实验四 设备管理

- 课程：操作系统
- 学院：信息工程学院
- 专业班级：计科2002
- 学号：2020010505
- 姓名：薛飞宇

一、实习目的

1. 模拟操作系统对IO设备管理
2. 加深了解设备管理的工作原理
3. 理解设备管理的原理及数据结构
4. 理解“银行家”算法
5. 编写和调试设备管理模拟程序，掌握设备管理算法
6. 阅读“demo.c”，加深对设备管理的理解

二、实习内容

任务1：代码阅读并调试实验

任务2：文件系统设计实验

任务3：页结构分析

三、实习任务及完成情况

任务1：代码阅读并调试实验

1. **阅读下面源代码，写出程序功能。**

```
#define _CRT_SECURE_NO_WARNINGS 1
#include<stdio.h>
#include<stdlib.h>

int Available[10];            //可使用资源向量
int Max[10][10];              //最大需求矩阵
int Allocation[10][10] = { 0 };       //分配矩阵
int Need[10][10] = { 0 };             //需求矩阵
int Work[10];                    //工作向量
int Finish[10];                   //是否有足够的资源分配，状态标志
int Request[10][10];          //进程申请资源向量
int Pause[10];
int arr[] = { 0 };         //各类资源总数
int List[10];
int i, j;
int n;                        //系统资源总数
int m;                        //总的进程数
int a;                        //当前申请的进程号
int l, e;                      //计数器
int b = 0, c = 0, f = 0, g;            //计数器
int z = 0;
```

```
int  securitycheck()     //安全性检测
{
    printf("\n\n");
    printf("\t\t\t※ 安全性检测 ※\n\n");
    if (n == 3)
    {
        printf("           工作向量        尚需求量        已分配        工作向量+已分
配\n进程 ");
        for (c = 1; c <= 4; c++)
        {
            for (j = 1; j <= n; j++)
            {
                printf("  %d类", j);
            }
        }
    }
    if (n == 2)
    {
        printf("        工作向量    尚需求量    已分配    工作向量+已分配\n进程 ");
        for (c = 1; c <= 4; c++)
        {
            for (j = 1; j <= n; j++)
            {
                printf("  %d类", j);
            }
        }
    }
    for (i = 1; i <= m; i++)
    {
        Pause[i] = Available[i];    //Pause[i]只是一个暂时寄存的中间变量，为防止在
下面安全性检查时修改到Available[i]而代替的一维数组
        Finish[i] = false;
    }
    for (g = 1; g <= m; g++)
    {
        for (i = 1; i <= m; i++)
        {
            b = 0;                  //计数器初始化
            Finish[i] == false;
            for (j = 1; j <= n; j++)
            {
                if (Need[i][j] <= Pause[j])
                {
                    b = b + 1;
                }
                if (Finish[i] == false && b == n)
                {
                    Finish[i] = true;
                    printf("\nP[%d] ", i);        //依次输出进程安全序列
                    for (l = 1; l <= n; l++)
                    {
                        printf("  %2d ", Pause[l]);
                    }
                    for (j = 1; j <= n; j++)
                    {
                        printf("  %2d ", Need[i][j]);
                    }
```

```
                    for (j = 1; j <= n; j++)
                    {
                        //Allocation[i][j]=Pause[j]-Need[i][j];
                        printf("  %2d ", Allocation[i][j]);
                    }
                    for (j = 1; j <= n; j++)
                    {
                        printf("  %2d ", Pause[j] + Allocation[i][j]);
                    }
                    for (l = 1; l <= n; l++)
                    {
                        Pause[l] = Pause[l] + Allocation[i][l];
                    }
                }
            }
        }
    }

    printf("\n\n");
    for (i = 1; i <= m; i++)
    {
        if (Finish[i] == true) f = f + 1;              //统计Finish[i]==true
的个数
    }
    if (f == m)
    {
        printf("safe state");
        printf("\n\n系统剩余资源量:    ");
        for (i = 1; i <= n; i++)
        {
        printf("   %d ", Available[i]);
        }
        f = 0;        //将计数器f重新初始化，为下一次提出新的进程申请做准备
        return 1;
    }
    else
    {
        printf("unsafe state ");
        for (i = 1; i <= n; i++)
        {
            Available[i] = Available[i] + Request[a][i];
            Allocation[a][i] = Allocation[a][i] - Request[a][i];
            Need[a][i] = Need[a][i] + Request[a][i];
        }
        return 0;
    }

}

void initialize()    //初始化
{
    printf("请输入系统的资源种类数: ");
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
    {
        printf("第%d类资源总数: ", i);
        scanf("%d", &arr[i]);
    }
```

```
    printf("请输入进程总数：");
    scanf("%d", &m);
    for (i = 1; i <= m; i++)
    {
        for (j = 1; j <= n; j++)
        {
            printf("进程P[%d]对第%d类资源的最大需求量：", i, j);
            scanf("%d", &Max[i][j]);
        }
    }
    for (i = 1; i <= m; i++)
    {
        for (j = 1; j <= n; j++)
        {
            printf("进程P[%d]对第%d类资源已分配数：", i, j);
            scanf("%d", &Allocation[i][j]);
            Need[i][j] = Max[i][j] - Allocation[i][j];
        }
    }
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= m; j++)
        {
            arr[i] -= Allocation[j][i];
        }
    }
    for (i = 1; i <= n; i++)
        Available[i] = arr[i];
    securitycheck();
}

void mainrequest()  //进程申请资源
{
    printf("请输入申请资源的进程：");
    scanf("%d", &a);
    for (i = 1; i <= n; i++)
    {
        printf("请输入进程P[%d]对%d类资源的申请量：", a, i);
        scanf("%d", &Request[a][i]);
        if (Request[a][i] > Need[a][i])
        {
            printf("\n出错！进程申请的资源数多于它自己申报的最大需求量\n");
            return;
        }
        if (Request[a][i] > Available[i])
        {
            printf("\nP[%d]请求的资源数大于可用资源数，必须等待\n", a);
            return;
        }

    }
    for (i = 1; i <= n; i++)
    {
        //以下是试探性分配
        Available[i] = Available[i] - Request[a][i];
        Allocation[a][i] = Allocation[a][i] + Request[a][i];
        Need[a][i] = Need[a][i] - Request[a][i];
    }
```

```
    int ret=securitycheck();
    if (ret == 1)
    {
        int key = 0;
        for (j = 1; j <= n; j++)
        {
            if (Need[a][j] == 0)
            {
                key++;
            }
        }
        if (key == n)
        {
            for (j = 1; j <= n; j++)
            {
                Available[j] += Allocation[a][j];
                Allocation[a][j] = 0;
            }
        }
    }
}

void mainshow()
{
    printf("\n\n");
    if (n == 3)
    {
        printf("           已分配        最大需求量        尚需要量 \n进程");
    }
    if (n == 2)
    {
        printf("        已分配    最大需求   尚需要量 \n进程");
    }
    for (i = 1; i <= 3; i++)
    {
        for (j = 1; j <= n; j++)
        {
            printf("  %d类", j);
        }
    }
    for (i = 1; i <= m; i++)
    {
        printf("\nP[%d]", i);
        for (j = 1; j <= n; j++)
        {
            printf("  %2d ", Allocation[i][j]);
        }
        for (j = 1; j <= n; j++)
        {
            printf("  %2d ", Max[i][j]);
        }
        for (j = 1; j <= n; j++)
        {
            printf("  %2d ", Need[i][j]);
        }
    }
    printf("\n\n系统剩余资源量:    ");
    for (i = 1; i <= n; i++)
```

```
    {
        printf("   %d ", Available[i]);
    }
    printf("\n");
}

void menu()
{
    printf("\n\n\t\t卐卍※§     银行家算法     §※卐卍\n");
    printf("\n\n\t\t\t1:初始化");
    printf("\n  \t\t\t2:进程进行资源申请");
    printf("\n  \t\t\t3:资源分配状态");
    printf("\n  \t\t\t4:退出程序");
    printf("\n\n\t\t\t\t\t 请输入你的选择: ");
}
int main()
{
    int key = 0;
    printf("\n\n");
    while (1)
    {
        menu();
        scanf("%d", &key);
        printf("\n\n");
        switch (key)
        {
        case 1:
            initialize();
            break;
        case 2:
            mainrequest();
            break;
        case 3:
            mainshow();
            break;
        case 4:
            printf("\n\n\t\t\t谢谢使用 \n");
            printf("\n\t\t\tSee you next time !\n\n\n");
            system("pause");
            return 0;
        }

    }
    system("pause");
    return 0;
}
```

程序功能：模拟银行家算法，用户根据不同的需求选择要进行的操作，包括初始化、进程进行资源申请、资源分配状态和退出。首先由用户进行初始化，输入系统资源种类及其数目，并输入进程及其对资源的需求量和已获得资源数；进行安全性检测，如果安全，可以对某个进程分配适当的资源，可以分配则可以执行，再进行安全性检测。否则分配失败。

