Description

For assignment 4, we're looking for you to be able to write software that *works with* a file system.

Specifically, you're going to be writing code that can read an exFAT file system, and extract files from that file system.

Assignment 4 is due Friday, April 22nd, 2021 at 11:59pm (Winnipeg time).

General submission requirements

Submissions tha d (i.e., you will
receive a score o

- All solutions epted.
- All solutions
  - Forget h a `Makefile`?
    Thankfu d find some
    very eas
  - Your `Mak`

    ```
    make cl
    ```

    is run in your directory, all the temporary files, **including** the executable should be removed (e.g., `rm -f my_prog`).
- All solutions must be compiled by issuing the command `make` in the directory containing the `Makefile`. No other build commands are acceptable, even if documented in the `README.md`.
- All solutions must include a Markdown-formatted `README.md` file that *minimally* describes how to run your submission.
  - Forget how to make a `README.md`? Never knew how to make `README.md`? Thankfully, you can go to https://readme.so/ and build a nice, Markdown-formatted `README.md` with a nice preview and some handy buttons to help you build it.
- All solutions must *run* to successful completion. Premature termination for

any reason (no obvious output, `Segmentation Fault`, `Bus Error`, etc.) is considered to be a solution implementation that does not run.

Code that runs for an unreasonably long time (e.g., > 30 seconds) without terminating is also considered to be a solution implementation that does not run.

- All solutions must *compile*. Code that does not compile will not be evaluated.
- Programs must produce no errors when compiled with **all** of the flags

  ```
  -Wall -Wpedantic -Wextra -Werror
  ```

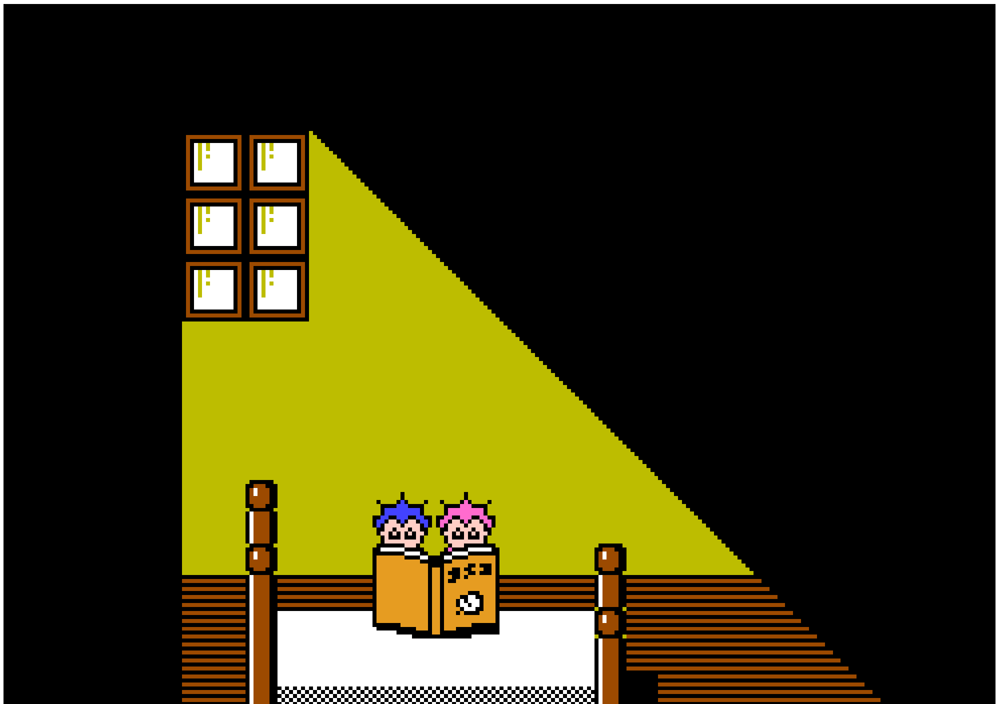
  Note that `-Werror` *prevents* your code from being **compiled** when warnings are present.

  If all of these flags are not in your `Makefile`, your submission will be treated as though it

- Your code m umanitoba.ca.
- No late subr e is enforced electronicall

Reminder: All ated similarity testing tool, alor oblems.

Implen
reader



Get pulled into reading!

In this question, you will write a program that reads an exFAT-formatted volume. For this assignment you are provided with a couple of volume images, but you are encouraged to read a USB flash, SDHC, or SDXC drive as a raw device (e.g., `/dev/sda3`), provided that you have physical access to a Linux machine.

exFAT documentation

The general stru ssed in class, so you should take ven't already.

