

摘要

在 Linux 操作系统中，使用了基于自主访问控制的用户权限管理系统。在这种权限控制模型下，Linux 的用户可以把自身的部分权限分配给其他用户，也可以自主设置其他用户对于该用户所拥有文件的访问权限。这种权限控制模型操作方便，但在安全性上存在较大风险。为了对 linux 进行安全加固，可以考虑引入具有更高安全性的基于强制访问控制的用户权限管理系统。因此，本项目的目标是通过系统调用函数重载的方法，实现一个基于强制访问控制的进程运行权限管理系统。由于在 linux 系统中，进程会继承用户的权限，因此本项目将通过对用户的权限设置实现对于进程的权限管理。

本项目的工作主要分为内核模块、后台服务器、前端 CLI、权限数据库与日志文件。内核模块功能有：对系统调用函数进行重载；使用 netlink 功能向后台服务器请求权限信息；通过权限信息判断是否允许相关系统调用函数执行。后台服务器功能有：常驻用户态后台运行；处理来自用户态模块的权限查询请求；处理来自前端 CLI 的权限管理操作；连接并管理权限数据库中的权限信息；将权限管理员权限操作记录写入用户态日志文件。前端 CLI 功能有：权限管理员通过操作前端 CLI 进行权限管理；提供 client 端使管理者可以通过命令行管理用户文件等的权限；管理的对象包括用户权限、文件目录权限、用户对网络等特殊对象的权限；管理操作包括设置、获取、删除权限。权限数据库功能有：保存用户和文件的权限信息。日志文件功能有：保存权限管理员权限操作记录，将操作按照天分割存储在不同文件，日志记录时首先获取并记录时间再根据类型（INFO、ERROR）等记录信息。

最终，本项目在 Ubuntu20.04 操作系统上完成，内核版本号为 5.15.0-xx-generic。本项目实现了目标的权限管理系统功能，通过了项目测试，实现了成功的基于强制访问控制的基于系统调用重载的程序运行权限管理系统开发。

目录

1	项目需求分析	1
1.1	需求分析	1
1.2	功能目标	1
2	项目总体设计	2
2.1	项目总体架构	2
2.2	信息交互	2
2.3	组成部分简介	3
3	内核模块	3
3.1	内核 Netlink 模块	3
3.2	内核 Hook 模块	5
4	后台服务器	5
4.1	用户态 Netlink 模块	5
4.2	数据库模块	6
4.3	Unix Socket 模块	6
4.4	日志模块	6
5	前端 CLI	6
6	权限数据库	7
7	日志文件	7
8	系统实现	7
8.1	实验环境	7
8.2	项目源文件代码结构	8
9	项目测试	9
9.1	测试前的准备	9
9.2	系统运行	10
9.3	测试用户权限与文件、目录权限	10
9.4	测试网络权限	14
9.5	测试 reboot 权限	15
9.6	测试系统日志权限	16

1 项目需求分析

1.1 需求分析

在 Linux 操作系统中，使用了基于自主访问控制的用户权限管理系统。在这种权限控制模型下，Linux 的用户可以把自身的部分权限分配给其他用户，也可以自主设置其他用户对于该用户所拥有文件的访问权限。

这种权限控制模型操作方便，但在安全性上存在较大风险。为了对 linux 进行安全加固，可以考虑引入具有更高安全性的基于强制访问控制的用户权限管理系统。因此，本项目的目标是通过系统调用函数重载的方法，实现一个基于强制访问控制的进程运行权限管理系统。

1.2 功能目标

由于在 linux 系统中，进程会继承用户的权限，因此本项目将通过对用户的权限设置实现对于进程的权限管理。

本项目实现的功能目标如下：

(1). 进程的文件、目录访问权限控制；

- 用户分为 4 个权限等级：1, 2, 3, 4；
- 文件和目录分为 4 个权限等级：1, 2, 3, 4；
- 权限等级数值越大，权限等级越高；
- 进程可以访问权限等级相当和权限等级较低的文件和目录。

(2). 进程的网络权限控制；

- 控制进程是否能够使用网络通信功能。

(3). root 用户进程的 reboot 权限控制；

- reboot 权限指包含系统重启、启用/禁用 Ctrl-Alt-Del 的一组权限，在 linux 的安全设置下，只有 root 用户有权限调用；
- 控制 root 用户是否能够使用 reboot 权限。

