

## Worcester Polytechnic Institute Digital WPI

---

Major Qualifying Projects (All Years)

Major Qualifying Projects

---

March 2015

# TAR Browser: TAR Archives as File Systems

Kyle P. Davidson  
*Worcester Polytechnic Institute*

Tyler Orrin Morrow  
*Worcester Polytechnic Institute*

Follow this and additional works at: <https://digitalcommons.wpi.edu/mqp-all>

---

### Repository Citation

Davidson, K. P., & Morrow, T. O. (2015). *TAR Browser: TAR Archives as File Systems*. Retrieved from <https://digitalcommons.wpi.edu/mqp-all/2117>

This Unrestricted is brought to you for free and open access by the Major Qualifying Projects at Digital WPI. It has been accepted for inclusion in Major Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact [digitalwpi@wpi.edu](mailto:digitalwpi@wpi.edu).



# WPI

## *TAR Browser: TAR Archives as File Systems*

A Major Qualifying Project Report

Submitted to the Faculty of

**Worcester Polytechnic Institute**

In partial fulfillment of the requirements for the

**Degree of Bachelor of Science**

Submitted on March 6, 2015

**Written by**

Kyle Davidson

Tyler Morrow

**Advised and Approved by**

Professor Michael Ciaraldi

## **Abstract**

The goal of the TAR Browser project was to develop a system to better extract single members from compressed TAR archives. To do so, the program must analyze a TAR archive (uncompressed or compressed with XZ or Bzip2 compression) and later use information from this analysis to extract a single desired member without decompressing much else. For many companies and users, this program can vastly improve efficiency when working with large TAR archives.

# Table of Contents

|  |     |
|--|-----|
| Abstract .....   | i   |
| Table of Contents .....                                    | ii  |
| List of Figures .....                                      | iii |
| 1. Introduction .....                                      | 1   |
| 2. Background .....  | 3   |
| 3. Development Process .....                               | 5   |
| 3.1. Initial Research and Planning .....                   | 5   |
| 3.2. Uncompressed TAR Analysis and Extraction .....        | 6   |
| 3.3. MySQL Database Implementation .....                   | 8   |
| 3.4. BZip2 Analysis and Extraction .....                   | 9   |
| 3.5. XZ Analysis and Extraction.....                       | 10  |
| 3.6. FUSE Research and Development .....                   | 11  |
| 4. Functionality .....                                     | 14  |
| 4.1. Database Preparation.....                             | 14  |
| 4.2. Archive Analysis.....                                 | 14  |
| 4.3. TAR Browser Filesystem .....                          | 16  |
| 5. Evaluation .....  | 18  |
| 5.1. Testing Procedures .....                              | 18  |
| 5.2. Accuracy Tests .....                                  | 18  |
| 5.3. Accuracy Test Results .....                           | 18  |
| 5.4. Efficiency and Performance Tests .....                | 19  |
| 5.5. Efficiency and Performance Test Results .....         | 20  |
| 5.6. Conclusions .....                                     | 24  |
| 5.7. Concerns.....   | 25  |
| 6. Future Work.....  | 28  |
| 6.1. Public Open Source Release .....                      | 28  |
| 6.2. XZ Support Maintenance.....                           | 28  |
| 6.3. Improvements and Expansion.....                       | 29  |
| 6.4. Final Thoughts .....                                  | 29  |
| Appendices .....   | 30  |
| Appendix A: TAR Browser Installation and Usage Guide ..... | 30  |
| Appendix B: TAR Browser SQL Database.....                  | 34  |
| Appendix C: Raw Data From Testing .....                    | 37  |
| References .....   | 42  |

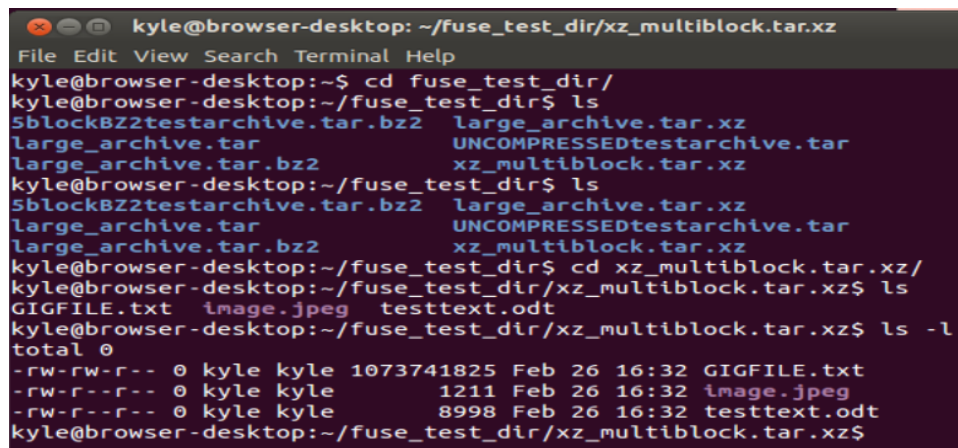
## List of Figures

|   |    |
|---|----|
| Figure 1.1: TAR Browser in action, navigating directories within an XZ archive.....                         | 1  |
| Figure 3.1: Output of the analysis program.....   | 7  |
| Figure 3.2: Exploring the contents of a TAR XZ archive .....  | 12 |
| Figure 5.1: Results of running “diff” on a file within several archives compared to the original file ..... | 19 |
| Table 5.1: Uncompressed TAR access times:.....  | 21 |
| Table 5.2: BZip2 Compressed TAR access times .....  | 22 |
| Table 5.3: XZ Compressed TAR access times .....   | 23 |
| Table 5.4: Uncompressed TAR access times (Subset of tests on second machine).....                           | 26 |
| Table 5.5: XZ Compressed TAR access times (Subset of tests on second machine) .....                         | 26 |

# 1. Introduction

The TAR Browser system aims to improve efficiency and performance when extracting members from large<sup>1</sup> TAR archives compressed with either XZ or BZip2 compression. This is useful in many development environments, particularly for companies which keep large nightly builds or backups. Oftentimes, developers will have to open older archives and extract a single member to work with. Compression formats such as Bzip2 and XZ do not natively support file indexing; thus, when decompressing a compressed archive to extract a particular file, TAR will decompress the entire archive in order to extract this one member, which can be very time consuming and CPU intensive. The goal of the TAR Browser system is to eliminate this wasted time by pulling out and decompressing only the blocks of the archive on which the desired file exists.

TAR Browser consists of three separate Linux programs, written in C, which share a MySQL database. The Archive Analysis program explores an input TAR archive (uncompressed, or compressed with BZip2 or XZ) and stores relevant information in a database. This database is first created through MySQL commands, then populated with tables through a small database preparation program. This program tests database access and creates the necessary tables, and is discussed in Appendix B. The client program allows the user to explore the archive as if it were a mounted directory, open read-only copies of files, and extract desired files efficiently. This is done through FUSE, a filesystem in userspace [Oseberg]. The FUSE system queries the database, utilizing the information for each archive and member in order to present the contents of the input archive as a read-only filesystem.

A terminal window titled 'kyle@browser-desktop: ~/fuse\_test\_dir/xz\_multiblock.tar.xz' showing a series of commands and their outputs. The user navigates from the root of the archive to a subdirectory and lists its contents. The terminal output is as follows:

```
kyle@browser-desktop: ~/fuse_test_dir/xz_multiblock.tar.xz
File Edit View Search Terminal Help
kyle@browser-desktop:~$ cd fuse_test_dir/
kyle@browser-desktop:~/fuse_test_dir$ ls
5blockBZ2testarchive.tar.bz2  large_archive.tar.xz
large_archive.tar             UNCOMPRESSEDtestarchive.tar
large_archive.tar.bz2        xz_multiblock.tar.xz
kyle@browser-desktop:~/fuse_test_dir$ ls
5blockBZ2testarchive.tar.bz2  large_archive.tar.xz
large_archive.tar             UNCOMPRESSEDtestarchive.tar
large_archive.tar.bz2        xz_multiblock.tar.xz
kyle@browser-desktop:~/fuse_test_dir$ cd xz_multiblock.tar.xz/
kyle@browser-desktop:~/fuse_test_dir/xz_multiblock.tar.xz$ ls
GIGFILE.txt  image.jpeg  testtext.odt
kyle@browser-desktop:~/fuse_test_dir/xz_multiblock.tar.xz$ ls -l
total 0
-rw-rw-r-- 0 kyle kyle 1073741825 Feb 26 16:32 GIGFILE.txt
-rw-r--r-- 0 kyle kyle 1211 Feb 26 16:32 image.jpeg
-rw-r--r-- 0 kyle kyle 8998 Feb 26 16:32 testtext.odt
kyle@browser-desktop:~/fuse_test_dir/xz_multiblock.tar.xz$
```

Figure 1.1: TAR Browser in action, navigating directories within an XZ archive

<sup>1</sup> In this paper, a large archive refers to a TAR archive of approximately 20 to 40 GB uncompressed.

This documentation will first cover the background details of this project, such as the origin of the problem, early decisions, and use cases. The development process will then be explored in more detail, going through the accomplishments, problems, and insights of each stage of development. This is followed by the functionality and technical details of the finalized TAR Browser system, as well as the accuracy and performance testing procedures. Final evaluations, as well as potential future work, will conclude this documentation.

Appendix A consists of an installation guide for TAR Browser. The TAR Browser source code can be found at the following Github repository:

<https://github.com/tomorrow-nf/tar-as-filesystem>

## 2. Background

The problem of slow file extraction with large TAR archives was posed by Sean Davidson, the storage development principal engineer at Dell in Nashua, New Hampshire and Kyle Davidson's father. The problem he presented is as follows: "To store static data efficiently and also be transportable across systems, compressed TAR files are used quite often. Accessing and extracting individual files inside a compressed TAR file can be time consuming and use precious file system resources, especially if they are very large compressed tar files consisting of tens of gigabytes." TAR Browser was developed to resolve this problem, but is not specifically designed for use within Dell, and is not a sponsored project.

Many users prefer to use the TAR archive format paired with some form of compression rather than creating a zipfile. This is because a zipfile is made for easy random access but has a lower compression ratio. When a zipfile is created, each file that it includes is compressed individually, meaning that an archive of many small files is barely compressed because there may not be many patterns in a single file. TAR utilized with a compression method such as BZip2 or XZ handles this by creating a single large file out of its input files. Any compression routine run on a TAR archive has a large amount of data to look at for compression opportunities. Since the compression format sees its input as a single file, no file markers or other flags exist in the compressed stream to point out where the members of the TAR archive exist, meaning the archive must be decompressed before its contents can be viewed.

There are many types of compression formats applicable to TAR archives. In addition to handling uncompressed TAR archives, TAR Browser supports BZip2 and XZ. These two compression formats were chosen primarily because of their widespread use and usage of more modern algorithms [Ellingwood]. Initially, GZip support was planned as well, but this was dropped early on in the development process in favor of more commonly used formats due to time constraints. Potential support for other compression methods such as GZip will be discussed under Future Work.

Basic use cases were drafted early on in this project. Users should be able to explore TAR archives as if they were directories, open read-only members within these archives, and extract a copy of a member. The extracted member must be identical to the actual member within the archive, including details such as file permissions. Users cannot modify anything within the TAR archive, such as inserting new members (including those which have been extracted from the archive, then edited), moving member locations, and deleting or creating new members and directories. This functionality is similar to that of GNOME's Archive Manager,



Ubuntu's File Roller, and Windows Explorer's handling of ZIP files. Users cannot update any information in the database through the FUSE program. A system running the analysis program can examine the contents of an archive, insert or modify information in the database, and create the database if it does not exist. The analysis program is incapable of editing the contents of an archive. These use cases outline the functionality of all three stages of TAR Browser: database preparation, archive analysis, and filesystem operations.

