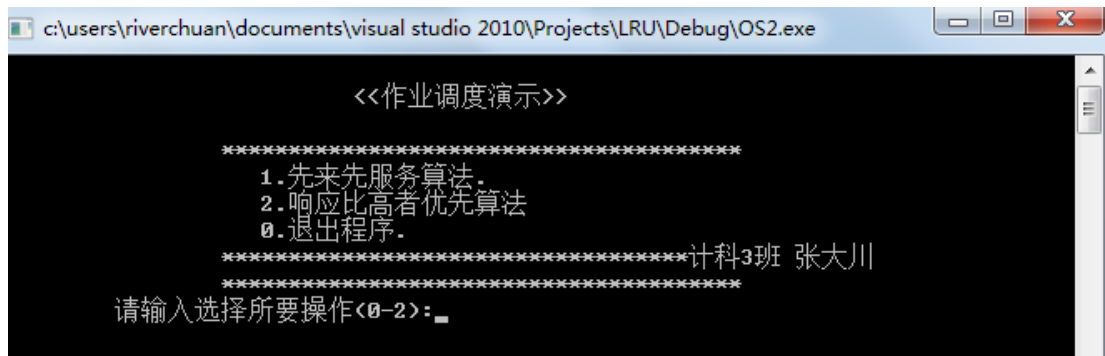




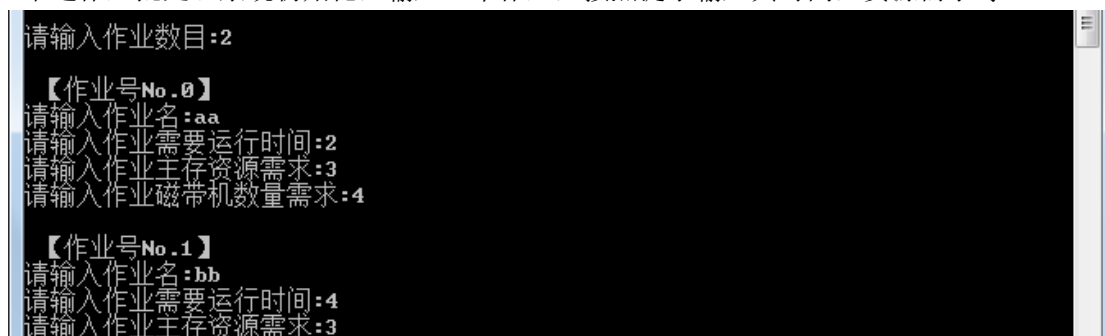


## 五、实验结果及分析

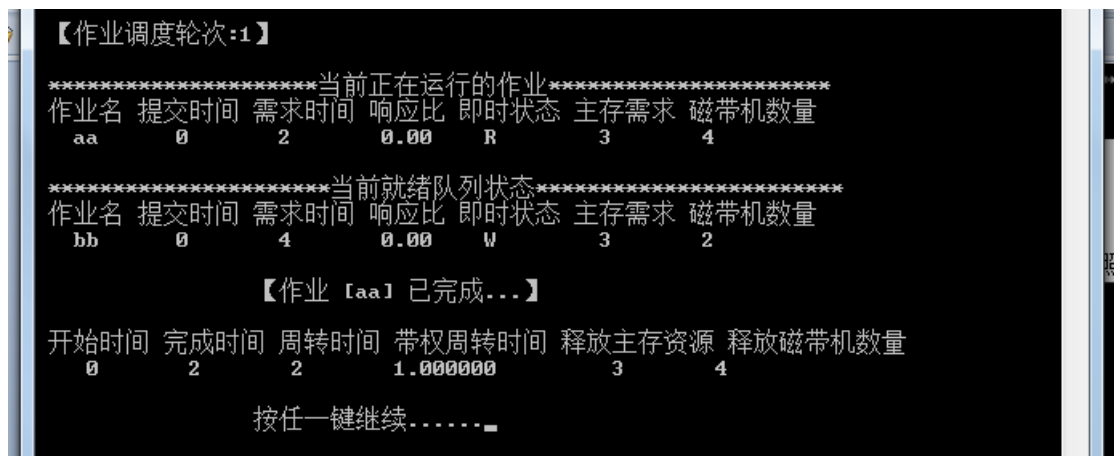
先选取《先来先服务算法》进行单道作业批处理



单道作业批处理系统初始化，输入三个作业，按照提示输入其时间，资源需求等：



第一轮作业调度，如图显示了每个作业提交时间、需求时间、响应比、即时状态、主存需求、磁带机数量等，每次调度完毕输出作业完成时间、周转时间、带权周转时间、释放的资源：



同样，第二轮调度，作业 cj11 完成，如下显示：

```
开始时间 完成时间 周转时间 带权周转时间 释放主存资源 释放磁带机数量
0 2 2 1.000000 3 4

按任一键继续.....h

【作业调度轮次:2】

*****当前正在运行的作业*****
作业名 提交时间 需求时间 响应比 即时状态 主存需求 磁带机数量
bb 0 4 0.00 R 3 2

*****当前就绪队列状态*****
【作业 [bb] 已完成...】

开始时间 完成时间 周转时间 带权周转时间 释放主存资源 释放磁带机数量
2 6 6 1.500000 3 2

按任一键继续.....
```

全部作业运行完毕，计算总的平均周转时间和带权周转时间：

```
<<全部作业已经完成...>>

作业数量 平均周转时间 带权周转时间
3 31.666667 9.772727
```

选取《响应比高者优先算法》进行单道批作业处理

```
c:\users\riverchuan\documents\visual studio 2010\Projects\LRU\Debug\OS2.exe

<<作业调度演示>>

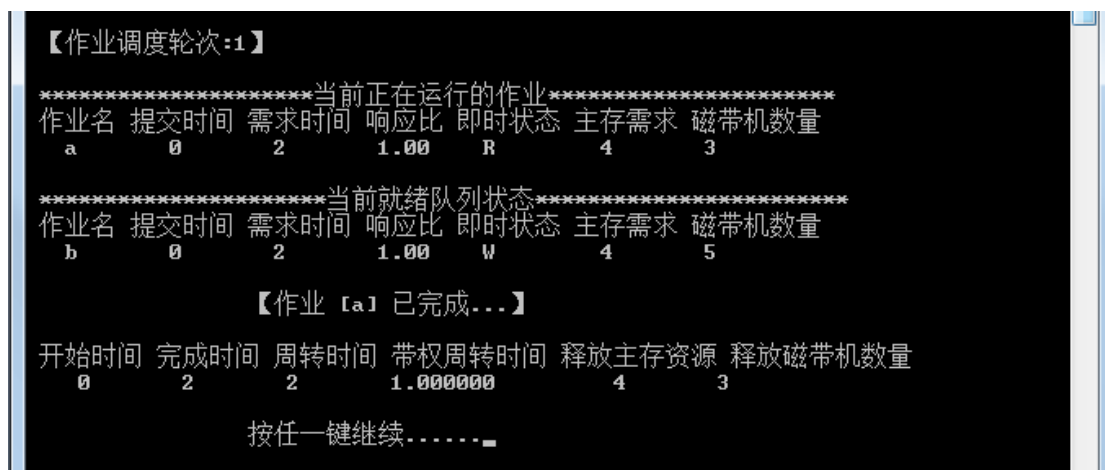
*****
1.先来先服务算法.
2.响应比高者优先算法
0.退出程序.
*****计科3班 张大川
*****

请输入选择所要操作(0-2):
```

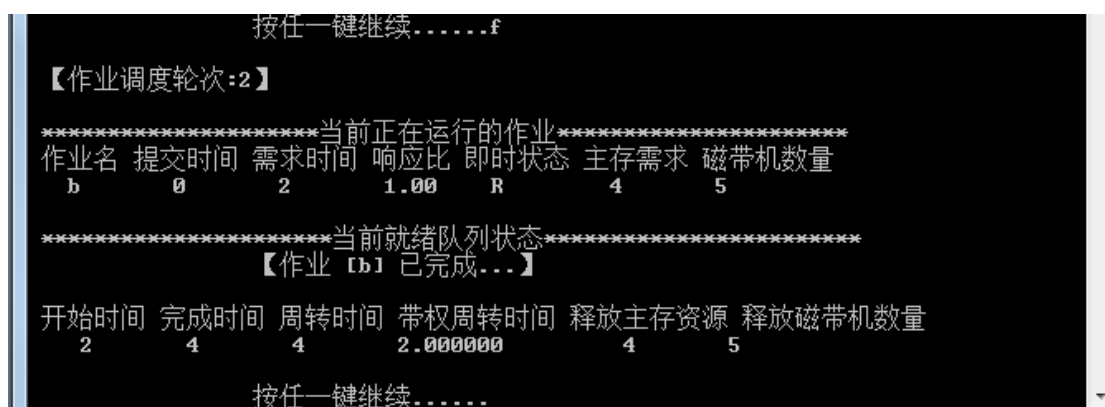
单道作业批处理系统初始化，输入三个作业，按照提示输入其时间，资源需求等：



第一轮作业调度，如图显示了每个作业提交时间、需求时间、响应比、即时状态、主存需求、磁带机数量等，每次调度完毕输出作业完成时间、周转时间、带权周转时间、释放的资源：



同样，第二轮调度，作业 b 完成，输出作业完成时间、周转时间、带权周转时间、释放的资源：



全部作业运行完毕，计算总的平均周转时间和带权周转时间：

<<全部作业已经完成...>>		
作业数量	平均周转时间	带权周转时间
3	37.333333	3.099715

## 六、调试总结

通过这次实验,我进一步加深了对各种调度算法的理解与应用，知识层面亦得到了升华。

## 七、比较各个算法的优缺点。

优缺点：

(1) 先来先服务算法：比较有利于长作业(进程),而不利于短作业。

(2) 最短作业优先算法：比较利于短作业，不利于长作业。SJF 算法能有效地降低作业的平均等待时间，提高系统吞吐量。SJF 算法完成未考虑作业的紧迫程度，因而又不能保证紧迫性作业（进程）会被及时处理。由于作业的长短只是根据用户提供的估计执行时间而定的，而用户又可能会有意或无意地缩短其作业的估计运行时间，致使该算法不一定能真正做到知作业优先调度。

## 八、源程序代码：

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define getpch(type) (type*)malloc(sizeof(type))

int Select;
int Systemtime = 0;          /*系统初始时间*/
int JCBnum = 0;              /*总作业数*/
int JCBtime = 0;             /*总周转时间*/
double JCBtotaltime = 0;     /*总带权周转时间*/

struct source{               /*定义资源需求结构*/
    char    memery[5];       /*主存需求*/
    int     machine;         /*磁带机数量*/
};

struct jcb {                 /* 定义作业控制块PCB */
    char    name[10];
    char    state;           /* 状态 */
    double  super;           /* 响应比优先权 */
    int     ntime;           /* 需要运行时间 */
    int     rtime;           /* 开始运行时间 */
};
```

