CSCI 5103 Project 4 Writeup

Anlei Chen & Shuo Jiang

1. Design:

The design mostly follows the structure described in Section 4: File System Interface. In addition to the required family system functions, five additional helper functions were added to organize and simplify the code. `inode_load` and `inode_save` load and save a specific inumber to and from disk. `get_free_block` scans the free map and returns the first available data block. `disk_bzero` zeros out the specified disk block. `mount_check` verifies whether the file system has been mounted before executing any file system function.

In addition, our implementation uses nested for loops to iterate through inode blocks in order to reduce unnecessary reads and writes. Although not required, we aimed to minimize disk access to improve performance. This optimization was applied mainly in the `fs_debug` and `fs_mount` functions. Run `make debug` to enable debug statements.

2. Testing:

Testing was conducted using `simplefs` with various disk image sizes. Valgrind was used throughout the entire testing process to ensure proper memory management. Both common and edge cases were tested and cross-checked against the provided `simplefs-solution` to confirm expected behavior. One notable difference is that, in our implementation, all functions (except `mount` and `format`) will fail if the disk is not mounted. A brief overview of testing procedures for each function are provided below:

fs_debug:

- Checked extensively against the solution using the provided image files.

fs_format:

- Check if the superblock has the correct information.
- Check that all inodes have been properly cleared.

fs_mount:

- No memory leaks occur when mounting multiple times.
- The file system and the corresponding free map are correctly initialized.

fs_unmount:

- If no file system has been mounted, no action is taken, and an error is returned.
- Properly frees memory and cleanly unmounts the existing file system.

fs_create:

- If all inode entries across the inode block are already in use, an error is returned.
- If inodes are available, the lowest available inumber is returned.
- The freemap is properly updated if a new inode block is allocated.
- Example commands:

```
{
   echo "format"
   echo "mount"
   for i in {1..129}; do
       echo "create"
   done
} | ./simplefs image.5 5
Output: create failed!
```

fs_delete:

- If the specified inode is invalid, no action is taken, and an error is returned.
- Only the specified inode is deleted, and no other disk blocks are affected.
- All direct and indirect pointers are properly freed.
- The freemap is properly updated when deleting large or sparse files.
- Example commands:

```
{
   echo "format"
   echo "mount"
   for i in {1..3}; do
       echo "create"
   done
   echo "delete 2"
   echo "delete 2"
} | ./simplefs image.5 5
Output: delete failed!
```

fs_getsize:

- If the specified inode is invalid, no action is taken, and an error is returned.
- If the file size is zero, but the inode is valid, 0 is returned.
- Example commands:

```
{
   echo "format"
   echo "mount"
   for i in {1..3}; do
       echo "create"
   done
   echo "getsize 0"
} | ./simplefs image.5 5
Output: inode 0 has size 0
```

fs_read:

- If the inode is invalid, no action is taken, and 0 is returned.
- If the offset is greater than the file size, no action is taken, and 0 is returned.
- If the length plus offset is greater than the file size, only bytes up to the end of the file are read.
- Reading from both direct pointers and indirect pointers are properly handled.
- Switching from using direct pointers to indirect pointers is properly handled.

- The return value contains the actual number of bytes read regardless of the length argument.
- File content is not affected or modified after a read.
- Example commands:

```
{
  echo "format"
  echo "mount"
  echo "create"
  echo "copyin test.file 0"
  echo "cat 1"
} | ./simplefs image.5 5
No output: no action is performed
```

fs_write:

- If the inode is invalid, no action is taken, and 0 is returned.
- If the resulting file size is greater than the max size, the data leftover data file is discarded.
- If data is written within the current file size, the file size will not increase.
- If data is written beyond the current file size, new data blocks are allocated as needed.
- If no free blocks are available, only the amount of bytes that can be written will be written.
- Writing to both direct and indirect pointers are properly handled.
- Switching from using direct pointers to indirect pointers is properly handled.
- Example commands of partial write:

```
{
   echo "format"
   echo "mount"
   echo "create"
   echo "copyin temp13k.txt 0"

} | ./simplefs image.5 5
Output: fs_write only wrote 12288 bytes, not 13000 bytes
```

- Example commands when there are no more free blocks:

```
{
  echo "format"
  echo "mount"
  echo "create"    # inode 0
  echo "create"    # inode 1
  echo "create"    # inode 2
  echo "create"    # inode 3
  echo "copyin test.file 0"
  echo "copyin test.file 1"
  echo "copyin test.file 2"
  echo "copyin test.file 3"
} | ./simplefs image.5 5
Output: fs_write only wrote 0 bytes, not 20 bytes.
```