Time-stamping computer events to report .1-msec accuracy of events in the Micro Experimental Laboratory

WALTER SCHNEIDER, ANTHONY ZUCCOLOTTO, and STEPHEN T. TIRONE
University of Pittsburgh, Pittsburgh, Pennsylvania

A critical problem in most real-time programming tasks is the verification that timing of the presented displays and recorded events occur at the expected times. It is often difficult for psychologists to verify critical timing because of the lack of costly specialized tools and technicians to measure events with millisecond accuracy. Algorithms utilized in a new Time Audit mechanism, added to the Micro Experimental Laboratory, are described. This new mechanism involves the recording and time stamping of all input and output events with 0.1-msec accuracy. This allows the experimenter to determine the initiation, duration, and termination of each event and also makes it possible to relate these events to the synchronization of the screen refresh cycle. A customized user interface provides a time log and event-tracing feature that enables nonprogrammers to determine the duration of command execution to the .1-msec level, allowing rapid, precise assessment of program execution. The resulting time log provides a detailed specification of the experimental events for debugging and a permanent record of the experimental procedure.

Computerized experimentation often requires critical monitoring of the timing of computer displays. It is difficult to time millisecond events precisely (see Brysbaert, 1990; Creeger, Miller, & Paredes, 1990; Segalowitz & Graves, 1990). Algorithms are available for millisecond timing (e.g., Graves & Bradley, 1988; Rensink, 1990; Westall, Perkey, & Chute, 1986), and software systems enable accurate timing through specially designed and optimized program calls (e.g., Schneider, 1989). These systems enable precise timing, provided that the experimenter codes the experimental sequence correctly and no variable operating system events (e.g., disk writes) occur (see Creeger et al., 1990). Unfortunately, almost all computer programming is highly error prone. To assess timing precision, it is critical to provide an easy and accurate method to verify millisecond timing of display events.

Granting agencies, journal editors, and the general public require that detailed records of experimental procedures be kept, to permit replication or auditing. For computerized experiments, to provide a reviewer with the source program offers an accurate specification that is difficult to verify without hours of study and extensive testing. It is also difficult for a faculty member to sit down with the code from a new graduate student and examine each timing loop to verify that the program performs as intended. In this paper, we describe procedures to allow debugging and auditing of time-critical program segments.

There are four methods available to access timing accuracy. The first method, slowing down the clock rate, allows the determination of event rates with hand timers.

For example, in Micro Experimental Laboratory (MEL; Schneider, 1988a, 1988b), the tick rate of the reaction time clock can be slowed from milliseconds to seconds. If two displays are to be presented with an interstimulus interval of 20 msec, the accuracy cannot be estimated reliably through observation of the screen. However, if the tick rate is increased from 1 msec to 1 sec, the timing can be checked with a watch to verify that 20 ticks have occurred. This technique is not useful for timing computer input/output events that are hardware dependent, such as the screen refresh rate or input delays (see Creeger et al., 1990; Paredes, Miller, & Creeger, 1990).

The second method involves the frequent execution of the critical timing component in a loop and the estimation of the elapsed time with an external reference such as a stopwatch. For example, in displaying a stimulus and a mask, one might program a loop of 1,000 executions and time the number of seconds required for executing the loop, thus providing an estimate for which seconds translate into milliseconds. As with the first method, this can be inaccurate if one is timing computer input/output events that are hardware dependent (e.g., the loop adds some small delay and the transfer from the end of the cycle to the beginning of the cycle may be different in a loop than when the subject initiates it).

The third method involves calling software timers that can time intervals and record the time between time check calls. This typically involves a sequence of steps such as starting a timer, performing an input/output operation such as reading a timer, waiting for a time interval that represents the best estimate of the remaining time to display the stimulus, presenting the stimulus on the next screen refresh, and recording the actual elapsed time on the timer (see Creeger et al., 1990). This is cumbersome, and it

Correspondence should be addressed to W. Schneider, Learning Research and Development Center, 3939 O'Hara Street, Pittsburgh, PA 15260 (e-mail: schneider@pitt.vms.cis.edu).

requires very careful coding if one wishes to get an accurate measurement. It also adds small delays to the program that would not be present with the software timers removed.

