XiX: Plan 9 in OCaml

Yoann Padioleau
yoann.padioleau@gmail.com

Abstract

XiX is a work-in-progress project to port parts of Plan 9 to OCaml: its build system `mk`, its shell `rc`, its ARM and MIPS toolchains `[5v][acl]`, but also its windowing system `rio`, and more.

1 Introduction

XiX, which stands for "XiX is XiX"[1], is a port of the C source code of major Plan 9 programs to OCaml[2], an industrial-strength functional programming language. Table 1 presents the current state of the project. The code of XiX is available at https://github.com/aryx/xix.

The first question about this project that comes to mind is: why? The main motivation for XiX actually comes from another project of mine called *Principia Softwarica*[3] (I submitted a separate paper to this workshop describing Principia Softwarica). In short, Principia Softwarica is a series of books explaining the code of Plan 9 programs that are essential to programmers, such as the linker, assembler, build system, and up to the windowing system. One strategy to fully understand a program, in order to explain its code in a book, is, perhaps surprisingly, to try to port it to another language. Indeed, during the porting of a program, it is tempting–because good programmers are lazy–to not implement certain features initially judged to be accessory, and instead focus on the main parts of the program. The process of discovering what those essential parts are already helps in understanding the original code. Moreover, very often the final ported program does not work at first because those features judged to be accessory turned out to be essential, which again helps to better understand the original code and its hidden subtleties.

I chose OCaml mainly because it is the programming language I am the most familiar with, and as it happens it is also my favourite. The port reached a point, however, where the XiX programs are actually useful on their own, independently of Principia Softwarica. Indeed, first, OCaml programs are usually portable and thanks to the OCaml compiler targeting multiple architectures and operating systems most XiX programs work without any extra effort on Linux, macOS, and Windows (plan9port[4] also helps to run Plan 9 programs on Linux and macOS), on amd64 and arm64 architectures. Second, the OCaml code being smaller (see Table 1) and arguably clearer, I found it easier to modify and I now usually experiment with new features (e.g., adding support for an `MKSHELL` environment variable in `mk`) first in OCaml, and only later port the feature to the original C code. Finally, OCaml programs are usually safer, as segmentation faults or buffer overflows cannot happen, for example.

2 Current state

The port of `mk` to OCaml, called `omk` (for `ocaml mk`), is almost complete, with only a few advanced features missing. The same is true for the shell `orc` and editor `oed`. In fact, the pair `omk/orc` is mature enough that it can now be used to drive the building process of XiX itself, dogfooding[5] those tools.

[1] XiX is the first project with a fully recursive acronym.
[2] https://ocaml.org
[3] https://github.com/aryx/principia-softwarica
[4] https://9fans.github.io/plan9port/
[5] The build of this LaTeX article was also driven by `omk`.

Category	Plan 9 Program(s)	LOC	XiX Program	LOC
Build system	`mk`	4350	`omk`	2240
Shell	`rc`	6500	`orc`	2190
Assembler	`5a va`	3020	`oas`	2190
Linker	`5l vl`	12980	`olk`	2910
C Compiler	`5c vc libcc`	28000	`occ`	5310
Editor	`ed`	1600	`oed`	1220
Graphics	`libdraw clock`	7280	`lib_odraw oclock`	2940
Communication	`lib9p`	2920	`lib_plan9`	1080
Windowing system	`rio`	5960	`orio`	2820
Utilities	`cat pwd wc`	173	`ocat opwd owc`	220
Total:		72783		23120

Table 1: Plan 9 and XiX Programs Statistics.
(**LOC** = lines of code, including code comments)

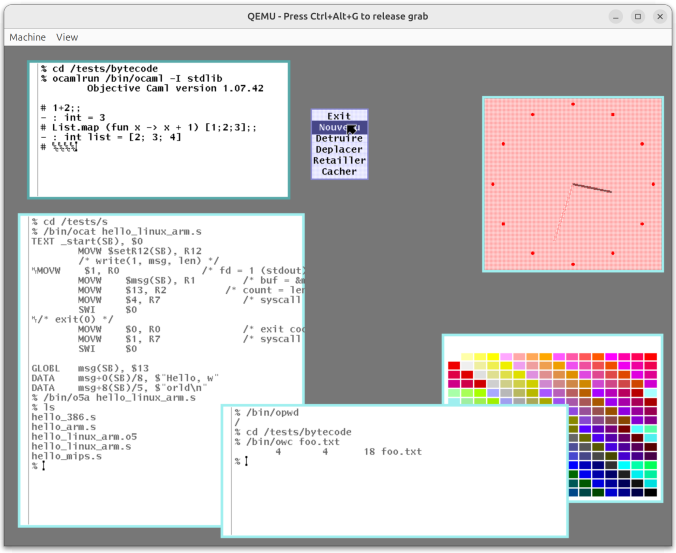

Figure 1: `orio` in action.

