PA06

Grace, Jayson
jaysong@unm.edu

Salas, Dominic
dsalas9@unm.edu

Saturday 14th November, 2015

Jayson Grace, Dominic Salas

Problem 1a

To identify a file by its magic number, we created this code:

```
**
 * @file    filetype.c
 * @author Jayson Grace (jaysong@unm.edu)
 * @author Dominic Salas (dominic.salas@gmail.com)
 * @date    11/05/2015
 * @brief  filetype program to identify files without
    dependence on its having an extension
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef int bool;
enum { false, true };

bool DEBUG = false;

/**
 * @brief Get the magic number from the input file
 * @param file The file to get the magic number from
 * @return Magic number
 */
unsigned char* getMagicNumber(FILE *file)
{
  long fileSize;
  unsigned char *fileInfo;
  fseek(file, 0, SEEK_END);
  fileSize = ftell(file);
  rewind(file);
  fileInfo = (char *) malloc(fileSize + 1);


  fread(fileInfo, sizeof(char), 4, file);
```

```
  if (DEBUG)
  {
    printf("%02hhx%02hhx%02hhx%02hhx\n", fileInfo[0],
       fileInfo[1], fileInfo[2], fileInfo[3]);
    if (fileInfo[0] == 0x1f)
    {
      printf("%x\n", fileInfo[0]);
    }
  }
  return fileInfo;
}

/**
 * @brief Get the file type based on its magic number
 * @param fileInfo The information about the file
 */
void getFileType(char* fileInfo)
{
  int i;
  char pdf[4] = {0x25,0x50,0x44,0x00};
  char jpg[4] = {0xff,0xd8,0xff,0x00};
  char elf[4] = {0x7f,0x45,0x4c,0x00};
  char tar[4] = {0x1f,0x8b,0x08,0x00};
  char sh[4] = {0x23,0x21,0x2f,0x00};
  char result[4];
  for (i = 0; i < 3; i++)
  {
    result[i] = fileInfo[i];
    if (DEBUG)
      printf("%c\n", fileInfo[i]);
  }
  result[3] = 0x00;
  if (strcmp(pdf, result) == 0)
    printf("PDF detected\n");
  else if (strcmp(sh, result) == 0)
    printf("sh detected\n");
  else if (strcmp(elf, result) == 0)
    printf("ELF detected\n");
  else if (strcmp(tar, result) == 0)
    printf("tar.gz detected\n");
  else if (strcmp(jpg, result) == 0)
```

```
    printf("jpg detected\n");
  else
    printf("Invalid file type input!\n");
}

/**
  @brief Entry into program
  @param argc Argument count
  @param argv Argument vector
  @return Success (or lackthereof) of program's
     execution
  */
int main(int argc, char **argv)
{
  char* fileInfo;
  if ( argc != 2 )
  {
    printf( "usage: %s filename", argv[0] );
  }
  else
  {
    FILE *file = fopen( argv[1], "r" );

    if ( file == 0 )
    {
      printf( "Could not open file\n" );
      return EXIT_FAILURE;
    }
    else
    {
      fileInfo = getMagicNumber(file);
      getFileType(fileInfo);
    }
  }
  free(fileInfo);
  return EXIT_SUCCESS;
}
```

