Requirements for minicomputer operating systems for human experimentation and an implementation on a 4K PDP-8 computer*

WALTER SCHNEIDER†

Indiana University, Bloomington, Indiana 47401

and

KARL W. SCHOLZ

University of Illinois, Champaign, Illinois 61820

A programming system is described which considerably simplifies preparation and running of experiments on PDP-8 computers.

The Department of Psychology at Indiana University has been using computers in psychological experimentation since 1965. In 1970, the second author, then at Indiana University, wrote an operating system for laboratory experimentation for PDP-8L computers. During the following 2 years of use of that and other systems, we have developed some important guidelines concerning computerized experimental control. On the basis of that experience, the operating system for the PDP-8 was recently rewritten.

We have developed a programming system which considerably simplifies preparation and running of experiments on PDP-8 computers. In this paper, we intend to (a) discuss software requirements for minicomputer experimentation, (b) describe the system we derived for PDP-8 machines, (c) illustrate the use of that system, (d) describe how to implement the system on your own PDP-8, and (e) add words of caution on the use and programming of a minicomputer.

SOFTWARE REQUIREMENTS FOR MINICOMPUTER EXPERIMENTATION

Standard compilers generally do not generate code which can meet the special requirements of experimental control. Special compilers which generate code to run experiments must meet three major requirements.

First, the compiler must be versatile enough to generate code to meet the E's needs. The compiler should provide the capacity for at least simple computation so that experimental contingencies can be programmed. To facilitate simple user control of external devices, special statements which control all the

devices connected to the computer should be provided. Commands for versatile sequencing through the program as well as conditional and probabilistic branching should be supported.

Second, the language of the compiler must be easy to learn and to use. The language should be constructed in such a way that a person familiar with computers could learn to use it in about a week. Further, error checking should be provided by the compiler, because a compiler which indicates those statements which are in error at the time of compilation will save the programmer hours in debugging. Finally, the compiler should do as much of the programming work as possible. For example, a compiler which allows one to specify latency collection by means of a single statement vs one requiring the user to store pointers and control the clock is much easier to use. The easier the compiler is to use, the better the chance of getting an experimental program operating in a day.

A third requirement is that the compiler must create efficient code. The compiler and the operating system should allow the user the ability to create and execute an arbitrarily complex process program with as little as 4K words of memory.

MINICOMPUTER OPERATING SYSTEM REQUIREMENTS

The code produced by the compiler is hosted by (run under control of) an operating system. It is the operating system that provides the special-purpose routines which perform most of the functions the user has programmed. The operating system must handle simultaneous I/O operations between peripheral devices and the computer. Few nonsophisticated computer users understand interrupt servicing and data transfer to peripheral devices from the computer. In order to eliminate the need for the user to learn and understand interrupt servicing and data transfer, the system should provide routines which the user can call. The system operating routines should allow the user to have several I/O operations occur simultaneously. Simultaneous I/O handling allows the user to input and output data for his program without having to pause during the ongoing operation of his experiment.

The operating system must maintain all necessary system clocks and timers. Most experiments consist of a sequence of events scheduled by a series of delays. The time base of the system (how often the clock is updated) determines the precision with which those delays can be controlled. Doorne and Sanders (1968) stated that the

*This research was supported in part by Indiana University Faculty Research Account 2631466 and by PHS MH 11219-03 and MH 12717-06.

†Requests for reprints or further information on HEPS should be addressed to the first author at Psychology Department, Indiana University, Bloomington, Indiana 47401. Operating system source and object tapes and user's manuals are available. Requests for information should include a description of the configuration of the target system.

presentation time of and between particular stimuli should be controllable to an accuracy of 1 msec; however, a millisecond clock could consume as much as 30% of the computer's time. Fortunately for most human experimental work, 100-msec interval timing is adequate. Hence, the system should count in 10ths, or if necessary 100ths, of a second. To allow the user to synchronize the start of interval timing exactly with events external to the computer, some method of resetting the clocks should be provided. To provide latency measures, the system routines must read the contents of the hardware clock, convert elapsed time into milliseconds, and pass the information to the user's program. Since human experimentation often requires simultaneous collection of several response latencies with millisecond accuracy, latency collection routines should allow the clock to be read several times so that several latencies can be collected during the same time period. Finally, the system should allow recording of either time of day or accumulated time from the start of the experiment.

The operating system should minimize the likelihood that the user can create a problem situation which he cannot easily diagnose and correct. In order to aid the user in the diagnosis of his problems, subroutines should check parameters to determine if those parameter values are reasonable. To facilitate debugging of the users' program, the system should allow the user to step through or execute portions of his program. Pointer manipulation should be handled in a way that prevents the user from deadlocking (entangling the pointers) in the system.