程序功能如下图：

2. **阅读代码，分析代码中算法并说明。**
   - 安全性检测算法：

     依次检测系统可使用的资源能否满足该进程使用，如果不能则等待。如果能则对该资源进行分配，将该进程设置为可以安全结束，并暂时回收该进程的资源。重复上述流程直到某个资源耗尽或所有进程都可以结束。

```
int  securitycheck()      //安全性检测
{
    printf("\n\n");
    printf("\t\t\t※ 安全性检测 ※\n\n");
    if (n == 3)
    {
        printf("          工作向量        尚需求量        已分配        工作向量+已分配\n进程 ");
        for (c = 1; c <= 4; c++)
        {
            for (j = 1; j <= n; j++)
            {
                printf("  %d类", j);
            }
        }
    }
    if (n == 2)
    {
        printf("        工作向量    尚需求量    已分配    工作向量+已分配\n进程 ");
        for (c = 1; c <= 4; c++)
        {
            for (j = 1; j <= n; j++)
            {
                printf("  %d类", j);
            }
        }
    }
    for (i = 1; i <= m; i++)
    {
        Pause[i] = Available[i];     //Pause[i]只是一个暂时寄存的中间变量，为防止在下面安全性检查时修改到Available[i]而代替的一维数组
        Finish[i] = false;
    }
    for (g = 1; g <= m; g++)
    {
```

```
        for (i = 1; i <= m; i++)
        {
            b = 0;                  //计数器初始化
            Finish[i] == false;
            for (j = 1; j <= n; j++)
            {
                if (Need[i][j] <= Pause[j])
                {
                    b = b + 1;
                }
                if (Finish[i] == false && b == n)
                {
                    Finish[i] = true;
                    printf("\nP[%d] ", i);         //依次输出进程安全序列
                    for (l = 1; l <= n; l++)
                    {
                        printf("  %2d ", Pause[l]);
                    }
                    for (j = 1; j <= n; j++)
                    {
                        printf("  %2d ", Need[i][j]);
                    }
                    for (j = 1; j <= n; j++)
                    {
                        //Allocation[i][j]=Pause[j]-Need[i][j];
                        printf("  %2d ", Allocation[i][j]);
                    }
                    for (j = 1; j <= n; j++)
                    {
                        printf("  %2d ", Pause[j] + Allocation[i][j]);
                    }
                    for (l = 1; l <= n; l++)
                    {
                        Pause[l] = Pause[l] + Allocation[i][l];
                    }
                }
            }
        }
    }

    printf("\n\n");
    for (i = 1; i <= m; i++)
    {
        if (Finish[i] == true) f = f + 1;              //统计Finish[i]==
true的个数
    }
    if (f == m)
    {
        printf("safe state");
        printf("\n\n系统剩余资源量:    ");
        for (i = 1; i <= n; i++)
        {
        printf("   %d ", Available[i]);
        }
        f = 0;        //将计数器f重新初始化，为下一次提出新的进程申请做准备
        return 1;
    }
    else
```

```
    {
        printf("unsafe state ");
        for (i = 1; i <= n; i++)
        {
            Available[i] = Available[i] + Request[a][i];
            Allocation[a][i] = Allocation[a][i] - Request[a][i];
            Need[a][i] = Need[a][i] + Request[a][i];
        }
        return 0;
    }

}
```

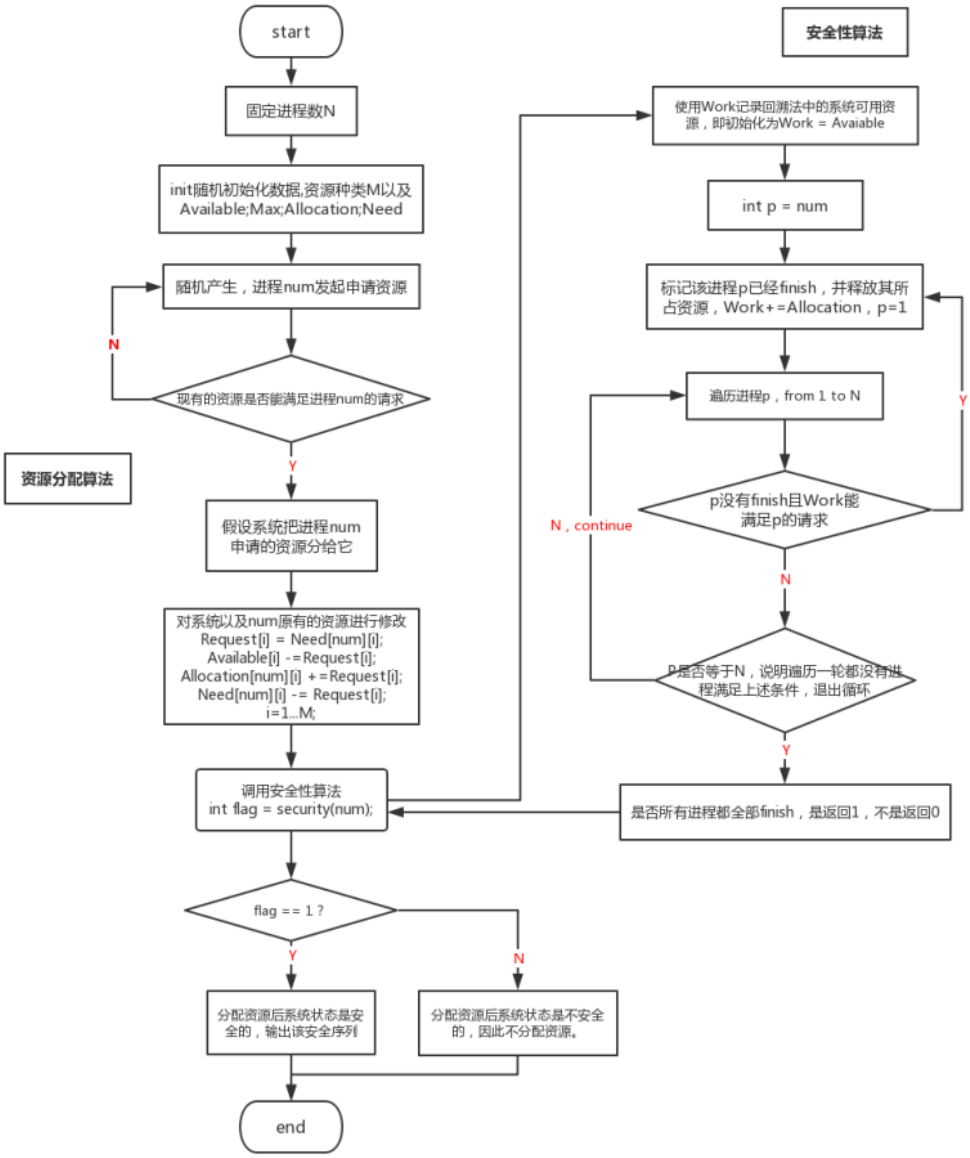
- 进程申请资源算法

  输入进程号及资源数目后，先判断其申请是否合理，如果不合理直接退出。尝试把系统可使用的资源分给该进程，并进行安全性检测，如果安全，则分配，否则不分配。

```
void mainrequest()  //进程申请资源
{
    printf("请输入申请资源的进程：");
    scanf("%d", &a);
    for (i = 1; i <= n; i++)
    {
        printf("请输入进程P[%d]对%d类资源的申请量：", a, i);
        scanf("%d", &Request[a][i]);
        if (Request[a][i] > Need[a][i])
        {
            printf("\n出错！进程申请的资源数多于它自己申报的最大需求量\n");
            return;
        }
        if (Request[a][i] > Available[i])
        {
            printf("\nP[%d]请求的资源数大于可用资源数，必须等待\n", a);
            return;
        }

    }
    for (i = 1; i <= n; i++)
    {
        //以下是试探性分配
        Available[i] = Available[i] - Request[a][i];
        Allocation[a][i] = Allocation[a][i] + Request[a][i];
        Need[a][i] = Need[a][i] - Request[a][i];
    }
    int ret=securitycheck();
    if (ret == 1)
    {
        int key = 0;
        for (j = 1; j <= n; j++)
        {
            if (Need[a][j] == 0)
            {
                key++;
            }
        }
        if (key == n)
```

