合肥工業大學

系统软件综合设计报告
操作系统分册

设计题目	36 访问控制技术
学生姓名	杨俊杰
学　　号	2016217692
专业班级	物联网工程 16-1 班
指导教师	田卫东
完成日期	2019.7.1

目录

1 课程设计任务、要求、目的

1.1 课程设计任务

访问控制技术：建立基于访问控制表的系统安全管理系统。

1.2 课程设计目的和要求

系统应该包含两个部分，一个部分是按内核代码原则设计的基于访问控制表的系统安全管理系统，由一系列的函数组成；另一个部分是演示系统，调用基于访问控制表的系统安全管理的相应函数，以让其运行，同时提供系统的控制台展示界面，包括用户创建、用户管理、文件（对象）创建、文件（对象）管理、用户登录、模拟读、写和运行文件。

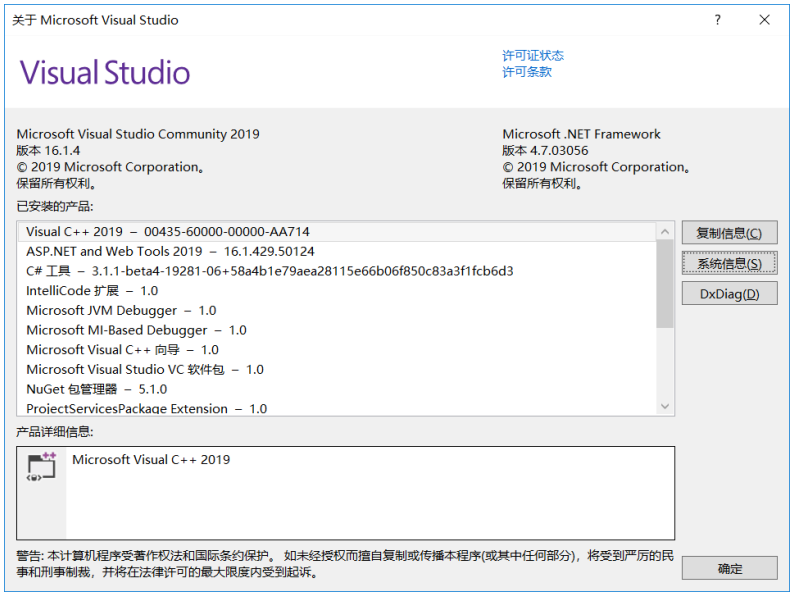
具体包括：

- 建立以用户组为管理域的访问控制表基本数据结构；
- 模拟操作系统中的若干对象，并可添加到访问控制表中；
- 建立操作系统的用户模型；
- 模拟用户登录及切换，访问对象时的受限情况。

2 开发环境

系统环境：Windows 10

IDE：Visual Studio 2019



编程语言：C/C++语言

3 相关原理

3.1 访问控制技术

访问控制技术是操作系统中一种重要的安全控制技术。

访问控制的目的是限制访问主体（用户、 进程、 服务等）对访问客体（文件、 系统等）的访问权限， 从而使计算机系统在合法范围内使用。

常见的访问控制模型有:

- 强制访问控制 (Mandatory access control: MAC)
- 自主访问控制 (Discretionary access control:DAC) （主流操作系统 Windows、Linux，防火墙 ACLs 等都是基于自主访问控制机制来实现访问控制）
- 基于角色的访问控制 (Role based access control : RBAC)
- 基于任务的访问控制模型(TBAC)
- 基于属性的访问控制模型
- 使用控制模型 UCON