Jayson Grace, Dominic Salas

To get a consistent set of tests to run, we created this bash script:

```
#!/bin/bash
#
# runTests.sh
#
# Run test for problem 1 of pa06
#
# Usage: bash runTests.sh
#
# Jayson Grace, jayson.e.grace@gmail.com, 11/8/2015
#
# Last update 11/8/2015 by Jayson Grace, jayson.e.
   grace@gmail.com
#

compile()
{
  reset
  gcc -g filetype.c -o filetype
}

pdfTest()
{
  wget http://help.adobe.com/en_US/reader/using/
     reader_X_help.pdf
  ./filetype reader_X_help.pdf
  mv reader_X_help.pdf reader_X_help
  ./filetype reader_X_help
  rm reader_X_help
}

jpgTest()
{
  wget https://upload.wikimedia.org/wikipedia/en/1/12/
     Never-Let-Me-Down.jpg
  ./filetype Never-Let-Me-Down.jpg
  mv Never-Let-Me-Down.jpg Never-Let-Me-Down
  ./filetype Never-Let-Me-Down
```

```
  rm Never-Let-Me-Down
}

tarTest()
{
  wget http://ftp.gnu.org/gnu/tar/tar-1.28.tar.gz
  ./filetype tar-1.28.tar.gz
  mv tar-1.28.tar.gz tar-1.28
  ./filetype tar-1.28
  rm tar-1.28
}

shTest()
{
  ./filetype runTests.sh
  cp runTests.sh runTests
  ./filetype runTests
  rm runTests
}

elfTest()
{
  ./filetype /bin/ls
}

compile
pdfTest
jpgTest
tarTest
shTest
elfTest
```

The output of the bash script is as follows:

```
Resolving help.adobe.com (help.adobe.com)...
    23.3.12.17, 23.3.12.10
Connecting to help.adobe.com (help.adobe.com)
    |23.3.12.17|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1930650 (1.8M) [application/pdf]
Saving to: reader_X_help.pdf

100%
[=====================================================>]
1,930,650    2.85MB/s    in 0.6s

2015-11-08 19:52:28 (2.85 MB/s) - reader_X_help.
    pdf saved [1930650/1930650]

PDF detected
PDF detected
--2015-11-08 19:52:28-- https://upload.wikimedia.org/
    wikipedia/en/1/12/Never-Let-Me-Down.jpg
Resolving upload.wikimedia.org (upload.wikimedia.org)
    ... 208.80.153.240, 2620:0:860:ed1a::2:b
Connecting to upload.wikimedia.org (upload.wikimedia.
    org)|208.80.153.240|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 18347 (18K) [image/jpeg]
Saving to: Never-Let-Me-Down.jpg

100%
[=====================================================>]
 18,347       68.7KB/s    in 0.3s

2015-11-08 19:52:28 (68.7 KB/s) - Never-Let-Me-Down.
    jpg saved [18347/18347]

jpg detected
jpg detected
```

```
--2015-11-08 19:52:28--  http://ftp.gnu.org/gnu/tar/tar
    -1.28.tar.gz
Resolving ftp.gnu.org (ftp.gnu.org)... 208.118.235.20,
    2001:4830:134:3::b
Connecting to ftp.gnu.org (ftp.gnu.org)
    |208.118.235.20|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3877043 (3.7M) [application/x-gzip]
Saving to:  tar -1.28.tar.gz

100%
[=====================================================>]
 3,877,043   2.13MB/s   in 1.7s

2015-11-08 19:52:30 (2.13 MB/s) -  tar -1.28.tar.gz
    saved [3877043/3877043]

tar.gz detected
tar.gz detected
sh detected
sh detected
ELF detected
```

Problem 1b

Magic numbers are used to specify what type of data is in a binary file. Because text files aren't binary files, they do not have a magic number. As a result, a file with a magic number can't be a text file.

