# Assignment 2

## More steganography

**Due date: 5th May, 2017**

This assignment builds on the code you created for Assignment 1. If your Assignment 1 code doesn't work, please make sure you read your marker's feedback on it. You can also come to consultation to get help. You don't need to get it working perfectly, but you do need to get it to hide and unhide messages. Even if your Assignment 1 code does hide and unhide messages properly, you should fix any memory leaks that your marker has identified.

There are three parts to this assignment, and we're releasing it at the start of Week 6. Task 1 can be attempted straight away, but Tasks 2 and 3 require some library functions that we won't cover in lectures until Weeks 7 and 8. We suggest starting with the functionality in Task 1 unless you feel like doing some indepent research.

### Task 1. Longer messages in multiple files

The program you wrote for Assignment 1 could only handle messages short enough to be hidden in one image, but what if you need to hide longer messages or use smaller images? One solution to this problem is to split the hidden message across multiple images.

Add a `-m` flag to both `hide` and `unhide`. This flag should work as follows:

```
hide -m number-of-files basename output-base-name
unhide -m basename
```

When given the `-m` flag, your `hide` program should look for a file named *basename*`-000.ppm` and hide as much of the message as will fit this image, outputting into *output-base-name*`-000.ppm`. If not all of the message has been hidden, then move on to *basename*`-001.ppm`, and so forth.

For example, suppose you want to hide your message, which is stored in a file called `secret`, in three pictures of kittens. You would rename your PPMs to `kittens-000.ppm`, `kittens-001.ppm`, and `kittens-002.ppm`. Then you would type

```
hide -m 3 kittens meow < secret
```

[1]

and the `hide` program would produce as output `meow-000.ppm`, `meow-001.ppm`, and `meow-002.ppm`. You would send these to your co-conspirators, who would type

```
unhide -m meow > secret
```

and the message would be extracted from the three input files.

#### Notes

It is obviously inconvenient that this system requires all the input images to follow a naming scheme. If you'd like to do something nicer then there are bonus marks available for implementing a different system – see below.

Your program should not assume anything about these images other than the filenames, e.g. you can't assume that they are all identical, or that they are all the same size. Do not attempt to hide anything in the headers, or they will no longer work as PPMs and your cover will be blown.

If the message requires fewer than *number-of-files* files, you may choose not to produce the corresponding output files, or you may produce output files that have no message encoded in them – it's up to you.

[1] The < character is doing I/O redirection, which you should have seen in the Week 3 lab. If you haven't gotten around to looking at that yet, maybe you should.

Depending on how you implemented Assignment 1, you may need to devise a new mechanism for determining where the message ends. Please bear this in mind.

You can assume that no more than 255 images will be needed; if your message is still too large to be encoded, your program is allowed to fail (although it should still fail *gracefully* rather than crash).

### Task 2. Parallel execution

Hiding messages in image files seems to work, but can be slow, especially when running on a computer small enough to be hidden in a shoe. Your co-conspirators have been wondering whether parallel execution to encode several messages at once might speed things up.

Implement a `-p` flag for `hide`. This flag should work as follows:

```
hide -p file
```

where *`file`* is the name of a file that contains the names of message files, image files, and desired output files, in this format:

```
firstmessage.txt  kittens.ppm meow.ppm
msg2 puppies.ppm woof.ppm
deathstar.plans artoo.ppm bleepbloop.ppm
```

Executing the above command should cause one process to spawn per line of this file, with each process hiding its message independently of the others. If given the sample file as input, the final results of the processing should be the same as separately running the following commands:

```
hide kittens.ppm meow.ppm < firstmessage.txt
hide puppies.ppm woof.ppm < msg2
hide artoo.ppm bleepbloop.ppm < deathstar.plans
```

You may assume that filenames do not contain spaces, are separated by whitespace, and that lines end with a newline (`'\n'`) character.

#### Notes

Your existing code probably reads its message from `stdin` rather than from a file pointer. If changing this would be awkward, there's a workaround – it is possible for a program to redirect its own `stdin` using the `dup2()` function:

```
FILE* myfile = fopen("new_stdin.dat", "r");
...
...
dup2(fileno(myfile), STDIN_FILENO);
```

Here, the `dup2()` function is overwriting `stdin`'s file descriptor with that of `myfile`. After this code executes, all functions that work with `stdin` will instead be working with `new_stdin.dat`, exactly as if the command had been run using I/O redirection[2]. We'll see `dup2()` again in Lecture 15, when we look at how pipes work. In the meantime, you are **not required** to use this technique if you don't feel it would make your program easier to write.

