TM-346

ARGONNE NATIONAL LABORATORY

A Comparative Study
of the
Fortran Development Environment
Provided by the
VAX/VMS and VAX/UNIX
Operating Systems

by

Richard C. Raffenetti

U of C-AUA-USDOE

APPLIED MATHEMATICS DIVISION

DISCLAIMER

The facilities of Argonne National Laboratory are owned by the United States Government. Under the terms of a contract (W-31-109-Eng-38) among the U. S. Department of Energy, Argonne Universities Association and The University of Chicago, the University employs the staff and operates the Laboratory in accordance with policies and programs formulated, approved and reviewed by the Association.

MEMBERS OF ARGONNE UNIVERSITIES ASSOCIATION

The University of Arizona
Carnegie-Mellon University
Case Western Reserve University
The University of Chicago
University of Cincinnati
Illinois Institute of Technology
University of Illinois
Indiana University
The University of Iowa
Iowa State University
The University of Kansas
Kansas State University
Loyola University of Chicago
Marquette University
The University of Michigan
Michigan State University
University of Minnesota
University of Missouri
Northwestern University
University of Notre Dame
The Ohio State University
Ohio University
The Pennsylvania State University
Purdue University
Saint Louis University
Southern Illinois University
The University of Texas at Austin
Washington University
Wayne State University
The University of Wisconsin-Madison

ARGONNE NATIONAL LABORATORY
Argonne, Illinois 60439

Applied Mathematics Division

A Comparative Study of the Fortran Development Environment Provided by the VAX/VMS and VAX/UNIX Operating Systems*

Richard C. Raffenetti

November 1979

Technical Memorandum No. 346

Intended Primarily for Internal Distribution

*Work performed under the auspices of the U.S. Department of Energy.

TABLE OF CONTENTS

Appendix page

Chapter 1

INTRODUCTION

The contents of this report are the result of a study to compare two operating systems which are available to control the Digital Equipment Corporation's (DEC) VAX-11/780 computer. The two operating systems are the VAX/VMS system from DEC and the UNIX system which is a product of AT&T's Bell Laboratories marketed by the Western Electric subsidiary. The information collected and presented here is intended to aid in the selection of an operating system to be used to control the VAX-11/780 hardware which will be operated in the Applied Mathematical Sciences Research Section of the Applied Mathematics Division (AMD). Support for this study was provided by the Applied Mathematical Sciences Research Section. The objectives of the study focus upon the Fortran development environment of each system. In other words, how good is either system as an environment in which numerical algorithms can be converted to programs and with the aid of which their mathematical properties can be studied. The scope of this study was to evaluate primary development tools including the file systems, the text editors, the Fortran compilers, and the linking loaders, and also execution performance. A number of other tools have also been evaluated. Of greatest secondary interest is the interactive debugging capability of either system.

The method by which the evaluation was carried out was to access an available system of each type by means of a 300 baud dial-up line and execute as many of the development tasks as seemed feasible in about a three-week period. The approach taken for each phase of the development process is described within each of the following sections. No attempt is made to be exhaustive in the evaluation of either system. Each system has many other useful features which could not be tested in the alloted time or described here.

The evaluation is presented in a comparative manner only to the extent that properties of the tools for each development process are described together, each in the same report section. Most conclusions as to which subsystem might be "better" would be entirely subjective. Therefore, such conclusions are avoided except when the outcome is clear, e.g., when one system has a defective feature or some feature is missing entirely.

The machines employed in this evaluation were the VAX/UNIX system being operated in the research group of Prof. Richard Fateman at the University of California, Berkeley and the VAX/VMS system being operated in the High Energy Physics Division at Argonne National Laboratory. Thanks are due Richard Fateman and Richard Newton of Berkeley and Jerry Gieraltowski of ANL for their aid in getting access to the machines and in answering questions.

This project also benefitted from discussions with Jack Dongarra, Burton Garbow, Paul Messina, and Jesse Wang of the AMD.

Chapter 2

LEARNING THE SYSTEMS

The first step in assessing the VAX operating systems was to learn how to use them. The availability and quality of documentation is by far the first consideration.

2.1 OFF-LINE DOCUMENTATION

Access to a complete set of VMS manuals provided all of the information necessary to log in and get started on that system. The "Primer" manual consisting of about eighty pages is sufficient to get a casual user started. It covers important features of the editor, the Fortran compiler, the MACRO assembler, the link editor, the file system, the file system utilities, and the command language including running a program and writing command procedures. Examples were a part of this brief but very useful document. Additional manuals are many and are well written. They are organized to provide fast access to the many optional features of the areas listed above. Examples are used widely. In addition, there are manuals which cover material of interest to the system programmer. VMS has protection capabilities which prevent unauthorized use of certain system services. System services can be called from a high-level language thus removing the need to program system tasks in assembler. An introduction to system capabilities is provided by the manual entitled "Summary Description."

Similar information about the UNIX system was less accessible to this project. The concepts and facilities which are provided are presented in the Bell System Technical Journal (BSTJ), volume 57, published in 1978. The entire issue is dedicated to a discussion of UNIX separate from and prior to the present implementation on the VAX-11/780. Of great value to this study in early work were two manuals which describe version 6 of UNIX which is implemented on the AMD PDP-11/70. One manual consists of a number of sub-documents which describe at greater length the main facilities of UNIX including the editor, Ratfor, the assembler, the C language, text processing, and the IO system. The second manual, the UNIX Programmer's Manual, consists of short descriptions of the function of each of the UNIX system commands, system calls, subroutines, special files, file formats and conventions, user maintained programs and subroutines, and system maintenance facilities. These descriptions include functions, syntax, options, files needed, references to documents or related commands, and known bugs. The environment of UNIX development was considered to be friendly and accordingly, convenient, foolproof protection facilities were not made available.