Jayson Grace, Dominic Salas

Problem 2

The following code makes up sizewatch.c:

```
/**
 * @file    sizewatch.c
 * @author Jayson Grace (jaysong@unm.edu)
 * @author Dominic Salas (dominic.salas@gmail.com)
 * @date    11/08/2015
 * @brief  Daemon to keep track of size changes that
    occur in a file over a 3 minute period.
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <time.h>
#include <syslog.h>

typedef int bool;
enum { false, true };

FILE * outputLog;

int fprintf(FILE *stream, const char *format, ...);

void getTime()
{
  time_t currentTime;
  struct tm * timeInfo;
  time(&currentTime);
  timeInfo = localtime(&currentTime);
  fprintf(outputLog, "[%d-%d-%d %d:%02d:%02d] ", (1 +
     timeInfo->tm_mon), timeInfo->tm_mday,  (1900+
     timeInfo->tm_year), timeInfo->tm_hour, timeInfo->
     tm_min, timeInfo->tm_sec);
```

```
}

/**
 * @brief Entry into program
 * @param argc Argument count
 * @param argv Argument vector
 * @return Success (or lackthereof) of program's
    execution
 */
int main(int argc, char* argv[])
{
  struct stat fileInfo;

  pid_t pid, sid;

  // Fork off parent
  pid = fork();

  // General error check for failure to generate child
     pid
  if (pid < 0)
  {
    exit(EXIT_FAILURE);
  }

  // Error check to determine if we can exit parent
     process
  if (pid > 0)
  {
    exit(EXIT_SUCCESS);
  }

  // Change file mode mask
  umask(0);

  // Open output log
  outputLog = fopen ("output.txt", "w+");

  // Create new SID for child process
  sid = setsid();
  if (sid < 0)
```

```
{
  exit(EXIT_FAILURE);
}

if(stat(argv[1], &fileInfo) == -1)
{
  fprintf(outputLog, "Error: Input file not found!\n
     ");
  return -1;
}

getTime();
fprintf(outputLog, "Monitoring File: %s\n", argv[1]);

off_t fileSize;
fileSize = fileInfo.st_size;
int i;

close(STDIN_FILENO);
close(STDOUT_FILENO);
close(STDERR_FILENO);

getTime();
fprintf(outputLog, "Initial Size: %d\n", (int)
   fileSize);

for(i = 0; i < 18; i++)
{
  if(stat(argv[1], &fileInfo) == -1)
  {
    fprintf(outputLog, "Error: The file has been
       moved or deleted!\n");
    return -1;
  }
  if((fileSize != fileInfo.st_size))
  {
    getTime();
    fprintf(outputLog, "Size Changed: Old: %d, New =
       %d\n", (int)fileSize, (int)fileInfo.st_size);
    fileSize = fileInfo.st_size;
  }
```

```
    sleep(10);
  }

  getTime();
  fprintf(outputLog, "Monitoring File terminated, BYE.\
    n");
  fclose(outputLog);
  return EXIT_SUCCESS;
}
```

While it is running, we make several modifications to test.txt.

The output is as follows:

```
[11-10-2015 10:28:28] Monitoring File: test.txt
[11-10-2015 10:28:28] Initial Size: 733
[11-10-2015 10:28:58] Size Changed: Old: 733, New = 371
[11-10-2015 10:29:38] Size Changed: Old: 371, New = 728
[11-10-2015 10:29:48] Size Changed: Old: 728, New = 668
[11-10-2015 10:30:28] Size Changed: Old: 668, New = 618
[11-10-2015 10:30:58] Size Changed: Old: 618, New = 774
[11-10-2015 10:31:18] Size Changed: Old: 774, New = 505
[11-10-2015 10:31:28] Monitoring File terminated, BYE.
```

Problem 3

To get started we created a bash script createVolume.sh, which creates our volume for us:

```
#!/bin/bash
#
# createVolume.sh
#
# Create a disk volume for problem 3 of pa06.
#
# Usage: bash createVolume.sh
#
# Jayson Grace, jayson.e.grace@gmail.com, 11/10/2015
#
```

```
# Last update 11/10/2015 by Jayson Grace, jayson.e.
    grace@gmail.com
#

dd if=/dev/zero of=./Jayson-Grace_Dominic-Salas_437disk
     bs=1k count=100
mkfs -t ext2 -N 16 -F Jayson-Grace_Dominic-
    Salas_437disk
```

