CS342 Operating Systems – Fall 2022
Project #4 – Linux ext2 file system

Assigned: Dec 4, 2022.
Due date: Dec 23, 2022, 23:59. Document version: 1.0

This project will be done in groups of two students. You can do it individually as well. You will program in C/Linux. Programs will be tested in Ubuntu Linux. Please start as soon as possible.

Part A (80 pts):

In this project you will develop a program that will access a disk in **raw mode** (*without* mounting it), and will print out some information from it. A **regular Linux file** will be acting as a **disk**. You will be given the disk (a Linux file). The disk is formatted with **ext2** Linux file system. It has just one directory in it: the root directory. There are no sub-directories in the root directory. All files in the root directory are regular files. The current Linux file system is not ext2. Since it is more complex than ext2, in this project you will work with ext2.

Your program will be called as `diskprint` and it will be invoked with the following parameters.

```
diskprint  DISKNAME
```

DISKNAME is the name of the regular Linux file that is acting as the disk. It will have ext2 file system in it. The block size is **4 KB**.

Your program will open the file DISKNAME and will access it block by block. For that you can use the `lseek()` and `read()` system calls. The `lseek()` system call allows you to set the file position pointer to anywhere (offset) in the file. Next read operation will take place from that position. Hence, given the block number, you can use `lseek()` to jump to the beginning of that block. For example, block 0 of the disk starts at offset 0 of the disk (Linux file). Block 1 starts at offset 4096, and so on. Then you can read the block into a buffer in memory. A buffer is simply an array of bytes (characters) that you will define in your program statically or dynamically. The buffer size should be at least 4 KB (block size).

Your program will parse the on-disk data structures of the ext2 file system and will print out the following information.

1. Print **super block** information. For this you need to access the superblock of the ext2 filesystem in the disk. Please read ext2 documentation to learn more about superblock structure and where to find it. You will print 10 fields that you will select from the superblock structure: one field per line.

2. Print the content of the **root directory**. The directory entries in the root directory should be visited one by one and for each directory entry, the contained filename should be printed out; one filename per line. You need to read ext2 documentation to learn about how to find the root directory and the structure of a directory entry.
3. Print the content of the **inode** of each regular file in the root directory. For that you need to access the inodes of these files. For an inode, 10 fields that you will select will be printed out (one field per line).

An example invocation of the program can be as follows.

```
diskprint disk1
```

The given disk (Linux file) is generated and formatted as follows. We first create a binary file containing some number of blocks (each 4 KB). For that we type:

```
dd if=/dev/zero of=disk1 bs=4K count=1000
```

This will generate a file called `disk1`. That is a regular Linux file that will act as the disk. It contains 1000 blocks. Each block is 4 KB. Hence the total size of the file (disk) is around 4 MB.

Then we format the disk with **ext2** file system. For that we type:

```
/sbin/mke2fs -t ext2 -r 0 -b 4096 -I 128 disk1 1000 -O
^ext_attr,^resize_inode,^dir_index,^sparse_super
```

Note that the above command is just a single line. The **mke2fs** command creates and initializes an ext2 file system in disk1. The inode size is specified as 128 bytes. We also specify the number of blocks (1000 in this case) and blocksize (4096 bytes). They are the same with the parameter values given to the `dd` command. The version of the ext2 file system is set to be 0 (original version). We also specify some options with the `-O` flag. These options are:

`^ext_attr`: do not use extended attributes feature.
`^resize_inode`: do not use resize feature.
`^dir_index`: do not create an index for a directory.
`^sparse_super`: do not use sparse superblock feature.

You can learn more about the mke2fs command using its manual page (man mke2fs).

Now the disk is ready to be accessed by your program. But it has no files in it. We place some files into the disk by first **mounting** the disk. First, we create a **mount point**, an empty sub-directory, in our project directory (or anywhere you wish). Let us assume the name of the mount point is `x`. We type "`mkdir x`" to create the sub-directory `x`. To mount the disk, we type:

```
sudo mount disk1 x
```

After mounting, we can now change into the directory `x`. Then we will be in the root directory of the mounted file system. If we see a subdirectory `lost+found` there, we can delete it. Then we create some files. For that we can use an editor or use the `touch` command. For example, we create a file called `file1.txt`. If it is empty, its size will be 0. Even its size would be zero, a file will have a directory entry allocated in the root directory and also an inode allocated. The inode of a file is the FBC (file control block) of the file. It keeps information about the file (file attributes).