You should also This document comprehensively our universal source of truth f

Volumes

Two volumes ha am. The two volumes are dist ME.md describing the t e in the volumes, and th

The graders may not use these volumes to evaluate your work.

Required commands

You should implement 3 commands that are all part of the same program: `info`, `list`, and `get`.

`info`

The `info` command will print information about the volume. The command, assuming your program is named `exfat`, would be

```
./exfat imagename info
```

Print out the following:

- Volume label.
- Volume serial number.
- Free space on the volume in KB.
- The cluster size, both in sectors and in bytes OR KB.

Note: 1KB ↔ 1024 bytes.

Volume label

The volume label is encoded as a directory entry in the root directory. You can find information about the volume label in section 7.3 Volume Label Directory Entry.

The Volume label is a Unicode-formatted string, please see Unicode and ASCII at the bottom for some additional information.

Volume serial

The volume seri nformation about it both in ions, and in section 3.1.11 Vo

You should prin .

Free space

exFAT uses allo unused clusters. You ca space in the volume (the nu clusters).

The allocation bitmap is encoded as a directory entry in the root directory. You can find information about the allocation bitmap in section 7.1 Allocation Bitmap Directory Entry.

You should use code that you wrote for lab 4 to determine this value.

Cluster size

The cluster size in sectors and the sector size in bytes can be found in the boot sector. You can find more information about both of these fields in 3.1 Main and Backup Boot Sector Sub-regions.

Remember: These fields are called ___Shift because you can use the left shift operator (<<) to quickly compute powers of two:

```
uint8_t x = 0x1 << 3; // 2^3 -> 8
```

`list`

The `list` command will recursively print all files and directories in the volume. The output should look *roughly* like the output from `tree` (try running `tree $YOUR_A2` on `aviary.cs.umanitoba.ca`).

You don't have to print out fancy Extended ASCII characters, but instead can use the – symbol to denote depth, where the number of – characters indicates the depth of the file/folder. Using the Extended ASCII set would be a nice touch, though.

The command, assuming your program is named `exfat`, would be

```
./exfat imagename list
```

Output should l[illegible]

```
egret.cs.umani[illegible]
File: file.txt[illegible]
Directory: fol[illegible]
- Directory: f[illegible]
-- Directory: [illegible]
--- File: thef[illegible]
File: file2.tx[illegible]
```

In this example, [illegible]: `file.txt` and `file2.txt`, and [illegible]ins a reference to one folder: `fo`[illegible] to one folder: `folder3`. The `fo`[illegible]`efile.txt`.

Order of files is [illegible]ore directories, for example). The recommended strategy here is to just print out files and folders in the same order that they are in in terms of the sequence of `DirectoryEntry` in the folders. Likewise, the sequence of directories that you follow is also not important. You're going to have to do this recursively, but it's up to you if you want to do a breadth-first or a depth-first implementation.

`get`

The `get` command will extract a complete file from the volume. The command, assuming your program is named `exfat`, would be

```
./exfat imagename get path/to/file.txt
```

The path you pass to your program is assumed to be an absolute path (it starts at the root directory), even if it doesn't have a leading `/`.

The path you pass to your program should be written out as a file to the same directory as your program with the same name as the file in the exFAT image. If the example above is executed, you should write the contents of the file at `path/to/file.txt` from the disk image to a file named `file.txt` in the same directory as the `./exfat` binary executable.

Implementation notes

The exFAT file system specification is fairly comprehensive and generally very good, but some parts of the documentation are not straightforward. Additionally, exFAT supports Unicode characters, and while that's an excellent property of a file system, working with Unicode characters in C isn't a great experience.

Below is some additional information that you can use to help guide you through working with ex

Sectors ↔ cl

The first chunk or Sub-regions) are organized in luster.

The FATs are als multiple sectors.

The data heap it not be the same size as a sector. right regions! You should cons ly converts from cluster nu

Root directory and cluster number offsets

An easily overlooked statement in 5.1 Cluster Heap Sub-region is

> Importantly, the first cluster of the Cluster Heap has index two, which directly corresponds to the index of `FatEntry[2]`.

That means that cluster indexing is effectively a 2-based array (what kind of a maniac came up with this?). This means that if, for example, the value in the boot sector for the root directory is 5, then the root directory is actually stored starting at cluster number 3 (5 – 2 = 3).

