THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 57, No. 6, July-August 1978
Printed in U. S. A.

UNIX Time-Sharing System:

No. 4 ESS Diagnostic Environment

By S. P. PEKARICH
(Manuscript received December 2, 1977)

Maintenance and testing of the Voiceband Interface and the Echo Suppressor Terminal in No. 4 ESS are aided by software in the 1A processor. A No. 4 ESS diagnostic environment was needed during the development phase of both the hardware and diagnostic software. The minicomputer chosen to support this environment was also required to support the development of more than one frame at the same time. Because of this requirement and other reasons, the UNIX operating system was selected for software support and development. This paper describes how the UNIX operating system was applied to this project.*

I. INTRODUCTION

Software in the 1A processor is used to maintain and test the Voiceband Interface and the Echo Suppressor Terminal in No. 4 ESS.[1] These testing programs were written in a high-level language oriented to the special requirements of diagnostics in an ESS environment. A No. 4 ESS diagnostic environment was needed during the development phase of both the hardware and diagnostic software. Digital Equipment Corporation's PDP-11/40 minicomputer and the UNIX operating system[2] were chosen to support this environment.

* UNIX is a trademark of Bell Laboratories.

II. VOICEBAND INTERFACE AND ECHO SUPPRESSOR TERMINAL

The Voiceband Interface[3] (VIF) provides an interface between analog transmission systems and the digital time-division switching network of No. 4 ESS.[4,5] A VIF contains up to seven active and one spare Voiceband Interface Units (VIU's). Each VIU terminates 120 four-wire, voice-frequency analog trunks and performs the analog-to-digital and digital-to-analog conversions necessary for interfacing with the time-division network.

The Echo Suppressor Terminal[6] (EST) is inserted (when required) between the VIF and the time division network. Through use of digital speech processing techniques and by operating in the multiplexed DS120 bit stream, the EST achieves about a 10:1 cost reduction over the analog echo suppressor it replaced.

III. MAINTENANCE OF VIF AND EST

Maintenance software for No. 4 ESS[7,8] can be functionally divided into three categories:

(*i*) Detect and recover from software malfunctions.
(*ii*) Detect and recover from hardware faults.
(*iii*) Provide error analysis and diagnostic programs to aid craftspersons in the identification and replacement of faulty modules.

This paper discusses how the UNIX operating system was applied to aid the development of diagnostic programs for VIF and EST.

Figure 1 shows the maintenance communication path between the 1A processor and the VIF and EST. The 1A processor issues maintenance commands to the VIF through maintenance pulse points from a signal processor.[5] The VIF replies to the 1A via the peripheral unit reply bus (PURB). The EST communicates with the 1A through a full peripheral unit bus[5] (PUB). EST commands are issued from the 1A via the peripheral unit write bus (PUWB), and the replies return by way of the PURB.

IV. NO. 4 ESS DIAGNOSTIC ENVIRONMENT UNDER THE UNIX SYSTEM

During the hardware and diagnostic software development phase of both the VIF and EST, a No. 4 ESS diagnostic environment had to

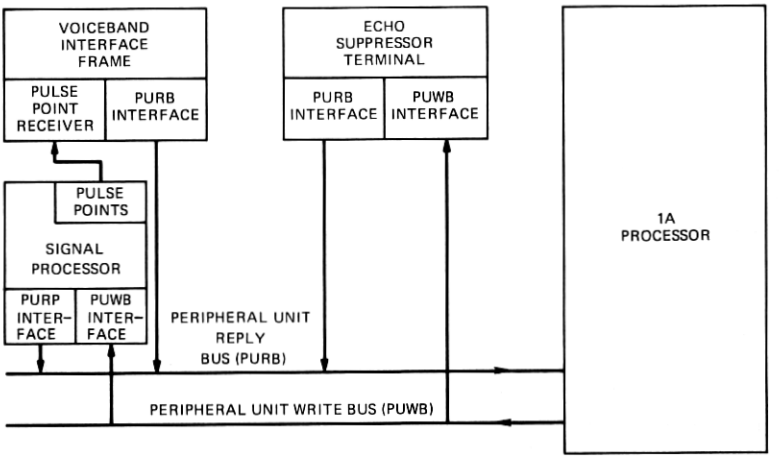

Fig. 1—Communication path between 1A Processor and VIF and EST.

