中国科学院大学
University of Chinese Academy of Sciences

RBAC访问控制实验

Role-Based Access Control LSM

郝淼

2024.4.19

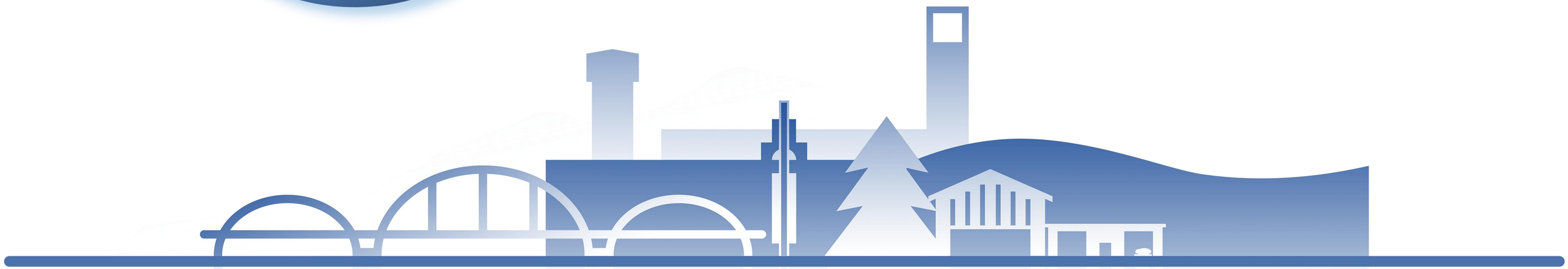
目录

CONTENTS

中国科学院大学
University of Chinese Academy of Sciences

01

RBAC策略



本实验中，用户（User）与角色（Role）是一对一关系，角色与权限（Permission）是多对多关系。因此，用户、角色与权限的关系可以表示为：

$$用户 \rightarrow 角色 \Rightarrow 权限$$

进一步可以将权限分解：

$$权限 = 接受性 + 操作 + 客体$$

可以用多重元组$c \in (U \times R \times A \times Op \times O)$表示“一个用户具有某个权限”

用户（User）

角色（Role）

接受性（Acceptability）

操作（Operation）

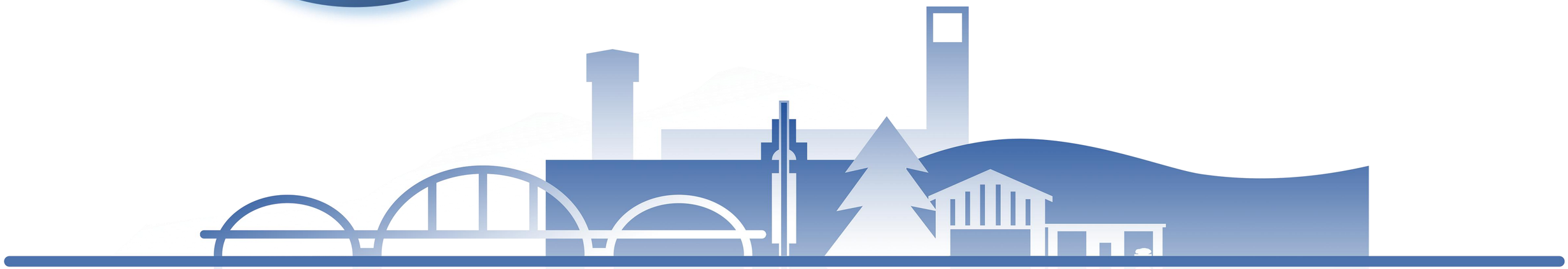
客体（Object）

数据库中存储若干这样的五元组，当U_i通过Op_j访问O_k时，检查数据库中否存在$(U_i, R_l, A_m, Op_j, O_k)$:

