OS LAB 5 文件系统

个人信息

姓名：郑凯琳

学号：205220025

邮箱：205220025@smail.nju.edu.cn

1. 实验进度

完成了所有内容，但不成功。

2. 实验过程

（1）完善系统调用处理（内核支持文件读写）

代码修改位置：kernel/kernel/irqHandle.c

syscallOpen

功能：文件的打开

实现细节：

1. 变量初始化

初始化了一些必要的变量，包括文件路径、父节点和目标节点的索引、基地址等。

2. 读取 Inode

调用 readInode 函数尝试读取目标文件的 Inode。如果文件存在，则根据标志位进行相应的处理；否则，尝试创建新文件。

处理已存在的文件

- 如果目标文件存在，首先检查 O_DIRECTORY 标志。如果设置了该标志且目标文件不是目录，则返回错误；反之亦然。
- 接下来，检查文件表中是否有可用的条目，如果有，则将文件状态设为使用中，并返回文件描述符；否则返回错误。

处理新文件创建

- 如果目标文件不存在且 O_CREATE 未设置，则返回错误。
- 如果 O_DIRECTORY 未设置，则解析路径，找到父目录，并创建常规文件。
- 如果 O_DIRECTORY 设置了，首先检查路径的末尾是否有 '/'，并删除它。然后找到父目录，并创建目录文件。
- 如果创建 inode 失败，则返回错误。否则，将新的 inode 添加到文件表中。

syscallWrite

功能：对标准输出和文件写操作的处理

实现细节：

文件写操作处理

- 检查文件描述符是否在文件范围内（即 fd >= MAX_DEV_NUM 且 fd < MAX_DEV_NUM + MAX_FILE_NUM）。
- 如果是文件描述符，则通过 fd -= MAX_DEV_NUM 将文件描述符转换为文件数组索引。
- 检查文件数组中的文件状态是否为活动（file[fd].state == 1）。
- 如果文件状态为活动，调用 syscallWriteFile 进行文件写操作。
- 否则，设置当前进程的 eax 寄存器为 -1，表示操作失败。

错误处理

- 如果文件描述符不在有效范围内，直接设置当前进程的 eax 寄存器为 -1，表示操作失败。

syscallRead

功能：对标准输出和文件读操作的处理

实现细节：

文件读操作处理

- 检查文件描述符是否在文件范围内（即 fd >= MAX_DEV_NUM 且 fd < MAX_DEV_NUM + MAX_FILE_NUM）。
- 如果是文件描述符，则通过 fd -= MAX_DEV_NUM 将文件描述符转换为文件数组索引。
- 检查文件数组中的文件状态是否为活动（file[fd].state == 1）。
- 如果文件状态为活动，调用 syscallReadFile 进行文件读操作。
- 否则，设置当前进程的 eax 寄存器为 -1，表示操作失败。

错误处理

- 如果文件描述符不在有效范围内，直接设置当前进程的 eax 寄存器为 -1，表示操作失败。

syscallWriteFile

实现了有效的文件写入逻辑，通过合理的块管理和缓冲区控制，保证了文件数据的正确写入和 inode 信息的更新。在分配新块、读取块数据、写入数据和更新 inode 信息等方面，通过逐步处理，确保了函数的稳定性和正确性。

（1）权限检查和初始化：

- 首先检查文件的写入权限，如果未设置，则直接返回错误。
- 初始化必要的变量，包括用户进程的基地址、用户进程缓冲区、待写入的字节数、缓冲区等。

（2）写入文件过程：【实现部分】

- 如果待写入的字节数小于等于 0，则直接返回错误。
- 进入循环，通过块索引和偏移量确定当前要写入的位置。
- 如果当前块索引超过了 inode 分配的块数，则分配新的块。
- 将当前块的数据读入缓冲区，计算当前块剩余可写入的空间。
- 如果当前块剩余空间足够，将剩余的所有字节写入缓冲区；否则尽可能多地写入当前块。
- 将写入的数据缓冲区写回磁盘。
- 更新剩余待写入字节数、当前块索引、当前块内偏移量、缓冲区索引等变量。