be supported. Since a 1A processor was not available for the development of VIF and EST, a minicomputer was used to simulate the diagnostic functions. Figure 2 shows the No. 4 ESS diagnostic support environment which was created. Special hardware units

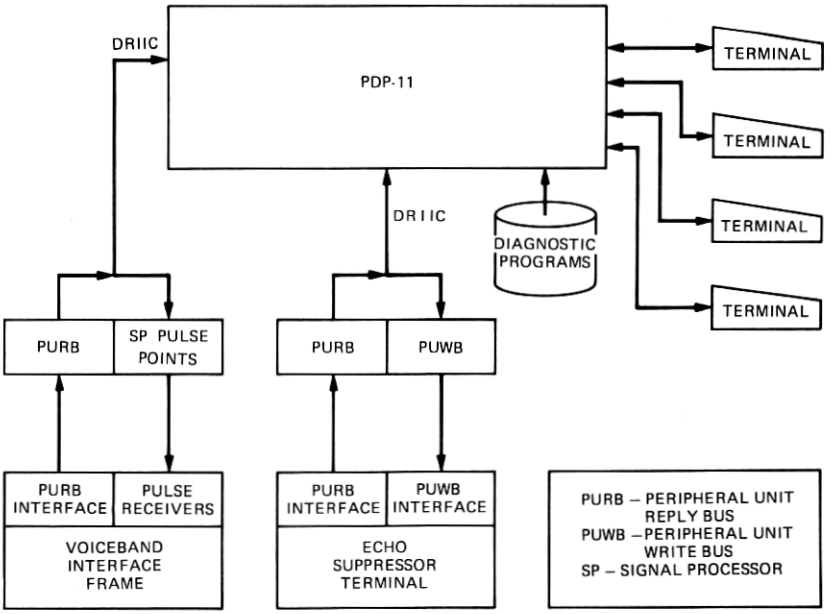

Fig. 2—No. 4 ESS diagnostic support environment.

were developed to provide electrically compatible interfaces representing the peripheral unit bus and the signal processor maintenance pulse points. These units are controlled by the minicomputer through standard computer interfaces. The minicomputer then performs the diagnostic functions of the 1A processor in a No. 4 office. By issuing commands to the special interface units, the minicomputer simulates the 1A processor's transmission of diagnostic instructions to the individual frames. The majority of the diagnostic software for VIF and EST was developed in this diagnostic environment.

Software development began under the disk operating system (DOS), supplied by the vendor. DOS is a "single-user" system; that is, only one software designer can use the machine at any given time. This limitation was acceptable early in the project when all the computer time was dedicated for software development. However, as support software became available, the hardware and diagnostic software test designers became heavy users of the system.

At the start of the project, it was realized that the minicomputer system was required to support more than one frame. In addition, support software development effort was still continuing, which now presented a problem in scheduling the minicomputer system. Two alternate solutions were considered. The first was to purchase another minicomputer to support the development effort on the second frame. A disadvantage of this proposal was that one of the minicomputer systems still had the scheduling problem with support software development and with supporting the frame. Also, supporting additional frames would cause the problem to arise again. The second alternative was to upgrade the minicomputer system so that it could support time-shared operations. This seemed a more economical way of supporting additional frames and support software development. The second alternative was chosen.

Two time-sharing systems were available for the PDP-11 computer, UNIX and RSX-11. The RSX-11 system was basically the single-user DOS, upgraded to support multiple users. Its main advantage was its upward compatibility with programs developed under DOS. The UNIX operating system, on the other hand, offered a better development environment and more support software tools than DOS. The C language was also available, which presented a very attractive alternative for developing new software. These advantages outweighed the disadvantage of having to modify the existing software developed under DOS. Therefore, the UNIX operating system was selected to support this project.