- 若存在且A_m为允许，则允许访问
- 若存在且A_m为拒绝，则拒绝访问
- 特别地，如果在数据库中没有查询到对应的元组，则允许访问

中国科学院大学
University of Chinese Academy of Sciences

02 实现



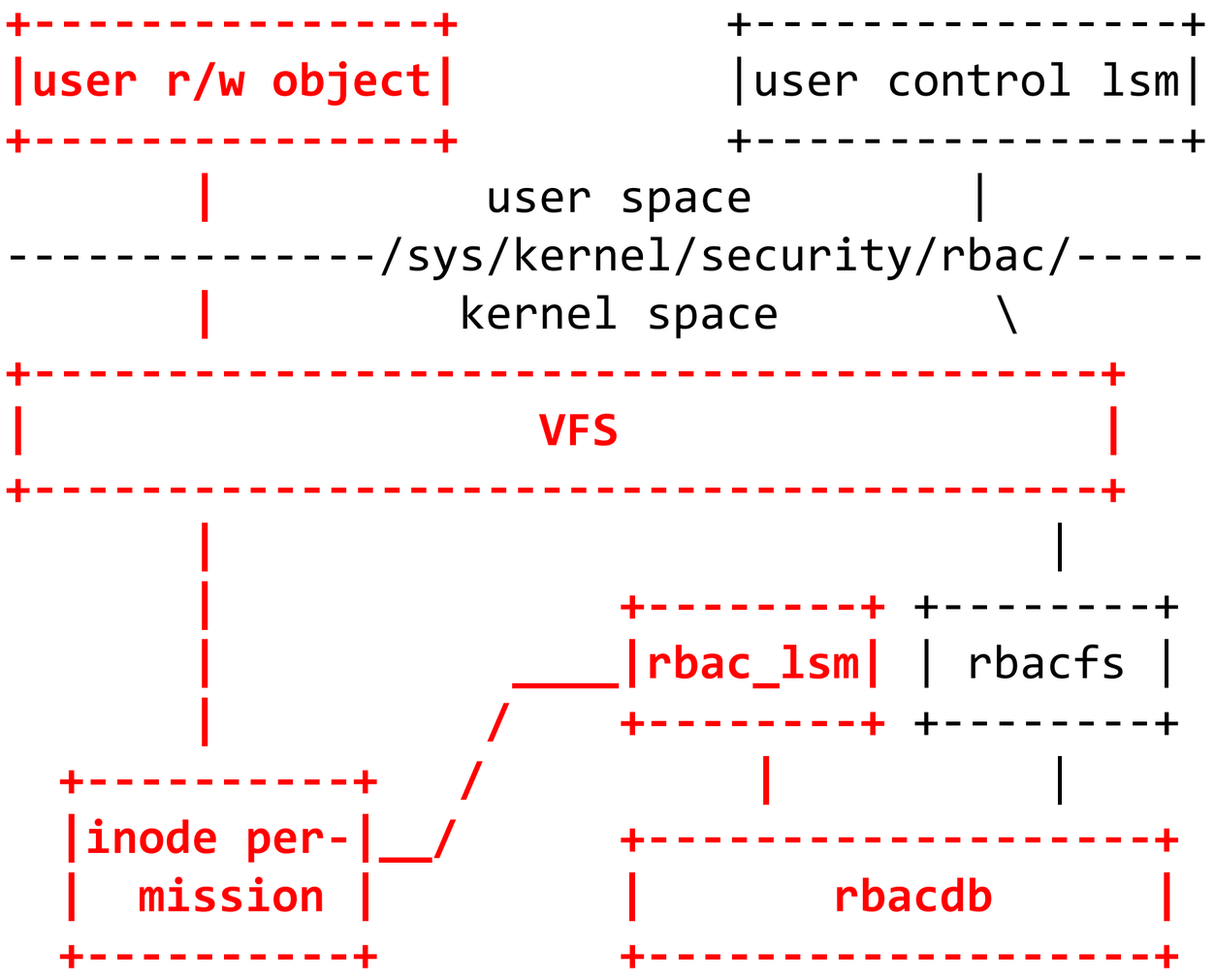
实验平台

实验平台：LoongArch 32 Reduced QEMU

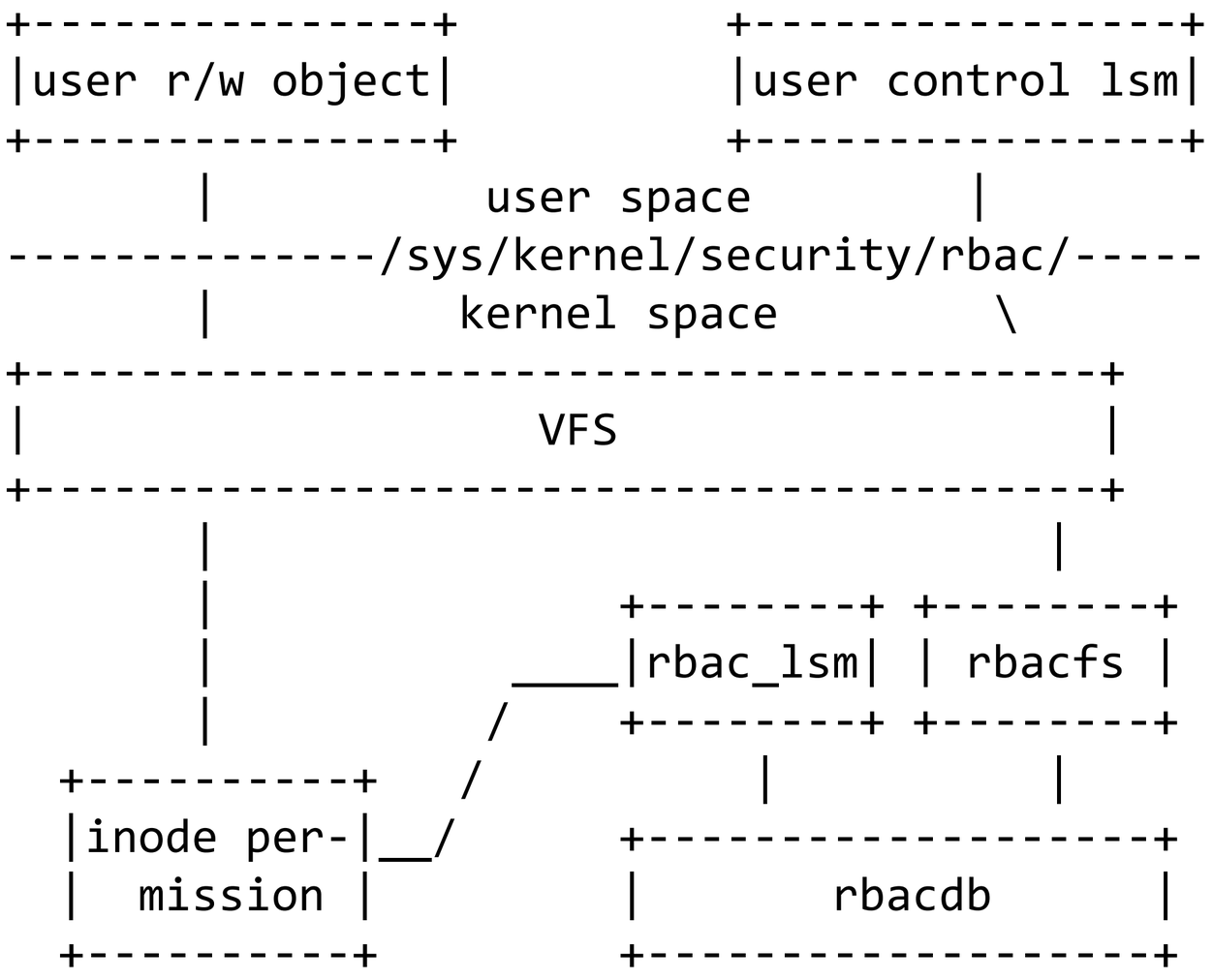
操作系统：BusyBox

内核版本：Linux 5.14.0-rc2

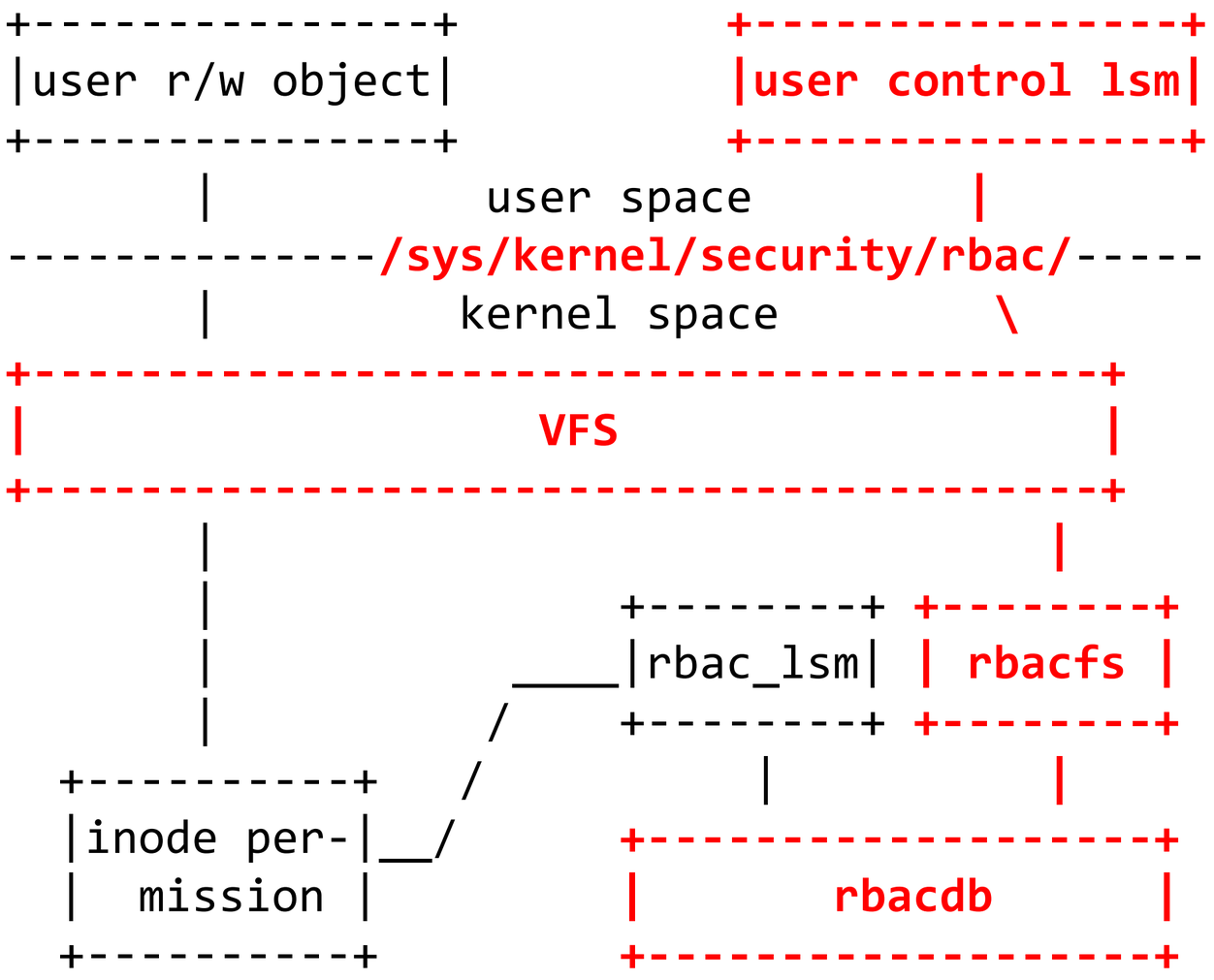
总体架构-访问控制


user r/w object
user control lsm
user space
/sys/kernel/security/rbac/
kernel space
VFS
rbac_lsm
rbacfs
inode per-
mission
rbacdb

总体架构


user r/w object
user control lsm
user space
/sys/kernel/security/rbac/
kernel space
VFS
rbac_lsm
rbacfs
inode per-
mission
rbacdb

总体架构- lsm 管理


user r/w object
user control lsm
user space
/sys/kernel/security/rbac/
kernel space
VFS
rbac_lsm
rbacfs
inode per-
mission
rbacdb

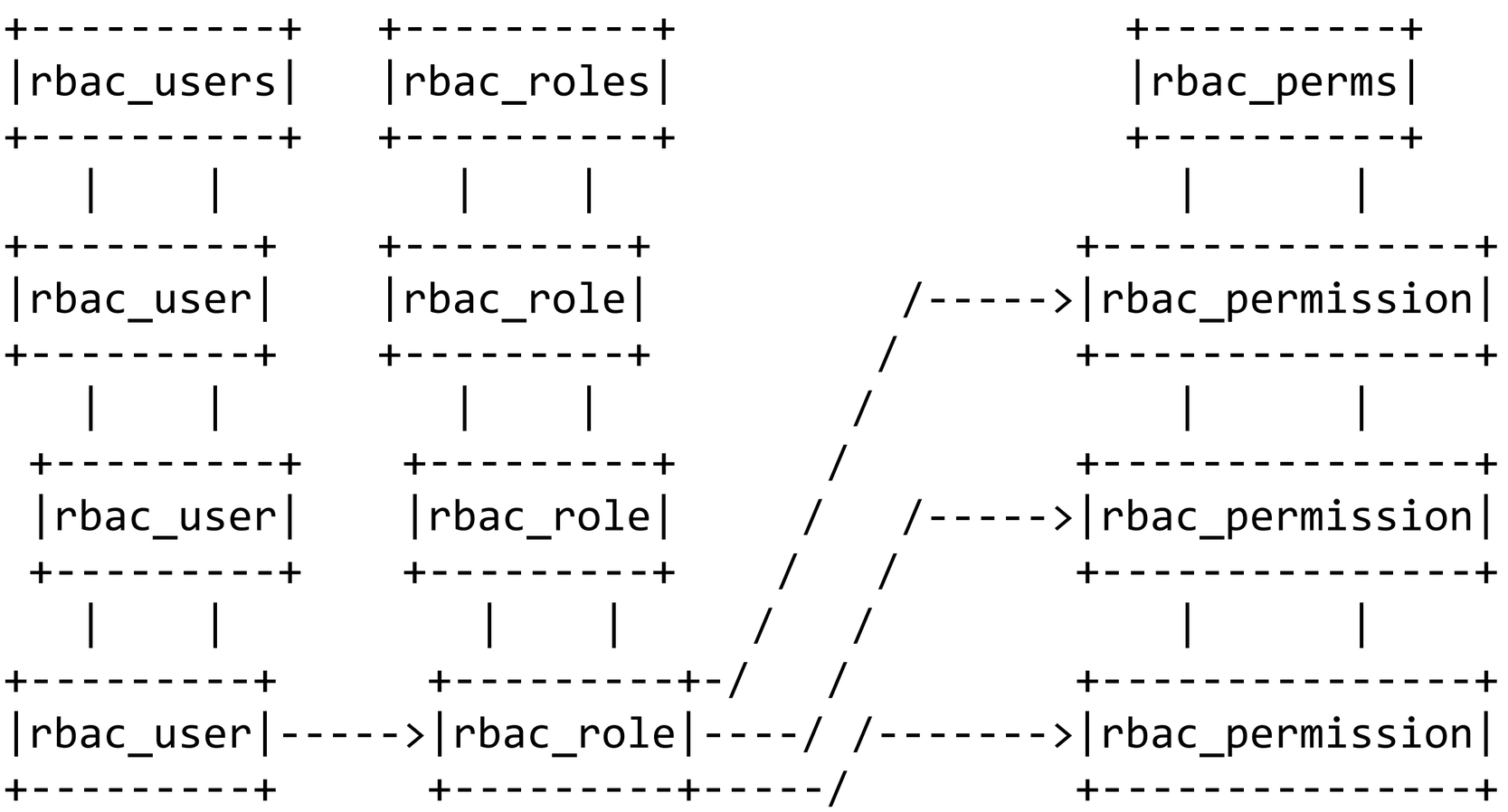
db 层负责实现内核数据对象的管理，向上提供接口供 fs 层和 lsm 进行使用。rbac-lsm 实现的内核数据对象包括：

```
struct rbac_user {
    uid_t               uid;
    struct rbac_role *role;
    struct list_head entry;
};
```

```
struct rbac_role {
    char                        name[ROLE_NAME_LEN];
    struct rbac_permission *perms[ROLE_MAX_PERMS];
    refcount_t                  ref;
    struct list_head            entry;
};
```

```
struct rbac_permission {
    int                 id;
    rbac_acc_t          acc;
    rbac_op_t           op;
    rbac_obj_t          obj;
    char                *obj_path;
    refcount_t          ref;
    struct list_head    entry;
};
```

db 层实现


rbac_users
rbac_roles
rbac_perms
rbac_user
rbac_role
rbac_permission
rbac_user
rbac_role
rbac_permission
rbac_user
rbac_role
rbac_permission

