Implementation of a Virtual File System

Group 03

Thomas Frick (03-150-927)
Matthias Ganz (04-862-850)
Philipp Rohr (04-397-030)

April 2, 2013

Contents

Abstract

The *Virtual File System* was implemented during the course *Java and C# in depth.*

This version of the document describes the project at the final state of milestone 1. and so on...

1. The Full Model

After we did one more experiment that is explained in the appendix I could could examine the measured values and made some observations that are described in this section. Based on those observations I built the full model that will be the basis of a mean value analysis shown further in the document.

1.1. Definitions

blablab

1.2. Observations

This is a table (table 1) measured throughout the and a footnote [1]

N	R_{puts}	$R_{retrieves}$	R_{meas}	X_{meas}	X_{calc}	R_{calc}	Z_{calc}
32	69	112	181	176.5	176.8	181	0
64	72	113	185	344.7	345.9	186	1
96	81	115	196	486.4	489.8	197	1
128	98	119	217	586.8	589.9	218	1
160	124	130	254	628.3	629.9	255	1
192	161	156	317	603.9	605.7	318	1
224	213	204	417	536.4	537.2	418	1
256	251	230	481	531.2	532.2	482	1

Table 1: Measured (on client side) and calculated data of the whole system.

1.3. The Model

and a figure 1. fancy foobar

[1] Described in the appendix **??**

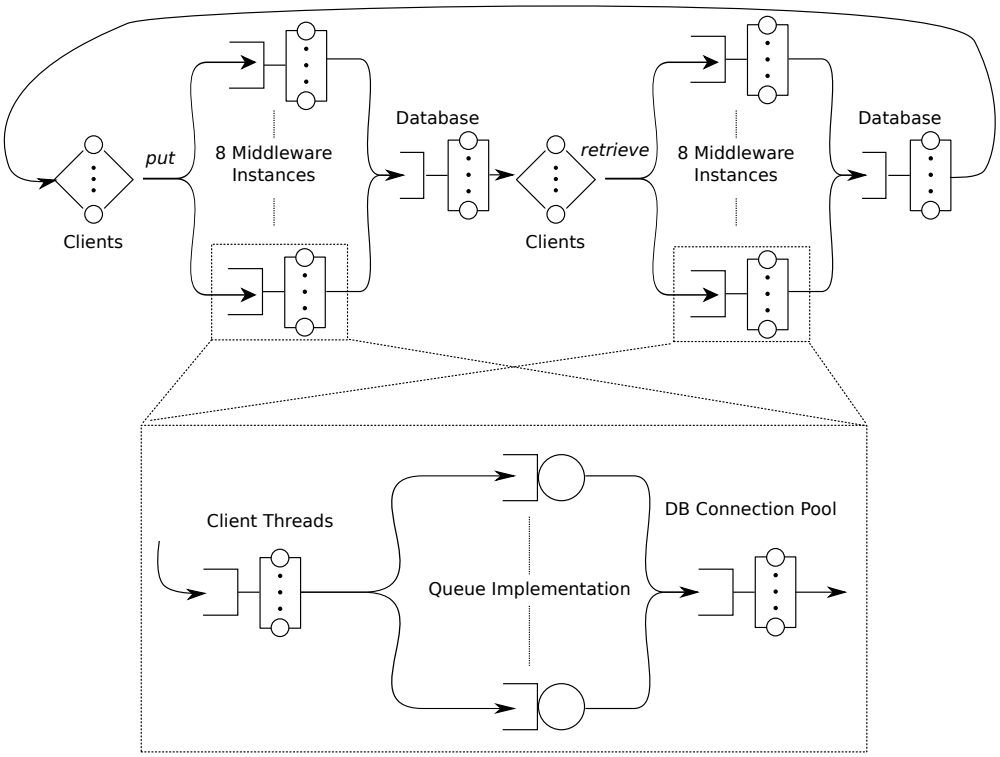

Figure 1: Full model of the system.

citation[2].

some math

$$\mu(n) = \begin{cases} n/S & \text{if } n = 1, 2, ..., m-1 \\ m/S & \text{if } n = m, m+1, ..., \infty \end{cases}$$

2. The File Format

This section describes the binary file format used by the file system inside a virtual disk. The file is separated into three major parts. The header, index and the data section. Each of them is described below.

2.1. Header Section

Name	Lenght	Description
Info	50 byte UTF-8 String	Contains something like Badger VFS 2013 V1.0
Version	10 byte UTF-8 String	Contains something like "1.0"
Compression used	20 byte UTF-8 String	null or indicates compression used for this file
Encryption used	20 byte UTF-8 String	null or indicates encryption used for this file
DirectorySectionOffset	long (8 byte)	File offset where our directory section starts
DataSectionOffset	long (8 byte)	File offset where our data section starts
SaltString	8 bytes	Salt used to hash username and password randomly string generated while creating this file
Password	xxx bytes	CryptoHash (SHA-whatever) of Password+SaltString

2.2. Directory Section

The directory section describes which files and folders belong to which parent directory. This section has a fixed size and contains so called DirectoryBlocks which also have a fixed size. This makes management an manipulation easy. To each directory belongs a B-Tree structure which lists all contained entries.

Directory Block

One Directory Block represents a Node in our B-Tree of order 2.