## 3. Development Process

### 3.1. Initial Research and Planning

The first two weeks of this project were spent researching the TAR archive format, comparing compression methods, and drafting up a development plan. Initial research led to the PTC MKS documentation regarding the format of TAR archives ["Format of Tar Archives."]. The official GNU TAR documentation provided more detailed information about the format [Free Software Foundation, Inc.]. The findings from both sources are discussed below. These sources proved to be sufficient in order to begin development on an uncompressed TAR analysis program.

Our research led us to the conclusion that TAR archives do not have their own header files, but each member within the archive has a header. These headers include information for each member such as the member's filename, size, access permissions, UID, and GID. Each block within a TAR archive is guaranteed to be 512 bytes in length. TAR archives also include designated "end of archive" blocks, two blocks consisting of repeating zero bytes. It was clear that TAR Browser would need most of this information in order to function properly.

To examine the structure of a TAR archive, the Linux command `od` was used. This program dumps a file into standard output [Meyering, Jim]. `od` was run on a TAR archive with the `-c` option in order to display printable characters. This TAR archive was a small test archive; it was uncompressed and consisted of several images, text files, and directories. This printed out the bytes in a TAR archive as ASCII characters, including the contents of each field of the TAR headers. This simple analysis proved the accuracy of early research of the TAR format.

Several issues were discovered early on. For one, within a TAR archive, multiple members could have the same names. This is not an issue though, as the TAR header for each member includes the file path within the TAR archive. Taken as a whole string, the filepath plus the filename within an archive is always unique. Furthermore, planned GZip support had to be dropped, as it would clearly take too long and would not be very useful, as mentioned previously. BZip2 and XZ support would take priority, and GZip support was planned to be added if time permitted.

At the end of this phase, a development plan was put in place. Basic analysis and extraction programs for uncompressed TAR archives would first be built, as well as the necessary SQL database. While a simple extraction program would be written for each supported format, it should be noted that the early extraction programs were intended as proofs of concept; the logic is used in the final product, but none of these extraction programs are part

of the final submission. BZip2 was selected as the first compression method to work with, as research into this format showed that the BZip2 source is formatted with the intention of being imported into other programs. XZ support would follow, and the final stage would be to develop the FUSE filesystem.

## 3.2. Uncompressed TAR Analysis and Extraction

Support for uncompressed TAR archives was the logical first step in developing TAR Browser. Regardless of whether an archive is compressed or not, TAR Browser needs to be able to pull out a single block, decompress it if necessary, and store information about this block. As such, starting development with uncompressed TAR support would prove to be useful when working with BZip2 and XZ support, as the fundamental procedures are very similar.

TAR Browser, when complete, would need to perform two main functions: analyzing an archive and storing this information in a database, and querying this database in order to display or extract a desired member. Development on uncompressed TAR support began with the analysis program. After performing any necessary checks to ensure the input file is a valid TAR archive, the program would need to analyze the first member header in the archive. The header of a TAR archive member holds vital information such as the filename and path of a member within the archive, the member's length in bytes, and owner and permission information ["Format of Tar Archives"]. As no decompression is necessary, any navigation and reading involved in analyzing an archive could be performed with `fread()`, `fseek()`, and `fwrite()`. The program would store the contents of each header field, and then the rest of the archive would need to be analyzed block by block in a similar fashion. The number of bytes and gigabytes read from an archive are stored separately so that the entire archive can be analyzed in one pass without overflow regardless of the size of the archive. Finally, the end of the archive can be checked. TAR archives always end in 1024 zero bytes, so a simple memory comparison would show whether the archive has been fully examined.

At this point in development, the MySQL database was not yet implemented. The analysis program would simply print out information regarding the members within the archive, as shown in figure 3.1.

```
kyle@browser-desktop:~/uncomptar/tar-as-filesystem$ ./build
/analyze_tar ./test/tarwithimage.tar
entry header offset: 0 GB and 0 bytes
entry name: image.jpeg
file length string: 00000002273
file length int: 1211
link flag: 0
link name:
ustar flag: ustar
file prefix:
data begins at 0 GB and 512 bytes

entry header offset: 0 GB and 2048 bytes
entry name: TEST1.txt
file length string: 00000000017
file length int: 15
link flag: 0
link name:
ustar flag: ustar
file prefix:
data begins at 0 GB and 2560 bytes

entry header offset: 0 GB and 3072 bytes
entry name: TEST2.txt
file length string: 00000000017
file length int: 15
link flag: 0
link name:
ustar flag: ustar
file prefix:
data begins at 0 GB and 3584 bytes
```

*Figure 3.1: Output of the analysis program*

As shown by this output, the initial TAR analysis program was successful. Much of this information would not become useful until later in development, such as the link information. As well, the initial program discarded information regarding the UID, GID, and mode; it was not until development began on FUSE that it became apparent that this information should be retained and not overwritten by FUSE, as a member extracted through TAR Browser must be identical to the member within the archive.

The next step would be to develop a simple extraction program. This program would eventually be scrapped, but it was implemented as a proof of concept to show that TAR Browser would be capable of using information about a single member in order to work with it. FUSE would end up utilizing similar queries and functionality; opening a read-only member of an archive is, in essence, extracting the member to a temporary location. The extraction program would only need information regarding the size of the member and the member's offset within the archive (in figure 3.1 above, this is output as "data begins at X GB and Y bytes"). A simple loop of reading and writing from the beginning of this offset until the bytes read is equal to the size of the member would prove to be successful. TAR Browser would be capable of extracting a member without processing the rest of the archive.

At this point, it became clear that the goals of TAR Browser were feasible. By utilizing offset information from a simple analysis, random access reading and extracting from a TAR archive was possible. The next step would be to implement the database. In theory, this should

not affect the functionality of TAR Browser. Instead of just printing the data, it would be stored into the database, and rather than accepting values in the commandline to extract a member, this program would query the database for this information.

### 3.3. MySQL Database Implementation

Both the analysis and extraction programs would next be modified to utilize a MySQL database<sup>2</sup>. A simple database creator program was put together. This program checks whether the database already exists. If the database does exist, it is updated; otherwise, an error is returned. The database itself must be created through MySQL beforehand. The final version of the database consists of six tables. These tables have remained unchanged throughout the majority of TAR Browser's development cycle. The Archive List table holds information about analyzed archives. Each supported archive format has its own table as well. BZip2 and XZ utilize a second table for block information, but they will be explained shortly. It is worth noting that all three archive tables have a similar structure; archive and member name information is stored, followed by information on where each member is located within the archive. BZip2 and XZ utilize block map tables in addition to archive tables. All six tables can be viewed in detail in Appendix B.

To fine tune the database implementation, a CNF file was created. The use of a CNF file allows the specifying of a database name, username, password, and other options for connecting to a MySQL database without the need to modify the source code of a program. The person who creates the specific SQL database for the program to use must also alter the CNF file to be able to access that database. This file allows for more flexibility of the TAR Browser database.

Once the Archive List and Uncompressed TAR tables were built, MySQL support could be added into the existing analysis and extraction programs. The basic functionality of each program would remain mostly unchanged. Both programs would first need the necessary database connection steps. Instead of printing out file information, the analysis program would store information regarding the archive itself into Archive List, and information about each member into the Uncompressed TAR table. The extraction program would then query the database for offset information instead of taking in hard-coded values. Implementing the database connection and queries did not affect the core functionality of these two programs, as

---

<sup>2</sup> For more information, see the official MySQL documentation here:  
<http://dev.mysql.com/doc/refman/5.7/en/index.html>

predicted. With these changes implemented, uncompressed TAR support was essentially complete until the addition of FUSE, and BZip2 would be the next focus.

### 3.4. BZip2 Analysis and Extraction

Compressed archive support was the natural progression from working with uncompressed TAR archives. BZip2 was chosen as it appeared to be simpler to work with. As well, the processes used to analyze a BZip2 archive would likely carry over to XZ support with some modifications; both formats utilize compressed blocks, and in theory, TAR Browser should be able to decompress single blocks through random access. TAR Browser would need to perform several important tasks for both analysis and extraction. Similarly to uncompressed TAR support, the program must be able to go through the archive block by block to record information about each member of the archive, such as the byte offsets of each block and the members located within each block. The key difference when compared to uncompressed TAR support is that each block of a BZip2 compressed archive must be decompressed in the analysis process. To extract a member, the program must be able to seek to the desired compressed block based on the results of a database query and decompress this block, as well as any other blocks containing parts of the member file.

Initially, the BZip2 API was utilized for TAR Browser development. In theory, API functions should allow the program to read in a single block utilizing the blocksize field in the BZip2 archive header. This method proved to be infeasible due to BZip2's block structure. When a BZip2 archive is created, the data is first run-length encoded before being split into equal size blocks [Taylor, James, "SeekingInBzip2Files."]. The uncompressed size of a given block is unknown; the block size in a BZip2 header actually refers to the amount of run-length encoded data in the block, which is practically useless for TAR Browser's purposes.

A bit of research led us to an open source, public domain program, Seek-BZip2, developed by Taylor Labs at John Hopkins University. This program outputs the bit offsets and uncompressed sizes of each block of a BZip2 archive by partially decompressing each block. Seek-BZip2 can also seek to the beginning of a block and decompress that block, which is exactly what TAR Browser would need to do. Seek-Bzip2 was altered to store a list of block locations and uncompressed sizes, and then implemented into TAR Browser. We confirmed with James Taylor that this was an acceptable route to take, as he released Seek-BZip2 to the public domain [Taylor, James. "Requesting Permission to Use Code."].

The modified Seek-BZip2 program adapted into TAR Browser essentially performed the exact functionality TAR Browser would need, all within a small, public domain program. It would

be unnecessary to write a new program that performs the exact same task, so Seek-BZip2 was slightly modified and implemented into TAR Browser. As well, licensing would not be an issue upon release, as the program is in the public domain. BZip2 support was essentially complete as a result of Seek-BZip2.

### 3.5. XZ Analysis and Extraction

Initial research into the XZ format showed that, for our purposes, a TAR XZ archive is similar in structure to a TAR BZip2 archive. An XZ archive consists of a series of streams, which each hold a number of blocks. Each block has its own XZ header [The Tukaani Project. “The .xz File Format.”]. Thus, XZ support would be extremely similar to BZip2 support; TAR Browser would need to read one block at a time from a TAR XZ archive and store information required for random access to each member in the MySQL database. This information could then be queried by the FUSE filesystem once implemented.

However, several issues were discovered early on in this process. For one, XZ archives consist of just one block unless the user creating the archive explicitly specifies a block size or runs the XZ program in a multi-threaded mode [The Tukaani Project. “The .xz File Format.”]. As such, TAR Browser would not be any more useful than conventional extraction methods for single-block archives. However, it is suggested in the official XZ documentation that any program utilizing XZ random access reads such as TAR Browser should recommend the usage of multi-block archives; this is addressed in the usage guide in Appendix A. As well, at this point in time, XZ is still in active development, which will undoubtedly cause issues in the future.

XZ archives consist of one or more streams, each with a header, one or more blocks, an index, and a footer. In order to analyze an archive for the TAR Browser database, the block number and uncompressed size of each member needs to be stored. This information is stored within the index of an XZ stream. The routine for XZ support, therefore, is to parse the index of an archive, and for each member, save all information that will be required for random access and accurate extraction.