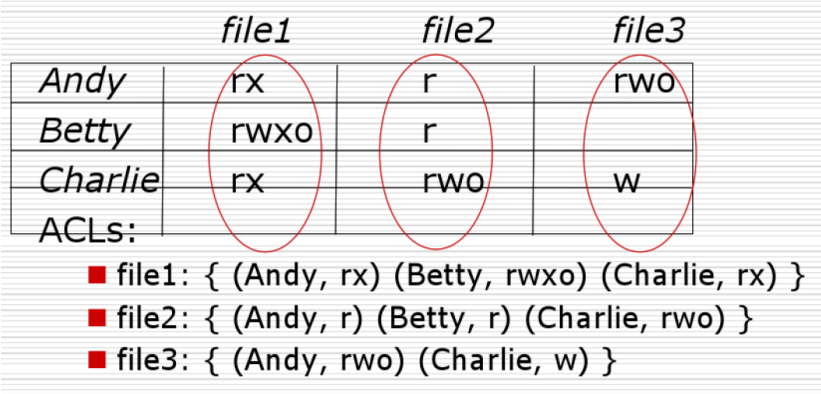
以上几种访问控制模型主要是 MAC 于 DAC。

强制访问控制：系统独立于用户行为强制执行访问控制， 用户不能改变他们的安全级别或对象的安全属性。 这种访问控制规则通常对数据和用户 按照安全等级划分标签， 访问控制机制通过比较安全标签来确定授予还是拒绝用户对资源的访问。 强制访问控制进行了很强的等级划分。

自主访问控制：自主访问控制机制允许对象的属主来制定针对该对象的保护策略。 通常 DAC 通过授权列表（或访问控制列表）来限定哪些主体针对哪些客体可以执行什么操作。如此将可以非常灵活地对策略进行调整，具有易用性与可扩展性。

3.2 访问控制表

本次课设所做的任务就是实现自主访问控制（DAC）中访问控制列表（Access Control List，ACL）的实现（DAC 还可基于保护位实现，在此不赘述）。



操作系统中的基于组的访问控制表是将用户分为多个组，针对每个组进行一些权限的限制。例如在 Linux 下,对一个文件(或者资源)可以进行操作的对象被分为三类: file owner(文件的拥有者),group(组,可以不是文件拥有者所在的组), other (其他)而对于每一类别又分别定

义了读、写、执行（read, write and execute ）权限以及特殊权限。但是这些权限只能搭配使用而已，如果想要对其其他用户组或者其他用户里的某一位用户设定一些不同的权限，就不行了。

Linu 上的 ACL 访问控制列表，主要的目的是在提供传统的 owner,group,others 的 read,write,execute 权限之外的局部权限设定。ACL 可以针对单个用户，单个文件或目录来进行 r,w,x 的权限设定，特别适用于需要特殊权限的使用情况。简单地来说，ACL 就是可以设置特定用户或用户组对于一个文件/目录的操作权限，可以实现灵活的权限管理，除了文件的所有者，所属组和其它人，可以对更多的用户设置权限。简单地来说 ACL 就是可以设置特定用户或者用户组对于一个文件/文件夹的操作权限。

4 系统结构和主设计思路

描述所设计系统的系统架构，主要的算法思想、流程图等。

本项目分为两个大模块。

第一个模块是库文件部分。

通过基于链表实现的用户表、数组实现的访问控制表（组表）和链表实现的文件链表，组成了整体模型。每个用户有一个属性为所属的组。基于组，对文件进行管理。每个用户只能访问自己组的文件对象。每个文件对象也可以进行 r/w/x 权限控制。

主要有函数：add_user()添加用户、int init()初始化用户表、add_ob()添加对象或修改对象、int user_read()用户读对象、user_write()用户写对象、user_execute()用户运行对象、 show_fun()显示用户表和组表（调试用）、user_log()用户登录

第二个模块是测试文件部分。

主要通过设置函数调用库文件内的函数，实现与用户进行创建用户、创建文件、登录用户、读/写/运行文件等的交互。主要包括管理员添加用户、管理员添加对象、用户读对象、用户写对象、用户执行对象、用户登录、用户选择。