fs 层基于 securityfs 机制，在 /sys/kernel/security/rbac 下建立了一些列虚拟文件，实现了用户与 lsm 的交互功能：

文件名	属性	说明
enable	读写	获取/改变 rbac-lsm 使能
user	只读	获取已添加的用户信息
role	只读	获取已添加的角色信息
perm	只读	获取已添加的权限信息
ctrl	只写	添加、删除内核数据对象，改变其间的关系

fs 层实现- enable

可以通过向 enable 文件中写 0 或 1 改变 rbac-lsm 的使能状态，其中 0 表示关闭；1 表示开启。

```
# cat enable
rbac: enabled
# echo 0 > enable
# cat enable
rbac: disabled
# echo 1 > enable
# cat enable
rbac: enabled
#
```

fs 层实现- rbac_enable_read


```
extern int rbac_enable;

static ssize_t rbac_enable_read(struct file *file, char __user *buf,
                                size_t size, loff_t *ppos)
{
        ...
        sprintf(kbuf, "rbac: %s\n", rbac_enable ? "enabled" : "disabled");
        ...
}
```

fs 层实现- rbac_enable_write


```
static ssize_t rbac_enable_write(struct file *file, const char __user *buf,
                                 size_t size, loff_t * ppos)
{
        ...
        switch (kbuf[0]) {
        case '0':
                rbac_enable = 0;
                break;
        case '1':
                rbac_enable = 1;
                break;
        default:
                ret = -EINVAL;
        }
        ...
}
```

fs 层实现- rbac_fs_init


```
static int __init rbac_fs_init(void)
{
        ...
        for (i = RBAC_ENABLE; i < RBAC_FP_TYPE_NUM; i++) {
                rbac_files[i].fp =
                        securityfs_create_file(rbac_files[i].namep,
                                               rbac_files[i].mode, rbac_dir,
                                               NULL, &rbac_files[i].ops);
                if (IS_ERR(rbac_files[i].fp)) {
                        ret = PTR_ERR(rbac_files[i].fp);
                        goto out;
                }
        }
        ...
}

fs_initcall(rbac_fs_init)
```

fs 层实现- user

读 user 以获取目前 rbac-lsm 已知的用户信息。

```
# cat user
uid: 0 acts as role "admin"
uid: 100
#
```

目前 rbac-lsm 已经添加了 2 个用户，其中 uid 为 0 的用户已经绑定到了名为 admin 的角色上； uid 为 100 的用户未绑定到任何角色。

fs 层实现- role

读 role 以获取目前 rbac-lsm 中已添加的角色信息。

```
# cat role
admin
        perm[0] id: 0
guest
#
```

目前 rbac-lsm 已经添加了 2 个角色，其中名为 admin 的角色已经绑定了一个 id 为 0 的权限；名为 guest 的角色未绑定任何权限。

读 perm 以获取目前 rbac-lsm 中已添加的权限。

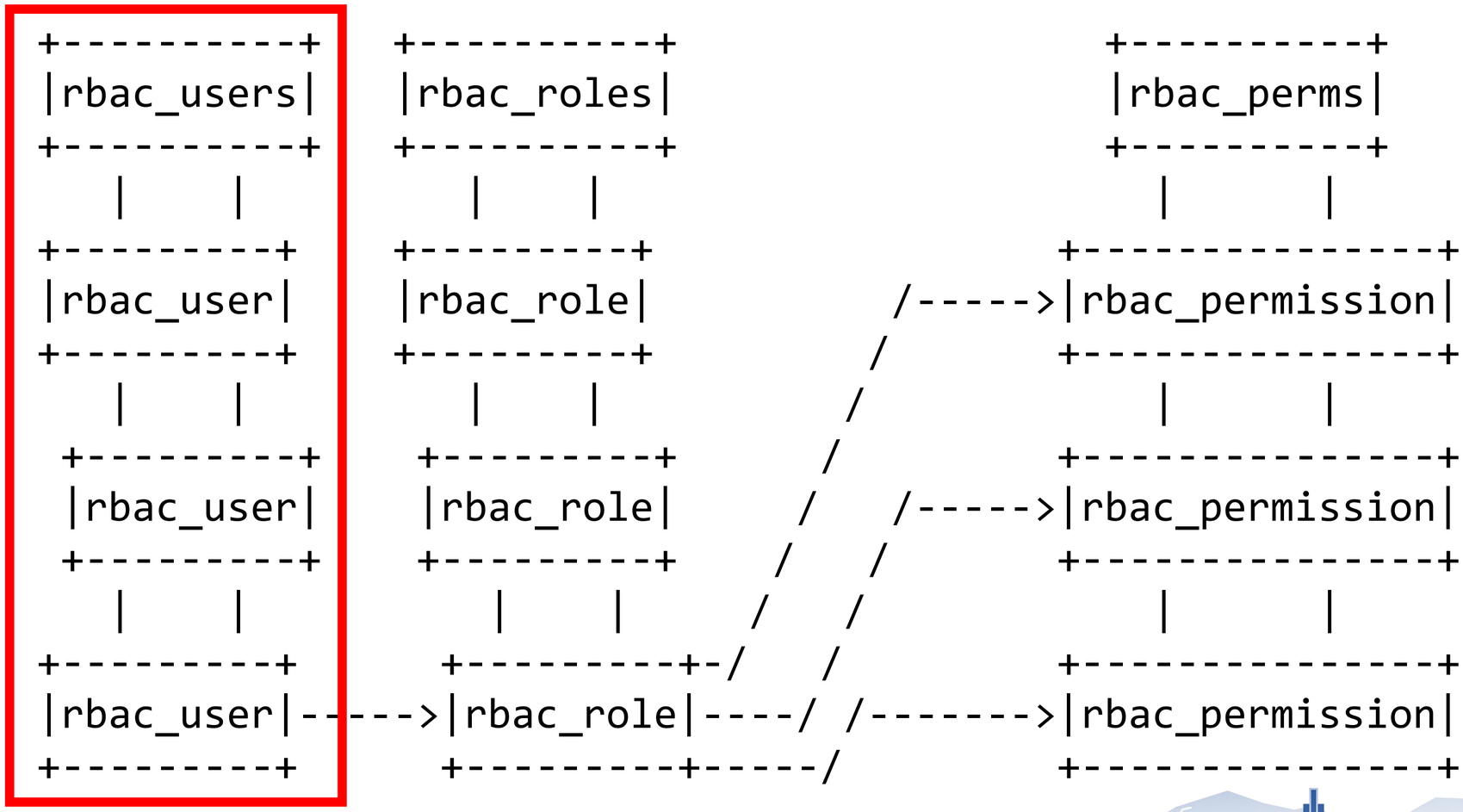
```
# cat perm
[0]: deny write on /init
[1]: accept read on /
[2]: deny read on /init
[3]: deny write on /
# 
```

目前 rbac-lsm 已经添加了 4 条权限，其中 id 为 0 的权限表示不允许写 /init；id 为 1 的权限表示允许读 /；id 为 2 的权限表示不允许读 /init；id 为 3 的权限表示不允许写 /。