This is where a major issue was encountered. The LZMA API included with the XZ source code proved to be useful for decompression, but the API does not support the type of random access reads TAR Browser would need. Lasse Collin, the lead developer of XZ, was contacted in regards to obtaining the information we would need from an archive. He suggested that we cannibalize a `parse_indexes()` function from the XZ source code itself: “The functionality of `parse_indexes()` should have been put into liblzma (or an I/O library) long

ago, but I don't get much development done. For now adapting `parse_indexes()` into your code is the best way" [Collin, Lasse. "XZ – Random Access Reads."].

While this solution certainly felt like a workaround, it proved to work successfully. This will need to be updated in the future and is discussed further under Future Work, but for the initial release of TAR Browser, this method proved to be sufficient. Various aspects of the XZ format that were originally suspected to be difficult to work with, such as filters and multibyte integers, were all handled smoothly by the LZMA API when decompressing a single block. Once a TAR XZ archive could be analyzed, a simple extraction program was put together as a proof of concept once again. This extraction program was virtually identical to the BZip2 extraction program, as the formats are extremely similar for our purposes. With uncompressed TAR, BZip2, and XZ supported by TAR Browser in its current state, we could now begin development on the FUSE filesystem.

## 3.6. FUSE Research and Development

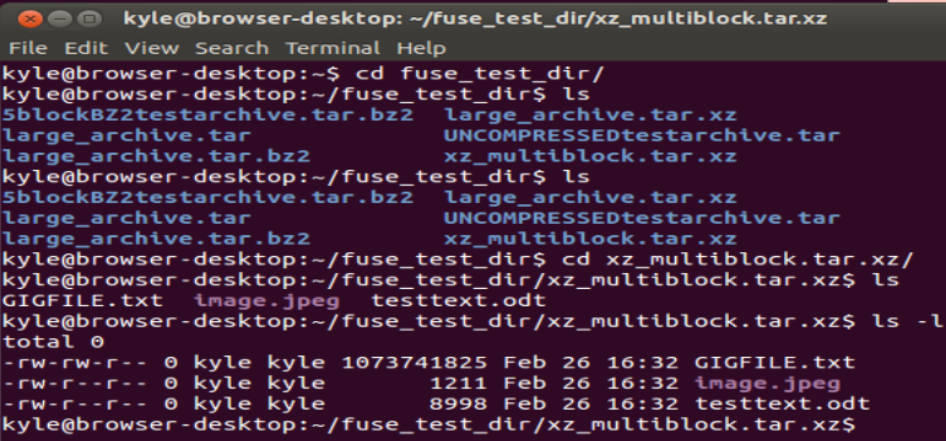
With support for the three planned formats completed, work could begin on the FUSE filesystem. In short, FUSE can "mount" a directory, and when working inside this "mounted" directory kernel operations can be intercepted and handled by functions defined in the FUSE client/userspace program [Oseberg]. A user can then explore this filesystem in their file manager of choice. The FUSE used in TAR Browser would need to be able to mount an archive, allowing the user to explore this archive as if it were a directory. Any operations used with this FUSE would need to query the database and use this information to display or work with the archive's members appropriately.

The first step in FUSE development was to identify the functions that TAR Browser would need to work with. As the program operates on a TAR archive, the filesystem would need to be entirely read-only, so any functions that perform operations to change or write to the filesystem, such as `write()`, `chmod()`, and `mkdir()`, should return errors.

Once these unneeded functions had been defined, returning the appropriate errors if the user tries to write anything within the archive, the next step of development was to create a functioning FUSE filesystem that would allow the user to browse through TAR archives as if they were directories. This required several functions to be implemented. These functions are `getattr()`, which allows a program to acquire the statistics of a file; `access()`, which allows a program to test the existence of a file as well as the file's read and write permissions; `readlink()`, which essentially dereferences a symbolic link; and `readdir()`, which reports



the contents of a directory. Using these functions, a “view-only” filesystem, capable of exploring the inner structure of TAR archives, could be generated and tested.

A terminal window titled 'kyle@browser-desktop: ~/fuse\_test\_dir/xz\_multiblock.tar.xz'. The terminal shows a series of commands and their outputs. The user navigates to the directory and lists files, showing a mix of tar and xz compressed files. Then, they navigate into the xz\_multiblock directory and list its contents, which are GIGFILE.txt, image.jpeg, and testtext.odt. Finally, they run 'ls -l' to show detailed file permissions and metadata for these three files.

```
kyle@browser-desktop: ~/fuse_test_dir/xz_multiblock.tar.xz
File Edit View Search Terminal Help
kyle@browser-desktop:~$ cd fuse_test_dir/
kyle@browser-desktop:~/fuse_test_dir$ ls
5blockBZ2testarchive.tar.bz2  large_archive.tar.xz
large_archive.tar             UNCOMPRESSEDtestarchive.tar
large_archive.tar.bz2        xz_multiblock.tar.xz
kyle@browser-desktop:~/fuse_test_dir$ ls
5blockBZ2testarchive.tar.bz2  large_archive.tar.xz
large_archive.tar             UNCOMPRESSEDtestarchive.tar
large_archive.tar.bz2        xz_multiblock.tar.xz
kyle@browser-desktop:~/fuse_test_dir$ cd xz_multiblock.tar.xz/
kyle@browser-desktop:~/fuse_test_dir/xz_multiblock.tar.xz$ ls
GIGFILE.txt  image.jpeg  testtext.odt
kyle@browser-desktop:~/fuse_test_dir/xz_multiblock.tar.xz$ ls -l
total 0
-rw-rw-r-- 0 kyle kyle 1073741825 Feb 26 16:32 GIGFILE.txt
-rw-r--r-- 0 kyle kyle      1211 Feb 26 16:32 image.jpeg
-rw-r--r-- 0 kyle kyle      8998 Feb 26 16:32 testtext.odt
kyle@browser-desktop:~/fuse_test_dir/xz_multiblock.tar.xz$
```

Figure 3.2: Exploring the contents of a TAR XZ archive

Finally, to enable true read-only access within the filesystem, the `open()` and `read()` functions would need to be defined. In the case of the TAR Browser filesystem, the `open()` function was written to work differently from its standard Linux counterpart. Normally, this function returns a file descriptor that can be used by other system calls such as `read()` and `write()`. However, this would not be possible within an archive. The files shown by this filesystem do not actually exist on disk, and so they cannot have an open file handle, which is required by the native `open()` call. Instead, the TAR Browser FUSE `open()` function queries the MySQL database for the existence of the member within the archive. If the file is found to exist in the database, a unique ID number is returned as a file handle. This ID can be used to quickly re-query the desired file’s information from the database. The `read()` function, which normally reads bytes from a file descriptor, utilizes the ID returned from `open()` to query block and member data offsets from the database. This uses code recycled from the “proof of concept” extraction functions from early development in order to decompress the member if necessary and read the requested member data from the TAR archive holding this member.

Of note is that each of the functions mentioned, with the exception of `open()` and `read()`, function independently and do not share information. These functions could operate in any order. As such, each function initiates a separate MySQL database connection, queries the necessary tables to retrieve information, and finally closes the MySQL connection before returning. This also acts as a failsafe. If the MySQL database crashes or closes due to an error, the FUSE functions will only fail until the database recovers. If FUSE initiates a single

connection instead, then it will continue to run but will not recover when the database issue is resolved.

Once the FUSE filesystem was implemented, the first release of TAR Browser was essentially complete. The next step would be to evaluate TAR Browser in terms of accuracy and performance.

## 4. Functionality

### 4.1. Database Preparation

As discussed in section 3.3, the MySQL database preparation program creates the necessary tables for the database. This must be run before the archive analysis or TAR Browser programs. The MySQL database must already exist; the name of the database to use is set in `TarBrowserConnectOptions.cnf`; a sample CNF file is included with the TAR Browser source code. This CNF file also allows specification of a username, password, and host. The Prepare Database program checks for a database on the system by the name defined in the CNF file. If this database exists, the tables in Appendix B are created if they do not exist. Once the database has been built, it can be utilized by the other two programs of the TAR Browser package.

### 4.2. Archive Analysis

The final archive analysis program is mostly left unchanged from the completion of XZ support. All three archive analysis programs (uncompressed TAR, BZip2, and XZ) are now functions which can be called from a single executable based on the type of archive that is being analyzed. This main executable is also responsible for checking whether the archive is a valid file.

Each of the three analysis functions begins by connecting to the database and beginning a transaction. The `ArchiveList` table is queried for an archive with this name. If it does not exist, it is added, and if it does, the user is prompted as to whether they want to overwrite the existing information in the database. If the user does overwrite this information, every table corresponding to this archive type is purged of tuples concerning this archive (for example, for a BZip2 archive, `Bzip2_files`, `Bzip2_blocks`, and `ArchiveList` are cleared of this archive).

For an uncompressed TAR archive, the program begins reading data from the archive. When a member's header is found, the header is parsed and stored into a "headerblock" data structure. This structure holds all of the fields within a TAR header. There is the possibility that the member's full path within the TAR archive or the target path of a link is too long to be held in the buffer allocated in a TAR header. In this case, the member name displayed in the TAR header is `“./.@longlink”` and the data corresponding to this TAR header is the filename path or link path of the next real TAR entry. If the `“./.@longlink”` is found, the next real TAR header evaluated has its appropriate field, its name or link target, overwritten by the data of the longlink

entry. The program, now pointing at the end of the header, records information from the header as well as its current position in bytes and gigabytes (which corresponds to the position of the member), inserts this information into the database, skips the actual data of this member, then begins reading more data. This will be the next header in the archive unless the program has reached the end of archive blocks, in which case analysis completes and if no errors have occurred the SQL transaction is committed.

TAR BZip2 and TAR XZ archives function extremely similarly. Each function incorporates a block decompression function appropriate for the archive file type; this helper function reads and decompresses a single block of the archive into memory. The XZ version uses the `grab_block()` function from the cannibalized XZ source code and requires only a file name and a block number. The BZip2 version requires the file name and block number, but also uses a “blockmap” struct, defined by `Seek-BZip2`, in order to know the offsets of each block. Aside from the differences in decompression, the rest of the analysis for BZip2 and XZ archives is remarkably similar; the blockmap structure is even used in XZ and functions as intended, despite being written for BZip2. Each analysis function begins with a database connection, starting a transaction, and querying `ArchiveList` for the existence of this archive, identically to the process used in uncompressed TAR analysis. Before looping through the archive, the first block is decompressed so we have some initial data to work with. The rest of the archive is then processed very similarly to analyzing an uncompressed TAR archive. The analysis is performed in the same manner as described in for uncompressed TAR archives above, with one major exception. Instead of reading from a file, data is read from a decompressed block. If at any point the amount of data left in the block is not enough to fill a complete TAR header or skip over the TAR entry’s data, the program reads as much data as it can, discards the block, decompresses the next block, and reads the remaining data from the beginning of the newly decompressed block. Every time a member within the TAR archive is analyzed, an insertion query is then generated based on information within its header, as well as information obtained about what blocks this member lives in. When analysis of the archive is finished, the blockmap is also stored into the database under the appropriate “\_blocks” table. Refer to Appendix B for the information stored into these tables.

In addition, an option to reduce the output of the program was incorporated. With an additional argument, the user can now limit the output they see to just the prompts requiring user input and error messages. Without this argument, information about each member in the archive being analyzed is printed to standard output. Refer to the Usage Guide in Appendix A for an example.

## 4.3. TAR Browser Filesystem

As TAR Browser works through FUSE, the program provides the infrastructure for a computer's file manager to view the inner members of analyzed TAR archives as if they had been extracted to the filesystem. Any program or text editor may then open read-only copies of the files displayed inside these TAR archives. FUSE does this by intercepting system calls using a kernel module and redirecting those system calls to user defined functions where we can program the functions to behave in a different way. What follows is a discussion of the behavior of the overridden functions.

