Proposing a Unix Portability Experiment

*Below is a memo that Steve Johnson and I gave to Elliot Pinson, our boss at the time, proposing the idea of buying a new machine to test the idea of porting Unix to a new architecture. It is, unfortunately, undated. Internal evidence from the document ("around March, 1976") suggests that it must date to ca. the beginning of 1976. The modified-date of the file as I found it is in 1978, but the file was certainly copied or fiddled, because the paper on the outcome of the experiment, "Portability of C Programs and the UNIX System", BSTJ **1**7 6, July-August 1978, confirms that the proposed machine (an Interdata 8/32) arrived in April 1977.*

Johnson's PCC compiler, which would be used for the Interdata project and would later be used for the VAX (Unix 32V) here and at Berkeley (BSD) and then also by important startups like MIPS, SGI, Sun, was evidently in its own late formative stage at the time of writing.

This memo, of course, wasn't a surprise to Pinson. He'd been quite supportive of the idea, and the memo follows a familiar style: researchers want to spend money and time on something, management wants reassurance that the researchers have a moderately coherent idea. Writing this was our equivalent of writing a DARPA or NSF grant proposal, but one that was already a lock.

To explain probably-unfamiliar references: ALTRAN was a language and system for doing symbolic algebra, primarily on polynomials and rational forms; it was mostly the brainchild of W. Stanley Brown. The whole thing was written in hyper-portable Fortran. #3ESS was a not-widely-deployed telephone Central Office for small communities; it used a unique processor (the AP-3). At the time it looked like fun and value for the Bell System for us to move Unix to it, but this didn't pan out.

Unix Portability

S. C. Johnson
D. M. Ritchie

What We Want to Do

We propose a project with three major goals:

1. to refine and extend the C language to make most C programs portable to a wide variety of machines, mechanically identifying non-portable constructions;
2. to write a compiler for C which can be changed without grave difficulty to generate code for a variety of machines;
3. to revise or recode a substantial portion of the Unix system in portable C, detecting and isolating machine dependencies, and demonstrate its portability by moving it to another machine.

From pursuing each goal we hope to attain a corresponding benefit:

1. improved understanding of the proper design of languages which, like C, operate on a level close to that of real machines but which can be made largely machine-independent;
2. a C compiler which can be adapted to other machines (independently of Unix), and which puts in practice some recent developments in the theory of code generation;
3. a working, if perhaps crude and incomplete, implementation of Unix on at least one other machine, with the hope that other implementations will be fairly straightforward.

What We Cannot Do

The common theme here is portability on a bold, apparently unprecedented, scale: we are attempting to make `portable' a multi-access, multi-programming operating system complete with enough utility programs to make it useful as a production tool. It is clear that the degree of portability promised cannot approach that of ALTRAN, for example, which can be brought up with a fortnight of effort by someone skilled in local conditions but ignorant of ALTRAN itself. We do not plan that the C language be bootstrapped by means of a simple interpreter of an intermediate language; instead an acceptably efficient code generator must be written. The compiler will indeed be designed carefully so as to make changes easy, but for each new machine it will inevitably demand considerable skill even to decide on data representations and run-time conventions, let alone code sequences to be produced.

Likewise, although we hope to isolate the machine dependent portions of the operating system into a set of primitive routines,

implementation of these primitives will involve deep knowledge of the most recondite aspects of the target machine, including the details of I/O operations and memory protection and relocation.

Next, the sheer bulk of code potentially involved is quite great, on the order of 100,000 lines, counting standard Unix utilities written in C but not subroutines or assembly-language routines and interfaces. Even if many of these utilities are dispensible at first, even if they are mostly portable anyway, even if we are able to discover non-portable constructs mechanically and unerringly, there will still be plenty of work in transporting them.

In view of the intrinsic difficulties of our project, we feel well justified in rejecting a number of sub-goals which might seem otherwise defensible. Thus, we cannot hope to make a portable Unix system compatible with software, file formats, or inadequate character set that may already exist on the machine to which it is moved; to promise to do so would impossibly complicate the project and, in fact, might destroy the usefulness of the result. If Unix is to be installed on a machine, its way of doing business must be accepted as the right way; afterwards, perhaps, other software can be made to work.

Where We Are

Each of the goals we mentioned initially has some characteristic problems associated with it. Since we have devoted careful attention only to the portability of C programs, and have uncovered several difficulties therein, we will discuss that aspect of the project.

Most of the C difficulties stem from the twin assumptions that pointers and integers require the same space in storage, and that pointers to an object of a given type (for example integers) are usable as pointers to a more finely-divided type (for example characters). The first assumption is stated explicitly in the C definition, and happens to be realistic on the only machines for which C has been completely implemented (PDP-11, H6000, S/370). It is used when two structures, which differ by substitution of an integer for a pointer, are assumed to be congruent; when the number `0' is used as an argument to a function expecting a pointer, to indicate a null value; and when a function delivering a pointer remains undeclared (and thus implicitly integer-valued). This assumption fails on some potentially interesting machines, such as the ESS #3 CC (a.k.a AP-3), which has 16-bit integers and 20-bit pointers.

The second assumption, that various sorts of pointers are compatible, is visible most dramatically in the current I/O interfaces to the operating system. The `read' call, for example, is equally happy to receive a character pointer as a destination for bytes as it is an integer pointer through which to store integers. This assumption too fails on many machines, including most word-oriented 16-bit machines as well as the Sigma 5.

If C, like PL/1 and Algol 68, required declarations of functions and of their arguments to be visible at the time of their use, the effects of these assumptions would be largely mitigated because the compiler could insert the appropriate conversions. It remains to be seen whether the burden of inserting such declarations is adequately compensated for. In any event, we have a useful tool (the `lint' program) for discovering inappropriate associations between integers and pointers; it can be modified, if need be, to produce the required declarations automatically.

Work on the portable C compiler is well under way; it uses the YACC-based `front-end' which has existed for some time. Examination of the operating system proper has not begun.

What We Want

We would like to obtain a new machine, and have easy access to it from the Research Unix machine, around March, 1976; by that time the new compiler should be working well. Two factors will govern the choice of this machine. It should not be so similar to the PDP-11 that the exercise becomes trivial, nor should it be so peculiar that the result is unnecessarily complicated or inefficient. Moreover, if the machine is attractive in its own right, then even the initial implementation of portable Unix may find an immediate market in the Bell System and elsewhere.