After creating some files, we are done. We go back to the parent directory of sub-directory `x`. Then we **unmount** the file system. For that we type:

```
sudo umount x
```

Now the disk (file system) is unmounted. We can now run our program to access the disk in raw mode, without mounting (we should not mount the disk while accessing with our program). Our program should open the disk (the Linux file) using the `open()` system call. Then it should access the blocks of it using `lseek()` and `read()` system calls. You can access for example, block 0. You will find the superblock somewhere there (read the ext2 documentation). Then, after reading and parsing superblock information, you will be able to access other file system structures (inodes and directory entries) in the disk, by accessing the respective blocks. After accessing and parsing the directory entry for a file, you will be able to learn its inode number, which will allow you to find and access the inode of the file.

Below is a sample code about how to access a block.

```
#define MAXFILENAME 256
#define BLOCKSIZE 4096

int main(int argc, char *argv[]) {
     char devname[MAXFILENAME];
     strcpy (devname, argv[1]);
     int fd;
     int n;
     off_t offset;
     int blocknum;
     unsigned char buffer[BLOCKSIZE];

     fd = open (devname, O_RDONLY);
     if (fd < 0) {
          printf ("can not open disk file\n");
          exit(1);
     }
     blocknum = 0;
     offset = 0 * blocknum;
```

```
    lseek(fd, offset, SEEK_SET);
    n = read (fd, buffer, BLOCKSIZE);
    if (n == BLOCKSIZE) {
        printf ("block read success\n);
    }
}
```

You will access (read) the disk in blocks (block by block). That means unit of data transfer (reading) from the disk should be one disk block (4 KB). You will read the related blocks to extract the desired information. Then you will print out the desired information to the screen. After that your program will close the disk file and will terminate.

You can also create one or more disk files and put content (files) into them with the same method that is explained above to do more tests. You should use the same method, because this is how we will create our test disks.

In Linux kernel, the implementation of ext2 file system is using the header file **ext2.h**. You can have a look to that file to see the data structures used.

Part B – Experiments (20 pts):

Do some timing experiments. For example, you can measure the time it takes to format a disk (Linux file). You can repeat this for various disk sizes. You can also measure the time your program requires to parse a disk file and print out the desired information. You can repeat some experiments to see the effect of disk caching (caching of disk blocks in memory). Try to explain your results. Write a report and have it in PDF form in the folder that you tar and gzip and upload.

Submission:

Put all your files into a directory named with your Student ID. If the project is done as a group, the IDs of both students will be written, separated by a dash '-'. In a README.txt file, write your name, ID, etc. (if done as a group, all names and IDs will be included). The set of files in the directory will include README.txt, Makefile, and program source files. We should be able to compile your program by just typing make. No binary files (executable files) will be included in your submission. Then tar and gzip the directory, and submit it to Moodle.

For example, a project group with student IDs 21404312 214052104 will create a directory named "21404312-214052104" and will put their files there. Then, they will tar the directory (package the directory) as follows:

tar cvf 21404312-214052104.tar 21404312-214052104

Then they will gzip the tar file as follows:

gzip 21404312-214052104.tar

In this way they will obtain a file called 21404312-214052104.tar.gz. Then they will upload this file into Moodle. For a project done individually, just the ID of the student will be used as file or directory name.

References:

1) Second Extended File System (ext2) of Linux:
https://www.nongnu.org/ext2-doc/ext2.html

2) ext2 information in Wikipedia: https://en.wikipedia.org/wiki/Ext2.

3) ext2 header file in Linux kernel:
https://lxr.linux.no/linux+v6.0.9/fs/ext2/ext2.h

Tips and Clarifications:

- Start early. Work incrementally.