2-29-2016

Zion File System Simulator

Robert Adams
Grand Valley State University, adamsr@gvsu.edu

Frederic Paladin
Grand Valley State University

Journal of Computer and Communications, 2016, 4, 10-19
Published Online April 2016 in SciRes. http://www.scirp.org/journal/jcc
http://dx.doi.org/10.4236/jcc.2016.44002

Zion File System Simulator

Frederic Paladin, D. Robert Adams

School of Computing and Information Systems, Grand Valley State University, Allendale, MI, USA
Email: frederic.paladin@gmail.com, adamsr@gvsu.edu

Received 30 January 2016; accepted 20 March 2016; published 23 March 2016

Abstract

File systems are fundamental for computers and devices with data storage units. They allow operating systems to understand and organize streams of bytes and obtain readable files from them. There are numerous file systems available in the industry, all with their own unique features. Understanding how these file systems work is essential for computer science students, but their complex nature can be difficult and challenging to grasp, especially for students at the beginning of their career. The Zion File System Simulator was designed with this in mind. Zion is a teaching and experimenting tool, in the form of a small application, built to help students understand how the I/O manager of an operating system interacts with the drive through the file system. Users can see and analyze the structure of a simple, flat file system provided with Zion, or simulate the most common structures such as FAT or NTFS. Students can also create their own implementations and run them through the simulator to analyze the different behaviors. Zion runs on Windows, and the application is provided with dynamic-link libraries that include the interfaces of a file system and a volume manager. These interfaces allow programmers to build their own file system or volume manager in Visual Studio using any .NET language (3.0 or above). Zion gives the users the power to adjust simulated architectural parameters such as volume and block size, or performance factors such as seek and transfer time. Zion runs workloads of I/O operations such as "create," "delete," "read," and "write," and analyzes the resulting metrics including I/O operations, read/write time, and disk fragmentation. Zion is a learning tool. It is not designed for measuring accurate performance of file systems and volume managers. The robustness of the application, together with its expandability, makes Zion a potential laboratory tool for computer science classes, helping students learn how file systems work and interact with an operating system.

Keywords

File System, Simulation, I/O Manager, Volume Manager, Operating System

1. Introduction

File systems are among the most important parts of an operating system. It is the structure and method of organization for data in a storage unit. Without it, the content of a disk is nothing but a long stream of meaningless bytes. A file system keeps track of where these bytes are, which ones are related to others, as well as file metadata. Learning about file systems is essential in any computing program. Not only is it very important in the Information Technology field, but the concepts of data structures and algorithms behind them can also be used in other, higher level, applications. File systems, especially modern ones such as exFAT or NTFS, can be difficult for some students to fully understand. This is the impetus for the Zion project.

Zion is a simulator that is targeted to computing students to help them understand I/O operations and experiment "hands on" with file systems. The application is provided with examples of a simple file system and a volume manager, so that students can see the complex tasks that the operating system has to do in order to send and receive I/O packets. Users can run workloads, which represents a sequence of "create," "delete," "read," or "write" operations, and analyze the results of these instructions. On each workload session Zion shows the time that was spent to read and write files, how many I/O operations were performed, along with additional information. Zion displays a diagram of the volume, divided in blocks, where users can see where a particular file was written. The expandability of the Zion framework allows students to implement and run more sophisticated file systems. Users can create a virtual NTFS or Unix File System, for instance, or they can even invent new ones. The goal of this project is to provide students and professors a tool that could potentially be used in laboratory assignments for Operating Systems or Computer Architectures classes.

This paper is organized as follows. Section 2 provides context for Zion by examining existing file system simulators. Section 3 discusses the design of Zion itself, while Section 4 covers the expandability features. Section 5 provides an evaluation of Zion, and Section 6 points to areas of future work.

2. Background and Related Work

Zion is not the only file system simulator available to students. Two applications in particular are very similar: Modern Operating Systems Simulators (MOSS) by Ray Ontko, and Java File System Simulator by Moazan.

MOSS File System Simulator [1], written in Java, was designed to show how Linux operating systems work behind the scenes. It is a command-based application that has a set of the most important UNIX system calls such as "creat," "open," "read," and "write," but also "mkdir" and "cp." This tool allows users to, among other things, simulate the creation of a file by specifying filename, block size, and blocks. MOSS not only shows the block details output in the terminal screen, but it also creates the actual file on the disk. While this tool is very powerful and it implements a great variety of system functions, it does not let the users execute more than one instruction at the time. Users can create batch files to run multiple instructions, but the tool does not show an overall statistics of such operations.

Java Files System Simulation [2] is an application that can run on any machine and, unlike MOSS, it has a graphical user interface. This application is simple to use and it has a very intuitive interface. Java Files System Simulation allows users to create a virtual drive and copy and paste files from the computer. Unfortunately there is not much documentation available on how new file systems can be created. While there is a way to load file systems from the computer (.fs files), it is not clear to what extends users can customize these file systems.