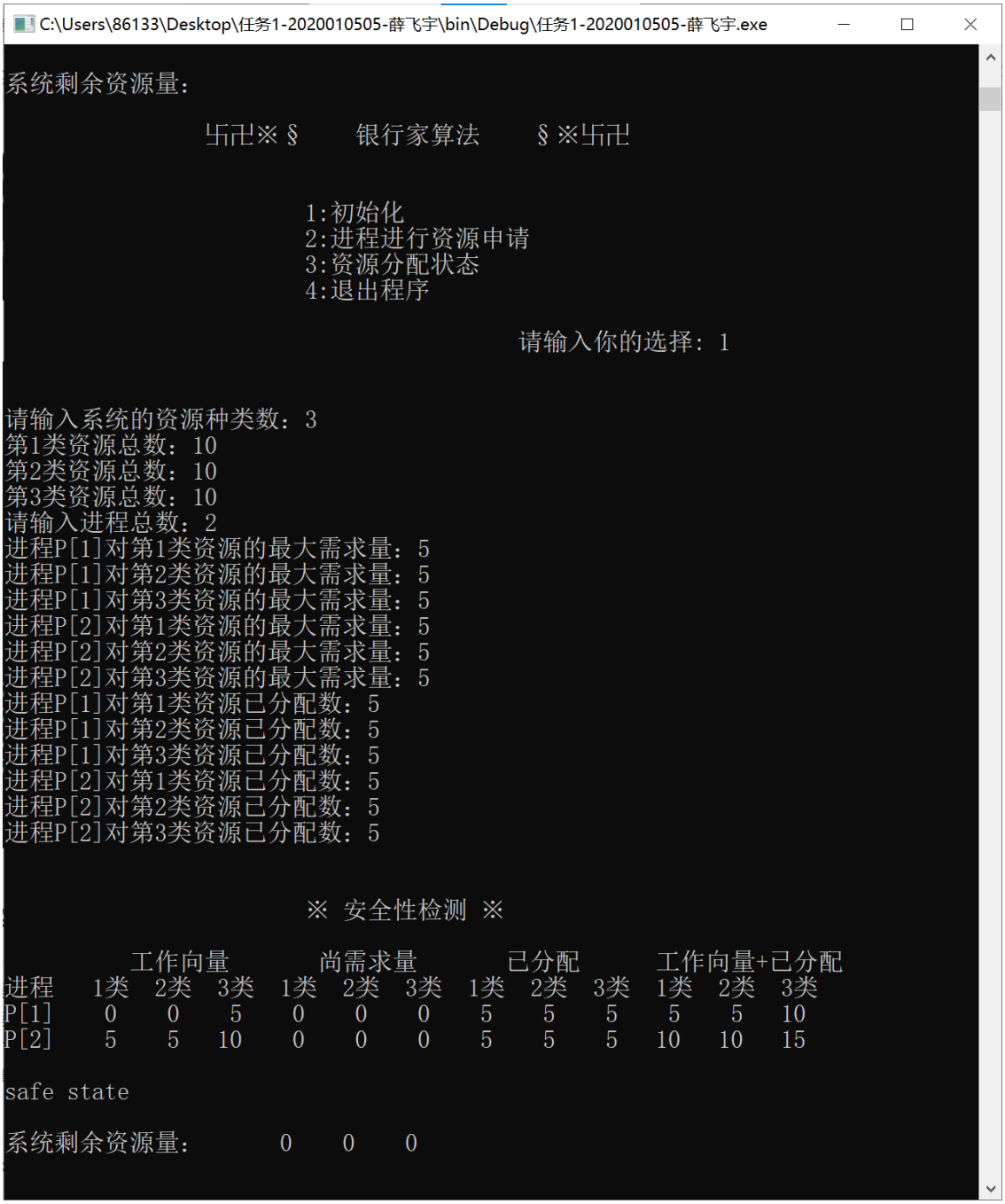
```
        {
            for (j = 1; j <= n; j++)
            {
                Available[j] += Allocation[a][j];
                Allocation[a][j] = 0;
            }
        }
    }
}

void mainshow()
{
    printf("\n\n");
    if (n == 3)
    {
        printf("           已分配        最大需求量        尚需要量 \n进
程");
    }
    if (n == 2)
    {
        printf("        已分配    最大需求   尚需要量 \n进程");
    }
    for (i = 1; i <= 3; i++)
    {
        for (j = 1; j <= n; j++)
        {
            printf("  %d类", j);
        }
    }
    for (i = 1; i <= m; i++)
    {
        printf("\nP[%d]", i);
        for (j = 1; j <= n; j++)
        {
            printf("  %2d ", Allocation[i][j]);
        }
        for (j = 1; j <= n; j++)
        {
            printf("  %2d ", Max[i][j]);
        }
        for (j = 1; j <= n; j++)
        {
            printf("  %2d ", Need[i][j]);
        }
    }
    printf("\n\n系统剩余资源量:    ");
    for (i = 1; i <= n; i++)
    {
        printf("   %d ", Available[i]);
    }
    printf("\n");
}
```

- 算法流程图:

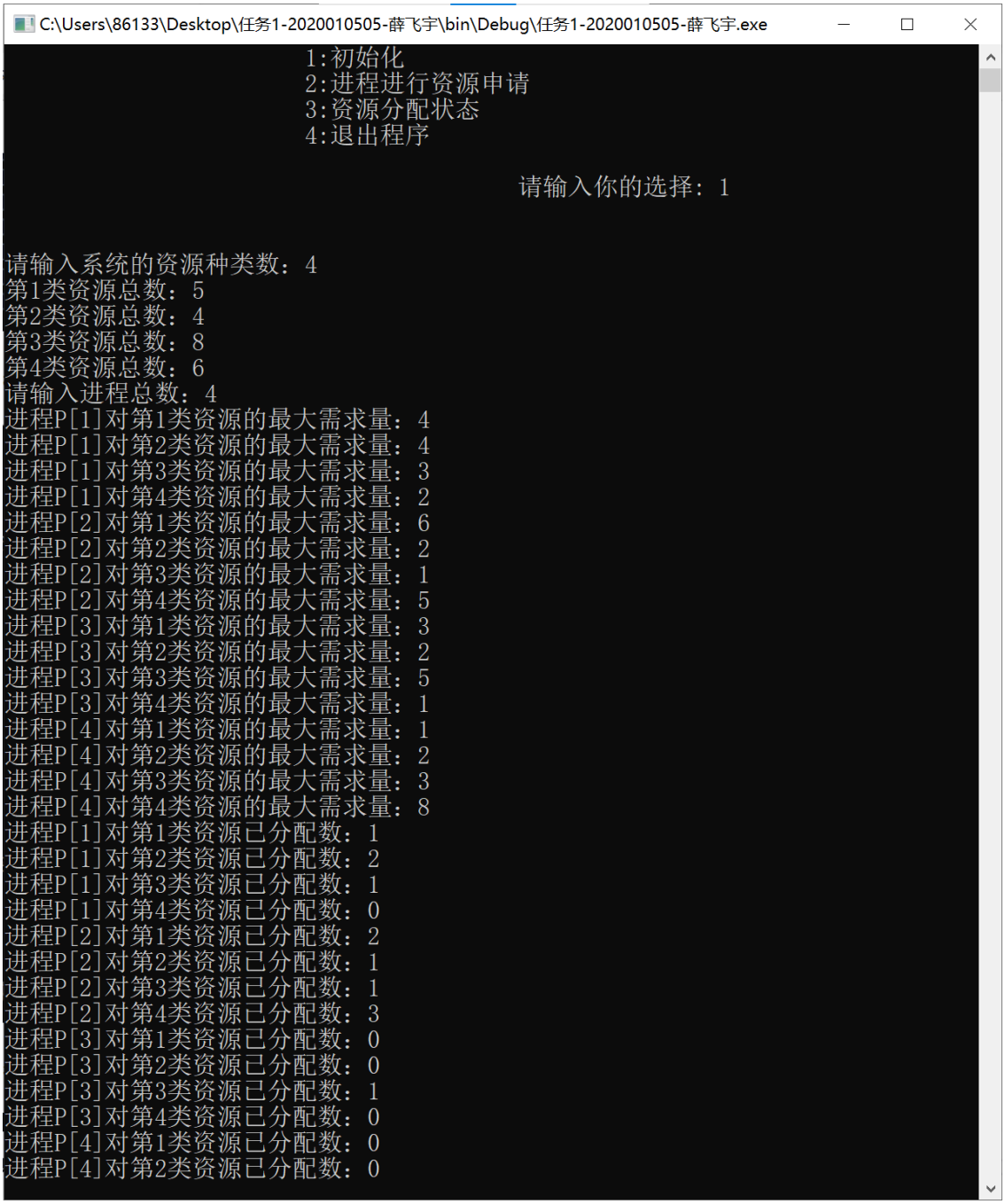


3. **调试并运行代码，写出结果。**

1.1 安全状态


C:\Users\86133\Desktop\任务1-2020010505-薛飞宇\bin\Debug\任务1-2020010505-薛飞宇.exe
系统剩余资源量:
银行家算法
1:初始化
2:进程进行资源申请
3:资源分配状态
4:退出程序
请输入你的选择: 1
请输入系统的资源种类数: 3
第1类资源总数: 10
第2类资源总数: 10
第3类资源总数: 10
请输入进程总数: 2
进程P[1]对第1类资源的最大需求量: 5
进程P[1]对第2类资源的最大需求量: 5
进程P[1]对第3类资源的最大需求量: 5
进程P[2]对第1类资源的最大需求量: 5
进程P[2]对第2类资源的最大需求量: 5
进程P[2]对第3类资源的最大需求量: 5
进程P[1]对第1类资源已分配数: 5
进程P[1]对第2类资源已分配数: 5
进程P[1]对第3类资源已分配数: 5
进程P[2]对第1类资源已分配数: 5
进程P[2]对第2类资源已分配数: 5
进程P[2]对第3类资源已分配数: 5
※ 安全性检测 ※
工作向量 尚需求量 已分配 工作向量+已分配
进程 1类 2类 3类 1类 2类 3类 1类 2类 3类 1类 2类 3类
P[1] 0 0 5 0 0 0 5 5 5 5 5 10
P[2] 5 5 10 0 0 0 5 5 5 10 10 15
safe state
系统剩余资源量: 0 0 0

