Purpose

`hblock(1)` is a shell script, available on homebrew, that blocks ads, beacons and malware sites. It does this by editing `/etc/hosts` and setting the IP address for such sites to 0.0.0.0. The issue is that hblock sometimes adds sites to `/etc/hosts` that are needed.

This executable fixes such issues by adding good DNS hosts to the exclusion list (`/etc/hblock/allow.list`) and removing the corresponding entry from `/etc/hosts`. It will also optionally flush the DNS cache and restart the `mDNSResponder` daemon.

Design Considerations

Where possible, will try to maintain the same structure in the C program as exists in the bash script, such as function and variable names.

File Locations

- Binary executable (`fix-hostfile`) located in `/usr/local/bin`
- Manpage (`fix-hostfiles.1`) located in `/usr/local/share/man/man1`
- API documentation located in project folder as `fix-hosts-apidoc.pdf` and `fix-hosts-apidoc.html`. Doc source files in `docs` folder.

Arguments

- **restore** : restores original hosts file, displays output
- **prep** : creates copy of original hosts file, displays output, calls `hblock(1)`
- DNS name to add with `-a` switch

Switches

- `-a, --add` : add DNS entry to allow.list, delete it from hosts
- `-f, --flush` : flush DNS cache and restart the `mDNSResponder` service
- `-h, --help` : Display usage

`main()`

- Handle switches
- Handle arguments
- Handle actions

`void usage(const char *program)`

- Display help to user

`int updateHostsFiles(const char *src, const char *dst, Action action)`

- Modify /etc/hosts

- PREP is essentially cp hosts{,-ORIG}
- RESTORE is the inverse
- if `(action == ACTION_PREP)`, run `hblock(1)`

```
int addDnsName(const char *hblock_dir, const char *dns_name,
const char *allow_file)
```

- Validate DNS name.
- Add valid DNS name to hblock exception list.
- Run hblock(1), which achieves the same result (i.e., removing the good DNS name from /etc/hosts) without the need for sed(1).

```
int dnsFlush(void)
```

- Verify running on macOS (Darwin).
- Flush DNS cache.
- Restart mDNSResponder daemon.
- if `action = ACTION_PREP`, run `hblock(1)`.

Results

At the start I believed that this program should have inferior performance to the bash script (`fix-hostfiles.sh`). First, there's the cost of instantiating a new process image. Second, part of the functions in the code required `system(3)`.

Therefore, the primary goal of this project was to see how well these same functions could be performed in a C program. While bash scripts are useful and easy, C is nicer to code in – at least for me.

In practice I didn't find the performance difference to be meaningful. It turned out that instantiation isn't as bad as it is for GUI programs, which are often surprisingly slow on macOS, despite plenty of free memory and CPU. Plus the program itself is small enough that runtime after instantiation isn't material to a human user.

Lessons

1. **Separate Folders**: It's better to have separate folders for each project as VSC does better with this.
    - Initially I tried having both the C program and bash executable in the same directory, but this caused complications with both VSC and git.
2. **basename**: There's a limitation in `basename(3)`; it reuses the same memory address. This messed up subsequent use of the library call within the same translation unit.
    - In `main.c` I had used `char *program = basename[argv[0])`, but this meant that the address pointed to was overwritten whenever

I called `basename()` again. E.g., when calling `fprintf(stderr, "%s, %d: argc: %d, optind: %d\n", basename(__FILE__), __LINE__, argc, optind);` subsequent, the address was overwritten and now the original value for `program` was lost.

- From the manpage: The `basename()` function returns a pointer to internal storage space allocated on the first call that will be overwritten by subsequent calls. `basename_r()` is therefore preferred for threaded applications.
  - In reality, `basename_r()` is preferred whenever you want to persist the returned string.
- The fix was: `char program[PATH_MAX]; basename_r(argv[0], program);`
  - `PATH_MAX` requires `#include <limits.h>`
- Interestingly, the memory address used by `basename()` does *not* persist over translation units. Therefore, I'm free to use `basename()` for the `fprintf()` calls I use for debugging and it works as expected; i.e., each TU gets its own unique memory address for calls to `basename()`.

3. **Rewriting**: The saying "it's not the writing but the rewriting" is true for coding as well.

   - I was surprised to discover things that I missed when creating the bash version of this. In retrospect, these changes should have been self evident.
   - For example, I had two functions (`copyHostsFile` and `restoreHostsFile`) in the bash script. Only when writing this is C did it become plainly obvious that these two functions should be in a single function (`updateHostsFiles`).

4. **Manpage**: I tried to write the manpage in markdown and then use `pandoc(1)` to create the manpage. This *mostly* worked, but introduced limitations in the output that ultimately proved not worth it. Having an existing manpage as a template and using that became easy enough without sacrificing control of the resulting output.

5. **Doxygen**: It's good to wait until the code is fully completed before adding doxygen API comments.

   - I'm ambivalent about whether I prefer these doc comments in the .c file or the corresponding .h.
   - I like the cleaner look of the c files sans api doc comments, but if these are in the .h file, the reader has to bounce back to the header file to see the api doc.
   - Also, for the doxygen VSC extension to work in the .h file requires that you explicitly name the variables in the header file; e.g., `void usage(const char *program)` instead of just `void usage(const char *)`.