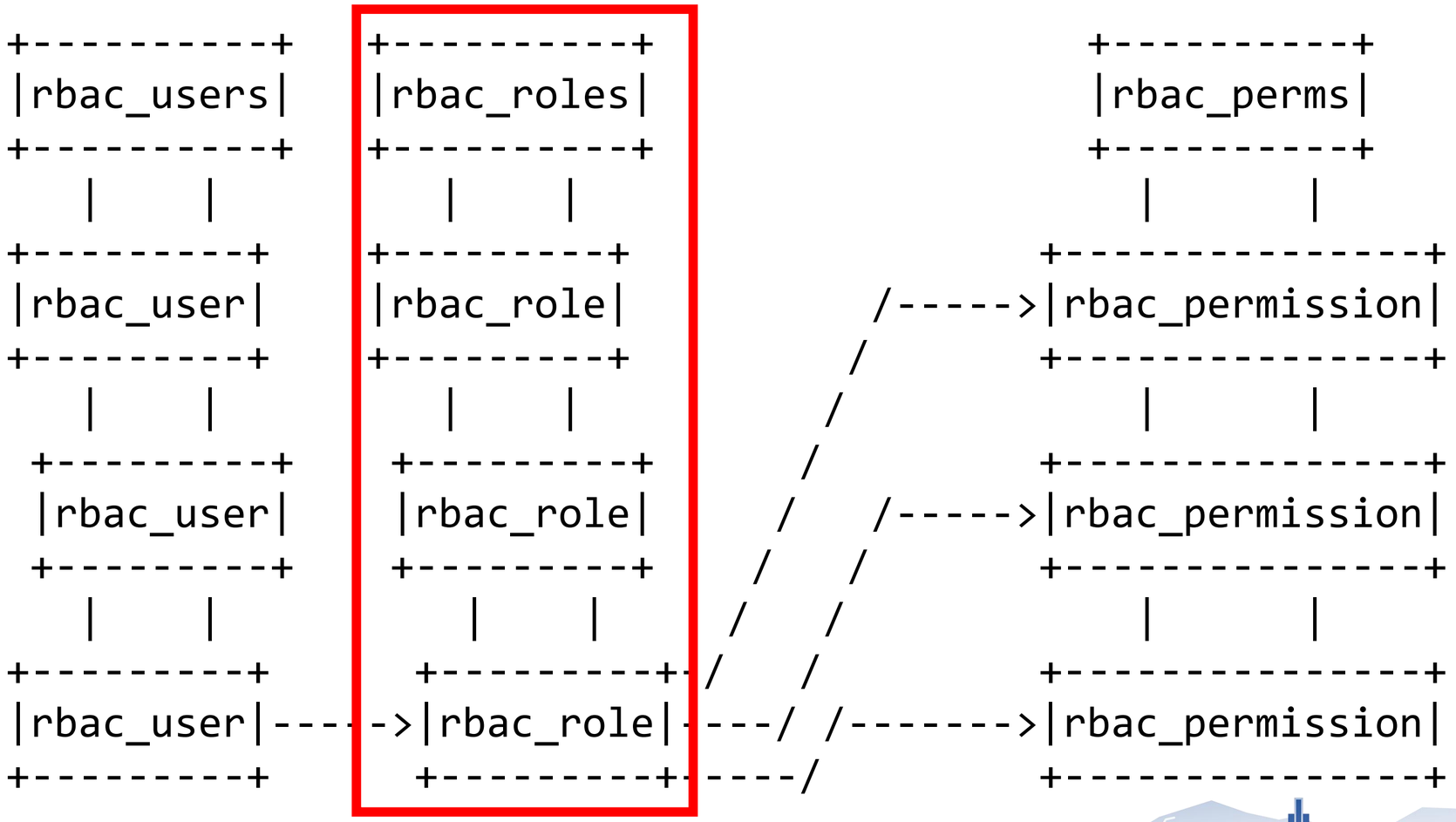
向 ctrl 写命令以实现对内核数据对象的修改，支持的命令包括：

- [add|remove] user UID：[添加|删除]用户，其 uid 为 UID
- [add|remove] role NAME：[添加|删除]名字为 NAME 的角色
- [add|remove] perm ACC OP OBJ：[添加|删除]接受性为 ACC，操作为 OP，客体为 OBJ 的权限
- [un]register UID NAME：将 uid 为 UID 的 user 和名字为 NAME 的 role（解）绑定
- bind ID NAME：将 id 为 ID 的权限绑定到名字为 NAME 的 role 上
- unbind RID NAME：将 id 为 ID 的权限到名字为 NAME 的 role 的绑定解除