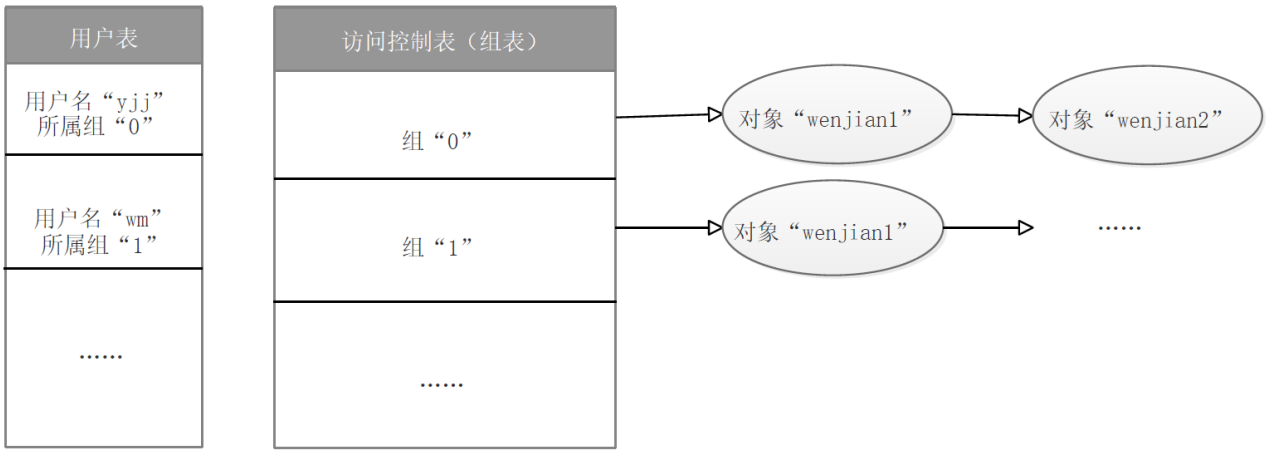


图 1 用户表和访问控制表模型示意图

5 程序实现---主要数据结构

描述所设计系统的关键数据结构。
（1）用户模型链表

```
typedef struct user_table/*用户模型链表*/
{
    char cName[str_size];/*用户名*/
    char secret[str_size];/*用户密码*/
    int group_num; /*所属用户组*/
    struct user_table *next;/*指向本结构体类型的指针类型*/
}user;
```

（2）对象模型

```
typedef struct object_model /*对象模型*/
{
    char oName[str_size]; /*对象名*/
    int permission; /*访问(r)、修改(w)、运行权限(x)*/
    struct object_model *next;
}myobj;
```

6 程序实现---主要程序清单

（1）库文件

① 定义用户表与组表

```
（2） user* user_head; /*用户表头*/
（3） myobj* group_table[5] = { NULL,NULL,NULL,NULL,NULL }; /*用户组表（访问控制表）
      */
```

②初始化用户表

```
/*
函数名：init
函数作用：初始化用户表
参数：无
返回：1（成功）
*/
int init()
```

```
{
    //extern user* user_head;
    user_head = (user*)malloc(sizeof(user));
    char name[str_size] = "yjj";
    char secret[str_size] = "123";
    memcpy(user_head->cName, name, sizeof(char) * str_size);
    user_head->group_num = 0;
    memcpy(user_head->secret, secret, sizeof(char) * str_size);
    user_head->next = NULL;
    return 1;
}
```

③添加用户

```
/*
函数名：add_user
函数作用：添加用户
参数：姓名、用户组、密码
返回：1(成功),0（失败）
*/
int add_user(char *name, int group, char *secret)
{
    //extern user* user_head;
    user *t = user_head, *in;
    if (t == NULL)
        return 0;
    while (t->next != NULL)
    {
        t = t->next;
    }
    in = (user*)malloc(sizeof(user));
    memcpy(in->cName, name, sizeof(char) * str_size);
    in->group_num = group;
    memcpy(in->secret, secret, sizeof(char) * str_size);
    in->next = NULL;
    t->next = in;
    return 1;

}
```

④添加对象或修改对象

```
/*
```

```
2.  函数名：add_ob
3.  函数作用：添加对象或修改对象
4.  参数：对象名、所属用户组名、权限(rwx)
5.  返回：1(第一个对象成功)2(add 成功)3(修改成功)
6.  */
7.  int add_ob(char* name, int group, int permission)
8.  {
9.      if (group_table[group] == NULL) // 该用户组的第一个对象
10.     {
11.         group_table[group] = (myobj*)malloc(sizeof(myobj));
12.         memcpy(group_table[group]->oName, name, sizeof(char) * str_size);
13.         group_table[group]->permission = permission;
14.         group_table[group]->next = NULL;
15.         return 1;
16.     }
17.     else
18.     {
19.         myobj* in = group_table[group];
20.         while (in != NULL && strcmp(in->oName, name)) // 是否该对象节点已经存
    在，存在则修改(strcmp 函数相等则返回 0)
21.             in = in->next;
22.         if(in != NULL ) // 对象已存在
23.         {
24.             memcpy(in->oName, name, sizeof(char) * str_size);
25.             in->permission = permission;
26.             return 3;
27.         }
28.         else // 对象要新建
29.         {
30.             in = group_table[group];
31.             while (in->next != NULL)    // 找到最后一个节点
32.                 in = in->next;
33.             myobj* t = (myobj*)malloc(sizeof(myobj));
34.             memcpy(t->oName, name, sizeof(char) * str_size);
35.             t->permission = permission;
36.             t->next = NULL;
37.             in->next = t;
38.             return 2;
39.         }
40.
41.     }
42.
43. }
```

