PROJECT 5B:
FILE SYSTEM - WRITING

Minimum Requirements: None

TEST DISTRIBUTION

Public tests – 39 tests | 192 points

Release tests – 0 tests | 0 points

Secret tests – 0 tests | 0 points

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

GOALS

It is time to add the write part of the file system.

User should be able to create, update, and delete files and directories.

This project builds directly on top of 5A.

FUNCTIONS TO IMPLEMENT

- static int gfs3_Write (struct File *file, void *buf, ulong_t numBytes)
- static int gfs3_Open (struct Mount_Point *mountPoint, const char *path, int mode, struct File **pFile)
- static int gfs3_Create_Directory (struct Mount_Point *mountPoint, const char *path)
- static int gfs3_Delete (struct Mount_Point *mountPoint, const char *path, bool recursive)
- static int gfs3_Sync (struct Mount_Point *mountPoint)

SYSTEM CALLS TO MODIFY

- Sys_Delete
- Sys_CreateDir
- Sys_Sync

BUFFER

Get_FS_Buffer
/* modify buffer->data */
Modify_FS_Buffer /* mark the buffer been modified */
Release_FS_Buffer

When writing to disk blocks remember to sync on close.

BITMAP

- How do we know which block is free?
  - BITMAP!
- Bitmap exists in the data blocks inside the second inode.
- You should read in the bitmap at mounting time and keep it up to date.
- The in-use block bitmap is your friend for finding and releasing disk blocks.
  - See Section 1.2.5 of the spec for details

GFS3_OPEN

- If a file does not exist and (mode & O_CREATE) == True, create it
- If a containing directory does not exist throw an error (ENOTFOUND)
- Split the path to directory name and file name
- Retrieve the directory inode
- Find a free writable inode (ref count is 0) for the file. Set up the parameters for our new file and write the inode on the disk.
- If inodes are used up, return ENOMEM
- Create a dirent representing the file and write it in the dirent blocks in the directory inode's data block.
    - You can use a helper function from both this and gfs3_Write here
    - Remember to update parent inode, write the new inode to disk

GFS3_CREATE_DIRECTORY

- If the directory already exists return EEXIST
- If the parent directory doesn't exist return ENOTFOUND
  - Directories should not be created recursively.
- Split the path and get the parent directory's inode
- This should be the same routine for gfs3_OPEN, make a helper function
  - Find a free writable inode for the new directory
  - Create a dirent for the new directory and add it to the parent directory inode's extents
  - Modify parent inode
- Make dirents in the new directory for '.' and '..'
  - This is the only difference when creating a file and creating a directory.

GFS3_WRITE

- [RECOMMENDED]
  - Create a helper function to write to data blocks
    - Let's you bundle the same work between writing files and directories
  - Create a helper function that updates inode for you.
- Use File→filePos to determine the starting location
- If the existing extents can hold all your data (from start to start + size), then just overwrite the existing file blocks.
- If not, there are a few options for extending the size of the file:
  - Check if there's free space after the last file block
  - Find a larger contiguous chunk of free blocks (with Find_First_N_Free), move all file blocks to these free blocks and release old blocks
  - Add a new extent
    - Remember, this needs to work even if writing to the same file multiple times. You can't always add a new extent

GFS3_DELETE

- Delete the dirent entry from parent directory
  - You can just zero out name_length and the inode number to mark this dirent as invalid
  - Make sure how your code for finding dirents from a path works with how you delete
- Decrement inode reference count
- If inode reference count drops to zero (always true in the scope of this project)
  - Release the inode on disk (set ref count, size, start_block, length_block to 0)
  - Release all disk blocks, mark corresponding bit in free disk block bitmap to 0

GFS3_SYNC

- Return Sync_FS_Buffer_Cache() (already provided to you, check bufcache.c)
- Pass the cache you created during mount as the parameter

GFS3_CLOSE

- You should always sync before close if you ever write to the disk!
- Safe to just call sync every time before you close

NOTE

- Remember to do sanity checks and free memory when necessary
- Don’t forget to use Release_FS_Buffer after using a buffer, especially now that we are writing
- If a file has size expect there are some data blocks.