```

    int    ptime;          /*提交时间*/
    int    ftime;          /*完成时间*/
    source *needsources;   /*资源需求链*/
    struct jcb* link;      /* 下一个作业控制块的地址 */
} *ready=NULL, *run, *p;

typedef struct jcb JCB;
typedef struct source SOURCE;

void display()
{
    int c;
    do
    {
        system("cls");
        printf("\n\t\t\t\t\t <<作业调度演示>>\n");
        printf("\n\t\t\t\t\t*****\t\t");
        printf("\n\t\t\t\t\t 1. 先来先服务算法.");
        printf("\n\t\t\t\t\t 2. 响应比高者优先算法");
        printf("\n\t\t\t\t\t 0. 退出程序.");
        printf("\n");
        printf("\t\t*****计科3班 张大川");
        printf("\n\t\t*****\t\t\n");
        printf("\t请输入选择所要操作(0-2):");
        scanf("%d", &c);
        system("cls");
    } while (c<0 || c>3);
    switch(c)
    {
        case 0:
            exit(0);
            break;
        case 1:
            Select=1;

            break;
        /* case 2:
            Select=2;*/

            break;
        case 2:
            Select=2;
            break;
    }
}

```

```

        default:
            break;
    }
}

void sort() /* 建立对进程进行优先级排列函数*/
{
    JCB *first, *second, *temp;
    int insert=0;
    switch(Select)
    {

    case 1:                /*先来先去服务算法*/
        if(ready==NULL)    /*队首空插入队首*/
        {
            p->link=ready;
            ready=p;
        }
        else                /*否则插入队尾*/
        {
            first=ready;
            second=ready->link;
            while(second!=NULL)
            {
                first=first->link;
                second=second->link;
            }
            first->link=p;
        }
        break;

    /* case 2:

        if((ready==NULL) || ((p->ntime)<(ready->ntime))) /*队首
        {
            p->link=ready;
            ready=p;
        }
        else                // 往后搜索适当的位置插
        {
            first=ready;
            second=first->link;
            while(second!=NULL) // 插入队伍中间

```

```

        {
            if((p->ntime)<(second->ntime))
            {
                p->link=second;
                first->link=p;
                second=NULL;
                insert=1;
            }
            else
            {
                first=first->link;
                second=second->link;
            }
        }
        if(insert==0) first->link=p;    // 插入队尾
    }
    break;    */

case 2:                /*响应比高者优先算法*/
                        /*响应比=(等待时间+需求时间)/需求时间*/

    temp=ready;
    p->super=(double) (Systemtime-p->ptime+p->ntime)/(double) (p->ntime);    /*计算插入队列作业优先权*/
    while(temp)                /* 计算就绪队列作业优先权 */

    {

temp->super=(double) (Systemtime-temp->ptime+temp->ntime)/(double) (temp->ntime);
        temp=temp->link;
    }

    /*响应比越大优先权越高*/
    if((ready==NULL) || ((p->super)>(ready->ntime)))    /*队首*/
    {
        p->link=ready;
        ready=p;
    }
else                /* 往后搜索适当的位置插入*/
    {
        first=ready;
        second=first->link;
        while(second!=NULL)    /*插入队伍中间*/
        {

```



```

        if((p->super)>(second->super))
        {
            p->link=second;
            first->link=p;
            second=NULL;
            insert=1;
        }
        else
        {
            first=first->link;
            second=second->link;
        }
    }
    if(insert==0) first->link=p;    /* 插入队尾*/
}

break;

default:
    break;
}
}

```

```

void input() /* 建立作业输入函数*/
{
    int i,num;
    SOURCE *q;
    printf("\n\n请输入作业数目:");
    scanf("%d",&num);
    JCBnum+=num;                /*总作业数*/
    for(i=0;i<num;i++)
    {
        printf("\n 【作业号No. %d】 \n",i);
        p=getpch(JCB);
        q=getpch(SOURCE);
        printf("请输入作业名:");
        scanf("%s",p->name);
        getchar();
        printf("请输入作业需要运行时间:");
        scanf("%d",&p->ntime);
        getchar();
        printf("请输入作业主存资源需求:");
        scanf("%s",q->memery);
        getchar();
    }
}

```

```

        printf("请输入作业磁带机数量需求:");
        scanf("%d",&q->machine);
        p->needsources=q;
        p->ptime=Systemtime;          /*作业提交时间为系统默认时间*/
        p->super=0;
        p->ftime=0;                    /*作业完成时间*/
        p->state='W';
        p->link=NULL;
        sort();
    }
}

```

```

void disp(JCB *pr) /*建立作业显示函数*/
{
    printf("\n 作业名 提交时间 需求时间 响应比 即时状态 主存需求 磁带机数量 \n");
    printf("    %s\t",pr->name);
    printf("    %d\t",pr->ptime);
    printf("    %d\t",pr->ntime);
    printf("    %.2f\t",pr->super);
    printf("    %c\t",pr->state);
    printf("    %s\t",pr->needsources->memery);
    printf("    %d\t",pr->needsources->machine);
    printf(" \n");
}

```

```

void check() /* 建立作业查看函数 */
{
    JCB* pr;
    printf("\n *****当前正在运行的作业*****"); /*显示
当前运行作业*/
    disp(run);
    pr=ready;
    printf("\n *****当前就绪队列状态*****"); /*显
示就绪队列状态*/
    while(pr!=NULL)
    {
        disp(pr);
        pr=pr->link;
    }
}

```

```

void destroy() /*建立作业撤销函数*/
{
    JCB *pr=run;
    run->ftime=Systemtime;
    printf("\n\t\t 【作业 [%s] 已完成...】 \n",run->name);
    printf("\n 开始时间 完成时间 周转时间 带权周转时间 释放主存资源 释放磁带机数量\n");
    printf("    %d\t",run->rtime);
    printf("    %d\t",run->ftime);
    printf("    %d\t",Systemtime-run->ptime);
    printf("    %f\t", (double) (Systemtime-run->ptime)/(double)run->ntime);
    printf("    %s\t",run->needsources->memery);
    printf("    %d\t",run->needsources->machine);
    printf(" \n");
    JCBtime += Systemtime-run->ptime;                /*更新总周转时间*/
    JCBtotaltime += JCBtime/(double)run->ntime;        /*更新总带权周转时间*/
    run=run->link;
    free(pr);
    pr=NULL;
}

```

```

void running() /* 建立作业运行函数*/
{
    run->rtime=Systemtime;    /*作业开始时间为系统当前时间*/
    Systemtime += run->ntime; /*更新系统时间*/
    destroy();
}

```

```

void main() /*主函数*/
{
    int h=0;
    char ch;
    display();
    input();
    Systemtime=0;    /*初始系统时间*/
    while(ready!=NULL)
    {
        ch=getchar();
        h++;
        printf("\n 【作业调度轮次:%d】 \n",h);
        run=ready;
        ready=ready->link;
        run->link=NULL;
    }
}

```

```
run->state='R';
check();
running();
printf("\n\t\t 按任一键继续.....");
ch=getchar();
}

printf("\n\n\t\t <<全部作业已经完成...>>\n");
printf("\n\t\t 作业数量 平均周转时间 带权周转时间 \n");
printf("\t\t %d\t", JCBnum);
printf("    %f\t", (double)JCBtime/(double)JCBnum);
printf("    %f\t", (double)JCBtotaltime/(double)JCBnum);
printf(" \n");
ch=getchar();
}
```

实验 三 题目 动态分区分配方式的模拟 第    周星期   

### 一、实验目的

了解动态分区分配方式中的数据结构和分配算法,并进一步加深对动态分区存储管理方式及其实现过程的理解

### 二、实验内容和要求

(1) 用 C 语言分别实现采用首次适应算法和最佳适应算法的动态分区分配过程和回收过程。其中,空闲分区通过空闲分区链(表)来管理;在进行内存分配时,系统优先使用空闲区低端的空间。

(2) 假设初始状态下,可用的内存空间为 640KB,并有下列的请求序列:

- 作业 1 申请 130KB
- 作业 2 申请 60KB
- 作业 3 申请 100KB
- 作业 2 释放 60KB
- 作业 4 申请 200KB
- 作业 3 释放 100KB
- 作业 1 释放 130KB
- 作业 5 申请 140KB
- 作业 6 申请 60KB
- 作业 7 申请 50KB
- 作业 8 申请 60KB

请分别采用首次适应算法和最佳适应算法进行内存的分配和回收,要求每次分配和回收后显示出空闲内存分区链的情况。

### 三、实验主要仪器设备和材料

#### 实验环境

硬件环境：个人台式机 Windows7

软件环境：C 语言编程环境, vs 2010

### 四、实验设计方案及原理

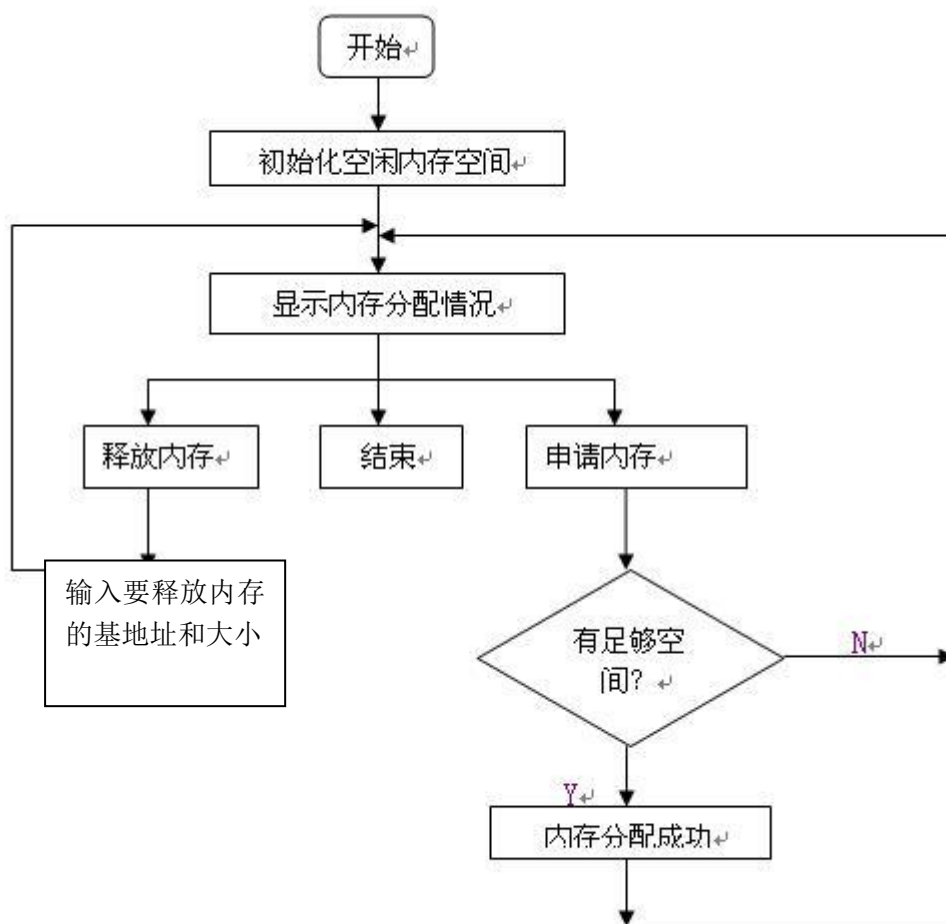
#### 1、循环首次适应算法

在该算法中，把主存中所有空闲区按其物理地址递增的次序排列。在为作业分配存储空间时，从上次找到的空闲分区的下一个空闲分区开始查找，直到找到第一个能满足要求的空闲区，从中划出与请求的大小相等的存储空间分配给作业，余下的空闲区仍留在空闲区表或链中。

#### 1、实验步骤

- (1) 初始化空闲分区；
- (2) 反复对现有的空闲分区进行进程创建和撤消，即内存分配和回收；
- (3) 退出。

#### 3、流程图



## 五、实验结果及分析

打开程序，运行界面如下：

