Introduction, and How I Do My Computing

Brett Gilio

<2020-07-27 Mon 19:07>

Contents

1 What is this blog about?

Well, after much request and many failed attempts I have finally crossed the edifice that is putting together a website. Friends of mine have long harassed me to put something together, to promote myself, and keep a journal of my thoughts for browsing. This is my best attempt at doing that, and I hope it can be useful for others who are passing through.

In lieu of formalities, my name is Brett Gilio, I am an aspiring mathematician and computer science something-or-other. In some other terminology, I care deeply about the relationship between formal mathematical logic and software development. I, as many of us do, sense a worrying devastation that is encroaching upon the world of software and computing. In many regards, these worries are ethical ones in that our computing has been effectively saturated and inundated with the unfortunate side-effects of surveillance capitalism.

It is becoming increasingly hard to appreciate a time where our computers were not connected to the internet continuously, because applications now adays tacitly demand exactly this. Furthermore, our rights to appreciate software in its one auditable essence has turned into a problem of propriety.

Without rehashing the eternal dogma of Richard M. Stallman and the GNU Project[1], the substrate of our computing world has been corrupted.

In terms of non-ethical problems, we have a great tendency in our hyper-capitalistic society to privilege profit maximization, as that is the essence of capitalism at its core. However, with such an agenda being at the center of an economic model, contrary to the claims of free-market hyper-efficiency, software has become increasingly less efficient[2]. One does not need to be a software engineer nor a statistician to appreciate this fact. Software in the 21st century continues to be sluggish in performance, with erroneously abundant side-effects, heavy in memory weight but without offering a semblance of type-safety, or even single-threaded memory safety.

This has become common-place in an industry that privileges software design practices that have not taken the time to properly investigate these very foundational issues that were at one time of great importance to highly revered computer scientists and mathematicians. Today, industry has adopted stateful, imperative logic with little regard to safety because of that profit motive. That software must be designed with weak safety because it takes too much time and care to craft something that is at the same time elegant, and deterministic in its behavior. Churn, churn, churn out that next feature!

These are all topics I want to touch on in this blog as it continues to develop. There is a plethora of content, and angles with which to dissect these problems. On the other hand, there is a lot of software, mathematical theory, and programming languages that I would like to spend time appreciating on this blog, for I feel they offer a path that needs to be forged in the "correct" direction. (Hint: It very likely does not have anything to do with Rust.)

2 How I do my computing.

Turning to a semi-related topic, I want to take time to express how I do my computing on a daily basis. Generally, my daily computer is a Lenovo Thinkpad T430. This is a nice computer as it is durable, quiet, has a nice battery life, and is alright in its computing power. The reason why I choose to use this computer over other, higher-end laptops has to do with my personal philosophy on software use.

Software freedom is always at the center of my computing mentality. I only want to use software that I can be assured is going to respect me, as

[1]`https://www.gnu.org/`

[2]`https://tonsky.me/blog/disenchantment/`

a user. For this reason, I use only distributions of GNU/Linux that use wholly free/libre software including the Linux-libre kernel. The Linux-libre kernel is wonderful precisely because it removes all of the problematic and unethical components from the vanilla Linux kernel that would impede my computer from having the high degree of ethicality I expect it to have. The T430 is generally free of problems when it comes to compatibility with the Linux-libre's blobless kernel. The only issue that i've had to resolve is wlan drivers.

By default, the T430 comes built in with integrated Intel iwlwifi card for wlan functionality. But as the iwlwifi driver code is not liberated I can not use it to achieve wireless functionality. As such I have purchased a ThinkPenguin Wireless USB dongle which uses the Atheros 802.11n PCI/PCI-E chip[3]. This chipset is usable with the ath9k driver which is fully compatible with Linux-libre!

My choice of distribution for GNU/Linux is the GNU Guix System[4], a project of which I am a regular contributor. I want to save my thoughts on the Guix project for another post, but for the lay reader it is a system and package manager that handles package management and system integration very differently than your Debian, Arch, or Ubuntu. I have never experienced a cleaner, more hygienic and reproducible system in my days of computing.

Additionally, packaging for GNU Guix is such a breeze with some very simple Scheme syntax. The power of this language makes it both abstractable to handle complex computations pertaining to a package derivation, as well as being simple enough to understand for a person with very little programming experience.

```
(define-public polyml
  (package
    (name "polyml")
    (version "5.8.1")
    (source (origin
              (method git-fetch)
              (uri (git-reference
                    (url "https://github.com/polyml/polyml")
                    (commit (string-append "v" version))))
              (file-name (git-file-name name version))
              (sha256
               (base32
```

[3]Looking at their shop, I don't see this listed anymore

[4]https://guix.gnu.org/

```
              "1ag7n55ym1zxksi89dvs17j6iaa58v4mg47s92zpa1b49k4fql1k"))))
    (build-system gnu-build-system)
    (inputs
     `(("gmp" ,gmp)
       ("lesstif" ,lesstif)
       ("libffi" ,libffi)
       ("libx11" ,libx11)
       ("libxt" ,libxt)))
    (arguments
     '(#:configure-flags
       (list "--with-system-libffi=yes"
             "--with-x=yes"
             "--with-threads=yes"
             "--with-gmp=yes")
       #:phases
       (modify-phases %standard-phases
         (add-after 'build 'build-compiler
           (lambda* (#:key make-flags parallel-build? #:allow-other-keys)
             (define flags
               (if parallel-build?
                   (cons (format #f "-j~d" (parallel-job-count))
                         make-flags)
                   make-flags))
             (apply system* "make" (append flags (list "compiler")))))))
    (home-page "https://www.polyml.org/")
    (synopsis "Standard ML implementation")
    (description "Poly/ML is a Standard ML implementation.  It is fully
compatible with the ML97 standard.  It includes a thread library, a foreign
function interface, and a symbolic debugger.")
    ;; Some source files specify 'or any later version'; some don't
    (license
     (list license:lgpl2.1
           license:lgpl2.1+))))
```