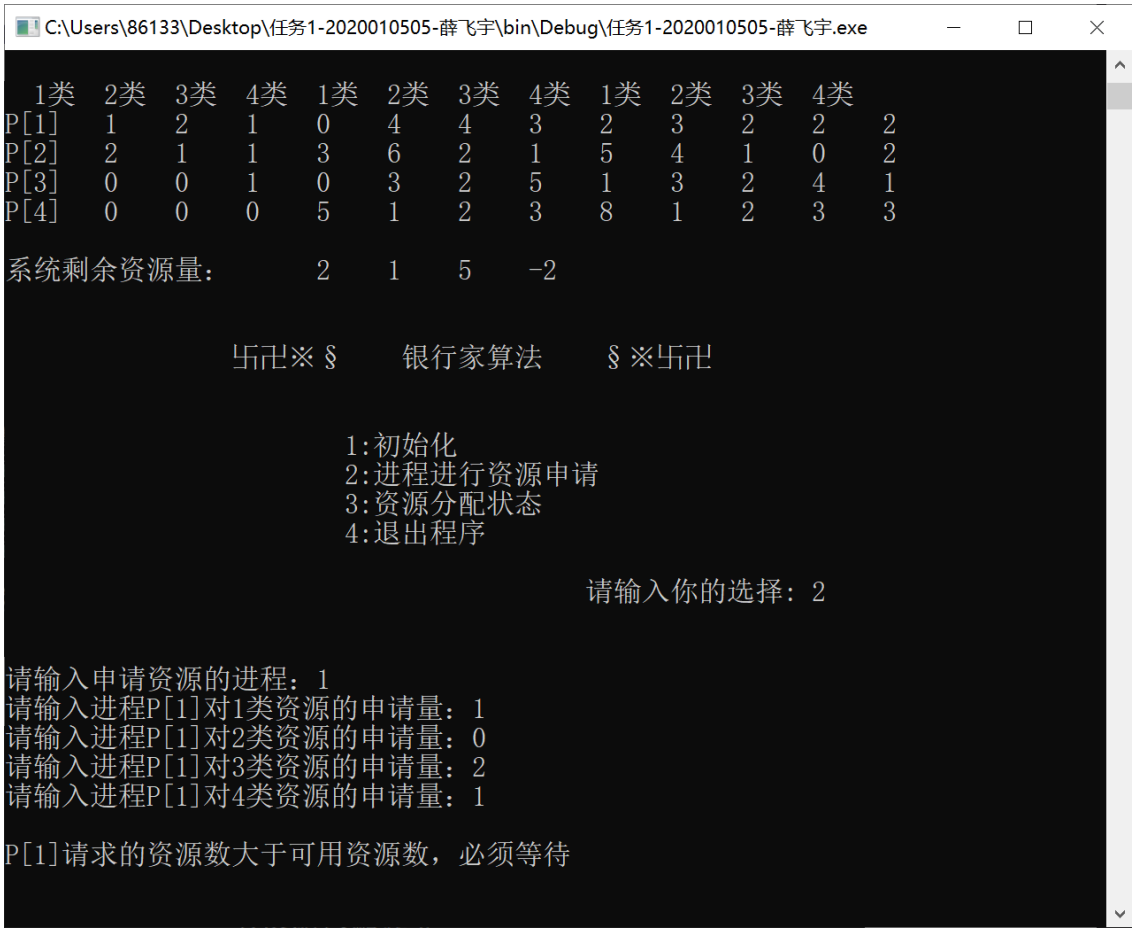
1.2 初始化



1.3 安全性检测及查询资源分配状态——不安全状态

1.4 进程进行资源申请

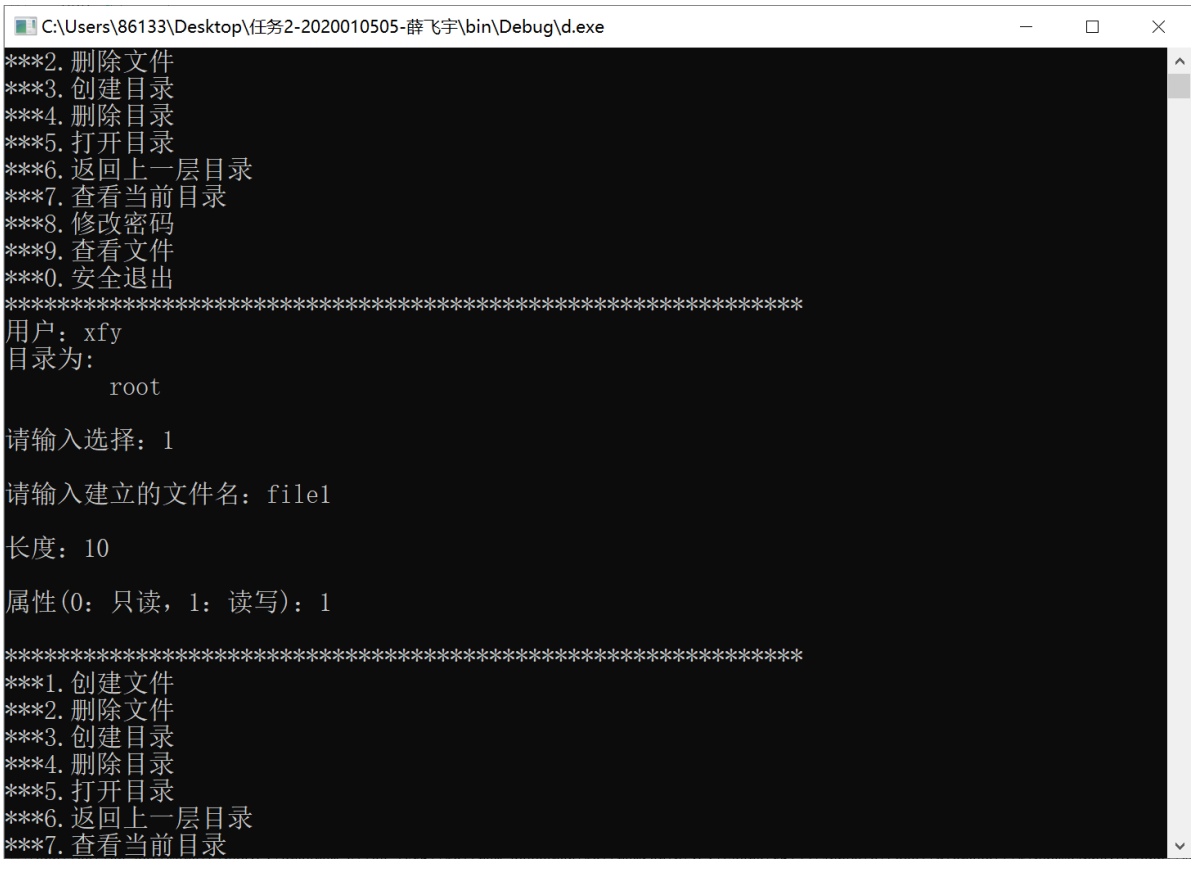
1.5 不安全的资源申请请求



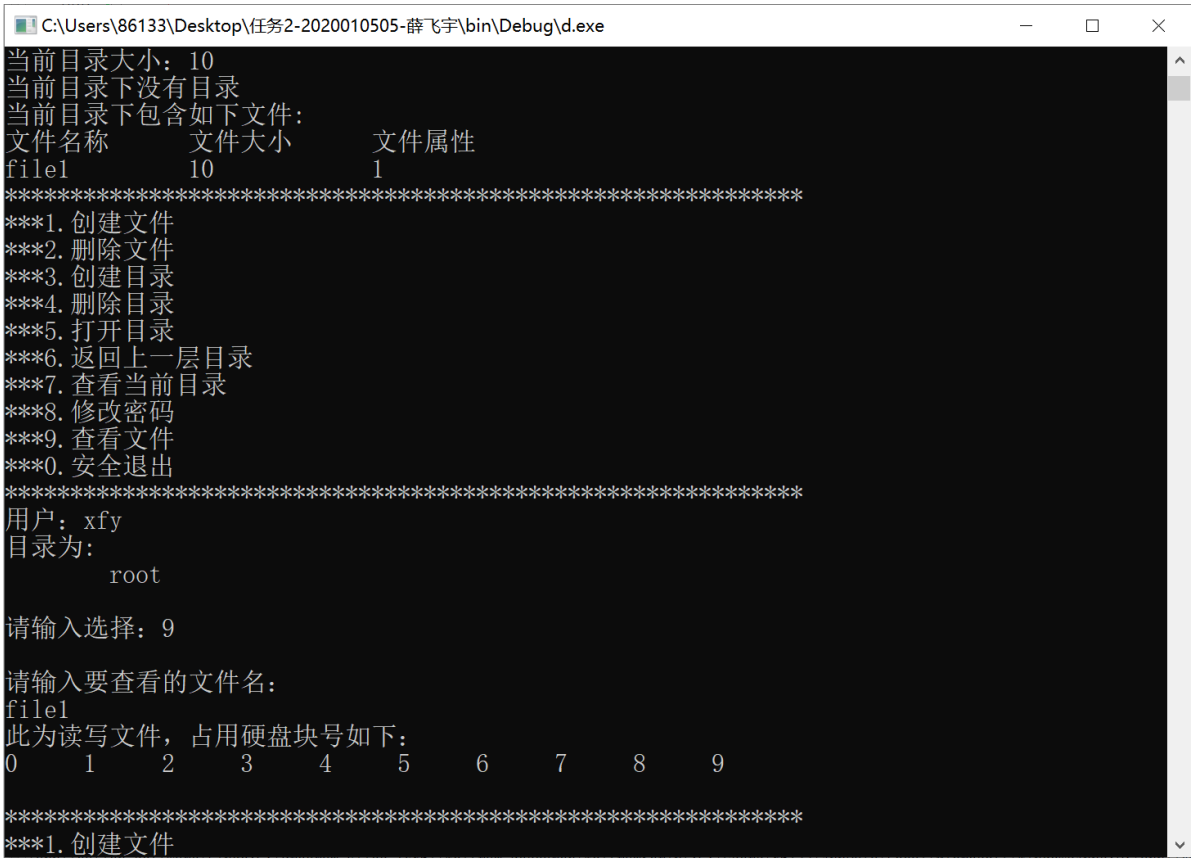
任务2：文件系统设计实验

测试程序

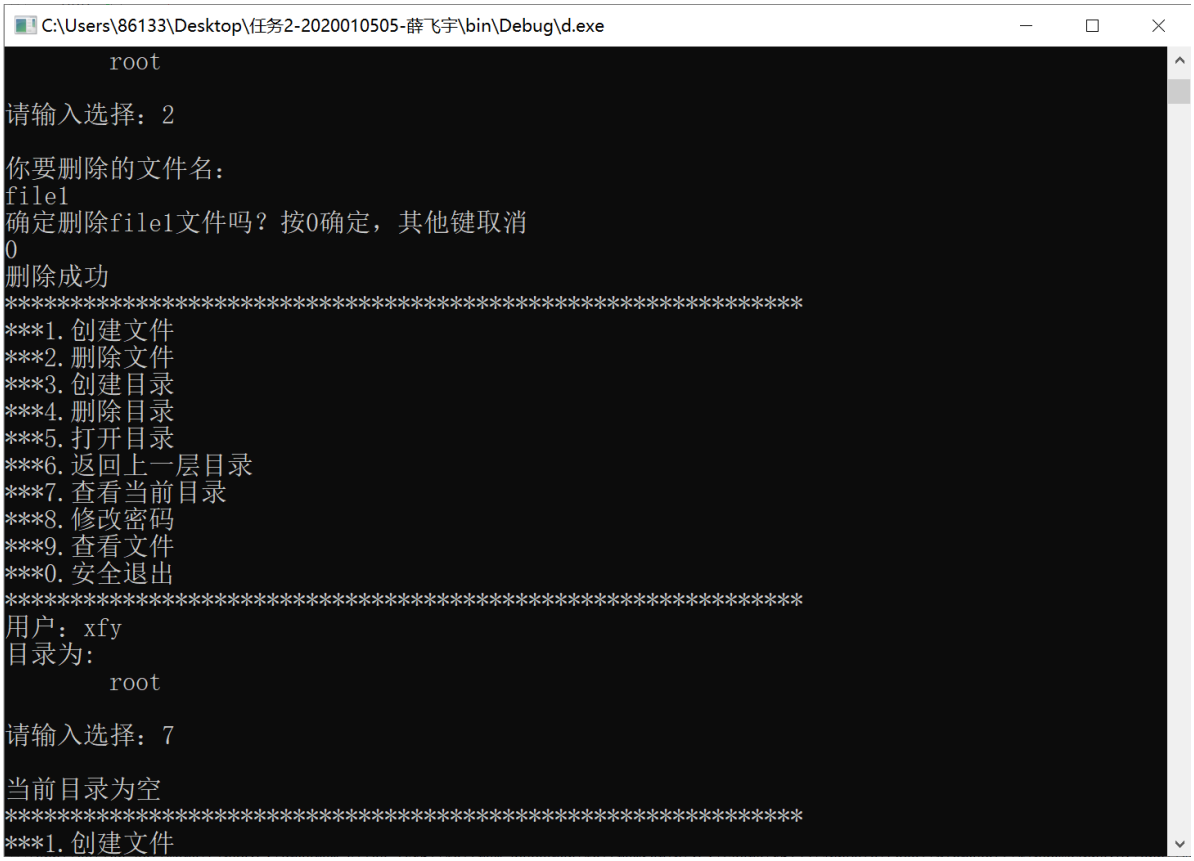
2.1 创建文件file1



2.2 查看当前目录下的文件file1



2.3 删除文件file1



源代码如下：

```
#include<string.h>
#include<stdlib.h>
#include<iostream>
#include<iomanip>
```

```
#if _MSC_VER>1000
#pragma once
#endif  //_MSC_VER>1000

using namespace std;
extern int disk_block[10000];
extern int disk_empty;
typedef struct UFD      //存储文件信息
{
    char name[10];      //文件名
    int attribute;      //属性
    int length;         //长度
    int a[10];      //为文件本身分配10个空间
    int *p1;            //一级索引，100个空间
    int (*p2)[100];     //二级索引，100*100个空间
    struct UFD *next;
}UFD;
typedef struct DIR      //存储目录信息
{
    DIR* above;         //上一结点
    char name[10];
    int length;
    DIR *next;          //下一结点
    UFD *File_head;     //此目录下的文件指针
    DIR *Dir_head;      //此目录下目录链表指针
}DIR;

class Cuse              //定义管理用户目录的类
{
    DIR *now;           //当前目录
    UFD *Fhead;         //文件的头结点
    DIR *Dhead;         //根目录的目录链头结点
    char code[10];      //密码
    char name[10];      //用户名
    int length;         //用户空间大小
    int status;         //是否获得空间
public:
    void set_status(int);
    int dele_user();
    int dis_file();     //显示文件所占外存块号
    int dis_dir(DIR *d);//当前路径
    int get_length();
    char const *get_name();
    char const *get_code();
    int get_status();
    int set_user(char *,char *);//设置用户名与密码
    DIR *get_now();
    int dele_file(UFD *f);      //删除文件
    int dele_dir(DIR*);         //删除目录
    Cuse();                     //构造
    ~Cuse();                    //析构
    int goback();               //返回上级
    int dis_now();              //显示当前目录
    int new_file();
    int new_dir();
    int open_dir();
    int open_file();
    int first_dele_file();      //删除文件的前部分工作
```

```
    int first_dele_dir();        //删除目录的前部分工作
    int set_code();
};

class Cdisk{                     //用户类
public:
    Cuse user[5];        //用户个数最多为5
    char code[10];
    int dis_disk();
    int first_dele_user();
    int dele_user(int);
    int new_user();              //查看当前用户与外存空间使用情况，后创建新用户
    int set_code();              //设置新密码
    int login(char);                 //登陆
    Cdisk();
    virtual~Cdisk();             //虚函数，析构
};

int disk_block[10000];
int disk_empty;
Cdisk::Cdisk()       //管理磁盘的类，构造函数
{
    int i=0;
    char code[10]="123456";
    for(i=0;i<10000;i++)         //初始化所有磁盘块为空闲
        disk_block[i]=0;
    //this->user[0].set_user("student","123");//默认一个用户
    disk_empty=10000;
    cout.setf(ios::left);    //设置输出方式
}
Cdisk::~Cdisk()      //析构
{
}

int Cdisk::dele_user(int i)      //Cdisk类dele_user的构造
{
    Cuse C;
    C=user[i];
    user[i].dele_user();         //调用Cuse类的成员函数 int dele_user()
    return 1;
}

int Cdisk::dis_disk()        //检查磁盘信息
{
    int i=0;
    cout<<setw(14)<<"用户名"<<setw(14)<<"占用空间大小"<<endl;
    for(i=0;i<5;i++)
        if(user[i].get_status()==1)      //存在的用户的信息
            cout<<setw(14)<<user[i].get_name()<<setw(14)<<user[i].get_length()
<<endl;
        cout<<"已用空间："<<10000-disk_empty<<endl<<"剩余空间："<<disk_empty<<endl;
        return 1;
}
int Cdisk::login(char b)         //登陆
{
    char n[10],c[10];
    int i;
    if(b=='1')
```

```
    {
        cout<<"用户：管理员"<<endl;
        cout<<"密码：默认\n"<<endl;
        system("pause");
        return 1;
    }
    else
    {
        if(!user[0].get_status())   //当前不存在用户
        {
            i=0;
            cout<<"当前用户为空，欢迎注册！"<<endl;
            user[i].set_status(1);      //为新用户分配权利
            cout<<"请输入用户名："<<endl;
            cin>>n;
            cout<<"请输入密码："<<endl;
            cin>>c;
            user[i].set_user(n,c);      //调用Cuse的成员函数，传递用户名与密码

            cout<<"恭喜，创建用户成功！"<<endl;
            return i;
        }
        else
        {
            cout<<"用户名:";
            cin>>n;
            cout<<"密码：";
            cin>>c;
            cout<<endl;
            for(i=0;i<5;i++)    //查找是否存在此用户
            {
                if(user[i].get_status())        //存在方比较
                    if(!strcmp(n,user[i].get_name()))   //相等时为0，此判断为匹配
                        if(!strcmp(c,user[i].get_code()))       //密码匹配
                        {
                            cout<<"登陆成功！"<<endl;
                            cout<<"欢迎"<<user[i].get_name()<<"登陆"<<endl;
                            return i;
                        }
                        else
                        {
                            cout<<"密码错误"<<endl;
                            return -1;
                        }
            }
            cout<<"此用户不存在！"<<endl;
            return -1;
        }
    }
}

int Cdisk::set_code()       //设置新密码
{
    char temp1[10],temp2[10];
    cout<<"请输入原密码"<<endl;
    cin>>temp1;
    if(strcmp(temp1,code))      //无疑是针对当前用户进行操作，故直接code
    {
```

```
        cout<<"原密码错误！"<<endl;
        return 0;
    }
    while(1)
    {
        cout<<"请输入新密码："<<endl;
        cin>>temp1;
        cout<<"请再次输入新密码："<<endl;
        cin>>temp2;
        if(strcmp(temp1,temp2))
        {
            cout<<"两次密码不相同，请重输！"<<endl;
            break;
        }
        cout<<"密码设置成功！"<<endl;
        strcpy(code,temp1);     //保存新密码
        break;
    }
    return 1;
}

int Cdisk::new_user()       //准备创建新用户
{
    char n[10],c[10];
    int i=0,j;
    for(i=0;i<5;i++)
        if(user[i].get_status()==0)     //是否有此用户，此尚未存在
            break;
        if(i==5)
        {
            cout<<"已经达到最大用户5个，不能再创建！"<<endl;
            return 0;
        }
        user[i].set_status(1);      //为新用户分配权利
        cout<<"请输入用户名："<<endl;
        cin>>n;
        if(i>0)         //已有其它用户存在
        {
            for(j=0;j<i-1;j++)
                if(!strcmp(user[j].get_name(),n))
                {
                    cout<<"此用户名已存在！"<<endl;
                    return 0;
                }
        }
        cout<<"请输入密码："<<endl;
        cin>>c;
        user[i].set_user(n,c);      //调用Cuse的成员函数，传递用户名与密码
