Zoroaster: A multiprogramming system for psychological research

J. L. LEWIS, S. J. BOIES, and G. W. OSGOOD
University of Oregon, Eugene, Oregon 97403

The development of a multiprogramming system for psychological research was undertaken on a PDP-9 computer with 8k memory. The needs of the users and the time demands on the computer raised several important questions: (1) Can a practical timesharing system be developed with only 8k memory? (2) Can critical timing functions be maintained? (3) Can easy access be provided (e.g., Fortran) to all experimental devices? (4) Can a system be designed which will take advantage of and be compatible with most of the standard DEC-provided software (e.g., Fortran compiler, loader)? The system was successfully developed by using a rapid within-core swapping technique, standard I/O routines, many general-purpose handlers and subroutines, and fully utilizing available core.

INTRODUCTION AND BACKGROUND

The computer-controlled Cognitive Studies Laboratory at the University of Oregon is built around a PDP-9 computer with 8k core memory. The system configuration includes two Teletypes, two Dectapes, and paper-tape I/O. In addition, an analog-digital converter, oscilloscopes, a variety of displays, keyboards, and other psychological apparatus are interfaced to the machine.

The experimental methods in our lab involve the use of a wide variety of audio and visual displays. For example, an experiment examining the effects of divided attention might require the S to perform two tasks at once. The primary task might be the presentation of a warning signal (e.g., a "+") on the scope followed ½ sec later by a letter (e.g., capital "A"). A second letter (e.g., small "a"), rotated 90 deg, might be presented 1 sec later, with the S required to press one key to indicate that the two letters had the same name. A secondary task might be the presentation of a tone at specified times during the primary task, with the S required to press a key with his foot to shut it off. In an experiment of this type, the warning interval and the interstimulus intervals require accurate timing. In addition, the major dependent variable, the S's reaction time, requires accurate timing. Secondary information such as electrophysiological data might also be collected, using an analog-digital converter. Finally, feedback information in the form of reaction time and accuracy might be displayed on the oscilloscope.

One of the major motivating factors in designing a multiprogramming system was the constant demand on the computer time. Although a large number of Es wanted to use the computer to control experiments, the typical user had had little experience with computers and had minimal programming skills.

The possibility of implementing a timesharing system raised several important questions: (1) Can a practical timesharing system be developed with only 8k memory? (2) Can critical timing functions be maintained? (3) Can easy access be provided (e.g., Fortran) to all experimental devices? (4) Can a system be designed which will take advantage of and be compatible with most of the standard DEC-provided software (e.g., Fortran compiler, loader)?

SYSTEM DESIGN

Zoroaster, our multiprogramming system, provides an affirmative answer to all of the above questions. The system was designed to perform a rapid swap between two core resident experimental programs, one of which could be written in Fortran. (It would be necessary to modify the standard Fortran object-time system to be reentrant to permit both user programs to be Fortran.) The design philosophy has included the provision of a large number of small timesharing handlers and standard subroutines which can take over all standard operations and fully utilize all of the available core space.

RAPID SWAPPING

The basis for the timesharing system is an "egg timer" which causes a program interrupt every 2 msec and TSM, a timesharing monitor and interrupt handler. The system is loaded and started with the standard loader by use of a dummy main program which does nothing but provide TSM with the starting addresses of the two programs, then transfers control to TSM. TSM does the setups and initialization of devices. It also keeps track of the necessary information to restore control to each program and to determine which program has control. Every 2 msec TSM swaps control from one program to the other. This provides each program with 2 msec execution time out of every 4 msec. This kind of approach was used to permit accurate timing and to provide maximal use of computing time. We are able to provide both users with 2-msec accuracy for timing intervals and ½-sec accuracy for timing external events, such as reaction time. Interrupt handling is guaranteed within 100 msec. Although we do not have a disk system, it should be emphasized that we would not have attempted disk swapping even if we had a disk.

STANDARD I/O

I/O during timesharing is handled by two system programs, TSIO and TSDTIO. TSIO is a timesharing routine which handles numeric (integer) or alphanumeric (ASCII) I/O for the Teletypes, reader, punch, and oscilloscopes. Input is assumed to be ASCII and is either packed in 5/7 ASCII if alphanumeric transfer is requested or converted to binary if integer is indicated. Likewise, output is always in ASCII, being converted as necessary. Access to TSIO in Fortran is via slightly modified READ and WRITE statements and in MACRO-9 by a JMS* TSIO passing up to four arguments: word address, word count, .DAT slot number, and integer field width. Writing on the oscilloscope is accomplished exactly as if writing on the Teletype except that the .DAT slot refers TSIO to a small oscilloscope handler. TSIO has the disadvantage of its restricted use (rather like having only A, H, and I formats in Fortran), but it has the advantage of being 2,100 octal words shorter than BCDIO, which it replaces, and it supports two programs.

TSDTIO is a block-oriented, time-shared, dynamically buffered, asynchronous write, error-handling, input-output, binary Dectape handler. TSM assigns Program 1 to DT1 and Program 2 to DT2. Reading is accomplished in Fortran by CALL SEEKS (IBLK), where IBLK is the Dectape block number where reading is to begin. Entry at SEEKS causes TSDTIO to request buffer space from SPACEL, a dynamic space-allocation program (unless buffer space exists from a previous call), and to transfer IBLK into that buffer. Any legal binary READ statement may follow. The READ causes appropriate transfer from the buffer to the user's program. If the READ (or subsequent READs) exhausts the Dectape buffer, the next block is automatically read in.

