Coding simultaneous experiments as independent FORTRAN subroutines

FRED A. MASTERSON

University of Delaware, Newark, Delaware 19711

Simultaneous experiments can be coded as separate FORTRAN subroutines which pass requests for critical times and events to a coordinating subroutine named CHECK. These requests are stored in a "request table." When all the experiments are temporally dormant, CHECK loops through the request table, checking each request. Whenever CHECK discovers that one of the requested times or events has occurred, it CALLs the subroutine of the appropriate experiment. In addition to supporting simultaneous experiments, this system permits a simpler style of programming experiments than that possible in unenhanced FORTRAN. This simplicity is largely due to the fact that the "loop and check" control structures are implemented in the CHECK subroutine and hence are transparent to the programmer.

A strong case can be made for using FORTRAN to program computer-controlled experiments (Gilbert, 1979). FORTRAN is one of the most widely used and readily available high-level programming languages. FORTRAN systems generally contain extensive libraries of functions and subroutines. Since FORTRAN is compiled, run-time code executes rapidly.

Pitted against the above virtues are two potential weaknesses. The first is that FORTRAN programs for experimental control are cluttered with loops: loops to check if certain events have occurred and loops to check if specific time intervals have timed out. Programs would be simpler and shorter if loop-and-check control structures were transparent to the programmer. The second weakness involves timesharing. Many behavioral experiments do not require the high response speed of a dedicated computer and, thus, could share time. However, many mini- and microcomputers do not support multijob operating systems, and for them timesharing must be achieved in a different manner. One solution is to incorporate two or more simultaneous experiments in a single FORTRAN program. At best, this is an awkward solution. It is more natural to code the experiments as independent FORTRAN subroutines, to be executed under the control of a master coordinating subroutine.

The current system permits timesharing between several experiments in a single-job environment, abolishes time and event checking loops from primary source code, and allows the programmer to write each experiment as a separate procedure coded in a high-level language (FORTRAN). This system has been implemented on a Digital Equipment Corporation LSI-11 with 24k words, running DEC's RT-11 operating system and DEC's RT-11/FORTRAN-IV compiler.

OVERVIEW OF THE SYSTEM

Each experiment is coded as a separate FORTRAN

subroutine. An additional FORTRAN subroutine, named CHECK, is the loop-and-check specialist mentioned above. Each experiment subroutine can communicate to CHECK a description of important impending times or events. After communicating one or more descriptions, an experiment subroutine terminates by means of a FORTRAN RETURN command. The experiment will not be reactivated (CALLED) until one of the requested times or events occurs.

All requests for times or events are written in a data structure we call the "request table." The CHECK subroutine continuously loops through the table, checking if any of the requested times or events have occurred. Given an occurrence, CHECK activates (CALLs) the experimental subroutine that entered the now satisfied request.

Here, then, is the overview of the system. There are experiment subroutines that briefly activate, turn stimuli on or off, record responses and latencies, enter requests for future times and/or events, and then suspend activation. These experiment subroutines remain dormant while the CHECK subroutine keeps track of their requests. Whenever a requested event or time period occurs, CHECK activates the experiment that entered the request. At that point, all the power of FORTRAN is available for analyzing the current state of affairs and deciding what needs to be done next. Then the experiment subroutine enters new requests and once more terminates execution, passing control back to CHECK.

The Request Table

Experiment subroutines communicate their requests to the request table (see "How a Request is Made"). The table stores all necessary information concerning each awaited time or expected event. This information include the followings: (1) The number code of the experiment that made the request. (2) The FORTRAN line number in the experiment's subroutine to which

control will be transferred when the requested time or event occurs. (3) The priority ranking of the request—The CHECK subroutine checks higher priority requests before it checks those of lower priority. Thus, the system can favor requests for which timing is critical over requests for which accurate timing is less essential. (4) The time at which the request was entered by the experiment—This information enables CHECK to compute the time elapsed from when the request was entered until the request was satisfied. If the request was a response event, the elapsed time is the response latency. (5) The type of request—Requests can be categorized depending on whether they are requests for times or for events. (6) Detailed characterization of the request—If the request is for a time, the time value is stored in the request table. If the request is for an event, two 16-bit words are stored in the table. The first is a mask representing the bits in the sense line word that currently are of interest to the experiment. These bits correspond to switches, photocells, pushbuttons, proximity sensors, etc. The second 16-bit word portrays the values of these sense line bits that the experiment wishes to detect. This representation is disjunctive: The experiment will be reactivated when any of the sense line bits specified by MASK assume value(s) specified by PATTERN.

Each request is laid out across a row of the request table. The row numbers are ranked in terms of the priority ratings. CHECK checks requests stored in higher priority rows before checking requests stored in rows of lower priority.