⑤显示用户表和组表（调试用）

```
/*
函数名：show_fun
函数作用：显示用户表和组表（调试用）
参数：无
返回：无
*/
void show_fun(void)
{
    user* t = user_head;
    cout << "------------用户表------------" << endl;
    while (t != NULL)
    {
        cout << "用户名：" << t->cName << " 所在组：" << t->group_num << " 密码：" << t->secret << endl;
        t = t->next;
    }
    cout << "------------用户组表(访问控制表)------------" << endl;
    for (int i = 0; i < 5; i++)
    {
        if (group_table[i] != NULL)
        {
            myobj* j = group_table[i];
            cout << "组号："<<i;
            while (j != NULL)
            {
                cout << "---->对象名：" << j->oName << " 权限：" << j->permission;
                j = j->next;
            }
            cout << endl;
        }
    }
}
```

⑥用户登录

```
/*
函数名：user_log
函数作用：用户登录
参数：用户名、密码
返回：1(成功) 0（用户不存在）2(密码错误)
```

```
*/
int user_log(char* user_name, char* secret)
{
    user* p_user = user_head;
    while (p_user != NULL && strcmp(p_user->cName, user_name))
        p_user = p_user->next;
    if (p_user == NULL)
        return 0; // 用户不存在
    else
    {
        if (strcmp(p_user->secret, secret) == 0) // 密码符合
            return 1;
        else
            return 2; // 密码错误
    }
}
```

⑦用户读对象

```
/*
函数名：user_read
函数作用：用户读对象
参数：用户名、对象名
返回：1(成功) 0（用户不存在）2(无权限)3(对象不可见)
*/
int user_read(char* user_name, char* ob_name)
{
    user* p_user = user_head;
    while (p_user != NULL && strcmp(p_user->cName, user_name))
        p_user = p_user->next;
    if (p_user == NULL)
        return 0;  // 用户不存在
    else
    {
        myobj* t = group_table[p_user->group_num];
        while (t != NULL && strcmp(t->oName, ob_name))
            t = t->next;
        if (t == NULL)
            return 3;  //没有此对象
        else
        {
            if (t->permission & 0x04)
                return 1;   // 有读权限，第三位为 1
            else
```

```
                return 2;   // 无读权限
        }
    }
}
```

⑧用户写对象

```
/*
函数名：user_write
函数作用：用户写对象
参数：用户名、对象名
返回：1(成功) 0（用户不存在）2(无权限)3(对象不可见)
*/
int user_write(char* user_name, char* ob_name)
{
    user* p_user = user_head;
    while (p_user != NULL && strcmp(p_user->cName, user_name))
        p_user = p_user->next;
    if (p_user == NULL)
        return 0;  // 用户不存在
    else
    {
        myobj* t = group_table[p_user->group_num];
        while (t != NULL && strcmp(t->oName, ob_name))
            t = t->next;
        if (t == NULL)
            return 3;  //没有此对象
        else
        {
            if (t->permission & 0x02)
                return 1;   // 有写权限，第二位为 1
            else
                return 2;   // 无权限
        }
    }
}
```

⑨用户运行对象

```
/*
函数名：user_execute
函数作用：用户运行对象
参数：用户名、对象名
```

```
返回：1(成功) 0（用户不存在）2(无权限)3(对象不可见)
*/
int user_execute(char* user_name, char* ob_name)
{
    user* p_user = user_head;
    while (p_user != NULL && strcmp(p_user->cName, user_name))
        p_user = p_user->next;
    if (p_user == NULL)
        return 0;  // 用户不存在
    else
    {
        myobj* t = group_table[p_user->group_num];
        while (t != NULL && strcmp(t->oName, ob_name))
            t = t->next;
        if (t == NULL)
            return 3;  //没有此对象
        else
        {
            if (t->permission & 0x01)
                return 1;   // 有运行权限，第一位为 1
            else
                return 2;   // 无权限
        }
    }
}
```