Directory entries

Files, directories, and some FS metadata are encoded as directory entries in the data heap region. The description of how to interpret types of directory entries is not great (what the heck does "Benign primary" even mean???).

While you actually can figure out the specific `EntryType` values by building up sequences of bits for each type of `DirectoryEntry`, it's easier to just look up these values somewhere else. An admittedly spammy looking page selling some software called "Active UNDELETE" has a nice list of the exFAT directory entry types that includes actual numeric values that you can use for comparing against the `EntryType` field in the directory entry.

The directory entries that correspond to one single file will exist as **three** separate, but sequential directory entries:

1. A file directory entry.
2. *Exactly 1* stream extension directory entry.
3. *At least 1* file name directory entry.

You need to reac … name of the file
that you're looki…

Note: Individual … e cluster, but
an entry set for … ords, if the
cluster chain for … uster 3 and
cluster 10), the f … n cluster 3, but
the file name di…

Unicode and…

Many fields in th … ngs. That
means that char … ided that your
Unicode string c … / لا أتحدث اللغة
العربية, 沒有中文, ਕੋਈ ਪੰਜਾਬੀ ਨਹੀਂ, and no emoji 🦖), you can trivially convert a Unicode string into a regular C string by stripping off the top 8 bits:

```
/**
 * Convert a Unicode-formatted string containing only ASCII characters
 * into a regular ASCII-formatted string (16 bit chars to 8 bit
 * chars).
 *
 * NOTE: this function does a heap allocation for the string it
 *       returns, caller is responsible for `free`-ing the allocation
 *       when necessary.
 *
 * uint16_t *unicode_string: the Unicode-formatted string to be
 *                           converted.
 * uint8_t   length: the length of the Unicode-formatted string (in
 *                    characters).
 *
 * returns: a heap allocated ASCII-formatted string.
 */
static char *unicode2ascii( uint16_t *unicode_string, uint8_t length )
{
    assert( unicode_string != NULL );
    assert( length > 0 );

    char *ascii_string = NULL;

    if ( unicode_string != NULL && length > 0 )
    {
        // +1 for a NULL terminator
        ascii_string = calloc( sizeof(char), length + 1);

        if ( a
        {
            // [illegible] in the
            //
            fo
            {
                [illegible] i];
            }
            // [illegible] string.
            as
        }
    }

    return ascii_string;
}
```

You are permitted to use this code fragment in your submission.

Boot sector

We provided this to you for lab 4, but here's the complete boot sector `struct` for your convenience:

```
#pragma pack(1)
#pragma pack(push)
typedef struct MAIN_BOOT_SECTOR
{
    uint8_t jump_boot[3];
    char fs_name[8];
    uint8_t must_be_zero[53];
    uint64_t partition_offset;
    uint64_t volume_length;
    uint32_t fat_offset;
    uint32_t fat_length;
    uint32_t cluster_heap_offset;
    uint32_t cluster_count;
    uint32_t first_cluster_of_root_directory;
    uint32_t volume_serial_number;
    uint16_t fs_revision;
    uint16_t fs_flags;
    uint8_t bytes_per_sector_shift;
    uint8_t sectors_per_cluster_shift;
    uint8_t number_of_fats;
    uint8_t drive_select;
    uint8_t pe
    uint8_t re
    uint8_t bo
    uint16_t b
} main_boot_se
#pragma pack(p
```

As with lab 4, yo on.

Binary files

Remember that inary files. With that in mind, yo you used for parsing an ELF- hose tools include

- The uintX_t family of integers from stdint.h.
- The hexdump command-line tool.

In assignment 1 we were working with binary files where the layout of the structures in the file conditionally changed based on the type of file that it was (e.g., 32-bit vs 64-bit files). The exFAT file system **does not** have any structures that are sized conditionally. That means that you can actually use (and are recommended to use) the “read the entire struct” strategy. Remember: this means that you need to use preprocessor directives (#pragma pack) to make sure that the compiler doesn’t change the layout of your struct.

Evaluation

Implementation

5 points are awarded for code quality and design:

Level	Description
0	The code is very poor quality (e.g., no comments at all, no functions, poor naming conventions for variables, etc).
1–3	The code is *low* quality, while some coding standards are applied, their uses is inconsistent (e.g., inconsistent use of comments, *some* functions but functions might do too much, code is repeated that *should* be in a function, etc). This is the maximum level you can earn if the implementation of your program is substantially incomplete.
4–5	The co onsistently throug

Each command s, for a total of 15 points (5 × 3)

Level	
0	No att s not run or compil
1–2	The su f the requir ted.
3–4	The su minor parts are stil
5	The submitted code is complete, all major functionality works as expected.