An operating system should decrease the chance of the user writing an experiment in which the data collected is in error without the user becoming aware of it. An example of this would be that of the user inadvertently resetting the system clock while collecting a latency. Whenever possible, the operating system should be programmed to detect such errors.

Finally, the operating system should be constructed in a modular fashion to simplify modification. There should be space available so that the sophisticated user can write and add subroutines to the operating system or modify and replace existing routines.

ANALYSIS SYSTEM

Once an experiment is completed, the user must analyze his data. Minicomputers with their small word size make calculations of other than trivial summary statistics difficult. The minicomputer experimental software should generate data tapes which can be read by standard minicomputer programming systems (e.g., FORTRAN or BASIC). This will enable the user to write programs to analyze the means, totals, and variances of the conditions of his experiment. For complex analysis, minicomputer software should provide some means for the transfer of data from the minicomputer to a large central computer facility.

Human Experimental Programming System

At this point, the Human Experimental Programming System (HEPS) developed for a PDP-8 4K computer will be described. The system (Fig. 1 consists of a 1K FORTRAN compiler, DEC's Octal Debugging Technique (ODT) program, and an interrupt servicing operating system. Figure 1 illustrates the flow of information between various I/O devices and the operating system. Teletype input is checked by the keyboard monitor before being passed to the appropriate program. When specific control characters are typed on the system Teletype, the keyboard monitor initiates one of the three programs in core. The FORTRAN compiler generates machine code commands, which are stored in the FORTRAN program execution area. The FORTRAN run-time monitor controls execution of ongoing experiments. To facilitate debugging of programs, the user can transfer control to ODT. Once under control of ODT, the user can execute sections of his program and display or change any portion of core.

The HEPS compiler is an interactive resident compiler which allows the user great versatility in a syntax similar to FORTRAN. The compiler was made resident in order to free the user of the tedious sequence of operations consisting of having to read in the editor, correct and punch a new source program, read in the compiler, read in the source program, punch the object program, and finally read in the operating system and source program code and execute the program. In order to correct an error in the HEPS system, the user merely reenters the compiler and types the corrected statement.

The compiler accepts statements of the form <label, command, parameter, and comment>. For example, one might code

```
100 IF(J) 105, 110, 130'COMMENT.
```

The statement label "100," is not merely an identifier, but also indicates the hard core location of the statement. The "IF" indicates the type of operation to be performed, i.e., the calling of the IF subroutine at execution time. The J, 105, 110, and 130 are parameters of the IF subroutine which executes the intended conditional branch. This standard form for all statements allows a very small (about 1K) compiler to generate the appropriate code. The compiler supports READ, WRITE, FORMAT, IF, DO, GO TO, ARITHMETIC (for one-word integer addition, subtraction, division, and multiplication), SET (constant initialization), O (octal or machine code constants), KONTINUE (continue), XIT (idle until interrupted), and call statements. Basically, these statements compile into a jump and store address to the appropriate subroutine. The compiler checks each statement for errors before generating the code and storing it in the execution area.

The compiler meets most of the requirements outlined earlier. It is very versatile: Special subroutines

Fig. 1. Flow of information between various I/O devices and the operating system.

for interval timing, latency collection, random number generation, array permutation, array indexing, double-word addition and division, and I/O handling for a system Teletype, peripheral Teletype, high-speed reader, and a digital interface are included. Also, the user can write and include machine language subroutines and/or insert machine code in his main program.

The compiler language is easy to learn and easy to code. Since the language appears very similar to FORTRAN, a person familiar with FORTRAN could learn to use the system within a week.

The generated code is quite efficient. Since the user's program consists basically of subroutine calls, 200 FORTRAN-type statements can fit in the 976 words available for the execution program. The modular construction of the compiler and the system allows the user to utilize unneeded subroutine core areas for data storage.

The system allows easy debugging. First, ODT allows the user to easily examine any portion of core and execute any subset of his program. Second, since the compiler is core resident, the user can change or add instructions and reexecute without recompiling the entire program.

A major weakness of the system is that it does not fully check parameters of subroutine calls. By the incorrect specification of parameters in a subroutine call, the user could purge the operating system.

The operating system (see Fig. 2) is interrupt driven and can handle input and output simultaneously without requiring a pause in experimental execution. All interrupts cause a branch to the interrupt servicing routine which determines which device caused the interrupt and transfers control to the routine to input or output one word of data or update the clock. Control is never returned to the user on an interrupt. The user's program is queued for execution and is not actually executed until all interrupt requests are serviced.

