Lab5-1-Extra-SSD

题目背景

以下部分仅用于帮助理解 SSD 的相关概念，具体的实现要求请以“题目描述”及后续说明为准。

本题要求你模拟一个简化版本的 SSD （固态硬盘）驱动。

SSD （固态硬盘）的特点是对于每个存储了数据的闪存块，必须先擦除才能写入。同时闪存块是有擦写次数上限的，这个擦写次数上限就是闪存的寿命。为了平衡闪存各个物理块的擦写次数， SSD 引入了磨损平衡机制，保证闪存中各个块的擦写次数大体上一致，避免某些块被过度使用。为了简化问题，我们规定在本题中， SSD 的读、写、擦除都是以块为单位，每块的大小都是 $512Bytes$ ；且对于每个物理块，必须**先擦除**，才能写入数据。

对于机械硬盘来说，其读写都是在具体的扇区上操作，无需擦除。当前的文件系统基本上是针对机械硬盘来开发的，因此为了屏蔽固态硬盘和机械硬盘的这个差异，固态硬盘中引入了闪存转换层（FTL），把闪存的操作方式转换为传统硬盘以 $512Bytes$ 扇区为单位的操作方式。

我们定义如下几个概念：

- **逻辑块号**：文件系统读写的逻辑硬盘块编号，在物理上不一定连续，每个块的大小为 $512Bytes$
- **物理块号**： SSD 实际读写的闪存的物理编号，在物理上连续，每个块的大小都是 $512Bytes$
- **闪存映射表**：记录逻辑块号对应物理块号信息的表
- **物理块位图**：记录各个物理块的状态信息

题目描述

在本题中，评测程序通过逻辑块号访问 SSD ，每个逻辑块大小为 $512Bytes$ 。实际读取逻辑块时，这个逻辑块号会通过闪存映射表转换为物理块号。同时需要在物理块位图维护每个物理块的状态，物理块有不可写、可写两种状态。初始时所有块均为可写状态，当一个可写的块被写入后，其会变为不可写状态，此时物理块只读不允许写入，只有通过擦除才能重新变为可写状态。

> 注意区分这里的 “物理块” 和块缓存中的 “磁盘块” 的区别，这里的 “物理块” 实际上指的是 IDE 磁盘的扇区

- 首先，你需要在代码中定义如下数据结构：
  - **闪存映射表**：记录逻辑块号对应物理块号信息的表
  - **物理块位图**：记录各个物理块是否可写。每次向可写的物理块中写入数据后，都需要标记物理块为不可写
  - **物理块累计擦除次数表**：记录各个物理块的累计擦除次数
- 其次，我们定义对 SSD 的如下操作：
  - **初始化硬盘**：清空闪存映射表，在物理块位图中将 SSD 的所有物理块初始化为可写状态，SSD 各物理块的累计擦写次数清 0。（保证只在开始时调用一次）
  - **读取逻辑块**：查询闪存映射表，找到逻辑块号对应的表项。如果为空，则返回 -1 ； 若不为空，就去读取对应的物理块，返回物理块内的数据，并返回 0。
  - **写入逻辑块**：查询闪存映射表，找到逻辑块号对应的表项。若为空（初次写），则分配一个可写的物理块，在闪存映射表中填入此映射关系，将数据写入该物理块，并标记此物理块为不可写；若不为空（覆盖写），则擦除原来的物理块并清除该逻辑块号的映射，并按相同方法分配一个物理块向其写入数据。

- **擦除逻辑块**：查询闪存映射表，找到逻辑块号对应的表项。若为空，不做处理；若不为空，则将对应的物理块擦除（擦除表示将物理块数据全部清 0，下同）并清除此映射关系。
    - 同时，你还需要按如下要求实现一个简易的**磨损平衡机制**：

> 1. 维护每个物理块的累计擦除次数：初始时所有块的累计擦除次数都为 0，之后每擦除一次，其累计擦除次数加 1。
> 2. 在分配新的物理块时，始终选择**可写**的物理块当中**累计擦除次数最小**的那一个（如有多个，则选择**物理块号最小**的那一个）。
> 3. **静态磨损平衡**：上面的机制会有不合理的情形。比如说一个逻辑块初始时只写了一次，之后就不再更改了，那么其所对应的物理块擦除次数始终为0，导致这个磁盘块的擦写次数没有被充分利用。
>
>   因此，我们规定如果找到的**具有最小擦除次数的可写**物理块（记为块A）擦除次数**大于等于5**，表明现在空闲块已经进入**高负荷状态**。此时需要：
>
>   - 检测所有不可写的物理块（保证此时存在不可写的块），找到其中累计擦除次数最少的块（如有多个，则选择物理块号最小的那一个）（记为块B）
>   - 将块B的内容写入到块A中
>   - 将块A标记为不可写
>   - 更新物理块B原先对应的逻辑块的映射为物理块A
>   - 擦除块B的数据（同时将B的累积擦除次数加1，并标记B为可写）
>   - 将块B作为可写的空闲块分配出来
>
>   这样，原来B中不被频繁修改的数据就在A中存放了，有利于磨损平衡。

