University of Northern Iowa
UNI ScholarWorks

Honors Program Theses | Student Work

2018

A graphical file system visualization tool for operating systems

Jacob Bundt
University of Northern Iowa

Let us know how access to this document benefits you

Copyright ©2018 Jacob Bundt
Follow this and additional works at: https://scholarworks.uni.edu/hpt

 Part of the Computer and Systems Architecture Commons, and the Data Storage Systems Commons

Recommended Citation

Bundt, Jacob, "A graphical file system visualization tool for operating systems" (2018). *Honors Program Theses*. 348.
https://scholarworks.uni.edu/hpt/348

This Open Access Honors Program Thesis is brought to you for free and open access by the Student Work at UNI ScholarWorks. It has been accepted for inclusion in Honors Program Theses by an authorized administrator of UNI ScholarWorks. For more information, please contact scholarworks@uni.edu.

Offensive Materials Statement: Materials located in UNI ScholarWorks come from a broad range of sources and time periods. Some of these materials may contain offensive stereotypes, ideas, visuals, or language.

A GRAPHICAL FILE SYSTEM VISUALIZATION TOOL FOR OPERATING SYSTEMS

A Thesis Submitted

In Partial Fulfillment

Of the Requirements for the Designation

University Honors with Distinction

Jacob Bundt

University of Northern Iowa

December 2018

This Study by: Jacob Bundt

Entitled: A Graphical File System Visualization Tool for Operating Systems

has been approved as meeting the thesis or project requirement for the Designation University Honors with Distinction

_____ _____

Date Dr. Sarah Diesburg, Honors Thesis Advisor

_____ _____

Date Dr. Jessica Moon, Director, University Honors Program

Abstract

Fileshark was developed for the primary purpose of aiding in the teaching of students in file system related courses, particularly at the undergraduate level, by providing a visual representation of often abstract file system ideas. Many teachers find it difficult to teach the file system in detail in a typical operating systems classroom environment due to its abstract nature and the inability to truly observe what is occurring within the file system. Fileshark provides an easy to use interface that can be used by both students and professors to visualize the internal workings of the file system and the processes that allow the file system to read, store and delete data. Such a program also has applications for systems forensics and system security related research. The application described in the following project is the result of those aims. The future goal of this project is for the application to be tested in the classroom setting with the goal of continued implementation into file system related curriculum.

I. Introduction

Within a computer there are many complex processes that must be carried out for even the smallest task to be carried out. Of the many portions of computers that manage these processes the file system stands out as not only one of the most important process managers, but also one of the least understood aspects of the system. The file system is responsible for managing the storage, naming, access rights, and metadata for files and directories. Almost every process revolving around files and directories is coordinated by the file system. Despite its importance, many teachers find it difficult to teach the file system in detail in a typical operating systems classroom environment due to its abstract nature and the inability to truly observe what is occurring within the file system. Due to this difficulty, the research contained in this thesis revolves around the development of a

web-based application for the purpose of aiding in the teaching of students enrolled in file system related courses, particularly at the undergraduate level. The purpose of the project was to create an easy to use interface that can be used by both students and professors to visualize the internal workings of the file system and the processes that allow the file system to read, store and delete data. The resulting program was entitled Fileshark.

Along with educational benefits, this program was designed to be an adaptable tool that has broader application to any other fields that observe file system data. For example, system forensics is a branch of criminal forensics that pertains to criminal evidence that can be found in computers and their internal storage. Fileshark can be used to study where and how data is being stored thus making it possible to find hidden information in the system storage. Another area is system security. Dr. Diesburg's dissertation used the information gathered from the internal workings of the program to study secure deletion habits. With Fileshark, we are capable of tracing deletion methods to see the completeness of deletion. This can help to make sure important information is not left on hard drives creating security risks. Once the broad applications are understood the necessity for a file system visualization tool becomes quite clear. These are a few of the reasons Fileshark was created. Through the process of its creation I learned a number of important skills—namely, programming in two new languages (HTML5 and JavaScript), basic networking tools, webpage design, and finally a better understanding of the file system.

II. Literature Review

To better understand what Fileshark does, we must first understand the basics of a file system. Currently, Fileshark is using a mounted EXT3 file system [14]. EXT3 is a member of the family

of file systems that are most commonly used by Linux operating systems. Also, the Linux operating system is used by a majority of servers on the Internet. Therefore, the use of EXT3 reaches a well-established and vast community of users. Figure 1 describes an overview of how the file system operates.

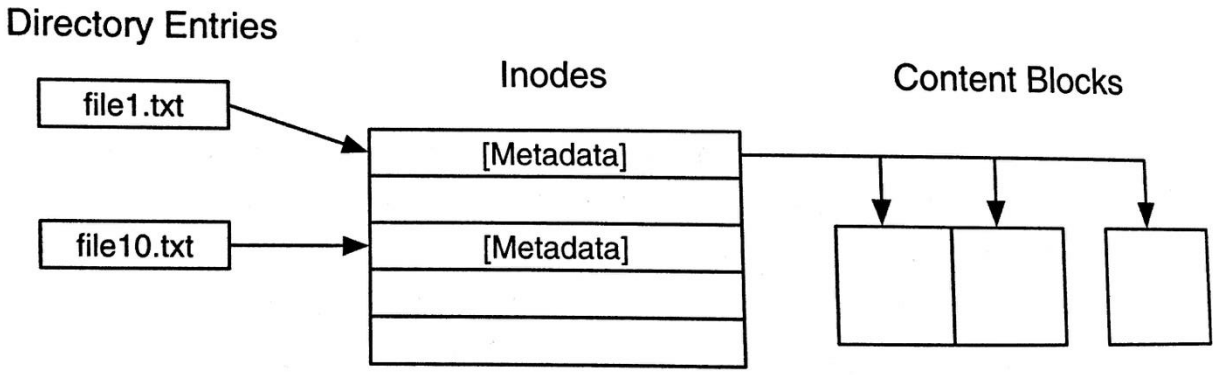


Figure 1: Relational Diagram of the EXT Family of File Systems [3]

Below are three important definitions that must be understood to understand how a file system operates.

- **Directories** – A directory stores a collection of directory entries. These directory entries are files and a pointer to the inode for the file. For example, as shown in figure 4 the directory contains file1.txt and file 10.txt. These files point to a set of inodes.
- **Inodes** – Inodes (or index nodes) store the metadata for the files and directories in memory. In figure 4 each file in the directory points to a block of inode data containing metadata.
- **Metadata** – Metadata describes the data you are trying to access. In figure 4, file1.txt points to the metadata stored in the inodes. The metadata stores information about the data such as where it can be found and how it is written so that when accessed it is easily read by the

system. The metadata is crucial in understanding how the operating system uses memory. By examining the metadata the storage of data both in where it is stored and how it is stored can be observed.

Understanding the role each of these components plays helps users visualize how the system operates and understand the importance of observing this behavior. The best way to observe this is visually as can be done using Fileshark. Table 1 below [4] shows how the file system sends write and delete actions to the storage drivers. (Note that only low-level WRITE operations are shown, even though a file is being deleted. When a file is deleted, only new metadata is written.)

Table 1: File system trace showing what happens when a file is written to disk and deleted.