```

c:\users\riverchuan\documents\visual studio 2010\Projects\LRU\Debug\text1.exe
-----动态分区分配方式-----
-----张大川3110005966-----
Please select a number(1,2,0)
1--首次适应算法
2--最佳适应算法
0--中止程序
1_
  
```

The screenshot shows the command-line interface of the program. The title bar indicates the file path: c:\users\riverchuan\documents\visual studio 2010\Projects\LRU\Debug\text1.exe. The program displays a menu for selecting a memory allocation algorithm: 1 for First Fit (首次适应算法), 2 for Best Fit (最佳适应算法), and 0 to terminate the program (中止程序). The user has entered '1'.

按照题目所给的作业请求序列，对程序进行运行，以下为运行结果：

首次适应算法部分：

1. 作业 1、2、3 分别申请内存空间

This is an example for FF:  
 作业 1 申请 130 k 的内存空间

内存链的状态为:  
 分配的地址为: 0    分配的空间: 130k    运行的作业号:1    内存已分配

作业 2 申请 60 k 的内存空间

内存链的状态为:  
 分配的地址为: 0    分配的空间: 130k    运行的作业号:1    内存已分配  
 分配的地址为: 130    分配的空间: 60k    运行的作业号:2    内存已分配

作业 3 申请 100 k 的内存空间

内存链的状态为:  
 分配的地址为: 0    分配的空间: 130k    运行的作业号:1    内存已分配  
 分配的地址为: 130    分配的空间: 60k    运行的作业号:2    内存已分配  
 分配的地址为: 190    分配的空间: 100k    运行的作业号:3    内存已分配

## 2. 作业 1 释放空间，作业 4、5 申请空间

作业 1 释放 130k 的空间

内存链的状态为:  
 分配的地址为: 0    分配的空间: 130k    运行的作业号:1    内存空闲  
 分配的地址为: 130    分配的空间: 60k    运行的作业号:2    内存已分配  
 分配的地址为: 190    分配的空间: 100k    运行的作业号:3    内存已分配

作业 4 申请 200 k 的内存空间

内存链的状态为:  
 分配的地址为: 0    分配的空间: 130k    运行的作业号:1    内存空闲  
 分配的地址为: 130    分配的空间: 60k    运行的作业号:2    内存已分配  
 分配的地址为: 190    分配的空间: 100k    运行的作业号:3    内存已分配  
 分配的地址为: 290    分配的空间: 200k    运行的作业号:4    内存已分配

作业 5 申请 140 k 的内存空间

内存链的状态为:  
 分配的地址为: 0    分配的空间: 130k    运行的作业号:1    内存空闲  
 分配的地址为: 130    分配的空间: 60k    运行的作业号:2    内存已分配  
 分配的地址为: 190    分配的空间: 100k    运行的作业号:3    内存已分配  
 分配的地址为: 290    分配的空间: 200k    运行的作业号:4    内存已分配  
 分配的地址为: 490    分配的空间: 140k    运行的作业号:5    内存已分配

## 3. 作业 4 释放空间。此时内存不足，不能分配为作业 6、7、8 分配空间。



```
没有足够的内存空间为作业6分配

内存链的状态为:
分配的地址为: 0    分配的空间: 130k    运行的作业号:1    内存空闲
分配的地址为: 130  分配的空间: 60k     运行的作业号:2    内存已分配
分配的地址为: 190  分配的空间: 100k    运行的作业号:3    内存已分配
分配的地址为: 290  分配的空间: 200k    运行的作业号:4    内存空闲
分配的地址为: 490  分配的空间: 140k    运行的作业号:5    内存已分配

没有足够的内存空间为作业7分配

内存链的状态为:
分配的地址为: 0    分配的空间: 130k    运行的作业号:1    内存空闲
分配的地址为: 130  分配的空间: 60k     运行的作业号:2    内存已分配
分配的地址为: 190  分配的空间: 100k    运行的作业号:3    内存已分配
分配的地址为: 290  分配的空间: 200k    运行的作业号:4    内存空闲
分配的地址为: 490  分配的空间: 140k    运行的作业号:5    内存已分配

没有足够的内存空间为作业8分配

内存链的状态为:
分配的地址为: 0    分配的空间: 130k    运行的作业号:1    内存空闲
分配的地址为: 130  分配的空间: 60k     运行的作业号:2    内存已分配
分配的地址为: 190  分配的空间: 100k    运行的作业号:3    内存已分配
分配的地址为: 290  分配的空间: 200k    运行的作业号:4    内存空闲
```

最佳适应算法部分:

```
c:\users\riverchuan\documents\visual studio 2010\Projects\LRU\Debug\text1.exe
-----动态分区分配方式-----
-----张大川3110005966-----
Please select a number(1,2,0)
1--首次适应算法
2--最佳适应算法
0--中止程序
1_
```

1. 从首先适应算法退出, 按照最佳适应算法, 为作业 2 申请内存空间, 释放作业 1 的空间, 为作业 3 申请空间。

```

This is an example for optimist method;
作业 1申请130 k的内存空间

内存链的状态为:
分配的地址为: 0    分配的空间: 130k    运行的作业号:1    内存已分配

作业 2申请60 k的内存空间

内存链的状态为:
分配的地址为: 130    分配的空间: 60k    运行的作业号:2    内存已分配
分配的地址为: 0    分配的空间: 130k    运行的作业号:1    内存已分配

作业 1释放 130k 的空间

内存链的状态为:
分配的地址为: 130    分配的空间: 60k    运行的作业号:2    内存已分配
分配的地址为: 0    分配的空间: 130k    运行的作业号:1    内存空闲

作业 3申请100 k的内存空间

内存链的状态为:
分配的地址为: 130    分配的空间: 60k    运行的作业号:2    内存已分配
分配的地址为: 190    分配的空间: 100k    运行的作业号:3    内存已分配
分配的地址为: 0    分配的空间: 130k    运行的作业号:1    内存空闲

```

2. 为作业 4、5 申请空间。

```

作业 4申请200 k的内存空间

内存链的状态为:
分配的地址为: 130    分配的空间: 60k    运行的作业号:2    内存已分配
分配的地址为: 190    分配的空间: 100k    运行的作业号:3    内存已分配
分配的地址为: 0    分配的空间: 130k    运行的作业号:1    内存空闲
分配的地址为: 290    分配的空间: 200k    运行的作业号:4    内存已分配

作业 5申请140 k的内存空间

内存链的状态为:
分配的地址为: 130    分配的空间: 60k    运行的作业号:2    内存已分配
分配的地址为: 190    分配的空间: 100k    运行的作业号:3    内存已分配
分配的地址为: 0    分配的空间: 130k    运行的作业号:1    内存空闲
分配的地址为: 490    分配的空间: 140k    运行的作业号:5    内存已分配
分配的地址为: 290    分配的空间: 200k    运行的作业号:4    内存已分配

```

3. 作业 5、6、7 分别申请空间

```
c:\users\riverchuan\documents\visual studio 2010\Projects\LR0\Debug\text1.exe

作业 4申请200 k的内存空间

内存链的状态为:
分配的地址为: 130  分配的空间: 60k  运行的作业号:2  内存已分配
分配的地址为: 190  分配的空间: 100k  运行的作业号:3  内存已分配
分配的地址为: 0  分配的空间: 130k  运行的作业号:1  内存空闲
分配的地址为: 290  分配的空间: 200k  运行的作业号:4  内存已分配

作业 5申请140 k的内存空间

内存链的状态为:
分配的地址为: 130  分配的空间: 60k  运行的作业号:2  内存已分配
分配的地址为: 190  分配的空间: 100k  运行的作业号:3  内存已分配
分配的地址为: 0  分配的空间: 130k  运行的作业号:1  内存空闲
分配的地址为: 490  分配的空间: 140k  运行的作业号:5  内存已分配
分配的地址为: 290  分配的空间: 200k  运行的作业号:4  内存已分配

没有足够的内存空间为作业6分配

内存链的状态为:
分配的地址为: 130  分配的空间: 60k  运行的作业号:2  内存已分配
分配的地址为: 190  分配的空间: 100k  运行的作业号:3  内存已分配
分配的地址为: 0  分配的空间: 130k  运行的作业号:1  内存空闲
分配的地址为: 490  分配的空间: 140k  运行的作业号:5  内存已分配
```

## 六、调试总结及心得体会

通过这次的实验，我进一步加深了对内存管理的理解与应用，也掌握了固定式分区分配的存储管理。刚开始不知怎么做，最后经过本身的努力，终于把这个程序搞定了。知识层面亦得到了提升。

## 七、比较两种分配特点：

首次适应算法：该算法倾向于优先利用内存中低址部分的空闲分区，从而保留了高址部分的大空闲区，这给以后到达的大作业分配大的内存空间创造了条件。其缺点是低址部分不断被划分，会留下许多难以利用的很小的空闲空间，而每次查找又都是从低址部分开始的，这无疑会增加查找可用的空闲分区时的开销。

## 八、源程序代码：

```
#include <iostream>
#include <stdlib.h>
#include <time.h>

using namespace std;
struct memory
{
    struct memory *former;
    int address;//地址
    int num;//作业号
    int size;//分配内存大小
    int state;//状态0表示空闲1表示已分配
```