The style of the documentation of UNIX is brief and sometimes rather sketchy. It would appear though that a user who is well acquainted with UNIX design concepts would not find this to be true. Indeed it has been possible to infer the actual method of operation or syntax of a command option, for example, by its similarity to concepts of the same type used in other commands.

2.2 ON-LINE DOCUMENTATION

All UNIX documentation comprising the two manuals described previously is available on-line in machine-readable form. The few UNIX Programmer's Manual pages which describe any command can be displayed at the terminal using the "MAN" command. One may also receive a brief, one-line description of each command by using the "WHATIS" command. In addition, the WHATIS database which is in alphabetical order forms a suitable table of contents for the manual. Whereas the entire manual is easily available it is a bit difficult to make best use of all UNIX facilities in an effective way without having even an out-of-date hard copy of the manual on hand. The table of contents provides the supplementary up-to-date source document which informs the user of new and updated features. Besides the UNIX Programmer's Manual, the more extended documents described in an earlier section can be formatted and displayed from on-line or archive files. The extent of this latter material might force an installation to keep it off-line.

The equivalent facility in VMS is provided by the "HELP command. This does not provide access to the VMS manuals but to similar information stored in a hierarchical manner. Qualifiers to this command provide access to various hierarchy levels or to functional areas of the descriptive material. VMS provides information for all of the commands via the HELP facility.

In addition to its "MAN" command, UNIX sometimes prompts the user for more information and at the same time allows him/her to obtain help. In this case the help document which is produced contains a summary of the MAN document.

2.3 TUTORIALS

UNIX provides two kinds of tutorial documentation. The document manual contains sections which are either explicitly tutorial by title or implicitly tutorial by their method of description. In addition, the "LEARN" command of UNIX provides access to interactive tutorial sessions by which one can be taught one of a number of useful topics including the file system, the text editor, macros, and typesetting of mathematical equations. In these sessions the user employs UNIX in true computer-aided instruction. VMS does not have such tutorials but the extensive documentation does contain many examples which serve as a kind of tutorial.

2.4 MAIL

The mail facility of UNIX is yet another source from which to obtain information about system facilities and by which to relay problems to the system manager or other users. For the former purpose the existence is presupposed of a person with sufficient knowledge, time, and interest to respond to one's questions. One may also communicate directly with someone who is also logged in but in such a case a real-time response may not be available. The VMS system does not have a mail facility although one had been implemented on the VMS system which was used in these comparisons. The source of this software was the Stanford Linear Accelerator Center (SLAC).

Chapter 3

BASIC SYSTEM ELEMENTS

Any interactive system has as its basic operations the manipulation of files which contain source programs, data, text, etc. This section describes the facilities of the UNIX and VMS operating systems with regard to these operations.

3.1 THE FILE SYSTEMS

Recognizing the important role that file manipulation takes in interactive computing both UNIX and VMS provide a set of file utilities which are available as primary commands and not as a utility subsystem as in PDP-11 systems. The file systems are also similar in that users are assigned a file directory and may create several levels of subdirectory as needed to separate their files by project, usage, etc. The file systems are hierarchical and all directories can be traced back to a master file directory. In accord with the friendly environment concept, UNIX users may change their directory upwards as well as downwards by employing the symbolic representation of each current directory's next higher level directory, "..". A UNIX user may therefore gain access to the entire file system without knowledge of the system's contents. This is thwarted somewhat by the fact that even though they are elements of a directory listing, subdirectory names need not look different from file names. The VMS system allows users to change their directory but there is no way to make the master file directory the default. Thus to search the file system one must instead obtain a list of user directories to gain access to them. Subdirectory elements in a directory listing are easily distinguished by the suffix "DIR". Both VMS and UNIX provide protection features but as described earlier, the protection concept was not a design goal and therefore UNIX capabilities in this regard are lacking in utility. When the user is working in the environment of a given default directory he may refer to files in another directory by giving the fully qualified file name including directory names. Both systems have this feature.

3.2 THE COMMAND LANGUAGES

Operations which manipulate directory elements (files, subdirectories, or command procedures) are referred to as system commands. Examples are: "FOR" which compiles a Fortran source file and produces an object file and "CAT" which displays (types) a file at a terminal. Both

systems provide a rich assortment of commands for use in carrying out system provided processes or tasks. In both systems all common tasks are primary commands which do not require the user to drop into a subsystem before entering further commands. Obvious exceptions on both systems are the text editor systems. VMS also provides the "MCR" command which gives VAX users access to RSX-11M system components in a manner which is compatible with the RSX-11M operating system. In this fashion, development of RSX-11M software can be carried out on the VAX hardware.

Both systems provide a realization of the concept of command procedures implemented as files containing properly formatted commands in a simple "command language" syntax. Parameters may be substituted at run-time in a procedure. Nine parameters are available in UNIX and eight in VMS. The procedures consist of commands and control structures such as loops, logical tests, and branching. Procedures may also be nested, i.e., they may call other procedures. In its command language UNIX also provides the concept of a "pipe", a device which allows communication between processes without the explicit existence of a file. Synchronization of the access of each command process to the pipe buffer is controlled by the operating system. This device simplifies the organization of procedures which process discardable intermediate information in a sequential fashion. Creation and deletion of such intermediate files may be an obstruction to efficiency because of limitations posed both by IO devices and by peripheral IO media.