（2）测试文件

①管理员添加用户

```
void adduser_interface()
{
    char username[str_size] = "0";
    int group = 0;
    char se[str_size] = "0";
    cout << "开始添加用户......" << endl;
    cout << "username:";
    cin >> username;
    cout << "group:";
    cin >> group;
    cout << "secret:";
    cin >> se;
    if(add_user(username, group, se))
```

```
        cout << "用户组: " << group << "下用户: " << username << "添加成功!" << endl;
    else
        cout << "用户组: " << group << "下用户: " << username << "添加失败!" << endl;

}
```

②管理员添加对象

```
void addob_interface()
{
    char obname[str_size] = "0";
    int group = 0;
    int permission = 0;
    cout << "开始添加对象......" << endl;
    cout << "obname: ";
    cin >> obname;
    cout << "group: ";
    cin >> group;
    cout << "permission: ";
    cin >> permission;
    switch (add_ob(obname, group, permission))
    {
    case 1: cout << "用户组: "<<group<<"新建第一个对象: "<< obname <<"成功!" << endl; break;
    case 2: cout << "用户组: " << group << "新对象: " << obname << "添加成功!" << endl; break;
    case 3: cout << "用户组: "<<group<<"下对象: "<<obname<<"权限修改成功!" << endl; break;
    default: cout << "未知错误! " << endl;
    }
}
```

③用户读对象

```
void userread_interface(char* username)
{
    //char username[str_size] = "0";
    char obname[str_size] = "0";
    cout << "开始读对象......" << endl;
    //cout << "username:";
    //cin >> username;
```

```
    cout << "obname: ";
    cin >> obname;
    switch (user_read(username, obname))
    {
        case 0: cout << "用户不存在！" <<endl; break;
        case 1: cout << "读成功！" << endl; break;
        case 2: cout << "无权限！" << endl; break;
        case 3: cout << "对象不可见！" << endl; break;
        default: cout << "未知错误！"<<endl;
    }

}
```

④用户写对象

```
void userwrite_interface(char* username)
{
    char obname[str_size] = "0";
    cout << "开始写对象......" << endl;
    //cout << "username:";
    //cin >> username;
    cout << "obname: ";
    cin >> obname;
    switch (user_write(username, obname))
    {
    case 0: cout << "用户不存在！" << endl; break;
    case 1: cout << "写成功！" << endl; break;
    case 2: cout << "无权限！" << endl; break;
    case 3: cout << "对象不可见！" << endl; break;
    default: cout << "未知错误！" << endl;
    }
}
```

⑤用户执行对象

```
void userexecute_interface(char* username)
{
    char obname[str_size] = "0";
    cout << "开始运行对象......" << endl;
    //cout << "username:";
    //cin >> username;
    cout << "obname: ";
    cin >> obname;
```

```
    switch (user_write(username, obname))
    {
    case 0: cout << "用户不存在！" << endl; break;
    case 1: cout << "运行成功！" << endl; break;
    case 2: cout << "无权限！" << endl; break;
    case 3: cout << "对象不可见！" << endl; break;
    default: cout << "未知错误！" << endl;
    }
}
```

⑥用户登录

```
void userlog_interface(void)
{
    char username[str_size] = "0";
    char secret[str_size] = "0";
    cout << "开始用户登录......" << endl;
    cout << "username: ";
    cin >> username;
    cout << "secret: ";
    cin >> secret;
    switch (user_log(username, secret))
    {
    case 1: cout << "用户: "<<username<<"登录成功!" << endl;
            user_select(username);  // 传入用户名
            break;
    case 0: cout << "用户不存在！" << endl; break;
    case 2: cout << "密码错误！" << endl; break;
    default: cout << "未知错误！" << endl;
    }
}
```

⑦用户选择

```
void user_select(char* username)
{
    int select2;
    while (true)
    {
        cout << "请选择操作（1 读对象 2 写对象 3 运行对象 6 用户注销）: ";
        cin >> select2;
        switch (select2)
        {
```

```
        case 1: userread_interface(username); break;
        case 2: userwrite_interface(username); break;
        case 3: userexecute_interface(username); break;
        case 6: return;
        default: cout << "未知错误！" << endl;
                 return;
        }
    }
}
```