How a Request is Made

Two subroutines are used to enter requests: WAIT and LOOK. The first has the calling syntax:

CALL WAIT (TIME, LINE, PRIORITY)

The effect of this subroutine is to enter a request that the calling experiment subroutine be reactivated after the interval TIME is timed out and that the subroutine should resume control at FORTRAN line number LINE. PRIORITY is an integer between 1 (highest priority) and 10 (lowest priority).

The event requesting subroutine has the calling syntax:

CALL LOOK (MASK, PATTERN, LINE, PRIORITY)

The requested event is that one or more of the sense line bits specified by MASK have the values (0 or 1) specified by PATTERN. When the event occurs, the experiment subroutine will be reactivated at FORTRAN line number LINE. PRIORITY is an integer between 1 (highest priority) and 10 (lowest priority).

Figure 1 shows an experiment-subroutine fragment that provides for timing an intertrial interval (ITI) of 60 sec. When the interval is up, control resumes at Line 3. Since the ITI need not be extremely precise,

```

      .
      .
      .
      ITI = 60.0
      .
      .
      .
      CALL WAIT (ITI, 3, 5)
      RETURN
3  -----
      .
      .
      .

```

Figure 1. The calling syntax for subroutine WAIT, which enters a timing request.

```

      .
      .
      .
      ON = "177777
      SWITCHES = "17
      .
      .
      .
      CALL LOOK (SWITCHES, ON, 4, 2)
      RETURN
4  -----
      .
      .
      .

```

Figure 2. The calling syntax for subroutine LOOK, which enters an event request.

a low priority is used. Figure 2 shows a fragment that requests the event that a subject press any one of four pushbuttons. The buttons are connected to the first 4 bits of the sense line word. The mask for these bits is 0001111 binary, or 17 octal. ON is simply a 16-bit word with all the bits set to 1 (177777 octal).

Once the request is entered via WAIT or LOOK, the experiment subroutine terminates via a RETURN statement. As a result, control is transferred to the CHECK subroutine. In general, an experiment subroutine can enter as many requests as necessary, but once all the requests are entered, the experiment subroutine must RETURN control back to CHECK.

How Experiments are Revived

When CHECK discovers that a requested time or event has occurred, it CALLS the appropriate experiment subroutine. In addition, CHECK passes the following information to the experiment subroutine: (1) The FORTRAN line number where control should recommence—This line number is used in a computed GOTO statement at the head of the experiment subroutine. The computed GOTO sends control to the desired line. (2) The time elapsed since the request was entered—If the request involved a response event, this elapsed

time is the response latency. (3) The value of the sense line word at the instant the request was satisfied—This information can be used to determine exactly which event occurred. For example, the code in Figure 2 asks for the event "one of the four pushbuttons is pressed." By examining the sense line word, the experiment subroutine can determine which of the four buttons was pressed by the subject.

Passing Information Through COMMON Memory

Most of the data communication between subroutines is accomplished with COMMON statements. For example, LOOK, WAIT, and CHECK share access to the request table via COMMON memory. Similarly, the FORTRAN line number, the elapsed time, and the current sense line word are passed to a reactivated experiment through COMMON memory.

The use of COMMON or global variables to share information has two advantages. First, it makes the communication of routine housekeeping data transparent to the programmer. Second, COMMON cuts down the time required for subroutine CALLs. The time involved in transferring control to a subroutine is an increasing function of the number of arguments of the subroutine. Thus, it saves time to transfer as much information as possible via COMMON (global) variables.

Communication With the Console Terminal

The final ingredient in our system is a set of mechanisms allowing the user to communicate with experiments via the console terminal. To this end, we have implemented a simple console command language for starting up and stopping experiments, sending messages to experiments, and examining the sense word, the relay word, and the request table. These commands are shown in Table 1. Notice that messages are sent to experiments by means of message buffers. Each experiment subroutine is assigned an 80-character string buffer to mediate console-to-experiment message passing.

An experiment can read a message placed in its message buffer by means of subroutine LISTEN. In

Table 2
FORTRAN Subroutines Used by Experiments
to Read or Send Messages

Subroutine Call	Effect
CALL LISTEN (STRING)	Causes the experiment to read its message buffer. The contents of the buffer are assigned into the variable named STRING. This allows experiments to receive messages created by the user via the SEND console terminal command (Table 1).
CALL TELL (STRING)	Causes a string of up to 80 characters to be displayed on the console terminal.
CALL TELLBK (LINE,STRING)	Similar to TELL. After the message is displayed, control is transferred back to the experiment—at FORTRAN line number LINE.

addition, an experiment can send messages to the console terminal via one of the subroutines TELL or TELLBK. As in the case of console-to-experiment communication, experiment-to-console message passing is mediated by the 80-character string buffers. The use of the subroutines for message transfer is outlined in Table 2.