The ARM and MIPS assembler `oas`, linker `olk`, and C compiler `occ` are far from complete. However, they can already be used to compile and assemble a toy `helloprintf.c` program, and to produce `a.out` and `ELF` binaries that can run respectively on Plan 9 and Linux.

Finally, the port of `rio`, `libdraw`, and `lib9p` reached the point where one can substitute `rio` with `orio` in Plan 9 and not notice any difference, like in Figure 1, which illustrates also the use of a few other XiX programs. Note that `orio` is the only program in the list that requires Plan 9 to run as it relies on features such as namespaces only provided by the Plan 9 kernel.

3 OCaml in Plan 9

To be able to run `orio`, an OCaml program, in Plan 9 requires first to port OCaml itself to Plan 9, but no one has done so. Fortunately, OCaml has two compilers: one producing optimized *native code*, called `ocamlopt`, and one producing a portable *bytecode* for a *virtual machine*, called `ocamlc`; this bytecode must then be interpreted by a bytecode interpreter, called `ocamlrun`. Both `ocamlc` and `ocamlopt` are written in OCaml,

but the bytecode interpreter is a small C program (12 000 LOC) with very few external dependencies. I just had to add a few `ifdef`s, add a few functions calling the Plan 9 C library, and finally use the Plan 9 C compiler to cross-compile this C code to produce an `ocamlrun` (`a.out`) binary that could run under Plan 9. Thanks to this `ocamlrun` you can then run `ocamlc` itself (it's a pure bytecode program) to compile other programs. To run `orio` requires to compile additional OCaml bindings to the UNIX C library, and link C code implementing OCaml cooperative threads using `select()`. Fortunately, I was able to use the Plan 9 `pcc` program and APE[6] to compile almost as-is this UNIX C code.

This work is available at https://github.com/aryx/ocaml-light. I started from an old version of OCaml (1.07) because it had fewer dependencies than recent versions, making the porting work less difficult. This old version though is good enough to compile XiX because the XiX code does not use any recent features of OCaml.

4 C vs OCaml

C is a great programming language, and a real improvement over assembly. It is arguably the best language to implement system programs[7] such as kernels, virtual machines, just-in-time compilers, etc. C can also be used as a "portable assembler"; this makes C a great language to implement efficiently core libraries, which can even be used from programs written in different programming languages. For many applications though, especially applications without strong constraints on memory, speed, or latency, it can be far more productive for the programmer to use higher-level languages. Programming in C is indeed error-prone, with recurring bugs such as buffer overflows, segmentation faults, or security holes. OCaml, a statically-typed functional language, is arguably more expressive and less error-prone than C; it is almost impossible to have many of the bugs mentioned above while programming in OCaml.

The numbers in Table 1 seem to indicate that for the programs that have been almost fully ported (`mk`, `rc`, `ed`, and `rio`), the ratio of LOC between C and OCaml is around 2, meaning OCaml is 2x more succinct than C. However, the comparison is not totally fair because there are still advanced features in the C versions not ported yet to OCaml, so the real ratio is probably closer to 1.5. This is far lower than what functional programmers zealots (including myself) would claim when comparing C to OCaml. In fact, I have been constantly surprised while reading and porting the code of those Plan 9 programs by the succinctness and ingenuity of the C code. For example, C lacks OCaml algebraic data types (ADTs, similar to tagged unions), pattern-matching, parametric polymorphism, closures, garbage collection, and more. Still, the code of the Plan 9 linkers use simple enums to represent the opcodes and abstract syntax trees (ASTs) of assembled programs, enums and simple comparison functions to represent a hierarchy of operand classes, a large literal array containing enum constants associated with a function using a simple switch to represent cases for machine-code generation. All of those together allow one to very succinctly encode the final machine-code generation phase of the linker. The OCaml code instead uses ADTs and simple pattern matching to encode the same machine-code generation, which looks more elegant and natural, but in the end the OCaml code is not that much smaller than the C code. It is, however, arguably clearer to understand.

5 Future work

An obvious future work is to finish `oas`, `olk`, and `occ` to reach parity with the original C code, and then go even further by targeting also Linux, macOS, and Windows on the AMD64, ARM64, and RISC-V architectures.

Another one is to try to port the Plan 9 kernel `9` to OCaml. I was able to run a minimal kernel[8] in OCaml on the bare metal linked with Plan 9 C libraries such as `libc` and `memdraw`; I managed to print `hello world`, but a lot remain to implement a full kernel in OCaml.

[6] https://9p.io/sys/doc/ape.html
[7] Maybe nowadays Rust and Zig are also strong candidates.
[8] https://github.com/aryx/xix/tree/master/kernel