        cout<<"恭喜，创建用户成功！"<<endl;
        return 1;
}

int Cdisk::first_dele_user()        //删除用户
{
    char n[10],c;
    int i;
    cout<<"请输入你要删除的用户名"<<endl;
    cin>>n;
```

```
    for(i=0;i<5;i++)        //在查找用户的同时，得到用户序号i
        if(!strcmp(user[i].get_name(),n)&&user[i].get_status())
            break;      //找到，跳出
        if(i==5)
        {
            cout<<"出错啦，此用户不存在！"<<endl;
            return 0;
        }
        cout<<"确认删除此用户？确认Y，取消任意键"<<endl;
        cin>>c;
        if(c!='Y'&&c!='y')
        {
            cout<<"已经取消删除！"<<endl;
            return 0;
        }
        this->dele_user(i);
        cout<<"用户删除成功"<<endl;
        return 1;
}

Cuse::Cuse()        //构造函数，初始化成员
{
    status=0;       //用户权利，即是否被创建标记
    length=0;       //空间
    now=0;          //当前目录
    Fhead=0;        //文件
    Dhead=0;        //目录
}

Cuse::~Cuse()       //析构，清除程序占用的内存
{
    disk_empty+=length;
    length=0;
    UFD *f=Fhead;
    DIR *d=Dhead;
    while(f!=0)     //文件
    {
        if(f->next==0)
        {
            this->dele_file(f);
            f=0;
            break;
        }
        while(f->next->next!=0)
            f=f->next;
        this->dele_file(f->next);
        f->next=0;
        f=Fhead;
    }
    while(d!=0)     //目录
    {
        if(d->next==0)
        {
            this->dele_dir(d);
            d=0;
            break;
        }
        while(d->next->next!=0)
```

```
            d=d->next;
        this->dele_dir(d->next);
        d->next=0;
        d=Dhead;
    }
}


int Cuse::new_file()        //建立新文件
{
    int i=0,j=0;
    UFD *f,*p=0;
    DIR *D;

    p=new UFD;              //开辟一个新的文件结构体
    if(p==0)
    {
        cout<<"无可用内存空间，创建文件失败！"<<endl;
        return 1;
    }
    cout<<"请输入建立的文件名：";
    cin>>p->name;
    if(now==0)          //根目录下的文件链
        f=Fhead;
    else                //当前目录下的文件链
        f=now->File_head;
    while(f!=0)         //检查是否文件重名
    {
        if(!strcmp(p->name,f->name))
        {
            cout<<"此文件已存在！"<<endl;
            return 0;   //退出
        }
        f=f->next;
    }
    cout<<"\n"<<"长度：";
    cin>>p->length;
    cout<<"\n"<<"属性(0：只读，1：读写)：";
    cin>>p->attribute;
    cout<<endl;
    if(p->length>disk_empty)        //空间不足
    {
        cout<<"文件太大，空间不足，当前空间为："<<disk_empty<<endl;
        delete p;
        return 0;
    }
    disk_empty=disk_empty-p->length;    //剩余空闲盘块
    //
    for(i=0;i<p->length&&i<10;i++)      //文件较小时，直接地址，文件数据盘块号
        for(j;j<10000;j++)              //位示图法
            if(disk_block[j]==0)        //空闲
            {
                p->a[i]=j;              //得到此空间
                disk_block[j]=1;        //标记为已分配出去
                j++;
                break;                  //跳出寻找，为文件分配下一空间
            }
            p->p1=0;                    //一级索引
```

```
            p->p2=0;                        //二级索引
            if(p->length>10)                //超过10，用一级索引
            {
                p->p1=new int[100];         //为一级索引分配100个空间
                for(i=10;i<p->length&&i<110;i++)
                    for(j;j<10000;j++)      //j，继续前面的值
                        if(disk_block[j]==0)
                        {
                            (p->p1)[i-10]=j;
                            disk_block[j]=1;
                            j++;
                            break;
                        }
                        if(p->length>110)           //超过110，得用二级索引
                        {
                            p->p2=new int[100][100];    //在一级索引的基础上，2维
                            for(i=110;i<p->length;i++)
                                for(j;j<10000;j++)      //j，继续前面的值
                                    if(disk_block[j]==0)
                                    {
                                        int m=(i-110)/100;      //行
                                        int k=(i-110)%100;      //列
                                        p->p2[m][k]=j;
                                        disk_block[j]=1;
                                        j++;
                                        break;
                                    }
                        }
            }
            if(now==0)  //根目录下的文件
            {
                p->next=Fhead;  //后继结点指向头，即新指点在前
                Fhead=p;    //新结点在头
            }
            else
            {
                p->next=now->File_head;
                now->File_head=p;
            }
            length+=p->length;  //用户总空间大小增加
            if(now!=0)          //子目录下空间大小增加
            {
                now->length+=p->length;
                D=now->above;   //上一级目录
                while(D!=0)     //上级目录（根目录已实现）空间增加
                {
                    D->length+=p->length;
                    D=D->above;     //逐级向上
                }
            }

            return 0;
}

int Cuse::new_dir()     //建立新目录
{
    DIR *p,*h;
    cout<<"请输入新目录的名字："<<endl;
```

```
    p=new DIR;
    cin>>p->name;        //目录名
    p->Dir_head=0;       //目录下的目录链为空
    p->length=0;         //
    p->File_head=0;      //目录下的文件为空
    if(now==0)           //当前为主目录
        h=Dhead;     //第一次时，h=0；指向目录链
    else
        h=now->Dir_head;//当前的目录链
    while(h!=0) //此目录下存在其它子目录
    {
        if(!strcmp(h->name,p->name))
        {
            cout<<"此目录已存在！"<<endl;
            return 0;
        }
        h=h->next;
    }
    if(now==0)       //当前为主目录
    {
        p->above=0;      //主目录里目录的上一结点为0
        p->next=Dhead;       //把原目录接入新目录后面（Dhead初始为0）
        Dhead=p;         //Dhead始终指向最新目录结点
    }
    else
    {
        p->above=now;        //当前目录为新目录的上一结点
        p->next=now->Dir_head;  //反序插入新目录
        now->Dir_head=p;     //更新目录链
    }
    cout<<"目录创建成功"<<endl;
    return 1;
}

int Cuse::goback()       //向上返回
{
    if(now==0)
    {
        cout<<"已是主目录，不能向上！"<<endl;
        return 0;
    }
    now=now->above;      //上一结点
    return 1;
}

int Cuse::open_dir()     //打开目录
{
    char name[10];
    DIR *p;
    if(now==0)  //当前主目录
        p=Dhead;
    else
        p=now->Dir_head;     //p指向目录链