Mount the file system					
Timestamp	**Action**	**Sector**	**Page ID**	**I-nodes**	**Sector Type**
1	WRITE	2	1398389		EXT3 SUPERBLOCK
2	WRITE	3	1398389		EXT3 SUPERBLOCK
Create file and sync					
Timestamp	**Action**	**Sector**	**Page ID**	**I-nodes**	**Sector Type**
3	WRITE	12290	1398723	12	DATA
4	WRITE	12291	1398723	12	DATA
5	WRITE	1052	1398393	8	JOURNAL SUPERBLOCK
6	WRITE	1053	1398393	8	JOURNAL SUPERBLOCK
7	WRITE	1054	1398393	8	JOURNAL DESCRIPTOR
8	WRITE	1055	1398393	8	JOURNAL DESCRIPTOR
9	WRITE	1056	1398391		INODE FREEMAP IN JOURNAL
10	WRITE	1057	1398391		INODE FREEMAP IN JOURNAL
11	WRITE	1058	1398389		GROUP DESCRIPTOR IN JOURNAL
12	WRITE	1059	1398389		GROUP DESCRIPTOR IN JOURNAL
13	WRITE	1060	1398391	12	INODE IN JOURNAL
14	WRITE	1061	1398391	12	INODE IN JOURNAL
15	WRITE	1062	1398391	2	INODE IN JOURNAL
16	WRITE	1063	1398391	2	INODE IN JOURNAL
17	WRITE	1064	1398633	2	DIR IN JOURNAL
18	WRITE	1065	1398633	2	DIR IN JOURNAL

19	WRITE	1066	1398720		BLOCK FREEMAP IN JOURNAL
20	WRITE	1067	1398720		BLOCK FREEMAP IN JOURNAL
21	WRITE	1068	1398726	8	JOURNAL COMMIT RECORD
22	WRITE	1069	1398726	8	JOURNAL COMMIT RECORD
23	WRITE	4	1398389		GROUP DESCRIPTOR
24	WRITE	5	1398389		GROUP DESCRIPTOR
25	WRITE	518	1398720		BLOCK FREEMAP
26	WRITE	519	1398720		BLOCK FREEMAP
27	WRITE	520	1398391		INODE FREEMAP
28	WRITE	521	1398391		INODE FREEMAP
29	WRITE	522	1398391	2	INODE
30	WRITE	523	1398391	2	INODE
31	WRITE	524	1398391	12	INODE
32	WRITE	525	1398391	12	INODE
33	WRITE	1024	1398633	2	DIR
34	WRITE	1025	1398633	2	DIR
Delete file and sync					
Timestamp	**Action**	**Sector**	**Page ID**	**I-nodes**	**Sector Type**
35	WRITE	1070	1398726	8	JOURNAL DESCRIPTOR
36	WRITE	1071	1398726	8	JOURNAL DESCRIPTOR
37	WRITE	1072	1398633	2	DIR IN JOURNAL
38	WRITE	1073	1398633	2	DIR IN JOURNAL
39	WRITE	1074	1398391	2	INODE IN JOURNAL
40	WRITE	1075	1398391	2	INODE IN JOURNAL
41	WRITE	1078	1398389		EXT3 SUPERBLOCK IN JOURNAL
42	WRITE	1079	1398389		EXT3 SUPERBLOCK IN JOURNAL
43	WRITE	1080	1398391	12	INODE IN JOURNAL
44	WRITE	1081	1398391	12	INODE IN JOURNAL
45	WRITE	1082	1398720		BLOCK FREEMAP IN JOURNAL
46	WRITE	1083	1398720		BLOCK FREEMAP IN JOURNAL
47	WRITE	1084	1398389		GROUP DESCRIPTOR IN JOURNAL
48	WRITE	1085	1398389		GROUP DESCRIPTOR IN JOURNAL
49	WRITE	1086	1398391		INODE FREEMAP IN JOURNAL
50	WRITE	1087	1398391		INODE FREEMAP IN JOURNAL
51	WRITE	1088	1398732	8	JOURNAL COMMIT RECORD
52	WRITE	1089	1398732	8	JOURNAL COMMIT RECORD
53	WRITE	2	1398389		EXT3 SUPERBLOCK
54	WRITE	3	1398389		EXT3 SUPERBLOCK
55	WRITE	4	1398389		GROUP DESCRIPTOR
56	WRITE	5	1398389		GROUP DESCRIPTOR
57	WRITE	518	1398720		BLOCK FREEMAP
58	WRITE	519	1398720		BLOCK FREEMAP

59	WRITE	520	1398391		INODE FREEMAP
60	WRITE	521	1398391		INODE FREEMAP
61	WRITE	522	1398391	2	INODE
62	WRITE	523	1398391	2	INODE
63	WRITE	524	1398391	12	INODE
64	WRITE	525	1398391	12	INODE
65	WRITE	1024	1398633	2	DIR
66	WRITE	1025	1398633	2	DIR
Un-mount the file system					
Timestamp	**Action**	**Sector**	**Page ID**	**I-nodes**	**Sector Type**
67	WRITE	1052	1398393	8	JOURNAL SUPERBLOCK
68	WRITE	1053	1398393	8	JOURNAL SUPERBLOCK
69	DELETE	1060	1398391	8	INODE IN JOURNAL
70	DELETE	1061	1398391	8	INODE IN JOURNAL
71	DELETE	1064	1398633	8	DIR IN JOURNAL
72	DELETE	1065	1398633	8	DIR IN JOURNAL
73	WRITE	1052	1398393	8	JOURNAL SUPERBLOCK
74	WRITE	1053	1398393	8	JOURNAL SUPERBLOCK
75	DELETE	1072	1398633	8	DIR IN JOURNAL
76	DELETE	1073	1398633	8	DIR IN JOURNAL
77	DELETE	1080	1398391	8	INODE IN JOURNAL
78	DELETE	1081	1398391	8	INODE IN JOURNAL
79	WRITE	1052	1398393	8	JOURNAL SUPERBLOCK
80	WRITE	1053	1398393	8	JOURNAL SUPERBLOCK
81	WRITE	2	1398389		EXT3 SUPERBLOCK
82	WRITE	3	1398389		EXT3 SUPERBLOCK

The scan in Table 1 display the processes issued in the file system to create and delete a small file (1K). The first step is mounting the file system. You can see the action write followed by a bit of metadata the sector and page ID's. This describes where the data is being written to. The sector type here shows that we are writing to a "superblock." The superblock is the beginning portion of the file system and describes the file system. It contains descriptions of file system sizes (block sizes, number of inodes, inodes per block etc.), configuration, where items such as inode tables are located and descriptors of available and used memory. There is also copies of the superblock stored throughout memory in case of corruption. The next set of commands creates the file. You see the

data, superblock, inodes, descriptors, and directories (dir) are updated. The next set of commands deletes the file. Again the directories, inodes, and superblocks are all overwritten. Finally, the commands are issued to unmount the file system. The unnecessary directories and inodes are eliminated via overwriting and the superblock is updated.

III. Source Review

There are a number of similar graphical visualization tools that share the common focus of visualizing the file system. However, none of these software tools are capable of visualizing the interactions and requests in a "live capture" form in which processes are displayed as they occur. Below is a list of a few of the similar software along with short descriptions of each product.

The Sleuth Kit (TSK) [12] operates in the computers command line and examines disk images. It is expandable through modules or as being used as a plugin to larger forensics tools. It avoids the operating system to find hidden or deleted files, it allows you to examine disk layout and then extract partitions. It is capable of displaying all data for files such as file attributes, file hashes, and file system and metadata structure details.