The system clock software maintains three 100-msec interval timers, a millisecond latency timer, and a 24-h clock. The clock can be synchronized to events external to the computer. Since the clock interrupts the CPU only once every 100 msec, it consumes less than 0.5% of the execution time. The interrupt servicing software uses a first-in-first-out communication queue which minimizes the chance of having system pointers confused (see Scholz, 1972). The independence of operating system components allows the elimination of sections of it when additional core is needed. Finally, the system allows easy initiating, aborting, and temporary halting of the compiler, experimental execution, and ODT program.

Data are analyzed in the system by taking program-generated data tapes and running them through DEC's 4K FOCAL or FORTRAN, or transferring the data to cards for analysis on a large computer.

Fig. 2. Operating system.

Fig. 3. Flow chart for experiment cited in text.

An Example

As an example of the use of the system, the familiar Sternberg experiment (1966) will be presented. This paradigm consists of S's learning a set of one to four digits, which are referred to as S's positive set. Then, on each trial of the experiment, S is presented a digit and required to make a response indicating whether the presented digit is a member of his positive set. The S's reaction time and response are recorded on each trial. The flow chart for this experiment is given in Fig. 3, and the code for the outlined section of the flow chart is given in Table 1. The statements are an example of the type of control the user has of the computer with his higher-level language.

Should the experiment not operate properly, the user can utilize ODT to facilitate debugging of the program. Under control of ODT, the user executes the program statement by statement until he determines the error. Thereafter, the user retypes the incorrect statement and attempts to execute the program again.

During the running of the experiment, the operating system allows simple initiation and termination of the user's program. To start the experiment, the user types CTRL "E," to abort the experiment CTRL "A." If the experiment must be temporarily halted, the user types CTRL "H."

Installing the System

The installation of the system is rather straightforward. After obtaining the appropriate materials and reading the implementation and users manual, the user would load the system and execute the ODT program. Under the control of ODT and by using the implementation manual, the system would be modified to operate with the installation's clock, input/output devices, and core configuration. Then a new system tape would be punched. Documentation and source listings are available so the user may add subroutines for special data manipulation and/or I/O control.

WORDS OF CAUTION

At this point, we would like to add a few words of caution about using a minicomputer. First, do not trust a mini, especially when collecting latencies. We

Table 1

```

210 CALL 86(40)'GET UNIFORM RANDOM NU. FROM 0-2047
220 40=40-10'SUBTRACT 1024 FROM IT
230 IF(40)260,260,240'BRANCH PROB. .5 TO 260
240 A RSET=ZERO'SET RESP. INDICATOR TO POS SET
245 CALL 91(5,1)'PERMUTE POSITIVE SET
250 A STIM=1'STORE POS. SET STIM TO BE PRESENTED
255 GO TO 280
260 A RSET=15'SET RESPONSE INDICATOR TO NEG. SET
265 CALL 91(5,5)'PERMUTE NEG. SET
270 A STIM=10'STORE NEG. SET STIM TO BE PRESENTED
280 WRITE 13,18'PRINT A LINE FEED TO WARN SUBJECT
285 CALL 78(300,55)'WAIT .5 SECONDS
288 XIT'WAIT
290 WRITE 13,STIM'PRESENT THE STIMULUS
305 CALL 78(400,200)'SET WAIT FOR RESPONSE UP 2 SEC.
310 CALL 95'INITIALIZE LATENCY COLLECTION INTERVAL
320 READ 12,14'WAIT FOR INPUT
330 CALL 96(X,X)'GET LATENCY
335 CALL 78(0,0)'TURN OFF 2 SEC. WAIT FOR RESP.

```

recommend that the user connect an external timer as a S and run the timer in the experiment to verify that the equipment and software are working properly. To illustrate the importance of such checking, we originally had a software bug which added ± 40 msec to a certain proportion of our latencies. This bug could not be detected by examination of the data, and only after setting up the timer could we determine the existence and cause of the bug. As another example, this past summer our crystal clock (purchased from a very popular computer company) began intermittently running 20% faster than the specified rate. Had the intermittent effect not be identified and had it occurred more in one condition of an experiment, it could have caused confounding in the data. Finally, since minicomputers have severe roundoff problems, one should always run through a set of data for which the results are known before analyzing other data with the program.

In summary, we have laid out what, after 2 years of using minicomputers, we feel are the important software requirements and have described the system which we designed in the spirit of these requirements.

REFERENCES

- Doorne, H. V., & Sanders, A. F. PSARP, a programmable signal and response processor. *Behavior Research Methods & Instrumentation*, 1968, 1, 29-32.
- Scholz, K. W. Computerized process control in behavioral science research. *Behavior Research Methods & Instrumentation*, 1972, 4, 203-208.
- Sternberg, S. High speed scanning in human memory. *Science*, 1966, 153, 652-654.