Both systems provide the capability of detaching a process from the terminal. This permits the user the ability to carry out other tasks using the terminal while he is waiting for an earlier process to finish. This may be thought of as a "submit" feature where the process goes to a batch-like system. The scheduling of such processes by the system is an important question. VMS allows control over scheduling by various prioritizing options. In UNIX there does not appear to be an analogous feature. UNIX treats all active processes as equal in terms of resource allocation. UNIX does however provide some features which appear useful to keep a process in inactive status until certain external conditions are met. The UNIX "shell" or command interpreter was a major design feature of the system and hence provides many significant capabilities of a utilitarian nature for the time-sharing environment.

VMS provides a card-oriented batch capability in addition to the time-sharing environment. Control of the batch processing is through command directives which intermix with source and data card decks.

Both systems have convenient and powerful command syntax which is similar. In UNIX an abbreviation, often comprising two or three characters, represents the command verb and is followed by keywords and operands which are usually file specifications. VMS provides longer commands which are sometimes two or three words but which may be abbreviated down to those characters which make the command designation unique. Keywords and operands follow as they do in the UNIX system. As might be expected, the specific syntax of the two systems is different.

Chapter 4

TEXT MANIPULATION

The time-sharing development environment requires a small number of utilities which support preparation and management of text comprising program input and output. These utilities deserve special emphasis since they provide the user with his primary interface to the development environment.

4.1 TEXT MANAGEMENT

The means of text management is the file system of each of the operating systems. Development projects on interactive systems ordinarily produce text files which are related in content. For example, if a Fortran program is compiled, then a file containing object code is produced in addition to a compilation output listing. Ordinarily the listing is diverted to a file for optional future use. In both systems files which are related are given different names by appending a qualifier following a "dot" separator. Fortran source files must be appended with the proper qualifier for the compiler but the qualifier need not be typed by the user when defaults are applicable. VMS uses a three character qualifier while UNIX uses only one. If a UNIX system process is creating a file with the same name as one which existed previously then the old one will be lost. VMS, on the other hand, maintains a file version number automatically and preserves old versions until they are removed by the user. A process will open and use a file with the highest version number unless the version is explicitly given. VMS thusly affords the user a means of self-protection; the burden on the user is to clean up old versions of his files. With the UNIX system the user is forced to maintain a copy of his file separately if he wishes to protect it. An archive facility in UNIX provides an easy way to do this.

Text files on both systems contain no blank fill to the right of the last character on a line and tabs can be used to remove some blanks which would appear to the left. Further means for compression or decompression of text files appear to not be a part of the standard VMS or UNIX systems.

4.2 EDITING

Both systems have text editors which afford the user a means to enter and modify text files. These files might be language source code, data, formattable text, etc. For this report the primary editor of each system was used: on UNIX the editor is entered by the command "ED" and in VMS the editor, called SOS, is also invoked by the command "EDIT" or "ED". Of interest to possible readers of this document is the fact that the alternate editors TECO and WYLBUR are implemented on the VMS system. TECO is invoked under the RSX-11M subsystem and WYLBUR (for the VAX) was developed by SLAC. Alternate experimental editors are available on the UNIX system as well as a subset of ED which is used during the symbolic debugging process to examine the source code file.

The UNIX editor is basically a context editor which maintains and can use line numbers also. During an editing session the line numbers change as a result of additions to or deletions from a text file. The editor syntax is simple and does not require an abundance of multi-keystroke characters. Slashes delimit strings and a comma separates the delimiters of a range, these being either strings, line numbers, or a combination. The characters "." and "$" stand for current line and last line respectively. The top line is always line one although a specification of "0a" allows appending text to the zeroth line (or equivalently before the top line). Heavily used editor concepts such as "find string", "substitute string", "print line", "step ahead or backwards" are extremely simple. The editor provides the ability to move text around in the text editor buffer as well as the ability to read text in from a text file and insert it into the current file being edited. A range of further capabilities whereby one can increment or decrement line specifications and use wildcards and other codes in strings rounds out a powerful editing capability.

A possible disadvantage of the UNIX editor stems from the recognition of case. Lower and upper case are distinguished by the editor in strings. Thus if text files have mixed case or are exclusively upper case the user will find himself employing the shift key rather often. However, it would appear however that if the working environment were exclusively a UNIX system then one might choose to employ lower case all the time except when upper case is explicitly required (e.g., in formattable text files). The Fortran compiler recognizes the lower case alphabet and moreover converts characters to upper case in the printing of literal strings.