Autopsy [2] is a free forensics tool that allows the user to see timestamp data and compile usage reports as to when certain events occurred on the computer, finding corrupt, hidden or dangerous files, searching a computer for keywords, extracting web data (history, bookmarks, cookies etc.), recovering deleted files from unallocated space, extracting metadata from pictures and scanning for threats. The program includes STIX and PhotoRec. Autopsy is the graphical interface to The Sleuth Kit, and it displays the data collected from the TSK commands. Figure 2 shows an example

of the interface. Autopsy is a forensics tool, meaning that it is an investigative software to help investigators collect data that is hidden or was intentionally or unintentionally destroyed. Its purpose is fighting cyber-crime or for criminal investigation but may be used for personal data recovery.

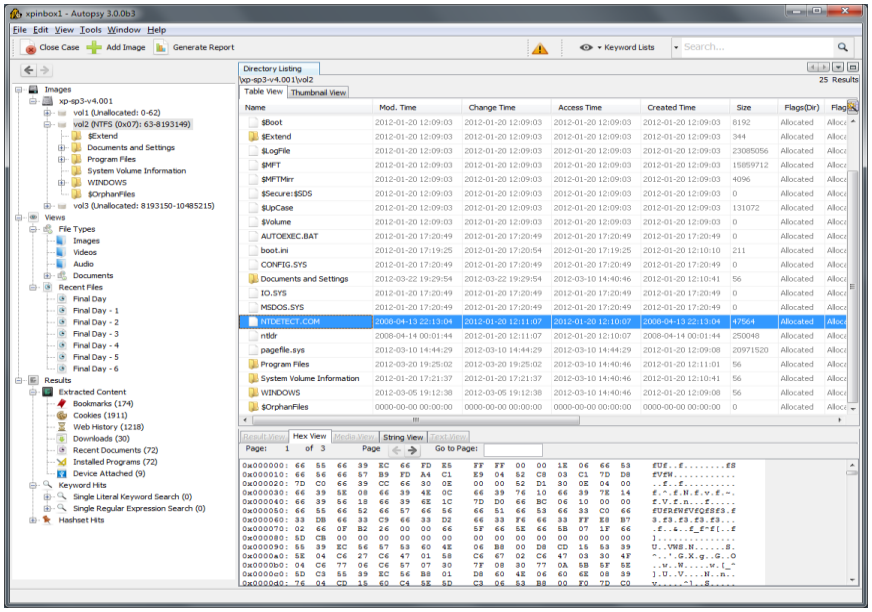


Figure 2: Autopsy Interface

File System Visualizer [6] is an older program that shows the files in directories in a 3D map. It shows where files are stored in the memory by showing directories as base rectangles with files viewed as blocks within the directory. The blocks are proportionate in size to their file size either in height or area depending on the view. There is also a 2D directory tree/file list interface that lays out the directories in a tree configuration to visualize sub-directories. Figure 3 shows the output visualization of the data storage.

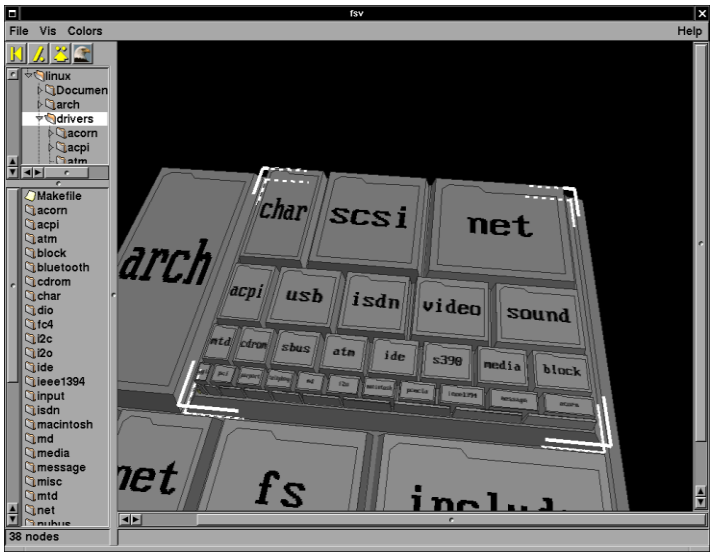


Figure 3: FSV Interface

Windows Directory Statistics (WDS) [15] is a program that scans the directory and creates a directory list, tree map, and an extension list. It displays a visual representation of the memory usage. The directory list and tree map visually display the files and directories while the extension list acts as a descriptor to show details about the system and its components. Figure 4 shows how the data is displayed in visual form. Each rectangle is a different file that is made proportionate in size to its file size. In the same way directories are formed out of these rectangles and include sub directories and other files. The colors represent file type.



Figure 4: WDS Interface

As detailed above, there is a variety of software that is designed to visualize the file system in its current state. Detailed below are a number of applications used specifically for education [8].

Alg_OS [9] is an educational tool designed for teaching students memory management and page replacement. Students are able to make changes and watch results in real time. Student data is then recorded for teaching purposes. The system is web based and is capable of creating models of memory that can be “accessed” using algorithms and commands.

Another approach to operating system education is the use of virtual machines [10]. Virtual machines can be combined with versioning systems (versioning systems track changes to work and allow collaboration for group members or teachers). Demonstrating techniques live on such a system can create a useful model of the operating system and kernel. Students are able to make changes without harming the outside system and teachers are able to observe what changes students make through versioning systems or shared access to the virtual machines.

Pintos [11] is a full educational operating system that comes with documentation and instructions for projects that can be carried out on the system. Students are able to interact with the system on actual hardware giving them full access to its components (scheduler, process manager etc.). Pintos was designed solely for the purpose of education.

Yet another approach to operating system education is the use of simulators such as SOsim [1]. SOsim creates a small basic operating system which includes a process manager, scheduler and

paged virtual memory. It uses a multi-programmed approach to teach users how the operating system components interact. Its basic design makes it easier to understand and use and its visual interface helps students visualize the ideas being taught.

After examining these tools, we have found that no software exists that would allow users to visualize what the file system is currently doing and how it is doing it. This is why we intend to create such a program to see this data in real time.

IV. Design and Methodology

This project was an extension of work done by Dr. Diesburg for her dissertation while attending Florida State University. Dr. Diesburg had created an application that worked with the file system to gather information about the procedures carried out by the file system to store, read and delete data. Her goal for this research was to work with secure deletion methods to increase security of data that is deleted by the user. Figure 5 is a diagram taken from Dr. Diesburg's dissertation work [5] that shows how her application gathers data. The application interfaces with the file system and gathers data including time stamps, drive operations, sector numbers, transaction numbers, memory pages, inode numbers, a description and the data that is being read, written or deleted. The data was then outputted into a spreadsheet for examination such as in Table 1. This information has many applications for use, but the application we chose to pursue is educational in nature.