Jayson Grace, Dominic Salas

Problem 3a

The output from running the command dumpe2fs Jayson-Grace_Dominic-Salas_437disk gives us this output, which includes the required information:

```
dumpe2fs 1.42.9 (4-Feb-2014)
Filesystem volume name:   <none>
Last mounted on:          <not available>
Filesystem UUID:          fde80887-9401-40d6-92fe
   -444052c81b2b
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      ext_attr resize_inode
   dir_index filetype sparse_super
Filesystem flags:         signed_directory_hash
Default mount options:    user_xattr acl
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              16
Block count:              100
Reserved block count:     5
Free blocks:              79
Free inodes:              5
First block:              1
Block size:               1024
Fragment size:            1024
Blocks per group:         8192
Fragments per group:      8192
Inodes per group:         16
Inode blocks per group:   2
Filesystem created:       Tue Nov 10 14:24:00 2015
Last mount time:          n/a
Last write time:          Tue Nov 10 14:24:00 2015
Mount count:              0
Maximum mount count:      -1
Last checked:             Tue Nov 10 14:24:00 2015
Check interval:           0 (<none>)
Reserved blocks uid:      0 (user root)
```

```
Reserved blocks gid:       0 (group root)
First inode:               11
Inode size:                128
Default directory hash:    half_md4
Directory Hash Seed:       2b892f10-82d6-4a43-aa54-83117
   f6b0e83


Group 0: (Blocks 1-99)
  Primary superblock at 1, Group descriptors at 2-2
  Block bitmap at 3 (+2), Inode bitmap at 4 (+3)
  Inode table at 5-6 (+4)
  79 free blocks, 5 free inodes, 2 directories
  Free blocks: 21-99
  Free inodes: 12-16
```

Problem 3b

In order to locate the root directory and display it's content we created the bash script 3b.sh:

```
#!/bin/bash
#
# 3b.sh
#
# Run code for problem 3b of pa06.
#
# Usage: bash 3b.sh
#
# Jayson Grace, jayson.e.grace@gmail.com, 11/10/2015
#
# Last update 11/10/2015 by Jayson Grace, jayson.e.
   grace@gmail.com
#

debugfs Jayson-Grace_Dominic-Salas_437disk -f 3
   bCommands
```

This script uses the commands listed in the file 3bCommands to get us our desired input through the debugfs utility. The 3bCommands file has the following commands:

```
imap ./
bd -f ./ 0
```

The subsequent output from running the 3b.sh bash script includes where the root directory is:

```
debugfs: imap ./
Inode 2 is part of block group 0
        located at block 5, offset 0x0080
```

as well as the content:

```
debugfs: bd -f ./ 0
0000  0200 0000 0c00 0102 2e00 0000 0200 0000
    ................
0020  0c00 0202 2e2e 0000 0b00 0000 e803 0a02
    ................
0040  6c6f 7374 2b66 6f75 6e64 0000 0000 0000  lost+
   found......
0060  0000 0000 0000 0000 0000 0000 0000 0000
    ................
*
```

Problem 3c

To create the text file, we created a python script, gen_abc.py:

```
#!/usr/bin/python

print 'A' * 800 + 'B' * 800 + 'C' * 800
```

Next, we create a copy of the disk before we right this information to it:

```
cp Jayson-Grace_Dominic-Salas_437disk Jayson-
   Grace_Dominic-Salas_437disk-before
```

Next, we created a bash script to automate the rest of the work, 3c.sh:

```
#!/bin/bash
#
# 3c.sh
#
# Run code for problem 3c of pa06.
#
```

```
# Usage: bash 3c.sh
#
# Jayson Grace, jayson.e.grace@gmail.com, 11/10/2015
#
# Last update 11/10/2015 by Jayson Grace, jayson.e.
   grace@gmail.com
#

python gen_abc.py > abc

debugfs -w Jayson-Grace_Dominic-Salas_437disk -f 3
   cCommands

rm abc
```

This was accompanied by 3cCommands:

```
write abc abc
```

This command was used to copy abc to the disk image:

```
debugfs: write abc abc
```