（3）写入最后一个块和更新 inode 信息：【实现部分】

- 循环结束后，将剩余的数据写入最后一个块，并更新 inode 的大小。
- 将更新后的 inode 信息写回磁盘。

（4）返回结果：

设置 eax 寄存器为成功写入的字节数，并更新文件的偏移量。

syscallReadFile

用于从文件系统中读取数据到用户进程的缓冲区中。在进行读取操作之前，它会检查文件描述符的权限，确保文件已经打开且具有读权限。然后，它根据文件的偏移量和用户指定的最大读取字节数，从磁盘加载数据块并将数据复制到用户进程的缓冲区中。

（1）权限检查：

首先，检查文件描述符对应的文件是否设置了读权限。如果没有设置读权限，函数将返回 -1，表示读取操作失败。

（2）变量初始化：

初始化变量 i、j，分别用于缓冲区中的字节偏移和索引，baseAddr 用于计算用户进程的基地址，str 是用户进程的缓冲区指针，size 是用户请求的最大读取字节数，buffer 是用于存储数据块的缓冲区，quotient 和 remainder 是文件偏移量的商和余数。

（3）读取文件元数据：【实现部分】

使用 diskRead 函数读取文件的 inode 结构。这个结构包含了文件的大小和分配的块数等信息。

（4）读取数据块：【实现部分】

- 进入循环，循环条件是还有剩余的字节需要读取。
- 根据文件偏移量计算当前块的索引 currentBlockIndex 和块内的起始偏移量 startOffset。
- 调用 readBlock 函数将当前块的数据读入缓冲区 buffer 中。

（5）处理当前块数据：【实现部分】

- 计算当前块剩余可读取的空间 readBlockSize。
- 如果剩余数据不足一个块，则只读取剩余的数据，并更新读取的字节数 size。
- 如果当前块剩余空间足够，将剩余的所有字节读入缓冲区 str 中。

（6）更新文件偏移量和返回值：【实现部分】

- 更新文件偏移量 file[sf->ecx - MAX_DEV_NUM].offset。
- 设置 pcb[current].regs.eax 返回成功读取的字节数 size。

（7）函数返回：

函数返回，读取操作完成。

syscallLseek

功能：用于在文件系统中定位文件偏移量。

根据用户给定的偏移量和相对位置 whence，它更新文件描述符对应的文件偏移量，并返回新的偏移量或者错误码。

实现步骤：

（1）参数解析：

- 获取函数参数 sf 中的文件描述符 i 和偏移量 offset。
- 初始化一个 Inode 结构体 inode 用于读取文件的元数据。

（2）错误检查：

- 检查文件描述符 i 是否在有效范围内，如果不在范围内或者对应的文件未打开（state 状态为 0），则返回错误码 -1。

（3）读取文件元数据：

- 使用 diskRead 函数读取文件的 inode 结构体，以获取文件的大小和其他元数据信息。

（4）根据 whence 进行定位：

根据 sf->ebx 的值进行 switch 分支判断，确定定位方式：

- SEEK_SET： 设置文件偏移量为给定的 offset 值。
- SEEK_CUR： 在当前偏移量的基础上增加 offset 值。
- SEEK_END： 设置文件偏移量为文件末尾加上 offset 值。

在每种情况下，更新文件描述符对应文件的偏移量 file[i - MAX_DEV_NUM].offset。

（5）返回新的偏移量：

将新的偏移量存入 sf->eax 寄存器中，以便返回给用户进程。

（6）函数结束：

函数执行完毕，返回定位后的偏移量或者错误码。

syscallClose

功能：文件的关闭

实现步骤：

（1）参数解析：

从函数参数 sf 中获取文件描述符 i。

（2）错误检查：

- 检查文件描述符 i 是否在有效范围内：
- 如果 i 小于 MAX_DEV_NUM 或者大于等于 MAX_DEV_NUM + MAX_FILE_NUM，则说明是设备文件，不能被关闭，返回错误码 -1。
- 如果 file[i - MAX_DEV_NUM].state 等于 0，表示该文件描述符对应的文件当前未使用，也返回错误码 -1。

