Devin Macalalad dtm97
David Gasperini dlg195

Benchmark

Simple Test:

[detached from 15036.fuse]
[dlg195@ls benchmark]$ ./simple_test
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

```
[dlg195@ls benchmark]$ ./simple_test
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
```

Test Case:

[detached from 15036.fuse]
[dlg195@ls benchmark]$ ./test_case
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 failure

```
[dlg195@ls benchmark]$ ./test_case
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 failure
```

Code

Readi:
calculates block number + offset within block using passed in inode number. Then reads entire block into a buffer and copies inode at specific offset

Writei:
calculates block number + offset within block using passed in inode number. Then reads entire block into a buffer, copies inode passed in into buffer at offset, then rewrites buffer to block

Dir_add:
First checks to see if dirent being added already exists by iterating over all valid blocks within parent directories direct_ptr. If not, an invalid dirent is searched for within valid blocks. If one is found, it is switched to valid and its name and inode number are set. If not, a new block of invalid dirents is allocated, and the first is set. The attributes of the parent directory inode are incremented accordingly.

Dir_remove:
Valid dirents of the blocks of the passed in inode are search, comparing each name to fname. If there is a match, the dirent is invalidated and its attributes are reset. The attributes of the parent directory inode are decremented accordingly.

Dir_find:
Reads in inode of inode number passed in. Iterates over all valid blocks in direct_ptr, checking all dirents within each until a match is found between the dirent name and fname.

Get_node_by_path:
The passed in path is tokenized by ‘/’. For each directory in the path, dir_find is called on it to check existence, and the next inode being searched is set (the initial inode being the root). At the end, the last inode number is read in.

Tfs_init:
The disk file is attempted to be opened. If it succeeds, the superblock is read in. Else, tfs_mkfs is called.

Tfs_destroy:
The superblock is freed, and the disk file closed.

Tfs_getattr:
Get_node_by_path is called to first guarantee the inodes existence and fills in the inode. If it exists, the entirety of the inodes vstat struct is copied.

Tfs_opendir:

Get_node_by_path is called to guarantee the directory's existence.

Tfs_readdir:
Get_node_by_path is called to first guarantee the directory's existence and fill in an inode. Then, every valid block in direct_ptr is iterated over, seeking out each valid dirent within each block. For each one, its name is passed into filler.

Tfs_mkdir:
The parent directory path and file name are separated. Get_node_by_path is called to first guarantee the directory's existence and fill in an inode representing the parent directory. Next, a new inode number is found and dir_add is called for the passed in directory name and this new inode number. Then, an inode is created for the new directory and is written to file.

Tfs_rmdir:
The parent directory path and file name are separated. Get_node_by_path is called to first guarantee the parent directory's existence and fill in an inode to represent it. Then it is called again to guarantee child directory's existence and fill in another inode to represent it. The data block bitmap is then cleared of all the data blocks the child holds. Next, the inode bitmap is cleared of the child directory's inode number. Both bitmaps are written back. Finally, dir_remove is called for the directory.

Tfs_create:
The parent directory path and file name are separated. Get_node_by_path is called to first guarantee the parent directory's existence and fill an inode to represent it. Next, an available inode number is acquired and dir_add is called with it and the passed in name. Finally, an inode for the new file is created and written.

Tfs_open:
Get_node_by_path is called to guarantee the file's existence.

Tfs_read:
Get_node_by_path is called to first guarantee the file's existence and fill an inode to represent it. Next, the beginning block/offset are calculated along with ending block/offset. Finally, all the data is read from those calculations on disk into the buffer.

Tfs_write:
Get_node_by_path is called to first guarantee the file's existence and fill an inode to represent it. Next, the beginning block/offset are calculated along with ending block/offset. Finally, all the data is written from the buffer to those calculations in the disk.

Tfs_unlink:
The parent directory path and file name are separated. Get_node_by_path is called to first guarantee the parent directory's existence and fill an inode to represent it. The same is done for the file. Next, the data bitmap is cleared for all data blocks the file has. Then, the inode bitmap is

cleared of the file's inode. Both are written to disk. Finally, dir_remove is called to remove the file's dirent in the parent directory.