Operating Systems
CSCI 5806

Spring Semester 2020 — CRN 21176 / 26762

Term Project — Step 1 — VDI File Access

Target completion date: Friday, January 31, 2020

Goals

- Provide the five basic file I/O functions to access disk space inside a VDI file.
- Create a structure or class to contain the data necessary to implement the five functions.

Details

You'll want a single entity — a structure or a class, either works — to represent a VDI file within your project. The intent is to collect the various data your project is going to need into one place for ease of use; if you're using C++ (you *are* using C or C++, right?) then a class can also contain the basic I/O functions as methods.

What do you need to keep track of in a VDI file? To implement the five functions you'll probably want at least these four items to be contained in your structure:

- The file descriptor for the VDI file. This is an `int`.
- A VDI header structure. Discussion of the structure can be found at **https://forums.virtualbox.org/viewtopic.php?t=8046**; C structures for the header can be found in the VirtualBox source code.
- The VDI translation map. This is optional; include it if you are going to enable dynamic VDI files (they're easy!), ignore it otherwise. This is just an array of integers, although you don't know the size in advance. The size is given in the VDI header, so you'll need to allocate this dynamically.
- A cursor. This is just an integer (of type `size_t`) that holds the location of the next byte to be read or written.

Once the structure / class is created, you'll need to implement a VDI version of each of the five basic I/O functions.

- **`struct VDIFile *vdiOpen(char *fn)`**
  Open the file whose name is given. The filename can be either a relative or absolute path. The function returns a pointer to a dynamically created VDI file structure (see above), or a null pointer if there was an error. The function should load the header and the translation map, set the cursor to `0` and set the file descriptor to whatever was returned from the `open()` system call.

  If you're using a class, then this can return a boolean to indicate success or failure of the open.

- **`void vdiClose(struct VDIFile *f)`**
  Close the file whose pointer is given. Deallocate any dynamically created memory regions.

- **`ssize_t vdiRead(struct VDIFile *f,void *buf,size_t count)`**
  Reads the given number of bytes from the given VDI file's disk space, placing the bytes in the given buffer. The location of the first byte read is given by adding the cursor to the start of the VDI file's data space. The starting location is given in the VDI header. Advance the cursor by the number of bytes read.
- **`ssize_t vdiWrite(struct VDIFile *f,void *buf,size_t count)`**
  Writes the given number of bytes to the given file, starting at the cursor (plus the data start location). Bytes are written sequentially and the cursor is advanced to the end of the written block. Bytes to be written are located in the given buffer.
- **`off_t vdiSeek(VDIFile *f,off_t offset,int anchor)`**
  Move the cursor of the given file to the given location, based on the offset and anchor values. If the resulting location is negative or larger than the disk size, do not change the value of the cursor.

If you are using a class, then the `VDIFile *` parameter is omitted.

You should also write a function that takes a pointer to a VDIFile as a parameter and displays its header fields in an easy-to-read manner. See example 1 below for a sample; your exact format may vary.

▹*Suggestions*

- `vdiSeek()` should only set the cursor in the `VDIFile` structure; it should *not* call `lseek()`.
- Reading and writing should be done one page at a time. While there are bytes left to be read or written, do the following tasks:
  1. Determine where within the current page reading or writing should begin.
  2. Determine how many bytes should be read or written in the current page.
  3. Determine the physical location of the page. This may involve page translation and/or page allocation. Questions: What if the page is not allocated? What if the page is marked as "all zeroes"?
  4. Use `lseek()` to go to the proper location within the physical page.
  5. Use `read()` or `write()` to read or write *only* the bytes within the current page.
  6. Advance the cursor by the number of bytes read or written; subtract the number of bytes read or written from the number of bytes remaining.
- If you are planning to write into the filesystem, consider using the `mmap()` and `munmap()` functions to load the VDI file header and translation map. These act like a write-through cache; they read the areas from the file into memory set up by the OS, and writing to them automagically writes back to the file.

▹*Example 1*

This is the output from my function **dumpVDIHeader(struct VDIFile *)**. The image name has a newline in it, so the closing square bracket is on its own line. The image comment is a 256-byte array, so I displayed it using `displayBufferPage()`. I wrote a function that converts a UUID from a 16-byte character array into the standard UUID hex format.

```
      Image name: [<<< Oracle VM VirtualBox Disk Image >>>
                  ]
       Signature: 0xbeda107f
         Version: 1.01
     Header size: 0x00000190  400
      Image type: 0x00000002
           Flags: 0x00000000
     Virtual CHS: 0-0-0
     Sector size: 0x00000200  512
     Logical CHS: 260-16-63
     Sector size: 0x00000200  512
      Map offset: 0x00100000  1048576
    Frame offset: 0x00200000  2097152
      Frame size: 0x00100000  1048576
Extra frame size: 0x00000000  0
    Total frames: 0x00000080  128
Frames allocated: 0x00000080  128
       Disk size: 0x0000000008000000  134217728
            UUID: e6d80707-6b24-46c1-04a7-65f685ebfafa
  Last snap UUID: 39261ab0-6367-49c1-0fbe-f010ff6ef1f1
       Link UUID: 00000000-0000-0000-0000-000000000000
     Parent UUID: 00000000-0000-0000-0000-000000000000
   Image comment:
Offset: 0x54
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|                |
  +-----------------------------------------------+  +----------------+

      Unused: 0
```