(4). 进程的系统内核日志权限控制；

- 在 linux 的安全设置下，普通用户可以查看系统内核日志，但只有 root 用户可以删除系统内核日志；

- 控制普通用户是否能够查看系统内核日志，控制 root 用户是否能够查看、删除系统内核日志。

(5). 日志记录。

- 将越权行为记录在系统内核日志；
- 将权限设置记录在用户态日志文件。

2 项目总体设计

2.1 项目总体架构

本项目是单主机项目，无网络拓扑架构。项目的总体架构图如图 1 所示：

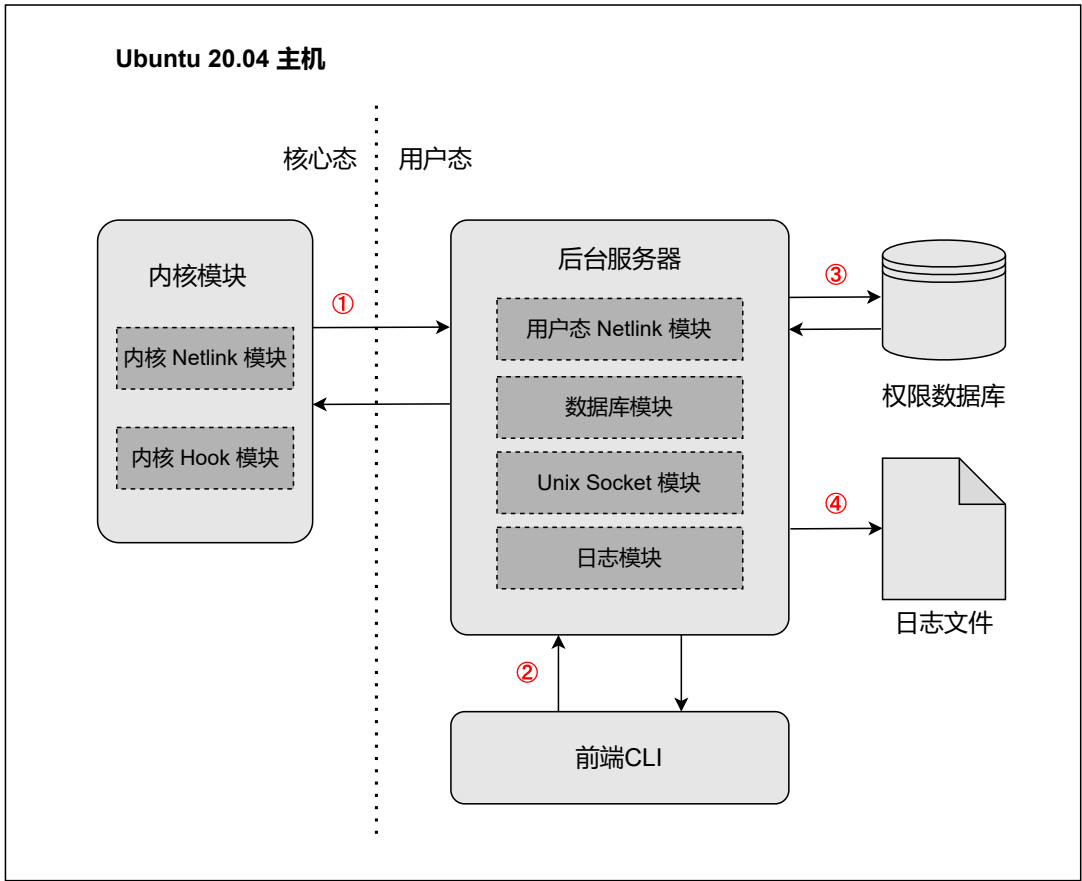


图 1: 项目总体架构图

2.2 信息交互

图 1 中包含的信息交互含义为：

- ①：内核模块向后台服务器进行权限查询，后台服务器返回查询结果；
- ②：权限管理员使用前端 CLI 对后端服务器进行权限设置、权限修改、权限删除、权限查询；
- ③：后台服务器对权限数据库进行权限设置、权限修改、权限删除、权限查询；
- ④：后台服务器将权限管理员的权限操作记录写入日志。

2.3 组成部分简介

本项目的主要组成分为：

- 内核模块：对系统调用函数进行重载；使用 netlink 功能向后台服务器请求权限信息；通过权限信息判断是否允许相关系统调用函数执行；
- 后台服务器：常驻用户态后台运行；处理来自用户态模块的权限查询请求；处理来自前端 CLI 的权限管理操作；连接并管理权限数据库中的权限信息；将权限管理员权限操作记录写入用户态日志文件；
- 前端 CLI：权限管理员通过操作前端 CLI 进行权限管理；提供 client 端使管理者可以通过命令行管理用户文件等的权限；管理的对象包括用户权限、文件目录权限、用户对网络等特殊对象的权限；管理操作包括设置、获取、删除权限；
- 权限数据库：保存用户和文件的权限信息；
- 日志文件：保存权限管理员权限操作记录，将操作按照天分割存储在不同文件，日志记录时首先获取并记录时间再根据类型（INFO、ERROR）等记录信息。