    cout<<"请输入你要打开的目录名："<<endl;
    cin>>name;
    //int flag=0;
    while(p!=0)
    {
```

```
        if(strcmp(p->name,name)==0)
        {
            now=p;  //找到目录，now标记
            return 1;
        }
        p=p->next;
    }
    cout<<"当前没此目录"<<endl;
    return 0;
}

int Cuse::first_dele_file() //删除文件的前面工作
{
    char temp[10],a[5];
    cout<<"你要删除的文件名："<<endl;
    cin>>temp;
    UFD *f=Fhead;   //数据文件头指针
    UFD *above=0;
    if(now!=0)
        f=now->File_head;   //当前目录下的数据文件
    while(f!=0)
    {
        if(!strcmp(f->name,temp))
            break;  //找到，跳出
        above=f;    //标记第一个文件
        f=f->next;
    }
    if(f==0)
    {
        cout<<"此文件不存在"<<endl;
        return 0;
    }
    cout<<"确定删除"<<f->name<<"文件吗？按0确定，其他键取消"<<endl;
    cin>>a;
    if(a[0]!='0')
    {
        cout<<"已取消删除文件。"<<endl;
        return 0;
    }
    disk_empty+=f->length;  //回收此数据文件的空间大小
    if(now==0)  //根目录
    {
        if(f==Fhead)
            Fhead=Fhead->next;
        else
            above->next=f->next;
    }
    else
    {
        DIR *d=now;
        while(d!=0)
        {
            d->length=f->length;
            d=d->above;
        }
        if(f==now->File_head)
            now->File_head=now->File_head->next;
        else
```

```
            above->next=f->next;
    }
    length-=f->length;
    this->dele_file(f);
    cout<<"删除成功"<<endl;
    return 1;
}

int Cuse::dele_file(UFD *f) //具体实现删除文件
{
    int i=0,m;
    for(i=0;i<10&&i<f->length;i++)  //回收文件具体空间，重置空闲
    {
        m=f->a[i];
        disk_block[m]=0;
    }
    if(f->p1!=0)    //回收一级索引
    {
        for(i=10;i<110&&i<f->length;i++)
        {
            m=f->p1[i-10];
            disk_block[m]=0;
        }
        delete[](f->p1);
    }
    if(f->p2!=0)    //回收二级索引
    {
        for(i=110;i<f->length;i++)
        {
            m=(f->p2)[(i-110)/100][(i-110)%100];
            disk_block[m]=0;
        }
        delete[](f->p2);
        delete f;
    }
    f=0;    //要删除的文件为空了
    return 1;
}

int Cuse::first_dele_dir()  //删除目录的前奏
{
    char n[10];
    char c;
    DIR *p,*above=0;
    p=Dhead;    //指向根目录下的目录链
    if(now!=0)
        p=now->Dir_head;    //指向当前目录下的目录链
    cout<<"要删除的目录名："<<endl;
    cin>>n;
    while(p!=0)
    {
        if(!strcmp(p->name,n))
            break;  //找到要删除的目录，跳出
        above=p;    //保存前一结点
        p=p->next;
    }
    if(p==0)
    {
```

```
        cout<<"没有这个目录！"<<endl;
        return 0;
    }
    cout<<"确定删除当前目录及此目标上的所有信息吗？按0确定，其他键取消"<<endl;
    cin>>c;
    if(c!='0')
        return 0;
    disk_empty+=p->length;  //回收磁盘空间
    if(now==0)
    {
        if(p==Dhead)    //如果是根目录下头结点，直接移动Dhead
            Dhead=Dhead->next;
        else
            above->next=p->next;//中间抽掉目标
    }
    else
    {
        if(p==now->Dir_head)
            now->Dir_head=now->Dir_head->next;
        else
            above->next=p->next;
        above=now;  //当前目录
        while(above!=0)
        {
            above->length-=p->length;//用above标记当前目录，进行大小更新
            above=above->above; //向上一级
        }
    }
    length-=p->length;  //根目录大小更新
    this->dele_dir(p);
    p=0;
    cout<<"删除成功！"<<endl;
    return 1;
}

int Cuse::dele_dir(DIR *p)  //具体实现删除目录的工作
{
    int flag=0;
    DIR *d=p->Dir_head; //当前目录下的目录链表
    UFD *f=p->File_head;    //当前目录下的文件链
    if(f!=0)
    {
        while(p->File_head->next!=0)//删除目录里的文件
        {
            f=p->File_head;
            while(f->next->next!=0)
                f=f->next;
            this->dele_file(f->next);
            f->next=0;
        }
        if(p->File_head->next==0)
        {
            this->dele_file(p->File_head);//删除头文件
            p->File_head=0;
        }
    }
    if(d!=0)    //删除目录下的目录
    {
```

```
        while(p->Dir_head->next!=0)
        {
            d=p->Dir_head;
            while(d->next->next!=0)
                d=d->next;
            this->dele_dir(d->next);
            d->next=0;
        }
        if(p->Dir_head->next==0)
        {
            this->dele_dir(p->Dir_head);//删除目录链头结点
            p->Dir_head=0;
        }
    }
    delete p;   //释放p占用的内存
    p=0;        //置0
    return 1;
}

int Cuse::dis_now() //显示当前目录
{
    DIR *d=Dhead;
    UFD *f=Fhead;
    if(now!=0)
    {
        d=now->Dir_head;    //当前目录下的目录链
        f=now->File_head;
    }
    if(d==0&&f==0)
    {
        cout<<"当前目录为空"<<endl;
        return 0;
    }
    cout<<"当前目录大小：";
    if(now==0)
        cout<<length;
    else
        cout<<now->length;
    cout<<endl;
    if(d==0)
        cout<<"当前目录下没有目录"<<endl;
    else
    {
        cout<<"当前目录下包含如下目录："<<endl;
        cout<<setw(14)<<"目录名称"<<setw(14)<<"目录大小"<<endl;
        while(d!=0)
        {
            cout<<setw(14)<<d->name<<setw(14)<<d->length<<endl;
            d=d->next;
        }
    }
    if(f==0)
        cout<<"当前目录下没有文件"<<endl;
    else
    {
        cout<<"当前目录下包含如下文件:"<<endl;
        cout<<setw(14)<<"文件名称"<<setw(14)<<"文件大小"<<setw(14)<<"文件属性"<<endl;
        while(f!=0)
```

```
        {
            cout<<setw(14)<<f->name<<setw(14)<<f->length<<setw(14)<<f-
>attribute<<endl;
            f=f->next;
        }
    }
    return 1;
}

int Cuse::open_file()   //打开文件
{
    char n[10];
    cout<<"请输入要打开的文件名"<<endl;
    cin>>n;
    UFD *f=Fhead;   //文件头指针
    if(now!=0)
        f=now->File_head;   //指向文件
    while(f!=0)
    {
        if(!strcmp(f->name,n))
        {
            cout<<"文件打开成功"<<endl;
            return 1;
        }
        f=f->next;
    }
    cout<<"当前目录无此文件"<<endl;
    return 0;
}

int Cuse::set_code()    //设置密码
{
    char a1[10],a2[10];
    cout<<"请输入原密码"<<endl;
    cin>>a1;
    if(strcmp(a1,code))
    {
        cout<<"密码错误"<<endl;
        return 0;
    }
    while(1)
    {
        cout<<"请输入新密码："<<endl;
        cin>>a1;
        cout<<"再次输入新密码:"<<endl;
        cin>>a2;
        if(strcmp(a1,a2))
            cout<<"两次输入密码不同，请重输！"<<endl;
        else
        {
            strcpy(code,a1);
            cout<<"密码修改成功！"<<endl;
            break;
        }
    }
    return 1;
}
```

```
DIR *Cuse::get_now()    //得到当前目录路径
{
