Lab Report on OCamlFUSE

Hanen Alkhafaji
`alkhafaji.2@wright.edu`
U00538764
w004hba

June 8, 2019

Abstract

This lab is intended to expose the user to the OCaml programming language and the OCamlFUSE project available on GitHub. There are familiarization tasks for the programming language, exploration tasks for the open source code of OCamlFUSE, and tasks for mounting a Google Drive storage on a Linux machine using OCamlFUSE.

pmateti: Well done! June 10, 2019

Contents

1 Objectives

1. Get comfortable with the OCaml programming language
2. Get comfortable with the OCamlFUSE source code
3. Successfully mount a Google Drive storage on a Linux machine using OCamlFUSE

2 Lab Experiment

pmateti: Describe each task.[1] Use subsections, and subsubsections.

2.1 Task: Online OCaml Lessons

2.1.1 Simple Expressions

2.1.2 Imperative Programming

2.1.3 Functions

2.1.4 New Examples by Hanen

2.1.5 Critique by Hanen

1. Lesson 1 - Simple Expressions This covered computing numeric values, defining strings, working with arrays, string manipulation, and defining and operating on tuples. Tuples can be made up of different data types. There are some functions built-in for tuples that have two elements. The functions presented in the tutorial were `fst` for getting the first element and `snd` for getting the second.

2. Lesson 2 - Imperative Programming `let` is the keyword used to set the results of some computation to a named variable. However, once a variable is set to a particular value using the `let` keyword, it cannot be modified to a different value. A compilation error results. The way to get around this constraint is to use the keyword `ref` on the right side of the `let` statement. That reference can then be modified. (`let x = ref 42;;`) `printf` is used similar to C to print formatted text to the terminal. Looping syntax is very similar to many other program languages, except when looping through a series of numbers backwards the word `downto` is used as opposed to `to` in the ascending direction. For comparison of values, the output is a boolean of either `true` or `false`. These greater than, less than, equal, or not equal to comparisons are not limited to only numeric values. The equal and not equal comparison symbols are similar to VB where a single equal sign represents an equivalence comparison while a less than sign followed by a greater than sign represents a non-equivalence comparison.

[1] In the courses, we had a Lab Assignment written up.

The only limitation is that there isn't support for comparing values of different types. However, functions like `string_of_int` can be used to convert an integer to a string so that it can be safely compared to another string. `if then else` logic is very straight forward. `while` loops logic is also easy to understand and uses a `while do done` structure.

3. Lesson 3 - Functions Functions can be defined in one line pmateti: Huh? using the `let` keyword very similar to defining a variable, but it takes arguments. One argument can be provided or several arguments in a tuple. Calling these functions is exactly the same as all other programming languages. Multiple values can be returned from a function by returning a tuple. Defined functions can also be called in a partial manner. This almost seems like extension methods from the C# world. A function that takes two arguments can be called with only one argument. However, it takes into account the value present at the time in which its called and uses that as the second parameter. `let mul x y = x * y`
`let double = mul 2`
`double 8` Anonymous functions are lambda expressions. They are functions defined without a name. These are useful for generating inline functions to be passed as a parameter to a function. In this case, they do not need to be assigned an identifier. Functions such as `List.map` and `List.fold_left` are useful for combining the power of anonymous functions and iterators to get a task done efficiently by iterating over a list and performing an operation on each value as the iterator visits each element.

4. pmateti: A couple of new examples of OCaml by Hanen?

pmateti: Give a brief opinion on the tutorial.

2.2 Task 2: Explore OCamlFUSE Source Code

`(https://github.com/astrada/google-drive-ocamlfuse)`

2.2.1 OCaml Development Environment

2.2.2 Study OCamlFUSE Source Code

1. OCaml development environment
(https://github.com/janestreet/install-ocaml)

   (a) Install opam
   `sudo add-apt-repository ppa:avsm/ppa`
   `sudo apt update`
   `sudo apt install -y opam m4` This was a success!

   (b) Initialize opam
   `sudo opam init -y --compiler=4.07.1` `sudo opam update -uy` `sudo echo $(opam env)`
   This was a success!

(c) Install libraries and tools `sudo opam install -y async core js_of_ocaml js_of_ocaml-ppx merlin utop ocp-indent` This installed so many things and took a while, but was a success!