3 内核模块

本项目中内核模块以 linux 内核模块的形式运行在核心态，其主要包含两个模块：

- 内核 Netlink 模块
- 内核 Hook 模块

3.1 内核 Netlink 模块

内核 Netlink 模块主要实现三个功能：

- (1). 实现内核态与用户态数据通信；

- (2). 实现内核态权限查询函数;
- (3). 实现内核态权限查询进程与 netlink 消息处理进程之间的数据传递与进程同步。

为了实现功能，内核 Netlink 模块使用了以下结构体：

```
// 在netlink标准消息的基础上添加的基础消息结构
struct prm_nlmsg {
    struct nlmsgghdr nlh;
    u32  msg_len;
    u8   msg_data[PAYLOAD_MAX_SIZE];
};

// 用户态与核心态之间发送的消息的结构
struct prm_msg {
    s32  index;    // 在模块中使用atomic_t的值，为了减少处理，取值范围是signed int
    u32  type;     // 消息类型
    u32  ino;      // inode编号
    u32  uid;      // 用户uid
    s32  p_type;   // 权限类型
    s32  result_type; // 权限查询结果
    u64  sem_msg_ptr; // 消息标识
};

// 用户内核态进程间数据传递的结构体
struct sem_msg {
    u32  status;
    s32  data;    // 取值与prm_msg.result_type一致
    struct semaphore sem;
};
```

为实现内核态与用户态数据通信的功能，本模块在加载时使用 static int k2u_socket_create(void) 函数建立 Netlink 端口，在卸载时使用 static int k2u_socket_close(void) 清理 Netlink 端口，在使用中通过 prm_nlmsg 结构体与 prm_msg 结构体进行消息传递，通过 int k2u_send(char *buf, size_t len) 函数向用户态发送消息，通过 static void netlink_message_handle(struct sk_buff *skb) 处理用户态返回的消息。

为实现内核态权限查询函数，本模块使用 prm_msg 结构体搭载权限查询参数，通过调用 int check_privilege(unsigned long ino, uid_t uid, int p_type, int *result) 向用户态发起权限查询请求。出于稳定性考虑，本模块会记录用户态模块的上线情况，如果用户态模块未上线，则不会发送权限查询请求，直接返回“用户态未上线”的查询结果；如果用户态权限查询结果返回超时或出错，在累积三次后，内核态会将用户态模块标记为“未上线”，需要用户态模块重新发起连接。

为实现进程间数据传递与进程同步，本模块使用结构体 sem_msg 作为进程间数据传递的结构体，在发送的权限查询消息与返回的权限查询结果中添加指向 sem_msg 结构体的指针，通过消息量与内核态共享内存空间的特性，实现内核态权限查询进程与 netlink 消息处理进程之间的数据传递与进程同步。

3.2 内核 Hook 模块

本项目内核 Hook 模块实现以下功能：

- (1). 内存保护修改；
- (2). 系统调用函数重载；
- (3). 系统调用函数修改。

内核 Hook 模块在加载时查找系统调用表地址，取消内存写保护，保存原始系统调用函数地址，重载系统调用函数，开启内存写保护。内核 Hook 模块在卸载时取消内存写保护，恢复原始系统调用函数，关闭内存写保护。

本模块重载的系统调用函数及其修改作用如下：

- (1). `sys_openat()`, `sys_read()`, `sys_write()`: 进行文件、目录的访问控制；
- (2). `sys_reboot()`: 进行 reboot 权限的访问控制；
- (3). `sys_socket()`: 进行网络权限的访问控制；
- (4). `sys_execve()`: 进行系统内核日志的访问控制。

4 后台服务器

本项目中后台服务器以后台程序的形式运行在用户态，其主要包含四个模块：

- 用户态 Netlink 模块
- 数据库模块
- Unix Socket 模块
- 日志模块

4.1 用户态 Netlink 模块

用户态 Netlink 主要功能是与核心态 Netlink 模块进行对接，辅助实现核心态 Netlink 模块的功能。因此，用户态 Netlink 模块与核心态 Netlink 模块共享 `prm_nlmsg` 与 `prm_msg` 两个结构体。

用户态 Netlink 模块在后台服务器启动后，向核心态发送 Netlink 消息，以建立核心态 Netlink 模块与用户态 Netlink 模块的连接。

在用户态 Netlink 模块收到来自核心态的权限查询请求后，将会调用数据库模块进行权限查询，将权限查询的结构返回给核心态 Netlink 模块。

4.2 数据库模块

在数据库的选择上，我们使用了 sqlite3 数据库存储数据并对数据进行增删查改。

在数据库的设计上，我们共设计了若干张表，其中每两张表进行配对。一张表负责记录对象（如文件、进程）的等级，另一张表格记录用户对该类对象的等级。每张表均有两个字段，分别为 id/key 和 level。以文件举例，给定某文件号，从文件数据库中查询该文件对应的等级；再给定用户 uid，在用户-文件数据库中查询该用户对文件这一类对象的权限等级，之后进行相应的操作。