题目要求

实验提供代码

请将以下函数的声明粘贴到 $fs/serv.h$ 中：

```
void ssd_init();
int ssd_read(u_int logic_no, void *dst);
void ssd_write(u_int logic_no, void *src);
void ssd_erase(u_int logic_no);
```

函数实现

在本题中，你需要使用 IDE 0号硬盘（`diskno` = 0）的第$0-31$个扇区来模拟 SSD 的第 $0-31$个物理块，之后根据 *题目描述部分* 的描述，在 $fs/ide.c$ 中实现下列函数：

- 初始化硬盘：

  `void ssd_init();`

    - 功能：初始化闪存映射表、物理块位图、物理块累计擦除次数表。
- 读取逻辑块：

  `int ssd_read(u_int logic_no, void *dst);`

    - `logic_no`：逻辑块号
    - `dst`：硬盘数据要加载到的内存地址
    - 功能：读取逻辑块 `logic_no` 的数据到 `dst`，成功返回 0，失败返回 -1
- 写入逻辑块：

`void ssd_write(u_int logic_no, void *src);`

- `logic_no`：逻辑块号
- `src`: 要写入到硬盘中的数据的起始内存地址
- 功能：向逻辑块号对应的硬盘块写入以 `src` 为起始地址的 512字节 数据

- 擦除逻辑块：

`void ssd_erase(u_int logic_no);`

- `logic_no`：逻辑块号
- 功能：擦除逻辑块 `logic_no` 上的数据，并取消 `logic_no` 到物理块号的映射关系

评测时将会调用上面的接口做一系列操作，验证操作执行结果是否正确，并会在操作的任何阶段读取硬盘数据以检查实现的正确与否。

为了实现上面的函数，你可能还需要自行实现以下两个函数，但我们不对函数的定义做具体要求：

> - 擦除物理块：将全 0 的数据写入到物理块中，物理块累计擦除次数加一，并置为可写状态
> - 分配新物理块：按“题目描述”部分的方法分配，需要用到物理块位图和物理块累计擦除次数两个信息

注意事项

- 物理块和逻辑块的编号范围均为 $0-31$，保证在调用 `ssd_read`、`ssd_write`、`ssd_erase` 时传入的逻辑块号在此范围内。
- 你需要调用 `ide.c` 中的 `ide_read` 和 `ide_write` 两个函数来完成对物理块的读写，其中 `diskno` 参数始终设为 0，`secno` 对应要读或写的物理块的编号。
- 在覆盖写磁盘块时，你应该先擦除原磁盘块，后分配空闲磁盘块，避免出现无空闲磁盘块可分配的情况。
- 你需要保证在每次擦除物理块上的数据时，都把其累计擦除次数加一。
- 闪存映射表、物理块位图、累计擦除次数表这些数据结构都只需要在内存中实现，无需存储到硬盘中。
- 你需要自行规定闪存映射表项为空的条件，不要用物理块编号为 0 表示空，避免与逻辑块映射到物理块 0 的情况冲突。（如可用 $0xFFFFFFFF$ 表示映射为空）

测试点和分数说明如下：

测试点序号	评测内容	分数
1	样例	10
2	保证每个逻辑块最多写一次（不覆写）	25
3	保证所有物理块擦写次数均小于5，不涉及静态磨损平衡测试	25
4	保证所有物理块擦写次数均小于5，不涉及静态磨损平衡测试	20
5	强测（300次擦写），涉及静态磨损平衡测试	20

> 上面的“擦写”指的是物理块被“擦除-写入”的次数。物理块每被擦除、写入一次，算一次擦写。

Lab5-2-Extra-Symbolic link

题目背景 & 预备知识

在 Linux 系统中，符号链接（Symbolic link）是一种特殊的文件类型，用来表示指向另一个文件路径的“指针”，类似于 Windows 系统中的快捷方式。如果你对符号链接的知识有所了解，可以跳过这一节的内容。

在磁盘上，符号链接的物理内容为其指向的文件的路径。符号链接本身可能会指向另一个符号链接。当我们使用 `open()` 函数打开一个符号链接的时候，实际上打开的是其最终指向的文件，返回最终指向的文件的文件描述符。