The VMS editor employs a more complicated syntax which differs substantially from editors in use on the CCF machines. This editor appears to be primarily a line oriented editor although context searches and string substitutions are possible. Ranges of text however cannot be specified by the use of context string delimiters. In the text manipulations tested for this report no need was found for the use of optional page numbering which might be considered as another level of line numbering. The added syntax necessary to specify pages would seem to be a burden on the user and no advantages are apparent at this time. Although this feature is optional, the editor is said to sometimes insert

page numbers, for instance, when a file is being copied. Page marks may be deleted. Strings are delimited only by the escape character (which prints as "$") and line ranges require a colon as a separator. The characters "↑", ".", and "*" conveniently represent the top, current, and bottom lines respectively in the absence of page marks. The absence of straightforward defaults is annoying. For example, to print the current line one must enter "P." rather than simply "P". Another rather annoying characteristic is that the substitution command functions globally unless something is done to avoid it. One alternative is to specify "decide mode" which will interrogate the user prior to making each substitution. A second alternative is to enter "alter mode" which allows changes to be made directly on the text without the use of the substitute command. In this mode one can skip over words and delete or change characters very easily. (It should be noted also that at least one of the UNIX system editors which is experimental at the time of this report has an equivalent capability). The VMS editor is also aware of case. String matching commands will match lower case input to either lower or upper case text unless the "exact" keyword is specified. On substitution the case of the replacement string is followed literally. As with the UNIX system, VMS Fortran recognizes the lower case alphabet so that a user, to avoid mixing case, might choose to develop source code exclusively in the lower case alphabet. A "SET TERMINAL" command affords the user a means of describing his terminal's real or imposed capabilities to the VMS system. Thus the question of case utilization can be resolved without users being forced to adopt methods of operation which are foreign or undesirable to them. A range of wildcard-type codes usable in a variety of commands provides powerful editing capabilities.

A significant disadvantage of the VMS editor in connection with the Fortran compiler is that the line numbers of the editor do not agree with the Fortran line numbers. This is a severe problem when one must debug the code. The error messages produced in compilation, linking, and execution contain the Fortran compiler-produced line numbers. The editor therefore does not effectively complement the Fortran development environment. In the UNIX system the Fortran compiler apparently adopts the line numbering of the source which is in accord with the editor. More will be said about the debugging process in a later section of this report.

In addition to SOS the VMS system has a batch-style editor, also keying on line numbers, which is used to maintain system source code. System updates are provided using the SLP editor subsystem.

4.3 LISTINGS

Both the VMS and UNIX systems offer facilities to obtain file listings. Both printed and displayed files are included in this category. The terminal file listing is obtained by use of the "TYPE" command in VMS whereas the "CAT" command is used in UNIX. It is interesting to note that the VMS terminal IO recognizes the presence of carriage con-

trol characters and makes proper use of them, including the "+". The UNIX system displays the carriage control characters as part of the listing.

A print command in both systems allows the user to dispose file listings to a system printer. In VMS the list file produced by the Fortran compiler is rather useful considering the previous discussion of line numbers. This listing contains both the editor line numbers of the source code and those used for reference by the Fortran compiler. Access to a printer is invaluable in the development process. Both systems provide formatting features and time and date stamping for printer files. These files are turned over to a spooling system. Whether or not the spooled print queue is managed by a scheduling algorithm which is different from FIFO has not been investigated.

4.4 TEXT PROCESSING

A detailed description of the features of the text processing systems is not within the scope of the present report. However it is perhaps widely known that the UNIX system supports a collection of text processing and text formatting capabilities. In addition, there is included an equation formatter for typesetting applications. In contrast, VMS has no system-supplied software for this purpose.

Chapter 5

FORTRAN LANGUAGE SUPPORT

The Fortran language is of primary interest in this report because it will probably continue to be the primary development language in the Applied Mathematical Sciences Research Section of the AMD. Accordingly, this section focuses attention on the Fortran compiler, on subsystems for loading or linking object code, and on facilities for maintaining load-time libraries. A later section of this report will deal with the execution and interactive debugging of such programs.

To test the reliability of the Fortran compilers, a source code which solves an ordinary, one-dimensional, boundary-value problem numerically was transferred to each system. The main control program code consists of about 110 lines of Fortran code (excluding comments) and conforming to the "66" standard. The code calls two EISPACK subroutines called BISECT and TINVIT consisting of 136 and 126 Fortran statements respectively. The main program had been written and debugged on the CCF using the IBM Fortran H Extended compiler. Both VAX compilers provide a code optimization option, the effect of which will be discussed in a section of this report describing run-time or execution characteristics.

The Fortran compilers on both systems produced correct object code for this program as evidenced by execution of the loaded code which produced identical results to those obtained on an IBM 3033. Further tests of the compilers were to introduce syntax errors and to rename subroutines so as to cause both compile-time and load-time diagnostics. Care was taken to not disable the production of warning messages either by choice or default. Whereas this testing was not exhaustive, some interesting results did emerge.

5.1 THE FORTRAN COMPILERS

The Fortran compiler supplied with the VMS system is called Fortran IV-Plus and is an implementation of the Fortran 66 standard with extensions, many of which are found in the Fortran 77 standard. Not all syntax errors were detected by this compiler. Those detected included unbalanced parentheses, an undefined statement number, the redefinition of a "DO loop" index within its range, and a double comma preceding the increment variable in a DO statement. In the case where errors were detected, the internal statement number was given in addition to a short message and a fragment of the source statement including or near to the erroneous syntax. This fragment is valuable because of the editor's use of line numbers different from the Fortran compiler's statement numbers.

Thus it can be used for context searching. Errors not detected included two other occurrences of double commas, one in an IO list and one in the argument list of a subroutine call.(1) In addition, the interactive compilation messages do not warn of unused variables or undefined variables. A compilation listing containing cross-reference maps can be produced on a file for later use and disposal. Optimized object code is produced by default. The user's guide lists nineteen kinds of optimization which are carried out. The default may be overridden for debugging.