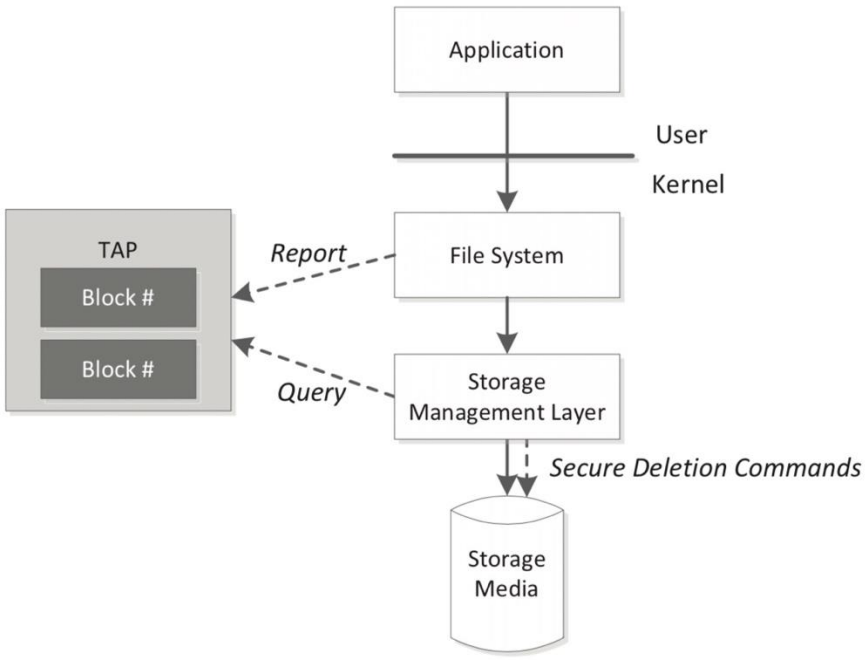


Figure 5: The basic framework of Dr. Diesburg's application

The 2018 spring semester was devoted to researching a number of aspects of the project. The first goal was to make sure a program did not already exist that did the work we were trying to do. We found many similar programs as cited in the literature review but none that implemented our purpose of an educational tool that displayed live file system data.

The next stage of our research was on possible implementation languages and platforms. We decided that we wanted to pursue creating an easy-to-access webpage and programming in HTML5 with as little Java Script as possible to fill in the gaps. HTML5 is the newest addition to the html language family. HTML5 was not a language I was familiar with prior to this project. Despite the steep learning curve we were able to develop a basic knowledge of the syntax and semantics of the language prior to the start of designing the application.

In our initial meetings to discuss possible designs for our program, a few necessary requirements were laid out. Due to the educational nature of the application, the tool needed to be easy enough for students and teachers to be able to quickly learn and use. The application would also need to be laid out in such a way that information could be easily accessed and easy to read. Finally, the goal was for the application to provide real time feedback to user commands such that the results could be viewed as they are processed by the file system.

For the original design, we knew we wanted to create a visual interface for Dr. Diesburg's application. This would entail three parts. First, we would need to receive information from Dr. Diesburg's process running on a remote server. Second, we would need to parse this information and place it in table form. Finally, we would need to display the table. Based on this implementation, one background process and two visual portions were needed. The initial design, as seen in Figure 6, began with a basic window interface that works in much the same way as any other program on a computer. Within the window we needed a series of buttons to control the connection to the server with Dr. Diesburg's process.

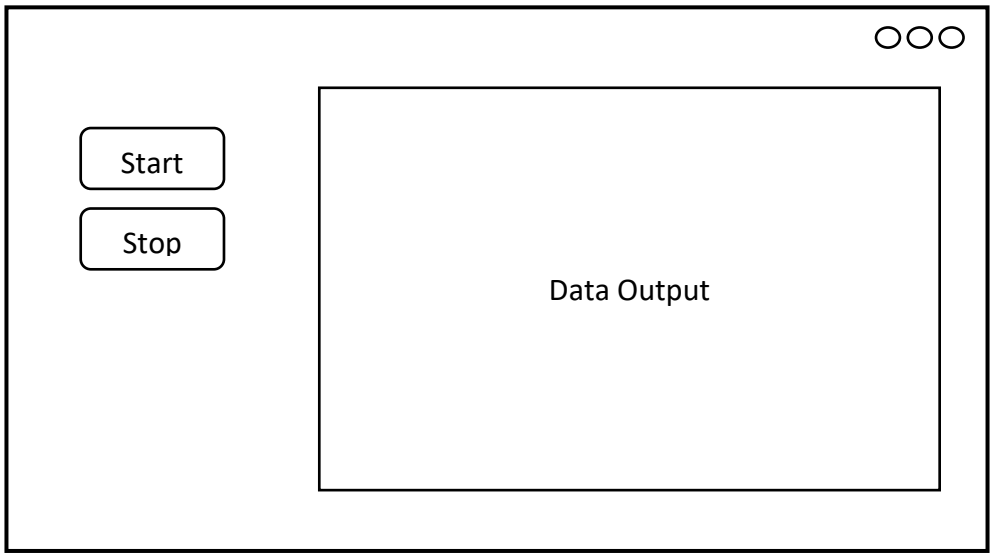


Figure 6: Design #1

We initially had two buttons: start (connect to the server and begin receiving data), and stop (stop receiving data). Finally, we added an area for the final output.

Our initial plan was to reuse open source code from a similar software called WireShark [16]. WireShark is a packet sniffing software that captures data being passed through the network and displays it to the user. The software captures all basic components of the packets including destinations, sources, packet contents etc. (similar to metadata in a file system). Figure 7 shows the Wireshark interface. The interface shows the captured packets in real time. In our program, these are equivalent to the data traces found in Table 1.

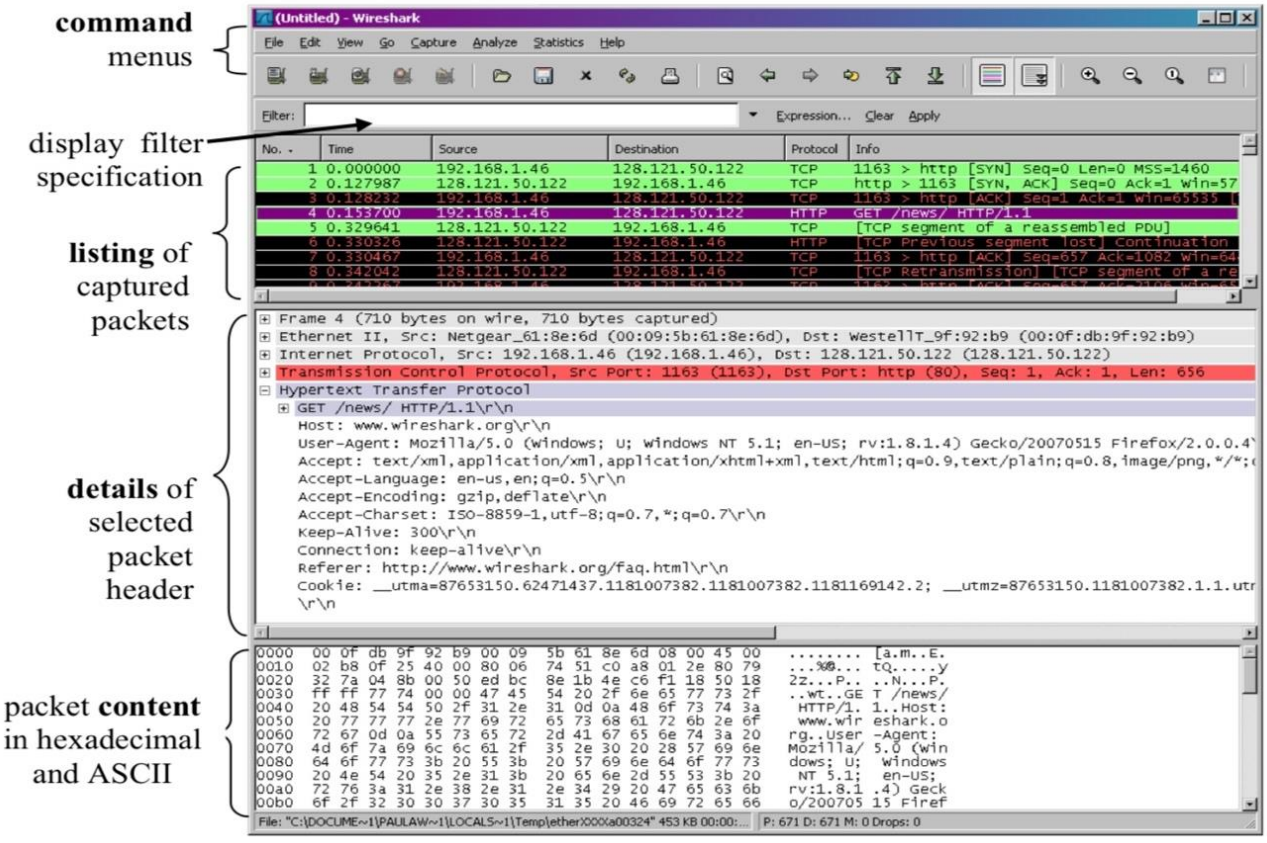


Figure 7: Wireshark Interface

In Figure 7, we also see that the user can click on a packet and see an expanded set of details. In the case of Fileshark, this would be similar to the metadata and the data that is being read or written. With this basis, we planned on simplifying the interface slightly while displaying similar output and keeping basic functionality.

