RU File System

Abhinay Reddy Vongur (av730)
Anmol Sharma (as3593)

Benchmark Results

simple_test.c:

```
TEST 1: File create Success
TEST 2: File write Success
TEST 3: File close Success
TEST 4: File read Success
TEST 5: File unlink success
TEST 6: Directory create success
TEST 7: Sub-directory create success
Benchmark completed
The elapsed time is 0.001240 seconds
```

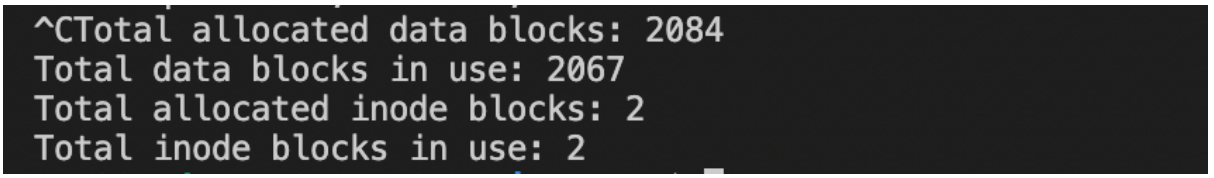
Blocks used after benchmark execution

```
^CTotal allocated data blocks: 33
Total data blocks in use: 17
Total allocated inode blocks: 2
Total inode blocks in use: 2
```

test_cases.c:

```
TEST 1: File create Success
TEST 2: File write Success
TEST 3: File close Success
TEST 4: File read Success
TEST 5: File unlink success
TEST 6: Directory create success
TEST 7: Directory remove success
TEST 8: Sub-directory create success
TEST 9: Large file write success
TEST 10: Large file read Success
Benchmark completed
The elapsed time is 0.022102 seconds
```

Blocks used after benchmark execution



Important Functions:

get_avail_ino():

- Using bio_read, read the inode bitmap from the disk.
- Iterate over the bitmap to check for available inodes. Corresponding bit will be 0 if an inode is available.
- When the first available inode is found, the bit is set to 1 and the bitmap block on disk is updated. The inode number corresponding to the bit is returned.

get_avail_blkno():

- Using bio_read, read the data bitmap from the disk.
- terate over the bitmap to check for available data blocks. Corresponding bit will be 0 if a data block is available.
- When the first available data block is found, the bit is set to 1 and the bitmap block on disk is updated with latest bitmap. The data block number corresponding to the bit is returned.

readi():

- Using the inode number, calculate the block number where the inode will be. Then find the offset for the inode inside the block.
- Fetch the block from disk and read the information from calculated offset.
- Copy the data to the input inode pointer.

writei():

- Using the inode number, calculate the block number where the inode will be. Then find the offset for the inode inside the block.
- Fetch the block from disk into a buffer and copy the data in input inode pointer to the offset calculated.
- Write the buffer back to the block on disk using bio_write

dir_find():

- Get the directory inode using inode number received in input.
- Read the data block containing the directory entry table of the current directory.
- Iterate over all directory entries to evaluate if filename is present in directory, if found copy its attributes to dirent pointer

dir_add():

- Check if file/directory is already present using dir_find() function. If present return the ENOEXIST error code.
- Read the block containing the directory entry table from the disk.
- Iterate over all the directory entry table entries to check if there is an empty slot where the input entry can be added.
- If an empty slot is found, fille the directory entry with the given details. If no empty slot is found then return the error ENOSPC.
- Whenever an entry is added to the directory, the size in inode is incremented by the size of one directory entry.
- **Note: Currently we are supporting only one data block as directory entry table for a directory.**

dir_remove():

- Read the directory entry data block using bio_read()
- Iterate over all directory entry to check if directory entry exists for the file/directory that needs to be removed.
- Reset all attributes of the particular directory entry and set the valid bit to 0.
- Update the directory entry block on disk and inode information of the directory.

rufs_mkfs():

- Initialise the Diskfile using dev_init()
- Initialise the superblock, inode bitmap, data block bitmap.
- Get an inode for root directory using get_avail_ino().
- Initialise the root inode and set the information to point to the root directory.
- Allocate a data block for directory entries of the root directory, initialise by adding . and .. as entries. Set the valid bit as 0 for the rest.
- Update the directory entry table data block number in inode and write the inode to disk.

rufs_readdir():

- Read the inode from the disk using get_inode_by_path() function. If it does not exist, return error ENOENT.
- Fetch the directory entry table data block.
- Iterate over all valid directory entries and copy all the entries into input buffer.

rufs_mkdir():