⑧主函数

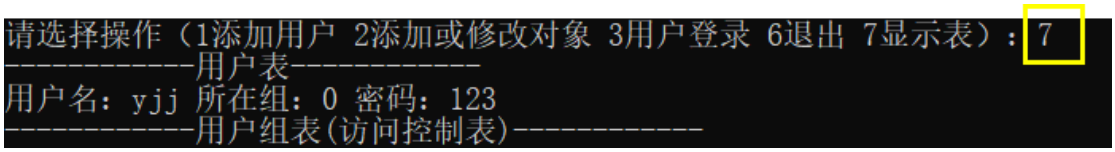
```
int main()
{
    init();
    int select;

    while (1)
    {
        cout << "请选择操作（1 添加用户 2 添加或修改对象 3 用户登录 6 退出 7 显示表）：";
        cin >> select;
        switch (select)
        {
        case 1: adduser_interface(); break;
        case 2: addob_interface(); break;
        case 3: userlog_interface(); break;
        case 6: return 0;
        case 7: show_fun(); break;
        default: cout << "未定义" << endl;
                return 0;
        }
    }
    return 0;
}
```

7 程序运行的主要界面和结果截图

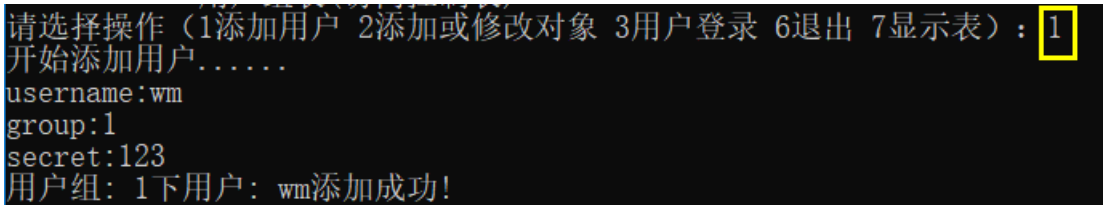
列出系统运行的主要界面和运行结果截图。

① 显示用户表和用户组表：



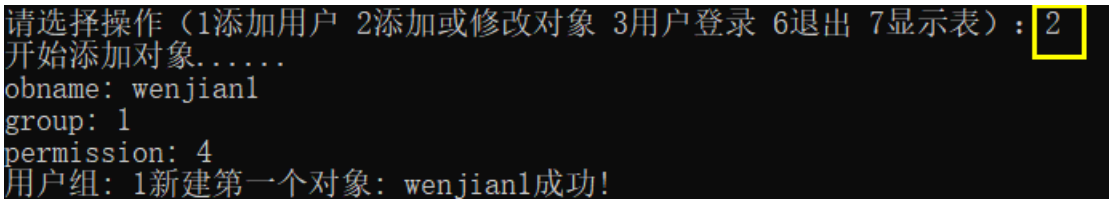
（系统默认存在一个用户“yjj”）

② 添加用户

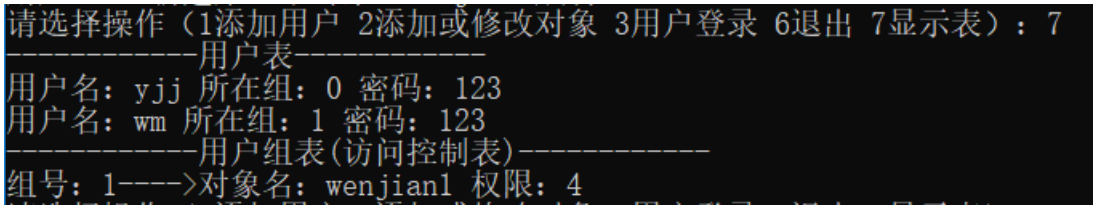


用户名为“wm”,所在组是组 1，密码是“123”