Functions that modify or change the state of the filesystem are programmed to return immediate errors. These functions are `mknod()`, `mkdir()`, `symlink()`, `unlink()`, `rmdir()`, `rename()`, `link()`, `chmod()`, `chown()`, `truncate()`, `utimens()`, `write()`, `statfs()`, and `fallocate()`. The functions that are implemented into the FUSE of TAR Browser share some common structure. They all begin by initiating a MySQL database connection and end by terminating it. Each function's behavior is split into three different possibilities based on what the function directed toward: the FUSE root ("/") directory, where the FUSE filesystem is mounted; an existing TAR archive (for example, "/archive.tar.xz"); or something within the tar archive (such as "/archive.tar/test.txt").

The FUSE `getattr()` function's expected behavior is the same as the `lstat()` function. It is expected to fill a file statistics structure with information about the file it is called on. Each of the three possibilities explained prior will populate this structure in a different way. In the FUSE of TAR Browser, root directory statistics are pulled from the directory where FUSE is mounted. In the case of an existing archive, the location of that archive on the system is queried from the database and the `lstat()` function is performed on the archive. If `getattr()` is called on an archive member, then all information about the member is queried from the database.

The FUSE `access()` function's expected behavior is nearly identical to its system-defined counterpart. The target can be tested for existence, as well as read, write, and execute permissions. In the case of a directory or a TAR archive (which the FUSE system views as a directory), if the tested access level contains write permissions, the `access()` call returns an error, as TAR Browser's filesystem will always be read-only. In the case of a member within a TAR archive, any access test that is not an existence or read-only test returns an error for the same reason.

The FUSE `readlink()` function dereferences a symbolic link and places the link target into a buffer. If called on the FUSE root directory or a TAR archive, an error is returned as these are not links in this system. If called on a member of a TAR archive, a MySQL query is sent to retrieve the member's information. If the member is not a link, then an error is returned. Otherwise, the link's target is placed into the buffer provided to the function.

The FUSE `readdir()` function populates a list with the contents of the directory it is called on. If called on the FUSE root directory, the MySQL Archive List table is queried and all existing archives are added to the list. Otherwise, a query with the appropriate pattern matching is sent to the table associated with the archive type being examined (such as CompXZ for compressed XZ files) and the results are added to the list.

The FUSE `open()` function expects the same behavior as the native `open()`, inserting a file handle into a file information structure. If called on the FUSE root directory an error is returned; this directory cannot be opened as a file. If called against an archive or archive member, the unique ID of the appropriate MySQL table is stored in the file structure for later use.

The FUSE `read()` function behaves similarly to the native `read()` function of Linux. This function places requested data into a provided buffer. If called on the FUSE root directory, an error is returned. If called on an archive, the MySQL table of archives is queried using the number stored in place of the file handle. From that query, the true path to the archive on the system is retrieved. The archive is then opened and read from using native `open()` and `read()` functions. If called on a member of a compressed archive, the archive's blockmap is queried from the database and stored in memory. In the case of a member of an uncompressed archive, the blockmap step is skipped, as this is unnecessary for an uncompressed archive. The information required to extract the member is then queried from the appropriate table of the database. The blocks the desired member exists on are then decompressed (if necessary) and the file's data is copied into the buffer provided to the `read()` function.

## 5. Evaluation

### 5.1. Testing Procedures

TAR Browser would need to be tested for accuracy and efficiency. The program would be useless if the member extracted from an archive is not identical to the member that is still within the archive. In terms of efficiency, recall one of the major goals of the TAR Browser project; accessing a single member within a large TAR archive is currently a slow process that TAR Browser aims to improve. In theory, extracting only the necessary blocks should be much faster than extracting an entire archive only to return the desired member, but this would need to be tested thoroughly. Finally, testing TAR Browser's toll on CPU performance would be valuable as well. While subpar performance would not be much of an issue if speeds are still greater than current extraction methods, extremely poor performance would be detrimental to the project; many machines would not be able to handle this software well.

### 5.2. Accuracy Tests

Accuracy tests were straightforward. The Linux `diff` command detects any differences between two files [Eggert et al.]. This command would be run on a member within a FUSE supported directory for an archive and the actual file extracted from this archive through default TAR extraction. These two files should be identical, and thus `diff` should not output anything. This process would be repeated for several different files for each type of archive; as no measurements are taken for these tests, formal trials were not deemed necessary.

### 5.3. Accuracy Test Results

Simple accuracy tests through `diff` proved that the members within the FUSE filesystem for an archive and files extracted from an archive through the TAR program were identical. As seen in figure 5.1, there is no output when running this command; `diff` only outputs text if there is a difference between the two files. As such, every bit of these two files is identical.

```
tomorrow@ubuntu: ~/Desktop/Link to tar-as-filesystem
tomorrow@ubuntu:~/Desktop/Link to tar-as-filesystem$ ./tarbrowser ~/TARBROWSER -d
FUSE library version: 2.9.2
tomorrow@ubuntu: ~/tar-as-filesystem/test/TEST DOCUMENTS
tomorrow@ubuntu:~/tar-as-filesystem/test/TEST DOCUMENTS$ diff image.jpeg ~/TARBROWSER/
9blockBZ2testarchive.tar.bz2/image.jpeg
tomorrow@ubuntu:~/tar-as-filesystem/test/TEST DOCUMENTS$ diff image.jpeg ~/TARBROWSER/
xztestarchive.tar.xz/image.jpeg
tomorrow@ubuntu:~/tar-as-filesystem/test/TEST DOCUMENTS$ diff image.jpeg ~/TARBROWSER/
UNCOMPRESSEDtestarchive.tar/image.jpeg
tomorrow@ubuntu:~/tar-as-filesystem/test/TEST DOCUMENTS$ _
```

Figure 5.1: Results of running “diff” on a file within several archives compared to the original file

This command was run on several files of several different uncompressed TAR, TAR BZip2, and TAR XZ archives, all with the same result. While there are no formal metrics that can be provided for these tests, these have sufficiently shown that the members output from TAR Browser are identical to those within the archives. Thus, TAR Browser’s accuracy is undisputed.

## 5.4. Efficiency and Performance Tests

Efficiency and performance testing would be performed in a similar manner. A comparison of access times was performed on a 2.7 gigabyte TAR archive created from the /usr directory of a Linux filesystem. TAR BZip2 and TAR XZ compressed archives were created with the same contents. The XZ archive was forced to include multiple blocks through XZ’s “--blocksize” option. Members were extracted from these archives through the TAR program, specifically by running `tar xvf <path to member>` on each archive. To achieve an accurate comparison with the TAR program’s extraction utility, the `cp` command was used on members within different archives through FUSE. GNU TAR in particular was tested, as the machine running the tests is an Ubuntu Linux virtual machine. Both commands end by writing a file to a different location, resulting in similar I/O overhead after the desired file has been loaded into memory. The goal is to speed up access times, so any system I/O time needed to load the file into memory counts against that method’s access time. These tasks would be measured through the native Linux `time` command, which outputs the elapsed time in minutes and seconds (the time a task takes to run completely), as well as the System and User CPU times.

All recorded tests were performed on the same computer within an Ubuntu Linux 12.04.5 virtual machine. This computer runs on an Intel Core2 Duo T6600 2.20 GHz CPU and is running



Windows 7 64-bit. The virtual machine, built in Oracle VirtualBox 4.3.6 r91406, is given 100% of resources of one CPU core and 1 GB of RAM.

Five members at various points within each archive were selected (these files were consistent with each of the three archives tested). The first and last members within these archives were included in these five selected members. Five trials were performed for each member of each archive type for each extraction method (TAR Browser versus `tar xvf <path to member>`).

## 5.5. Efficiency and Performance Test Results

The tables below show access time comparisons between FUSE access and native TAR commands for uncompressed TAR, TAR BZip2, and TAR XZ archives. The times are split into real time (real), user CPU time (UCPU), and system CPU time (SCPU).

Table 5.1: Uncompressed TAR access times:

| Block in archive:<br>Position/Total | Path to Member   | Average of FUSE access times*<br>(Five trials)                     | Average of GNU TAR access times* (Five trials)                     |
|-------------------------------------|--|--|--|
| 5 / 174797                          | usr/share/aptdaemon/lintian-fatal.tags                         | Real: 0.1496<br>UCPU: 0.0<br>SCPU: 0.0128                          | Real: 37.3018<br>UCPU: 0.4048<br>SCPU: 5.1080                      |
| 43700 / 174797                      | usr/share/icons/Humanity/actions/24/media-record.svg           | Real: 0.1688<br>UCPU: 0.0016<br>SCPU: 0.0112                       | Real: 36.9380<br>UCPU: 0.3600<br>SCPU: 4.9888                      |
| 87398 / 174797                      | usr/share/cups/doc-root/help/ref-snmp-conf.html                | Real: 0.1464<br>UCPU: 0.0016<br>SCPU: 0.0120                       | Real: 37.1464<br>UCPU: 0.3832<br>SCPU: 4.9672                      |
| 131098 / 174797                     | usr/src/linux-headers-3.2.0-70/arch/arm/mach-mmp/Makefile.boot | Real: 0.1608<br>UCPU: 0.0072<br>SCPU: 0.0096                       | Real: 37.2514<br>UCPU: 0.4136<br>SCPU: 4.9864                      |
| 174797 / 174797                     | usr/sbin/atd   | Real: 0.1246<br>UCPU: 0.0064<br>SCPU: 0.0104                       | Real: 37.0798<br>UCPU: 0.3880<br>SCPU: 5.0616                      |
| <b>TOTAL AVERAGES</b>               |  | <b>Real: 0.15004</b><br><b>UCPU: 0.0042</b><br><b>SCPU: 0.0112</b> | <b>Real: 37.1435</b><br><b>UCPU: 0.3899</b><br><b>SCPU: 5.0224</b> |

\* Real times reported in seconds. UCPU and SCPU times reported in clock ticks [Linux Man-pages Project].

Table 5.2: BZip2 Compressed TAR access times

| Block in archive:<br>Position/Total | Path to Member   | Average of FUSE access times*<br>(Five trials)                    | Average of GNU TAR access times* (Five trials)                          |
|-------------------------------------|--|---|---|
| 5 / 174797                          | usr/share/aptdaemon/lintian-fatal.tags                         | Real: 0.3066<br>UCPU: 0.0016<br>SCPU: 0.0136                      | Real: 365.7694<br>UCPU: 135.8460<br>SCPU: 217.3444                      |
| 43700 / 174797                      | usr/share/icons/Humanity/actions/24/media-record.svg           | Real: 0.3190<br>UCPU: 0.0032<br>SCPU: 0.0112                      | Real: 365.145<br>UCPU: 138.5128<br>SCPU: 214.6980                       |
| 87398 / 174797                      | usr/share/cups/doc-root/help/ref-snmp-conf.html                | Real: 0.2888<br>UCPU: 0.0016<br>SCPU: 0.0144                      | Real: 369.1124<br>UCPU: 141.4858<br>SCPU: 215.9180                      |
| 131098 / 174797                     | usr/src/linux-headers-3.2.0-70/arch/arm/mach-mmp/Makefile.boot | Real: 0.2916<br>UCPU: 0.0024<br>SCPU: 0.0096                      | Real: 375.3136<br>UCPU: 145.6212<br>SCPU: 218.3894                      |
| 174797 / 174797                     | usr/sbin/atd   | Real: 0.1850<br>UCPU: 0.0016<br>SCPU: 0.0144                      | Real: 369.8356<br>UCPU: 143.3066<br>SCPU: 215.2846                      |
| <b>TOTAL AVERAGES</b>               |  | <b>Real: 0.2782</b><br><b>UCPU: 0.0021</b><br><b>SCPU: 0.0126</b> | <b>Real: 369.0352</b><br><b>UCPU: 140.9545</b><br><b>SCPU: 216.3269</b> |

\* Real times reported in seconds. UCPU and SCPU times reported in clock ticks [Linux Man-pages Project].

