

天天做早操队

也可查看 [README.pdf](#)

队伍信息

- 高校名称：合肥工业大学
- 队伍ID：T202410359994653
- 队伍名：天天做早操队
- 队员：王腾昊 刘诗雨
- 指导老师：田卫东 周红鹃
- 选题方向：OS内核赛道-实现一个小型内核
- 项目名称：基于XV6-K210的文件系统权限管理

代码基础

- 代码基础：基于华中科技大学 xv6-K210 仓库代码
- 代码框架链接：<https://gitee.com/hustos/xv6-k210>
- 代码基础作为知名国赛代码框架，仅包含最基本的引导/启动模块和简单的文件系统与进程管理等功能。

运行平台

- qemu 5.1.0 模拟器平台
- riscv64-unknown-elf-toolchain
- RUSTBI Bootloader

实现功能

1. 参照 Linux 权限模型，实现了基于 FAT32 文件系统的文件权限管理模型。
2. 实现了类 Linux 用户/用户组模型，实现用户的动态绑定，能够添加用户/修改激活用户/查看用户信息。
3. 优化了相关用户工具程序，提升了使用体验。
4. 修复了有关 Bootloader 硬件兼容性的 bug。

工作量

添加或修改约 2800 行代码

系统实现：

启动界面

[illegible]

具体功能

1. 文件权限管理

- 实现了基于FAT32文件系统的文件权限管理模型，包括文件权限的读取、设置和检查。
- 类似于UNIX文件访问矩阵，设置了 `owner`、`Grouper` 和 `other` 三种权限，每种权限包括读、写和执行权限。
- 对于每种权限给出了其修改的接口 `chmod`。
- 除权限矩阵外，还设置了每个文件的 `owner_id` 和 `Grouper_id`，用于用户分类权限的检查。
- 给出了每个文件的所有者和所属用户组的修改和获取接口 `chgrp` 和 `chown`。

由于FAT32本身不支持类UNIX的文件权限管理机制，所以在实现过程中，我们将FAT32的无用和保留字段替换为权限矩阵的值，通过设置该字段，在不修改原有FAT32文件系统的基础上完成了文件权限的各类机制和功能。

```
1 typedef struct short_name_entry {
2     char        name[CHAR_SHORT_NAME];
3     uint8       attr;
4     uint8       _nt_res;
5     uint8       _crt_time_tenth;
6     uint16      perm_num;           // 权限矩阵
7     uint16      owner_id;          // 所有者用户id
8     uint16      group_id;          // 所有者用户组id
9     uint16      fst_clus_hi;
10    uint16      _lst_wrt_time;
11    uint16      _lst_wrt_date;
12    uint16      fst_clus_lo;
```

```
13     uint32      file_size;
14 } __attribute__((packed, aligned(4))) short_name_entry_t;
15
```

2. 用户/用户组管理

用户和用户组管理与文件权限管理是相辅相成的，在本项目中，我们实现了以下用户管理功能：

- 实现了类Linux用户/用户组模型，包括用户和用户组的动态绑定。
- 实现了用户和用户组的添加、修改和查看接口 `new_user`、`change_user` 和 `find_user`。
- 用户和用户组信息存储在内存中，通过 `id` 接口获取当前用户id，通过 `get_now_user` 接口获取当前用户信息。
- 实现了用户密码的鉴权登录和切换。

在用户管理中，我们维护了一个ACB（Access Control Block）表，里面存储每个用户的信息，在系统初始化时，自动加载一个名为root并具有root权限的用户。

