Supplement: Writing ls(1)

Leonard König, April 12, 2021

Paths in UNIX

You are tasked to write your own small version of the UNIX program **ls(1)**. It makes sense to get yourself acquainted with the original tool, beforehand.

Try the following invocations and understand the output. Take special care to understand the meaning of '`.`' and '`..`', consult the visualisation **Figure 1** and/or do some research on your own:[1]

```
$ ls
$ ls .
$ ls -l
$ ls -l /
$ ls -l /etc/..
$ ls -l /etc/../home/./../
$ ls -la ~
```

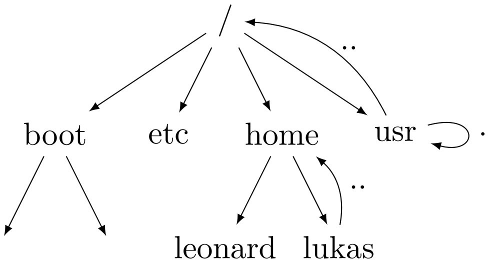

Figure 1 UNIX directory tree

Note: Don't worry, these pecularities of UNIX are already handled by the helper functions later on! However, these are not exposed by Linux *by default!* As with some earlier exercises, you should specify *before* any includes that you want to use the UNIX/POSIX standard from September 2008.[2]

```
#define _POSIX_C_SOURCE 200809L
```

[1] Extra: Write a program that simply prints the first argument and pass it '~', what's happening?

[2] Actually, functions like `getopt(3)` and `fstatat(3)` are older, however `scandir(3)` is a rather recent addition.

Parsing arguments

The ls program you are supposed to write shall accept arguments in the following fashion:

```
ls [-l|-a] [directory]
```

The first part are the *options* and you can pass anything from nothing to combined options like '`-la`' or '`-al`'. *After* these you can pass *non-option arguments*, in this case an optional directory, and if none is passed it should act as the argument was '`.`'. It would be tiring to deal with all those combinations manually, so instead you should use **getopt(3)**.

Make yourself acquainted with it, particularly read the linked manual page which also includes an example![3] Adapt it to your needs, it will prove as a good starting point for your own program. Make sure this part works before continuing with the next step.

Listing directory contents

After parsing the options and non-option arguments, you know which directories contents you are supposed to print, whether you should include files starting with a dot and/or sort them. Luckily there's also a function in UNIX that can just do that[4]: **scandir(3)**.

As with `getopt(3)`, there's a great example in the linked documentation which you should try to understand and use as the basis for this step. Again, make sure your code works, before proceeding!

Printing the file metadata

The final part of your program is to print the modification time if requested. Any files metadata can be queried using the function **fstatat(3)**.[5] You need to pass it an empty metadata structure which it will then 'fill' with meaningful data.

Again, the documentation contains example code of its usage. However, there, not `scandir(3)` is used but the less flexible variant `readdir(3)` which doesn't support sorting or filtering, but it's still useful to understand the functions behavior.

[3] What options does the example option string allow? What's the meaning of the first colon (`:`) and what the meaning of the others (they are different)?

[4] Yes, even sorting the entries (as part of a linked list) and filtering those with a leading dot!

[5] You could also try to use `stat(3)`, but then you'd need to concatenate the path to the passed direcory with each of its entries. This is inefficient and a huge source of bugs.

Sample outputs

I've included a few sample outputs, but you do not need to make your implementation behave exactly the same.

```
$ ls
ls.tex
Makefile
ls.c
```

```
$ ls .
ls.tex
Makefile
ls.c
```

Listing 2 ls on current working dir

```
$ ls -la /etc
Tue Jun  2 21:09:15 2020 .
Tue Jun  2 21:09:15 2020 ..
Tue Jun  2 21:09:15 2020 .hidden_file
Mon Jun  1 13:13:17 2020 boot
Sun May 31 12:38:45 2020 dev
Mon Jun  1 22:51:48 2020 etc
Mon Oct  7 18:43:56 2019 home
Sun May 31 12:38:19 2020 proc
Tue Feb 18 16:24:03 2020 root
Sun May 31 12:38:20 2020 sys
Tue Jun  2 21:07:47 2020 tmp
Mon Jun  1 22:51:48 2020 usr
Sun May 31 12:38:44 2020 var
```

Listing 3 Sorted output with modification time & Print hidden files