Table 5.3: XZ Compressed TAR access times

| Block in archive:<br>Position/Total | Path to Member   | Average of FUSE access times*<br>(Five trials)                     | Average of GNU TAR access times* (Five trials)                         |
|-------------------------------------|--|--|--|
| 5 / 174797                          | usr/share/aptdaemon/lintian-fatal.tags                         | Real: 3.7720<br>UCPU: 0.0008<br>SCPU: 0.0240                       | Real: 199.8374<br>UCPU: 58.8464<br>SCPU: 134.4202                      |
| 43700 / 174797                      | usr/share/icons/Humanity/actions/24/media-record.svg           | Real: 3.4630<br>UCPU: 0.0008<br>SCPU: 0.0240                       | Real: 198.4076<br>UCPU: 59.0072<br>SCPU: 132.7284                      |
| 87398 / 174797                      | usr/share/cups/doc-root/help/ref-snmp-conf.html                | Real: 6.0990<br>UCPU: 0.0024<br>SCPU: 0.0240                       | Real: 196.6448<br>UCPU: 58.7888<br>SCPU: 131.0488                      |
| 131098 / 174797                     | usr/src/linux-headers-3.2.0-70/arch/arm/mach-mmp/Makefile.boot | Real: 3.4012<br>UCPU: 0.0008<br>SCPU: 0.0256                       | Real: 196.9826<br>UCPU: 58.0526<br>SCPU: 130.9584                      |
| 174797 / 174797                     | usr/sbin/atd   | Real: 3.6910<br>UCPU: 0.0024<br>SCPU: 0.0232                       | Real: 203.1924<br>UCPU: 61.0520<br>SCPU: 134.5644                      |
| <b>TOTAL AVERAGES</b>               |  | <b>Real: 4.08524</b><br><b>UCPU: 0.0014</b><br><b>SCPU: 0.0242</b> | <b>Real: 199.0130</b><br><b>UCPU: 59.1494</b><br><b>SCPU: 132.7440</b> |

\* Real times reported in seconds. UCPU and SCPU times reported in clock ticks [Linux Man-pages Project].

## 5.6. Conclusions

These tests have shown that TAR Browser is both accurate and efficient. Accuracy has already been discussed; the files output by TAR Browser are identical to those extracted from an archive through traditional means. Any corruption would have been reported by `diff`. TAR Browser therefore accurately “extracts” members from uncompressed TAR, compressed TAR BZip2, and compressed TAR XZ archives.

Tables 5.1, 5.2, and 5.3 all demonstrate remarkably similar data in terms of both efficiency and performance. TAR Browser is clearly much more efficient at extracting a file than the native TAR tools. Actual running times for TAR Browser’s tasks were less than a second on average for uncompressed archives and for compressed TAR BZip2 archives, and about 4 seconds for TAR XZ archives. In contrast, `tar xvf` extraction took about 37 seconds for uncompressed archives, over 6 minutes for TAR BZip2 archives, and about 3.5 minutes for TAR XZ archives. The dramatic differences in time are a result of TAR Browser’s extraction methods. GNU TAR decompresses and extracts the entire archive, then copies the desired member to a new location outside of the archive. TAR Browser only handles the necessary blocks, decompressing and copying only the desired member. The overhead of working with the entire archive is only performed once per archive with TAR Browser; this is done when analyzing the archive; accessing any member after analysis no longer has this overhead, while traditional extraction methods such as GNU TAR process the entire archive each time. Times were extremely consistent with each file tested, regardless of where the file was located within the archive being tested; this is true for both TAR Browser and GNU TAR.

TAR Browser is not only much faster than traditional extraction, but our data shows that it has a much lighter toll on CPU performance than GNU TAR as well. TAR Browser never took more than 0.0256 seconds in CPU time (SCPU for TAR XZ archives), whereas extraction from a compressed archive took over 100 seconds in CPU time for compressed archives. For uncompressed archives, GNU TAR typically took around 5 seconds System CPU time and 0.3899 seconds User CPU time. The difference between TAR Browser and GNU TAR when handling uncompressed archives is not too significant in our tests. However, at a large scale (for example, a 20 GB or greater archive), these differences may become much more noticeable; this will not be tested, as compressed archives are of greater concern for this project. In contrast, the difference in performance between TAR Browser and GNU TAR for both TAR BZip2 and TAR XZ archives is extreme. The reasoning is the same as for real time; GNU TAR decompresses the entire archive, then saves the desired member, while TAR Browser only

decompresses and extracts this member. Again, TAR Browser's analysis program does have this overhead, but is only performed once per archive (unless the archive is updated in the future).

## 5.7. Concerns

While we are extremely confident in TAR Browser's improvements over traditional extraction methods, there are always improvements that could be made with any study.

The overhead of the copying a file through Linux (and thus through FUSE) is comparable to that of copying a file once it has been extracted into memory. However, as no true parallel exists between TAR Browser and traditional file extraction aside from this, this data may not be completely reliable. Still, we believe it to be so, as both TAR Browser and GNU TAR are performing very similar tasks in these tests. The differences in overhead (after the file is in memory) between the two methods are negligible.

As well, only the GNU standard TAR program was tested against TAR Browser. POSIX TAR, as well as other programs such as File Roller or the Gnome Archive Manager, were not tested. More research could be done to compare TAR Browser to these alternatives. GNU TAR was chosen for comparison as it is included with most popular Linux distributions and is essentially the source of modern TAR functionality [Free Software Foundation, Inc.].

One major concern was that hardware played a role in the test results. These tests were performed on a fairly outdated machine. To be safe, a subset of these tests was run on a more powerful machine, a desktop with an Intel Core i7-3770K CPU (3.50 GHz) running a 32-bit Ubuntu 14.04 LTS virtual machine through VMWare Player 6.0.5 build-2443746. Only the first and last members of each archive were tested, and an average of three trials was taken. This data is inconclusive as a result of technical issues when trying to extract files from a TAR.BZ2 archive through GNU TAR.

Table 5.4: Uncompressed TAR access times (Subset of tests on second machine)

| Block in archive:<br>Position/Total | Path to Member                         | Average of FUSE access times*<br>(Three trials) | Average of GNU TAR access times*<br>(Three trials) |
|-------------------------------------|--|---|--|
| 5 / 174797                          | usr/share/aptdaemon/lintian-fatal.tags | Real: 0.2300<br>UCPU: 0.0000<br>SCPU: 0.0070    | Real: 22.6977<br>UCPU: 0.0120<br>SCPU: 1.1520      |
| 174797 / 174797                     | usr/sbin/atd                           | Real: 0.0190<br>UCPU: 0.0000<br>SCPU: 0.0010    | Real: 22.8653<br>UCPU: 0.0160<br>SCPU: 1.1520      |

\* Real times reported in seconds. UCPU and SCPU times reported in clock ticks [Linux Man-pages Project].

Table 5.5: XZ Compressed TAR access times (Subset of tests on second machine)

| Block in archive:<br>Position/Total | Path to Member                         | Average of FUSE access times*<br>(Three trials) | Average of GNU TAR access times*<br>(Three trials) |
|-------------------------------------|--|---|--|
| 5 / 174797                          | usr/share/aptdaemon/lintian-fatal.tags | Real: 1.6680<br>UCPU: 0.0000<br>SCPU: 0.0000    | Real: 64.5260<br>UCPU: 23.2760<br>SCPU: 40.4973    |
| 174797 / 174797                     | usr/sbin/atd                           | Real: 1.8453<br>UCPU: 0.0000<br>SCPU: 0.0000    | Real: 64.1653<br>UCPU: 22.8120<br>SCPU: 40.6853    |

\* Real times reported in seconds. UCPU and SCPU times reported in clock ticks [Linux Man-pages Project].

While data for TAR BZip2 could not be obtained on this machine, the data for tests run on uncompressed TAR and TAR XZ archives is similar to the data from our formal tests. The differences in both real time and CPU time were noticeably faster when compared to the tests run on the first machine, but the differences between FUSE and GNU TAR access times are still substantial for these two formats. We do not believe hardware played a major role in the formal TAR Browser tests.

Despite these concerns, we do not believe that there are any major issues with this study. TAR Browser outperformed GNU TAR in every test performed, and any potential issues with these tests appear to be minor.



## 6. Future Work

### 6.1. Public Open Source Release

TAR Browser will be released and maintained as an open source, public domain project. The current GitHub repository<sup>3</sup> is planned to be kept in place. TAR Browser does utilize XZ as a dependency, as well as modified code from Seek-BZip2, as discussed previously. Seek-BZip2 has been released to the public domain [Taylor, James. "Requesting Permission to Use Code."]. The LZMA libraries (liblzma) are in the public domain as well [The Tukaani Project. "XZ Utils."]. The files from which TAR Browser cannibalized XZ source code, are also in the public domain. No other code from TAR, BZip2, or XZ source is included in our project; the imported code and utilities are all considered public domain. FUSE itself is released under a GPL license, but no licensed code, such as the FUSE libraries, are included in the TAR Browser package.

### 6.2. XZ Support Maintenance

As TAR Browser will be an open source project, maintenance will be vital. In particular, XZ support will need to be improved down the road due to the cannibalization of the XZ source code. XZ is still in active development, and Lasse Collin specifically mentioned that our current methods to support this format will break in future releases of XZ: "There is one likely backward-incompatible future update that will break the current `parse_indexes()`: metadata support. I plan to add a new Stream type (indicated by a bit in Stream Flags) for Streams that hold arbitrary metadata. I don't have the exact details decided yet but I will try to keep it simple and thus relatively easy to take into account in `parse_indexes()`-like situation too. Very probably metadata will be used rarely, so it will rarely affect code that doesn't support it" [Collin, Lasse. "XZ – Random Access Reads].

As a result, the developers maintaining TAR Browser will have to keep up to date with XZ development and resolve these issues in the future. Luckily, BZip2 development has more or less ceased, so it should not be problematic due to future updates. The last update to BZip2 was version 1.0.6, released in September, 2010, so it seems unlikely that there will be large future updates [Seward, Julian].

---

<sup>3</sup> <https://github.com/tomorrow-nf/tar-as-filesystem>

## 6.3. Improvements and Expansion

TAR Browser was built fairly modularly. Introducing support for further compression formats would not be a very difficult task. Most compression methods utilize blocks in a similar way to BZip2 and XZ. As a result, all one would need to do is develop a method to seek to and decompress a single block of a compressed archive. The database would need to be updated, as well as the analysis program. Supporting other compression methods would not be a trivial task, but it is certainly a feasible expansion to TAR Browser in the future.

## 6.4. Final Thoughts

TAR Browser succeeds as a solution to the proposed problem. The members shown within the read-only filesystem are identical to those within the archives, as shown by our accuracy tests. Performance and efficiency testing has shown dramatic improvements in both categories when comparing TAR Browser to GNU TAR, one of the most commonly used TAR management programs. If TAR Browser is implemented as planned, users will undoubtedly notice these improvements in regular usage.

# Appendices

## Appendix A: TAR Browser Installation and Usage Guide

### Installation:

---

This installation guide assumes you are compiling TAR Browser from source. If you already have the executables (tarbrowser, prepareDatabase, and analyze\_archive), you do not need this guide.

