PROJECT 4A:
FILE SYSTEM – READING

Minimum Requirements: 12 public tests

TEST DISTRIBUTION

Public tests – 16 tests | 77 points

Release tests – 0 tests | 0 points

Secret tests – 0 tests | 0 points

Yes, no secrets, you have all the test code to debug.

WHAT'S EXPECTED

- The purpose of this project is to add a writable filesystem, GFS3, to GeekOS.
- Previous projects not required: can start with fresh build!
- Ensure Makefile.linux ties the second ide disk to a valid gfs3 image.
- `QEMU ?= $(QEMU_BIN) -hdb <image name> -smp 2 -icount 4 -rtc clock=vm -rtc base=2014-01-01T00:00:00 -device isa-debug-exit,iobase=0x501  -m $(MEM) -debugcon file:output.log -serial stdio`

FUNCTIONS TO IMPLEMENT

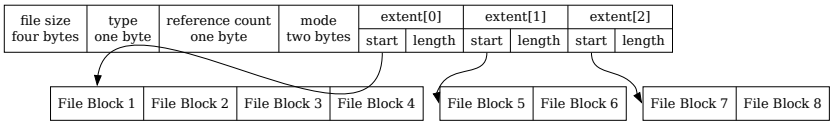
- `static int GFS3_Mount( struct Mount_Point *mountPoint)` `//can test code after implementing this`
- `static int GFS3_FStat( struct File *file, struct VFS_File_Stat *stat)`
- `static int GFS3_Read( struct File *file, void *buf, ulong_t numBytes)` `// test 35 after this`
- `static int GFS3_Seek( struct File *file, ulong_t pos)`
- `static int GFS3_Close( struct File *file)`
- `static int GFS3_FStat_Directory( struct File *dir, struct VFS_File_Stat *stat)`
- `static int GFS3_Close_Directory( struct File *dir)`
- `static int GFS3_Read_Entry( struct File *dir, struct VFS_Dir_Entry *entry)`
- `static int GFS3_Open( struct Mount_Point *mountPoint, const char *path, int mode, struct File **pFile)` `// test 33`
- `static int GFS3_Open_Directory( struct Mount_Point *mountPoint, const char *path, struct File **pDir)`
- `static int GFS3_Stat( struct Mount_Point *mountPoint, const char *path, struct VFS_File_Stat *stat)`

SYSTEM CALLS TO MODIFY

Refer to the spec for details. You can go through Sys_Close, Sys_Open, Sys_Read, Sys_Write to see how they are implemented. There are some structures and functions you might want to refer to:

- Sys_Mount: call Mount(vfs.c) to mount a file system
- Sys_Open_Directory: next_descriptor, Open_Directory (vfs.c), add_file_to_descriptor_table
- Sys_ReadEntry: VFS_Dir_Entry, VFS_Dir_Entry
- Sys_Stat: VFS_File_Stat, Stat(vfs.c)
- Sys_FStat: VFS_File_Stat, FStat(vfs.c)
- Sys_Seek: Seek(vfs.c)

INODES



- Data structure that stores information about file/directory.
- Types: `GFS3_DIRECTORY` or `GFS3_FILE`. Inodes can be both directories or files.
- The extents point to the disk blocks which store the actual data of the file if the type is `GFS3_FILE`, store dirents if the type is `GFS3_DIRECTORY`.
- File Blocks in an extent will be contiguous memory.
- Don't need to worry about mode.