The fourth method, monitoring with external hardware, provides accurate measurement, but it requires expensive external hardware and often the assistance of trained technical staff. Special hardware can be built to monitor hardware electrical potentials and acoustic or visual events. For example, Segalowitz and Graves (1990) measured keyboard and mouse delays with such hardware. Using storage oscilloscopes, one can record the delay between digital events. For example, by monitoring the red CRT analog output to determine when the voltage pulse occurs, one can measure the time until the first green signal. If the stimulus is not cyclic, a storage oscilloscope must be used. Setting up an oscilloscope to trigger on a computer display (e.g., only after the fixation cross appears on a display containing many characters) can be quite challenging and is often beyond the technical skills of most psychologists. It typically requires several hours to wire connections, record the waveforms, and interpret the results. Although important for developing experimental accuracy, this technique is unlikely to be used routinely in developing experiments.

All four methods are useful, but they have their limitations. What is needed is a relatively quick debugging option that allows the determination and verification of time-critical experimental events.

AUDIT TRAIL TIMING

In this paper, we describe an implementation of a new method called *time audit trail* recording. This involves time stamping all input/output events during the experiment and then providing the experimenter with a convenient interface to evaluate the timing data. If experimenters know every input and output event that has occurred, they can determine when stimuli have been presented and when the subject has produced responses. This enables verification of the experimental procedure. For the experiment to be timed effectively, the audit timing must add minimal load to the computer execution.

To be useful, time audit trails must be displayed in a way that enables the experimenter to identify the critical timing events. A list of all the input/output events on the computer can be quite long, burying the experimenter in pages of output that may hinder the interpretation of the event sequence. Therefore, procedures for searching output lists and marking sections of interest should be provided.

To provide audit trail timing in MEL, a series of routines has been added to log, process, and display timed events. The logging of events required recording the end of all input and output events with 0.1-msec accuracy. The timing was accomplished by reading the hardware and software counts of the IBM 6253/8254 timer chip that counts a 1.190-MHz rate (see, e.g., Creeger et. al, 1990;

Graves & Bradley, 1988). This count is then converted to 0.1-msec intervals. The reading of the clock requires a small amount of time (about 15–20 μ sec on a 386 20-MHz machine).

The technical requirements for time-stamping without disrupting timing include three points. First, the time-stamping process must involve no disk input/output during critical timing (see Creeger et al., 1990, for a discussion of disk access variability). Second, time stamping must have minimal dynamic data storage requirements so that new memory allocations and transfers are minimized. Third, time stamping should enable easy labeling of critical time components to enable fast and perhaps automated data analysis of time periods.

Separation of experimental trial generation and execution improves presentation speed, reduces timing variability, and facilitates time stamping. For example, the MEL system has separated the generation and the execution of time-critical events. All subject input/output commands are preprocessed to minimize execution time before the trial begins (e.g., the screen display is built in memory, so that only memory-to-screen move operations occur during the trial; see Schneider, 1988a, 1988b, 1989, 1990). The preprocessing can produce speed improvements of as much as 20 times over the use of standard BIOS writes to the screen (e.g., to write an entire screen requires 198 msec using BIOS calls and 9 msec using MEL on an IBM 386 20-MHz PC). The list of real-time commands is called an *event table*. Since all the memory storage and most of the computation occurs during the preprocessing and construction of the event table rather than during the execution of the event table, the timing variability occurs in the intertrial interval rather than during the stimulus presentation portion of the trial (e.g., the length of the intertrial interval may vary typically by 100 msec, but the timing of events during the trial is precise to the millisecond). The use of an event table facilitates time stamping. At the beginning of the trial, all the output events are already stored in memory and need not be copied during the trial. All that needs to be recorded is the precise time when the command was executed.