对于数据库的操作，分为数据库的创建与删除、表的创建与删除、对象等级的增删查改、用户等级的增删查改。

4.3 Unix Socket 模块

本项目用户端 client 和服务端 server 使用 UNIX Socket 通信,使用/tmp/client.socket 和/tmp/server.socket 作为客户端和服务端使用的 socket 路径，使用 socket() 函数在函数中创建 socket，服务器使用 listen() 等待客户端连接并使用 accept() 接受客户端连接，客户端使用 connect() 连接服务器进行操作。

客户端连接服务器后，每次运行仅仅读取一个命令并进行一次操作，通过命令行解析参数后构造 request 结构体并通过 send() 函数发送。服务器使用 recv() 函数接受结构体做出操作后构造 rsq 结构体返回给服务器，服务器获得结果后记录日志并通过 CLI 向用户展示结果。操作完成后，客户端和服务端分别使用 close() 关闭连接。

4.4 日志模块

本项目使用日志模块记录客户端和服务器的操作信息与错误信息。

log.h 中使用 logwrite() 日志接口,调用接口可以根据级别写入日志,级别包括 INFO、DEBUG、WARN、ERROR 等等，日志文件保存在 log/文件夹下。

调用日志接口时首先获取当前日期，根据日期分割存储的日志文件，不同日期的日志保存在不同文件下。保存时向文件中追加内容，内容前增加时间戳和日志级别。

客户端和服务端在接收发送消息、进行其他操作时记录日志并设置级别为 INFO，在出现任何预计到的问题时同样调用日志接口保存错误信息并设置级别为 ERROR。

5 前端 CLI

本项目中，前端 CLI 是提供给权限管理员进行权限管理的用户态可执行程序。

前端 CLI 使用 getopt() 函数解析权限管理员的命令行参数，通过 Unix Socket 与后台服务器通信，实现用户和文件等的权限管理操作，运行一次 CLI 命令只进行一次操

作，操作对象包括用户权限、文件权限、用户对网络等特殊对象的权限，操作类型包括设置、获取、删除权限。

命令行解析部分使用 `getopt()` 函数配合 `while()` 循环读取 CLI 命令，其中：-s 代表设置权限，后面必须接一个数字代表权限等级，-g 代表获取权限，-d 代表删除权限，-u 代表用户后面必须接一个用户名，-f 代表文件等后面可以接一个文件路径也可以接一个数字 5/6/7 代表网络等特殊对象，-h 会显示帮助信息，此外任何参数错误导致解析错误都会显示帮助信息。

命令行解析完成后会对用户名和文件路径进行处理，转换为用户号和文件 inode 号，再构造 req 数据结构后通过 UNIX Socket 发送给服务器并等待服务器返回。

服务器收到消息后根据操作类型调用数据库操作函数从而操作数据库，并将调用结果构造 rsp 结构返回给用户端。

对于服务器返回结果，首先判断其是否成功，如果成功对于或者权限操作再具体解析返回值获取权限值，最后将结果通过 CLI 输出给用户，这样一次操作完成后则关闭 socket，下次操作需要再次调用 `client.exe`。

6 权限数据库

权限数据库是保存在文件系统中的 sqlite3 格式的数据库文件，由后台服务器通过数据库模块进行连接与管理。

7 日志文件

日志文件是保存在文件系统中的一组文本文件，由后台服务器生成，记录权限管理员通过前端 CLI 的管理记录。

日志文件存储在 `log/` 文件夹下，按照日期来组织文件，不同日期的日志位于不同的文件。

每次调用 `logwrite()` 日志接口会向日志文件中增加一条日志，其结构包括：时间戳、日志级别、日志信息，client 和 server 的所有用户态操作和报错都会记录在日志中

8 系统实现

8.1 实验环境

本项目使用 Ubuntu 20.04 desktop 作为实验主机，操作系统内核版本为 5.15.0-xx-generic。

在 Ubuntu 20.04 desktop 的发行版基础上，本项目需要额外安装的依赖为：

- build_essential: 本项目依赖的编译工具与多种基础依赖库;
- libsqlites-dev : 本项目的数据库依赖函数库。

本项目使用 vscode 作为开发工具进行开发。

本项目使用的编译工具为 GNU Make 4.2.1, 使用的编译器为 gcc (Ubuntu 9.4.0-1ubuntu1 20.04.1) 9.4.0。

8.2 项目源文件代码结构

本项目 GitHub 链接为<https://github.com/FengweiZhang/SystemDesign-IS415-dev>。其中, 作为课程源代码文件提交的 src 目录代码结构如下:

```
src/                # 项目源代码
  kernel/           # 内核态源代码
    Makefile
    prm_error.h      # 内核态通用返回值定义
    prm_hook.c       # 内核态hook模块
    prm_hook.h
    prm_module.c     # 内核态模块主文件
    prm_netlink.c    # 内核态netlink模块
    prm_netlink.h
  user/             # 用户态源代码
    Makefile
    server.c         # 后端服务器, Unix Socket模块
    client.c         # CLI客户端
    operation.h
    socket_error.h
    database.c        # 数据库模块
    database.h
    database.db       # 权限数据库
    databaseExtension.c # 数据库模块
    databaseExtension.h
    log/             # 日志文件目录
    log.c            # 日志模块
    log.h
    log.conf         # 日志配置文件
    prm_error.h      # 同内核态prm_error.h, 仅用于user_netlink.c文件内部
    user_netlink.c    # 用户态netlink模块
    user_netlink.h
    test/            # 测试用函数、文件与脚本
```