    return now;
}

int Cuse::set_user(char *n,char *c)//建立用户与密码
{
    strcpy(name,n);
    strcpy(code,c);
    status=1;
    return 1;
}

void Cuse::set_status(int b)//标记分配
{
    status=b;
}

int Cuse::get_status()//探取是否分配
{
    return status;
}

const char* Cuse::get_code()//得到密码
{
    return code;
}

const char* Cuse::get_name()//得到名字
{
    return name;
}

int Cuse::get_length()//得到长度
{
    return length;
}

int Cuse::dis_dir(DIR *d)//显示目录
{
    if(d==0)
        return 0;
    if(d->above!=0)
        this->dis_dir(d->above);//递归调用此功能
    cout<<" "<<d->name<<'\n';
    return 0;
}

int Cuse::dis_file()//查看文件
{
    int i;
    char n[10];
    UFD *f=Fhead;
    if(now!=0)
        f=now->File_head;
    cout<<"请输入要查看的文件名："<<endl;
    cin>>n;
    while(f!=0)
```

```
    {
        if(!strcmp(n,f->name))  //找到此文件，跳出
            break;
        f=f->next;
    }
    if(f==0)
    {
        cout<<"当前目录下没此文件"<<endl;
        return 0;
    }
    if(f->attribute==0)
        cout<<"此为只读文件，";
    else
        cout<<"此为读写文件，";
    cout<<"占用硬盘块号如下："<<endl;
    for(i=0;i<f->length&&i<10;i++)      //直接存放的
    {
        cout<<setw(6)<<f->a[i];
        if((i+1)%10==0)
            cout<<endl;
    }
    for(i=10;i<f->length&&i<110;i++)    //一级索引存放的
    {
        cout<<setw(6)<<f->p1[i-10];
        if((i+1)%10==0)
            cout<<endl;
    }
    for(i=110;i<f->length;i++)  //二级索引存放的
    {
        cout<<setw(6)<<f->p2[(i-110)/100][(i-110)%100];
        if((i+1)%10==0)
            cout<<endl;
    }
    cout<<endl;
    return 1;
}

int Cuse::dele_user()//删除用户
{
    length=0;   //用户空间置0
    Fhead=0;
    Dhead=0;
    now=0;
    status=0;
    return 1;
}


int main()
{
    char c;
    Cdisk D;        //管理员类的对象
    int i=1,n,flag=1,t=0;

    while(flag)
    {
        cout<<"******************欢迎使用文件系统*************************"
<<endl;
```

```
        cout<<endl;
        cout<<"***                  1.管理员登陆"<<endl;
        cout<<"***                  2.用户登陆"<<endl;
        cout<<"***                  3.退出系统"<<endl;
        cout<<endl;
        cout<<"*****************************************************************"
<<endl;

        cout<<"\n请输入选择："";
        cin>>c;
        switch(c)
        {
        case '1':
            n=D.login(c);
            flag=1;
            system("cls");
            cout<<"管理员登陆成功！"<<endl;
            while(flag)
            {

cout<<"*****************************************************************"<<endl;
                cout<<"***1.创建用户"<<endl;
                cout<<"***2.删除用户"<<endl;
                cout<<"***3.查看当前用户"<<endl;
                cout<<"***4.修改密码"<<endl;
                cout<<"***5.返回登陆窗口"<<endl;

cout<<"*****************************************************************"<<endl;
                cout<<"请选择："";
                cin>>c;
                cout<<endl;
                switch(c){
                case '1':
                    D.new_user();
                    break;
                case '2':
                    D.first_dele_user();
                    break;
                case '3':
                    D.dis_disk();
                    break;
                case '4':
                    {
                        if(t==0)    //t作标记，防止重复设置密码出错
                            strcpy(D.code,"123");
                        D.set_code();
                        t++;
                        break;
                    }
                case '5':
                    {
                        flag=0;
                        system("cls");
                    }
                    break;
                default:
                    cout<<"请输入1-5！"<<endl;
                }
```

```
            }
            flag=1;
            break;
        case '2':
            n=D.login(c);
            if(n==-1)
                break;
            while(flag)
            {

cout<<"*****************************************************************"<<endl;
                cout<<"***1.创建文件"<<endl;
                cout<<"***2.删除文件"<<endl;
                cout<<"***3.创建目录"<<endl;
                cout<<"***4.删除目录"<<endl;
                cout<<"***5.打开目录"<<endl;
                cout<<"***6.返回上一层目录"<<endl;
                cout<<"***7.查看当前目录"<<endl;
                cout<<"***8.修改密码"<<endl;
                cout<<"***9.查看文件"<<endl;
                cout<<"***0.安全退出"<<endl;

cout<<"*****************************************************************"<<endl;
                cout<<"用户："<<D.user[n].get_name()<<'\n'<<"目录为:\n    root\n";
                D.user[n].dis_dir(D.user[n].get_now());
                cout<<endl;
                cout<<"请输入选择：";
                cin>>c;
                cout<<endl;
                switch(c)
                {
                case '1':
                    D.user[n].new_file();
                    break;
                case '2':
                    D.user[n].first_dele_file();
                    break;
                case '3':
                    D.user[n].new_dir();
                    break;
                case '4':
                    D.user[n].first_dele_dir();
                    break;
                case '5':
                    D.user[n].open_dir();
                    break;
                case '6':
                    D.user[n].goback();
                    break;
                case '7':
                    D.user[n].dis_now();
                    break;
                case '8':
                    D.user[n].set_code();
                    break;
                case '9':
                    D.user[n].dis_file();
                    break;
```

```
                case '0':
                    {
                        flag=0;
                        system("cls");
                    }
                    break;
                default:
                    cout<<"请输入0-9"<<endl;
                }
            }
            flag=1;
            break;
        case '3':
            flag=0;
            break;
        default:
            cout<<"请输入1-3! "<<endl;
            }
        }
}
```