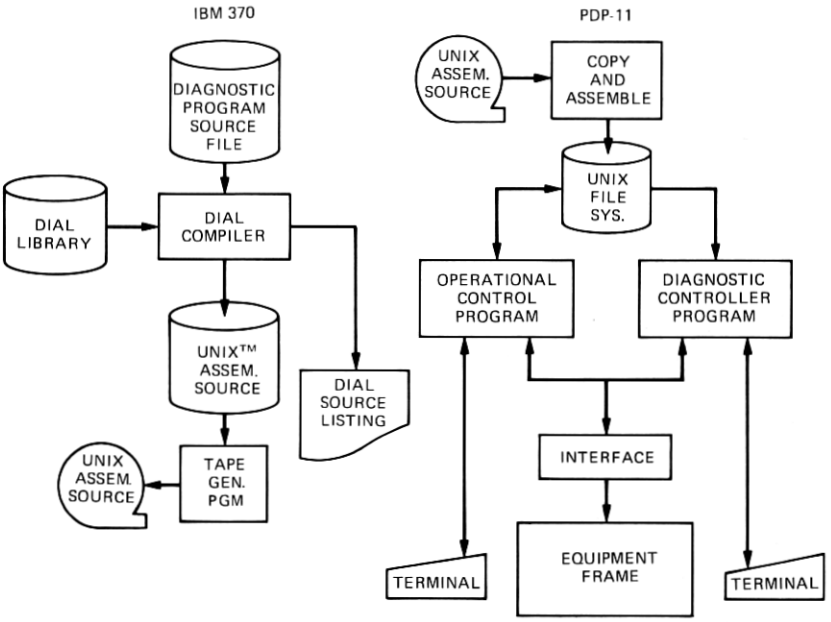

Fig. 3—Software for No. 4 ESS diagnostic environment.

V. THE SOFTWARE SYSTEM

Figure 3 is a block diagram of the overall software system used to support the No. 4 ESS diagnostic environment. It consists of an off-line compiler for a special diagnostic programming language (described below), a run-time monitor for execution and debugging of diagnostic programs, an on-line compiler for hardware and software debugging aids (denoted as Operational Control), and support programs for creating and maintaining the diagnostic data base. The entire software system on the minicomputer was named PADS (*P*DP-11 *A*ided *D*iagnostic *S*imulator).

5.1 DIAL compiler

DIAL (*Dia*gnostic *L*anguage)[8] is a high-level programming language that allows a test designer to write a sequence of test instructions with data using macro calls. The language was developed to meet the special requirements of diagnostics in the ESS environment. DIAL statements can be divided into two classes: testing statements and general purpose statements. Testing statements

are used to issue maintenance commands to a specified peripheral unit in a No. 4 ESS office. The general purpose statements are similar to most other high-level languages. They manipulate data, do arithmetic and logical functions, and control the program execution flow. The diagnostic programs for both the VIF and EST were written in DIAL.

Several DIAL compilers are available to diagnostic program designers. Each compiler produces code for a different application. The compilers of interest to the VIF and EST test designers are the DIAL/ESS and the DIAL/PADS compiler. The DIAL/ESS compiler, developed using the TSS SWAP (*Sw*itching *A*ssembler *P*rogram),[9] produces a data table which is interpreted by a diagnostic control program[8] in the No. 4 ESS office. The DIAL/PADS compiler, developed using VMSWAP (*V*irtual *M*emory SWAP), produces code for a PDP-11. This compiler, which runs on the IBM 370 computer, produces PDP-11 assembly language source code which is acceptable to the UNIX assembler. The assembly language code is subsequently transported to the UNIX file system where it is assembled. The resultant object modules are usable with the run-time monitor in PADS.

Consideration was given to implementing the DIAL/PADS compiler directly on the UNIX system. This would have eliminated the need for the IBM 370 computer in the diagnostic development effort. All software development could have been performed on the UNIX system. However, because of the lack of the necessary staff and computer resources, this approach was abandoned.