源文件中, 除 user/test 目录下部分测试程序来自网络以外, 全部文件由本项目自主编写完成。

内核态中, prm_hook.c、prm_module.c、prm_netlink.c 一同编译生成内核模块 prm.ko。

用户态中, server.c、database.c、databaseExtension.c、user_netlink.c 共同编译生成用户态后台程序 server.exe; client.c、log.c 编译生成用户态 CLI client.exe。

9 项目测试

本项目的测试主要用于检测1.2中实现的功能。

测试在 Ubuntu 20.04 操作系统上实现，内核版本号为 5.15.0-xx-generic。

9.1 测试前的准备

在进行测试前，需要进行编译，编译指令如下：

```
# 编译内核模块
cd src/kernel
make

# 编译用户态程序
cd ../user
make

# 编译测试样例
cd test
make
```

此外，为了进行权限控制，本项目使用 ubuntu 用户进行权限设置，对五个新用户的权限进行控制，新用户如下：

```
pity:x:1001:1001::/home/pity:/bin/bash
pity1:x:1002:1002::/home/pity1:/bin/bash
pity2:x:1003:1003::/home/pity2:/bin/bash
pity3:x:1004:1004::/home/pity3:/bin/bash
pity4:x:1005:1005::/home/pity4:/bin/bash
```

图 2: 用于测试的用户

为了防止 linux 本身的权限控制产生影响，需要对 user/test/testfile 目录下的文件进行 linux 系统的权限设置，设置结果如下：

```
ubuntu@ubuntu:~/SystemDesign-IS415-dev/src/user/test/testfile$ ls -lh
total 20K
drwxrwxrwx 2 ubuntu ubuntu 4.0K Dec  5 20:29 t2
-rw-rw-rw- 1 ubuntu ubuntu  7 Dec  5 06:05 test1
-rw-rw-rw- 1 ubuntu ubuntu  7 Dec  5 06:05 test2
lrwxrwxrwx 1 ubuntu ubuntu  5 Dec  5 20:29 test2.l -> test2
-rw-rw-rw- 1 ubuntu ubuntu  7 Dec  5 06:05 test3
-rw-rw-rw- 1 ubuntu ubuntu  7 Dec  5 06:05 test4
```

图 3: 设置测试文件的 linux 权限

可见，所有 linux 用户在 linux 的权限控制下对于这些测试文件拥有完全相同的权限。

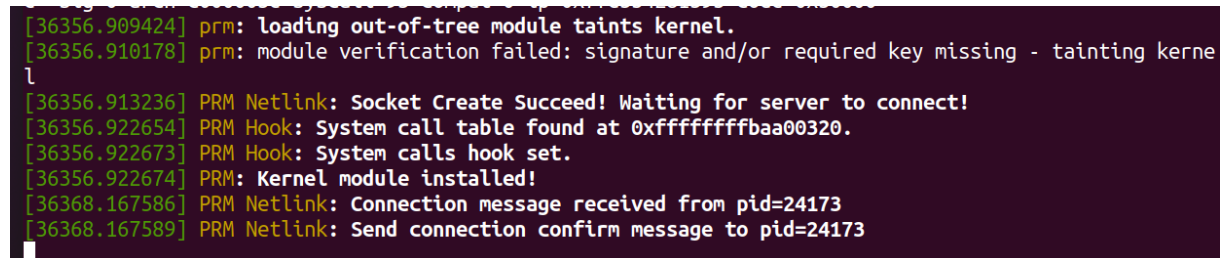
9.2 系统运行

使用如下指令加载内核态模块和运行用户态服务进程

```
# 安装内核模块
cd src/kernel
make ins

# 运行用户态服务进程
cd ../user
sudo ./server.exe
```

加载内核模块、启动用户态后台服务器后，内核日志输出：



```
[36356.909424] prm: loading out-of-tree module taints kernel.
[36356.910178] prm: module verification failed: signature and/or required key missing - tainting kernel
[36356.913236] PRM Netlink: Socket Create Succeed! Waiting for server to connect!
[36356.922654] PRM Hook: System call table found at 0xffffffffbba00320.
[36356.922673] PRM Hook: System calls hook set.
[36356.922674] PRM: Kernel module installed!
[36368.167586] PRM Netlink: Connection message received from pid=24173
[36368.167589] PRM Netlink: Send connection confirm message to pid=24173
```