The UNIX Fortran compiler conforms to the Fortran 77 language standard and has many of the useful extensions available in other Fortran compilers. For example, it includes nearly all of the 66 standard features as a subset. Conflicting elements such as "onetrip do loops" are accessible via a keyword option to the compiler. All deliberate syntax errors were detected by the compiler including unbalanced parentheses, the redefinition of a "DO loop" index in its range, double commas in an IO list, an argument list, and a "DO loop", and an undefined statement number. The compiler did warn of the presence of an unused local variable but did not point out a variable which was undefined prior to use. A listing containing the usual cross-reference maps of variables, statement numbers, and other language elements cannot be produced by any option. Other software development tools provide some or all of these useful facilities. The C language code which is produced as an intermediate step is optimized optionally. No optimization is carried out at the Fortran language level.

5.2 THE LINKING LOADERS

The command "LINK" in the VMS system accepts as parameters keywords and several object code files and user libraries. A task module is built which then may be executed. The linker points out unresolved references and tells the interactive user which program unit was referring to them. Several keyword options control the content of an optional image map file. The useful cross-reference map of program units and storage elements is available. The image may be executed notwithstanding the presence of unresolved variables. By default the linker includes information enabling the execution monitor to produce a traceback in the case of a run-time error.

In the UNIX system linking is carried out subsequent to compilation unless only compilation is requested. Moreover, a list of files consisting of only object programs may be loaded using the compiler command. The file "type" triggers the use of the compiler and loader or just the loader. The advantage of not using the load command is that the Fortran default libraries are known to the compiler command but not

---

(1)The absence of arguments in a subroutine call is purposeful in this system. The compiler inserts zero values for the arguments. Thus, optional parameters in system service routines may be omitted.

to the loader. If unresolved references are encountered the interactive user is made aware but no specifics are given as to which program unit made the reference. If caused by a spelling error the user will have to scan all of his source files to find the reference. In the event of an unresolved reference the module produced is not executable. Apparently this cannot be disabled.

Neither linker carries out checking of argument lists either for mismatch of length or type of variable. Such checking has not been common among linking loaders; there seem to be no real standards in this area. Problems due to such mismatches are generated at run-time and will be discussed in a subsequent section.

5.3 USER LIBRARIES

In the VMS system a "LIBRARY" command produces a user library from a file or files containing one or more object modules. Each object module becomes a separate recognizable entity in the library. Modules may be added, deleted, replaced, and extracted. For easy management a keyword option produces an alphabetized list of modules along with the addition date and time. Library disk space management is carried out by the user by means of other keyword options. User libraries form additional parameters for the "LINK" command.

In the UNIX system the creation and maintenance of user libraries constitutes one type of usage of the file archive facility. Files consisting of single object modules can be added to the archive file and are recognizable thereafter as object modules. Deletion, replacement, and extraction are available in addition to a facility by which the order of modules may be changed. This latter ability is required because in the UNIX linking process each library is searched sequentially only once. If a later module were to reference an earlier module a linking error due to non-resolution of the reference would occur. An archive directory which displays the order of the modules is easily obtained. The date and time of addition is also given.

5.4 FORTRAN PREPROCESSORS

Whereas testing the preprocessors available to UNIX or VMS users is beyond the scope of this report some information was obtained. The UNIX system supports three preprocessors which are front-ended to Fortran via the ordinary compiler command. An optional macro preprocessor may precede either Ratfor (rational Fortran) or EFL (extended Fortran language) processing with the file suffix distinguishing between the two extended language alternatives. To complement this facility a "STRUCT" processor translates Fortran into the Ratfor dialect. With VMS there is no system provided preprocessor although the MORTRAN preprocessor was implemented on the test machine by SLAC.

Some of the uses of a macro preprocessor are available in modern Fortran language concepts. These are the "INCLUDE" statement which incorporates the contents of a file at some point in a source code compilation and the "PARAMETER" statement which allows the user to redefine Fortran constants apart from their occurrence in the code. Both Fortran compilers implement these features which should be most useful in the Fortran development environment.

Chapter 6

PROGRAM EXECUTION

The present section deals with the execution and run-time debugging of programs developed in the UNIX and VMS Fortran environments. Apart from errors in the development of algorithm logic, which are errors of analysis, certain oversights on the part of the programmer continue to cause run-time errors which can be most difficult to find. The permissiveness of the Fortran language is a contributing factor in this regard and thus some compilers implement optional features which aid in the isolation of such problems. The utility of run-time error messages will be reviewed as well as the performance of a system-supplied symbolic debugging system.

Both the UNIX and VMS systems produce executable modules or "process images" as output of the linking process. These images end up as files stored on the default device and using supplied or default naming conventions. In the UNIX system one simply uses the full file name as a command to invoke the execution of a process. In the VMS system the command is "RUN fn" where fn is an unqualified file name having the proper type suffix of "EXE".

Both systems also allow the interactive user to initiate processes which execute in a mode which is considered to be detached from the terminal. That is, the user can subsequently use the terminal to carry out other command tasks. This is similar to the submit feature of VMS which places a process on the batch queue. Batch processes and detached processes are handled differently by the VMS process scheduler.

6.1 RUN-TIME ERROR MESSAGES

As has been described earlier, intentional errors were put into an otherwise correct Fortran code to permit observation of run-time error handling. These included mismatch of argument lists, undefined variables, and the exceeding of array bounds.