Messages are read from or sent to the console terminal only when there is nothing more important to do. Message I/O can occur only when CHECK has failed to find a ripened time or event request. Furthermore, messages to be displayed on the console terminal are transmitted one character at a time so that experimental requests can gain control at any time during the output of a message.

A SIMPLE EXAMPLE

Figure 3 displays a subroutine for discriminated appetitive barpressing. The onset of a lamp indicates the availability of food reinforcement. Pressing the bar will then procure food and also turn off the lamp. After the ITI, this procedure repeats, until the total number of trials (NTRIAL) has been played out.

The line numbers on the far left have been added for reference purposes and play no role in the subroutine. The computed GOTO in reference line 9 is the first executable FORTRAN statement. It passes control to the desired FORTRAN line number whenever the experiment subroutine is CALLED by CHECK.

FORTRAN line number 1 (reference line 12) is the initial entry to begin the experiment. Parameters are defined, such as sense-word and relay-word bits, and the trial number counter is initialized to zero. FORTRAN line number 2 (reference line 22) begins the block that starts a new trial. Here, the trial number counter is incremented and the lamp signal is turned on by subroutine TURN. This subroutine's arguments, LAMP and ON, play the roles of mask and pattern, respectively. The mask specifies the bits of the relay word to be altered. The pattern specifies their new values. In this

Table 1
Console Terminal Commands

Command	Action
GO m,n	Starts the mth experiment at FORTRAN line number n
STOP m	Removes all requests made by experiment m from the request table, thus effectively terminating the experiment
SEND m/<message>	Places a message up to 80 characters long in the message buffer belonging to experiment m
?W	Displays the "state of the world" on the console terminal: the bits of the sense and relay words, and the clock time
?T	Displays the request table on the console terminal

```

01      SUBROUTINE LEVER
02      COMMON/ESYS/LINE, IDENT, DRVIN, MSKDRV, DRVO, TNOW, TELPS
03      INTEGER*2 DRVIN, DRVO, TNOW (2), TELPS (2)
04      INTEGER*2 LATNCY (200), TRIAL, NTRIAL, IARG (10)
05      INTEGER*2 FEEDER, LEVER, LAMP, ON, OFF
06      LOGICAL*1 STRING (80)
07      REAL*4 ITI
08      C
09      GOTO (1,2,3,4,5,6) LINE
10      C
11      C   S E T   F I X E D   P A R A M E T E R S :
12      1   OFF = 0
13         ON = "177777
14         LEVER = 2**3
15         LAMP = 2**6
16         FEEDER = 2**7
17         TRIAL = 0
18         ITI = 60.0
19         NTRIAL = 100
20      C
21      C   S T A R T   A   T R I A L :
22      2   TRIAL = TRIAL + 1
23         CALL TURN (LAMP, ON)
24         CALL LOOK (LEVER, ON, 3, 2)
25         RETURN
26      C
27      C   A   L E V E R   P R E S S :
28      3   CALL TURN (LAMP, OFF)
29         CALL TURN (FEEDER, ON)
30         CALL WAIT (.05, 4, 5)
31         LATNCY (TRIAL) = IJCVT (TELPS)
32         IF (TRIAL .EQ. NTRIAL) GOTO 100
33         CALL WAIT (ITI, 2, 10)
34         RETURN
35      C
36      C   T U R N   O F F   F E E D E R   P U L S E
37      4   CALL TURN (FEEDER, OFF)
38         RETURN
39      C
40      C   E N D   O F   S E S S I O N :
41      100 CALL TELL ('LEVER: FINISHED...TYPE "GO 2,5" TO SEE DATA')
42         RETURN
43      C
44      C   P R I N T   O U T   D A T A :
45      5   NBLOCK = NTRIAL/10
46         DO 200 I=1,NBLOCK
47            DO 300 J=1,10
48               IARG (J) = LATNCY (J + 10*(I-1))
49               CALL SICVT (IARG, 10, STRING)
50               CALL TELLBK (6, STRING)
51               RETURN
52            6   CONTINUE
53            200 CONTINUE
54      C
55      C
56         RETURN
57      END

```

Figure 3. A real example: discriminated barpressing. The left line numbers are for convenient reference and play no role in the actual subroutine.

case, LAMP is a 16-bit word with only one nonzero bit: the 6th. Consequently, TURN sets the 6th bit of the relay word to the bit value specified in the pattern word ON. Since all bits of ON equal 1 (including the 6th), the net effect is to set the 6th bit of the relay word to 1.