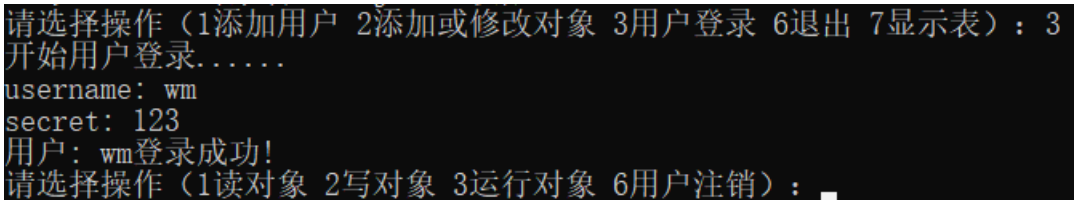
③ 添加文件（对象）



此时再次使用显示表功能（功能 7）可以看到刚刚添加的用户和文件已经添加成功。权限 4 代表 r/w/x 的权限设置分别为 1/0/0

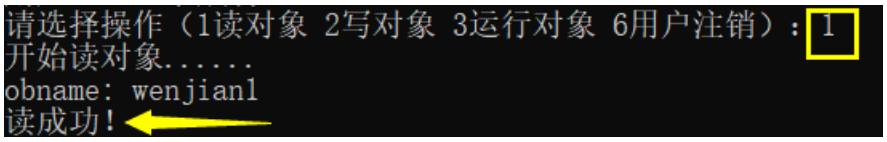


④ 登录用户

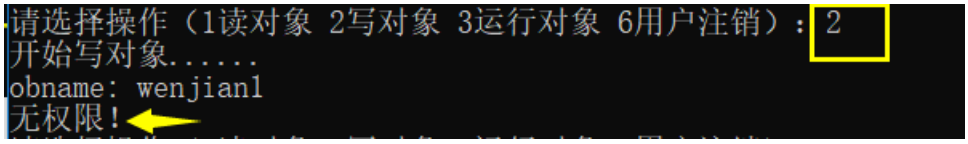


⑤ 模拟读文件成功的状况

使用登录的属于组 1 的用户“wm”尝试读组 1 的文件对象“wenjian1”

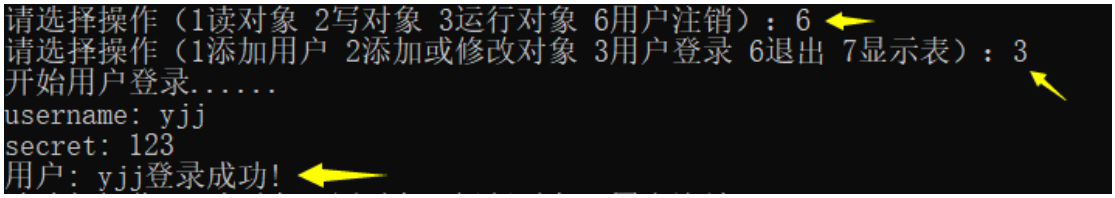


⑥ 模拟写文件权限受阻的状况

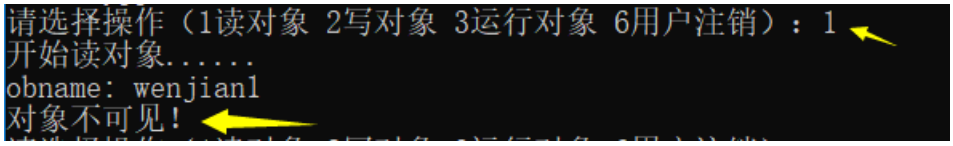


⑦ 模拟访问其他组的文件权限受阻的状况

注销原有用户，登录另外一个用户“yjj”



尝试读“wenjian1”。由于访问控制表（组表）的限制，属于组 0 用户“yjj”无法看见属于组 1 的“wenjian1”。



8 总结和感想体会

本次课程设计使我进一步了解了操作系统中的访问控制技术尤其是 linux 下的访问控制技术。

系统软件综合设计-操作系统部分的课程设计，让我在短时间内通过软件工程的思想完全自主设计一个操作系统的访问控制模型，使我的工程实践能力得到了提升，课设的顺利验收也给了我对未来的自信和对这条道路的肯定。由于要求不能使用 STL，必须自己设计数据结构，使得我重新复习了数据结构的相关知识，对它们有了新的掌握。

通过这次课程设计，我学会了：

- Windows 下库文件 lib 项目的创建和与控制台项目的关联；
- 通过自主设计数据结构，加深了对数组和链表的理解和应用；
- 了解了主流操作系统（Windows、Linux）中的访问控制技术及其实现方法，特别是本次设计中所完成的基于组的访问控制表的技术；
- 通过查询资料，也了解了许多老师上课没有具体讲解的具体原理知识。对实际 OS 有了更深一步的了解。

9 参考文献

[1]秦超,段云所,陈钟. 访问控制原理与实现[J].网络安全技术与应用,2001(05):54-58.
[2]王加森. 基于 LINUX 的安全操作系统[D].西南交通大学,2002.
[3]罗鑫. 访问控制技术与模型研究[D].北京邮电大学,2009.
[4]王利. 基于 Linux 的访问控制增强技术研究与实现[D].解放军信息工程大学,2008.