Our next design focused on combining the two actions of inputting commands to create the output and displaying the output. In order to do so, two interfaces were needed. We would need an interface for displaying output visually and an interface for inputting commands. The commands entered cause actual file system calls to the file system which in turn correlate to the data output by the program. For this reason, we shifted our focus to the design in Figure 8.

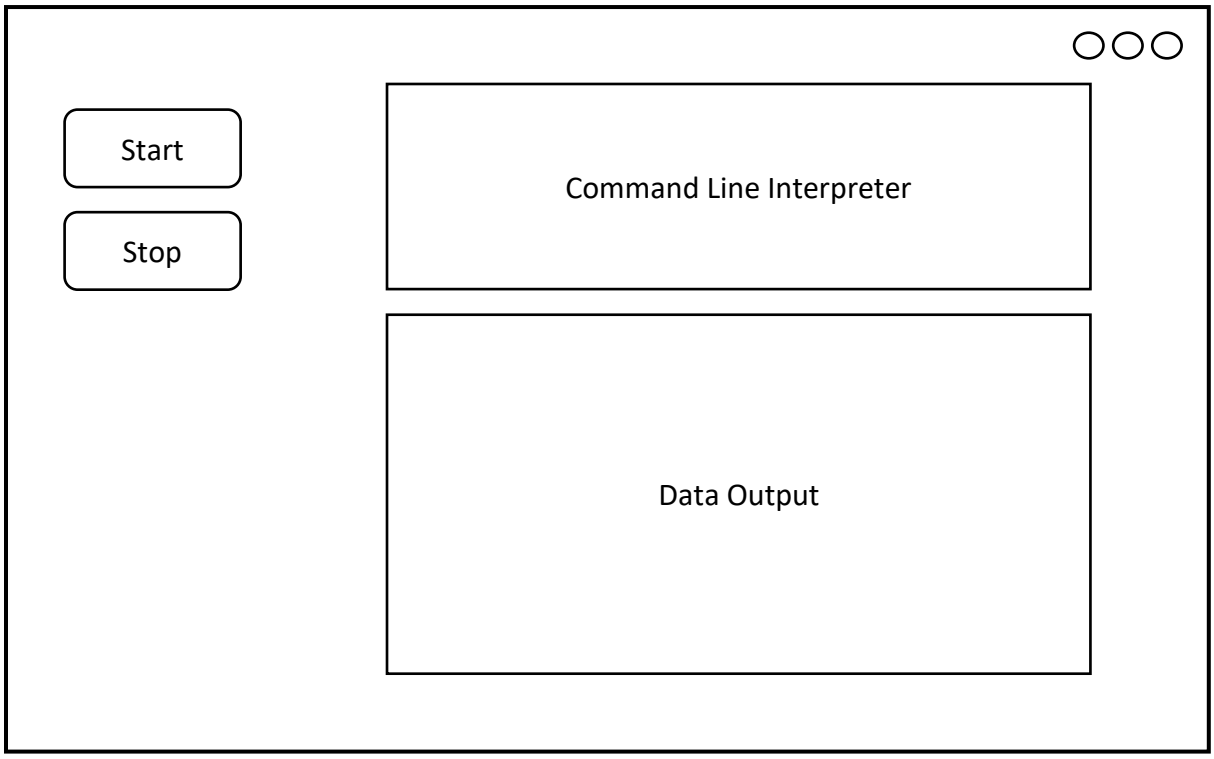


Figure 8: Design #2

In the second design, we kept the same basic interface as the first design, but added a command line interface that would allow the user to enter the commands and see the results in the same window. Unfortunately, due to time constraints and a steep learning curve, we had to eliminate the command line interface to focus on core features of the program. After this shift in focus we returned to our more basic original design and focused more on the user interface as opposed to the backend processes.

Figure 9 shows the final interface for the application. As you can see, we kept the original start and stop buttons but relabeled them connect and disconnect. These buttons allow us to connect to the server and keep it open while starting and stopping the output. When we are finished we can disconnect and clear the table using the disconnect button and by refreshing the webpage. The table output is formatted in CSS to be simple to read and easy to understand. We decided to use a browser window as our basic window due to the flexibility of implementation as explained in the next section.

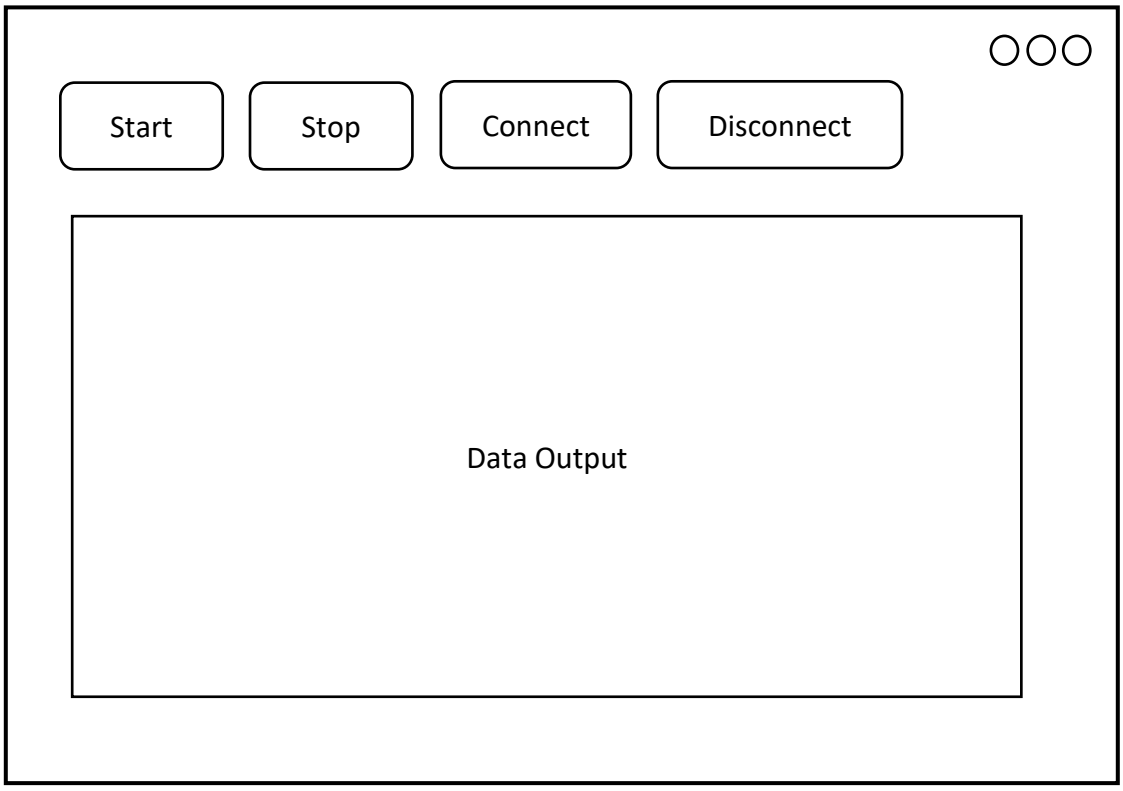


Figure 9: Final Design

Figure 10 shows a diagram describing the interactions that occur within the Fileshark framework. The base setup for the application is a webserver provided by Dr. Diesburg running Apache (a web server) [7]. The server contains both the code for the front-end web page and the virtual machine running the file tracing code.