The trial start block ends with a request for a leverpress event (reference line 24). Subroutine LOOK instructs the CHECK subroutine to reactivate the experiment at FORTRAN line number 3 when a leverpress

occurs. Then, in reference line 25, the experiment RETURNS control and remains dormant until a leverpress occurs.

FORTTRAN line number 3 (reference line 28) begins the block entered when the lever response finally occurs. The lamp is turned off and the feeder is turned on. Then, in reference line 30, a request is entered to activate the block at FORTRAN line number 4 when .05 sec times out. The sole purpose of this block (reference lines 37-38) is to terminate the 50-msec pulse to

the feeder. In reference line 31, the response latency is recorded in a data vector LATNCY. TELPS is the elapsed time value automatically passed back to the experiment when a request is satisfied. (IJCVT is a DEC SYSLIB function that converts TELPS from RT-11's 4-bit integer representation to the 2-bit integer representation.) The logical IF statement in reference line 32 tests to see if the desired number of trials have been run. If not, subroutine WAIT schedules the ITI and requests that control be returned to FORTRAN line number 2 at the end of the ITI. If all the trials have been run, control flows to FORTRAN line number 100, where a message is sent to the console terminal to the effect that the experiment is finished.

Reference lines 45-53 transfer data to the console terminal. Each successive block of 10 latencies is converted to an ASCII string via subroutine SICVT (reference line 49). Then, TELLBK sends the string to the experiment's message buffer (reference line 50). At this point the experiment subroutine is suspended, and control is RETURNed to CHECK (reference line 51). CHECK prints the message one character at a time whenever there is nothing more important to do. After CHECK has printed the last character of the message string on the console terminal, and assuming no higher priority requests become satisfied, CHECK will CALL the experiment subroutine and send control to FORTRAN line number 6 (the first argument of TELLBK in reference line 50). When this happens, control passes back to the outer DO-loop (reference line 46), and the next 10 latencies are similarly processed.

Though not shown here, data also can be written to floppy disk using the DEC SYSLIB routine IWRITE. This subroutine operates asynchronously, and so does not interfere with real-time experimental control.

EVALUATION

We have used the system for 2 years. Over this period we have come to appreciate more and more the use of a general-purpose checking routine which handles all experimental requests. The experiment subroutines are short and simple because all time and event checking loops are hidden in the general-purpose checking routine. The use of subroutines WAIT and LOOK to enter requests contributes to a highly comprehensible and readable programming style.

A major advantage of the system was not fully anticipated. Employment of a centralized time and event checking specialist allows the programmer to decompose an experiment into two or more parallel processes. For example, many experiments increment a time period index at regular intervals. Using the current system, this can be managed by a short block of code that is completely independent of the rest of the sub-

routine. The benefit is that such mundane chores as timing periods and incrementing time period indices can be totally ignored while the programmer concentrates on coding the main experimental contingencies.

A current example of the parallel perspective from our laboratory involves testing warning signals and safety signals against a Sidman avoidance baseline. The test phase of the experiment is easily viewed as two relatively independent processes: (1) a process governing the presentation of the test probe signals and (2) a process governing the Sidman avoidance schedule. These two processes have been programmed as two independently executing blocks of the experiment subroutine. Thus, we can code test probe delivery and the Sidman schedule as parallel processes with nearly independent flow of control. In general, decomposition of the experiment into parallel-execution modules simplifies program construction.

The main disadvantage of the system involves response time. Each experiment must share time with the others. Furthermore, coordinating the experiments and their requests requires additional time. In our experience running up to four simultaneous animal learning experiments, the system's responsivity falls within 20 msec. As mentioned earlier, many behavioral experiments can tolerate relatively brief delays.

More time-intensive experiments can be run individually. Even without timesharing, the system provides the advantages of hidden loops and decomposition into parallel processes. In its current form, the responsivity of the system running a single experiment is within 5 msec. That figure undoubtedly can be improved by coding such basic subroutines as CHECK, WAIT, and LOOK in assembly language.

Another problem with the current system is the primitive nature of the console terminal commands. Instead of keeping a list, for each experiment subroutine, of FORTRAN line numbers for entering different blocks of the experiment, the following standard has been adopted for all experiments. FORTRAN line numbers 1, 2, 3, and 4 are reserved for: (1) initializing the parameters of an experiment, (2) beginning the experiment, (3) dumping the experiment's data, and (4) aborting the experiment. Having established this standard, we plan in the near future to implement the console commands: INIT *n*, BEGIN *n*, DATA *n*, and ABORT *n*, where *n* is the number of an experiment.

REFERENCE

- GILBERT, L. H. Computer-controlled experiments: The case for FORTRAN. *Behavior Research Methods & Instrumentation*, 1979, 11, 593-595.

(Received for publication April 8, 1981;
revision accepted September 10, 1981.)