图 4: 加载内核模块，启动用户态后台服务器后内核日志输出

9.3 测试用户权限与文件、目录权限

9.3.1 设置权限

首先需要设置用户权限与文件、目录权限，目标是设置成以下权限等级：

- (1). 为 pity1, pity2, pity3, pity4 用户分别设置 1, 2, 3, 4, 等级的权限
- (2). 为 test1, test2, test3, test4 文件分别设置 1, 2, 3, 4 等级的权限
- (3). 为 test2.l 设置 2 等级权限
- (4). 为 t2 目录设置 2 等级权限

使用测试脚本进行权限设置：

```
# 在src/user/test目录下
sh ./set_pri.sh
```

set__pri.sh 脚本内容如下：

```
#!/bin/bash

cd ..

# 设置用户的权限
echo "\n设置用户权限"
```

```
sudo ./client.exe -s 1 -u pity1
sudo ./client.exe -s 2 -u pity2
sudo ./client.exe -s 3 -u pity3
sudo ./client.exe -s 4 -u pity4

# 设置文件的权限
echo "\n设置文件权限"
sudo ./client.exe -s 1 -f ./test/testfile/test1
sudo ./client.exe -s 2 -f ./test/testfile/test2
sudo ./client.exe -s 3 -f ./test/testfile/test3
sudo ./client.exe -s 4 -f ./test/testfile/test4
sudo ./client.exe -s 2 -f ./test/testfile/test2.1
sudo ./client.exe -s 2 -f ./test/testfile/t2

# 查看用户权限
echo "\n查看用户权限"
sudo ./client.exe -g -u pity1
sudo ./client.exe -g -u pity2
sudo ./client.exe -g -u pity3
sudo ./client.exe -g -u pity4

# 查看文件权限
echo "\n查看文件权限"
sudo ./client.exe -g -f ./test/testfile/test1
sudo ./client.exe -g -f ./test/testfile/test2
sudo ./client.exe -g -f ./test/testfile/test3
sudo ./client.exe -g -f ./test/testfile/test4
sudo ./client.exe -g -f ./test/testfile/test2.1
sudo ./client.exe -g -f ./test/testfile/t2
```

权限设置结果如下图5 所示。可以看到，目标用户与目标文件、文件夹被设置为指定的权限。

9.3.2 测试对文件的访问控制

测试不同用户对文件的访问权限：

```
# 在src/user/test目录下
su pity1 ./test_rw.sh
su pity2 ./test_rw.sh
su pity3 ./test_rw.sh
su pity4 ./test_rw.sh
```

其中，test_rw.sh 脚本内容如下，其主要功能是依次尝试写入与读取每个测试文件：

```
#!/bin/bash

echo -e "\nwhoami"
whoami

echo -e "\ntry to write"
./test_write.out
```

```
ubuntu@ubuntu:~/SystemDesign-IS415-dev/src/user/test$ sh ./set_pri.sh

设置用户权限
[sudo] password for ubuntu:
set user pity1 level : 1
set user pity2 level : 2
set user pity3 level : 3
set user pity4 level : 4

设置文件权限
file inode : 2359777, set file ./test/testfile/test1 level : 1
file inode : 2359778, set file ./test/testfile/test2 level : 2
file inode : 2359779, set file ./test/testfile/test3 level : 3
file inode : 2359780, set file ./test/testfile/test4 level : 4
file inode : 2359778, set file ./test/testfile/test2.l level : 2
directory inode : 2359847, set file ./test/testfile/t2 level : 2

查看用户权限
get user pity1 level : 1
get user pity2 level : 2
get user pity3 level : 3
get user pity4 level : 4

查看文件权限
file inode : 2359777, get ./test/testfile/test1 level : 1
file inode : 2359778, get ./test/testfile/test2 level : 2
file inode : 2359779, get ./test/testfile/test3 level : 3
file inode : 2359780, get ./test/testfile/test4 level : 4
file inode : 2359778, get ./test/testfile/test2.l level : 2
directory inode : 2359847, get ./test/testfile/t2 level : 2
```