In the VMS system, the presence of an undefined variable showed up at run-time as an "arithmetic trap" because the variable was used later as the denominator in a floating-point divide. A traceback was produced which explicitly named the Fortran statement number at which the trap occurred. It was established that VMS initializes variables to zero. A shortened argument list in a subroutine call produced an "access violation" when the subroutine referred to the missing variable. The automatic traceback produced the information by which the error could be

recognized. In a second argument list where a spurious variable was inserted, the message and traceback which was produced referred to an "adjustable array dimension error." Again the statement number of the offending code was given. A double comma in the same argument list led to an additional floating-divide error during execution which was resolved on the basis of the message but which should have been caught as a syntax error at compile-time. The error of exceeding an array dimension gave rise to an "access violation" trap. Recompilation with the bounds checking feature and subsequent execution of the new module produced a trap and the message "subscript out of range."

In the UNIX system identical source code errors produced the following behavior. The presence of the undefined variable caused the program to execute to completion with no error messages. Aside from some printed table headings the output of the program was missing although the correct number of iterations had been performed. Investigation revealed that UNIX had likewise initialized the undefined variable to zero but carrying out arithmetic with it evidently did not cause error exceptions. The shortened execution time implied that some of the code was not being executed. Execution with array space too small for the problem caused "segmentation violation" and "illegal unit number" messages to appear. Rerunning with the compiler's subscript checking feature produced the "subscript out of range" message along with additional data pinpointing the exact problem. Execution with a shortened argument list in a call statement produced the rather cryptic message "bus error." Finally, execution with a spurious variable inserted into a subroutine call argument list caused no error exceptions but the information returned from the subroutine turned out to be incorrect as evidenced by the printed output. Any time a run-time error was printed the additional message "core dumped" was produced. A core image file appeared in the user's default directory also. This image file can be used by the symbolic debugger to further isolate problems. Use of the debugger facility is described in a later section.

6.2 INPUT AND OUTPUT

Both the VMS and UNIX systems default logical units five and six to the terminal for input and output respectively. Fortran 77 and Fortran IV-Plus both support the "READ" and "PRINT" statements which also refer to the default input and output units respectively. In addition, Fortran IV-Plus supports the additional statements "ACCEPT" and "TYPE" for purely terminal oriented applications. These also default to the appropriate device. Both systems also allow the user to equivalence his default input and output units to symbolic file names at execution time using the command language facilities. This is particularly useful for detached process execution. If a file other than the default is used, e.g., logical unit n, then a file named "FORT.n" or "FOR00n.DAT" is opened and/or created in the UNIX and VMS systems respectively. The user need not be concerned about additional file oriented operations such as open, close, disposition, etc., although these facilities are available through Fortran subroutine calls to the IO system.

6.3 SYMBOLIC DEBUGGING SYSTEMS

Both operating systems provide symbolic debugging systems which may be employed optionally to monitor and control the execution of a Fortran program in an interactive mode. One can monitor a process' activity and the values of variables by either name or location. Values of the variables may be changed to resolve run-time conditions. In order to use the facilities one must prepare by compiling and/or linking and having the appropriate debug option keyword or parameter present. Subsequently, the process is executed and the interaction may begin.

In the UNIX system the "SDB" debugger is invoked by the command "SDB fn" where fn names the file containing the process image. The user is prompted for breakpoint commands and subcommands which allow a range of desirable facilities. Subcommands specify what is to be examined during or at a breakpoint and may conveniently be attached to the breakpoint command for uninterrupted tracing. The process can be executed in single-step mode which means line-by-Fortran-line. The debugger system is able to display the line of Fortran code corresponding to the point in the program where execution is stopped. Moreover, an editor-like subsystem allows the user to examine the source file. As described previously, the line numbers of the source code are used by the compiler and therefore the line numbers produced by the symbolic debugger system are in agreement with the editing facility. At the time when the material for this report was being gathered the facility by which a variable's value may be changed did not appear to be working properly. In the case where a process terminates with the message "core dumped" the symbolic debugger can be invoked and will use the core image file in addition to the process image file. Internal pointers will be set to the source line which caused the dump to be produced. Otherwise the pointers will be set to the first executable statement. This capability was not tested.

The VMS symbolic debugging facility is made a part of the process image when the debug keyword is specified to the linker. The user simply executes his process as usual and receives a debugger prompt. The facility supports three kinds of execution interruptions. These are "BREAK-", "TRACE-", and "WATCH-points." The first two are similar capabilities to those found in the UNIX facility. The WATCH-point allows the user to view all changes of a variable's value without specifying the source code location of such changes. It is a most desirable facility but at the time when this material was being gathered it appeared to be defective. The debugging session went into some kind of loop or other unknown state. The user of this debugger may also step through the execution line-by-Fortran-line. A prime disadvantage of the system is that internal statement numbers generated by the Fortran compiler are needed to specify certain interruption points. In the absence of a convenient listing or printing facility one can only scan the compiler listing using an editor. Such usage would be rather inconvenient at best.

Both debugging facilities have defects but still provide an interactive user with a powerful capability. Each facility is relatively easy

to use. Various means are afforded the user to add or delete the debugging code without necessarily backtracking all the way to the compiler. All program modules need not contain the debugging code; those modules so equipped will have the capability of interactive debugging.