Both MOSS and Java Files System Simulation are very powerful tools. However, they do not allow users to run a workload of instructions and show statistical information regarding the I/O operations. They do not display details such as the amount of bytes that are read and written, or the time that a particular operation takes (MOSS does show the bytes and blocks, but only for one instruction at the time). Zion, on the other hand, has functionality to load several hundreds of instructions, and show useful analytical information (both numerical and graphical) after each workload session. The interfaces provided with the applications, together with sample code, allow students to easily expand Zion and create new files systems or volume managers.

3. Zion File System Simulator

Zion is a Windows application that allows users to run workloads of I/O instructions such as "create," "delete," "read," and "write" through a simulation of a file system. Users can see where files are being stored in a virtual volume, and they can analyze statistical information including the amount of time needed to complete such instructions, the number of I/O operations, as well as the space used in the volume and the disk fragmentation. Zion is an instructive tool and students can experiment with different workloads. For instance, they can stress a file system with several "creates" and "deletes" to see how the file system handles and minimizes volume fragmentation. An additional improvement over existing simulators is the ability to create new volume types and new file systems, and to dynamically load them into the application through dynamic-link libraries (DLLs).

3.1. Usage

A user starts the application by choosing the volume and the file system they want to use, and customizing these components. The implementation of these components is completely independent of the application framework itself, therefore the customization parameters available depend on the specific type of components that have been loaded into Zion. Out of the box Zion comes with the Simple FS and Standard Disk modules, which are two separate DLLs. Simple FS is a very basic version of a FAT file system. It uses a file allocation table to keep track of where each file (first address) is stored on the disk. It has no maximum number of files that can be stored, unlike FAT and NTFS. However, it does not know the concept of folders: Simple FS is a flat file system. Standard Disk is an example of a volume that, like hard disk drives, is divided into blocks and sectors. The size of the disk can be dynamically set by the user using the parameters provided. Zion runs on Windows machine, and it was designed to mimic as much as possible the internals of Microsoft's most famous operating system. Two of the most important sources of information used to build Zion are "*Windows System Programming*" [3] and "*Windows Internals*" [4].

3.2. Parameters

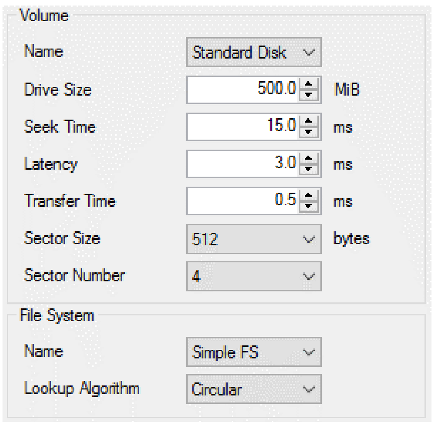
Zion displays a set of adjustable parameters for the volume and file system. With the Standard Disk, the user can specify the drive size, the size of a sector (currently locked to 512 bytes, like in most real life implementations), and the number of sectors for each block, shown in **Figure 1** (the icons used are from [5]).

To provide a more realistic feeling, the user can also specify values for average seek time, latency, and transfer time. Physical disks need to move their magnetic heads to the right track (seek time), then the disks have to

rotate so that the head can access the specific sector (latency), and then the data has to be written/read (transfer time). Zion lets the user specify the average time that these operations take, and then it uses these parameters to calculate runtime statistics.

3.3. Disk Format

Once the disk parameters have been set, the volume needs to be mounted and then formatted, like in actual computers. This is done by means of the menu buttons "Mount" and "Format" (**Figure 2**). The flags next to the file system and volume names denote that the mounting and formatting operations were successful (**Figure 3**). The number of blocks in the disk is calculated based on the disk size and the sector size (**Figure 4**). A block is the smallest allocation unit of a volume. The format process not only creates these allocation units, but it also marks the blocks that are reserved by the file system (this needs to be included in the file system implementation). Zion then displays the graphical representation of the volume showing the blocks in the disks after the formatting process (**Figure 5**).


Volume

Name	Standard Disk	
Drive Size	500.0	MiB
Seek Time	15.0	ms
Latency	3.0	ms
Transfer Time	0.5	ms
Sector Size	512	bytes
Sector Number	4	

File System

Name	Simple FS
Lookup Algorithm	Circular

Figure 1. Volume parameters.

