Operating Systems
CSCI 5806

Spring Semester 2025 — CRN 22968

Term Project — Step 3 — Low-Level Ext2 Access

Target completion date: Friday, February 24, 2025

Goals

- Provide functions to provide low-level access to an ext2 file system contained within a VDI file.
- Create a structure or class to contain the data necessary to implement these functions.

Details

In this step, we bridge the gap between "raw" data — disk partitions and byte / sector-level access — and "structured" data — file system-level access.

For convenience, you will still want two functions to open and close the file and a structure to hold necessary information. The functions are:

- **`struct Ext2File *open(char *fn)`**
  Use the partition-level `open()` function to open the given VDI file. Access the partition table and look for the first partition whose `type` is `0x83`, which indicates a Linux filesystem. Use that partition.
- **`void close(struct Ext2File *f)`**
  Close the file whose pointer is given. Deallocate any dynamically created memory regions.

Low-level ext2 access involves three structures — blocks, superblocks and block group descriptors.

Blocks

All space in an ext2 partition is divided into fixed-size blocks. The size of the blocks is determined by the superblock. With one exception, all disk access is performed by reading or writing entire blocks. To that end, you will need two block access functions:

- **`bool fetchBlock(struct Ext2File *f,uint32_t blockNum, void *buf)`**
  Read the given block number from the file system into the buffer. Return `true` if successful, `false` if the read fails.
- **`bool writeBlock(struct Ext2File *f,uint32_t blockNum, void *buf)`**
  Write the buffer to the given block in the file system. Return `true` if successful, `false` if the write fails.

There is one slight quirk in how the disk space is laid out; the main superblock (see next section) is *always* located in block zero. However, in a 1KB file system, that is the second physical 1KB block of space. The `s_first_data_block` field in the superblock indicates how many blocks to skip over to access block zero. The field is always 1 for 1KB file systems and 0 for all other block sizes.

Superblocks

The superblock is the main data structure in a UNIX file system. A good description of the structure can be found at https://www.nongnu.org/ext2-doc/ext2.html.

The main superblock is always located at an offset of 1 024 bytes from the start of the disk partition, regardless of block size. Backup copies of the superblock are stored at various locations throughout the partition (see next section), always at the beginning of a block. You should read the main superblock and store it in the structure you create for this step.

There are two important values that the superblock does not directly contain, but need to be calculated from values in the superblock. The first value is the file system's block size. It is derived from the `s_log_block_size` field: $b = 1024 \cdot 2^{\texttt{s\_log\_block\_size}}$. The second value is the number of block groups, derived from the **`s_blocks_count`** and **`s_blocks_per_group`** fields:
$n = \lceil \texttt{s\_blocks\_count}/\texttt{s\_blocks\_per\_group} \rceil$. Calculate these values and store them in the structure you create for this step.

You should write two functions for superblock access:

- **`bool fetchSuperblock(struct Ext2File *f,uint32_t blockNum, struct Ext2Superblock *sb)`**
  Read the superblock found in the given block number from the file system into the buffer. Return `true` for success, `false` for failure.
- **`bool writeSuperblock(struct Ext2File *f,uint32_t blockNum, struct Ext2Superblock *sb)`**
  Write the superblock to the given block. Return `true` for success, `false` for failure.

For these, use partition-level `lseek()`, `read()` and `write()` to access the main superblock in block 0; use `fetchBlock()` and `writeBlock()` to access other copies of the superblock. Verify that you have read a valid superblock by checking the `s_magic` field, it should be `0xef53`.

Block Groups and Their Descriptors

Blocks are split into block groups; groups act as a crude form of low-level disk access optimization, as the system typically tries to place all of the data blocks for one file in one block group. Each block group contains the following items:

- A copy of the superblock, if the block group number is 0, 1 or a power of 3, 5 or 7. This is always contained in the first block of the group.
- A copy of the *block group descriptor table*, an array of block group descriptors. This array begins in the second block of the group and has as many blocks as necessary to hold the table. The table is stored contiguously (no gaps between entries). Copies are stored in the same groups that have superblock copies.
- A single block containing a bitmap of used and unused blocks in the group.

- A single block containing a bitmap of used and unused inodes in the group.
- A portion of the inode array.
- Data blocks.

A *block group descriptor* is a small structure that contains the block numbers of a group's bitmaps and the first block of the inode table, along with the number of unused blocks and inodes. See the link in the previous section for information about this structure.