As it goes, with GNU Guix, a system which leverages a programming language in the family of LISP; my editor of choice is GNU Emacs. I am a semi-confident user of GNU Emacs. I do virtually all of my computing inside of this editor/environment/pseudo-operating system. While the extensibility of the Emacs Lisp programming language makes the editor such a joy to work with (and also a frustrating step-brother), I simply cannot divorce myself from the all-encompassing integration you get. I do my project planning, financial planning, email, git, software development, presentations, and more in GNU Emacs.

To further oust myself as a LISP junky, I use the X11 StumpWM[5] environment as my daily window manager. The mutable top-level in the Common Lisp implementation that StumpWM uses makes it very easy to have an environment that I can configure and re-configure on-the-fly. I do not have to spend time recompiling, and restarting my environment to tweak it how I want. The SLIME Swank server plugs right into a GNU Emacs process, and from there my environment can be changed in real time!

As far as networking goes, I spend most of my time on Telegram and IRC. For Telegram, I use a package named `telega.el`[6] which I co-maintain with brilliant Emacs Lisper and generally nice fellow, Evgeny Zajcev. You can find more details about this Telegram client in the Projects page of my website. For IRC I use the Freenode servers, and an IRC bouncer called ZNC[7]. If you are a regular user of IRC and are not using the ZNC bouncer, you are seriously missing out. From there I connect to the bouncer using the GNU Emacs client `erc`.

In terms of email and newsfeeds, I trust none other than the `gnus` newsreader and email client. Again, leveraging the power of the GNU Emacs environment, and Emacs Lisp, the `gnus` client features plentiful options for organization, communication, and niceties (such as snippet highlighting) that you simply will not find in other solutions. I have a particular disdain for webmail forms of email management, and while other local clients are fine enough for basic use the power of `gnus` is simply unparalleled in my opinion.

As far as my choices of programming languages goes, I unsurpsingly turn to the LISP family (Scheme, Emacs Lisp, Common Lisp) for a majority of my "light" programming. For more serious projects that I want to devote a degree of rigor, and ensure that performance and safety are well considered my choice tends to be some variation of C, OCaml, and/or Standard ML. I have familiarity with other purely-functional and impure functional programming languages such as Haskell, Mercury, F#, and Idris but they have never been my "go-to" selection.

The tooling for each of these languages is variable. As expected, C takes the mark for the most complete tooling. I use the CCLS LSP integration for virtually all of my work that involves the C programming language. For

[5]`https://stumpwm.github.io/`

[6]`https://github.com/zevlg/telega.el`

[7]`https://wiki.znc.in/ZNC`

OCaml I turn to the oft-trusted `tuareg` + `merlin` combination. Scheme has the fantastic geiser mode, which leverages the Scheme top-levels directly. Similarly, Common Lisp has two major options, SLIME & SLY, of which I go with the former. For all s-exp-based programming I trust the structured editing mode `paredit`. The only language I use regularly that does not feature much tooling would have to be Standard ML (though Mercury and Prolog come very close).

Lastly, for developing websites I try to stay as close to "bloat-free" as one can reasonably get. I have such high concerns for the world of browser-as-an-OS and Javascript-all-the-things, and what it will do to de-establish our friendly local, optimizable, and configurable free software solutions. In the past I have used the Scheme programming language in combination with David Thompson's Haunt[8] parser-generator. For this website I decided to change it up with the suggestion of another friend Alexandru-Sergiu Marton to try the Org-mode parser-generator.

3 Final thoughts.

I am more excited now that I have a first blog post finished to see what will come of this website and blog. I am eager to see what kind of reader base I will garner, if any. Additionally, I am hopeful that the readers who do approach my blog who are unfamiliar with functional programming or the other topics I address will do so with an open mind. On the other side, I hope that readers who are very familiar with all of what I have to say are not afraid to send me emails with thoughts, revisions, and suggestions for things I may have overlooked!

Especially when it comes to the more proof-intensive posts I will need all of the review I can get!

P.S. I love reading blogs. If you have a blog and think it might interest me, I want to include you in my blogroll[9]! Send me an email, or contact me on IRC (or Mastodon if I have one by then).

Have a response?

Responses and discussion pertaining to any of the blog entries on my website are welcome! Start a discussion on the mailing list by sending an email to

[8] `https://dthompson.us/projects/haunt.html`
[9] ~~`https://brettgilio.com/blogroll.html`~~

~brettgilio/blog-discussion@lists.sr.ht.

Errata:

- *<2020-07-28 Tue 18:06>* Include F# in the list of programming languages I have studied, but do not use regularly.
- *<2020-07-28 Tue 18:09>* Modify the footnote URL for the official GNU website to utilize HTTPS.
- *<2020-07-28 Tue 19:16>* Add a horizontal rule break between relevant sections.
- *<2020-07-28 Tue 19:38>* `s/approaching/encroaching upon`, more descript language in first section.