任务3：驱动程序设计

1. demo.c文件

```
#include <linux/config.h>    /*头文件及变量等定义*/
#include <linux/module.h>
#include <linux/init.h>
#include <linux/kernel.h>       /* printk() */
#include <linux/fs.h>                /* everything... */
#include <linux/errno.h>         /* error codes */
#include <linux/types.h>         /* size_t */
#include <linux/proc_fs.h>
#include <linux/fcntl.h>            /* O_ACCMODE */
#include <linux/poll.h>             /* COPY_TO_USER */
#include <asm/system.h>       /* cli(), *_flags */


#define DEVICE_NAME   "demo"
#define demo_MAJOR        249
#define demo_MINOR        0
#define MAX_BUF_LEN       20
static char drv_buf[20];


/* ************************************************************************/

static int demo_open(struct inode *inode, struct file *file)      {
    MOD_INC_USE_COUNT;
    sprintf(drv_buf,"device open sucess!\n");
    printk("device open sucess!\n");
    return 0;
}
static int  demo_release(struct inode *inode, struct file *filp)  {
    MOD_DEC_USE_COUNT;
```

```
    printk("device release\n");
    return 0;
}

/*****************************************************************/

static ssize_t  demo_read(struct file *filp, char *buffer, size_t count,
loff_t *ppos)  {
    if(count > MAX_BUF_LEN)
        count=MAX_BUF_LEN;
    copy_to_user(buffer, drv_buf,count);
    printk("user read data from driver\n");
    return count;
}
static ssize_t  demo_write(struct file *filp,const char *buffer, size_t
count)  {
    copy_from_user(drv_buf , buffer, count);
    printk("user write data to driver\n");
    //your code here
    return count;
}


/*****************************************************************/

static int demo_ioctl(struct inode *inode, struct file *file,
                  unsigned int cmd, unsigned long arg)
{
    switch(cmd){
        case 1:{printk("ioctl runing  \n");
                        printk("runing command 1 \n");
                        break;}
        case 2:{printk("ioctl runing  \n");
                        printk("runing command 2 \n");
                        break;}
        default:
            printk("error cmd number\n");break;
    }
    return 0;
}



/*****************************************************************/

static struct file_operations demo_fops = {
    owner:  THIS_MODULE,
    write:  demo_write,
    read:   demo_read,
    ioctl:  demo_ioctl,
    open:   demo_open,
    release:    demo_release,
};

/*****************************************************************/
```

```
static int __init demo_init(void)   {
    int result;
    SET_MODULE_OWNER(&demo_fops);
    result = register_chrdev(demo_MAJOR, "demo", &demo_fops);
    if (result < 0) return result;
    printk(DEVICE_NAME " initialized\n");
    return 0;
}

static void __exit  demo_exit(void) {
    unregister_chrdev(demo_MAJOR, "demo");
    printk(DEVICE_NAME " unloaded\n");
}



/*****************************************************************/
module_init(demo_init);
module_exit(demo_exit);
```

2. test.c文件

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <fcntl.h>
main()
{
    int fd, num;
    char buf[10];
                                    //打开"/dev/demo"
    fd = open("/dev/demo", O_RDWR, S_IRUSR | S_IWUSR);
    if (fd != -1 )
    {
                                      //初次读demo
      read(fd, buf, 10);
      printf("The demo is %s\n", buf);
   //写demo
    printf("Please input the string written to demo\n");
    scanf("%s", &buf);
    write(fd, buf, 10);
    read(fd, buf, 10);
    printf("The demo is %s\n", buf);
    ioctl(fd, 1, 1);
    close(fd);
    }
    else
    {
    printf("Device open failure\n");
    }
}
```

四、实验总结

1. 关于银行家算法需要三大信息的总结:每个进程可以请求每个资源的最大值；每个进程目前拥有多少分配的资源；系统目前有多少资源可用。只有当请求的资源量小于或等于可用的资源量且还未达到可以请求地最大值时，才能将资源分配给进程；否则，该过程将一直等到资源可用。
2. 关于安全性检测的总结：如果所有进程都有可能完成执行，则状态被认为是安全的。 由于系统无法知道一个进程何时终止，或者到那时它将请求多少资源，所以系统假设所有进程最终将尝试获取它们所声明的最大资源，并在不久之后终止。此外，如果一个进程在没有获得最大资源的情况下终止，它只会使系统变得更简单。 如果在安全状态要处理就绪队列，安全状态被认为是决策者。在给定该假设的情况下，该算法通过尝试找到允许每个请求获取其最大资源然后终止的过程的假设请求集来确定状态是否安全。
3. 银行家算法其实就是要设法保证系统动态分配资源后不会进入不安全状态，以避免死锁。即当进程提出资源请求且系统的资源能够满足该请求时，系统将判断满足此次资源请求后系统状态是否安全，如果判断结果为安全，则给该进程分配资源，否则不分配资源，申请资源的进程将阻塞。
4. 本次实验完成了对资源分配的安全性算法的调试、设计了简单的文件系统、最后在Linux系统下完成了驱动程序的设计。模拟操作系统对IO设备管理，加深了解设备管理的工作原理，理解设备管理的原理及数据结构，理解“银行家算法”，编写和调试设备管理模拟程序，掌握设备管理算法，阅读“demo.c”，加深对设备管理的理解，但是最后在驱动程序的设计中，发生了一些尚未解决的错误，等到时间充裕还需完善。