You will need to read the main copy of the descriptor table and store it in the structure you create for this step. Since the table size is unknown until calculating the number of block groups, you will have to allocate the table space dynamically.

To access the table, implement these two functions:

- **`bool fetchBGDT(struct Ext2File *f,uint32_t blockNum,struct Ext2BlockGroupDescriptor *bgdt)`**
  Read the block group descriptor table that begins at the given block number. Store the table in the array pointed to by **`bgdt`**. Return `true` for success, `false` for failure.
- **`bool writeBGDT(struct Ext2File *f,uint32_t blockNum,struct Ext2BlockGroupDescriptor *bgdt)`**
  Write the block group descriptor table to the file system starting at the given block number. Return `true` for success, `false` for failure.

Other Functions

You will probably want a function to display a superblock with the fields labeled and in text form and a function to display the values in the block group descriptor table. The following examples illustrate my version of these.

Example 1

This is the output from my step 3 program, on the fixed VDI file with 1KB blocks. It shows the superblock in readable form, then in byte form and then shows the block group descriptor table.

```
Superblock from block 0
Superblock contents:
Number of inodes: 32512
Number of blocks: 130048
Number of reserved blocks: 6502
Number of free blocks: 21487
Number of free inodes: 32175
First data block: 1
Log block size: 0 (1024)
Log fragment size: 0 (1024)
Blocks per group: 8192
Fragments per group: 8192
Inodes per group: 2032
Last mount time: Mon Jan 18 14:28:11 2021
Last write time: Mon Jan 18 14:30:29 2021
Mount count: 2
Max mount count: 65535
Magic number: 0xef53
State: 1
Error processing: 1
Revision level: 1.0
Last system check: Mon Jan 18 14:09:50 2021
Check interval: 0
OS creator: 0
Default reserve UID: 0
Default reserve GID: 0
First inode number: 11
Inode size: 128
Block group number: 0
Feature compatibility bits: 0x00000038
Feature incompatibility bits: 0x00000002
Feature read/only compatibility bits: 0x00000003
UUID: e4fbffca-0066-4bc1-abd9-09c867496c95
Volume name: []
Last mount point: [/media/csis/e4fbffca-0066-4bc1-abd9-09c867496c95]
Algorithm bitmap: 0x00000000
Number of blocks to preallocate: 0
Number of blocks to preallocate for directories: 0
Journal UUID: e4fbffca-0066-4bc1-abd9-09c867496c95
Journal inode number: 0
Journal device number: 0
Journal last orphan inode number: 0
Default hash version: 1
Default mount option bitmap: 0x0000000c
First meta block group: 0
```

```
Raw bytes from block 0 superblock:
Offset: 0x0
   00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f    0...4...8...c...
  +-----------------------------------------------+  +----------------+
00|00 7f 00 00 00 fc 01 00 66 19 00 00 ef 53 00 00|00|        f    S  |
10|af 7d 00 00 01 00 00 00 00 00 00 00 00 00 00 00|10| }              |
20|00 20 00 00 00 20 00 00 f0 07 00 00 4b e1 05 60|20|            K  ‘|
30|d5 e1 05 60 02 00 ff ff 53 ef 01 00 01 00 00 00|30|   ‘    S       |
40|fe dc 05 60 00 00 00 00 00 00 00 00 01 00 00 00|40|   ‘            |
50|00 00 00 00 0b 00 00 00 80 00 00 00 38 00 00 00|50|            8   |
60|02 00 00 00 03 00 00 00 e4 fb ff ca 00 66 4b c1|60|             fK |
70|ab d9 09 c8 67 49 6c 95 00 00 00 00 00 00 00 00|70|    gIl         |
80|00 00 00 00 00 00 00 00 2f 6d 65 64 69 61 2f 63|80|        /media/c|
90|73 69 73 2f 65 34 66 62 66 66 63 61 2d 30 30 36|90|sis/e4fbffca-006|
a0|36 2d 34 62 63 31 2d 61 62 64 39 2d 30 39 63 38|a0|6-4bc1-abd9-09c8|
b0|36 37 34 39 36 63 39 35 00 00 00 00 00 00 00 00|b0|67496c95        |
c0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01|c0|                |
d0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|d0|                |
e0|00 00 00 00 00 00 00 00 00 00 00 00 f4 71 bd c6|e0|             q  |
f0|0e de 42 c6 b3 c2 8f 19 91 28 e8 78 01 00 00 00|f0|  B      ( x    |
  +-----------------------------------------------+  +----------------+

Offset: 0x100
   00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f    0...4...8...c...
  +-----------------------------------------------+  +----------------+
00|0c 00 00 00 00 00 00 00 fe dc 05 60 00 00 00 00|00|           ‘    |
10|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|10|                |
20|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|20|                |
30|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|30|                |
40|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|40|                |
50|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|50|                |