#### Unpacking and Database Setup:

1. Download and unpack the TAR Browser source code from the TAR Browser Github repository:  
<https://github.com/tomorrow-nf/tar-as-filesystem/archive/master.zip>
2. Download the MySQL Client Library:  
<http://dev.mysql.com/downloads/connector/c/>
3. Set up a MySQL server or use an existing one:
  - a. Create a database on the server to be used by this program.
  - b. Create a user with permissions to access the database.
  - c. Alter the four fields of TarBrowserConnectOptions.cnf to the values necessary to access the database.
  - d. TarBrowserConnectOptions.cnf should remain in the same directory as the TAR Browser executable (tarbrowser).
  - e. Refer to the MySQL documentation if assistance is required with these steps.
4. Run the “make prepareDatabase” command utilizing the included makefile.
  - a. If making the prepareDatabase program fails, ensure that the MySQL client library was installed successfully.

#### Archive Analysis Setup:

1. Make sure a relatively current version of XZ Utils is available on your machine.
  - a. The currently installed version can be checked by running “xz -V”
  - b. We recommend version 5.2.0 or later. The latest version can typically be installed through your Linux distro’s package manager, or it can be found here:

<http://tukaani.org/xz/>

2. Run the makefile command “make analyze\_archive” command using the included makefile
  - a. If errors are produced, you may have to update your installed version of XZ Utils

**TAR Browser Setup:**

1. If FUSE is not already installed on your system download it from here:  
<http://fuse.sourceforge.net/>
  - a. Follow the official FUSE documentation in order to install FUSE
2. Run the command “make tarbrowser” using the provided makefile.
3. Create an empty directory which will become the mount point for the FUSE filesystem

## Usage:

---

Using TAR Browser effectively is a three-step process. The database must be prepared; this only needs to be done once per system. Archives must be analyzed, populating the database. The TAR Browser program can then be used to explore any archives that have been analyzed.

### Preparing the Database:

1. Run “./prepareDatabase” to create the necessary tables in your database.
  - a. This program will not delete any existing tables or table entries. If you wish to recreate a table or the entire database, the existing tables and entries must be removed through the MySQL commandline.
2. The “prepareDatabase” program will notify you if there is a problem setting up the database.
  - a. If any errors arise, ensure that the MySQL database and the CNF file are set up properly.

### Analyzing Archives:

1. The archive analysis program can be used to analyze any TAR archive (.tar), XZ Compressed TAR Archive (.tar.xz), or BZip2 Compressed TAR Archive (.tar.bz2) using the command “./analyze\_archive filename”.
  - a. You can add the -q option (for “Quiet”) to the end of analyze\_archive to only receive output when necessary: `./analyze_archive filename -q`
    - i. This is the only commandline option available with the archive analysis program. (-Q is also allowed, but is identical to its lowercase counterpart -q).
    - ii. With this option, output regarding archive and member details will not be shown. The user will be notified when the program completes and commits entries to the database, or an error occurs.
  - b. Each TAR archive must be analyzed separately. The analyze\_archive program will only accept one archive at a time.
  - c. One property to note is that XZ does not naturally allow random access. The “xz --list filename” command can be used to display the number of blocks in an XZ compressed file. If there is only one block, the file can still be analyzed but there is

- no overhead reduction from analyzing it. XZ allows you to create multi-block archives through the `--blocksize` option when compressing an archive. It is recommended to compress archives with a block size such that at least 20 blocks are created, or (for larger archives) each block is at least 200 MB. This is highly dependent on the size of the archive.
- d. If an error occurs during database communication, the archive analysis program will notify the user and roll back any transactions.

### **Browsing Archives:**

1. Mount the FUSE filesystem by using the command `./tarbrowser <FuseDir>` where `FuseDir` is the path to the directory where you wish to mount to (ex. `../../home/FuseDir`).
  - a. FUSE runs in the background by default.
  - b. If you wish to see a log of what FUSE is doing in the terminal add a `-d` argument to the previous command, this will cause FUSE to not run in the background and instead output information about running tasks to the terminal.
  - c. The FUSE filesystem process can be terminated by unmounting the directory in which it is loaded (`/FuseDir` in the example above).
2. You can now browse through any successfully analyzed archives as if they were uncompressed read-only directories from the directory mentioned in step 1. These can be browsed from the terminal or through most file explorers, such as Nautilus.
  - a. TAR Browser is intended for use by a single user or to be run on a server.

## Appendix B: TAR Browser SQL Database

This is a description of the MySQL tables used by the three programs described in this report.

The **ArchiveList** table contains a list of all archives that have been analyzed:

### *ArchiveList:*

---

|                    |   |
|--------------------|---|
| <b>ArchiveID</b>   | – An integer identification number associated with an entry in the table.   |
| <b>ArchiveName</b> | – The name of the archive file this entry relates to.   |
| <b>ArchivePath</b> | – The path within the system where the file referenced by ArchiveName exists.   |
| <b>Timestamp</b>   | – A string denoting the last time the archive file was modified. This is used to determine if the archive file is still valid for use by the program. |

---

The **UncompTar** table contains a list of all members found in uncompressed TAR archives, as well as the information required to extract them:

### *UncompTar:*

---

|                     |  |
|---------------------|--|
| <b>FileID</b>       | – An integer identification number associated with an entry in the table.                |
| <b>ArchiveID</b>    | – An integer that associates the table entry with an entry in the ArchiveList table.     |
| <b>ArchiveName</b>  | – The name of the archive file this entry exists inside.                                 |
| <b>MemberName</b>   | – The name of the archived member this entry refers to. (ex. File.txt)                   |
| <b>MemberPath</b>   | – The path to this archived member within the physical archive.<br>(ex. Dir_one/Dir_two) |
| <b>GBoffset</b>     | – The number of gigabytes that must be skipped to reach the file this entry refers to.   |
| <b>BYTEoffset</b>   | – The number of bytes that must be skipped after GBoffset to reach the desired file.     |
| <b>MemberLength</b> | – The bytes of data in this archived member.   |
| <b>LinkFlag</b>     | – Preserved link flag entry from the TAR header of this file.                            |
| <b>DirFlag</b>      | – A Y/N character flag indicating whether this entry refers to a directory or not.       |
| <b>Mode</b>         | – Preserved permission bits from the TAR header of this file.                            |
| <b>UID</b>          | – Preserved user ID from the TAR header of this file.                                    |
| <b>GID</b>          | – Preserved group ID from the TAR header of this file.                                   |
| <b>LinkTarget</b>   | – If the file referenced by this entry is a link this is path the link points to.        |

---

The **Bzip2\_files** table contains a list of all members found in compressed TAR BZip2 archives, as well as information required to extract them:

***Bzip2\_files:***

---

|                     |   |
|---------------------|---|
| <b>FileID</b>       | – An integer identification number associated with an entry in the table.   |
| <b>ArchiveID</b>    | – An integer that associates the table entry with an entry in the ArchiveList table.  |
| <b>ArchiveName</b>  | – The name of the archive file this entry exists inside.  |
| <b>MemberName</b>   | – The name of the archived member this entry refers to. (ex. File.txt)  |
| <b>MemberPath</b>   | – The path to this archived member within the physical archive.<br>(ex. Dir_one/Dir_two)  |
| <b>Blocknumber</b>  | – The compressed block where the first byte of the file’s data begins.  |
| <b>BlockOffset</b>  | – The number of bits that must be discarded before you reach the beginning of the block referred to by Blocknumber.                   |
| <b>InsideOffset</b> | – The number of bytes that must be discarded within the block referenced by Blocknumber to reach the first byte of this entry’s data. |
| <b>MemberLength</b> | – The bytes of data in this archived member.  |
| <b>LinkFlag</b>     | – Preserved link flag entry from the TAR header of this file.   |
| <b>DirFlag</b>      | – A Y/N character flag indicating whether this entry refers to a directory or not.  |
| <b>Mode</b>         | – Preserved permission bits from the TAR header of this file.   |
| <b>UID</b>          | – Preserved user ID from the TAR header of this file.   |
| <b>GID</b>          | – Preserved group ID from the TAR header of this file.  |
| <b>LinkTarget</b>   | – If the file referenced by this entry is a link this is path the link points to.   |

---

The **Bzip2\_blocks** table contains a list of all blocks in compressed BZip2 archives, as well as information required to extract them:

***Bzip2\_blocks:***

---

|                    |   |
|--------------------|---|
| <b>ArchiveID</b>   | – An integer that associates the table entry with an entry in the ArchiveList table.                                |
| <b>ArchiveName</b> | – The name of the archive file this entry exists inside.  |
| <b>Blocknumber</b> | – The compressed block within “ArchiveName” that this entry refers to.  |
| <b>BlockOffset</b> | – The number of bits that must be discarded before you reach the beginning of the block referred to by Blocknumber. |
| <b>BlockSize</b>   | – The number of uncompressed bytes that this block contains.  |

---



The **CompXZ** table contains a list of all members found in compressed TAR XZ archives, as well as the information required to extract them:

**CompXZ:**

---

|                     |   |
|---------------------|---|
| <b>FileID</b>       | – An integer identification number associated with an entry in the table.   |
| <b>ArchiveID</b>    | – An integer that associates the table entry with an entry in the ArchiveList table.  |
| <b>ArchiveName</b>  | – The name of the archive file this entry exists inside.  |
| <b>MemberName</b>   | – The name of the archived member this entry refers to. (ex. File.txt)  |
| <b>MemberPath</b>   | – The path to this archived member within the physical archive.<br>(ex. Dir_one/Dir_two)  |
| <b>Blocknumber</b>  | – The compressed block where the first byte of the file’s data begins.  |
| <b>BlockOffset</b>  | – Placeholder, currently unused as XZ contains this information in an index.  |
| <b>InsideOffset</b> | – The number of bytes that must be discarded within the block referenced by Blocknumber to reach the first byte of this entry’s data. |
| <b>MemberLength</b> | – The bytes of data in this archived member.  |
| <b>LinkFlag</b>     | – Preserved link flag entry from the TAR header of this file.   |
| <b>DirFlag</b>      | – A Y/N character flag indicating whether this entry refers to a directory or not.  |
| <b>Mode</b>         | – Preserved permission bits from the TAR header of this file.   |
| <b>UID</b>          | – Preserved user ID from the TAR header of this file.   |
| <b>GID</b>          | – Preserved group ID from the TAR header of this file.  |
| <b>LinkTarget</b>   | – If the file referenced by this entry is a link this is path the link points to.   |

---

The **CompXZ\_blocks** table contains a list of all blocks in compressed TAR XZ archives, as well as the information required to extract them:

**CompXZ\_blocks:**

---

|                    |   |
|--------------------|---|
| <b>ArchiveID</b>   | – An integer that associates the table entry with an entry in the ArchiveList table.                                |
| <b>ArchiveName</b> | – The name of the archive file this entry exists inside.  |
| <b>Blocknumber</b> | – The compressed block within “ArchiveName” that this entry refers to.  |
| <b>BlockOffset</b> | – The number of bits that must be discarded before you reach the beginning of the block referred to by Blocknumber. |
| <b>BlockSize</b>   | – The number of uncompressed bytes that this block contains.  |

## Appendix C: Raw Data From Testing

This section contains the raw data from the TAR Browser efficiency and performance tests. Evaluations were made based on the averages from this data. For more details regarding testing, refer to section 5: Evaluation.