图 5: 设置权限脚本输出

```
echo -e "\ntry to read"
./test_read.out
```

不同用户对文件的访问测试结果如图6。可以看到，用户只能访问权限等于或低于该用户的文件。

```
ubuntu@ubuntu:~/SystemDesign-IS415-dev/src/user/test$ su pity1 ./test_rw.sh
whoami
pity1

try to write
write test1 succeed
open test2 failed: Operation not permitted
open test2.l failed: Operation not permitted
open test3 failed: Operation not permitted
open test4 failed: Operation not permitted

try to read
read test1 succeed
open test2 failed: Operation not permitted
open test2.l failed: Operation not permitted
open test3 failed: Operation not permitted
open test4 failed: Operation not permitted
ubuntu@ubuntu:~/SystemDesign-IS415-dev/src/user/test$ su pity2 ./test_rw.sh
whoami
pity2

try to write
write test1 succeed
write test2 succeed
write test2.l succeed
open test3 failed: Operation not permitted
open test4 failed: Operation not permitted

try to read
read test1 succeed
read test2 succeed
read test2.l succeed
open test3 failed: Operation not permitted
open test4 failed: Operation not permitted
ubuntu@ubuntu:~/SystemDesign-IS415-dev/src/user/test$ su pity3 ./test_rw.sh
whoami
pity3

try to write
write test1 succeed
write test2 succeed
write test2.l succeed
write test3 succeed
open test4 failed: Operation not permitted

try to read
read test1 succeed
read test2 succeed
read test2.l succeed
read test2 succeed
open test4 failed: Operation not permitted
ubuntu@ubuntu:~/SystemDesign-IS415-dev/src/user/test$ su pity4 ./test_rw.sh
whoami
pity4

try to write
write test1 succeed
write test2 succeed
write test2.l succeed
write test3 succeed
write test4 succeed

try to read
read test1 succeed
read test2 succeed
read test2.l succeed
read test2 succeed
read test2 succeed
```

图 6: 不同用户访问文件结果

9.3.3 测试对目录的访问控制

通过以下指令进行对目录的访问控制测试

```
# 在src/user/test/testfile目录下
su pity1
cd t2
ls          # 尝试读取t2目录
exit

su pity2
cd t2
ls          # 尝试读取t2目录
exit
```

不同用户对目录的访问测试结果如图7。可以看到，文件夹 t2 权限为 2，权限为 1 的 pity1 无法访问 t2 下的内容，但权限为 2 的 pity2 可以访问 t2 下的内容。

```
ubuntu@ubuntu:~/SystemDesign-IS415-dev/src/user/test/testfile$ su pity1
pity1@ubuntu:/home/ubuntu/SystemDesign-IS415-dev/src/user/test/testfile$ cd t2
pity1@ubuntu:/home/ubuntu/SystemDesign-IS415-dev/src/user/test/testfile/t2$ ls
ls: cannot open directory '.': Operation not permitted
pity1@ubuntu:/home/ubuntu/SystemDesign-IS415-dev/src/user/test/testfile/t2$ exit
exit
ubuntu@ubuntu:~/SystemDesign-IS415-dev/src/user/test/testfile$ su pity2
pity2@ubuntu:/home/ubuntu/SystemDesign-IS415-dev/src/user/test/testfile$ cd t2
pity2@ubuntu:/home/ubuntu/SystemDesign-IS415-dev/src/user/test/testfile/t2$ ls
pity2@ubuntu:/home/ubuntu/SystemDesign-IS415-dev/src/user/test/testfile/t2$ exit
exit
ubuntu@ubuntu:~/SystemDesign-IS415-dev/src/user/test/testfile$
```

图 7: 不同用户访问文件夹结果

9.4 测试网络权限

运行网络权限测试脚本，对网络权限控制进行测试

```
# 在src/user/test目录下
sh ./test_net.sh
```

网络权限测试脚本内容如下：

```
#!/bin/bash

cd ..

echo "\n禁止pity的网络权限"
sudo ./client.exe -s 0 -u pity -f 6

echo "\npity尝试访问网络"
su pity -c ./test/test_net.out

echo "\n恢复pity的网络权限"
sudo ./client.exe -s 1 -u pity -f 6
```



```
echo "\npity尝试访问网络"
su pity -c ./test/test_net.out
```

网络测试输出如图8，用户 pity 先被禁止了网络的访问，无法访问网络，之后允许了 pity 的网络权限后，pity 得以获取网页内容。

```
ubuntu@ubuntu:~/SystemDesign-IS415-dev/src/user/test$ sh ./test_net.sh
禁止pity的网络权限
ban user pity permission of network!

pity尝试访问网络
socket: Operation not permitted

恢复pity的网络权限
give user pity permission of network!

pity尝试访问网络
HTTP/1.0 200 OK
Bdpagetype: 1
Bdqid: 0x87b09cbc0004c22a
Content-Type: text/html; charset=utf-8
Date: Sun, 25 Dec 2022 12:55:07 GMT
P3p: CP=" OTI DSP COR IVA OUR IND COM "
P3p: CP=" OTI DSP COR IVA OUR IND COM "
Server: BWS/1.1
Set-Cookie: BAIDUID=8D9C648A847BB89817375DCFFDCB7F45;FG=1; expires=Thu, 31-Dec-37 23:55:55 GMT; max-age=2147483647; path=/; domain=.baidu.com
Set-Cookie: BIDUPSID=8D9C648A847BB89817375DCFFDCB7F45; expires=Thu, 31-Dec-37 23:55:55 GMT; max-age=2147483647; path=/; domain=.baidu.com
Set-Cookie: PSTM=1671972907; expires=Thu, 31-Dec-37 23:55:55 GMT; max-age=2147483647; path=/; domain=.baidu.com
Set-Cookie: BAIDUID=8D9C648A847BB89810F9A2F5CC357C34;FG=1; max-age=31536000; expires=Mon, 25-Dec-23 12:55:07 GMT; domain=.baidu.com; path=/; version=1; comment=bd
Set-Cookie: BDSVRTM=28; path=/
Set-Cookie: BD_HOME=1; path=/
Set-Cookie: H_PS_PSSID=36545_37647_37689_37909_37623_37800_37948_37931_37902_26350_37881; path=/; domain=.baidu.com
Strict-Transport-Security: max-age=0
Traceid: 167197290735620887149777487122289443370
Vary: Accept-Encoding
X-Frame-Options: sameorigin
X-UA-Compatible: IE=Edge,chrome=1
```