▹*Example 2*

This is the VDI header from the sample VDI file. It's the same output as Step 0 Example 1.
`displayBuffer(f->header,400,0);`

```
Offset: 0x0
   00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f    0...4...8...c...
  +-----------------------------------------------+  +----------------+
00|3c 3c 3c 20 4f 72 61 63 6c 65 20 56 4d 20 56 69|00|<<< Oracle VM Vi|
10|72 74 75 61 6c 42 6f 78 20 44 69 73 6b 20 49 6d|10|rtualBox Disk Im|
20|61 67 65 20 3e 3e 3e 0a 00 00 00 00 00 00 00 00|20|age >>>         |
30|00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00|30|                |
40|7f 10 da be 01 00 01 00 90 01 00 00 02 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: 0x100
   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 10 00 00 00 20 00 00 00 00 00|50|                |
60|00 00 00 00 00 00 00 00 00 02 00 00 00 00 00 00|60|                |
70|00 00 00 08 00 00 00 00 00 00 10 00 00 00 00 00|70|                |
80|80 00 00 00 80 00 00 00 07 07 d8 e6 24 6b c1 46|80|            $k F|
90|                                               |90|                |
a0|                                               |a0|                |
b0|                                               |b0|                |
c0|                                               |c0|                |
d0|                                               |d0|                |
e0|                                               |e0|                |
f0|                                               |f0|                |
  +-----------------------------------------------+  +----------------+
```

▹*Example 3*

The third example shows the partition table from the sample VDI file. This is the same output as Step 0 Example 3. The function calls used here were:

```
char buffer[64];
struct VDIFile *f = vdiOpen(filename);
vdiSeek(f,446,SEEK_SET);
vdiRead(f,buffer,64);
displayBufferPage(pageBuf,64,190,256);
```

Output:

```
Offset: 0x100
   00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f    0...4...8...c...
  +-----------------------------------------------+  +----------------+
00|                                               |00|                |
10|                                               |10|                |
20|                                               |20|                |
30|                                               |30|                |
40|                                               |40|                |
50|                                               |50|                |
60|                                               |60|                |
70|                                               |70|                |
80|                                               |80|                |
90|                                               |90|                |
a0|                                               |a0|                |
b0|                                          00 20|b0|                |
c0|21 00 83 51 01 10 00 08 00 00 00 f8 03 00 00 00|c0|!  Q            |
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      |f0|                |
  +-----------------------------------------------+  +----------------+
```

▹*Example 4*

This is the VDI header from the sample dynamic VDI file. Much of the information is the same, but there are some differences.

```
      Image name: [<<< Oracle VM VirtualBox Disk Image >>>
                  ]
       Signature: 0xbeda107f
         Version: 1.01
     Header size: 0x00000190  400
      Image type: 0x00000001
           Flags: 0x00000000
     Virtual CHS: 0-0-0
     Sector size: 0x00000200  512
     Logical CHS: 0-0-0
     Sector size: 0x00000200  512
      Map offset: 0x00100000  1048576
    Frame offset: 0x00200000  2097152
      Frame size: 0x00100000  1048576
Extra frame size: 0x00000000  0
    Total frames: 0x00000080  128
Frames allocated: 0x00000018  24
       Disk size: 0x0000000008000000  134217728
            UUID: 554a3597-a01c-4d29-7e99-76bb25146767
  Last snap UUID: a5c4e0b0-d3db-4d52-7092-0d00dfe5c1c1
       Link UUID: 00000000-0000-0000-0000-000000000000
     Parent UUID: 00000000-0000-0000-0000-000000000000
   Image comment:
Offset: 0x54
   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|                |
  +-----------------------------------------------+  +----------------+

    Unused: 0
```

▹*Example 5*

The last example shows the first 64 entries from the dynamic VDI sample file's translation map. Note that many of the entries contain `0xffffffff` indicating the page is not allocated.

```
Offset: 0x100000
   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 01 00 00 00 12 00 00 00 13 00 00 00|00|                |
10|ff ff ff ff 02 00 00 00 ff ff ff ff ff ff ff ff|10|                |
20|ff ff ff ff 03 00 00 00 ff ff ff ff 14 00 00 00|20|                |
30|15 00 00 00 16 00 00 00 17 00 00 00 ff ff ff ff|30|                |
40|ff ff ff ff 09 00 00 00 ff ff ff ff ff ff ff ff|40|                |
50|ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff|50|                |
60|ff ff ff ff 04 00 00 00 ff ff ff ff ff ff ff ff|60|                |
70|ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff|70|                |
80|ff ff ff ff 0a 00 00 00 ff ff ff ff ff ff ff ff|80|                |
90|ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff|90|                |
a0|ff ff ff ff 05 00 00 00 ff ff ff ff ff ff ff ff|a0|                |
b0|ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff|b0|                |
c0|ff ff ff ff 06 00 00 00 ff ff ff ff ff ff ff ff|c0|                |
d0|ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff|d0|                |
e0|ff ff ff ff 07 00 00 00 ff ff ff ff ff ff ff ff|e0|                |
f0|ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff|f0|                |
  +-----------------------------------------------+  +----------------+
```