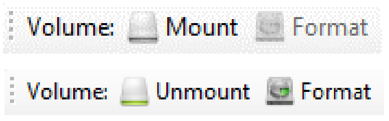
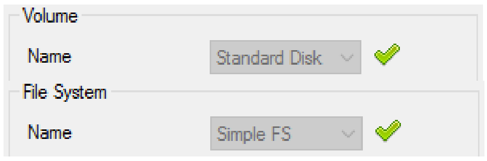
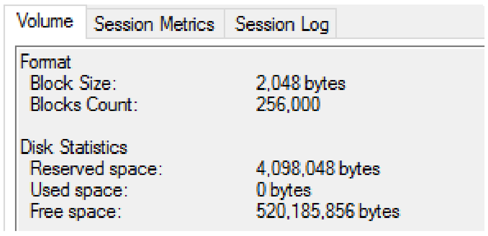


Figure 2. Volume parameters.



Name	Standard Disk ✓
Name	Simple FS ✓

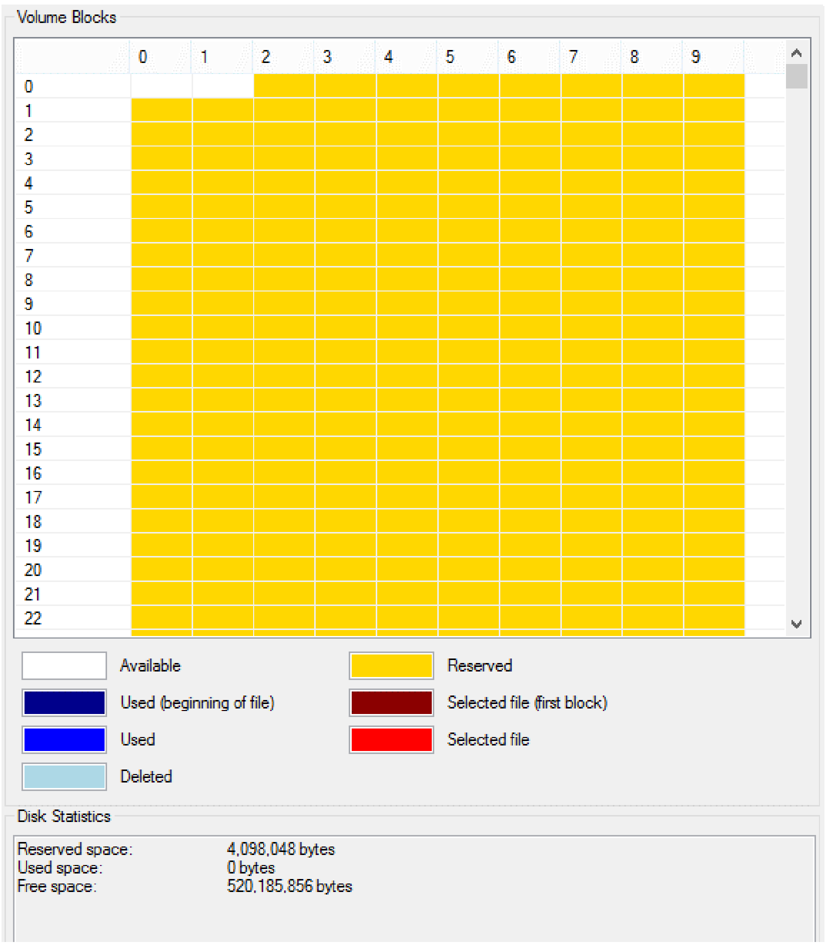
Figure 3. Volume and file systems flags.



```
Format
  Block Size:            2,048 bytes
  Blocks Count:          256,000

Disk Statistics
  Reserved space:        4,098,048 bytes
  Used space:            0 bytes
  Free space:            520,185,856 bytes
```

Figure 4. Format statistics.



```
Reserved space:     4,098,048 bytes
Used space:         0 bytes
Free space:         520,185,856 bytes
```

Figure 5. Diagram of the volume with statistical data.

3.4. Workload

A workload represents a series of instructions for the I/O manager. These instructions include CREATE, DELETE, READ, and WRITE, and they can be saved in a simple comma separated text file. Below is an example workload.

```
CREATE, file4.txt
CREATE, file5.txt
CREATE, file2.txt
WRITE, file2.txt, 1878606 bytes
DELETE, file5.txt
READ, file2.txt, 4412717 bytes
CREATE, file6.txt
DELETE, file2.txt
CREATE, file7.txt
WRITE, file4.txt, 1593981 bytes
DELETE, file7.txt
```

Each instruction has different parameters separated by a comma. The first parameter is the instruction type. The second is the file name, and the third one is the size. Size applies to WRITE and READ only, and it can be in one of the following units (not case sensitive):