图 8: 测试对用户网络的控制

9.5 测试 reboot 权限

运行 reboot 权限测试脚本，对 root 用户的 reboot 权限进行测试：

```
# 在src/user/test目录下
sh ./test_reboot.sh
```

reboot 权限测试脚本内容如下：

```
#!/bin/bash

cd ../

echo "\n禁止root用户reboot权限"
sudo ./client.exe -s 0 -u root -f 5

echo "\nroot用户尝试重启计算机"
sudo ./test/test_reboot.out

echo "\n启动root用户reboot权限"
```

```
sudo ./client.exe -s 1 -u root -f 5
```

reboot 系列权限测试结果如图9。可以看到，被禁用 reboot 权限后，root 用户无法重启电脑，恢复权限后可以重启，但是没有在图9中展示，因为电脑会直接关闭，已经在功能展示环节现场展示。

```
ubuntu@ubuntu:~/SystemDesign-IS415-dev/src/user/test$ sh ./test_reboot.sh
禁止root用户reboot权限
ban user root permission of reboot!

root用户尝试重启计算机
Reboot failed : Operation not permitted

启动root用户reboot权限
give user root permission of reboot!
```

图 9: 测试对 root 用户 reboot 权限的控制

9.6 测试系统日志权限

运行系统日志权限测试脚本，对系统日志权限进行测试：

```
# 在src/user/test目录下
sh ./test_dmesg.sh
```

系统日志权限测试脚本内容如下：

```
#!/bin/bash

cd ..

echo "\n禁止root日志访问权限"
sudo ./client.exe -s 0 -u root -f 7

echo "\nroot用户尝试删除日志"
sudo dmesg -C

echo "\n恢复root对日志权限"
sudo ./client.exe -s 1 -u root -f 7

echo "\nroot用户尝试查看日志"
sudo dmesg | tail -n 20

echo "\nroot用户尝试删除日志"
sudo dmesg -C

echo "\nroot用户检查日志"
sudo dmesg
```

系统日志的权限控制测试结果如图10所示。可以看到，先禁用了 root 的日志访问权限，之后 root 无法删除日志，在恢复 root 用户的日志访问权限后，root 用户先查看

日志，再删除日志，再次查看日志，发现日志已经被删除，说明 root 用户的日志访问权限已经恢复。

```
ubuntu@ubuntu:~/SystemDesign-IS415-dev/src/user/test$ sh ./test_dmesg.sh
```

禁止root日志访问权限

```
ban user root permission of kernel log!
```

root用户尝试删除日志

```
sudo: unable to execute /usr/bin/dmesg: Operation not permitted
```

恢复root对日志权限

```
give user root permission of kernel log!
```

root用户尝试查看日志

```
[38140.760581] Block: open REG file uid=1002 inode=2359778
[38140.760912] Block: open REG file uid=1002 inode=2359779
[38140.761283] Block: open REG file uid=1002 inode=2359780
[38140.765214] Block: open REG file uid=1002 inode=2359778
[38140.765407] Block: open REG file uid=1002 inode=2359778
[38140.765565] Block: open REG file uid=1002 inode=2359779
[38140.765721] Block: open REG file uid=1002 inode=2359780
[38146.406760] Block: open REG file uid=1003 inode=2359779
[38146.406914] Block: open REG file uid=1003 inode=2359780
[38146.413102] Block: open REG file uid=1003 inode=2359779
[38146.413214] Block: open REG file uid=1003 inode=2359780
[38151.222053] Block: open REG file uid=1004 inode=2359780
[38151.225132] Block: open REG file uid=1004 inode=2359780
[38324.794805] Block: open DIR uid=1002 inode=2359847
[38357.473500] Block: open DIR uid=1002 inode=2359847
[38478.498419] Block: net 1001
[38478.498794] Block: net 1001
[38478.501253] Block: net 1001
[38668.265650] Block: reboot 0
[38816.650321] Block: dmesg 0
```

root用户尝试删除日志

root用户检查日志

图 10: 测试对 root 用户系统日志权限的控制