Time stamping involves recording the completion or termination time of each event table operation. MEL uses a 48-bit clock to time the experiment in 0.1-msec intervals. On the completion of each event, the current time is read (requiring about 20 μ sec on a 20-MHz 386) and the 48-bit count is stored in memory. The system maintains a count of the number of events entered into the event table during the generation phase and uses this information to allocate a buffer before the trial begins. The required storage is 6 bytes $\times$ the number of events. Since the recording time is minimal (e.g., < .02 msec) and commands are buffered in the event table, the duration of each command is simply the time from the completion of the previous command to the start of the next command. This enables the determination of when the command output begins and when it is completed. For example, the sequence of a TONE command ending at 421.3

followed by a GRAPHICS _ DISPLAY command ending at 429.9 indicates that nothing appeared on the display until 421.3 msec, that the display was completed during the 8.6-msec interval, and that it was over at 429.9 msec. In addition to the time of the command execution, the source line number of the originating command is recorded to facilitate later debugging.

To determine actual refresh rates, it is critical to record screen events when the display raster scan or electron gun is at either the top or the bottom of the screen. Display scanning is usually crystal oscillator controlled. Hence, if the refresh rate is initially recorded at the beginning of the trial, all subsequent refreshes can be mathematically determined (e.g., on a VGA display, the refresh occurs every 14.29 msec). The program determines the refresh rate of the monitor at the beginning of the session or monitor mode switch (e.g., a change from EGA to VGA emulation mode altering scan frequency). The event table is not started until a top-of-screen event occurs. The beginning of the event table is logged. With the beginning scan and scan rate, the position of the scan can be determined for all the events of a trial. To determine when the scan line reaches a pixel on the display requires the calculation of the scan writing rate and the delay between the top-of-screen event and the scanning of the first pixel of the display. The actual refresh of the screen on a monitor occurs when the CRT's electronic gun activates the phosphor of a given pixel. The total time for an entire screen display is a function of the number and position of the pixels. In MEL, we estimate the display start time on the basis of a common point of the displays (i.e., the lower left pixel of the stimulus).

All subject input events must be time stamped to monitor the response component of the experiment. In MEL, each keyboard, response box, and mouse input is time stamped.

To facilitate program debugging, the experimenter can add *event messages* to the source code of the experiment to be time stamped. For example, before a critical masking event, the message "mask" can be inserted into the event table when the masking stimulus is displayed on the screen. When the event message is encountered during the execution of the event table, the time stamp is recorded. The event messages facilitate the searching of the time log file for the event message text string and determination of the intervals between critical events.

At the end of a trial, the event table and the time-stamping data are written to a file for processing after the debugging experimental session. This takes substantial time and disk space (e.g., the log of 20 sec of time stamping can require 5-8K of disk space). The time log data structure includes 6 bytes for the 48-bit clock time, 2 bytes for the command number, and 2 bytes for the line number originating the timed event.

Postprocessing programs are needed to facilitate viewing the data to verify timing. Three programs are utilized in MEL. The first is an event table dump program, TIMEOUT, that gives a time line listing of all the time

log data, listing the time of the completion of the command. An example of this output is shown in Figure 1. The program converts the event table commands, stored as integer command numbers, into command names. The output is complete but quite cumbersome to process. For example, to determine the duration of a display, the experimenter would have to determine the line numbers specifying when the display was written to the display memory, when the display palettes turned on or changed state, when the display was erased or overwritten, and so forth. This requires expert knowledge of the display hardware and control software.

The second program, TIMESHOW, interprets the time log data to relate the initiation and termination of events to the full event tables and the source code statements of the program. Figure 2 illustrates the basic display. On the left edge are *event traces*. These illustrate the relationships between events. The trace begins with the columns reading "wtlkiomr," which refer to waits, tones, latencies, keypresses, input port, output port, mouse, and response box events. The traces ABCD show display events. The event traces show the duration of the events. The "." state indicates that the event is inactive. The double line ("||") indicates that the event is active. For display events, there are thus three states, inactive ("."), in display memory but not visible ("|"), and visible ("||"). TIMESHOW reports the start, duration, and end state of the time trace for the line that the cursor is currently on. For display commands, the number of refreshes is also provided. Different classes of events are color coded to facilitate visual filtering of the data. The experimenter can search for specific text strings to identify cer-