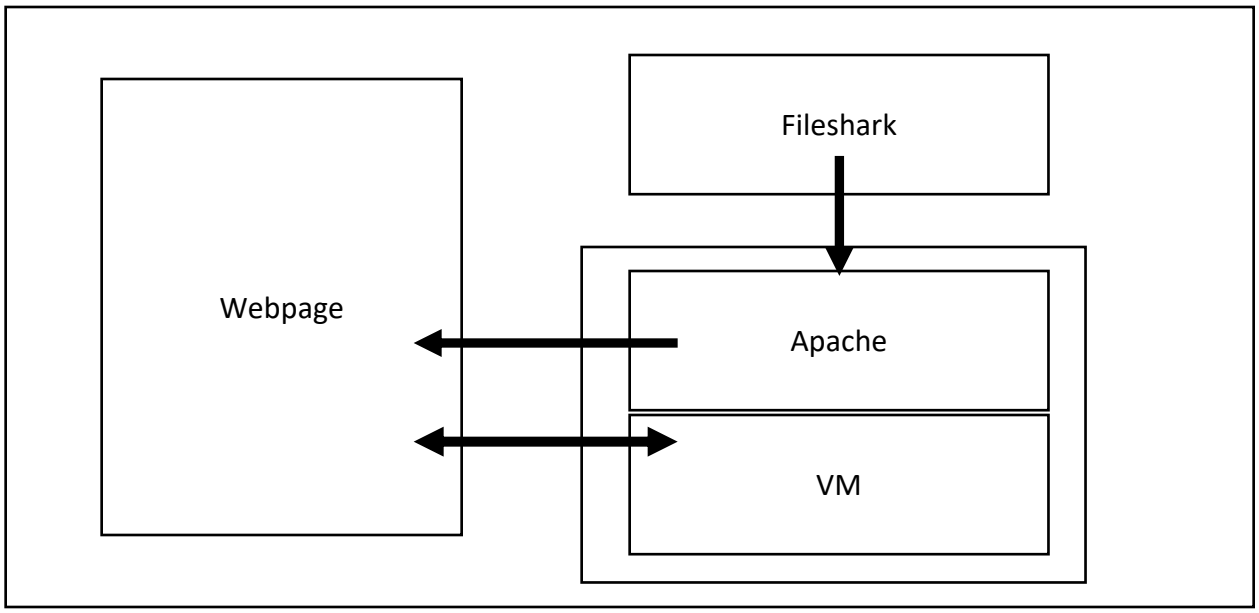


Figure 10: The Fileshark Framework

V. Implementation

While the program was in development and testing, we configured the Apache htaccess to require a login to access the web interface. However, due to the environment in which the webpage resides, the interface is accessible from any machine with Internet access when logged into the webpage. Using Apache, I configured the index.html file that Apache uses to display web content to run the code found in Appendix B. Apache sends the code to the webpage where it is interpreted. Figure 11 shows the Fileshark interface running in a browser window.

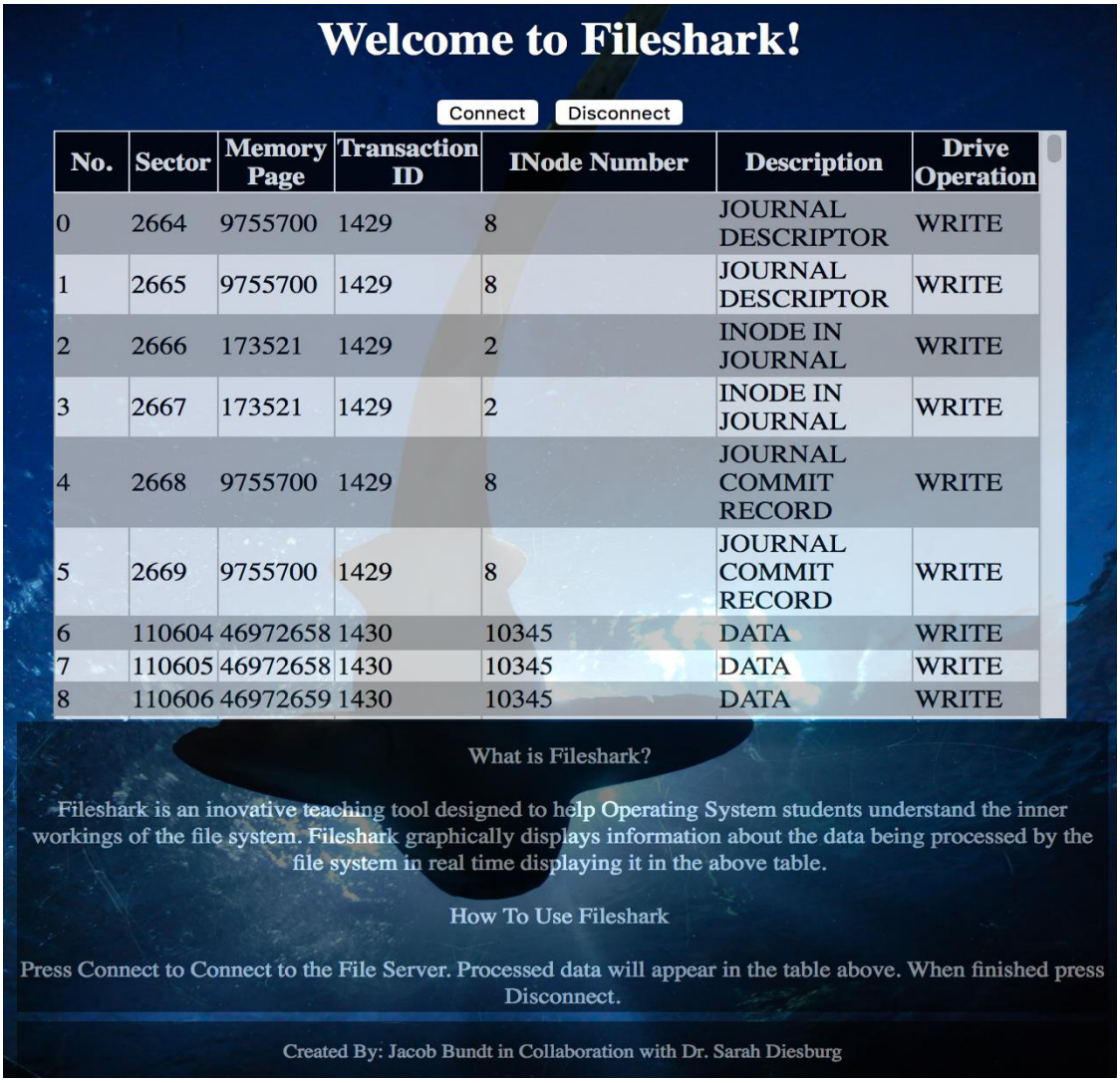


Figure 11: Final Webpage

At the top of the page, just below the title, the “Connect” and “Disconnect” buttons are visible. Pressing the “Connect” button establishes a websocket connection to the virtual machine. Disconnect tears down this connection and stops new data from being received. Below the buttons is the table interface. The table headers each describe a different piece of metadata that is contained in the file system call (more information about the headers is available in Appendix A). The table is scrollable to simplify the user experience. At the bottom of the page is a short description of Fileshark and a brief usage tutorial.