Name	Length	Description
DirectoryHeader	1 byte	Header information. This header makes it easy to determine whether a DirecotryBlock is in used or not (memory management)
DirectoryEntryBlock1	128 byte	The smaller key inserted into our B-Tree
DirectoryEntryBlock2	128 byte	The bigger key inserted into our B-Tree
DirectoryBlockLink1	8 byte	Points to another DirectoryBlock which contains keys smaller than DirectoryEntryBlock1
DirectoryBlockLink2	8 byte	Points to another DirectoryBlock which contains keys bigger than DirectoryEntryBlock1 but smaller than DirectoryEntryBlock2
DirectoryBlockLink3	8 byte	Points to another DirectoryBlock which contains keys bigger than DirectoryEntryBlock2

Directory Entry Block

Represents a single directory.

Name	Length	Description
Filename	112 byte	UTF-8 Filename String
DataBlockLocation	8 byte	Pointer to a DataBlock located in the Data Section. This DataBlock holds some meta information about the current directory
DirectoryEntryTreeRoot	8 bytes	Pointer to a DirectoryBlock located in the Directory Section. This referenced DirectoryBlock is the Root Block of a B-Tree containing all entries of that directory specified by the current Directory Entry Block. **This field containing a 0 indicates that this entry is a file not a directory**

2.3. Data Section

The data section is split into blocks where each of them is X bytes long. Each block contains some amount of data and points to a subsequent block

Block layout

Name	Length	Description
BlockHeader 0) Header-Bit (LSB) 1) not used 2) not used 3) not used 4) not used 5) not used 6) not used 7) not used	1 byte	 If set to 1 this is the first datablock of a file.
NextDataBlock	8 byte	Points to the start address of the next Datablock (linked list). 0 if this is the last DataBlock of a certain file or folder.
CreationDate	8 byte	UTC Time when this file was created *This field only exists if Header-Bit is set to 1*
DataLenght	4 byte	Indicates the number of data saved on this DataBlock.
Data	n byte	user data (may be encrypted/-compressed)

2.4. The root directory

3. The Compression

To reduce the data volume within the virtual disk, compression on each file can be enabled. Currently available compression algorithms are run length encoding [1] and LZ77 [3].

3.1. Run Length Encoding

The available 8bit run length encoding(rle) algorithm is a very simple form of data compression where multiple occurrence of the same byte were stored as a single byte value and the corresponding count. It is useful for simple graphic images like line drawings and icons.

3.2. LZ77

Abraham Lempel and Jacob Ziv introduced the LZ77 lossless compression algorithm in 1977. Newer compression methods such as GZIP or DEFLATE often use LZ77-based algorithms. The compression is achieved by replacing the data with a reference to an earlier existing copy in the data input stream. For that a window of a certain size is held in memory where existing copies of the current data are searched.

A. Glossary

VFS core The main Java library, that handles all the interaction with virtual disks and importing/exporting/storing files. It is used by the command line client and the gui.

Virtual Disk A virtual disk denotes a container file that is stored on the host file system. A virtual disk can be opened with the software that is developed during this project and stores the actual files. The file extension of the virtual disk is "*.bfs".

B. Command line client

The command line client allows the usage of the VFS core and is mainly intended to test the basic functionalities. The console runs either in management mode or in filesystem mode. The management mode is entered automatically when starting the command line client. It allows creating and disposing virtual disks. The filesystem mode is entered as soon as a virtual disk is opened.

TODO: DISCUSSION: sollen ganze ordner importiert und exportiert werden können? wird dies von der client-seite gehandelt?

B.1. startup

The command line client can be started as follows:

```
java -jar VFSCore.jar ch.eth.jcd.badgers.vfs.ui.VFSConsole
```

B.2. commands

Following commands can be used with the command line client in management mode:

- **create c:\path\to\disk.bfs 1024** creates virtual disk with a maximum quota of 1024 megabytes on the host system. The file may grow up to 1024 megabytes. **TODO**: more parameters are needed (encryption, compression, password if there is encryption)

- **dispose c:\path\to\disk.bfs** deletes the given virtual disk
- **open c:\path\to\disk.bfs** opens filesystem mode for the given virtual disk
- **exit** exits the console program

follwing commands can be used in filesystem mode:

- **ls** lists the contents of the current directory
- **cd dst** changes current directory to *dst* which must be either a child directory of the current path or ".."
- **mkdir dirName** creates a new directory *dirName* in the current path
- **mkfile fileName** creates a new empty file *fileName* in the current path - this is rather not usefull, as the "import" creates a file with content
- **rm file** deletes the entry denoted as *file*, it must be a child of the current path
- **cp src dst** copies the *src* file to *dst* as a child of the current path
- **mv src dst** moves the *src* file to *dst*
- **import ext_src dst** imports a *ext_src* from the host system to *dst*
- **export src ext_src** exports a *src* file to the host system *ext_dst*
- **find searchString** lists all filesystem entries below the current entry containing *searchString*
- **close** closes the filesystem mode, from now on management mode commands can be executed

List of Figures

List of Tables

References

[1] *Run Length Encoding.*

[2] Raj Jain. *The Art of Computer Systems Performance Analysis.* John Wiley and Sons, Inc., 1991.

[3] A. Lempel and J. Ziv. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory 23, 337-343*, 1997.