Submitting your assignment

Submissions will be made using the `handin` command available on all CS UNIX systems.

You should be submitting *at least* the following files:

- A `Makefile`.
- A `README.md` that includes a summary of how to compile and run your program (compiling *should* be "run `make`", but you should tell the grader how

to run your program, e.g., what arguments to pass to the program on the command line).

- Your solution for question 1 (probably just 1 `.c` file).

Please **do not** include any disk images in your submission.

If your files are in a folder named `my_a4`, you would run the command:

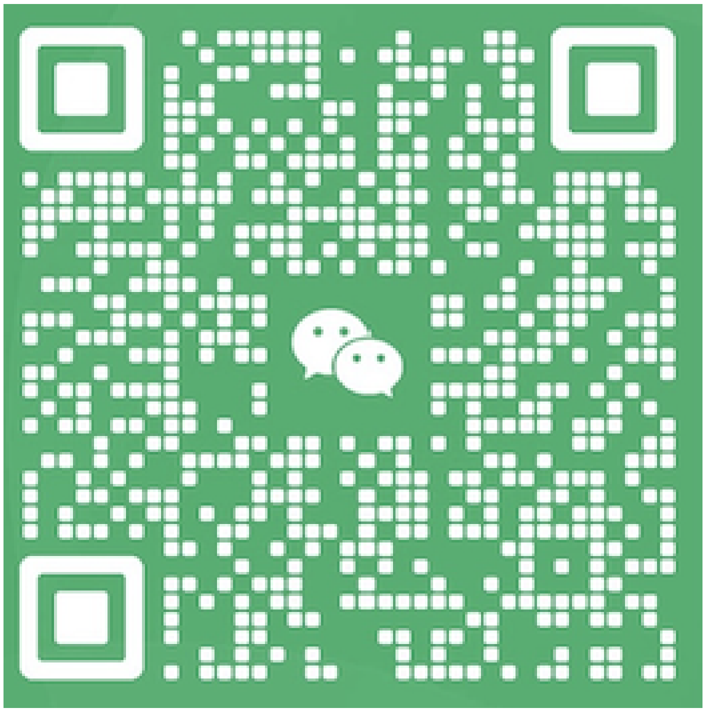
```
handin 3430 a4 my_a4
```

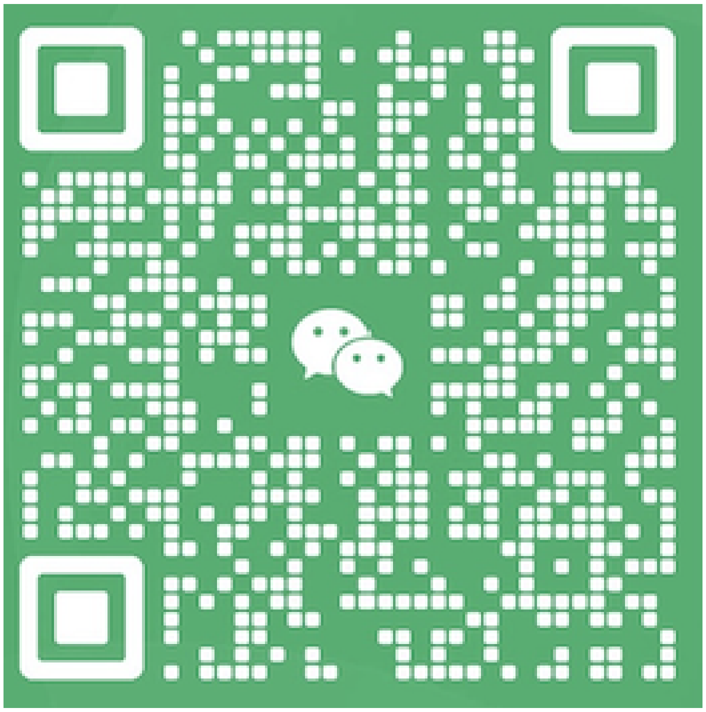
If you need more information, you can see `man handin`.

General Advice

Here's some gen :

1. The debugge olve this problem unt or `gdb`). The first questio or help is: "Have you tr nswer "no", I'll ask you (pol
2. Source contr osts an instance of GitLab th welcome to use private r





Reflect in ePortfolio

Open with docReader

You have viewed this to

Last Visited Apr 14, 2022 11:3