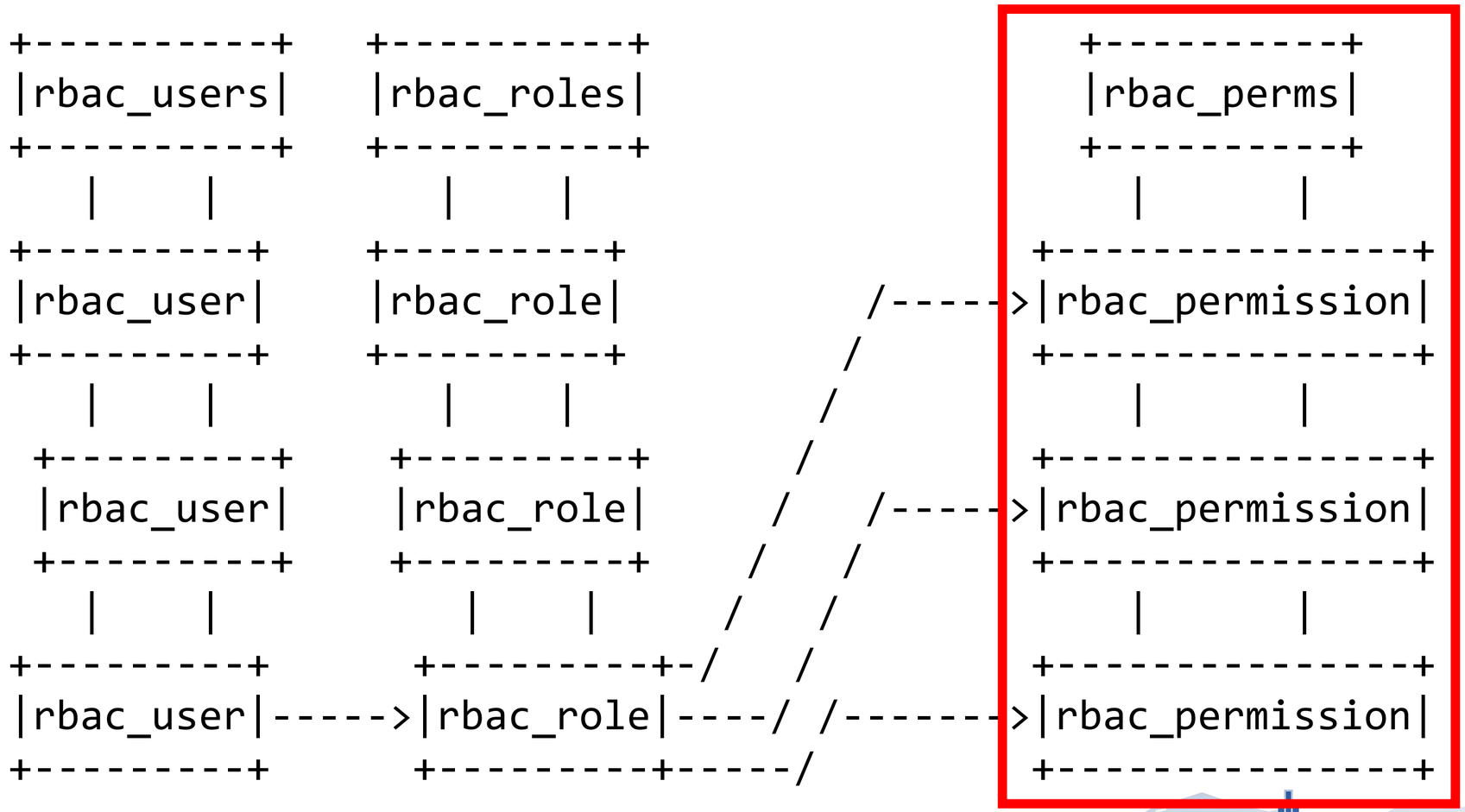
[add|remove] user UID



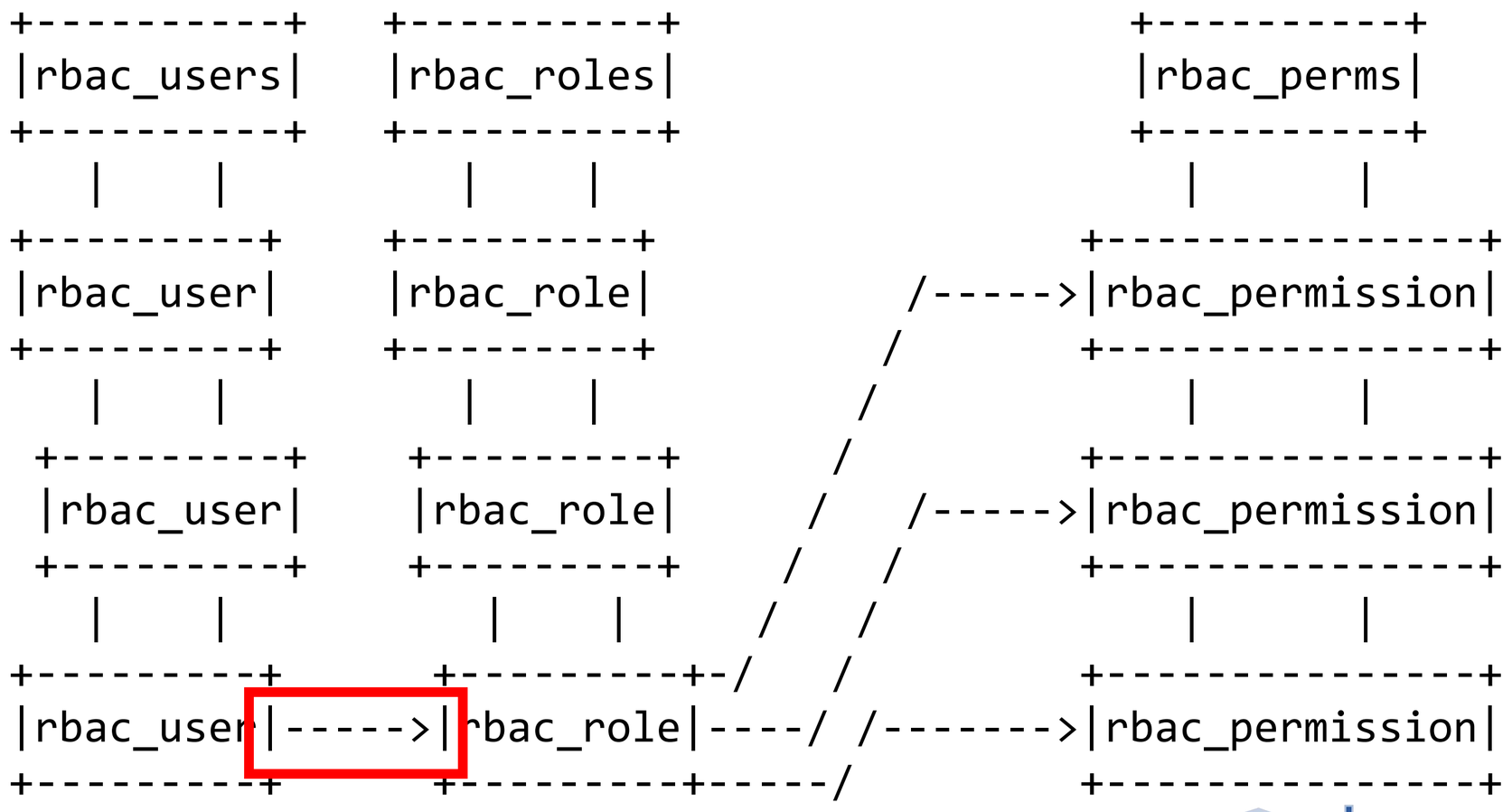
[add|remove] role NAME



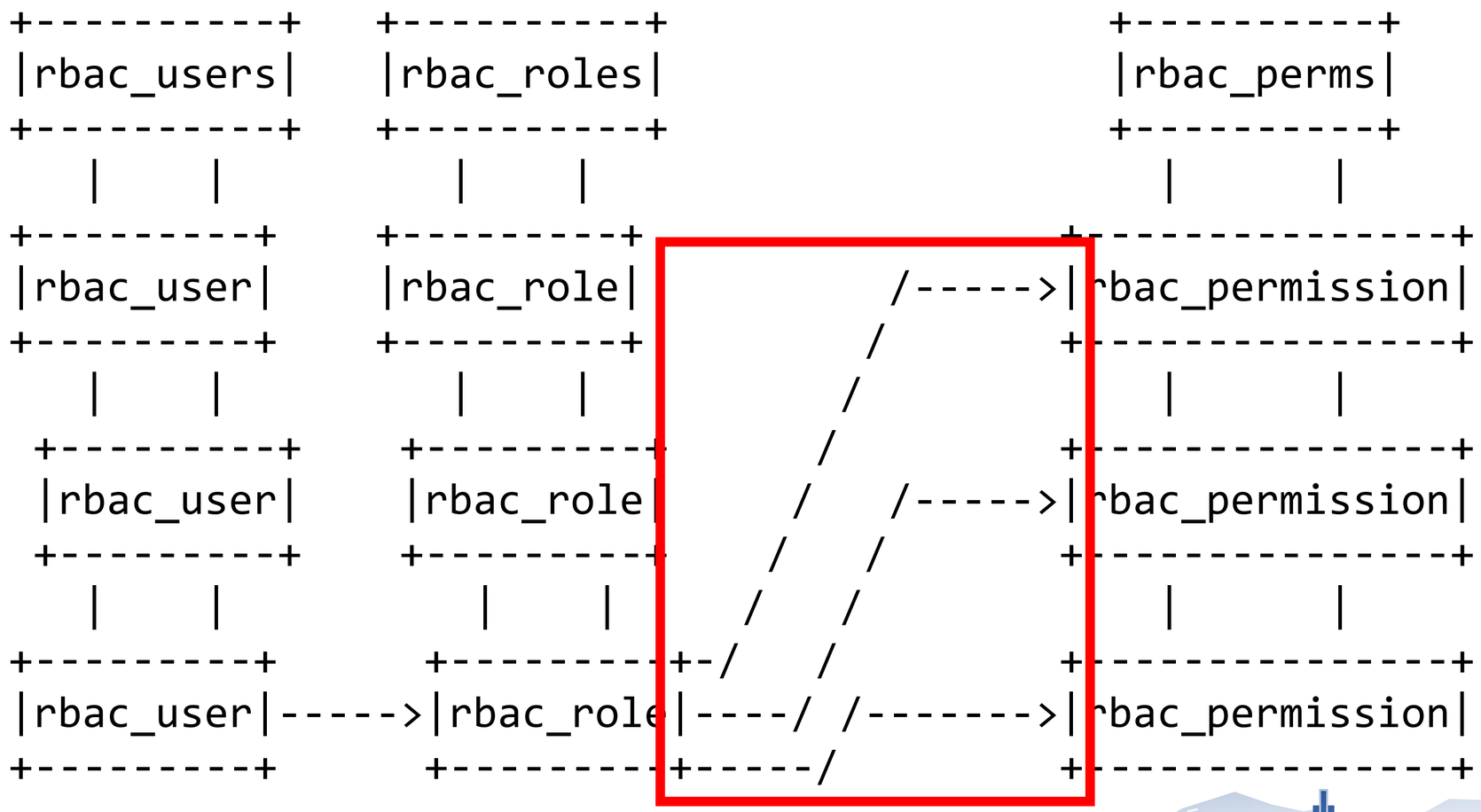
[add|remove] perm ACC OP OBJ



[un]register UID NAME



bind ID NAME; unbind RID NAME



lsm 通过将权限检查钩子挂在 inode_permission 上以接受/拒绝某次访问：

```
static int rbac_inode_permission(struct inode *inode, int mask)
{
        ...
        if (rbac_enable == 0)
                goto out;
        cred = current_cred();
        uid = from_kuid(cred->user_ns, cred->euid);
        ret = rbac_check_access(uid, inode, mask);
out:
        return ret;
}

static struct security_hook_list rbac_hooks[] = {
        LSM_HOOK_INIT(inode_permission, rbac_inode_permission),
};
```

lsm 通过将权限检查钩子挂在 inode_permission 上以接受/拒绝某次访问：

```
static int rbac_inode_permission(struct inode *inode, int mask)
{
        ...
        if (rbac_enable == 0)
                goto out;
        cred = current_cred();
        uid = from_kuid(cred->user_ns, cred->euid);
        ret = rbac_check_access(uid, inode, mask);
out:
        return ret;
}

static struct security_hook_list rbac_hooks[] = {
        LSM_HOOK_INIT(inode_permission, rbac_inode_permission),
};
```

rbac 开关检查，如果 rbac 关闭，则直接允许访问

lsm 实现- rbac_inode_permission

lsm 通过将权限检查钩子挂在 inode_permission 上以接受/拒绝某次访问：

```
static int rbac_inode_permission(struct inode *inode, int mask)
{
        ...
        if (rbac_enable == 0)
                goto out;
        cred = current_cred();
        uid = from_kuid(cred->user_ns, cred->euid);
        ret = rbac_check_access(uid, inode, mask);
out:
        return ret;
}

static struct security_hook_list rbac_hooks[] = {
        LSM_HOOK_INIT(inode_permission, rbac_inode_permission),
};
```

rbac 权限检查，根据 uid（用户）、mask（操作）和 inode（客体）查询是否具有访问权限

lsm 实现- rbac_check_access


```
int rbac_check_access(uid_t uid, struct inode *inode, int mask)
{
        ...
        user = rbac_get_user_by_uid(uid);
