VFS Core Report

Team Batman

April 7, 2014

1 Introduction

In this project we implemented a VFS. This was accomplished through the usage of three primary packages: VirtualDisk, FileManager, and FileSystem. VirtualDisk is built upon a java random acces file, it deals only with raw bytes and blocks of memory. FileManager comminucates to VirtualDisk, but is able to appreciate the different between the metadata of file and the data itself. It is aware of directorys and handles writing and reading files and directories. Encryption and compression are also handled in FileSystem. FileSystem is the top level interface where a user can write/read/create files and this connected to the CLI. The following sections will go over the design of each of these sections.

2 VirtualDisk

2.1 Into

The virtual disk package can be broken up into two primay components, the VirtualDisk and the BlockManager. The VirtualDisk is intended to provide the ability to read and write byte[] into memory, and read byte[] from memory, it knows essentially nothing except how to read and write. The BlockManager is much more robust; it is capable of writing blocks to memory and reading blocks from memory. The key aspect here is that it can give the apperence of continuous memory to the user, even if no such thing is availiable. Block manager also handles the issue of dynamic growth.

2.2 Design

2.2.1 VirtualDisk

This class is rather thin, it wraps a java RandomAccessFile object, that allows us to write and read from anywhere within our file at a specified long position. This class is heavily used by BlockManager in order to write and read raw byte[] from the disk.

2.2.2 BlockManager

The BlockManager has to handle three major functions: Write, Read, Delete. Although there is significantly more functionality in this class we will focus on these three functions since they are crucial to the usage of a VFS. As breifly mentioned above one of the major challenges (and in turn features) of this class is that it can write a single byte[] to noncontiguous memory and read that same byte[] later back to the user. This feature allows the file system to take up much less space since once we delete a file we can always begin to write the next file in that location (even if there isnt enough room for the entire file).

The main concept behind the BlockManager is, believe it or not, blocks. All of the memory on the computer is divided into blocks of a size (specified in the constants). Within each block the first 8 bytes is the size of the memory that is written in this block, the last 8 are the location of the next block of data in a continuous chunk are stored, continuous from a user perespective that is. The end of a chain of blocks is denoted by an address of 0.

Read, write and delete are all implemented recursively, they stop once the end of a chain of blocks is reached (or no data is left to be written). Some other functionality of this class includes, being able to find the location of the next free block, this ensures that we are always writing to the lowest block number. The ability to combine blocks, this essentially points the address of next in the first block to a specified block and some other functionality that FileManager utilizes.

The most important part of the BlockManager is that it gives us the illusion of contiguous memory no matter what actually happens underneath, and allows us to abstract away the nasty details of memory management in this system.

3 FileManager

3.1 Intro

FileManager is the first moment where the concept of a 'file' and a 'directory' begin to exist. For all of the classes below this on the stack everything is raw byte[], FileManager allows us to understand that a files and directories are made up of a single header block that is filled with MetaData object, that has all relevent information and a byte[] of the data contained as the contents of the file or the directory. In this section we will talk about the design of MetaData, FileManager.

3.2 Design

3.2.1 MetaData

3.2.2 FileManager

Files are viewed from the file system as a header block with data of a byte[] connected to it. THhe basic design of a file is meta data connected to a byte[] of content, hold whatever is inside the file. A directory has the same thing but the content stored in the byte[] is simpy a list of longs with the location of the children stored in this directory. Although files and directories are implmented very similar there are some suttle difference. The biggest difference is that the byte[] of a file is encrypted and compressed when put into data, where as the contents of a directory, although also a byte[], are left unencrypted and uncompressed. This was choosen becuase the are very often searching in in the tree, and being able to quickly read/write to directories is crucial for preformance. In both cases the meta data of a file and directory is not encrypted or compressed since it was designed to always take up a single block.

When writing a new file or directory the first step is to write the block with metadata then write the contents and combine the two. After this is complete the parent directory has to be updated to know that it has a new child. This means that the parents directories content must be read and re-written with a new location inside of it. Upon deletion of a file the same must happen but in reverse. When the contents of a file are updated the byte[] that the meta data points to is removed and replaced with the new byte[] and then the metadata is linked to this location in memory, this way the parent directory and the metadata do not have to update themselves in any way.

The FileManager also has the ability to deleterecursively, this is essentially removing a directory and its children's directory. In this way it will walk the file system from the current directory downward removing every file and calling it self on every directory. Another interesting function is the searching function. This function does a complete search for a specific path from the root. Given the complete path it is relatively quick operation to find any individual file we search each directory for the proper string in the path, and follow directory untill we get to the proper location.

4 FileSystem

4.1 Intro

The FileSystem is the highest level of FileSystem management. It is built upon the FileManager and provides a lot of the functionality one would expect from a file system. From here the CLI is able to perform any of the 13 commands currently suppported.

4.2 Design

The FileSystem is able to act like a true file system, the users do not need to know about metadata or anything about the underlying memory structure. In this way it supports operations such as writeFile, readFile, addNewDirectory and a number of others.