Chapter 7

EXECUTION SPEED STATISTICS

This section provides a comparison of the execution speeds of processes run on the two systems. Interest lies not only in the speed of the image which is produced by the Fortran system but also in the speeds of the Fortran compiler and linker themselves. Measurement of speeds is flawed by uncontrollable events relating to the multiprogramming environment. To the extent that was possible, the systems were timed under conditions of decreased system load. On the UNIX system this was achieved by carrying out tests between 6:30 and 8:30 A.M. UNIX local time. With the VMS system the perceived load seemed to never exceed about five users and the effect of their presence appeared minimal.

The UNIX system employed in these tests produces a summary of timing data following the execution of subsystems such as the Fortran compiler and linker and following the execution of user processes. Times included are user cpu time, system time (for IO), and elapsed wall-clock time. The VMS system provides a command by which elapsed wall-clock time may be obtained. In addition, the manager of the VMS system had implemented a pair of system calls by which a user could obtain the elapsed cpu time of his own process. Unfortunately the cpu time of the linker and the compiler is not explicitly available. The elapsed time, however, provides an upper bound by which some inference of the cpu performance of those subsystems might be drawn.

The following sections describe programs, referred to here as A, B, and C, which were employed to obtain data. During the timing experiments of program A the hardware configurations of the two systems were nearly equivalent. The configurations are described elsewhere in this report. When the experiments for programs B and C were carried out, the VAX machine with UNIX software had had a memory increase from .5Mbyte to 1.5Mbyte.

7.1 PROGRAM A

The initial tests of performance were carried out using the program described in Chapter 5, the numerical solution of an ordinary, one-dimensional, boundary-value problem on several equi-spaced grids. Both eigenvalues and eigenvectors were obtained for the resulting tridiagonal matrix problem using the BISECT and TINVIT subroutines from the EISPACK collection. By far the dominant characteristic of the processing is the manipulation of elements of a few one-dimensional arrays. The program required 5.22 cpu seconds on the IBM 3033 of the ANL Central Computing

Facility (CCF) with the workhorse EISPACK routines having been compiled using the Fortran H compiler with optimization level of two. This compiler provides many optimizing facilities and the resulting object code is considered to be highly optimized. The program is cpu bound with only occasional formatted output being produced on an output file.

In Table 1 are shown several cpu and elapsed times for the UNIX and VMS systems. The times are grouped according to whether or not the optimization feature of the respective compiler was used. In summary, the VMS optimized code is fastest being about seven times slower in cpu sec. than the IBM 3033. About 16% was gained by optimizing the VMS code. In contrast the UNIX code is about 9.5 times slower than the IBM 3033 and the gain from optimization is only about 12%. The cpu times comprising this data are relatively constant, exhibiting only small deviations from the averages. On the other hand, the elapsed times tabulated here exhibit large deviations from the average and trends are therefore less certain. Considering just the averages, in the UNIX data is seen an elapsed time roughly twice the cp time whereas the VMS elapsed times are 4/3 of the cp time.

Table 2 contains a short compilation of elapsed times for the compilation and linking of the entire source code. This consisted of the main program, the two EISPACK routines, and a dummy abend routine consisting of only a Fortran stop. On the average, compilation is nearly twice as fast using the VMS system and linking is about three times as fast.

TABLE 1

Comparison of UNIX and VMS execution times when running program A (times in sec.)

	UNIX		VMS	
	Elapsed Time	CPU Time	Elapsed Time	CPU Time
Unoptimized Code	168	56.5	50	42.9
	85	55.8	48	42.6
	75	56.7	67	42.9
	94	55.8	61	43.7
Average	106	56.2	57	43.0
Optimized Code	101	49.4	49	35.1
	121	49.6	69	35.6
	129	50.8	41	34.6
	79	49.3	41	35.0
	77	49.1	39	34.5
	81	48.6	43	35.2
Average	98	49.5	47	35.0

TABLE 2

Comparison of UNIX and VMS compilation and linking elapsed time for program A (times in sec.)

	Compilation		Linking	
	UNIX	VMS	UNIX	VMS
	89	34	26	8
	90	59	27	15
	91	33	27	6
	90	27	34	4
		62		7
		33		14
		29		6
Average	90	40	29	9

7.2 PROGRAM B

The program referred to as B was chosen to exercise the Fortran facilities for a case where doubly-dimensioned array elements dominate the arithmetic. To this end a driver program was written which solves linear equations using the widely known subroutines DECOMP and SOLVE. The problem was constructed artifically and consisted of solving systems of dimension n=10,20,...100 each a total of five times. The matrix solved is defined by

$$A(i,j) = 1 / \cosh(2(i-j)) \qquad |i-j| \leq 20$$
$$A(i,j) = 0.0 \qquad \text{otherwise}$$

and the right hand side in the system AX=b was taken to be b(i)=1.0,i=1,2,...n. The matrix A was computed only one time and its values were kept in a storage array to be used for all of the linear system solve iterations. Although the matrix is symmetric, no use was made of this property and the system was treated as being general. Fortran code for the driver, for DECOMP, and for SOLVE consists of 38, 36, and 28 statements respectively. The double precision (64-bit) version of all routines was used. The programs were likewise run on the IBM 3033 having been compiled using the Fortran H compiler with optimization level two. Execution time required 4.55 cpu seconds. Table 3 contains the results of the VAX-11/780 runs. The cpu time for the UNIX system runs is consistently greater than twice that of the VMS system. The elapsed time on the UNIX system varied slightly; on the VMS system the elapsed time was constant.

