02 文件系统 API 编程：实现目录克隆工具

1. 本节导读

在实验 01 中，我们观察了文件系统的静态结构。本实验我们将通过编写一个功能完整的目录复制工具 `my_cp`，掌握 Linux 内核提供的底层 I/O 接口。

你将亲自驱动内核接口完成文件的读写、元数据的迁移以及目录树的递归遍历。这不仅是编程练习，更是为了让你深刻理解：**操作系统是如何通过文件描述符（FD）管理字节流的，以及目录项（Dentry）是如何组织的。**

---

2. 课前基础知识储备：核心 API 手册

在编写程序之前，你需要掌握以下三类核心 API。请阅读其用法并观察示例。

2.1 文件流操作：open, read, write

Linux 采用"打开-读写-关闭"的模型。

```
#include <fcntl.h>
#include <unistd.h>

// 示例：将 src.txt 的内容复制到 dest.txt
int fd_in = open("src.txt", O_RDONLY);
int fd_out = open("dest.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);

char buf[1024];
ssize_t bytes;
while ((bytes = read(fd_in, buf, sizeof(buf))) > 0) {
    write(fd_out, buf, bytes);
}
close(fd_in);
close(fd_out);
```

【关键信息解读】

- `O_CREAT | O_TRUNC`：如果目标文件不存在则创建，存在则清空。
- `read` 的返回值：返回实际读取的字节数。如果返回 0 表示到达文件末尾（EOF）。
- `0644`：创建文件时的初始权限。

2.2 属性克隆：stat, chmod, chown

文件不仅有数据，还有权限和所有者属性，运行以下代码，尝试理解`stat`, `chmod`, `chown`命令。

```
#include <sys/stat.h>

struct stat st;
```

```
stat("source_file", &st); // 获取源文件属性

// 应用属性到目标文件
chmod("target_file", st.st_mode);           // 克隆权限模式
chown("target_file", st.st_uid, st.st_gid); // 克隆所有者与所属组
```

【关键信息解读】

- `st.st_mode` 中不仅包含 `755` 这样的权限，还包含了文件类型是目录还是普通文件。

【观察与思考】

- 为什么必须在文件写完后才调用 `chmod`？（提示：如果你先把目标文件设为只读，后续还能 `write` 吗？）

2.3 目录遍历：opendir, readdir

目录是一个包含多条"记录"的文件，每条记录是一个 `struct dirent`。

```
#include <dirent.h>

DIR *dir = opendir("my_dir");
struct dirent *entry;

while ((entry = readdir(dir)) != NULL) {
    printf("文件名: %s | 类型: %d\n", entry->d_name, entry->d_type);
}
closedir(dir);
```

【关键信息解读】

- `entry->d_name`：文件名。
- `entry->d_type`：文件类型。`DT_DIR` 表示目录，`DT_REG` 表示常规文件。
- 注意：每个目录下都有 `.` 和 `..`，递归时必须跳过它们。

---

3. 实验任务：编写智能目录复制工具 `my_cp`

目标：实现 `./my_cp <source_dir> <dest_dir>`，要求递归复制所有子目录及文件，并保持属性一致。

步骤 1：单文件复制函数

请先实现一个辅助函数，负责处理最基础的文件读写。

```
void copy_single_file(const char *src_path, const char *dest_path) {
    // 1. open 源文件 (O_RDONLY)
    // 2. open 目标文件 (O_WRONLY | O_CREAT | O_TRUNC)
    // 3. while(read) { write } 循环搬运数据
    // 4. 使用 stat 获取源文件属性
```

```
    // 5. 使用 chmod/chown 修改目标文件属性
    // 6. close 文件描述符
}
```

步骤 2：递归逻辑实现

这是实验的核心。你需要编写一个递归函数 `copy_directory`，相关流程可参考以下逻辑流：

- 创建目标目录：使用 `mkdir(dest_dir, mode)`。
- 打开源目录：使用 `opendir`。
- 循环读取目录项：
  - 使用 `readdir` 获取每一个 `entry`。
  - 判断：如果是 `.` 或 `..`，使用 `continue` 跳过。
  - 构造完整路径：使用 `sprintf(full_path, "%s/%s", dir_name, entry->d_name)`。
  - 分流处理：
    - 如果 `entry->d_type == DT_DIR`：递归调用自身。
    - 如果 `entry->d_type == DT_REG`：调用 `copy_single_file`。
    - 如果 `entry->d_type == DT_LNK`：打印 `[Link Ignore]` 并跳过。
- 关闭目录：`closedir`。

步骤 3：测试验证

编写完成后，请进行以下测试：

```
$ mkdir -p test_dir/inner
$ echo "hello" > test_dir/a.txt
$ chmod 700 test_dir/a.txt  # 设为仅所有者可读写执行
$ ./my_cp test_dir test_dest
$ ls -lR test_dest  # 验证权限和目录结构是否与 test_dir 一致
```

---

4. 实验进阶挑战

如果你已经完成了基础功能，请思考并尝试解决以下问题：

- 在拼接路径时，如果层级极深，`char path[1024]` 可能会溢出。如何安全地处理（提示：参考 `snprintf`）？
- 在 `read/write` 循环中，`buf` 的大小（如 1KB vs 4MB）对复制速度有什么影响？为什么？

---

5. 实验总结与后续预告

通过本次编程实验，你掌握了以下核心技能：

- **系统调用实践**：理解了内核如何通过 FD 维护进程与文件的状态。
- **递归文件系统操作**：学会了如何通过 `readdir` 配合递归算法手动“爬取”目录树。
- **元数据意识**：意识到文件不仅仅是数据，权限和所有权是操作系统安全体系的根基。

下一阶段预告： 目前我们依然在 Linux 提供的 API 之上开发。你是否好奇：当你调用 `write` 时，内核到底是如何把数据写到磁盘扇区的？在接下来的实验中，我们将深入**内核态**，在模拟磁盘上构建自己的 `Inode` 分配算法和块位图（Bitmap）。我们将从 API 的调用者，变成 API 的实现者。