A Literature Review on Portability of C Programs and the UNIX System

Smruti Sohani[1] Virendra Dixit[2]

[1,2]College of Professional Studies, Dr. APJ Abdul Kalam University, Indore, India

Abstract— Computer programs are portable to the extent that they can be moved to new computing environments with much less effort than it would take to rewrite them. In the limit, a program is perfectly portable if it can be moved at will with no change whatsoever. Recent C language extensions have made it easier to write portable programs. Some tools have also been developed that aid in the detection of no portable constructions. With these tools many programs have been moved from the PDP-11 on which they were developed to other machines. In particular, the UNIX† operating system and most of its software have been transported to the Interdata 8/32. The source-language representation of most of the code involved is identical in all environments.

Note— † At the time UNIX was a trademark of Bell Laboratories; today the mark is owned by the Open Group.

Key words: C language, UNIX, Operating system, portability

I. INTRODUCTION

C Programming is an ANSI/ISO standard and powerful programming language for developing real time applications. C programming language was invented by Dennis Ritchie at the Bell Laboratories in 1972. It was invented for implementing UNIX operating system. C is most widely used programming language even today. All other programming languages were derived directly or indirectly from C programming concepts. This tutorial explains all basic concepts in C like history of C language, data types, keywords, constants, variables, operators, expressions, control statements, array, pointer, string, library functions, structures and unions etc. A program is portable to the extent that it can be easily moved to a new computing environment with much less effort than would be required to write it afresh. It may not be immediately obvious that lack of portability is, or needs to be, a problem. Of course, practically no assembly-language programs are portable. The fact is, however, that most programs, even in high-level languages, depend explicitly or implicitly on assumptions about such machine-dependent features as word and character sizes, character set, file system structure and organization, peripheral device handling, and many others. Moreover, few computer languages are understood by more than a handful of kinds of machines, and those that are (for example, Fortran and Cobol) tend to be rather limited in their scope, and, despite strong standards efforts, still differ considerably from one machine to another.

The economic advantages of portability are very great. In many segments of the computer industry, the dominant cost is development and maintenance of software. Any large organization, certainly including the Bell System, will have a variety of computers and will want to run the same program at many locations. If the program must be rewritten for each machine and maintained for each, software costs must increase. Moreover, the most effective hardware for a given job is not constant as time passes. If a nonportable program remains tied to obsolete hardware to avoid the expense of moving it, the costs are equally real even if less obvious. Finally, there can be considerable benefit in using machines from several manufacturers simply to avoid being utterly dependent on a single supplier.

Most large computer systems spend most of their time executing application programs; circuit design and analysis, network routing, simulation, data base applications, and text processing are particularly important at Bell Laboratories. For years, application programs have been written in high-level languages, but the programs that provide the basic software environment of computers (for example, operating systems, compilers, text editors, etc.) are still usually coded in assembly language. When the costs of hardware were large relative to the costs of software, there was perhaps some justification for this approach; perhaps an equally important reason was the lack of appropriate, adequately supported languages. Today hardware is relatively cheap, software is expensive, and any numbers of languages are capable of expressing well the algorithms required for basic system software. It is a mystery why the vast majority of computer manufacturers continue to generate so much assembly-language software.

The benefits of writing software in a well-designed language far exceed the costs. Aside from potential portability, these benefits include much smaller development and maintenance costs. It is true that a penalty must be paid for using a high-level language, particularly in memory space occupied. The cost in time can usually be controlled: experience shows that the time-critical part of most programs is only a few percent of the total code. Careful design allows this part to be efficient, while the remainder of the program is unimportant. Thus, we take the position that essentially all programs should be written in a language well above the level of machine instructions. While many of the arguments for this position are independent of portability, portability is itself a very important goal; we will try to show how it can be achieved almost as a by-product of the use of a suitable language.

II. HISTORY

The Computing Science Research Center at Bell Laboratories has been interested in the problems and technologies of program portability for over a decade. Altran [3] is a substantial (25,000 lines) computer algebra system, written in Fortran, which was developed with portability as one of its primary goals. Altran has been moved to many incompatible computer systems; the effort involved for each move is quite moderate. Out of the Altran effort grew a tool, the PFORT verifier [4], that checks Fortran programs for adherence to a strict set of programming conventions. Most importantly, it detects (where possible) whether the program conforms to the ANSI standard for Fortran [5], but because many compilers fail to accept even standard-conforming programs, it also remarks upon several constructions that are legal but nevertheless no portable. Successful passage of a program through PFORT is an important step in assuring that it is portable. More recently, members of the Computer Science

Research Center and the Computing Technology Center jointly created the PORT library of mathematical software [6]. Implementation of PORT required research not merely into the language issues, but also into deeper questions of the model of floating point computations on the various target machines. In parallel with this work, the development at Bell Laboratories of Snobol4 [7] marks one of the first attempts at making a significant compiler portable. Snobol4 was successfully moved to a large number of machines, and, while the implementation was sometimes inefficient, the techniques made the language widely available and stimulated additional work leading to more efficient implementations.