        role = user->role;
        for (i = 0; i < ROLE_MAX_PERMS; i++) {
                perm = role->perms[i];
                if (perm != NULL && perm->obj == inode) {
                        if (mask & MAY_READ && perm->acc == ACC_DENY &&
                            perm->op == OP_READ) {
                                ret = -EPERM;
                                goto out;
                        }
                        ...
                }
        }
out:
        return ret;
}
```

通过 uid 查询用户信息，进一步获取当前用户的角色 role

lsm 实现- rbac_check_access


```
int rbac_check_access(uid_t uid, struct inode *inode, int mask)
{
        ...
        user = rbac_get_user_by_uid(uid);
        role = user->role;
        for (i = 0; i < ROLE_MAX_PERMS; i++) {
                perm = role->perms[i];
                if (perm != NULL && perm->obj == inode) {
                        if (mask & MAY_READ && perm->acc == ACC_DENY &&
                            perm->op == OP_READ) {
                                ret = -EPERM;
                                goto out;
                        }
                        ...
                }
        }
out:
        return ret;
}
```

遍历 role 中包含的每一条权限，根据 inode 和 mask 找到客体和操作与本次访问相同的权限

lsm 实现- rbac_check_access

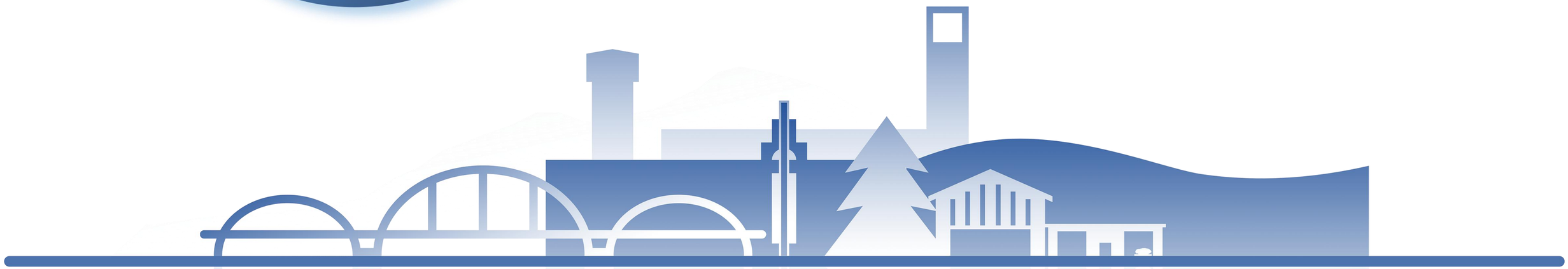

```
int rbac_check_access(uid_t uid, struct inode *inode, int mask)
{
        ...
        user = rbac_get_user_by_uid(uid);
        role = user->role;
        for (i = 0; i < ROLE_MAX_PERMS; i++) {
                perm = role->perms[i];
                if (perm != NULL && perm->obj == inode) {
                        if (mask & MAY_READ && perm->acc == ACC_DENY &&
                            perm->op == OP_READ) {
                                ret = -EPERM;
                                goto out;
                        }
                        ...
                }
        }
out:
        return ret;
}
```

