Project 5
CSS 430 Spring 2019
Manvir Singh & Margaret Connor

Project Specification

The purpose of this project is to build a replica of the Unix file system in the ThreadOS. Using a file system the user will be able to manage data on the disk by stream-oriented files instead of the direct access to blocks with ThreadOS's "kernel" calls. The file will provided the user with system calls to open, read, write, update the seek pointer of, close, delete, and get the size of a file.

To implement this the project creates a single level directory named "/" and predefined by the file system. We will create a number of classes to appropriately keep track and store files. The project is limited and works with the default file sizes of ThreadOS.

Source Code modified or created

- Superblock.java
- Kernel.java
- SysLib.java
- FileTable.java
- FileTableEntry.java
- Inode.java
- Directory.java
- TCB.java
- FileSystem.java

Descriptions

Superblock Class

The supper block class is the first block in memory located at block 0. It maintains information about the number of disk blocks, number of inodes, and freeblocks. The Superblock class is managed by the operating system only. The supper block class consists of 4 public functions that allow the operating system to: Synch the superblock to the disk, formats the disk and remaining nodes based on a given value, and get and return free blocks. By default there are 64 Inode blocks.

Inode Class

All blocks following the super block will contain inodes, one for every file. Based of the Unix inode our project's inodes consister 12 pointers, 11 of which point to direct blocks and the last to an indirect block. The Inodes also include the length of the corresponding file name, number of files, and a flag to indicate use of the file. The system is limited to 16 Inodes per block.

Inode contains functions that: writes an Inode to the disk, returns the block number of an Inode, updates an Inode, returns the index of an Inode, and deallocates a block.

Directory Class

The Directory keeps track of all the files in the root directory, each file consisting of a name (with 30 maximum characters) and the size of the file name. Each entry in the directory can be associated with an Inode. On creation the directory receives the max

number of Inodes that can be created and keeps track of which nodes are being used. The directory can read and write from a byte array, allocate and deallocate Inodes, and find the corresponding Inode number given a file name.

FileTableEntry Class & FileTable Class

The File table is a data structure that stores a vector of File Table Entries. This entry includes the seek pointer of this file, a reference to the inode corresponding to the file, the inode number, the count to maintain the number of threads sharing this file (structure) table, and the access mode. The seek pointer is set to the front or the tail of this file depending on the file access mode. The file system maintains the file (structure) table shared among all user threads.

The FileTable class contains functions that allow the program to allocate a new table entry for a file, deal locate a table entry from the FileTable structure, and check if the FileTable structure is empty.

FileSystem Class

The FileSystem class is the main driver for the Thread OS file system we've been building. An instance of the FileSystem class will act as a directory and follow all the rules listed above. The file class provides all the functions listed below.

- int format(int files); Use SysLib.rawwrite to write information into the DISK file. The parameter files specifies the maximum number of files the new volume can support. Specifically, this function should create the following on-disk structures: Superblock, Inodes, RootDirectory and freeList.
- int read(int fd, byte[] buffer); Reads up to buffer.length bytes from the file, currently open on FileDescriptor fd, starting at the current cursor. Return number of bytes actually read.
- int write(int fd, byte[] buffer) Writes contents of buffer to file fd, starting at the current cursor. Return number of bytes actually written.
- int seek(int fd, int offset, int whence); Moves the cursor by offset. Whence is any of SEEK_SET (0), SEEK_CUR (1) or SEEK_END (2). Clamp to 0 .. file length. Return updated cursor value.
- int close(int fd); Closes file fd. Flushes all in-cache Pages for that file to disk. Unregisters fd. Returns 0 or -1.
- int delete(String filename); Deletes the file called filename. If currently open, return -1. Else delete and return 0.
- int fsize(int fd); Return the size, in bytes, of file fd

Results

p5-master — java Boot — 99×48

```
Margarets-MacBook-Pro:p5-master margaretconnor$ java Boot
threadOS ver 1.0:
threadOS: DISK created
Type ? for help
threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)
-->l Test5
l Test5
threadOS: a new thread (thread=Thread[Thread-5,2,main] tid=1 pid=0)
1: format( 48 )...................successfully completed
Correct behavior of format.....................2
2: fd = open( "css430", "w+" )....successfully completed
Correct behavior of open.......................2
3: size = write( fd, buf[16] )....successfully completed
Correct behavior of writing a few bytes.........2
4: close( fd )....................successfully completed
Correct behavior of close......................2
5: reopen and read from "css430"..successfully completed
Correct behavior of reading a few bytes.........2
6: append buf[32] to "css430".....successfully completed
Correct behavior of appending a few bytes.......1
7: seek and read from "css430"....successfully completed
Correct behavior of seeking in a small file.....1
8: open "css430" with w+..........successfully completed
Correct behavior of read/writing a small file.0.5
9: fd = open( "bothell", "w" )....successfully completed
10: size = write( fd, buf[6656] ).successfully completed
Correct behavior of writing a lot of bytes....0.5
11: close( fd )...................successfully completed
12: reopen and read from "bothell"successfully completed
Correct behavior of reading a lot of bytes....0.5
13: append buf[32] to "bothell"...successfully completed
Correct behavior of appending to a large file.0.5
14: seek and read from "bothell"...successfully completed
Correct behavior of seeking in a large file...0.5
15: open "bothell" with w+.........successfully completed
Correct behavior of read/writing a large file.0.5
16: delete("css430").............successfully completed
Correct behavior of delete...................0.5
17: create uwb0-29 of 512*13......successfully completed
Correct behavior of creating over 40 files ...0.5
18: uwb0 read b/w Test5 & Test6...
threadOS: a new thread (thread=Thread[Thread-7,2,main] tid=2 pid=1)
Test6.java: fd = 3successfully completed
Correct behavior of parent/child reading the file...0.5
19: uwb1 written by Test6.java...Test6.java terminated
Correct behavior of two fds to the same file..0.5
Test completed
-->
```