（3）关闭文件或设备：

- 将 file[i - MAX_DEV_NUM].state 置为 0，表示文件描述符对应的文件不再使用。
- 将 file[i - MAX_DEV_NUM].inodeOffset、file[i - MAX_DEV_NUM].offset 和 file[i - MAX_DEV_NUM].flags 全部清零，以释放文件相关的资源和状态信息。

（4）设置返回值：

将 pcb[current].regs.eax 设置为 0，表示成功关闭文件或设备。

（5）函数结束：

返回函数执行结果。

syscallRemove

功能：文件的删除

实现细节：（TODO 部分）

（1）检查文件是否存在

- 调用 readInode 函数读取文件路径 str 对应的目标节点信息和偏移量 destInodeOffset。
- 如果文件不存在 (ret != 0)，设置错误码并返回。

（2）处理路径末尾的 '/'

- 如果路径以 '/' 结尾，则去掉末尾的 '/'。

（3）获取父节点路径

- 找到路径中最后一个 '/' 的位置，确定父节点路径 parentPath。
- 如果路径中不存在 '/' 或者最后一个 '/' 位于开头，则返回错误码。

（4）读取父节点信息

- 调用 readInode 函数读取父节点 parentPath 的信息和偏移量 fatherInodeOffset。
- 如果读取失败，设置错误码并返回。

（5）释放目标节点

- 根据 destInode 的类型（普通文件或目录），调用 freeInode 函数释放目标节点及其占用的磁盘空间。
- 如果释放失败，设置错误码并返回。

syscallStat

功能：完成了对文件或目录信息获取并返回的操作

实现细节：

（1）参数解析和初始化：

- sf->ecx 包含文件路径的地址，需要加上基地址 baseAddr 才能得到用户进程中的实际地址。
- sf->ebx 包含 struct stat 结构体的地址，也需要加上基地址 baseAddr。

（2）调用 stat 函数：

- 调用 stat(filename, &buffer) 函数获取文件或目录的信息，存储在 buffer 结构体中。

（3）处理获取到的文件信息：

- 将获取到的文件信息填充到 struct stat 结构体中。

（4）将 struct stat 结构体写回用户空间：

- 使用 memcpy 将 buffer 结构体的内容复制到用户空间中。

（5）设置返回值：

- 如果一切正常，设置 pcb[current].regs.eax 为 0，表示成功。

（2）用户程序

代码修改位置：app/main.c

ls

功能：一个简单的目录列表功能 (ls)，通过打开目录文件，读取目录内容到缓冲区 buffer，然后遍历缓冲区，输出有效的目录条目名称。

实现部分：

1. 转换缓冲区为 DirEntry 结构体数组。
2. 遍历当前缓冲区中的所有 DirEntry
- 循环遍历当前缓冲区中的所有 DirEntry 结构体。ret / sizeof(DirEntry) 计算出当前缓冲区中 DirEntry 的数量。
- 对于每一个 DirEntry，检查其 inode 是否不为 0，从而判断该条目是否有效。
  如果有效，则输出其名称 dirEntry[i].name。
3. 继续读取下一段数据到缓冲区，直到读取到末尾。

cat

功能：读取指定文件的内容并将其输出到标准输出。

实现部分：

1. 使用 read 函数从文件描述符 fd 中读取数据到缓冲区 buffer 中，每次最多读取 sizeof(buffer) 字节。
2. 循环遍历缓冲区中的每个字符，将每个字符输出到标准输出。
3. 每次循环完成后，通过 read(fd, buffer, sizeof(buffer)) 继续读取文件的下一段内容。

find

通过递归方式在指定目录 dir 中查找文件 file。

1. 首先获取 dir 目录下的文件状态信息，并根据文件类型选择不同的处理方式。
2. 如果是文件 (TYPE_FILE)，则调用 match 函数进行文件名匹配。

TODO 部分：

*

3. 如果是目录 (TYPE_DIRECTORY)，则打开目录并循环读取目录内容，处理每个目录条目：
- 将目录路径和目录条目名拼接为新的路径 name_buffer。
- 递归调用 find 函数，继续在子目录中查找文件。