VI. Future Work

To improve usability and functionality, a few more pieces could be added to the program. The goal is to add filtering and sort functionality so that users can search for specific types of data. Currently, the data field is far too large to display normally. This can be adjusted for by creating a hidden attribute tag for the data field. An option could be added to download the captured trace into a file that could be saved and reloaded.

Finally, the ultimate goal is to have this application tested in the classroom as a teaching tool. Curriculum must be designed for use with this tool to teach a file system module in a traditional computer science undergraduate course. Partner schools will be selected, with a group of operating system classes taught with and without this tool. Finally, learning will be evaluated via the same exam questions throughout both the classes that use the Fileshark tool and the control classrooms.

VII. Conclusion

At the end of the 2018 Fall semester, the alpha version of the Fileshark program has been completed and Fileshark is ready to begin testing in a classroom environment. Functionality that is currently supported by Fileshark includes parsing of incoming data, control over data connections, and basic user interfaces for data display. The final interface is shown in Figure 11. Through the process of Fileshark's creation, I learned a number of important skills: namely, programming in two new languages (HTML5 and JavaScript), basic networking skills including setting up an Apache webserver and the use of websockets, website design practices, experience with using virtual machines, and finally, a better understanding of the file system.

In conclusion, during the past two semesters, we have created a functional tool that can be used by educators to better teach students the internal workings of the file system. The tool consists of back-end code that can be used to create file system traffic and a web-based user interface that translates the file system traffic into an easy to read visual format. Fileshark is capable of reading and displaying file system data in a way that enables visual learning experiences that were previously not possible. This tool will not only help students learn in a visual way the internal processes of the file system, it will also help teachers improve their ability to teach abstract file system ideas, system forensic analysts to track how computers are storing data, and system security experts to track file system operations. Finally, Fileshark will continue to be expanded to meet user needs and to create a more detailed analysis of the processes it scans.

Appendix A

Appendix A consists of an XML trace that simulates what the Fileshark interface receives from the file trace software running on the back end. Each message sent to Fileshark is contained within a <rec> or record tag. The other tags in each message contain data as follows.

<ts>	Time Stamp shows when processes are carried out
<driveop>	Drive Operation shows what operation is being carried out on the drive
<sect>	Sector Number shows which sector of the memory is being accessed
<mempage>	Memory Page shows which memory page information is stored on
<trans>	Transaction ID shows the contained transaction number
<inodenum>	Inode Number shows which index number points to the metadata for the process
<desc>	Description shows a what part of the file system is being accessed
<data>	Data contains the data for each process

These tags are parsed by Fileshark and are added to the Output Table.

<rec><ts>0</ts><driveop>WRITE</driveop><sect>2</sect><mempage>2243171</mempage>
<trans>0</trans><inodenum>0</inodenum><desc> EXT3 SUPERBLOCK</desc><data>NULL
512-byte field</data></rec>
<rec><ts>0.01</ts><driveop>WRITE</driveop><sect>3</sect><mempage>2243171</mempag
e><trans>0</trans><inodenum>0</inodenum><desc>EXT3
SUPERBLOCK</desc><data>NULL 512-byte field</data></rec>
<rec><ts>0.02</ts><driveop>WRITE</driveop><sect>1052</sect><mempage>2243175</mem
page><trans>0</trans><inodenum>8</inodenum><desc> JOURNAL
SUPERBLOCK</desc><data>NULL 512-byte field</data></rec>
<rec><ts>0.03</ts><driveop>WRITE</driveop><sect>1053</sect><mempage>2243175</mem
page><trans>0</trans><inodenum>8</inodenum><desc> JOURNAL
SUPERBLOCK</desc><data>NULL 512-byte field</data></rec>
<rec><ts>0.04</ts><driveop>WRITE</driveop><sect>1054</sect><mempage>2243175</mem
page><trans>39</trans><inodenum>8</inodenum><desc> JOURNAL
DESCRIPTOR</desc><data>NULL 512-byte field</data></rec>
<rec><ts>0.05</ts><driveop>WRITE</driveop><sect>1055</sect><mempage>2243175</mem
page><trans>39</trans><inodenum>8</inodenum><desc> JOURNAL
DESCRIPTOR</desc><data>NULL 512-byte field</data></rec>
<rec><ts>0.06</ts><driveop>WRITE</driveop><sect>1056</sect><mempage>2243173</mem
page><trans>39</trans><inodenum>2</inodenum><desc> INODE IN
JOURNAL</desc><data>NULL 512-byte field</data></rec>
<rec><ts>0.07</ts><driveop>WRITE</driveop><sect>1057</sect><mempage>2243173</mem
page><trans>39</trans><inodenum>2</inodenum><desc> INODE IN
JOURNAL</desc><data>NULL 512-byte field</data></rec>
<rec><ts>0.08</ts><driveop>WRITE</driveop><sect>1058</sect><mempage>2243173</mem
page><trans>39</trans><inodenum>0</inodenum><desc> INODE FREEMAP IN
JOURNAL</desc><data>NULL 512-byte field</data></rec>
<rec><ts>0.09</ts><driveop>WRITE</driveop><sect>1059</sect><mempage>2243173</mem
page><trans>39</trans><inodenum>0</inodenum><desc> INODE FREEMAP IN
JOURNAL</desc><data>NULL 512-byte field</data></rec>
<rec><ts>0.1</ts><driveop>WRITE</driveop><sect>1060</sect><mempage>2243171</memp
age><trans>39</trans><inodenum>0</inodenum><desc> GROUP DESCRIPTOR IN
JOURNAL</desc><data>NULL 512-byte field</data></rec>
<rec><ts>0.11</ts><driveop>WRITE</driveop><sect>1061</sect><mempage>2243171</mem
page><trans>39</trans><inodenum>0</inodenum><desc> GROUP DESCRIPTOR IN
JOURNAL</desc><data>NULL 512-byte field</data></rec>
<rec><ts>0.12</ts><driveop>WRITE</driveop><sect>1062</sect><mempage>2243173</mem
page><trans>39</trans><inodenum>12</inodenum><desc> INODE IN
JOURNAL</desc><data>NULL 512-byte field</data></rec>
<rec><ts>0.13</ts><driveop>WRITE</driveop><sect>1063</sect><mempage>2243173</mem
page><trans>39</trans><inodenum>12</inodenum><desc> INODE IN