```

    struct memory *next;
};

typedef struct memory MEMORY;
MEMORY *mem;
const int size_min=10;//内存允许的最小空闲块的大小
bool is_optimist=false;//判断是否是最佳适应算法

void init();
void FF();
void alloc(MEMORY *,MEMORY *);//首次适应算法分配内存
void free(MEMORY *);//首次适应算法回收内存
void sort(MEMORY *);//对内存链进行排序
void insert(MEMORY *,MEMORY *);
void free_optimist(MEMORY *);
void print(MEMORY *);//打印内存链
void main()
{
    int i=0;
    while(1)
    {
        cout<<" -----动态分区分配方式-----"<<endl;
        cout<<" ----张大川3110005966-----"<<endl;
        cout<<("Please select a number(1,2,0)");
        cout<<("\n 1--首次适应算法");
        cout<<("\n 2--最佳适应算法"<<endl;
        cout<<"    0--中止程序"<<endl;
        cin>>i;

        if(i==1)
        {
            cout<<("\nThis is an example for FF:\n");
            is_optimist=false;
            init();
            FF();
        }
        else if(i==2)
        {
            cout<<("\nThis is an example for optimist method;\n");
            is_optimist=true;
            init();
            FF();
        }
        else if(i==0)

```

```

        {
            exit(1);
        }
    }
}

void init()
{
    mem=new MEMORY;
    mem->size=640;
    //mem->state=0;
    mem->former=0;
    mem->next=0;
}

void FF() //首次适应算法
{
    int i;
    int work[]={130, 60, 100, 200, 140, 60, 50, 60}; //作业序列
    //int assignment;
    MEMORY *running;
    for(i=0;i<sizeof(work)/sizeof(int);i++)
    {
        running=(MEMORY *)malloc(sizeof(MEMORY)); //初始化作业
        if(running!=NULL)
        {
            running->former=NULL;
            running->address=0;
            running->num=i+1;
            running->size=work[i];
            running->state=0;
            running->next=NULL;
            //cout<<"作业初始化成功"<<running->num<<endl;
            if(is_optimist==true) //最佳适应算法
            {
                //cout<<"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"<<endl;
                alloc(mem, running);
            }
            else //首次适应算法
            {
                alloc(mem, running);
            }
            print(mem);
            cout<<endl;
        }
    }
}
else

```

```

        cout<<"没有足够的内存空间"<<endl;

    if(rand()%3==1)
    {
        if(is_optimist==false)//首次适应算法
        {
            free(mem);
        }
        else//最佳适应算法
        {
            ::free_optimist(mem);
        }
    }
}

void free(MEMORY *ptr)//作业处理完后释放内存空间
{
    MEMORY *previous,*current;
    previous=ptr;
    current=previous->next;
    while(current!=NULL)
    {
        if(current->state==1&&rand()%3==1)
        {
            break;
        }
        previous=current;
        current=current->next;
    }
    if(current==NULL)
    {
        //cout<<"内存中没有任何作业!!!"<<endl;
        return;
    }
    else if(current->next==NULL)
    {
        if(previous->state==0)
        {
            MEMORY *temp;
            temp=current;
            previous->size=previous->size+current->size;
            previous->next=NULL;
            cout<<"作业 " <<(current->num)<<"释放 " <<(current->size)<<"k 的空间"<<endl;

```

```

        delete temp;
        print(mem);
    }
    else
    {
        current->state=0;
        cout<<"作业 " <<(current->num)<<"释放 " <<(current->size)<<"k 的空间"<<endl;
        print(mem);
    }
}
else if((current->next)->next==NULL)
{
    if(previous->state==0&&(current->next)->state==0)
    {
        MEMORY *temp1,*temp2;
        temp1=current;
        temp2=current->next;
        previous->size=previous->size+current->size+(current->next)->size;
        previous->next=NULL;
        cout<<"作业 " <<(current->num)<<"释放 " <<(current->size)<<"k 的空间"<<endl;
        delete temp1;
        delete temp2;
        print(mem);
    }
    else if(previous->state==0)//释放的地址空间前面有空闲块则把它和前面的合并
    {
        MEMORY *temp;
        temp=current;
        previous->size=previous->size+current->size;
        (current->next)->former=previous;
        previous->next=current->next;
        cout<<"作业 " <<(current->num)<<"释放 " <<(current->size)<<"k 的空间"<<endl;
        delete temp;
        print(mem);
    }
    else if((current->next)->state==0)//释放的地址空间后面有空闲块则把它和后面的空
闲块合并
    {
        MEMORY *temp;
        temp=current->next;
        current->size=current->size+(current->next)->size;
        current->state=0;
        current->next=NULL;
        cout<<"作业 " <<(current->num)<<"释放 " <<(current->size)<<"k 的空间"<<endl;

```

```

        delete temp;
        print(mem);

    }
    else//处理完的作业前后都没有空闲块时直接把它状态改为没分配
    {
        current->state=0;
        cout<<"作业 " <<(current->num)<<"释放 " <<(current->size)<<"k 的空间"<<endl;
        print(mem);
    }

}
else
{
    if(previous->state==0&&(current->next)->state==0)
    {
        MEMORY *temp1,*temp2;
        temp1=current;
        temp2=current->next;
        previous->size=previous->size+current->size+(current->next)->size;
        ((current->next)->next)->former=previous;
        previous->next=(current->next)->next;

        cout<<"作业 " <<(current->num)<<"释放 " <<(current->size)<<"k 的空间"<<endl;
        delete temp1;
        delete temp2;
        print(mem);
    }
    else if(previous->state==0)//释放的地址空间前面有空闲块则把它和前面的合并
    {
        MEMORY *temp;
        temp=current;
        previous->size=previous->size+current->size;
        (current->next)->former=previous;
        previous->next=current->next;
        cout<<"作业 " <<(current->num)<<"释放 " <<(current->size)<<"k 的空间"<<endl;
        delete temp;
        print(mem);
    }
    else if((current->next)->state==0)//释放的地址空间后面有空闲块则把它和后面的空闲块合并
    {
        MEMORY *temp;

```



```

        temp=current->next;
        current->size=current->size+(current->next)->size;
        current->state=0;
        ((current->next)->next)->former=current;
        current->next=(current->next)->next;

        cout<<"作业 " <<(current->num)<<"释放 " <<(current->size)<<"k 的空间"<<endl;

        delete temp;
        print(mem);

    }
    else//处理完的作业前后都没有空闲块时直接把它状态改为没分配
    {
        current->state=0;
        cout<<"作业 " <<(current->num)<<"释放 " <<(current->size)<<"k 的空间"<<endl;
        print(mem);
    }
}

void alloc(MEMORY *ptr, MEMORY *assign)//内存分配
{
    if(ptr->next==NULL)//内存没有作业运行
    {
        if(ptr->size>=assign->size)//内存空间大于作业所需空间
        {
            ptr->size=ptr->size-assign->size;//为内存分配空间
            assign->state=1;
            ptr->next=assign;
            assign->former=ptr;

            cout<<" 作业 " <<(assign->num)<<"申请" <<(assign->size)<<" " <<"k的内存空间"
            "<<endl;
        }
        else
        {
            cout<<"没有足够的内存空间为作业" <<(assign->num)<<"分配"<<endl;
            delete assign;
        }
    }
    else//内存中如果已经分配了空间
    {
        MEMORY *previous,*current;

```

```

previous=ptr;
current=previous->next;
while(current!=NULL)
{
    if(current->size>assign->size&&ptr->state==0)//如果当前内存空间大于作业所需空间并且内存没有被分配
    {
        break;
    }
    previous=current;
    current=current->next;
}

if(current==NULL)//空闲链中没有为作业分配所需的空间
{

    if(ptr->size>=assign->size)//内存中还有足够没分配的空间为此作业分配
    {
        assign->address =640-(ptr->size);//max+size_offset;//作业在内存中的首地址
        ptr->size=ptr->size-assign->size;
        assign->state=1;
        assign->former=previous;
        previous->next=assign;

        cout<<"作业 " <<(assign->num)<<"申请"<<(assign->size)<<" " <<"k的内存空间"<<endl;
    }
    else
    {
        cout<<"没有足够的内存空间为作业"<<(assign->num)<<"分配"<<endl;
    }
}
else//空闲链中有可为此作业分配的空间
{

    if((current->size-assign->size)<=size_min)//空闲链所具备的空间与作业所需空间大小差不多时
    {
        //直接把整个空闲块的空间分配给作业
        //否则从空闲块中
        current->num=assign->num;
        current->state=1;
        delete assign;//free(assign);
        //划出与作业等同的空间
    }
}

```

```

        cout<<"作业 " <<(current->num)<<"申请"<<(current->size)<<" " <<"k的内存
间"<<endl;
    }
    else//从空闲块中划分一块与作业大小等同的空间
    {
        current->size=current->size-assign->size;
        assign->state=1;
        assign->address=current->address+current->size;

        if(current->next==NULL)//此要分配的空间是空闲链的最后一个元素
        {
            assign->former=current;
            current->next=assign;
        }
        else
        {
            assign->next=current->next;
            (current->next)->former=assign;
            assign->former=current;
            current->next=assign;
        }
        cout<<"作业 " <<(assign->num)<<"申请"<<(assign->size)<<" " <<"k的内存空
间"<<endl;
    }
}

if((ptr->next)->next!=NULL&&is_optimist==true)
    sort(ptr);//排序由空闲块从小到大
//print(ptr);
}

void sort(MEMORY *ptr)
{
    MEMORY *temp=new MEMORY;
    temp->next=0;
    temp->former=0;

    while(ptr->next)
    {
        if((ptr->next)->next==NULL)//内存链中只有两个元素
        {

```

```

    MEMORY *p;
    p=ptr->next;
    ptr->next=NULL;
        insert(temp, p);
    }
else//内存链中有多个元素
{
    MEMORY *p;
        p=ptr->next;
        p->former=ptr;
        ptr->next=p->next;
        (p->next)->former=ptr;
    insert(temp, p);
}

}

ptr->next=temp->next;
(temp->next)->former=ptr;
delete temp;
}

void insert(MEMORY *queue, MEMORY *item)
{
    MEMORY *previous,*current;
    previous=queue;
    current=previous->next;
    while(current!=NULL && item->size>=current->size)
    {
        previous=current;
        current=current->next;
    }

    if(previous==queue)//所要插入的元素最小
    {
        if(queue->next==NULL)//内存链中只有一个元素
        {
            item->next=0;
            queue->next=item;
            item->former=queue;
        }
        else//内存链中有多个元素
        {
            item->next=queue->next;
            (queue->next)->former=item;

```