60|01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|60|                |
70|00 00 00 00 00 00 00 00 91 92 01 00 00 00 00 00|70|                |
80|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|80|                |
90|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|90|                |
a0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|a0|                |
b0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|b0|                |
c0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|c0|                |
d0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|d0|                |
e0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|e0|                |
f0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|f0|                |
  +-----------------------------------------------+  +----------------+

Offset: 0x200
   00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f    0...4...8...c...
  +-----------------------------------------------+  +----------------+
00|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|00|                |
10|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|10|                |
20|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|20|                |
30|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|30|                |
40|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|40|                |
50|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|50|                |
60|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|60|                |
70|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|70|                |
80|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|80|                |
90|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|90|                |
a0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|a0|                |
b0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|b0|                |
c0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|c0|                |
d0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|d0|                |
e0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|e0|                |
f0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|f0|                |
  +-----------------------------------------------+  +----------------+

Offset: 0x300
   00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f    0...4...8...c...
  +-----------------------------------------------+  +----------------+
00|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|00|                |
10|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|10|                |
20|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|20|                |
30|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|30|                |
40|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|40|                |
50|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|50|                |
60|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|60|                |
70|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|70|                |
80|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|80|                |
90|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|90|                |
a0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|a0|                |
b0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|b0|                |
c0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|c0|                |
d0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|d0|                |
e0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|e0|                |
f0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|f0|                |
  +-----------------------------------------------+  +----------------+


Block group descriptor table:
Block Block Inode Inode Free Free Used
Number Bitmap Bitmap Table Blocks Inodes Dirs
------ ------ ------ ----- ------ ------ ----
     0    259    260    261   1520   2019    2
     1   8451   8452   8453   1534   2032    0
     2  16385  16386  16387   1997   1904   34
     3  24835  24836  24837   2871   1836   60
     4  32769  32770  32771    768   2032    0
     5  41219  41220  41221   1534   2032    0
     6  49153  49154  49155   1791   2032    0
     7  57603  57604  57605   1279   2032    0
     8  65537  65538  65539      0   2032    0
     9  73987  73988  73989   1534   2032    0
    10  81921  81922  81923    256   2032    0
    11  90113  90114  90115   1024   2032    0
    12  98305  98306  98307    772   2032    0
    13 106497 106498 106499   1792   2032    0
    14 114689 114690 114691   1792   2032    0
    15 122881 122882 122883   1023   2032    0
```