`fork()` and `exec()` are potentially dangerous in the hands of novices[3]. Back up early and often.

Don't forget to document any known limitations on this functionality in your README.

We will cover the use of multiple simultaneous threads in Week 6.

[2] In fact, the shell implements I/O redirection by executing a `dup2()` call.

[3] See `https://en.wikipedia.org/wiki/Fork_bomb` for one reason why.

### Task 3. Before-and-after images

Some images might be more suitable than others for hiding messages in. It would be convenient to give a side-by-side display of what the original image looked like and what the image looks like after hiding the message.

Extend your `hide` program to accept a `-s` flag that causes the program to show side-by-side[4] images of the input PPM before hiding and the output PPM. You must use the Simple DirectMedia Library (SDL) to display these images.

#### Notes

You are not required to support *any combination* of the `-m`, `-p`, and `-s` options in the same run of the program, although you must support them all in the same executable. If the user tries to use more than one option at the same time, you may display a suitable error message and end the program.

We will cover the use of the SDLC library to display images in Week 7.

### Task 4. Bonus: using a directory with multiple files

It would be very convenient if, instead of making users rename files to match the scheme (and risking exposure of the secret message), they could simply point `hide` and `unhide` at a directory full of PPMs.

For two **bonus marks**, extend your `hide` and `unhide` programs to add a `-d` flag. This flag will work like the `-m` flag, except for the way it handles input and output.

The command line for `hide` should look like this:

`hide -d` *`inputdir outputdir`*

On execution, `hide` should check that *`inputdir`* exists and is a directory. It should also check to see whether *`outputdir`* exists. If it does, the program should check that it is a directory; if not, the program must create it.

Then, the program should hide the message in each file in *`inputdir`*. Instead of using a numerical sequence to determine the order, you should use *ASCII order* (i.e. the order given by `strcmp()`). Output files must have the same filename as the corresponding input files and must be saved in *`outputdir`*. As usual, the message should be read on `stdin`.

#### Notes

This functionality will require you to do some independent reading. In particular, you need to read about `libc`'s facilities for manipulating directories. You will find this information in your computer's `info` system, but you might find it more convenient to use this online version: `https://www.gnu.org/software/libc/manual/html_node/Accessing-Directories.html`.

You are not required to support any other command line parameters if `-d` is used, and again you may output a suitable error message and end the program if your user tries to use more than one parameter at once.

In order to get any bonus marks, you must have made a solid attempt at Tasks 1-3. Don't go chasing bonus marks if you haven't gotten the basic functionality working. Also, please bear in mind that having a well-structured, readable, robust program is going to get you more marks than implementing the bonus feature: see the marking scheme below.

[4] Yes, one above the other would also be fine – the main thing is to have both images on screen at one, so that the user can compare them.

### Submission instructions

You must submit a single archive file in gzipped tar format (`.tar.gz`) through Moodle. See `info tar` (or consult the internet) for information how to create a gzipped tar archive.

You must include a working `Makefile` with your submission. It should be possible for the marker to build all the executables by changing to the root directory of your submission and running `make` at the command line. The use of `make` and `Makefile`s will be covered in lectures in Week 4.

You should include a `README` file, in text format, that describes (at a minimum):

- your name and student ID number
- how to compile and run the program
- what functionality is and is not supported
- and any known bugs or limitations.

This information is to help your marker build and run your program. That means that it is to your advantage if it is accurate: don't claim that you have implemented something that's not actually there. You'll lose marks for not following the submission instructions. Remember, you want your marker to feel happy and generous!

### Marks breakdown

Marks for this assignment are broken down as follows:

- Basic functionality:
  - `-m`/`-p`/`-s` flags work: **8 marks**
  - I/O and command line parameters: **1 marks**
  - Submission guidelines followed, i.e. tarfile extracts properly and `Makefile` builds: **1 marks**
- Code quality:
  - error handling: **3 marks**
  - memory management, including avoiding overruns and memory leaks: **2 marks**
  - process handling, including avoiding zombie processes: **2 marks**
  - readability/usability (modularity, use of `.c` and `.h` files, quality of `README`, etc.): **3 marks**

This assignment is out of **20 marks**. An additional **2 marks** are available for adding support for the `-d` flag to `hide` and `unhide`.