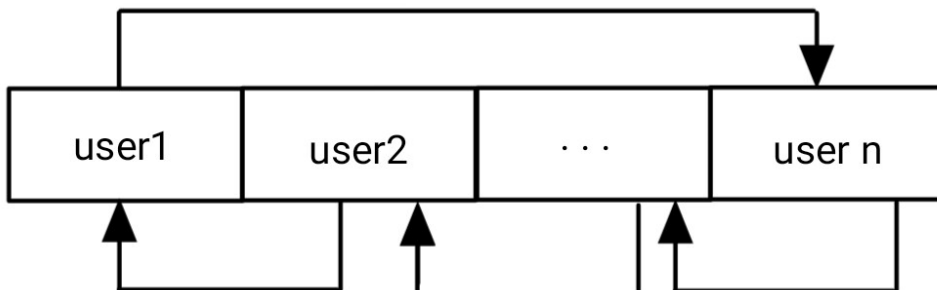
在后续通过维护用户链表完成用户数据的访问、修改、添加和删除。

我们不针对某个进程保存用户信息，而是针对整个系统设置当前系统运行状态的用户。

我们维护一个指针，指向 ACB 列表中现在激活的用户（Active User）（默认是root），在用户切换时，修改该指针即可。

文件访问和执行的鉴权也依赖当前系统运行在哪个用户下，需访问该指针确认系统当前状态是否有权限读/写/执行某一文件。

用户权限控制块形成的数据结构如下所示。



由于涉及临界资源操作，我们为ACB链表添加了信号量机制，防止越权访问造成的不可重复性。

3. 相关用户命令和系统调用

sh

由于涉及用户的显示，优化了命令行的显示模式，在最左侧显示当前激活的用户名。

```
root @ / $
```

ls

由于为文件增加了权限的管理，需要有方便的工具展示详细的文件权限。

故优化了ls的显示模式，ls有以下两种用法。

1.显示基本信息。

```
1 | ls
```

```
root @ / $ ls
FILE_NAME      TYPE
home           DIR
bin            DIR
README         FILE
adduser        FILE
cat            FILE
chgrp          FILE
chmod          FILE
chown          FILE
echo           FILE
find           FILE
grep           FILE
id             FILE
init           FILE
kill           FILE
```

2. 显示详细信息。

这里和Linux的显示模式类似。

- 最左一列显示d/-，d表示目录，-表示文件。
- 第二列显示权限矩阵，rwx分别表示读、写、执行权限。
- 第三列显示文件所有者。
- 第四列显示文件所有者用户组。
- 第五列显示文件大小。

```
1 | ls -l
```

使用效果

```
root @ / $ ls -l
```

Authority	Owner	Group	NAME	SIZE
drw-----	1	1	home	0
drw-----	1	1	bin	0
-rw-----	1	1	README	2104
-rw-----	1	1	adduser	27488
-rw-----	1	1	cat	26240
-rw-----	1	1	chgrp	27200
-rw-----	1	1	chmod	29600
-rw-----	1	1	chown	28128
-rw-----	1	1	echo	25064
-rw-----	1	1	find	27664
-rw-----	1	1	grep	29544
-rw-----	1	1	id	26616
-rw-----	1	1	init	25624
-rw-----	1	1	kill	25016

chmod

更改用户权限矩阵。用法与Linux类似

```
1 | chmod [num] filename # 其中num为三位数字，分别代表所有者、用户组和其他用户的权限。
```

使用前

-rw-----	1	1	who	25088
----------	---	---	-----	-------

使用后

-rwxrwxrwx	1	1	who	25088
------------	---	---	-----	-------

chown

更改文件所有者。用法与Linux类似

```
1 | chown [num] filename # 其中num为用户id，大小在1~65535之间。
```

使用前

-rwxrwxrwx	1	1	who	25088
------------	---	---	-----	-------

使用后

-rwxrwxrwx	123	1	who	25088
------------	-----	---	-----	-------

chgrp

更改用户所属的用户组。用法与Linux类似

```
1 | chgrp [num] filename # 其中num为用户id，大小在1~65535之间。
```

使用前

```
-rwxrwxrwx 123 1 who 25088
```

使用后

```
-rwxrwxrwx 123 1225 who 25088
```

adduser

添加用户，给定其用户名、密码，用户的描述，以及所属的用户组。

```
1 | adduser -n username -p password [-d describe -g groupid] # 其中用户描述和用户组可以缺省，由系统自动指定。
```

使用效果

```
root @ / $ adduser -n lsy -p lsywth -d testlogin -g 100
root @ / $ id
uid=1(root) gid=1(root) describe=root(root)
root @ / $ id lsy
#####
# Username:lsy:
# Userid: 1001
# Groupid: 100
# Describe: testlogin
#####
```

id

1. 显示当前激活的用户信息。

```
1 | id
```

使用效果

```
root @ / $ id
uid=1(root) gid=1(root) describe=root(root)
```

2. 给定某一用户名，给出其详细信息。

```
1 | id username
```

使用效果

```
root @ / $ id lsy
#####
# Username:lsy:
# Userid: 1001
# Groupid: 100
# Describe: testlogin
#####
```

su

切换用户，给定用户名和密码。

对ACB中的密码进行鉴权，若正确，更新当前激活用户。

```
1 | su
2 | [Password:]*****
3 | Correct
```

使用效果:

密码正确

```
root @ / $ su lsy
[Password:]lsywth
Correct
```

密码错误

```
lsy @ / $ su root
[Password:]123
Password error
[Password:]root
Correct
root @ / $
```

who

一元命令，简略地显示当前用户的信息（用户组、描述等）。

```
1 | who
```

使用效果:

```
root @ / $ who
uid=1(root) gid=1(root) describe=root(root)
```

系统功能验证

有权限(root)的访问

```
-rw----- 1 1 who 25088
-rw----- 1 1 xargs 26880
drw----- 1 1 etc 0
root @ / $ mv who whochanged
moving [who] to [whochanged]
```

无权限(user)的访问

```
-rwxr--r-- 1 1 wc 27448
lsy @ / $ wc
Permission denied in exec wc
Permission denied in exec wc
exec wc failed
lsy @ / $
```

特别地，只有具有root权限的用户才能创建用户。
其余功能在前文已完成叙述。

系统调用

为实现上述功能，我们设计了如下的系统调用，修改原有系统调用和函数原语若干。

秉承着系统调用数量最小化的原则，我们复用了一些系统调用函数，从而避免内核过于臃肿。

若无特别实现的系统调用，则具体的功能在用户态通过组合已有系统调用实现。

```
1 | uint64 sys_id(void);
2 | uint64 sys_get_now_user(void);
3 | uint64 sys_change_perm(void);
4 | uint64 sys_change_owner(void);
5 | uint64 sys_change_group(void);
6 | uint64 sys_find_user(void);
7 | uint64 sys_new_user(void);
8 | uint64 sys_change_user(void);
```


总结和感悟

在深入探索并尝试实现基于访问权限表的系统安全管理这一相关题目时，深刻体会到了操作系统中访问控制技术的重要性和复杂性。

在类Linux系统中，文件访问权限模型是保障系统安全的核心机制之一。通过文件权限位（rwx）的设置，系统能够精确地控制用户对文件的访问行为。在实现这一模型的过程中，对于权限管理的精细化和灵活性有了进一步的理解。无论是文件的所有者、所属组还是其他用户，都可以根据实际需求被赋予不同的访问权限，这种设计既保证了资源的有效利用，又有效防止了未经授权的访问和操作。

在模拟文件及目录的数据结构时，需要准确地表示文件和目录的层级关系，高效地管理文件的属性（如名称、大小、权限等），以及支持文件的创建、删除、移动等操作。

在操作系统中，用户和用户组是管理资源和权限的基本单位。通过建立用户和用户组模型，可以更灵活地控制不同用户对文件的访问权限。在实现这一功能的过程中，体会到了用户管理的复杂性。如何确保用户信息的准确性，如何防止用户之间的权限冲突，以及如何根据业务需求动态调整用户权限，都需要综合考虑和细致规划。

设计读、写和执行三种指令是模拟不同用户访问文件或目录时所发生的受限情况的关键步骤。在这一过程中，不仅要考虑指令的功能实现，还要关注指令的安全性和效率。例如，在实现写指令时，需要确保只有拥有相应权限的用户才能对文件进行写操作；在实现执行指令时，需要验证文件的可执行属性以及用户的执行权限。

在实现上述功能的过程中，我们体会到了操作系统中访问控制技术的重要性和复杂性，以及在设计和实现过程中需要注意的细节。它不仅关系到系统的安全性和稳定性，还直接影响到用户的使用体验和资源的利用效率。

作为大三刚刚结束操作系统课程学习的学生，本次比赛是我们一个宝贵的实践机会，我们将以此为基础，继续操作系统内核的开发，持续学习，学无止境。

下一步计划

- ☐ 重构文件系统，使得文件介质可迁移
- ☐ 实现临时提权
- ☐ 实现文件系统中的隐藏文件
- ☐ 优化进程与权限管理的关系

工作记录

见Commit记录

使用方法

确保本机已安装riscv-toolchain和qemu，然后执行以下命令：

```
1 make fs
2 make build
3 make run
```