- Find directory name and the base name of the path using dirname() and basename() functions.
- Get the inode for parent directory using get_inode_by_path() function. If it does not exist, return error ENOENT.
- Get available inode number and data block number to assign to the new directory. If data block/inode are not available, then return the error ENOSPC.
- Add a directory entry for the new directory in parent directory's entry table using dir_add() function. If unable to add then return the error code from dir_add()
- Initialise the directory entry table of the new directory, add . and .. as entries and set the valid bit to 0 for the rest of the entries.
- Write directory entry to the disk. Update the data block number in inode and write to disk.

rufs_rmdir():

- Separate parent directory and target directory pointer.
- Get the inode for target directory using get_inode_by_path(). If not found return error ENOENT.
- Check if the target directory is empty by checking the directory entries in directory entry table. If not empty return error ENOTEMPTY.
- Iterate over the direct pointers of the directory, clear the data block, unset the corresponding bit in data bitmap and write data bitmap to disk.
- Clear the inode and update the inode on disk as well. Unset the bit corresponding to the inode number in the inode bitmap.
- Fetch the parent directory inode by calling get_node_by_path()
- Call dir_remove() function to remove the directory.

get_inode_by_path():

- For the given path, count the number of levels starting from root directory using dirname() and basename() functions.
- Get the directory names for all the levels and base name for the given path.
- Starting from the first level, check if the entry of directory exists in parent directory at each level until the base level is reached. dir_find() function is used to check the entry.
- If at any level the entry is not present, then return ENOENT error.
- If entries for all the levels are present then the inode of the base level file/directory is copied into the input inode pointer.

rufs_create()

- This function is similar to the rufs_mkdir() function, except that the file is not allocated any data blocks when created.

rufs_write()

- Get the inode corresponding to the path using get_inode_by_path(). If not found then return ENOENT error.
- Check if offset exceeds current file size. If yes return 0.
- Calculate the starting data block for writing the data using offset information.
- Get the data block, depending on the size given and space left in the block write the data to the block and update size by subtracting the size of copied data. Move to the next block if needed. Repeat this process until size becomes 0.
- Allocate data blocks when the existing blocks are not enough to hold the data.
- When the direct pointer range exceeds, use indirect pointers and the their data blocks to write the information.
- If at any point, the remaining data can not be copied into the disk then the number of bytes copied until then is returned.
- Whenever the data is copied onto the disk, the inode of the file is updated to reflect the current size of the file.

rufs_read():

- Get the inode corresponding to the path using get_inode_by_path(). If not found then return ENOENT error.
- Check if offset exceeds current file size. If yes return 0.
- Calculate the starting data block for reading the data using offset information.
- Get the starting data block, depending on the size given and offset copy the data from block into the buffer and update size by subtracting the size of

copied data. Move to the next block if needed. Repeat this process until size becomes 0.
- When the range exceeds the direct pointers, use indirect pointers to access the data blocks and copy the data into buffer.
- If any of the direct/indirect pointer entries are -1, then the execution is stopped and size of data copied until then is returned.
- If at any point, the remaining data can not be copied into the disk then the number of bytes copied until then is returned.

rufs_unlink():

- This is same as rmdir() function, except that no check is performed to see if the file is empty.
- Any indirect pointer blocks used by the file are cleared and corresponding bit is unset.

Large File Support (Indirect Pointers)

Write

- When the write range exceeds the range covered by the direct pointers, indirect pointers are used to link more data blocks to the file.
- Based on the offset information, the indirect pointer index is calculated. Then the data block number at the index is read. This block holds direct pointer to other data blocks. Using the offset, corresponding index in the direct pointer block is calculated. The data block number at this index is read and the data is copies into that data block.
- If the indirect pointer is -1 then it means the entry is still not initialised. In that case, a new data block is allocated and and indirect pointer entry is filled with the data block number. Then the data block is read from disk as an integer array and all entries are set to -1. Next, for the first entry in data block, a new data block is allocated and the data block's number is filled in the entry. Now the final data block is used to copy the information into.

Read

- When the read range exceeds the range covered by the direct pointers, indirect pointers are used to access the data.
- Based on the offset information, the indirect pointer index is calculated. Then the data block number at the index is read. This block holds direct pointer to other data blocks. Using the offset, corresponding index in the direct pointer block is calculated. The data block number at this index is read and the data in the block is copied into the buffer.
- If the indirect pointer is -1 then it means the entry is still not initialised. In that case, the process is stopped and the size of data copied until then is returned.

Note:

- Currently the directory only supports the entries that can fit in one block. (can have 17 files/directories inside one directory). Test cases are changed to adhere to this limit.
- The code files for bonus part - Supporting large files - are uploaded separately as we wanted to make sure it does not cause any issues in the original functions. Filename: rufs-large-files.c
- We have made changes to the makefile to include some of the dependencies.