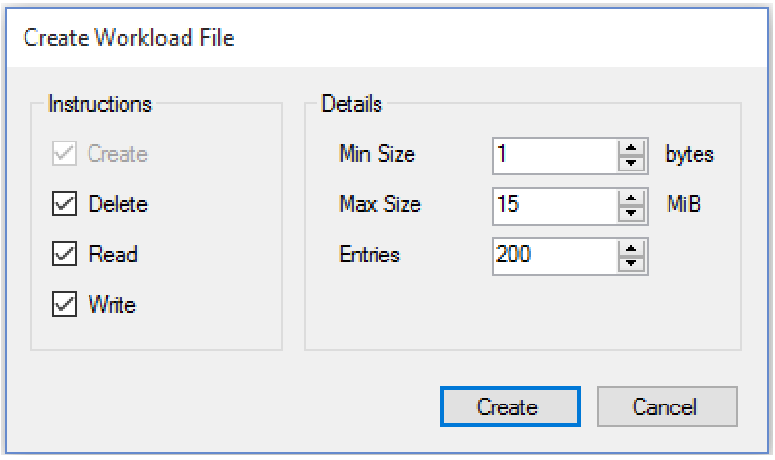
- "mb" or "megabyte": 1,000,000 bytes
- "mib" or "mebibyte": 1,048,576 bytes (2^{20} bytes)
- "kb" or "kilobyte": 1000 bytes
- "kib" or "kibibyte": 1024 bytes (2^{10} bytes)
- "bytes" (by default, if omitted)

Note: while the tool allows users to use MB/MiB as size units, files that are too big might not be written or read. This is because Zion actually allocates the same amount of data into memory for simulation purposes.

Although workloads can be created with any text editor, Zion also provides a tool to help the user create random instructions for the workload (**Figure 6**).

In the "Create Workload File" screen, users can indicate what instructions they want to include, the minimum and maximum size for read and write, and the number of entries. Notice that "create" is always selected. This is because the tool needs to have a reference to existing files in order to generate "delete," "read," and "write" instructions.

The "Open" button (**Figure 7**) loads the workload into the application, and the "Workload" tab of the GUI displays the list of these instructions with the total file count and size (**Figure 8**). Workloads can also be created on the fly by dragging and dropping existing files from your computer. For each file dropped into the list, the system generates two entries: CREATE and WRITE. The WRITE instruction will have the size of the file, and it will contain the actual file data.

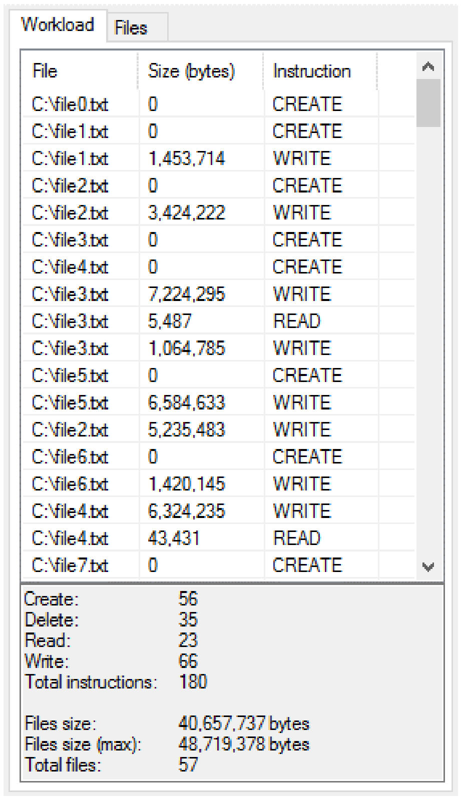

Create Workload File

Instructions
- [✓] Create
- [✓] Delete
- [✓] Read
- [✓] Write

Details
Min Size: 1 bytes
Max Size: 15 MiB
Entries: 200

Figure 6. Create workload screen.

Figure 7. Main workload functions.



File	Size (bytes)	Instruction
C:\file0.txt	0	CREATE
C:\file1.txt	0	CREATE
C:\file1.txt	1,453,714	WRITE
C:\file2.txt	0	CREATE
C:\file2.txt	3,424,222	WRITE
C:\file3.txt	0	CREATE
C:\file4.txt	0	CREATE
C:\file3.txt	7,224,295	WRITE
C:\file3.txt	5,487	READ
C:\file3.txt	1,064,785	WRITE
C:\file5.txt	0	CREATE
C:\file5.txt	6,584,633	WRITE
C:\file2.txt	5,235,483	WRITE
C:\file6.txt	0	CREATE
C:\file6.txt	1,420,145	WRITE
C:\file4.txt	6,324,235	WRITE
C:\file4.txt	43,431	READ
C:\file7.txt	0	CREATE

Create:	56
Delete:	35
Read:	23
Write:	66
Total instructions:	180
Files size:	40,657,737 bytes
Files size (max):	48,719,378 bytes
Total files:	57

Figure 8. Workload ready to run.

3.5. Session

With the volume properly mounted and formatted, and with a workload ready, the user can execute the instructions (Figure 9 and Figure 10).

