File System Implementation

06CS3008 Sweta Yamini
06CS3009 Naveen Kumar Molleti

April 30, 2009

Objective

To implement the UNIX filesystem (UFS).

Achieved

Implemented a filesystem which resembles the UFS including the features of single, double and triple indirection. We can also write beyond the EOF. We used C++ to write the code.

Implementation Details

- **The Disk** - We use a UNIX file named DISK to simulate a disk to impose our file system on. class Disk contains several methods which would let us simulate the working of a hard disk.
  - **Constructor Disk()** - It creates the DISK, if non-existent and initializes the superblock to zero.
  - **read()** - This function corresponds to the low-level disk read. It can read only a block of data. It is overloaded to simplify reading superblock, inodeblocks and indirection blocks.
  - **write()** - This function corresponds to the low-level disk write. It can only write a single block of data. It is overloaded to simplify writing superblock, inodeblocks, and indirection blocks.
  - **stop()** - Stops the disk. It takes in a boolean argument, and will delete the disk if passed a True argument.

  In our implementation, we have a block size of 256 bytes. And a total of 1024 blocks in our disk.

  Function prototypes:

  - void read(int blockNo, byte* buffer);
  - void read(SuperBlock* block);
  - void read(int blockNo, InodeBlock* block);
  - void read(int blockNo, IndirectBlock* block);
  - void write(int blockNo, byte* buffer);
  - void write(SuperBlock* block);
  - void write(int blockNo, InodeBlock* block);
  - void write(int blockNo, IndirectBlock* block);
  - void stop(bool removeFile);

- **The File System** - A File System on top of the disk is a logical structure provided to the user, for creating, reading and writing files.

  - **SuperBlock** - This is the first block on the disk. It contains information about the disk. It holds the size of the disk, the number of inode blocks, and the first block of the free blocks list. This is defined in *SuperBlock.h*
  - **Inodes** - These are data structures that contain metadata about files. They contain information about the owner, certain flags and size. But most importantly, they contain pointers to data blocks on the disk.
  - **Inode Blocks** - These inode blocks, consist of inodes.
  - **Indirect Blocks** - These only consist of pointers to data blocks.

    All the above three structures are defined in *Inode.h.* In our implementation, owing to a block size of 256 bytes, and an inode size of 64 bytes, every inode block consists of 4 inodes.
  - **Free Space** - The first block of the disk is the SuperBlock, followed by a predefined number of InodeBlocks. This is followed by a set of blocks called FreeList, which consist of pointers to free blocks on the disk.
  - **In-Memory data structures** - The File System maintains several in-memory data structures - the SuperBlock, the InodeBlock table, File Descriptor table, and the seek pointer table. The SuperBlock and the InodeBlock table are just in memory copies of the structures on the hard disk. The file descriptor table consists of the inode numbers of open files. The seek pointer table consists of the positions of the seek pointer in various files.

- **Implemented operations**

  - **formatDisk()** - Initializes the file system to a ”no files” state. Fills up the super block and writes into the free list.

  - **shutdown()** - Closes all files and shuts down ”DISK”
  - **create()** - Creates a file, makes an entry into the inode table. Note that this does not open the file. It returns the inode number of the newly created file.
  - **open()** - Opens an existing file. Takes in as parameter the inode number of the file (We do not use file names, we haven't yet implemented the directory structure). Makes an entry into the file descriptor table, and returns the file descriptor. If no entry is free in the table, returns -1.
  - **read()** - Takes in as arguments, the file descriptor, the buffer to read to, and the length of the buffer. Reads as many bytes as specified and writes into the buffer.
  - **write()** - Takes in as arguments, the file descriptor, the buffer to read from, and the length of the buffer. Reads as many bytes as specified, from the buffer, and writes to the disk.
  - **seek()** - Takes in as arguments, a file descriptor, offset to seek from/to, depending on a third parameter which takes values of SEEK_SET, SEEK_CUR, and SEEK_END, which retain the standard UNIX meanings.
  - **close()** - Takes in as argument a file descriptor, and closes the file, removing its entries, from the File Descriptor table, and the seek pointer table.
  - **delete()** - Takes in as an argument, a file descriptor, frees up all its blocks, adding them to the free list, and removing the entry of the inode in the inode block table.
  - **inumber()** - Returns the inode number of a file, given the file descriptor.
  - **close()** - Takes in as an argument, a file descriptor, and writes back the inode to disk, and removes entries from the in-memory tables.

    Above mentioned is the interface provided to the user. However, we also have two private functions
  - **allocate()** - This allocates a new free block to the file.
  - **free()** - This frees a block, and adds it to the free list.

- **Test Program** We write a program that can use the above mentioned modules to create a file, write some data into it, and then read from it.

- **Submitted files**

  - **Sources**

    * Disk.cpp

    * FileSystem.cpp
    * Inode.cpp
    * main.cpp
  - **Headers**
    * Disk.h
    * FileSystem.h
    * Inode.h
    * SuperBlock.h
    * misc.h
  - **Others**
    * Makefile
    * Report.pdf