(d) Test Installation

```
hanen@hanen:~$ git clone https://github.com/janestreet/install-ocaml
Cloning into 'install-ocaml'...
remote: Enumerating objects: 20, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 58 (delta 10), reused 14 (delta 6), pack-reused 38
Unpacking objects: 100% (58/58), done.
hanen@hanen:~$ cd install-ocaml/01-hello-world
hanen@hanen:~/install-ocaml/01-hello-world$ dune build hello_world.exe
hanen@hanen:~/install-ocaml/01-hello-world$ dune exec ./hello_world.exe
Hello, World
```

(e) Run Tests

```
hanen@hanen:~/install-ocaml/01-hello-world$
cd ../02-expect-tests
hanen@hanen:~/install-ocaml/02-expect-tests$
dune runtest
Done: 17/19 (jobs: 1)File "expect_test_example.ml",
line 1, characters 0-0:
diff (internal) (exit 1)
(cd _build/default && /usr/bin/diff -u
expect_test_example.ml expect_test_example.ml.corrected)
--- expect_test_example.ml      2019-06-08 15:37:18.946700012
-0400
+++ expect_test_example.ml.corrected    2019-06-08 15:37:21.494597583
-0400
@@ -2,5 +2,5 @@

let%expect_test _ =
let () = printf "foo" in
-  [%expect {| bar |}]
  +  [%expect {| foo |}]
     ;;
```

The test failed because there is a different in the actual results versus what was expected. The following commands will copy the results into what was expected and show that the tests will pass after that because there will no longer be a difference.

```
hanen@hanen:~/install-ocaml/02-expect-tests$ dune promote
Promoting _build/default/expect_test_example.ml.corrected to
expect_test_example.ml.
```

```
hanen@hanen:~/install-ocaml/02-expect-tests$ dune runtest
hanen@hanen:~/install-ocaml/02-expect-tests$ git diff
diff --git a/02-expect-tests/expect_test_example.ml b/
02-expect-tests/expect_test_example.ml
index 75a19d9..9bb1c70 100644
--- a/02-expect-tests/expect_test_example.ml
+++ b/02-expect-tests/expect_test_example.ml
@@ -2,5 +2,5 @@ open! Core

let%expect_test _ =
let () = printf "foo" in
-  [%expect {| bar |}]
  +  [%expect {| foo |}]
     ;;
```

(f) Editor Setup: Installed Visual Studio Code. Install a plugin for OCaml through Visual Studio Code by opening Visual Studio Code, pressing Ctrl+P, and entering `ext install hackwaly.ocaml`[2] into the text field. Once enter is pressed, Visual Studio code will automatically install the OCaml plugin.

1. Study OCamlFUSE source code source code I downloaded the source code from GitHub (https://github.com/astrada/google-drive-ocamlfuse) and opened it in Visual Studio Code.

   (a) ML versus MLI file The first thing I noticed was that there were duplicate files in the source folder that simply had different extensions. After some research, I found that the extension ML stands for Meta Language which is the umbrella programming language that contains OCaml. I suspect that the extension MLI stands for Meta Language Interface, but I cannot be certain of that since I was unable to find its expansion in my research. The reason I suspect it is Interface is because of what I observed when I looked at the files themselves. I opened bufferPool.ml and bufferPool.mli and compared them to each other. In the ML file, I found full function implementations while in the MLI file, I found a listing of function signatures. So, the MLI files must be interfaces that are used by other classes so as to hide the implementation in the ML from outsiders. pmateti: Compiled from ML to MLI.

   (b) Code Exploration

      i. There is a Make module in the concurrentGlobal file.
      ii. The file drive.ml appears to have the bulk of logic behind this implementation.
      iii. I picked a function from the buffer.mli interface file called `write_to_block` and did a project wide search for it. I found it referenced in the

[2]Insert citation.

drive.ml file as
`Buffering.MemoryBuffers.write_to_block` I do not understand why the case is different in the reference. Buffering when calling the function versus buffering in the definition of the file. I would not have made that connection before, but now am aware of it.

iv. Gapi shows up all over the code. It stands for Google API.

pmateti: Study OCamlFUSE source code: Study every file. Also, include sloccount.