ELAPSED	DURATION	LINE	COMMAND
=====	=====	=====	=====
4362.1	0.0	240	EVENT_MESSAGE
4362.1	12.9	254	WAIT_TOP
4375.6	13.4	254	COLOR_CLEAR
4375.8	0.2	269	DISPLAY
4375.8	0.0	270	EVENT_MESSAGE
4875.1	499.2	272	WAIT
4875.1	0.0	273	DISPLAY_OFF
4875.3	0.2	279	DISPLAY
4876.5	1.1	282	WAIT_TOP
4876.5	0.0	283	DISPLAY_ON
4876.5	0.0	284	EVENT_MESSAGE
4876.6	0.0	285	LATENCY
4876.7	0.1	287	KEY_IN
5376.1	499.4	288	WAIT
5376.2	0.2	289	DISPLAY
5376.2	0.0	295	EVENT_MESSAGE
5948.0	573.1	334	WAIT
5949.3	1.3	335	KEY_IN
5949.3	0.0	335	EVENT_MESSAGE
5993.8	0.0	336	EXECUTE
6005.2	11.4	167	WAIT_TOP
6018.6	13.4	167	COLOR_CLEAR
6018.7	0.1	296	DISPLAY
7518.1	1499.3	300	WAIT
7518.3	0.2	309	DISPLAY

Figure 1. TIMEOUT time stamping of commands.

Figure 2. TIMESHOW trace and command output.

tain events (e.g., find the presentation of the "+" fixation stimulus).

Trace A in Figure 2 shows the events for the display of the fixation, probe, and mask. The "+" is written at 4,375.8 msec. It is displayed until the occurrence of the display off event at 4,875.1 msec. At 4,875.3, the letter "B" is written to the screen but not yet visible (indicated by single line on Trace A). At 4,876.5 msec, the screen is turned on, showing the probe letter. At 5,376.2, the mask stimulus is presented, terminating the display. Latency timing is shown for the "l" trace starting at 4,876.6 with the LATENCY command and continuing until the response at 5,949.3, terminating the response with the input of a "3" key. The wait times in the experiment are illustrated by the "*" in the "w" trace. By scanning the wait latency and keypress input, the experimenter can easily identify when subject input occurs and whether the timing of the response is accurate.

The third program, TIMESTAT, allows searches for display event messages and statistical determination of the intervals between the messages. The program scans the time log file for event messages of the form \$nn, where nn is a number from 0 to 99. Figure 2 shows five event message events starting with "\$0 start" at 4,362.1 msec, and proceeding to "\$1 fixation" at 4,375.8 msec, then to "\$2 probe," and finally to "\$3 mask." The TIMESTAT

program scans the TIMESHOW output and builds a table in which each row is a trial and the interval between each \$nn event is shown (see Figure 3). Eight characters of the message following the \$0 event are displayed. At the bottom of the table are the mean and standard deviation of the duration of that event. An experimenter can quickly scan the table to determine the duration of the critical events and verify that different conditions have either the same timing or the expected difference in timing. When run with the conditionalization table option (specifying on the \$0 option a condition number such as "\$0@2" for the second type of trial), the TIMESTAT program will display the table for all conditions in chronological order and then produce separate tables for each condition. The mean and standard deviations for the event messages can then be compared across conditions.

Figure 3 shows a TIMESTAT output table. The first column gives the trial number. The second column gives the start time of the trial in milliseconds. The remaining columns give the elapsed times since the previous event messages. The mean and standard deviation of all the event messages are shown below. Note that the "fixation" duration varies across trials. This is to be expected, because it depends on the first top-of-screen event after the subject has pressed the "go" button. The probe duration ranges from 500.7 to 500.8 msec. In a masking