---

Uncompressed TAR FUSE access time:

usr/share/aptdaemon/lintian-fatal.tags

```
-- trial1 -- access time: 0m0.432, UCPU time: 0m0, SCPU time: 0m0.020
-- trial2 -- access time: 0m0.087, UCPU time: 0m0, SCPU time: 0m0.012
-- trial3 -- access time: 0m0.076, UCPU time: 0m0, SCPU time: 0m0.012
-- trial4 -- access time: 0m0.073, UCPU time: 0m0, SCPU time: 0m0.008
-- trial5 -- access time: 0m0.080, UCPU time: 0m0, SCPU time: 0m0.012
```

usr/share/icons/Humanity/actions/24/media-record.svg

```
-- trial1 -- access time: 0m0.416, UCPU time: 0m0.004, SCPU time: 0m0.016
-- trial2 -- access time: 0m0.104, UCPU time: 0m0, SCPU time: 0m0.012
-- trial3 -- access time: 0m0.087, UCPU time: 0m0, SCPU time: 0m0.008
-- trial4 -- access time: 0m0.101, UCPU time: 0m0.004, SCPU time: 0m0.008
-- trial5 -- access time: 0m0.136, UCPU time: 0m0, SCPU time: 0m0.012
```

usr/share/cups/doc-root/help/ref-snmp-conf.html

```
-- trial1 -- access time: 0m0.362, UCPU time: 0m0.004, SCPU time: 0m0.012
-- trial2 -- access time: 0m0.094, UCPU time: 0m0, SCPU time: 0m0.012
-- trial3 -- access time: 0m0.106, UCPU time: 0m0., SCPU time: 0m0.016
-- trial4 -- access time: 0m0.079, UCPU time: 0m0.004, SCPU time: 0m0.004
-- trial5 -- access time: 0m0.091, UCPU time: 0m0, SCPU time: 0m0.016
```

usr/src/linux-headers-3.2.0-70/arch/arm/mach-mmp/Makefile.boot

```
-- trial1 -- access time: 0m0.385, UCPU time: 0m0.004, SCPU time: 0m0.016
-- trial2 -- access time: 0m0.096, UCPU time: 0m0.004, SCPU time: 0m0.008
-- trial3 -- access time: 0m0.113, UCPU time: 0m0.008, SCPU time: 0m0.004
-- trial4 -- access time: 0m0.111, UCPU time: 0m0.012, SCPU time: 0m0.012
-- trial5 -- access time: 0m0.099, UCPU time: 0m0.008, SCPU time: 0m0.008
```

usr/sbin/atd

```
-- trial1 -- access time: 0m0.361, UCPU time: 0m0.008, SCPU time: 0m0.012
-- trial2 -- access time: 0m0.075, UCPU time: 0m0.012, SCPU time: 0m0.008
-- trial3 -- access time: 0m0.057, UCPU time: 0m0.004, SCPU time: 0m0.012
-- trial4 -- access time: 0m0.056, UCPU time: 0m0.008, SCPU time: 0m0.004
-- trial5 -- access time: 0m0.074, UCPU time: 0m0.000, SCPU time: 0m0.016
```

Bzip2 TAR FUSE access time:

usr/share/aptdaemon/lintian-fatal.tags

```
-- trial1 -- access time: 0m0.584, UCPU time: 0m0.000, SCPU time: 0m0.024
-- trial2 -- access time: 0m0.233, UCPU time: 0m0.008, SCPU time: 0m0.012
-- trial3 -- access time: 0m0.233, UCPU time: 0m0.000, SCPU time: 0m0.012
-- trial4 -- access time: 0m0.222, UCPU time: 0m0.000, SCPU time: 0m0.012
-- trial5 -- access time: 0m0.261, UCPU time: 0m0.000, SCPU time: 0m0.008
```

```
usr/share/icons/Humanity/actions/24/media-record.svg
-- trial1 -- access time: 0m0.536, UCPU time: 0m0.000, SCPU time: 0m0.016
-- trial2 -- access time: 0m0.260, UCPU time: 0m0.004, SCPU time: 0m0.008
-- trial3 -- access time: 0m0.292, UCPU time: 0m0.004, SCPU time: 0m0.012
-- trial4 -- access time: 0m0.254, UCPU time: 0m0.004, SCPU time: 0m0.008
-- trial5 -- access time: 0m0.253, UCPU time: 0m0.004, SCPU time: 0m0.012
```

```
usr/share/cups/doc-root/help/ref-snmp-conf.html
-- trial1 -- access time: 0m0.503, UCPU time: 0m0.000, SCPU time: 0m0.016
-- trial2 -- access time: 0m0.251, UCPU time: 0m0.004, SCPU time: 0m0.008
-- trial3 -- access time: 0m0.225, UCPU time: 0m0.000, SCPU time: 0m0.016
-- trial4 -- access time: 0m0.239, UCPU time: 0m0.004, SCPU time: 0m0.020
-- trial5 -- access time: 0m0.226, UCPU time: 0m0.000, SCPU time: 0m0.012
```

```
usr/src/linux-headers-3.2.0-70/arch/arm/mach-mmp/Makefile.boot
-- trial1 -- access time: 0m0.502, UCPU time: 0m0.004, SCPU time: 0m0.008
-- trial2 -- access time: 0m0.252, UCPU time: 0m0.004, SCPU time: 0m0.004
-- trial3 -- access time: 0m0.238, UCPU time: 0m0.000, SCPU time: 0m0.012
-- trial4 -- access time: 0m0.256, UCPU time: 0m0.004, SCPU time: 0m0.012
-- trial5 -- access time: 0m0.210, UCPU time: 0m0.000, SCPU time: 0m0.012
```

```
usr/sbin/atd
-- trial1 -- access time: 0m0.390, UCPU time: 0m0.000, SCPU time: 0m0.012
-- trial2 -- access time: 0m0.156, UCPU time: 0m0.000, SCPU time: 0m0.020
-- trial3 -- access time: 0m0.117, UCPU time: 0m0.004, SCPU time: 0m0.008
-- trial4 -- access time: 0m0.135, UCPU time: 0m0.004, SCPU time: 0m0.016
-- trial5 -- access time: 0m0.127, UCPU time: 0m0.000, SCPU time: 0m0.016
```

XZ TAR FUSE access time:

```
usr/share/aptdaemon/lintian-fatal.tags
-- trial1 -- access time: 0m4.672, UCPU time: 0m0.000, SCPU time: 0m0.020
-- trial2 -- access time: 0m3.525, UCPU time: 0m0.000, SCPU time: 0m0.028
-- trial3 -- access time: 0m3.636, UCPU time: 0m0.000, SCPU time: 0m0.032
-- trial4 -- access time: 0m3.560, UCPU time: 0m0.004, SCPU time: 0m0.020
-- trial5 -- access time: 0m3.467, UCPU time: 0m0.000, SCPU time: 0m0.020
```

```
usr/share/icons/Humanity/actions/24/media-record.svg
-- trial1 -- access time: 0m3.915, UCPU time: 0m0.000, SCPU time: 0m0.036
-- trial2 -- access time: 0m3.339, UCPU time: 0m0.000, SCPU time: 0m0.020
-- trial3 -- access time: 0m3.348, UCPU time: 0m0.000, SCPU time: 0m0.020
-- trial4 -- access time: 0m3.376, UCPU time: 0m0.000, SCPU time: 0m0.024
-- trial5 -- access time: 0m3.337, UCPU time: 0m0.004, SCPU time: 0m0.020
```

```
usr/share/cups/doc-root/help/ref-snmp-conf.html
-- trial1 -- access time: 0m7.729, UCPU time: 0m0.000, SCPU time: 0m0.024
-- trial2 -- access time: 0m5.709, UCPU time: 0m0.004, SCPU time: 0m0.020
-- trial3 -- access time: 0m5.705, UCPU time: 0m0.000, SCPU time: 0m0.032
-- trial4 -- access time: 0m5.641, UCPU time: 0m0.000, SCPU time: 0m0.024
-- trial5 -- access time: 0m5.711, UCPU time: 0m0.008, SCPU time: 0m0.020
```

```
usr/src/linux-headers-3.2.0-70/arch/arm/mach-mmp/Makefile.boot
-- trial1 -- access time: 0m3.827, UCPU time: 0m0.000, SCPU time: 0m0.020
-- trial2 -- access time: 0m3.355, UCPU time: 0m0.004, SCPU time: 0m0.024
-- trial3 -- access time: 0m3.252, UCPU time: 0m0.000, SCPU time: 0m0.024
-- trial4 -- access time: 0m3.326, UCPU time: 0m0.000, SCPU time: 0m0.036
-- trial5 -- access time: 0m3.246, UCPU time: 0m0.000, SCPU time: 0m0.024
```

usr/sbin/atd

-- trial1 -- access time: 0m4.032, UCPU time: 0m0.000, SCPU time: 0m0.020  
-- trial2 -- access time: 0m3.579, UCPU time: 0m0.000, SCPU time: 0m0.028  
-- trial3 -- access time: 0m3.600, UCPU time: 0m0.004, SCPU time: 0m0.020  
-- trial4 -- access time: 0m3.653, UCPU time: 0m0.004, SCPU time: 0m0.028  
-- trial5 -- access time: 0m3.591, UCPU time: 0m0.004, SCPU time: 0m0.020

Uncompressed TAR "xvf" access time:

usr/share/aptdaemon/lintian-fatal.tags

-- trial1 -- access time: 0m38.731, UCPU time: 0m0.452, SCPU time: 0m5.612  
-- trial2 -- access time: 0m37.132, UCPU time: 0m0.376, SCPU time: 0m4.976  
-- trial3 -- access time: 0m36.924, UCPU time: 0m0.412, SCPU time: 0m5.104  
-- trial4 -- access time: 0m37.129, UCPU time: 0m0.408, SCPU time: 0m4.892  
-- trial5 -- access time: 0m36.593, UCPU time: 0m0.376, SCPU time: 0m4.956

usr/share/icons/Humanity/actions/24/media-record.svg

-- trial1 -- access time: 0m36.544, UCPU time: 0m0.392, SCPU time: 0m4.928  
-- trial2 -- access time: 0m36.997, UCPU time: 0m0.400, SCPU time: 0m5.044  
-- trial3 -- access time: 0m37.400, UCPU time: 0m0.316, SCPU time: 0m5.020  
-- trial4 -- access time: 0m36.818, UCPU time: 0m0.396, SCPU time: 0m4.936  
-- trial5 -- access time: 0m36.931, UCPU time: 0m0.296, SCPU time: 0m5.016

usr/share/cups/doc-root/help/ref-snmp-conf.html

-- trial1 -- access time: 0m37.064, UCPU time: 0m0.388, SCPU time: 0m4.800  
-- trial2 -- access time: 0m37.111, UCPU time: 0m0.388, SCPU time: 0m5.060  
-- trial3 -- access time: 0m37.219, UCPU time: 0m0.376, SCPU time: 0m4.972  
-- trial4 -- access time: 0m37.413, UCPU time: 0m0.380, SCPU time: 0m5.108  
-- trial5 -- access time: 0m36.925, UCPU time: 0m0.384, SCPU time: 0m4.896

usr/src/linux-headers-3.2.0-70/arch/arm/mach-mmp/Makefile.boot

-- trial1 -- access time: 0m37.435, UCPU time: 0m0.452, SCPU time: 0m4.848  
-- trial2 -- access time: 0m37.509, UCPU time: 0m0.360, SCPU time: 0m5.048  
-- trial3 -- access time: 0m37.335, UCPU time: 0m0.372, SCPU time: 0m5.132  
-- trial4 -- access time: 0m37.130, UCPU time: 0m0.376, SCPU time: 0m5.020  
-- trial5 -- access time: 0m36.848, UCPU time: 0m0.508, SCPU time: 0m4.884

usr/sbin/atd

-- trial1 -- access time: 0m37.273, UCPU time: 0m0.452, SCPU time: 0m4.964  
-- trial2 -- access time: 0m37.208, UCPU time: 0m0.344, SCPU time: 0m5.060  
-- trial3 -- access time: 0m36.815, UCPU time: 0m0.404, SCPU time: 0m5.080  
-- trial4 -- access time: 0m37.027, UCPU time: 0m0.364, SCPU time: 0m5.116  
-- trial5 -- access time: 0m37.076, UCPU time: 0m0.376, SCPU time: 0m5.088