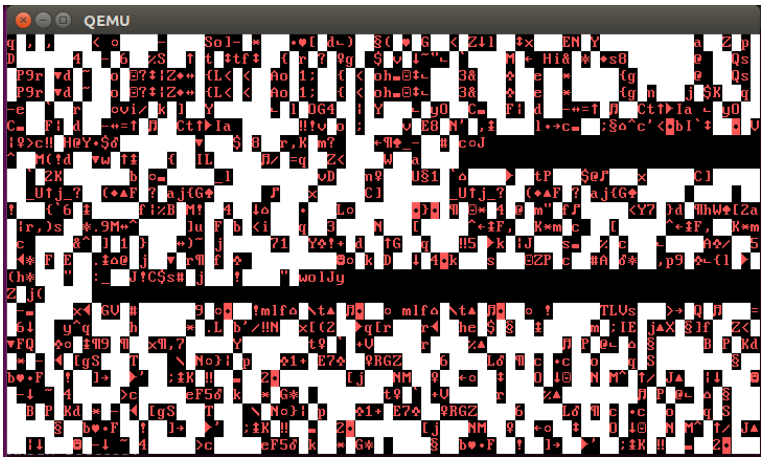
*

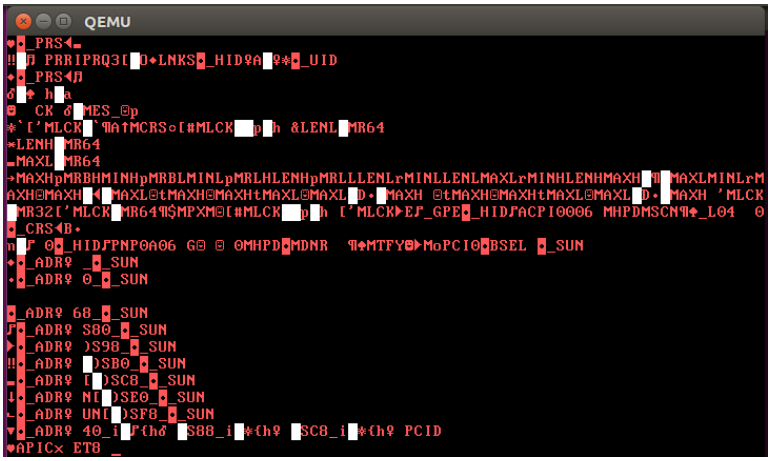
4. 其他文件类型暂不处理。

find 函数使用了系统调用 open、read、stat 等，以及自定义的字符串操作函数 stringLen、stringCpy 来操作文件和目录。

3. 实验结果

```
FORMAT success.
8191 inodes and 3070 data blocks available.
mkdir /boot
MKDIR success.
8190 inodes and 3069 data blocks available.
cp uMain.elf /boot/initrd
CP success.
8189 inodes and 3053 data blocks available.
mkdir /dev
MKDIR success.
8188 inodes and 3053 data blocks available.
touch /dev/stdin
TOUCH success.
8187 inodes and 3052 data blocks available.
touch /dev/stdout
TOUCH success.
8186 inodes and 3052 data blocks available.
mkdir /usr
MKDIR success.
8185 inodes and 3052 data blocks available.
mkdir /data/
MKDIR success.
8184 inodes and 3052 data blocks available.
touch /data/test.txt
TOUCH success.
8183 inodes and 3051 data blocks available.
mkdir /data/dir1
MKDIR success.
8182 inodes and 3051 data blocks available.
mkdir /data/dir1/dir11
MKDIR success.
8181 inodes and 3050 data blocks available.
touch /data/dir1/dir11/test.txt
TOUCH success.
8180 inodes and 3049 data blocks available.
mkdir /data/dir2
MKDIR success.
8179 inodes and 3049 data blocks available.
touch /data/dir2/test.txt
TOUCH success.
8178 inodes and 3048 data blocks available.
```




QEMU


QEMU