We can see from comparing the before and after images that the after image now has this at the end:

```
*
0005400 4141 4141 4141 4141 4141 4141 4141 4141
*
0005720 4242 4242 4242 4242 4242 4242 4242 4242
*
0005a40 4343 4343 4343 4343 4343 4343 4343 4343
*
0005d60 000a 0000 0000 0000 0000 0000 0000 0000
0005d70 0000 0000 0000 0000 0000 0000 0000 0000
```

Which in hex converts to our A's, B's and C's. We can also see with the hex dump the second block of data, which looks like this:

```
*
00004e0 0000 0000 0000 0000 0000 0000 d067 9e67
00004f0 5d81 f34c c185 744d b7d1 47af 0001 0000
0000500 000c 0000 0000 0000 937d 5647 0000 0000
```

```
0000510 0000 0000 0000 0000 0000 0000 0000 0000
*
```

Problem 3d

To write to the disk, we first opened debugfs with the -w parameter for our disk image. Next, we created multiple directories using mkdir. After making the fourth directory, we received the following error message: ext2fs_mkdir: Could not allocate inode in ext2 filesystem.

Upon creating the volume we have 5 open inodes to work with, because we specified that it be created with 16 inodes. Each file added to the disk uses an inode. Subsequently, when we wrote the abc file to the disk, we used one of the five remaining inodes. Each time we created a new directory, we were also using an inode. Once all of the inodes are used up, we are unable to create any new files on the volume.

Problem 3e

In order to run the code for catfilev, I have the 3e.sh script to mount the volume we've created:

```
#!/bin/bash

sudo mkdir /media/mount
sudo mount -t ext2 Jayson-Grace_Dominic-Salas_437disk /
   media/mount
```

The code for catfilev is as follows:

```
/**
 * @file    catfilev.c
 * @author Jayson Grace (jaysong@unm.edu)
 * @author Dominic Salas (dominic.salas@gmail.com)
 * @date    11/09/2015
 * @brief Print out content of file specified in mini
    file system by its inode number.
 * @params Number of inode; Path to directory with
    desired file to read
 */

#include <stdio.h>
```

```
#include <stdlib.h>
#include <dirent.h>
#include <string.h>

typedef int bool;
enum { false, true };

bool DEBUG = true;

/**
 * @brief Entry into program
 * @param argc Argument count
 * @param argv Argument vector
 * @return Success (or lackthereof) of program's
    execution
 */
int main(int argc, char **argv)
{
  if(argc == 1)
  {
    printf("Usage: %s [directory]\n", *argv);
    exit(0);
  }

  struct dirent *file;
  int inum = atoi(argv[1]);
  DIR *directory = opendir(argv[2]);
  while ((file = readdir(directory)) != NULL)
  {
    if (DEBUG)
    {
      printf("inode %d is for file %s\n", file->d_ino,
         file->d_name);
    }

    if (file->d_ino == inum)
    {
      int c;
      FILE *epicFile;

      char* filepath;
```

```
      // Add space for the new string
      filepath = malloc(strlen(argv[2])+1+4);
      // copy the path of the file into filepath
      strcpy(filepath, argv[2]);
      // add the name of the file to filepath
      strcat(filepath, file->d_name);

      epicFile = fopen(filepath, "r");
      if (epicFile)
      {
        while ((c = getc(epicFile)) != EOF)
          putchar(c);
        fclose(epicFile);
      }
    }
  }

  closedir(directory);
  return EXIT_SUCCESS;
}
```

The output from running this code with the parameters 12 (which corresponded to the location of the file abc) and /media/mount gave us the following output with our debug statement turned on:

```
node 16 is for file dir3
inode 12 is for file abc
800 A B and C's (For the sake of the grader, I'm not
   going to put 2400 characters in this code block, you
    know what's in here - it's the output for abc.)
inode 15 is for file dir2
inode 13 is for file yeah
inode 11 is for file lost+found
inode 2 is for file .
inode 14 is for file dir1
inode 2 is for file ..
```