01 文件系统架构观察与权限机制

1. 本节导读

在之前的实验中，我们主要与内存和 CPU 打交道。本实验我们将目光投向“持久化存储”。在 Linux 的哲学中，“一切皆文件”，这不仅包括磁盘上的代码和文档，还包括硬件设备、进程间通信的管道等。

本节我们首先以 Linux 上的常规文件和目录为例，观察操作系统是如何在底层组织这些数据的。我们将打破“文件名即文件”的直观印象，引入 **Inode** 与 **Dentry** 的概念。接着，我们将通过 **SUID** 实验观察 Linux 的安全权限体系。最后，我们将探讨文件系统的物理边界——即 Inode 和磁盘块的配额限制。这些观察将为你后续在自己的内核中实现简化版文件系统奠定理论基础。

---

2. 课前基础知识储备：Inode 与目录

2.1 常规文件与元数据

在用户看来，文件是一个可以通过名字访问的字节序列。但在内核眼中，文件由两部分组成：**元数据（Metadata）** 和 **数据块（Data Blocks）**。

我们可以使用 `stat` 工具来观察一个文件。请在终端执行：

```
$ echo "Hello OS" > test.txt
$ stat test.txt
```

【关键信息解读】

- **Inode**：文件的底层编号。内核内部并不通过 `test.txt` 这个名字找文件，而是通过这个编号。它就像是文件在磁盘上的“身份证号”。
- **Links**：硬链接数。表示有多少个文件名指向同一个 Inode。
- **Access/Modify/Change**：分别记录了文件的读取、内容修改、元数据修改时间。

【观察与思考】

- 记录下 `Inode` 编号。
- 观察 `Links`（链接数）的值是多少？
- `Access`/`Modify`/`Change` 三个时间戳分别在什么情况下会更新？

另外，我们可以通过执行以下命令查看系统限制：

```
$ df -h  # 查看磁盘字节空间（数据块）
$ df -i  # 查看 Inode 节点总数与已用数量
```

【关键信息解读】

- 在设计文件系统时，必须权衡 `Inode` 表的大小。`Inode` 过多会浪费磁盘空间，过少则限制了文件总数。

【观察与思考】

- 如果我们需要在磁盘上创建大量的小文件（例如 1 字节的文件），哪种资源会先耗尽？

2.2 目录：一种特殊的文件

你可能会好奇：如果 Inode 里不存文件名，那文件名存在哪？

答案是：存储在目录*。在 Linux 中，目录也是一种文件，它的内容是一张表，记录了文件名和 Inode 编号的映射关系。每一行被称为一个 **目录项 (Dentry)**。

---

3. 实验任务

任务 A：创建链接并对比

目标：理解硬链接与软链接的区别。

步骤 1：创建链接

```
ln test.txt hard_link          # 创建硬链接
ln -s test.txt soft_link       # 创建软链接（符号链接）
ls -i test.txt hard_link soft_link
```

【关键信息解读】

- 硬链接实际上只是在目录里增加了一个指向原有 `Inode` 的映射
- 软链接是一个独立的新文件，其内容存储的是目标文件的路径。

【观察与思考】

- `hard_link` 的 `Inode` 编号与原文件一致吗？`soft_link` 呢？
- 查看 `ls -l` 的结果，注意硬链接的“链接计数”发生了什么变化？

步骤 2：删除实验

```
rm test.txt
cat hard_link    # 是否还能读取？
cat soft_link    # 是否还能读取？
```

【关键信息解读】

- 在内核中，Inode 包含一个“引用计数”。执行`rm` 实际上是减少该计数值。只有当一个 Inode 的链接计数归零且没有进程占用时，文件内容才会被真正从磁盘释放。

【观察与思考】

- 为什么删除原文件后，硬链接依然能读取内容，而软链接失效了？

任务 B：特权进阶（SUID 实验）

目标：掌握 SUID 的实际意义。

在 Linux 中，普通用户可以修改自己的密码。但密码存储在 `/etc/shadow` 中，该文件只有 root 权限可写。普通用户是如何实现"越权"写入的？答案就是 **SUID (Set User ID)**。

步骤 1：准备环境（需 root 权限）

```
# 1. 创建一个只有 root 能读的机密文件
sudo sh -c 'echo "Top Secret" > /root/secret.txt'
sudo chmod 600 /root/secret.txt

# 2. 普通用户尝试读取
cat /root/secret.txt  # 应该提示 Permission denied
```

步骤 2：编写观察程序 我们需要一个程序来读取这个文件。请编写 `my_read.c`：

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
    printf("Real UID: %d\n", getuid());        // 运行该程序的真实用户 ID
    printf("Effective UID: %d\n", geteuid()); // 内核检查权限时使用的有效 ID

    // 尝试读取机密文件
    system("cat /root/secret.txt");
    return 0;
}
```

步骤 3：赋予权限 编译并设置权限：

```
gcc my_read.c -o my_read
sudo chown root:root my_read  # 确保所有者是 root
sudo chmod u+s my_read        # 赋予 SUID 位
./my_read                     # 以普通用户身份运行
```

【关键信息解读】

- 当文件设置了 SUID 位，进程在执行时的"身份"会临时切换为文件的所有者

【观察与思考】

- 查看程序输出的 Real UID（当前用户）和 Effective UID（实际发挥作用的用户）。
- 为什么此时普通用户可以读出 /root/secret.txt？
- 如果你编写的 SUID 程序允许用户输入参数（如 system(input)），会带来什么风险？

4. 实验总结与后续预告

通过本次观察实验，我们掌握了以下核心概念：

- **文件名不等于文件**：文件名只是 Dentry 中的一个字符串，底层由 Inode 唯一标识。
- **硬链接的本质**：多个文件名指向同一个 Inode。
- **权限的动态性**：通过 SUID，内核允许受控的特权提升。
- **物理布局的限制**：文件系统不仅要管数据，还要管元数据。

下一阶段： 在接下来的实验中，我们将不再依赖 Linux 强大的 ext4 或 xfs，而是在我们自己的简易内核中实现一个文件系统。我们会简化掉权限控制和多级目录，通过直接操作模拟磁盘，亲自实践如何将一个 Inode 对应到具体的磁盘块。