```

        item->former=queue;
        queue->next=item;
    }
}
else//定位到要插入的元素
{
    item->next=current;
    item->former=previous;
    if(current==NULL)
    {
        previous->next=item;
    }
    else
    {
        current->former=item;
        previous->next=item;
    }
}
}
}
void free_optimist(MEMORY *ptr)
{
    MEMORY *previous,*current;
    previous=ptr;
    current=previous->next;
    while(current!=NULL)
    {
        if(current->state==1&&rand()%3==1)
        {
            break;
        }
        previous=current;
        current=current->next;
    }
    if(current==NULL)
    {
        //cout<<"内存中没有任何作业!!! "<<endl;
        return;
    }
    else if(current->next==NULL)
    {
        if(previous->state==0&&((previous->address+previous->size)==current->address))
        {
            MEMORY *temp;

```

```

        temp=current;
        previous->size=previous->size+current->size;
        previous->next=NULL;
        cout<<"作业 " <<(current->num)<<"释放 " <<(current->size)<<"k 的空间"<<endl;
        delete temp;
        print(mem);
    }
else
{
    current->state=0;
    cout<<"作业 " <<(current->num)<<"释放 " <<(current->size)<<"k 的空间"<<endl;
    print(mem);
}
}
else if((current->next)->next==NULL)
{

    if(previous->state==0&&(current->next)->state==0&&((previous->address+previous->size)==current->address)&&((current->size+current->address)==(current->next)->address))
    {
        MEMORY *temp1,*temp2;
        temp1=current;
        temp2=current->next;
        previous->size=previous->size+current->size+(current->next)->size;
        previous->next=NULL;
        cout<<"作业 " <<(current->num)<<"释放 " <<(current->size)<<"k 的空间"<<endl;
        delete temp1;
        delete temp2;
        print(mem);
    }
else
if(previous->state==0&&((previous->address+previous->size)==current->address))// 释放的
地址空间前面有空闲块则把它和前面的合并
{
    MEMORY *temp;
    temp=current;
    previous->size=previous->size+current->size;
    (current->next)->former=previous;
    previous->next=current->next;
    cout<<"作业 " <<(current->num)<<"释放 " <<(current->size)<<"k 的空间"<<endl;
    delete temp;
    print(mem);
}
else

```

```

if((current->next)->state==0&&((current->size+current->address)==(current->next)->address))//释放的地址空间后面有空闲块则把它和后面的空闲块合并
{
    MEMORY *temp;
    temp=current->next;
    current->size=current->size+(current->next)->size;
    current->state=0;
    current->next=NULL;
    cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;

    delete temp;
    print(mem);
}
else
{

    if(previous->state==0&&(current->next)->state==0&&((previous->address+previous->size)==current->address)&&((current->size+current->address)==(current->next)->address))
    {
        MEMORY *temp1,*temp2;
        temp1=current;
        temp2=current->next;
        previous->size=previous->size+current->size+(current->next)->size;
        ((current->next)->next)->former=previous;
        previous->next=(current->next)->next;

        cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
        delete temp1;
        delete temp2;
        print(mem);
    }
    else
    if(previous->state==0&&(previous->address+previous->size)==current->address)//释放的地址空间前面有空闲块则把它和前面的合并
    {
        MEMORY *temp;
        temp=current;
        previous->size=previous->size+current->size;
        previous->state=0;
        (current->next)->former=previous;
        previous->next=current->next;
        cout<<"作业 "<<(current->num)<<"释放 "<<(current->size)<<"k 的空间"<<endl;
        delete temp;
    }
}

```

```

        print(mem);
    }
    else
if((current->next)->state==0&&((current->size+current->address)==(current->next)->address))//释放的地址空间后面有空闲块则把它和后面的空闲块合并
    {
        MEMORY *temp;
        temp=current->next;
        current->size=current->size+(current->next)->size;
        current->state=0;
        ((current->next)->next)->former=current;
        current->next=(current->next)->next;

        cout<<"作业 " <<(current->num)<<"释放 " <<(current->size)<<"k 的空间"<<endl;

        delete temp;
        print(mem);

    }
    else//处理完的作业前后都没有空闲块时直接把它状态改为没分配
    {
        current->state=0;
        cout<<"作业 " <<(current->num)<<"释放 " <<(current->size)<<"k 的空间"<<endl;
        print(mem);
    }
}
if((ptr->next)->next!=NULL)
    sort(ptr); //排序由空闲块从小到大
}

void print(MEMORY *ptr)
{
    MEMORY *temp;
    temp=ptr->next;
    cout<<"\n内存链的状态为: "<<endl;
    while(temp!=NULL)
    {
        cout<<"分配的地址为: " <<temp->address<<"    分配的空间: " <<temp->size<<"k"
            <<"    运行的作业号: " <<temp->num;
        if(temp->state==0)
        {
            cout<<"    内存空闲";
        }
        else
        {

```



```
        cout<<"    内存已分配";  
    }  
    cout<<endl;  
    temp=temp->next;  
}  
}
```

## 实验 四 题目 文件系统 第      周星期

### 一、实验目的

模拟文件系统实现的基本功能,了解文件系统的基本结构和文件的各种管理方法,加深理解文件系统的内部功能及内部实现。通过用高级语言编写和调试一个简单的文件系统,模拟文件管理的工作过程,从而对各种文件操作命令的实质内容和执行过程有比较深入的了解。

### 二、实验内容和要求

编程模拟一个简单的文件系统,实现文件系统的管理和控制功能。要求本文件系统采用两级目录,即设置主文件目录[MFD]和用户文件目录[UED]。另外,为打开文件设置运行文件目录[AFD]。设计一个 10 个用户的文件系统,每次用户可保存 10 个文件,一次运行用户可以打开 5 个文件,并对文件必须设置保护措施。在用户程序中通过使用文件系统提供的 Create、open、read、write、close、delete 等文件命令,对文件进行操作。

### 三、实验主要仪器设备和材料

#### 实验环境

硬件环境：个人台式机 Microsoft Windows 7

软件环境：C 语言编程环境, vs 2010

### 四、实验设计方案及原理

#### 1、实验原理

因为系统小，对文件目录检索使用简单的线性搜索，为了便于实现，对文件读写做了简化，在执行读写命令时，只修改读写指针，并不实际对文件进行操作，其中初始状态下，读指针=0，写指针=文件长度，文件保护码使用三位保护码：1：表示文件允许的读 2：表示文件可写 3. 表示文件可执行，创建文件时通过模拟磁盘的使用情况来实现。

#### 2、设计方案

程序采用两级目录结构，第一级设置主文件目录[MFD], 第二级为用户文件目录[UFD]，另外为打开文件设置运行目录[AFD]

```
struct mdf{                                /* 第一级：主目录 MDF */
    char name[10];                          /* 用户名 */
    UFD* directory;                        /* 文件目录指针 */
    }maindir[10];                          /* 用户数组 */
typedef struct mdf MDF;

struct ufd{                                /* 第二级：用户文件目录 UFD */
    char filename[10];                    /* 文件名 */
    char procode[3];                      /* 保护码 1:读 2:写 3:执行*/
    int length;                          /* 文件长度 */
    }*p;
typedef struct ufd UFD;

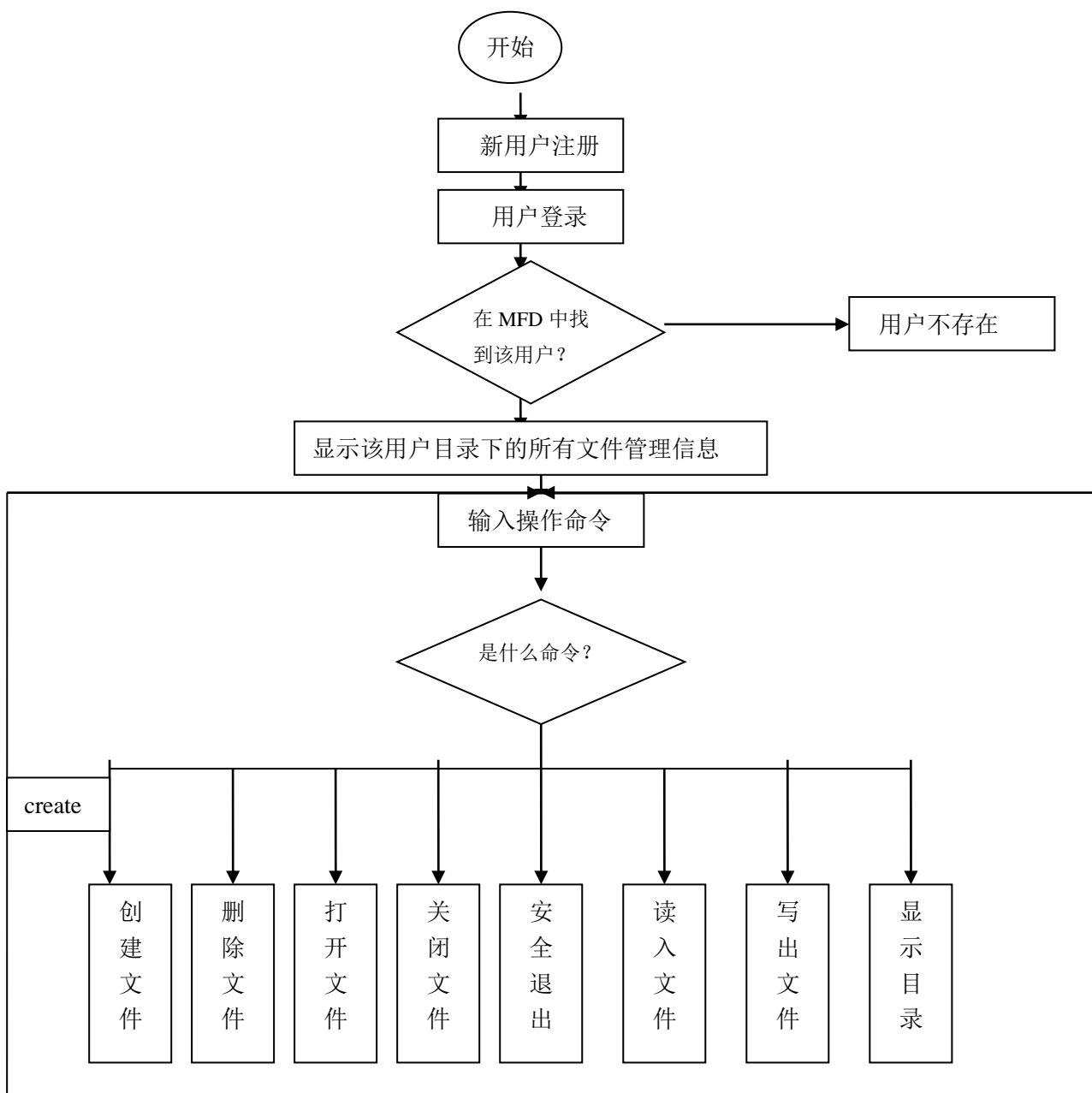
struct afd{                                /* 文件运行目录 */
    char filename[10];                    /* 文件名 */
    char procode[3];                      /* 保护码 1:读 2:写 3:执行*/
    int rw;                              /* 读写指针 初始状态读:0 写:文件长度 */
    }afd[5];
```

系统提供 8 条命令：create delete open close read write display quit，可以提供创建、删除、打开、关闭、读入、写出、显示、退出等功能。

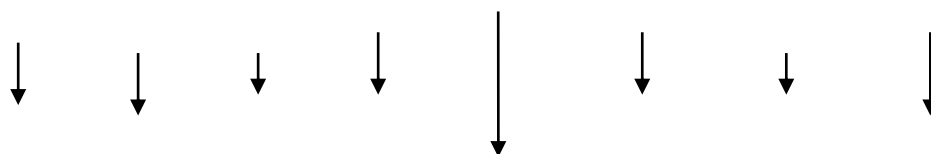
- ①、Create—在用户当前目录下创建一个文件，该文件的管理信息登录到用户文件信息管理模块中。  
命令格式：create
- ②、delete—删除当前用户目录下的一个文件，删除后文件无法打开。  
命令格式：delete
- ③、open—在当前用户目录打开某个文件并修改标志位，对文件各种操作都需先打开。  
命令格式：open

- ④、close—关闭用户运行的文件。执行该命令后，用户在系统中运行文件状态位被修改为关闭。  
命令格式：close
- ⑤、read—从用户已打开文件读信息，将文件信息读入并修改读取标志位，未打开文件无法读取。  
命令格式：read
- ⑥、write—向用户已打开文件写入信息，修改标志位，未打开文件无法写出操作。  
命令格式：write
- ⑦、display 用户目录文件列表显示。  
命令格式：display
- ⑧、quit 用户注销命令。当使用该命令时，用户退出系统。  
命令格式：quit

1、程序流程图：



delete	openn	close	quit	read	write	display
--------	-------	-------	------	------	-------	---------



## 五、实验结果及分析

文件系统初始界面，第一次运行必须注册用户，注册用户的信息将会被自动写到默认目录的 database.txt 文件，如果文件存在可以直接登陆不用注册，注册将会覆盖原来的数据：



注册信息演示，用户数量最大为 10:

```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

=====注册页面=====

请输入注册的用户数量:3
请输入第1个用户名:lzw
请输入第2个用户名:abc
请输入第3个用户名:fff
请输入lzw用户新建文件数<小于10>:3
请输入新建文件名:11111
请输入要建立的文件的属性<1:读,2:写,3:执行.>:1
请输入要构造的文件长度:12
请输入新建文件名:22222
请输入要建立的文件的属性<1:读,2:写,3:执行.>:1
请输入要构造的文件长度:23
请输入新建文件名:33333
请输入要建立的文件的属性<1:读,2:写,3:执行.>:1
请输入要构造的文件长度:21
请输入abc用户新建文件数<小于10>:2
请输入新建文件名:44444
请输入要建立的文件的属性<1:读,2:写,3:执行.>:1
请输入要构造的文件长度:23
请输入新建文件名:55555
请输入要建立的文件的属性<1:读,2:写,3:执行.>:1
请输入要构造的文件长度:10
请输入fff用户新建文件数<小于10>:1
请输入新建文件名:66666
请输入要建立的文件的属性<1:读,2:写,3:执行.>:1
请输入要构造的文件长度:15

=====注册完毕=====
```

注册结束会自动转到登陆界面，而所有用户和目录文件信息将会显示在登陆界面上方，可以选择登陆一个用户：

```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

  用户名  文件名  保护码  文件长度
  lzw     11111   1       12
         22222   1       23
         33333   1       21

  用户名  文件名  保护码  文件长度
  abc     44444   1       23
         55555   1       10

  用户名  文件名  保护码  文件长度
  fff     66666   1       15

=====登陆页面=====

请输入登录用户名:lzw

=====用户lzw目录下的文件=====

11111
22222
33333
```

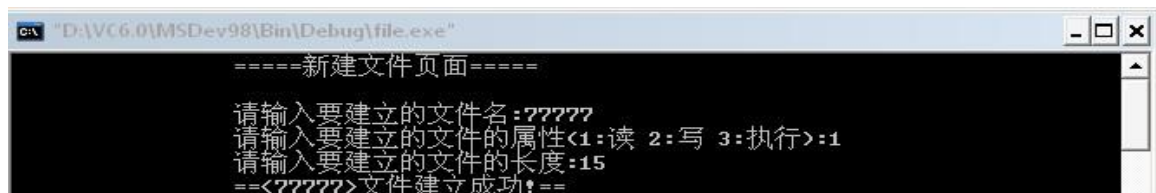
登陆成功则跳转至文件系统，可以输入相应命令对文件操作，这里测试先选择 create 命令新建文件

```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

<<文件系统演示>>

*****
create 新建文件.
delete 删除文件.
open   打开文件.
close  关闭文件.
read   读出文件.
write  写进文件.
display 显示用户列表.
quit   安全退出.
*****
操作命令:create delete open close read write display quit
请输入操作命令:create
```

新建文件页面，按要求输入名字、属性、长度等：




```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

=====新建文件页面=====

请输入要建立的文件名:?????
请输入要建立的文件的属性<1:读 2:写 3:执行>:1
请输入要建立的文件的长度:15
==<?????>文件建立成功!==
```

使用 open 命令打开一个文件，注意：对文件的所有读写等操作必须先打开文件，否则提示出错：



```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

<<文件系统演示>>

*****
create 新建文件.
delete 删除文件.
open   打开文件.
close  关闭文件.
read   读出文件.
write  写进文件.
display 显示用户列表.
quit   安全退出.
*****
操作命令:create delete open close read write display quit
请输入操作命令:open
```

在这里测试新建文件 88888：

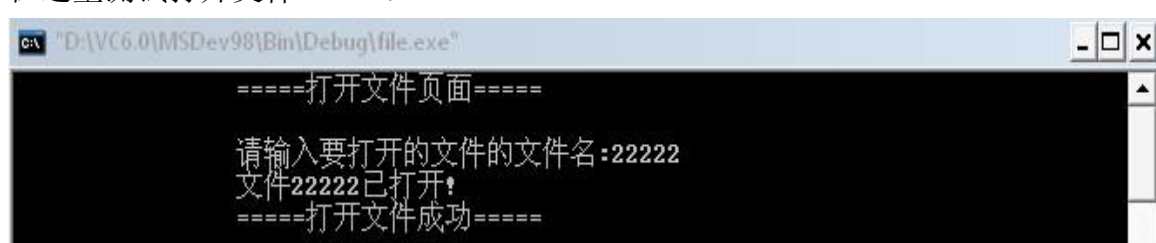


```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

=====新建文件页面=====

请输入要建立的文件名:88888
请输入要建立的文件的属性<1:读 2:写 3:执行>:3
请输入要建立的文件的长度:1
==<88888>文件建立成功!==
```

在这里测试打开文件 22222：

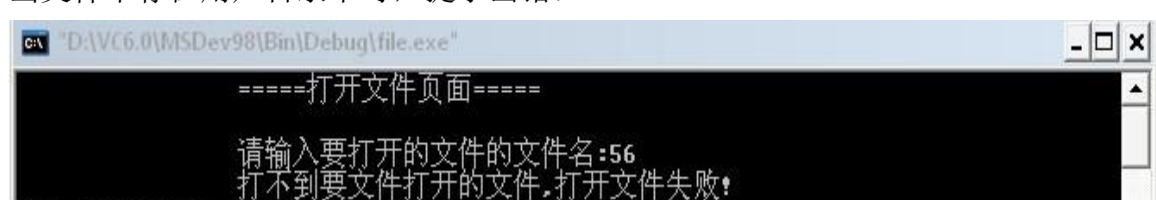


```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

=====打开文件页面=====

请输入要打开的文件的文件名:22222
文件22222已打开!
=====打开文件成功=====
```

当文件不存在用户目录下时，提示出错：



```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

=====打开文件页面=====

请输入要打开的文件的文件名:56
打不到要文件打开的文件.打开文件失败!
```

打开文件后，使用 read 命令读入文件，如果文件还没有在上一步打开，则操作提示出错：

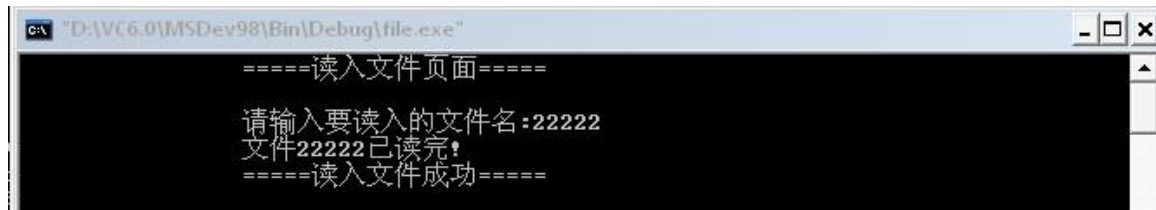


```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

=====读入文件页面=====

请输入要读入的文件名:656
操作错误, 当前文件?还没有打开!
```

成功读入文件 22222:

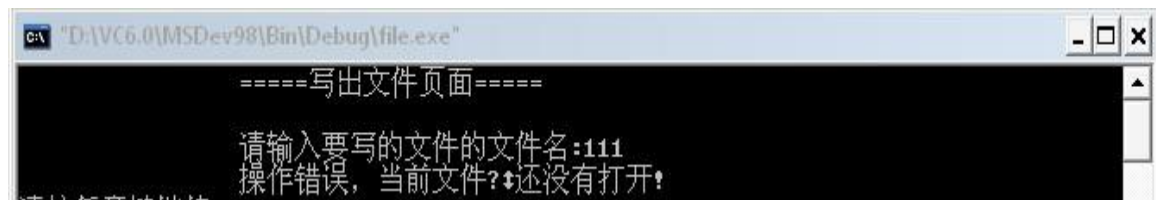


```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

====读入文件页面====

请输入要读入的文件名:22222
文件22222已读完!
====读入文件成功====
```

同样，使用 write 写出文件，如果文件也没有被打开则提示出错:



```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

====写出文件页面====

请输入要写的文件的文件名:111
操作错误, 当前文件?还没有打开!
```

成功写出文件 22222



```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

====写出文件页面====

请输入要写的文件的文件名:22222
文件22222已写完!
====写出文件成功====
```

使用 close 命令关闭文件，如果文件没有被打开则提示出错:

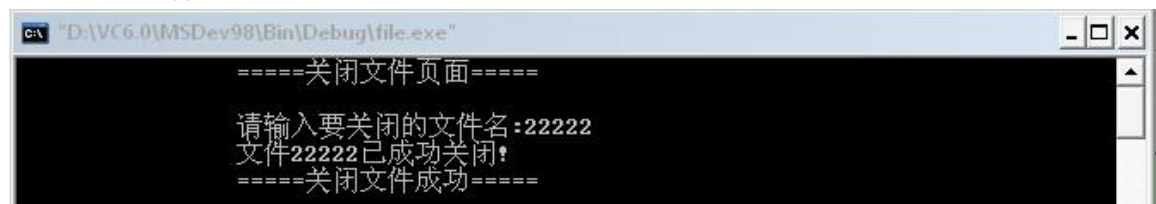


```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

====关闭文件页面====

请输入要关闭的文件名:77777
文件77777没有打开!关闭失败!
```

成功关闭文件 22222:

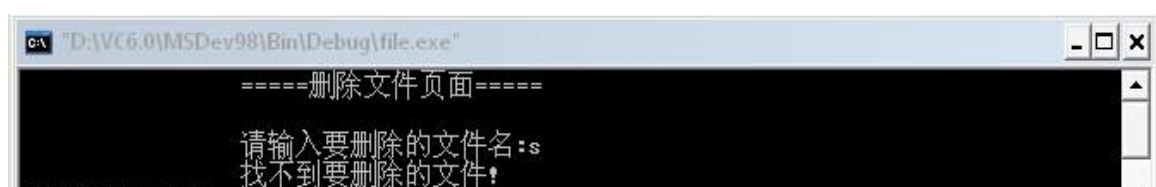


```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

====关闭文件页面====

请输入要关闭的文件名:22222
文件22222已成功关闭!
====关闭文件成功====
```

使用 delete 命令删除文件，找不到文件名字则提示出错:




```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

====删除文件页面====

请输入要删除的文件名:s
找不到要删除的文件!
```

成功删除用户一个文件 77777:



```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

====删除文件页面====

请输入要删除的文件名:77777
====文件删除成功!====
```

在主菜单使用 display 命令可以查看当前所有用户和文件详细信息:

```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

=====
用户名 文件名 保护码 文件长度
lzw    11111  1      12
       22222  1      23
       33333  1      21
       77777  1      15
       88888  3       1
       99999  2       2

用户名 文件名 保护码 文件长度
abc    44444  1      23
       55555  1      10

用户名 文件名 保护码 文件长度
fff    66666  1      15
=====
已经打开文件列表:
文件名 保护码 读写指针<为1表示已进行过写操作>
11111  1      0
=====
```

使用 quit 命令退出程序，则系统将自动保存当前操作结果：

```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

<<文件系统演示>>

*****
create 新建文件.
delete 删除文件.
open   打开文件.
close  关闭文件.
read   读出文件.
write  写进文件.
display 显示用户列表.
quit   安全退出.
*****
操作命令:create delete open close read write display quit
请输入操作命令:quit
当前目录信息已自动保存至文件database!
```

第二次运行程序，可以查看目录是否存在 database.txt 文件，如果存在直接使用登录功能可以获取上次的用户和文件信息：

```
C:\ "D:\VC6.0\MSDev98\Bin\Debug\file.exe"

=====主页面=====

1.注册用户
2.登陆用户
=====
请输入操作命令:2
```

登陆详细信息展示：



```
"D:\VC6.0\MSDev98\Bin\Debug\file.exe"

用户名 文件名 保护码 文件长度
lzw    11111  1      12
       22222  1      23
       33333  1      21
       77777  1      15
       88888  3       1
       99999  2       2

用户名 文件名 保护码 文件长度
abc     44444  1      23
       55555  1      10

用户名 文件名 保护码 文件长度
fff     66666  1      15

=====登陆页面=====

请输入登录用户名:lzw

=====用户lzw目录下的文件=====

11111
22222
33333
77777
88888
99999
请按任意键继续. . .
```

## 六、调试总结及心得体会

终于把文件管理的算法思想用 C 语言实现了，在实现的过程中，开始时无从下手，然后参考网上的资料，头脑中有了基本的数据结构，然后就按照自己设定的数据结构去实现文件打开 Create、delete、open、close、read、write 等操作，在实现的过程中遇到很多困难，但不断调试，不断修改，然后一步一步解决，最大的问题是 VS 2010 不怎么熟悉，很多快捷方式，查看方式都不懂，但它强大的调试功能还是不错的。

## 七、源程序代码：

```
#include<stdio.h>
#include<stdlib.h>
#include<iostream>
#define NULL 0
#define getpch(type,num) (type*)malloc(sizeof(type)*num)

struct ufd{
    /* 第二级：用户文件目录 UFD */
    char filename[10];    /* 文件名 */
    char procode[3];      /* 保护码 1:读 2:写 3:执行*/
    int length;           /* 文件长度 */
}*p;

typedef struct ufd UFD;

struct mdf{
    /* 第一级：主目录 MDF */
    char name[10];        /* 用户名 */
    UFD* directory;       /* 文件目录指针 */
    }maindir[10];         /* 用户数组 */

typedef struct mdf MDF;

struct afd{
    /* 文件运行目录 */
    char filename[10];    /* 文件名 */
    char procode[3];      /* 保护码 1:读 2:写 3:执行*/
    int rw;               /* 读写指针 初始状态读:0 写:文件长度 */
}afd[5];

int usernum, opennum;    /* 登陆用户数量 打开文件数量*/
int filenum[10];         /* 所有的用户目录下的文件数目*/
char username[10];       /* 当前登陆系统的用户名*/
char command[][10]={"create","delete","open","close","read","write","display","quit"};
/*文件操作命令*/

void login() /*用户注册，信息写入database文件*/
{
    int i,j;
    FILE *fp;
    int loginum;
    int creatnum[10];
    printf("\t\t====注册页面====\n\n");
```

```

printf("\t\t请输入注册的用户数量:");
scanf("%d",&loginum);
for(i=0;i<loginum;i++)
{
    printf("\t\t请输入第%d个用户名:",i+1);
    scanf("%s",maindir[i].name);
}
for(i=0;i<loginum;i++)
{
    printf("\t\t请输入%s用户新建文件数(小于10):",maindir[i].name);
    scanf("%d",&creatnum[i]);
    p=getpch(UFD,10);
    maindir[i].directory=p;
    for(j=0;j<creatnum[i];j++)
    {
        printf("\t\t请输入新建文件名:");
        scanf("%s",p[j].filename);
        printf("\t\t请输入要建立的文件的属性(1:读,2:写,3:执行.):");
        scanf("%s",p[j].procode);
        printf("\t\t请输入要构造的文件长度:");
        scanf("%d",&p[j].length);
    }
}

fp=fopen("database.txt","wb");
fwrite(&loginum,sizeof(int),1,fp); /* 写已注册用户数量到dat文件*/
for(i=0;i<loginum;i++)
{
    fwrite(maindir[i].name,sizeof(char)*10,1,fp); /* 写已注册用户名到dat文件*/
}
for(i=0;i<loginum;i++)
{
    fwrite(&creatnum[i],sizeof(int),1,fp); /* 写用户新建文件数量到dat文件*/
    p=maindir[i].directory; /* UFD获得文件目录指针*/
    fwrite(p,sizeof(UFD),10,fp); /* 写用户文件目录指针地址到dat文
件*/
}
fclose(fp);
printf("\n\t\t====注册完毕====\n");
printf("\n\t\t");
system("pause");
system("cls");
}

```

```

void init()      /*初始化读入login信息*/
{
    int i, j;
    FILE *fp;
    fp=fopen("database.txt", "rb");
    if (fp==NULL)
    {
        printf("\n\t警告:根目录不存在记录, 请先注册用户\n");
        login();
        fp=fopen("database.txt", "rb");
    }
    fread(&usernum, sizeof(int), 1, fp);      /* 读入已注册用户数量*/
    for (i=0; i<usernum; i++)
    {
        fread(maindir[i].name, sizeof(char)*10, 1, fp); /* 读入已注册用户名*/
        p=getpch(UFD, 10);
        maindir[i].directory=p;
    }
    for (i=0; i<usernum; i++)
    {
        fread(&j, sizeof(int), 1, fp);
        filenum[i]=j;      /*读入用户新建文件数量*/
        p=maindir[i].directory; /*获得文件目录指针*/
        fread(p, sizeof(UFD), 10, fp); /*读入用户文件目录指针地址p*/
    }
    opennum=0;      /*运行文件数*/
}

void createfile(int userid)      /*新建文件*/
{
    system("cls");
    char cfilename[10];
    int filelen, i;
    char code[3];
    if (filenum[userid]>=10) /*如果用户的文件已有10个*/
    {
        printf("\n\t\t用户的文件数量已达到上限, 即10个!\n");
        return ;
    }
    printf("\t\t====新建文件页面====\n\n");
    printf("\t\t请输入要建立的文件名:");
    scanf("%s", cfilename);
    for (i=0; i<filenum[userid]; i++) /*检查是否有同名文件, 有则提示然后返回*/
    {

```

```

        if(strcmp((p+i)->filename, cfilename)==0)
        {
            printf("\t\t文件名%s已存在!\n", cfilename);
            system("pause");
            system("cls");
            return ;
        }
    }
    printf("\t\t请输入要建立的文件的属性(1:读 2:写 3:执行):");
    scanf("%s", code);
    printf("\t\t请输入要建立的文件的长度:");
    scanf("%d", &filelen);
    strcpy(p[filenum[userid]].filename, cfilename); /*添加新文件名*/
    strcpy(p[filenum[userid]].procode, code);      /*添加保护码*/
    p[filenum[userid]].length=filelen;              /*添加文件长度*/
    filenum[userid]=filenum[userid]+1;              /*更新文件数量*/
    printf("\t\t==<%s>文件建立成功!==\n", cfilename);
    system("pause");
    system("cls");
}

void deletefile(int userid) /*删除文件*/
{
    system("cls");
    char dfilename[10];
    int i, j;
    printf("\t\t====删除文件页面====\n\n");
    printf("\t\t请输入要删除的文件名:");
    scanf("%s", dfilename);
    for(i=0; i<filenum[userid]; i++)
    {
        if(strcmp(p[i].filename, dfilename)==0) /*找到要删除的文件*/
        {
            for(j=i; j<filenum[userid]-1; j++)
            {
                strcpy(p[j].filename, p[j+1].filename);
                strcpy(p[j].procode, p[j+1].procode);
                p[j].length=p[j+1].length;
                if(j==filenum[userid]-1)
                {
                    strcpy(p[j+1].filename, " ");
                    strcpy(p[j+1].procode, " ");
                    p[j+1].length=0;
                    free(p+j+1);
                }
            }
        }
    }
}

```