Running a session consists of looping through each instruction and perform the appropriate operations. Alternatively, users can run a single instruction by selecting it and clicking “Run.” Based on the instructions in the workload, files are created, written, read, and deleted from the virtual volume. Because the list of files at the end of a session may not correspond to the list of files from the workload, the “Files” tab of the GUI shows all the files that are still present on the disk once the session is done (Figure 11). Clicking on the file name will display additional information such as creation, access, and write time (the data available depends on the actual implementation of the file system).

The “Volume Blocks” diagram (Figure 12) highlights the allocation units that are used and available, as well as additional information such as how much space has been taken by files, and the percent of disk fragmentation, although the actual data available depends on the implementation of the chosen file system (Figure 13).

3.6. Session Log and Metrics

Sessions logs and metrics are probably the most important outputs of a workload session (Figure 14 and Figure 15). Depending on user preference (see section “Additional Settings”), the session log can show different types of events. The tool always captures the instruction being executed with a timestamp. In addition, the log can show each time the file system performs one of the four operations (create, delete, read, and write), or when the volume manager writes to or reads from a block. In addition, the session metrics can include how many bytes were read and written, the total and average time spend, the number of I/O operations, and the lookup counts of

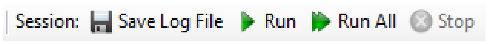
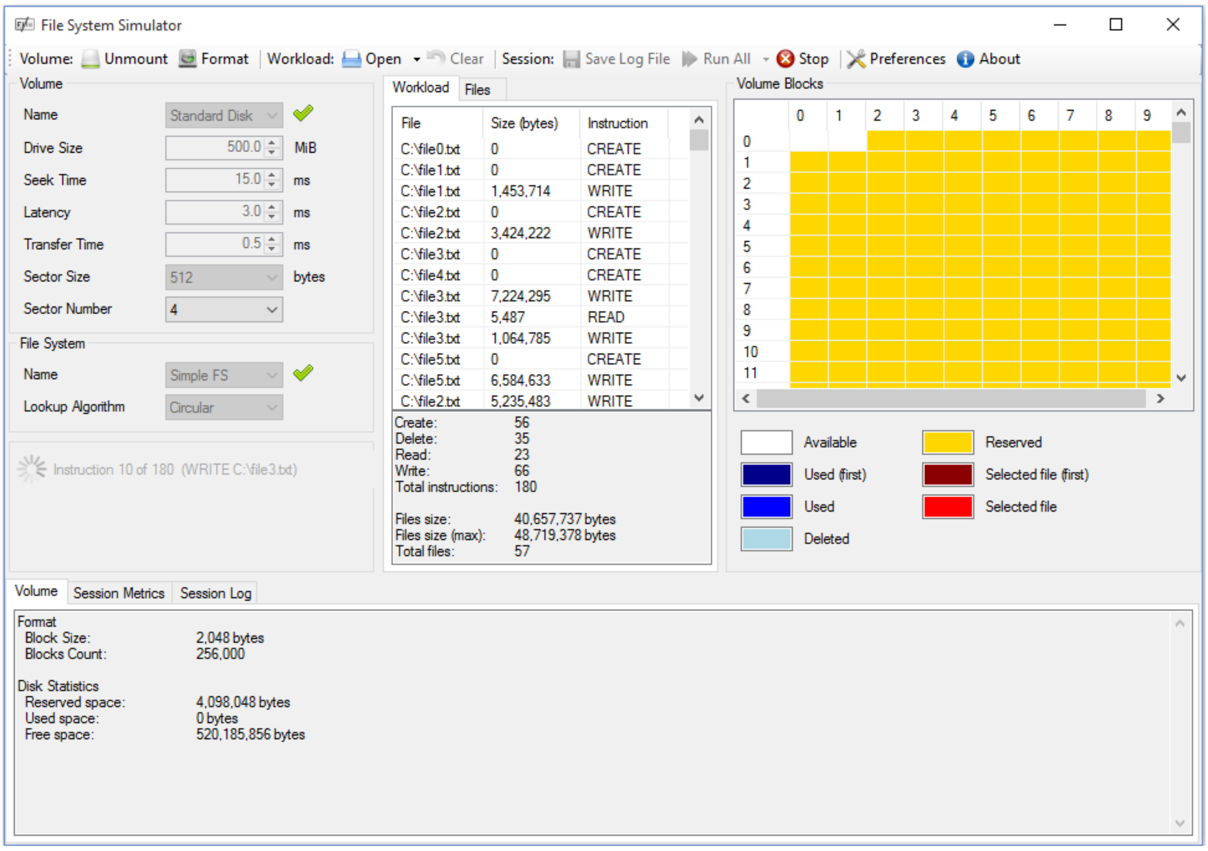


Figure 9. Session commands.