如果找到匹配的权限，且接受性为 ACC_DENY，则拒绝访问

中国科学院大学
University of Chinese Academy of Sciences

03

实验结果



实验结果 - 获取 rbac-lsm 初始状态


```
# cd /sys/kernel/security/rbac/
# cat user && cat role && cat perm
uid: 0 acts as role "admin"
admin
        perm[0] id: 0
[0]: deny write on /init
#
```

此时包含一个用户，它将 uid 为 0 的用户（即 root）绑定到了角色 admin 上；admin 绑定了 id 为 0 的权限；该权限禁止写 /init

实验结果 - 检查 rbac-lsm 是否工作


```
# cat enable
rbac: enabled
# cat /init
#!/bin/sh
RBAC=/sys/kernel/security/rbac
mount -t devtmpfs none /dev
mount -t proc proc /proc
mount -t sysfs sys /sys
mount -t securityfs securityfs /sys/kernel/security
exec 1> /dev/console 2> /dev/console < /dev/console
echo 1 > $RBAC/enable
echo add user 0 > $RBAC/ctrl
echo add role admin > $RBAC/ctrl
echo add perm d w /init > $RBAC/ctrl
echo bind 0 admin > $RBAC/ctrl
echo register 0 admin > $RBAC/ctrl
exec /linuxrc
# echo "add a new line" > /init
-sh: can't create /init: Operation not permitted
#
```

目前 rbac 已开启

实验结果 - 检查 rbac-lsm 是否工作


```
# cat enable
rbac: enabled
# cat /init
#!/bin/sh
RBAC=/sys/kernel/security/rbac
mount -t devtmpfs none /dev
mount -t proc proc /proc
mount -t sysfs sys /sys
mount -t securityfs securityfs /sys/kernel/security
exec 1> /dev/console 2> /dev/console < /dev/console
echo 1 > $RBAC/enable
echo add user 0 > $RBAC/ctrl
echo add role admin > $RBAC/ctrl
echo add perm d w /init > $RBAC/ctrl
echo bind 0 admin > $RBAC/ctrl
echo register 0 admin > $RBAC/ctrl
exec /linuxrc
# echo "add a new line" > /init
-sh: can't create /init: Operation not permitted
#
```

此时可以读 /init

实验结果 - 检查 rbac-lsm 是否工作


```
# cat enable
rbac: enabled
# cat /init
#!/bin/sh
RBAC=/sys/kernel/security/rbac
mount -t devtmpfs none /dev
mount -t proc proc /proc
mount -t sysfs sys /sys
mount -t securityfs securityfs /sys/kernel/security
exec 1> /dev/console 2> /dev/console < /dev/console
echo 1 > $RBAC/enable
echo add user 0 > $RBAC/ctrl
echo add role admin > $RBAC/ctrl
echo add perm d w /init > $RBAC/ctrl
echo bind 0 admin > $RBAC/ctrl
echo register 0 admin > $RBAC/ctrl
exec /linuxrc
# echo "add a new line" > /init
-sh: can't create /init: Operation not permitted
#
```

但是试图写 /init 时被拒绝了

实验结果 - 添加并修改权限


```
# echo add perm d r /init > ctrl && echo unbind 0 admin > ctrl && echo bind 1 ad
min > ctrl
# cat user && cat role && cat perm
uid: 0 acts as role "admin"
admin
        perm[0] id: 1
[0]: deny write on /init
[1]: deny read on /init
# cat /init
cat: can't open '/init': Operation not permitted
# echo "add a new line" > /init
#
```

添加一条对 /init 读的禁止权限，并将原来的权限替换为这条新的权限

实验结果 - 添加并修改权限


```
# echo add perm d r /init > ctrl && echo unbind 0 admin > ctrl && echo bind 1 ad
min > ctrl
# cat user && cat role && cat perm
uid: 0 acts as role "admin"
admin
        perm[0] id: 1
[0]: deny write on /init
[1]: deny read on /init
# cat /init
cat: can't open '/init': Operation not permitted
# echo "add a new line" > /init
#
```

权限 1 已经绑定到了名字为 admin 的 role 上，即禁止读 /init，由于解除了权限 0，现在该角色对应的用户可以写 /init

实验结果 - 添加并修改权限


```
# echo add perm d r /init > ctrl && echo unbind 0 admin > ctrl && echo bind 1 ad
min > ctrl
# cat user && cat role && cat perm
uid: 0 acts as role "admin"
admin
        perm[0] id: 1
[0]: deny write on /init
[1]: deny read on /init
# cat /init
cat: can't open '/init': Operation not permitted
# echo "add a new line" > /init
#
```

读 /init 访问被拒绝，但写 /init 访问被接受

实验结果 - 将权限改回


```
# echo unbind 0 admin > ctrl && echo bind 0 admin > ctrl
# cat user && cat role && cat perm
uid: 0 acts as role "admin"
admin
        perm[0] id: 0
[0]: deny write on /init
[1]: deny read on /init
# cat /init
add a new line
# echo "add a new line" >> /init
-sh: can't create /init: Operation not permitted
#
```

将刚刚新添加的权限 1 解绑，将权限 0 绑定

实验结果 - 将权限改回


```
# echo unbind 0 admin > ctrl && echo bind 0 admin > ctrl
# cat user && cat role && cat perm
uid: 0 acts as role "admin"
admin
        perm[0] id: 0
[0]: deny write on /init
[1]: deny read on /init
# cat /init
add a new line
# echo "add a new line" >> /init
-sh: can't create /init: Operation not permitted
#
```

名字为 admin 的 role 又具有了“禁止写 /init”的权限，没有了“禁止读 /init”的权限

实验结果 - 将权限改回


```
# echo unbind 0 admin > ctrl && echo bind 0 admin > ctrl
# cat user && cat role && cat perm
uid: 0 acts as role "admin"
admin
        perm[0] id: 0
[0]: deny write on /init
[1]: deny read on /init
# cat /init
add a new line
# echo "add a new line" >> /init
-sh: can't create /init: Operation not permitted
#
```

现在 uid 为 0 的用户又可以读 /init 了，并且 /init 中的内容已经覆盖为了刚刚写入的字符串，试图写 /init 时，访问被拒绝

实验结果 - 检查开关是否工作


```
# echo 0 > enable
# cat enable
rbac: disabled
# echo "add a new line" >> /init
# cat /init
add a new line
add a new line
#
```

关闭 rbac 开关

实验结果 - 检查开关是否工作


```
# echo 0 > enable
# cat enable
rbac: disabled
# echo "add a new line" >> /init
# cat /init
add a new line
add a new line
#
```

现在 uid 为 0 的用户既可以写 /init 了，又可以读 /init 了，访问不受 rbac 的限制

中国科学院大学
University of Chinese Academy of Sciences

感谢观看

Thank you for watching