TABLE 3

Comparison of UNIX and VMS execution times when running program B (times in sec.)

	UNIX		VMS	
	Elapsed Time	CPU Time	Elapsed Time	CPU Time
	97	92.1	49	45.28
	96	91.9	49	45.35
	97	92.0	49	45.22
	105	93.5	49	45.21
	94	91.8	49	45.17
	144	93.7	49	45.18
Average	105	92.5	49	45.23

7.3 PROGRAM C

The program referred to as C was supplied by Jack Dongarra of the Applied Mathematical Sciences Research Section. It consists of a test driver which is designed to compare several simultaneous linear equation subroutines. Included in the comparison are DECOMP and SOLVE, the LINPACK routines DGEFA and DGESL which use the basic linear algebra subroutines (BLAS), and analogous routines which have the BLAS in-line. The BLAS used by the LINPACK routines are DAXPY, DDOT, and DSCAL. The collection of routines comprises a mixture of array manipulation types since the BLAS serve to vectorize the time consuming operations. The driver program generates matrices A of dimension n=2,5,10,25,50,75,100 and right hand sides b consisting of random numbers. The system of each size greater than or equal to 25 is solved just one time by each of the subroutine groups employed. Dimension 25 is set as a standard to compute the number of solves carried out for the smaller systems. The exact number is adjusted so that as much total time to solve n x n systems ($n<25$) is used as to solve a single system of dimension 25. The adjustment is based on the $O(n^3)$ dependence of solve algorithms and not on computation time. Because the UNIX system did not have a Fortran callable cpu timing routine the data collected for program C is simply for execution of the entire program as has been done for programs A and B. In addition to the cpu and elapsed times, some information bearing on the non-cpu time caused by each system is presented. As before the computations were done using double precision (64-bit) data elements.

To highlight the possible effects of paging strategy in the two systems an additional parameter LDA was varied. This quantity is the dimension of the arrays in the driver program and is known to the subroutines as the "leading dimension of A." It is used to compute the memory location of array elements when a sub-array is being manipulated. As LDA increases, each column of A will span more virtual memory pages and hence the paging rate or IO time should change. This parameter was set both at 100 and 200 for a series of runs. Other than a change in the memory management I/O there is no change in the mathematical manipulations.

The data collected is presented in Table 4. It is interesting to point out the following. On the UNIX system the cpu time remained nearly constant with an increase in LDA. Likewise there would appear to be no significant effect on the system (IO) time or the elapsed time. Increases in system time closely parallel increases in elapsed time and can perhaps be attributed to system activities external to the immediate process. The array space required for this program is largely equal to $8n(n+2.5)$ bytes which for dimension 200 is 324 Kbytes and is well below the bounds of available physical memory. Thus during these timings the paging may not have been a factor. The VMS system data contains the numbers of page faults encountered during process execution. Changing LDA to 200 caused a significant increase in the page fault count. The array space for LDA equal to 100 is about 82 Kbyte which is within physical memory bounds. A page fault count of 211 would seem to not be serious in terms of causing system degradation. The increase of LDA to 200 caused the page fault count to jump to over 5000. Accompanying this

is a parallel increase of the cpu time and the elapsed time on the average by 1.8 and 14 seconds respectively. It would appear that a sizeable increase in paging does not degrade cpu or elapsed time significantly (under the conditions which existed when the data was obtained).

TABLE 4

Program C times (in sec.) and IO data

UNIX (1.5 Mbyte)

LDA=100			LDA=200		
CPU Time	System Time	Elapsed Time	CPU Time	System Time	Elapsed Time
79.5	1.0	83	79.5	1.4	87
79.1	1.4	89	79.7	1.0	85
79.8	2.8	112	79.6	1.5	102
79.8	2.3	101	81.4	5.6	134
80.1	1.0	85	79.2	1.1	84
79.5	0.8	82	79.3	1.2	83

VMS (.5 Mbyte)

LDA=100			LDA=200		
CPU Time	Page Faults	Elapsed Time	CPU Time	Page Faults	Elapsed Time
52.01	242	58	53.78	5086	61
52.29	211	58	54.66	4979	71
51.93	211	58	54.29	5648	76
53.83	211	60	55.31	6444	66
52.32	211	58	53.37	4563	60
54.03	211	64	55.84	5949	104

Appendix A

HARDWARE CONFIGURATIONS

In order to assess this comparison of the UNIX and VMS systems on the VAX-11/780 it is useful to know how the hardware is configured. The systems which were tested were implemented on somewhat different hardware and were operated by different individuals.

The UNIX system hardware consisted of the VAX-11/780 and the optional floating-point accelerator. The memory during the testing of program A consisted of .5 Mbyte which was later increased to 1.5 Mbyte. The testing of programs B and C was carried out in the latter environment. The system has two disks, one DEC RP06 used for files and a second System Industries (SI) disk attached to the UNIBUS and used for paging. The RP06 has 176 Mbyte of space and a peak transfer rate of 806 Kbyte/sec. About 80 Mbytes are available on the SI disk and the peak transfer rate is 1.2 Mbyte/sec,

The VMS system hardware consisted of the VAX-11/780 and the optional floating-point accelerator. The memory was .5 Mbyte and there was one DEC RM03 disk with 67 Mbyte of storage and a peak transfer rate of 1.2 Mbyte/sec.

During the testing both machines were accessed by 300 baud dial-up lines.