题目描述 & 题目要求

我们需要拓展现有的文件系统，使其能够打开符号链接最终指向的文件。

符号链接指向的路径（即符号链接文件的内容）满足下面几个要求：

- 符号链接指向的路径一定是**绝对路径**（mos 目前也只支持绝对路径的解析）。
- 符号链接指向的路径对应的文件要么是普通文件，要么是最终指向普通文件的符号链接。

  这蕴含了以下两点：

  - 符号链接不会指向一个目录，因此一个路径的各部分中的目录都不会是符号链接；
  - 符号链接可能指向的是一个符号链接，这个符号链接可能指向的也是一个符号链接，以此类推，但最终一定会指向一个普通文件。

评测点满足要求：

- 符号链接最终指向的普通文件一定存在。不要求进行错误处理。
- 链接关系不会产生环，例如，不会出现 a 指向 b，且 b 指向 a 的情况。
- 文件名的长度和路径长度满足 MOS 的要求，即文件名长度不会超过 $MAXNAMELEN$，路径长度不会超过 $MAXPATHLEN$（这两个宏定义在 user/include/fs.h 中）。

我们需要实现的内容

- 在 user/include/fs.h 中定义新的文件类型，即 `#define FTYPE_LNK 2`（不要使用其他数字作为链接文件的枚举编号）。
- 修改 tools/fsformat.c，使得 fsformat 工具在写入文件时根据传入的各个文件的类型来决定写入磁盘镜像的文件的类型（`FTYPE_LNK`、`FTYPE_DIR` 或 `FTYPE_REG`）。

  以下是一种修改的思路：

  - 将 `main()` 函数中调用的 `stat()` 函数改为 `lstat()`（参数不用改变）。`stat()` 或 `lstat()` 的作用都是将一个文件的相关信息储存在 `statbuf` 结构体中。它们的区别在于：`stat()` 函数会跟踪链接，会解析链接最终指向的文件；而 `lstat()` 不会跟踪链接，直接解析链接文件本身。而我们需要读取链接文件本身的信息，所以需要使用 `lstat()` 函数替换 `stat()` 函数（直接改函数名即可，这两个函数都是 Linux 的库函数）。
  - 可以仿照 `write_file()` 函数编写 `write_symlink()` 函数，实现向磁盘镜像写入符号链接文件的功能。你可以调用 Linux 库函数 `int readlink(char *pathname, char *buf, int bufsiz)` 来读取一个链接文件指向的目标路径（这个函数将路径 `pathname` 处的符号链接指向的目标路径写入 `buf` 指向的缓冲区，最多写入 `bufsiz` 个字节，返回值是写入的字节数量。详细说明可以在开发机中使用 `man 2 readlink` 命令查阅）。
    - 一种可能的实现框架如下所示：

      ```
      void write_symlink(struct File *dirf, const char *path) {
      ```

```
    struct File *target = create_file(dirf);
    // Your code here: 使用 readlink() 函数读取链接文件指向的路径，将其写入到下一个可用的磁盘块

    const char *fname = strrchr(path, '/');
    if (fname) {
        fname++;
    } else {
        fname = path;
    }
    // Your code here: 设置链接文件的文件名、大小（指向路径的字符串的长度）、类型属性

    save_block_link(target, 0, next_block(BLOCK_DATA));
}
```

  - 修改 `main()` 函数，在 `if else` 结构中增加一个分支，调用你编写的 `write_symlink()` 函数，使其不仅支持目录和普通文件的读取，还可以支持符号链接文件的读取。可以使用 `S_ISLNK(stat_buf.stmode)` 判断命令行参数对应的文件是否为符号链接。
  - 修改 `write_directory()` 函数，在 `if else` 结构中增加一个分支，调用你编写的 `write_symlink()` 函数，使其不仅支持目录和普通文件的读取，还可以支持符号链接文件的读取。结构体 `dirent` 的成员变量 `d_type` 可能会用到以下取值：`DT_DIR`（目录）、`DT_LNK`（符号链接）、`DT_REG`（普通文件）。
- 修改文件系统的实现，使其满足：用户程序使用 `open()` 函数打开一个符号链接文件的时候，实际上打开的是其最终指向的文件，返回最终指向的文件的文件描述符。被打开的文件可以正常地读写，就像直接打开了最终指向的文件一样。

测试点和分数说明如下：

测试点序号	评测内容	分数
1	样例	10
2	链接文件指向的文件一定是普通文件，并对打开文件进行读写测试	40
3	存在多级链接，可能链接到任意路径的文件或链接，并对打开文件进行读写测试	30
4	存在多级链接，可能链接到任意路径的文件或链接，并对打开文件进行读写测试	20