5.2 No. 4 ESS run-time environment

The PADS system allows the test designer to execute a DIAL program with the aid of a run-time monitor and debugging software package called DCON (*D*iagnostic *Con*troller). The debugging package in DCON provides a tool for evaluating diagnostic algorithms and debugging DIAL programs. DCON facilities allow the user to:

(*i*) Trace the execution of DIAL statements.
(*ii*) Pause before execution of each DIAL statement.
(*iii*) Pause at DIAL statements selected at run time.
(*iv*) Display and modify simulated 1A memory during a pause.
(*v*) Start execution of the DIAL program at any DIAL statement.
(*vi*) Skip over selected DIAL statements at run time.
(*vii*) Loop one or a group of DIAL statements at run time.

This last feature was especially useful for hardware troubleshooting. Using this debugging package, the test designer can follow the execution of a DIAL program while it is diagnosing the VIF or EST.

DIAL programs compiled by the DIAL/PADS compiler communicate data and status information to the diagnostic controller program. Under DOS, this communication link was established at run-time via the PDP-11 "trap" instruction. After switching to the UNIX operating system, the "emulator trap" (**emt**) instruction was used. The DCON process used the **signal()** system call to catch the **emt** instruction executed by the DIAL program. However, because of the large number of **emt** instructions executed in the DIAL program and the operating system overhead to catch and handle this signal, this method of dynamic linking between DCON and DIAL programs had to be abandoned. It was replaced by the jump subroutine instruction and loading a register with an address at run-time.

Maintenance instructions are sent to the VIF and the EST through general purpose I/O ports (DR11Cs) on the minicomputer. Originally, DCON relayed the maintenance instructions of DIAL programs using the **read()** and **write()** system services of the UNIX operating system. Before executing a DIAL program, DCON opened the appropriate files (general purpose I/O ports) and retained their file descriptors. All subsequent maintenance instructions requested by the DIAL program were handled by DCON as **read()** or **write()** requests to the file. Measurement of the I/O activity on these ports revealed that the VIF diagnostic program sent a large number of maintenance instructions. Hence, a large portion of the diagnostic run time was operating system overhead. Based on this observation, special system service routines were added to the UNIX operating system. These routines directly read and write the general purpose I/O ports. After implementing the I/O for maintenance instructions in this manner, the run time for the VIF diagnostic was reduced by more than half.

The PADS system simulates the automatic running of diagnostics as performed in a No. 4 ESS office. A complete diagnostic for a peripheral unit is normally written in many functional blocks called phases. Each phase is a self-contained diagnostic program designed to diagnose a small functional part of the unit. The phases of the diagnostic program are stored as load modules in the UNIX file system. The PADS system automatically searches the directory containing the diagnostic phases for the peripheral unit. Each phase is loaded into memory by PADS and execution control is given to it. At the termination of each phase, control is returned to the run-

time monitor program which will determine the next phase to be executed. The diagnostic phases for different frames are stored in separate subdirectories. The files are named using a predefined naming convention. This allows DCON to automatically generate the file name for the next diagnostic phase to be loaded and executed.

5.3 Support programs

The output from the DIAL/PADS compiler is UNIX assembler source code. The DIAL assembly source code is placed on a magnetic tape for transport to the PDP-11 system. A shell procedure on the UNIX system reads the tape and invokes the UNIX assembler. The output from the assembler is then renamed to the file name supplied by the user in the shell argument.

The UNIX shell program[10] is also used to interpret shell procedure files which simulate the automatic scheduling of diagnostics in the central office. These procedure files call special programs which monitor the equipment for a fault indication. After a fault is detected, the shell procedure calls in the DCON program to run the frame diagnostics.

5.4 On-line diagnostic aids

A software package known as "Operational Control" was developed under the UNIX system using the C compiler.[11] This program allows the user to issue operational commands to the frame by typing in programming statements at his terminal. These statements are compiled directly into PDP-11 machine code and may be executed immediately. Test sequences may be developed directly on-line with the frames. These programs can then be saved in files for future usage. This last feature was extremely easy to implement in the UNIX operating system. By altering the file descriptor from standard input or output to a file, common code reads and writes the program from the terminal or a file.