Example 2

Same output from the dynamic-allocation VDI file with 2KB block size.

```
Superblock from block 0
Superblock contents:
Number of inodes: 32512
Number of blocks: 65024
Number of reserved blocks: 3251
Number of free blocks: 11263
Number of free inodes: 32175
First data block: 0
Log block size: 1 (2048)
Log fragment size: 1 (2048)
Blocks per group: 16384
Fragments per group: 16384
Inodes per group: 8128
Last mount time: Mon Jan 18 14:27:58 2021
Last write time: Mon Jan 18 14:30:29 2021
Mount count: 2
Max mount count: 65535
Magic number: 0xef53
State: 1
Error processing: 1
Revision level: 1.0
Last system check: Mon Jan 18 14:12:51 2021
Check interval: 0
OS creator: 0
Default reserve UID: 0
Default reserve GID: 0
First inode number: 11
Inode size: 128
Block group number: 0
Feature compatibility bits: 0x00000038
Feature incompatibility bits: 0x00000002
Feature read/only compatibility bits: 0x00000003
UUID: 892a4485-0985-4cce-9b9a-29f238151c3a
Volume name: []
Last mount point: [/media/csis/892a4485-0985-4cce-9b9a-29f238151c3a]
Algorithm bitmap: 0x00000000
Number of blocks to preallocate: 0
Number of blocks to preallocate for directories: 0
Journal UUID: 892a4485-0985-4cce-9b9a-29f238151c3a
Journal inode number: 0
Journal device number: 0
Journal last orphan inode number: 0
Default hash version: 1
Default mount option bitmap: 0x0000000c
First meta block group: 0


Raw bytes from block 0 superblock:
Offset: 0x0
```

```
   00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f    0...4...8...c...
  +-----------------------------------------------+  +----------------+
00|00 7f 00 00 00 fe 00 00 b3 0c 00 00 ff 2b 00 00|00|             +  |
10|af 7d 00 00 00 00 00 00 01 00 00 00 01 00 00 00|10| }              |
20|00 40 00 00 00 40 00 00 c0 1f 00 00 3e e1 05 60|20| @   @      >  ‘|
30|d5 e1 05 60 02 00 ff ff 53 ef 01 00 01 00 00 00|30|   ‘    S       |
40|b3 dd 05 60 00 00 00 00 00 00 00 00 01 00 00 00|40|   ‘            |
50|00 00 00 00 0b 00 00 00 80 00 00 00 38 00 00 00|50|            8   |
60|02 00 00 00 03 00 00 00 89 2a 44 85 09 85 4c ce|60|         *D   L |
70|9b 9a 29 f2 38 15 1c 3a 00 00 00 00 00 00 00 00|70|  ) 8  :        |
80|00 00 00 00 00 00 00 00 2f 6d 65 64 69 61 2f 63|80|        /media/c|
90|73 69 73 2f 38 39 32 61 34 34 38 35 2d 30 39 38|90|sis/892a4485-098|
a0|35 2d 34 63 63 65 2d 39 62 39 61 2d 32 39 66 32|a0|5-4cce-9b9a-29f2|
b0|33 38 31 35 31 63 33 61 00 00 00 00 00 00 00 00|b0|38151c3a        |
c0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 3f 00|c0|              ? |
d0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|d0|                |
e0|00 00 00 00 00 00 00 00 00 00 00 00 22 94 4b f3|e0|            " K |
f0|cc d3 41 69 9b 2d 95 81 15 e2 6c 6d 01 00 00 00|f0|  Ai -    lm    |
  +-----------------------------------------------+  +----------------+

Offset: 0x100
   00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f    0...4...8...c...
  +-----------------------------------------------+  +----------------+
00|0c 00 00 00 00 00 00 00 b3 dd 05 60 00 00 00 00|00|           ‘    |
10|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|10|                |
20|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|20|                |
30|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|30|                |
40|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|40|                |
50|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|50|                |
60|01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|60|                |
70|00 00 00 00 00 00 00 00 22 93 01 00 00 00 00 00|70|        "       |
80|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|80|                |
90|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|90|                |
a0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|a0|                |
b0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|b0|                |
c0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|c0|                |
d0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|d0|                |
e0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|e0|                |
f0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|f0|                |
  +-----------------------------------------------+  +----------------+

Offset: 0x200
   00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f    0...4...8...c...
  +-----------------------------------------------+  +----------------+
00|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|00|                |
10|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|10|                |
20|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|20|                |
30|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|30|                |
40|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|40|                |
50|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|50|                |
60|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|60|                |
70|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|70|                |
80|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|80|                |
90|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|90|                |
a0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|a0|                |
b0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|b0|                |
c0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|c0|                |
d0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|d0|                |
e0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|e0|                |
f0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|f0|                |
  +-----------------------------------------------+  +----------------+

Offset: 0x300
   00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f    0...4...8...c...
  +-----------------------------------------------+  +----------------+
00|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|00|                |
10|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|10|                |
20|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|20|                |
30|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|30|                |
40|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|40|                |
50|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|50|                |
60|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|60|                |
70|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|70|                |
80|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|80|                |
90|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|90|                |
a0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|a0|                |
b0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|b0|                |
c0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|c0|                |
d0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|d0|                |
e0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|e0|                |
f0|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|f0|                |
  +-----------------------------------------------+  +----------------+


Block group descriptor table:
Block Block Inode Inode Free Free Used
Number Bitmap Bitmap Table Blocks Inodes Dirs
------ ------ ------ ----- ------ ------ ----
     0     65     66     67   4025   8115    2
     1  16449  16450  16451   2691   7804   94
     2  32768  32769  32770   1538   8128    0
     3  49217  49218  49219   3009   8128    0
```

Additional examples are in the step 3 log file in the repository. Those examples also display all copies of the superblock and block group descriptor tables for all six test files.