CS 4900/5900 Computer Security and Forensics

Programming Assignment 2

Analysis of a Linux Ext2 File System and Data Retrieval/Extraction

This document: Part 2: EXT2_BROWSER Tool

Due: Friday, March 29, 2019 (at 11:59pm)

EXT2_BROWSER TOOL:

This tool will require you to implement different functions, including methods to resolve path names, search through directories, read inodes for directories and regular files, and determine the data blocks associated with files and directories.

TECHNICAL DETAILS:

You were provided with a header file ("ext2fs.h") with all the necessary ext2 structures for this assignment: ext2_superblock, ext2_inode, ext2_group_desc, ext2_dir_entry_2.

Your tool will be called ext2_browser and should be called with two arguments (ext2_browser <ext2_file> <pathname>):

- ext2_file is the file that contains the ext2 file system
- pathname is the path to be resolved. This could be a path that ends in a file (example: /home/drews/testfile.txt) or a path that ends in a directory(example: /home/drews)

The first argument is the same as for the tool in the first part of this assignment. The pathname can be either referring to a regular file or a directory. If the path is for a regular file, you will output the contents of this file. If the path is for a directory, you will produce a listing of the contents of the directory that is the equivalent of the output of the Unix command 'ls -l'.

When resolving pathnames, I would recommend you utilize the C-standard library function 'strtok'. This function should be called with a token character '/'. It can be used to split a path like '/dir1/dir2/file' into the different components ('dir1', 'dir2', 'file').

Any pathname must always start with the root slash '/'. Remember that every file and directory in an ext2 file system is represented by exactly one inode.

- If the file is a regular file, this inode will contain information like file size, file permissions, owner, time of last modification, time of creation, time of last access and the location of the data blocks for this file.
- If the file is a directory ('directories are files'!), the inode for the directory contains the same kind of information as for a regular file, except that the data blocks (whose locations are listed in the inode) for a directory

When resolving a pathname like '/dir1/dir2/file.txt', you always start with the inode for the root directory '/'. This inode should always be the inode with the number 2. Note that inode numbers are used to index into the inode table. The inode table of an ext2 file system is a (very large) table (array) of inode structures. The inode table is divided (in equal sized parts) between the block groups of an ext2 file system (see lecture notes for more details). (**Note: the inode table index starts at 1! That means that index 1 is the first inode, not the second!**).

Back to the above example, to resolve '/dir1/dir2/file.txt', you will first need to access inode number 2 (i.e., the second inode). This inode describes the root directory '/' of the ext2 file system. The inode will point at the data block(s) for the directory. These blocks will contain directory records (see lecture slides for more details on directories) that are organized in a linked linear list. In our example, you will iterate through the individual directory records and try to find a match for the first sub-directory in our path (i.e., 'dir1'). If a match is found, the directory record contains the inode for this directory. If it is not found, the path cannot be resolved (and your tools should output an error message). In case of a match, you can continue resolving the pathname by accessing the inode for 'dir1' and search for a match for 'dir2'. Eventually, if all the sub-directories can be successfully resolved, you will find a directory record for the regular file 'file.txt'. This directory record contains the inode number for this regular file. This inode then contains the meta data for 'file.txt', including the pointers to the data blocks for this file.

REQUIREMENTS:

To summarize, your tool should be called using "./ext2_browser <ext2_file> <path>". It will resolve the <path> within the specified ext2 file <ext2_file>. A valid <path> should always start with a root slash '/'. If the path ends in a directory, you will produce an output equivalent to the Unix command 'ls -l'. If the path ends in a file name, output the contents of that file (you can simply use something like "printf("%c", buf[i])" in a loop, where buf[i] is the I'th data byte of the file).

Example output:

The following command displays the contents of the root directory of an example ext2 file system:

```
drews@pu2:~/extproject$ ./ext2_browser filesystems/fspart /

drwxrwxrwx      root         3072      Feb 15 19:59    .

drwxrwxrwx      root         3072      Feb 15 19:59    ..
```

```
drwx------      root          12288      Feb 14 08:53    lost+found

drwxrwxr-x      1000           1024      Feb 15 14:42    testdir1

-rw-rw-r--      1000             27      Feb 14 13:29    testfile2.txt

-rw-rw-r--      1000             33      Feb 14 13:29    testfile3.txt

drwxrwxr-x      1000           1024      Feb 15 14:42    testdir2

-rw-rw-r--      1000          20480      Feb 15 17:45    bigfile
```

