CMPSC 311 Assignment 4: Image Library

Out: April 4, 2014 Due: April 18, 2014

Purpose

This assignment is designed to give you further practice in understanding and creating abstractions, as well as implementing some simple binary file I/O. Given the specification for an image file format, you will create a small library which allows the user to read images from a file, draw them on a MultiTron screen, and save them to a new file.

Description

In this assignment you will write a small library to work with binary image files. This library will interface with the MultiTron driver which was implemented in the previous assignment. You will provide the functionality for reading an image from a file, writing an image to a file, and drawing an image with bit-masked transparency to the MultiTron.

File format

The "`3ii`" format is a fictitious image file format which stores three main pieces of information:

- A comment describing the image
- The image pixel data
- (Optional) A bitwise mask describing what parts of the pixel data to draw

The `3ii` file header begins at offset 0 in the file, and its structure is as follows (offset and size are in bytes):

Offset	Size	Description
0	4	Magic number `0x43530311` to identify file format
4	2	Width of the image in pixels
6	2	Height of the image in pixels
8	4	File offset where image data begins
12	4	File offset where comment string begins
16	2	Length of comment string
18	2	Feature bits (we will only implement `FEATURE_MASK`)

All multi-byte values in a `3ii` file are in *big-endian* byte order. You will need to read the `man` pages for `ntohs` and `ntohl` to learn to convert values appropriately, where "network byte order" means big-endian. (Attempting to do byte-order conversion by hand is not portable to different systems!)

Image mask

An image mask is a set of bits which describe whether a given part of an image is transparent or opaque. In our file format, each bit in the mask corresponds to the same bit in the data. When drawing the image, if the mask bit is 1, then the result should be taken from the data, and if the mask bit is 0, then the result should be taken from what is already on screen.

For example, if the byte on the screen is `01010101`, the data byte is `00000000`, and the mask byte is `00001111`, then the result which is drawn to the screen will be `01010000`. The mask tells us to take the first four bits from the screen and the last four bits from the data. This calculation should be done with the bitwise operators `&` and `|` in C.

In a `3ii` file, the `FEATURE_MASK` bit will be set in the "feature bits" field in the file header if the image file contains a mask. The data for the mask immediately follows the image data and is the same length. If the `FEATURE_MASK` bit is not set, then the file does not contain mask data, and the image is assumed to be opaque (i.e., every bit in the mask should be set to 1).

Verifier

The provided verifier program will run a number of unit tests on the implementation of the image library, using the provided images in the `images` directory, and output information about which tests have failed. The verifier also tests for correct handling of error conditions, so you may also see error messages in the output.

If you need to inspect the data in a particular image file to aid in debugging, you can do this using the `hexdump` utility, e.g.:

```
hexdump -C images/image2.3ii
```

If you cannot determine the source of the problem from the output of the verifier and inspecting any relevant images, you will need to use a debugger to locate the issue.

Procedure

1. Log into your virtual machine and open a terminal emulator.
2. Download the starter code and extract the tarball as you did in previous assignments:

   `http://www.cse.psu.edu/~djp284/cmpsc311-s14/docs/assign4.tar.gz`

   This will create a directory `assign4` containing these files:

   - `image.c`: Library implementation. All of your code goes here. *This is the only file you need to edit.*
   - `image.h`: Header file containing prototypes for the functions you will be implementing and a struct to store image state. You should not change this file.
   - `libmtdriver.a`: MultiTron and driver implementation, provided as a static library. (This is the 64-bit version; on a 32-bit machine, edit the Makefile to include `libmtdriver32.a` instead.) This library provides the `mtron_*` functions.
   - `driver.h`: Header file for `libmtdriver`.
   - `verify.c`: Program which runs unit tests on the library implementation.
   - `Makefile`: Makefile for the entire project. You can edit this if needed.

- `images/`: Directory of sample `3ii`-format images.

3. Comment all of your code as you write it to explain what the code is doing.

4. Implement the following four functions in `image.c`:

   - `image_init(img, fname)`: Opens the image file specified by the given filename and loads data from the file into the given image struct. This should allocate and initialize any memory needed by the struct based on the image dimensions found in the file and close the file when it is finished. Since `3ii` is a binary file format, you should use the low-level file descriptor I/O functions.
   - `image_destroy(img)`: Releases any resources that were allocated in `image_init`.
   - `image_save(img, fname)`: Creates a new image file with the given filename and writes the data from the given image struct into it using the `3ii` format. If the file already exists, it should be overwritten and truncated. The file should be closed when the function finishes. As with `image_init`, you should use low-level I/O for this function.
   - `image_draw(img, mtron, x, y)`: Draws the image at the given coordinates on the MultiTron. It is assumed that the MultiTron has already been powered up and initialized. If there is an image mask, it should be handled as described in the section on masks.

   Any function with an `int` return type should return 0 for success and nonzero if there is an error of any sort, including out-of-bounds values or malformed input files which do not adhere to the `3ii` format.

Submission

1. Run `make clean` to remove any build products (binaries, object files, etc.) from the directory.

2. Create a tarball containing the `assign4` directory, complete with the source code you have written. The tarball should be named `LASTNAME-psuid-assign4.tar.gz`, where `LASTNAME` is your last name in all capital letters and `psuid` is your PSU email ID (the one you would use to log into ANGEL). For example, if the instructor were submitting this assignment, he would run:

   ```
   $ tar -cvzf POHLY-djp284-assign4.tar.gz assign4/
   ```

3. Verify that the tarball contains all of your source code files by viewing the file list:

   ```
   $ tar -tvzf LASTNAME-psuid-assign4.tar.gz
   ```

4. Before sending the tarball, test it in a clean temporary directory using the following commands:

   ```
   $ mkdir /tmp/assign4test
   $ cp LASTNAME-psuid-assign4.tar.gz /tmp/assign4test
   $ cd /tmp/assign4test
   $ tar -xvzf LASTNAME-psuid-assign4.tar.gz
   $ cd assign4
   $ make
   $ ./verify
   ```

5. Send the tarball in an email with the subject line "311 submission" to both the instructor (`djpohly@cse.psu.edu`) and the TA for your section (listed on the course website). Put your own email in the Cc: line so you can be sure the email was sent properly and contains everything. **YOU ARE RESPONSIBLE FOR DOING THIS CORRECTLY.** Lost emails, emails not sent to both the instructor and TA, bad tarballs, and missing files are the student's responsibility and will be treated as late or missing. There will be no exceptions.

   This email should be sent by 11:59pm of the due date of the assignment; late submissions will be accepted up to **TWO** days late, with a penalty of 10% for each day. This late period is your extension; no further extensions will be granted.

Bonus Points

Implement the one remaining function, `image_get`. This function should retrieve image data from the MultiTron screen at the given coordinates, using the width and height already present in the image struct. The mask in the image struct should dictate which bits to retrieve; any other bits should be set to 0.

Reminder

As with all assignments in this class, you are prohibited from:

- Copying any content from the Internet.
- Discussing or sharing ideas, code, configuration, text, or anything else with others in the class.
- Seeking significant help from anyone inside or outside the class other than the instructor and TAs.

Likewise, you are responsible to protect your own code from falling into the hands of others. Consulting online sources is acceptable, but under no circumstances should *anything* be copied. Failure to abide by this requirement will result in dismissal from the class as described in the course syllabus.

Above all: when in doubt, be honest. If you do for any reason get significant help from a source, online, in person, etc., *document it in your submission* before we have to ask you about it. This will go a long way!