```

        }
    }

    filenum[userid]--;
    printf("\t\t====文件删除成功!====\n");
    system("pause");
    system("cls");
    return ;
}
}

printf("\t\t找不到要删除的文件!\n");
system("pause");
system("cls");
}

void openfile(int userid)          /*打开文件*/
{
    system("cls");
    int i;
    char ofilename[10];
    printf("\t\t====打开文件页面====\n\n");
    if(opennum>=5)
    {
        printf("\t\t打开超过5个文件!\n");
        return ;
    }
    printf("\t\t请输入要打开的文件的文件名:");
    scanf("%s", ofilename);
    for(i=0; i<opennum; i++)
    {
        if(strcmp(afd[i].filename, ofilename)==0) /*文件已经打开*/
        {
            printf("\t\t文件%s已打开!\n", ofilename);
            return ;
        }
    }
    for(i=0; i<filenum[userid]; i++)          /*在运行文件目录中找到要打开的文件*/
    {
        if(strcmp(p[i].filename, ofilename)==0) /*找到该文件*/
        {
            strcpy(afd[opennum].filename, ofilename); /*文件运行目录添加打开文件名*/
            strcpy(afd[opennum].procode, p[i].procode); /*文件运行目录添加文件保护码*/
            afd[opennum].rw=0; /*更新文件运行目录文件读写指针*/
            opennum++;
        }
    }
}

```

```

        printf("\t\t文件%s已打开!\n", ofilename);
        printf("\t\t====打开文件成功====\n\n");
        system("pause");
        system("cls");
        return ;
    }
}

printf("打不到要文件打开的文件, 打开文件失败!\n");
system("pause");
system("cls");
}

void closefile()          /*关闭文件*/
{
    system("cls");
    int i, j;
    char clfilename[10];
    if(opennum==0)        /*没有文件被打开过*/
    {
        printf("\t\t当前还没有打开文件!\n");
        return ;
    }
    printf("\t\t====删除文件页面====\n\n");
    printf("\t\t请输入要关闭的文件名:");
    scanf("%s", clfilename);
    for(i=0; i<opennum; i++)
    {
        if(strcmp(afd[i].filename, clfilename)==0) /*在运行文件目录中找到要关闭的文件*/
        {
            if(afd[i].rw==1)
            {
                printf("\t\t文件写回磁盘!\n");
            }
            for(j=i; j<opennum-1; j++)          /*更新文件运行目录, 关闭文件*/
            {
                strcpy(afd[j].filename, afd[j+1].filename);
                strcpy(afd[j].procode, afd[j+1].procode);
                afd[j].rw=afd[j+1].rw;
            }
            opennum--;
            printf("\t\t文件%s已成功关闭!\n", clfilename);
            printf("\t\t====关闭文件成功====\n\n");
            system("pause");
            system("cls");
        }
    }
}

```

```

        return ;
    }
}

printf("\t\t文件%s没有打开!关闭失败!\n", clfilename);
system("pause");
system("cls");
}

void readfile()          /*读入文件*/
{
    system("cls");
    int i;
    char rfilename[10];
    if(opennum==0)       /*没有文件被打开过*/
    {
        printf("\t\t操作错误, 当前还没有打开文件!\n");
        system("pause");
        system("cls");
        return ;
    }
    printf("\t\t====读入文件页面====\n\n");
    printf("请输入要读入的文件名:");
    scanf("%s", rfilename);
    for(i=0; i<opennum; i++)
    {
        if(strcmp(afd[i].filename, rfilename)==0) /*在运行文件目录中找到要读的文件*/
        {
            if(afd[i].procode[0]!='1') /*文件没有读权限, 将不执行读操作, 提示返回*/
            {
                printf("\t\t权限不足, 该文件不能读!读失败!\n");
                return ;
            }

            printf("\t\t文件%s已读完!\n", rfilename);
            printf("\t\t====读入文件成功====\n\n");
            system("pause");
            system("cls");
            break ;
        }
    }
}

void writefile()         /*写出文件*/
{
    system("cls");

```



```

int i;
char wfilename[10];
if(opennum==0)    /*没有文件被打开过*/
{
    printf("\t\t当前还没有打开文件!\n");
    system("pause");
    system("cls");
    return ;
}
printf("\t\t====写出文件页面====\n\n");
printf("\t\t请输入要写的文件的文件名:");
scanf("%s",wfilename);
for(i=0;i<opennum;i++)
{
    if(strcmp(afd[i].filename,wfilename)==0)/*在运行文件目录中找到要关闭的文件*/
    {
        if(p[i].procode[1]!='1')/*文件没有写权限,将不执行写操作,提示返回*/
        {
            printf("\t\t该文件不能写!写失败!\n");
            return ;
        }
        afd[i].rw=1;
        printf("\t\t文件%s已写完!\n",wfilename);
        printf("\t\t====写出文件成功====\n\n");
        return ;
    }
}
printf("\t\t操作错误,当前文件%s还没有打开!\n");
system("pause");
system("cls");
}

void displayinfo()    /*显示所有用户目录文件信息*/
{
    system("cls");
    int i,j;
    printf("\t\t=====\n");
    for(i=0;i<usernum;i++)    /*登陆页面用户列表显示*/
    {
        printf("\n\t用户名\t文件名\t保护码\t文件长度\n");
        printf("\t%s",maindir[i].name);
        for(j=0;j<filenum[i];j++)
        {
            if(j!=0)printf("\t");

```

```

printf("\t%s\t%s\t%d\n", maindir[i].directory[j].filename, maindir[i].directory[j].procode, maindir[i].directory[j].length);
    }
}
printf("\t=====\\n");
printf("\t已经打开文件列表:\\n\\t\\t文件名\\t保护码\\t读写指针(为1表示已进行过写操作)\\n");
for(i=0; i<opennum; i++)
{
    printf("%s\\t%s\\t%d\\n", afd[i].filename, afd[i].procode, afd[i].rw);
}
system("pause");
system("cls");
}

```

```

void savefile() /*保存操作*/
{
    /*开始把输入的构造信息写入到文件dat*/
    int i;
    FILE *fp;
    fp=fopen("database.txt", "wb");
    fwrite(&usernum, sizeof(int), 1, fp);
    for(i=0; i<usernum; i++)
    {
        fwrite(maindir[i].name, sizeof(char)*10, 1, fp);
    }
    for(i=0; i<usernum; i++)
    {
        fwrite(&filenum[i], sizeof(int), 1, fp);
        p=maindir[i].directory;
        fwrite(p, sizeof(UFD), 10, fp);
    }
    fclose(fp);
    printf("\t\\t当前目录信息已自动保存至文件database!\\n");
}

```

```

void main()
{
    int i, j, n, userid, choose=0, quit=0;
    char inputname[10];
    system("cls");
    printf("\\n\\t ----10级计算机科学与技术03班----");
}

```

```
printf("\n\t\t\t\t\t-----计科3张大川---\n");
printf("\n\t\t\t\t\t====主页面====\n");
printf("\n\t\t\t\t\t1. 注册用户");
printf("\n\t\t\t\t\t2. 登陆用户");
printf("\n\t\t\t\t\t===== \n");
printf("\t\t请输入操作命令:");
scanf("%d",&n);
if(n==1)
{
    login(); /*用户注册，用户文件信息写入dat文件*/
}
init(); /*信息初始化*/
system("cls");
for(i=0;i<usenum;i++) /*登陆页面用户列表显示*/
{
    printf("\n\t用户名\t文件名\t保护码\t文件长度\n");
    printf("\t%s",maindir[i].name);
    for(j=0;j<filenum[i];j++)
    {
        if(j!=0)printf("\t");
    }
    printf("\t%s\t%s\t%d\n",maindir[i].directory[j].filename,maindir[i].directory[j].procode,maindir[i].directory[j].length);
}
}
printf("\n\t\t\t\t\t====登陆页面====\n");
j=0;
while(j==0)
{
    printf("\n\t\t\t\t\t请输入登录用户名:"); /*用户登陆*/
    scanf("%s",username);
    for(userid=0;userid<usenum;userid++)
    {
        if(strcmp(maindir[userid].name,username)==0) /*用户存在*/
        {
            j=1;
            break;
        }
    }
    if(j==1)
    {
        break;
    }
    printf("\n\t\t\t\t\t输入的用户名不存在!\n"); /*用户不存在*/
}
```

```

}
p=maindir[userd].directory;                /*获得用户文件目录指针*/
printf("\n\t\t====用户%s目录下的文件====\n", username);
for(i=0; i<filenum[userd]; i++)
{
    printf("\n\t\t%s", p[i].filename);      /*显示该用户文件列表*/
}
printf("\n\t\t");
system("pause");
system("cls");
opennum=0;
while(true)                                /*操作主目录*/
{
    j=0;
    system("cls");
    printf("\n\t\t\t\t\t<<文件系统演示>>\n");
    printf("\n\t\t\t*****\t\t");
    printf("\n\t\t\t\t\t create 新建文件.");
    printf("\n\t\t\t\t\t delete 删除文件.");
    printf("\n\t\t\t\t\t open 打开文件.");
    printf("\n\t\t\t\t\t close 关闭文件.");
    printf("\n\t\t\t\t\t read 读出文件.");
    printf("\n\t\t\t\t\t write 写进文件.");
    printf("\n\t\t\t\t\t display 显示用户列表.");
    printf("\n\t\t\t\t\t quit 安全退出.");
    printf("\n\t\t\t*****\t\t\n");
    printf("\t\t\t\t\t操作命令:create delete open close read write display quit\n");
    printf("\t\t\t\t\t请输入操作命令:");
    scanf("%s", inputname);
    getchar();
    for(i=0; i<8; ++i)
    {
        if(strcmp(command[i], inputname)==0) /*命令存在*/
        {
            j=1;
            choose=i+1;
            break;
        }
    }
    if(j==0)
    {
        printf("输入命令不正确!");
        system("pause");
        continue;
    }
}

```

```
    }

    switch(choose)
    {

        case 1:
            createfile(userid);
            break;

        case 2:
            deletetfile(userid);
            break;

        case 3:

            openfile(userid);
            break;

        case 4:
            closefile();
            break;

        case 5:
            readfile();
            break;

        case 6:
            writefile();
            break;

        case 7:
            displayinfo();
            break;

        case 8:
            quit=1;
            break;

        default:
            break;
    }

    if(quit==1)break;
}
```

```
    savefile();  
}
```

