Dharmik Patel - dsp187
Dhruv Chaudhry - dc1533
ilab: cp.cs.rutgers.edu

Makefile:
  - make rufs: run this command to compile the rufs.c
  - make check_mt: run this command to check if the DISKFILE is mounted
  - make remove_mt: run this command to remove the mount. Helpfull when rufs exits without calling rufs_destroy()
  - make run_fuse: run this command to run our custom file System
  - make clean: remove all compiled files AND the DISKFILE. (erases our 'HDD')
  - our mount is at /tmp/dsp187/mountdir

Benchmarks:
- simple_test.c:
  - Run time: 1.4 milliseconds
  - Number of data blocks used: 123 dblocks
- test_cases.c:
  - Run time: 2.1 milliseconds
  - Number of data blocks used: 123 dblocks
- Notes:
  - Run time was calculated using clock() from time.h in milli seconds
  - Number of data blocks used, does not include superblocks, inode blocks, and bitmap blocks. It is only the data blocks.
    - just printed out the number of set bits in dblock_bm when rufs_destroy was called

Implementation:
  - There were 4 sets of functions that we implemented. The entry functions, the helper functions, dir functions and file functions
  - Entry Functions:
    rufs_mkfs:
      - this was called every time we were starting from an unformatted DISKFILE
      - it setup the root dir, with a single child dir '.'
      - we also setup the bitmaps with calloc to make sure they were all zero
    rufs_init:
      - the entry point of FUSE. if the disk was already formatted, we just loaded the superblock into mem.
      - this also served as a test bed for our helper functions before we implemented the dir and file functions
    rufs_destroy:
      - if the program exited with ^C, this function would get called, and free all the mem, and close the DISKFILE
      - we also printed the amount dblocks used at this point. see section above on Benchmarks.
    rufs_getattr:

- this would be called a lot of times by FUSE
- if a file/dir was not found, then it would return -ENOENT and set errno to ENOENT.
- Note on perm for REG files:
  - Abhilash, TA, said we can ignore the perms while creating a file for this project. so I just set them to rw-r--r--. the benchmarks create files with rw-rw-rw-. But that is not important for this project

- Helper Functions:

  my_print:
  - print only when DEBUG was set to 1, except the value of DEBUG did not matter in my_print_always().
  - these set of functions were a wrapper around printf, and helped us debug the code effectively with the different color outputs

  bitmap functions:
  - using the already provided bitmap functions, we implemented get_avail_ino() and get_avail_blkno().
  - every time we would run these 2 functions, we would make sure to do bio_read() and bio_write(), to ensure our changes were persistent on disk

  readi() and writei():
  - used to write and read inodes
  - made sure to calculate the proper offset needed since there could be multiple inodes in 1 inode block

  bio_read() and bio_write():
  - although these implementations were provided, we made sure to that every time we used, we would:
    1. make sure the buffer was of size BLOCK_SIZE
    2. if we were filling the buffer in from scratch, and not from bio_read(), we would calloc the mem used by buffer
    3. when writing data blocks that were stored in the direct pointers, we would always add 'sb->d_start_blk'

  dir_add():
  - used to add a file or subdir to a given parrent_dir
  - we would make sure to update the parrent_dir hardlink counter
  - we would also calloc a new block if needed to add a new dirent
  - thoroughly tested by the Benchmarks

  dir_find():
  - used to check if a file or subdir was in a given parrent_dir
  - make sure that the parrent_dir is a dir and not a file

  get_node_by_path():
  - we had a recursive approach to this
  - it would find a inode related to the absolute path given

  Once these were implemented, the rest of the functions were easy

- Dir Functions:

  rufs_opendir:
  - this would check if a given absolute path was a dir and if it existed

rufs_readdir:
- used to list out all the dirents in a given dir
- used the filler function to put the dirnames

rufs_mkdir:
- used to add a subdir at a given absolute path
- made sure that the dirname() was a dir and had enough space to add a dirent
- then we made a new dir with the normal subdirs '.' and '..' and made sure to link them properly.
  dot would link to basename dir
  dotdot would like to parrent_dir

- File Functions:

rufs_create:
- used to create a REG file
- did not allocate a data block yet, since the file size is 0 bytes

rufs_open:
- this would check if a given absolute path was a reg file and if it existed

rufs_write:
- write data to the data blocks
- made sure to use bio_read() on the starting block that contained the offset, to make sure all data before offset was preserved
- same for the block containing the end, so we don't write past offset+size
- all full blocks in the middle could just be overridden, so no need to waste bio_read
- if new data blocks allocated, made sure to also update the inode

rufs_read:
- read data from a file
- similar to rufs_write.
- we made sure to only start reading from the block containing the offset and stopped at the block containing offset+size