III. Portability of C Programs - Initial Experiences

C was developed for the PDP-11 on the UNIX system in 1972. Portability was not an explicit goal in its design, even though limitations in the underlying machine model assumed by the predecessors of C made us well aware that not all machines were the same [2]. Less than a year later, C was also running on the Honeywell 6000 system at Murray Hill. Shortly thereafter, it was made available on the IBM 310 series machines as well. The compiler for the Honeywell was a new product [8]. But the IBM compiler was adapted from the PDP-11 version, as were compilers for several other machines.

As soon as C compilers were available on other machines, a number of programs, some of them quite substantial, were moved from UNIX to the new environments. In general, we were quite pleased with the ease with which programs could be transferred between machines. Still, a number of problem areas were evident. To begin with, the C language was growing and developing as experience suggested new and desirable features. It proved to be quite painful to keep the various C compilers compatible, the Honeywell version was entirely distinct from the PDP-11 version, and the IBM version had been adapted, with many changes, from a by-then obsolete version of the PDP-11 compiler. Most seriously, the operating system interface caused far more trouble for portability than the actual hardware or language differences themselves. Many of the UNIX primitives were impossible to imitate on other operating systems; moreover, some conventions on these other operating systems (for example, strange file formats and record-oriented I/O) were difficult to deal with while retaining compatibility with UNIX. Conversely, the I/O library commonly used sometimes made UNIX conventions excessively visible--for example, the number 518 often found its way into user programs as the size, in bytes, of a particularly efficient I/O buffer structure.

Additional problems in the compilers arose from the decision to use the local assemblers, loaders, and library editors on the host operating systems. Surprisingly often, they were unable to handle the code most naturally produced by the C compilers. For example, the semantics of possibly initialized external variables in C was quite consciously designed to be implementable in a way identical to Fortran's COMMON blocks to guarantee its portability. It was an unpleasant surprise to discover that the Honeywell assembler would allow at most 61 such blocks (and hence external variables) and that the IBM link-editor preferred to start external variables on even 4096-byte boundaries. Software limitations in the target systems complicated the compilers and, in one case, the problems with external variables just mentioned, forced changes in the C language itself.

IV. The UNIX Portability Project

The realization that the operating systems of the target machines were as great an obstacle to portability as their hardware architecture led us to a seemingly radical suggestion: to evade that part of the problem altogether by moving the operating system itself. Transportation of an operating system and its software between non-trivially different machines is rare, but not unprecedented [9-13]. Our own situation was a bit different in that we already had a moderately large, complete, and mature system in wide use at many installations. We could not (or at any rate did not want to) start afresh and redesign the language, the operating system interfaces, and the software.

References

[1] B. W. Kernighan and D. M. Ritchie, the C Programming Language, Englewood Cliffs, N.J.: Prentice-Hall, 1978.
[2] D. M. Ritchie, S. C. Johnson, M. E. Lesk, and B. W. Kernighan, "UNIX Time-Sharing System: The C Programming Language," B.S.T.J., this issue, pp. 1991-2019.
[3] W. S. Brown, ALTRAN User's Manual, 4th ed., Murray Hill, N.J.,: Bell Laboratories, 1977.
[4] B. G. Ryder, "The PFORT Verifier," Software - Practice and Experience, (October-December 1974), pp. 359-377.
[5] American National Standard FORTRAN, New York, N.Y.: American National Standards Institute, 1966. (ANSI X3.9)
[6] P. A. Fox, A. D. Hall, and N. L. Schryer, "The PORT Mathematical Subroutine Library," AC M Trans. Math. Soft. (1978), to appear.
[7] R. E. Griswold, J. Poage, and I. Polonsky, The SNOBOL4 Programming Language, Englewood Cliffs, N. J.: Prentice-Hall, 1971.
[8] A. Snyder, A Portable Compiler for the Language C, Cambridge, Mass.: Master's Thesis, M.I.T., 1974.
[9] D. Morris, G. R. Frank, and C. J. Theaker, "Machine-Independent Operating Systems," in Information Processing 77, North-Holland (1977), pp. 819-825.
[10] J. E. Stoy and C. Strachey, "OS6--An experimental operating system for a small computer. Part 1: General principles and structure," Comp. J., 15 (May 1972), pp. 117-124.
[11] J. E. Stoy and C. Strachey, "OS6--An experimental operating system for a small computer. Part 2: Input/output and filing system." Comp. J., 15 (August 1972), pp. 195-203.
[12] D. Thalmann and B. Levrat, "SPIP, a Way of Writing Portable Operating Systems," Proc. ACM Computing Symposium (1977), pp. 452-459.
[13] L. S. Melen, A Portable Real-Time Executive, Thoth. Waterloo, Ontario, Canada: Master's Thesis, Dept. of Computer Science, University of Waterloo, October 1976.
[14] T. L. Lyon, private communication

[15] R. Miller, "UNIX - A Portable Operating System?" Australian Universities Computing Science Seminar (February, 1978).
[16] R. Miller, private communication.
[17] S. C. Johnson, "A Portable Compiler: Theory and Practice," Proc. 5th ACM Symp. on Principles of Programming Languages (January 1978).
[18] D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," B.S.T.J., this issue, pp. 1905-1929.
[19] D. M. Ritchie, "UNIX Time-Sharing System: A Retrospective," B.S.T.J., this issue, pp. 1947-1969. Also in Proc. Hawaii International Conference on Systems Science, Honolulu, Hawaii, Jan. 1977.

