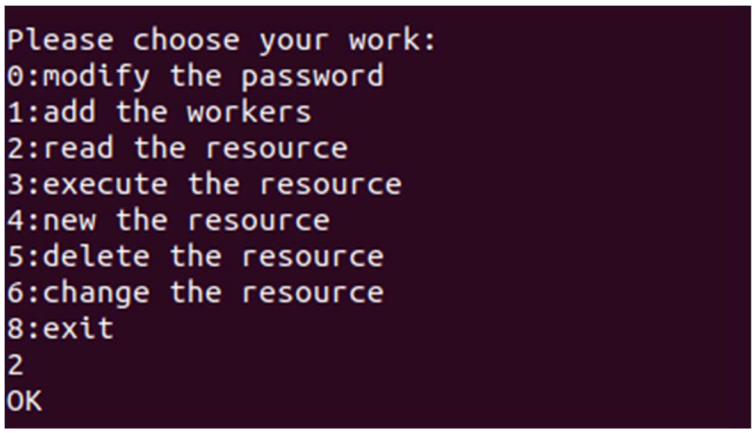
访问控制实验报告

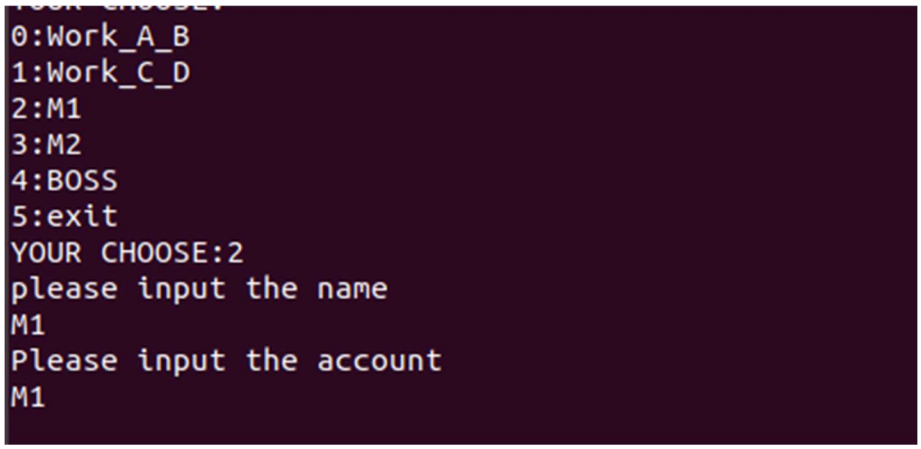
1 实验设计

为了更好地使用不同的情况，我设计了一个对本实验要求友好的 C 语言的隐式列表的 RBAC 以及一个扩展性极好的、所有列表都可以动态加入的显式列表的 C++语言的 RBAC。（这里所说的列表是指：用户集合表、角色集合表、权限集合表、角色与权限的分配表、用户与角色的分配表）

在具体在本实验中，我的系统首先使用者是 BOSS，他拥有对整个系统的绝对控制权，下表中所有权限他都拥有：



以及一些其他的权限，BOSS 可以添加任何员工，包括 M1、M2、A、C 类型的员工



而 M1 可以添加自己的下属，Work_A_B，不可添加其他人，M2 同样只可以添加自己的下属，而 ABCD 员工没法添加任何人，被添加的人只拥有添加者下属的权限，无法拥有其他权限。

```
YOUR CHOOSE:
0:Work_A_B
1:Work_C_D
2:M1
3:M2
4:BOSS
5:exit
YOUR CHOOSE:3
You are not permitted
```

在这里为了更好地适应本实验的情景，我将角色权限配置全部用代码完成，有较差的动态扩展性。
（注：此实验 power 表示权限，role 表示角色，object 表示用户主体）

```
struct ROLE W1 = {
        .r_name = "Work_A_B",
        .r_id = 0,
        .r_power = {
                ._read = 1,
                ._exe = 1,
                ._change = 0,
                ._del = 0,
                ._new = 0,
                ._other_1 = 0,
                ._other_2 = 0,
                ._other_3 = 0
        }
};

struct ROLE W2 = {
        .r_name = "work_C_D",
        .r_id = 1,
        .r_power = {
                ._read = 1,
                ._exe = 0,
                ._change = 1,
                ._del = 0,
                ._new = 1,
                ._other_1 = 0,
                ._other_2 = 0,
                ._other_3 = 0
        }
};
```

为了有较好的可扩展性，本实验我用 C++语言，用显式的列表写了可动态增加的 RBAC，可适用各种场合，RBAC 的数据结构在 rbac.h 头文件中，主函数的测试实现在 main.cpp 文件中。
首先，将实现相关的五个表以及相关的函数都可以动态分配加载：

```
 5
 6    //权限原子
 7   ⊞struct power { ... };
12
13    //角色原子
14   ⊞struct role { ... };
19
20    //用户原子
21   ⊞struct object { ... };
27
28    //权限集合
29   ⊞class power_table { ... };
77
78    //角色集合
79   ⊞class role_table { ... };
125
126  ⊞role_table::role_table() { ... }
129
130   //用户集合
131  ⊞class object_table { ... };
196
197  ⊞object_table::object_table() { ... }
200
201   //角色权限授权原子
202  ⊞struct role_power { ... };
207
208   //用户角色授权原子
209  ⊞struct object_role { ... };
214
215   //角色权限授权表
216  ⊞class role_power_table { ... };
253
254   //用户角色授权表
255  ⊞class object_role_table { ... };
292
```

```
// U R P 分别为用户集合，角色集合，权限集合
power_table P;
role_table R;
object_table U;

//PAR UAR 分别表示许可权与角色之间多对多的指派关系，用户与角色之间多对多的指派关系。
role_power_table PAR;
object_role_table UAR;
```

完全按照层次格式编写代码，所有的权限、角色、用户以及权限的配置都是动态可控的，这里也采用了第一个进入系统的人为管理员，相当于拥有 session 的功能。

```
char command[20];
cout << "Welcome here" << endl;
cout << "You are a manager of this system!" << endl;
while (1) {
    cout << "Please choose your operation:" << endl;
    cout << "1.add power\n2.add role\n3.add user\n4.add role_power\n5.add user_role\0.exit" << endl;
    cin >> command;
    if (command[0] == '0')
        break;
    switch (command[0]) {
```

其他设计与第一个 C 语言的设计相同。

注：c 语言版本:main.c

C+语言:main.cpp rbac.h