Event Message	0	1	2	3	4
Trial	ready	fixation	probe	mask	sub resp
1	4362.1	13.7	500.7	499.7	573.1
2	6014.8	1.7	500.7	499.9	594.7
3	8793.7	7.4	500.8	499.9	580.4
4	11147.4	12.3	500.8	499.8	640.4
5	12852.1	9.5	500.8	499.9	621.8
6	15518.9	5.8	500.7	499.9	559.5
7	18322.6	14.3	500.7	499.9	583.2
8	25208.3	11.3	500.8	499.8	586.3
9	27936.9	9.3	500.7	499.8	573.3
10	30578.2	6.2	500.7	499.9	591.1
<hr/>					
mean		9.1	500.7	499.9	590.4
SD		3.75	0.05	0.07	22.85

Figure 3. TIMESTAT statistical output of \$time events absolute time – relative time from last event message (in milliseconds).

study, it is critical that the masking period be fixed across conditions, and in the table this has only a .05-msec variability. The mask stimulus is similarly constant, with a mean of 499.9 msec and a standard deviation of .07 msec. The actual subject response time is expected to be variable in this run, showing a mean of 590.4 msec after the mask and a standard deviation of 22.85 msec. Such a timing table permits easy verification of the timing accuracy of the experiment.

CONCLUSION

The goal of time-auditing programs is to assess the accuracy of an experiment before the final experiment is run. Time stamping also provides audit information for others to use to check the accuracy of the program. The first time that we used time audit, we found timing errors in a program that had been used to collect data, that had been written by a programmer with 20 years of experience, and that was presumed to be bug free. Checking the time audit provides the laboratory supervisor with a complete specification of time-critical experiment events. This helps researchers sleep better and reduces the possibility of the nightmare in which, after a paper has been published and another researcher has failed to replicate an effect, one goes back over the code and finds that the original program was coded incorrectly. Time audit procedures can be incorporated into programs that utilize event tables for critical timing. This system will be included in the 1993 release of MEL 2.0.

REFERENCES

- BRYLSBAERT, M. (1990). A warning about millisecond timing in Turbo Pascal. *Behavior Research Methods, Instruments, & Computers*, **22**, 344-345.
- CREEGER, C. P., MILLER, K. F., & PAREDES, D. R. (1990). Micromanaging time: Measuring and controlling timing errors in computer-controlled experiments. *Behavior Research Methods, Instruments, & Computers*, **22**, 34-79.
- GRAVES, R., & BRADLEY, R. (1988). More on millisecond timing and tachistoscope applications for the IBM PC. *Behavior Research Methods, Instruments, & Computers*, **20**, 408-412.
- PAREDES, D. R., MILLER, K. F., & CREEGER, C. (1990). Graphic precision: Controlling stimulus displays on IBM PC-compatible computers. *Behavior Research Methods, Instruments, & Computers*, **22**, 319-322.
- RENSINK, R. A. (1990). Toolbox-based routines for Macintosh timing and display. *Behavior Research Methods, Instruments, & Computers*, **22**, 105-117.
- SCHNEIDER, W. (1988a). Micro Experimental Laboratory: An integrated system for IBM PC compatibles. *Behavior Research Methods, Instruments, & Computers*, **20**, 206-217.
- SCHNEIDER, W. (1988b). Micro Experimental Laboratory language reference manual. Pittsburgh, PA: Psychology Software Tools.
- SCHNEIDER, W. (1989). Enhancing a standard experimental delivery system (MEL) for advanced psychological experimentation. *Behavior Research Methods, Instruments, & Computers*, **21**, 240-244.
- SCHNEIDER, W. (1990). *MEL user's guide: Computer techniques for real time psychological experimentation*. Pittsburgh, PA: Psychology Software Tools.
- SEGALOWITZ, S. J., & GRAVES, R. E. (1990). Suitability of the IBM XT, AT, and PS/2 keyboard, mouse, and game port as response devices in reaction time paradigms. *Behavior Research Methods, Instruments, & Computers*, **22**, 283-289.
- WESTALL, R., PERKEY, M. N., & CHUTE, D. L. (1986). Accurate millisecond timing on Apple's Macintosh using Drexel's MilliTimer. *Behavior Research Methods, Instruments, & Computers*, **18**, 307-311.