2.3 Task 3: Mount Google Drive

2.3.1 Install OCamlFUSE

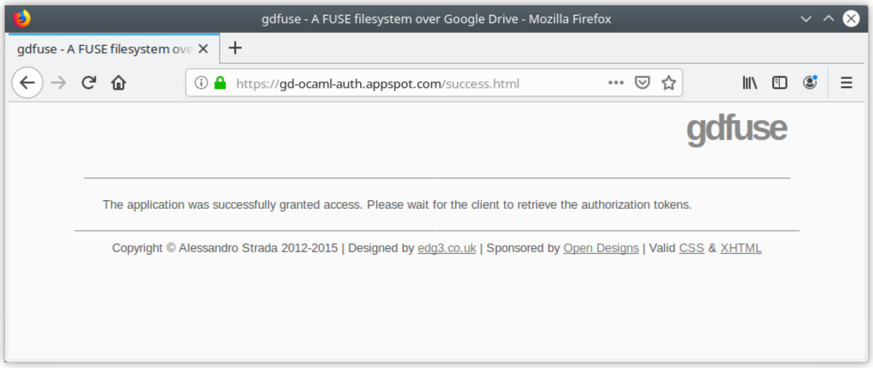
2.3.2 Allow Permissions through Browser

2.3.3 Make Directory and Mount Drive

1. Run commands to install OCamlFUSE

```
sudo add-apt-repository ppa:alessandro-strada/ppa
sudo apt-get-update
sudo apt-get install google-drive-ocamlfuse
sudo google-drive-ocamlfuse
```

2. Allow Permissions through Browser Allow gdfuse to see, edit, create, and delete all of your Google Drive files. Select Google account and allow access permissions.


gdfuse - A FUSE filesystem over Google Drive - Mozilla Firefox
gdfuse - A FUSE filesystem ove
https://gd-ocaml-auth.appspot.com/success.html
gdfuse

The application was successfully granted access. Please wait for the client to retrieve the authorization tokens.

Copyright © Alessandro Strada 2012-2015 | Designed by edg3.co.uk | Sponsored by Open Designs | Valid CSS & XHTML

Figure 1: Screenshot ocaml7.png

3. Make Directory and Mount Drive

```
hanen@hanen:mkdir ~/GoogleDrive
hanen@hanen:google-drive-ocamlfuse ~/GoogleDrive
hanen@hanen:cd GoogleDrive
hanen@hanen:ls
'2019-Hanen-CS 6970'    Misc
```

That final output matches the contents of that Google Drive in the browser as seen below. pmateti: Placement of figures is not quite in our control. So we must refer to afig by its number. See Figure 2

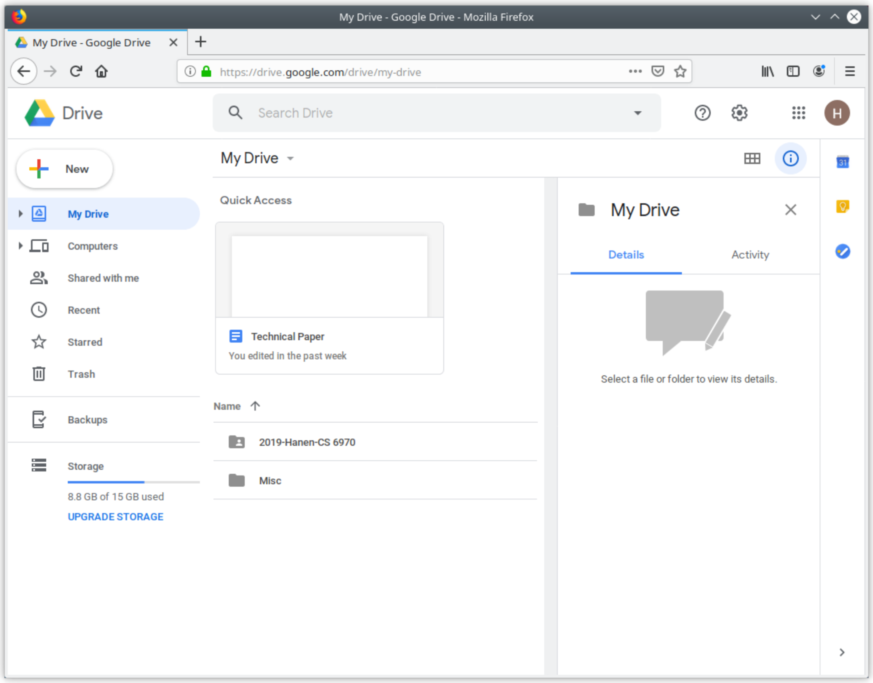


Figure 2: Content of GD as seen through the browser

3 Concluding Remarks

4 References

pmateti: We must have references. Properly. Author name, etc. I did a highly incomplete job below. Doing .bib can take an hour or two to learn – so do it in a way that Overleaf suggests.

Example Vangoor, Bharath Kumar Reddy, Vasily Tarasov, and Erez Zadok. "To FUSE or Not to FUSE: Performance of User-Space File Systems." In

15th USENIX Conference on File and Storage Technologies (FAST 17), pp. 59-72. 2017. PDF

OCaml Lessons `https://try.ocamlpro.com/`

Source Code `(https://github.com/astrada/google-drive-ocamlfuse)`