The parser for the Operational Control language is recursive. This type of parser was exceptionally easy to implement in C since the language allows recursive programming. The management of storage variables needed by the parser in recursive functions was automatically performed by the C compiler. This would have been a horrendous bookkeeping operation if a nonrecursive programming language had been used. The block structuring features of C made it easy to implement the parser from the syntax definition of

Operational Control. Using C, the on-line compiler was defined, implemented, and operational within a short time period.

VI. UNIX SYSTEM SUPPORT

The UNIX time-sharing system enables the minicomputer system to support more than one frame at a time. Each frame has a dedicated, general-purpose I/O port from the computer. The user supplies the frame selection information to the controlling program (either DCON or Operational Control). Then the software opens the appropriate I/O port for future communications with the frame.

Another feature provided by PADS is to allow the diagnostic debugging program to escape to the on-line compiler program. This is done in a manner such that when the user exits the on-line compiler, he immediately returns to DCON at the state at which it was left. This feature was very easy to implement under the UNIX operating system. By using the **fork()** and **exec()** system calls, the parent program (DCON) sleeps, waiting for the death of its child process (Operational Control). When the user exits from Operational Control, DCON is awakened and will continue its execution from the last state it was left in. The concept of placing one process to sleep and invoking another was not available on RSX-11.

VII. SUMMARY

The UNIX time-sharing system had many advantages over the RSX-11 system that was considered for the PADS system. Originally, PADS was developed under Digital Equipment Corporation's single-user Disk Operating System (DOS) using the macro assembler MACRO-11. When it became apparent that a second frame had to be supported, a time-sharing system was considered. At that time only two time-sharing systems were available for the PDP-11 computer, UNIX and RSX-11.

DEC's RSX-11 system is a disk operating system which supports multiple users. The basic advantage of RSX-11 was its upward compatibility with programs written to run under the disk operating system. However, this advantage was outweighed by the advantages the UNIX operating system presented.

In our opinion, the UNIX operating system provided a much better software development environment for our purpose than RSX-11. In particular, the C language is much better suited to systems programming than a macro assembler or Fortran. Also, many excellent

software tools such as the text editor and the linker existed on the UNIX system. For example, it was felt that with the aid of the text editor, all the programs that were written in MACRO-11 assembly language could easily be translated into the UNIX assembler language. This allowed the existing software to come up under the UNIX system in a short time period. Then, as time allowed, the existing software could be rewritten in C. All future software was slated to be written in C. The need to rewrite the software gave an opportunity to rethink the entire software project, this time with some experience. In the end, this led to vast improvements in the software packages.

REFERENCES

1. "1A Processor," B.S.T.J., *56* (February 1977).
2. D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," B.S.T.J., this issue, pp. 1905-1929.
3. J. F. Boyle, et al., "No. 4 ESS: Transmission/Switching Interfaces and Toll Terminal Equipment," B.S.T.J., *56* (September 1977), pp. 1057-1098.
4. "No. 4 ESS," B.S.T.J., *56* (September 1977).
5. J. H. Huttenhoff, et al., "No. 4 ESS: Peripheral System," B.S.T.J., *56* (September 1977), pp. 1029-1056.
6. R. C. Drechsler, "Echo Suppressor Terminal for No. 4 ESS," Intl. Conf. on Communications, *3* (June 1976).
7. P. W. Bowman, et al., "1A Processor: Maintenance Software," B.S.T.J., *56* (February 1977), pp. 255-289.
8. M. N. Meyers, et al., "No. 4 ESS: Maintenance Software," B.S.T.J., *56* (September 1977), pp. 1139-1168.
9. M. E. Barton, et al., "Service Programs," B.S.T.J., *48* (October 1969), pp. 2865-2896.
10. S. R. Bourne, "UNIX Time-Sharing System: The UNIX Shell," B.S.T.J., this issue, pp. 1971-1990.
11. 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.