Volume

Name	Standard Disk	✓
Drive Size	500.0	MiB
Seek Time	15.0	ms
Latency	3.0	ms
Transfer Time	0.5	ms
Sector Size	512	bytes
Sector Number	4	

File System

Name	Simple FS	✓
Lookup Algorithm	Circular	

Instruction 10 of 180 (WRITE C:\file3.txt)

File	Size (bytes)	Instruction
C:\file0.txt	0	CREATE
C:\file1.txt	0	CREATE
C:\file1.txt	1,453,714	WRITE
C:\file2.txt	0	CREATE
C:\file2.txt	3,424,222	WRITE
C:\file3.txt	0	CREATE
C:\file4.txt	0	CREATE
C:\file3.txt	7,224,295	WRITE
C:\file3.txt	5,487	READ
C:\file3.txt	1,064,785	WRITE
C:\file5.txt	0	CREATE
C:\file5.txt	6,584,633	WRITE
C:\file2.txt	5,235,483	WRITE

```
Create:             56
Delete:             35
Read:               23
Write:              66
Total instructions: 180

Files size:         40,657,737 bytes
Files size (max):   48,719,378 bytes
Total files:        57
```

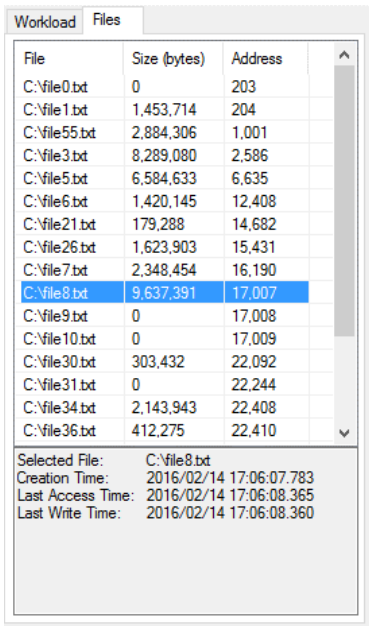
	0	1	2	3	4	5	6	7	8	9
0	⬜	⬜	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
1	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
2	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
3	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
4	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
5	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
6	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
7	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
8	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
9	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
10	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
11	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨

⬜	Available	🟨	Reserved
🟦	Used (first)	🟥	Selected file (first)
🟦	Used	🟥	Selected file
⬜	Deleted		

```
Format
 Block Size:         2,048 bytes
 Blocks Count:       256,000

Disk Statistics
 Reserved space:     4,098,048 bytes
 Used space:         0 bytes
 Free space:         520,185,856 bytes
```

Figure 10. Overview of the application with a session of workload in execution.



File	Size (bytes)	Address
C:\file0.txt	0	203
C:\file1.txt	1,453,714	204
C:\file55.txt	2,884,306	1,001
C:\file3.txt	8,289,080	2,586
C:\file5.txt	6,584,633	6,635
C:\file6.txt	1,420,145	12,408
C:\file21.txt	179,288	14,682
C:\file26.txt	1,623,903	15,431
C:\file7.txt	2,348,454	16,190
C:\file8.txt	9,637,391	17,007
C:\file9.txt	0	17,008
C:\file10.txt	0	17,009
C:\file30.txt	303,432	22,092
C:\file31.txt	0	22,244
C:\file34.txt	2,143,943	22,408
C:\file36.txt	412,275	22,410

```
Selected File:     C:\file8.txt
Creation Time:     2016/02/14 17:06:07.783
Last Access Time:  2016/02/14 17:06:08.365
Last Write Time:   2016/02/14 17:06:08.360
```

Figure 11. List of files available on the disk.

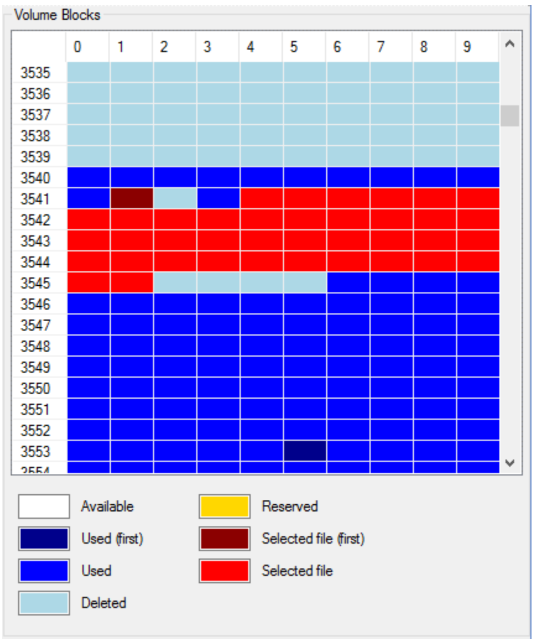
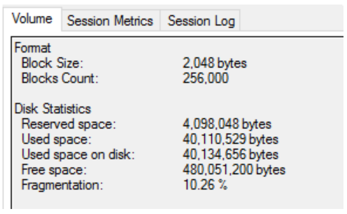