Bzip2 TAR "xvf" access time:

usr/share/aptdaemon/lintian-fatal.tags

-- trial1 -- access time: 6m3.097, UCPU time: 2m14.676, SCPU time: 3m35.989  
-- trial2 -- access time: 6m3.973, UCPU time: 2m16.273, SCPU time: 3m35.285  
-- trial3 -- access time: 6m3.745, UCPU time: 2m15.872, SCPU time: 3m35.105  
-- trial4 -- access time: 6m3.568, UCPU time: 2m15.480, SCPU time: 3m35.665  
-- trial5 -- access time: 6m14.464, UCPU time: 2m16.929, SCPU time: 3m44.678

```
usr/share/icons/Humanity/actions/24/media-record.svg
-- trial1 -- access time: 6m2.755, UCPU time: 2m14.436, SCPU time: 3m36.009
-- trial2 -- access time: 6m6.213, UCPU time: 2m18.537, SCPU time: 3m35.989
-- trial3 -- access time: 6m2.124, UCPU time: 2m19.937, SCPU time: 3m30.733
-- trial4 -- access time: 6m2.328, UCPU time: 2m18.345, SCPU time: 3m31.641
-- trial5 -- access time: 6m12.305, UCPU time: 2m21.309, SCPU time: 3m39.118
```

```
usr/share/cups/doc-root/help/ref-snmp-conf.html
-- trial1 -- access time: 6m3.664, UCPU time: 2m17.717, SCPU time: 3m34.041
-- trial2 -- access time: 6m5.119, UCPU time: 2m21.729, SCPU time: 3m31.897
-- trial3 -- access time: 6m9.946, UCPU time: 2m20.269, SCPU time: 3m34.225
-- trial4 -- access time: 6m22.675, UCPU time: 2m23.861, SCPU time: 3m46.698
-- trial5 -- access time: 6m8.158, UCPU time: 2m23.853, SCPU time: 3m32.729
```

```
usr/src/linux-headers-3.2.0-70/arch/arm/mach-mmp/Makefile.boot
-- trial1 -- access time: 6m2.369, UCPU time: 2m17.853, SCPU time: 3m33.229
-- trial2 -- access time: 6m26.235, UCPU time: 2m27.197, SCPU time: 3m47.810s
-- trial3 -- access time: 6m9.972, UCPU time: 2m23.273, SCPU time: 3m35.669
-- trial4 -- access time: 6m6.674, UCPU time: 2m21.493, SCPU time: 3m34.033
-- trial5 -- access time: 6m31.318, UCPU time: 2m38.290, SCPU time: 3m41.206
```

```
usr/sbin/atd
-- trial1 -- access time: 6m7.512, UCPU time: 2m22.717, SCPU time: 3m33.249
-- trial2 -- access time: 6m7.936, UCPU time: 2m21.509, SCPU time: 3m35.253
-- trial3 -- access time: 6m11.742, UCPU time: 2m22.957, SCPU time: 3m37.446
-- trial4 -- access time: 6m8.946, UCPU time: 2m23.461, SCPU time: 3m34.261
-- trial5 -- access time: 6m13.042, UCPU time: 2m25.889, SCPU time: 3m36.214
```

XZ TAR "xvf" access time:

```
usr/share/aptdaemon/lintian-fatal.tags
-- trial1 -- access time: 3m16.310, UCPU time: 0m57.644, SCPU time: 2m12.384
-- trial2 -- access time: 3m20.769, UCPU time: 0m59.272, SCPU time: 2m14.392
-- trial3 -- access time: 3m23.074, UCPU time: 0m59.056, SCPU time: 2m17.117
-- trial4 -- access time: 3m21.311, UCPU time: 0m59.056, SCPU time: 2m15.648
-- trial5 -- access time: 3m17.723, UCPU time: 0m59.204, SCPU time: 2m12.560
```

```
usr/share/icons/Humanity/actions/24/media-record.svg
-- trial1 -- access time: 3m15.960, UCPU time: 0m57.664, SCPU time: 2m11.760
-- trial2 -- access time: 3m11.250, UCPU time: 0m56.832, SCPU time: 2m7.984
-- trial3 -- access time: 3m22.115, UCPU time: 0m59.128, SCPU time: 2m16.277
-- trial4 -- access time: 3m26.148, UCPU time: 1m2.868, SCPU time: 2m16.249
-- trial5 -- access time: 3m16.565, UCPU time: 0m58.544, SCPU time: 2m11.372
```

```
usr/share/cups/doc-root/help/ref-snmp-conf.html
-- trial1 -- access time: 3m10.887, UCPU time: 0m57.196, SCPU time: 2m7.480
-- trial2 -- access time: 3m19.312, UCPU time: 1m0.004, SCPU time: 2m12.520
-- trial3 -- access time: 3m14.712, UCPU time: 0m58.184, SCPU time: 2m10.172
-- trial4 -- access time: 3m22.909, UCPU time: 1m0.468, SCPU time: 2m15.584
-- trial5 -- access time: 3m15.404, UCPU time: 0m58.092, SCPU time: 2m9.488
```

```
usr/src/linux-headers-3.2.0-70/arch/arm/mach-mmp/Makefile.boot
-- trial1 -- access time: 3m15.457, UCPU time: 0m57.692, SCPU time: 2m8.748
-- trial2 -- access time: 3m10.222, UCPU time: 0m55.987, SCPU time: 2m8.028
-- trial3 -- access time: 3m22.229, UCPU time: 0m59.396, SCPU time: 2m14.360
-- trial4 -- access time: 3m19.743, UCPU time: 0m58.548, SCPU time: 2m13.040
-- trial5 -- access time: 3m17.262, UCPU time: 0m58.640, SCPU time: 2m10.616
```

usr/sbin/atd

-- trial1 -- access time: 3m12.333, UCPU time: 0m56.492, SCPU time: 2m8.732  
-- trial2 -- access time: 3m21.216, UCPU time: 0m59.148, SCPU time: 2m15.068  
-- trial3 -- access time: 3m30.296, UCPU time: 1m5.308, SCPU time: 2m17.041  
-- trial4 -- access time: 3m34.247, UCPU time: 1m5.584, SCPU time: 2m20.473  
-- trial5 -- access time: 3m17.870, UCPU time: 0m58.728, SCPU time: 2m11.508

## References

Collin, Lasse. "How Does One Do Random Access?" *SourceForge*. N.p., 11 June 2010. Web. 13 Feb. 2015. <<http://sourceforge.net/p/lzmautils/discussion/708858/thread/4d71ae25/>>.

- Lasse Collin, the developer of XZ compression, describes how random access to an XZ compressed file can be accomplished.

Collin, Lasse. "XZ – Random Access Reads." E-mail correspondence with developer, November 2014.

- Email discussion regarding XZ random access reads. Quotes used with permission.

Davidson, Sean. Initial problem proposal. E-mail correspondence, November 2014.

- Sean Davidson proposed the problem of access times of members within large archives. This was discussed in person and through email.

Eggert, Paul, Mike Haertel, David Hayes, Richard Stallman, and Len Tower. "diff(1) - Linux Manual Page." *diff(1) - Linux Manual Page*. Free Software Foundation, Inc., Feb. 2015. Web. 27 Feb. 2015. <<http://man7.org/linux/man-pages/man1/diff.1.html>>

- Linux man page for the "diff" command. Describes the program's functionality, usage, and options.

Ellingwood, Justin. "An Introduction to File Compression Tools on Linux Servers." *An Introduction to File Compression Tools on Linux Servers*. DigitalOcean.com, 15 Apr. 2014. Web. 05 Sept. 2014. <<https://www.digitalocean.com/community/tutorials/an-introduction-to-file-compression-tools-on-linux-servers>>.

- Contains information on the compression utilities that come standard on Linux.

Free Software Foundation, Inc. "GNU Tar: An Archiver Tool." Free Software Foundation, Inc, 28

July 2014. Web. 25 Feb. 2015.

<[http://www.gnu.org/software/tar/manual/html\\_section/tar.html](http://www.gnu.org/software/tar/manual/html_section/tar.html)>

- Official documentation for the TAR archive format and archive program. Contains information regarding block size, header details, program options, etc.

"Format of Tar Archives." *PTC MKS Toolkit 9.6 Documentation Build 9*. N.p., n.d. Web. 07 Sept. 2014. <<https://www.mkssoftware.com/docs/man4/tar.4.asp>>.

- Contains information on TAR archive header formats.

Linux Man-pages Project. "time(1) - Linux Manual Page." time(1) - Linux Manual Page. Linux

Man-pages Project, 21 Feb. 2015. Web. 25 Feb. 2015.

<<http://man7.org/linux/man-pages/man1/time.1.html>>

- Linux man page for the "time" command. Describes the program's functionality, usage, and options.

Meyering, Jim. "od(1) - Linux Manual Page." od(1) - Linux Manual Page. Free Software Foundation, Inc., Feb. 2015. Web. 25 Feb. 2015.

<<http://man7.org/linux/man-pages/man1/od.1.html>>

- Linux manual page for the "od" command. Describes the program's functionality, usage, and options.

"MySQL 5.7 Reference Manual." *MySQL*. Oracle, n.d. Web. 13 Feb. 2015.

<<http://dev.mysql.com/doc/refman/5.7/en/index.html>>.

- Describes the functionality of the C API for MySQL.

Oseberg, Terje. "How FUSE Works." *SourceForge*. N.p., n.d. Web. 13 Feb. 2015.

<<http://fuse.sourceforge.net/doxygen/index.html>>.

- Details the inner workings of FUSE from both a system and source code level.

Seward, Julian. "Bzip2 and Libbzip2." Bzip2 : Home. N.p., 2015. Web. 28 Feb. 2015.

<<http://www.bzip.org/index.html>>

- Home page for the BZip2 project. Includes links to documentation.

Taylor, James. "Requesting Permission to Use Code." E-mail correspondence with developer.

14 Oct. 2014.



- E-mail to James Taylor requesting permission to implement Seek-BZip2 into TAR Browser. Permission granted, as Taylor considers his project to be public domain.

Taylor, James. "SeekingInBzip2Files." *BitBucket*. N.p., 26 July 2014. Web. 13 Feb. 2015. <[https://bitbucket.org/james\\_taylor/bx-python/wiki/IO/SeekingInBzip2Files](https://bitbucket.org/james_taylor/bx-python/wiki/IO/SeekingInBzip2Files)>.

- Describes the functionality of the Bzip-seek program developed by James Taylor at The Taylor Lab at John Hopkins University. Also contains a description of the bzip2 compression format, explicitly stating that a block actually contains an unknown amount of completely uncompressed data due to run length encoding.

The Tukaani Project. "The .xz File Format." *The .xz File Format*. The Tukaani Project, 27 Aug.

2009. Web. 25 Feb. 2015. <<http://tukaani.org/xz/xz-file-format-1.0.4.txt>>

- Official documentation for the XZ file format and XZ compression methods.

The Tukaani Project. "XZ Utils." *XZ Utils*. The Tukaani Project, 2015. Web. 27 Feb. 2015.

<<http://tukaani.org/xz/>>

- Overview of XZ Utils. "XZ Utils is free general-purpose data compression software with high compression ratio... XZ Utils are the successor to LZMA Utils."