Appendix B

Appendix B contains the code for Fileshark. Within the code comments describing the code are denoted by <!—Comment-->. Fileshark is written in HTML5 with a few portions in Javascript.

```
<html>

<!--Style the Webpage-->

   <style>
      header {
              color:#ffffff;
              font-size: 90%;
      }
      body {
              background-color: #000000;
              background-image: url(https://images.pexels.com/photos/67854/shark-fish-hammerhead-shark-aquarium-67854.jpeg);
              background-size: 100% 100%;
      }
      section {
              color: #ffffff;
              background-color: #000000;
              opacity: 0.5;
              position: absolute;
              width: 98%;
              height: 185px;
              bottom: 45px;
              font-size: 85%;
      }
      table {
              border-collapse: collapse;
              opacity: 0.8;
      }
      th {
              background-color:#000000;
              color:#ffffff;
              border: 1px solid #ffffff;
      }

      tr:nth-child(even) {
              background-color:#bbbbbb;
      }
      tr:nth-child(odd) {
              background-color:#ffffff;
      }
      td {
              border: 1px solid #bbbbbb;
      }
      tr:hover {
              background-color:#222222;
              color:#ffffff;
      }
      table tbody {
              display: block;
              height: 375px;
              width: 100%;
              overflow: auto;
      }
      footer {
              color:#ffffff;
              background-color:#000000;
              opacity:0.5;
              position: absolute;
              width: 98%;
              bottom: 1px;
              font-size: 75%;
      }
   </style>
```

```
!--The Title is Used in the Tab Menu and Bookmarks-->
   <head>
       <title>FileShark</title>
   </head>

!--A Header for the Webpage-->

   <header align="center">
       <h1>Welcome to Fileshark!</h1>
   </header>

!--These Commands Create the Connect and Disconnect Buttons and Assign their Functions to Carry Out When they are Clicked-->
   <body onload="connect()" align="center">
       <button onclick='connect()'>Connect</button>
       <button onclick="disconnect()">Disconnect</button>

!--These Calls Build the Output Table-->

       <table id="outputTable" align="center">
           <tr>
               <th>No.</th>
               <th>Sector</th>
               <th>Memory Page</th>
               <th>Transaction ID</th>
               <th>INode Number</th>
               <th>Description</th>
               <th>Drive Operation</th>
           </tr>
       </table>

       <script language="javascript" type="text/javascript">

<!--The Code in this Section Describes the Creation of the Websockets-->

               var wsUri = "ws://diesburg.cs.uni.edu:1030";
               var counter = -1;

<!--connect() When Called Opens a Websocket to diesburg.cs.uni.edu on Port 1030 and Displays a Connection Established Message-->

               function connect() {
                       websocket = new WebSocket(wsUri);
                       websocket.onopen = function(evt) {
                               alert('Connection Established');
                       };

                       <!--On Each Message Received Call parseXML to Add the Data to the Output Table-->

                       websocket.onmessage = function(evt) {
                               counter ++;
                               parseXML(evt.data);
                       };

                       <!--When the Websocket is Closed Notify the User-->

                       websocket.onclose = function(evt) {
                               alert("Disconnected");
                       };
               }

               <!--The Websocket also has the Capability of Sending Messages Back to the Server if Neccessary-->

               function sendMessage(message) {
                       alert("Sent: " + message);
                       websocket.send(message);
               }

               <!--When Called disconnet() Takes Down the Websocket-->

               function disconnect() {
                       websocket.close();
               }


<!--parseXML(xml) Takes in an XML String and Add's it to the Output Table as Table Data-->

               function parseXML(xml) {

                       <!--Create a DOM Parser to read in the XML and Add a Row to the Output Table-->

                       var parser, xmldoc, table;
                       table = document.getElementById("outputTable");
                       parser = new DOMParser();
                       xmlDoc = parser.parseFromString(xml, "text/xml");
                       var row = table.insertRow();
                       const counterIn = counter;

                       <!--Add Cells to Each Row-->

                       var count = row.insertCell(0);
                       var sect = row.insertCell(1);
                       var mempage = row.insertCell(2);
                       var trans = row.insertCell(3);
                       var inodenum = row.insertCell(4);
                       var desc = row.insertCell(5);
                       var driveop = row.insertCell(6);

                       <!--Add Data Pulled From XML to Each Cell-->

                       count.innerHTML = counterIn;
                       sect.innerHTML = xmlDoc.getElementsByTagName("sect")[0].childNodes[0].nodeValue;
```

```
                    mempage.innerHTML = xmlDoc.getElementsByTagName("mempage")[0].childNodes[0].nodeValue;
                    trans.innerHTML = xmlDoc.getElementsByTagName("trans")[0].childNodes[0].nodeValue;
                    inodenum.innerHTML = xmlDoc.getElementsByTagName("inodenum")[0].childNodes[0].nodeValue;
                    desc.innerHTML = xmlDoc.getElementsByTagName("desc")[0].childNodes[0].nodeValue;
                    driveop.innerHTML = xmlDoc.getElementsByTagName("driveop")[0].childNodes[0].nodeValue;
                }
        </script>
    </body>

<!--Text Section Describing Fileshark and How to Use It-->

    <section align="center">
        <p>
            What is Fileshark?
            <br>
            <br>
            Fileshark is an inovative teaching tool designed to help Operating System students understand the inner
            workings of the file system.Fileshark graphically displays information about the data being processed
            by the file system in real time displaying it in the above table.
            <br>
            <br>
            How To Use Fileshark
            <br>
            <br>
            Press Connect to Connect to the File Server. Processed data will appear in the table above. When finished press Disconnect.
        </p>
    </section>


<!--Footer Citing Creators-->

    <footer align="center">
        <p>Created By: Jacob Bundt in Collaboration with Dr. Sarah Diesburg</p>
    </footer>

</html>
```

Sources Cited

[1] Luiz Paulo Maia, Francis Berenger Machado, and Ageu C. Pacheco, Jr.. 2005. A constructivist framework for operating systems education: a pedagogic proposal using the SOsim. In *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education* (ITiCSE '05). ACM, New York, NY, USA, 218-222.

[2] Autopsy: https://www.sleuthkit.org/autopsy/. Accessed: 2018-03-20.

[3] Carrier, B. 2005. *File System Forensic Analysis*. Addison Wesley Professional.

[4] Diesburg, S.M. Per-File Full-Data-Path Secure Deletion for Electronic Storage. (2012).

[5] Diesburg, S., Meyers, C., Stanovich, M., Wang, A.-I.A. and Kuenning, G. 2016. Trueerase: Leveraging an auxiliary data path for per-file secure deletion. *ACM Transactions on Storage* (TOS). 12, 4 (2016), 18.

[6] fsv – 3D File System Visualizer: http://fsv.sourceforge.net/. Accessed: 2018-03-20.

[7] The Apache HTTP Server Project. (n.d.). Retrieved December 9, 2018, from https://httpd.apache.org/

[8] Garmpis, A. and Gouvatsos, N. 2012. Innovative teaching methods in Operating Systems: The Linux case. (Oct. 2012).

[9] Garmpis Aristogiannis 2010. Alg_OS—A web-based software tool to teach page replacement algorithms of operating systems to undergraduate students. *Computer Applications in Engineering Education*. 21, 4 (Nov. 2010), 581–585.

[10] Laadan, O., Nieh, J. and Viennot, N. 2010. Teaching operating systems using virtual appliances and distributed version control. (2010), 480.

[11] Love, R. 2010. *Linux Kernel Development, A thorough guide to the design and implementation of the Linux Kernel*. Addison Wesley Professional.

[12] Pfaff, B., Romano, A. and Back, G. 2009. The Pintos Instructional Operating System Kernel. *Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2009), 453–457.

[13] The Sleuth Kit: https://www.sleuthkit.org/sleuthkit/. Accessed: 2018-03-20.

[14] Tweedie, S. 2000. Ext3, journaling filesystem. *Ottawa Linux Symposium* (2000), 24–29.

[15] WinDirStat: https://windirstat.net/. Accessed: 2018-03-20.

[16] Wireshark · Go Deep.: https://www.wireshark.org/. Accessed: 2018-03-28.