INODES

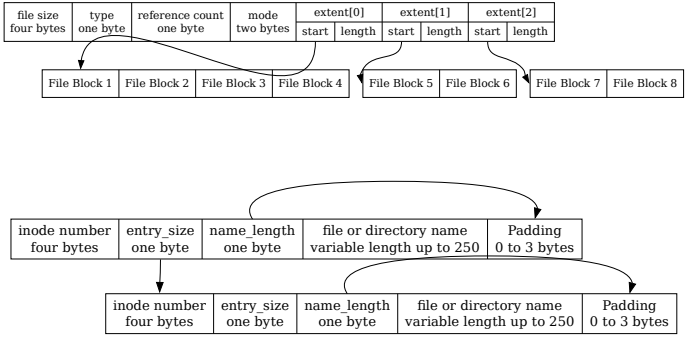
Inodes do not identify themselves. If you need to manipulate an inode, an inode number is needed, which can be found in dirents.

Dirent entries give us the file/directory name and inode number.

getInode(inode_num) ← highly recommended helper function to have!

- Find the block on disk holding this inode number.
- Use Get_FS_Buffer in FS_Buffer to read from disk. It will provide access to disk blocks with proper caching.
- Get the inode from the buffer and return it by computing the right offset.
- Be careful with dangling pointers.

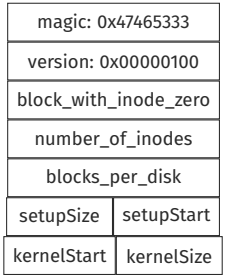
DIRENT (GFS3 DIRENT)



- Dirents are data structures used to store the **contents** of a **directory**. This is stored in the data blocks of a GFS3 DIRECTORY's inode.
- Black arrows in dirents are not pointers, they are here only to illustrate the size.

GFS3_MOUNT

- See pfat.c
- Define the structure of GFS3 instance (In-memory information describing a mounted GFS3 filesystem). Create and allocate a GFS3 instance, and store in mountpoint→fsdata.
- **Hint:** You need at least the superblock information in the instance and you can add whatever parameter you need for the project.
- To load superblock:
  - Load the boot block (first block), find the size of the block in fileio.h `SECTOR_SIZE`(Need this size to malloc a buffer to pass to Block_read). Use Block_Read in blockdev.c.
  - Load the superblock from the boot block. It should be stored at byte offset `PFAT_BOOT_RECORD_OFFSET` from the boot sector (use memcpy).
- Check the magic number of the superblock. The magic number will change if corrupted, will be 0 if not initialized (`GFS3_Magic`).



GFS3_MOUNT

- For cache (check struct “FS_Buffer_Cache”), you can use Create_FS_Buffer_Cache.
- Note you should always use bufcache.h API to read from disk except maybe for the boot block.
- Store the GFS3 instance in mountpoint→fsdata.
- Store mountpoint → ops properly using the mount point ops defined for you.

GFS3_OPEN

- Goal: Open a file given a path.
- e.g.: /home/test/file.c ← path
- You can get the instance of `GFS3_Instance` from mountPoint→fsData where you stored earlier in GFS3_Mount.
- Get inode number of the inode that contains the file.
  - Helpful to have the root directory inode as one of the fields in struct GFS3_Instance.
  - More details on next slide.

GFS3_OPEN

- Split path into segments by '/'
- Loop through dirents under current directory (access dirents through data blocks in inode)
- If the dirent name matches the segment (e.g. 'home')
  - If the segment is not the last segment
    - Repeat the process for the found dirent corresponding to the desired subdirectory (e.g. 'home')
  - If the segment is the last segment (e.g. test.c):
    - Create a struct GFS3_file (custom struct you should define, which stores all information to manipulate a file – **inode and inode number**)
    - Allocate struct File (Allocate_File() in vfs.c)
    - Store the file inode in the struct GFS3_file, and set file→fsdata to point to your GFS3_file
    - Store the struct File pointer you allocated to *pFile (you need to dereference the double pointer, it's changing the value of a pointer)