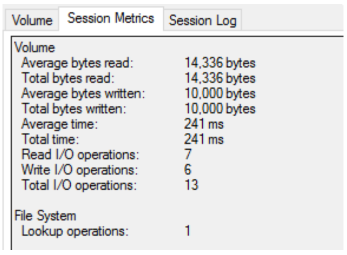
Figure 12. Volume blocks after the workload session is complete.



```
Volume   Session Metrics   Session Log
Format
  Block Size:               2,048 bytes
  Blocks Count:             256,000

Disk Statistics
  Reserved space:           4,098,048 bytes
  Used space:               40,110,529 bytes
  Used space on disk:       40,134,656 bytes
  Free space:               480,051,200 bytes
  Fragmentation:            10.26 %
```

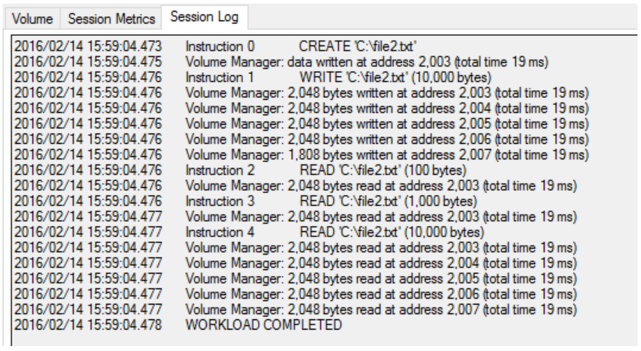
Figure 13. Volume format and statistics.



```
Volume   Session Metrics   Session Log
Volume
  Average bytes read:       14,336 bytes
  Total bytes read:         14,336 bytes
  Average bytes written:    10,000 bytes
  Total bytes written:      10,000 bytes
  Average time:             241 ms
  Total time:               241 ms
  Read I/O operations:      7
  Write I/O operations:     6
  Total I/O operations:     13

File System
  Lookup operations:        1
```

Figure 14. Session metrics.



```
Volume   Session Metrics   Session Log
2016/02/14 15:59:04.473   Instruction 0        CREATE 'C:\file2.txt'
2016/02/14 15:59:04.475   Volume Manager: data written at address 2,003 (total time 19 ms)
2016/02/14 15:59:04.476   Instruction 1        WRITE 'C:\file2.txt' (10,000 bytes)
2016/02/14 15:59:04.476   Volume Manager: 2,048 bytes written at address 2,003 (total time 19 ms)
2016/02/14 15:59:04.476   Volume Manager: 2,048 bytes written at address 2,004 (total time 19 ms)
2016/02/14 15:59:04.476   Volume Manager: 2,048 bytes written at address 2,005 (total time 19 ms)
2016/02/14 15:59:04.476   Volume Manager: 2,048 bytes written at address 2,006 (total time 19 ms)
2016/02/14 15:59:04.476   Volume Manager: 1,808 bytes written at address 2,007 (total time 19 ms)
2016/02/14 15:59:04.476   Instruction 2        READ 'C:\file2.txt' (100 bytes)
2016/02/14 15:59:04.476   Volume Manager: 2,048 bytes read at address 2,003 (total time 19 ms)
2016/02/14 15:59:04.476   Instruction 3        READ 'C:\file2.txt' (1,000 bytes)
2016/02/14 15:59:04.477   Volume Manager: 2,048 bytes read at address 2,003 (total time 19 ms)
2016/02/14 15:59:04.477   Instruction 4        READ 'C:\file2.txt' (10,000 bytes)
2016/02/14 15:59:04.477   Volume Manager: 2,048 bytes read at address 2,003 (total time 19 ms)
2016/02/14 15:59:04.477   Volume Manager: 2,048 bytes read at address 2,004 (total time 19 ms)
2016/02/14 15:59:04.477   Volume Manager: 2,048 bytes read at address 2,005 (total time 19 ms)
2016/02/14 15:59:04.477   Volume Manager: 2,048 bytes read at address 2,006 (total time 19 ms)
2016/02/14 15:59:04.477   Volume Manager: 2,048 bytes read at address 2,007 (total time 19 ms)
2016/02/14 15:59:04.478   WORKLOAD COMPLETED
```

Figure 15. Session log.

the file system (that is, how many lookups are performed in order to find a free entry in the allocation table, or the next address in chain of a file). The user must be aware that including this additional information can significantly slow down the session run, because the system has to capture and display more data.