Table 1
Program Segment Illustrating Use of TIMEL and KEYBRD

```
1       CALL LETTER (LET1)
2       MSEC = MSEC
3       CALL TIMEL
100     CALL LETTER (LET2)
4       CALL RESP (MASK,ITIME,
        IBIT,JTIME)
```

Writing is accomplished by the sequence CALL ENTERS (JBLK), where JBLK is the block number where writing is to begin followed by any number of standard WRITE statements, then CALL CLOSES (KBLK), where KBLK is the address where the next free block is to be returned. The basic strategy is to fill a buffer (requested by SPACEL) with WRITE requests. When the write buffer is full, its address is put in a queue, and a second buffer is requested and used. As the Dectape hardware is freed, the contents of the queued buffer is asynchronously output and the buffer area is released for subsequent (or other) use. This method eliminates the need for either user to wait for the Dectape hardware. The call to CLOSES causes a partially full buffer to be output. The dynamic buffering (SPACEL) optimizes the use of the previously inaccessible space over the loader and unused handlers. TSDTIO replaces BINIO, FILE, FIOPS, and DTA. It of course does not provide all of the functions of these programs, but it does have the advantage of requiring only 632 octal words plus buffer space. SPACEL is shared by other system and user programs and occupies only 102 octal words.

SPECIAL HANDLERS

In addition to our special I/O packages, many special-purpose handlers and subroutines are available to permit handling of routine functions. Examples of these are TIMEL and KEYBRD. TIMEL is a shared timing routine which provides concurrent timing of up to 10 intervals. Request for a timed interval (e.g., between the presentation of two letters) is initiated in FORTRAN in Table 1.

LETTER is assumed to be a subroutine used to control the display of a single letter coded by its argument. MSEC contains the value of the interval in milliseconds. The order of execution of the statements is represented by their numbers. Statement 1 causes the first letter to be displayed, 2 loads the accumulator (AC) with the value of MSEC, and 3 transfers control to TIMEL. TIMEL takes the value of the AC to set up a timing interval, saves the address of Statement 100, and returns control to Statement 4. At the end of the interval, a program interrupt transfers control to the handler portion of TIMEL, which transfers control to Statement 100. The return from LETTER is to TIMEL, which restores control to the instruction following the last one executed at the time of the timing interrupt. In addition to the 10 concurrent intervals, the handler portion of TIMEL can accommodate five levels of subroutine transfer. TIMEL permits a very powerful method of asynchronous software control (see Table 2).

KEYBRD is another standard subroutine which is used to free the main program from timing and recording reaction times. In addition to displaying the two letters at specified intervals, the request to begin timing is illustrated in Statement 4 of Table 1. The CALL RESP (MASK, ITEME, KEY, JTIME) causes the Ss' reaction times (from display of the first letter until a key is pressed) to be recorded. MASK indicates which keys on a standard keyboard are to be activated. The latency (in milliseconds) from the CALL until the keypress is put in ITIME. Similarly, the latency from the CALL until the key is subsequently lifted is put in JTIME. A code indicating which key was pressed is put into KEY. KEYBRD is also a shared routine and permits ½-msec accuracy, even if the "wrong" user has control at the time of the S's response.

FULL UTILIZATION OF CORE

The final procedure used to implement our timesharing system is the full utilization of core memory. This is accomplished by three methods. First is the special subroutine approach. Subroutines to handle standard functions can be written more efficiently by our programmer than by most of our users, and their use guarantees that long sequences of code will not be repeated by the novice (or lazy) programmer. Secondly, all subroutines and handlers are written to be reentrant, eliminating the need for duplicate code to handle the same function in both programs. Finally, normally unused space is retrieved. SPACEL provides dynamic allocation of buffer space over the loader and unused handlers. Several system programs use SPACEL (e.g., TSDTIO) and it is also available to the user.

Table 2
Illustration of Asynchronous Timing Control Available with TIMEL. The code in the main program initiates the sequence. MSEC milliseconds later control is passed to LITON, which turns on a light, sets up another timing interval, and returns. Ten msec later, control is passed to LITOFF, where the light is turned off and another interval is set up to call LITON 990 msec later. The effect is that a light will flash on for 10 msec once each second until ISW is set to nonzero. The main program is freed from the need to control the light except to start or stop the process.

```
Main Program Segment
        "
        "
        "
MSEC = MSEC
CALL TIMEL
CALL LITON (ISW)
        "

Subroutines
SUBROUTINE LITON (ISW)
IF (ISW .NE. 0) RETURN
CALL LITE (1)
MSC = 10
CALL TIMEL
CALL LITOFF (ISW)
RETURN

SUBROUTINE LITOFF (ISW)
CALL LITE (0)
MSC = 990
CALL TIMEL
CALL LITON (ISW)
RETURN
```

STANDARD DEC SOFTWARE

The standard DEC system is normally used. The only exception is replacement of I/O handling, as described earlier.

RESULTS

Evaluation can best be made in terms of the use. Users with only informal instruction in Fortran are able to make sophisticated use of the machine. Time used for experiments is high. It is not uncommon to share 8 to 10 h a day, thus doubling the subject-hours during the peak period of demand.