The following command displays the contents of the sub-directory 'testdir2':

```
drews@pu2:~/extproject$ ./ext2_browser filesystems/fspart /testdir2

drwxrwxr-x      1000           1024      Feb 15 14:42    .

drwxrwxrwx      root           3072      Feb 15 19:59    ..

-rwxrwxr-x      1000           8600      Feb 14 18:48    a.out

-rw-rw-r--      1000           5709      Feb 14 18:07    main.c

-rwxrwxr-x      1000           8600      Feb 14 18:49    hw

-rw-rw-r--      1000         102400      Feb 15 14:42    randtest2
```

The following command displays the contents of the file '/testfile2.txt':

```
drews@pu2:~/until2010/extproject$ ./ext2_browser filesystems/fspart
/testfile2.txt

222222222

222222

2222

22

2
```

HINTS:

<u>READING INODES</u>

The following code shows how to read in a given inode (inumber) from an ext2 file system. The example assumes that the block descriptor information (see Part 1 of this project) is stored in an array 'ext2_gd':

```
// note: inodes_per_group are from the superblock
```

```
unsigned int i_group = ( inumber - 1 ) / inodes_per_group;

unsigned int i_index = ( inumber - 1 ) % inodes_per_group;

unsigned int i_offset = block_size * ext2_gd[i_group].bg_inode_table +
(i_index * inode_size);


struct ext2_inode ext2_in;

fseek(fptr, i_offset, SEEK_SET);

fread((char*)&ext2_in, sizeof(struct ext2_inode), 1, fptr);
```

DIRECTORY LISTING

The inode contains important information (meta data) for a file or directory. Three time/date's are stored in the inode: time of file creation ('i_atime'), time of last modification (i_ctime), time of last access (i_mtime). These times are stored in a time format known as "time_t". It is usefule to convert these "time_t" types to another Unix time format known as 'struct tm'. This format makes it easier to print the times/dates in a user-friendly format. The following code demonstrates how to do this:

```
#include <time.h>

struct ext2_inode in_t; // I assume this contains the inode

time_t a_rawtime = in_t.i_atime;

char date[255]; // buffer to store time/date string

struct tm* a_time_tm;

atime_tm = localtime(&a_rawtime); // convert from time_t to tm format

strftime(date, 255, "%b %d %R", atime_tm);

printf("%s\n", date);
```

The access permissions for a file are stored in the 'i_mode' member of an inode structure. They represent the 'rwx' rights that are displayed by the Unix command 'ls -l'. See the lecture slides for more details on how to interpret these file permissions. The following code fragment shows conceptually how to test for individual file permissions:

```
const unsigned int rights[9]={S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP,
S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH};

const char* crights[3] = {"r", "w", "x"};

for(i = 0; i < 9; i++)

          {

            if( in_t.i_mode & rights[i] )

            {

              printf("%s", crights[i % 3]);

            }else

            {

              printf("-");

            }

          }
```

LARGE FILES

To get the full 100% for this part of the project, I expect your tool to support files up to a size that requires single indirect and double indirect blocks (see lecture slides for more information). For a block size of 1024 bytes, this means you should support a file size of up to:

$$12 \cdot 1024 + \frac{1024}{4} \cdot 1024 + \frac{1024}{4} \cdot \frac{1024}{4} \cdot 1024 \approx 67Mb$$

For a block size of 4096 bytes, this means you should support a file size of up to:

$$12 \cdot 4096 + \frac{4096}{4} \cdot 4097 + \frac{4096}{4} \cdot \frac{4096}{4} \cdot 4096 \approx 4.3\text{Gb}$$

Note that the number 12 in the above equations represents the number of direct block pointers. The numbers $\frac{1024}{4}$ (or $\frac{4096}{4}$) represent the number of block pointers that can be stored in a given block of size 1K or 4K.

REFERENCES:

- Lecture slides (ext2 file system slides)
- Header file with ext2 structures: ext2_superblock, ext2_inode, ext2_group_desc, ext2_dir_entry_2 (In a directory /home/drews/ext2_filesystems on the prime machines)
- I will put my own tool (ext2_browser) into the directory '/home/drews/ext2_filesystems'
- Ext2 files: I will put a number of different ext2 file systems in my home directory on the prime machines under: /home/drews/ext2_filesystems
- Technical documentation at www.osdev.org:
  https://wiki.osdev.org/Ext2

SUBMISSION:

Submit your program using the submission tool again. Submit your program by first changing into your project directory and then using the following command (on **p2.cs.ohio.edu**):

p2:> /home/drews/bin/490submit 3

If you need to overwrite a submission use the '-f' switch:

p2:> /home/drews/bin/490submit -f 3

Note that you will need a makefile. Also do not submit an ext2 file system file. This won't work since the submission tool has a maximum file size that can't be exceeded!