The amount of information in the session log and metrics depend on the actual implementation of both file system and volume manager. For instance, the file system has to keep track (and send) the amount of total I/O operations to Zion in order to display it at the end of the workload session. Another example is the StandardDisk implementation of a volume. This particular implementation does not allow the volume manager to append data into blocks. Therefore the file system has to do additional reads and writes in order to append data into an existing file. This explains the discrepancy between the bytes from the session metrics and the size from the workload. A session log can be exported into a text file by clicking "Save Log File", or Zion can automatically save all session log files.

3.7. Additional Settings

The "Preferences" screen (**Figure 16**) shows the application settings. Users can define what information to include in a session log ("Include File System Log" and "Include Volume Manager Log" check boxes), whether or not the user interface is automatically updated (although this will significantly slow down the session run), or if they want to automatically save the session log and metrics in a text file at the end of each workload session. The "Color Highlight" drop down allows users to define if the blocks in the Volume diagram are colored based on the different block types ("used" versus "reserved" versus "free"), or by file name.

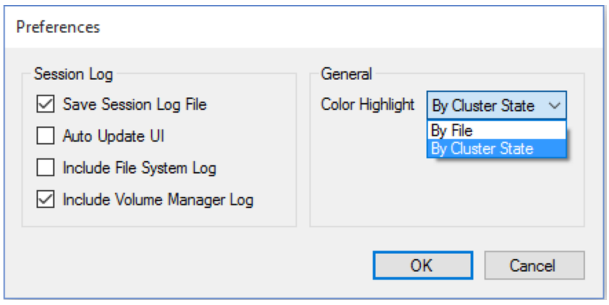
4. Expandability

One of the key strengths of Zion is its expandability. Students can easily create their own file systems or simulate the most popular ones such FAT, NTFS, or Unix File System. All they need to do is create a project in any .NET language (the minimum required is 3.0), import the two interfaces IFileSystem and IVolume, and create their own implementation. Upon startup, Zion scans the folder in which the application is executed and it loads all the available assemblies that have the "FileSystem" or "Volume" in their assembly information.

The interfaces IFileSystem and IVolume were deliberately designed to be as intuitive as possible (see Github Repository). File systems and volumes communicate information to Zion by raising appropriate events (for instance, Write Completed). Zion captures the events and associated data to display metrics

5. Conclusions and Future Work

File systems are an essential component of operating systems, and therefore they are an building block for computer science students. The complexity of file systems can make it challenging for students to learn the various structures and features of file systems available in the industry. Zion was designed to help reduce this learning curve. The goal of the Zion project is to provide a robust, easy to use, and flexible laboratory tool for computer science classes. The Zion File System Simulation application is designed as a learning tool for computing students. It is not intended for measuring accurate performance of file systems and volume managers, although more complex implementations could lead to this. While the tool shows information such as total time and I/O operations, this is just indicative information, and it may not be realistic if the implementation skips steps (such as a circular scan) on certain operations. Students may find it hard to notice differences in metrics from one file


Preferences

Session Log
[✓] Save Session Log File
[ ] Auto Update UI
[ ] Include File System Log
[✓] Include Volume Manager Log

General
Color Highlight: By Cluster State

Figure 16. Preferences screen.

system implementation to another, unless these file systems are substantially different in the way they handle the I/O operations.

Zion is an instructive tool for students who are learning operating systems and computer architectures, and its educational power is at different level. The learning experience could be as little as running workloads and observing the results, reading the code of actual file systems and volume implementations (learning more about data structures and algorithms), or actually creating their own implementations. Zion provides a range of choices to educators.

Zion certainly has room for improvement. Future work could include providing additional file systems and volumes, perhaps a simulation of actual FAT or NTFS. New volume managers could simulate rotating disks using different scheduling algorithms such as Circular Scan or Shortest Seek Time First. Whether used as it is, or expanded with new components, Zion has the potential to become a teaching tool in Operating Systems or Computer Architectures laboratory classes.

The installer, the file system and volume interfaces, as well as part of the source code, are available online (GitHub Repository [6]).

References

[1] Ontko, R. (n.d.) MOSS File System Simulator. http://www.ontko.com/moss/filesys/user_guide.html

[2] Moazan (n.d.) Java File System Simulation. http://sourceforge.net/projects/javafilesystem/

[3] Hart, J. (2010) Windows System Programming. 4th Edition, Addison-Wesley, Boston.

[4] Russinovich, M., Ionescu, A. and Solomon, D. (2012) Windows Internals. 6th Edition, Microsoft Press, Redmond.

[5] Oxygen Team. The Set of Icons Used in Zion Is “Oxygen Icons” by Oxygen Team. http://www.iconarchive.com/show/oxygen-icons-by-oxygen-icons.org.html

[6] Github Repository. https://github.com/fredericpaladin/Zion