GFS3_READ

- **Goal:** Reads data from the current position in file
- **Parameters:** `struct File *file, void *buf, ulong_t numBytes`
- Get the start pos(file→filePos) and calculate the end position.
- Retrieve the GFS3 file and the GFS3 instance you stored earlier in File→fsdata and File→mountpoint→fsdata.
- Get the inode from the GFS3 file (If you are not storing the inode in GFS3 file, then get the inode using the inode num)
- Walk through the data blocks and get the block for the start position.
- It may be possible that file data is stored in more than one data block so you may have to read from multiple data blocks. For each data block, read proper size of data from the file into the buffer.
- Return the number of bytes read (Could make it to a helper function)

GFS3_READ_ENTRY

- **Goal:** Read a directory entry from an open directory
- To read all files under a directory, you should keep calling the function until it returns `VFS_NO_MORE_DIR_ENTRIES`
- So you want to keep your progress across calls to read_entry (by using file→filepos). When it cannot find more entries, should return `VFS_NO_MORE_DIR_ENTRIES`

Steps:

- Retrieve the GFS3 file and the GFS3 instance (same file→fsdata, file→mountpoint→fsdata)
- Get the inode from the struct GFS3 file (here file is a directory)
- Loop through dirents making use of entry_length (check struct `GFS3_dirent`)
- Skip dirents until you hit the dirent after filepos
- Increment filepos after that
- Copy the name of the dirent to the VFS_Dir_Entry

GFS3_STAT & GFS3_FSTAT

- Get metadata of file named by given path or metadata for given file.
- GFS3_Stat:
  - You can get GFS3_Instance from mountPoint→fsData (remember we stored it in GFS3_Mount)
  - Get the inode from the path (like you did in GFS3_Open)
- GFS3_FStat:
  - You can get GFS3_File from file→fsData
  - Get the inode of the file
- It should be easy if you have implemented the helper function of getting an inode.
- Set VFS_File_Stat *stat correctly.

GFS3_FSTAT_DIRECTORY

- Goal: Stat operation for an already open directory
- You can get GFS3_File from dir→fsData
- Get the inode of the directory
- Set VFS_File_Stat *stat correctly

GFS3_OPEN_DIRECTORY

- Similar with GFS3_Open, but be careful with the parameters in Allocate_File (mainly pass the proper mode)

GFS3_SEEK

- **Parameter:** `struct File *file, ulong_t pos`
- Set up filePos of a file

GFS3_CLOSE & GFS3_CLOSE_DIRECTORY

- **Goal:** Close a file or a directory
- Free all the resources you've allocated
- I.e. Free the GFS3 File

GFS3_DISK_PROPERTIES

- Use mountPoint to retrieve your GFS3 file system instance
- Fill in block_size and blocks_in_disk (can get this info from superblock in GFS3 instance)

NOTE

- Remember to do sanity checks and free memory when necessary.
- Use Release_FS_Buffer to release a buffer after using one, even if you're just reading it.
- Find root inode: Use `GFS3_INUM_INODE` to get the inode number of the root.
- Create 2 structs – GFS3_file and GFS3_instance
- Check Mount_Point struct, Block_Read function
- The 0th inode is at the very beginning of block **block_with_inode_zero**. The inode addresses are like an array on the disk, the next inode address is just after the previous one. But inodes will not span blocks.
- In the Dirent diagram in the spec, entry_size also includes the size of this field itself.
- **Test Files to run** – gfs3tst.c

TESTS

- test-r-Basic_Seek: gfs3tst 16
- test-r-Basic_Stat: gfs3tst 18
- test-r-Close_Twice: gfs3tst 9
- test-r-Illegag_FD: gfs3tst 10
- test-r-Mount: gfs3tst 1
- test-r-Open_Existing_Dir: gfs3tst 34
- test-r-Open_Existing_File: gfs3tst 33
- test-r-Open_Inexisting_File: gfs3tst 2
- test-r-Read_Entry: gfs3tst 27
- test-r-Read_File: gfs3tst 35
- test-r-Recursive_Stat: gfs3tst 29
- test-r-Seek: gfs3tst 17
- test-r-Stat: gfs3tst 19
- test-r-Stat_Dir: gfs3tst 20